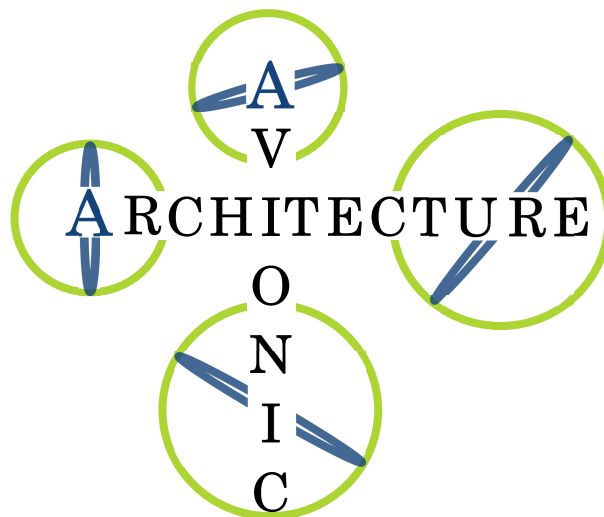


Dokumentation

Avionic Architecture



Teilnehmer:

Jörn Bellersen

Martin Bornhold

Marco Braun

Henning Elbers

Niklas May-Johann

Thomas Nordlohne

Jenny Inge Röbesaat

André Schaadt

Patrick Schmale

Steven Schmidt

Sebastian Vander Maelen

Markus Franz Wieghaus



Inhalt

Abbildungsverzeichnis	XIV
Tabellenverzeichnis	XV
Abkürzungsverzeichnis	1
1 Einleitung	1
1.1 Motivation	1
1.2 Die Aufgabenstellung	2
1.3 Auswahl der nicht kritischen Anwendung	3
1.3.1 Audio-Streaming	6
1.3.2 Lightshow	8
1.3.3 Wärmebildkamera	9
1.3.4 Objekterkennung	11
1.3.5 Fazit	13
1.4 Das Szenario	15
2 Projektmanagement	20
2.1 Projektmodalitäten	20
2.2 Projektablauf	20
2.2.1 Grober Projektaufbau	20
2.2.2 V-Modell	22
2.2.3 Phasen des V-Modells	23
2.2.4 Arbeitspakete der einzelnen Phasen	24
2.3 Organisationsstruktur	27
2.3.1 Verantwortlichkeitsbereiche innerhalb der Projektgruppe	27
2.4 Projektressourcen	30
2.4.1 Zeitliche Ressourcen	30
2.4.2 Finanzielle Ressourcen	30
2.4.3 Kosten des Gesamtprojekts	31
3 Voruntersuchungen	32
3.1 Theoretische Grundlagen zu Multikoptern	32
3.1.1 Vergleich der Multikopterarten	33
3.1.2 Flugmechanik von Multikoptern	35
3.1.3 Sensorik	38
3.2 Regelungstechnische Grundlagen	42
3.2.1 Unterschied zwischen Steuerung und Regelung	42
3.2.2 Komponenten und Signale in Regelkreisen	43



3.2.3	Reglertypen	45
3.2.4	Regelung von Multikoptern	49
3.3	Digitale Bildverarbeitung	50
3.3.1	Methodiken der Objekterkennung	50
3.3.2	Methodiken der Objektverfolgung	53
3.4	Kamerapositionierung und -stabilisierung	60
3.4.1	Optische Bildstabilisierung	60
3.4.2	Mechanische Bildstabilisierung	61
3.4.3	Elektronische Bildstabilisierung	62
3.4.4	Kardanische Aufhängung	62
3.5	Trajektorienplanung und -beschreibung	65
3.5.1	Kinematische Ketten	65
3.5.2	Vorwärts-Kinematik	66
3.5.3	Rückwärts-Kinematik	66
3.5.4	Koordinatentransformation	66
3.5.5	Parametrisierung nach Denavit Hartenberg	68
3.5.6	Quaternionen	69
3.6	Kameratypen und -schnittstellen	71
3.7	Drahtlose Datenübertragung	73
3.7.1	ZigBee	73
3.7.2	Bluetooth 4.0	74
3.7.3	Wireless Local Area Network	75
3.7.4	Fazit	76
3.8	Rechtliche Grundlagen	77
4	Systemdefinition	79
4.1	Verwendete Komponenten innerhalb des Systems	79
4.1.1	ASG	79
4.1.2	Das AEVV-System	79
4.2	Rahmenbedingungen für das Szenario	80
4.3	Anwendungsfälle	81
4.3.1	Multikopter steuern	82
4.3.2	Videoaufnahme durchführen	87
4.3.3	Objektverfolgung durchführen	89
4.3.4	Kamerabild & Telemetriedaten verarbeiten	92
4.3.5	Erweitertes Szenario	96
5	Risikoanalyse	99



6	Anforderungsspezifikation	105
6.1	Anforderungen an das ASG	106
6.2	Anforderungen an das AEVV-System	108
6.3	Anforderungen an die Fernsteuerung	111
6.4	Anforderungen an die Telemetriestation	112
6.5	Anforderungen an die Umwelt	112
6.6	Optionale Anforderungen	113
6.7	Sicherheitsanforderungen	114
6.8	Spezifikation der Echtzeitanforderungen	115
6.8.1	Erhebung der Echtzeitanforderungen für das ASG	116
6.8.2	Erste Evaluierung der Echtzeitanforderungserhebung für das ASG	122
6.8.3	Erhebung der Echtzeitanforderungen für das AEVV-System	123
6.8.4	Erste Evaluierung der Echtzeitanforderungserhebung für das AEVV-System	128
6.9	Anforderungen an die Ressourcen	131
6.9.1	Benötigte Rechenleistung	131
6.9.2	Benötigter Speicherbedarf	131
6.9.3	Benötigte Bus- bzw. I/O-Zugriffe	132
7	Architektur und Entwurf	133
7.1	Mixed-Critical Systems	133
7.1.1	Aktuelle Forschung	133
7.1.2	Herausforderungen	134
7.1.3	Extrafunktionale Eigenschaften	136
7.1.4	Vorteile von MCS	137
7.1.5	Entwurf von MCS	138
7.1.6	Fazit für diese Projektgruppe	141
7.2	Architekturentwurf	142
7.2.1	Gesamtsystem und Quadropter	142
7.2.2	ASG	143
7.2.3	AEVV-System und Telemetriestation	146
7.2.4	Beschreibung der Interfaces	148
7.3	Auswahl und Beschreibung der Hardware	153
7.3.1	Prozessorplattform	154
7.3.2	Kamera	155
7.3.3	Gimbal	158
7.3.4	Auswahl der Telemetriestation	160
7.3.5	Funkverbindung zur Telemetriestation	161
7.3.6	Multikopter	161
7.3.7	Auswahl der Sensoren	166



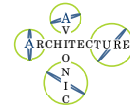
7.3.8	Verteilerplatine	170
7.4	Plattformdesign	172
7.4.1	Rechen- und Speicherkomponenten	173
7.4.2	Sensoren und Aktoren	174
7.4.3	Blockdesign in Vivado	175
7.4.4	Mixed-Criticality und Safety-Critical Aspekte im Plattformdesign	183
7.5	Mapping	185
7.5.1	Komponenten und Schnittstellen, die gemappt werden müssen	185
7.5.2	Mapping auf die Ressourcen	187
8	Entwicklungsumgebung	188
8.1	Vivado	188
8.1.1	2013.4	188
8.1.2	2014.1	188
8.1.3	2014.4	189
8.2	Vivado SDK	189
8.2.1	Versionen	189
8.2.2	Debugging	189
8.3	Buildroot	190
8.4	CAMeL-View	191
8.4.1	Analyse	191
8.5	Wrapper „Simplified Simulation Framework“	192
8.5.1	Anpassungen für den SIL-Test	192
8.5.2	Anpassungen für den Zynq	193
9	Implementierung der Teilsysteme	195
9.1	Integration der Hardware-Plattform	195
9.1.1	Anpassung des TE0703-04 Carrierboards	195
9.1.2	Aufbau der Verteilerplatine	195
9.2	Board Support Package	201
9.2.1	Bootvorgang	201
9.2.2	First Stage Bootloader	203
9.2.3	Second Stage Bootloader: U-Boot	203
9.2.4	Devicetree	205
9.2.5	Linux Kernel	205
9.2.6	Der PPM-IP-Core	205
9.3	Dataminer	209
9.3.1	Akkuwächter	209
9.3.2	Peripherie Controller	210
9.3.3	Fernsteuerungsrückkanal	211



9.3.4	Temperatursensor Controller	213
9.3.5	Funkfernsteuerungs Controller	214
9.3.6	Lagebestimmung	216
9.3.7	BRAM Controller	218
9.3.8	Dataminer Schedule	220
9.4	Sensordatenverarbeitung	223
9.4.1	Winkeldefinition	223
9.4.2	Koordinatentransformation von Sensorchip zu Quadrocopter	224
9.4.3	Winkelgeschwindigkeiten im erdfesten Koordinatensystem	225
9.4.4	Sensordatenfusion über Komplementärfilter	226
9.5	ASG	236
9.5.1	Modellbasierte Entwicklung des ASG	236
9.5.2	Motoransteuerung	249
9.5.3	BRAM Controller	250
9.6	Realisierung des AEVV-Systems	251
9.6.1	Die AEVV-Verwaltungseinheit	252
9.6.2	Konfiguration des AEVV-Systems	254
9.6.3	Wrapper-Klassen für die Bibliotheken der Subkomponenten	255
9.6.4	Realisierung der Avionik-Schnittstelle zum ASG	256
9.6.5	Datenhaltung der Telemetrieparameter im AEVV-System	257
9.6.6	Aufbau einer Netzwerkverbindung zur Telemetriestation	259
9.6.7	Übertragungsprotokoll für das Versenden der Telemetriedaten	259
9.6.8	Erweiterung der Telemetriedaten innerhalb des AEVV-Systems	260
9.6.9	Starten des AEVV-Systems	261
9.6.10	Streaming Server	261
9.6.11	Objekterkennung	262
9.6.12	Gimbalcontroller	265
9.7	Realisierung einer Telemetriestation	274
9.7.1	Gestaltung der graphischen Oberfläche	275
9.7.2	Erzeugung eines virtuellen Access-Points	276
10	Validierung und Verifikation	279
10.1	Testgrundlagen	279
10.1.1	Statischer Test	279
10.1.2	Dynamischer Test	283
10.1.3	Teststufen des V-Modells	286
10.1.4	Modellgetriebenes Testen	288
10.1.5	Reglertests	291
10.1.6	Ermittlung der Ausführungszeiten	294



10.2	Testvorgehen beim ASG	300
10.2.1	Komponententest beim ASG	300
10.2.2	Integrationstest beim AGS	304
10.2.3	Systemtests beim ASG	306
10.2.4	Validierung der Echtzeitanforderungen beim ASG/Dataminer	320
10.3	Testvorgehen beim AEVV	322
10.3.1	Komponententests	323
10.3.2	Integrationstests des AEVV-Systems	327
10.3.3	Systemtests	329
10.4	Testergebnisse	329
10.4.1	Testergebnisse ASG	330
10.4.2	Testergebnisse AEVV	336
11	Ausblick	341
12	Fazit	343
	Literaturverzeichnis	345
A	Anhang	354
A.1	Gantt-Diagramm	354
A.2	Vivado Hardware Design	358
A.3	Debug-Pinzuordnung	360
A.4	Ausführungszeiten Dataminertasks	361



Abbildungsverzeichnis

1.1	Multikopter als „fliegender Postbote“ [Pre]	1
1.2	Systemarchitektur "Mixed-Criticality"	3
1.3	Szenario Objekterkennung	11
1.4	Systemarchitektur mit Objektverfolgung	14
1.5	Wettkampf bei der RoboCup Soccer Standard Platform League 2010 [rc214]	15
1.6	Spielumgebung des Szenarios	16
1.7	Nullposition der Kamerapositioniereinheit	17
1.8	Start des Szenarios	17
1.9	Mögliche Spielsituation im Szenario	18
1.10	Mögliche Spielsituation im erweiterten Szenario	19
2.1	V-Modell	23
2.2	Wichtige Teams im Laufe des Projektes dargestellt auf einem Zeitstrahl mit Hervorhebung einzelner Meilensteine	25
3.1	Trikopter auf der Hannover Messe 2013	33
3.2	Quadrokopter Quadro XL	33
3.3	Hexa- und Octokopter	34
3.4	Definition Koordinatensystem gemäß MPU 9051	37
3.5	Definition Koordinatensystem der MPU 9051	38
3.6	Illustrierung der magnetischen Deklination	40
3.7	Darstellung eines offenen Grundregelkreises	43
3.8	Darstellung eines geschlossenen Grundregelkreises	43
3.9	Einheitssprung	45
3.10	Sprungantworten verschiedener Reglertypen	46
3.11	Optische Täuschung: Dalmatiner	51
3.12	Ablauf der digitalen Objekterkennung.	53
3.13	Der schematische Ablauf der Objektverfolgung.	53
3.14	Das Ausgangsbild.	54
3.15	Bild nach Anwendung des Median-Filters.	55
3.16	Die roten Farbwerte	55
3.17	Binarisierter Rotwert	56
3.18	Die Ausgabe nach der Erosion.	56
3.19	Ausgabe nach der Dilatation.	57
3.20	Erkannte Segmente nach dem BLOB-Algorithmus	57
3.21	Originalbild überlagert mit der Extraktion	58
3.22	Skizze einer Bildaufnahme bei einem <i>unbewegtem</i> Objektiv <i>mit</i> Bildstabilisation	60
3.23	Skizze einer Bildaufnahme bei einem <i>bewegtem</i> Objektiv <i>ohne</i> Bildstabilisation	61



3.24	Skizze einer Bildaufnahme bei einem <i>bewegtem</i> Objektiv <i>mit</i> Bildstabilisation	61
3.25	Funktionsweise der mechanischen Bildstabilisation in einer Kamera.	61
3.26	Funktionsweise eines Gimbal	62
3.27	Funktionsweise eines Gimbal mit Kamera	63
3.28	2-Achs-Gimbal und 3-Achs-Gimbal	64
3.29	Skizze einer Servo-Regelschleife	64
3.30	Darstellung der Rotation von Quaternionen, [Quaa]	70
4.1	Systemarchitektur mit AEVV-System	80
4.2	Use-Case: Systemfunktionen	81
4.3	Use-Case: Multikopter steuern	82
4.4	Use-Case: Videoaufnahme durchführen	87
4.5	Use-Case: Objektverfolgung durchführen	90
4.6	Use-Case: Kamerabild & Telemetriedaten verarbeiten	93
4.7	Use-Case: Erweitertes Szenario	96
6.1	Verlauf der Traceability innerhalb des Projekts	105
6.2	Prozess für die Echtzeitanalyse (blau: Phase; grau: Phasenergebnis)	115
6.3	abstrahierter Ablauf der Höhen- und Lageregelung (funktionale Software)	117
6.4	einfacher Lesezugriff auf einem I ² C-Bus [Eled]	118
6.5	Segmentierung eines 16-Bit Sensorwertes für einen Lesezugriff beim I ² C-Protokoll	119
6.6	abstrahierter Arbeitsablauf des AEVV-Systems	124
6.7	Zusammenhang zwischen Bildwinkel der Kamera und des Abstands zum Objekt (roter Kreis)	125
6.8	Zusammenhang zwischen Abstand d des Objekts und der Winkeländerung $\frac{\alpha}{2}$ der Positioniereinheit (blaues Dreieck)	128
6.9	abstrahierter Arbeitsablauf des AEVV-Systems	129
6.10	Verlauf von c_{Algo} in Abhängigkeit der Distanz beim Bildwinkel von 100°	130
7.1	Sicherheitskritische und zeitkritische Applikationen	135
7.2	Herausforderungen an Multi Core Plattformen	140
7.3	Komponentendiagramm des Gesamtsystems	143
7.4	Komponentendiagramm des Quadropters	144
7.5	Komponentendiagramm des ASGs	145
7.6	Datenflussgraph des ASGs	145
7.7	Komponentendiagramm des AvionicControllers	146
7.8	Datenflussgraph des AvionicControllers	146
7.9	Komponentendiagramm des AEVV-Systems	147
7.10	Datenflussgraph des AEVV-Systems	147
7.11	Komponentendiagramm der Telemetriestation	148
7.12	Blockschaltbild des Industrieboards	154



7.13	Mobius ActionCam 1080p [Mob]	156
7.14	Blockschaltbild Kamera	157
7.15	DYS SMART Gopro BL Gimbal [Hob]	158
7.16	Blockschaltbild Gimbal	160
7.17	Motorströme unterschiedlicher Propeller bei unterschiedlichem Schub	164
7.18	Blockschaltbild Multikopter	165
7.19	Blockschaltbild Verteilerplatine	170
7.20	Anordnung der Rechen- und Speicherkomponenten im Plattformdesign	173
7.21	Anordnung der Sensoren und Aktoren im Plattformdesign	174
8.1	Designprozessschritte in Vivado (entnommen aus [?])	188
8.2	Grafische Elemente in CAMEL-View	191
8.3	Ablaufdiagramm Framework	192
8.4	Ablaufdiagramm Simulationsschleife	194
9.1	Schaltplan Verteilerplatine	196
9.2	Aufbau Verteilerplatine	201
9.3	Ablauf des Bootvorgangs	202
9.4	Sperrung des SLCR_LOCK Registers	203
9.5	Befehls-Aliase zum Setzen der User-LED1 Funktionalität	204
9.6	Fehlerhafter und fehlerfreier Devicetree Eintrag für die Ethernet Schnittstelle des TE0720	206
9.7	Input und Outputs der PPM-IP-Core-Logik	206
9.8	PPM-IP-Core-Logik Ablauf	207
9.9	Komponentendiagramm Dataminer	209
9.10	Zustandsdiagramm der GAM Senderoutine	212
9.11	Fernsteuerung MC-20	214
9.12	Ablauf Sensordatenverarbeitung	217
9.13	BRAM Kommunikation zwischen den Teilsystemen	219
9.14	Ablaufdiagramm der Dataminersoftware	220
9.15	Flugfestes (blau) und erdfestes (rot) Koordinatensystem [Buc]	223
9.16	Mapping Koordinatensystem MPU9150 in Zwischenkoordinatensystem	225
9.17	Komplementärfilter	226
9.18	Komplementärfilter Nickwinkel	228
9.19	Nick- und Roll-Achse in Ruhelage	231
9.20	Nick- und Roll-Achse mit Bewegung	231
9.21	Ausrichtung des Erdmagnetfeldvektors	232
9.22	Kompass 2D: Berechnung der Ausrichtung [STMb]	233
9.23	Kompass 3D: Beispiel $\phi=0$ und $\theta < 0$	233
9.24	Blockschaltbild Flugregelung	237
9.25	Lageregelung in CAMEL-View	238

9.26	Struktureller Aufbau des Kompensationsreglers in CAMEL-View	239
9.27	Höhenregelung in CAMEL-View	240
9.28	CAMEL-View Code des Regelfehlers für die Höhenregelung	241
9.29	Berechnung der Motorstellwert in CAMEL-View	242
9.30	Zustandsautomat der Motorstellwerte	243
9.31	Physikalisches Modell des Quadropters in CAMEL-View	245
9.32	3D-Modell des Quadropters in CAMEL-View	246
9.33	Modell für Code-Generierung	248
9.34	Ablaufplan des selbst implementierten C-Code	248
9.35	I ² C Sequenz zum Schreiben der Drehgeschwindigkeit eines Motors	250
9.36	I ² C Sequenz zum Setzen der Motordrehrichtung im BL-Controller	250
9.37	Statische Struktur des AEVV-Systems (grob)	252
9.38	Ablaufdiagramm für das Starten des AEVV-Controllers	253
9.39	Interner Zustandsautomat im AEVV-Controller	254
9.40	Schnittstelle zum BRAM der Avionik durch Memory-Mapped I/O und einem <code>user-mode</code> I/O-Gerätetreiber	256
9.41	Aufbau des Videopipeline im Streaming Server.	263
9.42	Definition der Winkelbereiche des Gimbals	266
9.43	Schematische Darstellung der Bildung der Regelfehler	267
9.44	Abhängigkeit zwischen Ballabstand, Brennweite und Sensorgröße	269
9.45	Definition der Segmente der Mehrpunkt-Regelung	271
9.46	Übersicht über die Steuerung des Gimbalcontrollers	273
9.47	Graphische Oberfläche (GUI) für die Telemetriestation (Ausschnitt)	276
10.1	Kontrollflussgraph	282
10.2	<i>V-Modell</i>	286
10.3	Übersicht zum modellgetriebenem Testen (nach [HS09])	288
10.4	Ablauf beim Model In The Loop	289
10.5	Polstellen eines stabilen Systems [Orl11]	292
10.6	Polstellen eines grenzstabilen Systems [Orl11]	292
10.7	Polstellen eines instabilen Systems [Orl11]	292
10.8	Analyse des Zeitverhaltens [SZ11]	293
10.9	Analyse der Genauigkeit [SZ11]	293
10.10	Aufbau des WCET-Analyzers ait der Firma AbsInt	296
10.11	Analyse der Ausführungszeiten mittels Messung	298
10.12	<i>Testvorgehen beim ASG</i>	300
10.13	Zweigabdeckung der Komponente <i>LimitationTheta</i>	302
10.14	Verlauf der Position des Quadropters (Testfall STMIL??)	308
10.15	Verlauf der Winkel (Testfall STMIL??)	308
10.16	Aufbau MIL-Test	309



10.17	Peripherie MIL-Test	309
10.18	SPI-Schnittstelle innerhalb des CAMEL-View-Modells für den MIL-Test	311
10.19	Detaillierte SPI-Schnittstelle des Blocks <code>dataProcessing</code> für den MIL-Test	311
10.20	Ablaufdiagramm MatlabScript	313
10.21	Geplanter HIL-Testaufbau	314
10.22	Modifiziertes Konzept für den HIL-Test	315
10.23	Aufbau HIL-Test	316
10.24	Ablaufdiagramm für die SPI-Kommunikation während des HIL-Tests	318
10.25	SPI-Schnittstelle innerhalb des CAMEL-View-Modells für den HIL-Test	319
10.26	Ablauf: Evaluierung der Echtzeitanforderungen	321
10.27	Aufbau des AEVV-Systems	327
10.28	Verlauf der Winkel ohne Filterung der Nullen (Testfall STMIL??)	331
10.29	Verlauf der Winkel mit Filterung der Nullen (Testfall STMIL??)	332
10.30	SIL-Test mit 0.002s als Zeitschritt	332
10.31	SIL-Test mit 0.002s als Zeitschritt	333
10.32	SIL-Test mit 0.002s als Zeitschritt	334
10.33	Stellwerte für den linken Motor	334
10.34	Absoluter und relativer Fehler	335
10.35	Anweisungsüberdeckung des Gimbalcontrollers	337
10.36	Anweisungsüberdeckung des Streaming Servers	337
10.37	Anweisungsüberdeckung des AEVV-Controllers	338
A.1	Ganttogramm des Projekts Avionic Architecture (Seite 1)	354
A.2	Ganttogramm des Projekts Avionic Architecture (Seite 2)	355
A.3	Ganttogramm des Projekts Avionic Architecture (Seite 3)	356
A.4	Ganttogramm des Projekts Avionic Architecture (Seite 4)	357
A.5	Vivado Hardware Design (rechte Seite)	358
A.6	Vivado Hardware Design (linke Seite)	359
A.7	τ_1 Maximale Ausführungszeit der Aktualisierung der Lagewerte	361
A.8	τ_1 Maximale Ausführungszeit um die Lagewerte in das BRAM zu schreiben	361
A.9	τ_2 Maximale Ausführungszeit um die Temperaturen auszulesen	361
A.10	τ_3 Maximale Ausführungszeit um die Spannungen zu ermitteln	362
A.11	τ_4 Maximale Ausführungszeit um das PPM-Signal zu ermitteln	362
A.12	τ_5 Maximale Ausführungszeit für das Schreiben der Daten in das GAM-Daten Frame	362
A.13	τ_6 Maximale Ausführungszeit für das Schreiben der Peripheriedaten in das BRAM	362
A.14	τ_7 Maximale Ausführungszeit für das Setzen des akustischem Signals im GAM-Daten Frame	363
A.15	τ_8 Maximale Ausführungszeit für das Senden des GAM-Frames	363



A.16 τ_9 Maximale Ausführungszeit für das Ein-/Ausschalten der Peripherie . . . 363



Tabellenverzeichnis

1.1	Bewertung von Audiostreaming als nicht sicherheitskritische Task	7
1.2	Bewertung der Lichtshow als nicht sicherheitskritische Task	8
1.3	Bewertung von Wärmebildkamera als nicht sicherheitskritische Task	10
1.4	Bewertung von Objektverfolgung als nicht sicherheitskritische Task	12
1.5	Bewertung der nicht sicherheitskritischen Tasks im Überblick	13
2.1	Grober Aufbau des kompletten Projektes	22
2.3	Rollenverteilung innerhalb der Projektgruppe.	30
3.1	Vor- und Nachteile der Multikopterarten (vgl. [Bü13])	34
3.2	Bandbreiten typischer Audio/Video-Schnittstellen	72
3.3	Vergleich der vorgestellten Funkprotokolle	76
3.4	Übersicht über rechtliche Bedingungen	77
4.1	Use-Case: Flughöhe erhöhen/verringern	83
4.2	Use-Case: Multikopter vorwärts/rückwärts bewegen	83
4.3	Use-Case: Multikopter seitwärts bewegen	84
4.4	Use-Case: Multikopter um eigene Achse rotieren	84
4.5	Use-Case: Sensorwerte aufbereiten	85
4.6	Use-Case: Stellwerte für Motoren berechnen & aufbereiten	85
4.7	Use-Case: Flughöhenregelung aktivieren	86
4.8	Use-Case: Flughöhenregelung deaktivieren	87
4.9	Use-Case: Videoaufnahme aktivieren	88
4.10	Use-Case: Kamerabild bereitstellen	88
4.11	Use-Case: Videoaufnahme deaktivieren	89
4.12	Use-Case: Objektverfolgung aktivieren	90
4.13	Use-Case: Objekt erkennen	91
4.14	Use-Case: Kameraposition nachführen	91
4.15	Use-Case: Objektverfolgung deaktivieren	92
4.16	Use-Case: Sensorwerte aufbereiten	93
4.17	Use-Case: Telemetriedaten an Fernsteuerung senden	94
4.18	Use-Case: Telemetriedaten auf der Fernsteuerung anzeigen	94
4.19	Use-Case: Kamerabild & Telemetriedaten senden	95
4.20	Use-Case: Kamerabild & Telemetriedaten anzeigen	95
4.21	Use-Case: Flugrichtung vorschlagen	97
4.22	Use-Case: Torlinien erkennen	97
4.23	Use-Case: Anzahl geschossener Tore mitzählen	98
4.24	Use-Case: Spielstand senden	98
5.1	Skalierung von Auftrittswahrscheinlichkeiten und Schadenswirkungen	99
5.2	Risikomatrix	99



5.3	Skalierung des Risikos	99
5.4	Risikotabelle	101
6.1	Eindeutige Abkürzungen für die Anforderungserhebung	106
6.2	Anforderungen an das ASG	106
6.3	Anforderungen an das AEVV-System	109
6.4	Anforderungen an die Fernsteuerung	111
6.5	Anforderungen an die Telemetriestation	112
6.6	Anforderungen an die Umwelt	112
6.7	Optionale Anforderungen an das Gesamtsystem	113
6.8	Anforderungen an die Sicherheit	114
6.9	Zusammenfassung der Zeitabschätzungen vom dem ASG	122
6.10	Zusammenfassung der zeitlichen Constraints des AEVV-Systems	128
7.1	Mobius ActionCam Daten	156
7.2	DYS 3-Achs Gimbal Daten	159
7.3	Zusammensetzung Traglast	162
7.4	BMP085 Modes	169
7.5	Vivado Komponenten	176
7.6	Identifizierte Komponenten, die gemappt werden müssen	185
7.7	Identifizierte Schnittstellen, die gemappt werden müssen	186
9.1	Pinzuordnung zwischen der Verteilerplatine (VP) und dem Xilinx Zynq 7020200	
9.2	Interpretation der angezeigten GAM-Daten auf der Fernsteuerung (FS) . .	212
9.2	Interpretation der angezeigten GAM-Daten auf der Fernsteuerung (FS) (Fortsetzung)	213
9.3	Zuordnung der PPM Kanäle zu den Fernbedienungsfunktionen	215
9.4	Interpretation des Fehlerdatums des Dataminers	218
9.4	Interpretation des Fehlerdatums des Dataminers (Fortsetzung)	219
9.5	Auflistung der Tasks und deren Ausführungszeiten	221
9.6	Ermittelte Reglerparameter gemäß Ziegler/Nichols Einstellregeln	247
9.7	Empirisch ermittelte Reglerparameter für C-Code	249
10.1	Überblick der Teilmodelle mit den Anzahlen an Komponenten, Zweigen und Testfällen	302
10.2	Überblick der Integrationstestfälle: AttitudeController und HeightController	305
10.3	Auflistung der zu empfangenen Daten über SPI im MIL-Test	310
10.4	Auflistung der auszutauschenden Daten über SPI im HIL-Test	316
10.5	Testabdeckung ASG	319
10.6	Testabdeckung Sicherheitsanforderungen	320
10.7	Messungen der Ausführungszeiten	321
A.1	Debug-Pinzuordnung zwischen der Verteilerplatine (VP) und dem Xilinx Zynq 7020	360



1 Einleitung

In den Masterstudiengängen Informatik sowie Eingebettete Systeme und Mikrorobotik an der Carl von Ossietzky Universität Oldenburg findet planmäßig im zweiten Semester eine Projektgruppe statt. Im Rahmen dieser Projektgruppe sollen die Teilnehmer im Team eine komplexe Aufgabe aus dem Bereich der Informatik umsetzen. Primär werden Kenntnisse im Projektmanagement und Erfahrungen in der Arbeit als Team erworben. Die Projektgruppe dauert ein Jahr und es wird von den Teilnehmern erwartet sich innerhalb dieses Jahres als geschlossenes Team mit der Lösung dieser Aufgabe zu befassen. Bedingt durch die Teilnehmeranzahl von bis zu zwölf Personen werden hierbei auch Fähigkeiten wie Problemlösungskompetenzen und Konfliktbewältigung trainiert.

Die Projektgruppe Avionic Architecture (PGAA) wird im Sommersemester 2014 und Wintersemester 2014/15 von der Abteilung Eingebettete Hardware- und Software-Systeme unter der Leitung von Prof. Dr.-Ing. Wolfgang Nebel und Prof. Dr. rer. nat. Achim Rettberg in Zusammenarbeit mit dem OFFIS e. V.¹ organisiert. Als Betreuer sind der Projektgruppe Malte Metzdorf, Henning Schlender und Sören Schreiner zugeordnet. Diese Projektgruppe richtet sich in erster Linie an Studenten der Studienrichtung Eingebettete Systeme und Mikrorobotik. In diesem Studiengang befinden sich sowohl Absolventen des Bachelorstudiengangs Informatik als auch anderer gleichwertiger Studiengänge wie z.B. Elektrotechnik, Technische Informatik oder Mikrosystemtechnik.

1.1 Motivation

Multikopter sind heutzutage in vielen universitären und industriellen Einrichtungen Gegenstand aktueller Forschungsgebiete. Die Verwendung unbemannter Flugobjekte bietet neue Anwendungsmöglichkeiten und kann die Kosten, welche ansonsten durch kostspielige Zusatzhardware entstehen, reduzieren.

Das Einsatzgebiet erstreckt sich hierbei von fliegenden Postboten (s. Abbildung 1.1) oder Pizzalieferanten über die Vermessung, Kartierung und Inspektion von Geländeflächen und industrieller Anlagen bis hin zum fliegenden Kameramann für professionelle Filmaufnahmen aus der Luft [rq, cf, SPE, Gmba].



Abbildung 1.1: Multikopter als „fliegender Postbote“ [Pre]

Neben den kommerziellen Anwendungen

ist auch im Hobbybereich ein steigender Trend in den Multikopteranwendungen zu erkennen. Dies wird durch Open-Source-Projekte und einschlägige Foren, welche einen den Einstieg in die Welt der unbemannten Flugobjekte erleichtern, weiter begünstigt. Gerade

¹Oldenburger Forschungs- und Entwicklungsinstitut für Informatik



im Bereich der Luftbildaufnahmen ist ein wachsendes Interesse zu verzeichnen. Nicht zuletzt durch Crowdfunding-Projekte, welche mit Hilfe eines Multikopters und einer hochauflösenden Kamera unvergleichbare Videos aus unterschiedlichsten Blickwinkeln aufzeichnen [Ind].

1.2 Die Aufgabenstellung

In Rahmen der PGAA soll ein „mixed critical“ System entwickelt werden, bei dem eine sicherheitskritische und eine nicht sicherheitskritische Anwendung gleichzeitig auf der selben Plattform verarbeitet werden. Als Basis hierfür dient ein Multikopter, für den eine sicherheitskritische Avionik kombiniert mit einer rechenintensiven nicht sicherheitskritischen Anwendung umgesetzt werden sollen.

Der Systementwurf soll hierbei modellgetrieben erfolgen. Der Vorteil der modellgetriebenen Entwicklung in diesem Kontext ist vor allem die Qualitätssicherung. Das Systemmodell kann in der Simulation getestet werden. Näheres zum modellgetriebenen Systementwurf wird in der späteren Architekturphase erläutert.

Sicherheitskritische Anwendung

Als sicherheitskritische Anwendung soll die „Avionik“ eines Multikopters entwickelt werden. Der Begriff „Avionik“ leitet sich aus dem lateinischen Wort „Avis“ (dt. Vogel) und „Elektronik“ ab, und bezeichnet im allgemeinen Sprachgebrauch die Gesamtheit aller elektronischen und elektrischen Systeme eines Fluggerätes.

In diesem Projekt ist „Avionik“ konkreter zu verstehen als die Gesamtheit der Subsysteme zur Steuerung und Regelung des Multikopters.

Die Avionik verarbeitet die Signale der Fernbedienung und setzt diese in die gewünschte Fluglage und den gewünschten Auftrieb des Multikopters um. Hierzu werden die Rotoren auf Basis verschiedener Sensorwerte (Gyroskope, Beschleunigungssensoren, Kompass) entsprechend angesteuert (siehe Kapitel 3.1).

Die Ansteuerung der Motoren ist höchst sicherheitskritisch, da ein jederzeit stabiles und kontrollierbares Flugverhalten Voraussetzung ist, um potentielle Beschädigungen der Umgebung, aber auch des Multikopters selbst zu verhindern. Vor allem Menschen in der Umgebung des Multikopters dürfen in keinster Weise gefährdet werden.

Nicht sicherheitskritische Anwendung

Die nicht sicherheitskritische Anwendung kann frei gewählt werden. Als Einschränkung ist eine hohe Rechenintensität gefordert. Im Nachfolgenden Kapitel wird die Auswahl der nicht sicherheitskritischen Anwendung genauer erläutert.

In der Abbildung 1.2 ist eine mögliche grobe Systemarchitektur dargestellt. Auf der obersten Ebene wird zunächst die Funktionalität des Systems in Blöcken dargestellt. Die beiden Blöcke "Avionic" und "Non-Critical Application" sind noch sehr abstrakt und werden in später folgenden Kapiteln weiter verfeinert.

Unterhalb der funktionalen Übersicht liegt die "Processing Elements"-Ebene. Da in der Spezifikationsphase noch keine unnötigen Einschränkungen bezüglich der zu verwendenden Hardwarekomponenten gemacht werden sollen, wird diese Ebene erst in späteren Kapiteln weiter verfeinert.

Auf der untersten Ebene sind die Sensoren und Motoren dargestellt. Auch hier sollen noch keine Einschränkungen gemacht werden. In Kapitel 1.3.5 wird auch diese Ebene schrittweise verfeinert und es wird dargestellt welche Sensoren und Motoren genau verwendet werden.

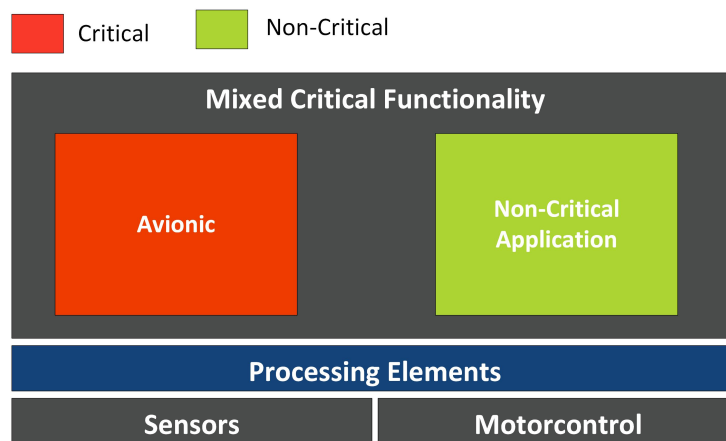


Abbildung 1.2: Systemarchitektur "Mixed-Criticality"

Diese noch sehr abstrakte Systemarchitektur, die sich nach Interpretation der Aufgabenstellung heraus kristallisiert hat, ermöglicht mit der Verfeinerung eine schrittweise Spezifikation einzelner Teilsysteme, ohne das Gesamtsystem hierbei aus den Augen zu verlieren.

1.3 Auswahl der nicht kritischen Anwendung

In diesem Abschnitt werden mögliche nicht sicherheitskritische Tasks vorgestellt und deren Eignung zur Umsetzung in der PGAA systematisch bewertet. Für die nachfolgend beschriebenen Kriterien wird eine 3-stufige Bewertung (gut, mittel und schlecht) vorgenommen.

1.3.0.1 Kosten

Die Projektgruppe hat nur ein beschränktes Budget zur Verfügung, weshalb auch die Kosten nicht vernachlässigt werden dürfen. Die Anschaffungskosten der notwendigen Hardware-



und Software-Komponenten werden zwischen 200€ - 300€ als "mittel" eingestuft. Liegen die Anschaffungskosten unter 200€ wird dies als "gut" bewertet, über 300€ als "schlecht".

1.3.0.2 Gewicht

Das Gewicht der zusätzlich benötigten Peripherie ist zum einen durch die Tragkraft des Multikopters limitiert; zum anderen bedeutet ein höheres Gewicht immer auch eine Verkürzung der Flugzeit bei gleicher Akkukapazität. Traglasten von ca. 500 g sind für Multikopter typischerweise unproblematisch. Ein Gewicht unter 400 g wird als "gut" bewertet, zwischen 400 g und 600 g als "mittel" und mehr als 600 g als "schlecht".²

1.3.0.3 Hardware-Schnittstellen

Die Hardwarekomponenten für die nicht kritische Task werden an die verarbeitende Elektronik angeschlossen. Folgende Eigenschaften werden hierbei positiv bewertet:

1. Die Schnittstellen sind standardisiert
2. Die Schnittstellen sind leicht zu implementieren
3. Es sind verschiedene Schnittstellen verfügbar, sodass im Falle von unvorhergesehenen Problemen eine Alternative genutzt werden kann

Ist keine der oben genannten Eigenschaften erfüllt, wird dies als "schlecht" bewertet, bei ein oder zwei erfüllten Eigenschaften als "mittel", und wenn alle erfüllt sind als "gut".

1.3.0.4 Software Support

Je nach Applikation können ggf. vorhandene Bibliotheken genutzt werden, was den Implementierungsaufwand u. U. reduziert. Wenn ein Großteil des Codes manuell geschrieben werden muss, wird dies als "schlecht" bewertet. Je mehr **IPs** (IP: "Intellectual Property") genutzt werden können, desto besser ist die Bewertung der jeweiligen Task.

1.3.0.5 Power

Die zusätzlich benötigten Hardwarekomponenten verbrauchen Energie. Wenn die Energie über den Akku des Multikopters bereit gestellt werden muss, verringert sich dadurch die maximale Flugzeit. Da der Einfluss auf die Flugzeit allerdings stark abhängig vom später gewählten Gesamtsystem und der Kombination von Rotoren und Motoren sowie dem Gesamtgewicht abhängig ist, erfolgt die Bewertung über den Vergleich der verschiedenen Tasks untereinander. Die Task mit dem höchsten Energiebedarf wird "schlecht" bewertet, die mit dem niedrigsten "gut".

1.3.0.6 Empfindlichkeit

Bei einem Multikopter treten im Flugbetrieb starke Vibrationen auf. Bewertet wird, wie

²Viele Multikopter werden für Foto- bzw. Videoaufnahmen eingesetzt. Sehr leichte Kameras und sogenannte Gimbals zur Stabilisierung des Kamerabildes wiegen in Summe ca. 500 g. Deswegen wurde dieses Gewicht als unproblematisch angenommen.

stark der Einfluss dieser Vibrationen auf die nicht kritische Anwendung ist. Wenn die Vibrationen keinen oder kaum Einfluss auf die nicht kritische Anwendung haben, wird dies "gut" bewertet. Wenn zwar ein Einfluss da ist, der aber ausgeglichen werden kann, wird dies "mittel" bewertet. Ein extremer Einfluss von Vibrationen, der zu schwerwiegenden Problemen bei der nicht kritischen Anwendung führen kann wird "schlecht" bewertet.

1.3.0.7 Rechenintensität

Ein Projektgruppenziel ist es, eine möglichst rechenintensive nicht kritische Anwendung umzusetzen, um die Leistungsfähigkeit des "Mixed-Criticality"-Systems gut herausarbeiten zu können. Eine hohe Rechenintensität wird daher positiv bewertet.

1.3.0.8 Komplexität

Da die Arbeitszeit der Projektgruppe auf einen kurzen Zeitraum limitiert ist, wird auch die Komplexität bzw. Machbarkeit der Umsetzung einer nicht kritischen Anwendung im Auswahlprozess stark berücksichtigt. Bewertet wird die Abschätzung des Entwicklungsaufwandes. Folgende Aspekte werden hierzu berücksichtigt:

1. zeitlicher Umfang des Entwicklungsprozesses
2. benötigtes Fachwissen zur Umsetzung
3. Parallelisierungsmöglichkeiten im Entwicklungsprozess

1.3.0.9 Effekt

Die Arbeitsmoral in einer Projektgruppe hängt sehr stark mit dem Entwicklungsziel und der sich daraus ergebenden Motivation zusammen. Mit dem Kriterium "Effekt" soll auch dieser Aspekt im Auswahlprozess berücksichtigt werden. Bewertet wird hierbei, ob die nicht kritische Anwendung der Projektgruppe subjektiv "gefällt". Dieses Kriterium unterliegt daher einer sehr subjektiven Bewertung.

1.3.0.10 Risiko

Der Entwicklungsprozess beinhaltet viele Risiken, z. B. :

1. Überschreitung des zeitlichen Rahmens
2. Überschreitung des finanziellen Rahmens
3. vollständiges Scheitern von Teilprojekten

Bewertet wird, ob und wie flexibel auf Fehlentwicklungen reagiert werden kann. Folgende Aspekte werden hierbei positiv bewertet:

1. gute Abschätzbarkeit der Risiken im Voraus
2. gute Ausweichmöglichkeiten bei Scheitern von Teilprojekten
3. Wiederverwendbarkeit von Hardware beim Ausweichen auf Alternativen
4. kleines Risiko für das vollständige Scheitern von Teilprojekten (subjektiv geschätzt)



1.3.1 Audio-Streaming

1.3.1.1 Idee Das Ziel ist es eine Audio-Botschaft möglichst schnell und einfach zu verbreiten, vor allem in Umgebungen, die mit anderen Mitteln nur schwer zugänglich sind.

1.3.1.2 Konzept Über eine Funkverbindung wird ein Audio-Stream von einer Basisstation zu dem Multikopter übertragen. Der Multikopter ist mit einem Lautsprecher ausgestattet und dient als Wiedergabeplattform. Für die Umsetzung werden als Kernkomponenten ein Lautsprecher, zwei Funkmodule und entsprechende Audio-Hardware benötigt.

1.3.1.3 Bewertung Die benötigte Hardware ist durchaus verbreitet und nicht besonders kostenintensiv. Es gibt für die digitale Verarbeitung von Audio-Signalen eine gute Softwareunterstützung und standardisierte Hardwareschnittstellen. Das größte Problem ist das hohe Gewicht und die hohe Leistungsaufnahme, die hauptsächlich durch den leistungsfähigen Lautsprecher und ggf. Verstärker verursacht werden. Es gibt bereits ausgereifte Konzepte und Softwarebibliotheken für die Verarbeitung von Audio-Streams. Daher ist die Komplexität der zu entwickelnde Software eher gering einzuschätzen. Der Schwerpunkt liegt hier auf der korrekten Verwendung und Kombination der Hardware.

Es wird eine Beispielkonfiguration betrachtet, welche neben einem SoundLink Mini Speaker System von Bose aus zwei XBee-Modulen zur kabellosen Datenübertragung besteht. Als Audio-Stream wird eine MP3-Signal mit 196 Kbps gewählt. Die maximale Bandbreite eines XBee-Module wird mit 250 Kbps angegeben. Diese und weitere in der Tabelle 1.1 verwendeten Daten stammen aus den Datenblättern bzw. Beschreibungen der Komponenten [Gmbc, Gmbd].



Tabelle 1.1: Bewertung von Audiostreaming als nicht sicherheitskritische Task

Kriterium	Abschätzung	Bw.
Kosten	<ul style="list-style-type: none"> • Lautsprecher: ca. 200 € (Bose SoundLink Mini Speaker System) • Funkverbindung: 2x XBee ca. 52 € 	○
Gewicht	<ul style="list-style-type: none"> • Lautsprecher: 670 g inkl. Akku • Halterung: ca. 100 g • Zusatzhardware: ca. 50 g 	–
Hardware Schnittstellen	<ul style="list-style-type: none"> • Ansteuerung Lautsprecher: 3,5 AUX-Eingang oder Bluetooth • Ansteuerung XBee: UART 	○
Software Support	<ul style="list-style-type: none"> • Bibliotheken für die Audioverarbeitung und XBee Anwendungen sind vorhanden 	+
Power	<ul style="list-style-type: none"> • Energieversorgung des Lautsprechers über integrierten Akku bis zu 7 Stunden • XBee: max. 50mA @ 3,3V 	+
Empfindlichkeit	<ul style="list-style-type: none"> • unempfindlich gegenüber Vibrationen 	+
Rechenintensität	<ul style="list-style-type: none"> • MP3-Stream auf dem Processor des Multikopter decodieren und weiterverarbeiten 	○
Komplexität	<ul style="list-style-type: none"> • gut im Projektzeitraum umzusetzen • gut zu parallelisieren • Fachwissen zu MP3 dekodieren und Wireless-Datenübertragung benötigt 	+
Effekt	<ul style="list-style-type: none"> • viel positives Feedback bei Vorschlag dieser Projektidee 	+



Tabelle 1.1: Bewertung von Audiostreaming als nicht sicherheitskritische Task (Fortsetzung)

Kriterium	Abschätzung	Bw.
Risiko	<ul style="list-style-type: none"> • Risiken hinsichtlich benötigter Lautstärke sind schwer im Voraus abzuschätzen • keine gute Ausweichmöglichkeiten: wenn Ausgangsleistung zu gering, Verstärker verwenden; Gewicht und Energieverbrauch steigen 	–

1.3.2 Lightshow

1.3.2.1 Idee Multikopter fliegt Manöver und macht dabei eine Lightshow.

1.3.2.2 Konzept Für die Lightshow wird der Multikopter mit RGB-LEDs ausgestattet. Der Multikopter fliegt autonom eine vordefinierte Strecke und steuert dabei die LEDs mit unterschiedlicher Frequenz an, sodass diese in verschiedenen Farben passend zum Manöver leuchten.

1.3.2.3 Bewertung Der Hauptkostenpunkt sind die LED-Streifen mit ca. 50 € pro 540 mm [SE]. Das Gewicht der LED-Streifen und der zusätzliche Hardwareaufwand zur Ansteuerung ist gering. Kritisch ist jedoch das Sicherheitsrisiko, da der Multikopter autonom Manöver fliegt und die nicht sicherheitskritische Task in die Flugalgorithmen eingreift. Somit wird die Task sicherheitskritisch.

Tabelle 1.2: Bewertung der Lichtshow als nicht sicherheitskritische Task

Kriterium	Abschätzung	Bw.
Kosten	<ul style="list-style-type: none"> • LED-Streifen: 50 € pro 54 cm 	+
Gewicht	<ul style="list-style-type: none"> • LED-Streifen: <100 g 	+
Hardware Schnittstellen	<ul style="list-style-type: none"> • Ansteuerung der RGB-LEDs über drei Signalleitungen und einer Versorgungsleitung 	+
Software Support	<ul style="list-style-type: none"> • es wird kein besonderer Software Support benötigt 	+
Power	<ul style="list-style-type: none"> • Leistungsaufnahme: 12 W pro 54 cm-LED-Streifen 	–

Tabelle 1.2: Bewertung der Lichtshow als nicht sicherheitskritische Task (Fortsetzung)

Kriterium	Abschätzung	Bw.
Empfindlichkeit	<ul style="list-style-type: none"> Vibrationen haben keinen Einfluss 	+
Rechenintensität	<ul style="list-style-type: none"> gering: nur Scheduling der LED-Ansteuerung notwendig 	-
Komplexität	<ul style="list-style-type: none"> sehr geringe Komplexität Ansteuerung der LEDs nach vorgegebenem Muster Weiterreichen des Manövers (Positionsdaten) an die Avionic 	+
Effekt	<ul style="list-style-type: none"> mit nur einem Multikopter wirkt es eher langweilig -> kaum Motivation 	-
Risiko	<ul style="list-style-type: none"> LEDs könnten bei Tageslicht zu dunkel leuchten 	+

1.3.3 Wärmebildkamera

1.3.3.1 Idee Das Ziel ist es, Brandherde in einem Waldgebiet frühzeitig zu erkennen und die Feuerwehr zu warnen.

1.3.3.2 Konzept An dem Multikopter wird über eine Positionierungseinheit eine Wärmebildkamera angebracht. Wenn ein Brandherd erkannt wird, wird über eine Funkschnittstelle ein Warnsignal mit den Brand-Koordinaten gesendet.

1.3.3.3 Bewertung Es gibt speziell für Multikopter ausgelegte Wärmebildkameras, die keinen eigenen Akku besitzen und nur wenige 100 g wiegen. Diese speziellen Kameras sind unempfindlich gegenüber Vibrationen. Neben der Kamera werden noch zwei Funkmodule und eine Kamera-Positionierungseinheit benötigt, welche das Gesamtgewicht weiter erhöhen. Die Komplexität der Task kann beliebig angepasst werden (erkennen von Objekten, erhöhen der Auflösung wenn möglich, etc.) und als Bildverarbeitungssoftware kann die Open-Source-Bibliothek OpenCV verwendet werden [ope].

Beispielhaft wird hier die Wärmebildkamera thermoIMAGER TIM 160 von der Firma Micro-Epsilon, zwei XBee Module und ein 3-Achs-Gimbal von PowerParts UAV als mögliche Hardwarekonfiguration angenommen [Hin, KG].



Tabelle 1.3: Bewertung von Wärmebildkamera als nicht sicherheitskritische Task

Kriterium	Abschätzung	Bw.
Kosten	<ul style="list-style-type: none"> • Wärmebildkamera ca. 6600 € • Funkverbindung: 2x ZigBee ca. 52 € • Gimbal: ca. 230,€ 	–
Gewicht	<ul style="list-style-type: none"> • Kamera: 250 g inkl. Objektiv • Halterung: ca. 300 g • Zusatzhardware: 50 g 	–
Hardware Schnittstellen	<ul style="list-style-type: none"> • Ansteuerung der Sollstellung des Gimbals • XBee 	+
Software Support	<ul style="list-style-type: none"> • Videobearbeitung mit OpenCV • Bibliotheken für XBee Anwendungen sind vorhanden 	○
Power	<ul style="list-style-type: none"> • Kamera: max. 12 W • Gimbal: abhängig von der Regelung • XBee: max. 50mA @ 3,3V 	○
Empfindlichkeit	<ul style="list-style-type: none"> • unempfindlich gegenüber Vibrationen 	+
Rechenintensität	<ul style="list-style-type: none"> • Auswerten des Bildes • Objekterkennung 	+
Komplexität	<ul style="list-style-type: none"> • OpenCV Objekterkennung • Gimbal- und Kameraschnittstellen Programmierung 	–
Effekt	<ul style="list-style-type: none"> • lässt sich gut präsentieren • Anwendungsbereiche sind immer aktuell 	+
Risiko	<ul style="list-style-type: none"> • Ohne Sponsoren ist diese Anwendung nicht umsetzbar 	○

1.3.4 Objekterkennung

1.3.4.1 Idee Eine weitere mögliche nicht sicherheitskritischer Task ist die Erkennung von Objekten mit Hilfe einer am Multikopter angebrachten Kamera. Durch die Kamera werden dabei Bilder der Umgebung aufgenommen und unter Verwendung diverser Bilderverarbeitungsmethoden analysiert. Daraus ergeben sich zahlreiche Anwendungsmöglichkeiten, wie z.B. die Stau-Erkennung im Straßenverkehr oder das Scannen von Gebäuden zur Erstellung von 3D-Modellen. Ein weiteres Szenario wäre der Einsatz des Multikopter als „Fliegender Schiedsrichter“ bei Sportveranstaltungen.

1.3.4.2 Konzept Die Kamera erkennt ein zuvor definiertes Objekt und verfolgt dieses Unabhängig von den Bewegungen des Multikopters. Der Multikopter wird weiterhin von einem Piloten gesteuert. Die Videosequenzen werden über eine Funkverbindung an eine Bodenstation gesendet, welche das Live-Bild an weitere Endbenutzer weiterleitet.

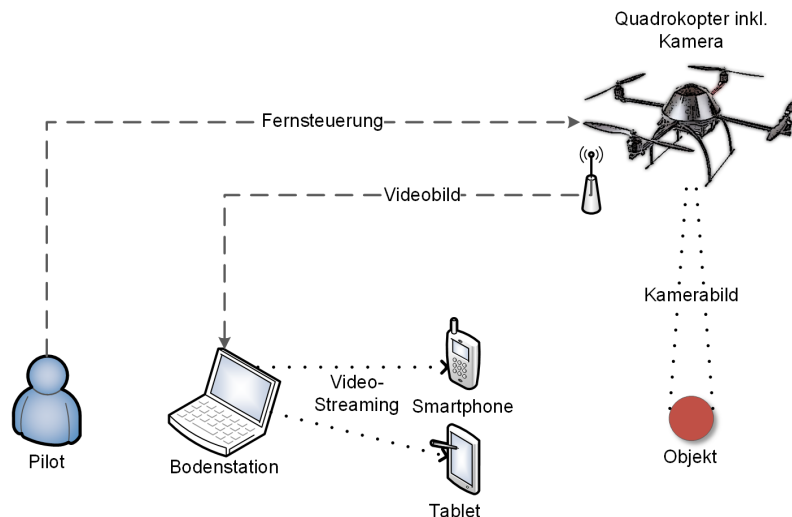


Abbildung 1.3: Szenario Objekterkennung

1.3.4.3 Bewertung Wie zuvor bei der Wärmebildkamera wird auch hier eine Positionierungseinheit für die Kamera, welche einen Live-Video-Out besitzen muss, benötigt. Für die Bildverarbeitung steht die Open-Source Bibliothek OpenCV zur Verfügung. Es ist u.a. eine Version für Embedded Systeme vorhanden, die diverse Algorithmen zur Objekterkennung enthält. Diese Algorithmen sind i.d.R. sehr rechenintensiv und erfordern eine hohe Performance. Dadurch entsteht das Risiko, dass die Rechenleistung des im Multikopter verbauten Prozessors nicht ausreicht. In diesem Fall besteht die Möglichkeit, die Objekterkennung auf die Bodenstation auszulagern.

Es wird eine Beispielkonfiguration betrachtet, bei der ein 3-Achsen-Gimbal, zwei XBee Module und als Kamera die Action Cam 100 von Rollei zum Einsatz kommen [Inca].



Tabelle 1.4: Bewertung von Objektverfolgung als nicht sicherheitskritische Task

Kriterium	Abschätzung	Bw.
Kosten	<ul style="list-style-type: none"> • Kamera: ca. 250 € • Funkverbindung: 2x ZigBee ca. 52 € • Gimbal: ca. 230 € 	–
Gewicht	<ul style="list-style-type: none"> • Kamera: 74 g ohne Batterien • Halterung: ca. 300 g • Zusatzhardware: ca. 50 g 	○
Hardware Schnittstellen	<ul style="list-style-type: none"> • Ansteuerung der Sollstellung des Gimbals • ZigBee 	○
Software Support	<ul style="list-style-type: none"> • Videobearbeitung mit OpenCV • Bibliotheken für XBee Anwendungen sind vorhaben 	○
Power	<ul style="list-style-type: none"> • Kamera: Versorgung über internen Akkumulator • Gimbal: abhängig von der Regelung 	○
Empfindlichkeit	<ul style="list-style-type: none"> • unempfindlich gegenüber Vibrationen 	+
Rechenintensität	<ul style="list-style-type: none"> • Auswerten des Bildes • Objekterkennung 	+
Komplexität	<ul style="list-style-type: none"> • OpenCV Objekterkennung • Gimbal- und Kameraschnittstellen Programmierung 	–
Effekt	<ul style="list-style-type: none"> • lässt sich gut präsentieren • Anwendungsbereiche sind immer aktuell 	+
Risiko	<ul style="list-style-type: none"> • Bei zu hohem Rechenaufwand der Bildverarbeitung muss diese ausgelagert werden. 	+



1.3.5 Fazit

Um für eine Abstimmung möglichst alle Vor- und Nachteile der jeweiligen nicht kritischen Anwendungen berücksichtigen zu können, werden in Tabelle 10.6 alle Bewertungen übersichtlich dargestellt.

Tabelle 1.5: Bewertung der nicht sicherheitskritischen Tasks im Überblick

Kategorie	Kosten	Gewicht	Schnittstellen	Softw. Support	Power	Empfindlichkeit	Rechenintensität	Komplexität	Effekt	Risiko
Musik Streaming	○	-	+	+	-	+	-	+	○	-
Lightshow	+	+	+	+	-	+	-	+	-	+
Wärmebild	-	-	+	○	○	+	+	-	+	○
Objektverfolgung	-	○	○	○	○	+	+	-	+	+

1.3.5.1 Musikstreaming

Als Hauptkritikpunkt sticht das hohe Risiko hervor. Die Ausgewählten Lautsprecher haben zwar einen eigenen Akku, mit dem die Lautsprecher lange genug betrieben werden können, doch falls die Lautstärke zu gering ist, muss zusätzlich eine Endstufe verbaut werden. Diese bringt zusätzliches Gewicht mit und verbraucht darüber hinaus Leistung, die vom Akku des Multikopters bereitgestellt werden muss. Beides beeinträchtigt die Flugdauer des Multikopters, sodass man vermutlich einen Trade-Off zwischen Flugdauer und Lautstärke der Audioübertragung in Kauf nehmen muss. Abschließend kommt hinzu, dass es keine Alternativenanwendung gibt, falls die Idee des Streamings im späteren Verlauf der Entwicklung unerwartet nicht umsetzbar ist.

1.3.5.2 Lightshow

Die Lightshow ist generell ungeeignet, da das fliegen von Flugmanövern diese Anwendung zu einer sicherheitskritischen macht. Zudem ist die benötigte Rechenleistung sehr gering, was ein sehr wichtiges Kriterium ist. Außerdem ist der optische Vorzeigeeffekt, der das Publikum ins Staunen versetzen sollen, sehr gering. Eine Lightshow ist nur dann eine Alternative, wenn die Lightshow in Verbindung mit mehreren Multikoptern im Schwarm umgesetzt wird.

1.3.5.3 Wärmebildkamera Die Aufgabe, den Multikopter in Waldgebieten zur Erkennung von Brandherden einzusetzen (siehe Kapitel 1.3.3), zeigt mehrere Probleme auf. Erstens lässt sich ein solches Szenario schwierig simulieren bzw. nachbilden und zweitens

sprengt der Anschaffungspreis für eine Wärmebildkamera das Budget der Projektgruppe. Sofern die Kamera keinen eigenen Akkumulator besitzt, muss deren Leistungsaufnahme von den Akkumulatoren des Multikopters gespeist werden. Bei dem bereits genannten Beispiel der thermoIMAGER TIM 160, handelt es sich dabei um ca. 12W.

1.3.5.4 Objekterkennung

Die letzte zu betrachtende Aufgabe ist die Objekterkennung im Flug (s. Kapitel 1.3.4). Für diese Herausforderung ergibt sich das größte Spektrum an konkreten Anwendungen und Alternativlösungen. Außerdem kristallisieren sich Arbeitspakete aus diversen Bereichen wie zum Beispiel Bildverarbeitung, Nachrichtentechnik und Datenaufbereitung heraus. Falls eine Online-Auswertung der Bilder nicht möglich ist, kann dies offline auf einer Basisstation durchgeführt werden.

Ein weiterer Vorteil ist, dass es Kameras mit separaten Akkumulatoren gibt, sodass die Akkumulatoren des Multikopters nicht zusätzlich belastet werden.

1.3.5.5 Ergebnis

Unter Berücksichtigung aller oben genannten potentiellen Risiken wurde eine Abstimmung durchgeführt. Hierbei hat sich eine deutliche Mehrheit der Projektgruppenmitglieder für die Objektverfolgung als nicht kritische Anwendung entschieden.

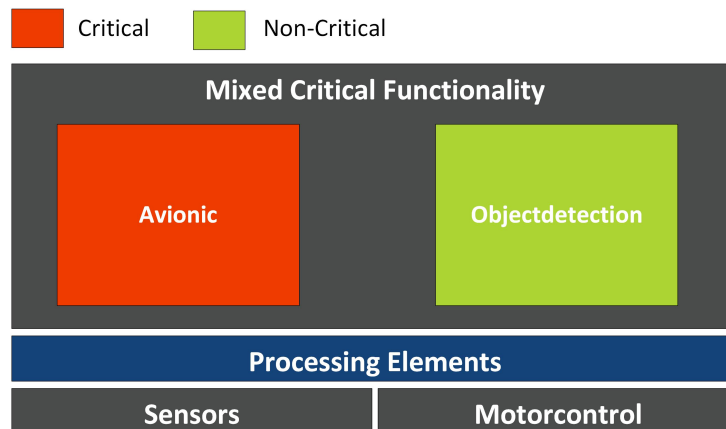


Abbildung 1.4: Systemarchitektur mit Objektverfolgung

Das in Kapitel 1.2 eingefügte Architektur-Diagramm kann daher bereits verfeinert werden (s. Abbildung 1.4). Die Objektverfolgung wird dem Diagramm als nicht kritische Anwendung hinzugefügt.

1.4 Das Szenario

Zur Demonstration der Funktionalität des Gesamtsystems dient das nachfolgend beschriebene Szenario. Als einen geeigneten Anwendungsbereich für diese Vision bietet sich Roboterfußball an (s. Abbildung 1.5). Mittlerweile existieren schon erste internationale Wettkämpfe, beispielsweise der RoboCup oder die jährlich stattfindende Weltmeisterschaft durch die Federation of International Robot-Soccer Association (FIRA) [rc214]. Damit eine besondere Liveübertragung von dem Fußballspiel den Zuschauern oder den Spielerteams zur Verfügung gestellt werden kann, bietet sich eine „fliegende Kamera“ an.

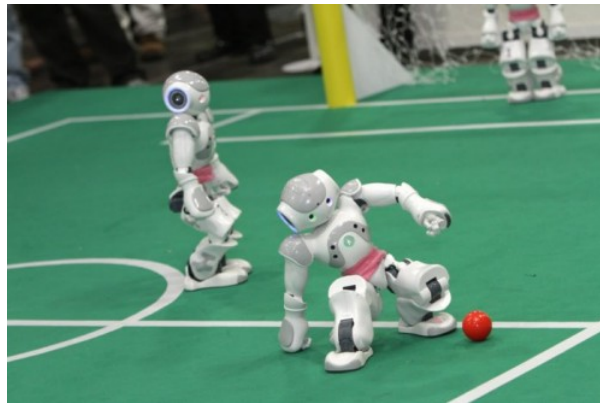


Abbildung 1.5: Wettkampf bei der RoboCup Soccer Standard Platform League 2010 [rc214]

Der Fokus im Grundszenario liegt in der Objekterkennung und einer autonomen Kameraführung über eine Kamerapositioniereinheit, die am Multikopter befestigt ist. Dieser wird am Seitenrand von einem Piloten ferngesteuert. Des Weiteren wird eine Liveübertragung des Kamerabildes an eine Telemetriestation³ gesendet. Im erweiterten Szenario wird das Fußballspiel konkretisiert und es werden Torlinien hinzugefügt. Der Multikopter fungiert in diesem Fall zusätzlich als Torlinienrichter. Ebenfalls soll dem Piloten über einen Live-stream eine Flugrichtung vorgeben werden, um den Ball optimal verfolgen zu können.

Bei diesem Szenario handelt es sich bei dem zu verfolgendem Objekt um einen Ball. Um die Umwelteinflüsse zu minimieren, spielt sich der Ablauf in einem geschlossenen, gut beleuchteten Raum ab. Damit ein ausreichend großes Sichtfeld der Kamera aufgebaut werden kann, muss der Multikopter eine gewisse Flughöhe erreichen können. Folglich bietet sich als Räumlichkeit eine Sporthalle an.

³Diese Telemetriestation repräsentiert eine Station, welche auf einem Display neben den Telemetriedaten auch die Liveübertragung darstellt.

Innerhalb der Sporthalle wird ein Spielfeld definiert. Dieses Feld repräsentiert genau die Fläche, in der die Objektverfolgung stattfindet und der Ball durch zwei Spieler hin- und her gespielt wird (s. Abbildung 1.6)⁴.

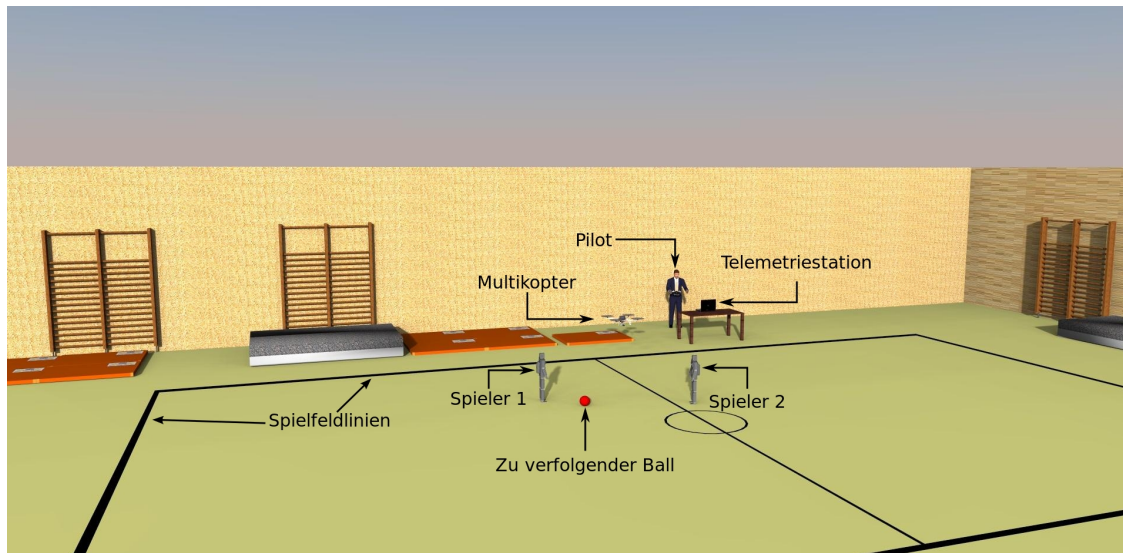


Abbildung 1.6: Spielumgebung des Szenarios

Während der gesamten Ballverfolgung soll sich das Objekt stets im Zentrum des Kamerabildes befinden. Dies wird durch eine am Multikopter befestigten Kamerapositioniereinheit erreicht.

Durch das Sichtfeld der Kamera wird eine „Tracking Area“ gebildet. Sofern sich der Ball innerhalb dieses Bereichs befindet, kann der Ball erkannt und zur Fokussierung verfolgt werden (s. Abbildung 1.8).

Während der Durchführung des Szenarios können im Spielablauf folgende Situationen auftreten:

Situation 1 Ball sichtbar: Die Kamera detektiert den zu verfolgenden Ball und wird autonom nachgestellt, sodass der Ball fortwährend im Fokus des Bildes ist.

Situation 2 Ball ist kurzzeitig verdeckt: Da ein kurzzeitiges Verdecken des Balls z. B. durch einen Fußballroboter sehr wahrscheinlich ist, wird die Kamera in diesem Fall ihre ursprüngliche Ausrichtung zunächst behalten. Wenn nach einer bestimmten Zeit der Ball nicht wieder auffindbar ist, tritt Situation 3 ein.

Situation 3 Ball nicht im Bild sichtbar: Die Kamera sucht autonom das Spielfeld nach dem Ball ab, bis dieser gefunden wird. Sobald dieser erneut detektiert werden konnte, wird die Objektverfolgung aufgenommen und es tritt Situation 1 ein.

⁴Zur graphische Darstellung werden als Multikopter ein Quadrocopter und als Kamerapositioniereinheit ein Gimbal mit drei Achsen gewählt

Das Szenario wird gestartet, indem sich der Multikopter am Seitenrand in einer ausreichenden Flughöhe befindet. Dabei muss eine freie Sicht auf den Ball gewährleistet werden. Zu diesem Zeitpunkt ist die Videoaufnahme und Objektverfolgung noch deaktiviert. Die bedeutet, dass die Kamera sowie die Kamerapositioniereinheit stromlos geschaltet sind. Ist die Flughöhe erreicht, wird zuerst die Bildaufnahme durch den Piloten aktiviert, die Kamerapositioniereinheit fährt autonom in Nullposition und behält diese unabhängig von der Multikopterlage bei. In der Nullposition ist die Kamerapositioniereinheit in 45° Winkel zum Lot in Flugrichtung ausgerichtet (s. Abbildung 1.7).

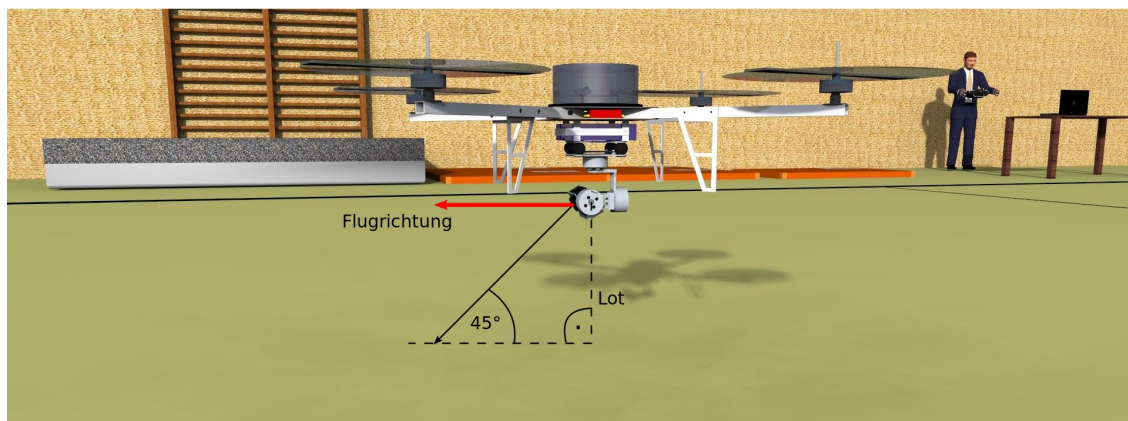


Abbildung 1.7: Nullposition der Kamerapositioniereinheit

Des Weiteren wird die Kamera eingeschaltet, sodass das aufgenommene Bild an die Telemetriestation übertragen wird. Danach schaltet der Pilot die Objektverfolgung ein. Das Spielfeld wird nun nach den Ball mit Hilfe der Kamera und der Positioniereinheit abgesehen, bis der Ball gefunden wird (s. blaue Pfeile in der Abbildung 1.8).

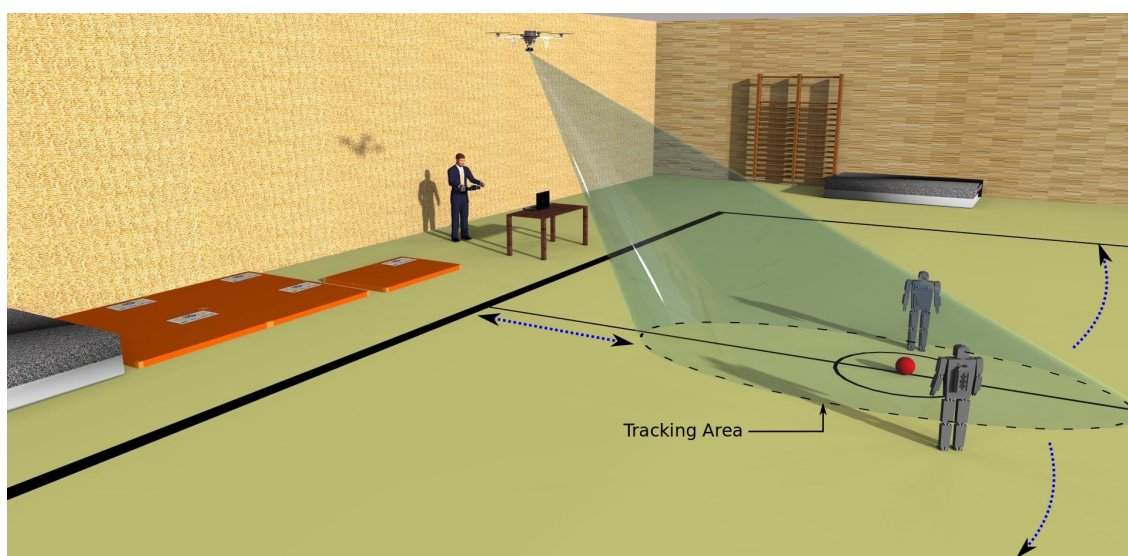


Abbildung 1.8: Start des Szenarios

Daraufhin spielen sich zwei Personen den Ball langsam, in Bodennähe, innerhalb des Spielfeldes zu. Die Kamerapositioniereinheit wird dabei ständig autonom nachgestellt, um den Ball möglichst im Zentrum des Bildes zu behalten. Währenddessen kann der Pilot den Multikopter frei bewegen (s. Abbildung 1.9).

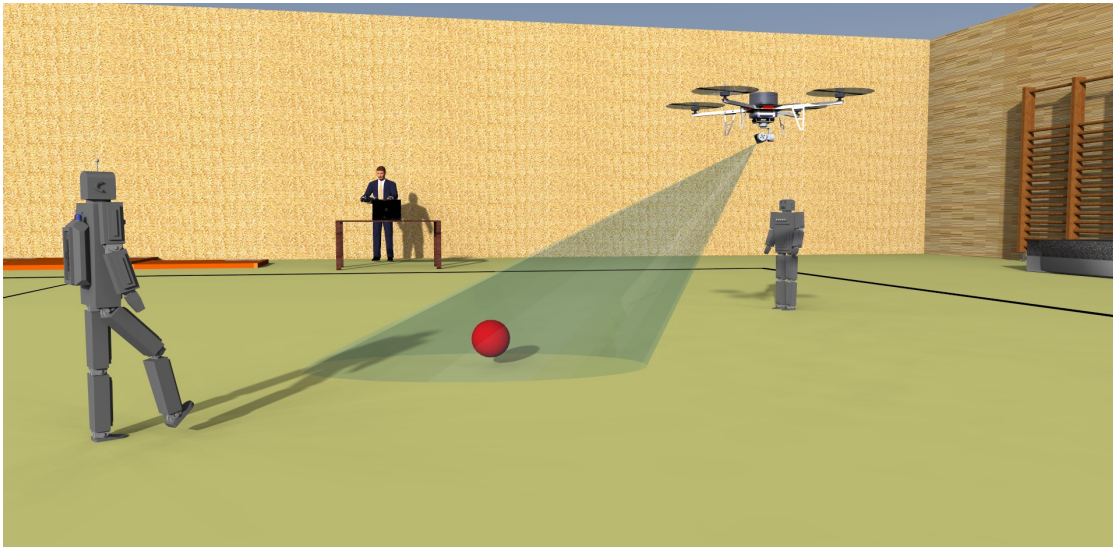


Abbildung 1.9: Mögliche Spielsituation im Szenario

Das Kamerabild samt der Objektverfolgung wird fortlaufend über eine gesonderte Funkverbindung an eine Telemetriestation gesendet, welche das aktuelle Bild auf einem Ausgabebildschirm darstellt. Neben dem aktuellen Kamerabild sollen Telemetriedaten des Multikopters an die Station gesendet werden. Dazu zählen u. a.

- Flughöhe,
- Akkustand,
- Reglerwerte der einzelnen Motoren,
- Sensorwerte,
- Lagewinkel
- etc.

Damit der Pilot permanent die Daten „Flughöhe“ und „Akkustand“ überwachen kann, werden diese separat an die Fernsteuerung gesendet.

Das Szenario wird standardmäßig beendet, indem der Pilot die Objektverfolgung (Kamera fährt in Nullposition) und die Videoaufnahme (Kamera und Positioniereinheit stromlos schalten) deaktiviert und schließlich die manuelle Landung des Multikopters durchführt (außerhalb des Spielfeldes).

Das erweiterte Szenario

Das in Kapitel 1.4 beschriebene Szenario kann in den Kontext eines Fußballspiels eingebettet werden. Durch zwei farblich unterschiedlich gekennzeichnete Linien werden die zwei Tore konkretisiert (s. Abbildung 1.10). Durch die digitale Bildverarbeitung ist es möglich, erzielte Tore zu erkennen und mitzuzählen. Der Spielstand ist an die Telemetriestation zur Anzeige weiterzuleiten. Das System kann somit als Torrichter eingesetzt werden.

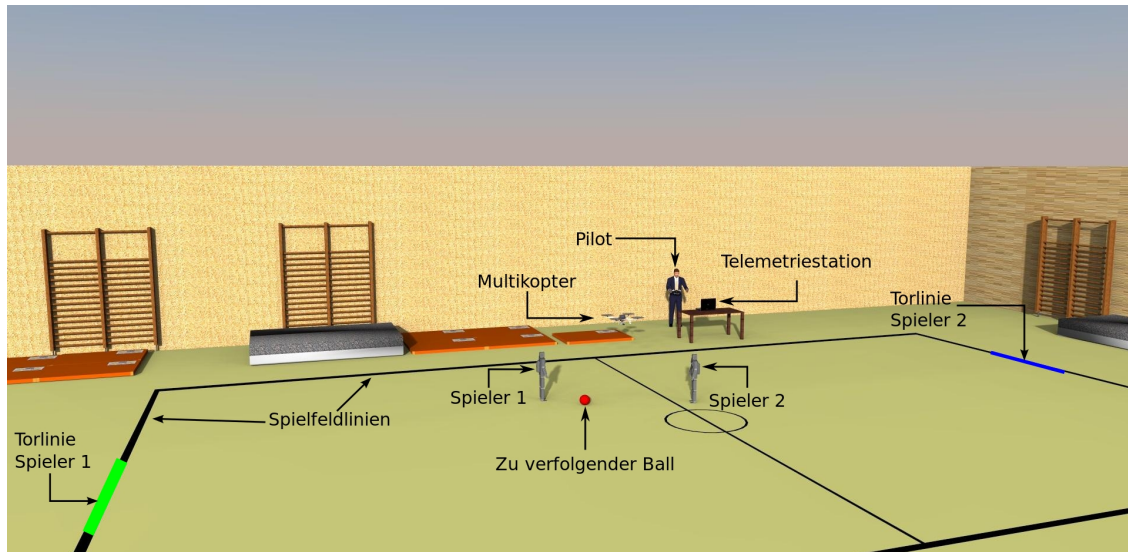


Abbildung 1.10: Mögliche Spielsituation im erweiterten Szenario

Ist die Objekterkennung aus technische Gründen (z. B. Objekt zu weit von der Kamera entfernt) nicht mehr möglich, wird dem Piloten eine Flugrichtung (Vorzugsflugrichtung) vorgeschlagen, um den Ball schnellstmöglich wiederzufinden. Diese Situation muss sich klar von der Situation 3 abgrenzen und die Kamerapositionierungseinheit muss die aktuelle Position halten.



2 Projektmanagement

In diesem Kapitel werden alle für das Projektmanagement wichtigen Informationen über die Projektgruppe Avionic Architecture zusammengefasst. Es beginnt mit den Projektmodalitäten, die die Rahmenbedingungen des Projektes beleuchten. Anschließend findet man den Projektablauf, also die Benennung und die Beschreibung des Ablaufs der einzelnen Phasen des Projektes. Danach folgt die Erläuterung der Organisationsstruktur, in der die Teammitglieder organisiert sind. Zuletzt wird auf die vorhanden personellen, zeitlichen und finanziellen Ressourcen des Projektes eingegangen.

2.1 Projektmodalitäten

Dieses Projekt ist Teil der Projektgruppe Avionic Architecture an der Carl von Ossietzky Universität Oldenburg. Sie wird von der Abteilung Hardware/Software-Entwurfsmethodik der Universität angeboten und zusätzlich vom OFFIS e. V. unterstützt. Im folgenden werden die Universitäts- und OFFIS-Mitarbeiter, welche die Ansprechpartner für unsere Projektgruppe sind, als Kunden bezeichnet, da ihnen unter anderem diese Position im Rahmen des Projektes zukommt. Weitere Informationen bezüglich der Rahmenbedingungen des Projekts sind im Kapitel 2.3.1 zu finden.

2.2 Projektablauf

In diesem Abschnitt wird der Ablauf des Projektes, also seine Phasen, deren Reihenfolge und deren Zusammenhang erläutert. Zu Beginn erfolgt eine sehr grobe Übersicht über den Aufbau des Projektes. Anschließend werden die angerissenen Projektteile gesondert erläutert. Zuerst wird das Vorgehensmodell, nach dem im Projekt vorgegangen wird, beleuchtet. Anschließend wird der Projektablauf noch einmal detaillierter anhand der Arbeitspakete, die erledigt werden müssen, dargestellt.

2.2.1 Grober Projektaufbau

Vorbereitungsphase In der Zeitspanne vom 01.04.2014 bis zum 08.05.2014 wurde das Projekt vorbereitet. In dieser Zeit wurde ein Proposal erstellt, in dem die Realisierbarkeit des Projektes geprüft und die Definition der Aufgabe verfeinert wurden. Außerdem wurden der Umfang und der Inhalt der nicht kritischen Task festgelegt. Darüber hinaus wurde die Aufgabenverteilung im Team festgelegt, eine Voruntersuchung verschiedener Vorgehensmodelle vorgenommen, um später eine Auswahl treffen zu können. Schlussendlich wurde der Projektplan erstellt und die Hauptphase des Projektes eingeleitet. Das entstandene Proposal kann im Anhang dieses Dokumentes eingesehen werden.



Projektphase Die Projektphase besteht aus vier Teilphasen, wie in Tabelle 2.1 dargestellt. Alle vier Phasen werden in der Zeit vom 09.05.14 bis zum 31.03.2015 bearbeitet, wobei die genauen Daten der oben genannten Tabelle entnommen werden können.

In der ersten Teilphase wird die Spezifikation an den Multikopter und die nicht sicherheitskritische Anwendung erstellt. Darüber hinaus werden Betriebs- und Umweltbedingungen erarbeitet, die vorliegen müssen, damit der Multikopter seine Spezifikation erfüllt. Anschließend werden sicherheitskritische Teile der Spezifikation herausgearbeitet, die separat zu dem nicht sicherheitskritischen Aufgabenbereich behandelt werden. In dieser Phase entstehen bereits Systemtestfälle mit denen der Prototyp später getestet wird. Außerdem werden Grundlagen in den Bereichen Flugphysik, Regelungstechnik und Hardware gesammelt und einzelne Personen arbeiten sich in die Software, die im Projekt verwendet werden soll ein.

In der nächsten Teilphase, Architekturentwurf, werden die Teilkomponenten des Projektes identifiziert und deren Beziehung zueinander festgelegt. Ferner werden die Schnittstellen zwischen den Komponenten definiert. Parallel dazu fällt die Entwicklung und das Testen der Regler für das Flugverhalten des Multikopters an. Außerdem werden Integrationstests erstellt, um die Kommunikation zwischen den Teilkomponenten, aus denen der Multikopter besteht, später damit zu testen.

In der Feinentwurfs- und Implementierungsphase muss der Prototyp gemäß der erarbeiteten Architektur in CAMEL-View implementiert werden. Darüber hinaus muss eine Hardwareplattform entwickelt werden und die Komponenten der Architektur müssen auf die Hardwareplattform abgebildet werden.

In der Testphase kann der erste Prototyp mit einer Software-in-the-Loop-Simulation (SIL-Simulation)⁵ getestet werden.

Anschließend wird der generierte Code auf die Hardware gespielt und mit Hilfe der Hardware-in-the-Loop-Simulation (HIL-Simulation)⁶ getestet. Im Anschluss folgen die ersten Tests mit dem vollständigen Prototyp unter realen Betriebsbedingungen.

In der letzten Phase folgt die Kundenabnahme in Form einer Präsentation und der Aushängung einer Dokumentation.

Abschlussphase Eine Abschlussphase, wie in den meisten Projekten üblich, gibt es bei diesem Projekt nur in einer abgeschwächten Form. Diese Abschlussphase wird von Seiten der Projektgruppe nicht dokumentiert und die Phase liegt nicht im Projektzeitplan, sondern schließt sich danach an. Inhaltlich wird es ein Feedback für den Kunden geben, das helfen soll, künftige Projektgruppen besser zu gestalten und die Teammitglieder erhalten eine Bewertung ihrer Leistungen innerhalb der Projektgruppe. Es wird außerdem Feedback für die Teammitglieder geben, damit sie ihre Bewertung nachvollziehen können.

⁵Bei der SIL-Simulation wird vorhandener Sourcecode in einer virtuellen Simulationsumgebung ausgeführt (vgl. [SZ06, S. 281]).

⁶Bei der HIL-Simulation werden vorhandener Sourcecode und Hardwarekomponenten in einer virtuellen Simulationsumgebung betrieben (vgl. [SZ06, S. 283]).



Tabelle 2.1: Grober Aufbau des kompletten Projektes

Phasenname	Phaseninhalt	Phasenbeginn / -ende
Vorbereitungsphase	Erstellen des Proposals	01.04.2014 / 08.05.2014
Projektphase	Anforderungsdefinition	09.05.2014 / 06.08.2014
	Architekturentwurf	07.08.2014 / 22.10.2014
	Feinentwurf & Implementierung	23.10.2014 / 14.01.2015
	Test	14.01.2015 / 31.03.2015
Abschlussphase	Dokumentation, Bewertung & Feedback	nach dem 31.03.2015

2.2.2 V-Modell

Im Grunde arbeitet die Projektgruppe nach einem leicht modifizierten V-Modell. Das V-Modell umfasst neun Phasen, die in einem "V" angeordnet sind. Die linke Seite des "V" umfasst alle Aktivitäten, die zur Planung und zum Entwurf des Systemdesigns durchgeführt werden müssen. Die Phasen werden im Laufe dieses Prozesses von links oben an durchlaufen, wie in Abbildung 2.1 anhand der schwarzen, durchgehenden Pfeile dargestellt. Dabei muss darauf geachtet werden, dass keine Funktionalität, die in der Anforderungsdefinition erhoben worden ist, beim Durchlaufen der Phasen nicht abnimmt. Es darf nur Funktionalität hinzugefügt werden. In diesen Phasen werden darüber hinaus Testfälle erstellt, mit deren Hilfe die erarbeiteten Spezifikationen später validiert⁷ werden können. Die Spitze am untersten Ende des "V" stellt die Phase der Implementierung des Designs dar. Die rechte Seite des "V" stellt die Integration und das Testen des Systems dar. Hierbei werden in jeder Phase die Testfälle durchgeführt, die der jeweiligen Phase im "V" gegenüberliegen. Wenn sich zu diesem Zeitpunkt herausstellt, dass das System Fehler enthält, muss auf die linke Seite des "V" zurückgekehrt werden. Es wird in die Phase zurückgekehrt, in welcher der Testfall, der den Fehler aufgezeigt hat, erstellt worden ist, wie in Abbildung 2.1 anhand der hellgrauen, waagerechten Pfeile zu sehen. Der Fehler muss in genau dieser Phase behoben werden, da er in dieser Phase ursprünglich in das System hinein gekommen ist. Eine Behandlung des Fehlers in einer anderen Phase, entspricht dem Erstellen eines Workarounds und nicht dem Lösen des ursprünglichen Problems. Nach dem Auftauchen eines Fehlers müssen alle folgenden Phasen erneut durchlaufen werden. Es muss also mindestens die Spitze und die komplette rechte Seite des "V" erneut durchlaufen werden.

Ist die Phase "Systemtest" durchlaufen erfolgt die Präsentation und der Abschluss der Projektgruppe. Abnahmetests werden, nach Wunsch des Kunden, im Rahmen dieses Projektes nicht erstellt, da der Kunde regelmäßig und häufig Einblick in das gesamte Projekt hat.

⁷Nicht-formaler Beweis, dass das System der Spezifikation entspricht

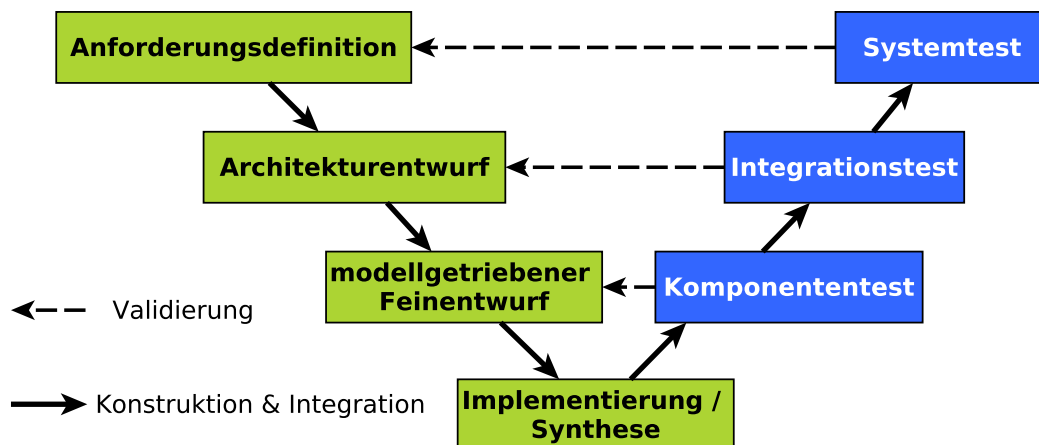


Abbildung 2.1: V-Modell

2.2.3 Phasen des V-Modells

Im Folgenden werden die Phasen des V-Modells, die in Abbildung 2.1 zu sehen sind, kurz beschrieben.

Anforderungsdefinition In der Anforderungsdefinition wird das Verhalten des Systems spezifiziert. Es wird festgelegt, welche Funktionalitäten das System bereitstellen muss und unter welchen Bedingungen es verwendet werden kann. In der Regel werden danach die Abnahmetests des Systems spezifiziert. Im Rahmen der Projektgruppe entfällt dieser Punkt jedoch, da keine Abnahmetests durchgeführt werden. Außerdem findet eine Einarbeitung in die Software, die verwendet werden soll und die informatikfremden Themenbereiche mit denen das Projekt zusammenhängt statt.

Architekturentwurf In der Architekturphase wird das spezifizierte Verhalten des Systems in funktionale Teile des Systems übersetzt. Es wird festgelegt, wie das System das spezifizierte Verhalten erreichen kann. Außerdem werden Systemtestfälle erstellt. Darüber hinaus werden die Teilkomponenten des Systems identifiziert und ihr Verhalten spezifiziert. Anschließend werden deren Schnittstellen definiert und entsprechende Testfälle für den Integrationstest erstellt.

Modellgetriebener Feinentwurf Im modellgetriebenen Feinentwurf werden die zuvor identifizierten Teilkomponenten funktional spezifiziert. Es wird festgelegt, wie das zuvor spezifizierte Verhalten innerhalb jeder einzelnen Komponente umgesetzt wird. Die sicherheitskritische Anwendung wird ab dieser Phase modellgetrieben entwickelt, die nicht sicherheitskritische Anwendung wird wie in der klassischen Softwareentwicklung verfeinert, ohne ein Modell zu erstellen. Den Abschluss dieser Phase bildet die Erstellung von Testfällen für den Komponententest.

Implementierung / Synthese In der Programmierungsphase wird das zuvor erarbeitete Design in die Zielsprache überführt.



Komponententest In dieser Phase werden die Komponenten, mit den im Feinentwurf spezifizierten Tests, auf ihre Korrektheit überprüft.

Integrationstest In der Integrationstestphase werden die Schnittstellen zwischen den Komponenten mit Hilfe der zuvor erstellten Integrationstestfälle getestet.

Systemtest In dieser Phase wird mit Hilfe von Systemtestfällen das vollständige System gegen die Anforderungen getestet.

2.2.4 Arbeitspakete der einzelnen Phasen

Um strukturiert arbeiten zu können, werden alle Aufgaben soweit möglich zu thematisch ähnlichen Arbeitspaketen zugeordnet. Diese Arbeitspakete werden dann von jeweils einem Team bearbeitet. Jedes dieser Teams hat einen Teamleiter, der selbst an Aufgaben mitarbeitet, aber auch als Ansprechpartner für den Kunden und andere Teamleiter zur Verfügung steht. Der Teamleiter ist auch für die Zeitplanung und die Delegation der Aufgaben des Teams verantwortlich. Eine Person kann mehreren Teams angehören und gleichzeitig in ihnen arbeiten.

Die wichtigsten Teams und die Zeitspanne, über die sie gearbeitet haben sind in Abbildung 2.2 zu sehen. Es folgt eine kurze Beschreibung der Teams und ihrer Aufgaben.

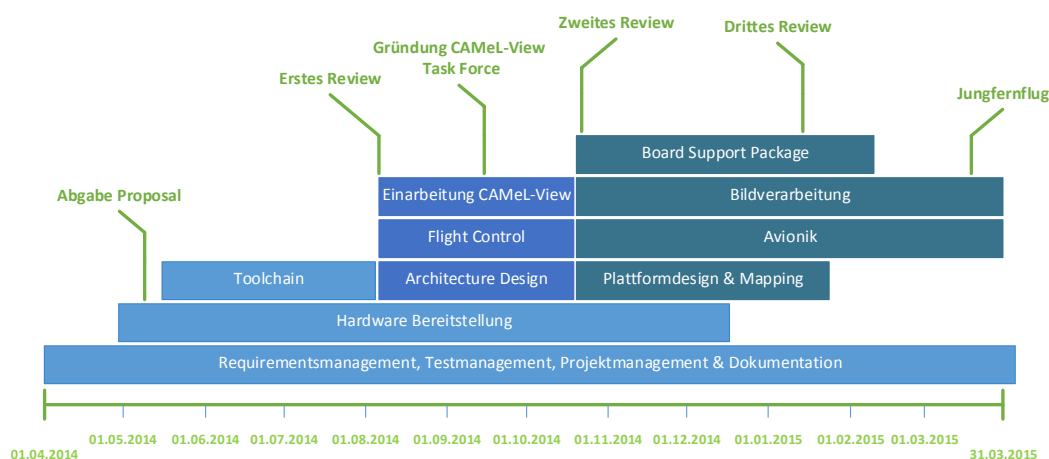
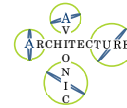


Abbildung 2.2: Wichtige Teams im Laufe des Projektes dargestellt auf einem Zeitstrahl mit Hervorhebung einzelner Meilensteine

Teams die das ganze Projekt über aktiv waren

- **Requirementsmanagement** Aufgaben sind zu Beginn des Projektes die Erfassung der Anforderungen und die Ausarbeitung der Vision. Im Projektverlauf müssen Anforderungen nachgepflegt werden.
- **Testmanagement** Zu Anfang muss in diesem Team ein Testplan erstellt werden. In der ersten Hälfte des Projektes müssen dann entsprechende Testfälle erstellt werden. In der zweiten Hälfte des Projektes sollen Test durchgeführt und protokolliert werden. Diese Aufgabe kann in die Teams delegiert werden, die das zu testende Fragment erstellt haben. Somit bleibt die Überwachung des Testfortschritts und die Qualitätssicherung beim Testen vorrangige Aufgabe in der zweiten Hälfte des Projektes.
- **Projektmanagement** In den Anfängen des Projektes müssen viele Regeln aufgestellt und durchgesetzt werden. Die Arbeitsumgebung muss festgelegt und eingerichtet



tet werden und ein Zeitplan muss erstellt werden. Dieser Zeitplan muss im Laufe des Projektes angepasst werden.

- **Dokumentation** Dokumentation muss in allen Teams durchgängig erledigt werden.

Teams der Phasen "Erstellen des Proposals" und "Anforderungsdefinition"

- **Hardwarebereitstellung** Dieses Team wählt die Hardware, die für das Projekt eingekauft werden muss aus, bestellt diese und testet ihr Funktion, um sie ggf. umzutauschen oder andere Hardware stattdessen zu kaufen.
- **Toolchain** In diesem Team werden die Möglichkeiten die die Zynq-Plattform bietet bewertet. Unter anderem muss ermittelt werden, ob es möglich ist ein Betriebssystem zu installieren und wie der Zynq mit Code bespielt werden kann.

Teams der Phase "Architekturentwurf"

- **Architecture Design** Dieses Team erstellt das Architekturdesign für das gesamte Projekt. Es werden Komponenten identifiziert und deren Schnittstellen untereinander festgelegt.
- **Flight Control** Erste Flugregler werden in diesem Team erstellt und getestet.
- **Einarbeitung in CAMEL-View** Da im Feinentwurf CAMEL-View verwendet werden soll, um die Avionik zu implementieren, muss zu diesem Zeitpunkt der Umgang mit der Software erlernt und das Wissen darüber in der Gruppe weitergegeben werden. Innerhalb dieser Phase wurde ermittelt, dass CAMEL-View ein sehr komplexes Programm ist, das eventuell nicht für den Einsatz in der Projektgruppe geeignet ist. Aus diesem Grund wurde das Team in eine CAMEL-View Task Force umgewandelt, die sich intensiv mit CAMEL-View beschäftigen sollte, um festzustellen, ob es alle von uns an es gestellten Anforderungen erfüllen kann. Nach der Umwandlung in die CAMEL-View Task Force standen dem Team mehr Arbeitskräfte zur Verfügung und alle Mitglieder des Teams wurde soweit möglich von anderen Aufgaben befreit.

Teams der Phase "Feinentwurf, Implementierung und Test"

- **Plattformdesign & Mapping** Dieses Team führt die Arbeit des Teams Architecture Design fort. Es hat die Aufgabe ein Plattformdesign auf Basis des Zynq Chips zu erstellen und anschließend die im Architecture Design Team erarbeiteten Komponenten auf dieses Plattformdesign zu mappen.
- **Avionik** Das Team Avionik führt die Arbeit von zwei Teams fort. Die Regelalgorithmen und das Know-How aus dem Team Flight Control werden mit Hilfe der Vorarbeit der CAMEL-View Task Force in CAMEL-View umgesetzt. Später werden auch die Tests der Avionik durch dieses Team durchgeführt und protokolliert.



- **Bildverarbeitung** Das Bildverarbeitungs-Team erstellt den Feinentwurf für die Bildverarbeitung und setzt diesen dann in C-Code um. Das Testen der Bildverarbeitung gehört darüber hinaus zu den Aufgaben dieses Teams.
- **Board Support Package** Dieses Team setzt die Arbeit des Toolchain Teams fort. Es bildet die Schnittstelle zwischen den Entwicklungsteams und der Plattform, so dass fertige Prototypen sofort auf der entsprechenden Hardware getestet werden können. Zu den Aufgaben gehören unter anderem die Auswahl eines geeigneten Betriebssystems, dessen Installation, der Testaufbau von Hardware für den HIL-Test und die Installation von verwendeten Bibliotheken.

2.3 Organisationsstruktur

Wie in Kapitel 2.2.3 beschrieben wird die Gruppe in kleinere Teams unterteilt, um strukturiert arbeiten zu können. Viele Projektgruppenmitglieder haben neben ihrer Arbeit in einem oder mehreren Teams noch einen Verantwortungsbereich, der für die gesamte Projektgruppe relevant ist.

2.3.1 Verantwortungsbereiche innerhalb der Projektgruppe

Verantwortung in einem Bereich zu übernehmen bedeutet Ansprechpartner für andere Gruppenmitglieder und Betreuer zu sein, immer über den aktuellen Fortschritt im besagten Bereich Bescheid zu wissen und die Qualitätssicherung in diesem Bereich zu übernehmen. Über diese speziellen Verantwortungsbereiche hinaus ist jedes Gruppenmitglied Entwickler. Als solcher arbeitet es, gemäß seinen persönlichen Fähigkeiten, an bestimmten Teilen des Projektes mit, ohne dass ein Bezug zu seiner speziellen, zusätzlichen Rolle besteht. Folgende Rollen wurden an Gruppenmitglieder verteilt, wobei ein Stellvertreter die entsprechende Position bei Unpässlichkeit vertritt und Teile ihres Aufgabengebietes nach Absprache übernimmt :

Projektleitung und Stellvertreter	Die Gruppenmitglieder mit diesen Rollen haben die Aufgabe das Projekt zu leiten und zu planen. Dazu gehören die Erstellung eines Projektplans und die Verwaltung der Ressourcen (s. Kapitel 2.3.1). Außerdem sind sie dafür verantwortlich, dass die Zeit- und Kostenpläne eingehalten werden.
Phasenleiter	Der Phasenleiter hat die Aufgabe die aktuelle Projektphase, in der er gewählt ist, vorzubereiten und bezüglich Zeitplanung und Qualitätssicherung zu überwachen.



Beauftragter für Versionskontrolle	In dieser Rolle ist man für die Versionsverwaltung des Projektes mit Github verantwortlich. Die Aufgaben reichen von der Einrichtung von Git, über die Pflege, bis hin zur Beratung der Gruppe bei Schwierigkeiten.
Testbeauftragter und Stellvertreter	Als Testbeauftragter oder dessen Stellvertreter ist man für die Qualitätssicherung im Bereich Testing verantwortlich. Man entscheidet über die Testmethoden, behält die Qualität der Testfälle im Blick und kontrolliert die Testabdeckung, also die Vollständigkeit der Testszenarien.
Dokumentationsbeauftragter und Stellvertreter	Die Gruppenmitglieder die diese Rollen innehaben kümmern sich um die Bereitstellung von Vorlagen und Hilfestellung für Latex. Darüber hinaus achten sie darauf, dass die Dokumentation der Aktivitäten voranschreitet und vollständig ist.
Beauftragter für Öffentlichkeitsarbeit und Stellvertreter	Diese Rollen befassen sich mit der Darstellung des Projektes nach außen. Zu den Aufgaben gehören die Einrichtung und Pflege einer Homepage sowie die Vorbereitung von Messebesuchen als Aussteller.
Sozialmanager	Der Sozialmanager übernimmt die Organisation von Aktivitäten zur Verbesserung der Betriebsklimas und ist Ansprechpartner für Gruppenmitglieder, die in einen zwischenmenschlichen Konflikt geraten sind, den er daraufhin zu lösen versucht.
Kassenwart	Als Kassenwart kümmert man sich um die von der Gruppe bereitgestellten Finanzmittel, welche zusätzlich zu den Finanzmitteln, die vom Auftraggeber für das Projekt bereitgestellt wurden, vorhanden sind. Man muss einen Überblick über die Finanzlage dieser Geldmittel haben und kann bedingt über deren Ausgabe entscheiden, da über Ausgaben hauptsächlich demokratisch in der Gruppe entschieden wird.



Teamleiter

Jedes Gruppenmitglied mit dieser Rolle vertritt Team innerhalb der Projektgruppe. Teams werden, wie oben erwähnt, gebildet, um koordiniert Aufgabenpakete bearbeiten zu können. Zu den Aufgaben eines Teamleiters gehören die Verteilung der Arbeit, die innerhalb des Teams anfällt, auf die Mitglieder und die Kommunikation mit anderen Teams. Außerdem ist er für die Qualitätssicherung innerhalb seines Teams verantwortlich. Da Teams mitunter nur kurz bestehen, ist diese Rolle meist eine kurzfristige Aufgabe und kann neben einer weiteren Rolle angenommen werden, ohne dass eine Vernachlässigung einer der beiden Aufgabenbereiche zu befürchten ist.

Die Verteilung der Rollen auf die Gruppenmitglieder kann Tabelle 2.3 entnommen werden.

Rolle	Name des Rolleninhabers
Projektleitung	Steven Schmidt
Stellvertretende Projektleitung	André Schaadt
Phasenleiter	Thomas Nordlohne
Beauftragter für Versionskontrolle	Martin Bornhold
Stellvertretender Beauftragter für Versionskontrolle	Patrick Schmale
Testbeauftragter	Jenny Röbesaat
Stellvertretender Testbeauftragter	Sebastian Vander Maelen
Dokumentationsbeauftragter	Markus Franz Wieghaus
Stellvertretender Dokumentationsbeauftragter	Henning Elbers
Beauftragter für Öffentlichkeitsarbeit	Thomas Nordlohne
Stellvertretender Beauftragter für Öffentlichkeitsarbeit	Jörn Bellersen
Sozialmanager	Niklas May-Johann
Kassenwart	Niklas May-Johann
Teamleiter	<i>Wechselnde Teammitglieder</i>

Tabelle 2.3: Rollenverteilung innerhalb der Projektgruppe.

2.4 Projektressourcen

In diesem Abschnitt werden die finanziellen und die zeitlichen Ressourcen, die der Projektgruppe zur Verfügung stehen, dargestellt.

2.4.1 Zeitliche Ressourcen

Das Projekt läuft über zwei Semester vom 01.04.2014 bis zum 31.03.2015. Jeder der zwölf Teilnehmer erhält nach der geltenden Prüfungsordnung der Universität Oldenburg für einen erfolgreichen Abschluss der Projektgruppe zwölf Kreditpunkte pro Semester. Je sechs Kreditpunkten liegen rechnerisch acht Wochenarbeitsstunden zu Grunde. Somit ergibt sich über den gesamten Zeitraum von zwei Semestern ein Wochenarbeitszeit von $16 \frac{\text{Stunden}}{\text{Woche}}$ pro Person. Da die Projektgruppe zwölf Teilnehmer hat ergeben sich zeitliche Ressourcen von 9984 Mannstunden für das 52 Wochen dauernde Projekt. Davon müssen lediglich circa 32 Mannstunden Urlaub pro Person abgezogen werden. Damit verbleiben 9600 Mannstunden für das gesamte Projekt.

2.4.2 Finanzielle Ressourcen

Der Projektgruppe stehen insgesamt 1000 € durch den Kunden zur Verfügung. Ein Teil dieses Kapitals wurde vom Kunden bereits für die Anschaffung eines Xilinx Zynq 7020 Evaluation Board verwendet, dass der Projektgruppe uneingeschränkt zur Verfügung steht. Zu diesen Kosten von 230 € kommt ein Sicherheitspuffer von 270 €, der vom Kunden



einbehalten wird. Es verbleiben 500 €, die von der Projektgruppe mit Zustimmung des Kunden für das Projekt ausgegeben werden können.

Zusätzlich hat sich die ENERCON GmbH aus Aurich bereit erklärt die Projektgruppe mit Hardware mit Kosten von bis zu 700 € zu unterstützen.

Somit stehen der Projektgruppe insgesamt 1200 € an Finanzmitteln zur Verfügung.

2.4.3 Kosten des Gesamtprojekts

Es folgt eine Auflistung der wichtigsten Posten als gerundete Werte.

Posten	Preis
Quadropter	schon vorhanden
Fernsteuerung	schon vorhanden
Kamera	110 Euro
Gimbal	366 Euro
Trägerboard	205 Euro
Verteilerplatine	140 Euro
Verbrauchsmaterial & Testequipment	240 Euro
TestRig Box & CAMEL-View Lizenzen	Kostenlos durch iXtronics gestellt
Summe	1061 Euro



3 Voruntersuchungen

Dieses Kapitel behandelt die zur Durchführung des Projekts notwendigen theoretischen Grundlagen. Dabei wird zunächst die Theorie von Multikoptern betrachtet. Anschließend wird auf die Grundlagen der Regelungstechnik eingegangen, da diese für ein stabiles Flugverhalten von Multikoptern essentiell sind. In den Kapiteln 3.3 bis 3.7 stehen die theoretischen Grundlagen für die Objekterkennung im Mittelpunkt. Dazu gehören die digitale Objektverfolgung, die Positionierung und Stabilisierung von Kameras, ein Vergleich unterschiedlicher Kameratypen und -schnittstellen sowie eine Gegenüberstellung verschiedener, drahtloser Übertragungsarten. Im darauf folgenden Kapitel wird ein Überblick über die rechtlichen Grundlagen bezüglich dem Fliegen eines Multikopters gegeben.

3.1 Theoretische Grundlagen zu Multikoptern

In den letzten Jahren gab es große Fortschritte in der Entwicklung von autonomen unbemannten Flugsystemen (engl.: Unmanned Aerial Systems (UAS)⁸). Insbesondere im gewerblichen Bereich, beispielsweise bei der Aufnahme von Luftbildern, kommen diese Flugsysteme verstärkt zum Einsatz. Multikopter stellen eine Unterklasse der UAS dar. Sie erzeugen ihren Auftrieb durch Rotoren und zählen deshalb gemäß Luftverkehrsgesetz (LuftVG §1 (2)) zu den Drehflüglern.

Die ersten Entwicklungen auf dem Gebiet der Multikopter gehen auf den Anfang des 20. Jahrhunderts zurück. Bereits im Jahr 1907 entwickelten die französischen Wissenschaftler Louis und Jaques Bréguet den *Gyroplane No. 1*, den ersten Helikopter zur Personenbeförderung mit vier Rotoren [Lei]. Jedoch rückten Multikopter erst durch die Fortschritte auf den Gebieten der Sensorik und Akkumulatoren in der jüngsten Vergangenheit wieder in den Fokus diverser Unternehmen (wie z.B. Mikrokopter, Ascending Technologies, Microdrones) und Forschungseinrichtungen (z.B. MIT, ETH Zürich, TU München, TU Berlin). Am weitesten verbreitet sind Multikopter mit vier in einer Ebene angeordneten Rotoren, die sog. Quadrokopter. Der Name leitet sich aus dem lateinischen Wort *quadrum* (dt: Viereck) und dem griechischen Wort *pteron* (dt.: Flügel) ab. Neben den Quadrokoptern existieren auch Multikopter mit drei (Trikopter), sechs (Hexakopter), acht (Oktokopter) oder mehr Rotoren. Im weiteren Verlauf erfolgt deshalb zunächst ein Vergleich der unterschiedlichen Multikopterarten. Anschließend wird auf das Flugprinzip, die Sensorik und die Regelung dieser Flugsysteme eingegangen.

⁸veraltet engl.: Unmanned Aerial Vehicle (UAV). Heutzutage hat sich der Begriff UAS durchgesetzt, da zu einem unbemannten Flugsystem nicht nur das Luftfahrzeug (UAV) gehört, sondern auch eine Bodenkontrollstation, ein Bediener, eine Funkstrecke und andere Teilsysteme.[Pul11]

3.1.1 Vergleich der Multikopterarten

Um einen Multikopter stabil in der Luft halten zu können, sind mindestens drei Rotoren notwendig, was zu der Entwicklung von Trikoptern führte. In Abbildung 3.1 ist ein Modell eines solchen Trikopters dargestellt. Aufgrund der ungeraden Rotoranzahl kann das Drehmoment bei diesen Flugsystemen nur mit Hilfe schwenkbarer Motoren kompensiert werden. Dies erschwert die Regelung des Flugverhaltens, da die drei Ausleger eines Trikopters bei horizontalen Bewegungen nicht entkoppelt voneinander betrachtet werden können. Trikotter mit drei schwenkbaren Motoren besitzen jedoch den Vorteil, dass sie ihre Lage beim waagerechten Fliegen nicht verändern müssen und sind deshalb insbesondere für Kameraaufnahmen gut geeignet. [Bü13, S. 58 ff]



Abbildung 3.1: Trikotter auf der Hannover Messe 2013 [Plu13]

Die meisten Multikopter sind mit vier, symmetrisch im Quadrat angeordneten, Rotoren ausgerüstet. In Abbildung 3.2 ist ein solcher Quadroptopter dargestellt. Aufgrund der geraden Rotoranzahl können zwei gegenüberliegende Rotoren im mathematisch positiven Drehsinn und die verbleibenden beiden gegenüberliegenden Rotoren im mathematisch negativen Drehsinn betrieben werden. Dadurch kann das Drehmoment dieser Flugsysteme bei gleicher Drehzahl der Rotoren vollständig neutralisiert werden. Außerdem ist es möglich die Achsen eines Quadroptopters bei der Bewegung in horizontaler Richtung entkoppelt voneinander zu betrachten, was die Regelung vereinfacht.



Abbildung 3.2: Quadroptopter Quadro XL [Quab]

Um die Ausfallsicherheit technischer System zu erhöhen, werden diese häufig redundant ausgelegt. Dieses Prinzip wird auch im Bereich der Multikopter angewandt und kommt bei Hexa- und Octokoptern zum Einsatz. Beispiele solcher Modelle sind in Abbildung 3.3 zu finden. Eine Besonderheit dieser Flugsysteme ist, dass sie selbst beim Ausfall eines Rotors noch ein stabiles Flugverhalten aufweisen. Dies gilt jedoch nur unter der Voraussetzung, dass an allen Seiten des Schwerpunkts des Systems noch Kraft ausgeübt werden kann. Beim Hexakopter kann dadurch ein Rotor ausfallen, während beim Oktokopter zwei beliebige Rotoren ausfallen können. Des Weiteren erzeugen die Hexakopter 1,5-mal und die Octokopter 2-mal so viel Schub wie die Quadrokopter und eignen sich deshalb zum Transport größerer Lasten. [Bü13, S. 60 f]



Abbildung 3.3: Hexa- (links) und Octokopter (rechts) [Pli]

In Tabelle 3.1 werden die Vor- und Nachteile der unterschiedlichen Multikopterarten noch einmal zusammengefasst.

Tabelle 3.1: Vor- und Nachteile der Multikopterarten (vgl. [Bü13])

	Trikopter	Quadrokopter	Hexa-/Octokopter
Vorteile	<ul style="list-style-type: none"> • geringere Materialkosten • für Kameraanwendungen prädestiniert 	<ul style="list-style-type: none"> • hohe Verbreitung • automatische Kompensation des Drehmoments • vergleichsweise einfache Regelung 	<ul style="list-style-type: none"> • redundante Auslegung der Motoren • stabiles Flugverhalten bei Rotorausfall • hohe Schubkraft
Nachteile	<ul style="list-style-type: none"> • schwenkbare Achse notwendig • komplexe Regelung • instabiles Flugverhalten bei Rotorausfall • geringe Schubkraft 	<ul style="list-style-type: none"> • instabiles Flugverhalten bei Rotorausfall 	<ul style="list-style-type: none"> • höhere Materialkosten • komplexer Aufbau • aufwändige Inbetriebnahme und Wartung • hoher Energieverbrauch • hohes Gewicht

3.1.2 Flugmechanik von Multikoptern

Die physikalische Schnittstelle zwischen dem Multikopter und seiner Umwelt bilden die an Bord des Multikopters installierten Sensoren und die Motoren, auf deren Motorwellen die Rotoren montiert sind. Unter dem Begriff Umwelt ist das Medium 'Luft' zu verstehen, in dem sich der Multikopter bewegt.

Multikopter erhalten ihren Auftrieb durch eine der Erdanziehung entgegenwirkende Kraft, die durch die Rotation der Motoren und der daran gekoppelten Rotoren erzeugt wird. Die Motoren besitzen einen charakteristischen Drehzahlbereich $[n_{min}, \dots, n_{max}]$, in dem diese mit der Drehzahl n arbeiten. Die Rotoren besitzen einen definierten Durchmesser d und eine definierte Steigung H . Die Steigung eines Rotors beschreibt die Wegstrecke, die der Rotor entlang der Achse, die normal zu einer durch die Rotorblätter aufgespannten Ebene liegt, während einer vollen Umdrehung zurücklegt. Aus diesen Faktoren lässt der so genannte Volumenstrom Q mit folgender Formel berechnen:

$$Q = \dot{V} \left[\frac{m^3}{s} \right] = \pi [1] \cdot \frac{d^2}{4} [m^2] \cdot H [m] \cdot n \left[\frac{1}{s} \right] \quad (3.1)$$

Wird der Volumenstrom auf ein bestimmtes Medium der Dichte ρ bezogen, so wird dieser mit folgender Gleichung in den Massenstrom \dot{m} überführt:

$$\dot{m} \left[\frac{kg}{s} \right] = \dot{V} \left[\frac{m^3}{s} \right] \cdot \rho \left[\frac{kg}{m^3} \right] \quad (3.2)$$

Ein möglicher Ansatz den Auftrieb zu beschreiben, der einem Multikopter bei einem Flug in Luft widerfährt, führt über das 3. Newton'sche Axiom (Wechselwirkungsprinzip). Newtons 3. Axiom besagt, dass Kräfte immer paarweise auftreten [Nol06, S. 123 ff]. Wirkt ein Körper α eine Kraft auf Körper β aus, so übt der Körper β gleichzeitig eine Kraft auf Körper α mit dem selben Betrag und entgegengesetztem Vorzeichen aus.

Werden diese theoretischen Überlegungen auf den Multikopter und den durch die Rotoren erzeugten Volumenstrom Q in einem Medium der Dichte ρ übertragen, lässt sich der Auftrieb bestimmen. Die Masse, die in einem Zeitintervall Δt durch den Massenstrom der Rotoren ausgestoßen wird, wird im Folgenden als m_{mol} bezeichnet. Wendet man das Wechselwirkungsprinzip auf den Massenstrom der Rotoren an, so bedeutet dies konkret, dass die Luftmoleküle der Masse m_{mol} mit der Geschwindigkeit \vec{v}_{mol} ausgestoßen werden. Um die Ausstoßgeschwindigkeit \vec{v}_{mol} zu erreichen, wirkt die Beschleunigung a für eine gewisse Zeit t_a auf die einzelnen Moleküle. Daraus ergibt sich durch das 1. Newton'sche Gesetz [Nol06] die Kraft \vec{F} mit:

$$\vec{F} \left[\frac{kg \cdot m}{s^2} \right] = m_{mol} [kg] \cdot \vec{a} \left[\frac{m}{s^2} \right] \quad (3.3)$$



Durch die sich drehenden Rotorblätter wird die Kraft \vec{F} erzeugt, die auf die einzelnen Moleküle der Luft einwirkt. Durch das Wechselwirkungsprinzip wirkt jedoch gleichermaßen eine Kraft $-\vec{F}$ von den Molekülen auf die Rotorblätter zurück, welche die Rotorblätter von den Molekülen wegbewegt. Zusätzlich wirkt das Schwerfeld der Erde mit der Kraft \vec{F}_g auf den Multikopter, welche der rückwirkenden Kraft der Moleküle entgegenwirkt. Aufgrund des Gewichtsunterschiedes von einem Molekül und der gesamten Konstruktion eines Multikopters sind viele dieser kleinen Kräfte nötig, um die Erdanziehungskraft \vec{F}_g zu überwinden. Ist die Summe der Beträge der rückwirkenden Kräfte der Moleküle größer als \vec{F}_g , so beginnt der Multikopter an Höhe zu gewinnen. Überwiegt der Kraft \vec{F}_g , so beginnt der Multikopter zu sinken.

Da jedoch aus dem Volumenstrom die Geschwindigkeit \vec{v}_{mol} bekannt ist, wird im Weiteren nicht die Kraft \vec{F} beschrieben, sondern der aus Geschwindigkeit und Masse resultierende Impuls [Nol06, S. 122 f] \vec{p}_{mol} . Aus der folgenden Formel ergibt sich damit der Impuls \vec{p}_{mol} aus Geschwindigkeit und Masse der ausgestoßenen Moleküle zu:

$$\vec{p}_{mol} \left[\frac{kg \cdot m}{s} \right] = m_{mol} [kg] \cdot \vec{v}_{mol} \left[\frac{m}{s} \right] \quad (3.4)$$

Dieser Impuls \vec{p}_{mol} lässt sich über das 2. Newton'sche Axiom [Nol06, S. 122 f] mit der Kraft \vec{F} in Beziehung setzen:

$$F_{mol}^{\rightarrow} \left[\frac{kg \cdot m}{s^2} \right] = \frac{d\vec{p}_{mol}}{dt} \left[\frac{kg \cdot m}{s} \right] \quad (3.5)$$

Durch diesen Ansatz lässt sich außerdem beschreiben, dass der Multikopter bei zunehmender Höhe und gleichbleibender Drehzahl der Motoren nicht weiter an Höhe gewinnen kann. Denn bei steigender Flughöhe des Multikopters sinkt die Dichte ρ . Daraus folgt mit Formel 3.1, dass der Volumenstrom und damit der Impuls (Formel (3.4)) des Massenauswurfs verringert wird. Infolgedessen steigt der Multikopter bei gleichbleibender Drehzahl aufgrund der geringeren Dichte bei zunehmender Höhe nicht weiter, sondern bleibt bei einer Grenzdichte ρ_{grenz} in der Luft stehen. Um weiter an Höhe zu gewinnen muss die Drehzahl des Multikopters gesteigert werden, um den Impuls wieder zu erhöhen.



Abbildung 3.4: Definition Koordinatensystem gemäß MPU 9051 [Teb]

Das sogenannte Gieren, welches einer Rotation um die z-Achse aus Abbildung 3.4 entspricht, wird durch ein auf den Quadrokopter wirkendes Drehmoment M erzeugt. Drehen alle Motoren mit der selben Geschwindigkeit, so ist die Summe der Drehmomente gleich Null. Dies ist dadurch zu beschreiben, dass sich die Drehmomente aufheben, die von den unterschiedlichen Rotationsrichtungen der Motorpaare erzeugt werden. Im Folgenden werden Motor (1) und (2) als $Paar_y$ und Motor (3) und (4) als $Paar_x$ bezeichnet.

In Abbildung 3.4 ist eine einfache Architektur eines Multikopters dargestellt, welcher über vier Motoren verfügt, die jeweils an den Enden eines ”+” montiert sind. Wird dieses Modell zur Beschreibung der Rotation um die z-Achse herangezogen, so lässt sich eine Drehung im mathematisch positivem Sinn dadurch erzeugen, dass die Drehzahl des $Paar_y$ gesteigert wird, während die Drehzahl des $Paar_x$ verringert wird. Dadurch überwiegt das Drehmoment von $Paar_y$ dem des $Paar_x$. Die Drehmomentdifferenz lässt den Multikopter mit einer Kraft \vec{F}_M , die abhängig von der Länge des Motorsauslegers ist, entgegen dem Uhrzeigersinn rotieren. Ein Vertauschen der Drehzahlen der Motorpaare lässt den Multikopter in die entgegengesetzte Richtung rotieren. Aufgrund dessen, dass der Auftrieb eines Paares durch ein Erhöhen der Drehzahlen steigt und der Auftrieb des anderen Paares durch ein Verringern der Drehzahlen sinkt, bleibt die Summe des erzeugten Auftriebs konstant. Dadurch ändert sich die Flughöhe des Multikopters nicht signifikant.

Das sogenannte Nicken und Rollen, welches dem Kippen und die x- bzw. y-Achse entspricht, kann ebenfalls mit Hilfe der Abbildung 3.4 beschrieben werden. Soll der Multikopter in eine beliebige Richtung fliegen, so muss die Drehzahl des Motors verringert werden, welcher sich in Flugrichtung befindet. Zusätzlich muss die Drehzahl des Motors gesteigert werden, welcher sich entgegen der gewünschten Flugrichtung befindet. Die Summe der einzelnen durch die Motoren erzeugten Auftriebe ändert sich nicht. Trotzdem verliert der Multikopter aufgrund des sich einstellenden Anstellwinkels an Höhe, da eine Komponente des Auftriebs zusätzlich den Multikopter in den Vorwärtsflug einwirkt. Der daraus resultierende Auftrieb ist um den Betrag dieser Komponente verringert. Der Höherverlust kann dadurch ausgeglichen werden, dass der gesamte Auftrieb um den Betrag der in den Vorwärtsflug eingehenden Komponente erhöht wird.



3.1.3 Sensorik

Sensoren lassen sich grob in zwei verschiedene Arten der Messwertbeschreibung gruppieren [Hea12]. Eine Art beschreibt die aufgenommenen Werte absolut zu einem Referenzkoordinatensystem. Das Referenzkoordinatensystem ist ortsfest und die aufgenommenen Daten des Sensors werden in einem globalen Bezug zu diesem Koordinatensystem gestellt.

Eine weitere Möglichkeit ist die Beschreibung der Messwerte in einem Koordinatensystem, welches sich an dem Sensor selbst orientiert. Diese Sensoren liefern eine Information, dass sich die Messgröße relativ zum Sensor verändert hat, jedoch ohne diese Daten in einen globalen Bezug zu stellen.

3.1.3.1 Beschleunigungssensoren

Beschleunigungssensoren geben ihre Lage in absoluten Koordinaten im Bezug zum Schwerfeld der Erde an. Es gibt verschiedene Bauformen dieser Sensoren, die Beschleunigungen entweder entlang einer, zweier oder allen drei Raumdimensionen messen können [Hea12]. Die Darstellung der Messwerte erfolgt dabei entlang des kartesischen Koordinatensystems, wie in Abbildung 3.5 dargestellt.

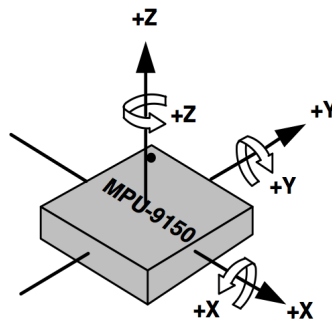


Abbildung 3.5: Definition Koordinatensystem der MPU 9051 [Inv]

Ein Beschleunigungssensor sollte im gemeinsamen Schnittpunkt der Roll-, Nick-, und Gierachse montiert werden, um Zentrifugalkräfte, die bei einem Kippen um eine oder mehrere Achsen auftreten, zu vermeiden oder zumindest zu verringern.

Beschleunigungssensoren werden im Aufbau häufig als „Micro Electro Mechanical Systems (MEMS)“ hergestellt. Bei dieser Art Aufbau ist eine definierte Masse in allen drei Raumrichtungen durch Federn aufgehängt. Dadurch wird ein gedämpftes Feder-Masse-System hergestellt, welches durch einwirkende Kräfte ausgelenkt wird. Durch den physikalischen Aufbau bildet das Feder-Masse-System gleichzeitig eine Kapazität entlang jeder einzelnen Achse. Die Auslenkung der Masse entlang einer oder mehrerer Raumrichtungen kann durch die Änderung der Kapazität direkt ermittelt werden. Eine Kapazitätsänderung erfolgt dadurch, dass sich die Abstände der Elektroden des aufgebauten Kondensators ändern und

somit eine messbare Größe erzeugen. Durch die vektorielle Interpretation der wirkenden Kraft und den ermittelten Beträgen der Komponenten des Beschleunigungsvektors entlang der kartesischen Raumachsen, kann der Betrag und die Richtung der angreifenden Beschleunigung ermittelt werden.

3.1.3.2 Gyroskop

Gyroskope, auch Kreisel genannt, sind in der Lage, Winkelbeschleunigungen entlang einer spezifischen Achse zu messen (Abbildung 3.5). Von diesen Sensoren existieren ebenfalls Bauformen, die Winkelbeschleunigungen in einer, zweier oder allen drei Raumdimensionen detektieren können. Der von dieser Art Sensoren ausgegebene Messwert ist eine relative Größe, die sich auf das interne Koordinatensystem des Sensors bezieht [Hea12]. Es ist nicht wie mit einem Beschleunigungssensor möglich, die globale Lage eines Multikopters mit einem Gyroskop zu bestimmen, da jeweils die Rotation um eine Achse bestimmt wird. Ohne dass weitere Informationen, wie die Null-Lage mit in die Auswertung einfließen, kann die Regelung eines Multikopters die Ausgangslage nicht bestimmen. Dennoch ist es möglich, eine mehr oder weniger gut funktionierende Regelung mit einem Gyroskop zu implementieren. Dazu werden aerodynamische Effekte genutzt, die dämpfend auf das Flugverhalten wirken. Durch den bei einer Bewegung herrschenden Luftwiderstand wird ein übermäßig starkes Kippen um die Roll- bzw. Nickachse verhindert, da der Luftwiderstand eine zur Bewegungsrichtung entgegengesetzte Kraft erzeugt, die den Multikopter entgegen des Neigungswinkels rotieren lässt. Der Betrag der Effekte ist jedoch recht gering, sodass sich ein mehr oder weniger starkes Hin- und Herpendeln als Resultat einstellt.

Wenn Gyroskope in Ruhe verharren, entsteht der Effekt einer *Drift*, die einer Scheinbewegung entspricht, obwohl sich der Sensor in Ruhe befindet. Diese Scheinbewegung bzw. die Drift ist auf Temperaturschwankungen zurückzuführen, die von außen auf den Sensor wirken [Bü13]. Dazu zählt beispielsweise das Auftreffen von Sonnenstrahlen auf das Sensorelement.

Gyroskope gibt es im Bereich der Multikopter im Wesentlichen in zwei Ausführungen: Die älteren Bauformen, die auf dem piezoelektrischen Effekt beruhen und die moderneren MEMS-Bauformen. Bei den piezoelektrischen Gyroskopen werden Piezokristalle verwendet, die sich einerseits beim Anlegen einer Spannung verformen und andererseits beim Verformen eine elektrische Spannung erzeugen. Bei einem Gyroskop dieser Bauform werden beide Eigenschaften benötigt. Bei dieser Bauform werden drei quaderförmige Piezokristalle verwendet, die jeweils paarweise an den Kanten miteinander verbunden sind. An eines der drei Piezoelemente wird eine sinusförmige Spannung angelegt, die das Element anregt. Durch den Gyroskopischen-Effekt [Bü13, S. 15] werden die beiden anderen Piezoelemente bei einer Drehung des Sensorelements ebenfalls in Schwingung versetzt, die gemessen werden kann. Die gemessene Spannung ist proportional zum Betrag der wirkenden Winkelgeschwindigkeit. Bei den MEMS-Gyroskopen werden ebenfalls zwei Piezoelemente an-

geregt, die sich in einer Ebene befinden. Bei dieser Bauform wird jedoch der Coriolis-Effekt genutzt, um die Winkelgeschwindigkeit, indirekt durch die bei einer Rotation wirkenden Coriolis-Kraft, zu messen.

3.1.3.3 Kompass

Ein Kompass, auch Magnetometer genannt, gibt die Ausrichtung absolut zum Nordpol an. Dabei ist jedoch zwischen dem *geographischen* Norden und dem *geodätischen* Norden zu unterscheiden. Der geographische Norden ist auf Landkarten angegeben und zeigt auf den Punkt, in dem die Meridianlinien auf der Nordhalbkugel zusammenlaufen. Ein Kompass hingegen, zeigt zum geodätischen Norden. Zwischen diesen beiden unterschiedlich gerichteten Größen (Vektoren) spannt sich, je nach Position auf der Erdoberfläche, ein Winkel auf. Dieser Winkel zwischen dem geographischen und magnetischen Norden wird *magnetische Deklination* genannt. Neben der örtlichen Abhängigkeit kommt noch eine zeitliche Veränderung der magnetischen Deklination hinzu.

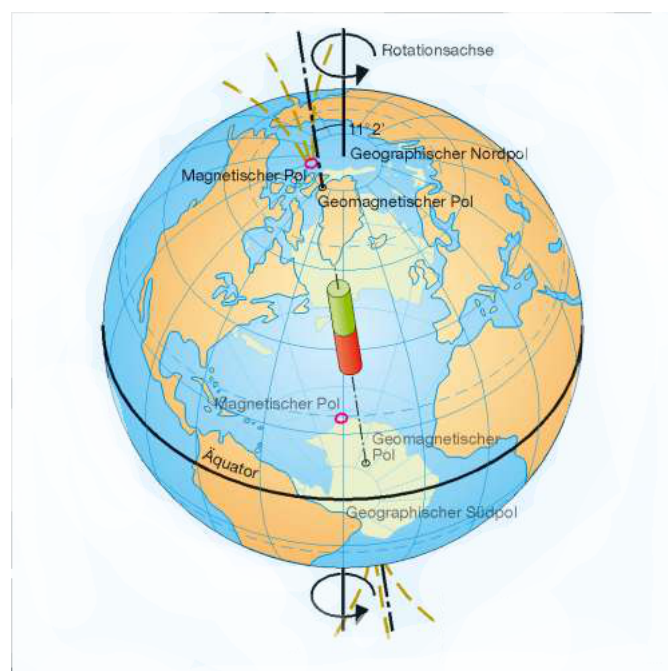


Abbildung 3.6: Illustrierung der magnetischen Deklination [Gmbb]

In Abbildung 3.6 ist die magnetische Deklination anhand eines Ausschnitts einer abstrakten Weltkarte illustriert.

Die Funktionsweise eines MEMS-Kompass beruht auf dem Hall-Effekt [Bü13, S. 31]. Der Hall-Effekt beschreibt, dass eine elektrische Spannung an einem leitfähigen Material entsteht, wenn sich dieses Material in einem Magnetfeld bewegt. Die Hall-Spannung entsteht dabei im Leiter sowohl zum Stromfluss wie auch zu den Magnetfeldlinien lotrecht. Die auf diese Weise entstehende Spannung kann dazu genutzt werden, die absolute Ausrichtung



zum magnetischen Norden zu bestimmen. Dabei ist die erzeugte Spannung proportional zum Winkel, der zwischen der Referenzrichtung des Kompasses und dem magnetischen Nordpol der Erde aufgespannt wird.

Als Magnetfeld für einen Kompass dient in erster Linie das Erdmagnetfeld zum Erzeugen dieser Hall-Spannung. Statische, sich zeitlich nicht ändernde Magnetfelder können durch eine Kalibrierung herausgefiltert werden. Ein Kompass ist jedoch auch sehr störanfällig gegenüber dynamischen Magnetfeldern. Dynamische Magnetfelder sind häufig künstlich erzeugt und treten durch Transformatoren, stromdurchflossene Leiter oder andere magnetische Quellen auf. Es ist daher äußerst wichtig, den Kompass an einem Ort zu platzieren, wo er so wenig dynamischen Störeinflüssen wie möglich ausgesetzt ist.

3.1.3.4 Luftdrucksensor

Ein Luftdrucksensor misst den Luftdruck einer Umgebung, in dem er sich befindet. Der Luftdruck auf Meereshöhe beträgt rund 1013 mbar. Aufgrund von Umwelteinflüssen, wie dem Vorüberziehen von Hoch- oder Tiefdruckgebieten, ändert sich jedoch dieser Referenzwert von 1013 mbar. Dadurch ist keine korrekte Bestimmung der Höhe über längere Zeit möglich.

Anhand der barometrischen Höhenformel [Aac] lässt sich aus diesem Messwert die Höhe über dem Meeresspiegel bestimmen.

$$p_h = p_0 \cdot e^{\left(-\frac{p_0 \cdot g \cdot h}{p_0}\right)}, \quad \text{mit } g = 9.81 \left[\frac{m}{s^2}\right] \wedge p_0 = \text{Luftdruck in Bodenhöhe} \quad (3.6)$$

Eine vereinfachte Form der barometrischen Höhenformel ist der barometrische Höhenstufe. Die barometrische Höhenstufe beschreibt die Höhendifferenz, bei der der Luftdruck um 1 hPa abnimmt [dlr]. Auf Höhe des Meeresspiegels liegt die barometrische Höhenstufe bei ungefähr 8 m. Bei zunehmender Höhe vergrößert sich die barometrische Höhenstufe aufgrund der abnehmenden Dichte des Mediums. Da die Flugzeit eines Multikopters durchschnittlich ca. 10 bis 20 Minuten beträgt, kann dies vereinfacht als statischer Fall angenommen werden, dass sich die Umwelteinflüsse innerhalb dieser kurzen Zeitspanne nicht sonderlich stark verändern. Wird diese Annahme getroffen, kann zum einen gesagt werden, dass eine konstante Aussage über den Luftdruck getroffen werden kann. Zum anderen kann gesagt werden, dass Barometer einen absoluten eindimensionalen Messwert im Bezug auf die Höhe über Normal-Null (Meeresspiegelhöhe) liefern, bei der der Sensor beim Einschalten kalibriert wurde.

Im Inneren eines Luftdrucksensors befindet sich eine Kapsel, in der sich ein Referenzdruck befindet. Diese Kapsel ist durch eine flexible Membran von der Umgebung und dem dort herrschenden Druck isoliert. Auf der Membran befindet sich ein Dehnmessstreifen, welcher in der Lage ist, den Grad der Verformung der Membran zu messen. Verändert sich der äußere Luftdruck, wird die Membran verformt, wodurch sich der Dehnmessstreifen ebenfalls



verformt und seinen Widerstandswert ändert, wodurch ein proportionaler Zusammenhang zwischen Druck- und Widerstandsänderung entsteht.

Um den Multikopter mit einer verwertbaren Höheninformation versorgen zu können, ist es nötig das Barometer vor jedem Start zu initialisieren, um die während der Initialisierung gemessene Höhe (gemessener Luftdruck) als 0-Meter-Marke setzen zu können.

3.1.3.5 Global Navigation Satellite System

Global Navigation Satellite Systems (GNSS) ist ein Oberbegriff für satellitengestützte Navigationssysteme. Beispiele für ein GNSS sind das amerikanische Global Positioning System (GPS), das russische Global Navigation Satellite System (GLONASS) und das europäische Galileo-System.

Das Prinzip hinter einem satellitengestützten Navigationssystem liegt in der Bestimmung der Signallaufzeiten und deren Laufzeitunterschiede. Die von mehreren bekannten Satelliten ausgesendeten Daten, mit exakten und synchronen Zeitstempeln aller Satelliten, werden auf einem definierten Punkt auf der Erdoberfläche empfangen und ausgewertet. Durch die Unterschiede der Signallaufzeiten und der zum Zeitpunkt des Aussendens bekannten Position der Satelliten im Orbit, lässt sich eine absolute Position auf der Erdoberfläche bestimmen. Bei einfachen GNSS-Empfängern, die für den zivilen Bereich erhältlich sind, beträgt die Genauigkeit ca. 5 - 10 m [Hea12].

Eine besondere Form ist das differentielle GNSS welches ein Referenzsignal in die Positionsbestimmung einbezieht, um die Unsicherheit der eigenen Positionsbestimmung zu verringern. Das Differenzsignal wird von einer fixen Station auf der Erde ausgesendet, deren Position mit hoher Genauigkeit bekannt ist. Ein differentieller GNSS-Empfänger besitzt ein zusätzliches Empfangsmodul, welches das Referenzsignal mit einem integrierten Empfangsmodul auswerten kann und die Positionskorrektur durchgeführt werden kann. Die Genauigkeit eines differenten GNSS nimmt mit zunehmenden Abstand zur Referenzstation ab und liegt in etwa im Intervall von 2 - 5 m [Hea12].

3.2 Regelungstechnische Grundlagen

Im folgenden Abschnitt werden Grundlagen und Begrifflichkeiten der Regelungstechnik erläutert. Es wird speziell auf einige Reglertypen und deren Anwendung eingegangen, wobei deren spezifische Vor- und Nachteile an einigen Beispielen erläutert werden.

3.2.1 Unterschied zwischen Steuerung und Regelung

Unter Steuerung ist die Veränderung eines Prozesses zu verstehen, wobei kein Wissen über die Auswirkung der Veränderung der geregelten bzw. der gesteuerten Größe in den

Prozess zurückgegeben wird. Eine Steuerung (s. Abbildung 3.7) wird daher oft auch als offener Regelkreis bezeichnet [Phi04].

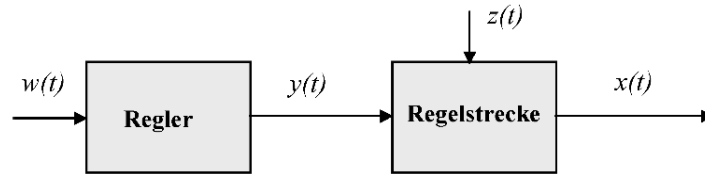


Abbildung 3.7: Darstellung eines offenen Grundregelkreises [Phi04]

Von einer Regelung wird immer dann gesprochen, wenn ein geschlossener Regelkreis vorliegt. Ein geschlossener Regelkreis zeichnet sich dadurch aus, dass ein Vergleich zwischen dem angestrebten Sollwert und dem momentanen Istwert der Regelung durchgeführt wird. Im Folgenden wird genauer auf Regler und die internen Strukturen eines geschlossenen Regelkreises eingegangen.

3.2.2 Komponenten und Signale in Regelkreisen

In Abbildung 3.8 ist ein geschlossener Grundregelkreis dargestellt. Der Grundregelkreis besteht aus den Grundkomponenten eines Reglers, einer Regelstrecke und dessen Rückführung, welche wiederum als Eingang in den Regler zurückgeführt wird. Durch die Rückführung ist sichergestellt, dass der Regler auf Veränderungen reagieren kann.

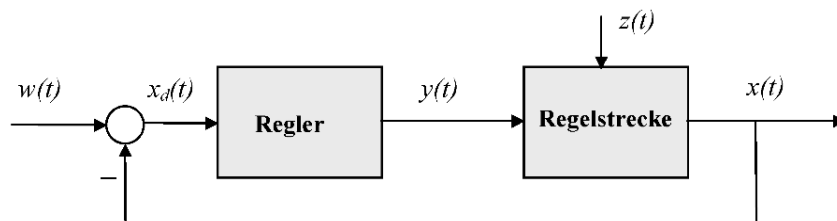


Abbildung 3.8: Darstellung eines geschlossenen Grundregelkreises [Phi04]

Um diese Struktur jedoch genau verstehen zu können, ist es notwendig, die Signale und deren Fluss innerhalb eines Regelkreises (s. Abbildung 3.8) zu kennen.

3.2.2.1 Regelgröße

Die Regelgröße $x(t)$ ist der Ausgang des Regelkreises (Istwert) und gleichzeitig die Prozessgröße, die von einem Messwertempfänger (Sensor) gemessen wird und über die Rückführung wieder in den Regelkreis eingekoppelt wird.

3.2.2.2 Führungsgröße

Als Führungsgröße $w(t)$ wird der Zustand des Systems beschrieben, welcher sich (mög-

lichst genau) durch die Regelung einstellen soll. Die Führungsgröße wird von außen in den Regelkreis eingespeist und besitzt einen definierten Sollwert

3.2.2.3 Regelfehler

Der Regelfehler $e(t)$ – hier in Abbildung 3.8 als $x_d(t)$ bezeichnet – bildet sich aus der Differenz zwischen Führungsgröße und Regelgröße, gemäß folgender Formel:

$$e(t) = x_d(t) = w(t) - x(t) \quad (3.7)$$

Der auf diese Weise errechnete Regelfehler dient als Eingang für den Regler. Ziel einer Regelung ist es, den Regelfehler zu minimieren, sodass er sich im Idealfall zu Null ergibt. Dies ist jedoch nur eine rein mathematische Betrachtung. In der Praxis ist es unmöglich, dass der Regelfehler exakt Null wird, welches beispielsweise durch Messrauschen der Sensoren zu begründen ist.

In einer praktischen Sicht auf den Regelfehler wird ein Toleranzband um den Endwert herum definiert, in dem sich schließlich der Regelfehler bewegt.

3.2.2.4 Stellgröße

Als Stellgröße $y(t)$ wird der Ausgang des Reglers bezeichnet. Die Stellgröße ist gleichzeitig auch der Eingang in die Regelstrecke und wird durch einen Aktuator physikalisch in die Strecke eingekoppelt. Im Falle einer Fluglageregelung wird die Regelstrecke durch den Multikopter und dessen physikalisches Modell substituiert.

Mit Hilfe dieser Signale ist es möglich, die Funktionsweise und die Wirkung des Reglers und der Regelstrecke genauer zu beschreiben. Außerdem kann die Wirkung einer Störung auf die Regelstrecke erläutert werden.

3.2.2.5 Regler

Ein Regler bildet den Betrag des eingehenden Regelfehlers $e(t)$ auf die Stellgröße $y(t)$ ab. Das grundlegende Ziel eines Reglers besteht in der Anpassung der Regelgröße $x(t)$ an die Führungsgröße $w(t)$. Ein Regler kann auf mehrere Arten implementiert werden. Eine nähere Beschreibung der Reglertypen wird in Kapitel 3.2.3 gegeben.

3.2.2.6 Regelstrecke

Als Regelstrecke ist das System zu verstehen, welches durch den Regler beeinflusst werden soll. In die Strecke wird die Stellgröße $y(t)$ hineingegeben, worauf die Strecke mit der Regelgröße $x(t)$ antwortet. Die Regelgröße ist die Prozessgröße des Systems, welche von den Sensoren aufgenommen wird und als Rückführung in die Bildung des Regelfehlers erneut einfließt. Bei einer Diskretisierung des Reglers bedeutet dies, dass die $i - 1$ 'te Regelgröße in die i 'te Regelschleife eingebunden wird.

Bei der Implementierung einer Fluglageregelung eines Multikopters ist der mechanische,

elektronische und allgemein der physikalische Aufbau des Multikopters selbst als Regelstrecke zu verstehen. So antwortet beispielsweise ein Multikopter mit einem doppelt so großen Motor-zu-Motor Abstand anders, als ein Multikopter mit einfachem Motor-zu-Motor Abstand, welcher den selben Regler mit den selben Reglerparametern verwendet.

3.2.2.7 Störfunktion

Um äußere Einflüsse, wie beispielsweise Umwelteinflüsse in das Modell des zu regelnden Systems einbinden zu können, wird die Störfunktion $z(t)$ in den Regelkreis integriert. Die Störfunktion greift an der Regelstrecke an, welche die Regelgröße direkt beeinflusst. Ein Beispiel für eine Störfunktion, die auf einen Multikopter einwirkt, ist ein auftretender Seitenwind. Dieser Seitenwind übt eine Kraft auf den Multikopter aus, um bewirkt so eine Änderung seines Zustandes (Kippen um eine oder mehrere Achsen). Dadurch wird eine Änderung der Regelgröße hervorgerufen, welche von einem geeigneten Sensor aufgenommen wird und in den Regelprozess einfließt. Auf diese Weise wird eine neue Stellgröße des Reglers berechnet, welche der Störfunktion entgegenwirkt.

Wird der einwirkende Seitenwind als Böe verstanden, welche schlagartig aufhört eine Kraft auf den Multikopter auszuüben, so wird der Multikopter beim Abflauen der Böe ebenfalls um eine oder mehrere Achsen kippen. Bei diesem Vorgang entfällt das Wirken der Störfunktion, auf die sich der Regler eingestellt hat, schlagartig. Diese Änderung der Regelgröße wird ebenfalls von einem Sensor erfasst, wodurch die Stellgröße im nächsten Zyklus des Regelkreises erneut angepasst wird.

3.2.3 Reglertypen

In den folgenden Abschnitten werden einige Regler vorgestellt und deren Stärken und Schwächen dargestellt. Dabei wird genau auf die jeweilige Funktionsweise eingegangen. Die verwendeten Formeln beziehen sich auf digitale Regler, daher wird das zeitkontinuierliche t durch ein zeitdiskretes k ersetzt. Zur Beschreibung der Reaktion eines Reglers auf die Veränderung der eingehenden Größe wird der so genannte Einheitssprung σ verwendet [Phi04].

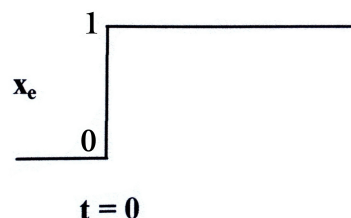


Abbildung 3.9: Einheitssprung – dient zum Erhalt der Sprungantwort [Phi04]

Bei dem Einheitssprung wechselt der Betrag der Führungsgröße sprunghaft von 0 auf 1, wie in Abbildung 3.9 zu sehen ist. Mathematisch kann der Einheitssprung wie folgt beschrieben werden:

$$\sigma = \begin{cases} 1, & \text{für } t \geq 0 \\ 0, & \text{für } t < 0 \end{cases} \quad (3.8)$$

Bei der theoretischen Betrachtung des Einheitssprungs der Führungsgröße hat diese vor dem Zeitpunkt $t = 0$ den Betrag 0. Zum Zeitpunkt $t = 0$ wechselt der Betrag sprunghaft auf 1, wobei die Steigung der Funktion σ zu diesem Zeitpunkt unendlich groß ist. In der Praxis kann keine unendlich große Steigung einer physikalischen Größe erzeugt werden, daher wird der Einheitssprung als eine Rampenfunktion angesehen, welche nach einer gewissen Zeitspanne Δt den Endwert von 1 erreicht.

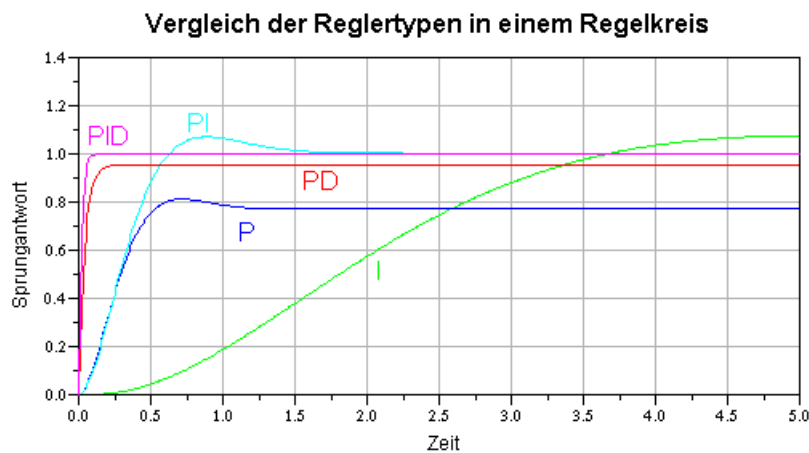


Abbildung 3.10: Sprungantworten verschiedener Reglertypen [rob]

3.2.3.1 P-Regler

Der einfachste Regler ist der so genannte proportionale Regler (P-Regler). Dieser Regler verstärkt den eingehenden Regelfehler $e(k)$ mit einem Proportionalitätsfaktor k_P , um die Stellgröße $y(k)$ zu generieren. Die mathematische Modellierung eines P-Reglers ist in der folgenden Formel dargestellt:

$$y_P(k) = K_P \cdot e(k) \quad (3.9)$$

Aus der Formel (3.9) ist ersichtlich, dass ein großer Regelfehler eine große Stellgröße hervorruft, während ein geringer Regelfehler nur eine kleine Stellgröße produziert. Wie in Abbildung 3.10 ersichtlich, führt ein P-Regler mit einem großen Verstärkungsfaktor zu einem starken Schwingen mit leicht gedämpfter Amplitude. In besonderen Fällen kann es auch zu einer sich aufschaukelnden Schwingung mit steigender Amplitude kommen. Schwingt ein Regler mit konstanter Frequenz und Amplitude, so spricht man vom aperi-



odischen Grenzfall. Bei der Verwendung eines P-Reglers kommt es sehr häufig dazu, dass der Endwert der Regelung nicht erreicht wird. Das bedeutet, dass sich die Regelgröße auch für den Fall $\lim_{k \rightarrow \infty}$ nicht der Führungsgröße anpasst und eine Regeldifferenz entsteht, die auch als Schleppabstand bezeichnet wird.

3.2.3.2 D-Regler

Ein differenzierender Regler (D-Regler) bildet im zeitkontinuierlichen die Ableitung des Eingangssignals $e(t)$ zu einem bestimmten Zeitpunkt t . Im zeitdiskreten wird die Differenz aus dem aktuellen Regelfehler $e(k)$ und den vorherigen Regelfehler $e(k-1)$ gebildet.

$$y_D(k) = K_D \cdot [e(k) - e(k-1)] \quad (3.10)$$

Aus Gleichung 3.10 ist ersichtlich, dass eine starke Änderung des Regelfehlers eine große Stellgröße hervorruft. Die Intensität mit der der D-Anteil in die Regelung einfließt, wird mit Hilfe des Differenzierbeiwerts K_D festgelegt. Ein D-Regler erzeugt ebenfalls wie der P-Regler, einen Schleppabstand zwischen Führungsgröße und Stellgröße, wenn die Änderung zwischen k' ten und $k-1'$ ten Regelfehler gleich Null ist. Im Gegensatz zum P-Regler benötigt der D-Regler einen definierten Anfangswert für $e(k-1)$ im Startzustand wenn gilt: $k=0$ und die Differenz zwischen k' ten und $k-1'$ ten Regelfehler gebildet wird.

3.2.3.3 PD-Regler

Um die Vorteile des proportionalen und differentiellen Reglers zu vereinen, können diese zusammenschaltet werden.

In folgender Gleichung ist die mathematische Modellierung eines diskreten PD-Reglers dargestellt:

$$y_{PD}(k) = K_P \cdot e(k) + K_D \cdot [e(k) - e(k-1)] \quad (3.11)$$

Die Sprunganwort eines PD-Reglers ist in Abbildung 3.10 dargestellt. Da weder der P- noch der D-Regler einen statischen Regelfehler erkennen, erzeugt der PD-Regler ebenfalls wieder einen Schleppabstand zwischen Führungs- und Regelgröße.

3.2.3.4 I-Regler

Bei dem integrierenden Regler (I-Regler) wird der Regelfehler $e(t)$ im zeitkontinuierlichen integriert, welches einer Summation des zeitdiskreten Regelfehlers $e(k)$ entspricht. Der I-Regler kann durch seine Funktion als ein Speicher angesehen werden, welcher den letzten Wert der Regelgröße $x(k)$ speichert. Die Intensität mit der der integrale Anteil in die Regelung einfließt, hängt von dem Betrag des Integrierbeiwerts K_I ab.

$$y_I(k) = K_I \cdot \sum_{i=0}^{k-1} e(k) \quad (3.12)$$

Das Wirkprinzip eines Integrators verleiht dem I-Regler eine Art von Trägheit, welches sich in der Dämpfung hochfrequenter Eingangssignale niederschlägt.

Der integrale Regler benötigt ebenfalls einen definierten Anfangswert für die Integration bzw. Summation. Zusätzlich ist es jedoch bei der Implementation eines integralen Anteils wichtig, dass eine Begrenzung der Stellgröße vorgenommen wird, sodass der errechnete Betrag die zulässige maximale Stellgröße nicht überschreitet und so gegebenenfalls eine Beschädigung der Regelstrecke hervorruft. Des Weiteren würde es sehr lange dauern, bis der I-Regler einen extrem hohen Betrag durch seine träge Eigenschaft ausgeregelt hat. Ein I-Regler ist in der Lage, den auftretenden Schleppabstand eines P- bzw. D-Anteils zu eliminieren und somit die Differenz zwischen Führungsgröße und Regelgröße auszugleichen. Eine Sprungantwort eines I-Reglers durch die Anregung eines Einheitssprungs ist in Abbildung 3.10 dargestellt.

3.2.3.5 PI-Regler

Wird der proportionale Anteil mit dem integralen Anteil zu einem Regler kombiniert, entsteht der so genannte PI-Regler. Das Hauptmerkmal dieses Reglers besteht in der Verbindung der Eigenschaft des P-Reglers, schnell auf eine Änderung des Regelfehlers reagieren zu können mit der Eigenschaft des I-Reglers, den Schleppabstand zwischen Führungs- und Regelgröße zu minimieren.

In der Gleichung (3.13) ist die mathematische Beschreibung des PI-Reglers dargestellt.

$$y_{PI}(k) = K_P \cdot e(k) + K_I \cdot \sum_{i=0}^{k-1} e(k) \quad (3.13)$$

Die Sprungantwort eines PI-Reglers ist in Abbildung 3.10 dargestellt. Die Verwendung eines PI-Reglers eignet sich genau dann besonders gut, wenn die zu regelnde Strecke ein differenzierendes Verhalten aufweist.

3.2.3.6 PID-Regler

Bei dem PID-Regler handelt es sich um eine Kombination des proportionalen, integralen und differentiellen Anteils zu einem gemeinsamen Regler. Dieser Regler ist der in der Industrie am weitesten verbreitete Regler. Durch die der Gleichung (3.14) dargestellte mathematische Modellierung des PID-Reglers ist es möglich, den differentiellen Anteil abzuschalten in dem das K_D auf Null gesetzt wird. Die Grafik Abbildung 3.10 zeigt die Sprungantwort eines PID-Reglers.

Im Folgenden ist die mathematische Beschreibung des PID-Reglers dargestellt.

$$y_{PID}(k) = K_P \cdot e(k) + K_I \cdot \sum_{i=0}^{k-1} e(k) + K_D \cdot [e(k) - e(k-1)] \quad (3.14)$$

Durch den im PID-Regler eingebetteten PD-Regler ist es möglich sehr schnell auf Änderungen des Regelfehlers $e(k)$ zu reagieren. Der PID-Regler ist jedoch durch den enthaltenen integralen Anteil nicht ganz so schnell wie ein reiner PD-Regler. Jedoch ist es mit einem PID-Regler im Gegensatz zum reinen PD-Regler möglich, den während einer Regelung auftretenden Schleppabstand durch den I-Anteil zu minimieren.

3.2.4 Regelung von Multikoptern

Durch das Implementieren von Regelungsstrukturen auf einer geeigneten Hardware ist es möglich ein System, wie beispielsweise ein Multikopter, zu regeln. Für einen modellgetriebenen Entwicklungsansatz ist es von Vorteil die Regelungsstrukturen digital aufzubauen. Im gleichen Zug ist es ebenfalls sinnvoll, eine digitale Sensorik zu verwenden, da dabei zusätzliche Hardware zur A/D-Wandlung entfällt.

Der Sollwert für die Regelung wird von der Funkfernbedienung vorgegeben, welcher über den Empfänger in das Regelungssystem eingebunden wird. Es kann unter verschiedenen Einzelkomponenten unterschieden werden, welche sich zu einem gesamten Regelungssystem zusammensetzen. Die Einzelkomponenten können beispielsweise aus einer Lage-, Höhen-, Positions- und Orientierungsregelung bestehen. Es eignet sich nicht jeder Sensor um jede der beschriebenen Regelungen durchzuführen. Für eine spezifische regelungstechnische Aufgabe muss zum einen ein passender Regler ausgesucht werden. Zum anderen ist es ebenfalls wichtig, einen passenden Sensor auszuwählen, dessen Messwert für die regelungstechnische Aufgabe einen adäquaten Nutzen hat. Es kann unter Umständen notwendig sein, dass Messwerte mehrerer Sensoren durch die so genannte Sensor-Fusion zusammengeführt werden. Dies ist beispielsweise bei der Anwendung einer Lageregelung der Fall. Hierbei werden die Daten eines Beschleunigungssensors und eines Gyroskops fusioniert um eine beständige Regelung zu erzeugen. Hierbei wird die Gewichtung der Messwerte des Beschleunigungssensors angehoben, wenn sich der Multikopter nur sehr langsam bewegt oder auf der Stelle schwebt. In diesem Zustand werden die kontinuierliche Drift der Gyroskope durch den Beschleunigungssensor korrigiert. Bei zunehmender Bewegungsgeschwindigkeit und großem Betrag der ausgegebenen Winkelbeschleunigung der Gyroskope, verschiebt sich die Gewichtung weiter zum Gyroskop hin, sodass dessen Messwerte größeren Einfluss an der Lageregelung gewinnen. Bei einer Lageregelung sollten Regler mit einem schnellen Führungsverhalten verwendet werden. Aus den in Kapitel 3.2.3 vorgestellten Reglertypen sollte ein PI- oder PID-Regler ausgesucht werden.

Zur Umsetzung einer Höhenregelung ist es sinnvoll einen PI-Regler zu verwenden, welcher bei einer durch den Piloten herbeigeführten Höhenänderung den Fehler des integralen Anteils arretiert und erst dann wieder freigibt, wenn der Steig- oder Sinkflug beendet ist. Dadurch wird sichergestellt, dass sich der Fehler nicht aufsummiert, wenn eine gewollte Höhenänderung durch den Piloten ausgeübt wird. Als Sensorik eignet sich für diese Aufgabe eine Fusion aus einem Barometer und dem Z-Anteil eines Beschleunigungssensors. Das



Barometer gibt hierbei eine absolute Höhe im Bezug auf die bei der Kalibrierung getroffenen Höhe der Startposition an. Der Z-Anteil des Beschleunigungssensors, welcher sich entlang des Vektors der Erdbeschleunigung richtet, wird dazu verwendet, um künstlich herbeigeführte Luftdruckschwankungen oder das Rauschen des Barometers zu unterdrücken. Bei einer Implementierung einer Orientierungsregelung (gemeint ist das Gieren um die Z-Achse) bietet sich eine Fusionierung aus den Daten eines Magnetometers und den Messwerten der Winkelbeschleunigung um die Z-Achse eines Gyroskops an. Das Magnetometer sorgt dafür, dass die absolute Ausrichtung im Bezug auf den magnetischen Norden gegeben ist. Statische Magnetfelder können gemessen und bei der Regelung heraus gerechnet werden. Die sich dynamisch verändernden Magnetfelder können durch das Einbeziehen der Daten des Gyroskops kompensiert werden. Wenn sich der Messwert des Magnetometers verändert, obwohl das Gyroskop keine Veränderung anzeigt, kann davon ausgegangen werden, dass keine Änderung der Ausrichtung stattgefunden hat, da sich der Multikopter dafür um die Z-Achse rotiert haben muss, welches durch die Daten des Gyroskopes widerlegt ist. Für diese Art der Regelung ist ein PI-Regler ebenfalls gut geeignet.

Um die Einzelkomponenten zusammenführen zu können und die Motoren entsprechend der Stellwerte ansprechen zu können, ist ein so genannter Mixer notwendig. Die Aufgabe des Mixers besteht beispielsweise darin, die drei Werte der Lageregelungen entlang jeder einzelnen Raumachse derart zu verrechnen, dass daraus für jeden Motor eine einzige Stellgröße entsteht, die über die Motortreiber in eine Bewegung umgesetzt werden kann.

3.3 Digitale Bildverarbeitung

Da die nicht sicherheitskritische Task einen Algorithmus zur Objektverfolgung mit Hilfe einer Kamera darstellt, werden in diesem Abschnitt die Grundlagen und möglichen Arten der digitalen Objekterkennung in Videosequenzen näher beschrieben. Die Erkennung eines Objektes in der digitalen Bildverarbeitung bildet die Grundlage der Objektverfolgung, die im Anschluss erklärt wird.

3.3.1 Methodiken der Objekterkennung

Die digitale Objekterkennung ist von großem Interesse in vielen Bereichen des Alltags. Von der Erkennung von Gesichtern bei der Digitalkamera, bis hin zu dem autonom fahrenden Auto von Google.[Sch14] Überall spielt der Erkennungsprozess eine große Rolle. Es ist außerdem immer wieder ein Forschungsthema in vielen Bereichen. Beispielsweise ist die automatische Erkennung von Tumoren ein Forschungsgebiet der Medizin.[LL04]

Der Objekterkennungsprozess ist in der Computertechnik eine komplexe Aufgabe. Im Gegensatz zum Menschen, bei dem der Objekterkennungsprozess beim Betrachten eines Bil-

des, sehr intuitiv und ohne Probleme abläuft, kann der Computer seine Informationen nur aus den einzelnen Pixeln beziehen.

Der Mensch hingegen, kann beim Betrachten eines Bildes Informationen aus seinem Vorwissen mit einfließen lassen.



Abbildung 3.11: Optische Täuschung: Dalmatiner [dog]

Für einen Computer ist Abbildung 3.11 nur eine Anordnung von weißen und schwarzen Flecken, ebenso auf den ersten Blick für den Menschen. Betrachten wir nun dieses Bild mit dem Wissen, dass dort ein Hund abgebildet ist, kann dieser ohne Probleme und mit Umrissen erkannt werden. Das Gehirn stellt diese Umrisse im Kopf wieder her. Dies setzt aber voraus, dass der Mensch schon einmal zuvor einen Hund gesehen hat. Der Computer hingegen ist von den einzelnen Pixeln abhängig. Diese stellen streng genommen nur die Lichtintensität des betrachteten Bildes dar, denn die Aufnahme eines Bildes oder Videos zur digitalen Weiterverarbeitung erfolgt üblicherweise mit Hilfe eines CMOS-Sensors. Dieser besteht aus Fotodioden, Kondensatoren, Verstärkern und Datenleitungen. Dabei wird eine Spannung mithilfe des Photoeffekts erzeugt. Diese ist proportional zur eingehenden Lichtmenge. Somit können solche Sensoren nur zwischen hell und dunkel unterscheiden. Die Farben werden mit Hilfe von roten, grünen und blauen Farbfiltern, dem sogenannten Bayer-Filter, erzeugt. Diese Filter lassen nur die bestimmten Lichtfarben durch und leiten diese auf CMOS-Sensoren weiter.

Die Helligkeit und Farbe des Objektes hängen somit stark von dessen Reflexionsvermögen und der Beleuchtung ab. Es ist schwierig für einen Computer zwischen Objekten zu unterscheiden, wenn es in dem Bild viele ähnliche Objekte oder Teilverdeckungen gibt. Hier braucht es eine andere Vorgehensweise als beim Menschen. Beispielsweise ist eine starke Änderung der Helligkeit oder Farbe, zweier benachbarter Pixel, ein Hinweis für den Computer, dass die Pixel zu verschiedenen Objekten gehören. Auch die schrittweise Änderung der Helligkeit mehrerer Pixel kann ein Hinweis auf Schattierungen sein. Der Computer hängt somit stark von den Konfigurationen des Menschen ab und ist eher eingeschränkt lernfähig.



Das einfachste Vorgehen um ein Objekt zu erkennen wäre, dass zu erkennende Objekt zu fotografieren und dieses dann mit einem anderen Bild, worauf sich dieses Objekt befindet, zu vergleichen. Dies ist jedoch nicht sehr effizient, da das Objekt nur erkannt wird, wenn es auf dem fotografierten Bild ähnlich dargestellt ist und sich darüber hinaus in der gleichen Entfernung befindet. Es benötigt einen anderen Ablauf, sodass das Objekt auch erkannt wird, wenn sich der Blickwinkel, die Entfernung oder die Ausrichtung des Objektes ändert. In der digitalen Objekterkennung gibt es jedoch keinen einheitlichen Prozess zu Verarbeitung, der immer ausgeführt wird, sondern nur einen Leitfaden, der je nach Anwendung angepasst werden muss. Dies sind Schritte, die nacheinander ausgeführt werden und in jedem Schritt die Komplexität bzw. den Informationsgehalt erhöhen.

Im ersten Schritt werden die Pixeldaten von einer Kamera aufgenommen. Dabei ist der Begriff Kamera sehr weitläufig, denn es kann sich, je nach Anwendungsgebiet, um eine Farbkamera, eine Schwarzweißkamera oder eine Infrarotkamera handeln.

Sind die Pixeldaten aufgenommen worden, so muss im nächsten Schritt das Bild vorverarbeitet werden. Hier werden möglichst viele Störungen wie Rauschen⁹ und ungleichmäßige Bildausleuchtung entfernt. Bei echtzeitfähigen Systemen werden FPGAs eingesetzt, worauf Hardwarekomponenten implementiert werden, die diese Vorverarbeitungsschritte realisieren. Diese Art der Vorverarbeitung ist deutlich schneller, als eine Vorverarbeitung durch einen Prozessor.

Der nächste Schritt wird als Segmentierung bezeichnet. Hier reduziert man die Bildinformationen auf relevante Objekte. Meist werden hiermit die Objekte vom Hintergrund getrennt. Dies geschieht zum Beispiel durch Binarisierung. Ist dies nicht möglich, kann auch nach anderen Eigenschaften wie Form, Lage, oder Größe der relevanten Objekte gesucht werden.

Die Merkmalsextraktion ist der nächste Schritt. Hier werden die gefundenen Objekte auf bestimmte Merkmale untersucht. Darunter wird wiederum zwischen deskriptive und abstrakte Merkmale unterschieden. Deskriptive Merkmale sind anschauliche Eigenschaften wie z. B. Flächen, Umfang oder Farbe. Die abstrakten Merkmale werden meistens aus Transformationen gewonnen wie z. B. der zweidimensionalen Fouriertransformation. Wurden die Objektmerkmale gefunden, werden diese gespeichert.

Als letzter Schritt steht die Klassifizierung der Objekte an. Hier werden die Merkmale betrachtet und einer Objektklasse zugeordnet. Diese Zuordnung findet mit wissensbasierten Systemen¹⁰ statt. Dies sind meist numerische Klassifikationsverfahren oder neuronale Netze. Anhand der klassifizierten und somit erkannten Objekte wird eine Ausgabe wie ein Stellbefehl oder eine visuelle Darstellung, getätigt. Das Abbildung 3.12 veranschaulicht noch einmal dieses Vorgehen. [? LL04, Bre13]

⁹Unerwünschte Bildfehler, die während der Aufnahme des Bildes entstehen.

¹⁰Datenbanksysteme, die Wissen modellieren und zur Verfügung stellen

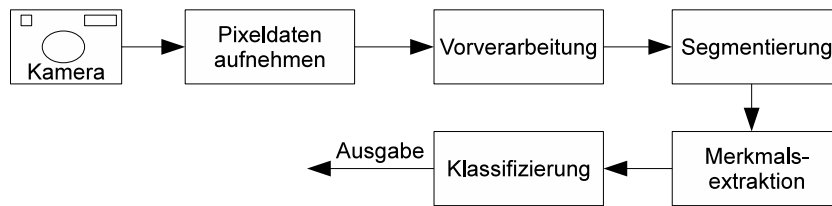


Abbildung 3.12: Ablauf der digitalen Objekterkennung.

3.3.2 Methodiken der Objektverfolgung

Da nun ein grober Überblick über die Vorgehensweise der digitalen Objekterkennung unter Anwendung eines Computersystems gegeben worden ist, werden in diesem Abschnitt einige Algorithmen dargestellt, um mit Hilfe von Pixeldaten Objekte zu erkennen und diese dann zu verfolgen. Die Objektverfolgung ist im Grunde nichts anderes, als das Erkennen von Objekten in einer Bildfolge. Eine Besonderheit liegt darin, dass nach der ersten Objekterkennung die Position des Objektes bekannt ist und für die nächsten Berechnungen mit eingebracht werden kann. Die Berechnungskomplexität kann verringert werden, indem nur noch der Bildausschnitt betrachtet wird, wo sich das Objekt voraussichtlich befindet. Die Abbildung 3.13 zeigt den schematischen Ablauf.

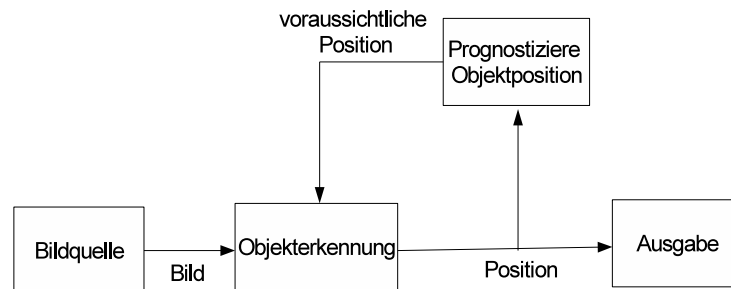


Abbildung 3.13: Der schematische Ablauf der Objektverfolgung.

Es gibt viele verschiedene Ansätze Objekte zu verfolgen. Ein Ansatz ist zum Beispiel das Template Matching. Dabei wird ein Template, d.h. eine Abbildung des gesuchten Objektes, mit den Input verglichen und dadurch nach dem Objekt gesucht. Wurde das Objekt gefunden, wird nur noch in einem Bereich gesucht, wo sich das Objekt in Zukunft voraussichtlich befindet, der sogenannten Region of Interest. Dabei ist das Suchen des Templates in dem Originalbild mit Hilfe der Kreuzkorrelation realisiert. Dies kann unter anderem auch im Frequenzbereich nach einer Fouriertransformation ausgeführt werden. Das Tem-

plate Matching im Frequenzbereich bringt den Vorteil, das es unempfindlich gegenüber Bildrauschen ist. Der Nachteil vom Template Matching ist, dass sich die Entfernung sowie die Rotation des gesuchten Objektes nicht verändern darf. Ansonsten wird das Objekt nicht gefunden. Abhilfe schaffen hier nur mehrere Templates, die verschiedene Rotationen des gesuchten Objektes darstellen. Diese müssen, wenn das gesuchte Objekt rotiert, alle ausprobiert werden.

Ein weiterer Ansatz ist zum Beispiel die Berechnung des Gleichgewichtspunktes eines Objektes. Dafür ist es nötig, dass sich das gesuchte Objekt von dem Hintergrund abhebt, sodass dies leicht erkannt werden kann. Um dies zu veranschaulichen, wird der Objekterkennungsprozess anhand eines Beispiels durchgeführt. Das Ausgangsbild wird in Abbildung 3.14 dargestellt und wurde mit einer Kamera aufgenommen und automatisch im JPEG Format gespeichert. Somit ist der erste Schritt der Verarbeitungskette, die Pixeldaten aufnehmen, abgeschlossen. Verfolgt und erkannt werden soll das rote Kart mit Mario als Fahrer.

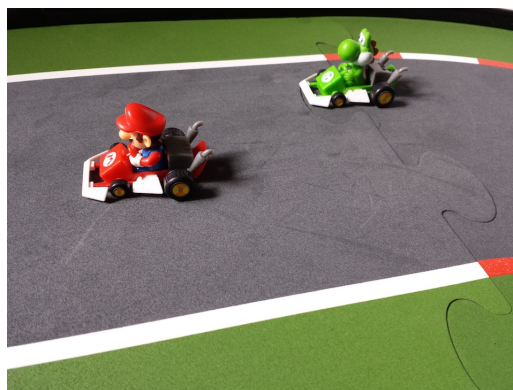


Abbildung 3.14: Das Ausgangsbild.

Nun muss eine Methode gefunden werden, wie das Objekt verfolgt werden kann. Hier stehen die Farben ins Auge. Für dieses Bild wäre es am Besten, wenn die blaue Latzhose von Mario als Erkennungsmerkmal ausgewählt wird, da die blaue Farbe im Bild einzigartig ist. Jedoch ist das Blau nur in bestimmten Winkeln sichtbar und somit für die Objektverfolgung mit Hilfe einer Kamera nicht geeignet. Die rote Farbe jedoch sticht heraus und ist auch aus verschiedenen Winkeln sichtbar. Also ist dies ein guter Ausgangspunkt. Leider ist die rote Farbe nicht nur beim Kart vorhanden sondern auch am Straßenrand. Dies ist ein Problem und muss bei dem Objekterkennungsprozess beachtet werden.

Im Vorverarbeitungsschritt wird das Bild vorbereitet. In Abbildung 3.15 wird die Ausgabe nach dem ersten Filter für die Vorverarbeitung dargestellt.

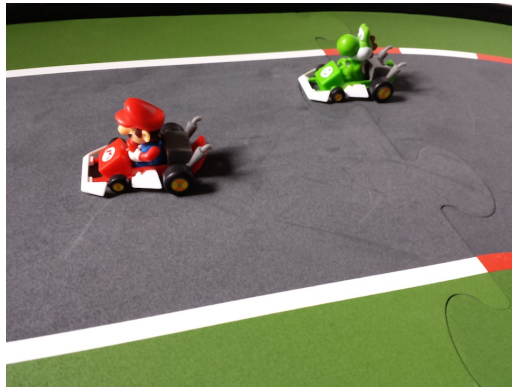


Abbildung 3.15: Bild nach Anwendung des Median-Filters.

Es wurde ein Median-Filter angewendet um Bildrauschen zu verringern und die Farben zu verdeutlichen. Der Median-Filter betrachtet die Farbe eines Pixels und die der benachbarten Pixel, berechnet die durchschnittliche Farbe der benachbarten Pixel und gibt diese dem aktuellen Pixel. Die Anzahl der betrachteten Nachbarpixel wird vorher festgelegt. Im nächsten Schritt der Vorverarbeitung wird die rote Farbe aus dem Originalbild herausgefiltert. Abbildung 3.16 zeigt die Rotwerte des Bildes.

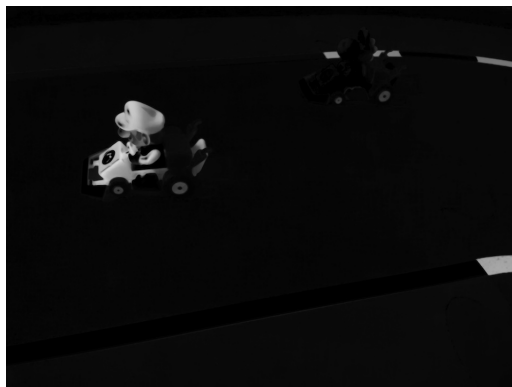


Abbildung 3.16: Die roten Farbwerte

Nun ist der Vorverarbeitungsschritt abgeschlossen. Es folgt die Segmentierung. Um möglichst viele rote Farbwerte die unrelevant sind zu löschen, werden die roten Farbwerte binarisiert also nur als schwarz oder weiß dargestellt. Dabei wird ein Farbbereich festgelegt der als Rot angesehen wird. Dieser wird weiß eingefärbt. Alle Farbwerte, die nicht in diesem Bereich liegen, werden schwarz eingefärbt. Abbildung 3.17 zeigt die Ausgabe dieses Schrittes.



Abbildung 3.17: Binarisierter Rotwert

Hier wird deutlich das nicht nur das Kart binarisiert wurde, sondern auch die Seitenstreifen. Dies lässt sich durch das sogenannte Opening minimieren. Es werden morphologische Operation auf dem Bild ausgeführt. Dabei werden im ersten Schritt die weißen Flächen durch Erosion verkleinert bzw. ganz gelöscht und danach durch die Dilatation die noch übrig gebliebenen weißen Flächen wieder vergrößert. Somit wird die Anzahl der nicht relevanten Objekte verringert. Die Abbildungen 3.18 und 3.19 zeigen die daraus resultierenden Ausgaben.

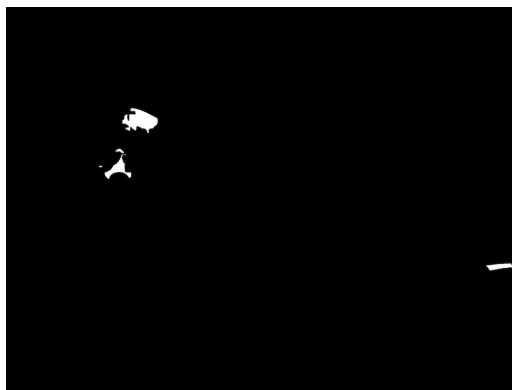


Abbildung 3.18: Die Ausgabe nach der Erosion.



Abbildung 3.19: Ausgabe nach der Dilatation.

Wichtig dabei ist, dass durch die Dilatation die Fläche, die von Interesse ist, groß und vor allem zusammenhängend ist. Dies ist die Vorbereitung für den nächsten Schritt, die Merkmalsextraktion. Hier wird der so genannte BLOB (Binary Large Object) Algorithmus angewendet. Dieser durchsucht das komplette Bild Pixel für Pixel und prüft das Pixel, welches sich links, oberhalb des aktuell betrachteten Pixels. Ist dieses Pixel weiß, gehören die Pixel zusammen und es wird es in einer Tabelle gespeichert. Wenn einer von den benachbarten Pixel weiß ist, so gehört der aktuelle Pixel auch zu dieser Fläche. Somit findet dieser Algorithmus, in binarisierten Bildern, zusammenhängende Segmente und speichert die Position und Größe. Aus diesem Grund findet der Algorithmus, angewendet auf das Beispiel, zwei zusammenhängende Flächen. Dies wird in der Abbildung 3.20 noch einmal verdeutlicht, in der die erkannten Segmente farblich markiert wurden.



Abbildung 3.20: Erkannte Segmente nach dem BLOB-Algorithmus

Als letzter Schritt folgt nun die Klassifizierung der gefundenen Segmente. Hierbei ist, wie oben erwähnt, der Gleichgewichtsschwerpunkt der gefundenen Fläche ausschlaggebend und dient als Klasseneinteilung. Eine Möglichkeit wäre ebenso das Kart durch die Anzahl der Pixel der gefundenen Segmente zu Klassifizieren. Ab einer bestimmten Anzahl an Pixel, also ursprünglich, ab einer bestimmten Menge an roter Farbe im Bild, wird infolgedessen

entschieden, dass dieses Segment das Kart darstellt. Abbildung 3.21 zeigt noch einmal das Originalbild überlagert von der Extraktion.



Abbildung 3.21: Originalbild überlagert mit der Extraktion

Nun könnte mit dieser Information visuell die Ausgabe getätigt werden, indem ein Rechteck um das erkannte Kart gezogen wird, um somit die erfolgreiche Objekterkennung anzudeuten. Für die nächste Berechnung könnte die Berechnungskomplexität verringert werden, indem nur noch der linke Teil des Bildes für die Objekterkennung herangezogen wird. Einziger Nachteil bei diesem Beispiel ist, dass bei Verdeckung des Karts nicht sichergestellt werden kann, dass das Kart erkannt wird. Das liegt daran, dass die rote Farbe nicht einzigartig im Bild ist und dieser Algorithmus somit den Seitenstreifen als Kart identifizieren könnte.

Dies ist nur ein Beispiel, wie ein möglicher Objekterkennungsprozess und Verfolgung ablaufen könnte. Es gibt viele unterschiedliche Herangehensweisen ein Objekt zu erkennen. Beispielsweise über markante Kanten etc. . Diese sind jedoch sehr anwendungsspezifisch und müssen für jede Anwendung einzeln herausgearbeitet werden, da es für die Objekterkennung nicht die eine richtige Lösung gibt.

Für dieses Projekt wird aller Voraussicht nach ein ähnliches Verfahren, wie das hier vorgestellte, zur Objektverfolgung genutzt werden. Da ein Ball verfolgt werden soll, wird voraussichtlich kein Kantentracking angewendet, sondern ein Tracking nach einer Farbe, die sich stark vom Hintergrund abhebt. Möglicherweise ist die Erosion und Dilatation der binarisierten Farbwerte auch nicht notwendig, wenn die gewählte zu trackende Farbe des Balles einzigartig auf dem Spielfeld des Szenarios vorhanden ist. Die Verwendung von einer einzigartigen Farbe im Spielfeld hätte auch den Vorteil, dass Teilverdeckungen des Balles kein Problem für das Objekttracking darstellen würden, da der Algorithmus davon ausgehen kann, dass es sich um den Ball handelt. Nur komplette Verdeckungen würden diesem Verfahren Schwierigkeiten machen, denn dann wird die Farbe nicht mehr erkannt. Hier könnte nur die Einrechnung der letzten bekannten Position und die daraus folgende Position zum erfolgreichen Tracken des Balles bei Verdeckungen führen. Je nach dem was



für eine Kamera bzw. Bildgröße verwendet wird, könnte es auch Variationen im Vorverarbeitungsschritt geben, um verschiedene Störungen wie Rauschen zu minimieren.

Zu bedenken ist weiter, dass es notwendig ist, diese Prozessschritte für eine Objektverfolgung für jeden Frame der Kamera auszuführen. Das impliziert, dass dies sehr schnell ausgeführt werden muss und je nach Bildgröße und Frames pro Sekunde Hardware voraussetzt, die viele Operationen schnell abarbeiten kann. Gerade bei der Objekterkennung gibt es Operationen, die auf jeden Pixel angewendet werden müssen. Als Beispiel sei hier der Median-Filter, mit einer Fenstergröße von 3×3 , genannt, der folglich neun Operationen pro Pixel ausführen muss. Ein Full-HD Videobild (1920×1080 Pixel) besitzt insgesamt 2.073.600 Pixel somit müssen für die Anwendung unseres Beispiel-Filters 18.662.400 Operationen für ein Frame stattfinden. Bei beispielsweise 30 Frames pro Sekunde müssen diese Operationen sowie weitere Operationen der Objekterkennung wie Segmentierung, Merkmalsextraktion und Klassifizierung alle 33,33 ms abgearbeitet sein. Für aktuelle Desktop-CPU's ist dies kein Problem, jedoch wird im Multikopter aus Gründen der elektischen Leistungseinsparung nicht solch eine CPU verbaut sein, so dass eine andere energiesparende Lösung benötigt wird. Hier bietet sich beispielsweise eine parallele Abarbeitung auf Hardwarebasis mit Hilfe eines FPGAs an. [hSF12]

3.4 Kamerapositionierung und -stabilisierung

Um ein Objekt unabhängig von der Flug- und Ausrichtung des Multikopters zu verfolgen, wird ein Subsystem benötigt, welches die individuelle Positionierung einer Kamera erlaubt. Dabei muss das Subsystem die Kamera vom Multikopter entkoppeln, sodass die Flugmanöver und die vom Fluggerät erzeugten Vibrationen während des Fluges nicht die Aufnahme der Kamera beeinflussen. Im Folgenden werden unterschiedliche Möglichkeiten aufgezeigt, um das Kamerabild während des Multikopterfluges zu stabilisieren und auszurichten.

3.4.1 Optische Bildstabilisierung

Bei dieser Bildstabilisierung messen Gyroskope die Vibrationen direkt im Objektiv der Kamera und stellen diese Informationen eine Auswertelogik bereit. Basierend auf diesen Informationen steuert die Auswertelogik eine bewegliche Linse innerhalb des Objektivs an. Somit werden die Vibrationen entlang der horizontalen und vertikalen Achse der Kamera ausgeglichen.

Die drei Abbildungen 3.22, 3.23 und 3.24 veranschaulichen die Funktionsweise einer solchen Bildstabilisierung in einem Objektiv mit vier Linsen, von denen die dritte (plan-konkave) Linse beweglich ist. Die rote Linie in den Abbildungen symbolisiert den Lichteinfall (das Bild), welches während der gesamten Belichtungszeit einer Bildaufnahme mittig auf die Bildebene treffen sollte (s. Abbildung 3.22). Wird das Objektiv nun während der Aufnahme bewegt, trifft das Bild nicht mittig auf die Bildebene und es ist verschwommen (s. Abbildung 3.23). Um diesen Effekt auszugleichen, wird während der Aufnahme die bewegliche Linse kontinuierlich so ausgerichtet, dass das Bild während der gesamten Belichtungszeit mittig auf die Bildebene projiziert wird (s. Abbildung 3.24).[Can]

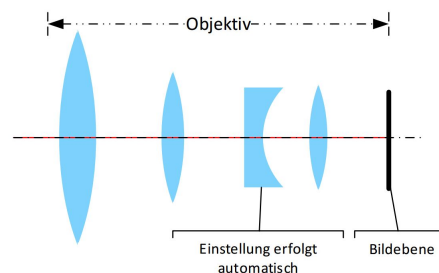


Abbildung 3.22: Skizze einer Bildaufnahme bei einem *unbewegtem* Objektiv mit Bildstabilisation

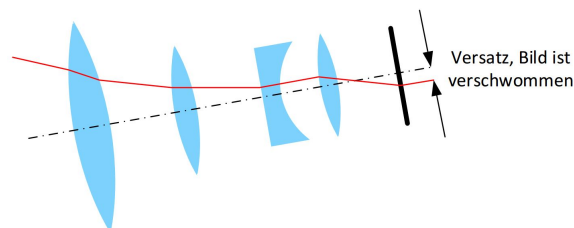


Abbildung 3.23: Skizze einer Bildaufnahme bei einem *bewegtem* Objektiv *ohne* Bildstabilisation

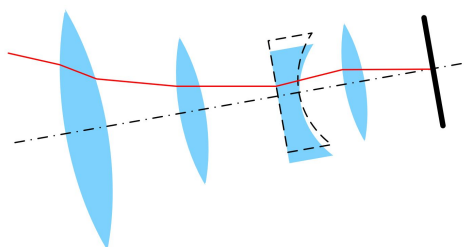


Abbildung 3.24: Skizze einer Bildaufnahme bei einem *bewegtem* Objektiv *mit* Bildstabilisation

Der Vorteil eines Objektivs mit Bildstabilisation ist, dass das Objektiv unabhängig von der Kamera eingesetzt und somit für beliebige Kameras verwendet werden kann. Des Weiteren ist das stabilisierte Bild (bei einer Spiegelreflexkamera) direkt durch den Kamerasucher zu sehen, was bei einer mechanischen Bildstabilisierung nicht der Fall ist (s. Kapitel 3.4.2).

3.4.2 Mechanische Bildstabilisierung

Neben der optischen Bildstabilisierung ist eine Bildstabilisierung direkt in der Kamera möglich (mechanische Bildstabilisierung). Dabei wird das gleiche Prinzip, wie unter Abschnitt Kapitel 3.4.1 beschrieben, angewandt. Der Unterschied besteht lediglich darin, dass anstatt einer beweglichen Linse direkt der Bildsensor bewegt wird und die Bildstabilisierung somit unabhängig von dem verwendeten Objektiv ist (s. Abbildung 3.25).



Abbildung 3.25: Funktionsweise der mechanischen Bildstabilisation in einer Kamera [Gro]

3.4.3 Elektronische Bildstabilisierung

Bei der elektronischen Bildstabilisierung wählt die Kamera automatisch eine erhöhte Lichtempfindlichkeit in Verbindung mit einer kurzen Verschlusszeit. Dadurch wirkt sich z. B. das Zittern einer menschlichen Hand, welche die Kamera hält, kürzer auf das Bild aus. Die dadurch entstehenden Bilder werden später während der Signalverarbeitung im Bildprozessor nachgearbeitet. Bei dieser Methode kann es zu erhöhtem Rauschen und Artefakten in den Bildern kommen, daher ist sie weniger effektiv als die o. g. Maßnahmen [Sch06].

Die drei vorgestellten Bildstabilisierungsmechanismen helfen lediglich die Kamera gegen Vibrationen zu schützen – häufig dahingehend optimiert, um die "Zitter"-Frequenz der menschlichen Hand herauszufiltern (bei der optischen und mechanischen Variante). Die Frequenz der Multikoptervibrationen unterscheidet sich von der "Zitter"-Frequenz, daher ist darauf zu achten, ob bei Verwendung der optischen oder mechanischen Bildstabilisierung auch der gewünschte Effekt der Bildstabilisierung eintritt. Damit die Kamera unabhängig von der Multikopterorientierung und dem Multikopterflug im Raum positioniert werden kann, ist eine weitere Vorrichtung nötig.

3.4.4 Kardanische Aufhängung

Abbildung 3.26 zeigt das Prinzip eine kardanische Aufhängung (auch Gimbal genannt). Ein Gimbal ist eine Vorrichtung, bei der ein Körper in mehreren (hier drei) zueinander rechtwinkligen Achsen drehbar gelagert ist. Der Körper befindet sich im inneren Ring und kann zum einen durch seine Schwerpunktlage oder durch eine elektronische Ansteuerung eine vorgegebene Stellung im Raum beibehalten, auch wenn die kardanische Aufhängung in der Lage beeinflusst wird.

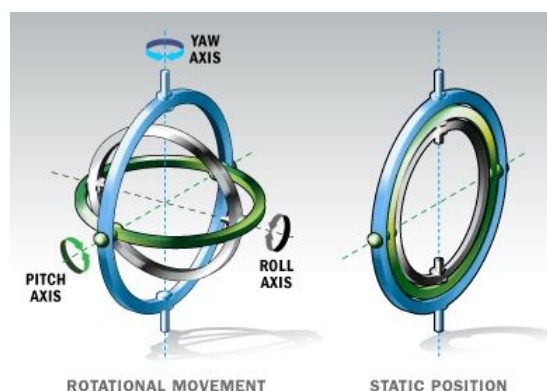


Abbildung 3.26: Funktionsweise eines Gimbal

Ein Problem, das bei einem Gimbalssystem auftritt, ist der sog. Gimbal-Lock. In diesem Fall ist die Pitch-Achse um $\pm 90^\circ$ gedreht und die Roll-Achse überschneidet sich mit der

Yaw-Achse (s. Abbildung 3.26, rechts). Somit fehlt ein Freiheitsgrad um das Objekt in eine gewünschte Lage zu bewegen.

Auf der linken Seite der Abbildung 3.26 ist der Gimbal so ausgelegt, dass bei Drehung der Yaw-Achse die Pitch- und Roll-Achse mit bewegt werden. Bei Drehung der Pitch-Achse bewegt sich nur die Roll-Achse mit. Eine Bewegung der Roll-Achse hat keine Auswirkung auf die beiden anderen Achsen. Die Achsenabhängigkeit sieht also wie folgt aus: Yaw- → Pitch- → Roll-Achse.

Als Beispiel zur Verdeutlichung des Problems wird als Objekt eine Kamera herangezogen, welche in Ausgangslage mit ihrem Objektiv in Richtung Roll-Achse ausgerichtet ist. Der Gimbal-Lock tritt in einem System mit der Achsenabhängigkeit Yaw- → Pitch- → Roll-Achse auf, wenn die Kamera in Richtung Yaw-Achse ausgerichtet ist (s. Abbildung 3.27).

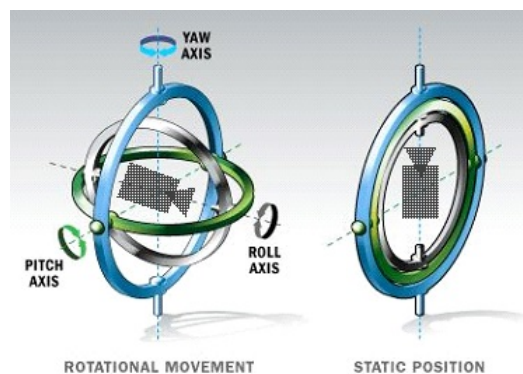


Abbildung 3.27: Funktionsweise eines Gimbal mit Kamera

Wird jedoch der gesamte Gimbal um 90° gedreht, sodass die Yaw-Achse nun der mittlere und die Pitch-Achse der äußere Ring ist, ändert sich die Achsenabhängigkeit zu: Pitch- → Yaw- → Roll-Achse. Somit würde ein Gimbal-Lock auftreten, wenn die Kamera in Richtung Pitch-Achse ausgerichtet ist.

Um einen Gimbal-Lock zu vermeiden, sollte die Achsenabhängigkeit des Gimbal so ausgewählt werden, dass dieser in der Anwendung möglichst selten auftritt. Ebenfalls ist es sinnvoll z. B. den Winkel der Pitch-Achse bei einer Achsenabhängigkeit Yaw- → Pitch- → Roll-Achse auf $\pm 85^\circ$ zu beschränken (wenn es die Applikation zulässt), damit der Gimbal-Lock von vornherein ausgeschlossen werden kann. Eine andere Möglichkeit, die das Problem lösen kann, ist es, einen vierten Ring hinzuzufügen, welcher das Gimbalssystem allerdings komplizierter in der Regelung macht [HMS07, S. 95 ff] [Str].

Der elektronisch Gimbal wird über eine Steuerungseinheit geregelt. Die Steuerungseinheit vergleicht die Ist-Lage der Objektaufgabe, welche über einen Gyrosensor ermittelt wird, mit der vorgegebenen Soll-Lage. Die Soll-Position kann typischerweise über Pulsweiten Modu-

lation (PWM) Signale der Steuerungseinheit für jede Achse vorgegeben werden. Tritt eine Differenz auf, werden die Motoren, welche die Achsen des Gimbals steuern, nachgeregelt. Eine Differenz tritt auf, wenn eine neue Soll-Position eingestellt wird, aber auch, wenn Störeinflüsse wie Vibrationen auf den Gimbal wirken.

Speziell für Kameraanwendungen in Zusammenhang mit Multikoptern sind zwei unterschiedliche elektronische Gimbalssysteme gängig – die 2-Achs und 3-Achs Gimbals (s. Abbildung 3.28). Die Varianten unterscheiden sich in der Anzahl der Freiheitsgrade (Achsen). Der 2-Achs-Gimbal hat zwei Achsen (Roll und Pitch) wohingegen beim 3-Achs-Gimbal die Yaw-Achse hinzukommt.



Abbildung 3.28: 2-Achs-Gimbal [pp2] und 3-Achs-Gimbal [rc2]

Als Motoren werden Servomotoren oder bürstenloser direkt Antriebsmotor (BLDC) eingesetzt. Ein Servomotor besteht aus mehreren Komponenten, welche zusammen eine Regelschleife bilden (s. Abbildung 3.29).

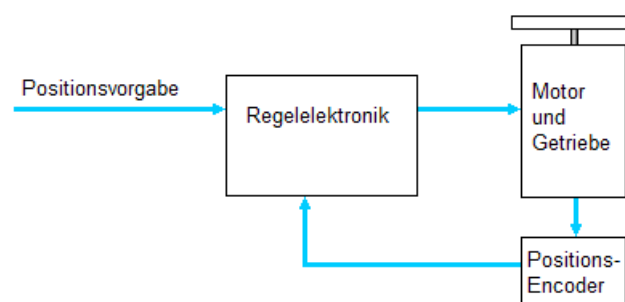


Abbildung 3.29: Skizze einer Servo-Regelschleife

Der Positionsenncoder ist häufig als mechanisches mit dem Getriebe gekoppeltes Potentiometer ausgeführt und als Motoren kommen BLDCs, synchrone oder asynchrone Gleichstrommotoren zum Einsatz. Unabhängig von der Motorart werden Servomotoren typischerweise mit einem 50 Hz Rechtecksignal angesteuert und das Servoprotokoll wird standardmäßig von vielen Flugcontrollern unterstützt. Dies bedeutet, dass der zusätzliche Be-

rechnungsaufwand reduziert wird.

Im Gegensatz dazu benötigen BLDCs eine spezielle Motoransteuerung, womit die Komplexität hard- und softwareseitig ansteigt. Bedingt durch die hochfrequente Ansteuerung der BLDCs (52 kHz¹¹ [STMa]) und dem typischerweise fehlenden Getriebe (BLDCs werden ohne Getriebe direkt an einer Gimbal-Achse befestigt), reagiert ein Gimbalssystem mit BLDCs schneller auf Störeinflüsse oder Solllageänderungen als ein Gimbalssystem mit Servomotoren [Unib] [Mika].

Des Weiteren ist es sinnvoll den Gimbal über schwingungsreduzierende Elemente am Multikopter zu befestigen. Dadurch werden von vornherein die Vibrationen des Multikopters und somit der Regelungsaufwand des Gimbals reduziert.

Für das unter Kapitel 1.4 genannte Szenario ist ein 3-Achs-Gimbalsystem mit BLDC Motoren zu bevorzugen. Ein solcher Gimbal gewährleistet (von den hier vorgestellten Varianten) den größtmöglichen Freiheitsgrad in der Kameraausrichtung und ein stabiles Kamerabild. Des Weiteren ist darauf zu achten, dass die Gimbalsteuereinheit eine externe Positionierung der Soll-Lage unterstützt, damit die Kamera kontinuierlich neu ausgerichtet und das Objekt verfolgt werden kann.

3.5 Trajektorienplanung und -beschreibung

In diesem Abschnitt werden die notwendigen Werkzeuge und Methoden beschrieben, um eine Trajektorienplanung der Kamerapositionierungseinheit durchzuführen.

3.5.1 Kinematische Ketten

Durch eine kinematische Kette wird die Stellung eines Endeffektors im Bezug auf ein gewähltes Bezugskordinatensystem angegeben. Dabei wird auf die Beschreibung der Orientierung und die Position des Endeffektors eingegangen. Die Orientierung beschreibt die rotatorische Ausrichtung zum Bezugskordinatensystem. Die Position beschreibt die translatorische Verschiebung zwischen Endeffektor und Bezugskordinatensystem. Der Endeffektor, kann beispielsweise ein Greifer eines Manipulators oder wie im Kontext dieser Projektgruppe der Photosensor der Kamera des Gimbals sein. Der Endeffektor ist oft über Gelenke mit der Basis verbunden, in der sich gewöhnlich auch das Bezugskordinatensystem befindet. Die Gelenke des Gimbals werden durch die Motoren der einzelnen Achsen repräsentiert. Diese Gelenke sind in der Lage, rotatorische Bewegungen zu vollführen und verändern dadurch die Orientierung des Bezugskordinatensystems. Wird jedes einzelne Gelenk mit einem eigenen Koordinatensystem versehen, so wird bei einer Drehung um eine Achse das Koordinatensystem der Achse i zu dem Koordinatensystem der vorange-

¹¹max. Frequenz eines typischen drei Phasen Motortreibers (L6234)



gangenen Achse $i-1$ gedreht. Die dazu notwendigen Rotationsmatrizen für die einzelnen Achsen sind in Kapitel 3.5.4 beschrieben.

Eine mögliche Beschreibungsform der Zusammenhänge zwischen den Bewegungen der Gelenke und der daraus resultierenden Endeffektorstellung bildet die Darstellung durch die Denavit-Hartenberg-Parameter.

3.5.2 Vorwärts-Kinematik

Die Vorwärts-Kinematik wird auch als direkte Kinematik bezeichnet, da mittels dieser Kinematik aus den Gelenkwinkeln auf direktem Wege die Pose des Endeffektors berechnet werden kann. Die Pose des Endeffektors beschreibt die Orientierung und Position relativ zum Bezugskoordinatensystem. Mathematisch wird die Transformation mittels der Vorwärts-Kinematik und einer seriellen Multiplikation der Denavit-Hartenberg-Matrizen beschrieben. Durch die serielle Multiplikation der einzelnen Matrizen für die Gelenkübergänge entstehen numerische Ungenauigkeiten, da die einzelnen Komponenten in jedem Schnitt mit den Rundungsfehlern aus dem vorherigen Schritt multipliziert werden. Dadurch entsteht bei der gesamten Berechnung ein absoluter Fehler, der sich nach dem Fehlerfortpflanzungsgesetz zusammensetzt.

3.5.3 Rückwärts-Kinematik

Die Rückwärts-Kinematik wird auch als inverse Kinematik bezeichnet und bildet das Gegenstück zur direkten Kinematik. Bei der inversen Kinematik wird versucht, aus der Pose eines Endeffektors die Gelenkstellungen zu berechnen. Diese Berechnung liefert nicht immer eindeutige Ergebnisse, da es mehrere Gelenkstellungen geben kann, die dieselbe Pose des Endeffektors beschreiben. Die Lösung dieses Problems kann auf zwei Arten erfolgen:

1. analytische Lösung
2. numerische Lösung

Eine analytische Lösung eines vorliegenden Problems ist nur in Spezialfällen möglich. Die Grade der Polynome, die bei einem analytischen Ansatz aufgestellt werden, folgen 2er-Potenzen. Der Grad eines solchen Polynoms bezieht sich auf die Komplexität des physikalischen Aufbaus eines kinematischen Systems. Eine geschlossene analytische Lösung ist lediglich für Polynome bis zum 4ten Grad bekannt [Hei14]. Numerische Lösungsverfahren bedienen sich häufig dem Gradientenabstiegsverfahren, welches jedoch einen Startwert und hinreichend viele Iterationsschritte benötigt, um eine Lösung zu approximieren.

3.5.4 Koordinatentransformation

Bei einer Koordinatentransformation werden die Koordinaten eines Punktes von einem Koordinatensystem ins andere übertragen. Dies ist formal gesehen der Übergang der Ur-



sprungs koordinaten (x_0, x_1, \dots, x_n) zu $(x'_0, x'_1, \dots, x'_n)$ im neuen Koordinatensystem.

Betrachten wir nun 2 kartesische Koordinatensysteme K_0 und K_1 , welche eine gemeinsame Z-Achse haben. Der Winkel zwischen diesen beiden Koordinatensystemen sei α , was einer Drehung um die z-Achse entspricht. Im 2 Dimensionalem Raum gibt es nur einen Rotationswinkel zwischen den Koordinatensystemen. Den analogen Fall im 3 Dimensionalen Raum werden wir uns später anschauen.

Sei nun p_0 ein Punkt im Koordinatensystem K_0 und p_1 der selbe Punkt im neuen Bezugskoodinatensystem, so ergibt sich in Vektorschreibweise folgender Zusammenhang:

$$p' = R \cdot p$$

mit

$$R = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Im 3 Dimensionalem Raum ist die Stellung eines Objektes eindeutig durch 6 Parameter charakterisiert.

$$P = (x \quad y \quad z \quad w_1 \quad w_2 \quad w_3) \quad (3.15)$$

Die Parameter x, y, z geben hierbei die Verschiebung in Richtung der entsprechenden Achsen an. Bei den Parametern w_1, w_2, w_3 gibt es 2 unterschiedliche Interpretationsweisen.

Euler-Winkel:

- w_1 Drehung um die Z-Achse des Bezugssystems
- w_2 Drehung um die veränderte X-Achse
- w_3 Drehung um die veränderte z-Achse

Yaw Pitch Roll

- w_1 Drehung um die z-Achse
- w_2 Drehung um die y-Achse
- w_3 Drehung um die z-Achse

Für die Drehung um die Koordinatenachsen ergeben sich folgende Matrizen:

$$R_z = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}$$

Bei der Yaw-Pitch-Roll Notation ergibt sich folgende Gesamtrrotationsmatrix:

$$R_{ges} = R_x(w_3) \cdot R_y(w_2) \cdot R_z(w_1)$$

Die Multiplikation erfolgt hier von links nach rechts. Im Gegensatz hierzu erfordert Euler-Notation eine Verknüpfung zum Zwischenkoordinatensystem. Es ergibt sich somit folgende Gesamtrrotationsmatrix:

$$R_{ges} = R_z(w_1) \cdot R_x(w_2) \cdot R_z(w_1)$$

Die Multiplikation erfolgt hier von links nach rechts.

3.5.5 Parametrisierung nach Denavit Hartenberg

Die Denavit-Hartenberg-Transformation ist ein mathematisches Verfahren, welches auf Grundlage von Matrizen die Überführung von Koordinatensystemen innerhalb kinematischer Ketten beschreibt. Sie kommt insbesondere bei der Vorwärtskinematik von Robotersystemen zum Einsatz.

Um die Übergänge zwischen den Koordinatensystemen zu parameterisieren werden die 4 Parameter $(\delta, \eta, l, \alpha)$ verwendet. Diese haben folgende Bedeutung:

- δ Drehung um die Z-Achse
- η Verschiebung entlang der Z-Achse
- l Verschiebung entlang der X-Achse
- α Drehung um die X-Achse

Auf Grundlage dieser Parameter ergibt sich folgende Gesamtrrotationsmatrix:



$$\begin{pmatrix} \cos(\delta) & -\sin(\delta) \cdot \cos(\alpha) & \sin(\delta) \cdot \sin(\alpha) & l \cdot \cos(\delta) \\ \sin(\delta) & \cos(\delta) \cdot \cos(\alpha) & -\cos(\delta) \cdot \sin(\alpha) & l \cdot \sin(\delta) \\ 0 & \sin(\alpha) & \cos(\alpha) & \eta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die einzelnen Matrizen, welche die Übergänge parametrisieren können anschließend multipliziert werden, um den Übergang des Gesamtsystem aufgrund der Stellwinkel der Gelenke darzustellen.

3.5.6 Quaternionen

Quaternionen sind Quadrupel, die aus vier Komponenten bestehen. Ein Quaternion q besteht aus den Komponenten $\{q_0, q_1, q_2, q_3\}$. Die Komponente q_0 bildet den reellen Anteil und die Komponenten $\{q_1, q_2, q_3\}$ zusammen den dreidimensionalen vektoriellen Anteil. Aufgrund des dreidimensionalen Vektorteils werden Quaternionen auch als hyperkomplexe Zahlen bezeichnet. Folgende Gleichung zeigt die Schreibweise eines Quaternion:

$$q = q_0 \cdot 1 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k \quad , \text{ mit } q_0, q_1, q_2, q_3 \in \mathfrak{R} \quad (3.16)$$

Die vierdimensionalen Quaternionen bilden einen Vektorraum über den realen Zahlen \mathfrak{R} mit den Basiseinheitsvektoren $\{1, i, j, k\}$. Des Weiteren gilt, wie aus den komplexen Zahlen bekannt:

$$i^2 = j^2 = k^2 = -1 = i \cdot j \cdot k \quad (3.17)$$

Quaternionen besitzen eine eigene Algebra, um Berechnungen zu beschreiben. Die Algebra der Quaternionen ist assoziativ, jedoch nicht kommutativ [JK13]. Diese Nicht-Kommutativität wird durch die folgenden Gleichungen veranschaulicht:

$$i \cdot j = +k, \quad j \cdot k = +i, \quad k \cdot i = +j \quad (3.18)$$

$$j \cdot i = -k, \quad k \cdot j = -i, \quad i \cdot k = -j \quad (3.19)$$

Bei der Rechnung mit Quaternionen werden diese typischerweise normiert. Die Normierung erfolgt, wie bereits durch die Vektorrechnung bekannt, durch die Division des Quaternion mit dessen Betrag. Die folgende Formel beschreibt die Bildung des Betrages $\|q\|$ des Quaternion q :

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \quad (3.20)$$

Ein normiertes Quaternion besitzt die Länge 1 und wird daher auch Einheitsquaternion genannt.



In Abbildung 3.30 ist die Rotation zwischen zwei Quaternionen visuell veranschaulicht dargestellt. Die geometrische Interpretation der Rotation eines Quaternion bildet eine Kugel, auf dessen Oberfläche sich die Werte bewegen. Bei normierten Quaternionen liegen der werte auf der Oberfläche der Einheitskugel.

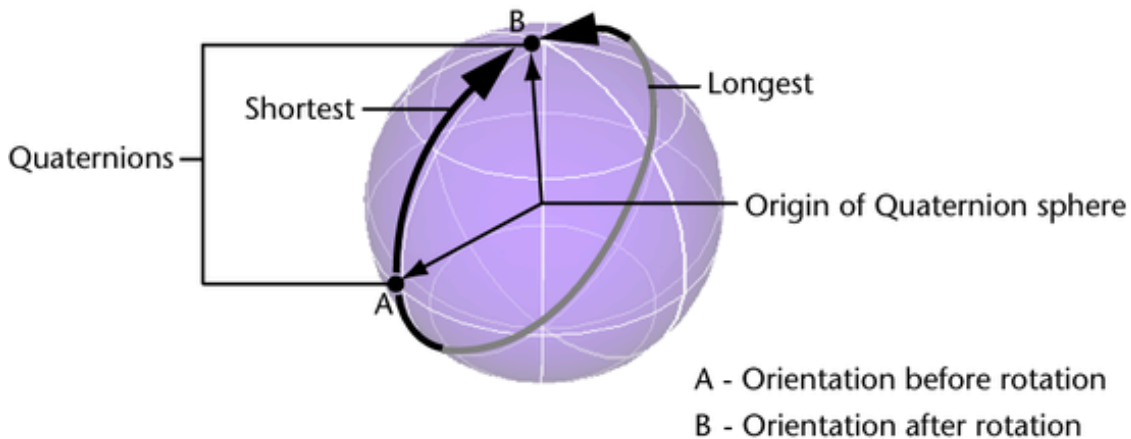


Abbildung 3.30: Darstellung der Rotation von Quaternionen, [Quaa]

In der Abbildung 3.30 ist ein Quaternion A dargestellt, welches durch Anwendung einer Rotation in das Quaternion B übergeht. Die Rotation um den Kugelmittelpunkt und die damit verbundene Verschiebung auf der Kugeloberfläche ist gut sichtbar. Wie auch bei Drehungen durch andere Verfahren, gibt es einen kurzen und einen langen Winkel um gedreht werden muss, damit sich die gewünschte Pose einstellt.

Die Addition von Quaternionen erfolgt komponentenweise, analog zu den komplexen Zahlen. Bei der Multiplikation von Quaternionen wird jede Komponente eines Quaternion mit allen Komponenten des anderen Quaternion multipliziert. Dabei ist Bezug auf die Gleichung 3.17 zu nehmen, um die Vorzeichen der konkatenierten Komponenten richtig zu setzen.

Mit den Quaternionen können im Gegensatz zu der Beschreibung durch die Stellungsmatrizen, lediglich rotatorische und keine translatorischen Bewegungen beschrieben werden. Der Vorteil beim Verwenden von Quaternionen besteht in der eindeutigen Beschreibung von Gelenkstellungen. Der Gimbal-Lock, wie in Kapitel 3.4.4 beschrieben, tritt bei den Quaternionen nicht mehr auf, da die Achsstellungen eindeutig beschrieben sind und dadurch keine Freiheitsgrade verloren gehen. Des Weiteren bieten Quaternionen im Vergleich zu den Stellungsmatrizen den Vorteil, dass numerische Ungenauigkeiten nicht mehr auftreten, da die Berechnungen parallel durchgeführt werden und nicht seriell, wie die Multiplikation mehrerer Stellungsmatrizen.



3.6 Kameratypen und -schnittstellen

Das zu verfolgende Objekt wird mit Hilfe einer Kamera aufgenommen, welche einen Live-Videostream über eine Audio/Video-Schnittstelle an eine Auswertelogik weiterleitet.

Die Nettodatenrate (C in $[\frac{bit}{s}]$) des Videostreams hängt von der Auflösung (P), der Bildwiederholungsrate (f_r in $[\frac{1}{s}]$) und der Bittiefe bzw. Farbtiefe (b in $[bit]$) der verwendeten Kamera ab und lässt sich wie folgt berechnen.

$$C = P \cdot f_r \cdot b \quad (3.21)$$

Diese Nettodatenrate kann mit Hilfe von Videokompressionsverfahren (z.B. mit dem Moving Picture Experts Group (MPEG) Verfahren) reduziert werden [Unia].

Aus diesem Grund ist ein Kriterium der Audio/Video-Schnittstellenauswahl die benötigte Bandbreite. Tabelle 3.2 gibt die maximale Bandbreiten für typische Kameraschnittstellen wieder [Wei08, S. 1018 ff].



Tabelle 3.2: Bandbreiten typischer Audio/Video-Schnittstellen

Schnittstelle	Bandbreite
HDMI ¹² (ab HDMI 1.3a)	8, 16 $\frac{G\text{Bit}}{s}$
Ethernet (100 Base-T)	100 $\frac{M\text{Bit}}{s}$
USB ¹³ (2.0)	480 $\frac{M\text{Bit}}{s}$
Firewire (IEEE 1394b)	800 $\frac{M\text{Bit}}{s}$

Um ein Livebild der Kamera zu erhalten, muss diese eine Audio/Video-Schnittstelle mit "Liveout"-Funktion anbieten. Das bedeutet, dass über diese Schnittstelle das aktuelle Bild angezeigt wird, welches auf den Bildsensor der Kamera fällt.

Bei Kameras wird unter anderem zwischen folgenden Typen unterschieden:

- Kompaktkameras
- Bridgekameras
- spiegellose Systemkameras
- Spiegelreflexkameras

Da neben einer Schnittstelle mit "Liveout"-Funktion ebenfalls Gewicht und Robustheit der Kamera (bei Verwendung in einem Multikoptersystem) im Vordergrund stehen, ist eine Kompaktkamera, genauer Outdoor-Kamera, zu bevorzugen.

Im Allgemein zeichnen sich Kompaktkameras durch ihre kleine Bauweise, ihr geringes Gewicht und ihre einfache Bedienung aus. Bei den speziellen Outdoor-Modellen kommt noch die Robustheit der Kamera gegen Umwelteinflüsse hinzu. Diese Modelle sind z. B. wasserfest und so konzipiert, dass die verwendete Hardware der Kamera gegen Erschütterung geschützt ist.

Eine solche Kamera ist beispielsweise die GoPro Hero 3+ Silver Edition der Firma GoPro Inc. Diese Kamera wiegt ca. 74 g bei einer Abmessung von 60 x 40 x 30 Millimeter und bietet folgende Audio/Video-Schnittstellen:

- HDMI
- Composite Video/Audio
- Wireless Local Area Network (WLAN) b/g/n

Der HDMI-Ausgang und die WLAN-Schnittstelle unterstützt dabei die "Liveout"-Funktion. Im Videomodus (sowohl National Television Systems Committee (NTSC) als auch Phase-Alternating-Line (PAL)) können folgende Auflösungen eingestellt werden:

¹²High-Definition Multimedia Interface (HDMI)

¹³Universal Serial Bus (USB)



- 1080p bis 60 fps
- 960p bis 60 fps
- 720p bis 120 fps
- Wide Video Graphics Array (WVGA) bis 120 fps

Um niedrigere Nettodatenraten zu erreichen, besteht die Möglichkeit, geringere Framerraten einzustellen. [GoP]

3.7 Drahtlose Datenübertragung

Da der Multikopter die Videodaten nach der Videoverarbeitung zum Nutzer streamen soll, muss dafür eine geeignete drahtlose Datenübertragung vorhanden sein. In diesem Kapitel werden einige Protokolle vorgestellt, die zur drahtlosen Übertragung von Videodaten geeignet sind. Ein Augenmerk wurde auf die Energieeffizienz und Übertragungsdatenrate gelegt, um den Akkumulator des Gerätes nicht zu sehr zu beanspruchen und die Flugzeit somit zu maximieren, sowie dem Nutzer ein ansehnliches Bild bereitstellen zu können. Des Weiteren ist die mögliche Reichweite der drahtlosen Übertragung ein wichtiger Punkt für die Anwendung im Zusammenhang mit dem Multikopter.

3.7.1 ZigBee

Das ZigBee Protokoll basiert auf dem IEEE 802.15.4 Standard, welches die Bitübertragungsschicht sowie die Sicherungsschicht des OSI Modells bereits definiert. Die übrigen Schichten werden vom ZigBee Standard definiert. Es kann im 868 MHz-, 915 MHz-, sowie im 2,4 GHz-Band arbeiten und hat je nach gewählter Frequenz eine Übertragungsrate von 40 Kbit/s bis zu 250 Kbit/s (bei 2,4 GHz). Im ZigBee Protokoll gibt es drei Gerätearten die wiederum in zwei Klassen unterteilt sind. Die beiden Klassen heißen Full Function Devices und die Reduces Function Devices. Zu den Full Function Devices zählen der Koordinator und der Router. Die Reduce Function Devices stellen die End-Knoten dar. In jedem ZigBee Netzwerk gibt es genau einen Koordinator. Dieser verwaltet das Netzwerk, indem er z. B. die drahtlose Verbindung startet und Beitrittsanträge innerhalb des Netzwerks an neue Knoten verteilt. Der Koordinator muss die ganze Zeit aktiv sein, d. h. er sollte mit einer Netzspannung betrieben werden. Ein Router kann in einem ZigBee Netzwerk vorhanden sein, muss es aber nicht. Er dient dazu, das Netzwerk zu vergrößern und die Daten der End-Knoten aufzunehmen. Dadurch muss der Router, ebenso wie der Koordinator die ganze Zeit, in der das Netzwerk aktiv ist, bereit sein.

Die End-Knoten sind die Geräte im Netzwerk, die energiesparend sind. Diese befinden sich die meiste Zeit in einem Schlaf-Modus, wo sie nur wenig Energie benötigen. Liegen neue Daten zum Versenden bereit, wachen die End-Knoten auf, schicken die Daten zu einem übergeordneten Gerät, etwa einem Router oder Koordinator und gehen danach wieder



in den Schlaf-Modus. Der Router und der Koordinator müssen die ganze Zeit aktiv sein, damit die End-Knoten die Daten zu ihnen schicken können. Dort werden sie zwischengespeichert und danach zum Ziel weitergeleitet. Somit können verschiedene Topologien, wie etwa eine Stern-, Maschen-, oder Baumtopologie aufgebaut werden.

Das ZigBee Netzwerk wird vor allem in der Industrie sowie in der Haus-Automatisierung eingesetzt, um Sensordaten zu erfassen und auszuwerten. Die Reichweite wird mit 60 m bis 300 m angegeben. Dieses ist das wohl bekannteste Protokoll, welches auf dem IEEE 802.15.4 Standard aufsetzt. Es gibt noch viele weitere Protokolle, die auf diesem Standard aufsetzen, wie z. B. WirelessHART. Da aber die Protokolle auf 802.15.4 aufsetzen sind sie in der Datenrate beschränkt auf maximal 250 Kbit/s. Voraussichtlich wird diese Datenrate nicht für ein angemessenes Bild ausreichen. Des Weiteren ist das Energiesparen durch Schlafen der End-Knoten für einen Videostream nicht möglich, da hier eine kontinuierliche Datenübertragung benötigt wird und nur die Knoten im Energiesparmodus gehen können, wenn keine Datenübertragung aktiv ist. Somit braucht es einen anderen Standard für das drahtlose Streamen von Videodaten. [Wie13]

3.7.2 Bluetooth 4.0

2010 wurde Bluetooth 4.0 oder auch Bluetooth Low Energy genannt, in den Bluetooth Standard integriert. Es nutzt fast die selben Technologien wie der alte Bluetooth Standard. Somit arbeitet es im 2,4 GHz Frequenzband, teilt diese in 40 je 2 MHz breite Kanäle (beim alten Bluetooth Standard waren es 79 je 1 MHz breite Kanäle) und besitzt Frequenz Hopping Spread Spectrum, welches es ermöglicht im laufenden Betrieb des Netzwerkes den Kanal zu wechseln, um Interferenzen zu vermeiden. Bluetooth bildet als Topologie ein sogenanntes Piconet. Dabei sind Punkt-zu-Punkt und Punkt-zu-Multipunkt Verbindungen möglich. Der Ersteller des Netzwerkes ist der Master, der die Adressen verwaltet und die vorhandenen Geräte synchronisiert. Alle anderen Netzwerkteilnehmer übernehmen die Rolle von Slaves. In einem Piconet können maximal acht Teilnehmer vorhanden sein. Teilnehmer eines Piconets können mit anderen Teilnehmern eines anderen Piconet-Netzwerkes kommunizieren. So können sich bis zu zehn Piconet-Netzwerke zu einem Scatternet zusammenschliessen. Die maximale Distanz zweier Netzwerkteilnehmer wird mit 50 m angegeben. Dabei wird eine effektive Datenrate bei Bluetooth 4.0 von 0,27 MBit/s erreicht. Beim normalen Bluetooth (ab Version 3.0) wird maximal eine Datenrate von 54 MBit/s erreicht.

Bluetooth hat ein breites Anwendungsspektrum und wird z. B. für Headsets, Tastaturen oder auch Mobiltelefone eingesetzt. Um dieses breite Spektrum abzudecken, gibt es im Bluetooth Standard sogenannte Profile, um sofort zu erkennen, welches Gerät vorliegt. Dabei kann ein Gerät mehrere Profile aufweisen.

Bluetooth 4.0 besitzt im Gegensatz zum alten Standard zwei verschiedene Chip-Typen, die in unterschiedlichen Modi arbeiten: dual-mode und single-mode. Beim dual-mode ist



der alte Bluetooth Standard sowie der neue Bluetooth Low Energy Standard integriert. Beim Single Mode ist nur der Bluetooth 4.0 Standard integriert. Die Besonderheit von Bluetooth Low Energy liegt darin, dass sich die Geräte die meiste Zeit, ähnlich wie beim ZigBee Standard, im Schlaf-Modus befinden um so Energie zu sparen. Sendet nun ein Bluetooth 4.0 Gerät, wird ein gleicher Energieverbrauch wie beim klassischen Bluetooth erreicht. [blu], [Wie13]

3.7.3 Wireless Local Area Network

Das Wireless Local Area Network oder auch W-Lan genannt, basiert auf der Gruppe der 802.11 Standards, die verschiedene Abwandlungen auf Basis von Ethernet haben. Es ist das am weitesten verbreitete drahtlose Netzwerk. Ein Standard ist der 802.11ac der nur im 5 GHz Frequenzbereich arbeitet und eine effektive Übertragungsrate von maximal 400 MBit/s aufweist. 802.11n ist ebenfalls noch weit verbreitet und arbeitet im 2,4 GHz und im 5 GHz Frequenzbereich. Dieser Standard liefert eine maximale effektive Datenübertragungsrate von 200 MBits/s. Dies ist zudem noch stark abhängig von der eingesetzten Antennentechnik. Die maximale Übertragungsreichweite liegt bei den Standards 802.11ac bei 50 m und bei 802.11n bei 100 m. Die möglichen Verbindungs-Topologien sind Punkt-zu-Punkt Verbindungen, die sogenannte Independent Basic Service Set. Hier verbinden sich zwei W-Lan Clients direkt miteinander. Die üblichste Topologie von W-Lan ist die Basic Service Set, wo ein Access Point das kabelgebundene Ethernet-Netzwerk drahtlos für W-Lan Clients bereitstellt. Das Extended Service Set verbindet zwei Ethernet Netzwerke drahtlos miteinander. Bei dieser Verbindung können sich keine weiteren drahtlosen Teilnehmer verbinden.

Die letzte mögliche Art ist das Wireless Distribution System. Hier nimmt ein sogenannter W-Lan Repeater das W-Lan Signal auf, bereitet dies neu auf und sendet das Signal weiter. Mit dieser Technik kann die Reichweite des W-Lan Netzwerkes vergrößert werden.

Der derzeit schnellste W-Lan Standard ist der 802.11ad oder auch Wireless Gigabit genannt, der nicht abwärtskompatibel zu den anderen Standards ist. Dieser bietet eine theoretisch maximale Übertragungsrate von 7 GBit/s, wobei jedoch einige Firmen schon versuchen die Geschwindigkeit auf 20 GBit/s zu erhöhen. Einziger Nachteil von Wireless Gigabit ist die geringe Reichweite von nur 10 m. Dies liegt daran, dass dieser Standard im 60 GHz Frequenzband arbeitet und hier die Dämpfung auf das Funksignal sehr stark ist (20 db/km). Deswegen wird 802.11ad nur für Punkt-zu-Punkt Verbindungen benutzt, und soll in Zukunft die Kabel für verschiedene Geräte wie z.B. Fernseher und Mediaplayer überflüssig machen. W-Lan wird somit vor allem dort eingesetzt, wo große Datenmengen übertragen werden müssen.



3.7.4 Fazit

Durch die Anforderung, dass ein Videostream von einem Multikopter zu einer Telemetriestation gesendet werden soll, wird die Auswahl eines geeigneten Protokolls stark eingeschränkt. Dies hat den Hintergrund, dass für das Streamen eines Videos, je nach Videogröße, eine möglichst große Datenübertragungsrate benötigt wird. Dem Nutzer des Systems soll ein anschauliches Bild, also eines mit einer möglichst hohen Auflösung, bereitgestellt werden. Darüber hinaus sollen weitere Informationen drahtlos an die Telemetriestation gesendet werden. Deshalb sollte ein Protokoll gewählt werden, das eine möglichst hohe Datenübertragungsrate, sowie eine möglichst große Reichweite bietet. Dies hat den Vorteil, dass ein Puffer vorhanden ist, falls sich im Laufe des Projektes Anforderungen ändern und größere Datenmengen übertragen werden sollen. Somit fällt ZigBee als mögliches Funkprotokoll raus, da dies nur eine Übertragungsrate von maximal 250 KBit/s bietet. Ein weiterer wichtiger Punkt bei der drahtlosen Übertragung sind Interferenzen mit der Fernbedienung des Multikopters und anderen Geräten. Da diese im 2,4 GHz Frequenzband arbeiten, muss eventuell auf einen anderen Frequenzbereich ausgewichen werden, um Störungen zu vermeiden. Es wäre deshalb von Vorteil, wenn mehrere Frequenzen vom Protokoll unterstützt würden. Durch dieses Problem fällt auch Bluetooth als Standard zur Datenübertragung heraus, da dies nur im 2,4 GHz Frequenzband arbeiten kann. Folglich bleibt nur noch W-Lan als möglicher Funkübertragungsstandard übrig. Hier kristallisiert sich vor allem der W-Lan 802.11n Standard heraus, da dieser im 2,4 GHz sowie im 5 GHz Frequenzband arbeiten kann, eine hohe Übertragungsrate (max 200 MBit/s) und eine große Reichweite (100 m) bietet. Da dieser Standard im Gegensatz zum W-Lan 802.11ad Standard weit verbreitet ist und seit längerem auf dem Markt ist als W-Lan 802.11ac, gibt es für diesen Standard günstige USB-Funkstick Lösungen, die leicht eingesetzt werden können und eine Möglichkeit darstellen, die Hardware des Multikopters mit einer W-Lan Schnittstelle auszurüsten. Somit ist der W-Lan 802.11n Standard, der Standard, der in diesem Projekt voraussichtlich für das Streamen eines Videos einsetzen wird. Als nächstes folgt eine Tabelle, die die vorgestellten Übertragungsprotokolle miteinander vergleicht. [wlaa],[wlab],[wlac],[wlad],[wlae]

Tabelle 3.3: Vergleich der vorgestellten Funkprotokolle

Protokoll	Frequenzband	Übertragungsrate	Reichweite
ZigBee	868 MHz 915 MHz 2,4 GHz	40 Kbit/s bis 250 Kbit/s	60 m bis 300 m
Bluetooth	2,4 GHz	(3.0) max. 54 MBit/s (4.0) eff. 0,27 MBit/s	50 m

Tabelle 3.3: Vergleich der vorgestellten Funkprotokolle (Fortsetzung)

Protokoll	Frequenzband	Übertragungsrate	Reichweite
W-Lan 802.11n	2,4 GHz 5 GHz	200 MBit/s	100 m
W-Lan 802.11ac	5 GHz	400 MBit/s	50 m
W-Lan 802.11ad	60 GHz	7 GBit/s	10 m

3.8 Rechtliche Grundlagen

In diesem Kapitel werden auf rechtliche Grundlagen eingegangen, welche bei der Verwendung eines Multikopters berücksichtigt werden müssen. Die rechtlichen Grundlagen werden durch das Luftverkehrsgesetz (LuftVG) und Luftverkehrsordnung (LuftVO) geregelt.

Grundsätzlich ist bei der Nutzung von unbemannten Fluggeräten zwischen den gewerblichen und privaten Gebrauch zu unterscheiden. In Tabelle 3.4 sind einige für dieses Projekt wichtige Kriterien aufgelistet. Weitere Informationen sind den o. g. Regelwerken zu entnehmen. [Lufa][Lufb]

Tabelle 3.4: Übersicht über rechtliche Bedingungen

Kriterium	Privat	Gewerblich
Gewicht Fluggerät	Bei einem Gewicht > 5 kg wird eine Aufstiegserlaubnis benötigt.	Aufstiegserlaubnis wird immer benötigt.
Pilot	Keine gesetzlichen Vorschriften, jeder darf einen Multikopter steuern.	
Flugumgebung	Nicht geflogen werden darf der Multikopter <ul style="list-style-type: none"> • innerhalb eines 1,5 km Radius von Flughäfen entfernt • in Flugverbotszonen • über Atomkraftwerke, Menschenansammlungen, Unfallstellen (Regelung nach Bundesland unterschiedlich) • über Grundstücke ohne Genehmigung 	Nicht geflogen werden darf der Multikopter <ul style="list-style-type: none"> • innerhalb eines 1,5 km Radius von Flughäfen entfernt • in Flugverbotszonen • über Atomkraftwerke, Menschenansammlungen, Unfallstellen (Regelung nach Bundesland unterschiedlich) • über Grundstücke ohne Genehmigung

Tabelle 3.4: Übersicht über rechtliche Bedingungen (Fortsetzung)

Kriterium	Privat	Gewerblich
max. Flughöhe und -weite	<ul style="list-style-type: none"> • Flugweite innerhalb der Sichtweite (ca. 200 bis 300 Meter) • Flughöhe je nach Bundesland (30 bis 100m) 	<ul style="list-style-type: none"> • Flugweite innerhalb der Sichtweite (ca. 200 bis 300 Meter) • Flughöhe je nach Bundesland (30 bis 100m)
Schadensersatz	Der Pilot haftet für alle Schäden.	
Einschränkungen bei Bildaufnahmen	<ul style="list-style-type: none"> • Bilder von Gebäuden dürfen im privaten Umfeld gezeigt werden (Urheberrecht des Architekten) • Bilder von Gebäuden dürfen im Internet veröffentlicht werden, sofern sie nicht die Rückseite oder Innenhof zeigen. • Bilder von militärisch relevanten Bereichen (§ 109 StGB) 	<p>Professionelle Aufnahmen benötigen die Erlaubnis des Grundstückbesitzers. Auch wenn der Zugang zu privaten Zwecken gestattet ist.</p>



4 Systemdefinition

In diesem Kapitel wird zunächst eine Aufteilung des Gesamtsystems in die Teilsysteme der sicherheitskritischen und nicht sicherheitskritischen Anwendung vorgenommen. Danach werden die relevanten Rahmenbedingungen erläutert, die sich u. a. aus der Aufgabenstellung (s. Kapitel 1.2) und aus dem Szenario (s. Kapitel 1.4) ergeben. Den Abschluss des Kapitels bilden die aufgestellten Anwendungsfälle, die die Funktionalität des Gesamtsystems definieren. Diese stammen ebenfalls aus dem Szenario und dienen als Grundlage für die Spezifikation der Anforderungen.

4.1 Verwendete Komponenten innerhalb des Systems

Das Gesamtsystem lässt sich gemäß der Aufgabenstellung (s. Kapitel 1.2) in eine sicherheitskritische Avionik und eine nicht sicherheitskritische Anwendung unterteilen. Für die nicht sicherheitskritische Anwendung wurde im Kapitel 1.3.5 die Verfolgung von Objekten gewählt. Im folgenden Kapitel erfolgt eine Abgrenzung dieser beiden Teilsysteme. Um zwischen diesen innerhalb des Projekts zu differenzieren, werden eindeutige Bezeichnungen für die zwei Teilsysteme eingeführt.

4.1.1 ASG

Die sicherheitskritische Avionik besteht aus diversen Hard- und Softwarekomponenten, die für die Regelung des Flugverhaltens des Multikopters Voraussetzung sind. Als einheitliche Bezeichnung für diese Komponenten wird der Begriff Avionik Steuergerät (ASG) eingeführt.

Neben den Sensoren und Motoren des Multikopters umfasst das ASG eine Verarbeitungseinheit zur Ausführung der Regelungsalgorithmen sowie das Empfangsmodul für die Verbindung mit der Fernsteuerung. Die Hauptfunktion des ASG ist es, die Steuerbefehle des Piloten in die entsprechenden Bewegungen des Multikopters zu transformieren und parallel dazu ein stabiles Flugverhalten sicherzustellen. Hinzu kommen das Versenden von Telemetriedaten an die Fernsteuerung und die Bereitstellung dieser Daten für die nicht sicherheitskritische Anwendung.

4.1.2 Das AEVV-System

Um zusammenhängende Funktionalitäten sowie deren zugehörige Hardwarekomponenten der nicht sicherheitskritischen Anwendung zu einer Einheit zusammenfassen zu können, wird die Bezeichnung Aufnahme-, Erkennungs-, Verfolgungs- und Versendungssystem (AEVV-System) eingeführt.

Das AEVV-System umfasst alle Funktionalitäten und deren Hardwarekomponenten der nicht kritischen Task, die am oder im Multikopter aufzufinden sind. Konkret handelt es

4.2 Rahmenbedingungen für das Szenario

sich dabei um die Bildaufnahme, die Erkennung eines Objekts auf dem Kamerabild, die Verfolgung des detektierten Objekts sowie das Versenden des Kamerabildes und weiterer Daten an die Telemetriestation. Das Empfangen der Daten auf der Telemetriestation gehört nicht mehr zum Funktionsumfang des AEVV-Systems.

Die erste Funktionalität ist die Aufnahme eines Kamerabildes. Ziel ist es, ein digitales Kamerabild zur Verfügung zu stellen.

Die zweite Funktionalität bildet die Erkennung eines Objektes auf dem bereitgestellten Bildmaterial. Dazu wird das Kamerabild nach einem definierten Objekt untersucht, welches sich dafür in Form und Farbe stark von seiner Umgebung abheben muss.

Die Verfolgung des erkannten Objektes repräsentiert die dritte Funktionalität. Aufgabe dieser Funktion ist es, zu gewährleisten, dass das erkannte Objekt im Zentrum des Kamerabildes bleibt. Dieses Justieren soll durchgeführt werden, sobald sich das Objekt im Randbereich des Bildes befindet.

Die vierte Funktionalität besteht aus dem Versenden des Kamerabildes und weiterer Daten an die Telemetriestation.

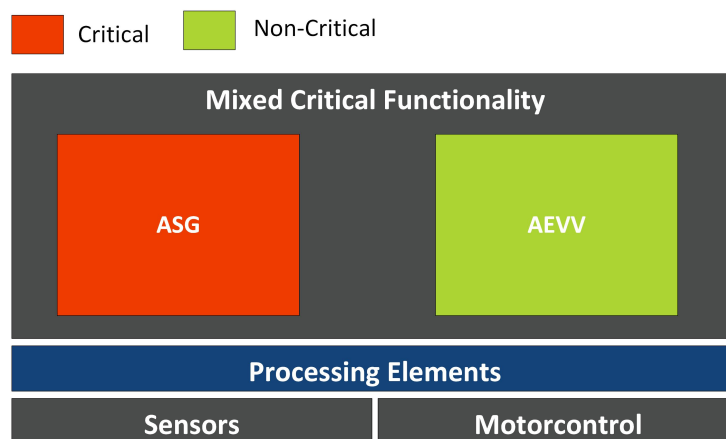


Abbildung 4.1: Systemarchitektur mit AEVV-System

Durch die Einführung dieser Komponenten lässt sich die eingangs definierte Systemarchitektur verfeinern (s. Abbildung 4.1). Die nicht sicherheitskritische Anwendung wird durch das AEVV-System und die Avionik wird durch das ASG spezifiziert.

4.2 Rahmenbedingungen für das Szenario

In diesem Kapitel werden die Rahmenbedingungen erläutert, welche die allgemeinen Vorbedingungen darstellen. Dadurch soll ein sicheres Flugverhalten des Multikopter gewährleistet werden. Diese Bedingungen lassen sich u.a. aus dem Szenario ableiten.

Wie in Kapitel 1.4 beschrieben, ist es eine Möglichkeit das Gesamtsystem als Torlinienrichter beim Roboterfußball einzusetzen. Dementsprechend wird der Multikopter in einem

4.3 Anwendungsfälle

geschlossenem Raum eingesetzt. Um ungewollte Luftströme durch die Rotoren zu vermeiden, muss sich der Multikopter auf einer ausreichenden Höhe befinden. Dadurch wird eine Beeinflussung des Balls durch den Wind vermieden.

Die Dimension eines Spielfeldes wird in den Regelwerken der einzelnen Ausrichter genauer beschrieben [Com14].

Während der Durchführung in einer Sporthalle muss für eine ausreichende Belichtung gesorgt werden, damit die ausgewählte Kamera ein verwendbares Farbbild für die digitale Bildverarbeitung liefert. Des Weiteren hebt sich seine Farbe stark von der Umgebung ab. Die Geschwindigkeit, mit der der Ball gespielt wird, darf eine Maximalgeschwindigkeit von 2.0 m s^{-1} nicht überschreiten, sodass das AEVV-System diesen immer erkennen kann.

Für eine erfolgreiche Objekterfassung und Objektverfolgung muss sich der Multikopter in seiner Arbeitshöhe¹⁴ befinden.

4.3 Anwendungsfälle

Aus dem Szenario in Kapitel 1.4 lassen sich verschiedene Anwendungsfälle ableiten, die im folgenden Abschnitt betrachtet werden sollen. Ein Überblick über die Funktionen des gesamten Systems ist in Abbildung 4.2 zu finden.

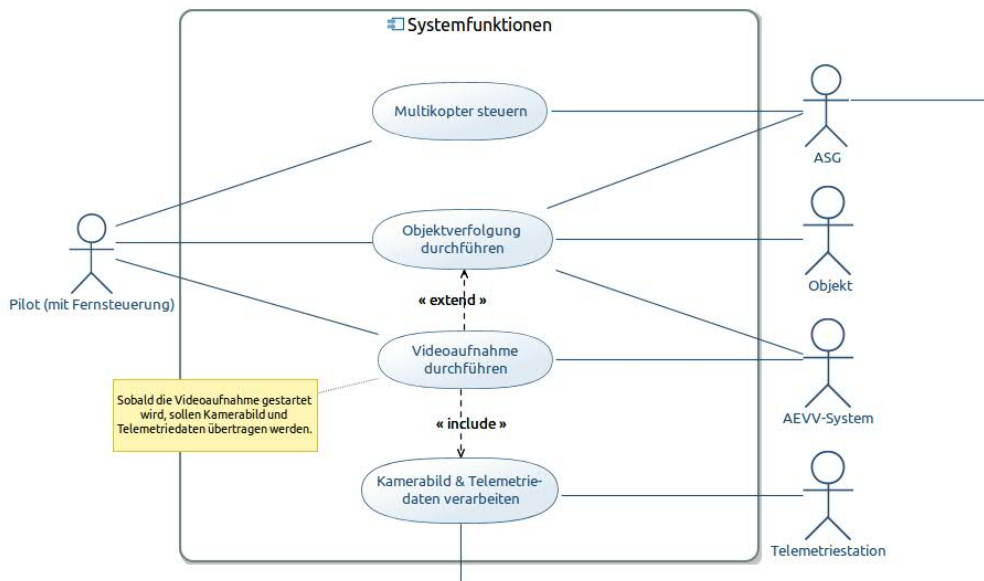


Abbildung 4.2: Use-Case: Systemfunktionen

Der Pilot hat die Möglichkeiten den Multikopter zu steuern, die Videoaufnahme durchzuführen und die Objektverfolgung auszuführen. Damit der Anwendungsfall *Objektverfolgung durchführen* korrekt ablaufen kann, muss der Anwendungsfall *Videoaufnahme durchführen*

¹⁴Die Arbeitshöhe ist eine Mindestflughöhe und richtet sich nach dem Szenario, sodass dieses erfolgreich durchgeführt werden kann.

ebenso ausgeführt werden. Aus dem im Kapitel 1.4 aufgestellten Szenario geht hervor, dass eine Liveübertragung des Kamerabildes zur Telemetriestation erfolgen soll. Dieser Sachverhalt wird durch den Anwendungsfall *Kamerabild & Telemetriedaten verarbeiten* dargestellt.

Da die Systemfunktionen in diesem Diagramm noch sehr abstrakt sind und z.T. weitere Anwendungsfälle enthalten, wird im weiteren Verlauf jede Systemfunktion in einem separaten Use-Case-Diagramm betrachtet.

4.3.1 Multikopter steuern

In Abbildung 4.3 ist der Anwendungsfall *Multikopter steuern* verfeinert. Der Akteur *Multikopter* aus Abbildung 4.2 wird hierfür in die Akteure *Sensoren* und *Motoren* aufgeteilt. Es wird deutlich, dass der Pilot in der Lage ist, die Flughöhe des Multikopters zu erhöhen bzw. zu verringern und dessen Position zu verändern. Dazu kann er den Multikopter sowohl vorwärts und rückwärts als auch seitwärts bewegen. Des Weiteren hat er die Möglichkeit, den Multikopter um die eigene Achse zu rotieren. Alle Anwendungsfälle erfordern, dass die Stellwerte für die Motoren vom System berechnet werden. Die Grundlage dafür bilden die Informationen der Sensoren, die im Anwendungsfall *Sensorwerte aufbereiten* verarbeitet werden.

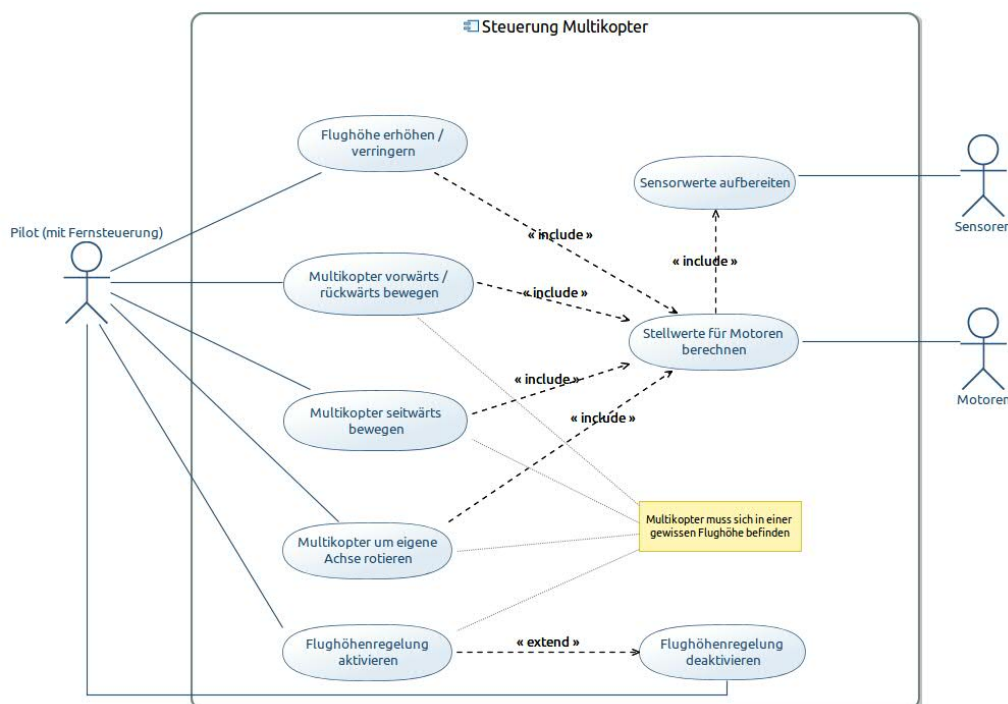


Abbildung 4.3: Use-Case: Multikopter steuern

In den nachfolgenden Tabellen wird jeder einzelne Use-Case aus der Abbildung 4.3 im Detail definiert.

Tabelle 4.1: Use-Case: Flughöhe erhöhen/verringern

UC001:	Flughöhe erhöhen/verringern
Hauptakteur, Akteur:	<u>Pilot</u> , Sensoren und Motoren
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich im betriebsbereiten Zustand. • Verbindung zwischen Multikopter und Fernsteuerung ist aufgebaut.
Nachbedingungen:	<ul style="list-style-type: none"> • Multikopter hat die vorgegebene Höhe eingenommen. • Dabei werden externe Einflüsse berücksichtigt, sodass sich der Multikopter in einer horizontale Lage (Nullstellung der Roll- & Nick-Achse) hält.
Fachlicher Auslöser:	Pilot benutzt zur Höhenveränderung die Joysticks auf der Fernsteuerung.
Ablauf:	<ul style="list-style-type: none"> • Pilot sendet die Höhenveränderung über die Fernsteuerung an den Multikopter. • Multikopter interpretiert die Signale und vergleicht die Soll- mit der Ist-Stellung. • Multikopter regelt die Motoren, sodass die vorgegebene Höhe eingenommen wird.

Tabelle 4.2: Use-Case: Multikopter vorwärts/rückwärts bewegen

UC002:	Multikopter vorwärts/rückwärts bewegen
Hauptakteur, Akteur:	<u>Pilot</u> , Sensoren und Motoren
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich im betriebsbereiten Zustand. • Verbindung zwischen Multikopter und Fernsteuerung ist aufgebaut.
Nachbedingungen:	<ul style="list-style-type: none"> • Multikopter hat die vorgegebene Sollstellung durch eine Vorwärts- bzw. Rückwärtsbewegung eingenommen und regelt diese weiterhin. • Dabei werden externe Einflüsse berücksichtigt, sodass sich der Multikopter in einer horizontale Lage (Nullstellung der Roll- & Nick-Achse) hält.
Fachlicher Auslöser:	Pilot benutzt zur Lageänderung die Joysticks auf der Fernsteuerung.



Tabelle 4.2: Use-Case: Multikopter vorwärts/rückwärts bewegen (Fortsetzung)

UC002	Multikopter vorwärts/rückwärts bewegen
Ablauf:	<ul style="list-style-type: none"> • Pilot sendet die Lageänderung über die Fernsteuerung an den Multikopter. • Multikopter interpretiert die Signale und vergleicht die Soll- mit der Ist-Stellung. • Multikopter regelt die Motoren, sodass die Soll-Stellung durch eine Vorwärts-/Rückwärtsbewegung eingenommen wird.

Tabelle 4.3: Use-Case: Multikopter seitwärts bewegen

UC003:	Multikopter seitwärts bewegen
Hauptakteur, Akteur:	<u>Pilot</u> , Sensoren und Motoren
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich im betriebsbereiten Zustand. • Verbindung zwischen Multikopter und Fernsteuerung ist aufgebaut.
Nachbedingungen:	<ul style="list-style-type: none"> • Multikopter hat die vorgegebene Soll-Stellung durch eine Seitwärtsbewegung eingenommen. • Dabei werden externe Einflüsse berücksichtigt, sodass sich der Multikopter in einer horizontale Lage (Nullstellung der Roll- & Nick-Achse) hält.
Fachlicher Auslöser:	Pilot benutzt zur Lageänderung die Joysticks auf der Fernsteuerung.
Ablauf:	<ul style="list-style-type: none"> • Pilot sendet die Lageänderung über die Fernsteuerung an den Multikopter. • Multikopter interpretiert die Signale und vergleicht die Soll- mit der Ist-Stellung. • Multikopter regelt die Motoren, sodass die Soll-Stellung durch eine Seitwärtsbewegung eingenommen wird.

Tabelle 4.4: Use-Case: Multikopter um eigene Achse rotieren

UC004:	Multikopter um eigene Achse rotieren
Hauptakteur, Akteur:	<u>Pilot</u> , Sensoren und Motoren
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich im betriebsbereiten Zustand. • Verbindung zwischen Multikopter und Fernsteuerung ist aufgebaut.



Tabelle 4.4: Use-Case: Multikopter um eigene Achse rotieren (Fortsetzung)

UC004	Multikopter um eigene Achse rotieren
Nachbedingungen:	<ul style="list-style-type: none"> • Multikopter hat sich mit dem vorgegebenen Winkel um die eigene Achse gedreht. Dabei nimmt er permanent eine horizontale Lage (Nullstellung der Roll- & Nick-Achse) ein.
Fachlicher Auslöser:	Pilot benutzt zur Rotation um die eigene Achse des Multikopters den Joystick der Fernsteuerung.
Ablauf:	<ul style="list-style-type: none"> • Pilot sendet die Werte zur Rotation über die Fernsteuerung an den Multikopter. • Multikopter interpretiert die Signale und vergleicht die Soll- mit der Ist-Stellung. • Multikopter regelt die Motoren, sodass die Soll-Stellung durch eine Rotation eingenommen wird.

Tabelle 4.5: Use-Case: Sensorwerte aufbereiten

UC005:	Sensorwerte aufbereiten
Hauptakteur, Akteur:	Sensoren
Vorbedingungen:	<ul style="list-style-type: none"> • Die Sensorik befindet sich in einem betriebsbereiten Zustand.
Nachbedingungen:	<ul style="list-style-type: none"> • Die aktuellen Sensorwerte liegen dem System in einer geeigneten Formatierung zur Weiterverarbeitung vor.
Fachlicher Auslöser:	Das System möchte zur Regelung, welche entweder automatisch oder durch den Piloten eingeleitet wird, die aktuellen Sensorwerte auslesen.
Ablauf:	<ul style="list-style-type: none"> • Das System fordert die benötigten Sensorwerte an. • Das System bereitet die Sensorwerte in einem geeigneten Format zur internen Weiterverarbeitung auf.

Tabelle 4.6: Use-Case: Stellwerte für Motoren berechnen & aufbereiten

UC006:	Stellwerte für Motoren berechnen & aufbereiten
Hauptakteur, Akteur:	Motoren
Vorbedingungen:	<ul style="list-style-type: none"> • Die Motoren und die Sensorik befinden sich in einem betriebsbereiten Zustand.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Motoren haben die vorher vom System berechneten Stellwerte empfangen und geregelt.



Tabelle 4.6: Use-Case: Stellwerte für Motoren berechnen & aufbereiten(Fortsetzung)

UC006	Stellwerte für Motoren berechnen & aufbereiten
Fachlicher Auslöser:	Das System möchte zur Regelung, welche entweder automatisch oder durch den Piloten eingeleitet wird, den Motoren die aktuellen Stellwerte übermitteln.
Ablauf:	<ul style="list-style-type: none"> • Das System fordert die benötigten Sensorwerte an (vgl. UC005). • Das System ermittelt mit Hilfe der Sensorwerte die aktuellen Stellwerte Weiterverarbeitung auf. • Das System übermittelt den Motoren die jeweiligen Stellwerte.

Tabelle 4.7: Use-Case: Flughöhenregelung aktivieren

UC007:	Flughöhenregelung aktivieren
Hauptakteur, Akteur:	Pilot
Vorbedingungen:	<ul style="list-style-type: none"> • Eine Verbindung zwischen Funksteuerung und Multikopter existiert bereits. • Der Multikopter befindet sich auf seiner Arbeitshöhe. • Die Regelung der Flughöhe ist deaktiviert. • Mit dem Gasknüppel wird die Schubkraft für die Motoren vorgegeben.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Regelung der Flughöhe ist aktiviert. • Mit dem Gasknüppel wird die derzeitige Flughöhe des Multikopters verringert oder erhöht.
Fachlicher Auslöser:	Der Pilot möchte, dass der Multikopter seine derzeitige Flughöhe hält.
Ablauf:	<ul style="list-style-type: none"> • Der Pilot aktiviert über den Schalter auf der Funksteuerung die Flughöhenregelung. • Die aktuelle Flughöhe des Multikopters wird ermittelt. • Diese Flughöhe wird, solange keine Änderungswünsche durch den Piloten eintreffen, dauerhaft geregelt.



Tabelle 4.8: Use-Case: Flughöhenregelung deaktivieren

UC008:	Flughöhenregelung deaktivieren
Hauptakteur, Akteur:	Pilot
Vorbedingungen:	<ul style="list-style-type: none"> • Eine Verbindung zwischen Funksteuerung und Multikopter existiert bereits. • Der Multikopter befindet sich auf seiner Arbeitshöhe. • Die Regelung der Flughöhe ist aktiviert. • Mit dem Gasknüppel wird die derzeitige Flughöhe des Multikopters verringert oder erhöht.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Regelung der Flughöhe ist deaktiviert. • Mit dem Gasknüppel wird die Schubkraft für die Motoren vorgegeben.
Fachlicher Auslöser:	Der Pilot möchte, dass der Multikopter die Flughöhenregelung beendet.
Ablauf:	<ul style="list-style-type: none"> • Der Pilot deaktiviert über den Schalter auf der Funksteuerung die Flughöhenregelung.

4.3.2 Videoaufnahme durchführen

In Abbildung 4.4 wird die Systemfunktion *Videoaufnahme durchführen* ausführlicher dargestellt. Sobald der Pilot die Videoaufnahme aktiviert hat, wird von der Kamera fortlaufend ein Kamerabild bereitgestellt. Dies erfolgt so lange, bis der Pilot die Videoaufnahme wieder deaktiviert.

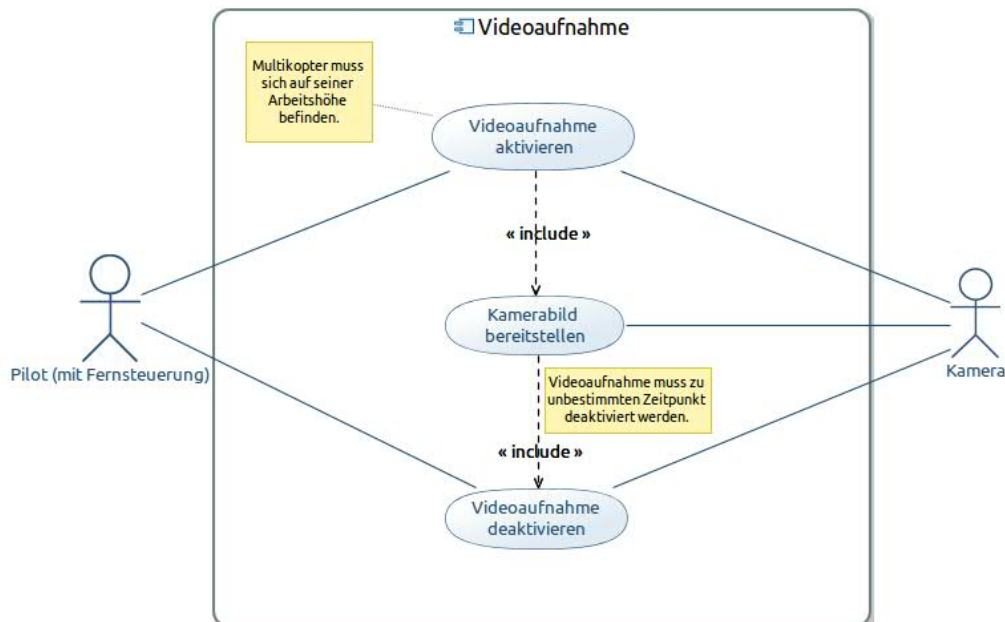


Abbildung 4.4: Use-Case: Videoaufnahme durchführen

In den nachfolgenden Tabellen wird jeder einzelne Use-Case aus der Abbildung 4.4 detaillierter betrachtet.

Tabelle 4.9: Use-Case: Videoaufnahme aktivieren

UC009:	Videoaufnahme aktivieren
Hauptakteur, Akteur:	<u>Pilot</u> , Kamera
Vorbedingungen:	<ul style="list-style-type: none"> • Die Kamera und ihre Positioniereinheit befindet sich in einem betriebsbereiten Zustand. • Die Videoaufnahme und die Kamerapositioniereinheit sind deaktiviert und somit stromlos.
Nachbedingungen:	<ul style="list-style-type: none"> • Kamera wurde eingeschaltet und ist bereit für die Aufnahme des Kamerabildes. • Kamera wurde durch die Positioniereinheit in Nullposition gefahren (s. Abbildung 1.7).
Fachlicher Auslöser:	Pilot möchte ein Videobild anzeigen lassen.
Ablauf:	<ul style="list-style-type: none"> • Pilot betätigt den Schalter auf der Fernsteuerung zur Aktivierung der Videoaufnahme. • Kamera wird eingeschaltet.

Tabelle 4.10: Use-Case: Kamerabild bereitstellen

UC010:	Kamerabild bereitstellen
Hauptakteur, Akteur:	<u>Pilot</u> , Kamera
Vorbedingungen:	<ul style="list-style-type: none"> • Die Kamera befindet sich in einem betriebsbereiten Zustand. • Kamera wurde bereits eingeschaltet.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Kamera liefert kontinuierlich Videodaten an ein Processing Element zur internen Weiterverarbeitung. • Videodaten werden vom Processing Element an die Telemetriestation gesendet.
Fachlicher Auslöser:	Pilot möchte ein Videobild anzeigen lassen.
Ablauf:	<ul style="list-style-type: none"> • Videoaufnahme der Kamera wird gestartet. • Kamera zeichnet kontinuierlich ihr Videobild auf und stellt dem Processing Element die Daten zur internen Weiterverarbeitung bereit.



Tabelle 4.11: Use-Case: Videoaufnahme deaktivieren

UC011:	Videoaufnahme deaktivieren
Hauptakteur, Akteur:	Pilot, Kamera
Vorbedingungen:	<ul style="list-style-type: none"> • Die Kamera und ihre Positioniereinheit befinden sich in einem betriebsbereiten Zustand. • Kamera wurde bereits eingeschaltet.
Nachbedingungen:	<ul style="list-style-type: none"> • Videoaufnahme der Kamera wurde beendet. • Kamera und Kamerapositioniereinheit wurden ausgeschaltet und sind somit stromlos.
Fachlicher Auslöser:	Pilot möchte die Anzeige des Videobildes beenden.
Ablauf:	<ul style="list-style-type: none"> • Videoaufnahme der Kamera wird beendet. • Kamera wird ausgeschaltet.

4.3.3 Objektverfolgung durchführen

Einen sehr ähnlichen Aufbau besitzt die Systemfunktion *Objektverfolgung durchführen*, die in Abbildung 4.5 dargestellt ist. Der Akteur *AEVV-System* aus Abbildung 4.2 wird hier durch die Akteure *Kamera* und *Positioniereinheit* repräsentiert. Der Pilot hat wiederum die Möglichkeit, die Objektverfolgung zu aktivieren, sofern die Videoaufnahme bereits aktiviert wurde. Infolgedessen versucht das System mit Hilfe der Kamera ein definiertes Objekt im Kamerabild zu erkennen. Bewegt sich das zu erkennende Objekt, führt die Positioniereinheit die Kameraposition automatisch nach, sodass das Objekt weiterhin im Fokus bleibt. Genau wie bei der Videoaufnahme hat der Pilot auch bei der Objektverfolgung jederzeit die Möglichkeit, diese zu deaktivieren.

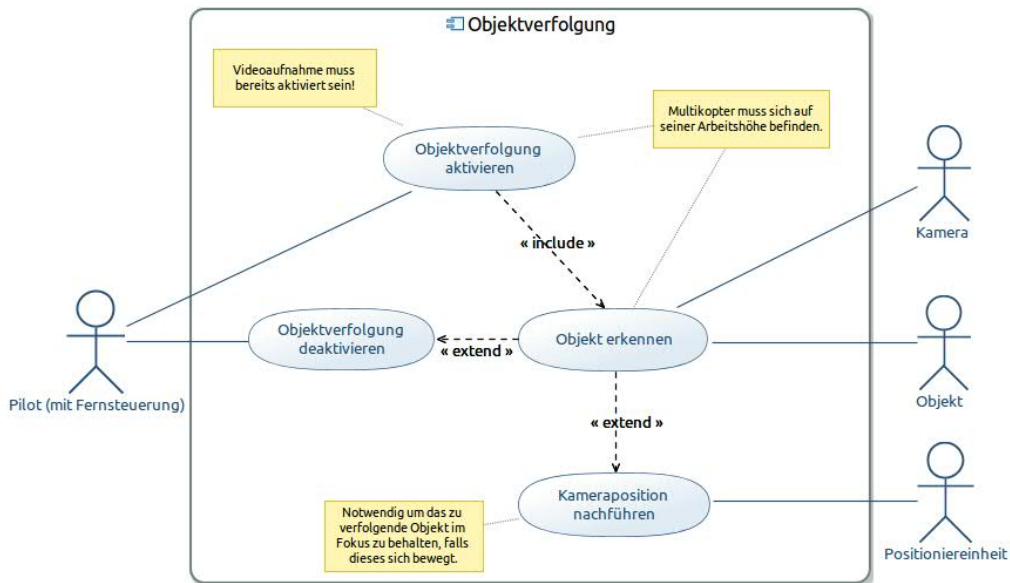


Abbildung 4.5: Use-Case: Objektverfolgung durchführen

In den nachfolgenden Tabellen wird jeder einzelne Use-Case aus der Abbildung 4.5 detailliert definiert.

Tabelle 4.12: Use-Case: Objektverfolgung aktivieren

UC012:	Objektverfolgung aktivieren
Hauptakteur, Akteur:	Pilot
Vorbedingungen:	<ul style="list-style-type: none"> • Pilot hat die Videoaufnahme bereits aktiviert (vgl. UC009). • Es wird permanent ein Videobild zur Weiterverarbeitung zur Verfügung gestellt. • Kamerapositioniereinheit befindet sich in Nullstellung (s. Abbildung 1.7).
Nachbedingungen:	<ul style="list-style-type: none"> • Objektverfolgung wurde aktiviert und versucht, das definierte Objekt zu detektieren.
Fachlicher Auslöser:	Pilot möchte die Videoverfolgung eines definierten Objekts starten.
Ablauf:	<ul style="list-style-type: none"> • Pilot betätigt den Schalter für die Objektverfolgung auf der Fernsteuerung. • Das AEVV-System zur Objekterkennung wird aktiviert.



Tabelle 4.13: Use-Case: Objekt erkennen

UC013:	Objekt erkennen
Hauptakteur, Akteur:	Kamera, Objekt
Vorbedingungen:	<ul style="list-style-type: none"> • Objektverfolgung wurde erfolgreich aktiviert (vgl. UC012). • Es wird permanent ein Videobild zur Weiterverarbeitung zur Verfügung gestellt. • Das definierte Objekt befindet sich im Sichtfeld der Kamera.
Nachbedingungen:	<ul style="list-style-type: none"> • Definiertes Objekt wurde im Bild erkannt. • Definiertes Objekt befindet sich im Zentrum des Bildes.
Fachlicher Auslöser:	Pilot möchte die Videoverfolgung eines definierten Objekts starten.
Ablauf:	<ul style="list-style-type: none"> • Das AEVV-System greift auf die bereitgestellte Videodaten zu. • Das AEVV-System versucht das definierte Objekt zu erkennen.

Tabelle 4.14: Use-Case: Kameraposition nachführen

UC014:	Kameraposition nachführen
Hauptakteur, Akteur:	Kamerapositioniereinheit, Objekt
Vorbedingungen:	<ul style="list-style-type: none"> • Objektverfolgung wurde erfolgreich aktiviert (vgl. UC012). • Kamerapositioniereinheit ist betriebsbereit und eingeschaltet. • Das definierte Objekt hat sich aus dem Fokus der Kamera bewegt.
Nachbedingungen:	<ul style="list-style-type: none"> • Das zu verfolgende Objekt befindet sich erneut im Fokus der Kamera.
Fachlicher Auslöser:	Pilot möchte die Videoverfolgung eines definierten Objekts starten.
Ablauf:	<ul style="list-style-type: none"> • Abstand zwischen Kamerafokus und dem zu verfolgenden Objekt ermitteln. • Sollwerte für die Positioniereinheit berechnen und an sie übermitteln. • Positioniereinheit fährt auf die errechnete Soll-Position.



Tabelle 4.15: Use-Case: Objektverfolgung deaktivieren

UC015:	Objektverfolgung deaktivieren
Hauptakteur, Akteur:	<u>Pilot</u>
Vorbedingungen:	<ul style="list-style-type: none"> Objektverfolgung wurde erfolgreich aktiviert (vgl. UC012).
Nachbedingungen:	<ul style="list-style-type: none"> Objektverfolgung wurde erfolgreich beendet. Kamerapositioniereinheit befindet sich in Nullposition (s. Abbildung 1.7).
Fachlicher Auslöser:	Pilot möchte die Videoverfolgung eines definierten Objekts wieder beenden.
Ablauf:	<ul style="list-style-type: none"> Pilot betätigt den Schalter für die Objektverfolgung auf der Fernsteuerung. Die Verfolgung des Balls durch das AEVV-System wird deaktiviert. Kamera wird durch die Kamerapositioniereinheit in Nullstellung gefahren.

4.3.4 Kamerabild & Telemetriedaten verarbeiten

Die letzte Systemfunktion *Kamerabild & Telemetriedaten verarbeiten* wird in Abbildung 4.6 ausführlicher behandelt. Dazu müssen zunächst die Informationen der Sensoren aufbereitet werden. Diese werden anschließend zusammen mit dem Kamerabild an die Telemetriestation gesendet. Die Aufgabe dieser ist es, das empfangene Kamerabild und die Telemetriedaten anzuzeigen.

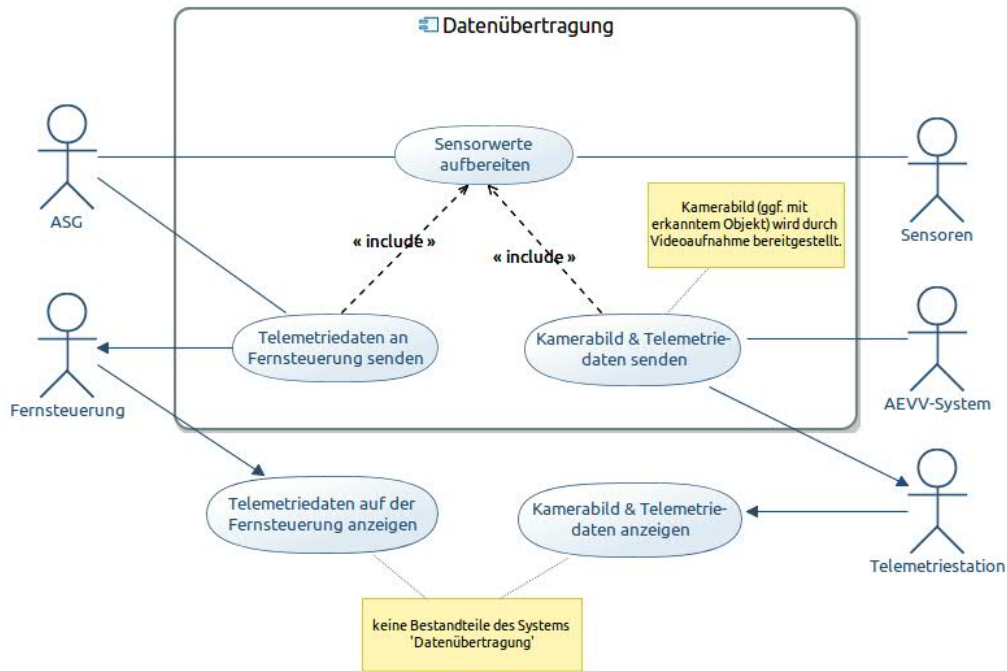


Abbildung 4.6: Use-Case: Kamerabild & Telemetriedaten verarbeiten

In den nachfolgenden Tabellen wird jeder einzelne Use-Case aus der Abbildung 4.6 detaillierter betrachtet.

Tabelle 4.16: Use-Case: Sensorwerte aufbereiten

UC016:	Sensorwerte aufbereiten
Hauptakteur, Akteur:	ASG, Sensoren
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter ist betriebsbereit.
Nachbedingungen:	<ul style="list-style-type: none"> • Sensorwerte für Akkustand und Flughöhe stehen zur Verfügung. • Sensorwerte können versendet werden.
Fachlicher Auslöser:	Akkustand und Flughöhe müssen dem Piloten kontinuierlich zur Verfügung gestellt.
Ablauf:	<ul style="list-style-type: none"> • Messwerte der vorhandenen Sensoren werden vom ASG ausgelesen. • ausgelesene Sensorwerte werden ggf. normiert und in die entsprechenden SI-Einheiten umgerechnet.



Tabelle 4.17: Use-Case: Telemetriedaten an Fernsteuerung senden

UC017:	Telemetriedaten an Fernsteuerung senden
Hauptakteur, Akteur:	ASG, Fernsteuerung
Vorbedingungen:	<ul style="list-style-type: none"> • Sensorwerte wurden erfolgreich aufbereitet (vgl. UC016).
Nachbedingungen:	<ul style="list-style-type: none"> • Telemetriedaten Akkustand und Flughöhe wurden an die Fernsteuerung versendet.
Fachlicher Auslöser:	Akkustand und Flughöhe müssen dem Piloten kontinuierlich zur Verfügung gestellt werden, damit dieser den Multikopter sicher steuern kann.
Ablauf:	<ul style="list-style-type: none"> • Die Sensorwerte Akkustand und Flughöhe werden in ein Datenframe gebündelt. • Das Frame wird über den Rückkanal der Fernsteuerung an diese gesendet. • Dieser Vorgang wird kontinuierlich wiederholt.

Tabelle 4.18: Use-Case: Telemetriedaten auf der Fernsteuerung anzeigen

UC018:	Telemetriedaten auf der Fernsteuerung anzeigen
Hauptakteur, Akteur:	Fernsteuerung
Vorbedingungen:	<ul style="list-style-type: none"> • Telemetriedaten wurden vom ASG an die Fernsteuerung versendet (vgl. UC017).
Nachbedingungen:	<ul style="list-style-type: none"> • Die empfangenen Datenframes werden aufbereitet und auf der Anzeige der Fernsteuerung ausgegeben.
Fachlicher Auslöser:	Akkustand und Flughöhe müssen dem Piloten kontinuierlich zur Verfügung gestellt werden, damit dieser den Multikopter sicher steuern kann.
Ablauf:	<ul style="list-style-type: none"> • Datenpakete über den Rückkanal der Fernsteuerung empfangen. • Akkustand und Flughöhe auf der Anzeige der Fernsteuerung darstellen. • Vorgang für jedes empfangene Datenframe wiederholen.



Tabelle 4.19: Use-Case: Kamerabild & Telemetriedaten senden

UC019:	Kamerabild & Telemetriedaten senden
Hauptakteur, Akteur:	<u>AEVV-System, Telemetriestation</u>
Vorbedingungen:	<ul style="list-style-type: none"> • Sensorwerte wurden erfolgreich aufbereitet (vgl. UC016). • Kamerabild steht zur Verfügung. • Funkverbindung zwischen AEVV-System und Telemetriestation vorhanden.
Nachbedingungen:	<ul style="list-style-type: none"> • Kamerabild & Telemetriedaten wurden erfolgreich an die Telemetriestation versendet.
Fachlicher Auslöser:	Kamerabild und Telemetriedaten sollen in einer Telemetriestation angezeigt werden.
Ablauf:	<ul style="list-style-type: none"> • Die Sensorwerte Akkustand und Flughöhe sowie das Kamerabild werden in einem Datenframe gebündelt. • Das Datenpaket wird an das Funkmodul des AEVV-Systems weitergeleitet. • Das Funkmodul sendet das Datenpaket an die Telemetriestation. • Vorgang wird kontinuierlich wiederholt.

Tabelle 4.20: Use-Case: Kamerabild & Telemetriedaten anzeigen

UC020:	Kamerabild & Telemetriedaten anzeigen
Hauptakteur, Akteur:	<u>Telemetriestation</u>
Vorbedingungen:	<ul style="list-style-type: none"> • AEVV-System sendet kontinuierlich Datenframes an die Telemetriestation (vgl. UC019).
Nachbedingungen:	<ul style="list-style-type: none"> • Kamerabild & Telemetriedaten werden auf der Anzeige der Telemetriestation ausgegeben.
Fachlicher Auslöser:	Kamerabild und Telemetriedaten sollen in einer Telemetriestation angezeigt werden.
Ablauf:	<ul style="list-style-type: none"> • Das Datenframe wird über das Empfangsmodul der Telemetriestation empfangen. • Aus dem Datenframe wird das Kamerabild, der Akkustand und die Flughöhe extrahiert. • Die Werte Akkustand und Flughöhe werden in das Kamerabild eingefügt. • Das resultierende Bild wird auf der Anzeige der Telemetriestation ausgegeben. • Vorgang wird für jedes empfangene Datenframe wiederholt.

4.3.5 Erweitertes Szenario

In diesem Abschnitt werden die Anwendungsfälle für das im Kapitel 1.4 beschriebene erweiterte Szenario erläutert. Das entsprechende Use-Case-Diagramm ist in Abbildung 4.7 dargestellt. Um die Einbettung in den Kontext eines Fußballspiels zu ermöglichen, muss das AEVV-System die Anzahl der erzielten Tore erfassen. Dies wird durch den Anwendungsfall *Anzahl geschossener Tore mitzählen* abgedeckt. Dazu ist es jedoch erforderlich, dass das AEVV-System die unterschiedlichen Torlinien erkennt (s. Anwendungsfall *Torlinien erkennen*). Parallel dazu wird der aktuelle Spielstand an die Telemetriestation übertragen. Im erweiterten Szenario existiert noch ein weiterer Anwendungsfall: *Flugrichtung vorschlagen*. Dieser wird ausgeführt, falls das Objekt die „Tracking Area“ des AEVV-Systems verlassen hat. In diesem Fall muss das AEVV-System dem Piloten eine Vorzugsflugrichtung vorschlagen, in welcher sich wahrscheinlich das gesuchte Objekt befindet.

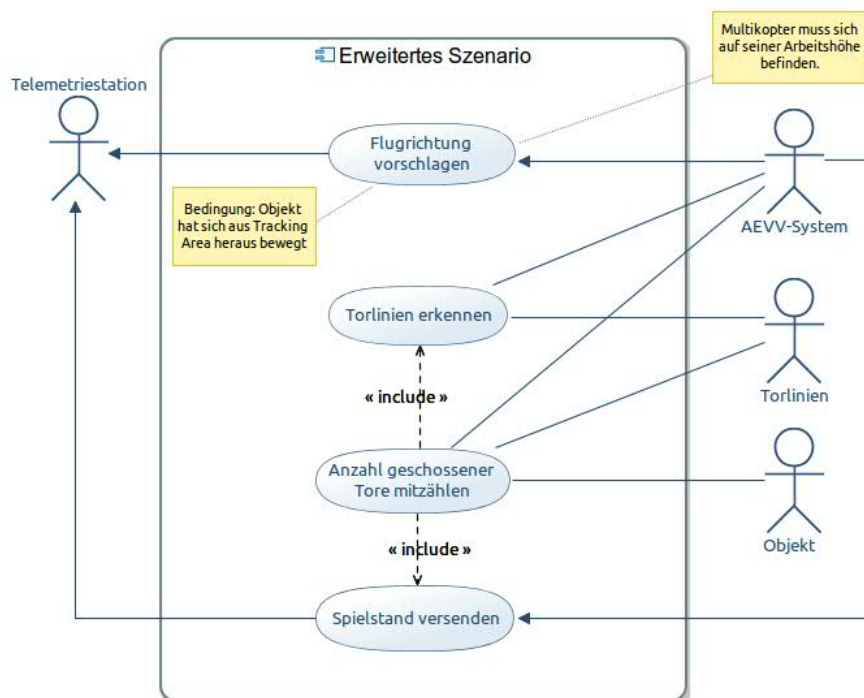


Abbildung 4.7: Use-Case: Erweitertes Szenario

In den nachfolgenden Tabellen wird jeder einzelne Use-Case aus der Abbildung 4.5 detaillierter definiert.



Tabelle 4.21: Use-Case: Flugrichtung vorschlagen

UC021:	Flugrichtung vorschlagen
Hauptakteur, Akteur:	AEVV-System, Pilot
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich in der Luft. • Videoaufnahme und Objektverfolgung sind aktiv. • Gesuchtes Objekt hat sich aus der „Tracking Area“ des AEVV-Systems heraus bewegt.
Nachbedingungen:	<ul style="list-style-type: none"> • Flugrichtung in der sich das Objekt wahrscheinlich befindet, wird dem Piloten vorgeschlagen.
Fachlicher Auslöser:	Pilot muss das gesuchte Objekt mit dem Multikopter verfolgen.
Ablauf:	<ul style="list-style-type: none"> • Bewegungsrichtung des gesuchten Objekts wird ermittelt. • Objekt wird nicht mehr erkannt, da es sich außerhalb der Tracking Area des AEVV-Systems befindet. • AEVV-System schlägt dem Piloten die zuletzt bekannte Bewegungsrichtung des Objekts als Flugrichtung vor.

Tabelle 4.22: Use-Case: Torlinien erkennen

UC022:	Torlinien erkennen
Hauptakteur, Akteur:	AEVV-System, Torlinien
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich in der Luft über dem Spielfeld. • Videoaufnahme und Objektverfolgung sind aktiv.
Nachbedingungen:	<ul style="list-style-type: none"> • Zwei Torlinien im Spielfeld werden erkannt.
Fachlicher Auslöser:	AEVV-System muss die Torlinien detektieren.
Ablauf:	<ul style="list-style-type: none"> • Das AEVV-System greift auf die bereitgestellten Videodaten zu. • Das AEVV-System versucht die Torlinien mit Hilfe der digitalen Bildverarbeitung zu erkennen.



Tabelle 4.23: Use-Case: Anzahl geschossener Tore mitzählen

UC023:	Anzahl geschossener Tore mitzählen
Hauptakteur, Akteur:	Kamerapositioniereinheit, Objekt, Torlinien
Vorbedingungen:	<ul style="list-style-type: none"> • Multikopter befindet sich in der Luft, sodass er die jeweilige Torlinie auf Spielfeld erkennen kann. • Videoaufnahme und Objektverfolgung sind aktiv. • benötigte Torlinie im Spielfeld wurde erkannt (vgl. UC022).
Nachbedingungen:	<ul style="list-style-type: none"> • Die erzielten Tore werden gezählt.
Fachlicher Auslöser:	AEVV-System muss den Spielstand verfolgen.
Ablauf:	<ul style="list-style-type: none"> • Sobald sich das Objekt aus dem Spielfeld über eine Torlinie bewegt, wird dies als Tor gezählt.

Tabelle 4.24: Use-Case: Spielstand senden

UC024:	Spielstand senden
Hauptakteur, Akteur:	AEVV-System, Telemetriestation
Vorbedingungen:	<ul style="list-style-type: none"> • Die erzielten Tore werden gezählt (vgl. UC023).
Nachbedingungen:	<ul style="list-style-type: none"> • Spielstand wurde an die Telemetriestation gesendet.
Fachlicher Auslöser:	AEVV-System muss den Spielstand aktualisieren.
Ablauf:	<ul style="list-style-type: none"> • AEVV-System sendet den aktuellen Spielstand an die Telemetriestation. • Beim Erkennen eines Tors wird der Spielstand aktualisiert.



5 Risikoanalyse

In diesem Kapitel werden mögliche Risiken, welche im Laufe des Projektes auftreten können, aufgelistet. Die Risiken wurden von den einzelnen Teilgruppen, welche aus Experten in den jeweiligen Teilgebieten bestehen, erfasst und an das Projektmanagement weitergeleitet.

Ein Risiko wird hinsichtlich seiner Auftrittswahrscheinlichkeit und der Schadenswirkung bewertet (s. Tabelle 5.1).

Tabelle 5.1: Skalierung von Auftrittswahrscheinlichkeiten und Schadenswirkungen

Skalierung Auftrittswahrscheinlichkeit	
1	sehr unwahrscheinlich
2	unwahrscheinlich
3	wahrscheinlich
4	sehr wahrscheinlich
Skalierung Schadenswirkung	
1	unbedeutend
2	gering
3	erheblich
4	katastrophal

Mit Hilfe einer Risikomatrix (s. Tabelle 5.2) und unter Berücksichtigung der Auftrittswahrscheinlichkeit, sowie der Schadenswirkung, wird ein Gesamtrisiko für jedes aufgelistete Risiko bestimmt. Dieses wird, wie in Tabelle 5.3 dargestellt, skaliert.

Tabelle 5.2: Risikomatrix

Auftrittswahrscheinlichkeit	4	5	6	7	8
	3	4	5	6	7
	2	3	4	5	6
	1	2	3	4	5
		1	2	3	4
		Schadenswirkung			

Tabelle 5.3: Skalierung des Risikos

Skalierung des Risikos	
3	vernachlässigbar
4	gering
6	mittel
>6	hoch

Für jedes Risiko werden mögliche Gegenmaßnahmen vorgestellt und daraufhin das Risiko neu bewertet.



Diese Risiken werden in vier Bereiche unterteilt:

- Sicherheitskritische Risiken,
- technische Risiken,
- organisatorische Risiken und
- Teamrisiken.

Unter den *sicherheitskritischen Risiken* sind die Risiken zu verstehen, welche die technische Sicherheit des Systems betreffen. Bei Auftreten eines solchen Fehlerfalles kann es zu Umwelt- und Personenschäden kommen. In der Tabelle 5.4 sind nur die sicherheitskritischen Risiken aufgelistet, welche aufgrund des zeitlichen Rahmens innerhalb dieser Arbeit nicht weiter verfolgt werden. Im Kapitel 6.7 werden die Risiken, welche in diesem Projekt bearbeitet werden, näher erläutert.

Die *technischen Risiken* beziehen sich auf Hard- und Softwarerisiken, welche während der Entwicklungsphase auftreten können.

Risiken, die den Projektablauf z. B. bezüglich der Zeitplanung betreffen, werden unter *organisatorischen Risiken* aufgelistet.

Unter *Teamrisiken* sind die Risiken zu verstehen, welche bei den einzelnen Teammitgliedern auftreten könnten.

Folgende Tabelle 5.4 listet mögliche Risiken auf, die im weiteren Verlauf des Projektes auftreten könnten.

Legende:

W: Wahrscheinlichkeit

A: Auswirkung

RA: Risikoampel

RRA: Restrisikoampel

Tabelle 5.4: Risikotabelle

Nr.	Risikobeschreibung	W.	A.	RA	Gegenmaßnahme	W.	A.	RRA
Sicherheitskritische Risiken								
RI001:	Motor Ausfall / Rotorverlust	1	4	5	Algorithmus zur Erkennung eines defekten Motors bzw. Rotorverlust implementieren. Bei Erkennung eines solchen Fehlers Notlandung einleiten.	1	2	3
RI002:	Zerstörung des Akkus durch Überlastung während des Fluges	1	3	4	Akku überdimensionieren, Temperatur überwachen, Akku luftig anbringen (Konvektionskühlung), Stromaufnahme begrenzen und messen	1	1	2
RI003:	Elektromagnetische Interferenzen beeinflussen Lageregelung	2	4	6	Leitungen schirmen, Galvanische Trennung, DC/DC- Wandler, sternförmiger Massepunkt	1	4	5
RI004:	Ausfall von Sensoren	1	4	5	Qualitativ hochwertige Sensoren kaufen, Sensoren redundant auslegen, Fehler in Sensorwerten erkennen	1	2	3
Technische Risiken								
RI005:	Platine während der Entwicklungsphase falsch angesteuert → Bauteile und/oder Leiterkarte können zerstört werden	2	4	6	Verfügbarkeit spezieller Ersatzbauteile und Ersatzplatinen sicherstellen	2	2	4

Tabelle 5.4: Risikotabelle (Fortsetzung)

Nr.	Risikobeschreibung	W.	A.	RA	Gegenmaßnahme	W.	A.	RRA
RI006:	Kameraschnittstelle kann nicht aus-gewertet werden	2	4	6	Die Kamera muss eine alternative Schnittstelle anbieten	1	2	3
RI007:	Kamerapositionierungseinheit hat einen zu hohen Stromverbrauch	2	3	5	Regelparameter anpassen, Gewicht verringern durch alternative Komponenten	1	3	4
RI008:	Regler-Stellgröße über 100%	2	2	4	Auslage der Regler beachten, Auswahl der Motoren beachten	1	2	3
RI009:	Objekterkennung in Bilddaten nicht umsetzbar, wegen fehlerhaften Al-gorithmus oder schlechten Lichtver-hältnissen	2	4	6	Anforderungen an Umwelt und Al-gorithmus stellen, Objekt erkennbar färben	2	1	3
RI010:	Funkverbindung zur Telemetriesta-tion bricht ab oder Daten gehen ver-loren	2	2	4	Wechsel des Funkstandards, Sicher-heitsprotokoll implementieren	2	1	3
RI011:	Bandbreite der Funkverbindung zur Telemetriestation ist zu gering	2	2	4	Wechsel des Standards, Weniger Daten übertragen (geringere Auflö-sung, Komprimierung)	1	2	3
RI012:	Positioniereinheit erfüllt nicht die Anforderungen, durch fehlerhafte Installation oder Mechanik	3	2	5	Austausch der Mechanik, Gewicht der Kamera gering halten	2	2	4
RI013:	Inkompatible Schnittstellen zwi-schen Teilsystemen	2	4	6	Schnittstellenadapter, Standardi-sierte Schnittstellen verwenden	1	2	3

Tabelle 5.4: Risikotabelle (Fortsetzung)

Nr.	Risikobeschreibung	W.	A.	RA	Gegenmaßnahme	W.	A.	RRA
RI014:	Kein akzeptables Bild der Kamera, durch Erschütterungen, Flugeschwindigkeit oder unzureichenden Zoom	2	3	5	Flugeschwindigkeit reduzieren, schwingungsreduzierende Elemente verwenden	1	3	4
RI015:	Torlinien können nicht erkannt werden	1	3	4	Gut erkennbare Farbe der Linien, Belichtung des Raumes anpassen	1	2	3
RI016:	Ungenauigkeit in der Torerkennung	3	3	6	Torgröße verringern, Multikopter muss durch Piloten besser positioniert werden	2	2	4
RI017:	Spannungseinbrüche durch Lastschwankungen	3	3	6	Entkopplung von Leistungselektromotoren, Pufferkondensatoren einsetzen	1	3	4
Organisatorische Risiken								
RI018:	Risiken werden falsch eingeschätzt	2	4	6	Risiken durch mehrere Leute und in einem iterativen Prozess abschätzen	1	4	5
RI019:	Projektziele werden nicht erreicht bzw. das Ergebnis trifft nicht die erfassten Anforderungen	2	3	5	Regelmäßige Absprache mit den Betreuern und Teammeetings	1	3	4
RI020:	Zeitmanagement des Projektplans ist unzureichend	2	4	6	Einplanen von Zeitpuffern bei kritischen Meilensteinen	2	1	3
RI021:	Während des Projektes werden neue Anforderungen gestellt	1	4	5	Aufgabenplanung und regelmäßige Absprache mit den Betreuern	1	2	3
Teamrisiken								

Tabelle 5.4: Risikotabelle (Fortsetzung)

Nr.	Risikobeschreibung	W.	A.	RA	Gegenmaßnahme	W.	A.	RRR
RI022:	Krankheitsbedingter Ausfall eines Projektgruppenmitglieds	2	3	5	Übernahme der Arbeit durch andere Teammitglieder	2	2	4
RI023:	Motivation des Projektgruppenmitglieds lässt mit der Zeit nach	1	4	5	Meilensteine in geringeren Abständen aufeinander folgen lassen um Erfolgserlebnisse zu erreichen. Soziale Events nutzen um Team neu zu motivieren.	1	2	3

6 Anforderungsspezifikation

In diesem Kapitel wird der Systemkontext für das Projekt Avionic Architecture abgegrenzt, indem die nötigen Anforderungen spezifiziert werden. Diese lassen sich aus dem im Kapitel 1.4 definierten Szenario und den dazugehörigen Anwendungsfällen (s. Kapitel 4.3) ableiten. Die daraus resultierenden Anforderungen decken den Entwicklungsrahmen für die anschließenden Phasen ab und dienen der späteren Testphase als Grundlage für die Erstellung von Testfällen bzw. -szenarien.

Dazu werden funktionale und nicht funktionale Anforderungen für die folgenden fünf Teilbereiche erhoben:

- Avionik Steuergerät (ASG),
- AEVV-System,
- Fernsteuerung,
- Telemetriestation und
- Umwelt.

Zu einer besseren Nachvollziehbarkeit und einer eindeutigen Verfolgbarkeit wird jede Anforderung aus einer ID, einer Bezeichnung und einer Beschreibung konstruiert. Eine ID bzw. eine einheitliche Abkürzung (vgl. Tabelle 6.1) ermöglicht, gerade in späteren Phasen des Projekts, eine eindeutige Zurückverfolgbarkeit (im Folgenden Traceability genannt), z. B. von den Anforderungen über die Implementierung bis hin zu den Testfällen.

Für dieses Projekt ist die Struktur der Traceability in Abbildung 6.1 dargestellt.

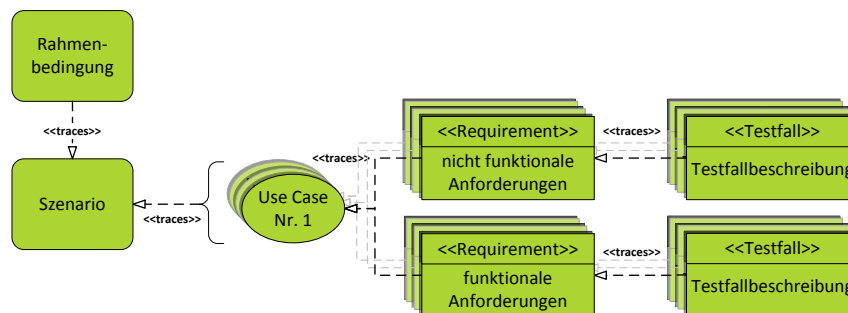


Abbildung 6.1: Verlauf der Traceability innerhalb des Projekts

Das ausgewählte Szenario bildet den Grundstein für die Funktionalität des Gesamtsystems. Aus dem Szenario resultieren alle Anwendungsfälle, aus denen sich daraufhin die funktionalen und nicht funktionalen Anforderungen ableiten lassen. Dabei hat jede Anforderung eine Referenz auf den dazugehörigen Anwendungsfall. Für die spätere Testphase wird zu jeder Anforderung mindestens ein Testfall angelegt, welcher auf die Anforderung verweist.

Tabelle 6.1: Eindeutige Abkürzungen für die Anforderungserhebung

Abkürzung	Bedeutung
FM	Funktionale Anforderung an das ASG
NM	Nicht funktionale Anforderung an das ASG
FA	Funktionale Anforderung an das AEVV-System
NA	Nicht funktionale Anforderung an das AEVV-System
FF	Funktionale Anforderung an die Fernsteuerung
NF	Nicht funktionale Anforderung an die Fernsteuerung
FT	Funktionale Anforderung an die Telemetriestation
NT	Nicht funktionale Anforderung an die Telemetriestation
FO	Optionale funktionale Anforderung
NO	Optionale nicht funktionale Anforderung
RB	Anforderung an die Umwelt
SR	Sicherheitsanforderung

6.1 Anforderungen an das ASG

Im folgenden Abschnitt werden die Anforderungen an das ASG erhoben.

Tabelle 6.2: Anforderungen an das ASG

Bez.	Anforderung	Ref.
FM001:	Multikopter fliegen Der Multikopter muss sich in der Luft halten und fortbewegen können. Dabei muss er ein stabiles Flugverhalten aufweisen.	UC001 - UC004
FM002:	Multikopter steuern Der Pilot muss jederzeit in der Lage sein den Multikopter über seine Fernsteuerung zu steuern.	UC001 - UC004
FM003:	Empfangen von Steuerbefehlen Das ASG muss über ein Empfangsmodul die von der Fernsteuerung gesendeten Steuerbefehle empfangen.	FM002

Tabelle 6.2: Anforderungen an das ASG (Fortsetzung)

Bez.	Anforderung	Ref.
FM004:	Interpretation der Steuerbefehle Das ASG muss die empfangenen Steuerbefehle interpretieren und in Führungsgrößen für die Roll-, Nick- und Gierachse, sowie den Schub umwandeln.	FM003
FM005:	Bestimmung der Lage Das ASG muss die aktuelle Lage des Multikopters im Raum bestimmen können.	FM001
FM006:	Bestimmung der Flughöhe Das ASG muss die aktuelle Flughöhe des Multikopters bestimmen können.	UC007 UC008 FM008
FM007:	Autonome Regelung der Achsen Auf Basis der Führungsgrößen der Roll-, Nick und Gierachse soll eine autonome Lageregelung durchgeführt werden. Dazu müssen die Stellwerte der Motoren aus den Führungsgrößen berechnet werden.	UC006 FM001 FF003
FM008:	Autonome Regelung der Flughöhe Bei aktivierter Flughöhenregelung soll der Multikopter bei unveränderten Führungsgrößen seine aktuelle Flughöhe beibehalten.	FM006 FF003
FM009:	Motorstellwerte weitergeben Die berechneten Stellwerte müssen an die Motoren weitergegeben werden.	FM007
FM010:	Sensorwerte zur Bestimmung der Lage Aus den Sensoren muss die Lage des Multikopters bestimmt werden.	UC005 FM005
FM011:	Sensorwerte zur Bestimmung der Flughöhe Aus den Sensoren muss die Flughöhe des Multikopters bestimmt werden.	UC016 FM006
FM012:	Sensorwerte zur Bestimmung des Akkustands Aus den Sensoren muss der Akkustand des Multikopters bestimmt werden.	UC016



Tabelle 6.2: Anforderungen an das ASG (Fortsetzung)

Bez.	Anforderung	Ref.
FM013:	Schnittstelle zwischen den Systemen ASG und AEVV Zum Austausch der Telemetriedaten und Steuersignale muss es eine Schnittstelle zwischen dem ASG und dem AEVV-System geben. Unter Steuersignalen wird in diesem Kontext die Aktivierung und Deaktivierung von Bildaufnahme und Objektverfolgung verstanden.	FA007
FM014:	Abgleich des Akkustands Das ASG muss den Akkustand kontinuierlich mit einem kritischen Wert vergleichen.	FM012
FM015:	Warnung bei kritischem Akkustand Bei Erreichen oder Unterschreiten des kritischen Werts des Akkustands muss das ASG eine Warnmeldung über den Rückkanal an die Fernsteuerung versenden.	FM014
NM001:	Funkverbindung ASG - Fernsteuerung Das ASG muss mit einem Transceiver zur Kommunikation mit der Fernsteuerung ausgestattet werden.	FM003
NM002:	Energieeffiziente Regelung Die Regelalgorithmen müssen effizient hinsichtlich des Energieverbrauchs sein, sodass die Flugdauer für das Szenario ausreicht.	Kapitel 1.4
NM003:	Maßeinheit der Flughöhe Als Maßeinheit für die Flughöhe wird die SI-Einheit Meter verwendet.	FA007
NM004:	Maßeinheit des Akkustands Als Maßeinheit für den Akkustand wird die SI-Einheit Volt verwendet.	FA007
NM005:	Minimale Spannung des Akkus Die minimale Spannung des Akkus darf 3.2 V/ Zelle nicht unterschreiten.	FM015

6.2 Anforderungen an das AEVV-System

Im folgenden Abschnitt werden die Anforderungen an das AEVV-System erhoben.



Tabelle 6.3: Anforderungen an das AEVV-System

Bez.	Anforderung	Ref.
FA001:	Bildaufnahme aktivieren und deaktivieren Bei Eingang des Steuersignals für die (De-)Aktivierung der Bildaufnahme muss diese (de-)aktiviert werden. Dabei wird jedes Mal die Kamera durch ihre Positioniereinheit in Nullposition gefahren. Bei Deaktivierung werden sie stromlos geschaltet.	UC009 + UC011 FM013 FF001
FA002:	Objekterkennung aktivieren und deaktivieren Bei Eingang des Steuersignals für die (De-)Aktivierung der Objekterkennung muss diese (de-)aktiviert werden.	UC012 + UC015 FM013 FF002
FA003:	Kamerabild bereitstellen Die eingeschaltete Kamera muss dem AEVV-System fortwährend Bilddaten zur Verfügung stellen.	UC010
FA004:	Objekt erkennen Wenn die Objekterkennung eingeschaltet ist, muss fortwährend versucht werden ein vorher definiertes Objekt in den Bilddaten zu erkennen.	UC013
FA005:	Funkverbindung AEVV-System - Telemetriestation Es muss eine drahtlose Verbindung zwischen dem AEVV-System und der Telemetriestation bereitgestellt werden.	FA006 FA007
FA006:	Kamerabild versenden Das AEVV-System muss die von der Kamera empfangenen Bilddaten über die drahtlose Verbindung an die Telemetriestation streamen.	UC019
FA007:	Senden der Telemetriedaten Das AEVV-System muss die Telemetriedaten über die drahtlose Verbindung an die Telemetriestation senden.	UC019
FA008:	Positionierung der Kamera Eine Positioniereinheit muss für die Kamera installiert werden, um diese unabhängig vom Multikopter positionieren zu können.	FA009



Tabelle 6.3: Anforderungen an das AEVV-System (Fortsetzung)

Bez.	Anforderung	Ref.
FA009:	Zentrierung des Objektes Wenn sich das zu verfolgende Objekt aus dem Zentrum der Kamera bewegt, muss die Positioniereinheit diese automatisch neu justieren, sodass sich das Objekt wieder im Zentrum befindet.	UC014
NA001:	Genauigkeit der Positionierungseinheit Die Positioniereinheit muss eine ausreichende Genauigkeit besitzen, um das Objekt im Zentrum des Kamerabildes zu fokussieren.	FA008
NA002:	Reaktionszeit der Positioniereinheit Die Positioniereinheit muss eine ausreichende Reaktionszeit für das Szenario besitzen.	FA008 Kapitel 1.4
NA003:	Erschütterungsresistenz der Kamera Die Kamera muss unempfindlich gegenüber Erschütterungen/Vibrationen sein, welche z. B. beim Multikopterflug auftreten können.	FA008
NA004:	Einhaltung der Nutzlast Das Gesamtgewicht der Peripherie darf die Nutzlast des Multikopters nicht überschreiten. Die konkrete Nutzlast ist von der Wahl des Multikopters abhängig.	FM001
NA005:	Einbau der Peripherie Das Start- und Landeverhalten des Multikopters darf durch die montierte Peripherie nicht behindert werden.	Kapitel 5
NA006:	Bandbreite der Funkverbindung zur Telemetriestation Es muss eine Funkverbindung gewählt werden, die genügend Bandbreite bereitstellt um die Bilddaten der Kamera an die Telemetriestation zu streamen.	FA005 NA007
NA007:	Auflösung des Videostreams Die zur Telemetriestation gestreamten Bilddaten müssen mindestens eine WVGA-Auflösung (mind. 720 × 540 Pixel) aufweisen.	FA003

Tabelle 6.3: Anforderungen an das AEVV-System (Fortsetzung)

Bez.	Anforderung	Ref.
NA008:	<p>Verdeckung des Objekts</p> <p>Ist das zu erkennende Objekt für einen Zeitraum von mehr als 4 Sekunden verdeckt, wird die Kamera in Nullposition gebracht. Der Pilot ist in der Verantwortung den Multikopter so zu positionieren, dass das zu erkennende Objekt wieder im Sichtfeld der Kamera ist.</p>	FA004

6.3 Anforderungen an die Fernsteuerung

Im folgenden Abschnitt werden die Anforderungen an die Fernsteuerung erhoben.

Tabelle 6.4: Anforderungen an die Fernsteuerung

Bez.	Anforderung	Ref.
FF001:	<p>Schalter für die Bildaufnahme</p> <p>Die Fernbedienung muss einen Schalter für das Ein- und Ausschalten der Bildaufnahme bereitstellen.</p>	UC009 + UC011 FA001
FF002:	<p>Schalter für die Objekterkennung</p> <p>Die Fernsteuerung muss einen Schalter für das Ein- und Ausschalten der Objekterkennung bereitstellen.</p>	UC012 + UC015 FA002
FF003:	<p>Schalter für die Flughöhenregelung</p> <p>Die Fernsteuerung muss einen Schalter für das Ein- und Ausschalten der autonomen Flughöhenregelung bereitstellen.</p>	UC007 UC008 FM008
FF004:	<p>Senden der Telemetriedaten</p> <p>Der Multikopter muss seine Telemetriedaten über den Rückkanal der Fernsteuerung an diese versenden.</p>	UC017
FF005:	<p>Empfangen der Telemetriedaten</p> <p>Die Fernsteuerung muss die Telemetriedaten des Multikopters über die Funkverbindung empfangen.</p>	FF004
FF006:	<p>Anzeigen der Telemetriedaten</p> <p>Die empfangenen Telemetriedaten müssen auf der Anzeige der Fernsteuerung ausgegeben werden.</p>	UC018
NF001:	<p>Aktualisierung der Telemetriedaten</p> <p>Die Telemetriedaten müssen sekundlich aktualisiert werden.</p>	FF006



6.4 Anforderungen an die Telemetriestation

Im folgenden Abschnitt werden die Anforderungen an die Telemetriestation erhoben.

Tabelle 6.5: Anforderungen an die Telemetriestation

Bez.	Anforderung	Ref.
FT001:	Bildschirm der Telemetriestation Die Telemetriestation muss einen Bildschirm für die Anzeige der Bild- und Telemetriedaten besitzen.	UC020
FT002:	Empfangen der Daten Die Telemetriestation muss die zu ihr übermittelten Daten empfangen.	UC019 FA005
FT003:	Visualisierung der empfangenen Daten Die Telemetriestation muss die zu ihr übermittelten Daten auf dem Bildschirm darstellen.	FT002

6.5 Anforderungen an die Umwelt

Im folgenden Abschnitt werden die Anforderungen an die Umwelt erhoben. Diese werden aus dem Kapitel 4.2 abgeleitet.

Tabelle 6.6: Anforderungen an die Umwelt

Bez.	Anforderung	Ref.
RB001:	Flugort Der Multikopter darf nur in einem großen, leeren und abgeschlossenen Raum (z.B. Turnhalle) geflogen werden. Die Grundfläche des Raumes muss mindestens so groß sein wie das Spielfeld. Die Höhe muss mindestens für das Szenario ausreichen.	Kapitel 4.2
RB002:	Belichtung Der Raum, in dem der Multikopter geflogen wird, muss für die Kamera und Objektverfolgung geeignet ausgeleuchtet sein.	Kapitel 4.2
RB003:	Anzahl der Bälle Auf dem Spielfeld darf nur mit einem Ball gespielt werden.	Kapitel 4.2

Tabelle 6.6: Anforderungen an die Umwelt (Fortsetzung)

Bez.	Anforderung	Ref.
RB004:	Maximale Geschwindigkeit des Balls Der Ball darf eine Geschwindigkeit von 2 m s^{-1} (tbc) nicht überschreiten.	Kapitel 4.2
RB005:	Farbe des Balls Der Ball muss sich in seiner Farbe deutlich von der gesamten Umgebung unterscheiden.	Kapitel 4.2
RB006:	Größe des Balls Der Ball muss einen Umfang von 58 bis 60 cm besitzen.	Kapitel 4.2
RB007:	Gewicht des Balls Das Gewicht des Balls muss mindestens so groß sein, dass dieser nicht durch den vom Multikopter erzeugten Luftstrom bewegt wird.	Kapitel 4.2
RB008:	Größe des Spielfelds Um einen ausreichenden Spielfluss zu gewährleisten, muss das Spielfeld mindestens 8x5 Meter groß sein.	Kapitel 4.2

6.6 Optionale Anforderungen

Im folgenden Abschnitt werden optionale Anforderungen an das Gesamtsystem erhoben, welche sich aus dem erweiterten Szenario (s. Kapitel 1.4) ergeben.

Tabelle 6.7: Optionale Anforderungen an das Gesamtsystem

Typ	Bez.	Anforderung	Ref.
AEVV-System	FO001:	Vorzugsflugrichtung vorschlagen Das AEVV-System muss dem Piloten eine Vorzugsflugrichtung vorschlagen, wenn die Objekterkennung aus technischen Gründen nicht mehr möglich ist, um den Ball möglichst schnell wiederzufinden.	Kapitel 1.4
	FO002:	Vorzugsflugrichtung an Telemetriestation senden Das AEVV-System muss die Vorzugsflugrichtung an die Telemetriestation senden.	FO001

Tabelle 6.7: Optionale Anforderungen an das Gesamtsystem (Fortsetzung)

Typ	Bez.	Anforderung	Ref.
	FO003:	Spielstand erkennen Überschreitet der Spielball eine der Torlinien, muss für das entsprechende Team der Punktestand aktualisiert werden.	NO001
	FO004:	Spielstand an Telemetriestation senden Der Spielstand muss an die Telemetriestation gesendet werden.	FO003
Telemetriestation	FO005:	Spielstand anzeigen Der Spielstand muss auf der Telemetriestation angezeigt werden.	FO004
	FO006:	Weiterleiten der Bilddaten Die Telemetriestation kann die Bilddaten über einen Webbrowser zur Verfügung stellen.	FT002
Umwelt	NO001:	Farbe der Torlinien Die Torlinien müssen sich in ihrer Farbe deutlich von der Umwelt und untereinander unterscheiden.	-

6.7 Sicherheitsanforderungen

Im folgenden Abschnitt werden die Sicherheitsanforderungen erhoben. Um den Umfang für eine Projektgruppe beherrschbar zu halten, wird nur die Störung der Kommunikation zwischen der Fernsteuerung und dem ASG betrachtet.

Tabelle 6.8: Anforderungen an die Sicherheit

Bez.	Anforderung	Ref.
SR001:	Erkennen einer Störung Das ASG muss eine Störung der Funkverbindung zur Fernsteuerung selbstständig erkennen.	-
SR002:	Reaktion auf eine Störung Beim Eintreten einer Störung muss das ASG die Roll- und Nickachse des Multikopters in die Nullposition bringen und die Flughöhe halten.	SR001



6.8 Spezifikation der Echtzeitanforderungen

Die Spezifikation der Echtzeitanforderungen ist ein wichtiger Bestandteil für alle echtzeitkritischen Systeme. Durch diesen Schritt werden alle echtzeitkritischen Tasks in einen zeitlichen Zusammenhang gebracht. Dadurch kann eine spätere Scheduling-Analyse garantieren, dass ein System seine Deadline nicht verletzt und somit seine Ergebnisse korrekt liefert.

In diesem Projekt gibt es die zwei Teilsysteme (s. Kapitel 4)

- ASG und
- AEVV.

Neben der Spezifikation der Echtzeitanforderungen müssen diese Anforderungen auch evaluiert werden. Dieser gesamte Prozess wird im folgenden Echtzeitanalyse genannt (s. Abbildung 6.2). Innerhalb dieser Echtzeitanalyse durchlaufen die betroffenen Anforderungen bestimmte Phasen, die im folgenden genauer erklärt werden.

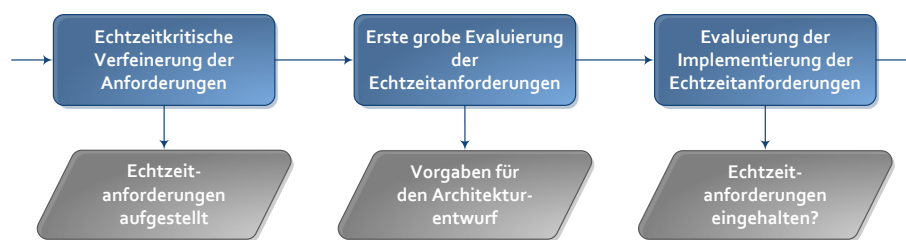


Abbildung 6.2: Prozess für die Echtzeitanalyse (blau: Phase; grau: Phasenergebnis)

Während dieses Prozesses werden drei Phasen durchgeführt:

Phase 1: Die erste Phase ist die Erhebung der Echtzeitanforderungen. Dazu zählt die Ermittlung der betroffenen Tasks aus den aufgestellten Anforderungen (s. Kapitel 6) sowie die Abschätzung der Echtzeitanforderungsaspekte. Für jedes dieser Teilsysteme wird eine einzuhaltende Deadline definiert, gefolgt von ersten Abschätzungen über die möglichen Ausführungszeiten der einzelnen Aufgaben.

Nachdem die betroffenen Anforderungen ermittelt wurden, werden sie in einen zeitlichen Abhängigkeitsgraph (s. Abbildung 6.3 und Abbildung 6.6) gebracht. Danach wird jede Komponente echtzeitkritisch betrachtet, sodass daraus eine Deadline abgeleitet werden kann. Dieser Schritt wird in der Regel durch Worst-Case-Abschätzungen durchgeführt.

Phase 2: Innerhalb dieser Phase wird eine erste grobe Evaluierung durchgeführt, um eine Aussage über die Realisierbarkeit dieser Echtzeitanforderungen zu treffen. Diese Schlussfolgerung soll als Grundlage für das anliegende Architektur-Mapping dienen (s. Kapitel 7.5).



Phase 3: Die letzte Phase findet während der Implementierungsphase statt. Hierbei werden die tatsächlichen Ausführungszeiten ermittelt und anhand der definierten Echtzeitaspekte evaluiert (s. Kapitel 10.4.2.1). Bei Nichteinhaltung der Echtzeitanforderungen werden Gegenmaßnahmen und Optimierungen vorgeschlagen.

6.8.1 Erhebung der Echtzeitanforderungen für das ASG

Im Rahmen des ASGs sind folgende Anforderungen aus dem Kapitel 6 betroffen, die in den folgenden Abschnitten einer zeitlichen Verfeinerung unterzogen werden:

- FM010: Sensorwerte zur Bestimmung der Lage,
- FM011: Sensorwerte zur Bestimmung der Flughöhe,
- FM007: Autonome Regelung der Achsen,
- FM008: Autonome Regelung der Flughöhe,
- FM009: Motorstellwerte weitergeben,
- FM003: Empfangen von Steuerbefehlen,
- FM014: Abgleich des Akkustands und
- FF004: Senden der Telemetriedaten.

Die Worst-Case-Abschätzungen für dieses System werden anhand von praktischen Erfahrungen sowie von verwendeten Erwartungswerten aus der aktuellen Forschung und der Industrie abgeleitet.

6.8.1.1 Autonome Höhen- und Lageregelung

Damit der Multikopter ein stabiles und sicheres Flugverhalten (s. FM001) aufweisen kann, muss sich dieser einer dauerhaften, autonomen Höhen- (s. FM008) und Lageregelung (s. FM007) unterziehen. Hinter diesem Prozess verbirgt sich eine Kette von Folgeprozessen, deren Ausführungsreihenfolge in Abbildung 6.3 dargestellt wird.

Bevor die eigentliche Regelung durchgeführt wird, müssen alle relevanten Sensoren ausgelesen werden. Die Gyroskope und der Beschleunigungssensor sind dabei die wichtigsten Sensoren und werden folglich immer für jeden Durchlauf des Regelkreislaufrs einmal ausgelesen. Die Werte des Barometers und des Hallsensors werden nicht so häufig auf den Sensorchip aktualisiert. Demnach werden diese Sensoren nicht für jeden Durchlauf des Regelkreises erneut ausgelesen. In der Abbildung 6.3 wird das Erfassen der Sensorwerte sequentiell dargestellt¹⁵.

Liegen die aktuellen Werte der Sensorik vor (s. Abbildung 6.3, roter Rahmen), werden die Lagewinkel und die Position entlang der z-Achse (Istwerte) aktualisiert. Nachdem die Sensoren ausgelesen wurden, wird daraus die aktuelle Lage und Flughöhe als Rückführgrößen berechnet (s. Abbildung 6.3, grüner Rahmen). Anhand der durch die Funksteuerung

¹⁵Unter gewissen Umständen kann diese Aufgabe auch teilweise parallel ausgeführt werden. Dies ist z. B. möglich, wenn Sensoren über eigene, dedizierte Busse angeschlossen werden.

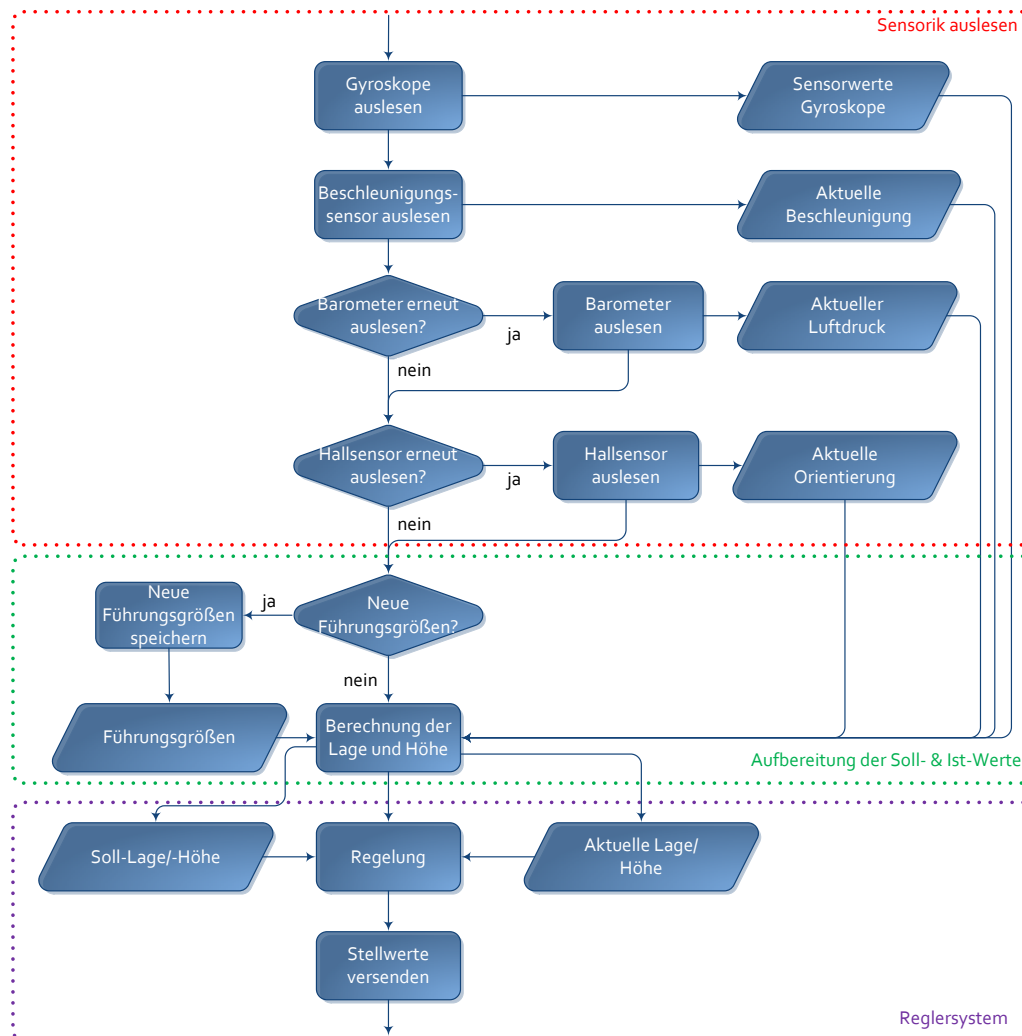


Abbildung 6.3: abstrahierter Ablauf der Höhen- und Lageregelung (funktionale Software)

eingestellten Führungsgrößen und der ermittelten Rückführgrößen der Sensoren kann die Regelung durchgeführt werden (s. Abbildung 6.3, violetter Rahmen). Daraufhin werden die ermittelten Stellwerte an die einzelnen Motorregler gesendet.

Um eine Deadline für diese Hauptaufgabe zu definieren, wird für jede Teilaufgabe eine Periode geschätzt. Da es sich i.d.R. um eine sequentielle Abarbeitung handelt, ist die Periode der letzten Aufgabe die Deadline des gesamten Systems, da innerhalb dieser Zeit alle Aufgaben abgearbeitet sein müssen.

Bei all diesen Tasks handelt es sich um periodische Tasks, welche einer gewissen Abhängigkeit unterliegen. Die Aufgabe für das Versenden der Motorstellwerte kann erst durchgeführt werden, wenn die Regelung beendet wurde. Diese wiederum kann nur gestartet



werden, wenn die Soll- und Istwerte des Systems vorliegen. Dies hat zur Folge, dass die Periode T_{motor} für das Versenden der Motorstellwerte gleichzeitig die Gesamtperiode T_{ges} der autonomen Regelung widerspiegelt. Innerhalb dieser Periode müssen sowohl die vorherigen Tasks als auch diese Aufgabe selbst terminieren. Daraus lässt sich die folgende Bedingung für die Ausführungszeiten c_i der Tasks bei einer sequentiellen Abarbeitung ableiten:

$$\sum_{i=1}^n c_i + c_{motor} < T_{ges} \quad | \quad n := \text{Anzahl der vorherigen Tasks} \quad (6.1)$$

Um eine erste Abschätzung für diese Gesamtperiode zu erlangen, wird für jeden dieser Prozesse eine Periode ermittelt. Die Festlegung dieser Perioden bzw. Deadlines soll in den folgenden Abschnitten vertiefend erläutert werden.

Gyroskope und Beschleunigungssensoren auslesen Gyroskope und Beschleunigungssensoren müssen für die Regelung permanent ausgelesen werden (s. FM010). Diese kommunizieren typischerweise über einen Inter-Integrated-Circuit (I²C)-Bus, welcher i.d.R. mit einer Taktung von 400 kHz betrieben wird. Jeder dieser zwei Sensoren liefert 3 Messwerte – jeweils einen für die Roll-, Nick- und Gier-Achse. Folglich müssen für zwei Sensoren insgesamt 6 Messwerte ausgelesen werden, welche häufig 16 Bit lang sind. Der I²C-Bus verschickt Frames¹⁶ mit der Größe von 9 Bits (8 Datenbits + 1 Acknowledgde-Bit) über den Bus. Des Weiteren muss für einen einfachen Lesezugriff ein Protokoll-Overhead in Kauf genommen werden. Dieser besteht aus folgenden vier Frames [Eled]:

1. Slaveadresse (SA) inkl. Schreibzugriff auf den Bus,
2. Registeradresse (RA),
3. Slaveadresse (SA) inkl. Lesezugriff auf den Bus und
4. zu lesende Daten vom Slave.

Um schließlich ein 8-Bit Datenwert zu akquirieren, müssen pro Datenframe insgesamt vier Frames verschickt werden (s. Abbildung 6.4).

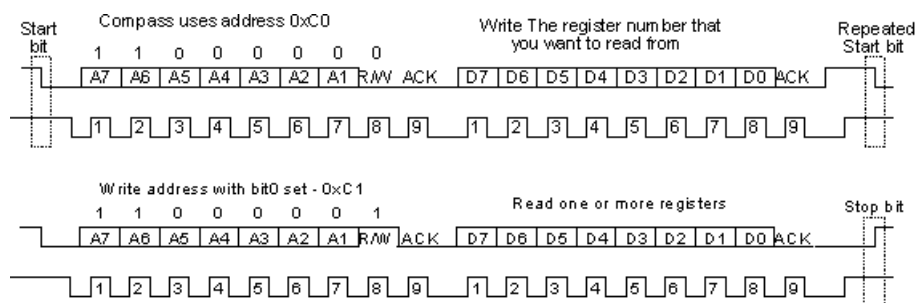


Abbildung 6.4: einfacher Lesezugriff auf einem I²C-Bus [Eled]

¹⁶Ein Frame ist ein digitales Datenpaket, welches Dateninformation beinhaltet. Typischerweise besteht ein Frame aus 8-10 Bits. Folglich werden Daten in mehrere kleine, zusammenhängende Frames aufgeteilt, welche zusammengesetzt wieder die Ursprungsinformation darstellen.

Konkret ergibt sich für die Übertragung von sechs Sensorwerten im Worst-Case-Fall folgende Anzahl von Bits (s. Abbildung 6.5) [Eled]:

$$8 \cdot I^2C \text{ Frames} \cdot 9 \frac{\text{Bits}}{I^2C \text{ Frame}} \cdot 6 = 432 \text{ Bits} \quad (6.2)$$

Bei einer Busfrequenz von 400 kHz errechnet sich die Gesamtdauer wie folgt:

$$432 \text{ Bits} \cdot \frac{1}{400 \text{ kHz}} = 1.08 \text{ ms} \quad (6.3)$$

Theoretisch könnte man mit einer Frequenz von ca. 1 kHz die Gyroskope und den Beschleunigungssensor auslesen.

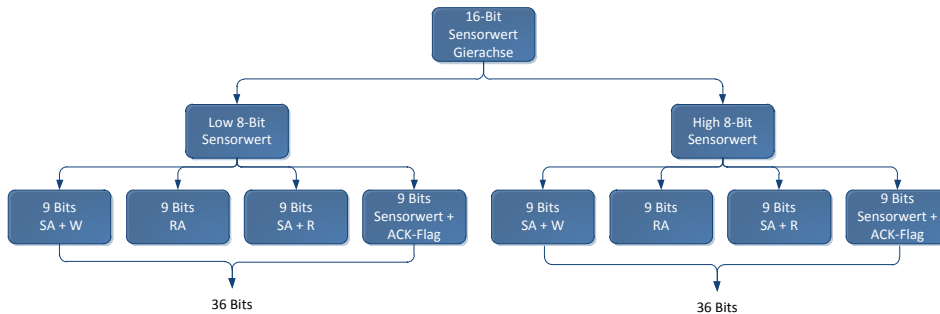


Abbildung 6.5: Segmentierung eines 16-Bit Sensorwertes für einen Lesezugriff beim I²C-Protokoll

Doch die Übertragung lässt sich mit Hilfe des Burst-Modus¹⁷ optimieren. Dadurch entfällt bei der Akquirierung des zweiten Datenpakets das Versenden der I²C-Frames für die Adressierung. Folglich werden anstatt acht nur noch fünf I²C-Frames benötigt:

$$5 \cdot I^2C \text{ Frames} \cdot 9 \frac{\text{Bits}}{I^2C \text{ Frame}} \cdot 6 = 270 \text{ Bits} \quad (6.4)$$

Folglich ergibt sich folgende theoretische Abtastfrequenz:

$$270 \text{ Bits} \cdot \frac{1}{400 \text{ kHz}} = 0.675 \text{ ms} \approx 1.5 \text{ kHz} \quad (6.5)$$

Unter Berücksichtigung von möglichen Pausen zwischen den einzelnen Übertragungen sowie des Prozesses für das Auslesen der Hallsensoren und des Barometers ist die Akquirierung der beiden Sensoren mit einer Periode von 2 ms als realistisch zu betrachten. Für vertiefende Informationen über die Grundlagen des I²C-Busses siehe [Eled].

¹⁷Der Burst-Modus erlaubt es, bei gleicher Slave- und Registeradresse mehrere Datenpakete direkt hintereinander über den gleichen Bus zu verschicken, ohne erneut die Adressen verschicken zu müssen (vgl. [Eled]).



Barometer und Hallsensor auslesen Das Auslesen des Barometers für den aktuellen Luftdruck und der daraus resultierenden Flughöhe (s. FM011) sowie das Auslesen der Hallsensoren als Kompass für die Orientierung (s. FM010) geschehen hier ebenfalls über einen I²C-Bus. Da es sich hierbei häufig (im Worst-Case-Fall) um jeweils ein bis drei 16-Bit Sensorwerten handelt, können diese theoretisch mit einer schnelleren Rate als 1 kHz abgefragt werden. Da diese Messwerte jedoch nicht so häufig auf dem Chip aktualisiert werden, wird eine größere Periode in Betracht gezogen. Häufig befinden sich diese im Intervall von 5 ms – 25 ms (vgl. [Pul11]; [Sch10], S. 68; [Bos09], S. 10) ab. Um eine präzise Auswertung der Sensoren mit möglichst wenig Rauschen zu erhalten, wird die Auflösung mit der geringsten Rauschentwicklung vorausgesetzt, sodass sich z. B. beim BMP085-Sensor eine Aktualisierung alle 25.5 ms ergibt [Bos09]. Unter Berücksichtigung des Protokoll-Overheads und der Pausen wird für das Auslesen des Barometers eine Periode von 30 ms definiert.

Durch Abwägung diverser Anwendungsbeispiele (vgl. [Pul11]; [Sch10], S. 68) ergibt sich für einen Hallsensor eine Periode von 10 ms.

Um die Ausführungszeit für das Auslesen des Luftdrucks abschätzen zu können, muss ein 16-Bit Wert ausgelesen werden. Verwendet man die Formel 6.4 für nur ein 16-Bit Wert, ergibt sich bei einer Taktrate von 400 kHz des I²C-Busses eine Dauer von gerundet 200 μ s. Analog dazu lässt sich die Ausführungsdauer für das Auslesen des Hallsensors ermitteln. Dieser besteht i.d.R. aus drei 16-Bit Werten. Unter Anwendung der Formel 6.4 für drei 16-Bit Werte, ergibt sich bei einer Taktrate von 400 kHz des I²C-Busses eine Dauer von gerundet 400 μ s.

Berechnung der Lage und Höhe Im Anschluss des Auslesens der Sensorwerte müssen aus diesen erhaltenen Rohdaten die aktuelle Flughöhe sowie -lage berechnet werden. Diese Aufgabe muss jedes Mal nach der Sensordatenakquirierung durchgeführt werden. Demnach wird diese Task mit der gleichen Periode wie die vorherige Aufgabe (2 ms) durchgeführt. Über die benötigte Ausführungszeit lässt sich in dieser Phase noch keine Aussage treffen.

Regelung Die Definition der Periode der Regelung (s. FM007) beruht auf Erfahrungen und Empfehlungen aus der Praxis, welche eine Regelfrequenz zwischen 125 Hz und 500 Hz voraussetzen (vgl. [Sch10], S. 68). Für dieses Projekt wird durch Abwägung der verschiedenen Werte eine Periode von 2 ms angenommen. Die Ausführungsdauer der Regelung kann in dieser Phase noch nicht abgeschätzt werden. Die Implementierung des Algorithmus' für die Regelung in der späteren Phase ist dafür entscheidend.



Motorstellwerte verschicken Da sich neue Stellwerte der einzelnen Motoren aus der Regelung ergeben, ist es sinnvoll, im Anschluss jeden Durchlaufs der Regelung die Motorstellwerte an die jeweiligen Motoren zu versenden (s. FM009). Folglich besitzt dieser Prozess die gleiche Periode wie die Regelung¹⁸.

Um eine erste Abschätzung über die Ausführungsdauer dieser Aufgabe treffen zu können, muss die Zeitdauer für das Versenden von jeweils einem Byte an n Motoren berechnet werden. Dabei sind für einen Schreibvorgang von 8 Bit laut [Eled] drei I²C-Frames vonnöten. Das Versenden von einem Motorstellwert als 16 Bit-Wert dauert folglich ca. 0.09 ms (s. Formel 6.6).

$$\frac{4 \cdot I^2C \text{ Frames} \cdot 9 \frac{\text{Bits}}{I^2C \text{ Frame}} \cdot n}{400k\text{Hz}} = 0.09\text{ms} \cdot n \quad (6.6)$$

Da die Motoren über einen separaten I²C-Bus angeschlossen sind, kann dieser Prozess nebenläufig durchgeführt werden. Dadurch können direkt nach der Regelung neue Sensorwerte und Führungsgrößen der Funksteuerung ausgelesen werden.

6.8.1.2 Steuerbefehle der Funksteuerung empfangen

Anhand vergangener Forschungsthemen ([Sch10], S. 68) und durchgeführter Messungen genügt es, alle 20 bis 30 ms abzufragen, ob neue Steuerbefehle durch den Piloten vorhanden sind (vgl. FM003). Speziell für dieses Projekt wurde durch Abwägung und eigenen Messungen eine feste Periode von 20 ms für diese Aufgabe definiert.

6.8.1.3 Akkustand überwachen und Telemetriedaten an die Funksteuerung senden

Da bei diesen beiden Prozessen wichtige Informationen ermittelt und an den Piloten verschickt werden (s. FM014 und FF004), welche jedoch durch den Piloten erst einmal wahrgenommen werden müssen, reicht eine sekundliche Aktualisierung der Daten auf der Funksteuerung vollkommen aus. Eine höhere Aktualisierungsrate würde das System unnötig mehr belasten und ein Mensch würde aufgrund seiner begrenzten Reaktionszeit diese Aktualisierungen gar nicht wahrnehmen.

¹⁸Theoretisch ist es auch möglich, die Periode der Regelung kürzer als die des Versendens der Motorstellwerte durchzuführen, sodass sich der Regler nachträglich einschwingen kann (Aufgabe des I-Anteils). Dieser Gedanke wird hier nicht weiter verfolgt.



6.8.2 Erste Evaluierung der Echtzeitanforderungserhebung für das ASG

In Tabelle 6.9 sind alle betroffenen Prozesse mit einem Echtzeitaspekt und deren ermittelten Perioden T_i zusammengefasst aufgelistet. Die Werte für die Ausführungszeiten c_i sind in dieser Phase des Projekts, sofern dies möglich ist, entweder gemessen (vgl. Kapitel 7.3.7) oder geschätzt.

Tabelle 6.9: Zusammenfassung der Zeitabschätzungen vom dem ASG

Nr.	Bezeichnung	T_i	c_i	Ref.
1	Autonome Höhen- und Lageregelung			
1.1	Gyroskope auslesen	2 ms	1 ms	FM010
1.2	Beschleunigungssensor auslesen			
1.3	Barometer auslesen	30 ms	0.2 ms	FM011
1.4	Hallsensor/Kompass auslesen	10 ms	0.4 ms	FM010
1.5	Berechnung der Lage und Flughöhe	2 ms	-	FM010, FM011
1.6	Regelung	2 ms	?	FM007, FM008
1.7	Motorstellwerte verschicken (an n Motoren)	2 ms	$0.09ms \cdot n$	FM009
2	Steuerbefehle von Funksteuerung empfangen	20 ms	-	FM003
3	Akkustand überwachen	1 s	-	FM014
4	Telemetriedaten an Funksteuerung senden	1 s	-	FF004

Da das ASG mit einer Regelfrequenz von 500 Hz arbeiten soll, ist die Periode der Regelung gleichzeitig die Deadline des ASGs. Daraus folgt, dass die Gesamtperiode T_{ges} der Höhen- und Lageregelung gleich der Periode T_{16} der Nr. 1.6 aus Tabelle 6.9 ist.

Es ergibt sich für T_{ges} eine Zeit von $T_{ges} = T_{16} = 2ms$. Eingesetzt in die Formel 6.1 lässt sich die Bedingung für die Ausführungszeiten c_i auf

$$c_{11} + c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17} + x < 2ms \quad (6.7)$$

verfeinern. Die maximale Ausführungszeit c_{ges} für die Höhen- und Lageregelung darf die Gesamtperiode des Systems von 2 ms nicht überschreiten¹⁹. Der Wert x entspricht einer durchschnittlichen Ausführungsdauer für die Tasks Nr. 2 - 4 (s. Tabelle 6.9), die im Worst-Case-Fall ebenfalls zu berücksichtigen ist. Das bedeutet, dass die Ausführungszeiten c_{11}

¹⁹Die maximale Ausführungszeit hängt streng genommen auch noch von dem verwendeten Scheduling-Algorithmus sowie weiteren Overhead wie z. B. Interrupts ab. Für diese Berechnung wird dieser Gedanke vernachlässigt.



bis c_{17} die zeitliche Grenze von 2 ms nicht zu 100% ausnutzen dürfen. Ein konkreter Wert für x lässt sich aber in dieser Phase noch nicht ermitteln.

Abgeleitet aus Formel 6.7 ergibt sich für die maximale Ausführungszeit der Regelung folgende Bedingung:

$$c_{16} < 2ms - c_{11} - c_{12} - c_{13} - c_{14} - c_{15} - c_{17} - x \quad (6.8)$$

Durch Aufsummierung der Ausführungszeiten fällt auf, dass sich eine Gesamtperiode von 2 ms durch einen reinen sequentiellen Ablauf nicht realisieren lässt. Wie bereits erwähnt, kann der Prozess für das Verschicken der Motorstellwerte nebenläufig zu den anderen Aufgaben ausgeführt werden. Außerdem lässt sich das Auslesen der Sensorwerte über den I²C-Bus sowie das Aufbereiten dieser optimieren. Ein Prozess kann die neuen Werte der Sensoren anfragen, aufbereiten und in einen Speicher schreiben. Parallel dazu kann ein anderer Prozess (Regelung) diese Werte aus dem Speicher lesen, sodass die Regelung nicht auf das Anfragen der Sensorwerte warten muss.

Mit der Formel 6.8 kann in einer späteren Phase evaluiert werden, ob die von der Projektgruppe entwickelte Regelung diese Zeitbedingung einhält.

6.8.3 Erhebung der Echtzeitanforderungen für das AEVV-System

Zuerst sei zu erwähnen, dass die aufgestellten zeitlichen Einschränkungen an das AEVV-System eine weiche Echtzeit darstellen. Da es sich hierbei letzten Endes um eine Objektverfolgung und eine Videoübertragung handelt, kommt es bei einer Verletzung der Deadlines nur zu einem Qualitätsverlust der Videoübertragung.

Im Rahmen des AEVV-Systems sind folgende Anforderungen aus dem Kapitel 6 betroffen, die in den folgenden Abschnitten einer zeitlichen Verfeinerung unterzogen werden:

- FA003: Kamerabild bereitstellen,
- FA004: Objekt erkennen,
- FA006: Kamerabild versenden,
- FA009: Zentrierung des Objekts und
- NA002: Reaktionszeit der Positioniereinheit.

Die entscheidende Komponente in diesem System ist der Algorithmus für die Objekterkennung. Folglich wird für diese Komponente eine Deadline d_{aevv} bestimmt, innerhalb der das System das Objekt erkennt und im Mittelpunkt des Bildes fixiert haben muss. Verletzt die Erkennung bzw. die Zentrierung diese Deadline, kann das System nicht gewährleisten, dass das Objekt rechtzeitig verfolgt werden kann. Dies hat zur Folge, dass die Gesamtausführungszeit c_{ges} des AEVV-Systems innerhalb dieser Grenze liegen muss. Neben der Ermittlung der Deadline für die Objekterkennung (s. Kapitel 6.8.3.3) werden für die restlichen Komponenten die Ausführungszeiten c_i abgeschätzt.



In Abbildung 6.6 sind die zeitlichen Abhängigkeiten der benötigten Komponenten für die Erfüllung der Anforderungen dargestellt. Die Abbildung zeigt, dass jede Komponente eine gewisse Zeit c_i benötigt, bevor die nächste Aufgabe starten kann.

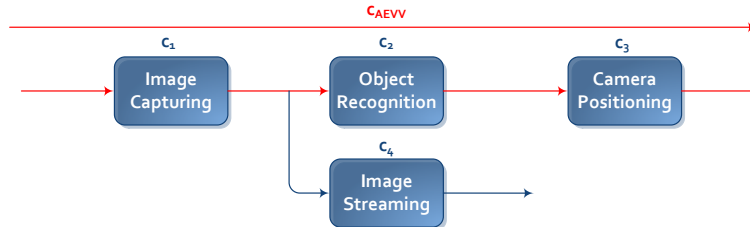


Abbildung 6.6: abstrahierter Arbeitsablauf des AEVV-Systems

Um die Startzeitpunkte der Komponenten zeitlich zu entkoppeln, kann der Prozess für das Bildverschicken parallel zur Objekterkennung gestartet werden. Dadurch muss diese nicht auf die Terminierung der Objekterkennung warten und kann eine bessere Übertragungsrate gewährleisten. Ein weiterer Vorteil der Parallelisierung ist, dass beide Komponenten mit unterschiedlichen Frameraten und Bildgrößen arbeiten können.

Der kritische Pfad (s. Abbildung 6.6, rote Pfeile) betrifft für dieses System nur die oberen drei Komponenten, da diese für die Objekterkennung und -verfolgung zuständig sind. Diese Aufgaben müssen zeitliche Einschränkungen einhalten. Die Gesamtausführungsdauer des kritischen Pfads beträgt c_{aeuv} .

Das Übertragen der Bilder verläuft nebenläufig und ist für die Objekterkennung irrelevant.

6.8.3.1 Image Capturing

Die Bereitstellung des Kamerabildes (s. FA003) ist in erster Linie abhängig von der Framerate f_{fps} ²⁰ der Kamera. Außerdem beeinflusst die Übertragungsrate f_U des Übertragungsmediums, über das die Kamera mit dem System angeschlossen ist, sowie die Speichergröße eines Frames d_B die Bereitstellung des Kamerabildes. Damit bei gegebener Framerate ein Bild rechtzeitig zur Verfügung gestellt wird, muss folgende Bedingung erfüllt sein [NKLL10]:

$$c_{IC} = \frac{d_B}{f_U} \wedge c_{IC} < \frac{1}{f_{fps}} \quad (6.9)$$

Die Zeit, die das System benötigt, um das Bild zur Verfügung zu stellen, wird durch c_{IC} dargestellt.

²⁰Frames Per Second (fps) ist eine Einheit, die angibt, wie viele Einzelbilder pro Sekunde die Kamera bereitgestellt. Für ein ruckelfreies Video sollte die Framerate mindestens 24 fps betragen.

6.8.3.2 Image Streaming

Die Videoübertragung kann die Bilder direkt von der vorherigen Komponente abgreifen und sie somit mit der gleichen Übertragungsrate f_U zur Station versenden. Demnach richtet sich die Übertragungsrate f_T zur Telemetriestation nach der Framerate f_{fps} bzw. f_U .

6.8.3.3 Object Recognition

Die Deadline für die Objektdetektion ist genau die zeitliche Grenze, bis zu der das AEVV-System die Aufgaben Ballerkennung und -zentrierung erledigt haben muss. Ansonsten kann nicht gewährleistet werden, dass der Ball rechtzeitig im Bild zentriert wird. Die Zeit, die das System benötigt, um den Ball zu erkennen, wird im Folgenden mit c_{Algo} deklariert. Um eine Aussage mit möglichst wenigen Abhängigkeiten über die zeitlichen Einschränkungen für die Objekterkennung (s. FA004) zu treffen, wird ein vereinfachtes Modell definiert (s. Abbildung 6.7) [FBE⁺13]. Für dieses Modell gelten folgende Annahmen:

- Annahme 1: Der Ball wird zum ersten Mal am Rand des Bildbereichs erkannt. Das bedeutet, dass der Ball sich bis zu dem jeweiligen Zeitpunkt noch nicht im Bildwinkel der Kamera befindet.
- Annahme 2: Die Bewegungsrichtung des Balls ist linear und seine Geschwindigkeit ist konstant.
- Annahme 3: Die Ballgeschwindigkeit ist kleiner als die maximale Bewegungsgeschwindigkeit der Kamerapositioniereinheit.
- Annahme 4: Der Ball bewegt sich stets parallel zur Kamera, sodass er sich nicht auf die Kamera zubewegt.

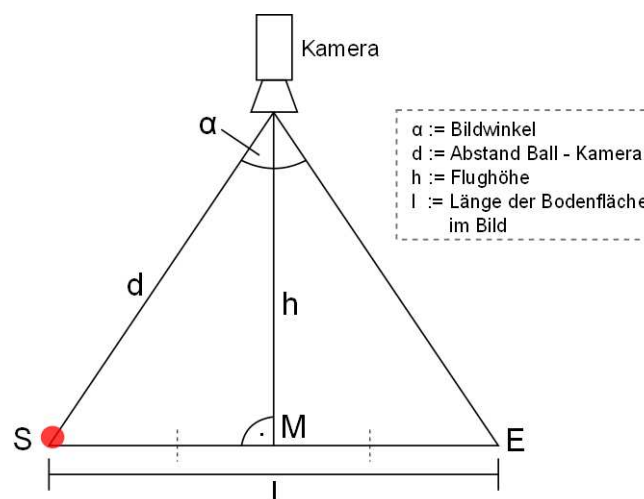


Abbildung 6.7: Zusammenhang zwischen Bildwinkel der Kamera und des Abstands zum Objekt (roter Kreis)



Tritt der Ball zum ersten Mal am Punkt S in den Bildwinkel α der Kamera auf (s. Abbildung 6.7), hat er eine Distanz d zur Kamera. Die Frist für die Erkennung des Balls lässt sich auf zwei unterschiedlichen Stufen durchführen:

1. Fall: Innerhalb der gesamten Strecke l muss der Ball mindestens einmal erkannt werden.
2. Fall: Innerhalb der halben Strecke l (bis zum Punkt M) muss der Ball mindestens einmal erkannt werden.

Der Fall 2 hat den Vorteil, dass sich der Ball früher im Mittelpunkt der Kamera befinden kann. Dies wirkt sich zu Lasten der Deadline aus, welche dadurch verkleinert wird. Fall 1 beschreibt den Worst-Case-Fall für die Erkennung des Objekts. Wird der Ball nicht innerhalb dieser Strecke erkannt, verschwindet der Ball aus dem Bildwinkels, ohne ihn auch nur einmal detektiert zu haben.

Als Ausblick kann in Betracht gezogen werden, die Kamera nur innerhalb der gestrichelten Linie in Abbildung 6.7 zu bewegen, sodass das aufgenommene Video „ruckelfreier“ wird. Dieser Gedanke wird in dieser Arbeit aber nicht weiterverfolgt.

Die Bestimmung der Deadline lässt sich u. a. in Abhängigkeit der Höhe herleiten. In dieser Arbeit soll die Bestimmung unabhängig von der Höhe durchgeführt werden, da sie für die Evaluation keine Rolle spielt. Dafür wird die Annahme getroffen, dass das aufgespannte Dreieck der Kamera ein gleichschenkliges ist. Daraus folgt für die maximale Strecke l im Fall 1 folgende Formel:

$$l_1 = \sqrt{2 \cdot d^2 - 2d^2 \cdot \cos \alpha} = d \cdot \sqrt{2 \cdot (1 - \cos \alpha)} \quad (6.10)$$

Die Gleichung für den Fall 2 lässt sich aus der Formel 6.10 herleiten (s. Formel 6.11).

$$l_2 = \frac{l_1}{2} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{2} \quad (6.11)$$

Wenn der Ball die Geschwindigkeit v_B aufbringt, ergibt sich folgende Deadline d_{obj1} im Fall 1 [NKLL10]:

$$d_{obj1} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{v_B} \quad (6.12)$$

Analog zur Formel 6.12 ergibt sich für den Fall 2 die Deadline d_{obj2} wie folgt:

$$d_{obj2} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{2 \cdot v_B} \quad (6.13)$$

Als Erweiterung des Modells soll die Geschwindigkeit v_K des Multikopters berücksichtigt werden. Diese kann die Deadline verkürzen oder verlängern. Folgen beide Objekte der gleichen Richtung, verlängert sich die Zeit, in der das Objekt erkannt werden kann. Durch



Überlagerung der Geschwindigkeiten verkleinert sich somit die resultierende Gesamtgeschwindigkeit v_{ges} (s. Formel 6.14).

$$v_{ges} = v_B - v_K \quad (6.14)$$

Verwendet man das Ergebnis der Formel 6.14 als Betrag²¹ und setzt es in die Gleichungen 6.12 und 6.13 ein, erhält man die Formeln zur Bestimmung der Deadline für die Objekterkennung und -verfolgung (s. Formel 6.15 und 6.16).

$$d_{obj1} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{|v_B - v_K|} \quad (6.15)$$

$$d_{obj2} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{2 \cdot |v_B - v_K|} \quad (6.16)$$

Die Deadline d_{obj1} bzw. d_{obj2} ist genau die Zeit, innerhalb der das AEVV-System ihre gesamten Aufgaben (den Ball auf dem Bild erkennen und den Ball im Bild zentrieren) erledigt haben muss. Im Folgenden wird d_{obj1} durch d_{aevv} ersetzt.

6.8.3.4 Camera Positioning

Die letzte Aufgabe des AEVV-Systems aus Abbildung 6.6 ist die Positionierung der Kamera, sodass der Ball im Mittelpunkt des Bildes bleibt (s. FA009 und NA002). Die dafür benötigte Positioniereinheit besitzt eine gewisse Reaktionszeit t_{verz} und die Motoren haben eine Winkelgeschwindigkeit ω . Diese Zeit muss bei der Einhaltung der Deadline für die Objekterkennung und -verfolgung berücksichtigt werden.

In Abbildung 6.8 ist die Winkeländerung $\frac{\alpha}{2}$ für die Kamerapositioniereinheit eingezeichnet (blaues Dreieck). Demnach muss im Worst-Case-Fall der Fokus der Kamera (Punkt M) zum Punkt S wandern. Bei gegebener Winkelgeschwindigkeit ω und der Reaktionszeit t_{verz} ergibt sich eine maximale Zeitdauer von c_{Gimbal} (s. Formel 6.17).

$$c_{Gimbal} = \frac{\alpha}{2 \cdot \omega} + t_{verz} \quad (6.17)$$

²¹Für die Objekterkennung ist es irrelevant, ob sich der Multikopter oder der Ball schneller bewegt. Der Betrag der resultierenden Geschwindigkeit ist entscheidend für die zeitliche Bestimmung der Deadline.

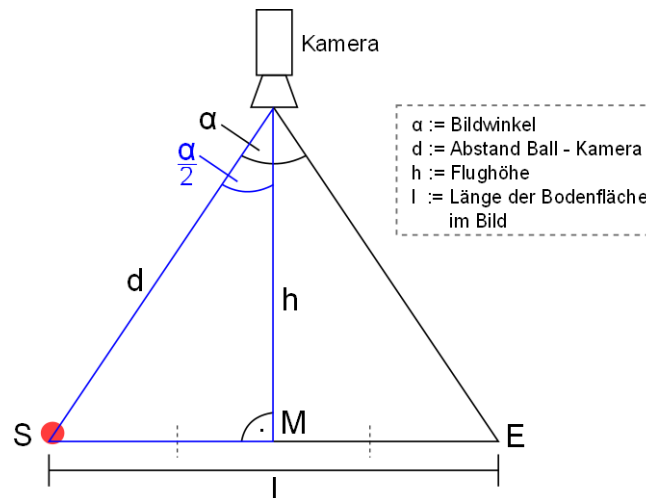


Abbildung 6.8: Zusammenhang zwischen Abstand d des Objekts und der Winkeländerung $\frac{\alpha}{2}$ der Positioniereinheit (blaues Dreieck)

6.8.4 Erste Evaluierung der Echtzeitanforderungserhebung für das AEVV-System

Abgesehen von „Image Streaming“, dessen Zeit sich nach der Bereitstellung der Bilder richtet, sind in Tabelle 6.10 die ermittelten Zeiten und Deadlines noch einmal zusammengefasst. Dabei ist die Dauer für den Algorithmus der Objekterkennung noch die Unbekannte, die im späteren Verlauf des Projektes zu ermitteln ist.

Tabelle 6.10: Zusammenfassung der zeitlichen Constraints des AEVV-Systems

Nr.	Bezeichnung	c_i	d_i	Ref.
1	Image Capturing	$c_{ic} = \frac{d_B}{f_U}$	$d_{aevv} = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{ v_B - v_K }$	FA003
2	Object Recognition	$c_{Algo} = ?$		FA004
3	Camera Positioning	$c_{Gimbal} = \frac{\alpha}{2 \cdot \omega} + t_{verz}$		FA009, NA002



Um die gesamte Ausführungsdauer für das System AEVV zu verdeutlichen, wird noch einmal die Abbildung 6.6 aufgegriffen und mit den ermittelten zeitlichen Einschränkungen verfeinert (s. Abbildung 6.9).

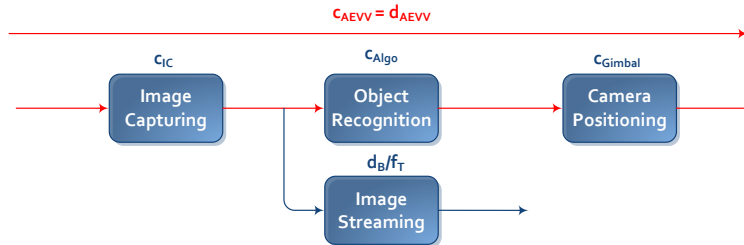


Abbildung 6.9: abstrahierter Arbeitsablauf des AEVV-Systems

Aus Abbildung 6.9 und der ermittelten Deadline d_{aeuv} lässt sich für die Gesamtausführungsdauer c_{aeuv} schlussfolgern, dass $c_{aeuv} < d_{aeuv}$ gelten muss. Ersetzt man diese Ungleichung durch die einzelnen Ausführungszeiten ergibt sich folgende Formel:

$$c_{ic} + c_{Algo} + c_{Gimbal} < d_{aeuv} \quad (6.18)$$

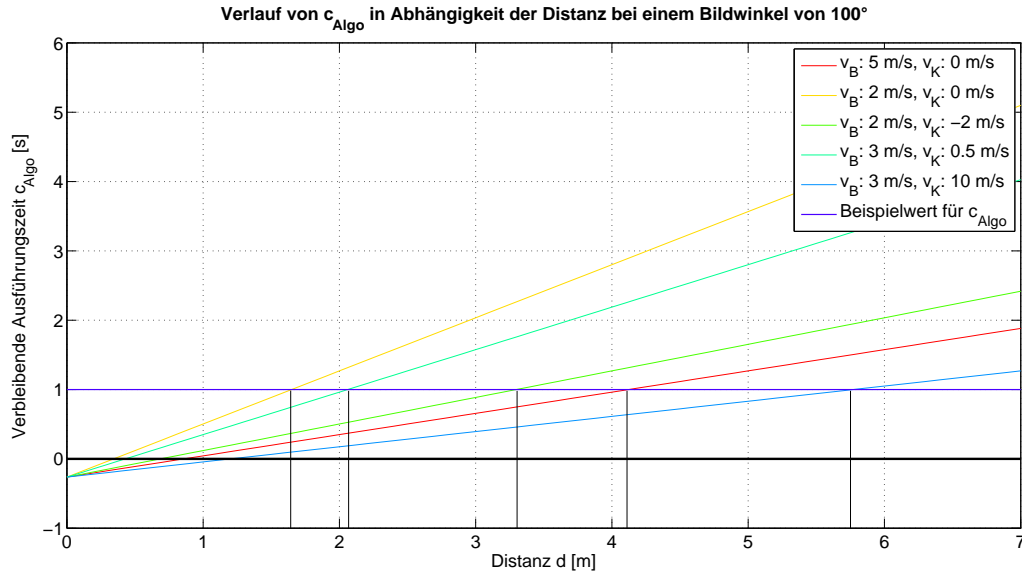
Da die Einhaltung der Deadline stark von der Implementierung des Algorithmus' für die Objekterkennung abhängt, soll die Evaluierung der Echtzeitanalyse mit Hilfe der Bedingung 10.7 durchgeführt werden.

$$c_{Algo} < d_{aeuv} - c_{ic} - c_{Gimbal} \quad (6.19)$$

In Situationen, die dem aufgestellten Modell entsprechen und die dessen Annahmen einhalten, kann bei Erfüllung der Formel 10.7 vom AEVV-System gewährleistet werden, dass das Objekt rechtzeitig erkannt und fokussiert wird.

Ausführungszeit c_{Algo} als Funktion in Abhängigkeit von der Distanz

Betrachtet man mit Hilfe der Formel 10.7 die benötigte Zeit für den Algorithmus als Funktion in Abhängigkeit der Distanz d , lassen sich Aussagen über Mindestdistanz zum Ball festhalten.

Abbildung 6.10: Verlauf von c_{Algo} in Abhängigkeit der Distanz beim Bildwinkel von 100°

Die Abbildung 6.10 zeigt eine Auswertung der Funktion

$$c_{Algo}(d) = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{|v_B - v_K|} - \frac{d_B}{f_U} - \frac{\alpha}{2 \cdot \omega} - t_{verz} \quad (6.20)$$

bei unterschiedlichen Geschwindigkeiten für den Ball (v_B) und den Multikopter (v_K) mit Hilfe von folgenden fiktiven Beispielparametern für

- die Winkelgeschwindigkeit $\omega = \frac{2000^\circ}{s}$,
- die Reaktionszeit $t_{verz} = 0.06$ s,
- die Übertragung f_U über USB 2.0 mit ca. 28 MB/s bei einer Bildgröße d_B von 5 MB,
- den Blickwinkel $\alpha = 100^\circ$ und
- der Ausführungszeit $c_{Algo} = 1$ s.

Grundsätzlich lässt sich aussagen, dass die Mindestdistanz d steigen muss, sobald die Differenzgeschwindigkeit $|v_B - v_K|$ steigt. Diese Schlussfolgerung deckt sich mit der Überlegung, dass bei gleicher Distanz aber schnelleren Geschwindigkeit weniger Zeit zur Verfügung steht, um den Ball zu detektieren. Bei einer Ballgeschwindigkeit v_B von $3 \frac{m}{s}$ und einer Multikoptergeschwindigkeit v_K von $0.5 \frac{m}{s}$ muss der Multikopter eine Mindestdistanz von ca. 3.5 m zum Ball haben, um eine rechtzeitige Ballverfolgung zu gewährleisten.

6.9 Anforderungen an die Ressourcen

Durch die Spezifikation der (nicht-) funktionalen Anforderungen entstehen gewisse Erwartungen an die benötigten Ressourcen. In diesem Abschnitt soll eine erste Worst-Case-Abschätzung über die benötigte Rechenleistung, über den Speicherbedarf sowie über die Buszugriffe getroffen werden.

Vorab lässt sich festhalten, dass die digitale Bildverarbeitung aufgrund der Bildgröße und der komplexeren Algorithmen den größten Einfluss auf diese Anforderungen hat.

6.9.1 Benötigte Rechenleistung

In der Praxis erwiesen sich Controller wie z. B. der 32Bit ARM9-Prozessor der Firma *STMicroelectronics* mit 96MHz (vgl. [Sch10], S. 7f) oder der *Atmel AT89C5131* USB-Controller (vgl. [Pul05], S. 66) als völlig ausreichend für die Flugregelung.

Die Rechenleistung für eine digitale Bildverarbeitung ist abhängig von der Framerate der Kamera, von der Auflösung der Bilder sowie von den verwendeten Algorithmen. Typischerweise beträgt die Framerate 30 fps, sodass alle 33 ms ein neues Bild zur Verfügung steht. Innerhalb dieser Zeitspanne muss die Bildverarbeitung durchgeführt werden.

Laut Anforderung NA007 beträgt die minimale Bildauflösung 720×540 Pixel, sodass innerhalb des Zeitraums von 33 ms 388.800 Pixel analysiert werden müssen. Als erster Anhaltspunkt resultiert daraus, dass der Prozessor für die Verarbeitung eines Pixels maximal nur ca. 84 ns benötigen darf. Zu dieser Zeitdauer müssen mögliche Pausen, Kommunikationsoverhead, technische Einschränkungen wie Übertragungsraten sowie Verzögerungszeiten der Kamerapositioniereinheit mit einkalkuliert werden.

6.9.2 Benötigter Speicherbedarf

Laut Anforderung NA007 beträgt die Bildauflösung 720×540 Pixel, sodass bei einer Farbtiefe von 24 Bit eine Speichergröße von ca. 9 MB unkomprimiert pro Bild entsteht. Da ggf. jeweils ein Bild für die Objekterkennung und eins für den Livestream benötigt wird (s. Kapitel 6.8.3), summiert sich der Speicherbedarf für die Bilder auf 18 MB. Für den Worst-Case-Fall, dass die Bilder einmal zwischengespeichert werden müssen, ergibt sich ein Gesamtspeicherbedarf von 36 MB. Darüber hinaus wird noch ein dynamischer Speicher für die Algorithmen benötigt, welcher jedoch in diesem Stadium des Projekts noch nicht ermittelt werden kann.

Für die Avionik müssen die aktuellen Sensorwerte zwischengespeichert werden. Diese Werte werden i. d. R. in einem 16-Bit Integer abgelegt, sodass bei den Telemetriedaten Akkustand, drei Winkel, Flughöhe, Motorstellwerte eine Speichergröße von 96 Bits nötig ist. Da unter Umständen eine Historie der Werte benötigt wird, ergibt sich bei weiteren 20 Variablen pro Wert eine Gesamtgröße von 2 kBit. Auch für die Algorithmen der Avionik fällt ein Speicherbedarf an, welcher aber ebenfalls noch nicht ermittelt werden kann.

Der Speicherbedarf der Avionik ist gegenüber dem der Bildverarbeitung erheblich klein, sodass dieser für die weitere Abschätzung vernachlässigt wird.

6.9.3 Benötigte Bus- bzw. I/O-Zugriffe

Um mit den angeschlossenen Sensoren kommunizieren zu können (s. FM010, FM011), muss die verwendete Plattform eine Schnittstelle für einen I²C-Bus bereitstellen. Des Weiteren wird ein SPI-Bus zur Spannungsüberwachung der Akkumulatoren benötigt (s. FM014). Ein weiterer I²C-Bus wird für das Verschicken der Motorstellwerte gefordert (s. FM009). Wie in Anforderung FM013 dargestellt, muss die Plattform einen gemeinsamen Bus bzw. einen Buszugriff auf einen gemeinsamen Speicher für beide Teilsysteme zur Verfügung stellen.

Um eine bidirektionale Kommunikation zwischen der Funksteuerung und dem ASG zu realisieren, wird jeweils ein PPM-Signal für die Steuerbefehle (s. FM003), ein weiterer für die Positionierung der Kamera (s. FA008) sowie eine UART-Schnittstelle als Rückkanal zur Funksteuerung (s. FF004) benötigt. Laut Anforderung FA001 muss es möglich sein, die Kamerapositioniereinheit ein- bzw. auszuschalten. Dieses lässt sich nicht über den vorgesehenen Bus für die Positioniereinheit realisieren. Folglich wird dafür ein Output-Pin benötigt.

Um eine Kamera an die Plattform anzuschließen, wird typischerweise eine Universal Serial Bus (USB)-Schnittstelle benötigt. Des Weiteren wird ein Funkmodul für das Versenden der Telemetriedaten benötigt, welcher häufig über die USB-Schnittstelle angeschlossen wird. Insgesamt wird folglich ein USB-Host mit zwei USB-Ports vorausgesetzt.



7 Architektur und Entwurf

7.1 Mixed-Critical Systems

Viele Avionik Systeme sind ein sogenanntes Cyber-Physical System (CPS). Solche Systeme sind immer den physikalischen Gesetzen unterworfen (physical) und benötigen somit auch eine berechnende Einheit (cyber), welche das System steuert [BBB⁺09]. Ein Fehler in einem CPS kann zu schwerwiegenden Folgen für die Umwelt des Systems führen, wie z. B. Verletzungen am Menschen. Auch ein Quadrocopter ist ein CPS und kann bei unsachgemäßer Regelung schwere Verletzungen oder Sachschäden verursachen. Deswegen ist es für dieses Projekt wichtig, eine saubere Trennung vom sicherheitskritischen (ASG) und vom nicht sicherheitskritischen Teil (AEVV-System) zu gewährleisten.

Ein solches System, welches sicherheitskritische und nicht sicherheitskritische Komponenten auf derselben Plattform vereint, wird als Mixed-Critical System (MCS) bezeichnet. Im folgenden sei dazu eine Definition gegeben [BBB⁺09]:

A mixed-critical system is an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform.

Unter dem Begriff *safety-critical* kann im Kontext von Avionik-Systemen z. B. die Motorregelung verstanden werden. *Mission-critical* bezeichnet Funktionen wie die Navigation oder Kommunikation des Fluggerätes zur Bodenstation. Des Weiteren repräsentiert *non-critical* Anwendungen, die weder für die Flug- noch für die Missionssicherheit von Bedeutung sind. Im Kontext der Projektgruppe wird die Motorregelung und die Kommunikation mit der Fernsteuerung als *safety-critical* bzw. als sicherheitskritisch eingestuft, wohingegen die Objekterkennung als *non-critical* bzw. als nicht sicherheitskritisch eingeschätzt wird. Von Mixed-Criticality wird meistens in einer der folgenden Situationen gesprochen, wobei sich diese nicht zwingend gegenseitig ausschließen [EC112]:

1. Die Mischung von Realzeitanforderungen mit harten und weichen Deadlines²² auf einer Plattform.
2. Die Mischung von Applikationen mit unterschiedlichen Kritikalitätsleveln.

In dem vorliegenden Anwendungsfall dieser Projektgruppe treten beide Situationen auf.

7.1.1 Aktuelle Forschung

Für MCS gibt es diverse und vielfältige Anwendungsgebiete (z. B. Automobilindustrie, Luftfahrtindustrie, Medizin, Verkehrskontrolle, usw.) und dank der beschriebenen Vorteile

²²Das Verpassen einer harten Deadline hat katastrophale Auswirkungen. Beim Verletzen einer weichen Deadline würden nur Qualitätsverluste (z. B. Jitter) auftreten.



solcher Systeme ist dieses Thema auch in vielen Projekten Forschungsgegenstand. Davon werden im Folgenden exemplarisch einige kurz vorgestellt:

CONTREX Das CONTREX-Projekt startete erst im Oktober 2013 und untersucht das Design von eingebetteten MCS unter Beachtung von extrafunktionalen Eigenschaften wie Realzeitfähigkeit, Power, Temperatur und Zuverlässigkeit [TOG⁺].

CERTAINTY Dieses Projekt beschäftigt sich mit dem Entwurf von Methodiken und Tools für das Design, die Validierung und Synthese von MCS [EC112].

RECOMP Hier wird an Methoden, Tools und Plattformen geforscht, die eine kosteneffiziente Zertifizierung von MCS ermöglichen [EC112].

DREAMS Dieses Projekt startete ebenfalls im Oktober 2013 und beschäftigt sich mit der Entwicklung einer europäischen Referenzarchitektur für MCS [TOG⁺].

Alle diese Projekte werden von der EU gefördert, was deren Bedeutsamkeit unterstreicht. Eine Plattform, welche für gemischt kritische Systeme gut geeignet ist, ist die CompSOC Plattform. Sie bietet eine virtuelle Ausführungsplattform für eine Applikation, um ein unabhängiges Design und Verifikation zu gewährleisten. Der CompSOC-Design-Flow errechnet automatisch ein Mapping von Benutzern (steht in diesem Kontext für: Berechnung, Kommunikation) auf Ressourcen (z. B. Prozessor) und das zugehörige Scheduling-Budget (prozentualer Anteil) für jeden Benutzer auf der betrachteten Ressource [GAC⁺].

7.1.2 Herausforderungen

Kritikalität lässt sich, angelehnt an die oben definierten zwei Situationen, im Groben in zwei Domänen betrachten: Zeit und Sicherheit. Das Kritikalitätslevel einer Applikation oder eines Systems steigt auch mit steigenden Anforderungen in beiden Bereichen. Dazu siehe Abbildung 7.1. Applikationen, die sowohl viele Funktionen mit Sicherheitsanforderungen als auch zeitkritische Funktionen aufweisen, haben das höchste Kritikalitätslevel (obere rechte Ecke der Abbildung 7.1). Dazu gehört auch der Avionik Bereich. Das Kritikalitätslevel der sicherheitskritischen Anwendung in diesem Projekt ist demnach sehr hoch, da einerseits einige Anforderungen sicherheitskritisch sind (siehe Kapitel 6.7) und andererseits mehrere Funktionen existieren, die in einer bestimmten Zeitspanne Berechnungen durchgeführt haben und plausible Werte liefern müssen (siehe Motorregelung).

Beim Entwurf eines MCS geht es darum, im Konflikt stehende Anforderungen gegeneinander abzuwägen. Auf der einen Seite führt eine starke Partitionierung des Systems zu erhöhter Sicherheit, da sicherheitskritische Funktionen z. B. ihren eigenen Prozessor oder Speicher zur Verfügung haben. Auf der anderen Seite würde ein geteilter Ressourcenzugriff zwischen den Anwendungen mit unterschiedlichen Kritikalitätsleveln zu einer höheren Performanz führen, da die Daten der anderen Applikation direkt bereitgestellt werden können

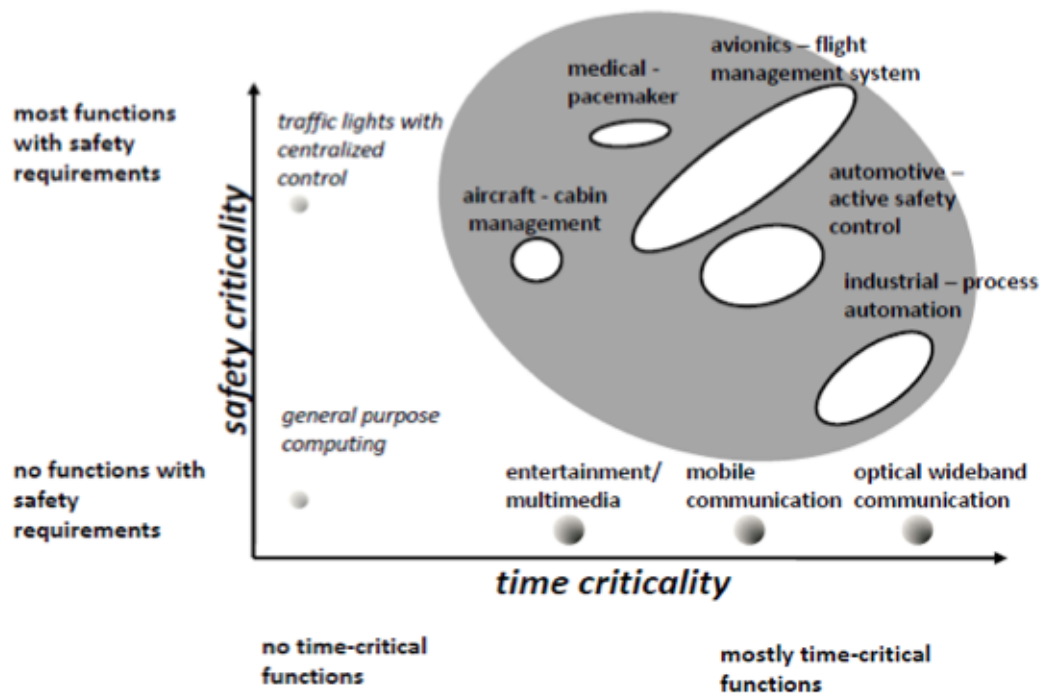


Abbildung 7.1: Sicherheitskritische und zeitkritische Applikationen [EC112]

[BD14]. Dabei ist allerdings zu beachten, dass nicht beabsichtigte Zugriffe auf gemeinsam genutzte Ressourcen unter allen Umständen zu vermeiden sind. Dadurch könnte die sicherheitskritische Anwendung in ihrer Funktionalität eingeschränkt und gestört werden. Eine weitere Herausforderung liegt in der Fehlertoleranz des MCS. Da es sich bei diesem System meist um ein System-on-a-Chip (SoC), folglich um einen physischen Chip, handelt, bedeutet ein Ausfall dieses Chips den Zusammenbruch des gesamten Systems. Man spricht in diesem Zusammenhang von einem Single Point Of Failure (SPOF). Redundanz wäre eine passende aber meist kostenintensive Möglichkeit, um diesen SPOF aufzulösen. In diesem Projekt wird aus zeitlichen Gründen eine redundante Implementierung nicht berücksichtigt, da dies einen zu hohen Entwicklungsaufwand bedeuten würde. Außerdem würde Redundanz ein gesteigertes Gewicht bedeuten, wodurch die Traglast des Quadropters u. U. überschritten werden könnte.

Die Zertifizierung von einem MCS ist ein wichtiges und aktuelles Thema in der Forschung. Mit diesem Themenbereich beschäftigen sich derzeit mehrere von der EU geförderte Projekte, u. a. CONTREX, CERTAINTY, MultiPARTES, DREAMS und PROXIMA [TOG⁺]. In dieser Projektgruppe beschränkt sich die Korrektheitsprüfung des Systems allerdings nur auf Komponenten-, Integrations- und Systemtests der sicherheitskritischen und nicht sicherheitskritischen Anwendung (siehe Kapitel 10.2 und Kapitel 10.3).

Abgeleitet aus den Herausforderungen sollen folgende Punkte besonders beachtet werden:



- Die sicherheitskritische Anwendung benötigt exklusiven Zugriff auf die Sensorwerte, die Daten der Fernsteuerung und die Motorregler.
- Die gemischt kritischen Tasks brauchen einen gemeinsamen Speicherbereich, der bei möglichen schreibenden Zugriffen gegenseitigen Ausschluss bieten muss.

Allgemein lässt sich feststellen, dass der Entwurf eines MCS neue Herausforderungen an die Entwickler stellt, wodurch neue Entwurfsmethodiken benötigt werden. Außerdem müssen Tools entwickelt werden, die den Entwickler insbesondere bei der Verifikation und Validierung des Systems unterstützen.

7.1.3 Extrafunktionale Eigenschaften

Dieses Kapitel liefert Hintergrundinformationen zu extrafunktionalen Eigenschaften bzgl. MCS. Diese sind von besonderer Bedeutung beim Modellieren, Simulieren und Analysieren von besagten Systemen. In dieser Projektgruppe wird das System aber nicht bzgl. der nachfolgenden Eigenschaften optimiert. Ein solches Vorgehen wäre zu umfangreich.

Eine extrafunktionale Eigenschaft ist der Energieverbrauch eines Systems [BD14]. Besonders bei mobilen Geräten wie Smartphones oder Tablets spielt dies gerade beim Entwurf eine entscheidende Rolle. Eine längere Betriebsdauer des Gerätes kann durch einen im Systementwurf berücksichtigten optimierten Energieverbrauch erzielt werden. Ein MCS hat den Vorteil, meist eine in sich geschlossene Plattform und keine zusätzlichen Recheneinheiten zu besitzen. Hierdurch kann sich der Energieverbrauch verringern. Auch die Flugdauer des in diesem Projektes verwendeten Quadrocopter profitiert somit von der eingesetzten MCS Plattform.

Auch die Temperatur oder die Hitzeentwicklung eines MCS muss bei seiner Entwicklung berücksichtigt werden [BD14]. Innerhalb des CONTREX Projektes werden insbesondere Energieverbrauch und Temperatur eines Chips auf Systemebene näher untersucht und Simulationen dazu auf einer virtuellen Plattform durchgeführt. Durch Floorplanning²³ und thermische Analysen kann der Energieverbrauch und die Hitzeentwicklung optimiert werden [Gru]. Um die dynamische Verlustleistung z. B. von einem MCS zu berechnen, wird folgende Formel benötigt [Gru]:

$$P_{dyn}(t) = \frac{1}{2} \overline{C}(t) F_{clk}(t) V_{dd}(t)^2 \quad (7.1)$$

Dabei steht $\overline{C}(t)$ für die Kapazität, $F_{clk}(t)$ für die Frequenz (Taktung) und $V_{dd}(t)$ für die Spannung über die Zeit des betrachteten Systems.

Eine für die Sicherheit des Systems wichtige Eigenschaft ist die Zuverlässigkeit. Um diese zu steigern, sollen z. B. die Signalwege minimiert werden. Doch diese Herangehensweise

²³Floorplanning ist ein Optimierungsproblem, bei dem Bauteile in einem System so angeordnet werden sollen, dass möglichst kurze Signalwege entstehen.



stell auch noch in der heutigen Zeit ein Problem dar, denn in der Automobilindustrie sind immer noch ca. 2 km an Kabeln in modernen Autos verlegt [EC112]. Ein MCS hat aufgrund seiner Kompaktheit einige Vorteile gegenüber verteilten Systemen bzgl. der Zuverlässigkeit.

Das Zeitverhalten ist eine ebenso wichtige Eigenschaft. Hierbei sind ungewollte Seiteneffekte von eigentlich separierten Funktionen zu vermeiden [EC112][TOG⁺].

Eine meist schwer zu verhindernde extrafunktionale Eigenschaft ist die Alterung. Damit ist der natürliche Verschleiß des Systems gemeint, der z. B. durch Elektromigration (Materialtransport in einem festen Leiter durch Bewegung der Ionen) entsteht [HGE⁺14]. Eine mögliche Gegenmaßnahme ist u. a. Power Gating, bei dem gerade inaktive Gatter-Blöcke abgeschaltet werden, sodass dort kein elektrischer Strom fließt und dadurch auch die Alterungseffekte reduziert werden.

Das Unterbinden von Abweichungen bei der Produktion eines Chips ist ebenfalls eine extrafunktionale Eigenschaft und führen zu Endprodukten mit teils unterschiedlichen physikalischen Eigenschaften. Man spricht hierbei von Prozessvariation. Dieser Effekt tritt insbesondere bei kleineren Strukturgrößen verstärkt auf und folglich auch bei Mehrkern Prozessoren, die meist Basis eines MCS sind.

Zusammenfassend lässt sich festhalten, dass extrafunktionale Eigenschaften insbesondere beim Entwurf von MCS wichtig sind. Die Effekte treten bei solchen Systemen meist noch stärker auf, sodass durch Nichtbeachtung die Funktionalität und auch die Lebensdauer der Systeme deutlich reduziert werden kann. Allerdings bieten MCS auch Vorteile, welche im folgenden Kapitel detaillierter erläutert werden.

7.1.4 Vorteile von MCS

Die Entwicklung von Prozessoren geht aufgrund der steigenden Hitzeentwicklung und der damit zusammenhängenden Probleme mit der Kühlung sowie den immer kleiner werdenden Strukturgrößen in Richtung Mehrkernarchitekturen bzw. Multi-Core-Prozessoren. Dieser Trend wird durch das Moore's Law unterstrichen, welches besagt, dass sich die Anzahl an Transistoren in integrierten Schaltkreisen alle 2 Jahre verdoppelt. Im Endverbraucher-Markt haben sich diese Modelle schon fast ausnahmslos durchgesetzt. Auch im Entwurf von sicherheitskritischen eingebetteten Systemen sollen mittlerweile Mehrkernarchitekturen verwendet werden, da solche Architekturen einige Vorteile bieten, die im folgenden Text vorgestellt werden. Da Mehrkernprozessoren auch Basis vieler MCS sind, so wie auch in dieser Projektgruppe, betreffen die genannten Vorteile auch diese Art von Systemen.

1. Das Gewicht und Volumen solcher Systeme sinkt im Vergleich zu äquivalenten verteilten Architekturen [Gon13].



2. Die Produktionskosten sinken [Gon13], da einerseits weniger Material benötigt wird und andererseits die Produktion im Allgemeinen vereinfacht wird, weil bei derselben Anzahl an Produktionsschritten am Ende mehrere Prozessorkerne entstanden sind.
3. Auch Redundanz im Kontext von Sicherheit kann einfacher und kosteneffektiver erreicht werden, da weniger gleiche Komponenten benötigt werden [EC112].
4. Flexible Festlegung der am besten passende Laufzeitumgebung für jeden einzelnen Kern aufgrund der Freiheitsgrade [Gon13].

7.1.5 Entwurf von MCS

Bei dem Entwurf eines MCS müssen unter allen Umständen ungewollte Interferenzen zwischen der sicherheitskritischen und der nicht sicherheitskritischen Anwendung verhindert werden. In diesem Kapitel sollen einige Herangehensweisen vorgestellt werden, die dieses Ziel verfolgen. Dazu wird nicht mehr allgemein von Anwendungen gesprochen, sondern es ist notwendig, auf dieser Ebene von einzelnen Tasks zu sprechen. Hierbei kontrolliert die Anwendung mehrere Tasks, die auf dem Prozessor ausgeführt werden.

Eine erste Maßnahme, um sicherheitskritische und nicht sicherheitskritische Tasks unterscheiden zu können, ist die Einführung von sogenannten Kritikalitätslevel. Dazu sei folgende Definition gegeben:

Definition 1 *Ein System besteht aus einer endlichen Anzahl an Komponenten K . Jede Komponente besitzt ein Kritikalitätslevel L und enthält eine endliche Menge sporadischer Tasks τ_i . Jeder Task ist dabei durch die folgenden Eigenschaften eindeutig definiert: Seine Periode T_i , Deadline D_i , Ausführungszeit C_i und das Kritikalitätslevel L_i .*

Das System kann zwischen verschiedenen Kritikalitätsleveln wechseln, abhängig von dem Verhalten der Tasks. Tasks, die eine kürzere Periode haben, bekommen bei den in MCS hauptsächlich verwendeten Scheduling Strategien (wie z. B. Rate Monotonic) eine höhere Priorität zugewiesen. Das bedeutet, dass bei einer aktivierenden Task mit einer kürzeren Periode das System in ein höheres Kritikalitätslevel wechselt. Meist handelt es sich um zwei Stufen:

1. High (HI): Tasks haben längere Ausführungszeiten oder kürzere Perioden.
2. Low (LO): Tasks haben kürzere Ausführungszeiten oder längere Perioden.

Oft ist auch nur der Wechsel in ein höheres Kritikalitätslevel erlaubt (detailliertere Informationen s. [BD14], Seite 7 ff.).

Die erste wichtige Arbeit zum Mixed-Criticality Scheduling auf Einkernprozessoren kam von Steve Vestal [Ves07]. Er hat festgestellt, dass das Deadline Monotonic Schedulingverfahren für feste Perioden und Ausführungszeiten zwar optimal ist, aber nicht bei Mixed-Criticality Scheduling. Bei letzterem existieren unterschiedliche Startzeiten und variable



Ausführungszeiten der Tasks²⁴. Weiterhin ist Earliest Deadline Late (EDL) beim Scheduling von periodischen und aperiodischen (sicherheitskritischen und nicht sicherheitskritischen) Tasks einem statischen Schedulingverfahren vorzuziehen und sogar optimal, allerdings nicht bei unterschiedlichen Kritikalitätsleveln unter den Tasks [BD14]. Diese Beobachtungen unterstreichen die Schwierigkeiten und die neuen Herausforderungen beim Scheduling von gemischt kritischen Tasks auf einer Plattform.

Vestal schlägt als Lösung ein statisches Schedulingverfahren mit vorheriger Bestimmung der Prioritäten der Tasks durch Audsley's Priority Assignment Algorithmus vor, der für Einkernprozessoren optimal ist [DRRG10]. Der Algorithmus wird in Quelle [Aud91], Seite 12, vorgestellt. Vestals Verfahren trägt den Namen: SMC-NO (static mixed criticality with no run-time monitoring). Dabei ist ein Wechsel der Kritikalitätslevel während der Laufzeit verboten, was ein noch sehr restriktives Verfahren zeigt.

Eine weitere passende Schedulingstrategie ist das Slack Scheduling bei Systemen mit zwei Kritikalitätsleveln. Die Tasks mit dem niedrigeren Kritikalitätslevel laufen im Slack (nicht genutzte Zeitslots) von Tasks mit höherem Kritikalitätslevel. Der Slack ist dabei wie folgt definiert:

$$\text{slack}(t) = D_i - t - C_i(t) \quad (7.2)$$

$C_i(t)$ steht für die verbleibende Ausführungszeit von Task $\tau_i(HI)$ zum Zeitpunkt t (vgl. [Ret14], Kapitel 4). D_i ist hier die vorher schon definierte Deadline von Task τ_i .

Ein passendes statisches Schedulingverfahren ist Rate Monotonic, bei dem die Tasks mit kürzeren Perioden (also einem höheren Kritikalitätslevel) auch eine höhere Priorität bekommen. Mehrere Tasks desselben Kritikalitätslevels können dabei auch untereinander unterschiedliche Prioritäten bekommen, die aber alle höher sein müssen als die höchste Priorität im nächst niedrigeren Kritikalitätslevel. Wenn Tasks trotz hoher Kritikalität eine lange Periode haben, kann das Verfahren der Period Transformation angewandt werden. Dies ermöglicht es, die Periode und Ausführungszeit zu teilen, sodass aus einem Task zwei Teile entstehen. Dieses Vorgehen kann solange wiederholt werden, bis die Taskteile die gewünschte Periodenlänge aufweisen und dadurch eine höhere Priorität erreicht wurde [Ves07]. Die Sub-Tasks werden dadurch automatisch direkt hintereinander ausgeführt und können nur durch Tasks mit höherer Priorität unterbrochen werden. Dieses Verfahren muss bei einem unterliegenden statischen Schedulingverfahren (wie etwa beim Rate Monotonic) vor der Laufzeit durchgeführt werden.

Die erste wichtige Arbeit zum Mixed-Criticality Scheduling auf Mehrkernprozessoren kam von Anderson u. a. [ABB09].

²⁴Unterschiedliche Ausführungszeiten treten auf, da diese von der Worst Case Execution Time (WCET) Analyse abhängen und diese für höhere Kritikalitätslevel konservativer abgeschätzt wird, d. h. bei einem Wechsel des Kritikalitätslevel ändern sich auch die Ausführungszeiten der Tasks

In Abbildung 7.2 ist der traditionelle Ansatz dargestellt, der eine strikte Trennung der sicherheitskritischen und nicht sicherheitskritischen Anwendungen vorsieht, dem neuen Ansatz einer gemischt kritischen Multi-Core Plattform gegenüber gestellt. Ein großer Vor-

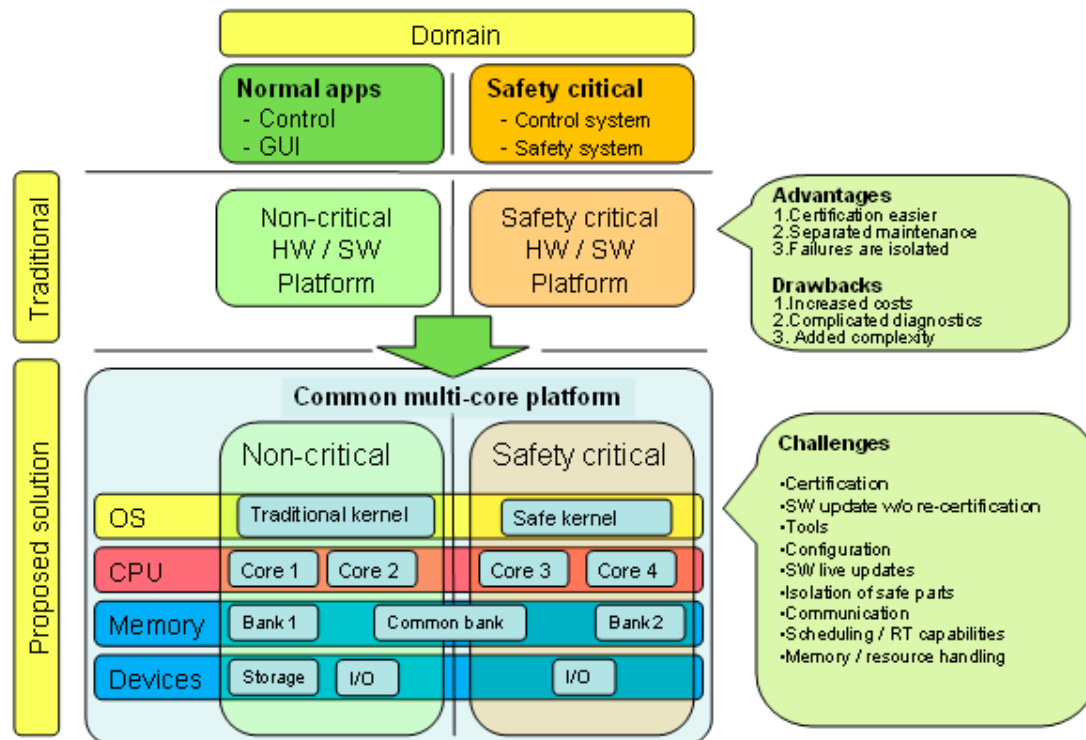


Abbildung 7.2: Herausforderungen an Multi-Core Plattformen, Quelle: <http://atcproyectos.ugr.es/recomp/>

teil der traditionellen Herangehensweise ist die Fehlerisolierung, wohingegen die Kosten steigen. Auf der Multi-Core Plattform kann man sehr gut den in Kapitel 7.1.4 genannten Vorteil der Flexibilität nachvollziehen. Jede der beiden Anwendungen hat seinen eigenen Betriebssystemkernel und seine zugeordneten Prozessorkerne, sowie Arbeitsspeicher. Um Daten untereinander austauschen zu können, gibt es einen geteilten Speicherbereich (in Abbildung 7.2 Common bank genannt). Genau Diese Struktur bewirkt die hauptsächliche Schwierigkeit solcher Systeme, da sie sich hier gegenseitig beeinflussen können.

In [ABB09] werden fünf Kritikalitätslevel eingeführt, die jeweils ein unterschiedliches Schedulingverfahren beinhalten. Jeder Prozessor, dem eine Komponente (beinhaltet wieder mehrere Tasks) eines der folgenden Kritikalitätslevel zugeordnet wird, wendet das dahinter liegende Verfahren an:

A Statisches Schedulingverfahren



- B** Teilweise präemptives EDF²⁵
- C** Global präemptives EDF
- D** Global präemptives EDF
- E** Heuristisches Verfahren (z. B. Round Robin)

Um eine geplante Kommunikation auf dem Bus zu ermöglichen, kann der Einfachheit halber ein Zeitschlitzverfahren namens Time division multiple access (TDMA) angewendet werden. Hierfür wird der Bus für eine zu definierende Zeiteinheit für ein bestimmtes Kritikalitätslevel freigegeben und danach gewechselt (Round Robin). Innerhalb dieses Kritikalitätslevels kann wiederum das First Come First Serve (FCFS)-Prinzip verwendet werden. Die Zuordnung, welche Task auf welchem Prozessor ausgeführt werden soll, kann bei fehlenden Erfahrungswerten z. B. einfach durch einen evolutionären Algorithmus entschieden werden [BD14].

Der entscheidende Punkt eines Entwurf einer Multi-Core Plattform liegt in der Verwaltung der geteilten Ressourcen wie z. B. dem Shared Memory (siehe Abbildung 7.2). Dazu gibt es zwei Verfahren, die beim Scheduling von periodischen und aperiodischen Tasks mit Ressourcenzugriffen schon erprobt sind und nur durch die Vererbung des Kritikalitätslevels erweitert wurden [BD14]:

1. Priority and Criticality Inheritance Protocol (PCIP)
2. Priority and Criticality Ceiling Protocol (PCCP)

Einen deutlich restriktiveren Ansatz bietet der Standard ARINC 653²⁶, der die Ressourcenpartitionierung und das Zeitmanagement von Avionik-Systemen regelt. Dabei werden der sicherheitskritischen und der nicht sicherheitskritischen Applikation dedizierte Prozessorbereiche zugeordnet. Diese Bereiche müssen physisch voneinander getrennt sein, sodass in jeder Einheit nur Funktionen des gleichen Kritikalitätslevels sind. Der Datenfluss wird über einen Bus oder ein Betriebssystem verwaltet.

7.1.6 Fazit für diese Projektgruppe

Die für diese Projektgruppe bevorzugte Vorgehensweise beim Entwurf des MCS und der damit zusammenhängenden Trennung des ASGs und AEVV-Systems ist, angelehnt an ARINC 653, eine zeitliche und örtliche Trennung der Systeme. Dies hat zur Folge, dass jede Task die ihm zugesicherte Ausführungszeit auf dem Prozessor erhält, unabhängig von den anderen Tasks. Des Weiteren wird verhindert, dass Tasks auf schon allokierten Speicher schreibend zugreifen, wodurch Interferenzen verursacht werden. Die örtliche Trennung

²⁵Die Tasks mit der jeweils frühesten Deadline werden als erstes ausgeführt und können Tasks mit späterer Deadline unterbrechen.

²⁶Zu erwerben unter: <https://store.aviation-ia.com/>



beinhaltet, dass geteilte Ressourcen möglichst vermieden werden sollten. Falls sie doch notwendig sind, sollte nur eine Komponente schreibend darauf zugreifen und die anderen sich auf einen lesenden Zugriff beschränken. Weiterhin muss das ASG ein höheres Kritikalitätslevel als das AEVV-System besitzen. Auf einem eventuell verwendeten Bus sollte eines der im Kapitel 7.1.5 vorgestellten Schedulingverfahren angewendet werden. Beim manuellen Mapping auf die in dieser Projektgruppe vorliegende Plattform muss weiterhin beachtet werden, dass genug Kapazitäten auf den Ressourcen vorhanden sein müssen, um jederzeit die Ausführung der sicherheitskritischen Anwendung zu gewährleisten.

Das tatsächliche und endgültige Mapping der Komponenten und die Platzierung der Sensoren und Aktoren befindet sich im Kapitel 7.5.

7.2 Architekturentwurf

Ziel des Architekturentwurfs ist die Abbildung der Systemfunktionalität auf die vorhandene Hardware. Hierzu wird das System zunächst in ein hierarchisches Modell von Komponenten überführt, die jeweils eine abgeschlossene Funktionalität repräsentieren. Dabei sind die Komponenten ASG, AEVV-System, Telemetriestation und Funksteuerung bereits aus der Systemdefinition im Kapitel 4 und dem Szenario in Kapitel 1.4 bekannt. Sie werden in die Hierarchie des Architekturmodells eingegliedert und weiter verfeinert. Die Kommunikation der Komponenten untereinander wird in Form von Schnittstellen erfasst. Für die Modellierung werden UML-Komponentendiagramme genutzt, wobei die Schnittstellen aus Kapitel 7.2.4 näher erläutert werden.

Das entstandene Modell wird in Kapitel 7.5 für die Abbildung der Funktionalität auf die vorhandene Hardware genutzt. Außerdem soll es soll der späteren Hard- und Softwareimplementierung als Vorlage für deren Struktur dienen.

7.2.1 Gesamtsystem und Quadrokooper

Die erste Hierarchieebene des Architekturmodells ist das Gesamtsystem, welches in Abbildung 7.3 dargestellt ist. Es verdeutlicht die Zusammenhänge der einzelnen Teilsysteme. Die Komponente *Remote Control* repräsentiert die Fernsteuerung und leitet die Steuersignale des Piloten an den Quadrokooper weiter. Außerdem sendet sie Steuersignale für die De- bzw. Aktivierung von der Bildaufnahme und der Objekterkennung. Weiterhin muss sie Telemetriedaten vom Quadrokooper empfangen und anzeigen können (siehe Kapitel 6.3). Die Komponente *Video Clients* stellt mögliche Endgeräte dar, die den Videostream von der Telemetriestation empfangen und anzeigen können, was in der optionalen, funktionalen Anforderung FO006 beschrieben ist.

Die Komponente *Quadrokooper* lässt sich in zwei weitere Teilsysteme aufteilen und ist in Abbildung 7.4 ausführlicher dargestellt. Sie repräsentiert den Quadrokooper und besteht

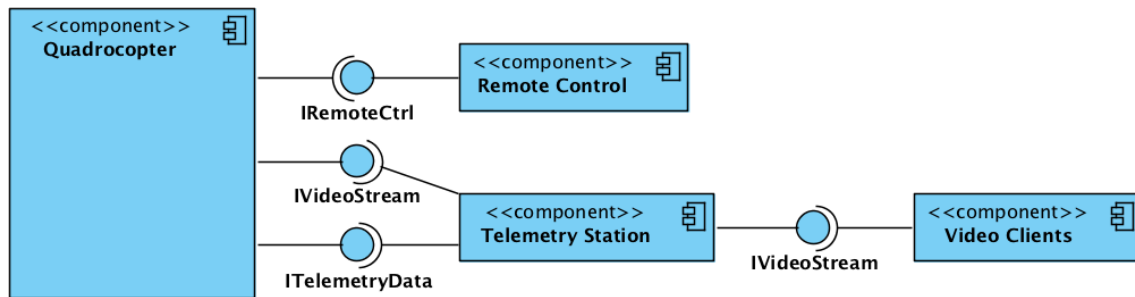


Abbildung 7.3: Komponentendiagramm des Gesamtsystems

aus dem AEVV-System, ASG und die benötigten Hardwarekomponenten. Die vom ASG und vom AEVV-System verwendeten Hardwarekomponenten sind grau hervorgehoben.

Das ASG verwendet die Motoren zur Steuerung des Quadropters und die Sensoren, um die Beschleunigung, die Höhe und die Lage im Raum zu erfassen. Außerdem wird das ASG um ein Funkmodul für die Verbindung zur Fernsteuerung erweitert.

Das AEVV-System hat eine Schnittstelle zum Gimbal und der Kamera, um ein Videosignal zu empfangen und die Kamera ausrichten zu können. Weiterhin benötigt das AEVV-System ein Funkmodul, um mit der Telemetriestation kommunizieren zu können.

Entsprechend den Schlussfolgerungen aus dem Kapitel 7.1.6 ist die IAvionic Schnittstelle der einzige Berührungspunkt der beiden Systeme. Der Aufbau von AEVV-System, ASG und Telemetriestation wird in den folgenden Kapiteln ausführlich beschrieben.

7.2.2 ASG

Hauptaufgabe des ASGs ist es aus der Ist-Situation des Quadropters und den Steuersignalen des Piloten eine entsprechende Ansteuerung der Motoren vorzunehmen. Dargestellt ist es in Abbildung 7.5. Die Erfassung der Ist-Situation geschieht durch die Komponente *Sensor Drivers*. Sie übernimmt die Konfiguration und Ansteuerung der Sensoren um deren Daten auszulesen. Die Komponente *Remote Control Transceiver* stellt eine bidirektionale Funkverbindung zur Fernbedienung zur Verfügung und empfängt darüber die Steuersignale des Piloten. Zusätzlich bietet sie einen Rückkanal zur Fernbedienung für die Darstellung von Telemetriedaten. Die Steuersignale und Sensordaten werden u. a. vom *Avionic Controller* abgerufen, der mit ihnen die Regelung der Fluglage durchführt und entsprechende Motorstellwerte generiert. Eine detailliertere Beschreibung dieser Komponente folgt. Die Stellwerte der Motoren werden an den *Motor Driver* übergeben, der diese in die Ansteuerung der Motoren umsetzt.

Zusätzlich führt das ASG eine kontinuierliche Überwachung der Flughöhe und des Ladezustands der Akkumulatoren durch. Die jeweiligen Werte und eine Warnung bei kritischem Ladezustand werden über den Rückkanal der Funkverbindung zur Fernbedienung übertragen. Diese Aufgabe wird durch die Komponente *Monitoring* umgesetzt. Weiterhin

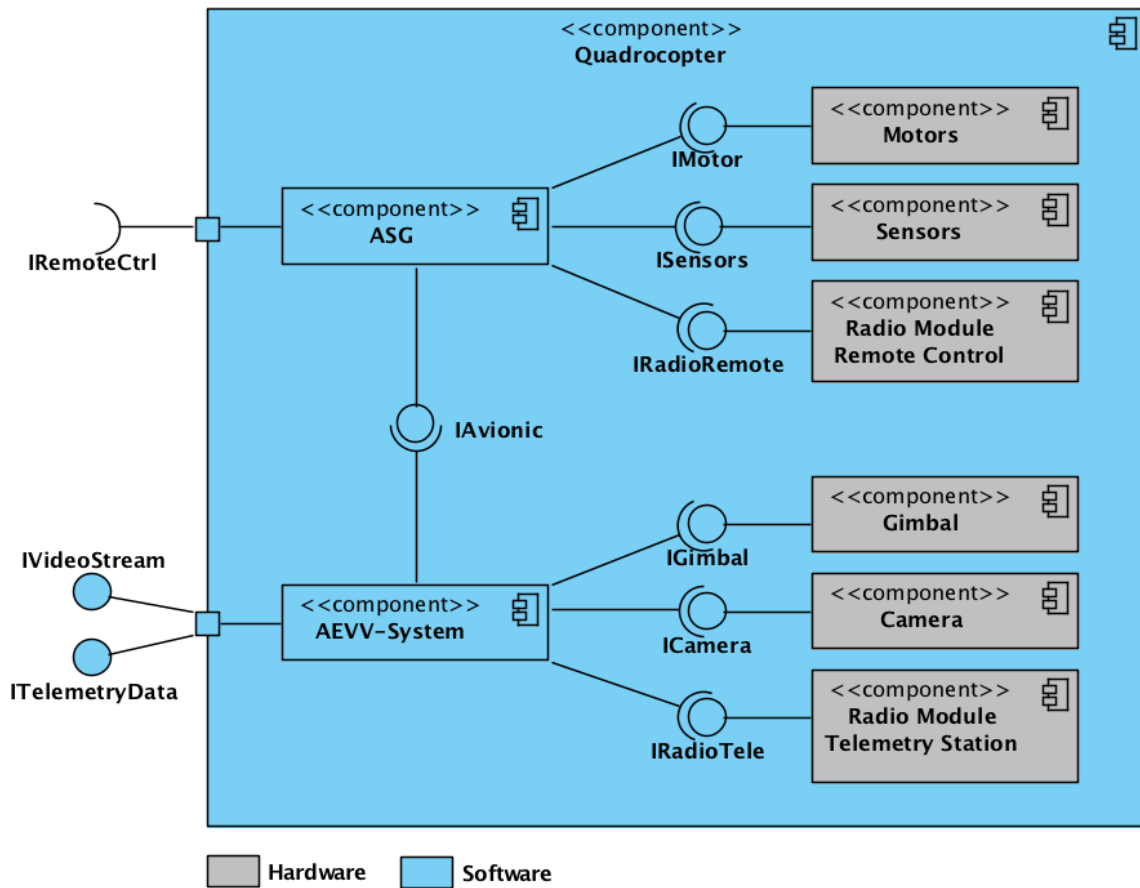


Abbildung 7.4: Komponentendiagramm des Quadropters

stellt diese Komponente die Sensordaten und Steuersignale (siehe Kapitel 1.4) dem AEVV-System zur Verfügung. Der exakte Datenfluss innerhalb des ASGs ist in Abbildung 7.6 dargestellt um die Zusammenhänge zwischen den Komponenten zu verdeutlichen.

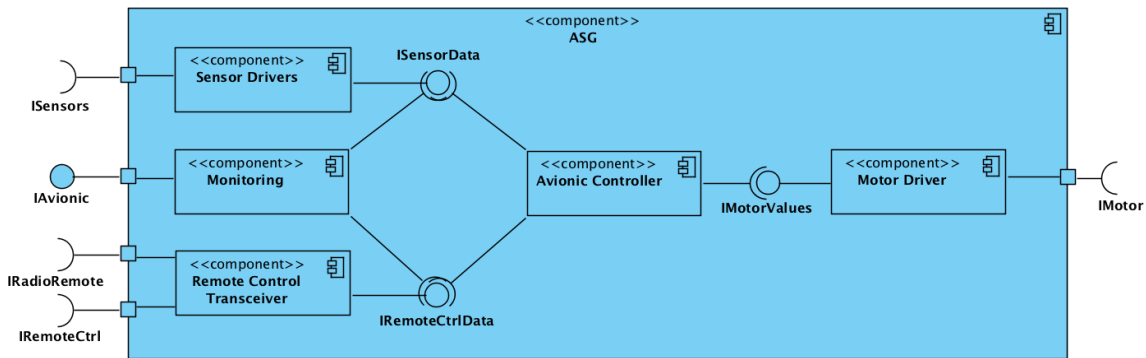


Abbildung 7.5: Komponentendiagramm des ASGs

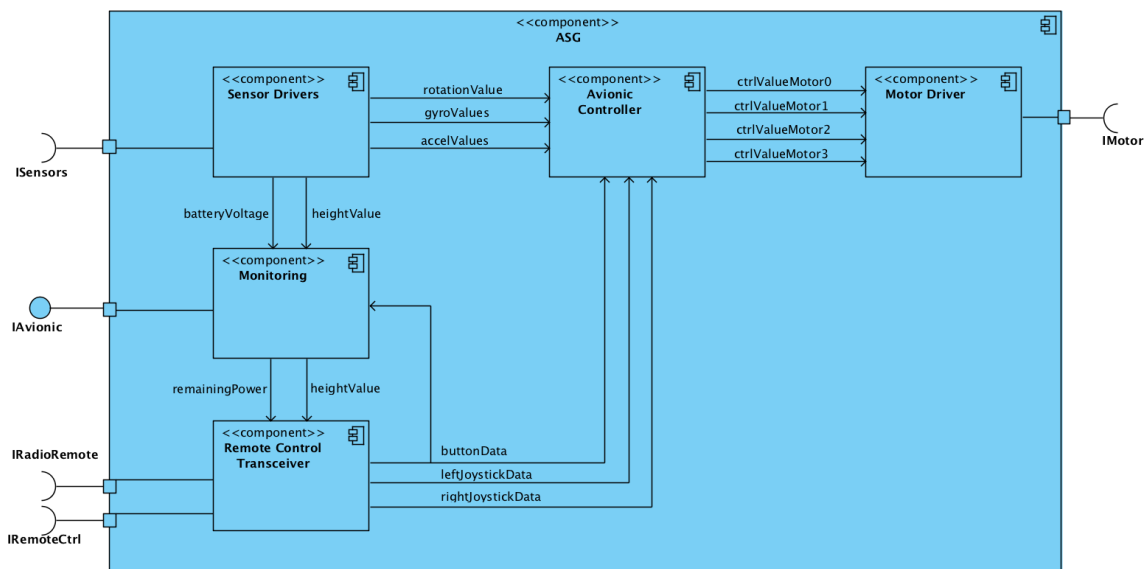


Abbildung 7.6: Datenflussgraph des ASGs

Der in Abbildung 7.7 dargestellte *Avionic Controller* übernimmt die Regelung des Quadropters anhand der von der Fernbedienung übermittelten Sollwerte. Dabei sind die folgenden Teilkomponenten für die Funktionalität des *AvionicControllers* wichtig. Die Komponente *Calculate Setpoints* ermittelt aus den Stellwerten der Joysticks und dem Status der Buttons der Fernbedienung die passenden Führungsgrößen für die Regelung. Die Komponente *Sensor Data Preprocessing* führt eine Vorverarbeitung der Sensordaten durch um evtl. auftretende Ausreißer in den Daten zu eliminieren. Der *Attitude Controller* übernimmt die Lageregelung anhand der von der Fernbedienung erhaltenen Führungsgröße und der von den Sensoren gelieferten Regelgröße. Der *Height Controller* ist für die Höhenregelung zuständig und ermittelt ebenfalls anhand der Führungs- und Regelgröße die

Stellgröße für die Motoren bzgl. der Höhe. Diese Komponente kann über die Fernbedienung manuell ein- und ausgeschaltet werden. Die Komponente *Motor Values Calculator* berechnet dann aus den einzelnen Stellgrößen der vorangegangenen Regelung die Stellwerte für die vier Motoren.

Um den Datenfluss im Detail nachvollziehen zu können, ist in Abbildung 7.8 ein Datenflussgraph des *Avionic Controllers* dargestellt.

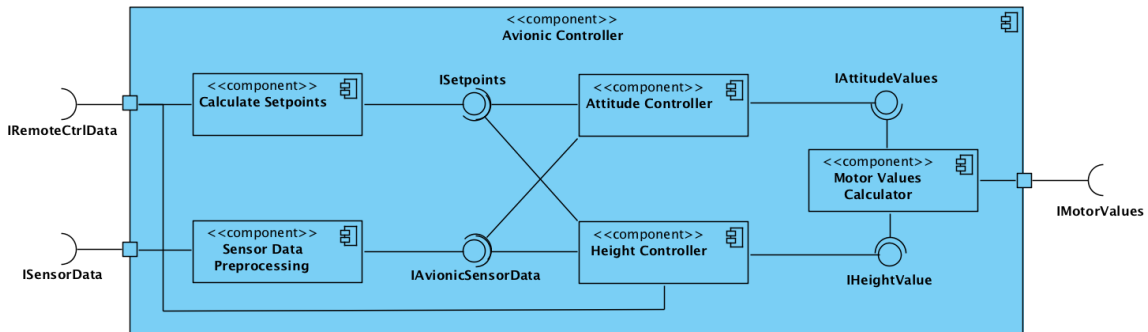


Abbildung 7.7: Komponentendiagramm des AvionicControllers

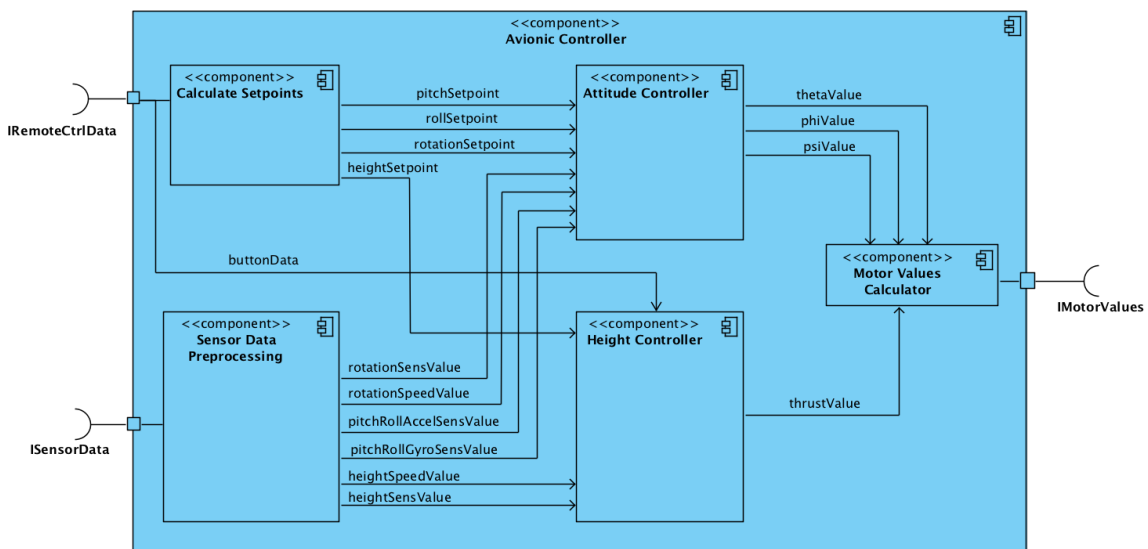


Abbildung 7.8: Datenflussgraph des AvionicControllers

7.2.3 AEVV-System und Telemetriestation

Das *AEVV-System* ist für die Videoverarbeitung, Steuerung der Kamera und Weiterleiten der Daten an die Telemetriestation zuständig. Die Subkomponenten werden im Folgenden beschrieben und sind in Abbildung 7.9 dargestellt. Der *AEVVController* stellt die Avionik Daten wie die Höhe des Quadropters oder den aktuellen Batteriestatus bereit. Diese Daten bekommt er vom *ASG*. Des Weiteren steuert er das Ein- und Ausschalten der Ka-

mera sowie der Objekterkennung. Der Übersicht halber wurden diese Schnittstellen nicht im Komponentendiagramm eingezeichnet, müssen aber bei der Implementierung beachtet werden. Der *Gimbal Controller* sendet Positionsvorgaben an den Gimbal, um die Kamera auf den Ball auszurichten. Der *Video Grabber* kommuniziert mit der Kamera und stellt den Videostream der Kamera bereit. Die *Object Detection* verwendet den Videostream, ermittelt die Position des Balls und stellt diese zur Verfügung. Der *Telemetry Data Collector* sammelt alle Sensordaten wie Flughöhe, Batteriestatus und die aktuelle Position des Balls im Kamerabild und stellt diese über das Funkmodul der Telemetriestation bereit. Der *Streaming Server* verwendet den Videostream vom *Video Grabber* und stellt diesen über das Funkmodul der Telemetriestation bereit.

Zur besseren Übersicht wurde vom *AEVV-System* ein Datenflussgraph angefertigt, dieser ist in Abbildung 7.10 zu sehen.

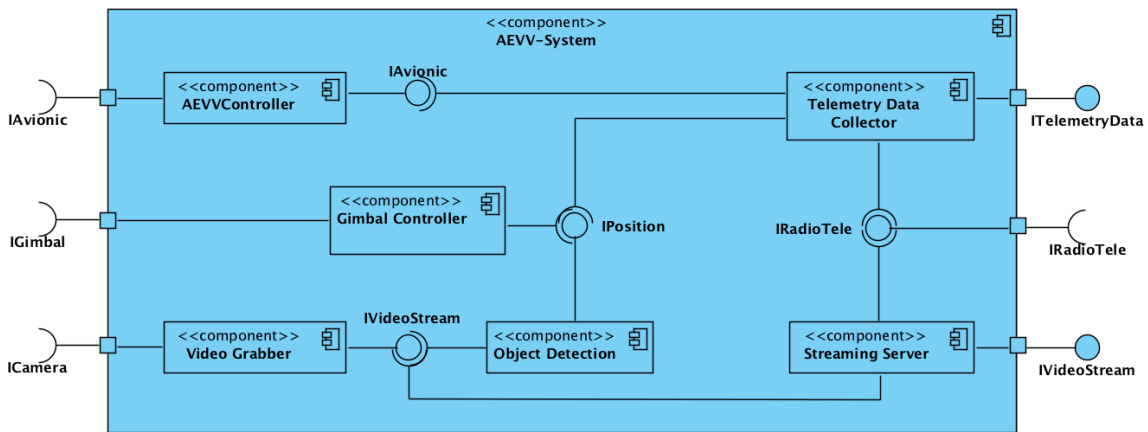


Abbildung 7.9: Komponentendiagramm des AEVV-Systems

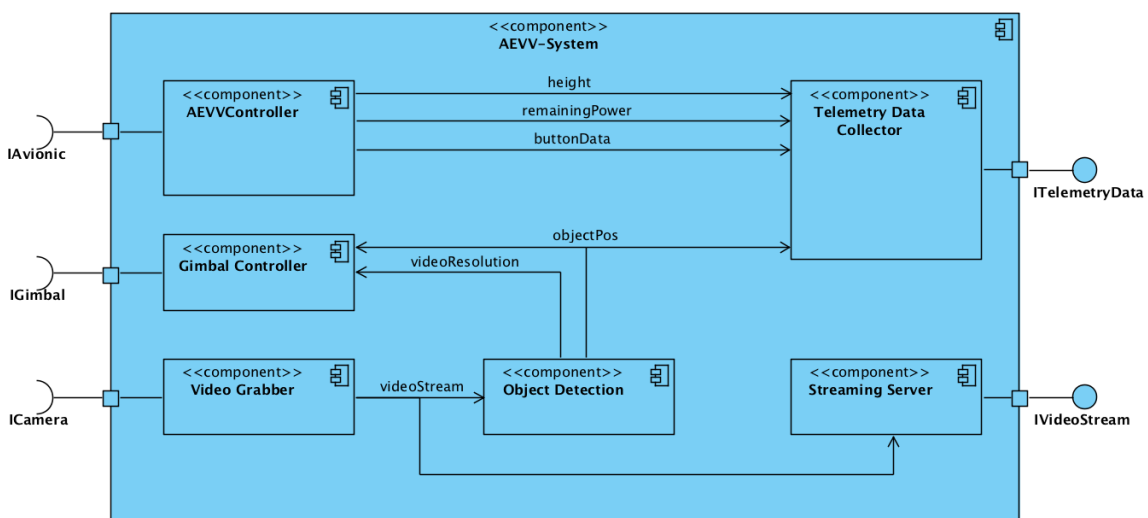


Abbildung 7.10: Datenflussgraph des AEVV-Systems



Die Telemetriestation nimmt Daten vom Quadrokopter entgegen und stellt sie dem Piloten auf einem Bildschirm zur Verfügung. Die Daten bestehen aus dem Kamerabild als Videostream und den Telemetriedaten des Quadrokopters (siehe Kapitel 1.4). Die benötigte Funktionalität wird durch die in Abbildung 7.11 dargestellten Teilkomponenten realisiert. Das *Radio Module Quadrocopter* stellt zusammen mit einem entsprechenden Gegenpart auf dem Quadrokopter die Funkverbindung bereit. Der *Streaming Server* nimmt den Videostream vom Quadrokopter entgegen und stellt ihn intern für die Weiterverarbeitung bereit. Wenn benötigt nimmt er eine Skalierung und Konvertierung des Videoformates vor. Die *Local Visualization* stellt das Kamerabild zusammen mit den Telemetriedaten in einer graphischen Oberfläche dar. Optional stellt die Telemetriestation den Videostream für Endgeräte zur Verfügung, diese Funktion wird vom *Webserver* übernommen. Die benötigte Netzwerkkonnektivität wird durch das *Radio Module Video Clients* zur Verfügung gestellt (siehe FO006).

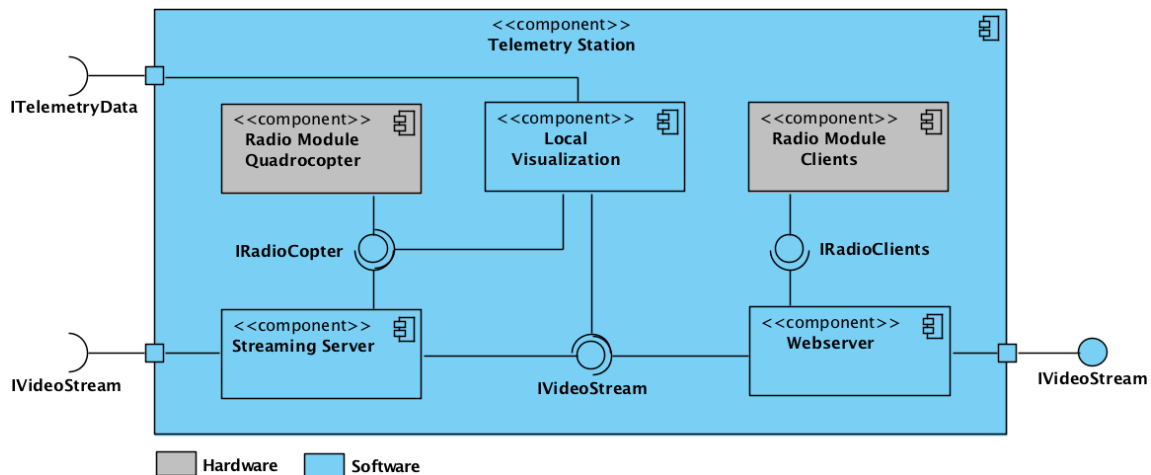


Abbildung 7.11: Komponentendiagramm der Telemetriestation

7.2.4 Beschreibung der Interfaces

In diesem Kapitel werden die Interfaces in tabellarischer Form dargestellt, dazu werden die jeweiligen Daten des Interfaces spezifiziert und genauer beschrieben. Weiterhin wird bei jedem Interface auf das zugehörige Komponentendiagramm verwiesen. Die Schnittstellen zu den Hardware Komponenten werden hier nicht im Detail beschrieben, da diese schon fest definiert sind und in den Datenblättern nachgelesen werden können.

Die wichtigste Schnittstelle ist im Folgenden die *IAvionic*-Schnittstelle, welche die Kommunikation zwischen dem ASG und dem AEVV-System beschreibt. Dieses Interface ist von besonderer Bedeutung, da darüber die Kommunikation zwischen dem sicherheitskritischen und dem nicht sicherheitskritischen System stattfindet. Beim Entwurf dieser Schnittstel-

le muss insbesondere auf die speziellen Anforderungen, die durch die Mixed Criticality entstehen, geachtet werden (siehe Kapitel 7.1.3).

System

IRemoteCtrl		
Daten	Beschreibung	Referenzen
Rohdaten der Fernbedienung	Steuerrohdaten, die enthalten, welche Eingaben auf der Fernbedienung gemacht wurden.	Abbildung 7.3, FM003

IVideoStream		
Daten	Beschreibung	Referenzen
videoStream	Videostream der Kamera mit einer Mindestauflösung von WVGA	Abbildung 7.3, NA007

ITelemetryData		
Daten	Beschreibung	Referenzen
attitude	Lage des Quadrokopter	Abbildung 7.3, FM009, FM010, FM011, FM012
heigth	Flughöhe des Quadrokopters	
batteryStatus	Ladezustand des Akkus	
motorCtrlValues	Reglerwerte, die an die einzelnen Motoren gesendet werden	
sensorValues	Werte, die aus den Sensoren ausgelesen werden	

Quadrokopter

IAvionic		
Daten	Beschreibung	Referenzen
attitude	Lage des Quadrocopter	Abbildung 7.4, FM010, FF001, FF002, FF003
buttonData	Informationen darüber, ob und welche Buttons auf der Fernbedienung gedrückt werden	

ASG

ISensorData		
Daten	Beschreibung	Referenzen
heightValue	Vom Höhengsensor erfasst Flughöhe (unskaliert)	Abbildung 7.5, FM010, FM011, FM012
rotationValue	Vom Kompass erfasst Rotation (unskaliert)	
GyroValues	Von den Gyroskopen erfasste Lage im Raum (unskaliert)	
AccelValues	Von den Beschleunigungssensoren erfasste Lage im Raum (unskaliert)	
batteryStatus	Ladezustand des Akkus	

IRemoteCtrlData		
Daten	Beschreibung	Referenzen
leftJoystickData	Stellwerte des linken Joysticks	Abbildung 7.5, FM011, FM014, FF001, FF002, FF003
rightJoystickData	Stellwerte des rechten Joysticks	
buttonData	Informationen darüber, ob und welche Buttons auf der Fernbedienung gedrückt werden	
height	Flughöhe (skaliert)	
remainingPower	Verbleibende Akkuleistung (skaliert)	

IMotorValues		
Daten	Beschreibung	Referenzen
ctrlValueMotor0	Stellwert für Motor 0	Abbildung 7.5, FM009
ctrlValueMotor1	Stellwert für Motor 1	
ctrlValueMotor2	Stellwert für Motor 2	
ctrlValueMotor3	Stellwert für Motor 3	

Avionic Controller



IAvionicSensorData		
Daten	Beschreibung	Referenzen
heightSensValue	Sensorwerte des Höhsensors	Abbildung 7.7, FM010, FM011
heightSpeedValue	Sensorwerte des Beschleunigungssensors bezüglich der Beschleunigung in Richtung der Z-Achse	
rotationSensValue	Sensorwerte des Kompasses	
rotationSpeedValue	Sensorwerte des Beschleunigungssensors bezüglich der Rotation um die Z-Achse	
pitchRollGyroSensValue	Sensorwerte des Gyroskops bezüglich X- und Y-Achse	
pitchRollAccelSensValue	Sensorwerte des Beschleunigungssensors bezüglich der Beschleunigung Richtung X- und Y-Achse	

ISetpoints		
Daten	Beschreibung	Referenzen
heightSetpoint	Führungsgröße der Höhe in skaliertem Darstellung	Abbildung 7.7, FM004
rotationSetpoint	Führungsgröße der Rotation in skaliertem Darstellung	
pitchSetpoint	Führungsgröße der Nickachse in skaliertem Darstellung	
rollSetpoint	Führungsgröße der Rollachse in skaliertem Darstellung	

IAttitudeValues		
Daten	Beschreibung	Referenzen
thetaValue	Winkel der Nickachse in skaliertes Darstellung	Abbildung 7.7, FM007
phiValue	Winkel der Rollachse in skaliertes Darstellung	
psiValue	Winkel der Gierachse in skaliertes Darstellung	

IHeightValue		
Daten	Beschreibung	Referenzen
thrustValue	Schub in skaliertes Darstellung	Abbildung 7.7, FM008

AEVV

IPosition		
Daten	Beschreibung	Referenzen
objectPos	X- und Y-Koordinaten an denen der Ball auf dem Bild erkannt worden ist	Abbildung 7.9, FA004, NA007
videoResolution	Auflösung des Videostreams	

7.3 Auswahl und Beschreibung der Hardware

In diesem Kapitel wird die Hardware beschrieben, die im weiteren Verlauf des Projektes verwendet wird. Dazu gehört:

- die Prozessorplattform,
- die Kamera,
- der Gimbal,
- die Sensorik,
- das Funkmodul zur Telemetriedatenübertragung,
- die Telemetriestation,
- der Multikopter,
- das Motherboard



7.3.1 Prozessorplattform

Die Prozessorplattform, auf der die Videoverarbeitung und die Regelung/Steuerung des Multikopters parallel verarbeitet werden (siehe Kapitel 1.2 Aufgabenstellung), muss unter anderem folgende Anforderungen erfüllen²⁷:

- hohe Rechenleistung
- niedriger Energieverbrauch
- möglichst geringes Gewicht
- kleine Baugröße
- ausreichend I/O-Pins
- Bereitstellung von Sensorschnittstellen (I²C, etc.)

Auf Basis dieser Anforderungen wird ein Xilinx Zynq 7020 ausgewählt. Im Folgenden wird die Entscheidung begründet.

Rechenleistung Mit dem Xilinx Zynq 7020 existiert eine sehr rechenstarke Multicore Plattform, die darüber hinaus einen FPGA-Anteil bereit stellt. Der Prozessor besteht aus zwei Advanced RISC Machines (ARM) Cortex-A9 Kernen mit jeweils 667 MHz. Der FPGA ist ein Artix-7 mit 85000 Logikzellen.

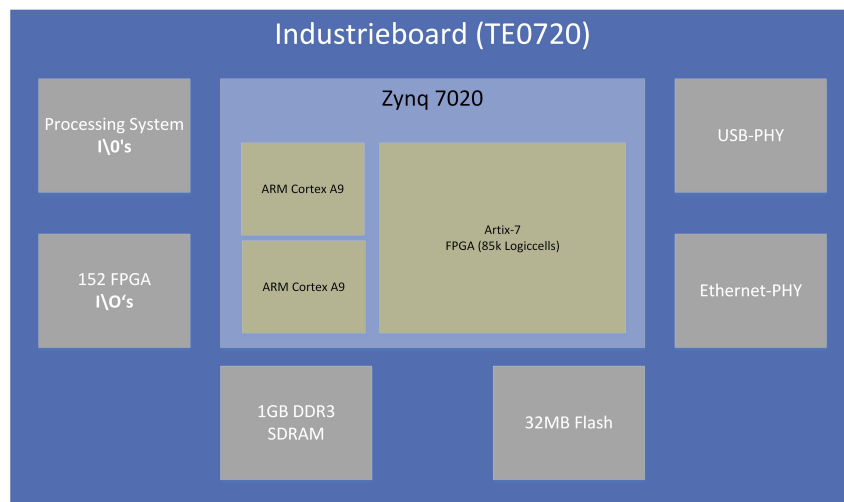


Abbildung 7.12: Blockschaltbild des Industrieboards

Von der Firma "Trenz Electronics" wird ein Industrieboard vertrieben, welches die Zynq 7020 Prozessorplattform um die in Abbildung 7.12 gezeigte zusätzliche Peripherie erweitert. Dazu gehört bspw. ein USB-PHY, 1 GB DDR3 SDRAM und 32 MB Flash Speicher. 1 GB RAM genügt den Anforderungen an die Ressourcen aus dem Kapitel 6.8.

²⁷Spezielle Anforderungen bezüglich der Mixed-Criticality werden im Kapitel 7.1 genauer beschrieben.



Im FPGA-Anteil kann speziell auf die Anwendung zugeschnittene Logik realisiert werden. Dadurch lässt sich die Effizienz des Gesamtsystems erhöhen.

I/Os Das Industrieboard stellt eine Vielzahl von konfigurierbaren I/Os zur Verfügung. Über den FPGA Anteil kann direkt auf 152 I/Os zugegriffen werden.

Energieverbrauch Eine hohe Rechenleistung steht immer im Kontrast zu einem niedrigen Energieverbrauch. Hohe Rechenleistung bedeutet i.d.R. hohe Taktraten. Hohe Taktraten führen tendenziell zu einer höherer Verlustleistung.

Das Industrieboard wird wie in Kapitel 7.3.8 beschrieben auf einem Carrierboard betrieben. Dieses benötigt eine 2 A Stromversorgung bei 5 V [Elea]. Daraus resultiert eine maximale Leistungsaufnahme von 10 W. 10 W sind für die bereitgestellte Rechenleistung akzeptabel.

Gewicht und Baugröße Aufgrund der Anwendung auf einem Multikopter spielen neben der Leistungsaufnahme auch die Größe sowie das Gewicht der Plattform eine wichtige Rolle. Das Gewicht des Carrierboards beträgt ca. 83 g und das des Industrieboards ca. 24 g. Das Carrierboard hat eine Baugröße von 64,5 x 100,0 mm und kann somit problemlos auf dem Fluggerät montiert werden (siehe Kapitel 7.3.6).

Schnittstellen Die Zynq 7020 Plattform bietet 14 Hardwareschnittstellen (darunter I²C, UART, CAN, etc.). Vor allem die I²C-Schnittstelle bietet sich zum Anschluss der benötigten Sensoren an. Weitere Schnittstellen können im FPGA instantiiert werden. Die entsprechenden FPGA I/Os werden dann zu Schnittstellen Pins.

7.3.2 Kamera

In diesem Unterkapitel werden die Kamera und die Kamera-Schnittstellen beschrieben, die in diesem Projekt verwendet werden.

Die Kamera ist Teil des AEVV-Systems und muss daher den entsprechenden Anforderungen aus dem Kapitel 6.2 genügen. Diese Anforderungen sind:

- FA003: Kamerabild bereitstellen
- NA003: Erschütterungsresistenz der Kamera
- NA007: Auflösung des Videostreams

Um die Erschütterungsresistenz (NA003) zu gewährleisten, ist, wie im Kapitel 3.6 vorgestellt, eine Outdoor-Kamera zu bevorzugen. Ein weiteres Auswahlkriterium ist, dass die Kamera eine digitale Schnittstelle mit "Liveout"-Funktion bereitstellt. Diese Schnittstelle sollte, neben geringem zusätzlichen Hardwareaufwand, von der Bildverarbeitungssoftware OpenCV unterstützt werden, welche im weiteren Verlauf des Projekts genutzt wird.

In diesem Projekt wird als Kamera die Mobius ActionCam 1080p verwendet (siehe Abbildung 7.13).



Abbildung 7.13: Mobius ActionCam 1080p [Mob]

Vorteil dieser Kamera gegenüber konkurrierenden Produkten ist, dass sie als USB-Webcam betrieben werden kann. Die USB-Schnittstelle stellt eine Livestream mit einer Auflösung von bis zu 1080p zur Verfügung (FA003, NA007). Dieser kann direkt (ohne weiteren Hardwareaufwand) mit dem USB-Port der Adapterplatine (siehe Kapitel 7.3.8) verbunden und mit Hilfe von OpenCV auf dem Prozessor weiterverarbeitet werden.

Ein weiterer Vorteil ist das geringe Gewicht und die geringen Abmessung der Mobius ActionCam, die sich damit gut für die Platzierung in einem Gimbal eignet. Die Eckdaten der Kamera sind in Tabelle 7.1 zusammengefasst (Daten von der Website [LLC] entnommen).

Tabelle 7.1: Mobius ActionCam Daten

Name	Mobius ActionCam 1080p
Abmessungen	61 x 35 x 18 mm
Linse	Standard 46°(720p)
Gewicht	39 g
Kosten ²⁸	89,00 €
Liveout	Ja (Webcam und A/V)
Schnittstelle	USB 2.0, MircoSD-Karten Slot

²⁸Händler: amazon Zugriff: 02.10.2014



Auflösung im Liveout Modus	einstellbar bis 1080p
Datenkompression	MPEG und H.264

Die Mobius Kamera kann über einen Graphical User Interface (GUI) konfiguriert werden. Darin können beispielsweise Video- und Audio-Einstellungen vorgenommen werden. Genauere Informationen sind der Website zu entnehmen²⁹. Geladen wird die Kamera ebenfalls über die USB-Schnittstelle, wobei der Ladestrom 140 mA beträgt.[LLC]

Die Kamera wechselt in den Webcam Modus, wenn keine SD-Karte installiert ist und das USB Kabel angeschlossen wird. In diesem Modus können die Bilddaten bei eingestelltem Videokompressionsverfahren H.264 mit einer Auflösung von maximal 1080p und 30 fps übertragen werden. Die Videodatenrate beträgt bei diesen Einstellungen ca. 30000 kbit/s (messtechnisch ermittelt).

USB 2.0 hat nach Spezifikation eine maximale Bandbreite von 480 Mbit/s. Damit ist der USB zu ca.

$$C_{USB} = \frac{100\% \cdot B_{CAM}}{B_{USB}} \quad (7.3)$$

$$C_{USB} = \frac{100\% \cdot 480 \frac{Mbit}{s}}{29,3 \frac{Mbit}{s}} \approx 6,1\% \quad (7.4)$$

ausgelastet. Die restliche Bandbreite kann für das Videostreaming genutzt werden (siehe Kapitel 7.3.5).

Das Blockschaltbild Abbildung 7.14 stellt die Schnittstellen zur Kamera dar.

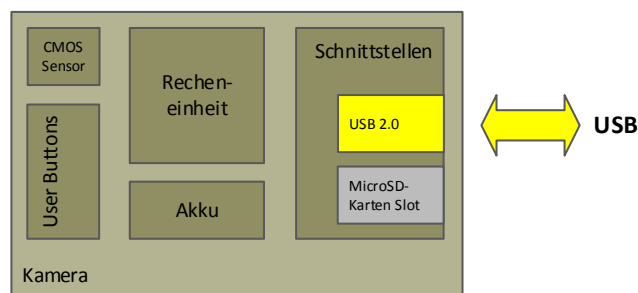


Abbildung 7.14: Blockschaltbild Kamera

²⁹In dieser Arbeit wird die GUI-Version 3.0.3.0 - und Firmwareversion 2.18 genutzt

7.3.3 Gimbal

In diesem Abschnitt wird der in diesem Projekt genutzte Gimbal zur Kamerapositionierung vorgestellt und die verwendeten Schnittstellen beschrieben. Der Gimbal ist Teil des AEVV-Systems und muss daher den entsprechenden Anforderungen aus dem Kapitel 6.2 genügen. Diese Anforderungen sind:

- FA008: Positionierung der Kamera
- FA009: Zentrierung des Objektes
- NA001: Genauigkeit der Positionierungseinheit
- NA002: Reaktionszeit der Positionierungseinheit
- NA005: Einbau der Peripherie

Um eine ausreichende Reaktionszeit sowie Genauigkeit der Kamerapositioniereinheit zu gewährleisten (NA001, NA002) und einen größtmöglichen Freiheitsgrad in der Kamerapositionierung zu erreichen (FA008), ist wie in Kapitel 3.4.4 vorgestellt, ein 3-Achs-Gimbal mit BLDC Motoren zu bevorzugen. Des Weiteren muss der Gimbal eine externe Positionierschnittstelle jeder Achse bereitstellen, um eine extern gesteuerte Positionierung der Kamera zu ermöglichen (FA009).

Ein solcher Gimbal ist beispielsweise der „DYS SMART Gopro BL Gimbal“ von DYS, welcher in diesem Projekt verwendet wird (siehe Abbildung 7.15).



Abbildung 7.15: DYS SMART Gopro BL Gimbal [Hob]

Dieser Gimbal ist speziell für die Montage unter einem Multikopter entworfen worden (NA005) und daher gut für den in dieser Arbeit verfolgten Einsatzzweck geeignet. Die Eckdaten des Gimbals sind in Tabelle 7.2 aufgelistet.



Tabelle 7.2: DYS 3-Achs Gimbal Daten

Name	DYS SMART Gopro BL Gimbal
Motor-Typ	Brushless
Anzahl Achsen	3
Gewicht	262 g
Kosten ³⁰	189,00 €
Controller Board	BaseCam Electronics SimpleBGC 32bit-Version
Unterstützt ext. Positionierung	alle drei Achsen, separat ansteuerbar
Art externer Positionierung	Analog, PWM, PPM, USB ³¹
Versorgungsspannung	7,4 V - 14,8 V

Vorteil dieses Gimbals ist neben dem geringen Gewicht, dass die Regelparameter für die Motoransteuerung vorkonfiguriert sind und der Gimbal somit ohne große Anpassung direkt verwendet werden kann. Sind Änderungen in der Gimbalkonfiguration nötig, können diese benutzerfreundlich über ein GUI vorgenommen werden. Die GUI bietet beispielsweise Einstellungsmöglichkeiten für:

- die Regelparameter,
- den maximalen Winkelausschlag der Achsen,
- die Spannungsüberwachung,
- den Livefeedback der Sensorwerte und
- die Firmware Updates der Controllerplatine usw.

Eine detailliertere Beschreibung des Funktionsumfangs des Gimbals bzw. der GUI ist dem User Manual zu entnehmen³². [elec]

Wie in Tabelle 7.2 dargestellt, kann der Gimbal über unterschiedliche Protokolle in der Ausrichtung gesteuert werden. In diesem Projekt wird die USB Schnittstelle verwendet, um die größtmögliche Flexibilität in der Gimbalansteuerung zu erreichen. Die USB Schnittstelle bietet als einzige die Möglichkeit, neben der Vorgabe der Soll-Positionen der Gimbalachsen, auch die aktuelle Lage der einzelnen Achsen zurück zu lesen.

Darüber hinaus ist es sinnvoll, die Versorgungsspannung des Gimbals auf 12 V zu begrenzen, um mögliche Hardwareschäden durch Überspannung zu vermeiden. Bei dieser

³⁰Händler: powerparts Zugriff: 02.10.2014

³¹Puls-Pausen-Modulation (PPM)

³²In dieser Arbeit wird die GUI- und Firmwareversion 2.40b7 genutzt.



Versorgungsspannung beträgt die Stromaufnahme ca. 500mA (messtechnisch ermittelt).

Folgendes Blockschaltbild (Abbildung 7.16) veranschaulicht die Schnittstellen zur Gimbalsteuerung, welche im weiteren Projektverlauf verwendet werden.

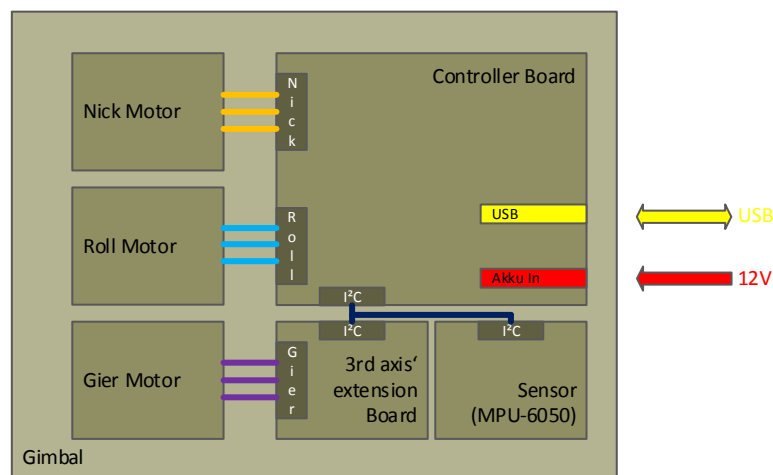


Abbildung 7.16: Blockschaltbild Gimbal

7.3.4 Auswahl der Telemetriestation

Aus der Anforderungsdefinition (siehe Kapitel 6) geht hervor, dass eine Telemetriestation benötigt wird, die den Videostream des AEVV-Systems sowie weitere Telemetriedaten empfangen und visualisieren kann.

Die Anforderungen mit Bezug zur Telemetriestation werden in Kapitel 6.2 und 6.4 beschrieben. Diese Anforderungen sind:

- NA007: Auflösung des Videostreams
- FT001: Bildschirm der Telemetriestation
- FT002: Empfangen der Daten
- FT003: Visualisierung der empfangenen Daten

Aus den Anforderungen FT001 und FT003 geht hervor, dass die Telemetriestation einen Bildschirm bereitstellen muss, auf dem der Videostream und weitere Telemetriedaten (Akkustand, Flughöhe, etc.) angezeigt werden. Da der Videostream laut NA007 mindestens eine WVGA-Auflösung aufweisen muss, muss auch die Telemetriestation in der Lage sein, den Videostream in gleicher Auflösung anzuzeigen.

Als mobile Telemetriestation bietet sich ein normaler Laptop an. Über den Anschluss von USB-Devices können nahezu alle üblichen Funkverbindungen aufgebaut werden. Über den Bildschirm lassen sich der Videostream und die Telemetriedaten anzeigen.



7.3.5 Funkverbindung zur Telemetriestation

Die Anforderungen an die Funkverbindung zwischen AEVV-System und Telemetriestation werden in Kapitel 6.2 und 6.4 beschrieben. Diese Anforderungen sind:

- NA007: Auflösung des Videostreams
- NA006: Bandbreite der Funkverbindung zur Telemetriestation
- FT002: Empfangen der Daten

In Kapitel 3.7 werden verschiedene drahtlose Übertragungsstandards gegenüber gestellt. Basierend auf dieser Voruntersuchung wird als Übertragungsstandard W-Lan 802.11n gewählt. Dieser Standard bietet mit 200 MBit/s eine ausreichende Übertragungsrate und mit 100 m eine ausreichende Reichweite.

An das in Kapitel 7.3.8 vorgestellte Carrierboard kann problemlos ein USB W-Lan 802.11n Device angeschlossen werden, da das Carrierboard eine USB-Schnittstelle bereitstellt. Die Verwendung der USB-Schnittstelle schränkt die Datenrate nicht weiter ein, da auf dem Carrierboard eine USB 2.0 Schnittstelle (480 MBit/s) zur Verfügung steht.

Bei einer hohen Videoauflösung müssen die Daten komprimiert werden, damit eine 200 MBit/s Datenrate ausreicht. Welcher Codec zur Kompression der Daten verwendet wird, wird im späteren Verlauf endgültig entschieden. Mit MPEG-2 lässt sich bspw. eine Auflösung von 1440×1080 Pixel (30 Frames/s) realisieren bei einer Datenrate von 80 MBit/s.

7.3.6 Multikopter

Dieses Unterkapitel beschreibt die Auswahl eines Multikopters unter Berücksichtigung der Traglast. Dabei muss der gewählte Multikopter die Anforderungen

- FM002: Multikopter fliegen
- NA004: Einhaltung der Nutzlast

einhalten.

Damit ein Multikopter fliegen kann, muss dieser genug Auftrieb erzeugen (siehe Kapitel 3.1.2). Der benötigte Auftrieb ist dabei proportional zu dem Gewicht des Multikopters. Dazu gehört sein Eigengewicht als auch die Traglast (FM002, NA004). Das zusätzliche Gewicht darf dabei die Flugdauer des Multikopters höchstens soweit reduzieren, dass diese weiterhin für die Durchführung des Szenarios (Halbzeit Robocup: 10 min [Com14]) ausreicht.

Das zusätzliche signifikante Gewicht, welches an den Multikopter montiert wird, setzt sich aus den in Tabelle 7.3 aufgelisteten Teilgewichten zusammen.



Tabelle 7.3: Zusammensetzung Traglast

Kamera	39 g
Gimbal	262 g
USB Hub ³³	40 g
Verteilerplatine (bestückt) ³³	120 g
Carrierboard (bestückt)	83 g
Logikakku	79 g
Zynq	24 g
Σ Traglast	647 g

Die zusätzliche Traglast m_{zusatz} , die der Multikopter zu heben hat, liegt somit bei ca. 647 g.

Im Hinblick auf das in Kapitel 1.4 vorgestellte Szenario und der zusätzlichen Traglast bietet sich die Verwendung eines Quadropters an. Dieser hat typischerweise eine höhere Schubkraft als ein Trikotter, wodurch der Quadropter größere Lasten tragen kann. Die o. g. Traglast ist jedoch noch nicht so groß, dass ein Hexakopter oder gar ein Oktokopter notwendig wäre. Des Weiteren sind Quadropters auf dem Markt am weitesten verbreitet und es gibt viele Informationen über solche Systeme.

Vom OFFIS wird der Projektgruppe ein Quadropter (Quadro XL von MikroKopter) zur Verfügung gestellt.

Dieser Quadropter m_{kopter} wiegt ohne Akku ca. 1 kg und das Gestell hat einen Durchmesser von ca. 60 cm. Als Flugakku ist ein 4S Lithium polymer Akku mit 6600 mAh verbaut, welcher weitere 715 g ($m_{flugakku}$) wiegt. Das Gewicht des Gesamtsystems m_{ges} beläuft sich somit auf:

$$m_{ges} = m_{zusatz} + m_{flugakku} + m_{kopter} \quad (7.5)$$

$$m_{ges} = 647 \text{ g} + 715 \text{ g} + 1000 \text{ g} = 2362 \text{ g} \quad (7.6)$$

Als Motoren kommen vier MK3638 Brushless Motoren zum Einsatz und als Propeller werden die 12x4,5 EPP-NY-Propeller genutzt.

³³Schätzwerte



Bei dem in Formel 7.5 errechnetem Gewicht muss jeder der vier Motoren eine Schub Th_N von

$$Th_N = \frac{m_{ges}}{N_{motor}} \quad (7.7)$$

$$Th_N = \frac{2362 \text{ g}}{4} \approx 590 \text{ g} \quad (7.8)$$

aufbringen.

Der Strom, den die Motoren benötigen um den o. g. Schub zu erzeugen, ist abhängig von den verwendeten Propellern. Diese unterscheiden sich bspw. in der Fläche sowie weichen und harten Materialien.

Durch die Größe der Propeller verändert sich die Menge der verdrängten Luft pro Umdrehung. Jedoch steigt das Gewicht mit der Größe der Propeller an und der Motor muss eine höhere Beschleunigungsleistung aufbringen. Kleine Propeller haben somit den Vorteil, dass sich der Quadrocopter agiler in der Luft bewegen kann, da das System nicht so träge ist. Die Motoren müssen jedoch schneller drehen (höhere Stromaufnahme) um den gleichen Schub zu erzeugen.

Weiche Propeller brechen nicht so schnell wie Propeller aus härten Materialien. Unter hoher Last verformen sich weiche Propeller jedoch schneller und fangen an zu schwingen. Dadurch sinkt der Wirkungsgrad und der Schub reduziert sich. Bei größeren Lasten sollte der Propeller also aus härterem Material bestehen.[Mikd]

Für die verwendeten Motortyp werden Kurven bereitgestellt, die einen mittleren Stromverbrauch eines Motors bei 16 V Versorgungsspannung ($U_{motor,16V}$) für unterschiedlichen Schub bei verschiedene Propellertypen aufzeigt [Mikc]. In der Abbildung 7.17 sind diese Kennlinien dargestellt. Für den 12x4,5 EPP-NY-Propeller ist keine Kennlinie verfügbar, daher wird als Referenz für die Motorstromaufnahme ein ähnlicher Propellertyp (APC12x3,8) herangezogen.

Wie in der Abbildung 7.17 zu erkennen, ist für einen Schub von 590 g bei den Propeller ein mittlerer Motorstrom von

$$I_{motor,16V} \approx 3,5 \text{ A}$$

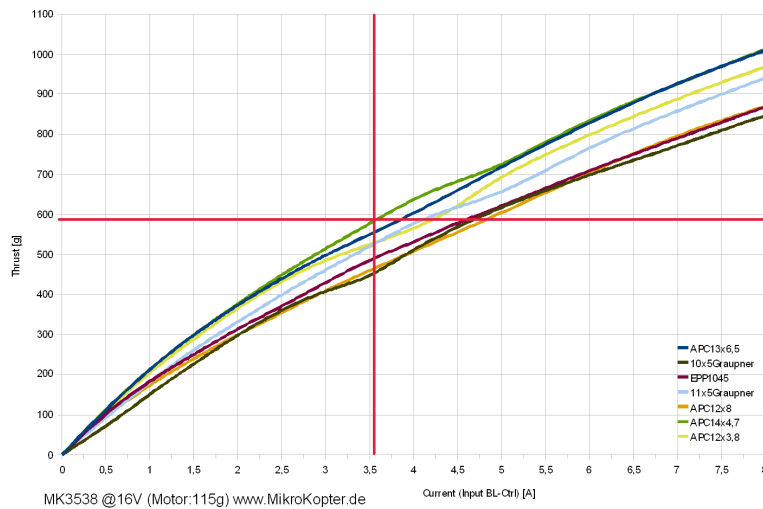


Abbildung 7.17: Motorströme unterschiedlicher Propeller bei unterschiedlichem Schub

zu erwarten. Bei der hier verwendeten Nennspannung U_{nenn} von 14,8 V liegt die Stromaufnahme eines Motors bei

$$\frac{U_{nenn}}{U_{motor,16V}} = \frac{I_{motor,16V}}{I_{motor}} \quad (7.9)$$

$$I_{motor} = \frac{I_{motor,16V} \cdot U_{motor,16V}}{U_{nenn}} \quad (7.10)$$

$$I_{motor} = \frac{3,5 \text{ A} \cdot 16 \text{ V}}{14,8 \text{ V}} \approx 3,8 \text{ A} \quad (7.11)$$

Mit Hilfe dieses Stroms und unter Berücksichtigung der anderen Verbraucher, die ebenfalls von dem Flugakku versorgt werden (siehe Kapitel 7.3.8), ist es möglich eine Flugzeit abzuschätzen. Die zusätzliche Hardware, welche an dem Flugakku angeschlossen ist und diesen signifikant belastet, ist der Gimbal. Der Gimbal hat bei 12 V (U_{gimbal}) eine Stromaufnahme (I_{gimbal}) von ca. 500 mA (siehe Kapitel 7.3.3). Daraus folgt, dass der Flugakku mit einem zusätzlichen Strom ($I_{zusatz,flug}$) von

$$\frac{U_{nenn}}{U_{gimbal}} = \frac{I_{gimbal}}{I_{zusatz,flug}} \quad (7.12)$$

$$I_{zusatz,flug} = \frac{I_{gimbal} \cdot U_{gimbal}}{U_{nenn}} \quad (7.13)$$

$$I_{zusatz,flug} = \frac{0,5 \text{ A} \cdot 12,0 \text{ V}}{14,8 \text{ V}} \approx 0,4 \text{ A} \quad (7.14)$$

belastet wird.

Somit liegt der Gesamtstrom des Flugakkus $I_{ges,flug}$ bei

$$I_{ges,flug} = I_{zusatz,flug} + I_{motor} \cdot N_{motor} \quad (7.15)$$

$$I_{ges,flug} = 0,4 A + 3,8 A \cdot 4 = 15,6 A. \quad (7.16)$$

Bei einer Akkukapazität des Flugakkus C_{flug} von 6600 mAh und 10% Verluste hervorgerufen durch Verwirbelungen oder weiteren Stromverbrauch von nicht berücksichtigten Bauteilen wie beispielsweise Isolatoren oder Motortreibern, liegt die Flugzeit t_{flug} bei ungefähr

$$t_{flug} = \frac{C_{flug}}{I_{ges,flug}} \cdot 90\% \quad (7.17)$$

$$t_{flug} = \frac{6,6 Ah}{15,6 A} \cdot 90\% \approx 0,38 h \approx 23 min. \quad (7.18)$$

23 min sind ausreichend für die Durchführung des in Kapitel 1.4 vorgestellten Szenarios. Folgendes Blockschaltbild Abbildung 7.18 veranschaulicht die Ansteuerung der Flugmotoren (siehe dazu Kapitel 7.3.8).

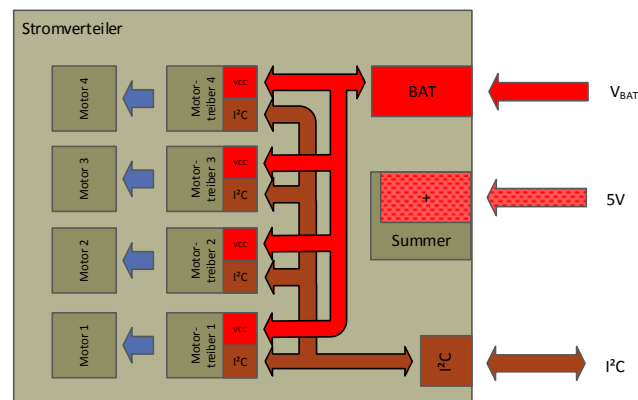


Abbildung 7.18: Blockschaltbild Multikopter



7.3.7 Auswahl der Sensoren

In den folgenden Abschnitten wird beschrieben, welche Sensoren für die Ermittlung der Fluglage ausgewählt und wie die Sensordaten für die Regelung aufbereitet werden. Die Sensoren sind Teil des ASGs und müssen daher den entsprechenden Anforderungen aus dem Kapitel 6.1 genügen. Diese Anforderungen sind:

- FM005: Bestimmung der Lage
- FM006: Bestimmung der Flughöhe
- FM010: Sensorwerte zur Bestimmung der Lage
- FM011: Sensorwerte zur Bestimmung der Flughöhe

Zur stabilen Regelung der Fluglage und der Höhe des Multikopters werden Sensoren benötigt, mit denen die momentane Fluglage und die Flughöhe erfasst werden können. Im Kapitel 3.2.4 wird beschrieben, dass zur Lageregelung Gyroskope und Beschleunigungssensoren fusioniert werden müssen. Des Weiteren ist zur Höhenregelung ein zusätzliches Barometer erforderlich.

7.3.7.1 Sensorik zur Erfassung der Lage

Während des Quadropterflugs muss zu jedem Zeitpunkt die aktuelle Fluglage, wie in Anforderung FM005 und FM010 gefordert, bestimmt werden.

Im Kapitel 3.2.4 wird beschrieben, dass zur Lagebestimmung die Drehrate um die drei Achsen über Gyroskope sowie die Beschleunigung in X-, Y- und Z-Richtung erfasst werden müssen.

Die Sensordaten, die zur Regelung des Quadropters genutzt werden, sind mit Störungen überlagert. Diese werden hauptsächlich von den hochfrequenten Motorströmen, die zu stark wechselnden Magnetfeldern führen, verursacht. Diese Magnetfelder stören die Elektronik in den Sensor-ICs³⁴. Bei einer digitalen Übertragung der Sensorwerte (z. B. via I²C-Bus) ist die Übertragungstrecke als solche robuster gegenüber elektromagnetischen Störeinflüssen als bei einer analogen Datenübertragung. Aus diesem Grund bietet sich die Verwendung von Sensoren mit digitalem Ausgang an.

Besonders vorteilhaft ist die Verwendung von integrierten Sensorsystemen bei denen 3-Achs-Beschleunigungssensoren und -Gyroskope bereits auf einem Chip kombiniert sind. Bei diesen Systemen reduziert sich zum einen der Verdrahtungsaufwand erheblich, da z. B. bei einer I²C-Schnittstelle zwei Leitungen ausreichen um eine Vielzahl an Sensorwerten abfragen zu können. Außerdem entfallen sehr viele Arbitrierungsvorgänge, da ein Businterface des Chips ausreicht. Somit kann eine schnelle und unkomplizierte Übertragung der Sensorwerte gewährleistet werden.

Zur Erfassung der Fluglage wird der **MPU9150** von "Invensense" verwendet. Auf diesem Chip sind ein 3-Achsen-Beschleunigungssensor, ein 3-Achsen-Gyroskop und ein 3-Achsen-

³⁴Integrated Circuit (IC)



Kompass integriert. Zur Berechnung der Fluglage sind allerdings nur die Beschleunigungssensoren und die Gyroskope erforderlich. Der Kompass kann genutzt werden, um die absolute Orientierung des Quadropters gegen geodätisch Nord zu erfassen.

Zeitverhalten MPU9150 Aus den Echtzeitanforderungen (siehe 6.8) ergibt sich, dass eine Aktualisierung der Sensordaten für die Lageregelung mit mindestens 500 Hz erfolgen muss.

Der MPU9150 stellt eine I²C-Schnittstelle zur Verfügung, welche mit maximal 400 kHz (fast-mode) betrieben werden kann. Allerdings lässt sich aus dieser Angabe nur grob abschätzen, wie viel Zeit die Übertragung der Sensorwerte wirklich in Anspruch nimmt, da bei I²C nach jeder Übertragung auf ein ACK gewartet wird (siehe 6.8.1.1). Zur Bestimmung der tatsächlichen Übertragungszeit werden die Sensoren daher an einem ATmega32 Mikrocontroller von Atmel getestet und die Übertragungszeit messtechnisch ermittelt. Hierzu wurde ein Pin, an den auch das Oszilloskop angeschlossen ist, zu Beginn der I²C-Sequenz auf High und bei Fertigstellung wieder auf Low gesetzt. Der I²C-Bustransfer für die zur Lageregelung benötigten Daten dauerte bei diesem Test ca. 500 μ s. Die Übertragungsgeschwindigkeit reicht demnach aus um eine 500 Hz Regelung realisieren zu können.

Wie häufig die Sensordaten im MPU9150 intern aktualisiert werden, lässt sich entweder in den entsprechenden Steuerregistern konfigurieren oder ist auf einen festen Werte eingestellt. Bei den Beschleunigungssensoren ist der Wert fest (nicht konfigurierbar) und beträgt 1 kHz. Die Samplerate der Gyroskope hingegen ist konfigurierbar und kann maximal 8 kHz betragen.

Berechnung der Lage Die Lage des Quadropters lässt sich sowohl aus den Werten der Beschleunigungssensoren als auch aus den Werten der Gyroskope bestimmen. Beides hat aber gravierende Nachteile, die im Folgenden näher beschrieben werden.

Die Lagebestimmung aus den Werten der Beschleunigungssensoren gestaltet sich, für ein ruhendes oder sich gleichförmig bewegendes System, vergleichsweise einfach. In diesem Fall wirkt ausschließlich die Erdbeschleunigung auf das System. Aus den gemessenen Beschleunigungswerten in X-, Y- und Z-Richtung kann die relative Richtung der wirkenden Erdbeschleunigung und somit auch die Lage des Quadropters in der Luft bestimmt werden.

Allerdings lässt sich aus diesen Werten nicht mehr sinnvoll die Lage bestimmen, wenn das gesamte System zusätzlich zur Erdbeschleunigung eine weitere Beschleunigung erfährt. Dies tritt bei jeder Bewegungsänderung auf. Ob dieser Fall gerade Eintritt lässt sich rechnerisch bestimmen. Dazu wird die wirkenden Beschleunigungen vektoriell aufaddiert und die insgesamt wirkende Beschleunigung betragsmäßig mit der Erdbeschleunigung verglichen. Wird diese stark über- oder unterschritten ist eine Bestimmung der Lage aus den

Beschleunigungssensorwerten nicht mehr möglich.

Auch aus den Sensorwerten der Gyroskope lässt sich die Lage des Quadropters berechnen. Wie im Kapitel 3.1.3 beschrieben, werden über die Gyroskope die Winkelgeschwindigkeiten um jeweils eine bestimmte Achsen gemessen. Nach einer anfänglichen Initialisierung der Null-Lage, werden die Gyroskopwerte integriert, um von den Winkelgeschwindigkeiten auf die relative Winkeländerung zur Null-Lage schließen zu können. Allerdings wird das Signal zur digitalen Signalverarbeitung in diskreten Zeitschritten quantisiert. Durch die Quantisierung entsteht ein Fehler, der immer mit integriert wird. Zudem wird angenommen, dass zwischen den Abtastzeitpunkten der Wert konstant bleibt, was ebenfalls nicht der Realität entspricht. Da sich diese Fehler durch die Integration aufsummieren, kann sich die ermittelte Lage von der tatsächlichen Fluglage entfernen. Dieser Effekt wird Drift genannt.

7.3.7.2 Sensorik zur Erfassung der Flughöhe

Während des Quadroptersfluges muss zu jedem Zeitpunkt die aktuelle Flughöhe, wie in Anforderung FM006 und FM011 gefordert, bestimmt werden.

Ein Luftdrucksensor misst den Luftdruck in der Umgebung, in der er sich befindet (siehe Kapitel 3.1.3). Mit zunehmender Höhe sinkt der Luftdruck. Bei einer statischen Höhe ist allerdings zu berücksichtigen, dass sich der Luftdruck darüber hinaus auch wetterabhängig ändern kann.³⁵

Eine Höhenänderung des Quadropters kann daher durch messen des Luftdrucks erfasst (siehe Kapitel 3.2.4) und die Höhe des Quadropters auf Basis dieser Sensorwerte geregelt werden.

Wie oben beschrieben, bietet die Verwendung von Sensoren mit digitalem Ausgang viele Vorteile, weshalb auch ein Barometer mit digitalem Ausgang gewählt wird. Genutzt wird der **BMP085** von Bosch Sensortec GmbH. Dieser Sensorchip stellt ebenfalls eine I²C-Schnittstelle zur Verfügung.

7.3.7.3 Zeitverhalten BMP085 In Tabelle 7.4 sind die Wandlungszeiten zur Bestimmung des Luftdruckes in verschiedenen Modes dargestellt, die der BMP085 zur Verfügung stellt. Die höchste Auflösung wird im "ultra high resolution"-Mode erreicht, bei dem 8 Messwerte verrechnet werden. Allerdings beträgt die Wandlungszeit im "ultra high resolution"-Mode 25.5ms und ist damit am längsten. Da jedoch für die Regelung der Flughöhe eine träge Regelung ausreicht, ist dieser Wert unkritisch.

³⁵Da das Szenario in einer Sporthalle stattfindet, werden Wettereinflüsse in diesem Projekt vernachlässigt.

Tabelle 7.4: BMP085 Modes

Mode	Parameter oversampling_setting	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [μ A]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

Jede Luftdruckmessung muss bei dem BMP085 einzeln gestartet werden. Dies geschieht über eine I²C-Nachricht, mit der entsprechende Bits in einem Control-Register gesetzt werden. Das Senden der Start-Nachricht dauert 80 μ s. Das Auslesen der Messwerte, nachdem die Messung abgeschlossen wurde, dauert weitere 180 μ s. Diese Zeiten wurden ebenfalls messtechnisch mit dem ATmega-Controller erfasst.

7.3.8 Verteilerplatine

Um den Prozessor mit dem Fluggerät zu verbinden, wird dieser über ein sog. Carrierboard (TE0703-04) der Firma Trenz Electronics auf einer Verteilerplatine angebracht. Diese Verteilerplatine muss die Anforderungen

- FM003: Empfangen von Steuerbefehlen
- FM005: Bestimmung der Lage
- FM006: Bestimmung der Höhe
- FM009: Motorstellwerte weitergeben
- FM012: Sensorwerte zur Bestimmung des Akkustandes
- FM015: Warnung bei kritischem Akkustand
- FA001: Bildaufnahme aktivieren und deaktivieren
- FA003: Kamerabild bereitstellen
- FA005: Funkverbindung zwischen AEVV-System - Telemetriestation
- FA008: Positionierung der Kamera

erfüllen, bzw. die entsprechenden Hardwarebeschaltung bereitstellen. Das in Abbildung 7.19 gezeigte Blockschaltbild stellt den Aufbau der Verteilerplatine dar.

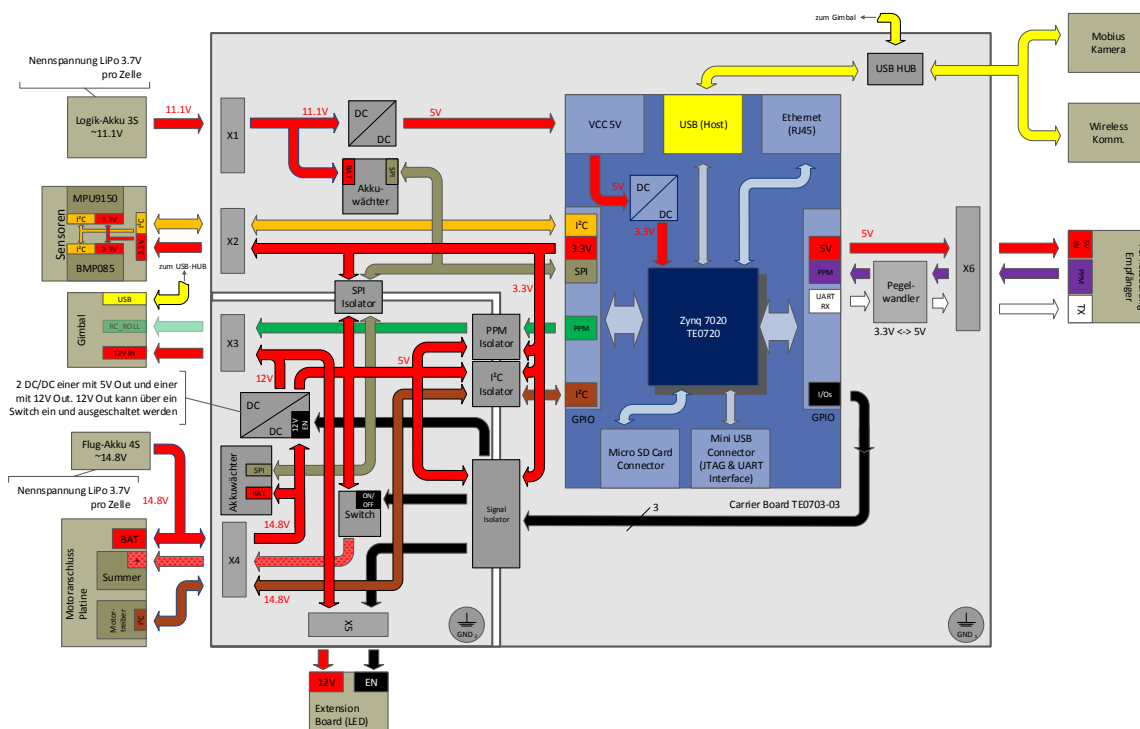


Abbildung 7.19: Blockschaltbild Verteilerplatine

Wie in Kapitel 7.3.1 vorgestellt, wird in dieser Arbeit als Prozessorplattform ein Zynq 7020 eingesetzt. Diese Prozessorplattform wird auf ein Carrierboard TE0703-04 der Firma



Trenz Electronics integriert. Das Carrierboard führt die GPIO Anschlüsse des Zynqboards heraus und stellt unter anderem einen USB-Anschluss zur Verfügung [Elea]. Das Verteilerboard integriert wiederum das Carrierboard, um die GPIOs für weitere Peripherie (z. B. die Sensoren) bereitzustellen.

Neben dem Quadrocopter wird ebenfalls eine Funkfernbedienung (Graupner MC-20) vom OFFIS zur Verfügung gestellt, welche mit dem Graupner GR16 Transceiver kommuniziert. Diese Kombination erlaubt eine bidirektionale Verbindung zwischen Quadrocopter und Funkfernbedienung. Zum Anschluss des Empfängers steht die Klemme X6 bereit, welche die Signal weiter an den Prozessor leitet (FM003, FM015).

Die Flughöhe und die aktuelle Lage des Quadrocopters werden über Sensoren bestimmt (siehe Kapitel 7.3.7). Diese Sensoren werden an Klemme X2 angeschlossen und kommunizieren über I²C (FM005, FM006).

Auf dem Quadrocopter befinden sich zur Ansteuerung der Motoren vier sog. BL-Ctrl, welche ebenfalls über I²C kommunizieren. Diese sind an einem Stromverteiler angeschlossen. Neben den I²C Signalen schleift die Stromverteilerplatine [Mikb] ein weiteres Signal zu einem Summer durch, welcher z. B. für ein Akkuwarnsignal verwendet werden kann. Diese Signale sowie die Flugakkuspannung werden an Klemme X4 angeschlossen (FM009).

Die Spannungsversorgung des Quadrocopters ist in zwei Teilbereiche unterteilt, damit mögliche Spannungsschwankungen durch hohe Stromspitzen (z. B. durch die Flugmotoren hervorgerufen) nicht den Prozessor beeinflussen. Die zwei Teilbereiche sind galvanisch voneinander getrennt und in der Abbildung 7.19 mit GND_1 und GND_2 gekennzeichnet. Teilbereich GND_1 wird von dem Logikakku versorgt und der Teilbereich GND_2 vom Flugakku. Der Logikakku ist an Klemme X1 angeschlossen und seine Spannung wird, wie der Flugakku (Klemme X4), mit Hilfe eines Spannungswächters überwacht (FM012). Der Logikakku besitzt eine Kapazität C_{logik} von 1100 mAh bei einer Nennspannung U_{flug} von 11,1 V. Der Logikakku wird signifikant von dem Prozessor inkl. Carrierboard, dem WLAN-Stick und der Mobius Kamera belastet. Die Gesamtstromaufnahme $I_{ges,komponenten}$ der Komponenten liegt damit bei

$$I_{ges,komponenten} = I_{carrier} + I_{WLAN} + I_{Mobius} \quad (7.19)$$

$$I_{ges,komponenten} = 2000 \text{ mA} + 100 \text{ mA} + 140 \text{ mA} \approx 2240 \text{ mA}. \quad (7.20)$$

Diese Geräte arbeiten alle im 5 V Bereich $U_{Komponenten}$. Die Nennspannung des Logikakkus wird dazu mit einem Step-Down-Wandler in den 5 V Spannungsbereich transformiert.

Dabei wird von einem Wirkungsgrad η_{DCDC} von 80% ausgegangen. Somit liegt die Stromaufnahme I_{logik} im 11,1 V Bereich des Akkus bei

$$\frac{U_{komponenten}}{U_{logik}} = \frac{I_{logik} \cdot \eta_{DCDC}}{I_{ges,komponenten}} \quad (7.21)$$

$$I_{logik} = \frac{U_{komponenten} \cdot I_{ges,komponenten}}{U_{logik} \cdot \eta_{DCDC}} \quad (7.22)$$

$$I_{logik} = \frac{5 V \cdot 2,24 A}{11,1 V \cdot 80\%} \approx 1,26 A. \quad (7.23)$$

Daraus folgt eine Flugzeit t_{flug} von

$$t_{flug} = \frac{C_{logik}}{I_{logik}} \cdot 90\% \quad (7.24)$$

$$t_{flug} = \frac{1,1 Ah}{1,26 A} \cdot 90\% \approx 0,87 h \approx 52 min. \quad (7.25)$$

Es wird von 10% Verlusten ausgegangen, die durch den nicht berücksichtigten Stromverbrauch der Isolatoren und der Versorgung des Empfängers entstehen.

Diese in Formel 7.24 ermittelte Flugzeit ist höher als die, welche im Kapitel 7.3.6 in der Formel 7.17 berechnet wurde und beschränkt daher die Flugdauer nicht weiter (unter der Bedingung, dass beide Akkumulatoren voll aufgeladen sind).

Der Gimbal wird über USB (siehe Kapitel 7.3.3) angesteuert. Dazu wird ein USB Kabel mit integriertem Isolator zwischen Gimbal und USB Hub verwendet. Die Klemme X3 stellt weitere Hardwareinterfaces für den Gimbal bereit – einen schaltbaren 12 V Ausgang zur Versorgung des Gimbals und einen PPM-Anschluss zur alternativen Gimbalsteuerung (FA008, FA001).

Ein Extensionboard [Mikb] kann an Klemme X5 angeschlossen werden. An dieser Klemme stehen 12 V Versorgungsspannung und ein Enable-Signal zur Verfügung. Das Extensionboard kann z. B. LEDs ansteuern, um einen kritischen Akkustand anzuzeigen.

Zum Anschluss der Kamera (siehe Kapitel 7.3.2) und der Wireless Kommunikation zur Telemetriestation (siehe Kapitel 7.3.5) steht ein USB-Hub zur Verfügung. Dieser ist mit der USB Schnittstelle auf dem Carrierboard verbunden (FA003, FA005).

7.4 Plattformdesign

In diesem Kapitel wird die Plattform beschrieben, auf der die Software später ausgeführt werden soll. Da auf dem Zynq ein FPGA zur Verfügung steht, kann eine Plattform zum

Großteil frei entwickelt werden. Für das Plattformdesign wird Xilinx Vivado verwendet. Mit Hilfe dieser Software werden Hardwarekomponenten ins Plattformdesign integriert, die in einer Bibliothek von Xilinx zur Verfügung gestellt werden. Das Plattformdesign wird in diesem Kapitel nur an Hand von vereinfachten Grafiken erläutert. Das vollständige Design als Vivado-Modell ist im Anhang zu finden (s. Abbildung A.5, Abbildung A.6).

7.4.1 Rechen- und Speicherkomponenten

In diesem Abschnitt werden die Rechen- und Speicherkomponenten und deren Verknüpfung untereinander im Plattformdesign erläutert. Der grundlegende Aufbau ist in Abbildung 7.20 zu sehen. Das vollständige Design befindet sich zur Ansicht im Anhang (s. Abbildung A.5, Abbildung A.6).

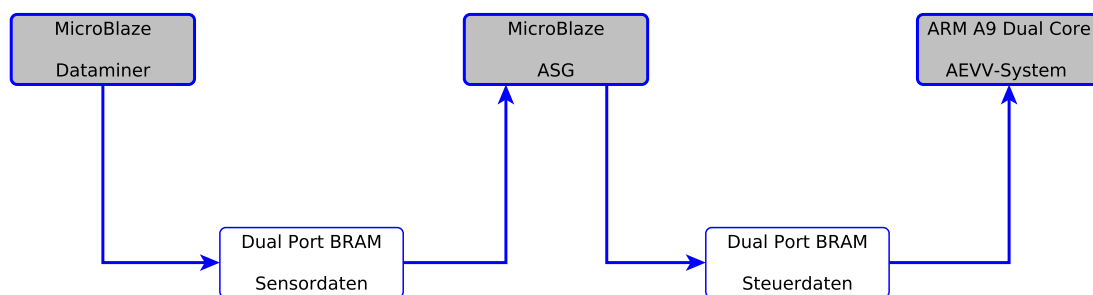


Abbildung 7.20: Anordnung der Rechen- und Speicherkomponenten im Plattformdesign

Bei den grau hinterlegten Hardwarekomponenten handelt es sich um Rechenkomponenten. Wie am oberen Rand der Hardwarekomponenten zu lesen handelt es sich um zwei MicroBlaze Softcores und einen ARM Cortex A9 Dual Core Prozessor. Am unteren Rand der Hardwarekomponenten steht ein beschreibender Name für die jeweilige Rechenkomponente. Die weiß hinterlegten Hardwarekomponenten sind Speicherkomponenten. Es handelt sich um zwei Dual Port BRAMs in denen Sensordaten bzw. Steuerdaten gespeichert werden. Die Hardwarekomponenten sind durch Datenleitungen verbunden, durch die Daten in die durch die Pfeile markierten Richtungen übertragen werden können.

Die Namen der drei Rechenkomponenten geben Hinweis auf ihre Funktion:

Bei der **AEVV-System-Hardwarekomponente** handelt es sich um einen ARM Cortex A9 Dual Core Prozessor auf dem das AEVV-System ausgeführt wird. Der Prozessor ist u. a. mit der NEON-Technologie versehen. Hierbei handelt es sich um Schaltungen, die das Ausführen von Multimediainhalten beschleunigen, was einen großen Mehrwert für das AEVV-System bringt.

Die **ASG-Hardwarekomponente** ist ein MicroBlaze Softcore, der in der programmierbaren Logik implementiert ist. Er ist relativ leistungsstark ausgelegt. Er verfügt u. a. über eine Floating-Point-Unit, einen Hardwaremultiplizierer, einen Hardwaredividierer und 128kByte Speicher. Auf ihm wird das ASG ausgeführt.

Die **Dataminer-Hardwarekomponente** ist ebenfalls ein MicroBlaze Softcore. Er ist identisch zur ASG-Hardwarekomponente konfiguriert aber auf ihm wird die Datenerfassung von den Sensoren ausgeführt.

7.4.2 Sensoren und Aktoren

In diesem Abschnitt wird erläutert, welche Sensoren und Aktoren im Design integriert sind und wie diese mit den Rechen- und Speicherkomponenten zusammenhängen. Der Aufbau ist in Abbildung 7.21 dargestellt. Jeder Aktor und Sensor ist dabei an einem Treiberbaustein angeschlossen, der in der Abbildung nicht dargestellt ist. Das vollständige Design befindet sich zur Ansicht im Anhang (s. Abbildung A.5, Abbildung A.6).

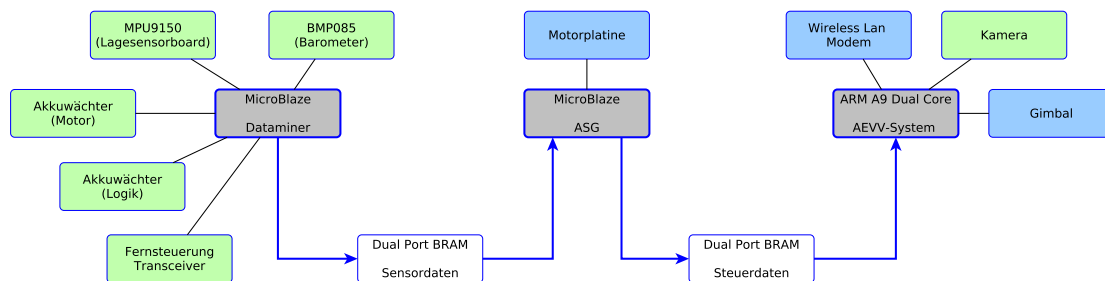


Abbildung 7.21: Anordnung der Sensoren und Aktoren im Plattformdesign

Bei den grün hinterlegten Komponenten handelt es sich um Sensoren und bei den blau hinterlegte um Aktoren oder Eingabegeräten. Alle Sensoren, die für die Lageregelung, also für die ASG Komponente, wichtig sind, werden an die Dataminerkomponente angeschlossen. Die Daten können dann in die Sensordata-BRAM-Komponente geschrieben werden. Von dort können sie durch die ASG-Komponente gelesen werden. Diese kann die von ihr errechneten Motorstellwerte direkt an die Motorplatine weitergeben. Außerdem kann sie Steuerdaten in die Kommunikation-BRAM-Komponente schreiben. Aus dieser kann die AEVV-System-Komponente die Steuerdaten abrufen, um entsprechend auf diese zu reagieren. Sie kann mit denen an ihr angeschlossenen Peripheriegeräten alle an sie gestellten Anforderungen erfüllen. Es folgt eine tabellarische Auflistung der Aktor- und Sensorkomponenten inklusive einer kurzen Beschreibung ihrer Aufgaben.



Komponente	Typ	Schnittstelle	Aufgabe
BMP085	Sensor	I2C	Sensor liefert Luftdruck, Temperatur und Kompasswert.
MPU9150	Sensor	I2C	Sensor liefert Beschleunigungswerte und Drehrate.
Akkuwächter (Energie)	Sensor	SPI	Sensor überwacht die Spannung der Energieversorgung der Motoren.
Akkuwächter (Logik)	Sensor	SPI	Sensor überwacht die Spannung der Energieversorgung der Logik.
Fernsteuerung Transceiver	Eingabegerät	PPM/Uart	Fernbedienung auf der die Steuerbefehle für den Quadrocopter aufgenommen werden. Darüber hinaus wird der Akkustand des Quadrocopters auf der Fernbedienung angezeigt.
Kamera	Sensor	USB	Kamera mit der es möglich ist das Spielgeschehen aufzunehmen. Mit Hilfe des Kamerabildes wird auch die Objektverfolgung realisiert.
Motorplatine	Aktor	I2C	An der Motorplatine sind die vier Motoren des Quadrocopters angeschlossen. Über sie wird die Leistung gesteuert, die jeder Motor erhält.
Wireless Lan Modem	Aktor	USB	Mit Hilfe des Wireless Lan Modems wird das Kamerabild und die Telemetriedaten an die Bodenstation gesendet.
Gimbal	Aktor	USB	Der Gimbal bewegt die Kamera, so dass das gewünschte Objekt immer im Bild bleibt.

7.4.3 Blockdesign in Vivado

In der nachfolgenden Tabelle 7.5 sind alle im Vivado-Blockdesign enthaltenen Blöcke mit der entsprechenden Komponente aus dem vereinfachten Design in Abbildung 7.21 und einer Beschreibung aufgeführt.



Tabelle 7.5: Vivado Komponenten

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
aevv_system	ARM A9 Dual Core AEVV-System	Dieser Block repräsentiert den ARM Zweikernprozessor, auf dem das AEVV-System ausgeführt wird.
net_gnd	ARM A9 Dual Core AEVV-System	Dieser Block zieht die SDIO0_CDN und SDIO1_CDN Pins (gehören zur Kommunikation mit der SD-Karte) des AEVV-Systems auf Masse.
rst_processing_system7_0_100M	-	Führt den Reset von folgenden Komponenten durch: Lokale Speicher der MicroBlaze Softcores, AXI Interconnects, alle Peripherie-Komponenten, ARM Cores und des MicroBlaze Debug Module.
aevv_system_axi_periph	ARM A9 Dual Core AEVV-System	AXI Interconnect zwischen dem AEVV-System und den am FPGA angeschlossenen Komponenten (Debug Led, Bram Controller, Timer, Debug Pins) ³⁶ .
axi_bram_ctrl_aevv	Verbindung zwischen Dual Port BRAM Steuerdaten und AEVV-System	Dieser BRAM Controller regelt den Zugriff des AEVV-Systems auf den lokalen Block RAM von AEVV-System und ASG und legt u. a. den Adressbereich fest.

³⁶Das WLAN Modul, die Kamera und der Gimbal sind hier nicht genannt, da sie direkt über die USB Schnittstelle am AEVV-System, also dem ARM Core angeschlossen sind.

Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
<code>blk_mem_gen_asg_aevv</code>	Dual Port BRAM Steuerdaten	Dieser Block stellt lokalen Speicher für das AEVV-System und das ASG zur Verfügung und bietet Ports an, über die geschrieben und gelesen werden kann. An den Ports werden die BRAM Controller angeschlossen.
<code>xlconstant_write_disabled_aevv</code>	Dual Port BRAM Steuerdaten	Es handelt sich bei diesem Block um eine Konstante, die den schreibenden Zugriff des AEVV-Systems auf den Block RAM zwischen ASG und AEVV-System verhindert.
<code>axi_timer_aevv</code>	ARM A9 Dual Core AEVV-System	Dieser Block stellt dem AEVV-System zwei 32 Bit-Timer zur Verfügung.
<code>axi_gpio_debug_led_aevv</code>	ARM A9 Dual Core AEVV-System	Dieser Block stellt dem AEVV-System eine Debug Led über GPIO zur Verfügung. Der entsprechende Pin kann in Tabelle 9.1 nachgelesen werden.
<code>axi_gpio_debug_pins_aevv</code>	ARM A9 Dual Core AEVV-System	Dieser Block stellt dem AEVV-System zwei Debug Pins über GPIO bereit. Die entsprechenden Pins können in Tabelle 9.1 nachgelesen werden.

Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
mb_asg	MicroBlaze ASG	Es handelt sich bei dieser Komponente um den in Logik implementierten 32 Bit-Prozessor, einen sogenannten Softcore. Auf diesem wird das ASG ausgeführt, also die Regelungsalgorithmen, um den Quadropter während des Fluges stabil zu halten und steuern zu können.
mb_asg_local_memory	MicroBlaze ASG	Lokaler Speicher für das ASG, in dem das auszuführende Programm liegt.
mb_asg_axi_periph	Verbindung zwischen Motorplatine und MicroBlaze ASG	AXI Interconnect zwischen dem ASG und den dort angeschlossenen Komponenten.
axi_bram_ctrl_asg_to_aevv	Verbindung zwischen Dual Port BRAM Steuerdaten und ASG	Dieser BRAM Controller regelt den Zugriff des ASG-Systems auf den lokalen Block RAM von AEVV-System und ASG und legt u. a. den Adressbereich fest.
axi_iic_motorboard	Verbindung zwischen Motorplatine und MicroBlaze ASG	Der AXI IIC Controller bietet die Schnittstelle zwischen den Motoren und dem MicroBlaze, auf dem das ASG implementiert ist.



Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
axi_timer_asg	MicroBlaze ASG	Dieser Block stellt dem ASG zwei 32 Bit-Timer zur Verfügung.
axi_gpio_debug_led_asg	MicroBlaze ASG	Dieser Block stellt dem ASG eine Debug Led über GPIO zur Verfügung. Der entsprechende Pin kann in Tabelle 9.1 nachgelesen werden.
axi_gpio_gimbal_ppm	-	Veralteter GPIO IP Core, der für das Steuern der Gimbalfunktion gedacht war. Kann für andere erweiterte Funktionen verwendet werden.
axi_gpio_debug_pins_asg	MicroBlaze ASG	Dieser Block stellt dem ASG zwei Debug Pins über GPIO bereit. Die entsprechenden Pins können in Tabelle 9.1 nachgelesen werden.
axi_bram_ctrl_asg_to_dataminer	Verbindung zwischen Dual Port BRAM Sensordaten und ASG	Dieser BRAM Controller regelt den Zugriff des ASGs auf den lokalen Block RAM von Dataminer und ASG und legt u. a. den Adressbereich fest.



Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
blk_mem_gen_asg_dataminer	Dual Port BRAM Sensordaten	Dieser Block stellt lokalen Speicher für den Dataminer und das ASG zur Verfügung und bietet Ports an, über die geschrieben und gelesen werden kann. An den Ports werden die BRAM Controller angeschlossen.
xlconstant_write_disabled_asg	Dual Port BRAM Sensordaten	Es handelt sich bei diesem Block um eine Konstante, die den schreibenden Zugriff des ASG auf den Block RAM zwischen Dataminer und ASG verhindert.
mb_dataminer	MicroBlaze Dataminer	Es handelt sich bei dieser Komponente um den in Logik implementierten 32 Bit-Prozessor, einen sogenannten Softcore. Auf diesem wird der Dataminer ausgeführt, also das zyklische Abfragen der einzelnen Sensoren und die Kommunikation mit der Fernsteuerung.
mb_dataminer_local_memory	MicroBlaze Dataminer	Lokaler Speicher für den Dataminer, in dem das auszuführende Programm liegt.



Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
<code>mb_dataminer_axi_periph</code>	Verbindungen zwischen Barometer, Lagesensorboard, Akkuwächtern, Fernsteuerung und dem Dataminer	AXI Interconnect zwischen dem Dataminer und den dort angeschlossenen Komponenten.
<code>mdm_dataminer</code>	-	MicroBlaze Debug Module, welches zum Debuggen des MicroBlaze benötigt wird. Unter anderem stellt dieser Block eine JTAG Schnittstelle für Software Debug Tools bereit und eine JTAG basierte UART Schnittstelle.
<code>axi_bram_ctrl_dataminer</code>	Verbindung zwischen Dual Port BRAM Sensordaten und Dataminer	Dieser BRAM Controller regelt den Zugriff des Dataminers auf den lokalen Block RAM von ASG und Dataminer und legt u. a. den Adressbereich fest.
<code>axi_intc_dataminer</code>	MicroBlaze Dataminer	Generiert die von der angeschlossenen Peripherie (hier der Dataminer Timer) ausgelösten Interrupts für den verbundenen MicroBlaze (hier der MicroBlaze Dataminer).
<code>axi_timer_dataminer</code>	MicroBlaze Dataminer	Dieser Block stellt dem Dataminer zwei 32 Bit-Timer zur Verfügung.

Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
axi_spi_battery_guard	Verbindungen zwischen den Akkuwächtern (Logik + Motor) und dem MicroBlaze Dataminer	SPI Schnittstelle zwischen den Akkuwächtern (Slaves) und dem MicroBlaze Dataminer (Master).
axi_iic_mpu	Verbindung zwischen dem MPU9150 (Lagesensorboard) und dem MicroBlaze Dataminer	IIC Schnittstelle zwischen dem Lagesensorboard und dem MicroBlaze Dataminer.
axi_iic_bmp	Verbindung zwischen dem BMP085 (Barometer) und dem MicroBlaze Dataminer	IIC Schnittstelle zwischen dem Barometer und dem MicroBlaze Dataminer.
axi_uartlite_remote_control	Verbindung zwischen Fernsteuerung Transceiver und MicroBlaze Dataminer	UART Schnittstelle für die bidirektionale Kommunikation vom MicroBlaze Dataminer zur Fernsteuerung, um die Telemetriedaten auf der Fernsteuerung empfangen und darstellen zu können.
ppm_interpreter_remote_control	Verbindung zwischen Fernsteuerung Transceiver und MicroBlaze Dataminer	PPM Schnittstelle für die Kommunikation von der Fernsteuerung zum MicroBlaze Dataminer, um die Eingabedaten empfangen und verarbeiten zu können.
axi_gpio_debug_leds_dataminer	MicroBlaze Dataminer	Dieser Block stellt dem Dataminer zwei Debug Leds über GPIO zur Verfügung. Die entsprechenden Pins können in Tabelle 9.1 nachgelesen werden.



Tabelle 7.5: Vivado Komponenten (Fortsetzung)

Name des Blocks in Vivado	Entsprechende Komponente in Abbildung 7.21	Beschreibung
axi_gpio_toggle_peripheral	MicroBlaze Dataminer	Dieser Block erlaubt das Ein- und Ausschalten des Extension Boards für die Leds, des Summers für akkustische Warnsignale und des Gimbals über GPIO.
axi_gpio_debug_pins_dataminer	MicroBlaze Dataminer	Dieser Block stellt dem Dataminer vier Debug Pins über GPIO bereit. Die entsprechenden Pins können in Tabelle 9.1 nachgelesen werden.
xadc_dataminer	MicroBlaze Dataminer	Der XADC Block erlaubt das Auslesen der Temperatur und der Versorgungsspannung des TE0720 Boards.

7.4.4 Mixed-Criticality und Safety-Critical Aspekte im Plattformdesign

In diesem Abschnitt werden Aspekte des Plattformdesigns beleuchtet, die die allgemeine funktionale Sicherheit verbessern oder die die Sicherheit in Bezug auf Mixed-Criticality verbessern. Im zweiten Teil wird ein Ausblick gegeben, welche Sicherheitsmaßnahmen darüber hinaus wirksam angewendet werden könnten, aber nicht umgesetzt wurden.

7.4.4.1 Realisierte Mixed-Criticality Aspekte Es stehen verschiedenen Berechnungskomponenten für den kritischen und nicht kritischen Teil der Anwendung zur Verfügung. So ist gewährleistet, dass Cache und RAM Speicher der Prozessoren getrennt sind. Es herrscht größtenteils eine Unabhängigkeit der Daten der beiden Teile. Alle Daten, die für beide Systeme relevant sind, können durch den Dual Port BRAM "Steuerdaten" ausgetauscht werden. Hierbei sind beide Berechnungskomponenten dem BRAM gegenüber Busmaster, allerdings kann die eine Berechnungskomponente nur Schreibzugriffe, die andere nur Lesezugriffe einleiten. Der Dual Port BRAM erlaubt gleichzeitiges Lesen und



Schreiben, sodass keine der Rechenkomponenten warten muss. Darüber hinaus kann es durch die zwei Rechenkomponenten nicht dazu kommen, dass der kritische Teil ausfällt, wenn der nicht kritische Teil fehlerbedingt nicht mehr weiterläuft.

Es ist keine Kommunikation vom nicht kritischen Teil zum kritischen Teil notwendig, da alle Sensoren, die vom kritischen Teil benötigt werden, im kritischen Teil der Architektur angeschlossen werden können. So können keine Sensordaten durch unnötige Weiterleitungen verfälscht werden.

7.4.4.2 Ausblick auf Mixed-Criticality Aspekte Zukünftig können Watchdogs implementiert werden. Diese könnten das schreiben von Daten überwachen, sodass sichergestellt ist, dass Daten zu den richtigen Zeitpunkten geändert werden. Am sinnvollsten wären diese Watchdogs entlang des Datenpfades von der Aufnahme der Sensordaten, bis zur Regelung der Motoren für die Flugregelung. Hier müsste jeder Datenzugriff mit einem Watchdog versehen werden, um sicherzustellen, dass die entsprechenden Daten rechtzeitig geschrieben werden.

Um die Korrektheit der Daten sicherzustellen könnten Sensoren redundant verbaut werden. Durch einen Voting-Algorithmus können dann Fehler in einzelnen Sensoren ausgeglichen werden.

7.5 Mapping

Beim Mapping wird in erster Linie die Überführung der identifizierten Komponenten und Schnittstellen auf die physikalische Hardware des Zynq realisiert. Dabei gibt es diverse Möglichkeiten, da viele verschiedene Rechen- und Kommunikationsressourcen auf der Plattform vorhanden sind (s. Kapitel 7.3.1). Folglich müssen beim Mapping Präferenzen gesetzt werden. Innerhalb dieser Projektgruppe werden diese auf die Handhabung der Mixed-Criticality und auf Ausführungsgeschwindigkeit der Anwendung gelegt.

Die Komponenten, die eine Rechenressource benötigen, und die Schnittstellen zwischen Komponenten wurden im Kapitel 7.2 identifiziert. Die Interfaces zu den Sensoren und zu den Aktoren wurden in Kapitel 7.3.8 definiert und werden insbesondere in Abbildung 7.19 dargestellt. Nachfolgend sind die ermittelten Komponenten und Schnittstellen noch einmal zusammengefasst. Hierbei ist zu beachten, dass aus der ASG-Komponente, wie sie in Abbildung 7.5 dargestellt ist eine weitere Komponente, die Dataminer-Komponente, herausgelöst wurde. Sie umfasst den SensorDriver, das Monitoring und den Remote Control Transceiver (vgl. Abbildung 7.5). Diese Module wurden aus der ASG-Komponente herausgelöst, weil sie für die Interaktion mit einem Teil der Sensoren und Aktoren und der Sensordatenfusion verantwortlich sind. Damit die ASG-Komponente die Sensordaten möglichst schnell verarbeiten kann wurde die Dataminer-Komponente darauf ausgelegt, alle Daten von den Sensoren abzuholen und sie in einen Speicher zu schreiben, auf den die ASG-Komponente Zugriff hat. Dies hat den Vorteil, dass keine Rechenkapazität für die Abholung und Fusionierung der Daten verloren geht, die bei der Berechnung der Motorstellwerte gebraucht wird. Darüber hinaus würde das Abbrechen und die Wiederaufnahme einer Kommunikation Protokoll-Overhead erzeugen, der bei einem Auslesen in einem Stück nicht auftreten würde. Dies spart Zeit ein.

7.5.1 Komponenten und Schnittstellen, die gemappt werden müssen

Komponenten

Tabelle 7.6: Identifizierte Komponenten, die gemappt werden müssen

Num.	Komponente	Beschreibung
1	ASG-Komponente	Entspricht der ASG-Komponente im Blockdiagramm 7.4.
2	Dataminer-Komponente	Ist Teil der ASG-Komponente im Blockdiagramm 7.4.
3	AEVV-Komponente	Entspricht der AEVV-Komponente im Blockdiagramm 7.4.

Die Aufteilung dieser Komponenten wird durch die Zuteilung von unterschiedlichen Kritikalitätsleveln verstärkt. Die ASG-Komponente enthält einen sicherheitskritischen An-



spruch, wohingegen die AEVV-Komponente nur eine rechenintensive Aufgabe erfüllen muss. Aus diesem Grund sollen die beiden Komponenten möglichst lose gekoppelt werden (s. Kapitel 7.1). Alle Unterkomponenten und deren Schnittstellen, die sich in der AEVV-Komponente (vgl. Kapitel 7.2) befinden, werden innerhalb der Implementierungsphase in Software realisiert.

Schnittstellen

Die Schnittstellen *IMotor*, *ISensors*, *IRadioRemote*, *IGimbal*, *ICamera* und *IRadioTele* aus Abbildung 7.4 werden in Abbildung 7.19 in Kapitel 7.3.8 präzisiert. Aus diesem Grunde wird in folgenden Tabelle nicht auf die Komponentendiagramme, sondern auf die Blockschaltbilder verwiesen. Zusätzliche wichtige Schnittstellen sind die Datenverbindungen zwischen der Dataminer- und der ASG-Komponente (vgl. *ISensorData* und *IRemoteCtrlData* in Abbildung 7.5) und zwischen der ASG- und der AEVV-Komponente (vgl. *IAvionic* in Abbildung 7.4). Alle weiteren Schnittstellen, die sich innerhalb der Komponenten aus Abbildung 7.4 befinden und nicht gesondert erwähnt werden, sind unter *weitere Schnittstellen* in der folgenden Tabelle aufgeführt. S

Tabelle 7.7: Identifizierte Schnittstellen, die gemappt werden müssen

Num.	Kommunikationsweg	Beschreibung
1	I ² C Schnittstelle	Schnittstelle zum Motortreiber
2	I ² C Schnittstelle	Schnittstelle zum Sensorboard
3	SPI Schnittstelle	Schnittstelle zu den beiden Akkuwächtern
4	PPM Schnittstelle	Schnittstelle zur Gimbalsteuerung
5	PPM Schnittstelle	Schnittstelle vom Fernsteuerungsempfänger
6	UART RX Schnittstelle	Schnittstelle zum Fernsteuerungsempfänger
7	USB Schnittstelle K	Schnittstelle zur Kamera
8	USB Schnittstelle W	Schnittstelle zum 802.11 Modem (s. Kapitel 7.3.5)
9	GPIO Schnittstelle	Steuersignale für LED, Summer und Gimbal
10	IAvionic Schnittstelle	Schnittstelle zwischen ASG- und AEVV-Komponente
11	ISensorData Schnittstelle	Schnittstelle zwischen Dataminer- und ASG-Komponente
12	IRemoteCtrlData Schnittstelle	Schnittstelle zwischen Dataminer- und ASG-Komponente
13	<i>weitere Schnittstellen</i>	Schnittstellen innerhalb der Dataminer- ASG-, und AEVV-Komponente



7.5.2 Mapping auf die Ressourcen

Die Komponenten müssen nun auf die Rechen- und Kommunikationsressourcen gemappt werden, die auf der in Kapitel 7.4 beschriebenen Plattform vorhanden sind. Die Namen der Rechenressourcen im Plattformdesign geben schon Aufschluss über ihre Funktion bzw. welche Komponente auf ihnen ausgeführt wird.

Mapping der Rechenressourcen

Die AEVV-Komponente wird auf dem ARM A9 Dual Core "AEVV-System" ausgeführt. Da die AEVV-Komponente viel Rechenleistung benötigen wird und der ARM A9 die leistungsstärkste CPU auf der Plattform ist, ist dies Zuordnung sinnvoll. Außerdem sind die meisten Peripheriegeräte, die die AEVV-Komponente benötigt, an dieser CPU angeschlossen. Alle übrigen Daten, die die Komponente benötigt, können ihr über den Dual Port BRAM "Steuerdaten" zugänglich gemacht werden. Darüber hinaus können die Multimedia Aufgaben der AEVV-Komponente von die Neon-Technologie des ARM Prozessors profitieren, indem sie schneller berechnet werden können.

Die ASG-Komponente wird auf dem MicroBlaze "ASG" ausgeführt. Er bietet genug Rechenleistung, hat Zugriff auf die Motorplatine und kann alle Sensordaten über den Dual Port BRAM "Sensordaten" erhalten.

Die Dataminer Komponente wird auf den MicroBlaze "Dataminer" gemappt. Er hat genug Rechenleistung und Zugriff auf alle Sensoren. Die Sensordaten können nach der Erhebung und Fusion mit Hilfe des Dual Port BRAMs "Sensordaten" an die ASG-Komponente versendet werden.

Mapping der Kommunikationsressourcen

Alle Sensoren und Aktoren sind schon auf der Plattform vorhanden wie in Abbildung 7.21 zu sehen. Entsprechend ist das Mapping der Schnittstellen auf der Plattform eindeutig. Die Schnittstellen sind die entsprechenden Verbindungen zwischen den Rechenressourcen und den Sensoren und Aktoren.

Die ISensorData und IRemoteCtrlData Schnittstellen werden auf den Dual Port BRAM "Sensordaten" gemappt. Er kann Daten zwischenspeichern und sorgt dafür, dass einzelne Daten immer vollständig gelesen und geschrieben werden. Darüber hinaus garantiert er, dass gleichzeitiges lesen und schreiben möglich sind. Diese Eigenschaften sind hilfreich, da keine unvollständigen Daten gelesen werden können. Außerdem können keine Verzögerungen durch das Warten auf Lese- oder Schreibzugriffe vorkommen, sodass die Echtzeitanforderungen genauer abgeschätzt werden können.

Das gleiche gilt für die IAvionic Schnittstelle. Sie wird auf den "Steuerdaten" Dual Port BRAM gemappt, der die gleichen Eigenschaften wie der "Sensordaten" BRAM hat.

8 Entwicklungsumgebung

8.1 Vivado

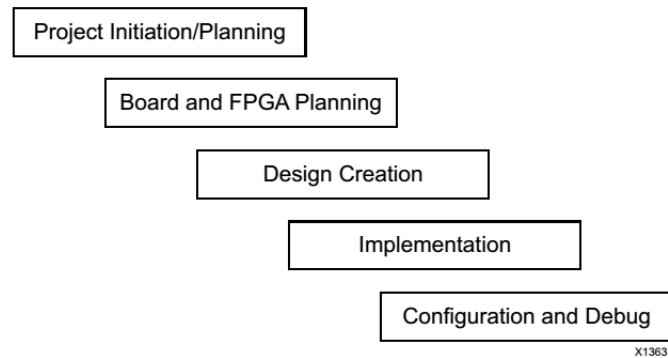


Abbildung 8.1: Designprozessschritte in Vivado (entnommen aus [?])

Vivado ist eine IDE zur Entwicklung von Systemen auf FPGA. Es enthält verschiedene Tools, die den Entwickler unterstützen sollen. Vivado wird in verschiedenen Versionen bereitgestellt. Für die Entwicklung in der Projektgruppe wird die WebPack-Version verwendet. Es ist mit dieser Version möglich, IPs zu integrieren und zu implementieren. Des Weiteren kann man dann die Software für die entsprechende Plattform in einem SDK entwickeln. Ebenfalls enthalten ist ein Simulator, mit dem das Verhalten der Hardware simuliert werden kann. Dadurch ist es möglich, Fehler zu finden und zu beseitigen. Ebenfalls lässt sich so auch das zeitliche Verhalten darstellen. Der Designprozess in Vivado ist in Abbildung 8.1 dargestellt. Die einzelnen Schritte können sich wiederholen, sofern dies nötig ist. Dieses Vorgehen findet sich auch in der Oberfläche von Vivado wieder.

Im Verlauf des Projektes wurde die Version mehrfach geändert. Die Gründe für die Updates werden in den folgenden Kapiteln näher beschrieben.

8.1.1 2013.4

Vivado 2013.4 war zu Beginn des Projektes die aktuelle Version. Wie oben beschrieben, dient Vivado dazu, eine Architektur zu designen und Software für diese Architektur zu programmieren. Ein Problem mit der Version ist, dass sie nicht mit der benötigten RTOS-Version kompatibel war. So war ein Wechsel nötig.

8.1.2 2014.1

Wie oben beschrieben, wurde auf die neueste Version umgestiegen. Diese wurde bis Ende des Jahres verwendet, da keine Probleme erwartet wurden. Zwischenzeitlich hatte sich herausgestellt, dass RTOS doch nicht für das Projekt geeignet ist. Die Version behält



man trotzdem bei, da zwischenzeitlich der PPM-Interpreter (9.2.6) erstellt worden ist, und ansonsten die neue Version keinerlei Nachteile hat, wurde die Version beibehalten.

8.1.3 2014.4

Die Board-Support-Package-Gruppe evaluierte gegen Ende noch den Einsatz von Petalinux. Die Petalinux-Tools werden immer in einer geade Versionsnummer veröffentlicht. Zum Zeitpunkt der Entscheidung war die aktuellste Version 2014.4. Diese wurde daher dann genutzt. Es stellte sich allerdings heraus, dass Petalinux, doch nicht geeignet ist. Es wurde daher eine eigene Linuxversion gebaut.

8.2 Vivado SDK

Das Vivado Software Development Kit ist eine auf Eclipse basierende Software. Sie ermöglicht es, Software für die einzelnen Plattformen in einem Design zu schreiben. Das SDK bietet die Möglichkeit den in Vivado generierten Bitstream auf den FPGA zu schreiben. Die Programme können dann auf der entsprechenden Plattform (ARM-Core oder Microblaze) ausgeführt werden. Ebenfalls enthalten ist ein Debugger, der bei der Entwicklung hilfreich ist.

8.2.1 Versionen

Die Versionen des SDK sind mit Vivado verknüpft. Für die Entwicklung gab es beim SDK keine relevanten Änderungen, weshalb an dieser Stelle auf eine Unterscheidung verzichtet wird.

8.2.2 Debugging

Die Möglichkeiten des SDK zum Debuggen sind vielfältig. An dieser Stelle werden die häufig verwendeten eingeführt.

8.2.2.1 System Debugger Das Xilinx SDK bietet die Möglichkeit Programme zu debuggen. Der System Debugger stellt dazu eine Verbindung zu dem Board Mittels JTAG her. Der System Debugger ermöglicht es verschiedene Systeme zu Debuggen. Es können sowohl Bare-Metal-Programme als auch Linux-Applikationen überprüft werden. Auch lassen sich die Programme auf verschiedenen Prozessoren debuggen. Es können sowohl Programme auf dem dualen ARM laufen, als auch auf verschiedenen MicroBlaze-Prozessoren.

8.2.2.2 GNU Tool Chain Support Ebenfalls wird vom SDK die GNU Tool Chain unterstützt. Um diese zu nutzen muss wird im Entwicklungsprozess ein Board Support Package (BSP) erstellt. Dieses enthält die benötigten Treiber und ermöglicht die Entwicklung von Stand-Alone-Applikationen. Mit Hilfe eines Assistenten lassen sich über JTAG dann die



Programme auf die Plattform bringen. Eine Zuordnung auf einzelne Kerne ist hier auch möglich.

8.2.2.3 Profiling Ebenfalls ist es möglich mit dem SDK ein Profiling der erstellten Plattform durchzuführen. Mit Hilfe des Profiling ist es möglich die Ausführungszeiten ganzer Programme oder einzelner Funktionen zu messen. Dazu verwendet das SDK das Programm *gprof*. Im Rahmen des Projektes sollte das Profiling genutzt werden, um die Echtzeitanforderungen zu überprüfen. Es wurde versucht Anhand von Anleitungen³⁷ das Profiling einzurichten. Da die Anleitung sich noch auf das EDK bezieht und neuere Beschreibungen fehlen, konnte ein Profiling nicht durchgeführt werden. Stattdessen wird ein Messbasierter Ansatz verwendet (10.2.4).

8.3 Buildroot

Buildroot ist ein Werkzeug mit dem sich eigene, speziell an die Hardware und Aufgabe angepasste embedded Linux Distributionen generieren lassen. Es ist sehr stark konfigurierbar und erweiterbar und unterstützt den Entwickler durch einen hohen Grad der Automatisierung. Buildroot bietet neben der Möglichkeit den Kernel und Bootloader für die Zielplattform zu kompilieren, eine große Auswahl an Open-Source Anwendungen von diversen Linux-Shells über Bibliotheken bis hin zu Grafischen Benutzeroberflächen. Buildroot lässt sich in mehreren Stufen über das vom Linux Kernel bekannte *make menuconfig* Interface konfigurieren. Zunächst gibt es die Buildroot Konfiguration selbst, dort sind z. B. Einstellungen bezüglich der Zielplattform, der zu verwendenden Cross-Compiler-Toolchain oder der zu installierenden Software Pakete vorzunehmen. Hier wird auch konfiguriert, ob Linux Kernel und Bootloader von Buildroot kompiliert werden und welche Versionen verwendet werden sollen. In weiteren Konfigurationsebenen lassen sich dann die Konfiguration der Linux-Shell Umgebung, des Linux Kernels und des Bootloaders vornehmen. Um keine Probleme mit einer externen Cross-Compiler-Toolchain zu bekommen, wird in Buildroot die Xilinx Toolchain verwendet, die zusammen mit der Vivado Design Suite installiert wird. Für die Integration der AEVV-System Software, lässt sich Buildroot sehr einfach um weitere Softwarepakete erweitern. Diese Pakete stehen dann in der bereits erwähnten Konfigurationsoberfläche zur Verfügung. Als Quelle eines Softwarepaketes kann Buildroot u. a. ein Repository oder lokalen Dateisystempfad verwenden, dies ist gerade während der Entwicklung des AEVV-Systems von Vorteil. Änderungen lassen sich so ohne großen Aufwand direkt testen. Neben dem Cross-Compiling übernimmt Buildroot auch die Aufgabe das Wurzeldateisystem zu erstellen. Hierzu wird ein minimales Template verwendet, in das jedes Softwarepaket bei seiner Installation zusätzlich benötigte Dateien oder Ordner anlegen kann.

³⁷http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/edk_prof.pdf

8.4 CAMEL-View

CAMEL-View ist Tool zur modellbasierten Entwicklung von mechatronischen Systemen. Es wird von der Firma iXtronics entwickelt und vertrieben. Für die Arbeit in der Projektgruppe wurden uns die Lizenzen zur Verfügung gestellt. Ebenfalls wurde uns eine s.g. Testrig-Box geliehen, welche insbesondere für den HIL-Test Kapitel 10.2.3.5 erforderlich ist. CAMEL-View ist ebenfalls in der Lage c-Code aus einem erstellten Modell zu generieren.

Um das Modell ausführen zu können benötigt man einen Wrapper (Kapitel 8.5). In dem generierten Code sind außerdem Informationen für die Simulationsoberfläche enthalten. Da dieser Code nicht zertifiziert ist, muss daher auch ein SIL-Test (Kapitel 10.2.3.4) durchgeführt werden, um zu verifizieren, dass dieser Code, dem Modell entspricht.

8.4.1 Analyse

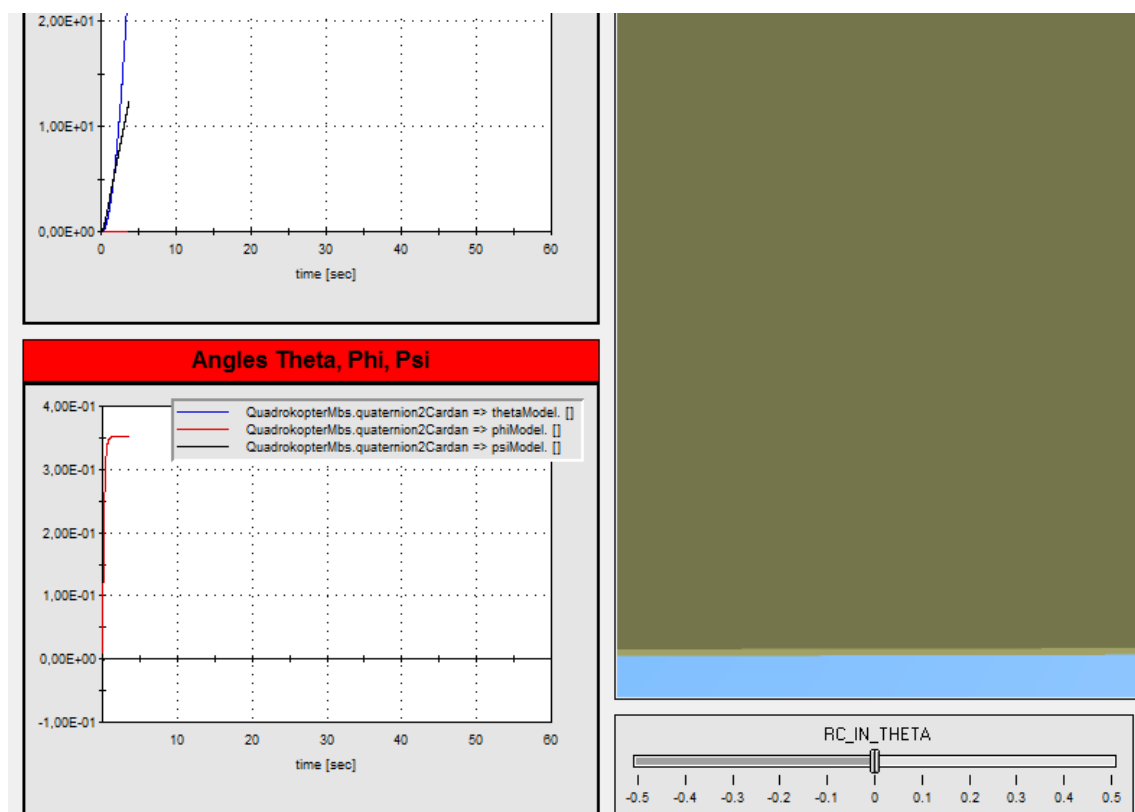


Abbildung 8.2: Grafische Elemente in CAMEL-View

Neben der Codegenerierung bietet CAMEL-View auch die Möglichkeit das Modell zu analysieren. Das Programm stellt dazu eine Oberfläche bereit, mit der man mit Hilfe von grafischen Elementen, wie z. B. Plots oder Schieberegler, Werte in das Modell eingeben bzw. sich ausgeben lassen kann. Abbildung 8.2

8.5 Wrapper „Simplified Simulation Framework“

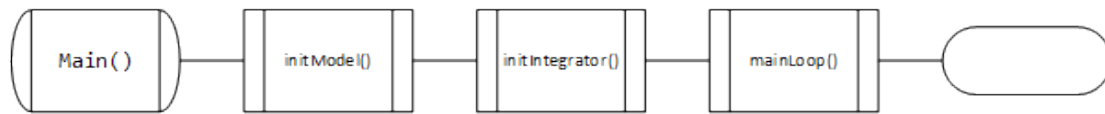


Abbildung 8.3: Ablaufdiagramm Framework

Das SimplifiedSimulationFramework stammt ebenfalls von der Firma iXtronics und wurde der Projektgruppe zur Verfügung gestellt um die Modelle aus CAMEL-View ausführen zu können, es handelt sich dabei also um einen Wrapper. Dieser Wrapper wird im Projekt an bestimmte Umgebungen angepasst. Zum einen sollen die Regler für die Flugregelung auf einem Microblaze ausgeführt werden und zum Anderen sollen die Modelle im SIL-Verfahren getestet werden. Für beide Anwendungen sind daher Anpassungen nötig, die in den jeweiligen Kapiteln Kapitel 9.5.1.7 und 8.5.1 beschrieben werden.

Das Framework beinhaltet verschiedene Bibliotheken, welche die Funktionalität beinhalten. Zum anderen gibt es ein paar Dateien, die Einstellungsmöglichkeiten, wie z. B. die Startzeit, beinhalten. Zur Simulation des Modells werden mathematische Funktionen, wie Winkelfunktionen, und Integratoren benötigt. Das Framework enthält die Integrationsverfahren nach Euler und Heun.

Der Ablauf des Frameworks ist in Abbildung 8.3 zu sehen. In der Main-Funktion werden zu erst der Speicher reserviert und das Model sowie der Integrator initialisiert. Als nächstes wird die Simulationsschleife aufgerufen, welche das Modell dann simuliert. Die Funktion `mainLoop()` wird wie oben beschrieben für die jeweiligen Aufgaben angepasst.

8.5.1 Anpassungen für den SIL-Test

Das Framework musste für den SIL-Test angepasst werden. Es müssen die CSV-Dateien, welche die Ergebnisse des MIL-Tests enthalten, eingelesen werden. Und des weiteren muss die Simulationsschleife angepasst werden. Der Ablauf ist in Abbildung 8.4 zu sehen.

8.5.1.1 Ausgabe verschoben In dem ursprünglichen Version des Wrappers befand sich die Ausgabe nach der Funktion `executeIntegrationStep()`. Dies führte dazu, dass beim SIL-Test die Ausgaben um einen Zeitschritt verschoben waren. Um dies zu korrigieren, wurde die Ausgabe der Werte zwischen dem Auswerten des Hauptschritts und den Integrationsschritten verschoben.

8.5.1.2 CSV-Parser Zum Einlesen der CSV-Datei wird eine bereits vorhandene Lösung verwendet³⁸. Der Parser kann CSV-Dateien einlesen und auch mit Überschriften umgehen. Ein Nachteil ist, dass der Parser nur eine Zeile als Überschrift akzeptiert. Dies

³⁸<http://sourceforge.net/projects/cccsvparser/>



wird in dem Framework umgangen in dem der Parser so initialisiert wird, dass die erste Zeile keine Überschrift ist, d.h. die erste Zeile enthält bereits Werte. Mit dem Befehl `Csv_destroy_row()` werden die drei Überschriftenzeilen ignoriert.

8.5.1.3 Spaltensuche Ein Problem ist, dass beim Export der Daten aus CAMEL-View die Reihenfolge der Spalten nicht immer die gleiche ist. Daher wird im Framework noch ein Array angelegt, in dem die Position der Werte für die Fernbedienung gespeichert wird. Daher wird in dem Framework erst noch die Zeile mit den Spaltenüberschriften durchsucht und eine Variable mit einem Mapping angelegt.

8.5.1.4 Runge-Kutta-Algorithmus Das Framework besitzt von sich aus nicht die Möglichkeit das Integrationsverfahren nach Runge-Kutta durchzuführen. Daher wurde dieses Verfahren in dem Wrapper integriert. Allerdings wird das Verfahren nicht genutzt, da sich herausgestellt hat, dass die Testrig-Bix dieses Verfahren nicht unterstützt. Um die Tests einheitlich zu halten wurde daher auf dieses Verfahren verzichtet. Der Runge-Kutta-Algorithmus wurde nach [FPL13] implementiert.

8.5.1.5 Aufruf des Frameworks Der Wrapper wurde dahingehend angepasst, dass es nun mit Aufrufparametern gestartet werden kann. Somit können nun die CSV-Dateien für die Ein- und Ausgabewerte, sowie die Schrittweite angegeben werden. Der Vorteil bei diesem Vorgehen ist, dass die SIL-Tests so in einem Batch-Verfahren durchgeführt werden können.

8.5.2 Anpassungen für den Zynq

Um die erstellten Regler auf dem Zynq ausführen zu können, muss der Wrapper dafür angepasst werden. Zum einen ist es nicht gewünscht, dass das Modell simuliert wird. Daher wird bei jedem Ausführungsschritt die Zeit auf Null zurückgesetzt. Des weiteren müssen die Daten aus dem BRAM, an die richtigen Stellen des Modells geschrieben werden. Dazu wird das Mapping des BRAM-Controllers verwendet. Um die Daten der Sensoren in das Modell einzufügen.

In verschiedenen Datenstrukturen, sind die Informationen über den Aufbau des Modells gespeichert. Darunter Name des Blocks, sowie der Ein- und Ausgänge und der Parameter. Über die Struktur `theSimModel` ist es möglich sich, durch das Modell zu 'hangeln'.

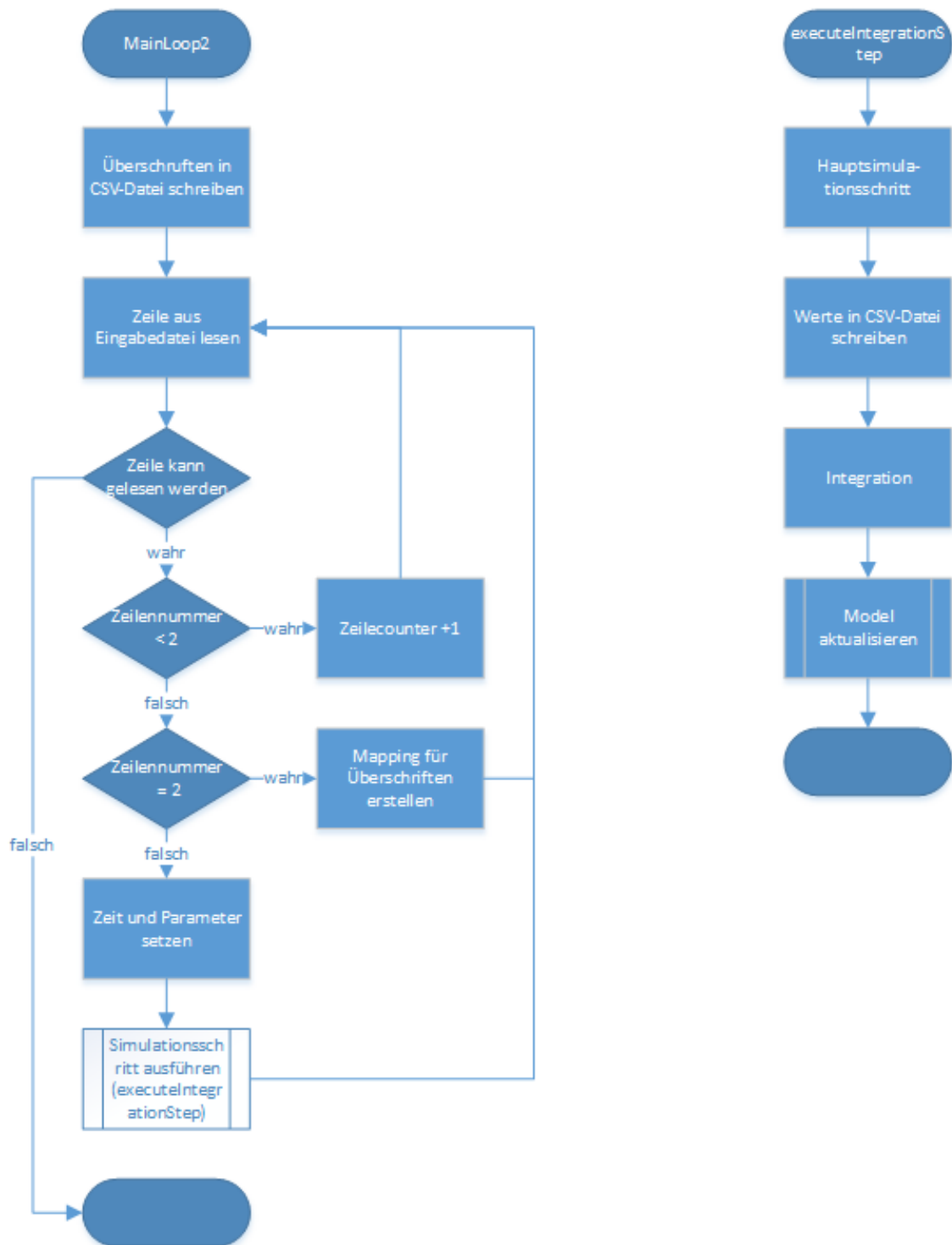


Abbildung 8.4: Ablaufdiagramm Simulationsschleife



9 Implementierung der Teilsysteme

In diesem Kapitel werden die einzelnen Systemkomponenten näher beschrieben und ihre Funktionsweisen erläutert.

9.1 Integration der Hardware-Plattform

Die Hardware-Plattform besteht aus mehreren Teilsystemen (Carrierboard, Prozessorboard und Verteilerplatine), welche im Folgenden erläutert werden.

9.1.1 Anpassung des TE0703-04 Carrierboards

Das Prozessorboard TE0720 wird über ein Carrierboard TE0703-04 an der entwickelten Verteilerplatine befestigt. Um alle Registerbänke auf dem verwendeten Prozessorboard zu verwenden, ist es nötig diese jeweils mit 3,3 V Versorgungsspannung zu versorgen. Auf dem Carrierboard wird die Registerbank 34 über den Jumper J5 mit 3,3 V versorgt. Die anderen drei Bänke (13, 33, 35) werden über die Lötbrücken R_{23} , R_{26} und R_{35} mit Spannung versorgt.

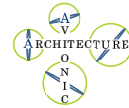
9.1.2 Aufbau der Verteilerplatine

In diesem Abschnitt wird näher auf die Verteilerplatine eingegangen, welche auf Basis des in Kapitel 7.3.8 vorgestellten Hardwarekonzeptes entwickelt wird. Wie bereits erwähnt, ist die Verteilerplatine in zwei galvanisch voneinander getrennten Stromkreisen (Logic-Part und Energy-Part) aufgeteilt. Der verwendete Prozessor wird über den Logic-Part versorgt. Um die galvanische Trennung aufrecht zu erhalten, werden die Steuersignale zwischen Logic- und Energy-Part über Isolatoren geleitet. In dem Schaltplan Abbildung 9.1 der Verteilerplatine ist ebenfalls der Logic- und Energy-Part klar voneinander abgegrenzt, um eine bessere Übersichtlichkeit zu gewährleisten.

Isolatoren

Als Isolatoren – mit IC_1 , IC_7 und IC_9 im Schaltplan gekennzeichnet – kommen Bausteine der ADuM-Familie der Firma Analog Device zum Einsatz. Neben der galvanischen Trennung übernehmen diese Chips ebenfalls die Aufgabe der Spannungswandlung. Der Prozessor auf dem Logic-Part arbeitet mit 3,3 V wohingegen die Peripherien, welche an dem Energy-Part angeschlossen wird (z. B. Motortreibern oder das Extensionboard), mit 5 V Signalen arbeiten. Des Weiteren wurde bei der Auswahl der Isolatoren darauf geachtet, dass diese kompatibel zu den maximale Datenrate der angeschlossenen Signale sind. Im Gegensatz zu den anderen Isolatoren, wird IC_7 über den N-Kanal Mosfet Q_9 mit Hilfe der Chip Select Leitung $CS2_1$ entsprechend ein- bzw. ausgeschaltet. Im ausgeschalteten Zustand besitzt $OUTD$ von IC_7 eine hochohmigen Zustand, wohingegen im eingeschalteten Zustand der Pegel abhängig von IND ist (High oder Low). Der hochohmige Ausgang wird

9 IMPLEMENTIERUNG DER TEILSYSTEME



9.1 Integration der Hardware-Plattform

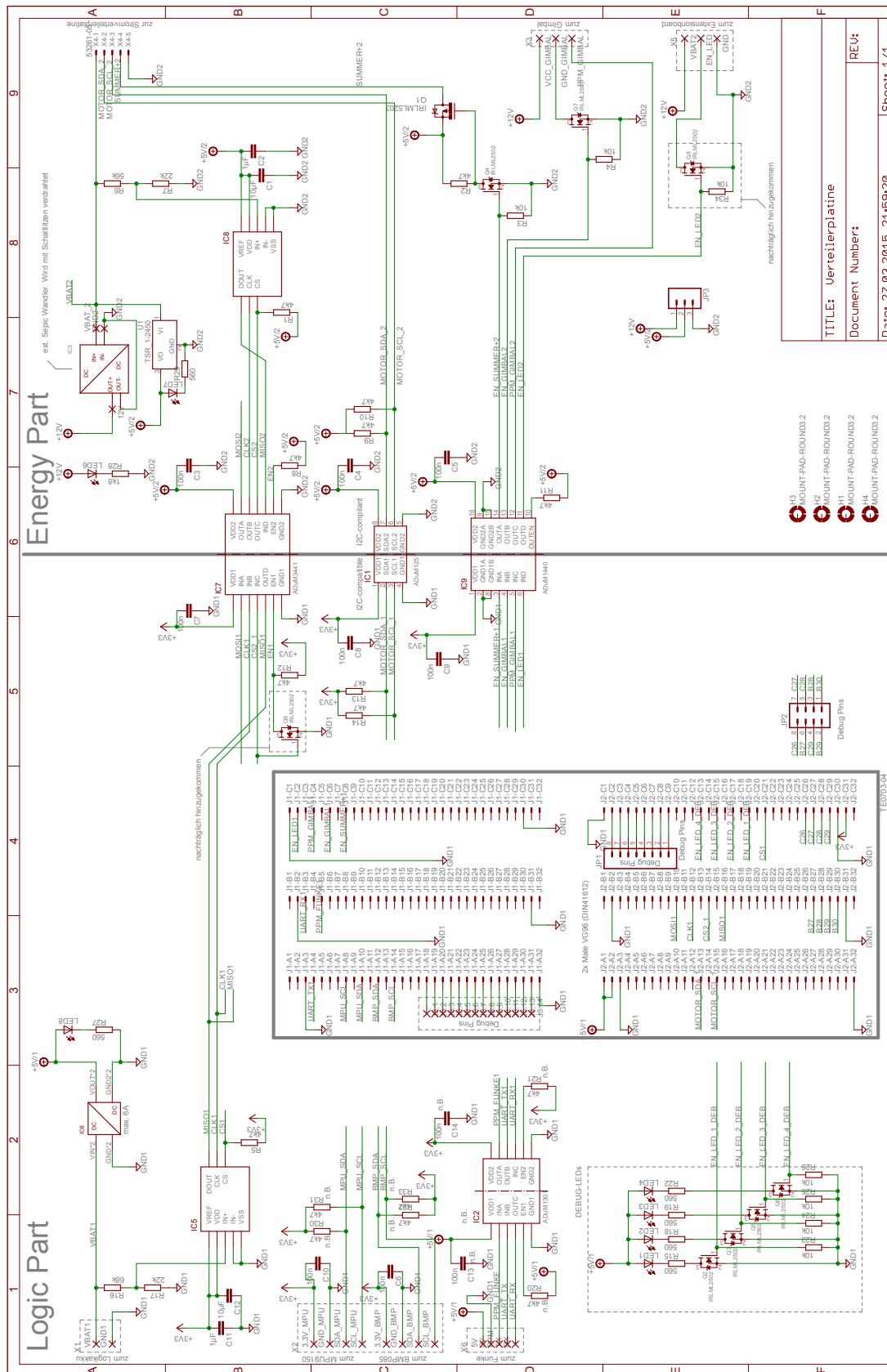


Abbildung 9.1: Schaltplan Verteilerplatine



benötigt, wenn mehrere Slaves über die *MISO1* Leitung Daten senden wollen. Ansonsten könnte der aktive Slave nicht den Pegel der Leitung ändern und die Daten würde fehlerhaft vom Master empfangen werden. IC_2 ist ebenfalls ein ADuM Baustein, der jedoch nur die Aufgabe der Spannungswandlung zwischen dem Funkreceiver und dem Prozessor hat. Dieser Baustein wird in diesem Projekt jedoch nicht genutzt, da der hier verwendete Receiver (GR-16 von Graupner) mit 3,3 V Pegeln arbeitet, und dient ausschließlich für mögliche Erweiterungen.

Um den USB-Anschluss zur Gimbalsteuerung gegenüber den Logic-Part zu isolieren, wird ein USB-Isolator³⁹ eingesetzt, der nicht im Schaltplan verzeichnet ist.

9.1.2.1 Logic-Part

Spannungsversorgung

Der Logic-Part wird über eine 3S Lithium-polymer Akkumulator, welcher an dem XT60 Stecker angeschlossen wird, über X_1 mit Spannung versorgt. Die Spannung des Akkumulators wird mit einem Step-Down-Wandler IC_6 auf 5 V gewandelt, um die Bauelemente auf dem Logic-Part mit Spannung zu versorgen. Step-Down-Wandler zeichnen sich im Allgemeinen durch ihren hohen Wirkungsgrad aus und werden daher häufig in batteriebetriebenen Produkten eingesetzt. Als Wandler kommt eine Spannungswandlerplatine von Pololu⁴⁰ zum Einsatz, welche die zur Wandlung benötigten elektrischen Bauteile auf kleiner Fläche komprimiert, einen maximalen Ausgangsstrom von 6 A und einen Wirkungsgrad von bis zu 90 % bei 12 V Eingangsspannung, 5 V Ausgangsspannung und 1 A Last liefert. Die 3,3 V Versorgungsspannung für die Isolatoren werden von dem Carrierboard zur Verfügung gestellt.

Akkuwächter

Als Akkuwächter IC_5 kommt ein MCP3201 der Firma Microchip zum Einsatz. Dieser Baustein ist ein 12-Bit Analog/Digital Converter⁴¹ mit integriertem SPI-Interface. Die Widerstände R_{16} und R_{17} dienen als Spannungsteiler, um die Spannung des Logic-Akkus auf die Messspannung zu verringern. Die Messspannung darf dabei maximal bei der Referenzspannung des Akkuwächters, hier 3,3 V, liegen. Ein geladener Lithium-polymer Akkumulator hat eine maximale Zellspannung von 4,2 V. Der 3 Zellen Logic-Akku hat somit eine maximale Spannung $U_{logicIn,max}$ von 12,6 V. Die maximale Messspannung $U_{logicMess,max}$ beläuft sich daher auf

$$\frac{R_{16} + R_{17}}{R_{17}} = \frac{U_{logicIn,max}}{U_{logicMess,max}} \quad (9.1)$$

$$U_{logicMess,max} = \frac{U_{logicIn,max} \cdot R_{17}}{R_{16} + R_{17}} = \frac{12,6 V \cdot 22 k\Omega}{68 k\Omega + 22 k\Omega} \approx 3,08 V. \quad (9.2)$$

³⁹USB-Isolator <http://www.watterott.com/de/USB-Isolator> Zugriff: 19.12.2014

⁴⁰Step-Down-Wandler: D24V60F5 <http://www.pololu.com/product/2865> Zugriff: 19.12.2014

⁴¹Analog/Digital Converter (ADC)



Bei der Referenzspannung $U_{logic,ref}$ von 3,3 V und bei einer Auflösung von 12-Bit beträgt die LSB⁴²-Auflösung LSB_{size}

$$LSB_{size} = \frac{U_{logic,ref}}{2^{12} \text{ bit}} = \frac{3,3 \text{ V}}{4096 \text{ bit}} \approx 0,8 \frac{\text{mV}}{\text{bit}}. \quad (9.3)$$

Aufgrund von Toleranzen in den Spannungsteiler-Widerständen von $\pm 1\%$ und Grundrauschen auf der Signalleitung, wird in diesem Projekt mit einer begrenzten Auflösung von 10-Bit gerechnet und die zwei LSBs vernachlässigt. Daraus ergibt sich eine reduzierte LSB-Auflösung $LSB_{size,red.logic}$ von

$$LSB_{size,red.logic} = \frac{U_{logic,ref}}{2^{10} \text{ bit}} = \frac{3,3 \text{ V}}{1024 \text{ bit}} \approx 3,2 \frac{\text{mV}}{\text{bit}}. \quad (9.4)$$

Sendet somit der MPU3201 einen digitalen Spannungswert ADC_{value} von 900, liegt eine Batteriespannung U_{Bat} von

$$U_{Bat} = ADC_{value} \cdot LSB_{size,red.logic} \cdot \frac{R_{16} + R_{17}}{R_{17}} \quad (9.5)$$

$$U_{Bat} = 900 \text{ bit} \cdot 3,2 \frac{\text{mV}}{\text{bit}} \cdot \frac{68 \text{ k}\Omega + 22 \text{ k}\Omega}{22 \text{ k}\Omega} = 11781,8 \text{ mV} \approx 11,78 \text{ V} \quad (9.6)$$

an.

Peripherieanschlüsse

Der Logic-Part bietet unter anderem Anschlüsse für zwei voneinander getrennte I²C Busse (X_2). An diesen werden der BMP085 und der MPU9150 angeschlossen. Für beide Bussen sind optional Pull-Up Widerstände ($R_{30} - R_{33}$) vorgesehen, welche bei Verwendung der genannten Sensoren jedoch nicht benötigt werden.

Des Weiteren bietet X_6 ein UART Interface zum Anschluss des Rückkanals der Funkverbindung an. Der verwendete Receiver arbeitet mit einem sog. „One Wire“-UART. Dazu werden die TX- und RX-Leitung des Interfaces über jeweils 1 k Ω zusammengeschaltet. Dadurch können die Daten nur im halbduplex Modus versendet werden. Die zusätzlichen Widerstände sind in der Leitung zwischen X_6 und Receiver integriert. Das PPM-Steuersignal zur Flugregelung wird ebenfalls an Klemme X_6 angeschlossen.

Das Carrierboard stellt eine USB-Schnittstelle bereit, an der über ein 4-Port USB-Hub die Kamera, der Gimbal und der WLAN-Stick angeschlossen sind.

Erweiterungsanschlüsse

Um ein komfortables Testen der Verteilerplatine und eine relativ einfache Erweiterung zu ermöglichen, sind diverse Anschlüsse des Carrierboards über JP_1 , JP_2 und J_3 herausgeführt. Des Weiteren sind vier LEDs ($LED_1 - LED_4$) an das Carrierboard angeschlossen, welche zu Testzwecken oder als Statusanzeige verwendet werden können.

⁴²least significant bit (LSB)

9.1.2.2 Energy-Part

Spannungsversorgung

Die Versorgung des Logik-Parts der Verteilerplatine erfolgt über einen 4 S Lithium-polymer Akkumulator, der an der Molexbuchse X_4 angeschlossen wird. Die benötigten Spannung von 12 V und 5 V werden von Spannungswandlern (IC_3 und U_1) übernommen. Die 5 V Spannung wird, wie auf dem Logic-Part, von einem Step-Down-Wandler erzeugt⁴³. Bei der 12 V Spannungsgenerierung wird ein sog. Single Ended Primary Inductance Converter (SEPIC)-Wandler⁴⁴ eingesetzt. Ein SEPIC-Wandler hat neben dem hohen Wirkungsgrad gegenüber linearen Spannungsreglern den Vorteil, dass dieser die Eingangsspannung herauf sowie herab transformieren kann. Somit ist auch bei Über- oder Unterschreitung der Ausgangsspannung gewährleistet, dass diese stabil bleibt. Ein Unterschreiten der Versorgungsspannung könnten in diesem Projekt z. B. durch hohe Motorströme oder durch einen niedrigen Akkustand erreicht werden.

Akkuwächter

Wie schon bei dem Logic-Part wird auf dem Energy-Part ein MCP3201 als Akkuwächter (IC_8) eingesetzt. Die Referenzspannung $U_{energy,ref}$ dieses Akkuwächters liegt bei 5 V. Durch den verwendeten 4 Zellen Akku auf dem Energy-Part steigt die maximale Spannung $U_{energyIn,max}$ auf 16,8 V an. Die maximale Messspannung $U_{energyMess,max}$ beläuft sich daher auf

$$\frac{R_6 + R_7}{R_7} = \frac{U_{energyIn,max}}{U_{energyMess,max}} \quad (9.7)$$

$$U_{energyMess,max} = \frac{U_{energyIn,max} \cdot R_7}{R_6 + R_7} = \frac{16,8 V \cdot 22 k\Omega}{56 k\Omega + 22 k\Omega} \approx 4,74 V. \quad (9.8)$$

Analog zum Logic-Part wird auch hier mit einer reduzierten Auflösung von 10-Bit gearbeitet. Somit beträgt die reduzierte LSB-Auflösung $LSB_{size,red.energy}$

$$LSB_{size,red.energy} = \frac{U_{energy,ref}}{2^{10} bit} = \frac{5 V}{1024 bit} \approx 4,9 \frac{mV}{bit}. \quad (9.9)$$

Peripherieanschlüsse

Neben der Versorgungsspannung werden an der Molexbuchse X_4 ebenfalls die I²C- Motortreiber angeschlossen. R_9 und R_{10} sind die entsprechenden Pull-Up Widerstände des Busses. An dem letzten Anschluss der Molexbuchse wird ein Summer angeschlossen, welcher über die High-Side Treiberstufe Q_1 und Q_4 geschaltet wird.

X_3 bietet eine Anschluss für den Gimbal. Die Versorgungsspannung des Gimbals kann

⁴³Step-Down-Wandler: TracoPower TSR 1-2450 <http://www.adafruit.com/product/1065> Zugriff: 19.12.2014

⁴⁴SEPIC-Wandler: Adjustable 1,2-30 V 12W http://www.lipoly.de/index.php?main_page=product_info&cPath=880_926_3306_3732&products_id=272901 Zugriff: 19.12.2014

9.1 Integration der Hardware-Plattform

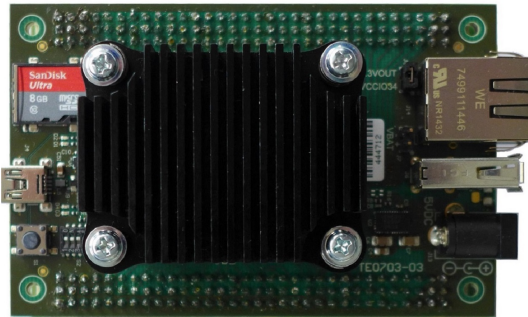
über Q_7 ein- bzw. ausgeschaltet werden. Der Gimbal wird über X_3 mit dem 2 polige JST-Buchse angeschlossen.

An X_5 wird das Extensionboard angeschlossen, welches die LEDs unterhalb des Quadropters ansteuert. Über Q_8 werden die LEDs geschaltet. Q_8 und R_{34} sind nicht auf dem ersten Layout der Verteilerplatine enthalten und sind nachträglich hinzugefügt worden. Tabelle 9.1 zeigt ein Mapping zwischen den I/Os der Prozessorplattform und den Schnittstellen auf der Verteilerplatine. Ebenfalls ist in der Spalte „Mapping“ eingetragen, an welche Softwarekomponente die I/Os angeschlossen sind.

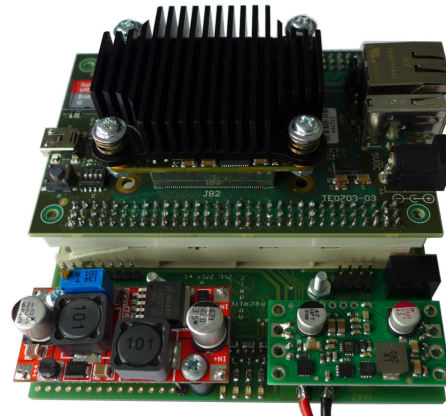
Tabelle 9.1: Pinzuordnung zwischen der Verteilerplatine (VP) und dem Xilinx Zynq 7020

Pin VP	Belegung	Dir.	Pin Zynq	Bank Zynq	Mapping
J1-A4	UART_TX1	IN	H19	Bank 35	Dataminer
J1-A8	MPU_SCL	OUT	D20	Bank 35	Dataminer
J1-A10	MPU_SDA	IN/OUT	B19	Bank 35	Dataminer
J1-A12	BMP_SDA	IN/OUT	E21	Bank 35	Dataminer
J1-A14	BMP_SCL	OUT	C15	Bank 35	Dataminer
J1-B3	UART_RX1	OUT	G17	Bank 35	Dataminer
J1-B5	PPM_FUNKE_1	IN	F18	Bank 35	Dataminer
J1-C2	EN_LED1	OUT	F16	Bank 35	Dataminer
J1-C4	PPM_GIMBAL_1	OUT	E15	Bank 35	Dataminer
J1-C6	EN_GIMBAL_1	OUT	G19	Bank 35	Dataminer
J1-C8	EN_SUMMER_1	OUT	G15	Bank 35	Dataminer
J2-A13	MOTOR_SDA_1	IN/OUT	V12	Bank 13	ASG
J2-A15	MOTOR_SCL_1	OUT	AA7	Bank 13	ASG
J2-B10	MOSI1	OUT	V8	Bank 13	Dataminer
J2-B12	CLK1	OUT	W11	Bank 13	Dataminer
J2-B14	CS2_1	OUT	Y6	Bank 13	Dataminer
J2-B16	MISO1	IN	Y9	Bank 13	Dataminer
J2-C13	EN_LED_4_DEB	OUT	AA9	Bank 13	AEVV
J2-C15	EN_LED_3_DEB	OUT	AA11	Bank 13	ASG
J2-C17	EN_LED_2_DEB	OUT	AA12	Bank 13	Dataminer
J2-C19	EN_LED_1_DEB	OUT	AA16	Bank 33	Dataminer
J2-C21	CS1	OUT	AA17	Bank 33	Dataminer

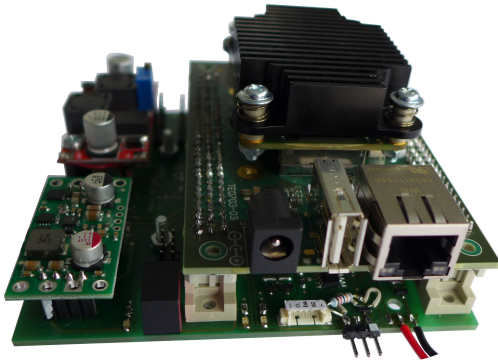
Eine weitere Tabelle A.1 im Anhang zeigt die Zuordnung von weiteren Debug-Pins auf der Verteilerplatine zur Prozessorplattform und Abbildung 9.2 zeigt die entwickelte Verteilerplatine.



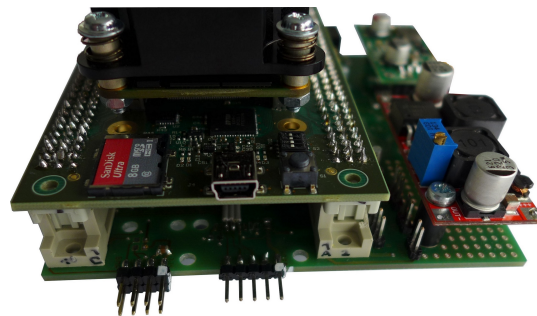
(a) Carrierboard mit aufgesteckter TE0720 Prozessorplattform



(b) Bestückte Verteilerplatine; Spannungsversorgung Energy-Part (rot), rechts Spannungsversorgung Logic-Part



(c) Unterhalb des Carrierboards Anschlüsse der Energy-Part Peripherie



(d) Unterhalb des Carrierboards Anschlüsse der Logic-Part Peripherie

Abbildung 9.2: Aufbau Verteilerplatine

9.2 Board Support Package

Das für den Betrieb benötigte Software-Grundgerüst wird als Board Support Package (BSP) bezeichnet. Es besteht aus mehreren Komponenten, die speziell an die vorliegende Hardware-Plattform angepasst sind und stellt das Verbindungsglied zwischen der Hardware und unseren Anwendungen ASG und AEVV-System dar.

In den folgenden Kapiteln werden die Komponenten vorgestellt und insbesondere auf Anpassungen und Konfiguration für die Hardware-Plattform eingegangen.

9.2.1 Bootvorgang

Der Bootvorgang umfasst alle Vorgänge vom Einschalten des Systems bis zu dem Zeitpunkt an dem das ASG und das Betriebssystem des AEVV-Systems ihre Arbeit aufnehmen. Das verwendete Xilinx Zynq SoC bietet eine große Flexibilität bei der Konfiguration des Boot-

prozesses. Es sind sechs verschiedene Bootmodi mit jeweils einem anderen Bootmedium als Quelle wählbar, die sich mit einer Vielzahl an mehrstufigen Softwarekonfigurationen aus Bootloadern, Betriebssystemen und Anwendungen kombinieren lassen (vgl. [Incc], Kapitel 6.1). Der für die Realisierung ausgewählte Prozess verwendet die SD-Karte als Bootmedium und setzt sich aus drei Phasen zusammen, die sequentiell ausgeführt werden.

In der ersten Phase wird das BootROM ausgeführt, eine Software die fest in das Xilinx Zynq SoC integriert ist und vom Benutzer nicht verändert werden kann. Es kommt auf beiden Prozessorkernen des ARM-Prozessors zur Ausführung, wobei sich der sekundäre Kern nach feststellen der eigenen Identität schlafen legt. Die eigentliche Ausführung wird auf dem primären Kern fortgesetzt. Das BootROM initialisiert das System entsprechend des ausgewählten Bootmodus um Zugriff auf das Bootmedium zu erlangen und lädt von dort den First Stage Boot Loader (FSBL) in einen On-Chip Memory (OCM). Sobald dies abgeschlossen ist, wird jeglicher Zugriff auf den Programmcode des BootROM deaktiviert und die Kontrolle an den FSBL übergeben. Der FSBL ist eine Software die im Xilinx SDK generiert wird und stellt die zweite Phase des Bootvorgangs dar. Er nimmt die weitere Initialisierung und Konfiguration des ARM-Prozessors und der Hardwareperipherie vor. Zudem programmiert er den FPGA und schließt damit den Bootvorgang des ASG ab. Dessen Ausführung beginnt, in dem das Reset-Signal des FPGA durch den FSBL aufgehoben wird. Der Bootvorgang für das AEVV-System wird mit dem Laden des Second Stage Boot Loader (SSBL) fortgesetzt, anschließend wird ihm die Kontrolle übergeben. In der dritten Phase wird der SSBL ausgeführt. Seine Aufgabe ist es, die Komponenten des AEVV-Betriebssystems vom Bootmedium in den Arbeitsspeicher zu laden und zu starten. Als SSBL kommt U-Boot zum Einsatz, ein Bootloader der häufig in eingebetteten Systemen eingesetzt wird und sowohl von Xilinx wie auch von Trenz Electronics für die Verwendung mit dem Xilinx Zynq SoC angepasst wurde. Nachdem alle Komponenten geladen wurden, startet der SSBL den Linux Kernel und übergibt ihm die Kontrolle über den ARM-Prozessor. Der Bootvorgang des Systems ist damit abgeschlossen, wobei das AEVV-System erst nach Abschluss der Linux Startphase betriebsbereit wird. Eine Übersicht des gesamten Ablaufs ist in Abbildung 9.3 dargestellt und zeigt die drei Software Komponenten mit ihren jeweiligen Kernaufgaben.

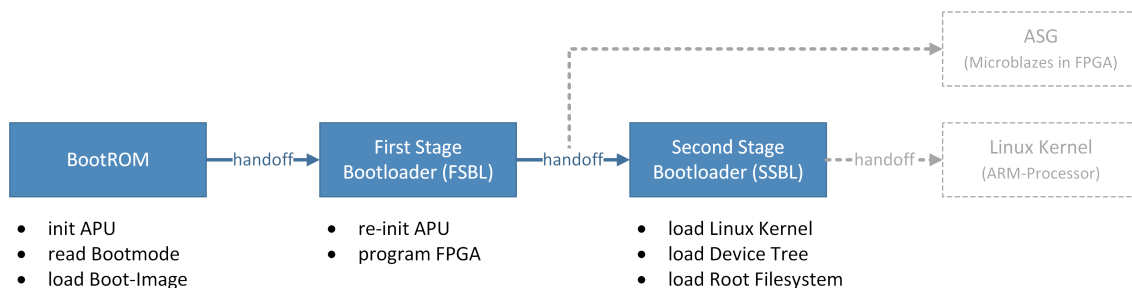


Abbildung 9.3: Ablauf des Bootvorgangs



9.2.2 First Stage Bootloader

Der FSBL ist die erste, vom Benutzer beeinflussbare Software, die auf dem ARM-Prozessor zur Ausführung kommt. Er wird vom Xilinx SDK automatisch generiert, als Eingabe wird die Hardwarebeschreibung aus der Vivado Design Suite benötigt. Über diesen Weg gelangt die Konfiguration des ARM-Prozessors und der Hardwareperipherie, die in Vivado vorgenommen wurde, in das System. Der FSBL wird vom BootROM in das 256 kb große OCM kopiert und ist auf 192 kb begrenzt. Seine Aufgabe ist es, das System entsprechend der Konfiguration aus Vivado zu initialisieren und den FPGA mit dem Bitstream vom Bootmedium zu programmieren. Abschließend lädt er den SSBL vom Bootmedium in den Arbeitsspeicher und übergibt ihm die Kontrolle über den ARM-Prozessor.

Anpassungen In der Standardkonfiguration des Xilinx Zynq SoC ist es möglich die FPGA Konfiguration durch einen Reset, ausgelöst durch Software auf dem ARM-Prozessor, zurückzusetzen. Dieser Reset wird beim Herunterfahren oder Neustarten des AEVV-Betriebssystems ausgelöst. Sollte dies während des Fluges auftreten, ist ein Absturz des Quadropters unvermeidlich. Um dies zu unterbinden wird das SLCR_LOCK Register vor dem Start des ASG und SSBL gesetzt. Ein Reset durch die Software auf dem ARM-Prozessor ist dann nicht mehr möglich (vgl. [Incc, S. 1587]). Den FSBL um die entsprechende Zeile Quellcode zu ergänzen fällt leicht, da bereits mehrere leere Funktionen vorgesehen sind die an kritischen Stellen im Programmablauf aufgerufen werden, sogenannte Hooks. Das SLCR_LOCK Register wird im FsbHookBeforeHandoff-Hook gesetzt, direkt vor dem Start des ASG und SSBL, die entsprechende Anweisung ist in Abbildung 9.4 hervorgehoben.

```

1  u32 FsbHookBeforeHandoff(void) {
    u32 Status;
3   Status = XST_SUCCESS;
    /*
5   * User logic to be added here.
    * Errors to be stored in the status variable and returned
7   */
    fsbl_printf(DEBUG_INFO, "In FsbHookBeforeHandoff function \r\n");
9   fsbl_printf(DEBUG_GENERAL, "Lock the SLCR_LOCK register \r\n");
    Xil_Out32(XPS_SYS_CTRL_BASEADDR + 0x00, 0x00000001);
11  return (Status);
    }

```

Abbildung 9.4: Sperren des SLCR_LOCK Registers

9.2.3 Second Stage Bootloader: U-Boot

Um das AEVV-Betriebssystem zu laden und starten wird U-Boot verwendet. Ein frei verfügbarer Bootloader, der häufig in Kombination mit dem Linux Kernel in eingebetteten



Systemen zum Einsatz kommt, entwickelt wird er von DENX Software Engineering. Xilinx bietet eine angepasste Version speziell für das Xilinx Zynq SoC an, jedoch bietet sie keine Unterstützung für das verwendete TE0720 GigaZee SoC Modul von Trenz Electronic. Eine lauffähige Version für das TE0720 Modul wird nur von Trenz Electronic selbst angeboten, wobei sie nicht mehr aktiv weiterentwickelt wird und mittlerweile über ein Jahr alt ist. Ein Versuch die Anpassungen in die aktuelle Version von Xilinx zu portieren schlug fehl, daher kommt in diesem System die Version von Trenz Electronic zum Einsatz.

Anpassungen Die Taktfrequenz des UART ist in der U-Boot Version von Trenz Electronic auf 50MHz eingestellt. Die Voreinstellung in der Vivado Design Suite und damit auch die Konfiguration der Hardware durch den FSBL ist jedoch auf 100MHz eingestellt. Dadurch kommt es zu fehlerhaften Terminal-Ausgaben über die UART-Schnittstelle, sobald der FSBL die Kontrolle an U-Boot übergibt. Um das Problem zu beheben wurde die Taktfrequenz in U-Boot von 50Mhz auf 100Mhz angehoben.

Eine Änderung, die bereits in die aktuellen U-Boot Versionen von DENX Software Engineering und Xilinx Einzug gehalten hat, betrifft das Speicherlayout für die Komponenten des AEVV-Betriebssystems. In der aktuellen Versionen von Trenz Electronic wird das Wurzeldateisystem noch vor Devicetree und Linux Kernel im Speicher abgelegt und darf maximal 10 MB groß sein. Ist es größer wird dadurch der Devicetree oder sogar der Linux Kernel überschrieben. Das Wurzeldateisystem in diesem Projekt ist mit etwa 28 MB zu groß für das Layout. Daher wurde es aus den aktuellen Versionen von U-Boot übernommen, so dass zunächst der Devicetree, dann der Linux Kernel und anschließend das Wurzeldateisystem im Arbeitsspeicher abgelegt werden. Die Größe des Wurzeldateisystems ist so nur noch durch den verfügbaren Arbeitsspeicher begrenzt.

Um dem Benutzer eine Rückmeldung über den Fortschritt zu geben wurde die User-LED1 des TE0720 GigaZee SoC Moduls verwendet. Sie ist über den System Management Controller des Moduls ansprechbar und ihre Funktion lässt sich konfigurieren. Neben weiteren Möglichkeiten lässt sie sich einschalten, ausschalten oder über einen Multiplexed-IO Kanal durch den ARM-Prozessor kontrollieren [Eleb]. Hierzu wurden die in Abbildung 9.5 dargestellten Befehls-Aliase definiert. Kurz bevor das AEVV-Betriebssystem vom Bootmedium in den Arbeitsspeicher kopiert wird, wird die LED eingeschaltet. Nachdem der Kopiervorgang abgeschlossen ist und bevor das Betriebssystem startet, wird sie auf die Kontrolle durch den ARM-Prozessor konfiguriert, wodurch sie erlischt. Der Benutzer kann so den Fortschritt von U-Boot erkennen.

```
led_on  = mii write 1a 5 7
led_off = mii write 1a 5 6
led_mio = mii write 1a 5 4
```

Abbildung 9.5: Befehls-Aliase zum Setzen der User-LED1 Funktionalität



9.2.4 Devicetree

Der Devicetree ist ein Datenstruktur in der Eigenschaften der Hardware wie z.B. die physikalischen Adressen, passende Treiber oder die Anzahl der CPU's für den Linux-Kernel hinterlegt sind. Er wird vor dem Start des Kernels in den Arbeitsspeicher geladen und seine Speicheradresse an den Kernel übergeben. Der Kernel ist so zu einem gewissen Grad unabhängig von Änderungen der Hardware und es muss nur der Devicetree aktualisiert werden. Der Devicetree für die vorliegende Hardware Plattform wird vom Xilinx SDK aus der Hardwarebeschreibung generiert, er liegt dann in als Textdatei mit einer C ähnlichen Syntax vor. Um diese Datei für den Kernel nutzbar zu machen, muss sie mit dem zum jeweiligen Kernel passenden Devicetree Compiler (DTC) in ein binäres Format umgewandelt werden. Der DTC befindet sich im Kernel-Repository und wird mit ihm zusammen kompiliert [Incb].

Anpassungen Der Eintrag für die Ethernet Schnittstelle des TE0720 GigaZee SoC Moduls wird von der Generierung nicht korrekt erstellt und muss manuell angepasst werden. Erst danach kann die Schnittstelle genutzt werden. Der fehlerhafte, generierte und fehlerfreie, manuell erzeugte Eintrag sind in Abbildung 9.6 dargestellt.

9.2.5 Linux Kernel

Der eingesetzte Linux Kernel stammt von der Firma Xilinx Inc. und wurde für die Verwendung mit dem Xilinx Zynq SoC angepasst und um entsprechende Treiber ergänzt. Er wird in der Version 3.17.0 verwendet, die von Xilinx für die Verwendung mit der Vivado Design Suite 2014.4 vorgesehen ist. Eine funktionstüchtige Basiskonfiguration, die als Grundlage für eigene Anpassungen verwendet werden kann, ist enthalten.

Anpassungen Für die korrekte Funktion der im AEVV-System eingesetzten Hardware mussten die entsprechenden Treiber für Gimbal, Kamera, W-LAN Stick und GPIO aktiviert werden.

9.2.6 Der PPM-IP-Core

Zur Übermittlung der Fernsteuerungsdaten vom Empfänger in das System, wird ein IP-Core benötigt, der das PPM-Signal vom Empfänger aufnimmt, interpretiert und dem System bereitstellt. Da von Xilinx kein fertiger IP-Core existiert, der diese Funktionalität bietet, wurde dieser mithilfe der Vivado Design Suite erstellt. Der detaillierte Aufbau dieses Cores wird im Folgenden erklärt. Abbildung 9.7 zeigt die eingehenden und ausgehenden Signale der PPM Logik.

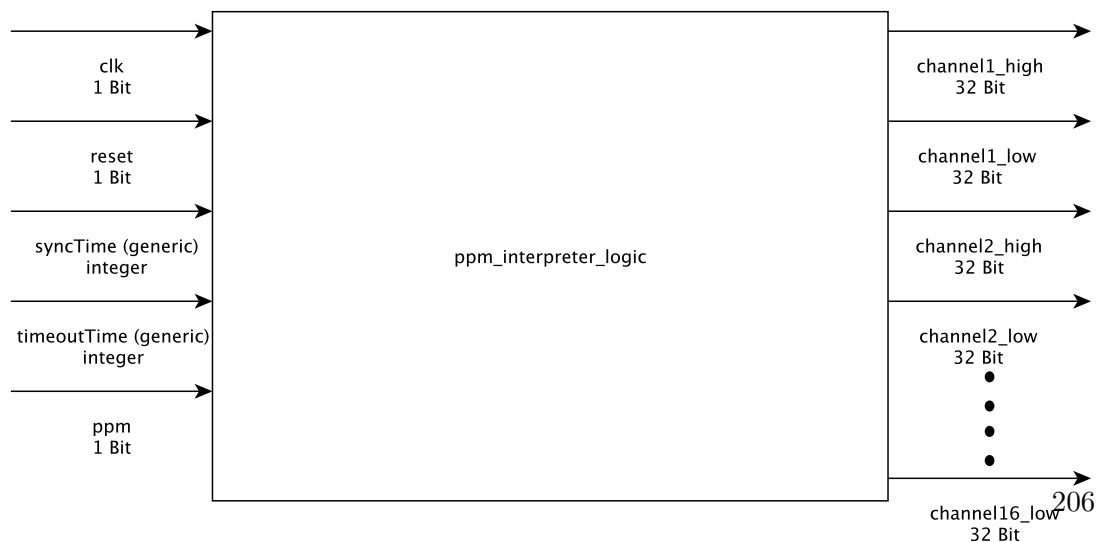
```

1  /* Fehlerhafter, automatisch erstellter Eintrag. */
   &gem0 {
3     local-mac-address = [00 0a 35 00 00 00];
     phy-mode = "rgmii-id";
5     status = "okay";
     xlnx,ptp-enet-clock = <0x69f6bcb>;
7     ps7_ethernet_0_mdio: mdio {
         #address-cells = <1>;
9         #size-cells = <0>;
     };
11 };

13 /* Fehlerfreier, manuell erstellter Eintrag. */
   &gem0 {
15     local-mac-address = [00 0a 35 00 00 00];
     phy-mode = "rgmii-id";
17     status = "okay";
     xlnx,ptp-enet-clock = <0x69f6bcb>;
19     phy-handle = <&phy0>;
     xlnx,enet-reset = "";
21     xlnx,eth-mode = <0x1>;
     ps7_ethernet_0_mdio: mdio {
23         #address-cells = <1>;
         #size-cells = <0>;
25         phy0: phy@0 {
             compatible = "marvell,88e1510";
27             device_type = "ethernet-phy";
             reg = <0x0>;
29             marvell,reg-init = <0x3 0x10 0xff00 0x1e 0x3 0x11 0xffff0 0xa>;
             linux,phandle = <0x2>;
31             phandle = <0x2>;
         };
33     };
};

```

Abbildung 9.6: Fehlerhafter und fehlerfreier Devicetree Eintrag für die Ethernet Schnittstelle des TE0720



Als Eingänge besitzt die Logik das CLK-Signal welches vom AXI-Bus mit 100 MHz bereitgestellt wird, ein Reset-Signal welches ebenfalls am AXI-Bus angeschlossen ist, das zu interpretierende PPM-Signal und zwei einstellbare Eingangssignale die die Synchronisationszeit für das PPM-Signal, sowie die timeout-Zeit die angibt wie lange der Core das Eingangssignal maximal interpretieren soll. Als Ausgänge besitzt der Core 32 Signale die jeweils die High- und Low-Zeiten in ns Auflösung für maximal 16 Kanäle bereitstellt. Der IP-Core ist als State Maschine aufgebaut der das eingehende PPM-Signal interpretiert. Dabei wird das Signal mit der eingehenden CLK-Rate, in diesem Fall mit 100 MHz abgetastet. Mit dieser Geschwindigkeit wird ebenfalls die State-Maschine getriggert. Abbildung 9.8 zeigt den Aufbau der State-Maschine des IP-Cores.

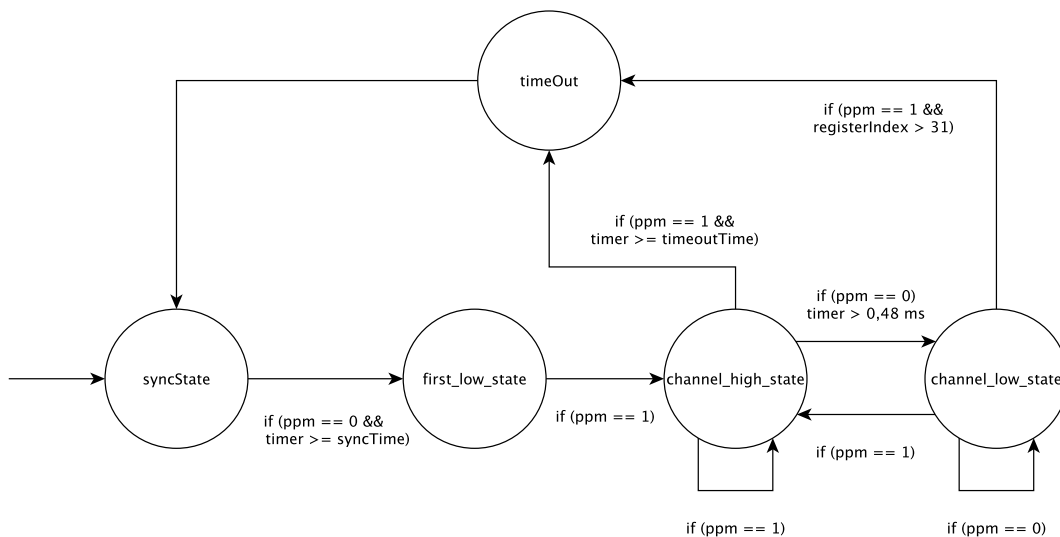


Abbildung 9.8: PPM-IP-Core-Logik Ablauf

Der Anfangszustand ist der syncState. In diesem wird so lange verharrt bis die Synchronisationszeit abgelaufen und danach das PPM-Signal auf low gesetzt wurde. Wird das PPM-Signal auf low gesetzt bevor die Synchronisationszeit abgelaufen ist, wird der interne Timer des Cores zurückgesetzt und es wird wieder auf die Synchronisation gewartet. War die Synchronisation erfolgreich, wird der Zustand gewechselt und im first_low_state auf ein High-Signal des PPM-Eingangs gewartet. Tritt das High-Signal auf, wird in den channel_high_state gewechselt. Hier wird die Zeit des High-Signals aufgezeichnet, solange das PPM-Signal auf High ist. Ist das PPM-Signal länger als die angegebene timeoutTime auf High, wird ein Timeout detektiert und es wird der Zustand zu timeOut gewechselt. Wechselt das PPM-Signal vor der timeoutTime auf low, wird der Zustand zu channel_low_state gewechselt und die High-Zeit für den jeweiligen Channel im internen RAM des Cores gespeichert. Um Störungen auf das PPM Signal herauszufiltern, muss das PPM Signal mindestens 0,48 ms High sein, da es sonst vorkommen kann, dass durch



die hohe Abtastrate eine Störung als Low interpretiert wird. Der `channel_low_state` ist ähnlich aufgebaut wie der `channel_high_state`, es wird hier die Low-Zeit des PPM-Signals aufgezeichnet. Wird in diesem Zustand das PPM-Signal auf High gesetzt wird die Low-Zeit im RAM gespeichert und der Zustand gewechselt. Wurden nun alle 32 RAM-Register der Ausgänge beschrieben, wird in dem `timeOut` Zustand gewechselt. Sind noch nicht alle Register beschrieben worden, wird wieder zum `channel_high_state` gewechselt und das PPM-Signal wird weiter interpretiert.

Im `timeOut` Zustand werden in einem Takt die 32 Werte im RAM-Register, an die 32 Ausgänge des IP-Cores gelegt und danach unbedingt in den `syncState` gewechselt, sodass die State Machine wieder von vorne beginnen kann.

Somit wurde ein PPM-Interpreter-IP-Core erstellt, der die High- und Low-Zeiten eines PPM-Signals, durch die Abtastrate von 100 MHz, auf 10 ns genau auswertet.

9.3 Dataminer

Dieser Abschnitt beschreibt die Funktionsweise des Dataminers. Die Aufgaben des Dataminers bestehen darin, mit der Peripherie zu kommunizieren, die Daten aufzubereiten und diese den anderen Teilsystemen über einen BRAM zur Verfügung zu stellen (siehe Kapitel 7.5).

Die in Abbildung 9.9 dargestellten Komponenten werden im folgenden näher beschrieben.

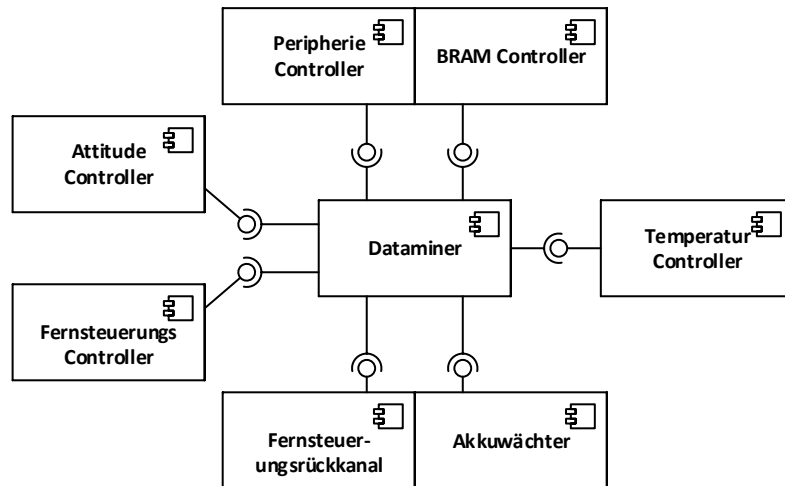


Abbildung 9.9: Komponentendiagramm Dataminer

9.3.1 Akkuwächter

Um ein starkes Unterschreiten von einer LiPo Zellspannung $U_{min,zell}$ von $3,2V$ zu vermeiden, werden zwei MCP3201 Akkuwächter eingesetzt, welche die beiden Versorgungsspannungen messen. Wie bereits erwähnt, wird als Logik Akku ein 3 Zellen N_{logik} und als Energie Akku ein 4 Zellen $N_{energie}$ LiPo-Akkumulator verwendet. Somit beläuft sich die unteren Schwellspannungen der beiden Akkus auf

$$U_{min,energie} = N_{energie} \cdot U_{min,zell} = 4 \cdot 3,2V = 12,8V \text{ und} \quad (9.10)$$

$$U_{min,logik} = N_{logik} \cdot U_{min,zell} = 3 \cdot 3,2V = 9,6V. \quad (9.11)$$

Der MCP3201 liefert über eine SPI Schnittstelle die ausgelesenen ADC Werte. Aus diesen ADC-Werten können, wie in Kapitel 9.1.2.1 und Kapitel 9.1.2.2 beschrieben, die tatsächlichen Spannungswerte der Akkus berechnet werden. Allerdings sind dort idealisierte Werte angenommen und es werden beispielsweise die möglichen Toleranzen in den Spannungsteiler-Widerständen nicht berücksichtigt. Aus diesem Grund wurde für die hier erstellte Bibliothek für den MCP3201 eine Zweipunkt-Kalibrierung vorgenommen. D. h. es wurden zwei tatsächliche Messpunkte genommen und daraus die Steigung der Geraden



ermittelt, die den Zusammenhang zwischen ADC-Wert und Spannung darstellen – jeweils für beide Spannungsbereiche. Für diese Vorgehensweise wurde ein linearer Zusammenhang zwischen ADC-Wert und Spannung angenommen. Weiterhin werden die ausgelesenen Spannungswerte mit der Methode des gleitenden Mittelwertes gefiltert, um hochfrequente Störeinflüsse zu vermeiden.

9.3.2 Peripherie Controller

Der Peripherie Controller steuert folgende Hardwarekomponenten abhängig von dem Dataminer Zustand an:

- ext. Erweiterungsboard
- Debug LED 1
- Debug LED 2
- Summer
- Gimbal

Das **Erweiterungsboard** steuert die LEDs an, welche unter dem Ausleger des Quadropters in Flugrichtung befestigt sind. Diese leuchten dauerhaft, wenn die Spannungen der Akkumulatoren oberhalb ihrer minimalen Spannung liegen. Ist dies nicht der Fall, blinken die LEDs im 2 Hz Takt.

Die **Debug LED 1** leuchtet kontinuierlich, wenn die Initialisierung der gesamten Komponenten am Dataminer abgeschlossen und fehlerfrei ist. Ist diese LED aus, ist die Initialisierung noch nicht abgeschlossen oder es ist währenddessen ein Fehler aufgetreten.

Die **Debug LED 2** blinkt mit einer Frequenz von 10 Hz, wenn der Dataminer arbeitet. Ist diese LED dauerhaft an oder aus, läuft der Prozessor aufgrund eines Fehlers nicht mehr weiter.

Der **Summer** hat die Aufgabe (ähnlich wie die LEDs unter dem Auslegen) die Umgebung über ein akustisches Signal bei Unterspannung der Akkumulatoren zu warnen. Befinden sich diese Spannungen im Betriebsbereich, ist der Summer aus.

Die Versorgungsspannung des **Gimbals** wird über den Dataminer ein- und ausgeschaltet. Nach der erfolgreichen Initialisierung der Komponenten wird die Versorgungsspannung des Gimbals eingeschaltet. Der Dataminer schaltet diese erst aus, wenn die Energiespannung unterhalb des kritischen Wertes liegt, um den Akku zu entlasten.

9.3.3 Fernsteuerungsrückkanal

Wie im Kapitel 9.1.2.1 vorgestellt, werden die Daten, welche an der Fernbedienung angezeigt werden sollen, über eine „One Wire“-UART Bus gesendet. Wie dem Kapitel 1.4 zu entnehmen, werden die Flughöhe sowie die Akkustände an die Fernsteuerung gesendet, damit diese permanent von dem Piloten überwacht werden können.

Die Firma Graupner vermarktet spezielle Module, welche unterschiedlichste Daten an die Fernbedienung schicken können. Diese Daten werden auf einem Display auf der Fernbedienung angezeigt. Eine solches Modul ist das General Air Modul (GAM)⁴⁵. Dieses kann unter anderem zwei unterschiedliche Messspannungen und Temperaturen sowie die aktuelle Höhe an die Fernbedienung senden. Des Weiteren können unterschiedliche akustische Warnsignal auf der Funkfernsteuerung aktiviert werden.

Um zusätzliche Hardware und somit Gewicht einzusparen, wird auf dem Zynq ein GAM emuliert. Dafür wird die UART Schnittstelle auf eine Baudrate von 19200 mit 8 Datenbits, kein Paritybit und einem Stopbit eingestellt.

Der Fernbedienungsempfänger sendet alle 200 ms eine Anfrage mit der GAM-ID 0x8D über den „One Wire“-UART Bus. Ist ein GAM an dem Bus angeschlossen, sendet dieser das Datenframe zurück an die Fernbedienung, welches daraufhin auf dem Display angezeigt wird. Dabei ist darauf zu achten, dass zwischen den einzelnen Daten des Frames eine Verzögerung von ca. 2 ms eingehalten wird. Diese Zeit wird von der Fernsteuerungseinheit benötigt, um das Datum zu verarbeiten. Des Weiteren muss eine Verzögerung von ca. 4 ms nach dem Empfangen der GAM-ID eingehalten werden, bevor das erste Datum des Frames gesendet werden darf. Diese Verzögerung wird von dem Receiver benötigt um den „One Wire“- UART Bus freizugeben.

Abbildung 9.10 zeigt das Zustandsdiagramm der Senderoutine der GAM-Bibliothek, welche alle 2 ms aufgerufen wird. Im Zustand IDLE wird geprüft, ob eine GAM-ID empfangen wurde. Ist eine GAM-ID im Empfangspuffer vorhanden, wird beim nächsten Aufruf in den WARTE Zustand gewechselt, um eine Verzögerung von 4 ms zu erreichen. In den darauffolgenden Aufrufen werden die Daten gesendet. Sind alle Daten gesendet, wird beim nächsten Funktionsaufruf der Empfangspuffer geleert⁴⁶. Im den darauffolgenden Aufruf, wird in den Anfangszustand IDLE gewechselt und auf der nächste Anfrage gewartet.

Die GAM-Daten werden auf der Fernbedienung im oberen Display und über unterschiedlichen Menüausgaben angezeigt (siehe Tabelle 9.2).

⁴⁵GAM: <http://www.graupner.de/de/products/33611/product.aspx> Zugriff: 02.01.2015

⁴⁶Das Leeren des Empfangspuffers ist nötig, da aufgrund der Hardwarebeschaltung des „One Wire“- UARTs die gesendeten Daten über die Kopplung der RX und TX Leitung des Busses ebenfalls im Empfangspuffer geschrieben werden.

9.3 Dataminer

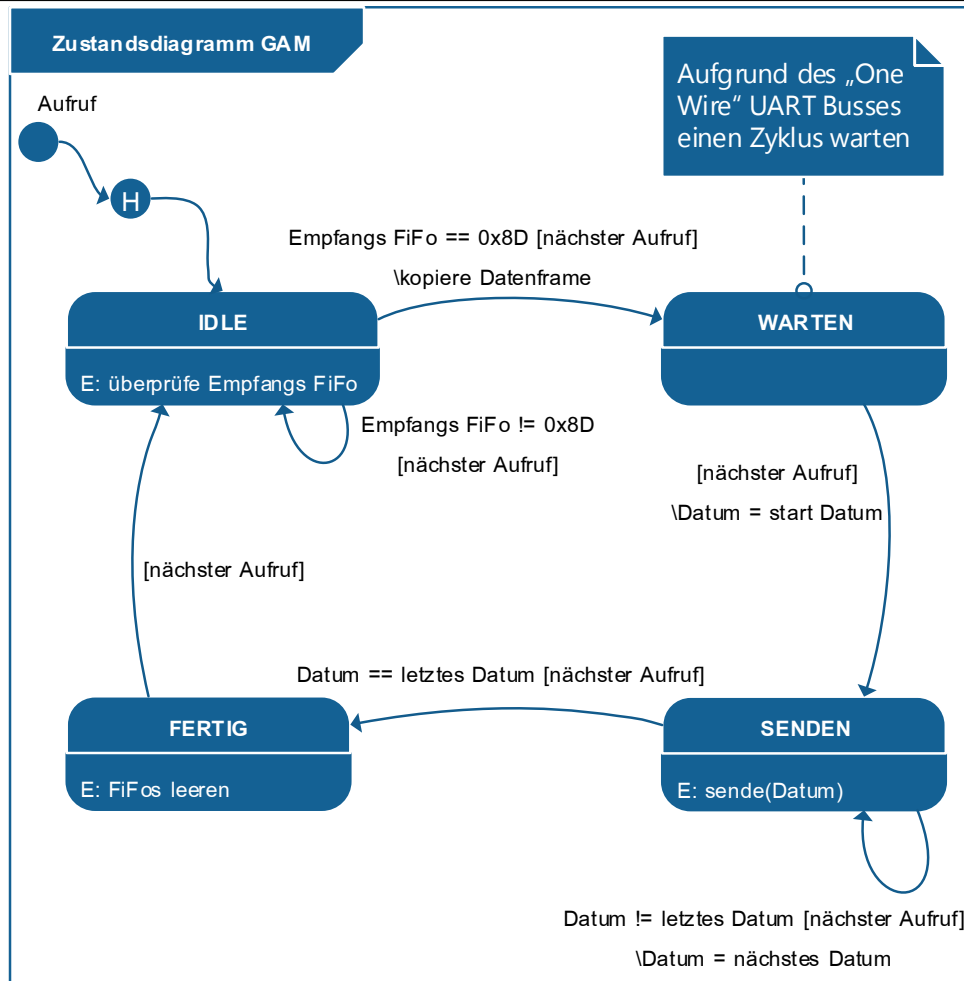







Abbildung 9.10: Zustandsdiagramm der GAM Senderoutine

Tabelle 9.2: Interpretation der angezeigten GAM-Daten auf der Fernsteuerung (FS)

FS Anzeige	Beschreibung
	<ul style="list-style-type: none"> • BAT1 11.1V = Logikspannung [V] • BAT2 15.2V = Energiespannung [V] • T1 52°C = Zynqtemperatur [°C] • T2 30°C = Umgebungstemperatur [°C] • ALT 0m = Flughöhe in [m] • -0m1 = Beschleunigung in [$\frac{m}{s}$] • 0m3 = Beschleunigung in [$\frac{m}{3s}$] • POWER <ul style="list-style-type: none"> – 15.2V = Energiespannung [V] – 6600 = Kapazität Energieakku [mAh] <p>Die restlichen Werte werden nicht genutzt.</p>

Tabelle 9.2: Interpretation der angezeigten GAM-Daten auf der Fernsteuerung (FS) (Fortsetzung)

FS Anzeige	Beschreibung
	<ul style="list-style-type: none"> • 15.1V = Energiespannung [V] • 6600 = Kapazität Energieakku [mAh] • 0m1 = Beschleunigung in [$\frac{m}{s}$]
	<ul style="list-style-type: none"> • 11.1V = Logikspannung [V] • +52°C = Zynqtemperatur [°C]
	<ul style="list-style-type: none"> • 15.2V = Energiespannung [V] • +30°C = Umgebungstemperatur [°C]
	<ul style="list-style-type: none"> • 0m = Flughöhe [m] • -0.0 $\frac{m}{s}$ = Steiggeschwindigkeit [$\frac{m}{s}$]
	<ul style="list-style-type: none"> • Cell0 = minimale Spannung einer Akkuzelle

9.3.4 Temperatursensor Controller

Der Zynq 7020 besitzt eine On-Chip Temperatursensor der über einen ADC ausgelesen wird. Die Genauigkeit dieses Sensors liegt bei $\vartheta_{zynq,error} \pm 4^{\circ}\text{C}$. Die ausgelesene Temperatur wird mit Hilfe der Methode des gleitenden Mittelwertes Tiefpass gefiltert. Dies bewirkt, dass hochfrequente Störeinflüsse herausgefiltert und nur die sich im allgemeinen langsam ändernde Temperatur berücksichtigt wird.

Die ermittelte Temperatur ist die sog. junction Temperatur des Zynqs, welche 125°C nicht überschreiten darf. Eine höhere Temperatur würde den Prozessor irreparabel beschädigen. Um dies zu vermeiden wird zyklisch die Temperatur des Zynqs ausgelesen und unter anderen über den Rückkanal der Fernbedienung dem Piloten mitgeteilt (s. Kapitel 9.3.3).

Die empfohlene junction Temperatur $\vartheta_{zynq,rec}$ liegt bei $\leq 85^{\circ}\text{C}$. Um eine Überhitzung des Prozessors zu vermeiden, wird eine Temperaturschwellwert $\vartheta_{zynq,thres}$ definiert, der bei Überschreitung ein akustisches Warnsignal auf der Fernsteuerung auslöst. Der Temperaturschwellwert wird, wie in Formel 9.12 dargestellt, berechnet.

$$\vartheta_{zynq,thres} = \vartheta_{zynq,rec} - |\vartheta_{zynq,error}| \quad (9.12)$$

$$\vartheta_{zynq,thres} = 85^{\circ}\text{C} - 4^{\circ}\text{C} = 81^{\circ}\text{C} \quad (9.13)$$

Somit hat der Pilot bei Überschreitung des Temperaturschwellwertes noch Zeit, um den Quadrokopter zu landen bevor die maximale junction Temperatur von 125°C überschritten wird.

Als zweite Temperatur wird die Umgebungstemperatur übertragen. Diese wird mit Hilfe des MPU9150 ermittelt und an die Fernbedienung übertragen.

9.3.5 Funkfernsteuerungs Controller

Als Fernbedienung wird die Graupner MC-20 verwendet. Diese sendet im 2,4 GHz Band ein 8 Kanal PPM Signal an den am Quadrokopter befestigten GR-16 Receiver (ebenfalls von Graupner). Abbildung 9.11 zeigt die genutzten Schalter und Hebel der Fernbedienung. Die Schalter befinden sich in Ausgangsstellung, wenn diese von dem Piloten weg zeigen. Dieser Zustand muss vor jedem Start des Quadrokopters eingestellt und überprüft werden.



Abbildung 9.11: Fernsteuerung MC-20

Tabelle 9.3 zeigt die Zuordnung der PPM-Kanäle zu den Funktionen der Fernbedienung. Ebenfalls wird die Interpretation dieser Signale aufgelistet.

Tabelle 9.3: Zuordnung der PPM Kanäle zu den Fernbedienungsfunktionen

PPM-Nr.	Zuordnung	Interpretation
0	Wert des Nickhebels	Nickbewegung des Quadropters
1	Wert des Gierhebels	Gierbewegung des Quadropters
2	Wert des Schubhebels	„Normal“ Schub der Motoren „Höhe halten“ Steig-\Sinkgeschwindigkeit
3	Wert des Rollhebels	Rollbewegung des Quadropters
4	SW ⁴⁷ 9	Ausgangsstellung Flugmodus „Normal“ Endstellung Flugmodus „Höhe halten“
5	SW 5/6	Ausgangsstellung Kamera und Objekterkennung aus („Power-Off Mode“) Mittelstellung Kamera an („Fixed Mode“) Endstellung Kamera und Objekterkennung ein („Searching/Tracking Mode“)
6	SW 1	Ausgangsstellung Motoren aus Endstellung Motoren an
7	unbelegt	-

Der Quadropters kann in zwei Betriebsmodi geflogen werden. Dem „normalen“ Modus, in dem der Schubhebel als solcher interpretiert wird (SW 9: Startstellung) oder im „Höhen halten“ Modus. Hier wird der Schubhebel als Steig- bzw. Sinkgeschwindigkeit interpretiert. Dabei wird beim umlegen des Schalters SW 9 auf Endstellung die aktuelle Höhe und die dazugehörige Hebelstellung als 0-Punkt interpretiert (siehe Kapitel 9.5.1.2 und Kapitel 9.5.1.3)

Die Kamera und die Objekterkennung können von dem Piloten über den Schalter SW 5/6 während des Fluges ein- bzw. ausgeschaltet werden. In Startstellung sind beide Komponenten aus („Power-Off Mode“). In Mittelstellung wird die Kamera eingeschaltet und der Gimbal wird im „Fixed Mode“ betrieben. In Endstellung wird neben der Kamera die Ob-

⁴⁷Schalter (SW)



jekterkennung aktiviert und der Gimbal wird im „Searching Mode“ oder „Tracking Mode“ eingestellt (siehe Kapitel 9.6.12).

Befinden sich alle Bedienelemente der Fernbedienung in Ausgangsstellung, kann der Pilot über den Schalter SW1 den Quadrokopter freigeben. D. h., dass sich erst jetzt die Motoren drehen dürfen. Dadurch wird erreicht, dass der Pilot den Quadrokopter bewusst freigeben muss und dieser vorher die Möglichkeit hat, die restlichen Fernbedienungselemente in eine definierte Position einzustellen. Ist der Quadrokopter einmal freigegeben, können die Motoren nur über die Kombination SW 1 in Startstellung und Schubhebel auf untersten Anschlag komplett ausgeschaltet werden. Andernfalls drehen die Motoren immer mit einem definierten Wert, ohne dass der Kopter abhebt. Diese Vorgehensweise reduziert hohe Anlaufströme, die in der Motoreinschaltphase entstehen.

Kommt es während des Fluges zu einem Verbindungsabbruch zwischen Fernsteuerung und Receiver, ist dieser so eingestellt, dass definierte PPM-Signale gesendet werden. Dabei sind die PPM-Kanalparameter so gewählt, dass diese den schwebenden Quadrokopter langsam zu Boden gleiten lassen.

9.3.6 Lagebestimmung

Auf dem Dataminer wird die jeweils aktuelle Fluglage bestimmt. Hierzu müssen die Sensorwerte ausgelesen und fusioniert werden. In Kapitel 7.3.7 wird beschrieben, welche Sensoren zur Lageerfassung notwendig sind. Im Folgenden wird beschrieben, wie der Ablauf der Sensordatenauswertung auf dem Dataminer umgesetzt wird. Eine detaillierte Beschreibung der Sensordatenverarbeitung findet sich in Kapitel 9.4.

Aus den Anforderungen (siehe Kapitel 6.8.2) ergibt sich, dass Nick- und Rollwinkel mit einer Geschwindigkeit von 500 Hz geregelt werden müssen. Daher müssen auch die benötigten Sensorwerte alle 2 ms aktualisiert werden. Die Berechnung des Gierwinkels und der aktuellen Flughöhe kann mit einer niedrigeren Frequenz erfolgen, da diese für einen stabilen Flug unkritischer sind und weniger hohe Anforderungen an die Dynamik des Systems gestellt werden.

Die zur Berechnung des Gierwinkels benötigten Magnetometerdaten müssen mindestens im 20 ms Takt aktualisiert werden. Die zur Höhenberechnung benötigten Barometerdaten sind alle 30 ms zu aktualisieren.

Aus diesen Anforderungen lässt sich ein 30 ms dauernder Ablaufplan erstellen, der die benötigten Aktualisierungsraten sicher stellt und in einer Endlosschleife aufgerufen wird. Hierzu werden die 30 ms zunächst in 2 ms Zeitschritte unterteilt. Sicherzustellen ist, dass in jedem Zeitschritt der MPU-9150 ausgelesen wird, der die Beschleunigungen und Drehraten bereit stellt. Außerdem muss die Berechnung der fusionierten Roll- und Nickwinkel ebenfalls in jedem Zeitschritt durchgeführt werden. Die restliche Zeit wird zum Starten

von Messungen auf dem BMP085 oder AK8975 bzw. zum Auslesen der Messergebnisse genutzt. In Abbildung 9.12 ist der verwendete Schedule dargestellt.

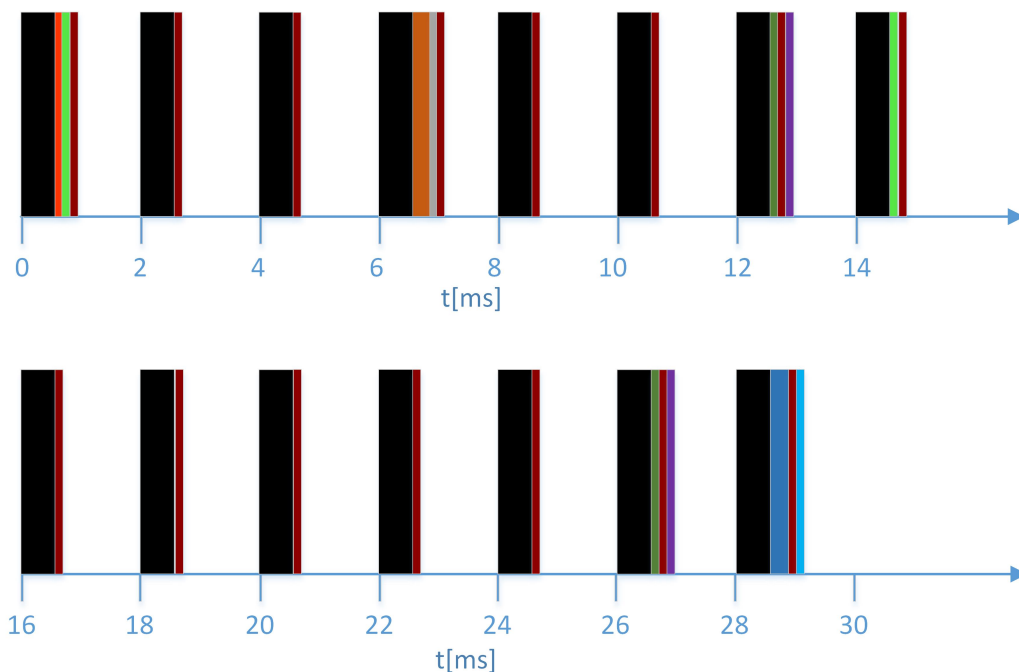
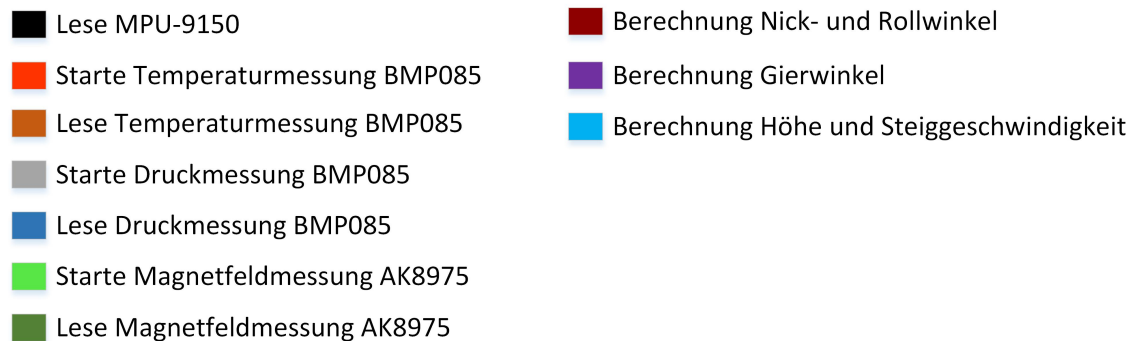


Abbildung 9.12: Ablauf Sensordatenverarbeitung

Der Schedule ist so ausgelegt, dass in jedem Zeitschritt noch ca. 1 ms Rechenzeit übrig bleibt. Ein kleiner Teil dieser Zeit wird für die anderen Dataminer-Aufgaben (siehe Kapitel 9.3) verwendet. Da diese Aufgaben lediglich wenige μs in Anspruch nehmen, bleibt in jedem Zeitschritt ca. 1 ms Puffer übrig.

Im C-Programm für den Dataminer wurde dieser Ablauf mit einer State-machine umgesetzt. Um auf Code-Ausschnitte in der Dokumentation zu verzichten, sei an dieser Stelle auf das entsprechende Doxygen-File verwiesen. Die C-Library wurde so gestaltet, dass lediglich die `updateAttitude()`-Funktion zyklisch im 2 ms Takt von einem Timer aufgerufen werden muss. Die interne State-machine steuert den Programmablauf, so dass die Anforderungen für die Aktualisierungsraten eingehalten werden.

9.3.7 BRAM Controller

Die Daten, welche von weiteren Teilsystemen außerhalb des Dataminers benötigt werden, sind im BRAM abzuspeichern. Diese sind:

- Signale der Fernsteuerung
- Telemetriedaten
- Daten der Flugsensoren

Die Flugregelung arbeitet mit 500 Hz. Die dafür benötigten Sensordaten werden alle 2 ms im BRAM aktualisiert. Daraus resultiert, dass die Regelung mit maximale 2 ms alten Daten arbeitet. Dies ist jedoch aufgrund des eher trägen Gesamtsystems unkritisch. Zu diesen Daten gehören

- der Roll Winkel [deg],
- der Nick Winkel [deg],
- der Gier Winkel [deg],
- die Roll Winkelgeschwindigkeit [deg/s],
- die Nick Winkelgeschwindigkeit [deg/s],
- die Gier Winkelgeschwindigkeit [deg/s],
- die Steiggeschwindigkeit [m/s²],
- die Flughöhe [m] und
- ein Flag, welches valide Daten anzeigt.

Die Telemetriedaten bestehen aus

- der aktuellen Logik- und Energiespannung,
- minimale Zellen-, Logik- und Energiespannung,
- aktuelle Zynq- und Umgebungstemperatur,
- maximale Zynq Temperatur,
- Fehlerdatum des Dataminers und
- die Anzahl der I²C-Kommunikationsfehler zwischen Flugsensoren und Dataminer.

Die Bedeutung des Fehlerdatums des Dataminers ist, wie in Tabelle 9.4 gezeigt, zu interpretieren.

Tabelle 9.4: Interpretation des Fehlerdatums des Dataminers

BitNr.	Wert	Beschreibung
0 (LSB)	0	Sensoren sind kalibriert
	1	Sensoren werden kalibriert
1	0	BRAM fehlerfrei
	1	BRAM Zugriffsfehler

Tabelle 9.4: Interpretation des Fehlerdatums des Dataminers (Fortsetzung)

BitNr.	Wert	Beschreibung
2	0	Logikspannung i.O.
	1	Logikspannung zu gering
3	0	Energiespannung i.O.
	1	Energiespannung zu gering
4	0	Zynqtemperatur i.O.
	1	Zynqtemperatur zu hoch
5	0	Initialisierung der Komponenten i.O.
	1	Initialisierung der Komponenten fehlgeschlagen

Um die Speicheradressen im BRAM zwischen den Teilsysteme (Dataminer, ASG, AEVV) identisch zu halten, wird ein globales Headerfile verwendet, welche alle Speicheradressen enthält. In Abbildung 9.13 ist die Kommunikation der Teilsysteme über die BRAMs dargestellt.

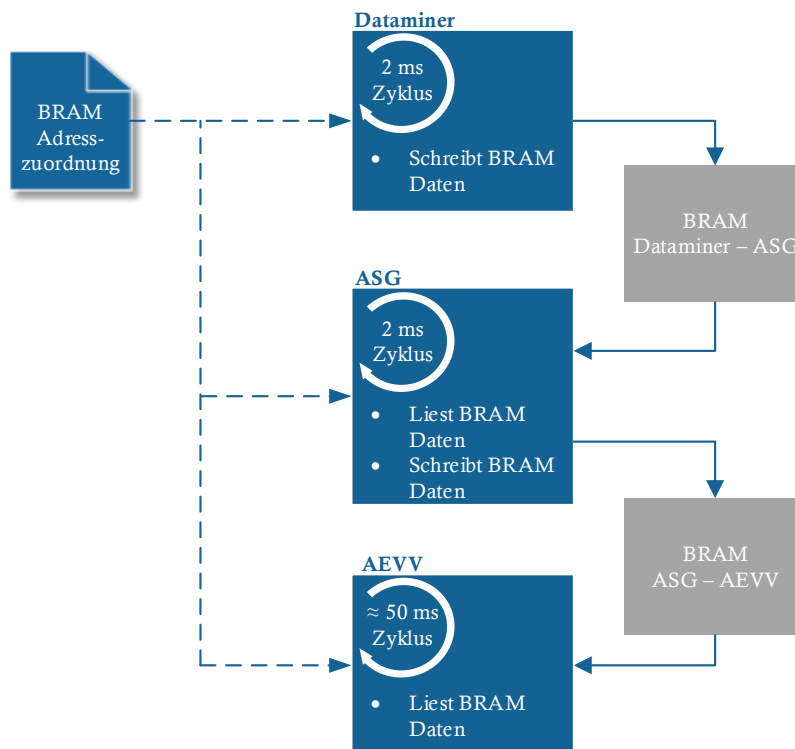


Abbildung 9.13: BRAM Kommunikation zwischen den Teilsystemen

9.3.8 Dataminer Schedule

Auf dem Dataminer werden die beschriebenen Komponenten sequentiell ausgeführt und in 9 Tasks τ_1 bis τ_9 aufgeteilt. Wie auf dem in Abbildung 9.14 dargestellten Ablaufdiagramm, werden nach dem Power-On des Dataminer Softcores die Komponenten initialisiert. Daraufhin wird, bei erfolgreicher Initialisierung, ein 2 ms Timer gestartet. Nach Ablauf der 2 ms, wird der Timer Interrupt ausgeführt, welcher die höchste Priorisierung in dem Dataminer-System hat. In der Interrupt Service Routine (ISR) des Timers werden die I²C Sensoren abgefragt und deren Datenfilterung durchgeführt. Diese Daten werden über einen BRAM dem ASG zur Verfügung gestellt (τ_1). Am Ende dieser ISR, wird ein Updateflag gesetzt. Dieses Flag wird in einer Endlosschleife abgefragt. Ist das Updateflag gesetzt, wird ein sequenzieller Ablauf durchgeführt, um die restliche Peripherie anzusprechen, diese Daten auszuwerten und entsprechend auf den BRAM und dem GAM-Daten Frame zu verteilen (τ_2 bis τ_9). Am Ende dieses Zyklus wird das Updateflag zurückgesetzt und auf den nächsten Timer Interrupt gewartet.

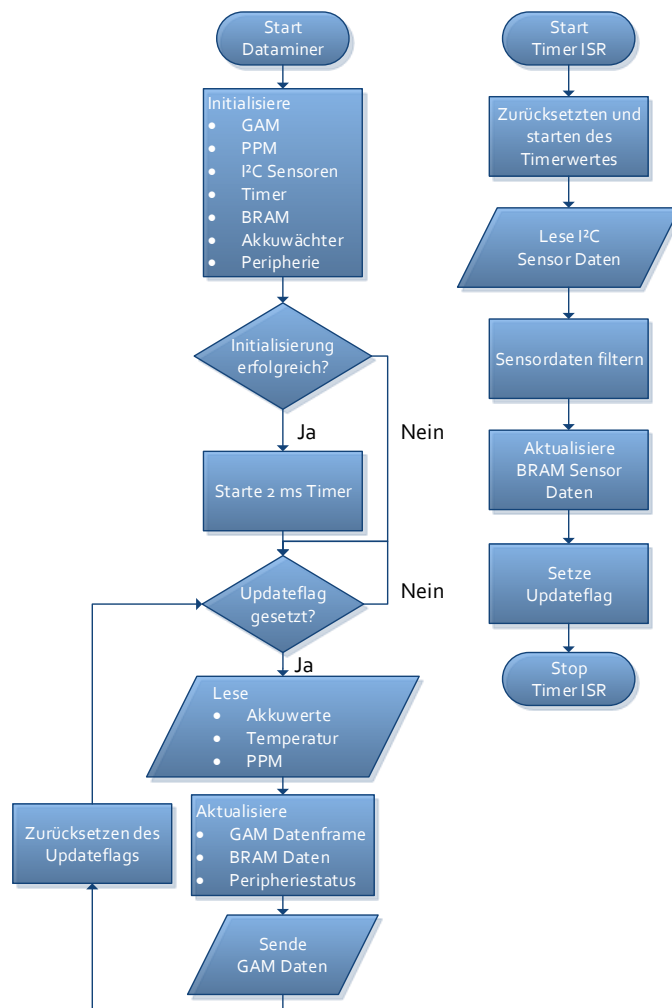


Abbildung 9.14: Ablaufdiagramm der Dataminersoftware



In Tabelle 9.5 werden die einzelnen Task näher beschrieben. Neben der Beschreibung werden ebenfalls die zugehörigen Worst Case Ausführungszeiten c_i (aufgerundete Werte) angegeben. Einige dieser Task haben z. B. interne Zustandsautomaten und somit unterschiedliche Ausführungszeiten, je nach Aufrufzyklus (siehe Abbildung A.7 – Abbildung A.16 im Anhang).

Tabelle 9.5: Auflistung der Tasks und deren Ausführungszeiten

Nr.	Beschreibung	c_i [μs]
τ_1	Liest und fusioniert die Sensorwerte (siehe Kapitel 9.3.6). Anschließend werden diese Werte im BRAM abgelegt.	800
τ_2	Auslesen der Zynq und der MPU9150 Temperatur (siehe Kapitel 9.3.4)	9
τ_3	Auslesen der Spannung des Logik- und Energieakkus (siehe Kapitel 9.3.1)	140
τ_4	Auslesen der PPM-Kanäle (siehe Kapitel 9.3.5)	8
τ_5	Die gemessenen Daten aus τ_1 bis τ_3 werden in das GAM-Daten Frame übernommen (siehe Tabelle 9.2)	39
τ_6	Schreiben der Daten aus τ_2 bis τ_4 in das BRAM, welche für das ASG und/oder dem AEVV System benötigt werden (siehe Kapitel 9.3.7)	35
τ_7	Diese Task wertet ein Fehlerdatum (siehe Tabelle 9.4) aus und setzt entsprechend das Datum für die akustische Ausgabe im GAM-Frame. Da bei der akustischen Warnung nur ein Status ausgegeben werden kann, sind die Warnungen wie folgt priorisiert. <ul style="list-style-type: none"> 1 τ_1 kalibriert die Sensoren 2 Fehler beim BRAM Zugriff 3 Unterschreitung einer Minimalspannung 4 Überschreiten der maximalen Zynqtemperatur 	2
τ_8	Innerhalb dieser Task wird die Senderoutine des Rückkanals der Fernbedienung durchgeführt (siehe Kapitel 9.3.3).	4
τ_9	Auf Basis der möglichen Fehler und des internen Status des Dataminers werden die im Kapitel 9.3.2 beschriebenen Komponenten an den GPIOs des Dataminers ein- bzw ausschaltet.	7

$$c_{sum} = \sum_1^9 c_i = \underline{\underline{1044}}$$

Die Summe der Worst Case Ausführungszeiten c_{sum} liegt bei 1044 μs . D. h. , dass der Dataminer innerhalb der 2 ms alle Tasks abarbeiten kann. Aufgrund des Timerinterrupts



9.3 *Dataminer*

ist gewährleistet, dass die Sensordaten zyklisch aktualisiert und dem ASG zur Verfügung gestellt werden.

9.4 Sensordatenverarbeitung

In diesem Kapitel wird die Verarbeitung der Sensordaten detailliert beschrieben. Das Kapitel ist unterteilt in:

- Winkeldefinition (Kapitel 9.4.1)
- Koordinatentransformation von Sensorchip zu Quadrokopter (Kapitel 9.4.2)
- Transformation der Winkelgeschwindigkeiten in das erdfeste Koordinatensystem (Kapitel 9.4.3)
- Sensordatenfusion über Komplementärfilter (Kapitel 9.4.4)

9.4.1 Winkeldefinition

Das in Kapitel 3.1.2 auf Abbildung 3.4 dargestellte Koordinatensystem wird zur Regelung des Quadrokopters verwendet. Im Folgenden werden die Winkel definiert, die als Eingangsgrößen der Lageregelung dienen.

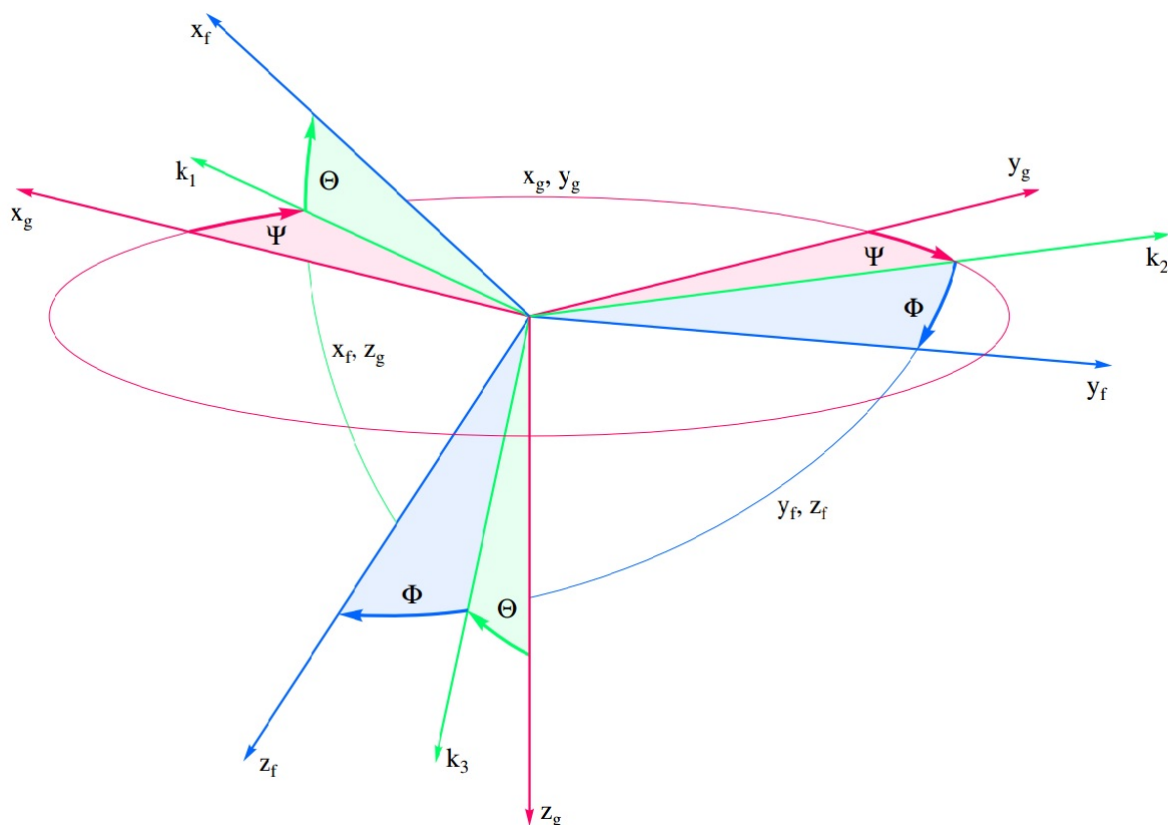


Abbildung 9.15: Flugfestes (blau) und erdfestes (rot) Koordinatensystem [Buc]

Die Lage eines Fluggerätes wird üblicherweise über die Winkel θ , ϕ und ψ beschrieben (siehe Abbildung 9.15). Zu sehen sind zwei Koordinatensysteme. In rot ist das erdfeste (geodätische) und in blau das flugfeste Koordinatensystem dargestellt. Das flugfeste Ko-



ordinatensystem bewegt sich mit dem Flugobjekt mit. Hierbei ist die Flugrichtung in Richtung der dargestellten x_f -Achse definiert. Die y_f -Achse würde bei einem Flugzeug in Richtung einer Tragfläche zeigen, was bei einem Quadrokopter allerdings nicht zutrifft. Die z_f -Achse ist nach unten zeigend definiert und zeigt daher bei horizontaler Fluglage zum Erdmittelpunkt.

Das erdfeste Koordinatensystem zeigt mit der z_g -Achse zum Erdmittelpunkt. Die x_f -Achse zeigt in Ausgangsflugrichtung und die y_f -Achse steht senkrecht auf den anderen beiden. Über die Winkel θ , ϕ und ψ wird nun beschrieben, wie das flugfeste Koordinatensystem (gedanklich am Quadrokopter befestigt) zum erdfesten Koordinatensystem ausgerichtet ist. Nach der z, y', x'' - Konvention erfolgt die Überführung des erdfesten in das flugfeste Koordinatensystem durch drei hintereinander durchgeführte Drehungen:

1. Drehung um die Z-Achse um ψ (Gierwinkel)
2. Drehung um die neue Y-Achse um θ (Nickwinkel)
3. Drehung um die neue X-Achse um ϕ (Rollwinkel)

Diese drei Drehungen sind in Abbildung 9.15 anschaulich dargestellt. Mit der ersten Drehung um die z_g -Achse um ψ wird die x_g -Achse in die k_1 -Achse überführt sowie die y_g -Achse in die k_2 -Achse. Der Winkel ψ beschreibt daher das Gieren des Quadrokopters. Mit der zweiten Drehung um θ wird die k_1 -Achse in die x_f -Achse überführt und die z_g -Achse in die k_3 -Achse. Der Winkel θ beschreibt das Nicken des Quadrokopters. Die dritte Drehung um die x_f -Achse beschreibt das Rollen des Quadrokopters. Hierbei wird die k_2 -Achse in die y_f -Achse überführt sowie die k_3 -Achse in die z_f -Achse. [Buc]

9.4.2 Koordinatentransformation von Sensorchip zu Quadrokopter

Bei der gewählten Einbauweise des MPU-9150 Sensorchips zeigt die Z-Achse des auf dem Chip definierten Koordinatensystems nach oben. Da nach Kapitel 9.4.1 die Z-Achse des flugfesten Koordinatensystems nach unten zeigt, wird zunächst ein Mapping der Sensorwerte auf ein Zwischenkoordinatensystem vorgenommen. Bei dem Zwischenkoordinatensystem zeigt die Z'-Achse zum Erdmittelpunkt. Das Mapping von MPU9150 auf das Zwischenkoordinatensystem wird in Abbildung 9.16 dargestellt.

Hierzu werden die Sensorwerte folgendermaßen gemappt:

$$Acc'_X = Acc_Y(MPU) \quad (9.14)$$

$$Acc'_Y = Acc_X(MPU) \quad (9.15)$$

$$Acc'_Z = -Acc_Z(MPU) \quad (9.16)$$

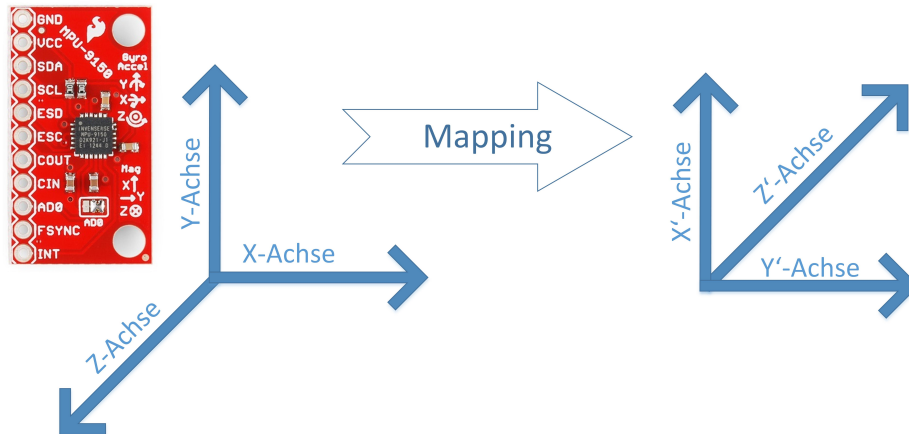


Abbildung 9.16: Mapping Koordinatensystem MPU9150 in Zwischenkoordinatensystem

$$Gyro'_X = Gyro_Y(MPU) \quad (9.17)$$

$$Gyro'_Y = Gyro_X(MPU) \quad (9.18)$$

$$Gyro'_Z = -Gyro_Z(MPU) \quad (9.19)$$

Um dieses Zwischenkoordinatensystem in das Quadrokoopterkoordinatensystem zu überführen, muss dieses im Anschluss noch gedreht werden, da dieses zum tatsächlichen Quadrokoopter-System noch 45° gedreht ist. Mit einer 2D-Rotation des Zwischenkoordinatensystems wird diese Drehung kompensiert.

Rotation der Beschleunigungssensorwerte:

$$\begin{pmatrix} Acc_x \\ Acc_y \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \cdot \begin{pmatrix} Acc'_X \\ Acc'_Y \end{pmatrix} \quad (9.20)$$

Rotation der Gyroskopsensorwerte:

$$\begin{pmatrix} Gyro_x \\ Gyro_y \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \cdot \begin{pmatrix} Gyro'_X \\ Gyro'_Y \end{pmatrix} \quad (9.21)$$

9.4.3 Winkelgeschwindigkeiten im erdfesten Koordinatensystem

Die gemessenen Winkelgeschwindigkeiten über die Gyroskope beziehen sich immer auf das flugfeste Koordinatensystem, da sich der Sensorchip auf dem Quadrokoopter mit bewegt. Diese Winkelgeschwindigkeiten müssen in Abhängigkeit der aktuellen Fluglage ausgewertet und den Winkeln θ , ϕ und ψ zugeordnet werden. An einem Beispiel kann schnell gezeigt werden, warum diese Transformation der Winkelgeschwindigkeiten vom flugfesten in das erdfeste Koordinatensystem notwendig ist.

Beispiel: Sei der Quadrocopter um 90° in der Rollachse ausgelenkt ($\phi = 90^\circ$). Dann führt eine Drehung im flugfesten Koordinatensystem um die Y-Achse nicht zu einem Nicken des Quadrocopters sondern zu einem Gieren. Die um die Y-Achse gemessene Winkelgeschwindigkeit betrifft dann den Winkel ψ .

Die Winkelgeschwindigkeiten $\dot{\theta}$, $\dot{\phi}$ und $\dot{\psi}$ werden aus den im flugfesten Koordinatensystem gemessenen Winkelgeschwindigkeiten $Gyro_x$, $Gyro_y$ und $Gyro_z$ wie folgt berechnet [Sch]:

$$\dot{\theta} = Gyro_y \cdot \cos(\phi) - Gyro_z \cdot \sin(\phi) \quad (9.22)$$

$$\dot{\phi} = Gyro_x + (Gyro_y \cdot \sin(\phi) + Gyro_z \cdot \cos(\phi)) \cdot \frac{\sin(\theta)}{\cos(\theta)} \quad (9.23)$$

$$\dot{\psi} = \frac{Gyro_y \cdot \sin(\phi) + Gyro_z \cdot \cos(\phi)}{\cos(\theta)} \quad (9.24)$$

9.4.4 Sensordatenfusion über Komplementärfilter

Viele, für die Regelung des Quadrocopters relevante Zustandsgrößen sind nicht direkt messbar, sondern müssen aus Sensorwerten indirekt abgeleitet werden. Hierzu gehört z.B. die Fluglage. Im Folgenden wird das Komplementärfilter vorgestellt, als eine Möglichkeit, Sensorwerte verschiedener Sensoren zur Ermittlung der gleichen Zustandsgröße zu fusionieren. Anschließend wird die Lageermittlung und die Höhenermittlung mit einem Komplementärfilter beschrieben.

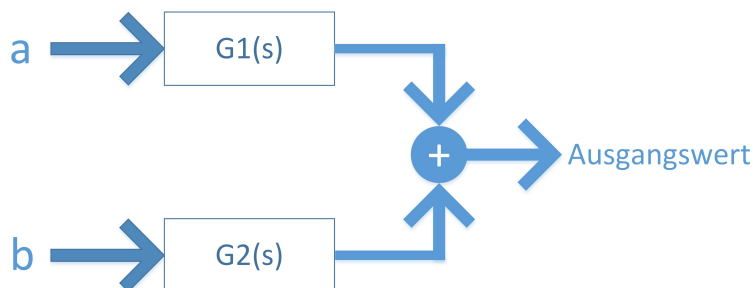


Abbildung 9.17: Komplementärfilter

Die Grundidee eines Komplementär-Filters ist es, unterschiedliche Sensoren für unterschiedliche Frequenzbereiche der gleichen Zustandsgröße zu nutzen. Die Auslegung des Filters geschieht hierzu im Laplace-Raum.

Die grundlegende Funktionsweise des Komplementärfilters ist in Abbildung 9.17 dargestellt. Seien a und b zwei Messungen der selben Zustandsgröße über unterschiedliche Sensoren, so wird über die Messung a mit der Übertragungsfunktion $G1(s)$ und die Messung b mit der Übertragungsfunktion $G2(s)$ multipliziert. Die beiden Produkte werden anschlie-



ßend addiert.[W.]

Die Übertragungsfunktionen $G_1(s)$ und $G_2(s)$ werden so ausgelegt, dass die Summe eins ergibt:

$$G_1(s) + G_2(s) = 1 \quad (9.25)$$

Sei nun $G_1(s)$ ein Tiefpassfilter 1. Ordnung so ergibt sich $G_2(s)$ nach Formel 9.26

$$G_2(s) = 1 - G_1(s) \quad (9.26)$$

als ein Hochpassfilter 1. Ordnung. Die Amplitudengänge $|G_2(j\omega)|$ und $|G_1(j\omega)|$ summieren sich für jede Frequenz $f = \frac{1}{2\pi \cdot \omega}$ zu eins. Durch die Bedingung Formel 9.25 ist somit sichergestellt, dass im Ausgangssignal alle Frequenzen gleichmäßig verstärkt werden.

Ein Komplementärfilter ist daher sehr gut geeignet um Gyroskopsensorwerte (verwendbar im hohen Frequenzbereich) mit Beschleunigungssensorwerten (verwendbar im niedrigen Frequenzbereich) sowie Magnetometerwerten (verwendbar im niedrigen Frequenzbereich) oder Barometerwerten (verwendbar im niedrigen Frequenzbereich) zu fusionieren.

9.4.4.1 Lagewinkelermittlung mittels Komplementärfilter Wie in Kapitel 7.3.7 beschrieben, ist zur Ermittlung der Fluglage eine Fusion von Beschleunigungssensordaten und Gyroskopdaten erforderlich. Ein Komplementärfilter bietet sich hierzu an, da die Beschleunigungssensorwerte nur für ein ruhendes (bzw. gleichförmig bewegtes) System sinnvoll ausgewertet werden können. Gyroskope haben sehr gute dynamische Eigenschaften, weisen allerdings keine Langzeitstabilität auf, da durch die Integration Messfehler immer weiter aufaddiert werden.

Die Auslegung des Filters ist hierbei für Roll- u. Nickwinkel gleich. Die Vorgehensweise wird nachfolgend anhand des Nickwinkels θ erläutert.

9.4.4.2 Ermittlung des Nickwinkels θ

Aus den Beschleunigungssensorwerten kann der Winkel θ über die *asin*-Funktion aus der Beschleunigung in X-Richtung (Acc_x) und der Erdbeschleunigung ermittelt werden.

$$\theta_{acc} = asin\left(\frac{Acc_x}{9.81 \frac{m}{s^2}}\right) \quad (9.27)$$

Aus den Gyroskopwerten kann der Nickwinkel durch die Integration der Winkelgeschwindigkeit $\dot{\theta}$ über die Zeit ermittelt werden. Die Berechnung von $\dot{\theta}$ aus den gemessenen Gyroskopwerten wird in Kapitel 9.4.3 beschrieben.

$$\theta_{Gyro} = \int \dot{\theta}_{Gyro} dt \quad (9.28)$$

Die abgeleiteten Winkel θ_{acc} und θ_{Gyro} werden nun über ein Komplementärfilter fusioniert. Zur Filterung der Beschleunigungssensorwerte wird hierzu die Übertragungsfunktion $G_1(s)$ als Tiefpass erster Ordnung ausgelegt (siehe Formel 9.29).

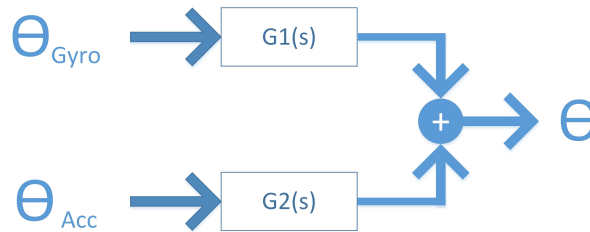


Abbildung 9.18: Komplementärfilter Nickwinkel

$$G_1(s) = \frac{1}{1 + Ts} \quad (9.29)$$

Die Grenzfrequenz, ab der das Tiefpassfilter um -3dB dämpft, kann durch Wahl der Zeitkonstante T bestimmt werden. Der Amplitudengang des Tiefpassfilters aus Formel 9.29 wird beschrieben durch:

$$|G_1(j\omega)| = \frac{1}{\sqrt{1 + \omega^2 T^2}} \quad (9.30)$$

Eine Dämpfung um 3dB ergibt sich nach Formel 9.30 bei

$$\omega_{gr} = \frac{1}{T} \quad (9.31)$$

Somit lässt sich die Zeitkonstante T mit $\omega = 2\pi f_{gr}$ für eine gewünschte Grenzfrequenz f_{gr} berechnen nach

$$T = \frac{1}{2\pi \cdot f} \quad (9.32)$$

Die Übertragungsfunktion $G_2(s)$ wird nach der Formel 9.33 bestimmt, so dass $G_2(s)$ einen zu $G_1(s)$ komplementären Frequenzgang aufweist.

$$G_2(s) = 1 - G_1(s) = 1 - \frac{1}{1 + Ts} = \frac{1 + Ts}{1 + Ts} - \frac{1}{1 + Ts} = \frac{Ts}{1 + Ts} \quad (9.33)$$

Für die Übertragungsfunktion des Gesamtsystems ergibt sich

$$\theta = \theta_{acc} \cdot G_1(s) + \theta_{Gyro} \cdot G_2(s) \quad (9.34)$$



Die Integration, die zur Ermittlung des Winkels aus den Gyroskopdaten notwendig ist, wird im Laplace-Raum zu einer einfachen Multiplikation mit $\frac{1}{s}$.

$$\theta = \theta_{acc} \cdot G_1(s) + \dot{\theta}_{Gyro} \cdot \frac{1}{s} \cdot G_2(s) \quad (9.35)$$

Die Laplace-Transformation eignet sich zur Beschreibung zeitkontinuierlicher Systeme. Da die Verarbeitung der Sensorwerte allerdings digital auf einer CPU in zeitdiskreten Schritten geschieht, ist einer Überführung der Laplacetransformierten in den sog. Z-Bereich notwendig. Mit Hilfe der Z-Transformation können zeitdiskrete Systeme mathematisch korrekt beschrieben werden.

Die Transformation dieser Systemübertragungsfunktion in den Z-Bereich zur Beschreibung zeitdiskreter Systeme wird über eine Ersetzung der Laplacevariable s nach der "Backwards-Differenz"-Methode vorgenommen [W.]:

$$s = \frac{1}{\Delta t} \cdot (1 - z^{-1}) \quad (9.36)$$

Hierbei ist Δt das Abtastintervall. Setzt man in die Formel 9.35 die Übertragungsfunktionen $G_1(s)$ und $G_2(s)$ ein und substituiert s (siehe Formel 9.36), so ergibt sich für die Übertragungsfunktion im Z-Bereich:

$$\theta = \theta_{acc} \cdot \frac{1}{1 + T(\frac{1}{\Delta t} \cdot (1 - z^{-1}))} + \dot{\theta}_{Gyro} \cdot \frac{1}{\frac{1}{\Delta t} \cdot (1 - z^{-1})} \cdot \frac{T(\frac{1}{\Delta t} \cdot (1 - z^{-1}))}{1 + T(\frac{1}{\Delta t} \cdot (1 - z^{-1}))} \quad (9.37)$$

Aus dieser Z-Transformierten kann die Differenzgleichung hergeleitet werden, die das System im Zeitbereich beschreibt und somit den zu implementierenden Algorithmus:

$$\theta_k = \alpha(\theta_{k-1} + \dot{\theta}_k \cdot \Delta t) + (1 - \alpha)\theta_{acc} \quad (9.38)$$

$$\text{mit } \alpha = \frac{\frac{T}{\Delta t}}{1 + \frac{T}{\Delta t}} \quad (9.39)$$

wie in Formel 9.38 zu sehen, steckt die Zeitkonstante T , über die die Grenzfrequenz des Hoch- und des Tiefpassfilters bestimmt wird, in dem Faktor α . Zur Auslegung dieses Komplementärfilters muss nun lediglich noch die Zeitkonstante T so gewählt werden, dass einerseits der Drift der Gyroskope über die Beschleunigungssensoren ausgeglichen werden kann. Andererseits darf der Einfluss der Beschleunigungssensoren nicht zu groß sein, da diese für schnelle Bewegungen keine verwendbaren Werte liefern.

9.4.4.3 Ermittlung des Rollwinkels ϕ

Der Rollwinkel wird genauso fusioniert wie der Nickwinkel (s. Kapitel 9.4.4.2). Aus den



Beschleunigungssensorwerten berechnet sich der Rollwinkel nach Formel 9.40 unter Einbeziehung des Winkels θ . Die Bestimmung des Rollwinkels aus den Gyroskopsensorwerten (Formel 9.41) und die anschließende Sensordatenfusion (Formel 9.42) ist äquivalent zur Bestimmung des θ -Winkels (s. Kapitel 9.4.4.2).

$$\phi_{Acc} = \text{asin}\left(\frac{-Acc_y \cdot \cos(\theta)}{9.81 \frac{m}{s^2}}\right) \quad (9.40)$$

$$\phi_{Gyro} = \int \dot{\phi}_{Gyro} dt \quad (9.41)$$

$$\phi_k = \alpha(\phi_{k-1} + \dot{\phi}_k \cdot \Delta t) + (1 - \alpha)\phi_{acc} \quad (9.42)$$

9.4.4.4 Ermittlung einer geeigneten Grenzfrequenz für Nick- und Rollwinkel

Um heraus zu finden, für welches α das Komplementärfilter sowohl gute dynamische Eigenschaften aufweist aber auch den Drift der Gyroskope kompensiert, wird ein Versuch durchgeführt. Zunächst wird das System im Ruhezustand mit den realen Sensorwerten betrieben. Der Faktor α wird auf 1 gesetzt. In diesem Zustand werden lediglich die Gyroskopwerte zur Winkelermittlung berücksichtigt ($\alpha = 1 \Rightarrow T \rightarrow \infty \Rightarrow f_{gr} = 0$). Durch den Drift der Gyroskopwinkel entfernt sich mit der Zeit die ermittelte Lage immer weiter von der tatsächlichen Lage. Im nächsten Schritt wird α immer weiter verkleinert, bis der Drift schließlich von dem größeren Einfluss der Beschleunigungssensorwinkel kompensiert werden kann. Dieser Punkt hat sich bei $\alpha = 0.98$ eingestellt, was einer Grenzfrequenz von $f_{gr} = 0.16 \text{ Hz}$ entspricht.

Die für den Versuch verwendete Hardware ist nur in der Lage ist, die Sensoren im 20 ms Takt abzufragen. Daher sind die ermittelten Ergebnisse nur als Richtwerte für die Sensordatenverarbeitung auf der Zielplattform zu interpretieren, bei der die Sensorwerte alle 2 ms abgerufen und fusioniert werden.

In Abbildung 9.19 ist der Messverlauf für $\alpha = 0.98$ und $\Delta t = 20 \text{ ms}$ über 5 s dargestellt. Es kann eine Rauschamplitude mit ca. $+0.035^\circ$ abgelesen werden. Diese Genauigkeit ist ausreichend für eine gute Regelung der Fluglage. Bei einer weiteren Verkleinerung von α steigt die Rauschamplitude deutlich an, weil die Beschleunigungssensorwerte wesentlich mehr Rauschen als die Gyroskopsensorwerte.

Zur Demonstration der guten dynamischen Eigenschaften der Sensordatenfusion wird eine Messung über 5 s gemacht, bei der die Nick-Achse zweimal leicht ausgelenkt wurde. Diese Messung ist in Abbildung 9.20 dargestellt. Gut zu erkennen ist, dass nach der zweimaligen Auslenkung die Null-Lage umgehend wieder richtig ermittelt wird. Das entwickelte Komplementärfilter hat somit gute dynamische und statische Eigenschaften und kann zur Lageermittlung der Quadrocopters genutzt werden.

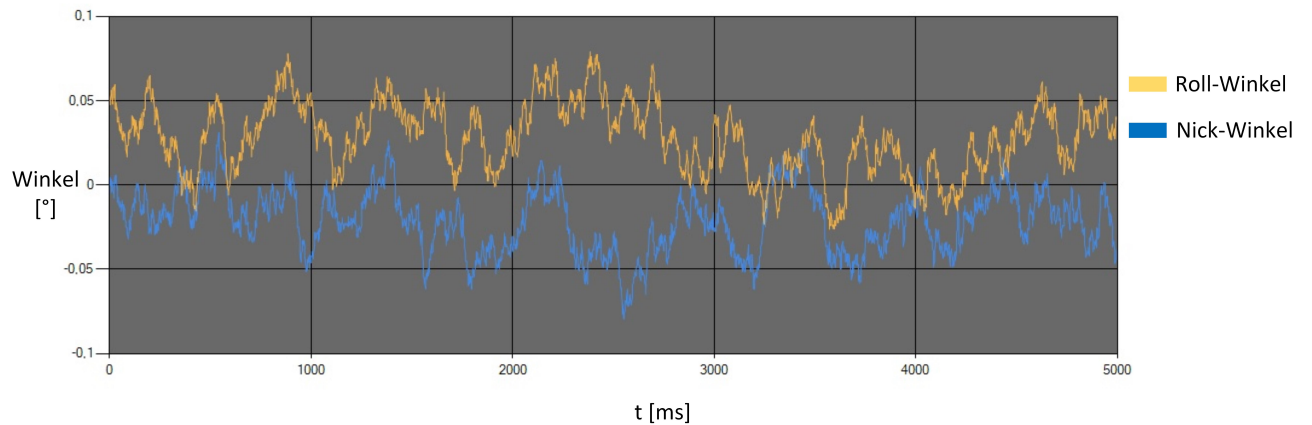


Abbildung 9.19: Nick- und Roll-Achse in Ruhelage

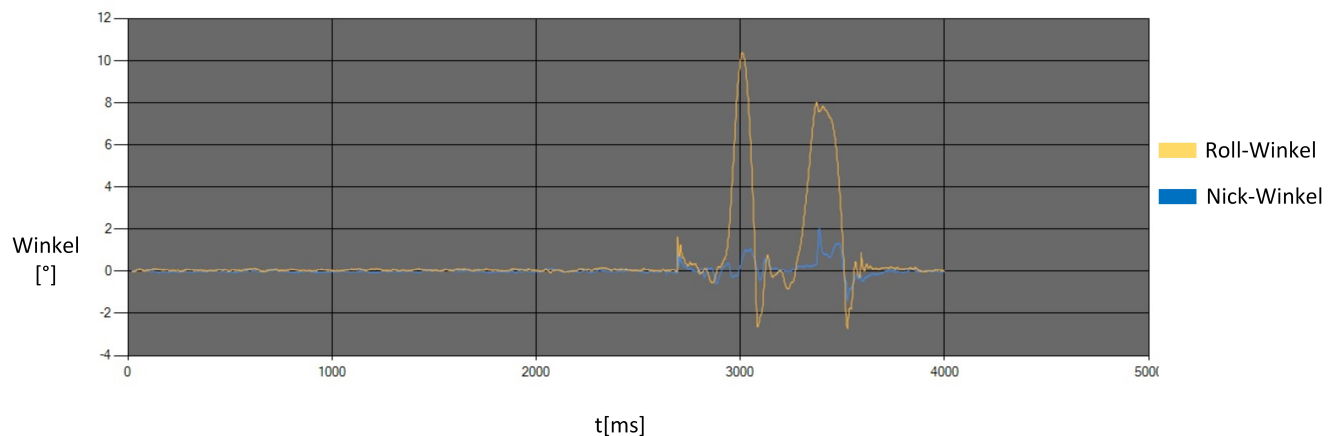


Abbildung 9.20: Nick- und Roll-Achse mit Bewegung

Auf der Zielplattform konnte die Grenzfrequenz noch weiter verringert werden, ohne dass die Winkelwerte weg driften. Dies ist bedingt durch die höhere Abtastrate von 500 Hz. Hier hat sich ein α von 0.999 als optimal heraus gestellt, was einer Grenzfrequenz $f_{gr} = 0.08 \text{ Hz}$ entspricht.

9.4.4.5 Ermittlung des Gierwinkels ψ

Zur Ermittlung des Gierwinkels werden ebenfalls in einem Komplementärfilter Gyroskopwerte und Magnetometerwerte fusioniert. Wie bei der Ermittlung der Nick- und Rollwinkel wird das Filter zunächst im Laplace-Raum ausgelegt und anschließend über die Z-Transformation in eine Differenzgleichung überführt. Hierbei werden die Gyroskopdaten für die hohen Frequenzen genutzt und der über das Magnetometer ermittelte Gierwinkel für die niedrigen Frequenzen. Im Folgenden wird zunächst beschrieben, wie die

Ausrichtung anhand der Magnetometerdaten berechnet werden kann. Anschließend wird die Sensordatenfusion beschrieben.

Lagekompensierter Kompass Mit Hilfe eines 3-Achs-Magnetometer ist es möglich, die Ausrichtung des Quadropters im Bezug zum magnetischen Norden zu messen. Der Magnetfeldvektor steht in Deutschland um $\alpha \sim 60^\circ$ nach unten geneigt, wobei die horizontale Komponente in Richtung des magnetischen Nordens zeigt (siehe Abbildung 9.21). Zur Ermittlung der Ausrichtung ist lediglich die horizontale Komponente von Belangen.

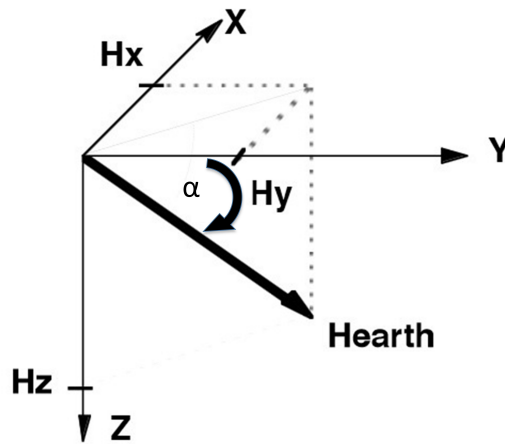


Abbildung 9.21: Ausrichtung des Erdmagnetfeldvektors

Wenn der Quadroptor parallel zur Erdoberfläche steht ($\theta=0$ und $\phi=0$), kann die Ausrichtung des Quadropters über die atan-Funktion aus der gemessenen magnetischen Feldstärke in X- und Y-Richtung ermittelt werden (siehe Formel 9.43) [STMb]. In Abbildung 9.22 ist die Ermittlung des Gierwinkels ψ aus den gemessenen Feldstärken M_{x1} und M_{x2} dargestellt.

$$\psi = \text{atan}\left(\frac{M_{y1}}{M_{x1}}\right) \quad (9.43)$$

Wenn der Quadroptor allerdings in Nick- oder Rollachse geneigt ist ($\theta \neq 0$ oder $\phi \neq 0$), muss die horizontale Komponente des Erdmagnetfeldes unter Einbeziehung der in Z-Richtung wirkenden Feldkomponente berechnet werden, da X- und Y-Achse nicht mehr in der horizontalen Ebene liegen (siehe Formel 9.44 ff.) [STMb].

$$M_{x2} = M_{x1} \cdot \cos(\theta) + M_{z1} \cdot \sin(\theta) \quad (9.44)$$

$$M_{y2} = M_{x1} \cdot \sin(\phi) \cdot \sin(\theta) + M_{y1} \cdot \cos(\phi) - M_{z1} \cdot \sin(\phi) \cdot \cos(\theta) \quad (9.45)$$

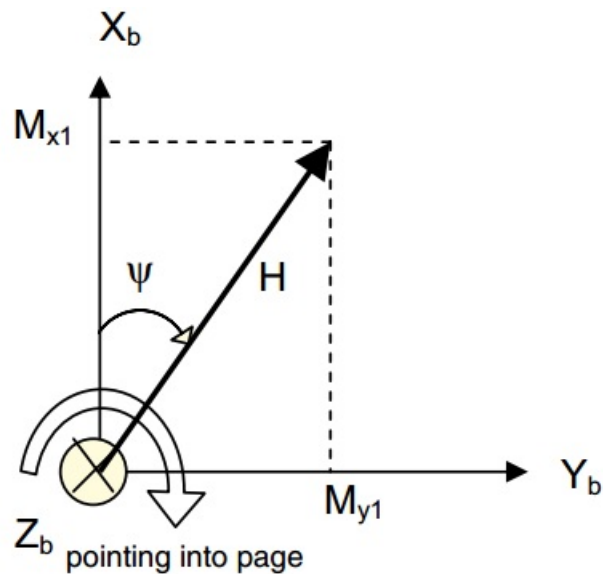
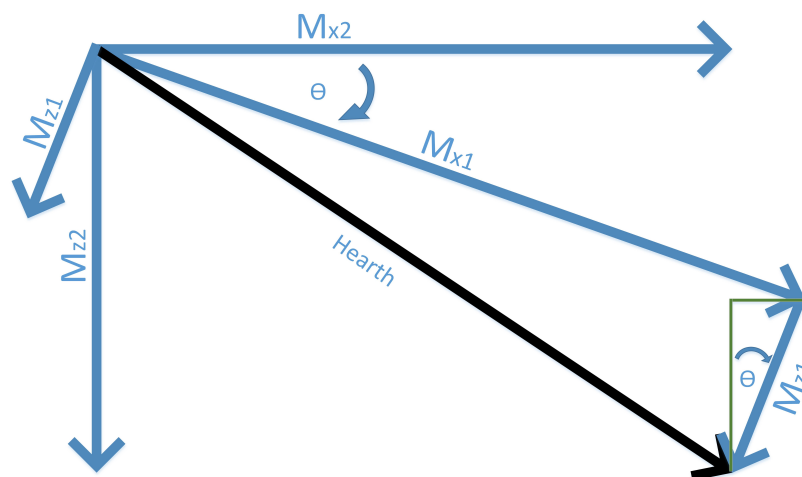


Abbildung 9.22: Kompass 2D: Berechnung der Ausrichtung [STMb]

Die Skizze auf Abbildung 9.23 zeigt beispielhaft, wie sich die Projektion M_{x2} des in X-Richtung (M_{x1}) und Z-Richtung (M_{z1}) gemessenen Magnetfeldvektors in die horizontale Ebene berechnet. Die dargestellte Skizze ist gültig für $\phi = 0$ und $\theta < 0$.

Abbildung 9.23: Kompass 3D: Beispiel $\phi=0$ und $\theta < 0$

Kalibrierung des Magnetometers Das Magnetometer muss kalibriert werden, um unterschiedliche Sensitivitäten der integrierten Hall-Sensoren für die einzelnen Achsen auszugleichen. Außerdem müssen statische Magnetfelder kompensiert werden, die sich als Offset niederschlagen.

Zur Kalibrierung der X- und Y-Achse wird dieses im eingebauten Zustand zunächst in



der Horizontalen gedreht. Aus den gemessenen Maximum und Minimum Werten wird für jede der beiden Achsen ein Offset und ein Wertebereich bestimmt. Bei dem verwendeten Kompass wurden folgende Maximum und Minimum Werte gemessen:

$$X_{Max} = 2 \quad (9.46)$$

$$X_{Min} = -119 \quad (9.47)$$

$$Y_{Max} = 63 \quad (9.48)$$

$$Y_{Min} = -56 \quad (9.49)$$

Diese Werte werden zur Bestimmung der Kalibrierten Magnetometerwerte eingesetzt:

$$X_{Offset} = \frac{(X_{Max} + X_{Min})}{2} \quad (9.50)$$

$$X_{Range} = X_{Max} - X_{Min} \quad (9.51)$$

$$X_{Kal} = \frac{X - X_{Offset}}{X_{Range}} \quad (9.52)$$

$$Y_{Offset} = \frac{(Y_{Max} + Y_{Min})}{2} \quad (9.53)$$

$$Y_{Range} = Y_{Max} - Y_{Min} \quad (9.54)$$

$$Y_{Kal} = \frac{Y - Y_{Offset}}{Y_{Range}} \quad (9.55)$$

Für die Z-Achse ist das gleiche Vorgehen mit einem 90° geneigten Quadrocopter notwendig. Hierbei wurden folgende Werte gemessen:

$$Z_{Max} = 61 \quad (9.56)$$

$$Z_{Min} = -62 \quad (9.57)$$

$$Z_{Kal} = \frac{Z - \frac{(Z_{Max} + Z_{Min})}{2}}{Z_{Max} - Z_{Min}} \quad (9.58)$$

Die so berechneten kalibrierten Werte sind Normiert auf einen Wertebereich von +1 bis -1 bei Messung der horizontalen Komponente des Erdmagnetfeldes. Für die Bestimmung des ψ -Winkels nach Formel 9.44ff werden die kalibrierten Werte genutzt.

Sensordatenfusion Gierwinkel Zur Bestimmung des Gierwinkels werden Gyroskopsensorwerte mit dem Winkel aus dem 3D-Kompass fusioniert. Hierbei wird, wie bei Nick- und Rollwinkel, ein Komplementärfilter eingesetzt.



$$\psi_{Mag} = \operatorname{atan}\left(\frac{M_{y2}}{M_{x2}}\right) \quad (9.59)$$

$$\psi_{Gyro} = \int \dot{\psi}_{Gyro} dt \quad (9.60)$$

$$\psi_k = \alpha(\psi_{k-1} + \dot{\psi}_k \cdot \Delta t) + (1 - \alpha)\psi_{Mag} \quad (9.61)$$

Auf der Zielplattform wird $\alpha=0.995$ verwendet, was einer Grenzfrequenz von 0.05 Hz bei einer Abtastrate von $\Delta t=15$ ms entspricht.

9.4.4.6 Ermittlung der Flughöhe Genau wie bei der Lageermittlung ist auch bei der Höhermittlung sowie zur Ermittlung der Steig- und Sinkgeschwindigkeit ein Komplementärfilter zur Sensordatenfusion einsetzbar. Eine Möglichkeit besteht darin, wie in Kapitel 7.3.7 beschrieben, die Flughöhe des Quadrocopters über eine Messung des Luftdrucks zu bestimmen. Die Ermittlung der Steig- und Sinkgeschwindigkeit kann in diesem Fall über eine Ableitung der Höhe geschehen. Allerdings ist der alleinige Einsatz von Barometern sehr ungenau, da diese sehr stark Rauschen. Der verwendete BMP085-Sensorchip (siehe Kapitel 7.3.7) hat eine Rauschamplitude von 0.3 m Effektivwert. Dies macht das Barometer alleine zur Ermittlung der Steig- und Sinkgeschwindigkeit unbrauchbar. Deswegen wird die Barometermessung mit der Höhenbestimmung über die Z-Beschleunigung fusioniert. Aufgrund der guten Langzeitstabilität aber der schlechten dynamischen Eigenschaften ist die Steig- und Sinkgeschwindigkeit, die über das Barometer ermittelt wird, lediglich für sehr niedrige Frequenzbereich sinnvoll verwendbar.

Genau gegenteilig verhält sich die aus den Beschleunigungswerten ermittelte Steig- und Sinkgeschwindigkeit. Aufgrund der notwendigen Integration driften die ermittelten Werte über lange Zeit immer weiter von den tatsächlichen ab. Daher können über die Beschleunigung ermittelte Steig- und Sinkgeschwindigkeiten nur für hohe Frequenzen genutzt werden.

Im ersten Schritt wird die Höhe auf Basis der Beschleunigung in Richtung der z_g -Achse (Z-Achse im geodätischen Koordinatensystem) berechnet (siehe Formel 9.62 [Sch]).

$$Acc_{zg} = -Acc_x \cdot \sin(\theta_n) + ([Acc_y \cdot \sin(\phi_n) + Acc_z \cdot \cos(\phi_n)] \cdot \cos(\theta_n)) \quad (9.62)$$

$$s_z = \int (Acc_{zg} - 9,81 \frac{m}{s^2}) dt \quad (9.63)$$

$$h_z = \int s_z dt \quad (9.64)$$

Die aktuelle Steiggeschwindigkeit s_z lässt sich nach Formel 9.63 über das Integral bestimmen. Zu berücksichtigen ist hierbei, dass die Erdbeschleunigung subtrahiert werden muss. Durch eine weitere Integration lässt sich die Flughöhe nach Formel 9.64 berechnen. Zur



Bestimmung der aktuellen Flughöhe wird die Höhe h_z aus den Beschleunigungsdaten mit der Höhe h_{baro} aus der Luftdruckmessung fusioniert (siehe Formel 9.65). Der Faktor α wird hierbei zu 0.7 gewählt. Dies entspricht einer Grenzfrequenz f_{gr} von 2.27 Hz bei einer Abtastrate von 30 ms.

$$h_k = \alpha(h_{k-1} + s_{z,k-1} \cdot \Delta t) + (1 - \alpha)h_{baro} \quad (9.65)$$

$$\text{mit } \alpha = \frac{\frac{T}{\Delta t}}{1 + \frac{T}{\Delta t}} \quad (9.66)$$

Die aktuelle Steiggeschwindigkeit wird durch eine Fusion von s_z und der Ableitung der berechneten Höhe bestimmt (siehe Formel 9.67). Hier wird $\alpha=0.99$ gesetzt ($f_{gr}=0.05$ Hz), da sonst das Rauschen des Barometers die Werte zu stark verfälschen würde.

$$s_k = \alpha(s_{k-1} + s_{z,k-1} \cdot \Delta t) + (1 - \alpha) \cdot \frac{h_k - h_{k-1}}{\Delta t} \quad (9.67)$$

9.5 ASG

In diesem Abschnitt wird neben der Modellierung des physikalischen Modells des Quadropters durch die Software CAMEL-View, auf die Beschreibung der Lage- und Höhenregelung, sowie die Motorwertberechnung eingegangen. Zusätzlich wird die Kommunikation zwischen dem ASG und dem AEVV-System durch einen eigenen BRAM beschrieben.

9.5.1 Modellbasierte Entwicklung des ASG

Da für die Entwicklung der sicherheitskritischen Anwendung ein modellgetriebener Ansatz verfolgt wird, wird in diesem Kapitel die Modellbildung der Avionik betrachtet. Als Entwicklungsumgebung kommt dabei die Software CAMEL-View des Unternehmens iXtronics GmbH zum Einsatz. CAMEL-View ist insbesondere für den modellbasierten Entwurf mechatronischer Systeme geeignet. Ähnlich wie die weit verbreitete Simulationssoftware MATLAB/Simulink ermöglicht CAMEL-View die objektorientierte Modellbildung mit Hilfe graphischer Blöcke. Dafür stehen verschiedene Toolboxen zur Regelungstechnik, zu Mehrkörper- und hydraulischen Systemen sowie zur Animation zur Verfügung. Des Weiteren unterstützt CAMEL-View die Verwendung von C-Code innerhalb der Modelle und kann somit in allen Phasen des modellbasierten Entwurfs - Model in the loop (MIL), Software in the loop (SIL) und Hardware in the loop (HIL) - verwendet werden.

Im weiteren Verlauf des Kapitels wird zunächst der allgemeine Aufbau des Modells beschrieben. Daraufhin wird auf die Implementierung der Lage- und Höhenregelung betrachtet. Es folgt die Beschreibung der Berechnung der Motorstellwerte. Anschließend wird näher auf die Entwicklung eines physikalischen Modells zur Validierung der entworfenen

Regler eingegangen. Den Abschluss des Kapitels bildet die Generierung von C-Code aus dem Modell.

9.5.1.1 Aufbau des Modells Das ASG kann als geschlossener Grundregelkreis (s. Kapitel 3.2.2) dargestellt werden, wie in Abbildung 9.24 gezeigt. Die Sollwerte für die Winkel Θ , Φ , Ψ und die Flughöhe des Quadropters gehen als Führungsgrößen in die Regelung ein. Mit Hilfe der Messwerte der Sensoren werden die Regelfehler berechnet, die wiederum als Eingangswerte für die Lage- und die Höhenregelung dienen. Die dadurch entstehenden Stellsignale werden vom Stellglied in die entsprechenden Motordrehzahlen umgerechnet. Diese bilden schließlich die Eingaben für die Regelstrecke. An dieser Stelle kommen auch die Grundsätze des modellbasierten Entwurfs zum Einsatz. Während der MIL-Phase wird die Regelstrecke als physikalisches Modell unter Verwendung von Differentialgleichungen beschrieben. In der späteren HIL-Phase wird die Regelstrecke dann durch den realen Quadropters ersetzt.

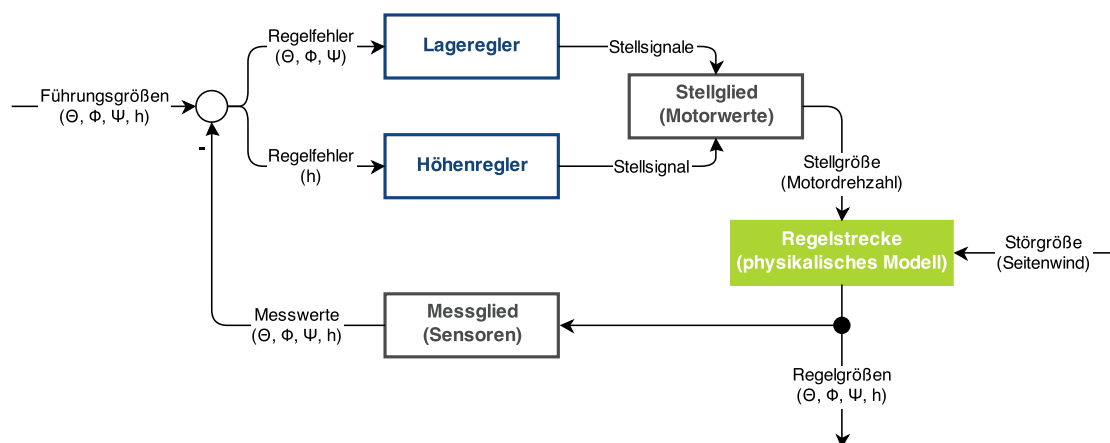


Abbildung 9.24: Blockschaltbild Flugregelung

9.5.1.2 Modellierung der Lageregelung Das Modell der Lageregelung wird ebenfalls in CAMEL-View implementiert und durch ein Objekt der Klasse `AttitudeControlClass` dargestellt. Die Lageregelung besteht im Wesentlichen aus drei Reglern, die unabhängig voneinander die Stellgrößen für die Rotation um die X-, Y- und Z-Achse berechnen. Diese Werte dienen wiederum als Grundlage für die Berechnung der Motorstellwerte, worauf im Kapitel 9.5.1.4 eingegangen wird.

In Abbildung 9.25 ist das Modell der Lageregelung in CAMEL-View dargestellt. Die Eingänge des Moduls werden durch die Sollwerte der einzelnen Fernsteuerungskanäle, sowie durch die bei der Sensorverarbeitung ermittelten Werte für die Winkel und Winkelgeschwindigkeiten der einzelnen Raumachsen gebildet. Da der Winkel θ einen definierten Bereich von $-\frac{\pi}{2}$ bis $+\frac{\pi}{2}$ und die Winkel ϕ und ψ einen Definitionsbereich von $-\pi$ bis

$+\pi$ besitzen, müssen die Winkelüberschläge im Grenzbereich berücksichtigt werden. Dazu wurden die Klassen WinkelueberschlagThetaClass, WinkelueberschlagPhiClass und WinkelueberschlagPsiClass implementiert. Des Weiteren werden die Sollwerte der Fernsteuerung mit Hilfe der Klasse LimitAngleClass begrenzt.

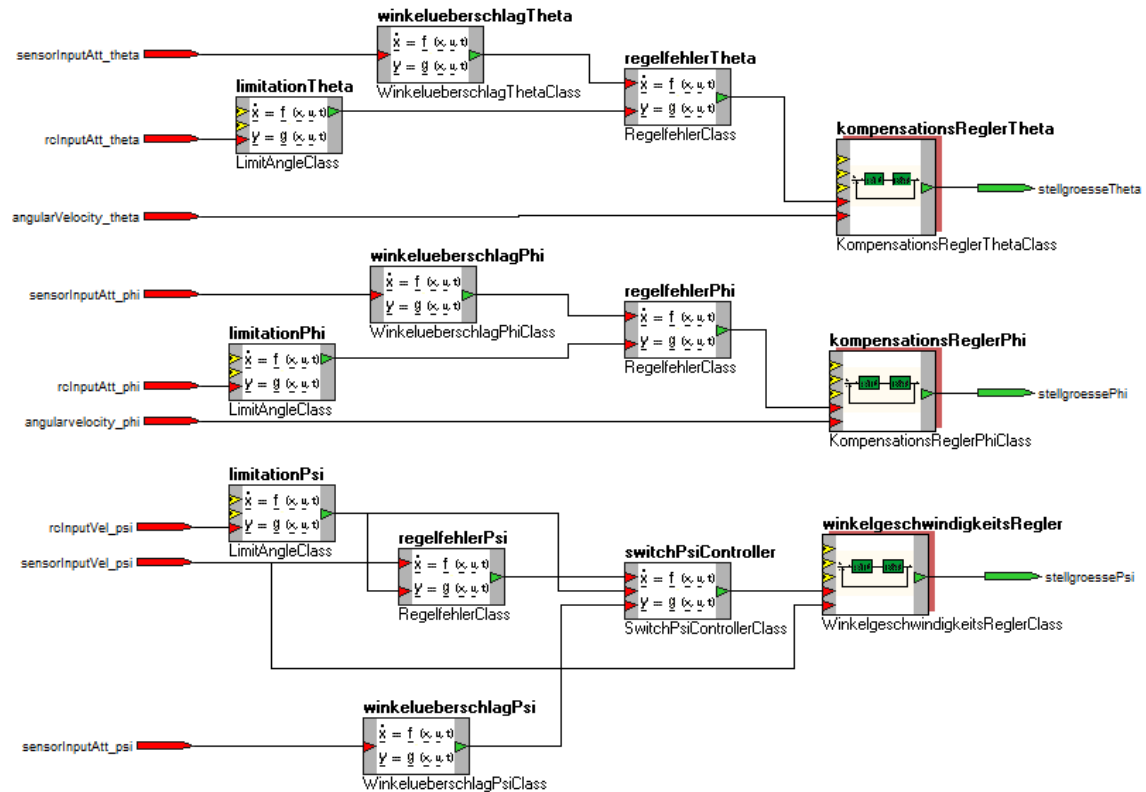


Abbildung 9.25: Lageregelung in CAMEL-View

Aus den Eingaben der Fernsteuerung (Sollwerte $w(t)$) und den fusionierten Sensorwerten der Winkel (Istwerte $x(t)$) werden anschließend die entsprechenden Regelabweichungen $e(t)$ gemäß Formel 9.68 berechnet. Dazu wurde auf Grundlage der in CAMEL-View vordefinierten Klasse `StateSpaceOdds` eine neue Klasse `RegelfehlerClass` erstellt.

$$e(t) = w(t) - x(t) \tag{9.68}$$

Bei den verwendeten Reglern handelt es sich um eine Art Kompensationsregler. Dieser Regler besteht aus drei wesentlichen Komponenten. Einem proportionalen Anteil, einem integralen Anteil und einem dämpfenden Anteil. Der Stellwert des integralen Anteils ist begrenzt, da der Stellwert für die Motorwerte ebenfalls begrenzt ist. Wenn der integrale Anteil nicht zusätzlich begrenzt wäre, dann würden sich zeitliche Verzögerungen ergeben,



wenn der Betrag des integralen Anteils größer als der maximale Motorstellwert werden würde. Die Dämpfung wird durch die Winkelgeschwindigkeit ν und einem Skalierungsfaktor D_s der jeweiligen Achse charakterisiert. Der proportionale und der integrale Anteil werden additiv zu einem PI-Anteil zusammengefasst. Der dämpfende Anteil wird von dem Betrag des PI-Anteils gemäß folgender Formel subtrahiert [Sch10, S. 51].

$$y_{Komp}(k) = K_P \cdot e(k) + K_I \cdot \sum_{i=0}^{k-1} e(k) - D_s \cdot \nu(k) \quad (9.69)$$

Befindet sich das System in Ruhe ($\nu = 0$), so ist der Betrag der Dämpfung gleich Null. Rotiert das System um eine spezifische Achse, so verhält sich die Dämpfung proportional zur einwirkenden Winkelgeschwindigkeit. Dadurch wird eine Kraft produziert, die der Rotation aufgrund des negativen Vorzeichens entgegenwirkt.

In Abbildung 9.26 ist das Modell des Kompensationsreglers in CAMEL-View dargestellt. Es handelt sich dabei um eine hierarchische Struktur, die auf der Klasse `MathematicHcs0dss` basiert. Die einzelnen Regleranteile wurden jeweils als eigene Klassen `PClass` für den P-Anteil, `IClass` für den I-Anteil und `DampingClass` für den Dämpfungs-Anteil realisiert. Der P-Anteil besitzt einen Parameter als Eingang durch den die Verstärkung der Regelabweichung definiert wird. Beim I-Anteil existieren zwei zusätzliche Parameter `Imin` und `Imax`. Diese dienen als Begrenzung und verhindern den zu starken Aufbau des I-Anteils, besser bekannt als Anti-Windup. Der Dämpfungs-Anteil besitzt als Eingang nicht den Regelfehler, sondern die Winkelgeschwindigkeit. Dessen Einfluss kann mit Hilfe des Parameters `Ds` kontrolliert werden. Nachdem die einzelnen Regleranteile gemäß Formel 9.69 addiert wurden, erfolgt eine Begrenzung der Stellgröße um eine zu starke Varianz der Motorstellwerte zu verhindern. Die Wahl geeigneter Parameter wird im Kapitel 9.5.1.6 beschrieben.

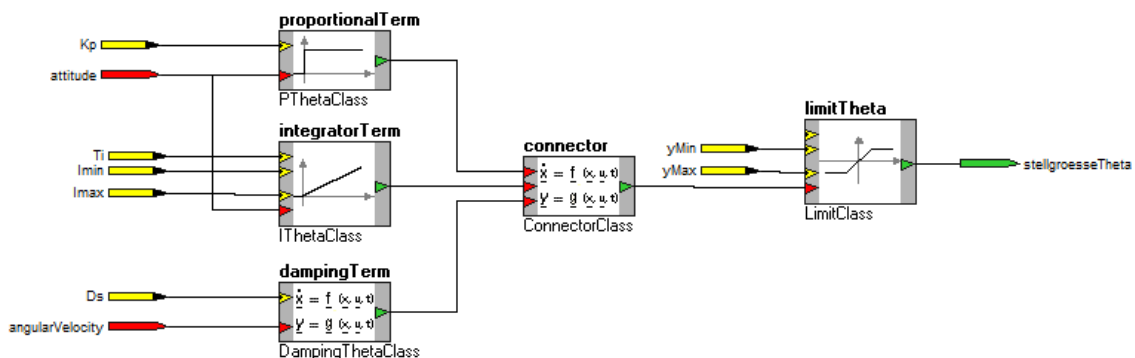


Abbildung 9.26: Struktureller Aufbau des Kompensationsreglers in CAMEL-View

Während für die Roll- und Nick-Achse des Quadropters die Regelung auf Grundlage der Winkel θ und ϕ erfolgt, wird für die Gier-Achse ein komplexerer Ansatz verfolgt. Solange sich der Gierstick der Fernsteuerung in der Mittelstellung befindet, d.h. der Quadropters nicht Gieren soll, erfolgt die Regelung anhand des Gierwinkels ψ . Beginnt der Pilot hingegen zu Gieren so wird auf Grundlage der Winkelgeschwindigkeit ν_ψ geregelt. Währenddessen wird der aktuelle Gierwinkel in einer Hilfsvariablen zwischengespeichert, damit bei Beendigung der Gierbewegung erneut auf diesen Winkelwert geregelt werden kann. Leider lässt sich die Hilfsvariable nur mit dem Wert 0 initialisieren. In der Realität würde dies dazu führen, dass der Quadropters sich nach dem Start stets nach dem geographischen Norden ausrichtet. Dieses Verhalten ist unsinnig und wurde durch einen Workaround in der Sensordatenverarbeitung korrigiert. Der Gierwinkel wird dabei intern mit einem Offset versehen, sodass die Ausgangsposition des Quadropters stets den Gierwinkel 0° aufweist. Das Umschalten der Regelung des Gierwinkels wurde im Modell in der Klasse `SwitchPsiControllerClass` implementiert (s. Abbildung 9.25).

9.5.1.3 Modellierung der Höhenregelung Neben der Lageregelung wurde im Rahmen der Projektgruppe auch eine Höhenregelung implementiert. Diese sorgt dafür, dass der Quadropters seine Flughöhe automatisch beibehält. Dabei sind zwei unterschiedliche Fälle zu unterscheiden. Ist die Höhenregelung deaktiviert, so wird der aktuelle Schubwert der Fernsteuerung an die Motoren weitergereicht. Bei aktiver Höhenregelung gibt der Schubstick der Fernsteuerung hingegen die Sink- bzw. Steiggeschwindigkeit des Quadropters vor. In diesem Fall werden mit Hilfe des Höhenreglers die entsprechenden Stellwerte für die Motoren berechnet. Das Modell des Höhenreglers ist in Abbildung 9.27 zu finden.

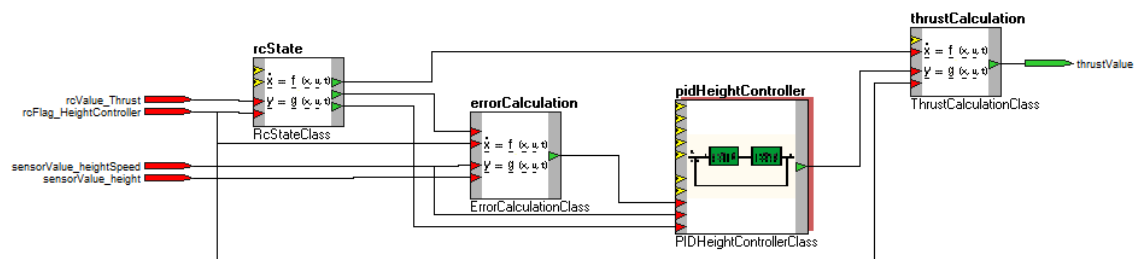


Abbildung 9.27: Höhenregelung in CAMEL-View

Die Eingangswerte für den Höhenregler sind der aktuelle Schubwert, die Schalterstellung für die Höhenregelung auf der Fernsteuerung, die aktuelle Flughöhe des Quadropters sowie dessen Steig-/Sinkgeschwindigkeit. Die Klasse `RcStateClass` überprüft, ob die Höhenregelung vom Piloten aktiviert wurde. Durch eine Fallunterscheidung wird dabei entweder der Schubwert der Fernsteuerung durchgereicht oder es wird der Sollwert für die Steig-/Sinkgeschwindigkeit des Quadropters berechnet. Dieser Wert geht in die Berechnung der Regelabweichung durch die Klasse `ErrorCalculationClass` ein. Dabei müssen



erneut zwei Fälle unterschieden werden. Falls die berechnete Steig-/Sinkgeschwindigkeit ungleich 0 ist, wird der Regelfehler anhand der gemessenen Steig- bzw. Sinkgeschwindigkeit des Quadropters geregelt. Sonst erfolgt die Berechnung des Regelfehlers auf Grundlage des gemessenen Höhenwerts um die Flughöhe des Quadropters beizubehalten (s. Abbildung 9.28).

```

StateSpaceOdss named: ErrorCalculationClass.
  input:      #(setpoint_heightSpeed)  on: ScalarOdss unit: 'm/s';
              #(isHeightControlActive) on: ScalarOdss;
              #(actual_heightSpeed)    on: ScalarOdss unit: 'm/s';
              #(actual_height)         on: ScalarOdss unit: 'm';
  output:     #(error_h)               on: ScalarOdss unit: 'm/s';

  auxiliar:   #(aux_height) on: ScalarOdss;
              #(aux_error)  on: ScalarOdss;

  auxiliarEquation:
    ((isHeightControlActive = 1) & (setpoint_heightSpeed = 0))
    isTrue: [aux_error := aux_height - actual_height]
    isFalse:[aux_height := actual_height;
             aux_error := setpoint_heightSpeed - actual_heightSpeed];

  outputEquation:
    error_h := aux_error;
end.

```

Abbildung 9.28: CAMeL-View Code des Regelfehlers für die Höhenregelung

Für die Höhenregelung wird erneut ein Kompensationsregler verwendet. Dieser entspricht weitestgehend dem im Kapitel 9.5.1.2 beschriebenen Modell. Es wurde lediglich eine Erweiterung beim I-Anteil hinzugefügt, da dieser für die Höhenregelung deaktivierbar sein muss. Dadurch wird gewährleistet, dass der I-Anteil nur beim Halten der Flughöhe einen Einfluss hat und beim Steigen bzw. Sinken des Quadropters nicht berücksichtigt wird. Im Modell wird dies realisiert indem die Ableitung auf 0 gesetzt und der Regelfehler somit nicht weiter aufsummiert wird.

Der Stellwert des Höhenreglers geht zusammen mit dem Schubwert in den Block `Thrust CalculationClass` ein. Bei deaktivierter Höhenregelung wird der Schubwert durchgereicht. Ist diese hingegen aktiv wird auf den Schubwert der Stellwert des Höhenreglers addiert. Mit Hilfe des Parameters `heightSpeedScaling` kann dieser noch skaliert werden.

9.5.1.4 Berechnung der Motorstellwerte Aus der bereits beschriebenen Lage- und Höhenregelung ergeben sich Stellgrößen für θ , ϕ , ψ und den Schub. Diese Werte müssen auf



die einzelnen Motoren des Quadropters verteilt werden. In Abbildung 9.29 ist die Berechnung dieser Motorstellwerte dargestellt. Zunächst werden die Eingangswerte mit Hilfe der Klasse `MotorLimitClass` begrenzt. Anschließend wird im Block `MotorValueSumClass` für jeden Motor der Stellwert entsprechend den eingehenden Stellgrößen berechnet. Dabei kommen folgende Formeln zur Anwendung:

$$M_{Front} = y_{Thrust} + y_{Theta} + y_{Psi} \quad (9.70)$$

$$M_{Rear} = y_{Thrust} - y_{Theta} + y_{Psi} \quad (9.71)$$

$$M_{Left} = y_{Thrust} + y_{Phi} - y_{Psi} \quad (9.72)$$

$$M_{Right} = y_{Thrust} - y_{Phi} - y_{Psi} \quad (9.73)$$

Der vordere Motor M_{Front} und der hintere Motor M_{Rear} sind für das Nicken des Quadropters verantwortlich. Deshalb geht bei deren Berechnung die Stellgröße y_{Theta} ein. Die Rollbewegung des Quadropters wird durch eine Veränderung der Drehzahlen des linken Motors M_{Left} und des rechten Motors M_{Right} realisiert. Deshalb wird hier die Stellgröße y_{Phi} einbezogen. Um den Quadropters zu Gieren müssen der vordere und der hintere Motor ein größeres bzw. kleineres Drehmoment als der rechte und der linke Motor aufweisen. Dies spiegelt sich in der Addition der Stellgröße y_{Psi} bei M_{Front} und M_{Rear} sowie deren Subtraktion bei M_{Left} und M_{Right} wider.

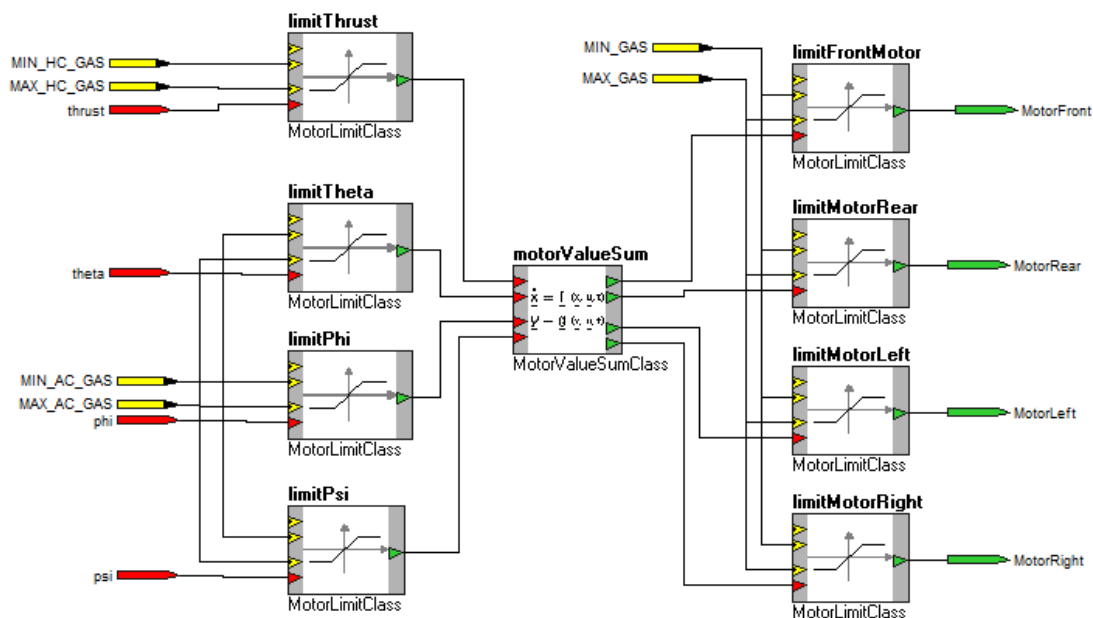
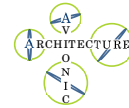


Abbildung 9.29: Berechnung der Motorstellwert in CAMEL-View



Wie in Kapitel 7.3.6 beschrieben, entwickelt jeder Motor, mit der verwendeten 12 x 4.5" Luftschraube, eine Kraft, um maximal 2,2 kg in die Luft zu heben. Dies entspricht ungefähr einer Schubkraft von 21,6 N. Bei der Berechnung der Motorstellwerte muss dieser Maximalwert berücksichtigt werden. Deshalb werden die Ausgänge der Klasse `MotorValueSumClass` erneut mit Hilfe der Klasse `MotorLimitClass` begrenzt. Somit ist sichergestellt, dass im Modell keine größeren Motorstellwerte auftreten als in der Realität.

Die berechneten Motorstellwerte dürfen nur auf die Motoren gegeben werden, wenn bestimmte Voraussetzungen erfüllt sind. Dadurch können wichtige Sicherheitsaspekte realisiert werden, wie beispielsweise dass die Motoren erst rotieren, sobald der Pilot den entsprechenden Schalter auf der Fernsteuerung betätigt hat. Aus diesem Grund wurde der in Abbildung 9.30 dargestellte Zustandsautomat zur Berechnung der Motorstellwerte implementiert.

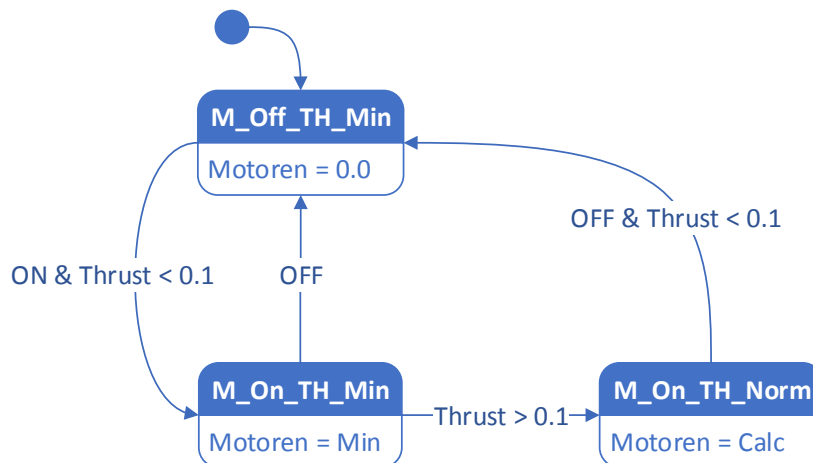


Abbildung 9.30: Zustandsautomat der Motorstellwerte

Im Initialzustand `M_Off_TH_Min` des Automaten werden keine Stellwerte an die Motoren weitergegeben. Erst wenn der Pilot die Motoren über die Fernsteuerung einschaltet und sich der Schubstick in der untersten Position befindet, geht der Automat in den Zustand `M_On_TH_Min` über. In diesem Zustand wird das Standgas auf die Motoren gegeben. Eine Weitergabe der berechneten Motorstellwerte erfolgt jedoch noch nicht. Dies ist erst dann der Fall, wenn der Pilot den Schub auf der Fernsteuerung erhöht. Der Automat geht dann in den Zustand `M_On_TH_Norm` über. In diesem Zustand werden die berechneten Stellwerte aus der Lage- und Höhenregelung direkt auf die Motoren gegeben. Dies ist der normale Flugzustand des Systems. Schaltet der Pilot die Motoren auf der Fernsteuerung aus, so werden die Motoren erst dann tatsächlich ausgeschaltet, wenn sich der Schubstick wieder in der untersten Position befindet. Dadurch wird ein Absturz des Quadropters verhindert, falls die Motoren während des Fluges vom Piloten deaktiviert werden.



9.5.1.5 Physikalisches Modell des Quadropters Das physikalische Modell des Quadropters stellt die Regelstrecke dar (s. Abbildung 9.24) und soll das reale Systemverhalten des Quadropters möglichst genau simulieren. Als Eingänge werden die Auftriebskräfte der vier Motoren F_1 bis F_4 verwendet. Daraus werden die Rotationswinkel des Quadropters und dessen Lage im dreidimensionalen Raum berechnet. Unter der Voraussetzung, dass sich der Quadropters in der +-Konfiguration befindet, können für die Berechnung der Lage die folgenden, vereinfachten Differentialgleichungen verwendet werden [Pul05, S. 90]:

$$m \cdot \ddot{x} = u_1 \cdot (\sin \Theta \cos \Psi + \sin \Phi \sin \Psi) - W_1 \cdot \dot{x}^2 \quad (9.74)$$

$$m \cdot \ddot{y} = u_1 \cdot (\sin \Phi \cos \Psi - \sin \Theta \sin \Psi) - W_2 \cdot \dot{y}^2 \quad (9.75)$$

$$m \cdot \ddot{z} = u_1 \cdot (\cos \Phi \cos \Theta) - m \cdot g - W_3 \cdot \dot{z}^2 \quad (9.76)$$

mit

$$u_1 = F_1 + F_2 + F_3 + F_4 \quad (9.77)$$

Die Konstante m beschreibt dabei die Gesamtmasse des Quadropters, während die Konstanten W_1 bis W_3 Widerstandswerte darstellen. Bei der Konstanten g handelt es sich um die Gravitationskonstante. Es wird deutlich, dass zur Berechnung der Lage die Rotationswinkel Θ für die Neigung des Quadropters entlang der Nick-Achse, Φ für die Neigung entlang der Roll-Achse und Ψ für die Drehung um die Gier-Achse notwendig sind. Diese gehen aus den folgenden, vereinfachten Differentialgleichungen für einen Quadropters in +-Konfiguration hervor [Pul05, S. 91]:

$$I_{xx} \cdot \ddot{\Theta} = l \cdot (-F_1 + F_3) - W_4 \cdot \dot{\Theta}^2 \quad (9.78)$$

$$I_{yy} \cdot \ddot{\Phi} = l \cdot (-F_2 + F_4) - W_5 \cdot \dot{\Phi}^2 \quad (9.79)$$

$$I_{zz} \cdot \ddot{\Psi} = l \cdot (M_1 - M_2 + M_3 - M_4) - W_6 \cdot \dot{\Psi}^2 \quad (9.80)$$

Neben den Widerstandswerten W_4 bis W_6 fließen die Längen l der Rotorausleger des Quadropters und die Trägheitsmomente I_{xx} , I_{yy} und I_{zz} in die Berechnung der Rotationswinkel ein. Während die Differentialgleichungen Formel 9.78 und Formel 9.79 die Auftriebskräfte der Rotoren verwenden, werden für den Winkel Ψ die entsprechenden Drehmomente M_1 bis M_4 verwendet.

Eine Übersicht des physikalischen Modells in CAMEL-View ist in Abbildung 9.31 zu sehen. Jede Differentialgleichung wurde in einer eigenen Klasse des Typs *StateSpaceOdds* implementiert. Für die Berechnung der Drehmomente aus den Auftriebskräften wurde ebenfalls eine eigene Klasse *ForceTorqueConverter* angelegt. Neben der Lage und den Winkeln des Quadropters werden in dem Modell auch die entsprechenden Winkelgeschwindigkeiten

und die Geschwindigkeiten in X-, Y- und Z-Richtung berechnet. Eine Erweiterung um die Beschleunigungen ist ebenfalls leicht möglich.

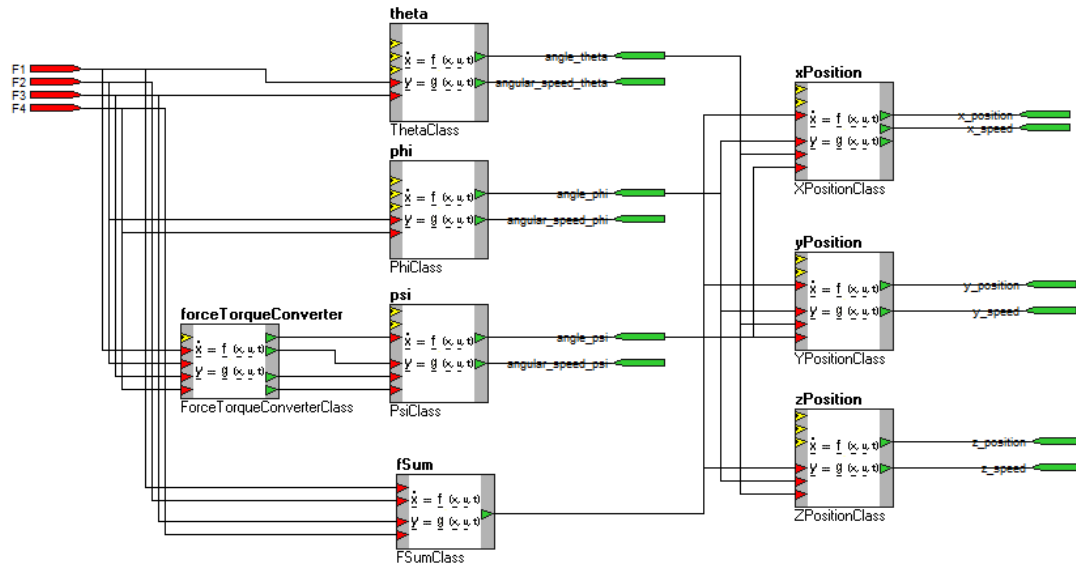


Abbildung 9.31: Physikalisches Modell des Quadropters in CAMEL-View

Bereits erste Tests mit diesem physikalischen Modell haben gezeigt, dass es nur einen eingeschränkten Gültigkeitsbereich besitzt. So lieferte es nur für kleine Winkel bis 5° korrekte Ergebnisse. Der Grund dafür ist, dass in den Differentialgleichungen Formel 9.74, Formel 9.75, Formel 9.76, Formel 9.78, Formel 9.79 und Formel 9.80 diverse Vereinfachungen vorgenommen wurden. Die Beschränkung auf kleine Winkel ist für das Testen der Lage- und Höhenregelung sowie die Ermittlung geeigneter Reglerparameter nicht geeignet. Die Implementierung des physikalischen Modells anhand von Differentialgleichungen ist deshalb nicht möglich.

Aus diesem Grund wurde entschieden einen anderen Modellierungsansatz zu verfolgen und den Quadropters durch ein Mehrkörpersystem zu modellieren. Es handelt sich dabei um mechanische Systeme einzelner Körper, die untereinander durch Gelenke verbunden sind. Die Dynamik des Systems wird durch die an den Gelenken und Körpern wirkenden Kräfte beschrieben. Mehrkörpersysteme kommen insbesondere dann zum Einsatz, wenn die Erstellung eines mathematischen Modells zu aufwändig erscheint. Des Weiteren ist die Modellierung von Mehrkörpersystemen einer der Haupteinsatzzwecke der Simulationssoftware CAMEL-View. Mit der Klasse `Multibody System Elements` steht dort sogar eine eigene Bibliothek für diese Systeme zur Verfügung.

Die Modellierung des Quadropters als Mehrkörpersystem ist trivial. Es handelt sich dabei lediglich um einen starren Körper, der keinerlei Gelenke besitzt und an dessen Auslegern die vier Auftriebskräfte F_1 bis F_4 wirken. In CAMEL-View wurde dies durch

die Verwendung der Klasse `RigidBodyMbs` realisiert. Als Eingänge enthält diese die vier Auftriebskräfte und das Drehmoment des Quadropters. Für jede Auftriebskraft wurde daraufhin ein separater `Frame` definiert, welcher die Lage des Kraftvektors relativ zum Schwerpunkt des Körpers beschreibt. Außerdem benötigt das Mehrkörpersystem Informationen über die Masse des Körpers und seine Trägheitstensoren. Als Ausgänge der Klasse `RigidBodyMbs` sind lediglich die Position des Körpers im dreidimensionalen Raum und dessen Orientierung fest definiert. Es können jedoch weitere Ausgänge wie beispielsweise die Winkelgeschwindigkeit oder die Beschleunigung des Körpers hinzugefügt werden. Dabei muss jedoch stets das richtige Bezugssystem gewählt werden. Für die Lageregelung des Quadropters spielen dessen Winkelgeschwindigkeiten eine wichtige Rolle. Deshalb wurde dem Mehrkörpersystem ein weiterer Ausgang mit der Winkelgeschwindigkeit im körperfesten Koordinatensystem `AngularVelocityAbsBcsMbs` hinzugefügt. Die Höhenregelung benötigt die Steig- bzw. Sinkgeschwindigkeit des Quadropters im ortsfesten Koordinatensystem, sodass ein weiterer Ausgang `VelocityAbsIcsMbs` erforderlich war.

Ein weiterer Vorteil von CAMEL-View bei der Modellierung von Mehrkörpersystem ist die einfache Erstellung einer 3D-Ansicht. Diese kann unter dem Menüpunkt `3D Graphic` in der Klasse `RigidBodyMbs` vorgenommen werden. Dadurch kann in der Simulation die Dynamik des Systems animiert werden. Im Rahmen der Projektgruppe wurde zunächst versucht, die Darstellung des Quadropters aus dem Szenario (s. Abbildung 1.7) zu verwenden. Dabei stellte sich jedoch heraus, dass diese Darstellung zu komplex ist und zu starken Verzögerungen in der Simulation führt. Deshalb wurde ein abstrakteres 3D-Modell des Quadropters gewählt, welches in Abbildung 9.32 zu finden ist.

Das Mehrkörpersystem liefert deutlich bessere Ergebnisse als das physikalische Modell mit Differentialgleichungen. Die Lage- und Höhenregelung kann dadurch in der Simulation überprüft werden.

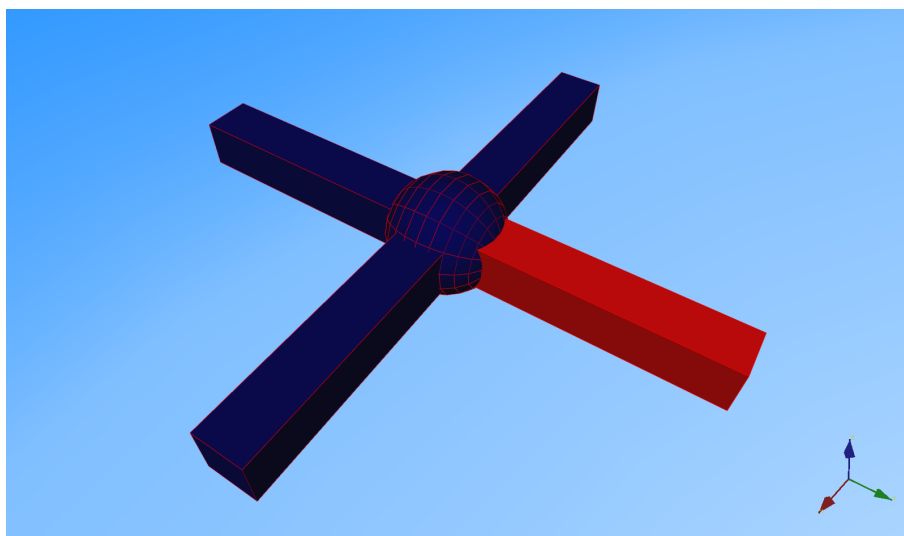


Abbildung 9.32: 3D-Modell des Quadropters in CAMEL-View



9.5.1.6 Ermittlung der Reglerparameter Zur Ermittlung der Reglerparameter wurde ein praktisch orientierter Ansatz verfolgt. Bei diesem Ansatz wurden die Parameter nicht mathematisch anhand von Übertragungsfunktionen und dem physikalischen Modell bestimmt, sondern heuristisch bzw. empirisch durch die Einstellregeln nach Ziegler/Nichols [LW12, S. 478 ff]. Dieses Vorgehen ist ein iterativer Prozess, welcher mehrere Schritte benötigt, um die optimalen Reglerparameter zu bestimmen.

Initial wurde lediglich ein P-Regler verwendet, um die kritische Verstärkung $K_{P_{krit}}$ zu bestimmen, bei der das System zu schwingen beginnt. Dieser gefundene Wert für die Verstärkung des Regelungssystems wurde gemäß [LW12, S. 478 ff] von 111 um 55% auf 50 verringert. Zur Ermittlung der Zeitkonstante T_i des integralen Anteils wurde die Periodendauer T_{krit} der entstandenen Schwingung gemessen. Aus diesem Wert konnte entsprechend den Einstellregeln von Ziegler/Nichols die Zeitkonstante T_i bestimmt werden. Um den Betrag der künstlichen Dämpfung einzustellen gibt es keine Faustregel die angewendet werden kann. Es wurde für die Dämpfung ein geringer Anfangswert gewählt, da die Dämpfung sehr kritisch im Bezug auf die Stabilität des Regelkreises anzusehen ist. Zur Ermittlung des optimalen Wertes wurde der Betrag schrittweise erhöht, bis sich ein Flugverhalten einstellte, welches ein leicht gedämpftes Verhalten aufzeigte. Der sich für den Quadrokopter geeignete Wert für die Dämpfung liegt bei 10.

Dieses Vorgehen wurde sowohl bei der Lage- als auch bei der Höhenregelung angewandt. Die ermittelten Reglerparameter sind in Tabelle 9.6 zu finden.

Tabelle 9.6: Ermittelte Reglerparameter gemäß Ziegler/Nichols Einstellregeln

	Lageregler Θ, Φ	Lageregler Ψ	Höhenregler
P-Anteil	50.0	37.0	7.0
I-Anteil	0.4	-	0.04
Dämpfung	10.0	1.0	3.0

9.5.1.7 Generierung des C-Code Für die Code-Generierung wird in CAMEL-View der Task-Assistent verwendet. Da für den Quadrokopter nur die Regler benötigt werden, wird ein neues Modell (siehe Abbildung 9.33) erstellt. Dieses Modell enthält dann lediglich die Blöcke, die für die Regelung benötigt werden. Die Blöcke für das physikalische Modell des Quadrokopters fallen weg. Der generierte Code wird dann in dem Wrapper (Kapitel 8.5) von iXtronics verwendet. Der Wrapper musste noch entsprechend für die Zynq-Plattform angepasst werden. Zum Beispiel ist die Speicherreservierung mit dem `malloc()`-Befehl implementiert worden. Dies verursacht jedoch Fehler bei der Ausführung auf dem Microblaze des ASG.

Die ersten Flugversuche mit dem generierten C-Code zeigten ein instabiles Flugverhalten des Quadrokopters. Dies ist insofern überraschend, da sich das System in der Simulation stabil verhält. Dabei muss jedoch berücksichtigt werden, dass die Motoren in der Simulati-

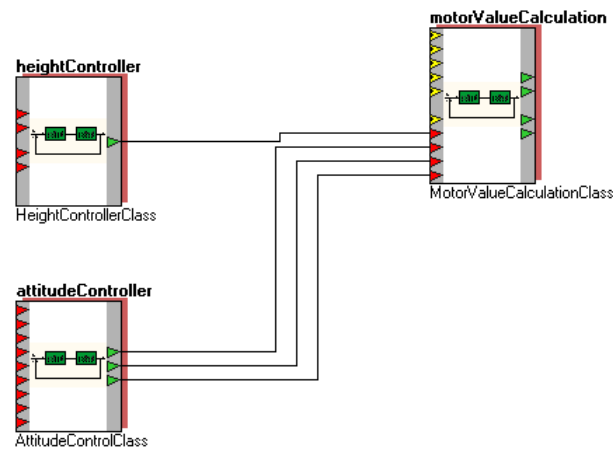


Abbildung 9.33: Modell für Code-Generierung

on nicht abgebildet wurden. Stattdessen wurden nur die Auftriebskräfte (s. Kapitel 9.5.1.5) betrachtet. Dennoch lässt sich die Diskrepanz zwischen Simulation und Realität nicht vollständig auf diese Beschränkung des Modells zurückführen, denn auch die Untersuchungen mit diversen Reglerparametern führten nicht zu einer Stabilisierung des Systems. Das instabile Flugverhalten des Quadropters wird demnach durch mehrere Faktoren hervorgerufen. Einerseits wird der Reglerentwurf durch CAMEL-View stark restriktiert, u.a. durch die fehlende Möglichkeit den I-Anteil zurückzusetzen (s. Kapitel 9.5.1.2). Andererseits erscheint die Ausführung des C-Code mit numerischen Integrationsverfahren unangemessen und führt zu signifikanten Verzögerungen in den Berechnungen. Aus diesen Gründen kann der aus CAMEL-View generierte C-Code im Rahmen dieser Projektgruppe nicht auf dem realen System ausgeführt werden.

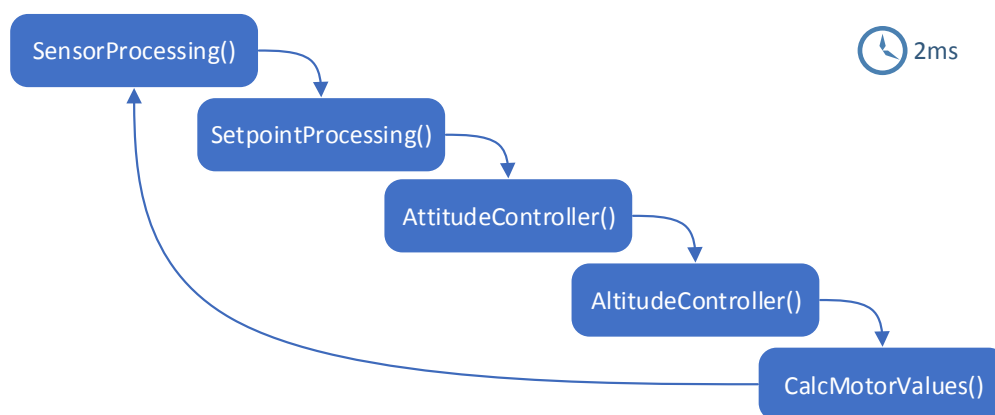


Abbildung 9.34: Ablaufplan des selbst implementierten C-Code



Um das Projektziel eines stabilen, flugfähigen Quadropters dennoch zu erreichen, hat sich die Projektgruppe dazu entschieden den C-Code für die Lage- und Höhenregelung selbst zu implementieren. Dabei konnten einige Artefakte früherer Arbeiten [Sch10] wiederverwendet werden. Der Ablauf des implementieren Code ist in Abbildung 9.34 dargestellt.

Zunächst werden mit Hilfe der Funktion `SensorProcessing` die aktuellen Sensordaten aus dem BRAM des DataMiner gelesen. Danach werden die Sollwerte der Fernsteuerung durch die Funktion `SetpointProcessing` entgegengenommen und ausgewertet. Es folgt der Aufruf der Lageregelung durch die Funktion `AttitudeController`, in der entsprechende Stellwerte für die Motoren des Quadropters berechnet werden. Anschließend wird die Höhenregelung durch die Funktion `AltitudeController` ausgeführt. Daraus resultieren erneut entsprechende Motorstellwerte. Diese werden im letzten Schritt mit Hilfe der Funktion `CalcMotorValues` an die Motoren gesendet (s. Kapitel Kapitel 9.5.2). Da die Regelungsfrequenz in den Anforderungen mit 500 Hz definiert wurde (s. 6.8), wird dieser Ablauf zyklisch alle 2 ms wiederholt.

Bei den Flugversuchen zeigte sich weiterhin, dass auch die im Kapitel 9.5.1.6 ermittelten Reglerparameter nicht korrekt sind und ein starkes Aufschwingen des Systems verursachen. Deshalb wurden empirisch neue Parameter ermittelt, die in Tabelle 9.7 zu finden sind. Mit diesen Änderungen konnte ein stabiles und sicheres Flugverhalten des Quadropters sichergestellt werden.

Tabelle 9.7: Empirisch ermittelte Reglerparameter für C-Code

	Lageregler Θ, Φ	Lageregler Ψ	Höhenregler
P-Anteil	11.0	25.0	30.0
I-Anteil	0.015	-	0.002
Dämpfung	1.65	10.0	5.0

9.5.2 Motoransteuerung

Jeder der vier BL-Motoren des Quadropters wird über einem BL-Controller⁴⁸ der Firma MikroKopter angesteuert. Diese Controller kommunizieren über einen I²C Bus mit der Prozessorplattform und haben eindeutige I²C-Adresse.

- I²C Adresse vorderer Motor: 0x52
- I²C Adresse hinterer Motor: 0x54
- I²C Adresse linker Motor: 0x56
- I²C Adresse rechter Motor: 0x58

⁴⁸BL-Controller v2.0 http://wiki.mikrokoetter.de/BL-Ctrl_2.0 Zugriff: 23.03.2015

Zum schreiben der Drehgeschwindigkeit steht ein 12-Bit Stellwert zur Verfügung. Die Sequenz zum Schreiben eines Stellwertes ist in Abbildung 9.35 dargestellt. Dabei beinhaltet das erste Datum die 8-MSBs und die obersten 3-Bit des zweiten Datum enthalten die 3-LSBs des Stellwertes. Die restlichen Bits des zweiten Datums müssen 0 sein. Sind diese ungleich 0, sind die BL-Controller im Konfigurationsmodus.

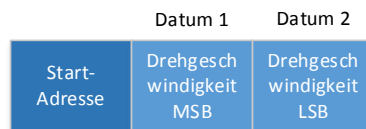


Abbildung 9.35: I²C Sequenz zum Schreiben der Drehgeschwindigkeit eines Motors

Der jeweilige Motortreiber wird mit Hilfe der in Abbildung 9.36 dargestellten Sequenz konfiguriert. Die abgebildete Sequenz zeigt die beispielhafte Konfiguration eines Motortreibers in die linke bzw. rechte Drehrichtung, welche bei nicht drehenden Motoren ausgeführt werden muss. Dabei werden die restlichen Einstellungsmöglichkeiten (Datum 5 - 8) ignoriert und nur die Drehrichtung wird in das EEPROM des BL-Controllers übernommen. Am Ende dieser Sequenz wird ein CRC-Datum gesendet, welches sich aus der Summe der vorherigen 9 Daten und dem Offset 0xAA berechnet.

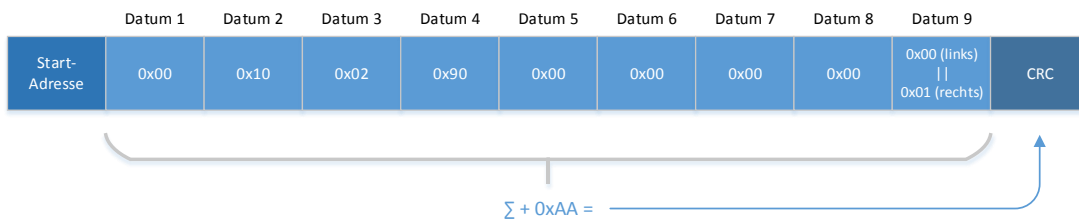


Abbildung 9.36: I²C Sequenz zum Setzen der Motordrehrichtung im BL-Controller

In diesem Projekt werden der vordere und der hintere Motor rechtsdrehend und der linke sowie der rechte Motor linksdrehend eingestellt. Weitere Informationen zu Ansteuerung der BL-Controller sind der Doxygen Dokumentation zu entnehmen.

9.5.3 BRAM Controller

Die gesamten Daten, welche der Dataminer dem ASG zur Verfügung stellt, werden in das BRAM zwischen ASG und AEVV kopiert. Dabei werden ebenfalls die BRAM Adressen aus `bram_address_map.h` (siehe Abbildung 9.13) verwendet. Nach dem Kopiervorgang werden zusätzlich noch die aktuellen Motorstellwerte und die für die Flugregelung benötigten angepassten Fernsteuerungswerte in das BRAM geschrieben. Diese Werte werden unter anderem zur Telemetriestation gesendet um diese zu überwachen (siehe Kapitel 9.7.1).



9.6 Realisierung des AEVV-Systems

Um eine Implementierung für das AEVV-System vorzunehmen, wird an die im Kapitel 7.2.3 erstellten Komponenten- und Datenflussdiagrammen angeknüpft. Um eine erste Übersicht zu schaffen, wird eine statische Struktur von dem gesamten AEVV-System erstellt. Das dazugehörige Klassendiagramm findet sich in Abbildung 9.37 wieder. Die bereits in den Anforderungen und in den Komponentendiagrammen aufgestellten Hauptaufgaben lassen sich ebenfalls im Klassendiagramm wiederfinden. Die Architektur des AEVV-Systems besteht aus folgenden Modulen:

- **CAevvController:** Der AEVV-Controller hat die Aufgabe, das AEVV-System zu konfigurieren, die Subkomponenten zu verwalten und den Datenaustausch zwischen diesen durch eine Interprozesskommunikation (IPC) bereit zu stellen.
- **CConfigurationManager:** Die Klasse der Konfigurationseinheit bietet Funktionen an, um Einstellungsparameter aus einer XML-Datei zu lesen.
- **IBramDriver:** Diese Schnittstelle ermöglicht einen einheitlichen Zugriff auf den BRAM des ASGs auf dem Microblaze.
- **IClient:** Die Aufgabe, die Telemetriedaten an die Telemetriestation zu versenden, soll durch das Interface IClient abgedeckt werden.
- **IDataStorage:** Diese Schnittstelle bietet Funktionen an, um die Telemetriedaten in einem Objekt zu kapseln und um den anderen Komponenten diese zur Verfügung zu stellen.
- **CTask:** Diese abstrakte Klasse stellt dem Controller Funktionen zum Starten und Stoppen der Subkomponenten bereit und bewirkt somit eine Generalisierung dieser für den Controller.
- **CVideoGrabber:** Diese Klasse ist ein Wrapper, um auf die Bibliothek des Video-Streamings zugreifen zu können.
- **CObjectDetection:** Diese Klasse ist ein Wrapper, um auf die Bibliothek der Objekterkennung zugreifen zu können.
- **CGimbalController:** Diese Klasse ist ein Wrapper, um auf die Bibliothek des Gimbal-Controllers zugreifen zu können.

In den folgenden Abschnitten werden die einzelnen Komponenten betrachtet und detaillierter erklärt.

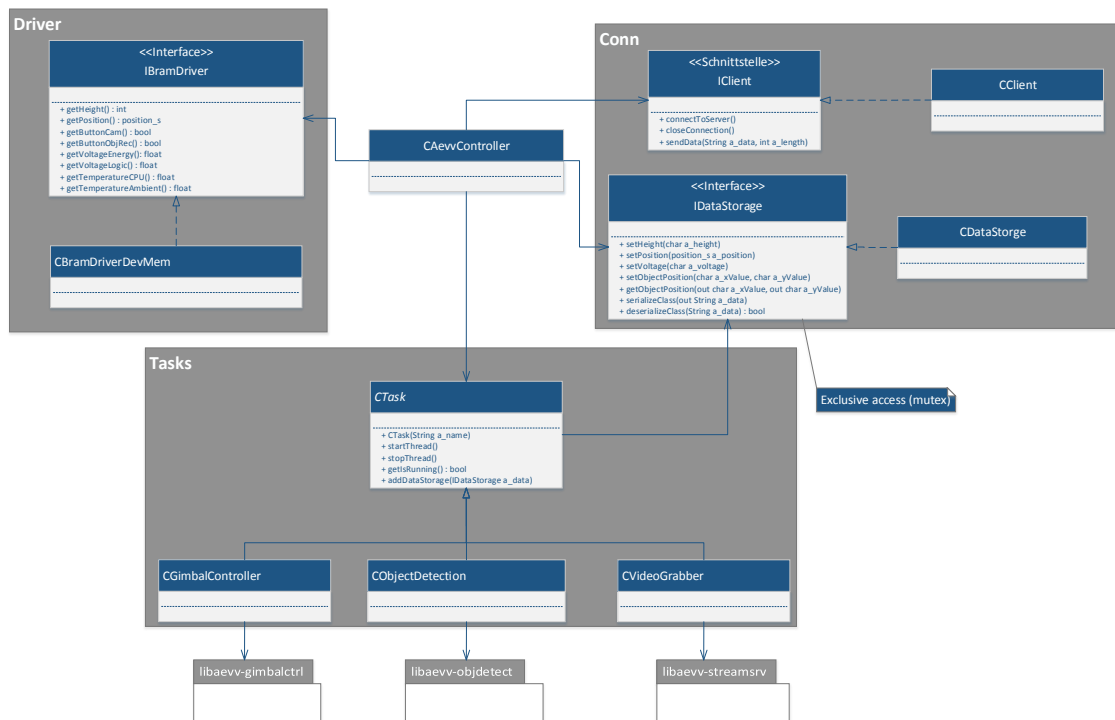


Abbildung 9.37: Statische Struktur des AEVV-Systems (grob)

9.6.1 Die AEVV-Verwaltungseinheit

Die Verwaltungseinheit des AEVV-Systems realisiert die Klasse `CAevvController`. Dessen Aufgaben sind u.a.

- Konfiguration des AEVV-Systems durch Einlesen einer XML-Datei,
- Subkomponenten (externe Bibliotheken) abhängig der Schalterstellungen auf der Funksteuerung starten und stoppen,
- Daten aus dem ASG-BRAM lesen,
- Netzwerkverbindung zur Telemetriestation aufbauen und
- Datenaustausch zwischen den Subkomponenten durch Interprozesskommunikation ermöglichen.

Durch den Methodenaufruf von `CAevvController::startAevv()` kann die Hauptroutine des AEVV-Controllers gestartet werden. Diese Methode besteht aus einer „Endlosschleife“, welche den Ablauf aus Abbildung 9.38 durchführt. Vor dem Betreten dieser Schleife wird eine Netzwerkverbindung aufgebaut und das periodische Versenden gestartet. Danach werden periodisch die neuen Daten aus dem BRAM des ASGs ausgelesen und im `CDataStorage`-Objekt abgelegt sowie die Aktualisierung des Zustands des AEVV-Systems (s. Abbildung 9.38, **Check Switch Control**) durchgeführt. Hinter der Aktivität **Check Switch Control** verbirgt sich ein Zustandsautomat, der in Abbildung 9.39 abgebildet ist.

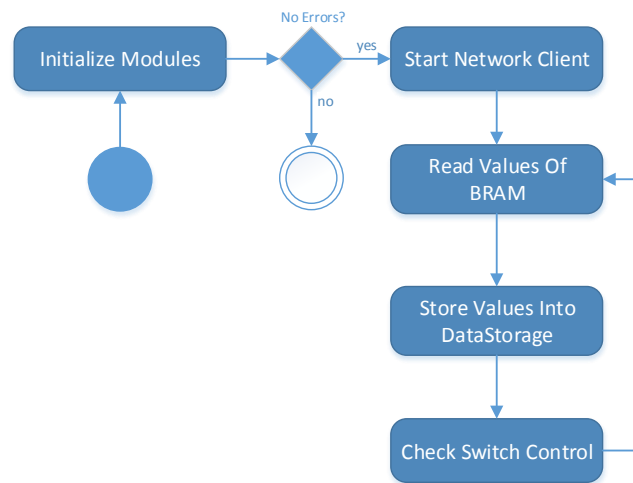


Abbildung 9.38: Ablaufdiagramm für das Starten des AEVV-Controllers

Eine Änderung des Zustands kann durch Modifikation der Schalterstellungen auf der Funksteuerung erreicht werden. Die Schalterstellungen werden ebenfalls aus dem BRAM gelesen. Der vollständige Zustandsautomat ist in Abbildung 9.39 dargestellt. Auf der Funksteuerung wird jeweils eine Schalterstellungen für das (De-) Aktivieren des Video-Streamings sowie eine für das (De-) Aktivieren der Objekterkennung und -verfolgung reserviert. Daraus ergeben sich folgende Zustände:

- **ALL_OFF**: Beide Schalter sind deaktiviert, folglich ist weder das Video-Streaming noch die Objekterkennung und -verfolgung aktiviert. Die Motoren des Gimbals sind ebenfalls ausgeschaltet.
- **CAM_ON_OBJ_OFF**: Der Schalter für die Videoübertragung ist aktiviert. Der Video-Stream zur Telemetriestation ist aktiv, die Motoren des Gimbals werden aktiviert und dieser wird in **Fixed-Mode** (s. Kapitel 9.6.12.1) gesetzt.
- **ALL_ON**: Der Schalter für die Objekterkennung ist ebenfalls aktiviert. Das Video wird ebenfalls an die Bildverarbeitung weitergeleitet, der Gimbal-Controller verlässt den **Fixed-Mode** und versucht das Objekt zu verfolgen.

Andere Zustände sind nicht relevant und mechanisch nicht möglich, da ein Schalter auf der Funksteuerung mit 3 Kippstellungen gewählt wird. Somit ist durch die Hardware und durch die Software gewährleistet, dass die Kamera immer eingeschaltet ist, bevor die Objekterkennung gestartet wird.

In den nachfolgenden Abschnitten werden die Funktionalitäten des AEVV-Controllers durch Verwendung von einzelnen Hilfsmodulen detaillierter erklärt.

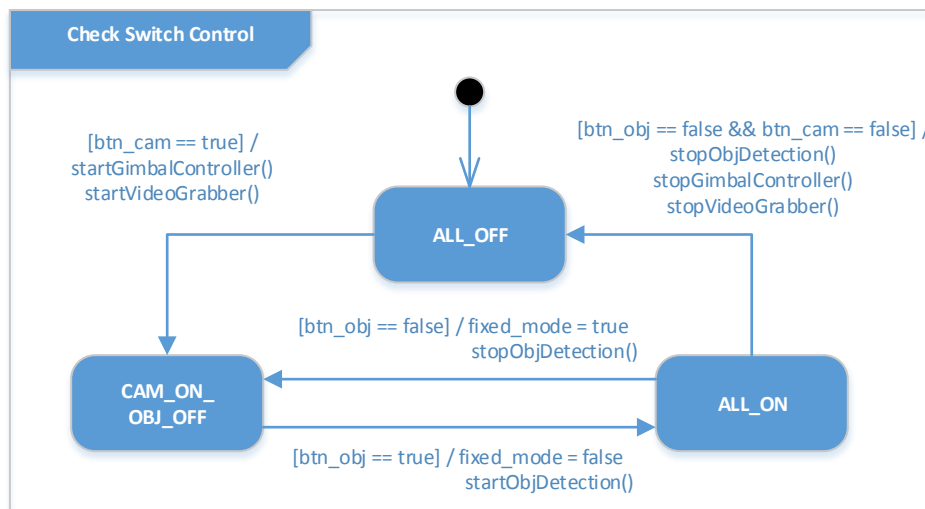


Abbildung 9.39: Interner Zustandsautomat im AEVV-Controller

9.6.2 Konfiguration des AEVV-Systems

Die Parameter für die Konfiguration des AEVV-Systems sollen über eine Konfigurationsdatei bereitgestellt werden. Das Einlesen dieser Parameter wird mit Hilfe der Klasse `CConfigurationsManager` durchgeführt. Diese stellt für jeden Parameter eine passende Funktion bereit. Bevor die Hauptroutine des AEVV-Controllers gestartet wird, müssen zuerst alle Module initialisiert und mit Hilfe dieser Klasse konfiguriert werden.

Die Konfigurationsdatei wird in Form einer XML-Datei angelegt. Der Grund für die Entscheidung einer XML-Datei ist die Existenz von fertigen quelloffenen Bibliotheken, die das Einlesen einer XML-Datei komfortabel gestalten, sowie die einfache Erweiterbarkeit der Datenstruktur. In diesem Projekt wird die Bibliothek *TinyXML-2*⁴⁹ verwendet. In Listing 1 ist ein Auszug aus der XML-Konfigurationsdatei abgebildet.

```

1 <ObjectDetection>
2 <MinRed>140</MinRed>
3 <MinGreen>25</MinGreen>
4 <MinBlue>100</MinBlue>
5 <MaxRed>185</MaxRed>
6 <MaxGreen>62</MaxGreen>
7 <MaxBlue>160</MaxBlue>
8 <SearchArea>25</SearchArea>
9 <!-- negative dilat size will deactivate the dilatation -->
10 <DilatSize>3</DilatSize>
11 </ObjectDetection>
  
```

⁴⁹Weiterführende Informationen zur verwendeten XML-Parser-Bibliothek lassen sich der folgenden Quelle <http://www.grinninglizard.com/tinyxml2/> (letzter Zugriff: 08.03.2015) entnehmen.



```

12 <DataMiner>
   <!-- Memory device in Linux to get access to the BRAM of ASG -->
14 <MemDevice>/dev/mem</MemDevice>
   </DataMiner>
16 <Telemetry>
   <!-- IP address and port of the telemetry station to send the telemetry
        data -->
18 <IpAddress>192.168.2.1</IpAddress>
   <Port>51717</Port>
20 </Telemetry>

```

Listing 1: Auszug aus der Konfigurationsdatei für das AEVV-System

Eine sehr gute Alternative zu XML ist JavaScript Object Notation (JSON)⁵⁰.

9.6.3 Wrapper-Klassen für die Bibliotheken der Subkomponenten

Damit der Zugriff auf die Bibliotheken für das Video-Streaming, für die Objekterkennung und für die Objektverfolgung vereinfacht werden kann, existieren folgende Wrapper-Klassen für die jeweiligen Aufgaben:

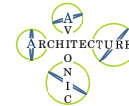
- CObjectDetection: Objekterkennung,
- CGimbalController: Objektverfolgung und
- CVideoGrabber: Video-Streaming.

Diese Wrapper-Klassen stellen u.a. eine Initialisierungsroutine für jede Komponente bereit, welcher diverse Konfigurationsparameter übergeben werden können. Diese Initialisierungsfunktionen werden vor dem Starten des AEVV-Systems aufgerufen.

Der AEVV-Controller startet die jeweilige Aufgabe zur Laufzeit als einen eigenständigen Thread⁵¹, sodass alle Subkomponenten quasi-parallel ausgeführt werden können. Innerhalb des Threads wird die jeweilige Hauptroutine der Subkomponente aufgerufen. Um solch ein Konzept zu ermöglichen, werden die Wrapper-Klassen durch die abstrakte Klasse CTask generalisiert. Die abstrakte Klasse stellt u.a. eine run()- und eine stop()-Funktion bereit, welche jeweils durch einen Wrapper realisiert und durch den AEVV-Controller aufgerufen wird.

⁵⁰ „Die Syntax von JSON ist einfacher gestaltet und erscheint daher oft lesbarer und insbesondere leichter schreibbar. In der Regel reduziert JSON auch den Overhead im Vergleich zu XML“ (http://de.wikipedia.org/wiki/JavaScript_Object_Notation, Letzter Zugriff: 09.04.2005).

⁵¹ Eine Aufgabe eines Prozesses kann in kleinere Tasks aufgeteilt werden, welche in mehreren Threads abgearbeitet werden können. Diese Threads sind diesem Prozess untergeordnet und können folglich durch das Betriebssystem parallel ausgeführt werden.



9.6.4 Realisierung der Avionik-Schnittstelle zum ASG

Die Schnittstelle zur Avionik ist notwendig, um die aktuellen Telemetriedaten des ASGs (Sensor-, Motorstellwerte) sowie die notwendigen Schalterstellungen zu erhalten. Diese sollen daraufhin über eine Netzwerkverbindung an die Telemetriestation versendet werden. Um solch eine Schnittstelle zu ermöglichen, werden die benötigten Daten in einem Block-Random-Access-Memory (BRAM) im Microblaze abgelegt. Der AEVV-Controller hat aufgrund der entwickelten Plattform-Architektur die Möglichkeit, lesend auf diesen BRAM zuzugreifen. Um diesen lesenden Zugriff auf den BRAM zu gewährleisten, wird der *Memory Mapped I/O*-Ansatz mit Hilfe des Linux-Betriebssystems verfolgt. Dadurch kann ein *user-mode I/O*-Gerätetreiber entwickelt werden. Dieser bildet einen gewählten Adressraum aus dem physikalischen Arbeitsspeicher (RAM) in den virtuellen Adressraum des aktuellen Prozesses ab. Bei diesem Adressraum handelt es sich um den vom benötigten BRAM vom ASG-Microblaze, welcher beim Betriebssystem unter einer definierten Adresse registriert werden kann. Diese kann der *devicetree*-Datei vom Betriebssystem entnommen werden, die beim Booten geladen wird. Nachdem der BRAM beim Betriebssystem registriert wurde, befindet sich dieser unter der bekannten Adressierung im physikalischen Arbeitsspeicher.

Der gewählte *user-mode I/O*-Gerätetreiber kann mit Hilfe von Linux-Bibliotheken den physikalischen Arbeitsspeicher als virtuelle Gerätedatei (`/dev/mem`) öffnen und durch die Linux-Funktion `mmap()` einen physikalischen Adressraum in den aktuellen Prozessadressraum abbilden. Voraussetzung für die Durchführung dieser Schritte ist, dass der auszuführende Prozess die jeweiligen Rechte (i.d.R. *root*-Rechte) besitzt. Dieses Vorgehen ist in Abbildung 9.40 dargestellt.

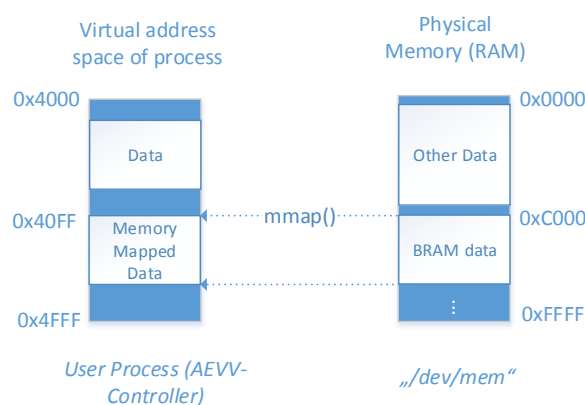


Abbildung 9.40: Schnittstelle zum BRAM der Avionik durch Memory-Mapped I/O und einem *user-mode I/O*-Gerätetreiber



Der `mmap()`-Funktion werden als Parameter u.a. die physikalische Startadresse des BRAMs (hier: `0xC000`) und die Zugriffsart (lesend oder schreibend) übergeben. Als Rückgabewert dieser Funktion erhält man die neue Startadresse im virtuellen Adressraum des Prozesses (hier: `0x40FF`). Da die Werte im BRAM des ASGs an fixe und bekannte Adressen abgelegt werden, können diese ebenfalls im BRAM-Treiber verwendet werden. Dazu muss der jeweilige Adress-Offset auf die neue virtuelle Startadresse addiert werden, um den jeweiligen Wert zu erhalten. Die benötigten Adressen für das AEVV-System liegen in der `bram_address_map.h`-Datei im `driver`-Ordner. Für den direkten lesenden Zugriff auf den Memory-Mapped I/O-Adressraum wird eine Template-Funktion namens `readReg(uint32_t a_offset)` innerhalb der Klasse `CBramDriverDevMem` bereitgestellt, welche den Inhalt an der jeweiligen Adresse zurückgibt. Über das Interface `IBramDriver` wird für jedes Telemetrie-Datum eine Funktion bereitgestellt, sodass sich andere Komponenten (hier: AEVV-Controller) einfach dieser Schnittstelle bedienen können.

Durch dieses Konzept ergeben sich folgende Vorteile [And]:

- Einfache Lösung, da keine Treiber beim Kernel registriert werden müssen
- Sinnvoll bei schnellen Lösungen, gerade bei Prototypen
- Aufgrund der Einfachheit eine portable und flexible Lösung

Jedoch verbergen sich hinter diesem Konzept auch Nachteile, welche nicht zu vernachlässigen sind [And]:

- Keine Sicherheit bei Speicherzugriffen, unerlaubte Zugriffe sind möglich
- Physikalische Adressen müssen bekannt sein

Um diesen Nachteilen entgegenzuwirken, gibt es noch eine weitere Option. Anstatt einen Treiber im `user-space` zu werden, kann ein richtiger Kernel-Treiber entwickelt werden [And]. Dieser muss beim Kernel registriert werden. Diese Methode zieht jedoch eine hohe Entwicklungszeit mit sich und wird im Rahmen dieser Projektarbeit nicht weiter verfolgt. In den nächsten Abschnitten wird auf die Datenhaltung und das Versenden der Telemetriedaten innerhalb des AEVV-Systems eingegangen.

9.6.5 Datenhaltung der Telemetrieparameter im AEVV-System

Damit die Telemetriedaten zentral und gekapselt im AEVV-System abgelegt werden können, gibt es ein Interface `IDataStorage`, welches von der Klasse `CDataStorage` realisiert wird. Der AEVV-Controller erzeugt ein einziges Objekt von dieser Klasse und stellt dies den anderen Komponenten zur Verfügung. Da die Subkomponenten durch Threads parallel ausgeführt werden und da folglich über die Zeit gleichzeitige Zugriffe auf dieses Objekt entstehen können, müssen die Methoden einen exklusiven Zugriff auf die Daten gewährleisten. Dadurch werden Seiteneffekte wie beispielsweise die Erzeugung von inkonsistenten Daten vermieden. Um dieses Konzept zu ermöglichen, gibt es zwei Ansätze:



Ansatz 1: Rückgabewert und Argumentenliste der Get- und Set-Methoden der Klasse als Referenzparameter mit Semaphoren ausstatten.

Ansatz 2: Rückgabewert und Argumentenliste der Get- und Set-Methoden der Klasse als Wertparameter (Kopien) der Objekte realisieren.

Ansatz 1 ist deutlich effizienter als der zweite, da nur mit Referenzen gearbeitet wird und somit keine neuen Kopien der Objekte angelegt werden müssen. Hierbei ist jedoch zu beachten, dass mehrere Threads auf der gleichen Referenz arbeiten, sodass ein gleichzeitiger Zugriff ausgeschlossen werden muss. Es können zwei Fälle eines gleichzeitigen Zugriffs auftreten, welche beide bei diesem Ansatz beachtet werden müssen:

Fall 1: Die Get- und Set-Methode zu der gleichen Datenstruktur aus `CDataStorage` werden simultan durch verschiedene Threads aufgerufen.

Fall 2: Während die Get- oder die Set-Methode zu einer Datenstruktur aufgerufen wird, ändert sich der Inhalt des Referenzparameters bzw. der Datenstruktur.

Das Auftreten dieser beiden Fälle kann durch Verwendung atomarer (nicht unterbrechbare) Operationen bzw. exklusiver Zugriffe durch Semaphoren ausgeschlossen werden. Für die Realisierung bedeutet dies, dass beispielsweise sowohl der Referenzparameter als auch das Klassenattribut mit jeweils einer eigenen Semaphore ausgestattet werden muss, welche gleichzeitig durch den Funktionsaufruf akquiriert werden müssen.

Der zweite Ansatz ermöglicht automatisch durch das *Call-By-Value*-Prinzip einen exklusiven Zugriff auf die Daten. In diesem Falle wird von jedem Datum erst eine Kopie erzeugt und dann zurückgegeben bzw. der Funktion als Parameter übergeben. Da auf einer völlig neuen Kopie Modifikationen vorgenommen werden, muss der Parameter selbst nicht mehr mit einer Semaphore ausgestattet werden. Somit kann der Fall 2 nicht mehr eintreten. Es ist lediglich sicherzustellen, dass bei der Rückgabe einer Datenstruktur diese nicht gerade durch einen Aufruf der Set-Methode durch einen anderen Thread modifiziert wird (Fall 1). Folglich müssen die Get- und Set-Methode zu einem Attribut weiterhin jeweils mit einem eigenen Semaphor ausgestattet werden. Innerhalb dieses Projekts wird der Ansatz 2 verfolgt.

Um den Entwickler zu signalisieren, dass Funktionen keine Modifikationen an Klassenattributen vornehmen und somit keine Seiteneffekte hervorrufen können, werden diese Funktionen bei der Deklaration mit dem C++-Konstrukt `const` deklariert.



9.6.6 Aufbau einer Netzwerkverbindung zur Telemetriestation

Eine weitere Anforderung an das AEVV-System ist das Versenden von Telemetriedaten über eine drahtlose Netzwerkverbindung an eine Telemetriestation. Für eine erfolgreiche Umsetzung dieser Anforderung wird eine konventionelle Socket-Verbindung verwendet. In dieser Ausarbeitung ist das AEVV-System ein Client und die Telemetriestation realisiert einen Server.

Es existiert der Ordner `conn` innerhalb der Software-Struktur, in dem sich das Interface `IClient` befindet. Diese Schnittstelle soll es anderen Modulen ermöglichen, eine Verbindung zu einem Server aufzubauen und an diesen Daten zu versenden. Eine Empfangsroutine ist innerhalb dieses Projekts nicht vorgesehen. Eine mögliche Realisierung dieses Interfaces wird durch die Klasse `CClient` durchgeführt, welche eine Transmission Control Protocol (TCP)-Socket-Verbindung zum Server aufbaut. Dazu wird die IP-Adresse und der Port, auf dem der Server lauscht, benötigt. Beliebige Daten können über die Klassenmethode `sendData()` versendet werden.

Speziell in diesem Projekt werden die Telemetriedaten über dieses Interface verschickt. Da diese Aufgabe als eine parallele, eigenständige Task angesehen werden kann, wird das Versenden der Telemetriedaten zyklisch durch einen separaten Thread durchgeführt. Damit diese Task eine ausführbare Thread-Funktion besitzt und gleichzeitig Zugriff auf die Telemetrie erhält, erbt die Klasse `CClient` von der abstrakten Klasse `CTask`. Diese stellt eine `run()`-Methode bereit, in der die Hauptroutine des Threads implementiert wird. Für die Klasse `CClient` bedeutet dies, dass innerhalb dieser Funktion der Verbindungsaufbau und das Versenden der Telemetriedaten realisiert wird. Außerdem wird ein Verbindungsabbruch bemerkt und folglich ein erneuter Verbindungsaufbau gestartet. Ein Verbindungsabbruch kann mit Hilfe der Bibliotheksfunktion `write()` aus `unistd.h` erkannt werden. Diese Funktion gibt als Rückgabewert die Anzahl an versendeten Bytes zurück oder, im Fehlerfall, eine negative Zahl. Tritt solch ein Fehlerfall ein, kann man daraus schließen, dass es Probleme mit der Netzwerkverbindung gibt.

Damit der Server und der Client eine einheitliche Interpretation der zu versendenden und der zu empfangenen Daten haben, muss eine Art Software-Protokoll eingeführt werden. Im nachfolgenden Abschnitt wird dies näher erläutert.

9.6.7 Übertragungsprotokoll für das Versenden der Telemetriedaten

Als Basistechnik für das Softwareprotokoll wird das Key-Value-Prinzip umgesetzt. Dies bedeutet, dass jedes Datum (Value) einen eindeutigen Schlüssel (Key) besitzt. Folglich wird dieser Schlüssel zusammen mit dem jeweiligen Datum über das Netzwerk versendet. Eine Methodik für das Austauschen dieser Daten ist das Versenden einer Anfrage vom Server an den Clienten. Der Server sendet ein `Request` mit dem jeweiligen Schlüssel an



den Client. Dieser wiederum antwortet mit einer **Response**-Nachricht, welche das jeweilige Datum beinhaltet.

Dieses Verfahren erzwingt ein erhöhtes Kommunikationsaufkommen, da beide Seiten aktiv senden und empfangen müssen. Außerdem ist es für diese Anwendung nicht optimal, da jedes Mal alle Daten benötigt werden. Folglich ist es sinnvoller, dass der Client (hier: das AEVV-System) autark und kontinuierlich alle Daten in einem Schritt an den Server (hier: Telemetriestation) sendet. Der Server hingegen wartet passiv auf neue Daten, die er entgegennehmen kann. Um solch eine Kommunikation zu ermöglichen, werden alle Daten der Klasse `CDataStorage` in eine Zeichenkette als Key-Value-Paar gespeichert und über die Senderoutine des Clients versendet.

Damit beide Seiten die Daten einheitlich interpretieren, wird folgendes Protokoll eingeführt:

- Jede Zeichenkette beginnt mit einem Startzeichen (vgl. `CDataStorage::START`) und schließt mit einem Endzeichen (vgl. `CDataStorage::END`) ab.
- Jedes Datum erhält einen eindeutigen Schlüssel (vgl. `CDataStorage::code_e`).
- Der Schlüssel und das Datum sind durch ein Trennzeichen voneinander getrennt (vgl. `CDataStorage::SEPARATOR`).
- Die Key-Value-Paare sind durch ein weiteres Trennzeichen voneinander getrennt (vgl. `CDataStorage::NEXT`).

Im aktuellen Entwicklungsstand sind folgende Symbole für die Parameter gewählt:

- `CDataStorage::START := '~'`,
- `CDataStorage::END := '$'`,
- `CDataStorage::SEPARATOR := '='` und
- `CDataStorage::NEXT := '|'`.

Ein Auszug aus dem Gesamtergebnis der Zeichenkette sieht wie folgt aus:

```
"~0=12.4000|1=11.3000|2=9.6000|3=11.3000|4=7.4000|5=43|...|33=200.2000$"
```

9.6.8 Erweiterung der Telemetriedaten innerhalb des AEVV-Systems

Falls der Fall eintreten sollte, dass weitere Telemetriedaten über die Netzwerkverbindung verschickt werden sollen, müssen gewisse Adaptionen im AEVV-System vorgenommen werden. Dazu müssen folgende Schritte durchgeführt werden:

1. Befindet sich das Datum im BRAM der ASG-Komponente? Falls ja, muss das Interface `IBramDriver` um die jeweilige Funktion und die Datei `bram_address_map.h` um die physikalische Adresse erweitert werden.
2. Das Interface `IDataStorage` muss mit einer Set- und einer Get-Funktion für das Datum erweitert werden. Des Weiteren muss ein Klassenattribut für das Datum mit einem passenden Datentyp angelegt werden.



3. Das neue Klassenattribut muss bei der Serialisierung und Deserialisierung des Objekts vom Typ `IDataStorage` berücksichtigt werden.
 - Die Enumeration `CDataStorage::code_e` um einen neuen Schlüssel erweitert werden.
 - In der Funktion `CDataStorage::serializeClass()` muss das neue Klassenattribut in den String eingepflegt werden.
 - In der Funktion `CDataStorage::mapValueToAttribute()` muss das neue Klassenattribut eingepflegt werden.
 - Der Zuweisungsoperator `CDataStorage::operator=()` muss um das neue Klassenattribut erweitert werden.
4. Einerseits können die Methodenaufrufe des neuen Klassenattributs innerhalb des AEVV-Controllers in der Funktion `updateDataStorage()` stattfinden, andererseits können diese innerhalb eines Threads der Subkomponenten aufgerufen werden.

9.6.9 Starten des AEVV-Systems

Der Quellcode zum AEVV-System lässt sich unter dem Pfad `<Pfad zum git>/sw/aevv/` auffinden. Unter diesem Pfad liegen die Bibliotheken für die drei Subkomponenten (Video-Streaming, Objekterkennung und Gimbal-Controller) sowie der AEVV-Controller selbst. Der AEVV-Controller kann durch den Aufruf von `aevv-controller` gestartet werden. Um diesen auf dem lokalen Rechner ausführen zu können, gibt es verschiedene Startoptionen. Hintergrund dieser Parameteroptionen ist, dass weder auf dem lokalen Rechner ein BRAM existiert noch zwingend die Funksteuerung vorhanden ist. Folgende Startparameter können beim Aufruf von `aevv-controller` übergeben werden:

- `-dev [cam_on | all_on | usr_ctrl]`: Der BRAM sowie die Schalterstellungen werden durch eine Fake-Klasse emuliert. Dabei kann der Video-Stream-Schalter durch `cam_on` bzw. der der Objekterkennungsschalter durch `all_on` eingeschaltet werden. Durch `usr_ctrl` kann der User die Schalter durch Konsoleneingaben ('1': Schalter ein, '0' Schalter aus) steuern.
- `-dF | -debugFrame`: Der Rechner mit der angegebenen IP-Adresse in der `config.xml` erhält einen zusätzlichen Video-Stream, auf dem das erkannte Objekt gekennzeichnet wird.
- `-help`: Ausgabe einer Hilfe innerhalb der Konsole.

9.6.10 Streaming Server

Der Streaming Server nimmt das Videobild von der Kamera über USB entgegen und stellt es der Objekterkennung und der Telemetriestation zur Verfügung. Für die Videoverarbeitung wurde das Multimedia Bibliothek `GStreamer` eingesetzt. `GStreamer` ist eine Sammlung von Elementen die jeweils eine spezielle Aufgabe erfüllen. Es gibt z.B. Elemente

um ein Video zu skalieren, zu komprimieren oder die Bildrate anzupassen. Die Elemente lassen sich untereinander verbinden um eine komplexe Funktionalität oder sogar eine ganze Anwendung zu erzeugen. Die Verbindung von mehreren Elementen wird als Pipeline bezeichnet. Die für den Streaming Server erstellte Pipeline ist in Abbildung 9.41 dargestellt. Zunächst wird das Kamerabild über das Element *v4l2src* von der Kamera in einer Auflösung von 1280x720 Pixeln, 30 Bildern pro Sekunde und codiert als Moving JPEG (MJPEG) entgegengenommen. Das Element *tee* dupliziert jedes empfangene Bild um es separat für die Telemetriestation und Objekterkennung bereitzustellen. Die einzelnen Bilder für Telemetriestation werden durch das Element *rtpjpegpay* in mehrere Pakete aufgeteilt, falls sie zu groß für ein UDP-Datenpaket sind. Anschließend werden die Bildpakete durch das Element *udpsink* in das Real Time Protocol (RTP) verpackt und über eine UDP Verbindung an die Telemetriestation gesendet. Für die Objekterkennung wird die Bildwiederholungsrate zunächst durch das Element *videorate* auf 15 Bilder pro Sekunde herabgesetzt um die Prozessorauslastung zu reduzieren. Anschließend werden die Bilder durch das Element *jpegdec* von MJPEG zu Video-Rohdaten dekodiert, die für die Objekterkennung benötigt werden. Um die Prozessorauslastung weiter zu reduzieren, werden die Bilder durch das Element *videoscale* auf die Hälfte der Auflösung runterskaliert. Abschließend werden die Bilder durch die Elemente *videoconvert* und *appsink* zunächst in ein kompatibles Pixelformat konvertiert und dann an die Objekterkennung weitergeleitet. Je nach Bedarf, können die Videoströme zur Telemetriestation und zur Objekterkennung separat voneinander pausiert und wieder abgespielt werden. Sind beide Videoströme unterbrochen, wird die komplette Pipeline angehalten und die Prozessorauslastung durch den Streaming Server reduziert sich von etwa 58 auf etwa zwei bis drei Prozent. Die Auslastung für den Videostrom zur Telemetriestation beträgt etwa 13 Prozent und etwa 45 Prozent für den Videostrom zur Objekterkennung.

9.6.11 Objekterkennung

Auf Basis der Voruntersuchungen aus Kapitel 3.3 wurde die Objekterkennung implementiert. Für die Zielplattform wurden allerdings einige Anpassungen gemacht. Da die Mobius Kamera Rauscharme Bilder liefert, wird im Vorverarbeitungsschritt die Bildfilterung weggelassen. Dadurch kann Prozessorlast reduziert werden. In diesem Schritt wird lediglich das Bild, wenn das Objekt im vorherigen durchlauf der Objekterkennung gefunden wurde, auf die Region of Interest verkleinert um die Prozessorlast die die Objekterkennung benötigt zu verringern. Der ROI Wert kann in der XML Konfigurationsdatei geändert werden. Dieser Wert gibt die Hälfte der Länge des Quadrates an, welches aus dem Bild geschnitten und analysiert wird. Nach dem Vorverarbeitungsschritt wird das Bild, bzw. der Bildausschnitt auf das Wesentliche segmentiert. Hier wird anhand der Farbe die ebenfalls in der XML Konfigurationsdatei festgelegt wurde, das Bild segmentiert, sodass am Ende ein Binäres Bild überbleibt, welches die festgelegte Farbe beinhaltet. Dies geschieht mit

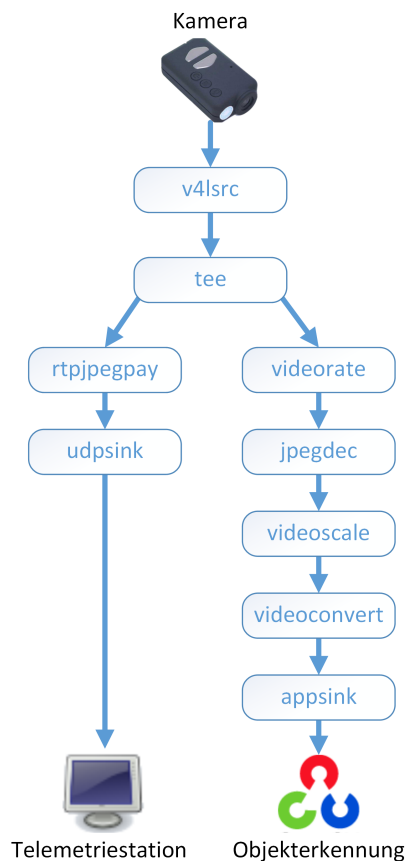


Abbildung 9.41: Aufbau des Videopipeline im Streaming Server.

der `inRange` Methode, die OpenCV bereitstellt. Um den erkannten Bereich künstlich zu vergrößern, kann durch Angabe der Dilatationsgröße die Dilatation des Bildes eingestellt, bzw. aktiviert werden. Ist diese Größe negativ, wird die Dilatation deaktiviert. Im nächsten Schritt wird der Blob Detection Algorithmus auf das binäre Bild angewendet. Hier wurde keine Library verwendet, sondern eigens entwickelter Programmcode genutzt. Nachdem die zusammenhängenden Bereiche des Bildes berechnet wurden, werden diese analysiert. Da der Ball laut Anforderungen eine einzigartige Farbe auf dem Spielfeld hat, kann davon ausgegangen werden, dass das erste erkannte Objekt der Ball ist. Daher wird nur das Label mit der Nummer eins analysiert und der Mittelpunkt dieses Labels ausgewertet. Das spart wiederum Performance und die Erkennung des Objektes ist robuster gegenüber Verdeckungen, da kleine farbige Bereiche im Bild ausreichen, um den Ball zu erkennen. Nach diesem Schritt wird der nächste ROI Bereich anhand der neuen erkannten Position berechnet. Wenn keine Position erkannt wurde, wird das komplette Bild im nächsten Schritt analysiert und minus eins als erkannte Position für X und Y ausgegeben. Wurde allerdings im letzten Schritt der Ball gefunden, und im diesen Schritt nicht, wird zuerst das komplette Frame noch einmal analysiert und nicht nur der kleine Bildausschnitt, da

9 IMPLEMENTIERUNG DER TEILSYSTEME



9.6 Realisierung des AEVV-Systems

der Ball eventuell im kompletten Bild zu sehen und nur aus den Bildausschnitt verloren gegangen ist. Wurde der Ball erfolgreich erkannt, wird die X- und Y-Position des Balles ausgegeben.

9.6.12 Gimbalcontroller

In diesem Kapitel wird auf die verschiedenen Modi eingegangen, die in dem Gimbal-Controller implementiert sind. Außerdem wird die Kopplung des Gimbal-Koordinatensystems an die Umwelt, sowie eine Methode zur Bestimmung der Entfernung des Balls eingegangen. Zusätzlich wird die verwendete Regelung detailliert beschrieben und auf die Anbindung an den AEVV-Controller eingegangen.

9.6.12.1 Betriebsmodi Der Gimbal-Controller verfügt über vier verschiedene Betriebsmodi. Diese Modi sind im Folgenden aufgelistet und anschließend näher beschrieben.

1. Power-Off-Mode
2. Fixed-Mode
3. Searching-Mode
4. Tracking-Mode

Der Power-Off-Mode wird für zwei Fälle verwendet. Der erste Fall entspricht dem Initialisierungszustand, wenn das AEVV-System eingeschaltet wird und sich der Schalter auf der Graupner MC20 Fernsteuerung in der Position Null (Schalterstellung vom Piloten weg) befindet. Der Zweite Fall tritt ein, wenn sich der Quadrocopter in der Luft befindet, der Ball gesucht oder getrackt wird und die Motoren des Gimbals aus irgendeinem Grund stromlos geschaltet werden sollen. Dies tritt beispielsweise dann auf, wenn der Energy-Akku eine kritische verbleibende Kapazität erreicht hat und Strom gespart werden muss.

Bei dem Fixed-Mode handelt es sich um das Anfahren einer voreingestellten Position, um eine definierte Startpose des Gimbals zu erhalten. Dazu wird die Nick-Achse auf $+45^\circ$ und die Gier-Achse auf 0° gefahren. Die Kamera schaut dann in Flugrichtung des Quadrocopters und blickt in einem 45° -Winkel auf die Szene. Dieser Modus wird dann verwendet, wenn das AEVV-System gestartet oder wenn die Objektverfolgung bzw. der Gimbal-Controller während des Fluges aus- und anschließend wieder eingeschaltet wird.

Der Searching-Mode wird dazu verwendet, das Spielfeld nach dem Zielobjekt, in unserem Anwendungsfall einem roten Ball, zu suchen. In Abbildung 9.42 ist der Arbeitsraum des Gimbals dargestellt, in dem die Kamera jede Position auf der Oberfläche der Halbkugel anfahren kann. Bei dem Searching-Mode folgt die Trajektorie der Kameralinse einem Rechtecksignal. Die Nick-Achse bewegt sich in einem Intervall zwischen 0 und $+90^\circ$ mit einer kontinuierlichen Bewegung. Bei jedem Wechsel der Bewegungsrichtung der Nick-Achse bewegt sich die Gier-Achse um ein Inkrement von 20° weiter. Auf diese Weise bewegt sich die Gier-Achse in einem Intervall von $\pm 180^\circ$.

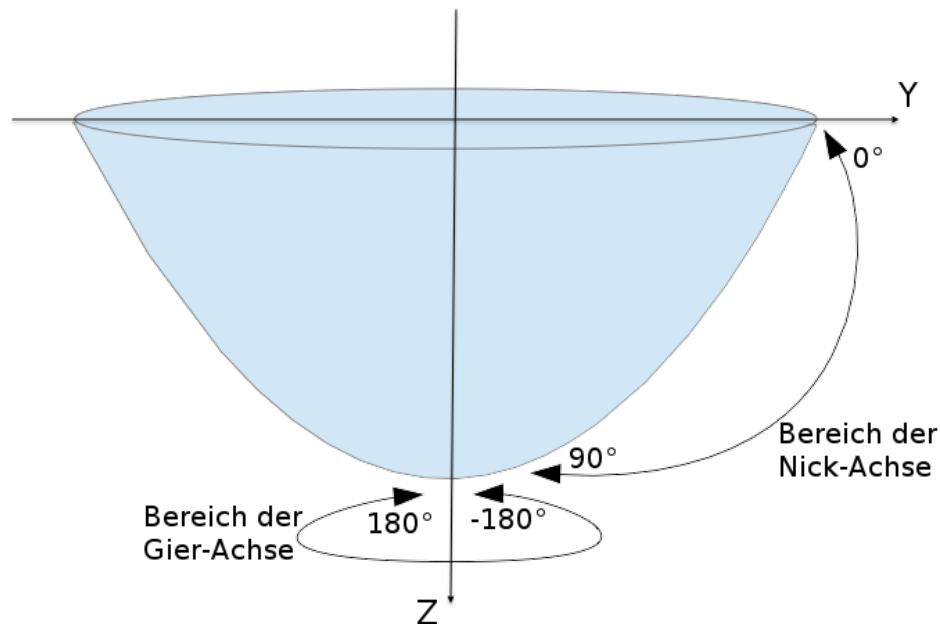


Abbildung 9.42: Definition der Winkelbereiche des Gimbals

Unterbrochen wird der Searching-Mode, sobald der Ball von der Objektverfolgung erkannt wurde und die Ballkoordinaten einen positiven Betrag aufweisen. Die aktuellen Koordinaten des Balls werden in einem Speicherbereich im AEVV-Controller abgelegt. Der Gimbal-Controller liest mit Hilfe einer Call-Back-Funktion aus diesem Speicherbereich aus, damit die Konsistenz der Daten sichergestellt ist und keine Probleme bei einem gleichzeitig auftretenden Lese- und Schreibzugriff entstehen können.

Wenn ein Ball durch die Mobius-Kamera erkannt wurde, wird der Tracking-Modus aktiviert. In der Tracking-Funktion sind zwei Regler implementiert, einen für die Y-Position (Nick-Achse) und einen für die X-Position (Gier-Achse). Die Auflösung, die für das Tracking verwendet wurde, liegt bei 640 x 360 Pixel. Die Regler versuchen jeweils den Regelfehler aus Bildmittelpunkt und aktueller Ballposition zu minimieren. Dadurch wird der Ball in den Bildmittelpunkt gerückt, wie in Abbildung 9.43 dargestellt. Die Aktualisierung der Ballposition erfolgt auf dieselbe Weise, wie bereits im Searching-Mode beschrieben wurde.

Die Regelung des Tracking-Modus erfolgt mit einer Frequenz von 10 Hz. Eine höhere Regelfrequenz würde eine schnellere Reaktion der Nachführung des Balls zur Folge haben, jedoch liegt laut Herstellerangaben die maximale Frequenz zur Aktualisierung der Stellwerte der Achsen bei den verwendeten 10 Hz [Gim]. In Kapitel 9.6.12.4 wird detailliert auf die im Tracking-Mode implementierte Regelung eingegangen.

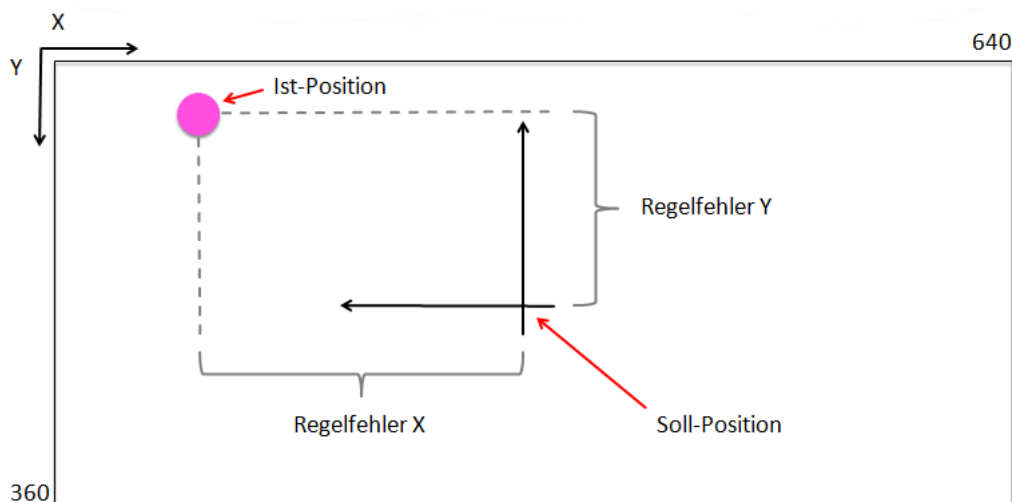


Abbildung 9.43: Schematische Darstellung der Bildung der Regelfehler

9.6.12.2 Kopplung des Gimbal-Koordinatensystems Zur korrekten Ausrichtung ist es notwendig, den Gimbal über ein Koordinatensystem mit der Umwelt zu verbinden. Die internen Stabilisierungsalgorithmen der Gimbal-Elektronik verwenden dazu Messwerte der auf der Rückseite der Kamera angebrachten MPU6050 Internal Measurement Unit. In dieser IMU ist ein 3-Achs Gyroskop, ein 3-Achs Beschleunigungssensor, sowie ein Temperatursensor verbaut. Der Gimbal-Controller im AEVV-System kann auf diese Werte jedoch nicht zugreifen, da diese nicht nach außen zugänglich gemacht werden. Neben der MPU6050 der Gimbal-Elektronik besteht die Möglichkeit, die Sensordaten aus der MPU9150 der Avionik zu verwenden. Diese können aus dem nicht sicherheitskritischen BRAM ausgelesen werden, welcher dem AEVV-System durch den MicroBlaze des ASG zur Verfügung gestellt wird (Abbildung 7.21). Dadurch ist es möglich, eine kinematische Kette aufzustellen, die die Translation und Rotation des Gimbal-Koordinatensystems im Bezug auf des Koordinatensystem der Avionik beschreibt. Auf die Charakteristika kinematischer Ketten wird in Kapitel 3.5.1 näher eingegangen. Die Verwendung kinematischer Ketten als direkte bzw. inverse Kinematik wird in dem Kapitel 3.5.2 bzw. Kapitel 3.5.3 thematisiert.

Eine Form der Darstellung dieser kinematischen Kette ist durch die Parameterisierung der Gelenkübergänge nach Denavit Hartenberg, wie in Kapitel 3.5.5 dargestellt, möglich. Die Translationen und Rotationen entlang der einzelnen Raumachsen werden hierbei durch sechsdimensionale Matrizen dargestellt, die intern mit Euler-Winkeln rechnen. Bei dieser Darstellungsform kann jedoch der in Kapitel 3.4.4 beschriebene Gimbal-Lock auftreten, welcher der dreidimensionalen kardanischen Aufhängung des Gimbals bei bestimmten Stellungen ein oder zwei Freiheitsgrade entzieht. Dadurch wird die Bewegungsfreiheit des Gimbals eingeschränkt und die Kamera kann dadurch nicht mehr alle Punkte des Arbeitsraumes anfahren.



Eine Alternative zur Berechnung des Übergangs des Avionik- in das Gimbal-Koordinatensystem bieten die in Kapitel 3.5.6 beschriebenen Quaternionen. Bei der Rechnung mit Quaternionen besteht nicht die Gefahr, Freiheitsgrade durch bestimmte Achsstellungen zu verlieren, da die Stellungen der Achsen über den kompletten Winkelbereich definiert sind. Der Nachteil der Quaternionen besteht darin, dass rotatorische, jedoch keine translatorischen Bewegungen dargestellt werden können. Deshalb eignen sich die Quaternionen alleine nicht für die hier auftretende Problemstellung. Abhilfe würde die Kombination von Quaternionen und den Stellungsmatrizen schaffen. Dazu würden die rotatorischen Anteile durch die Quaternionen beschrieben werden und die translatorischen Anteile durch die Stellungsmatrizen.

Da der Aufwand für die Aufstellung der kinematischen Kette jedoch zu hoch und die mathematische Implementierung zu komplex ist, wird eine andere Form der Kopplung an die Umwelt gewählt.

Die Positionsberechnung der Achsstellungen wird nicht mehr absolut an der aktuellen Ausrichtung des Quadropters definiert, sondern relativ über die Informationen aus dem Kamerabild. Es wird davon ausgegangen, dass die Gimbal-Elektronik die Kamera unabhängig von der aktuellen Lage des Quadropters immer in der definierten Pose ausrichtet. Dabei wurde jedoch festgestellt, dass die Yaw-Achse des Gimbals eine Drift aufweist, die die Kamera mit einer kontinuierlichen, langsamen Geschwindigkeit um die Yaw-Achse rotieren lässt. Diese Drift kann nicht kompensiert werden, da sich das Koordinatensystem des Gimbals mit der Drift mitbewegt und das System somit keine Rotation detektiert.

9.6.12.3 Entfernungsbestimmung zwischen Ball und Kamera Die Bestimmung der Entfernung zwischen Ball und Kamera dient dazu, die Verfahrgeschwindigkeiten der Achsen des Gimbals zu skalieren. Eine geringe Distanz zwischen Ball und Kamera erfordert schnelle Bewegungen der Achsen, um den Ball im Fokus des Kamerabildes zu halten. Bei großen Distanzen müssen die Geschwindigkeiten der Achsen verringert werden, damit sich diese nicht zu schnell bewegen und der Ball aus dem Fokus des Kamerabildes verloren geht. Eine zu große Skalierung der Geschwindigkeit würde die Achsen außerdem zum Schwingen bringen, wenn diese eine genaue Position anfahren wollen, in der sich der Ball befindet. Um Informationen über die Entfernung zwischen Ball und Kamera zu erhalten, ist es möglich, ein Verfahren aus der Bildgebung in der Medizintechnik zu verwenden. Es wird versucht, über einen bekannten Abstand zwischen Sensor und Objekt die Größe von bestimmten Gewebestrukturen im Körper zu bestimmen. Das inverse Vorgehen könnte dazu verwendet werden, den Kamera-Ball-Abstand zu bestimmen. Dazu ist es allerdings notwendig, die Charakteristika der Kamera auszunutzen. Im Falle des Gimbal-Controllers ist die Entfernung zwischen Ball und Kamera unbekannt, jedoch hat der Ball einen definierten Durchmesser. Deshalb kann das inverse Vorgehen im Bezug auf das erwähnte

medizinische Verfahren verwendet werden. Über die Anzahl der Pixel, die der Ball im Bild ausfüllt, ist es möglich, Rückschlüsse auf die Entfernung zu ziehen. Dazu sind allerdings Referenzmessungen notwendig, die die detektierte Pixelanzahl bei definierten Entfernungen ermittelt.

Außerdem müssen zur Berechnung des Abstandes einige technische Daten der Kamera bekannt sein. Es ist notwendig, die Brennweite und die Größe des in der Kamera verbauten Sensor-Chips zu kennen. Der Abstand des Balls kann anhand dieser Daten und der Pixelanzahl, die der Durchmesser des Balls im Bild einnimmt, berechnet werden. Folgende Formel beschreibt diesen mathematischen Zusammenhang:

$$D = f \cdot \frac{d_{Ball}}{s_{Sensor}} \quad (9.81)$$

In dieser Gleichung beschreibt D die Distanz zwischen Kameralinse und Ball. Dabei ist f die Brennweite der Kameralinse. Je größer die Brennweite ist, desto dichter und größer erscheint der Ball auf dem Bild. Der Parameter d_{Ball} gibt den Durchmesser des Balls an, während s_{Sensor} die physikalische Breite des Sensor-Chips beschreibt.

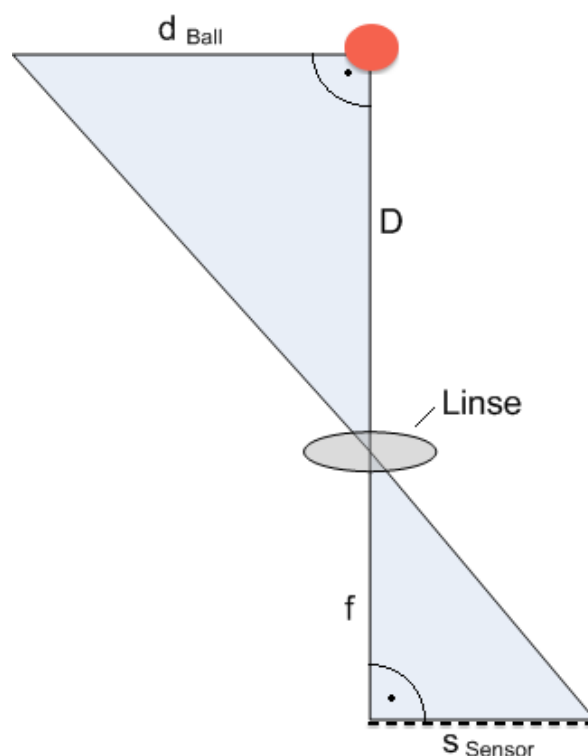


Abbildung 9.44: Abhängigkeit zwischen Ballabstand, Brennweite und Sensorgröße

In der Abbildung 9.44 ist der Zusammenhang der Formel 9.81 schematisch dargestellt.

Es wurde mehrfach versucht, Kontakt zum Hersteller aufzunehmen, um fehlende Kenn-



zahlen zu erfahren. Bei den Kennzahlen, die nicht ermittelt werden konnten, handelt es sich um die genaue Brennweite der Kamera bei der Verwendung der Linse B und die physikalischen Abmessungen des verwendeten Sensor-Chips. Da die Implementierung der Entfernungsbestimmung ohne diese Kennzahlen nicht möglich ist, konnte diese Methode nicht in das AEVV-System und den Gimbal-Controller implementiert werden.

9.6.12.4 Regelung der Achsen im Tracking-Mode Die Verbindung zwischen Gimbal-Controller im AEVV-System und der Gimbal-Elektronik wird über eine USB-Verbindung aufgebaut. Im Linux-Betriebssystem wird die Verbindung als serielle Schnittstelle angesehen, die durch die Gerätedatei in `/dev/Gimbal` repräsentiert wird. Eine detaillierte Beschreibung der Anbindung ist im Kapitel 9.6.12.5 vorhanden.

Die Firmware der Gimbal-Elektronik unterstützt zwei Modi, um die Befehle zum Gimbal zu senden und die Achsen zu verfahren. Diese Modi sind zum einen der Speed-Mode, bei dem eine gewünschte Winkelgeschwindigkeit vorgegeben wird, mit der die entsprechenden Achsen verfahren werden. Zum anderen ist das der Angle-Mode, bei dem ein absoluter Winkel vorgegeben wird, der von der jeweiligen Achse angefahren wird [Gim].

Um eine Regelung zum Verfolgen des Balls zu implementieren, ist die Definition eines Regelfehlers notwendig. Dazu wurde der Bildmittelpunkt als Referenz definiert und entspricht daher der Soll-Position. Die von dem Objektverfolgungsalgorithmus bestimmte Ballposition wird als Ist-Position in die Regelung eingeführt, um den Regelfehler als Differenz zwischen Soll- und aktueller Ist-Position zu bilden. Die X- und Y-Achse des Bildes werden separat von einander behandelt, um jeweils einen Regler für jede Achse zu implementieren. Um die Kamera dem Ball geregelt nachzuführen, wurde der Speed-Mode des Gimbal gesetzt und für die Nick- und Gier-Achse jeweils ein PI-Regler implementiert. Es wurde für die Implementierung ein PI-Regler verwendet, da ein sanftes Nachführen der Kamera gewünscht ist und der D-Anteil ein abruptes Anfahren der Achsen zur Folge haben würde. Der I-Anteil lässt die Achsen sanft anfahren und bremst die Achsen sanft wieder ab. Der I-Anteil ist jedoch vorsichtig zu behandeln, da das System sonst zu schwingen beginnt. Die Stellwerte der Regler wurden direkt als Geschwindigkeitsvorgaben für die Achsen verwendet. Dadurch stellt sich bei einem großen Regelfehler eine große Verfahrgeschwindigkeit für die Achsen ein. Wird der Regelfehler beim Anfahren der Position kleiner, verringert sich die Verfahrgeschwindigkeit ebenfalls proportional.

Bei dem Test dieser Regelungsarchitektur ist jedoch aufgefallen, dass sich die Roll-Achse nicht korrekt verhält und ein nichtdeterministisches Verhalten aufweist. Die Achse stellt zufällige Winkel, obwohl die Achse mit 0° initialisiert wurde. Die Stellwinkel unterscheiden sich bei jedem Durchlauf voneinander und sind nicht reproduzierbar.

Da der Speed-Mode nicht korrekt funktioniert, wurde auf den Angle-Mode umgestellt. Dabei wurden die reinen Algorithmen der PI-Regler unverändert beibehalten. Die Berechnungen der Motorstellwerte wurden jedoch abgeändert. Bei dieser Implementation

wird jeweils der aktuelle absolute Winkel für jede Achse gespeichert. Auf diesen absoluten Winkel wird in jedem Regel-Zyklus der Stellwert der Regler hinzuaddiert. Der Stellwert entspricht dem Inkrement eines relativen Winkels, um den die Achsen verfahren werden müssen, um die vorgegebene Position anzufahren. Die Kombination aus Angle-Mode und PI-Regler führte leider zu einem aufschwingenden und nicht stabilen Verhalten. Es hat sich herausgestellt, dass bei der Aktivierung des Angle-Mode zusätzlich noch ein Motion-Planner aktiviert wird, welcher die Trajektorien der einzelnen Achse des Gimbals berechnet [Gim]. Dieser Motion-Planner interfereert mit den PI-Reglern, sodass keine stabile Regelung bei der Verfolgung des Objektes möglich ist. Eine Anpassung der Regelparameter der Gimbal-Elektronik über die GUI des Herstellers, sowie die Anpassung der Regelparameter der PI-Regler hat das Problem nicht vollständig lösen können.

Abhilfe hat die Verwendung von Mehr-Punkt-Reglern geschaffen. Bei dieser Art von Reglern werden die Stellgrößen nicht dynamisch aus dem Regelfehler generiert, sondern mit definierten Intervallen verglichen. Liegt der Regelfehler in einem entsprechenden Intervall, wie in Abbildung 9.45 zu sehen, so wird der für das jeweilige Intervall definierte Wert als Stellgröße für die jeweilige Achse gesetzt.

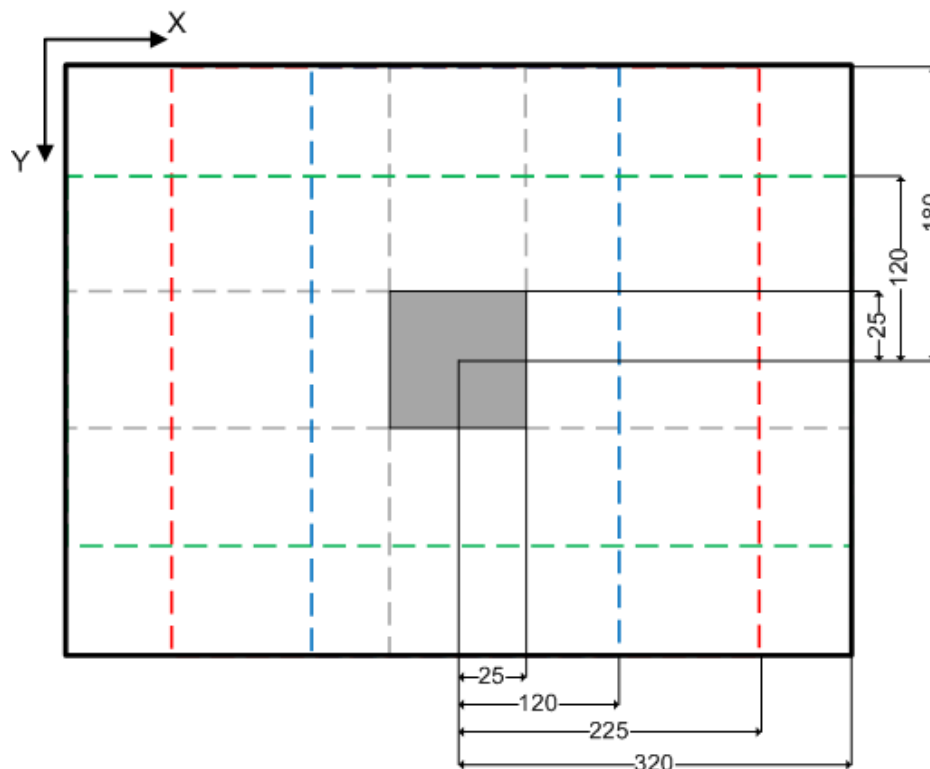


Abbildung 9.45: Definition der Segmente der Mehrpunkt-Regelung

In der Abbildung 9.45 ist das Prinzip der letztlich im Gimbal-Controller implementierten Regelung dargestellt. Es ist zu erkennen, dass für die Nick- und Gier-Achse eine unter-



schiedliche Anzahl von Intervallen vorgesehen ist. Dies ist auf das ungleiche Seitenverhältnis von 16:9 zwischen Breite und Höhe zurückzuführen. Das graue Quadrat in der Mitte mit einer Kantenlänge von 25 px spiegelt den Totbereich der Regelung wider. Befindet sich der von dem Objekterkennungsalgorithmus bestimmte Punkt des Balls in diesem Bereich, so findet keine Ansteuerung der Achsen statt, da der daraus entstehende Regelfehler sehr klein ist. Die X-Achse des Bildes, welche von der Gier-Achse bedient wird, ist insgesamt in sieben Segmente unterteilt. Ausgehend vom Bildmittelpunkt befinden sich jeweils drei Grenzlinien mit den Abständen ± 120 px, ± 225 px und ± 320 px symmetrisch zueinander. Das Segment zwischen ± 25 px und ± 120 px wird `INNER_SEGMENT` genannt und ist mit einer Verfahrensgeschwindigkeit von 1 gewichtet. Der Bereich von ± 120 px bis ± 225 px trägt den Titel `MID_SEGMENT` und enthält als Verfahrensgeschwindigkeit den Wert 2. Zwischen der Grenzlinie bei ± 320 px und dem Bildrand liegt das `OUTER_SEGMENT`, welches mit der Geschwindigkeit 3 versehen ist.

Entlang der Y-Achse, welche von der Nick-Achse des Gimbals bedient wird, sind neben dem Totbereich zwei weitere Segmente definiert. Das `INNER_SEGMENT` der Y-Achse ist äquivalent zu dem Segment der X-Achse und erstreckt sich zwischen ± 25 px und ± 120 px und dem Bildmittelpunkt. Die Gewichtung dieses Intervalls liegt ebenfalls bei 1. Das `OUTER_SEGMENT` Intervall erstreckt sich von ± 120 px bis zum Bildrand und wird mit 2 gewichtet. Dadurch ergeben sich mit dem Totbereich für die X-Achse insgesamt sieben und für die Y-Achse insgesamt fünf Segmente.

9.6.12.5 Anbindung an den AEVV-Controller Im Folgenden wird näher auf die Integration des Gimbalcontrollers in das gesamte AEVV-System eingegangen. Der AEVV-Controller verwaltet alle Teilsysteme des AEVV-Systems und übernimmt dadurch eine übergeordnete Managementaufgabe des Systems, durch ihn wird u. a. auch der Gimbalcontroller in seiner Funktion gesteuert. Der Ablauf dieser Steuerung ist in Abbildung 9.46 dargestellt. Die Entscheidung, wann der Gimbalcontroller in den Power-Off-Mode, den Fixed-Mode oder den Searching- bzw. Tracking-Mode wechselt trifft der Pilot durch die Fernsteuerung, deren Kommunikation zum AEVV-System durch den AEVV-Controller ermöglicht wird. Die einzige Ausnahme dazu bildet der Wechsel vom Searching- in den Tracking-Mode und umgekehrt. Hier findet der Wechsel automatisch im Gimbalcontroller statt. Wenn der Ball für eine längere Zeit im Tracking-Mode nicht gefunden werden kann, dann wird in den Searching-Mode gewechselt und wenn der Ball im Searching-Mode gefunden wurde, dann wird in den Tracking-Mode gewechselt. Die genaue Funktion der nachfolgend verwendeten Modi, kann in Kapitel 9.6.12.1 nachgelesen werden.

Bei der Initialisierung des Gesamtsystems wird auch der Gimbalcontroller initialisiert, dies geschieht durch den Aufruf der `init` Funktion durch den AEVV-Controller. Dadurch wird der Gimbal in den Power-Off-Mode versetzt, d. h. die Motoren werden stromlos

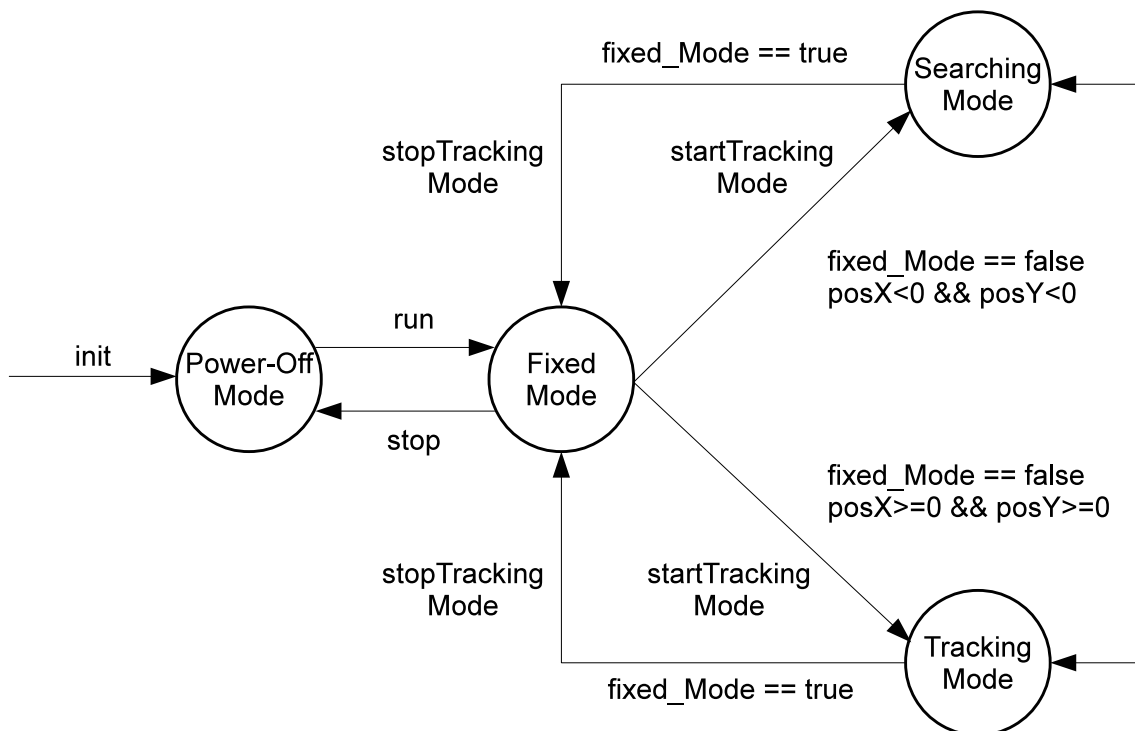


Abbildung 9.46: Übersicht über die Steuerung des Gimbalcontrollers

geschaltet. Eine weitere Aufgabe der `init` Funktion ist es die Referenzen auf die Ballpositionen in X- und Y-Achse zu erhalten, damit später bei der Ballerkennung automatisch in den Tracking-Mode gewechselt werden kann. Weiterhin werden die Segmente für den Mehrpunktregler anhand der Größe des vorliegenden Bildausschnittes dynamisch für die nachfolgende Regelung berechnet.

Wenn der Gimbalcontroller gestartet wird, durch die Funktion `run`, dann geht er standardmäßig in den Fixed-Mode. Dies wird, wie oben beschrieben, durch die Entscheidung des Piloten herbei geführt. Ebenso kann der Gimbalcontroller durch die Funktion `stop` auch wieder in den Power-Off-Mode versetzt werden.

Wenn der Pilot nun über die Fernsteuerung die Entscheidung trifft den Searching- bzw. Tracking-Mode zu aktivieren, ruft der AEVV-Controller die `startTrackingMode` Funktion auf, dadurch wird ein Flag (`fixed_Mode`) gesetzt und somit der Fixed-Mode verlassen. Ob danach in den Searching-Mode oder direkt in den Tracking-Mode gewechselt wird, hängt von der Position des Balles ab, die durch `posX` und `posY` in den jeweiligen Achsen ermittelt werden kann. Diese beiden Datenstrukturen enthalten die oben beschriebenen Referenzen auf die Ballpositionen. Wenn diese kleiner Null sind, also der Ball nicht gefunden wurde, dann wird in den Searching-Mode gewechselt. Ansonsten, wenn der Ball an den angegebenen Koordinaten gefunden wurde, wird in den Tracking-Mode gewechselt. Eine Rückkehr in den Fixed-Mode, ausgelöst durch die Fernsteuerung, wird erneut durch das Flag `fixed_Mode` ermöglicht. Dabei ist sowohl eine sofortige Rückkehr aus der Searching-



Mode Routine, als auch aus der Tracking-Mode Routine vorgesehen. Dies wird jeweils durch die Funktion `stopTrackingMode` im AEVV-Controller vorgenommen.

Es lässt sich also festhalten, dass der Pilot über die Fernsteuerung dem AEVV-Controller mitteilt, welchen der drei Modi (Power-Off-Mode, Fixed-Mode und Searching- bzw. Tracking-Mode) der Gimbalcontroller einnehmen soll. Der AEVV-Controller fragt den aktuellen Mode zyklisch vom Gimbalcontroller ab und kann ihn dann auch auf der Telemetriestation u. a. für den Piloten darstellen.

9.7 Realisierung einer Telemetriestation

Die Telemetriestation hat die Aufgabe, Telemetriedaten vom Quadrocopter zu empfangen und diese graphisch anzuzeigen. Des Weiteren soll das Videobild der Kamera als Live-Stream ebenfalls in einer graphischen Oberfläche angezeigt werden.

Um diese Anforderungen zu erfüllen, werden gewisse Hardware- und Software-Komponenten vorausgesetzt. In den folgenden Kapiteln sollen diese erwähnt werden. Außerdem soll auf die Gestaltung der graphischen Oberfläche (engl. graphical user interface (GUI)) sowie auf die Erzeugung eines drahtlosen Netzwerks durch die Telemetriestation detaillierter eingegangen werden.

Benötigte Hardware

Für die Realisierung einer Telemetriestation werden folgende Hardware-Komponenten benötigt:

- Notebook inkl. USB-Anschluss und
- zwei WLAN USB-Module gleicher Art im 5 GHz Frequenzband.

Benötigte Software

Neben der Hardware werden folgende Software-Komponenten auf dem verwendeten Notebook vorausgesetzt:

- Betriebssystem: Ubuntu 14.04 LTS,
- *hostapd*-Package zur Erzeugung eines virtuellen Access-Points,
- *dnsmasq*-Package zur Verwendung eines DHCP- bzw. DNS-Servers (optional),
- GTK+3-Toolkit zur Anzeige einer GUI,
- Glade v3 als GUI-Designer,
- GStreamer-Package (Version 1.0) und
- *log4cpp*-Package.

Abgesehen vom Betriebssystem und von *log4cpp* können alle weiteren Komponenten über das Software-Center bzw. über die Konsole mit dem Tool `apt-get install <package>` installiert werden.



9.7.1 Gestaltung der graphischen Oberfläche

Zur Anzeige einer graphischen Oberfläche wird das Toolkit GTK+ und zur graphischen Erstellung wird das Design-Tool *Glade* verwendet. Mit Hilfe von *Glade* können durch Verwendung von vordefinierten User-Interface-Elementen auf eine einfache Art und Weise eine graphische Oberfläche zusammengestellt werden.

Anforderungen an die GUI sind u.a.

- die Anzeige des Videobilds als Livestream und
- die Darstellung der Telemetriedaten vom Quadrokopter.

Ein erster Entwurf einer graphischen Oberfläche ist in Abbildung 9.47 dargestellt. Rot eingerahmt ist der Bereich für die Telemetriedaten. Dieser ist durch Register in folgende drei Bereiche gegliedert:

- *General*: Anzeige allgemeiner Informationen über die aktuellen Spannungen und Temperaturen auf dem Quadrokopter sowie Kommunikationsfehler auf den Microblazes.
- *QC Info*: Unter diesem Reiter werden die aktuellen Sensorwerte sowie die Motorstellwerte und Sollwerte von der Funksteuerung angezeigt.
- *Peripheral*: Unter diesem Reiter wird der aktuelle Modus des Gimbal-Controllers sowie die Schalterstellungen auf der Funksteuerung angezeigt.

Der Livestream der Kamera wird in dem blau eingerahmten Bereich angezeigt. Das Kameravideo wird durch GStreamer vom Quadrokopter über eine Pipeline an die Telemetriestation gesendet. Diese Pipeline ist an die IP-Adresse und einen Port gebunden. Innerhalb der GUI wird das empfangene Video in dem GUI-Element *drawing-area* angezeigt. Dazu wird dessen *window-handler* an die GStreamer-Pipeline durch das *XOverlay-Interface* gebunden. Detaillierte Informationen können der Quelle [Art] entnommen werden.

Der grüne Rahmen beinhaltet ein Logging-Fenster, welches neben den Log-Ausgaben der Telemetriestation auch die Log-Ausgaben des Quadrokopters anzeigen soll. Damit letztere Ausgaben überhaupt bei der Telemetriestation ankommen, wird folgendes Konzept verwendet:

- Die Software-Komponenten des AEVV-Systems verwenden `log4cpp`, um die Log-Nachrichten auszugeben.
- `Log4cpp` wird so konfiguriert, dass diese Nachrichten im *syslog* vom Linux-Betriebssystem, welches auf den ARM-Kernen läuft, gespeichert.
- Das *syslog* sendet alle eingehenden Nachrichten an die IP-Adresse der Telemetriestation. Diese Einstellung ist in der Datei `/etc/rsyslog.conf` vorgenommen worden.
- Auf der Telemetriestation muss das *syslog* ebenfalls angepasst werden. Dazu wird die Datei `/etc/rsyslog.d/50-default.conf` um folgende Zeile erweitert:

```
local0.*                -/var/log/pgaa.log
```

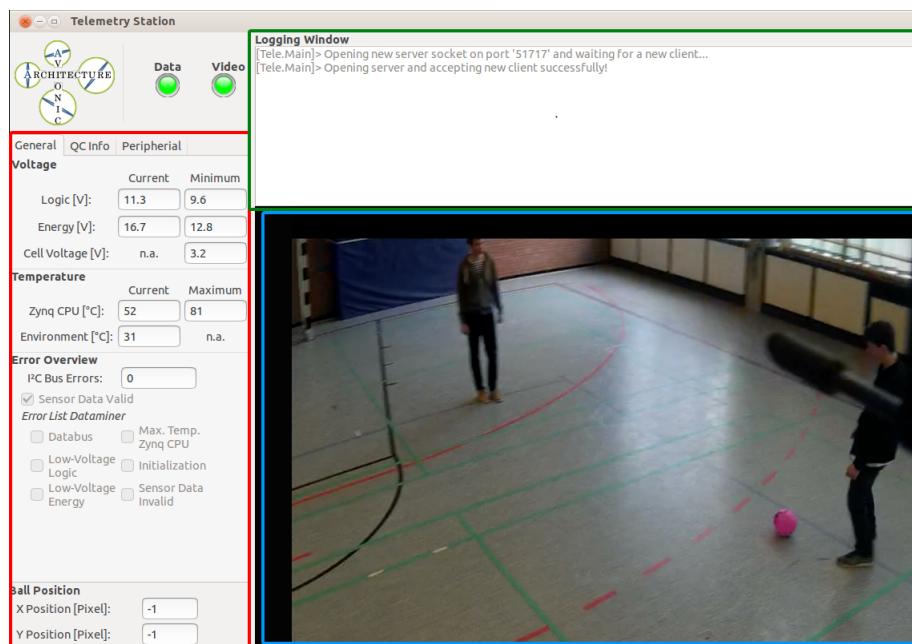


Abbildung 9.47: Graphische Oberfläche (GUI) für die Telemetriestation (Ausschnitt)

Die Software für die Telemetriestation wird soweit erweitert, dass ein paralleler Thread gestartet wird, welcher permanent die Datei `/var/log/pgaa.log` ausliest und dem Logging-Fenster hinzufügt. Innerhalb dieses Projekts konnte aus zeitlichen Engpässen das Konzept nur partiell umgesetzt werden.

Des Weiteren existieren in der GUI zwei Elemente in Form einer LED, um anzuzeigen, ob eine Socket-Verbindung bzw. der Videostream erfolgreich läuft. Für die LED des Videostreams ist ein Erkennungsmuster für ein erneutes Deaktivieren des Videostreams noch nicht implementiert. Als Ausblick könnte man für diesen Fall beim Schließen der Pipeline ein *End-of-Stream*-Signal über GStreamer verschicken.

Im folgenden Abschnitt soll detaillierter auf das Erzeugen eines virtuellen Access-Point in Software eingegangen werden.

9.7.2 Erzeugung eines virtuellen Access-Points

Zur Realisierung eines virtuellen Access-Points wird das Tool *hostapd* verwendet. Mit Hilfe dieser Software lässt sich in wenigen Schritten ein eigener, virtueller Access-Point in Software erzeugen. Bevor dieser Access-Point angelegt wird, stellt sich die Frage, welches Frequenzband ausgewählt werden. Hostapd bietet für eine WLAN-Kommunikation die Möglichkeit, zwischen den 802.11 a/b/g/n/ac Technologien bzw. zwischen dem 5 GHz oder dem 2.4 GHz Frequenzband zu unterscheiden. Um einen störungsfreien Betrieb zwischen der Fernsteuerung und dem Quadroptor im 2.4 GHz-Band gewährleisten zu können,



wird die WLAN-Kommunikation zwischen dem Quadrocopter und der Telemetriestation im 5 GHz-Band mit der 802.11 n-Technologie ausgewählt.

Sowohl die Telemetriestation als auch der Quadrocopter wird jeweils mit einem der beiden WLAN-Modulen ausgestattet.

Um den Anforderungen gerecht zu werden (s. Kapitel 7.3.5), werden zwei WLAN-Module der Firma *CSL* mit einem *Ralink*-Chipsatz verwendet. Der Hardwaretreiber ist in dem Paket `firmware-ralink` enthalten. Der Vorteil von einer Ubuntu-Distribution ist der bereits vorinstallierte Linuxtreiber `firmware-ralink`.

Damit ein eigener Access-Point eingerichtet werden kann, muss zuerst eine Konfigurationsdatei für `hostapd` erstellt werden. In dieser Datei können u.a. Parameter für das WLAN-Interface, für den Service Set Identifier (SSID), für den WLAN-Schlüssel, für den Funkkanal oder für die Verschlüsselungsart festgelegt werden. Ein Auszug aus solch einer Konfigurationsdatei ist in Listing 2 abgebildet.

```
interface=wlan1
2 driver=nl80211
  ssid=PGAA-AP
4 country_code=DE
  ieee80211d=1
6 ieee80211n=1
  hw_mode=a
8 channel=40
  ht_capab=[HT40-][SHORT-GI-20][SHORT-GI-40][TX-STBC][RX-STBC12]
10 require_ht=1
  wmm_enabled=1
12 macaddr_acl=0
  auth_algs=1
14 ignore_broadcast_ssid=0
  wpa=3
16 wpa_passphrase=PGAA2015
  wpa_key_mgmt=WPA-PSK
18 wpa_pairwise=TKIP
  rsn_pairwise=CCMP
```

Listing 2: Auszug aus der Konfigurationsdatei für `hostapd`

Diese Konfigurationsdatei kann unter `/etc/hostapd/hostapd.conf` abgelegt werden. Der Access-Point kann durch folgenden Konsolenbefehl gestartet werden:

```
sudo hostapd /etc/hostapd/hostapd.conf
```

Wichtig vor der Ausführung ist, dass der Dienst `hostapd` deaktiviert ist. Dies kann über folgenden Konsolenbefehl durchgeführt werden:

```
sudo service hostapd stop
```

Neben `hostapd` kann ein weiteres Software-Paket verwendet werden. Es heißt `dnsmasq`, welches einen DHCP- und DNS-Server aufsetzen kann. Dieser DHCP-Server weist Stationen,



die einen Verbindungsaufbau anfordern, eine Adresse in einem konfigurierten Adressbereich zu. Diese Konfiguration erfolgt in der Datei `/etc/dnsmasq.conf`. In dieser Konfigurationsdatei wird neben dem IP-Adressbereich auch der Gültigkeitszeitraum für eine vergebene Adresse festgelegt. Diese Software ist aber nicht zwingend notwendig, da auch eine statische IP-Adresse beim Client vergeben werden kann. Diese muss in gleichen Subnetz der Telemetriestation liegen. Das Subnetz der Telemetriestation kann in der Datei `/etc/network/interfaces` für das jeweilige WLAN-Interface (z.B. wlan1) festgelegt werden.



10 Validierung und Verifikation

Dieses Kapitel befasst sich mit der Validierung und der Verifikation des Quadropters. Da der Quadropter ein sicherheitskritisches System ist und ein fehlerhaftes Verhalten zu katastrophalen Folgen führen kann, ist das Beheben von Fehlern und das Erfüllen aller Anforderungen von immenser Bedeutung. In diesem Kapitel werden zunächst einige Voruntersuchungen bezüglich der Validierung und der Verifikation beschrieben. Anschließend wird auf die unterschiedlichen Testvorgehen vom ASG und vom AEVV-System eingegangen.

10.1 Testgrundlagen

Im Folgendem Abschnitt werden einige Testgrundlagen zur Validierung und zur Verifikation dargestellt. Nach einer kurzen Einleitung zum Thema Validierung und Verifikation werden einige statische und dynamische Testmethoden vorgestellt. Daraufhin werden die verschiedenen Teststufen des V-Modells genauer beschrieben und es wird auf das modellgetriebene Testen eingegangen. Abschließend werden die Themen Reglertest und Ermittlung der Ausführungszeiten präsentiert.

Für den weiteren Verlauf werden im Folgendem die Definitionen für Validierung, Verifikation und Testen eingeführt.

Definition Validierung: „Validierung ist der Prozess zur Beurteilung eines Systems oder einer Komponente mit dem Ziel festzustellen, ob der Einsatzzweck oder die Benutzererwartungen erfüllt werden. Funktionsvalidierung ist demnach die Prüfung, ob die Spezifikation die Benutzeranforderungen erfüllt, ob überhaupt die Benutzerakzeptanz durch eine Funktion erreicht wird.“ [JS13]

Definition Verifikation: „Verifikation ist der Prozess zur Beurteilung eines Systems oder einer Komponente mit dem Ziel festzustellen, ob die Resultate einer eingegebenen Entwicklungsphase den Vorgaben für diese Phase entsprechen. Software-Verifikation ist demnach die Prüfung, ob eine Implementierung der für den betreffenden Entwicklungsschritt vorgegebenen Spezifikation genügt.“ [JS13]

Definition Testen: „Unter Testen versteht man den Prozeß des Planens, der Vorbereitung und der Messung, mit dem Ziel, die Eigenschaften eines IT-Systems festzustellen und den Unterschied zwischen dem tatsächlichen und dem erforderlichen Zustand aufzuzeigen.“ [MP02]

Mit Testen soll insgesamt die Qualität eines Systems gesteigert werden, indem Fehler gefunden und behoben werden.

10.1.1 Statischer Test

Im Gegensatz zum dynamischen Test 10.1.2 wird das Testobjekt nicht mit Testdaten ausgeführt, sondern einer Analyse unterzogen. [SL05] Diese Analyse kann durch Personen



erfolgen oder aber mittels Unterstützung durch Werkzeuge. Bei dieser Analyse können alle Dokumente betrachtet werden, welche zur Erstellung der Software relevant sind. Ziel dieses Prozesses ist, Abweichungen von der vorhandenen Spezifikation oder Fehler in der Projektplanung frühzeitig zu finden, um den Entwicklungsprozess optimieren zu können. Statische Tests dienen somit der Prävention, um zu vermeiden, dass frühe Fehler die weitere Entwicklung negativ beeinflussen.

10.1.1.1 Reviews Reviews sind eine übliche Vorgehensweise zur Betrachtung und Prüfung von Dokumenten. Diese werden auf Grundlage der Dokumente durchgeführt, welche im Rahmen der Entwicklung von Software entstehen. Sie werden durch Kollegen durchgeführt und sollten nach Möglichkeit zeitnah nach Erstellung der Dokumente stattfinden. Ein geeigneter Zeitpunkt ist beispielsweise nach Abschluss einer der Phasen des V-Modells. Durch Reviews ergeben sich einige Vorteile:

- Fehler können durch ihre frühe Entdeckung kostengünstig beseitigt werden.
- Entwicklungszeiträume können verkürzt werden.
- Ungenauigkeiten in der Umsetzung der Kundenwünsche können frühzeitig erkannt werden.
- Da die Überprüfung im Team stattfindet, findet eine Kommunikation zwischen den Entwicklerteams statt.
- Durch die Arbeit im Team wird es notwendig, Sachverhalte so zu beschreiben, dass auch Außenstehende diese verstehen können.

Im Folgenden wird kurz auf die grundlegende Vorgehensweise eingegangen, mit welcher Reviews durchgeführt werden.

Planung In der Planung ist vom Manager des Reviews zu entscheiden, welche Dokumente einem Review unterzogen werden sollen. Hierbei ist vorher sicherzustellen, dass die ausgewählten Dokumente in ihrer entgeltigen Version vorliegen. Zudem ist für jedes Dokument abzuschätzen, welchen Zeitaufwand das Review in Anspruch nehmen wird und welche Personen geeignet sind, um dieses durchzuführen. Hierzu sind die Sichtweisen der jeweiligen Personen zu berücksichtigen, um die gewünschten Effekte beim Review zu erzielen. Beispielsweise können hier spezielle Teile des Dokumentes ausgewählt werden, welche spezifische Risiken für das Projekt beinhalten, oder aber das gesamte Dokument betrachtet werden.

Einführung In der Einführung oder Vorbesprechung werden die am Review beteiligten Personen mit den benötigten Informationen versorgt, die zur Durchführung des Reviews



nötig sind. Hierzu gehören zu den zu untersuchenden Dokumenten beispielsweise das Pflichtenheft, aber auch einzuhaltende Standards oder Richtlinien.

Reviewsitzung Die Reviewsitzung wird von einem Sitzungsleiter durchgeführt und ist auf einen festen Zeitrahmen beschränkt. Innerhalb dieser Sitzung darf jeder Teilnehmer frei seine Meinung äußern, sollte aber dabei stets klar machen, dass das Dokument und nicht der Autor einer kritischen Betrachtung unterzogen wird. Ziel der Sitzung ist es, das Dokument in Bezug auf Einhaltung von Vorgaben, Richtlinien und Standards sowie auf das Vorhandensein von Fehlern und Unstimmigkeiten zu prüfen. Das Ergebnis der Reviewsitzung sollte die Entscheidung sein, ob das Dokument unverändert akzeptiert, verbessert oder neu angefertigt werden muss.

Nachbereitung In der Nachbereitung wird der Autor in der Regel die gefundenen Mängel beseitigen. Es ist aber auch möglich, dass der Manager auf eigene Verantwortung entscheidet, das Dokument unverändert zu lassen.

Wiedervorlage Nach erfolgter Nachbereitung wird das überarbeitete Dokument erneut dem Manager vorgelegt, welcher dieses kontrolliert. Sollte das Ergebnis des ersten Reviews nicht akzeptabel gewesen sein, wird ein zweites Review durchgeführt.

10.1.1.2 Statische Analyse Im Gegensatz zum Review wird eine statische Analyse durch Werkzeuge durchgeführt. Ziel ist es hierbei, wie beim Review Fehler und Unstimmigkeiten in den Dokumenten zu finden. Im Gegensatz zur dynamischen Analyse wird auch bei diesem Verfahren keine Ausführung der zu analysierenden Dokumente vorgenommen. Damit eine automatisierte Auswertung durch Werkzeuge erfolgen kann, müssen die Dokumente in einem geeigneten Format vorliegen. Dies sind beispielsweise die in der Modellierung mittels UML vorkommenden Klassendiagramme zur Modellierung einer Softwarearchitektur. [CR07] Desweiteren lassen sich generierte Ausgaben in *HMTL* oder *XML* ebenfalls einer automatisierten Analyse unterziehen. Statische Analysewerkzeuge werden in der Regel vom Entwickler selbst eingesetzt. Dies kann vor oder während der Komponenten- und Integrationstests erfolgen und dient der Überprüfung, ob alle Richtlinien und Konventionen eingehalten wurden. Da diese Analyse mittels Werkzeugen und somit automatisch erfolgt, ist diese deutlich weniger zeitintensiv als Reviews.

Eines dieser Analysewerkzeuge ist beispielsweise der Compiler. Dieser führt eine Analyse des Quellcodes durch und überprüft die Syntax des Programmes. Darüber hinaus bieten die meisten Compiler zusätzliche Informationen wie beispielsweise:

- Verletzung der Syntax
- Konventions- und Standardsabweichungen



- Identifikation von nicht erreichbarem Code
- Prüfung auf typgerechte Wandlung von Datentypen
- Ermittlung von nicht deklarierten Variablen
- Über- und Unterschreiten von Feldgrenzen
- Anomalien im Kontrollfluss 10.1.1.2
- Anomalien im Datenfluss 10.1.1.2

Datenflussanalyse Eine Datenflussanalyse analysiert den Datenfluss zwischen Teilen eines Programmes. Hierbei werden Abhängigkeiten zwischen den Programmteilen identifiziert. Ziel ist beispielsweise die Identifikation von Programmfragmenten, welche nicht erreichbar sind. Hierdurch kann die Ausführungszeit von Programmen beschleunigt werden.

Das Grundprinzip einer Datenflussanalyse beruht auf einer Analyse des Kontrollflussgraphen. 10.1.1.2 Dessen Knoten bilden sogenannte Blöcke, in denen eine oder mehrere Anweisungen enthalten sind. Die Datenflussanalyse untersucht nun, welche Änderungen an den verwendeten Variablen vorgenommen werden. Zudem wird untersucht, wie sich die Daten zwischen den Blöcken ändern.

Kontrollflussanalyse Die Kontrollflussanalyse wird aufgrund des Kontrollflussgraphen vorgenommen. In Abbildung 10.1 ist ein beispielhafter Kontrollflussgraph dargestellt.

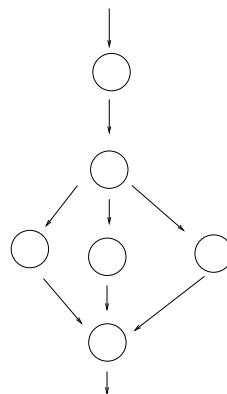


Abbildung 10.1: Kontrollflussgraph

Die einzelnen Knoten eines Kontrollflussgraphen stellen Anweisungsblöcke dar, in denen eine oder mehrere Anweisungen ausgeführt werden. Änderungen im Programmablauf werden durch bedingte Anweisungen herbeigeführt. So können beispielsweise auch Blöcke mehrmals durch Schleifen ausgeführt werden. Durch die Anschaulichkeit des Kontrollflussgraphen ist es möglich, komplexe Abläufe leichter zu erfassen und Anomalien festzustellen.



Die Analyse des Kontrollflussgraphen kann durch Personen aber auch Werkzeug gestützt erfolgen. Voraussetzung für eine erfolgreiche Analyse ist, dass der Kontrollflussgraph nicht von Hand erstellt wird, sondern durch Werkzeugunterstützung. Nur so kann gewährleistet werden, dass es sich tatsächlich um eine Eins-zu-eins Abbildung von Quellcode und Kontrollflussgraph handelt.

10.1.2 Dynamischer Test

Im Gegensatz zum statischen Test wird beim dynamischen Test das Testobjekt tatsächlich zur Ausführung gebracht. Hierzu muss ein lauffähiger Code vorliegen, welcher mit Eingangsdaten versehen wird. In den unteren Teststufen der Komponenten- und Integrationstests ist es oft notwendig, den Code in eine Testumgebung einzubetten, um diesen ausführen zu können. Meist muss zu diesem Zweck das aufrufende Programm simuliert werden, um die zu testende Komponente mit Eingangsdaten zu versorgen. Man spricht hierbei von einem Testtreiber.

Zu einem Testfall gehören immer:

- Vorbedingungen und Voraussetzungen zur Durchführung des Tests.
- Spezifikation des einzelnen Testfalls.
- Erwartetes Ergebnis.

Es macht wenig Sinn, einzelne Tests getrennt voneinander auszuführen. Diese werden nacheinander folgend ausgeführt. Hierzu kommen Testscripte zum Einsatz, mit welchen die erwarteten Ausgabewerte mit den tatsächlichen Werten verglichen werden. Entspricht der erwartete Wert dem tatsächlichen Ausgabewert, gilt der Testfall als bestanden. Es gibt zu diesem Zweck eine Vielzahl von unterschiedlichen Frameworks, welche dies ermöglichen. Eines davon ist das Google C++ Testframework. [Goo]

Zur Erstellung der Testfälle gibt es eine Reihe von Verfahren. Zwei dieser Verfahren, auf die nun eingegangen werden soll, sind Black- und Whiteboxtests.

10.1.2.1 Blackbox Tests Bei den Blackbox Verfahren ist der innere Aufbau des zu testenden Objektes nicht bekannt. Grundlage für diese Tests bilden die Spezifikation und die Anforderung des zu testenden Systems. Die Testfälle werden hierbei aus der Spezifikation abgeleitet. Ein vollständiger Test wäre, wenn man das zu testende Objekt mit allen möglichen Eingabedaten testet. Dies ist in der Praxis unmöglich, daher bildet man Äquivalenzklassen, um alle Szenarien der Eingabedaten abzudecken. 10.1.2.1

Befinden sich Fehler in den Anforderungen oder der Spezifikation des Systems, können Fehler im Testobjekt mit Blackboxtests nicht gefunden werden. Das Testobjekt verhält sich in diesem Fall wie vorgesehen, ist aber aufgrund der fehlerhaften Anforderungen selbst fehlerbehaftet. Um dem vorzubeugen, bieten sich vorherige Reviews an.(siehe Kapitel 10.1.1.1)



Zudem kann mit Blackboxtests ebenfalls nicht festgestellt werden, ob das Testobjekt zusätzliche Funktionalität bereitstellt, welche nicht durch die Spezifikation vorgesehen ist. Dies kann unter Umständen sicherheitsrelevante Fehler zur Folge haben. Andererseits kann durch den Blackboxtest sehr gut die Funktionalität eines Objektes getestet werden, weshalb dieser durchaus seine Berechtigung neben dem Whiteboxtest (10.1.2.2) hat.

Äquivalenzklassen Die Eingabedaten werden in Äquivalenzklassen unterteilt. Eine Äquivalenzklasse sind alle Eingabedaten, bei denen sich das zu testende Objekt gleich verhält. Es reicht an dieser Stelle, einen Kandidaten einer Äquivalenzklasse als Eingangswert zu verwenden, da davon ausgegangen wird, dass sich das Testobjekt für alle Elemente der Äquivalenzklasse gleich verhält.

Es sind hierbei nicht nur die gültigen Eingabewerte in Äquivalenzklassen zu unterteilen, sondern auch für ungültige Eingabewerte sind Äquivalenzklassen zu finden. Das systematische Vorgehen zum Finden der Äquivalenzklassen sieht wie folgt aus:

- Definitionsbereich der Eingangswerte ermitteln. (z.B. Wertebereich von Variablen)
- Definitionsbereich in gültige und ungültige Werte unterteilen.
- Die ermittelte Äquivalenzklasse in weitere Äquivalenzklassen einteilen. Hierbei werden Wertebereiche zusammengefasst, welche vom Testobjekt auf identische Weise verarbeitet werden.
- Für jede Äquivalenzklasse ist anschließend ein Repräsentant zu wählen. Zu diesem Repräsentanten wird dann ein Testfall erstellt.

Ebenfalls aussichtsreiche Kandidaten für einen Repräsentanten sind sicherlich Werte auf den Grenzen der Äquivalenzklassen. Auf die sogenannte Grenzwertanalyse wird in 10.1.2.1 noch einmal genauer eingegangen.

Grenzwertanalyse Zusätzlich zu den Testfällen, welche aus den Repräsentanten der Äquivalenzklassen entstehen, kann eine Grenzwertanalyse zusätzlich Testfälle liefern. Oft treten Fehlerzustände an den Grenzen der Wertebereiche von Variablen auf, da dort zusätzliche Fallunterscheidungen im Programmcode vorgenommen werden. Es ist daher sinnvoll, die Ränder der Wertebereiche einer gesonderten Analyse zu unterziehen. Für jeden Randwert wird sowohl der Randwert selbst, als auch die beiden benachbarten Werte, welche sich innerhalb und außerhalb der Äquivalenzklasse befinden, getestet. Für Fließkommazahlen muss hierbei zunächst eine hinreichende Genauigkeit gewählt werden. Das In- und Dekrement ist für die ausgesuchten Werte hinreichend klein zu wählen.



10.1.2.2 Whitebox Tests Bei Whiteboxtests liegt der Quellcode des zu untersuchenden Objektes offen. Ziel ist es, jede Zeile des Codes einmal zur Ausführung zu bringen, um eventuelle Fehler aufzudecken. Hierbei ist es unter Umständen nötig, den Code selbst manipulieren zu können. Hierbei soll eine zuvor festgelegte Abdeckung der Anweisungen im Code erfüllt werden. Hierzu stehen unterschiedliche Vorgehensweisen zur Verfügung:

- Anweisungsüberdeckung
- Zweigüberdeckung
- Pfadüberdeckung

Anweisungsüberdeckung Bei der Anweisungsüberdeckung soll eine zuvor festgelegte prozentuale Anzahl von Anweisungen zur Ausführung kommen. Hierzu müssen geeignete Testfälle identifiziert werden. In einem ersten Schritt wird von dem vorliegenden Quellcode ein Kontrollflussgraph (10.1.1.2) erstellt. Blöcke von sequentiell aufeinander folgenden Anweisungen werden hierbei zusammengefasst. Nach Ausführung der Testfälle ist nachzuweisen, welche Anweisungen ausgeführt wurden. Wurde die zuvor spezifizierte prozentuale Anzahl von Anweisungen erreicht, gilt der Testfall als bestanden. Hierzu bietet z.B. das Google Testframework bereits eine automatisierte Auswertung. Es ist hierbei auswertbar, wieviel Prozent der Anweisungen in den getesteten Klassen durch die durchlaufenden Testfälle abgedeckt wurden.

Zweigüberdeckung Die Zweigüberdeckung stellt die Analyse von bedingten Anweisungen in der Vordergrund. Hierbei soll eine komplette Abdeckung der durch bedingte Anweisungen hervorgerufenen Zweige erreicht werden. Anders als bei der Anweisungsüberdeckung spielt es hierbei keine Rollen, ob beispielsweise ein else-Zweig keine Anweisungen enthält. Bei der Zweigüberdeckung sind auch solche Pfade zu durchlaufen, da auch hierbei Fehler entstehen können. Die Zweigüberdeckung verlangt, dass bei bedingten Anweisungen beide bzw. bei Switch-Case-Anweisungen alle Pfade durchlaufen werden. Dies gilt ebenfalls für die Schleifenkörper und Rücksprunganweisungen von Schleifen, insbesondere ist hierbei auch die komplette Umgehung einer Schleife zu testen.

Pfadüberdeckung Die Pfadüberdeckung kommt zum Einsatz, wenn das Testobjekt komplex ist und insbesondere viele Schleifen enthält. Hierbei reicht eine Anweisungs- oder Pfadüberdeckung als angemessener Test nicht aus. Pfade bezeichnen die Abfolge von einzelnen Zweigen innerhalb des zu untersuchenden Programmes. Er beschreibt somit die Abfolge von einzelnen Programmteilen. Im Gegensatz dazu betrachtet eine Zweiganalyse nur die einzelnen Anweisungsfolgen innerhalb des Programmes. Bei der Pfadanalyse werden somit die Abhängigkeiten zwischen den einzelnen Zweigen betrachtet. Es ist klar, dass hierbei keine absolute Abdeckung aller Pfade getestet werden kann. Allerdings wird

das zu untersuchende Objekt mittels der Pfadüberdeckung wesentlich genauer getestet, da 100 % Anweisungüberdeckung nicht garantiert, dass alle Kanten(Zweige) des Kontrollflussgraphen ausgeführt wurden. Ebenso garantieren 100 % Zweigabdeckung nicht, dass es nicht noch weitere Pfade innerhalb des Testobjektes gibt, welche noch nicht durchlaufen wurden.

10.1.3 Teststufen des V-Modells

Wie bereits in Kapitel 2.2.2 beschrieben, orientiert sich die Projektgruppe beim Entwickeln und Testen des Quadropters an das V-Modell (Abb. 10.2).

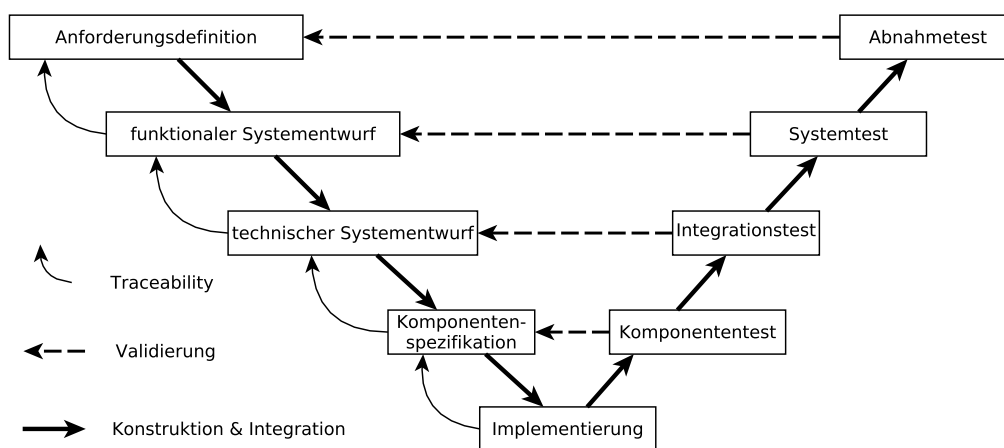


Abbildung 10.2: V-Modell

Beim V-Modell wird das Entwicklungsvorgehen auf dem linken Ast und das Testvorgehen auf dem rechten Ast des „V’s“ dargestellt. Das Testvorgehen gliedert sich dabei in vier Teststufen. Auf der untersten Teststufe befindet sich der Komponententest, auf den der Integrationstest, der Systemtest und der Abnahmetest aufbauen [SL05].

10.1.3.1 Komponententest Beim Komponententest, auch Unitest genannt, wird jede Komponente/ Unit zunächst isoliert von anderen Komponenten getestet. Dadurch werden komponentenexterne Einflüsse ausgeschlossen, wodurch beim Entdecken eines Fehlers die Ursache direkt auf die getestete Komponente zurückgeführt werden kann. [SL05]

Beim Komponententest ist die innere Struktur der Komponente dem Tester häufig bekannt, wodurch der Whitebox-Test (Vgl. 10.1.2.2) zum Einsatz kommen kann. Jedoch besteht nicht immer die Möglichkeit den Whitebox-Test zu verwenden, da die Komponente z.B. nicht einsehbar oder viel zu komplex ist. In diesem Fall wird oft der Blackbox-Test (Vgl. 10.1.2 verwendet. Der Zufallstest hingegen ist nicht zuverlässig genug bei der Findung von Fehlern und wird daher nur selten verwendet. [SL05]

Generell ist das manuelle Durchführen und Dokumentieren von Testfällen sehr aufwen-



dig, besonders wenn die Testfälle mehrmals durchgeführt werden müssen. Um das Durchführen und Dokumentieren von Testfällen zu automatisieren, gibt es verschiedene Unit-Testframeworks wie z.B. das Google Testframework.

10.1.3.2 Integrationstest Die zweite Teststufe des V-Modells ist der Integrationstest. Hier wird vorausgesetzt, dass die einzelnen Komponenten, die zuvor beim Komponententest getestet wurden, einwandfrei funktionieren. Für den Integrationstest werden die einzelnen Komponenten zu größeren Teilsystemen integriert. Schwerpunktmäßig wird beim Integrationstest das Zusammenspiel der Komponenten und dessen Schnittstellen getestet. Fehler können unter anderem bei integrierten Komponenten auftreten, wenn z.B. dessen Schnittstellenformate nicht passen [SL05]. Beim Integrationstest können sowohl Blackbox-, als auch Whitebox-Tests zum Einsatz kommen. Wie bereits erwähnt, werden beim Integrationstest die einzelnen Komponenten zu größeren Teilsystemen integriert. Bei der Integration der Komponenten gibt es verschiedene Vorgehensstrategien.

Eine Strategie ist die Ad-hoc Strategie. Hier werden die Komponenten in der (zufälligen) Reihenfolge der Fertigstellung integriert. Sobald Komponenten erfolgreich mit Komponententest getestet wurden, können diese integriert und dem Integrationstest unterzogen werden. Mit dieser Strategie kann bereits sehr früh damit begonnen werden erste Integrationstest durchzuführen [SL05].

Eine andere Vorgehensstrategie bei der Integration der Komponenten ist die Top-down-Integration. Hier werden die Komponenten auf oberster Systemebene zuerst zusammengefügt. Blöcke aus unteren Systemebenen werden zunächst durch Platzhalter ersetzt. Nachdem die ersten Integrationstests abgeschlossen wurden, werden immer wieder weitere Komponenten aus den darunter liegenden Systemebenen zu den Komponenten aus der obersten Systemebenen hinzugefügt und erneut getestet. Dieser Vorgang wird solange wiederholt, bis alle Komponenten zusammengefügt wurden und die Integrationstests alle erfolgreich abgeschlossen wurden [SL05].

Neben der Top-down-Integration gibt es noch die Bottom-up-Integration. Hierbei werden zuerst die Komponenten auf der untersten Systemebenen zu einem Teilsystem zusammengefügt und getestet. Nachdem das Teilsystem erfolgreich getestet wurde, werden weitere Komponenten hinzugefügt und getestet, bis alle Komponenten zusammengefügt wurden. Das „big bang“ ist eine weitere Integrationsmöglichkeit. Es werden alle Komponenten auf einmal zusammengefügt und auf einmal getestet. Jedoch treten die Fehler beim „big bang“ alle auf einmal auf, sodass es zum einen schwieriger ist das System zum Laufen zu bringen und zum anderen ist das Finden der Fehlerursache deutlich aufwendiger. Daher sollte „big bang“ möglichst vermieden werden.

10.1.3.3 Systemtest Der Systemtest ist die dritte Teststufe beim V-Modell und wird direkt nach den Integrationstests durchgeführt. Hierbei wird das gesamte System getestet. Nach dem Abschluss der Systemtests muss sichergestellt sein, dass das getestete System die zuvor definierten Anforderungen erfüllt. Bei Systemtests werden überwiegend Blackbox-Tests verwendet, da die innere Struktur des Gesamtsystems nicht immer bekannt ist. [SL05]

10.1.3.4 Abnahmetest Der Abnahmetest ist die letzte Teststufe beim V-Modell und wird im Anschluss an die Systemtests durchgeführt. Das Durchführen der Testfälle findet nach der Übergabe des Systems an den Kunden statt. Der Testschwerpunkt hierbei liegt auf das Verhalten des System in der Umgebung, in der es betrieben werden soll. In der Praxis wird der Abnahmetest zusammen mit dem Kunden durchgeführt und ist Voraussetzung für die rechtskräftige Übernahme des Systems. [SL05]

10.1.4 Modellgetriebenes Testen

Bei der modellgetriebenen Entwicklung bietet sich das modellgetriebene Testen an. Das modellgetriebene Testen kann bereits in einer sehr frühen Entwicklungsphase stattfinden, da für das Testen zunächst nur das Modell oder Teile des Modells benötigt werden und noch nicht das gesamte System [ZN09]. Dadurch können Fehler früher gefunden und behoben werden. Der Ablauf des modellgetriebenen Testen ist in Abbildung 10.3 zu sehen.

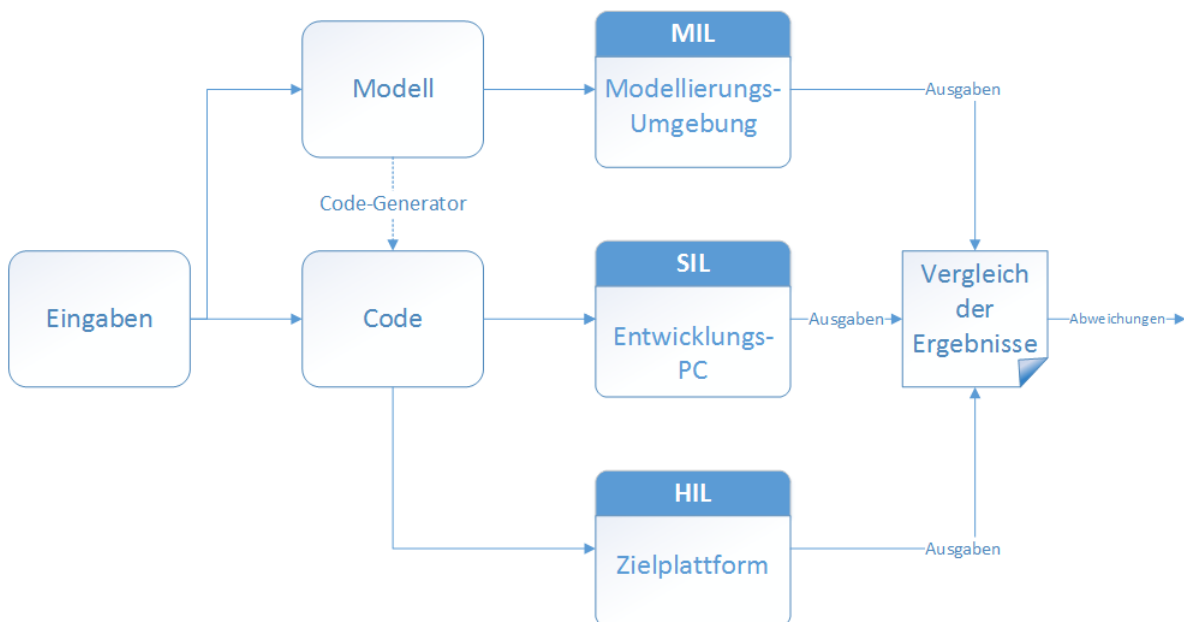


Abbildung 10.3: Übersicht zum modellgetriebenem Testen (nach [HS09])

Beim Model In The Loop Test (MIL-Test) werden Eingaben im Modell gemacht und das Modell wird in der Entwicklungsumgebung ausgeführt. Für den Software In The Loop

Test (SIL-Test) wird aus dem Modell Code generiert. Der generierte Code wird mit den gleichen Eingaben wie beim MIL-Test auf einem Entwicklungs-PC ausgeführt. Für den Hardware In The Loop Test (HIL-Test) wird der Code auf die Zielplattform gespielt und dort erneut mit den gleichen Eingaben ausgeführt. Anschließend werden die Ergebnisse vom MIL-, SIL- und HIL-Test miteinander verglichen. Die Abweichungen zwischen den Ausgaben sollten dabei sehr gering sein. [HS09]

10.1.4.1 Model In The Loop Test Beim Model In The Loop Test (MIL-Test) wird das entwickelte Modell (Controller Modell) vom System in der Testumgebung simuliert. Die Testumgebung ist meistens auch die Entwicklungsumgebung z.B. CAMEL-View oder Simulink. Das Modell kann zunächst in einer offenen Schleife (open-loop) ohne einem Physikalischen Modell simuliert werden. Das Testen in einer offenen Schleife wird häufig übersprungen und das Modell wird in einer geschlossenen Schleife (closed-loop) zusammen mit einem Physikalischen Modell simuliert (Abb. 10.4). Der Vorteil beim MIL-Test ist, dass eine sehr schnelle Entwicklung möglich ist, da bereits kleine Veränderungen am Modell sofort getestet werden können. [ZN09]

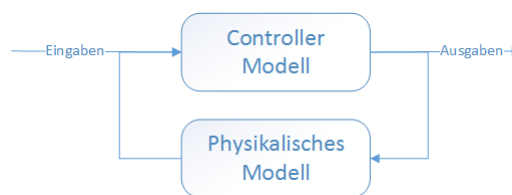


Abbildung 10.4: Ablauf beim Model In The Loop

10.1.4.2 Software In The Loop Test Nach dem MIL-Tests wird der Software In The Loop Test (SIL-Test) durchgeführt. Dabei wird sowohl das Controller Modell als auch das Physikalische Modell durch Code (meistens C- oder C++-Code) ersetzt. Der Code wird entweder aus den Modellen mittels einem Code-Generator generiert oder manuell geschrieben. Der Code kann wiederum in einer offenen Schleife (open-loop) ohne dem Code vom Physikalischen Modell oder in einer geschlossenen Schleife (closed-loop) mit dem Code vom Physikalischen Modell getestet werden. Beim SIL-Test werden die gleichen Testfälle wie beim MIL-Test durchgeführt. Anschließend werden die Testergebnisse vom SIL-Test mit denen vom MIL-Test verglichen. Mit den SIL-Tests soll sichergestellt werden, dass sich der Code identisch zum Modell verhält. [ZN09]

10.1.4.3 Hardware In The Loop Test Der Hardware In The Loop Test (HIL-Test) wird nach dem SIL-Test durchgeführt. Dafür wird der getestete Code auf die Zielplattform gebracht. Das Physikalische Modell wird dabei immernoch simuliert, wobei Echtzeitverhalten bei der Simulation des Physikalischen Modells vorausgesetzt wird [ZN09]. Die



Zielplattform interagiert mit dem Physikalischen Modell durch ihre digitalen und analogen Ein- und Ausgänge. Beim HIL-Test werden erneut die gleichen Testfälle wie bei den MIL- und SIL-Tests durchgeführt. Auch hier werden die Ergebnisse vom MIL-, SIL- und HIL-Test miteinander verglichen, um auch hier sicherzustellen, dass sich der Code auf der Zielplattform identisch zum Modell verhält. [HS09]

10.1.4.4 Integration des modellgetriebenen Testens in das V-Modell Das Model-In-The-Loop, das Software-In-The-Loop und das Hardware-In-The-Loop können auf jeder Teststufe des V-Modells durchgeführt werden.

Komponententest:

- **Model In The Loop:** Die Modellkomponenten werden in der Simulationsumgebung getestet.
- **Software In The Loop:** Aus den Modellkomponenten wird Code generiert. Der Code jeder Komponente wird einzeln auf einem Entwicklungs-PC, mit den gleichen Eingaben wie beim MIL-Test, getestet.
- **Hardware In The Loop:** Der Code von jeder Komponente wird einzeln auf die Zielplattform gespielt und auf dieser, mit den gleichen Eingaben wie beim MIL-Test, getestet.

Bei jedem Testdurchlauf werden die Eingaben und die Ausgaben in CSV-Dateien gespeichert. Nachdem die Tests durchgeführt wurden, werden die Ausgaben vom MIL-, SIL- und HIL-Test miteinander verglichen. Es wird erwartet, dass die Ausgaben identisch sind. Auf dieser Teststufe soll sichergestellt werden, dass sich der Code der verschiedenen Komponenten auf der Zielplattform identisch zu den Modellen der Komponenten verhält.

Integrationstest:

- **Model In The Loop:** Die einzelnen Komponenten werden zu größeren Teilmodellen zusammengefügt und anschließend in der Simulationsumgebung getestet.
- **Software In The Loop:** Aus den Teilmodellen wird Code generiert. Der Code von jedem Teilmodell wird einzeln auf einem Entwicklungs-PC, mit den gleichen Eingaben wie beim MIL-Test, getestet.
- **Hardware In The Loop:** Der Code von jedem Teilmodell wird einzeln auf die Zielplattform gespielt und auf dieser, mit den gleichen Eingaben wie beim MIL-Test, getestet.

Bei jedem Testdurchlauf werden die Eingaben und die Ausgaben erneut in CSV-Dateien gespeichert. Nachdem die Tests durchgeführt wurden, werden die Ausgaben vom MIL-, SIL- und HIL-Test miteinander verglichen. Es wird erwartet, dass die Ausgaben identisch



sind. Auf dieser Teststufe soll sichergestellt werden, dass sich der Code der Teilmodelle auf der Zielplattform identisch zu den Teilmodellen verhält.

Systemtest:

- **Model In The Loop:** Die einzelnen Teilmodelle werden zu einem Gesamtmodell zusammengefügt und anschließend in der Simulationsumgebung getestet.
- **Software In The Loop:** Aus dem Gesamtmodell wird Code generiert. Der Code vom Gesamtmodell wird auf einem Entwicklungs-PC, mit den gleichen Eingaben wie beim MIL-Test, getestet.
- **Hardware In The Loop:** Der Code vom Gesamtmodell wird auf die Zielplattform gebracht und auf dieser, mit den gleichen Eingaben wie beim MIL-Test, getestet.

Bei jedem Testdurchlauf werden die Eingaben und die Ausgaben erneut in CSV-Dateien gespeichert. Nachdem die Tests durchgeführt wurden, werden die Ausgaben vom MIL-, SIL- und HIL-Test miteinander verglichen. Es wird erwartet, dass die Ausgaben identisch sind. Auf dieser Teststufe soll sichergestellt werden, dass sich der Code vom Gesamtmodell auf der Zielplattform identisch zum Modell verhält.

10.1.5 Reglertests

Damit der Quadrocopter eine stabile Lage erreichen kann, werden Regler verwendet. Dafür müssen die Regelparameter so eingestellt werden, dass der Regelkreis stabil ist. Generell kann ein Regler mittels einem Einheitssprung angeregt werden. Die daraus resultierende Sprungantwort kann anschließend genauer analysiert werden. Die zu untersuchenden Kriterien, Stabilität, Zeitverhalten und Genauigkeit, werden im Folgenden genauer vorgestellt.

10.1.5.1 Analyse der Stabilität Bei der Analyse der Stabilität werden drei Fälle unterschieden: stabil, grenzstabil und instabil. Ein System ist stabil, wenn dieses in Ruhelage bleibt bzw. nach einer Anregung in die Ausgangsposition wieder zurückkehrt. Ein grenzstabiles System kehrt nach einer Anregung nicht mehr in die Ausgangsposition zurück. Der Ausgangswert ist jedoch begrenzt. Bei einem instabilem System steigt der Ausgangswert bei einer Anregung ins Unendliche. [Hei14]

Eine Methode zur Stabilitätsanalyse ist die Betrachtung der Polstellen [SZ11]. Zur Ermittlung der Polstellen wird die Nennerfunktion $N(s)$ der Übertragungsfunktion $G(s) = \frac{Z(s)}{N(s)}$ vom Regler benötigt. Die Polstellen sind wiederum die Nullstellen von $N(s)$. Die Polstellen können einen reellen und einen imaginären Anteil besitzen. Nachdem die Polstellen ermittelt wurden, können diese in der s -Halbebene dargestellt werden. Dabei ist ein System stabil, wenn dieses keine Polstellen mit einem positiven Realanteil besitzt und auch keine Polstelle auf der Imaginärachse liegt (Abb. 10.5). [Orl11]

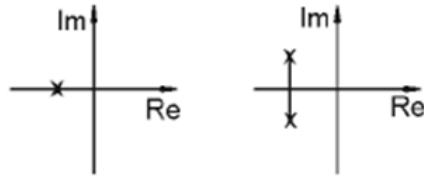


Abbildung 10.5: Polstellen eines stabilen Systems [Orl11]

Ein grenzstabiles System hat mindestens einen Pol auf der Imaginärachse, jedoch keinen Mehrfachpol auf der Imaginärachse. Zusätzlich darf sich kein Pol in der rechten s-Halbebene befinden (Abb. 10.6).

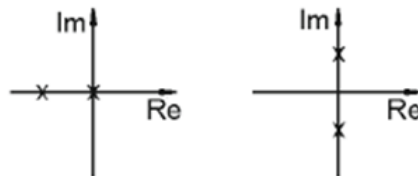


Abbildung 10.6: Polstellen eines grenzstabilen Systems [Orl11]

Ansonsten ist das System instabil (Abb. 10.7). [Orl11]

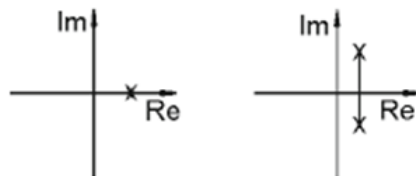


Abbildung 10.7: Polstellen eines instabilen Systems [Orl11]

10.1.5.2 Analyse des Zeitverhaltens Bei der Analyse des Zeitverhaltens werden meist zwei Zeiten betrachtet. Das ist zum einem die Anregelzeit und zum anderem die Ausregelzeit (Abb. 10.8). Die Anregelzeit ist ein Maß für die Schnelligkeit der Sprungantwort. Sie gibt die Zeit an, die das System nach einem Sprung braucht, bis es in einem definierten Toleranzband eintritt. Das Toleranzband ist meist definiert als ein Bereich von $\pm 2\%$ um den Sollwert. Die Ausregelzeit gibt die Zeit an, die das System benötigt, um dauerhaft im Toleranzbereich zu verbleiben. [SZ11]

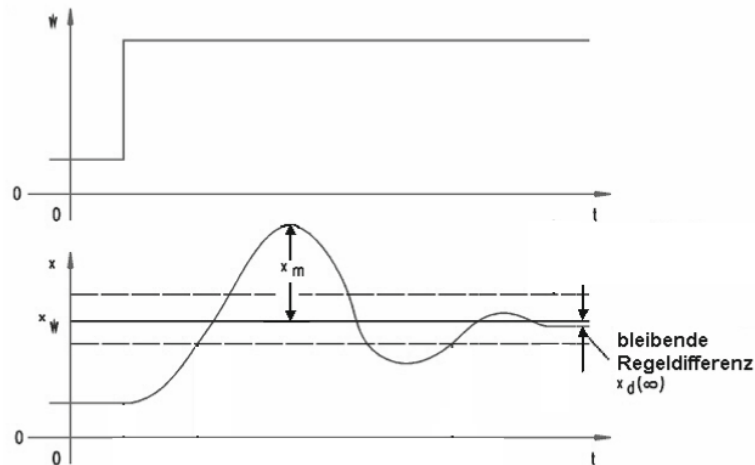


Abbildung 10.8: Analyse des Zeitverhaltens [SZ11]

10.1.5.3 Analyse der Genauigkeit Eine weitere wichtige Größe bei der Analyse von Reglern ist die Genauigkeit. Als stationäre Genauigkeit bezeichnet man das Verhalten, wenn der Regelfehler eines Systems nach einer Anregung wieder Null wird [Lun13]. Im Gegensatz dazu, wenn der Regelfehler nicht gegen Null strebt, betrachtet man die dauerhafte Regelabweichung. Die dauerhafte Regelabweichung beschreibt die dauerhafte Abweichung vom Sollwert (Vgl. 10.9). Eine weitere wichtige Größe ist die maximale Überschwingbreite. Sie gibt den maximalen Wert der Überschwingung an [SZ11]. Da ein stabiles System idealerweise nicht schwingt, sollte die maximale Überschwingung so gering wie möglich sein.

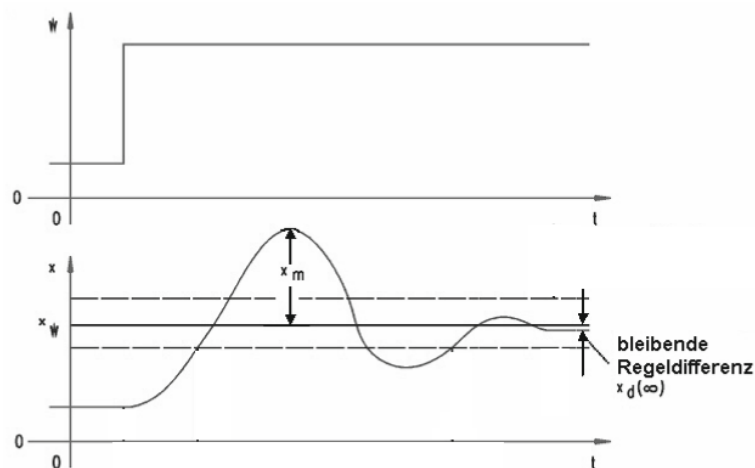


Abbildung 10.9: Analyse der Genauigkeit [SZ11]



10.1.6 Ermittlung der Ausführungszeiten

Das ASG regelt das Flugverhalten des Multikopters. Es handelt sich hierbei um ein sicherheitskritisches System, da ein Versagen unmittelbar zu hohen Kosten oder Verletzungen von Personen führt. Daher ist es wichtig, dass das System zu jeder Zeit die korrekte Reaktion auf von außen herangetragene Ereignisse aufweist. Hierbei ist es nicht nur wichtig, dass die Reaktion korrekt ist, sondern dass sie auch innerhalb des geforderten Zeitlimits (Deadline) erfolgt. Man spricht hier davon, dass das System harte Echtzeit erfüllen muss, da ein Überschreiten der Deadlines in keinem Fall zulässig ist. Die Zeiten, welche den Tasks für die nötigen Berechnungen zur Verfügung stehen, wurden bereits im Kapitel 6.8 definiert. Ein Überschreiten dieser WCETs ist beim ASG nicht zulässig.

Beim AEVV-System handelt es sich um ein weiches Echtzeitsystem, da ein Überschreiten von Deadlines lediglich Performance Einbrüche zur Folge hat, nicht aber zu katastrophalen Folgen führt. Bei der Objektverfolgung und dem zur Verfügungstellen des Videostreams ist es wichtig, dass diese im Mittel die geforderte Performance aufweisen, um ein flüssiges Bild sowie ausreichende Verfolgung des zu verfolgenden Objektes zu erreichen. Es ist hierbei wichtig, dass die geforderten WCETs im Mittel eingehalten werden.

Nun werden 2 Ansätze vorgestellt und verglichen, mit welchen validiert werden kann, ob die geforderten Ausführungszeiten beider Systeme tatsächlich eingehalten werden können. Mit diesen Ansätzen soll es möglich sein, zu validieren ob die Task die geforderten WCETs tatsächlich einhalten, wenn sie auf der Zielhardware(Target) laufen.

10.1.6.1 Statische Analyse der Ausführungszeiten Ziel der statischen Analyse von Ausführungszeiten ist es, aufgrund einer Analyse des Quellcodes und der Zielhardware, auf welcher der Quellcode ausgeführt wird, genaue Voraussagen machen zu können, welche oberen Schranken für die Ausführungszeiten zu erwarten sind. Hierzu spielt die Architektur der Zielhardware eine entscheidende Rolle.

Moderne Architekturen nutzen Pipelining und Caching. Dies hat zur Folge, dass die Ausführungszeit einer Instruktion maßgeblich von der vorherigen Instruktion abhängt. Hierdurch kann es zu sehr starken Schwankungen bei den Ausführungszeiten kommen. Ein Ausschalten dieser Mechanismen kann allerdings zu starken Performance-Einbrüchen führen. Dies hat zur Folge, dass die ermittelte obere Grenze für die Ausführungszeit deutlich über der in der Praxis auftretenden Grenze liegen kann. Daher ist es unerlässlich, diese Mechanismen bei einer statischen Ausführungszeiten-Analyse zu berücksichtigen. [Cor08] Zudem ist es nötig, bei einer statischen Analyse sämtliche Wertebereiche und Eingangskombinationen des zu analysierenden Tasks zu berücksichtigen, um sicherzustellen, den kritischen Pfad des zu analysierenden Programmes abzudecken, welcher für die WCET relevant ist.

Ein Werkzeug, welches in der Lage ist, diese Schwierigkeiten abhängig vom Target, auf dem der Code ausgeführt wird, zu berücksichtigen, ist der aiT-WCET Analyzer (aiT) der



Firma AbsInt. AbsInt hat mit ihrem Werkzeug aiT einen Worst-Case-Analyser auf den Markt gebracht, welcher in der Industrie große Akzeptanz gefunden hat. Zu den Kunden von AbsInt gehören laut Hersteller Angaben auf der firmen-eigenen Homepage namhafte Unternehmen wie z.B. Airbus Industries, Bosch, BMW, Nokia und IBM. [Abs]

Der WCET-Analyser aiT analysiert auf Grundlage eines ausführbaren Binärys des zu analysierenden Programmes und genauer Kenntnis der Architektur, auf welcher das Programm ausgeführt werden soll, statisch das Cache- und Pipelining-Verhalten des auszuführenden Tasks und ermittelt anhand der Wertebereiche der Eingangs- und Ausgangsvariablen die obere und untere Schranke für die Laufzeit des Tasks.

Als Eingangsdaten für die Berechnung der WCETs benötigt aiT lediglich ein Binary des Quellcodes, den Einstiegspunkt des Programmes sowie eine *AIS*-Konfigurationsdatei, welche Informationen über das Cache- und Pipelining-Verhalten des Targets enthält. Da aiT mit dem Binary arbeitet, ist es unabhängig vom Compiler und kann nahtlos in den bestehenden Entwicklungsprozess integriert werden. Die statische Analyse der Ausführungszeiten läuft in den folgenden Schritten ab, welche in Abbildung 10.10 dargestellt sind. [Cor08]

- **Rekonstruktion des Kontrollflusses** Im ersten Schritt wird das zu analysierende Programm vom CFG-Builder eingelesen und daraus der Kontrollfluss des Programms rekonstruiert. Hierbei werden die im Programm enthaltenen Instruktionen analysiert. Hierzu ist es nötig, dass genaue Informationen über den Instruktionssatz des zu verwendenden Targets bekannt sind. Andernfalls ist es nicht möglich, auch plattformabhängige Befehle interpretieren zu können. Ergebnis dieser Analyse ist eine CRL2-Datei.
- **Wertebereichsanalyse** Nun werden für alle Zugriffe die möglichen Werte der Register untersucht, die im Laufe des Programmes auftreten können.
- **Schleifenanalyse** Hier wird nun für alle im Programm vorkommenden Schleifen eine obere Grenze ermittelt. Dies erfolgt für alle Schleifen, welche nicht bereits vom Nutzer in der *AIS* Datei angegeben worden sind. Dies funktioniert aber laut Angaben des Herstellers nur für sehr einfache Schleifenkonstrukte. Daher ist es bei komplexeren Schleifenkonstrukten wichtig, die oberen Grenzen bereits in der *AIS* Datei anzugeben.
- **Cache-Analyse** In diesem Schritt wird für jeden Speicherzugriff klassifiziert, um welche Art von Speicherzugriff es sich handelt. Man unterscheidet hier zwischen Cache-/ und Speicherzugriffen. Die Cachezugriffe werden hierbei unter anderem in *always hit*, *always miss* etc klassifiziert. Das Werkzeug aiT ist hierbei in der Lage, für bestimmte Architekturen die durch einen Cache-Miss hervorgerufene Verzögerung zu berechnen.

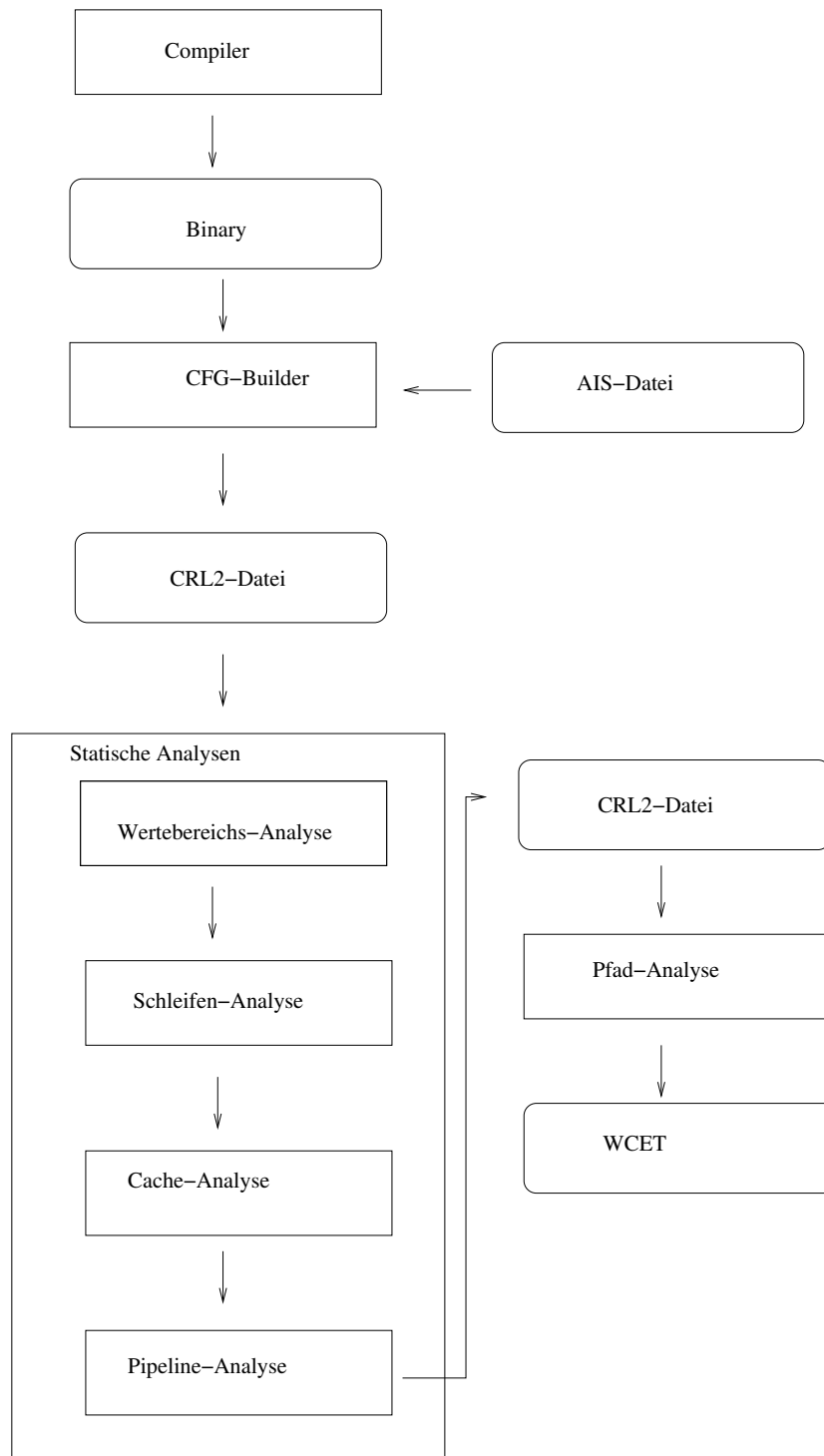


Abbildung 10.10: Aufbau des WCET-Analyzers ait der Firma AbsInt



- **Pipelining-Analyse** Hier wird das Verhalten der Pipeline-Architektur formal berechnet. Für jede Instruktion wird anhand aller möglichen Szenarien die Ausführungszeit berechnet. Dies beinhaltet sowohl den Zustand der Pipeline als auch den Cache-Inhalt. Beides hat einen Einfluss auf die WCET.

Ergebnis dieser Berechnungen ist eine weitere CRL2-Datei. Diese enthält für jeden Basisblock abhängig vom jeweiligen Cache- und Pipelinezustand die jeweilige Berechnungszeit.

- **Pfadanalyse** Die Pfadanalyse stellt den letzten Schritt der Ausführungszeitenanalyse dar. Diese analysiert, basierend auf der produzierten CRL2-Datei, alle möglichen Pfade des Programmablaufes und liefert als Ergebnis eine verlässliche WCET des analysierten Programms.

10.1.6.2 Dynamische Analyse Im Gegensatz zur statischen Analyse der Ausführungszeiten wird bei einer dynamischen Analyse versucht, durch Simulation oder Ausführung des zu analysierenden Programmes auf der Zielhardware eine obere Grenze für die Ausführungszeit zu ermitteln. [IL07]

Bei diesem Vorgehen wird eine Instrumentierung des Codes vorgenommen, um die Ausführungszeit während der Ausführung auf der Zielhardware zu messen. Hierbei hängt die Ausführungszeit von der geeigneten Wahl der Eingabeparameter ab. Pro Simulationsdurchlauf wird allerdings nur eine Kombination von Eingabewerten getestet. Daher ist es bei diesem Ansatz nötig, möglichst viele und geeignete Messreihen durchzuführen. Da pro Messung nur ein Kontrollflusspfad innerhalb des zu analysierenden Programmes durchläuft, ist es wichtig, vorab die kritische Pfade innerhalb des Programmes zu identifizieren.

Die durch diesen Ansatz ermittelten Ausführungszeiten stellen allerdings nur eine Abschätzung der WCET dar und keine exakte obere Schranke. Aufgrund der relativ geringen Anzahl unterschiedlicher Kontrollflusspfade innerhalb der zu analysierenden Programme in dieser Projektgruppe wird diese Näherung aber hinreichend genau sein. Bei den zu analysierenden Programmen handelt es sich um Regelalgorithmen, welche mit sehr wenigen unterschiedlichen Kontrollflusspfaden auskommen. Daher stellt die Pfadanalyse und somit die Wahl geeigneter Eingabeparameter kein Problem dar.

Im Folgenden wird ein messbasierter Ansatz durch Instrumentierung vorgestellt, welchen wir im Rahmen dieser Projektgruppe verwenden werden. Hierzu wird durch Instrumentierung des Quellcodes eine Messumgebung erzeugt, mit welcher die Ausführungszeiten der einzelnen Modelle sowie der Komponenten des AEVV-Systems auf der Zielhardware gemessen werden können. Zur Messung wird ein Timer über den Prozessortakt verwendet, weil hierbei ein Hardware-Timer benötigt wird.

Der Ablauf des messbasierten Ansatzes ist in Abbildung 10.11 dargestellt.

Vor Beginn der Messung werden die Daten im Cache und der Pipeline mit zufälligen Werten gefüllt, um zu verhindern, dass sich die Messungen gegenseitig beeinflussen. An-

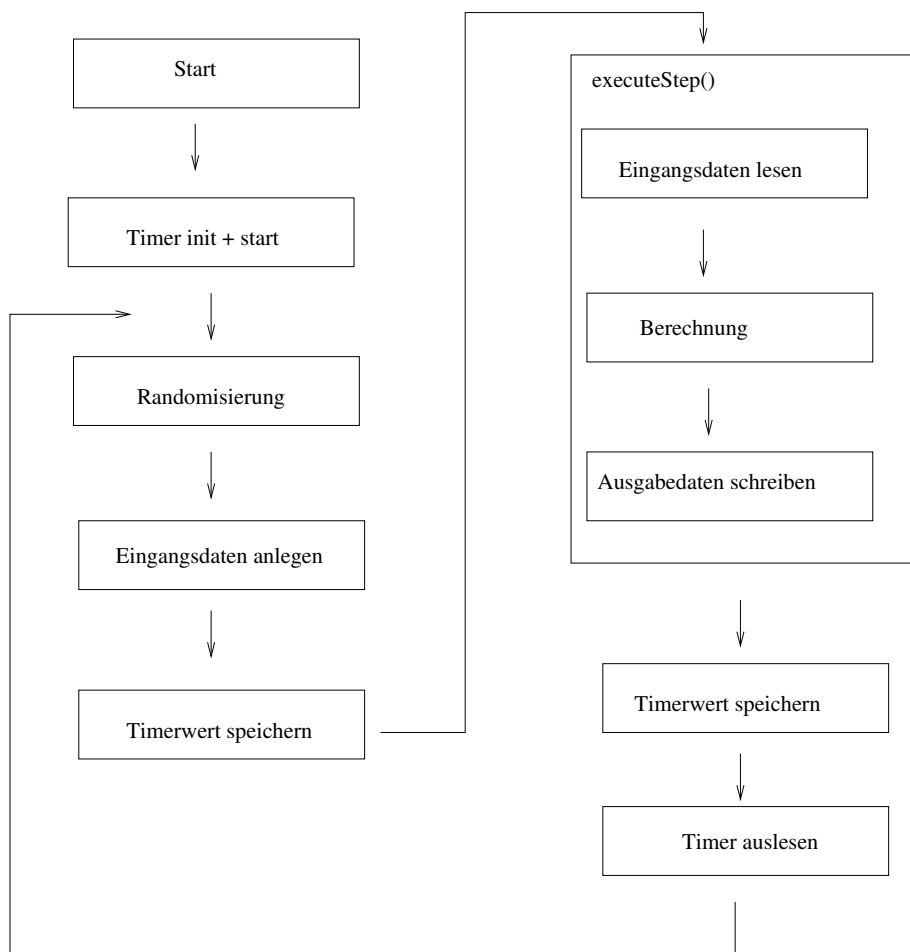


Abbildung 10.11: Analyse der Ausführungszeiten mittels Messung

schließlich werden die Eingangsdaten aus einem Array gelesen, in welchem sich zeilenweise die anzulegenden Stimulidaten befinden. Im Falle des ASG-Systems benötigen die zu testenden Modelle die Ausgangsdaten der vorherigen Berechnung, da es sich hierbei um Regelschleifen handelt. Daher werden in diesem Falle die Ausgabewerte des physikalischen Modells verwendet, mit welchem die Umgebung im Rahmen des HIL-Tests simuliert wird. Vor Aufruf des Modells wird der Timerwert gespeichert, um nach Abschluss der Berechnungen die vergangene Zeit ermitteln zu können. Mit den Stimulidaten als Eingangswerten wird nun die *executeStep* Funktion des Modells aufgerufen. Nach Abschluss der Berechnungen wird der Timerwert erneut ausgelesen und gespeichert. Hiernach erfolgt die Berechnung der Ausführungszeit durch Bildung der Timer Differenz.

10.1.6.3 Auswirkungen des Instrumentierten Codes auf die Ausführungszeit Das Vorgehen der Messung der Ausführungszeiten durch Instrumentierung bringt den Nachteil mit sich, dass hierdurch in das System eingegriffen wird. Durch die Ausführung des eingebrachten Codes wird der Zustand des Caches und der Pipeline verändert, was sich in den Ausführungszeiten bemerkbar macht. Um diese Effekte rückrechnen zu können, wird die Messung zusätzlich ohne den Aufruf der zu messenden Komponente ausgeführt. Hierdurch ist es möglich, die Zeiten zu ermitteln, die durch das Auslesen und Schreiben des Timers vergehen. Die Randomisierung des Caches und der Pipeline haben einen großen Einfluss auf die Ausführungszeiten. Daher kann die obere Grenze der Ausführungszeiten deutlich über der in der Praxis auftretenden Grenze liegen. Deshalb werden die Messungen erneut ohne Randomisierung durchgeführt und die Ergebnisse anschließend verglichen und interpretiert.



10.2 Testvorgehen beim ASG

Wie in den Kapitel 2.2.2 und 10.1.3 bereits vorgestellt wurde, wird beim Testvorgehen des ASGs nach dem V-Modell vorgegangen.

Das Vorgehen nach dem V-Modell wird mit dem modellbasiertem Ansatz kombiniert (Vgl. 10.1.4.4). In Abbildung 10.12 wird das grobe Vorgehen dargestellt. Zunächst wird auf Komponententestebene und Integrationstestebene nur MIL-Tests durchgeführt. Auf dieser Testebene werden keine SIL- und HIL-Test durchgeführt. Auf der Systemtesttestebene werden hingegen sowohl MIL-, SIL und HIL-Test durchgeführt, als auch Testfälle am realen System. Der Abnahmetest wird nicht durchgeführt.

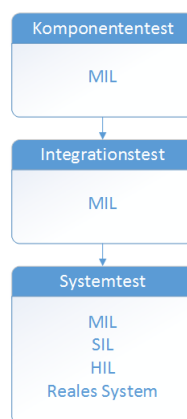


Abbildung 10.12: Testvorgehen beim ASG

Da das ASG ein sicherheitskritisches System ist, ist es von besonderer Bedeutung, dass bestimmte Ausführungszeiten eingehalten werden. Die Validierung dieser Ausführungszeiten wird im Abschnitt 10.2.4 beschrieben.

10.2.1 Komponententest beim ASG

Nachdem eine Komponente entwickelt wurde, muss diese ausreichend getestet werden, damit sichergestellt werden kann, dass diese auch einwandfrei funktioniert. Beim Testvorgehen werden die Komponenten aus dem CAMEL-View-Modell nur auf Modellebene getestet. Der Schwerpunkt liegt hier auf der Validierung des korrekten Verhaltens der einzelnen Komponenten. Dabei wird jede Komponente isoliert von den anderen Komponenten getestet. Es werden die Komponenten aus den folgenden Teilmodellen getestet:

- AttitudeController-stellgroesseTheta
- AttitudeController-stellgroessePhi
- AttitudeController-stellgroessePsi
- HeightController



- MotorValueCalculation

Für jede Komponente wird ein Testfalldokument angelegt, in dem für das Testen dieser Komponente alle relevanten Informationen notiert werden. Ein Testfalldokument für eine Komponente besteht aus folgenden Informationen:

- **Nr.:** Jedem Testfalldokument wird eine eindeutige Identifikation zugewiesen (z.B. KTASG002).
- **Bezeichnung:** Jedes Testfalldokument ist durch einen Bezeichner gekennzeichnet (z.B. LimitationTheta).
- **Testobjekt:** Hier wird das zu testende Objekt angegeben (z.B. AttitudeController-limitationTheta).
- **Beschreibung:** Hier wird die Funktion der Komponente beschrieben.
- **Anzahl der Zweige:** Hier wird die Anzahl der Zweige, die die Komponente aufweist, angegeben.
- **Vorbedingung:** Hier werden Vorbedingungen definiert, die vor dem Ausführen der Testfälle erfüllt sein müssen.
- **Testfall:** Jedem Testfall wird eine eindeutige Bezeichnung zugewiesen. Dieser setzt sich aus der Identifikation des Testfalldokuments und einem Buchstaben zusammen (z.B. KTASG002a).
- **Eingabewerte:** Bei jedem Testfall werden verschiedene Eingaben angegeben, die bei der Ausführung des Testfalls verwendet werden.
- **Erwartete Ausgabe:** Bei der Durchführung der Testfälle, durch die dazugehörigen Eingaben, werden an dieser Stelle die erwarteten Ausgaben notiert.

Bei jeder Komponente ist die innere Struktur bekannt, sodass White-Box-Tests verwendet werden. Bei der Erstellung der Testfälle ist das Ziel die vollständige Zweigabdeckung jeder Komponente. Dazu wird bei jeder Komponente die Anzahl der Zweige ermittelt. In Tabelle 10.1 sind die einzelnen Teilmodelle und dessen Anzahl an Komponenten, Zweigen und Komponententestfällen angegeben.

Damit ergibt sich eine Gesamtanzahl von 87 Zweigen. Jeder Zweig wird durch mindestens einen Testfall abgedeckt. Zusätzlich zur Zweigabdeckung werden weitere Testfälle gebildet, die insbesondere Grenzwerte testen. Insgesamt werden 205 erstellte Komponententestfälle durchgeführt.

Im Folgenden wird ein Beispiel eines Komponententestfalls anhand der Komponente *limitationTheta* des *AttitudeCotnrollers* vorgestellt:

Teilmodell	Anzahl der Komponenten	Anzahl der Zweige	Anzahl der Komponententestfälle
AttitudeControllerstellgroesseTheta	9	17	35
AttitudeControllerstellgroessePhi	9	17	35
AttitudeControllerstellgroessePsi	8	15	35
HeightController	8	13	57
MotorValueCalculation	9	25	43
Gesamt	43	87	205

Tabelle 10.1: Überblick der Teilmodelle mit den Anzahlen an Komponenten, Zweigen und Testfällen

Beispiel *LimitationTheta*: Die Komponente *LimitationTheta* begrenzt den Eingabewert.

Falls der Eingabewert kleiner als -0.349 ist, so wird der Ausgabewert auf -0.349 gesetzt bzw. falls der Eingabewert größer als 0.349 ist, auf 0.349. Im Falle, dass der Eingabewert zwischen -0.349 und 0.349 liegt, so ist der Ausgabewert gleich dem Eingabewert.

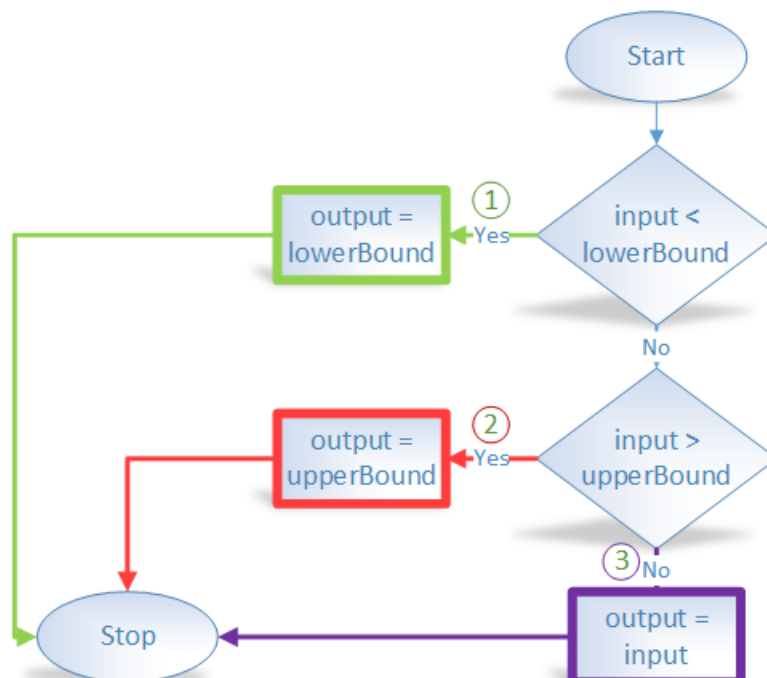


Abbildung 10.13: Zweigabdeckung der Komponente *LimitationTheta*



In Abbildung 10.13 wird ein Kontrollflussgraphen der Komponente *LimitationTheta* dargestellt. Anhand dieser Abbildung ist zu erkennen, dass es in der Komponente *LimitationTheta* drei Zweige gibt. Somit sind für eine vollständige Zweigabdeckung drei Testfälle notwendig:

1. $\text{input} < \text{lowerBound}$
2. $\text{input} > \text{upperBound}$
3. $\text{input} > \text{lowerBound}$ AND $\text{input} < \text{upperBound}$

Zusätzlich werden die Grenzwerte betrachtet. Mit den Werten $\text{lowerBound} = -0.349$ und $\text{upperBound} = 0.349$ ergeben sich die folgenden Grenzwerte:

- -0.350
- -0.349
- 0.349
- 0.350

Für die Komponente *LimitationTheta* sind mit den Grenzwerten auch gleichzeitig die Eingabewerte für die Zweigabdeckung abgedeckt. So dass sich folgende Testfälle für diese Komponente ergeben:

Testfall	Eingabewert	erwartete Ausgabe
KPAC002a	-0.350	-0.349
KPAC002b	-0.349	-0.349
KPAC002c	0.349	0.349
KPAC002d	0.350	0.349

Nach diesem Vorgehen sind für jede Komponente Testfälle erstellt worden. Die gesamten Komponententestfälle für das ASG sowie die dazugehörigen Testprotokolle sind im Testdokument im Abschnitt ?? dokumentiert.

Für die Durchführung der Testfälle wird CAMEL-View verwendet. Bei jeder Testfalldurchführung wird die CAMEL-View-Umgebung zunächst resettet. Anschließend werden die Eingabewerte, die in den Testfällen angegeben sind, in Textfeldern eingegeben. Anschließend wird die Simulation gestartet. Die Ausgaben werden in weiteren Textfeldern angezeigt. Die Ergebnisse der Testfalldurchführungen werden in Testprotokollen gespeichert. Ähnlich wie die Testfälle, werden in den Testprotokollen die Testfallnummer, die Bezeichnung, die Durchführung, die Parameterewrte, das erwartete Verhalten und die Erwartungswerte angegeben. Zusätzlich werden in den Testprotokollen die folgenden Angaben hinzugefügt:



- **Status:** Sofern der Testfall erfolgreich durchgeführt wurde, ist der Status des Testfalls 'bestanden', andernfalls 'nicht bestanden'.
- **Datum:** Hier wird angegeben, an welchem Tag der Testfall durchgeführt wurde.
- **Protokollant:** Hier wird der Protokollant, der den Testfall durchgeführt und protokolliert hat, angegeben.
- **tatsächlicher Wert:** Neben dem erwarteten Werten, werden im Testprotokoll die tatsächlichen Werte, die bei der Testdurchführung ausgegeben wurden, angegeben.
- **Testergebnis:** Hier wird das Testergebnis zusammengefasst.

Die Testprotokolle der Komponententestfälle befinden sich im Testdokument im Abschnitt ??.

10.2.2 Integrationstest beim AGS

Bei den Integrationstests werden die zuvor getesteten Komponenten zu größeren Teilmodellen zusammengefügt. Da jede Komponente sehr übersichtlich aufgebaut ist und nur wenige Operationen aufweist, wird der *big-bang*-Integrationsansatz verwendet. Es werden die folgenden Teilmodelle getestet:

- Attitudecontroller - stellgroesseTheta
- Attitudecontroller - stellgroessePsi
- Attitudecontroller - stellgroessePsi
- HeightController
- MotorValueCalculation
- Physikalisches Modell

Wie beim Komponententest, werden die Teilmodelle nur auf Modellebene getestet. Der Aufbau eines Integrationstestfall ist ähnlich dem eines Komponententestfalls. Es werden eine eindeutige Identifikationsnummer, eine Bezeichnung, das Testobjekt und die Vorbedingungen angegeben. Zusätzlich werden die folgenden Angaben zu jedem Testfall hinzugefügt:

- **Anzahl der abgedeckten Zweige:** Hier wird die Anzahl der Zweige aus den Komponenten, die durch diesen Testfall abgedeckt werden, angegeben.
- **Testdurchführung:** Hier wird die genauere Testdurchführung beschrieben. Es wird angegeben, wann welche Eingaben gemacht werden müssen.
- **Erwartetes Ergebnis:** Hier wird das erwartete Ergebnis beschrieben.

10.2.2.1 AttitudeController und HeightController Der Schwerpunkt bei den Teilmodellen der AttitudeController und dem HeightController liegt auf das Zusammenspiel der verschiedenen Komponenten miteinander und der korrekten Begrenzung verschiedener Werte. Als Ausgangspunkt für die Erstellung der Testfälle wird die Zweigabdeckung verwendet. Jeder Zweig aus jeder Komponente soll durch mindestens einen Testfall abgedeckt werden. Im Vergleich zu den Komponententests, bei denen nur zu Beginn einer jeden Testdurchführung Eingaben in der Simulation getätigt wurden, werden bei den Integrationstests auch zusätzlich während der Simulation neue Eingaben gemacht. Dadurch wird jeder Testfall umfangreicher. In Tabelle 10.2 ist zu erkennen, dass es mehr Zweige als Testfälle gibt. Da aber jeder Testfall durch die Mehrfacheingaben umfangreicher ist, deckt jeder Testfall mehr als einen Zweig ab.

Teilmodell	Anzahl der Zweige	Anzahl der Integrationstestfälle
AttitudeController-stellgroesseTheta	17	7
AttitudeController-stellgroessePhi	17	7
AttitudeController-stellgroessePsi	15	6
HeightController	13	5
Gesamt	52	25

Tabelle 10.2: Überblick der Integrationstestfälle: AttitudeController und HeightController

Die erstellten Testfälle testen, ob die Stellgrößen bzw. der Schub korrekt berechnet und begrenzt werden. Die Testfälle werden mit CAMEL-View durchgeführt. Während der Simulation werden die Eingabewerte geändert. Daraus resultiert eine Sprungantwort des Ausgabewerts. Bei der Sprungantwort wird analysiert, ob der Wert korrekt berechnet und begrenzt wird.

10.2.2.2 MotorValueCalculation Für das Teilmodell *MotorValueCalculation* sind sechs Testfälle erstellt worden, mit denen jeder der 25 Zweige im Teilmodell *MotorValueCalculation* abgedeckt wird. Mit diesen sechs Testfällen wird getestet, ob die Motorstellwerte korrekt berechnet und auch korrekt begrenzt werden. Die Testfälle werden ebenfalls mit CAMEL-View durchgeführt. Die Ergebnisse werden anschließend mit den erwarteten Ergebnissen verglichen.

10.2.2.3 Physikalisches Modell Da das Physikalische Modell aus einem Mehrkörpersystem besteht, kann dieses nicht in Komponenten eingeteilt und getestet werden. Daher wird das Physikalische Modell erst durch die Integrationstests getestet. Mit den Integra-



tionstest kann keine Zweigabdeckung erreicht werden, da das Physikalische Modell aus einem Mehrkörpersystem besteht. Daher werden nur die Drehbewegungen des Modells getestet. Das Physikalische Modell hat vier Eingabewerte, von denen jeder die Kraft eines der vier Motoren beschreibt. Je nachdem wie groß diese Eingabe-Kräfte sind, verändert sich die Lage des Physikalischen Modells. Für die Drehbewegungen Gieren, Nicken, Rollen, Steigen und Sinken sind Testfälle erstellt worden. Insgesamt gibt es acht Testfälle, die das Physikalische Modell testen. Der Aufbau der Testfälle ist identisch zu den anderen Integrationstestfällen. Es werden in den Testfällen die Eingaben für die vier Kräfte angegeben und beim erwartetem Ergebnis werden die erwartete Position, die erwarteten Winkel und die erwarteten Winkelgeschwindigkeiten angegeben. Nach dem die Testfälle mit CAMEL-View durchgeführt wurden, werden die Ergebnisse mit den erwarteten Ergebnissen verglichen.

Die gesamten Integrationstestfälle für das ASG sowie die dazugehörigen Testprotokolle sind im Testdokument im Abschnitt ?? dokumentiert.

10.2.3 Systemtests beim ASG

Bei den Systemtests wird das gesamte System getestet. Hier wird vorausgesetzt, dass alle getesteten Komponenten und Teilmodelle einwandfrei funktionieren. Zunächst wird in der CAMEL-View-Umgebung der Model-In-The-Loop-Test durchgeführt. Nach erfolgreichem Model-In-The-Loop-Test folgt der Software-In-The-Loop-Test und anschließend der Hardware-In-The-Loop-Test. Nachdem diese erfolgreich getestet wurden, folgen die Systemtests am realen Quadrokopter.

10.2.3.1 Vorgehen beim Model-In-The-Loop Beim Model-In-The-Loop-Test (MIL-Test) auf Systemebene wird das Gesamtsystem, welches aus den Lagereglern, dem Höhenregler, der Motorstellwerteberechnung und dem Physikalischen Modell besteht, getestet. Der Schwerpunkt bei diesem Testdurchgang liegt auf dem Reglerverhalten, da hier erstmals mithilfe des Physikalischen Modells die Regler im geschlossenen Regelkreis getestet werden.

Als Grundlage für die Erstellung der MIL-Testfälle dienen unter anderem die funktionalen Anforderungen an das ASG (Vgl. Kapitel 6.1). Da im Modell jedoch nicht alles modelliert ist, wie z.B. die Sensoren oder der Akku, können mit den MIL-Testfällen nicht alle funktionalen Anforderungen abgedeckt werden. Die MIL-Testfälle decken 8 von 15 funktionalen Anforderungen an das ASG ab.

Zunächst werden mit den Testfällen STMIL?? - STMIL?? einzelne Flugbewegungen wie das Steigen, Sinken, Nicken und Rollen getestet. Diese Flugbewegungen werden mit maximalen und minimalen Eingaben getestet. Darauf folgen zwei Testfälle (STMIL??, STMIL??) die den Höhenregler testen. Anschließend werden mit den Testfällen STMIL?? - STMIL??



die einzelnen Flugbewegungen miteinander kombiniert. Mit den Testfällen STMIL?? - STMIL?? werden die verschiedenen Flugbewegungen miteinander kombiniert und zusätzlich wird der Höhenregler aktiviert. Das Gieren wird mit den Testfällen STMIL?? - STMIL?? überprüft. Die letzten Testfälle (STMIL?? - STMIL??) beschreiben Szenarien, die der Quadroptopter in der Simulation fliegen soll. Zu den Szenarien gehören z.B. „die Halle abfliegen“ oder „Achten fliegen“. Ein weiteres Szenario ist ein Langzeittest, bei dem der Quadroptopter über einen längeren Zeitraum (Akkulaufzeit) verschiedene Manöver fliegt. Die Szenarien sind nur für Eingaben mit der Fernbedienung geeignet, da das gleichzeitige Ändern von mehr als einem Eingabewert in der Simulationsumgebung nicht möglich ist. Daher werden diese Testfälle nur einmal durchgeführt. Die beschriebenen Testfälle sind zur Validierung der Funktionen und des Reglerverhaltens entworfen worden und werden für den SIL- und HIL-Test nicht weiter verwendet. Für den Vergleich zwischen MIL-, SIL- und HIL-Test sind die drei Testfälle STMIL??, STMIL?? und STMIL?? erstellt worden. In diesen Testfällen werden verschiedene Manöver miteinander kombiniert, damit alle wesentlichen Funktionen, wie z.B. das Fliegen in verschiedene Richtungen oder das Aktivieren des Höhenreglers, abgedeckt werden.

Ein Testfall ist nach dem gleichen Schema wie die Komponenten- und Integrationstestfälle aufgebaut. Jeder Testfall besteht aus einer eindeutigen Identifikationsnummer, einem Bezeichner, dem Testobjekt, Vorbedingungen, der Testdurchführung und dem erwarteten Ergebnis. Zusätzlich werden bei den MIL-Testfällen noch die Anforderungen, die mit den jeweiligen Testfall abgedeckt werden, angegeben.

Die MIL-Testfälle sind im Testdokument im Abschnitt ?? zu finden.

Die MIL-Tests werden in der CamelView-Umgebung durchgeführt. Dabei ist jeder Testfall für zwei Testdurchläufe ausgelegt. Beim ersten Testdurchlauf werden die Eingaben direkt in die CamelView-Umgebung eingegeben. Beim zweiten Testdurchlauf werden die Eingaben mittels der Fernbedienung eingegeben. Dazu werden die Fernsteuersignale direkt in die Simulation eingespeist (siehe Kapitel 10.2.3.2 und Kapitel 10.2.3.3). Die unterschiedlichen Durchführungen bei beiden Durchgängen werden bei jedem Testfall bei der Testdurchführung beschrieben. Bei beiden Testdurchläufen werden bei jedem Testfall sowohl die Eingabewerte als auch die Ausgabewerte, sowie einige weitere Debugwerte, in CSV-Dateien gespeichert. Nachdem die Testfälle durchgeführt wurden, werden die CSV-Dateien analysiert. Bei der Analyse der CSV-Dateien wird untersucht, wie stark die Istwerte von den Sollwerten abweichen und wie lange es dauert bis die Regelgrößen in das jeweilige Toleranzband eintreten. Weiterhin wird untersucht wie stark das System schwingt und wie groß die maximalen Abweichungen sind. Bei den Winkeln Theta, Phi und Psi wird eine Abweichung von 0.1% toleriert. Bei der Höhe werden größere Abweichungen toleriert, da die Sensoren für die Höhenregelung eine geringere Genauigkeit aufweisen. Zusätzlich zu den CSV-Dateien werden die Winkel und die Positionen in Timeplots dargestellt (Abb. 10.14 und Abb. 10.15).

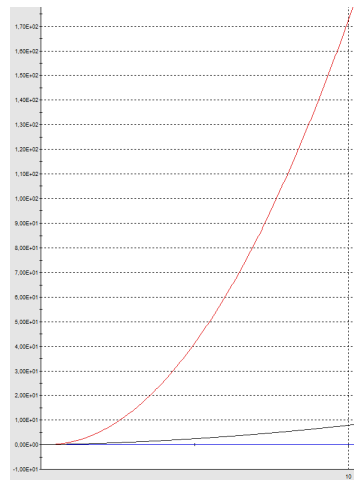


Abbildung 10.14: Verlauf der Position des Quadropters (Testfall STMIL??)

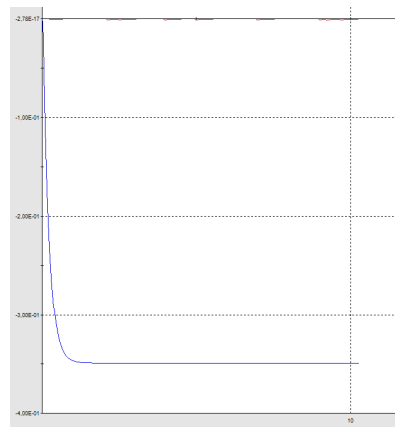


Abbildung 10.15: Verlauf der Winkel (Testfall STMIL??)

Die Ergebnisse der Analyse und die Timeplots werden in Testprotokollen dokumentiert. Zu jedem Testfall gibt es zwei Testprotokolle. Das erste Testprotokoll beschreibt Durchführung und die Ergebnisse des Testfalls, bei dem die Eingaben direkt in der Simulation eingegeben wurden, und das zweite Testprotokoll beschreibt die Ergebnisse der Testdurchführung bei dem die Eingaben mit der Fernbedienung gemacht wurden. Jedes MIL-Testprotokoll referenziert auf den dazugehörigen Testfall, um damit die Transparenz zu sichern. In jedem Testfall wird der Status, ob ein Testfall bestanden oder nicht bestanden wurde, der Protokollant, der das Testprotokoll erstellt hat, und das Datum, an dem der Testfall durchgeführt wurde, angegeben. Anschließend folgt eine Beschreibung der Durchführung, der verwendeten Eingabewerte und des erwarteten Verhaltens. Bei den MIL-Testprotokollen werden zusätzlich die Timeplots der Winkel und der Position dargestellt.

Die MIL-Testprotokolle sind im Testdokument in den Abschnitten ?? und ?? zu finden.

10.2.3.2 Einbindung der Funkfernsteuerung Für den MIL-Test müssen die Fernsteuerungssignale direkt in die Simulation eingespeist werden. Zur Anbindung von Hardware an die Simulationsumgebung hat der Hersteller iXtronics die TestRig-Box zur Verfügung gestellt. Da die TestRig-Box keinen PPM-Eingang für das Summensignal von der Funkfernsteuerung direkt zur Verfügung stellt, muss ein Transducer entwickelt werden (siehe Abbildung 10.16). Dieser dekodiert das PPM-Signal und stellt die einzelnen Kanäle über eine SPI-Schnittstelle zur Verfügung.

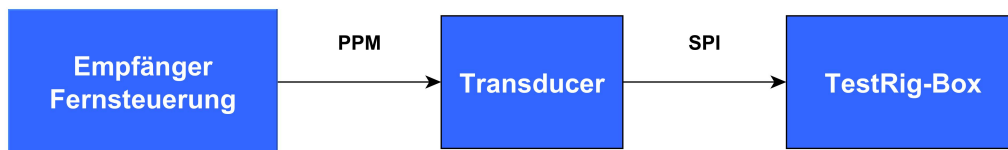


Abbildung 10.16: Aufbau MIL-Test

Als Controller für den Transducer wird ein ATmega32 von Atmel verwendet. Das Summensignal wird an einen externen Interrupt-Pin angeschlossen. Die Zeit zwischen zwei steigenden Flanken wird über einen Timer gemessen. Außerdem wird der ATmega32-Controller als SPI-Slave betrieben und stellt die ermittelten Kanalwerte im SPI-Datenregister zur Verfügung.

Bei der aktuellen Implementierung der Firmware für den Transducer werden die ersten 8 Kanäle aus dem Summensignal dekodiert und byteweise per SPI nacheinander zur Verfügung gestellt. In Abbildung 10.17 ist der Testaufbau für den MIL-Test dargestellt.

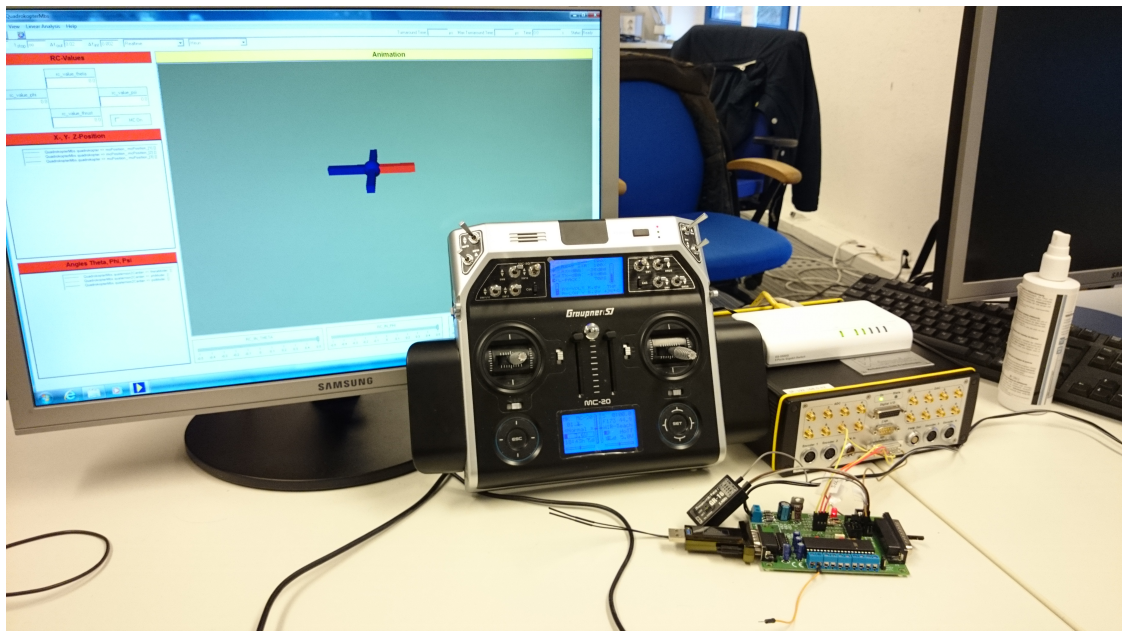


Abbildung 10.17: Peripherie MIL-Test

10.2.3.3 Datenaustausch über SPI Zur Durchführung der MIL-Tests ist es vorgesehen, die aktuellen Soll-Positionen von der Funksteuerung in das Modell zu laden. Dazu wurde bereits im vorherigen Abschnitt die benötigte Hardware erklärt. Damit das Modell diese Werte erhält, muss eine Kommunikationschnittstelle eingerichtet werden. Dazu wird die TestRig-Box verwendet, welche ein Serial Peripheral Interface (SPI) zwischen dem entwickelten Modell und der Funksteuerung anbietet. Folglich muss das verwendete Modell um einen Block für das SPI erweitert werden. Innerhalb dieses Blocks wird das SPI initialisiert, empfangene Daten aufbereitet und diese den Regler-Blöcken innerhalb des Modells zur Verfügung gestellt.

In Tabelle 10.3 sind die Daten aufgelistet, welche über das SPI im CAMEL-View-Modell empfangen werden müssen.

Tabelle 10.3: Auflistung der zu empfangenen Daten über SPI im MIL-Test

Datum	Speichertiefe
Sollwert Roll-Achse	1 Byte
Sollwert Nick-Achse	1 Byte
Sollwert Gier-Achse	1 Byte
Sollwert Schub	1 Byte
Schalterstellung Höhenregelung	1 Byte

Die Summe der Speichertiefe ergibt 5 Bytes (40 Bits), welche in jedem Zyklus übertragen werden müssen. Die von CAMEL-View mitgelieferten Treiber ermöglichen einen maximalen Datentransfer von 64 Bits bzw. 8 Bytes pro Zyklus. Somit können alle erforderlichen Daten in einem Integrationsschritt durchgeführt werden, welcher in diesem Falle 2 ms beträgt. Die SPI Taktfrequenz, die vom Master (hier: TestRig-Box) vorgegeben wird, muss in diesem Falle mindestens folgende Übertragungsrate f aufzeigen, damit die SPI-Kommunikation nicht die Deadline von 2 ms verletzt:

$$f > \frac{40 \text{ Bits}}{(2 - x) \text{ ms}} \quad (10.1)$$

Hierbei entspricht x die Zeitdauer, die das Modell benötigt, um alle weiteren Berechnungen in diesem Integrationsschritt ausführen zu können. Da diese Zeitdauer schwierig abzuschätzen ist, sollte eine großzügige Frequenz gewählt werden. Theoretisch müsste bei Vernachlässigung dieser Zeit die Übertragungsrate größer als 20 kHz sein. Während der Durchführung der MIL-Tests wurden die Daten mit einer Frequenz von 50 kHz übertragen. Der hierarchisch höchste SPI-Block `dataProcessing` ist in Abbildung 10.18 dargestellt. Dieser besitzt neben Ausgängen noch Einstellungsparameter zur komfortablen Konfigura-

tion des SPIs. Dadurch besteht die Möglichkeit, die Parameter außerhalb des Blocks zu verändern (s. Abbildung 10.18, gelbe Kästchen).

Die Ausgänge des C-Code-Blocks spiegeln die Daten aus Tabelle 10.3 wieder (s. Abbildung 10.18, grüne Kästchen). Jedes Datum ist einem bestimmten Ausgang zugeordnet und kann somit in das gesamte Modell integriert werden.

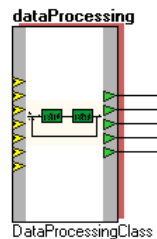


Abbildung 10.18: SPI-Schnittstelle innerhalb des CAMEL-View-Modells für den MIL-Test

In CAMEL-View ist die Realisierung des SPIs durch C-Code-Blöcke vorgesehen, da die jeweiligen Treiber in der Programmiersprache C vorliegen. Folglich verbergen sich hinter dem Block `dataProcessing` u.a. kleine C-Code-Blöcke zur Inbetriebnahme des SPIs (s. Abbildung 10.19). Um Zugriffe auf die SPI-Treiber-Funktionen zu erhalten, muss in jedem betroffenen C-Code-Block das Header-File `ix_mc02_io.h` aus der CAMEL-View-Installationsumgebung inkludiert werden. Diese werden durch den hierarchischen Block `spiInterface` von der Umgebung gekapselt. Dieser initialisiert das SPI mit dem vorgegebenen Parametern und bildet die empfangenen Rohdaten auf die zwei Ausgänge ab (hier: jeweils einen für die niederwertigen und einen für die höherwertigen 32 Bytes). Diese Daten werden danach durch den Block `remoteControlProcessing` für das Modell aufbereitet und anschließend gefiltert. Der Block `spiCore` hat noch keine größere Bedeutung. Momentan gibt er nur das aktuelle Slave-Select-Signal aus. Des Weiteren könnte er zukünftig noch weitere Aufgaben übernehmen.

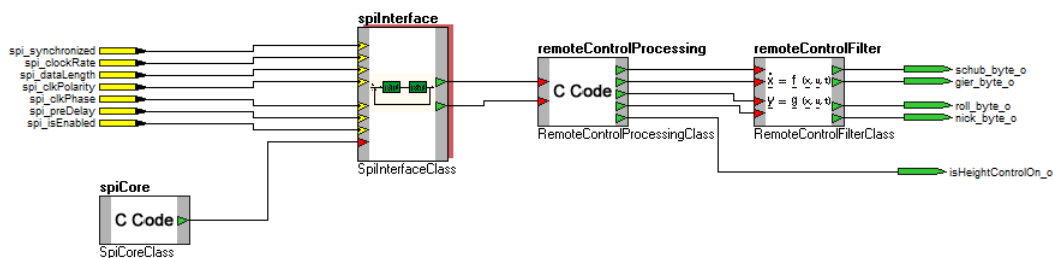


Abbildung 10.19: Detaillierte SPI-Schnittstelle des Blocks `dataProcessing` für den MIL-Test

Hinweis: Bei der Verwendung von C-Code-Blöcken ist es wichtig, die gesamte C-Routine nur im *Major-Step* bzw. im Hauptschritt des Integrationsschritts auszuführen. Dazu gibt



es eine Funktion namens `isMajorStep()`, die durch eine `if`-Abfrage auf ihre Wahrheit geprüft werden sollte.

10.2.3.4 Vorgehen beim Software-In-The-Loop Nachdem der MIL-Test erfolgreich durchgeführt wurde, wird der Software-In-The-Loop-Test (SIL) durchgeführt. Für den SIL-Test wird aus dem CAMEL-View-Modell, inklusiv dem physikalischen Modell, C-Code generiert. Da es sich bei CAMEL-View um ein nicht verifiziertes Werkzeug handelt, muss sichergestellt werden, dass sich der C-Code äquivalent zum Modell verhält. Dazu kann die Modelltransformation verifiziert oder das Verhalten des C-Codes validiert werden. Da die Verifikation der Modelltransformation sehr aufwendig ist und den Rahmen dieses Projekts weit übersteigen würde, wird nur die Validierung des Verhaltens durchgeführt. Hierbei wird überprüft, ob sich der C-Code äquivalent zum Modell verhält. Dazu werden die beim MIL-Test gespeicherten CSV-Dateien verwendet. Der C-Code wird mit den in den CSV-Dateien gespeicherten Eingaben ausgeführt und die Ausgaben vom C-Code werden in einer weiteren CSV-Datei gespeichert. Anschließend können die Ausgaben vom SIL-Test mit denen vom MIL-Test verglichen werden. Es wird erwartet, dass der durchschnittliche Unterschied zwischen den Ausgaben kleiner als 0.1 % ist.

Der SIL-Test wird nur auf der Systemtestebene durchgeführt, d.h. es wird der gesamte Regler mit dem physikalischen Modell getestet. Die Komponenten- und Integrationstests werden auf dieser Ebene nicht durchgeführt, da für jeden Test das Framework zum Ausführen des Modells angepasst werden muss.

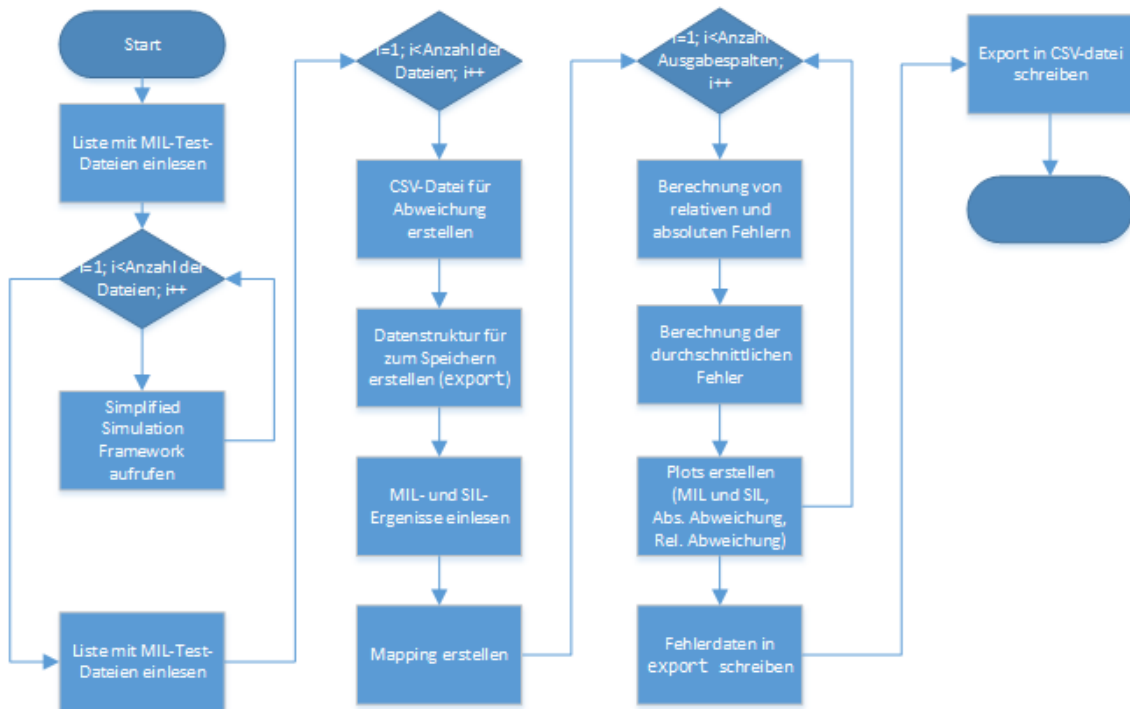


Abbildung 10.20: Ablaufdiagramm MatlabScript

Mit Hilfe von Matlab wird das Framework zur Ausführung des Modells mit entsprechenden Parametern aufgerufen. Anschließend werden sowohl die MIL- und SIL-Dateien eingelese und die Ergebnisse miteinander abgeglichen. Das Matlab-Script berechnet dazu die absoluten und relativen Fehler der einzelnen Werte. Zusätzlich werden in einer weiteren Datei die benötigten Daten für das Protokoll gespeichert. Abbildung 10.20

Nach [FPL12] ist der absolute Fehler die Differenz zwischen dem wahren und dem gemessenen Wert. Der relative Fehler ist eine Bezugsgröße auf den wahren Wert. Für den SIL-Test bedeutet dies, dass sich der absolute Fehler durch die Differenz der Werte vom SIL- und vom MIL-Test ergibt (Formel 10.2). Die Bezugsgröße für den relativen Fehler ist der Wert des MIL-Tests, so dass sich die Formel 10.3 ergibt.

$$F_{abs} = y_{sil} - y_{mil} \quad (10.2)$$

$$F_{rel} = \frac{F_{abs}}{y_{mil}} = \frac{y_{sil} - y_{mil}}{y_{mil}} \quad (10.3)$$

10.2.3.5 Vorgehen beim Hardware-In-The-Loop Im HIL-Test wird der Zielcode auf der Zielplattform ausgeführt. Das physikalische Modell wird weiterhin simuliert. Es stellt der Zielhardware Umgebungsparameter bereit und nimmt Motorstellwerte entgegen (s. Kapitel 10.1). Das ursprünglich geplante HIL-Test Konzept wird in Abbildung 10.21 dargestellt.

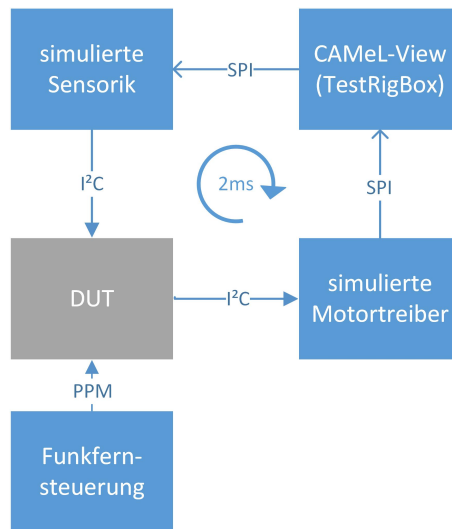


Abbildung 10.21: Geplanter HIL-Testaufbau

Aus der CAMEL-View Simulationsumgebung werden Sensordaten über SPI bereit gestellt. Dies geschieht mit Hilfe der TestRig-Box. Da die realen Sensoren über I²C-Schnittstellen an die Zielplattform angeschlossen sind, werden zusätzlich Sensoren simuliert. Diese stellen die Sensorwerte über I²C-Schnittstellen zur Verfügung. Die I²C-Register und Busadressen sind hierbei genauso wie bei den später verwendeten Hardware Sensoren. Dadurch können Buszugriffe mit getestet werden und der Programmcode auf der Zielplattform bleibt unverändert.

Die Sollwerte für Lage und Höhe werden der Zielplattform, wie im realen System, über die Fernsteuerung bereit gestellt. Auf Basis der Sensorwerte und der Sollwerte werden Motorstellwerte berechnet, die über eine I²C-Schnittstelle an simulierte Motortreiber weitergegeben werden. Diese Motortreiber stellen die berechneten Stellwerte der Simulation wieder über eine SPI-Schnittstelle zur Verfügung.

Bei dem oben vorgestellten ursprünglichen HIL-Test Konzept treten Probleme mit der SPI-Kommunikation auf, weshalb es nicht zur Durchführung der HIL-Tests genutzt werden kann. Bei CAMEL-View kann in jedem Integrationsschritt nur eine SPI-Kommunikation mit maximal 8 Datenbytes erfolgen. Da wesentlich mehr Daten übertragen werden müssen (siehe Kapitel 10.2.3.6) kann die Integrationsschrittweite von 2 ms nicht eingehalten werden, sondern muss deutlich reduziert werden. Außerdem gibt es Probleme mit den simulierten Motortreibern, da diese nach jedem Übertragenen SPI-Byte eine kurze Zeit benötigen, um die SPI-Datenregister neu zu setzen. Der SPI-Treiber für die TestRig-Box erlaubt jedoch keine Pause zwischen den einzelnen Bytes während eines Integrationsschrittes. So muss das Rücklesen der Motorstellwerte byteweise auf mehrere Integrationsschritte verteilt durchgeführt werden. Dies verringert die Zeit, die für einen Integrationsschritt übrig bleibt weiter. Insgesamt verringert sich die maximale Integrationsschrittweite auf 166 μ s. Unter

600 μ s pro Integrationsschritt kann das Modell auf der TestRig-Box allerdings nicht mehr sauber ausgeführt und die Simulationsansicht wird nicht mehr aktualisiert. Ein Durchführen der Tests ist daher nicht möglich.

Damit dennoch der Test auf der Zielplattform durchgeführt werden kann, wird ein modifizierter Ansatz gewählt. Es handelt sich dabei nicht mehr um einen klassischen HIL-Test, da die Rückkopplung („in-the-loop“) mit diesem neuen Konzept aufgelöst wird (s. Abbildung 10.22). Durch diese Auflösung wird die bekannte Problematik mit dem Zurücklesen der Motorstellwerte umgangen.

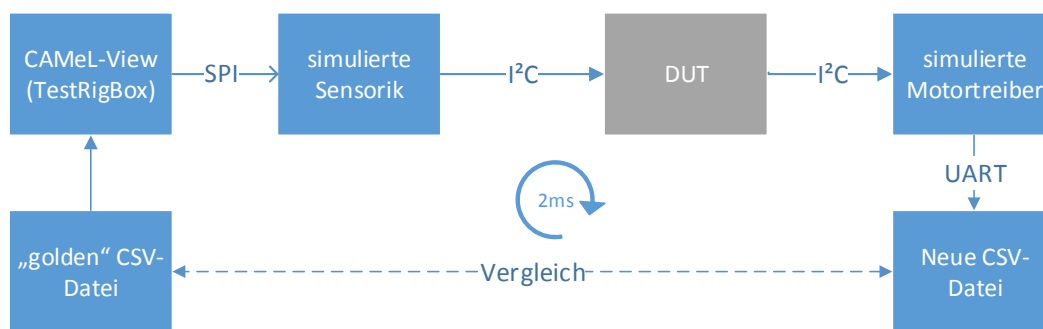


Abbildung 10.22: Modifiziertes Konzept für den HIL-Test

Die Rückkopplung entsteht durch das Versenden der neu berechneten Motorstellwerte. Diese werden in der neuen Methodik nicht wieder an die TestRig-Box bzw. an das physikalische Modell gesendet; sie werden stattdessen in einer externen CSV-Datei auf einem lokalen Rechner gespeichert. Für diesen Datenaustausch wird eine Universal Asynchronous Receiver Transmitter (UART)-Verbindung zwischen dem lokalen Rechner und der Zielplattform aufgebaut. Da die Eingabewerte äquivalent zu den vorherigen Tests sind, erhält das physikalische Modell die neuen Motorstellwerte aus einer vorherigen CSV-Datei. Dieser Ansatz weist einen weiteren Nutzen für die Auswertung der Ergebnisse auf. Durch die Erzeugung einer weiteren CSV-Datei mit den ermittelten Motorstellwerten können diese Werte mit den Ergebnissen aus vorherigen Tests verglichen werden. Um weiterhin die Konsistenz zu vorherigen Tests zu gewährleisten, werden die Eingabewerte (Sollwerte) der Funksteuerung aus einer vergangenen CSV-Datei ausgelesen und über SPI an die Avionik geschickt.

Der Aufbau für den HIL-Test wird in Abbildung 10.23 dargestellt.

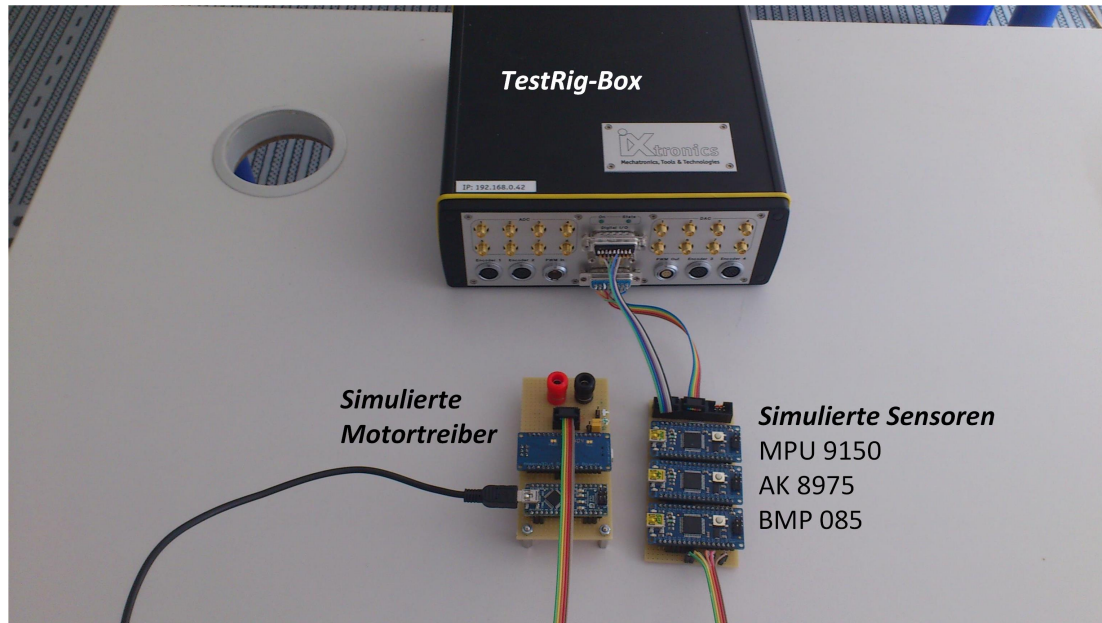


Abbildung 10.23: Aufbau HIL-Test

10.2.3.6 Datenaustausch über SPI Ähnlich wie bei dem MIL-Test (s. Kapitel 10.2.3.3) muss auch beim HIL-Test ein Datenaustausch stattfinden. In diesem Fall besteht dieser Austausch zwischen dem Modell und den Reglern auf der Zielarchitektur über die bekannte SPI-Schnittstelle. Das dazugehörige Konzept wird im Kapitel 10.2.3.5 erklärt.

Eine bidirektionale Kommunikation über SPI ist, wie bereits im Kapitel 10.2.3.2 erwähnt, nicht möglich. Folglich muss die Übertragung jedes Datums sequentiell durchgeführt werden. Eine Übersicht über die auszutauschenden Daten ist in Tabelle 10.4 dargestellt. Damit der HIL-Test vergleichbare Ergebnisse aufzeigen kann, müssen die Eingabewerte äquivalent mit denen aus den vorherigen MIL- und SIL-Tests sein. Deswegen werden die Sollwerte der Funksteuerung über eine vorherige CSV-Datei im Modell eingelesen und über SPI an die Zielarchitektur übertragen. Das Versenden der Motorstellwerte an das Modell ist nicht nötig, da diese Werte ebenfalls aus der vorherigen CSV-Datei geladen werden. Die neu generierten Motorstellwerte hingegen werden über die UART-Verbindung in einer separaten Datei gespeichert. Somit können die Motorstellwerte aus dem HIL-Test mit den Werten aus dem CAMEL-View-Modell verglichen werden.

Tabelle 10.4: Auflistung der auszutauschenden Daten über SPI im HIL-Test

Datum	Speichertiefe
Sensorwert der Beschleunigung (3 Achsen)	6 Bytes (3 x 2 Bytes)
Sensorwert der Gyroskope (3 Achsen)	6 Bytes (3 x 2 Bytes)
Sensorwert des Kompasses	2 Bytes

Tabelle 10.4: Auflistung der auszutauschenden Daten über SPI im HIL-Test (Fortsetzung)

Datum	Speichertiefe
Sollwerte für Roll-, Nick- und Gier-Achse	6 Bytes (3 x 2 Bytes)
Sollwert für den Schub	2 Bytes
Aktuelle Flughöhe	4 Bytes
Schalterstellung Höhenregelung	1 Byte
	Σ 27 Bytes

Da die Frequenz für die Lageregelung 500 Hz beträgt, müssen die Daten - mit Ausnahme von der Höhe und des Kompasses - innerhalb von 2 ms aktualisiert werden (21 Bytes). Wie bereits im Kapitel 10.2.3.3 erwähnt, erlauben die SPI-Treiber für die TestRig-Box einen Datenaustausch von 8 Bytes pro Integrationsschritt innerhalb der Simulation. Dies hat zur Folge, dass eine SPI-Kommunikation für die Lageregelung

$$\frac{21 \text{ Bytes}}{8 \text{ Bytes}} \approx 3 \text{ Integrationsschritte} \quad (10.4)$$

benötigt. Des Weiteren wird ein vierter Integrationsschritt benötigt, um die Werte für den Kompass und für die Höhe abwechselnd zu aktualisieren. Daraus lässt sich ableiten, dass ein Integrationsschritt der Simulation nur einen Viertel der Zeit von 2 ms betragen darf. Der daraus resultierende Ablauf ist in Abbildung 10.24 dargestellt.

Bei vier Integrationsschritten reicht es aus, die Simulation mit einer Periode von 500 μ s zu aktualisieren. Aus technischen Gründen ist es nicht möglich, solch eine Periode für die Integrationsschrittweite einzustellen. Diese lässt sich in CAMEL-View nur auf ein Vielfaches von 200 μ s konfigurieren, sodass das nächst kleinere Intervall 400 μ s ist. Um weiterhin die Aktualisierungsrate von 2 ms zu erhalten, wird ein „Dummy“-Frame eingefügt, welches die Gesamtperiodendauer von 1.6 ms auf 2.0 ms aufstockt. Dadurch wird die Vergleichbarkeit mit den vorherigen Ergebnissen aus den MIL- und SIL-Test gewährleistet, welche ebenfalls in CSV-Dateien gespeichert sind.

Das Versenden des Wertes für die Höhe bzw. für den Kompass wird mit jedem neuen Ablauf abwechselnd durchgeführt. Dadurch werden diese Werte alle 4 ms aktualisiert, was jedoch aufgrund der deutlich höheren Aktualisierungsrate der Sensoren (s. Kapitel 7.3.7) ausreichend ist.

Die SPI-Initialisierungsroutine ist bis auf die Übertragungsrate weiterhin die gleiche wie beim MIL-Test. Da bei diesem Test mehr Daten übertragen werden müssen, ergibt sich eine schnellere Mindestfrequenz f (s. Formel 10.5). Das x ist analog zum MIL-Test die Zeitdauer für die restlichen Berechnungen innerhalb der Simulation. Theoretisch müsste bei Vernachlässigung dieser Zeit die Übertragungsrate größer als 160 kHz sein. Für die Durchführung wird eine Übertragungsrate von 200 kHz vorgesehen.

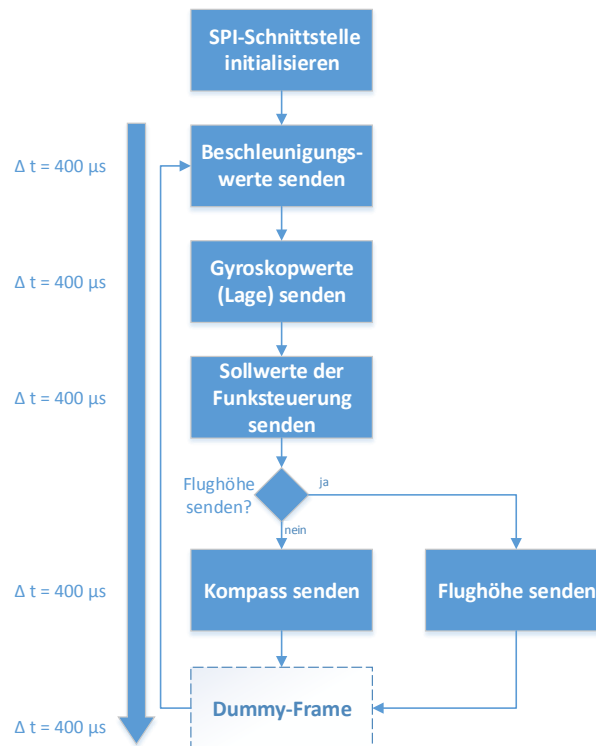


Abbildung 10.24: Ablaufdiagramm für die SPI-Kommunikation während des HIL-Tests

$$f > \frac{64\text{Bits}}{400\mu\text{s} - x} \quad (10.5)$$

In CAMEL-View wird dazu ein neues SPI-Modell erstellt, welches den Ablauf aus Abbildung 10.24 durch C-Code-Blöcke realisiert. Der SPI-Block aus CAMEL-View ist in Abbildung 10.25 dargestellt. Da nur Werte aus dem Modell an die Zielarchitektur gesendet werden, existiert für jedes Datum ein separater Eingang (rote Kästchen). Analog zum MIL-Test sind die Konfigurationsparameter für die SPI-Kommunikation außerhalb des SPI-Blocks einstellbar (gelbe Kästchen).

Als Ausblick für die Realisierung des HIL-Test im CAMEL-View kann der in Abbildung 10.25 dargestellte C-Code-Block analog zur Abbildung 10.19 modularer gestaltet werden. Neben der Initialisierungsroutine kann der Daten- und der Kontrollfluss sowie die Anwendungsdaten- und SPI-Logik voneinander durch separate Blöcke getrennt werden.

10.2.3.7 Systemtest am realem System Bei den Systemtests am realen System wird das Physikalische Modell durch den realen Quadrokoopter ersetzt. Die Systemtests am realem System dienen zur Überprüfung der in Kapitel 6.1 gestellten Anforderungen. Um sicherzustellen, dass auch alle Anforderungen an das ASG abgedeckt werden, wurde eine Abdeckungsmatrix erstellt (Tabelle 10.5). In dieser Abdeckungsmatrix ist zu erkennen,

10.2 Testvorgehen beim ASG

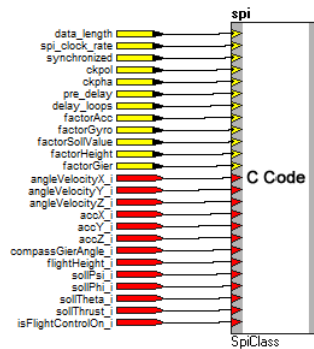


Abbildung 10.25: SPI-Schnittstelle innerhalb des CAMEL-View-Modells für den HIL-Test

dass jede Anforderung an das ASG durch mindestens einen Testfall abgedeckt wird. Die Anforderung FM013 beinhaltet Funktionen des AEVV-Systems, so dass Testfälle für das AEVV-System entworfen wurden, wodurch auch diese Anforderung vollständig abgedeckt wird.

Tabelle 10.5: Testabdeckung ASG

Testfall	FM001	FM002	FM003	FM004	FM005	FM006	FM007	FM008	FM009	FM010	FM011	FM012	FM013	FM014	FM015	NM001	NM002	NM003	NM004	NM005
STRS??					+															
STRS??	+	+	+	+	+		+		+	+						+				
STRS??	+	+	+	+	+		+		+	+						+	+			
STRS??	+	+	+	+	+	+	+	+	+		+					+		+		
STRS??						+		+			+							+		
STRS??												+		+	+				+	+
STRS??	+	+	+	+	+		+		+	+						+	+			
STRS??	+	+	+	+	+	+	+	+	+		+					+		+		
STAV??													+							
STAV??													+							
STAV??													+							

Mit dem Testfall STRS?? wird das Einschalten und das Kalibrieren getestet. Die Testfälle STRS?? - STRS?? testen das Starten, das Steuern des Quadropters und das Landen. Die darauf folgenden Testfälle STRS?? und STRS?? testen die Höhenregelung und das Senden der aktuellen Höhe an die Telemetriestation. Mit dem Testfall STRS?? wird das Erkennen eines kritischen Akkustandes getestet. Die Testfälle STRS?? - STRS?? testen das Fliegen verschiedener Manöver.



Aufgrund dessen, dass der Quadrocopter sehr schnell in der Beschleunigung ist und die Turnhalle, in der die Testfälle durchgeführt werden, relativ klein ist, ist das Testen mit maximalen Eingaben in der Fernbedienung nicht möglich.

Zusätzlich zu den Anforderungen an das ASG sollen die Sicherheitsanforderungen durch weitere Testfälle abgedeckt werden. Die Anforderung SR002 fordert jedoch, dass beim Auftreten einer Störung zwischen dem Quadrocopter und der Fernbedienung, der Quadrocopter diese Störung erkennt, die Sollwerte der Winkel Theta, Phi und Psi auf Null setzt und die Höhe hält. Da ein Testfall, der diese Anforderung abdecken würde, bei der Durchführung zu risikoreich ist, wird diese Anforderung nur eingeschränkt getestet. Das Halten der Höhe bei einer Störung wird deshalb nicht getestet. In Tabelle 10.6 ist zu sehen, dass die Sicherheitsanforderung SR002 nicht abgedeckt wird. Von den Anforderungen an das ASG und den Sicherheitsanforderungen konnten insgesamt 95.5% der Anforderungen abgedeckt werden.

Tabelle 10.6: Testabdeckung Sicherheitsanforderungen

Testfall	SR001	SR002
STRS??	+	

Die Testfälle werden aus Sicherheitsgründen, aber auch aufgrund der gestellten Anforderungen an die Umwelt (siehe Kapitel 6.5), in einer Turnhalle durchgeführt. Bei der Durchführung wird der Quadrocopter von einem Piloten gesteuert. Gleichzeitig werden die Werte, die an die Telemetriestation geschickt werden, von einer weiteren Person beobachtet.

10.2.4 Validierung der Echtzeitanforderungen beim ASG/Dataminer

Aufgrund der echtzeitkritischen Flugalgorithmen, muss garantiert werden, dass alle Deadlines eingehalten werden. Zum Evaluieren der zuvor in Kapitel 6.8 erhobenen Echtzeitanforderungen, wird ein messbasierter Ansatz verwendet. Aus den Anforderungen geht hervor, dass die maximale Ausführungszeit c_{ges} für das ASG die Gesamtperiode des Systems von 2ms nicht überschreiten darf.

In Abbildung 10.26 wird der Ablauf der Messung dargestellt. Zunächst wird der vollständige Programmcode auf die Hardware gespielt. Zum Messen der Ausführungszeit wird vor der Ausführung des zu messenden Codesegments ein Debugpin der Hardware auf High gesetzt. Anschließend wird der Programmcode ausgeführt. Nach dem Ausführen des Programmcodes wird der Debugpin zurück auf Low gesetzt. Mit Hilfe eines Oszilloskops wird die Dauer, die der Debugpin auf High gesetzt wurde, gemessen. Die Dauer gibt die Ausführungszeit des ausgeführten Programmcodes an.

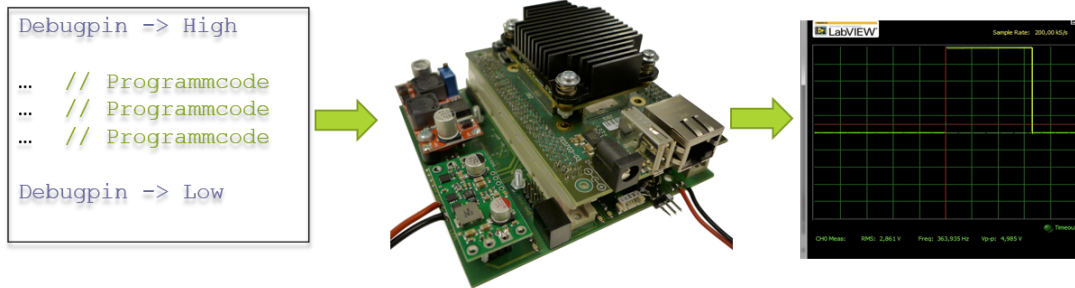


Abbildung 10.26: Ablauf: Evaluierung der Echtzeitanforderungen

Da im Code des ASGs wenige Verzweigungen oder Schleifen vorhanden sind, wurde händisch der längste Ausführungsweg ermittelt. Die entsprechenden Ausführungszeiten wurde für die WCET-Analyse mit Hilfe des o.g. Messsystems ermittelt und die Ergebnisse sind in Tabelle 10.7 dargestellt.

Bezeichnung	Ausführungszeit
bramWrite	$38\mu s$
altitudeController	$6.2\mu s$
attitudeController	$10.5\mu s$
calculateMotorValues	$37\mu s$
setpointProcessing	$19.5\mu s$
sensorProcessing	$10\mu s$
ASG-Gesamt	$121.2\mu s$

Tabelle 10.7: Messungen der Ausführungszeiten

Es ergibt sich für das ASG eine gesamte Ausführungszeit von $c_{ges} = 121.2\mu s$. Diese liegt unter den maximal erlaubten $2ms$, wodurch die Echtzeitanforderungen erfüllt werden.

Mit der selben Methode werden die Ausführungszeiten des Dataminer Sourcecodes ermittelt (siehe Kapitel 9.3.8). Diese liegen ebenfalls unterhalb der $2ms$ Deadline und genügen somit den gestellten Anforderungen.



10.3 Testvorgehen beim AEVV

Beim Testen des AEVV-System kann auf das Standard Vorgehen bei Softwaretests zurückgegriffen werden. Zunächst werden die einzelnen Komponenten einem Komponententest unterzogen, um zu testen, ob diese isoliert für sich alleine korrekt funktionieren. Anschließend werden die einzelnen Komponenten schrittweise integriert und deren Schnittstellen einem Integrationstest unterzogen. Nach erfolgreichem Abschluss der Integrationstests und der Integration zum Gesamtsystem kann dieses im Gesamtsystemtest getestet werden.

In Kapitel 7.2.3 wurde analysiert, aus welchen Komponenten das AEVV-System besteht. Aus diesem Architekturentwurf wird schnell deutlich, dass den Komponenten Object Detection und Gimbal Controller eine besondere Bedeutung zukommt. Dies sind die Komponenten, bei denen die meiste Implementierungsarbeit vorgenommen werden muss. Daher ist es wichtig, diese Komponenten besonders sorgfältig zu testen, da dort das höchste Fehlerpotential vorhanden ist.

Die komplette Entwicklung des AEVV-Systems erfolgt in C++. Es ist möglich, die einzelnen Komponenten bereits frühzeitig auf einem Entwicklungs-PC zu testen.

Jeder Testfall wird im Testdokument festgehalten. Hierbei sind für jeden Testfall folgende Informationen vorhanden:

- **Nr.:** Jedem Testfall wird eine eindeutige Identifikation zugewiesen (z.B. KTAV001).
- **Bezeichnung:** Jeder Testfall ist durch einen Bezeichner beschrieben (z.B. Testen des Verbindungsaufbaus des Gimbal Controllers.).
- **Bezug zu Anforderung:** Hier wird festgehalten, welche Anforderungen mit diesem Testfall validiert werden soll.
- **Komponente:** Hier wird die zu testende Komponente spezifiziert.
- **Vorbedingung:** Hier werden Vorbedingungen definiert, die vor dem Ausführen der Testfälle erfüllt sein müssen.
- **Durchführung:** Hier wird beschrieben, was wie zu testen ist.
- **Erwartetes Ergebnis:** Hier wird das erwartete Ergebnis des Testfalls spezifiziert. Beispielsweise die erwarteten Ausgabewerte.

Zur Dokumentation der Durchführung der Testfälle wird eine Exceltabelle verwendet. In dieser wird festgehalten, ob der Testfall bestanden wurde oder nicht. Diese Tabelle soll einen schnellen Überblick gewähren, welche Testfälle bereits bestanden wurden, um hieraus Rückschlüsse auf den derzeitigen Entwicklungsstand des Projektes ziehen zu können. Auch nicht bestandene Testfälle müssen dort festgehalten werden, um nach Beheben des Fehlers erneut durchgeführt werden zu können. Zudem wird in der Testdatenbank der Pfad zu den



Testquellen angegeben, um die Durchführung der Tests nachhalten zu können und diese zum Zwecke der erneuten Durchführung schnell zu finden.

Zusätzlich zum Eintrag in die Testdatenbank wird für jeden Testfall ein Protokoll angelegt, welches folgende Einträge enthält.

- **Protokollant:** Der Name der Person, die den Test durchgeführt hat.
- **Testfallnummer:** Jedem Testfall wird eine eindeutige Identifikation zugewiesen (z.B. KTAV001).
- **Status:** Status des Testfall bestanden oder nicht bestanden.
- **Datum:** Das Datum der Durchführung.
- **Bezeichnung:** Der Bezeichner des Testfalls.
- **Bezug zu Anforderung:** Hier wird festgehalten, welche Anforderungen mit diesem Testfall validiert werden soll.
- **Vorbedingung:** Hier werden Vorbedingungen definiert, die vor dem Ausführen der Testfälle erfüllt sein müssen.
- **Durchführung:** Hier wird beschrieben, was wie getestet wurde.
- **Erwartetes Ergebnis:** Hier wird das erwartete Ergebnis des Testfalls spezifiziert. Beispielsweise die erwarteten Ausgabewerte.
- **Tatsächliches Ergebnis:** Hier wird beschrieben, welches Ergebnis der Test lieferte. War das Verhalten wie erwartet oder abweichend?

10.3.1 Komponententests

An dieser Stelle werden die Komponententests des AEVV-System spezifiziert. Für jede Komponente wird angegeben, was genau bei der jeweiligen Komponente im Rahmen eines Komponententests zu testen ist. Zuvor wurden bereits alle Komponenten durch die jeweiligen Entwickler einem Unittest unterzogen, um ihre grundlegende Funktionalität sicherzustellen. Die Komponententests sollen nun sicherstellen, dass die Komponenten ihrer Spezifikation entsprechen.

Hierbei soll, sofern es die Art des Testfalles ermöglicht, das Unit Test Framework Boost [Fra] verwendet. Dies ist hilfreich, um Tests erneut durchführen zu können, falls durch gefundene Fehler Änderungen am Quellcode der Komponenten des AEVV-Systems vorgenommen werden müssen. Zudem können so alle Tests von zentraler Stelle aus gestartet werden, im Falle dieses Projektes aus dem Ordner *sw/aeev/build* heraus.



Als Maß für die Testabdeckung soll im Rahmen der Komponententests die Anweisungsüberdeckung verwendet werden. Hierzu kann das Werkzeug LCOV [LCO] verwendet werden. Nachdem die Testfälle als ausführbarer C++-Code vorliegen soll LCOV verwendet werden, um festzustellen, welche Anweisungen im Code noch nicht erreicht wurden. Dies kann anschließend analysiert werden, um weitere Testfälle zu schreiben. Ziel ist es, im Rahmen der Komponententests jede Anweisung im Code mindestens einmal zu durchlaufen. Die Auswertung hierzu findet sich in Kapitel 10.4.2.

10.3.1.1 Objekterkennung Die Objekterkennung besteht, wie in Kapitel Kapitel 3.3 definiert, aus mehreren Verarbeitungstufen. Diese Stufen werden zunächst im Verbund getestet. Erst wenn während des Testens Probleme identifiziert werden, ist es nötig, diese Stufen einzeln zu testen, um diese genauer eingrenzen zu können. Zunächst soll es aber genügen, die Objekterkennung als geschlossene Komponente zu betrachten.

Das Testen der Objekterkennung erfolgt in mehreren Stufen. Zunächst soll anhand vorgefertigter Bilder, welche einen Ball mit geringem Durchmesser in bekannter Position enthalten, getestet werden, ob die Objekterkennung die Position dieses zusammenhängenden Bereiches korrekt erkennt. Zunächst wird dies im Testfall KTAV?? mit weißem Hintergrund getestet.

In einem zweiten Schritt wird in Testfall KTAV?? dasselbe Vorgehen mit einer Hintergrundfarbe durchgeführt, die dem Szenario, welches in Kapitel 4.2 definiert wurde, angemessen ist. Hierdurch soll sichergestellt werden, dass die Objekterkennung auch bei farbigem Hintergrund in der Lage ist, den roten Bereich zu erkennen. Beide Testfälle umfassen mehrere Durchläufe mit unterschiedlichen Pixelpositionen.

Nachdem mit kleinen roten Bereichen getestet wurde, wird in Testfall KTAV?? die Erkennung des Schwerpunktes von größeren roten Bereichen getestet. Dies soll validieren, dass die Objekterkennung in der Lage ist, den Mittelpunkt zusammenhängender roter Bereiche zu ermitteln, die Werten entsprechen, die dem Szenario angemessen sind.

Anschließend wird im Testfall KTAV?? anhand konkreter mit der Kamera aufgenommener Bilder erneut die Erkennung der korrekten Position des roten Objektes getestet. Hierdurch soll getestet werden, dass die Objekterkennung auch mit Bildern, wie sie später im Livebetrieb vorkommen, korrekt arbeitet.

Nach der Überprüfung des Algorithmus anhand einzelner Bilder werden in den Testfällen KTAV?? bewegte Bilder verwendet, welche zuvor mit der Kamera aufgenommen wurden. Nach erfolgreichem Abschluss dieser Tests wird die Objekterkennung in den Testfällen KTAV??, KTAV?? und KTAV?? im Live-Betrieb getestet, zunächst mit unbewegtem Objekt, danach mit bewegtem Objekt. Diese Tests werden zweimal durchgeführt, einmal mit fest montierter Kamera, ein weiteres Mal mit nicht festmontierter Kamera. Hierbei wird die Kamera am Gimbal befestigt und dessen Selbststilisierung aktiviert. Durch den Tester ist



dann ein stabiles Flugverhalten zu simulieren. Dies dient dazu, um bereits an dieser Stelle eventuelle Probleme mit der Objekterkennung bei bewegter Kamera zu identifizieren.

Integriert ins AEVV System wird die Objekterkennung durch den AEVV-Controller de-/ und aktiviert. Dies wird in den Testfällen KTAV?? und KTAV?? getestet.

10.3.1.2 Testdurchführung Objekterkennung Damit die oben genannten Tests möglichst einfach und automatisiert durchgeführt werden konnten, wurde ein Skript erstellt welches die Objekterkennung im Terminal eines Linux System einzeln ausführt. Dies hat den Vorteil, dass die Bilder und Videos zur Testdurchführung einfach geladen und der Objekterkennung bereitgestellt werden können. Dadurch war es möglich, realistische Tests durchzuführen, die sich nahe am echten Betrieb des Systems befinden. Somit musste die Objekterkennung nicht für die Testfälle umgeschrieben werden, was nötig gewesen wäre, da der Streaming Server bzw. GStreamer hochintegriert in der Objekterkennung ist und die Initialisierung sowie die Bereitstellung eines Frames ansonsten stark verändert werden müsste.

Das Skript erstellt mithilfe des Programmes v4l2loopback⁵² eine virtuelle Kamera, die die oben genannten Bilder und Videos der Objektverfolgung bereitstellt. Diese Testbilder werden durch GStreamer in die virtuelle Kamera geladen. Aus Sicht der Objektverfolgung ist diese Kamera eine normale Kamera und gleichwertig wie die Mobius Kamera. Die Objekterkennung lässt sich unabhängig von dem AEVV-Controller starten, mit einer dazu geschriebenen Klasse, die weiterhin viele Einstellmöglichkeiten bietet, wie die zu erkennende Farbe, oder die Möglichkeit den DebugFrame an-, oder abzuschalten. Somit wurde ein Skript erstellt, dass alle oben genannten Testfälle automatisiert ausführt und erfolgreich abschliesst. Lediglich die Live Testfälle KTAV??, KTAV?? und KTAV?? mussten per Hand getestet werden, was mithilfe der Objekterkennung und der dazugehörigen ausführbaren Datei leicht auf dem Quadropter möglich war.

10.3.1.3 Gimbal Controller Der Gimbal Controller bekommt über das Interface Iposition die aktuelle Position des zu erkennenden Objektes sowie die Auflösung des Videostreams. Dieser ermittelt nun den korrekten Stellwert für die Winkel des Gimbals.

Zunächst ist die korrekte Ansteuerung des Gimbals durch die Software des Gimbalcontrollers zu testen. Zuerst ist der Verbindungsauf und Verbindungsabbau zur Hardware des Gimbals zu testen. Dies geschieht in den Testfällen KTAV?? und KTAV??. Im Testfall KTAV?? wird das Senden von Stellwerten an den Gimbal Controller getestet.

Nachdem durch die obigen Testfälle sichergestellt ist, dass die Ansteuerung des Gimbals wie erwartet funktioniert, wird der Gimbalcontroller auf seine eigentliche Funktionalität getestet. Dieser beinhaltet 3 Modi, welche zu testen sind. Zum einen den Tracking Mode, welcher einen Regler für die Gier- und Nickachse des Gimbals darstellt. Dieser ermittelt

⁵²<https://github.com/umlaeute/v4l2loopback>



die neuen Stellwerte für den Gimbal je nach Abweichung des zu verfolgenden Objektes vom Bildmittelpunkt. Der Tracking Mode wird aktiviert, sobald sich die von der Objekterkennung ermittelte Position in X-Achse und Y-Achse im positiven Bereich befinden.

Die Übergabe der aktuellen Werte für die X-Position und Y-Position wird in den Testfällen KTAV?? und KTAV?? getestet.

Der Searching Mode wird aktiviert, sobald die X- oder Y-Position des zu verfolgenden Objektes den Wert -1 hat. Dies ist der Wert, welchen die Objekterkennung liefert, sofern sie das zu verfolgende Objekt nicht mehr im derzeitigen Bildausschnitt finden kann. Der Gimbal beginnt dann den Arbeitsbereich abzusuchen. Hierbei ist insbesondere zu testen, ob der Wechsel zwischen den zur Verfügung gestellten Modis korrekt funktioniert. Dies geschieht in den Testfällen KTAV??m KTAV??, KTAV??, KTAV?? und KTAV??.

Der Fixed Mode muss den Gimbal auf eine feste Neutralposition stellen, dies wird im Testfall KTAV?? getestet.

Im Rahmen der Komponententests ist für den Tracking Mode zu ermitteln, ob der Gimbal Controller die korrekten Stellwerte liefert, nachdem er die Position und Auflösung eines Bildes mitgeteilt bekommen hat. Dies wird in Testfall KTAV?? durchgeführt. Bei diesen Testfällen ist vorab vom Tester zu berechnen, welche Stellwerte der Gimbal abhängig von seiner derzeitigen Position und der Abweichung des Objektes(x- und y-Position bezogen auf den Bildmittelpunkt) berechnen sollte.

Im Rahmen des Komponententests kann an dieser Stelle lediglich überprüft werden, ob die berechneten Stellwerte, mit den vom Entwickler erwarteten Stellwerten übereinstimmen. Ob die Regelung des Gimbals den Anforderungen an die Objektverfolgung tatsächlich genügt, kann erst im Rahmen der auf die Komponententests folgenden Integrationstests erfolgen. Insbesondere kommt der Integration der Komponenten Objekterkennung und Gimbal Controller daher eine besondere Bedeutung zu.

Der Searching Mode kann hingegen schon im Rahmen des Komponententest hinreichend getestet werden. Der Gimbal sucht hierbei den Arbeitsbereich ab. Dies wird im Testfall KTAV?? getestet.

Zusätzlich wird im Testfall KTAV?? getestet, ob das Deaktivieren/Aktivieren der Motoren des Gimbals korrekt funktioniert.

10.3.1.4 Streaming Server Der Streaming Server stellt das Kamerabild zur Verfügung, welches von der Kamera aufgenommen wird. Das aufgenommene Kamerabild wird hierbei in 2 Videostreams aufgeteilt und der Objekterkennung sowie der Telemtriestation zur Verfügung gestellt.

Zunächst ist die Initialisierung des Streaming Servers zu testen. Dies wird im Testfall KTAV?? zunächst anhand einer korrekten Konfiguration getestet. Im Testfall KTAV?? wird die Fehlerbehandlung der Konfiguration anhand einer fehlerhaften Konfiguration getestet.

Nun wird in den Testfällen KTAV??,KTAV??,KTAV??,KTAV?? und KTAV?? das Ein-/Ausschalten der einzelnen Streams des Streaming Servers getestet. Zunächst wird hierbei ein vorgefertigter Videostream als Eingangssignal genutzt.

Anschließend wird in den Testfällen KTAV??, KTAV??, KTAV??, KTAV?? und KTAV?? das Ein-/Ausschalten der einzelnen Streams erneut getestet. Allerdings dient diesmal ein von der angeschlossenen Kamera gelieferter Stream als Eingangssignal.

Abschließend wird im Testfall KTAV?? noch das Beenden des Streaming Servers getestet.

10.3.1.5 AEVVController Der AEVV-Controller dient als Konfigurationsmanager für alle anderen im AEVV-System verwendeten Komponenten. Desweiteren übernimmt er die De-/Aktivierung der Komponenten und stellt ihnen die vom ASG in den Block RAM geschriebenen Avionic Daten, wie die aktuelle Flughöhe und den Akkustand, zur Verfügung. Der AEVV Controller verwendet das Interface BramDriver, um das Auslesen aus dem Block Ram vorzunehmen. Dies wird im Testfall KTAV?? getestet. Mittels der Klasse ConfigurationManager werden die Konfigurationsparameter für das AEVV System aus einer XML-Datei ausgelesen. Dies wird in den Testfällen KTAV?? und KTAV?? getestet. Die ausgelesenen Werte aus dem Block Ram werden mittels der Klasse DataStorage gespeichert. Das Speichern wird im Testfall KTAV?? getestet.

10.3.2 Integrationstests des AEVV-Systems

Die Integrationstests haben zum Ziel, die Schnittstellen der einzelnen Komponenten zu testen. Es zu überprüfen, ob die Komponenten in der Lage sind, korrekt miteinander zu kommunizieren und ob sie im Zusammenspiel funktionieren. In Abbildung 10.27 ist noch einmal der Aufbau und die Kommunikation zwischen den Komponenten des AEVV-Systems dargestellt.

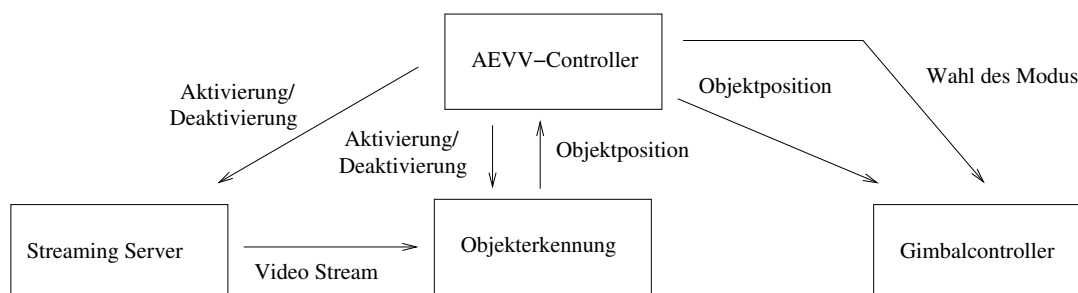


Abbildung 10.27: Aufbau des AEVV-Systems

Im Rahmen der Integrationstests des AEVV-Systems werden die Komponenten des Systems schrittweise zum Gesamtsystem integriert. Jeder Integrationstest deckt verschiedene Anforderungen an das Gesamtsystem ab. Ziel ist es, durch die Integrationstest alle Anforderungen abzudecken. Die Übersicht hierzu findet sich in Kapitel 10.4.2.



Dem AEVV-Controller kommt eine zentrale Rolle zu, da er die Aktivierung und Deaktivierung der Komponenten übernimmt. Zudem wird die von der Objekterkennung ermittelte Objektposition an den Gimbalcontroller weitergeleitet.

Die Funktionalität des AEVV-Controllers, des Gimbal Controllers und der Objekterkennung kann erstmals im Rahmen der Integrationstests im Zusammenspiel getestet werden. Insbesondere beim Gimbalcontroller wird erst hier deutlich werden, ob dieser den Anforderungen an das Gesamtsystem genügt. Nur im Zusammenspiel mit realen Videodaten, welche von der Objekterkennung ausgewertet werden, ist es möglich zu überprüfen, ob das Nachführen der Kamera durch den Gimbal Controller den Anforderungen an das Szenario genügt.

Im Folgenden werden die einzelnen Integrationstestfälle beschrieben.

Im Testfall ITAV?? wird das Zusammenspiel von Objekterkennung und des Streaming Servers anhand aufgenommener Videodaten getestet. Hier soll validiert werden, dass die Objekterkennung die Position des zu verfolgenden Objektes korrekt ermittelt, wenn ihr die Videodaten durch den Streaming Server zur Verfügung gestellt werden.

Hiernach wird in Testfall ITAV?? der Verbindungsaufbau von AEVV Controller zur Telemetriestaion getestet. Anschließend wird in ITAV?? die Datenübertragung von AEVV Controller zur Telemetriestation getestet.

Testfall ITAV?? dient dazu, den AEVV Controller im Zusammenspiel mit der Objekterkennung zu testen. Hier wird zunächst getestet, ob das Aktivieren und Deaktivieren der Objekterkennung funktioniert.

Desweiteren wird in Testfall ITAV?? überprüft, ob das Starten des Streaming Servers durch den AEVV Controller funktioniert.

Es ist nun im Testfall ITAV?? zu validieren, ob der AEVV Controller den Fixed Mode des Gimbal Controllers aktivieren kann.

In Testfall ITAV?? wird nun überprüft, ob der AEVV Controller die Position des zu verfolgenden Objektes korrekt dem Gimbal Controller zur Verfügung stellt.

Im Testfall ITAV?? wird nun das Verfolgen eines Objektes durch Objektverfolgung und Gimbal Controller getestet. Hierzu werden alle beteiligten Komponenten integriert. Hierbei ist insbesondere das Verhalten des Gimbalcontrollers zu testen. Da erst an dieser Stelle validiert werden kann, ob die Regelung des Gimbal stabil genug ist, um ein störungsfreies Verfolgen des Objektes zu gewährleisten, kommt diesem Test eine besondere Bedeutung zu. Im Sichtfeld der Kamera ist nun das zu verfolgende Objekt zu bewegen, um die Verfolgung durch die Objekterkennung und das Nachjustieren des Gimbals durch den Gimbalcontroller zu testen. Dies geschieht zunächst mit fest in der Luft fixierten Quadrokooper.

Anschließend wird in Testfall ITAV?? nun selbiges Vorgehen wiederholt, nur wird der Quadrokooper hier vom Tester in der Luft festgehalten und ein stabiles Flugverhalten simuliert. Dies dient dazu, um vorab Probleme zu identifizieren, die durch Bewegungen des Quadrokoopers im Flug entstehen.

Anschließend wird in den Testfällen ITAV?? und ITAV?? der Searching Mode des Gimbalcontrollers getestet. Hierzu wird das Objekt außerhalb des derzeitigen Sichtfeldes platziert und anschließend der Searching Mode des Gimbalcontrollers aktiviert. Der Gimbal fährt nun den abzusuchenden Bereich ab und muss im Zusammenspiel mit der Objekterkennung das zu verfolgende Objekt finden und dann die Verfolgung aufnehmen. Diese Tests werden ebenfalls bei fixiertem und in der Luft befindlichem Quadrocopter durchgeführt.

10.3.3 Systemtests

Es ist nötig, einige Funktionalitäten des AEVV-Systems in 2 Phasen zu testen. Bewegungen des Multikopters können nicht unerhebliche Störeinflüsse auf die Objekterkennung und Objektverfolgung haben. Daher ist es nötig, diese Funktionalitäten zunächst zu testen, ohne dass sich der Multikopter bewegt. Dies ist hilfreich, um eventuelle vorhandene Fehler genauer eingrenzen zu können. Deshalb werden einige Testfälle zunächst durchgeführt, während der Multikopter in der Luft fixiert ist. Anschließend werden dieselben Testfälle erneut durchgeführt, während sich der Multikopter im Flug befindet. Als Maß für die Testabdeckung soll bei den Systemtests auf die Abdeckung der Anforderungen an das Gesamtsystem zurückgegriffen werden. Die Übersicht hierzu findet sich in Kapitel 10.4.2. Die Testfälle STAV??, STAV??, STAV??, STAV?? und STAV?? überprüfen die korrekte Funktionsweise der Objekterkennung. Diese Testfälle werden nach erfolgreichem Abschluss erneut in der Luft durchgeführt. Während der Entwicklung können diese Tests bereits an einem Entwickler-PC durchgeführt werden, um die Grundfunktionalität des Systems sicherzustellen. Für die Abnahme müssen diese Tests auf der Zielplattform wiederholt werden.

Hiernach wird das AEVV-System getestet, während es sich im Flug befindet. Die Testfälle STAV??, STAV?? und STAV?? testen zunächst die Aktivierung des AEVV-Systems und das Senden des Kamerabildes an die Bodenstation. Die Testfälle STAV??, STAV??, STAV??, STAV?? und STAV?? testen die Objekterkennung und Verfolgung während des Fluges. Die Testfälle STAV?? und STAV?? testen die Übertragung der Bilddaten an die Bodenstation.

Im Testfall STAV?? wird die Positioniereinheit auf ihre Funktionalität getestet. Zusätzlich wird in den Testfällen STRS??, STRS??, STRS?? des ASGs sichergestellt, dass die Peripherie des AEVV-Systems keine negativen Auswirkungen auf das Flugverhalten des Multikopters hat.

10.4 Testergebnisse

In diesem Abschnitt werden die Testergebnisse von den durchgeführten Tests beschrieben.



10.4.1 Testergebnisse ASG

In diesem Abschnitt werden die Testergebnisse vom ASG vorgestellt. Es werden zuerst die Testergebnisse der Komponenten- und Integrationstests und anschließend die Testergebnisse der Systemtests beschrieben.

10.4.1.1 Komponenten- und Integrationstestergebnisse Jeder Komponenten- und Integrationstest wurde manuell in der CAMEL-View-Umgebung durchgeführt. Mit den durchgeführten Testfällen wurde getestet, ob jeder Block korrekt funktioniert. Zusätzlich wurde getestet, ob die Blöcke im Zusammenspiel funktionieren und ob bestimmte Ausgaben richtig begrenzt werden. Dabei konnte jeder Testfall erfolgreich durchgeführt werden. Mit dem Durchführen der Testfälle konnte, sowohl mit den Komponententests, als auch mit den Integrationstests, eine vollständige Zweigabdeckung aller Komponenten erreicht werden. Bei der späteren MIL-Testdurchführung wurden Fehler im *AttitudeController-stellgroessePsi* und im *HeightController* festgestellt. Daraufhin wurden die Testfälle von diesen beiden Teilmodellen angepasst, damit eine vollständige Zweigabdeckung weiterhin erreicht wird, und anschließend erneut erfolgreich durchgeführt. Mit der vollständigen Zweigabdeckung werden viele Fehlerquellen abgedeckt. Jedoch kann es durch ungünstige Eingabekombinationen dennoch zu Fehlern kommen. Um auch diese auszuschließen, wären andere Abdeckungskriterien, z.B. Pfadabdeckung, besser geeignet. Da die Testfälle aber alle manuell durchgeführt wurden, wäre der Aufwand einer Pfadabdeckung zu umfangreich geworden.

10.4.1.2 MIL-Test Die MIL-Tests wurden zuerst mit direkten Eingaben in CAMEL-View-Umgebung durchgeführt. Dabei sind folgende Fehler aufgetreten:

- Höhenregler: Beim Aktivieren des Höhenreglers hat sich der Höhenregler nicht auf die Sollhöhe eingeregelt. Dies lag daran, dass sich der Höhenregler nicht auf die Höhe sondern auf die Steiggeschwindigkeit eingeregelt hat. Dieser Fehler wurde erfolgreich behoben.
- Gieren: Der Quadrocopter stürzte in der CamelView-Simulation beim Gieren mit einer sehr geringen Winkelgeschwindigkeit ab. Dieser Fehler konnte auf eine fehlerhafte Winkelumrechnung der Entwicklungsumgebung zurückgeführt werden. Zum Testen des Gierens wurden die Rückführungen der Winkel Theta und Phi aus dem Physikalischen Modell gekappt, so dass das Gieren isoliert von den anderen beiden Reglern getestet werden konnte. Die isoliert durchgeführten Testfälle konnten erfolgreich getestet werden.
- Psi-Regelung: Der Winkel Psi veränderte sich bei Veränderungen der Winkel Theta und Phi leicht. Jedoch regelte sich der Psi-Winkel nicht wieder zurück auf den

Sollwinkel. Stattdessen blieb der Regelfehler konstant. Bei wiederholenden Winkeländerungen von Theta und Phi summierte sich der Regelfehler von Psi auf, so dass sich dieser in den Test um bis zu 17° veränderte.

Nachdem die Fehler behoben wurden, sind erneut Komponenten- und Integrationstests mit den entsprechenden Komponenten/Teilsystemen sowie die MIL-Tests erfolgreich durchgeführt worden. Aufgrund der eingeschränkten Eingabemöglichkeiten konnten die Testfälle STMIL??-STMIL??, die die Szenarien wie z.B. das „Achten“ fliegen beschreiben, nicht durchgeführt werden.

Beim zweiten MIL-Testdurchlauf, bei dem die Eingaben mit der Fernbedienung eingegeben wurden, schwankten die Werte stärker um die Sollwerte als beim ersten MIL-Testdurchlauf (ohne Fernbedienung). In Abbildung 10.28 werden die Winkelverläufe der Winkel Theta, Phi und Psi vom Testfall STMIL?? dargestellt.

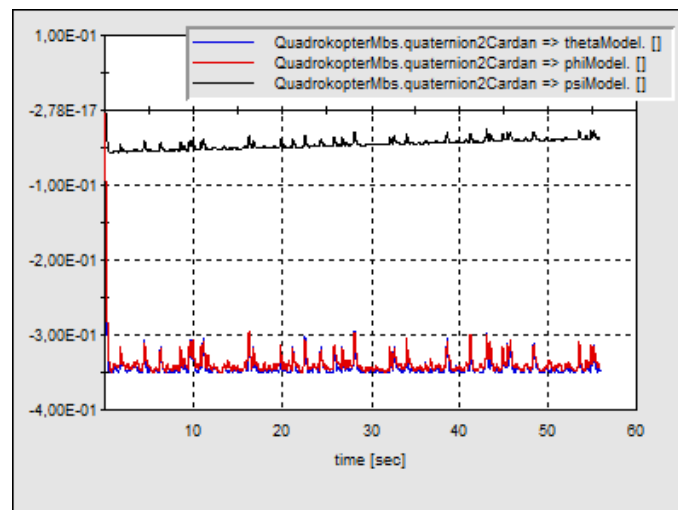


Abbildung 10.28: Verlauf der Winkel ohne Filterung der Nullen (Testfall STMIL??)

Die starken Schwankungen um den Sollwert sind dadurch zu erklären, dass über die SPI-Schnittstelle gelegentlich eine Null anstelle des Sollwerts geschickt wurde. Um dieses Problem zu umgehen, wurden die gesendeten Nullen im CAMEL-View Modell herausgefiltert. Anschließend wurden die MIL-Testfälle alle erneut und erfolgreich durchgeführt. Die Schwankungen waren dabei deutlich geringer (Abb. 10.29).

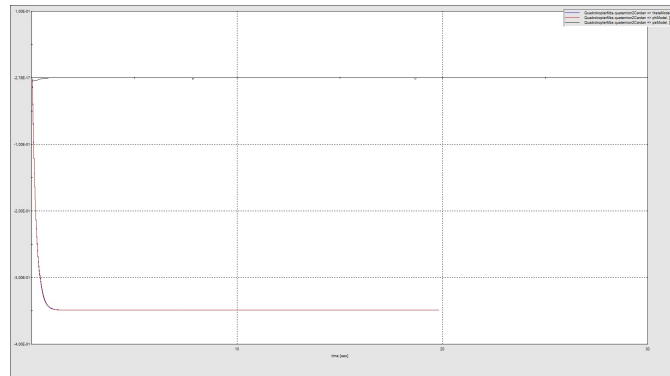


Abbildung 10.29: Verlauf der Winkel mit Filterung der Nullen (Testfall STMIL??)

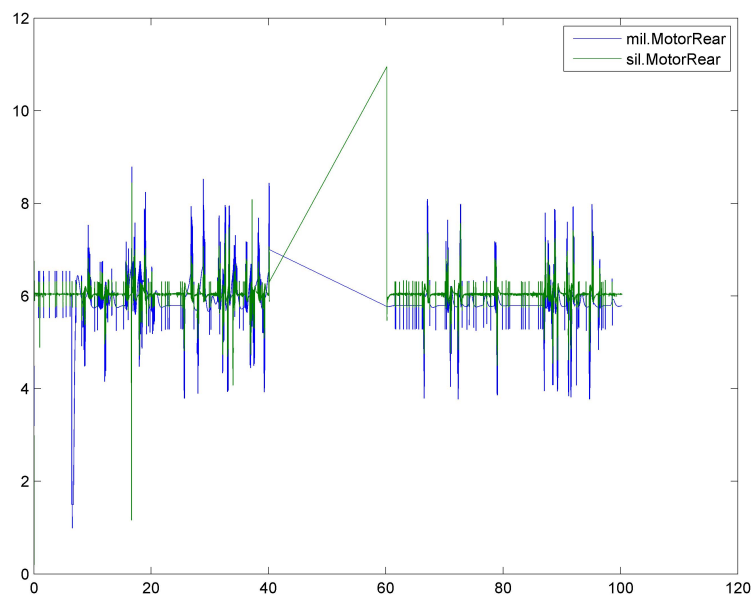


Abbildung 10.30: SIL-Test mit 0.002s als Zeitschritt

10.4.1.3 SIL-Test Bei der Ausführung des SIL-Tests sind mehrere Abweichungen aufgetreten. Zum einen stimmte die Schrittweite der Berechnungen in der Simulation mit denen aus der Ausgabe nicht überein. Dieses Problem konnte leicht durch eine Einstellung in CAMEL-View behoben werden.

Darüber hinaus kann die Testrig-Box die nötigen Daten nicht mit der gewünschten Regel-frequenz von 500Hz ausgeben. Dadurch entstehen Artefakte und das Modell funktioniert nicht richtig (siehe Abbildung 10.32).

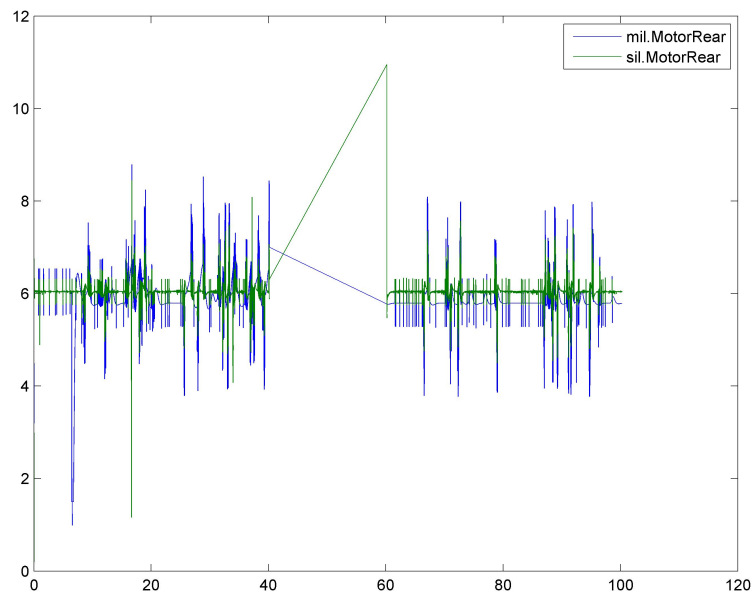


Abbildung 10.31: SIL-Test mit 0.002s als Zeitschritt

Ein weiteres Problem ist, dass die Ausgabefunktion im Wrapper an einer falschen Stelle stand. Dies führte dazu, dass die Eingabewerte zwar an der richtigen Stelle standen, die Ausgabewerte allerdings einen Zeitschritt zu früh ausgegeben werden. Dies führte zu einem Offset bei den Ausgabewerten. Durch ein Verschieben der Ausgabefunktion innerhalb des Codes des Wrappers wurden die Ausgabewerte an der richtigen Stelle ausgegeben.

Eine Schwierigkeit ist beim SIL-Test zum einen, dass die im MIL-Test berechneten und die ausgegebenen Zeitschritte unterschiedlich sind, sodass der SIL-Test nicht korrekt durchgeführt werden konnte. Aus diesem Grund wurden einige einfache Tests wiederholt. Als nächstes wurden die mit der Testrig-Box generierten MIL-Test-Ergebnisse verwendet. Leider kann die Box nur Zeitschritte ausgeben, die größer sind als $\Delta t < 0,1s$. Somit sind in diesem Falle Abweichungen vom eigentlichen Modell zu erwarten.

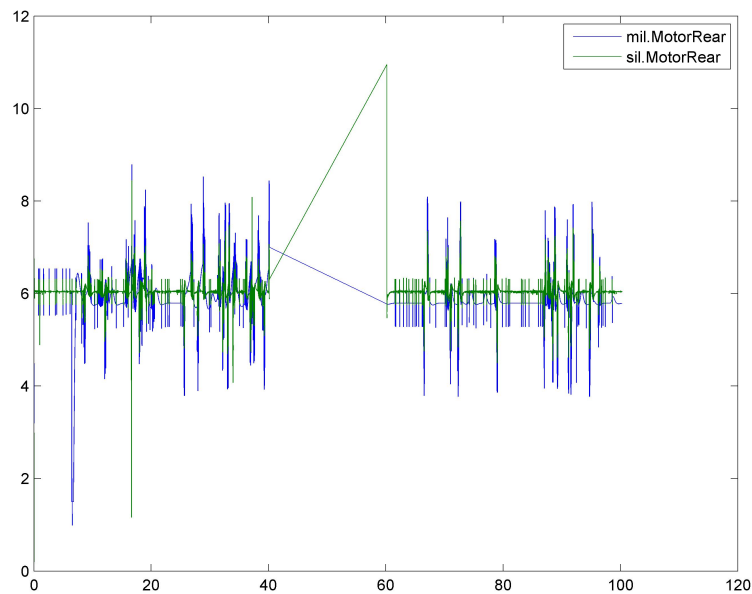


Abbildung 10.32: SIL-Test mit 0.002s als Zeitschritt

Auf Grund des oben beschriebenen Sachverhalts und der fortgeschrittenen Zeit wird nur ein MIL-Test mit der CAMEl-View Oberfläche durchgeführt. Dort können Werte für die Fernsteuerung eingegeben werden. Allerdings lässt diese Variante nur einfache Flugrichtungen zu. Als Beispiel dient hier ein Plot des dritten SIL-Tests. Auf den ersten Blick sieht es so aus, als ob sich MIL und SIL gleich verhalten. Abbildung 10.33 Bei genau-

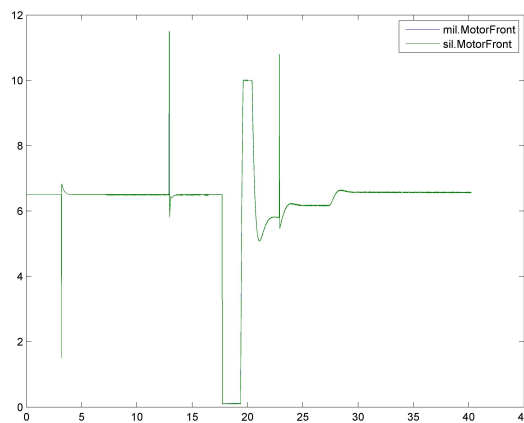


Abbildung 10.33: Stellwerte für den linken Motor

rer Betrachtung stellt man fest, dass die Werte um über 60% voneinander abweichen. In Abbildung 10.34 sind die absoluten und relativen Fehler an Hand von Graphen zu sehen.

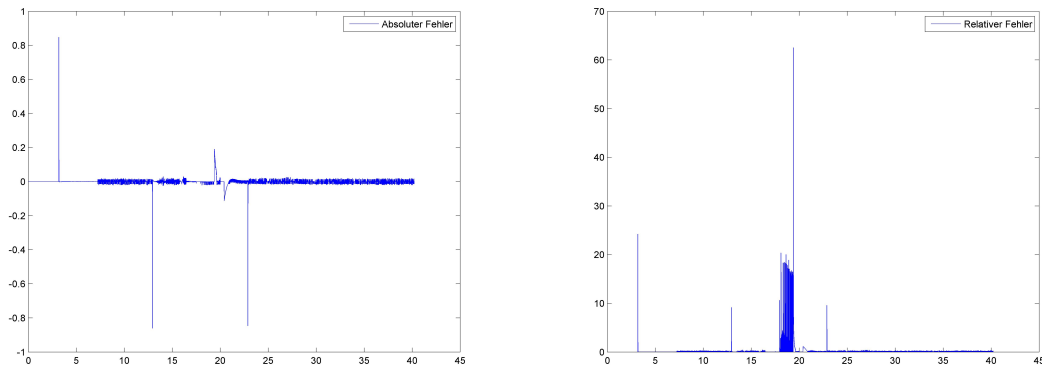


Abbildung 10.34: Absoluter und relativer Fehler

Diese Fehler sind bei allen betrachteten Werten zu beobachten. Daher gelten alle SIL-Tests als nicht bestanden.

10.4.1.4 HIL-Test Der HIL-Test konnte im Rahmen dieses Projekts aufgrund unerwarteter Komplikationen nicht mehr durchgeführt werden. Bei der Portierung des generierten Codes auf die Zielarchitektur traten diverse Probleme auf (s. Kapitel 9.5.1.7 und Kapitel 9.5.1.2). Aus zeitlichen Gründen wurde die Entscheidung getroffen, die Algorithmen der Regler eigenständig, basierend auf dem Modell, in der Programmiersprache C zu implementieren. Ein HIL-Test mit diesem Code liefert aufgrund fehlender Rückschlüsse auf die Ergebnisse der vorherigen Teststufen (MIL und SIL) keinen nennenswerten Mehrwert, weil bei einem modellbasierten Ansatz eine vollständige Äquivalenz zwischen Modell und Code nicht ohne Weiteres gewährleistet werden kann. Es können Abweichungen zwischen den Ergebnissen auftreten, obwohl der Quadrokopter mit beiden Implementierungen ein stabiles Flugverhalten aufweist. Dies liegt folglich an den unterschiedlichen Ansätzen bei der Realisierung der Regleralgorithmen und -parametern.

Des Weiteren ist das HIL-Konzept so gewählt, dass es keine Rückkopplung zum physikalischen Modell gibt (s. Abbildung 10.22). Würde trotzdem ein HIL-Test mit dem selbst implementierten Code durchgeführt werden, kann nicht ohne Weiteres validiert werden, ob dieser ein stabiles Flugverhalten in der Simulation aufweist.

10.4.1.5 Systemtest am realem System Die Systemtestfälle am Quadrokopter wurden in einer Turnhalle durchgeführt, um die in Kapitel 6.5 gestellten Anforderungen an die Umwelt einzuhalten. Alle Testfälle konnten erfolgreich durchgeführt werden. Im folgenden werden die Testergebnisse der Testfälle zusammengefasst:



- **STRS??**: Nach dem Einschalten des Quadropters wurden die Sensorwerte kalibriert und an die Telemetriestation gesendet. Die angezeigten Werte der Winkel waren Null ($\pm 1^\circ$).
- **STRS??, STRS??, STRS??**: Nachdem der Schub erhöht wurde, stieg der Quadropter stabil. Bei Eingaben der Sollwinkel flog der Quadropter entsprechend in die erwartete Richtung und wies dabei ein stabiles Flugverhalten auf. Beim Verringern des Schubs sank der Quadropter stabil zu Boden und landete unversehrt.
- **STRS??**: Der Quadropter flog stabil und der Höhenregler wurde eingeschaltet. Der Quadropter regelte sich auf die erwartete Höhe ein, wobei der Quadropter um bis zu einen Meter um die Sollhöhe schwank.
- **STRS??**: Nach dem Einschalten des Quadropters wurden die Sensorwerte kalibriert und an die Telemetriestation gesendet. Der angezeigte Wert der Höhe war Null ($\pm 0.3m$).
- **STRS??**: Der Quadropter wurde solange geflogen bis der Akku den kritischen Wert unterschritten hat. Es ertönte eine akkustische Warnmeldung.
- **STRS??, STRS??**: Der Quadropter flog verschiedene Manöver. Das Flugverhalten war dabei stabil.
- **STRS??**: Der Quadropter flog verschiedene Manöver, wobei der Höhenregler aktiviert war. Das Flugverhalten war dabei stabil und der Quadropter hat dabei die Sollhöhe gehalten ($\pm 1m$).
- **STRS??**: Der Quadropter steht auf dem Boden und die Verbindung zwischen Fernbedienung und Quadropter wurde unterbrochen. Die Sollwerte der Winkel wurden an die Telemetriestation gesendet und zeigten den Wert Null an.

10.4.2 Testergebnisse AEVV

An dieser Stelle werden die Testergebnisse des AEVV-Systems ausgewertet.

Die erstellten Komponententests wurden bei allen Komponenten erfolgreich durchgeführt. Für den Streaming Server, den AEVV-Controller sowie den Gimbalcontroller wurde das Google Testframework verwendet, um die Testfälle durchzuführen. Anschließend wurde mittels LCOV untersucht, welche Zeilen des Codes durch die durchgeführten Testfälle nicht erreicht wurden. Hierbei handelte es zumeist um durch die Testfälle nicht erreichte Randfälle oder Fehlerbehandlungen. In beiden Fällen wurden die Testfälle verfeinert, um die Anweisungsüberdeckung zu verbessern.

In den Grafiken 10.35 und 10.36 sieht man, dass die Anweisungsüberdeckung im Falle des Gimbalcontrollers und des Streaming-Servers fast vollständig erfüllt ist. Bei den nicht



abgedeckten Anweisungen handelt es sich um die Fehlerbehandlung von Fehlern, welche nicht ohne aufwendige Testtreiber provoziert werden konnten. Daher wurde im Rahmen dieser Projektgruppe darauf verzichtet.

LCOV - code coverage report

Current view: top level - gimbalcontroller		Hit	Total	Coverage
Test: gimbaltest-gimbalcov.info.cleaned	Lines:	268	291	92.1 %
Date: 2015-03-30	Functions:	40	40	100.0 %

Filename	Line Coverage	Functions
Gimbal.cpp	95.4 % (83 / 87)	100.0 % (17 / 17)
Gimbalcontroller.cpp	90.6 % (184 / 203)	100.0 % (22 / 22)
Gimbalcontroller.h	100.0 % (1 / 1)	100.0 % (1 / 1)

Generated by: *LCOV version 1.10*

Abbildung 10.35: Anweisungsüberdeckung des Gimbalcontrollers

LCOV - code coverage report

Current view: top level		Hit	Total	Coverage
Test: streamtest-streamcov.info.cleaned	Lines:	331	348	95.1 %
Date: 2015-03-26	Functions:	45	49	91.8 %

Directory	Line Coverage	Functions
common/include	100.0 % (9 / 9)	80.0 % (4 / 5)
stream-srv/include	100.0 % (10 / 10)	76.9 % (10 / 13)
stream-srv/src	94.8 % (312 / 329)	100.0 % (31 / 31)

Generated by: *LCOV version 1.10*

Abbildung 10.36: Anweisungsüberdeckung des Streaming Servers

Beim AEVV-Controller 10.37 wurde durch LCOV eine schlechtere Anweisungsüberdeckung ermittelt. Dies liegt daran, dass das Testen des BRAMs auf der Zielhardware ausgeführt wurde. Allerdings lassen sich dort ausgeführte C++ Programme nicht mittels LCOV analysieren, da dieses dort nicht ohne weiteres zur Ausführung gebracht werden konnte. Daher tauchen die Testfälle, welche den BRAM testen, nicht in der Anweisungsüberdeckung mit auf.

LCOV - code coverage report

Current view: top level - src/conn		Hit	Total	Coverage
Test: controltest-controlcov.info.cleaned	Lines:	341	367	92.9 %
Date: 2015-03-30	Functions:	56	61	91.8 %

Filename	Line Coverage	Functions
CClient.cpp	63.4 % (45 / 71)	66.7 % (8 / 12)
CDataStorage.cpp	100.0 % (296 / 296)	98.0 % (48 / 49)

Generated by: *LCOV version 1.10*

Abbildung 10.37: Anweisungsüberdeckung des AEVV-Controllers



Die Objekterkennung wurde mittels eines geschriebenen Shellsriptes getestet. Daher war es nicht möglich, eine automatisierte Auswertung der Anweisungsüberdeckung durchzuführen. Da es sich bei der Objekterkennung aber um ein streng sequentielles Programm handelt, konnte die Anweisungsüberdeckung von Hand abgeschätzt werden, diese liegt nach unten abgeschätzt bei 80 Prozent. Bei den nicht erreichten Anweisungen handelt es sich ausschließlich um Fehlerbehandlungen, die auftreten, wenn die Komponente nicht korrekt initialisiert wird. Da sichergestellt ist, dass der AEVV-Controller die Objekterkennung korrekt initialisiert, werden diese Fehlerbehandlungen im realen System nicht zur Ausführung kommen.

Abschließend bleibt festzuhalten, dass keine vollständige Anweisungsüberdeckung erreicht werden konnte. Das Betrachten der Anweisungsüberdeckung erwies sich aber als hilfreich beim Auffinden nicht betrachteter Grenzfälle in den Wertebereichen der Eingangsvariablen sowie dem Auffinden nicht durch die Tests abgedeckter Fehlerbehandlungen. Somit bleibt festzuhalten, dass die Anweisungsüberdeckung als alleiniges Maß für die Testabdeckung durch die Komponententests wenig aufschlussreich ist, bei der Durchführung der selbigen aber ihre Daseinberechtigung hatte.

Die Integrationstests wurden ebenfalls erfolgreich durchgeführt. Hierbei wurden einige Fehler in der Kommunikation zwischen dem AEVV-Controller und dem Gimbalcontroller deutlich. So kam es häufig vor, dass der Suchmodus des Gimbalcontrollers nicht korrekt aktiviert wurde. Dies konnte durch eine Verfeinerung der Kommunikation zwischen diesen Komponenten behoben werden. So war es an dieser Stelle z.B. nötig, die vom AEVV-Controller gesetzten Attribute innerhalb des Gimbals durch Semaphoren zu schützen. Nachdem diese Probleme behoben waren, konnten die restlichen Tests erfolgreich durchgeführt werden. In Tabelle ?? sieht man, dass durch die Integrationstests alle Anforderungen an das System abgedeckt wurden.

Die Systemtests wurden in 2 Stufen durchgeführt. Hierbei war es hilfreich, zu Entwicklungszwecken bereits auf einem Entwicklungs PC die Testfälle KTAV01 bis KTAV07 durchführen zu können. Hierdurch konnte die Funktionalität der Objekterkennung im Zusammenspiel mit AEVV-Controller und Gimbalcontroller bereits frühzeitig getestet werden. Insbesondere bei der Entwicklung des Gimbalcontrollers konnte das System durch diese Tests optimal auf seinen Einsatz im Gesamtsystem vorbereitet werden.

Anschließend wurden die Tests auf der Zielplattform wiederholt. Dort war es zunächst nötig die Auflösung des Videostreams so zu wählen, dass die Zielhardware noch verbleibende Ressourcen für die Berechnungen der Objekterkennung und des Gimbalcontrollers zur Verfügung hat. Diese Framerate auf dem Produktivsystem beträgt 30fps zur Telemetriestation und 15fps zur Objekterkennung. Dies erwies sich als optimaler Wert, um eine Auslastung der CPU zwischen 90 und 100Prozent zu erreichen.

Nachdem hier alle Parameter geeignet gewählt wurden, konnten die Systemtests STAV01 bis STAV07 erfolgreich auf der Zielhardware abgenommen werden.



Die Systemtests STAV08 bis STAV018 wurden während der Testflüge in der Turnhalle abgenommen. Alle Testfälle konnten erfolgreich abgenommen werden. Aus Tabelle ?? ist ersichtlich, dass alle Anforderungen an das AEVV-System durch die abgenommenen Testfälle abgedeckt wurden.

Insgesamt bleibt als Ergebnis der Testphase festzuhalten, dass das System erfolgreich getestet werden konnte und alle zuvor erhobenen Anforderungen erfüllen konnte.

??

10.4.2.1 Validierung der Echtzeitanforderungen des AEVV-Systems An dieser Stelle soll evaluiert werden, ab welcher Mindestdistanz d garantiert werden kann, dass das AEVV-System in der Lage ist, den zu verfolgenden Ball im Zentrum des Kamerabildes zu halten.

Die Echtzeitanforderungen des AEVV-Systems sollen hierbei nach der in Kapitel 6.8 aufgestellten Formel:

$$c_{IC} + c_{Algo} + c_{Gimbal} < d_{aevv} \quad (10.6)$$

validiert werden. Die Einhaltung der Deadline hängt maßgeblich von der Berechnungszeit der Objekterkennung ab. Daher soll die Formel 10.7 verwendet werden, um die Echtzeitanforderungen zu validieren. Bei Einhaltung dieser Formel kann garantiert werden, dass das zu verfolgende Objekt dauerhaft im Zentrum des von der Kamera aufgenommenen Bildes gehalten werden kann.

$$c_{Algo}(d) = \frac{d \cdot \sqrt{2 \cdot (1 - \cos \alpha)}}{|v_B - v_K|} - \frac{d_B}{f_U} - \frac{\alpha}{2 \cdot \omega} - t_{verz} \quad (10.7)$$

Zur Auswertung der Formel treffen wir die Annahme, dass sich das zu verfolgende Objekt mit einer Geschwindigkeit von $2m/s$ bewegt. Die Geschwindigkeit des Multikopters wird für diese Auswertung mit annähernd 0 angenommen. Dieser befindet sich also schwebend in der Luft.

Zusätzlich zur Verzögerungs- und Stellzeit des Gimbals muss noch die Berechnungszeit für den neuen Stellwinkel betrachtet werden. Diese wurde auf der Zielhardware gemessen und beträgt im Schnitt $58\mu s$. Nach oben abgeschätzt geht diese nun als $1ms$ in die Formel mit ein. Die Regelung, welche die Stellwerte des Gimbals berechnet, wird im Schnitt alle $100ms$ aufgerufen. Zudem ist die Hardware des Gimbals laut Hersteller nur alle $20ms$ in der Lage, einen Steuerbefehl zu interpretieren. Wir gehen für unsere Berechnung davon aus, dass die Hardware die vollen $20ms$ benötigt, um den Befehl zu interpretieren und kommen somit insgesamt auf eine Verzögerung des Gimbals von $t_{verz} = 0.100 + 0.2 + 0.01 = 0.121s$. Die Winkelgeschwindigkeit des Gimbals beträgt laut Herstellerangaben $0.120^\circ/s$ pro gewählter Geschwindigkeitsstufe. Bei der Regelung des Gimbals wird dieser mit einer Geschwindigkeit von 50 betrieben, was somit $50 \cdot 0.1221^\circ/s$ entspricht.



Der Öffnungswinkel der Kamera beträgt $\alpha = 46^\circ$, für die Übertragungsrate und Bildgröße werden $d_B = 3MB$ und $f_U = 28MB/s$ angenommen. Aufgelöst nach d ergibt sich somit für die Minstdistanz zum Ball:

$$d = \frac{|2|}{\sqrt{2 \cdot (1 - \cos(46))}} \cdot \left(\frac{3}{28} + \frac{46}{2 \cdot 50 \cdot 0.1221} + 0.121 \right) \quad (10.8)$$

Als Unbekannte bleibt nun die Berechnungszeit für die Objekterkennung und die Distanz zum Ball. Zunächst soll eine Minstdistanz ermittelt werden, welche benötigt wird, um die Formel erfüllen zu können. Betrachtet man die Werte für die Stellgeschwindigkeit des Gimbals, wird schnell klar, dass dieser zunächst die limitierende Komponente ist. Um für die Distanz zum Ball erstmalig einen positiven Wert zu erhalten, muss die Distanz zum Ball mindestens $d = 10.2m$ betragen.

Diese Minstdistanz muss nun noch erhöht werden, um zusätzlich noch die Berechnungszeit für die Objekterkennung durchführen zu können. Die Berechnungszeiten der Objekterkennung wurden ebenfalls auf der Hardware gemessen. Für diese ergibt sich im Schnitt die Zeit $c_{algo} = 158ms$.

Löst man die Formel 10.9 nun mit allen Werten auf, ergibt sich somit eine Minstdistanz von

$$d = \frac{|2|}{\sqrt{2 \cdot (1 - \cos(46))}} \cdot \left(\frac{3}{28} + \frac{46}{2 \cdot 50 \cdot 0.1221} + 0.121 + 0.158 \right) = 10.62m \quad (10.9)$$

ab dieser garantiert werden kann, dass das Objekt im Zentrum des Sichtfeldes gehalten wird. Im praktischem Arbeitsalltag sind durchaus niedrigere Distanzen zum verfolgtem Objekt möglich, da die Abschätzung den pessimistischsten Fall betrachtet, bei dem sich das Objekt dauerhaft mit der zulässigen Maximalgeschwindigkeit bewegt. In der Praxis werden aber nur geringe Distanzen überwunden, bevor das Objekt zum Stillstand kommt.



11 Ausblick

Sensorinitialisierung

Wenn während des Betriebes der Reset-Knopf auf dem Carrierboard betätigt wird, kann es beim erneuten Starten zu einem Fehlerfall kommen, in dem die I²C-Sensoren nicht mehr ausgelesen werden können. Dieser Fehler tritt z. B. dann auf, wenn der Reset-Knopf eine Kommunikationssequenz zwischen dem Dataminer und den Sensoren unterbricht. Das kann dazu führen, dass die Kommunikation in einem Punkt unterbrochen wurde, in der die Sensoren auf ein ACK warten, welches aufgrund des Neustarts nicht mehr auftritt. Ein solcher Fehler wird im Dataminer bei der Initialisierung erkannt und verhindert das Starten des Main-Timers. Dadurch, dass in dieser Situation das Valide-Flag der Sensoren FALSE ist, startet das ASG nicht mit seiner Flugregelung und die Motoren bleiben aus. Dieser Zustand lässt sich nur durch ein „Power ON/OFF“ durch den Logik-Akku beheben, welcher die Prozessorplattform sowie die Sensoren versorgt. Zur Zeit muss dieser Vorgang manuell durchgeführt werden. Eine Möglichkeit diesem Fehlerfall vorzubeugen, wäre mit Hilfe des Resetsignals die Versorgungsspannung der Sensoren über einen MOSFET-Schaltung ein- bzw. auszuschalten. Dadurch würden mit der Prozessorplattform ebenfalls die Sensoren neu gestartet werden.

Positionsregelung

Um mit dem Quadrocopter eine genauere Höhenmessung zu erlangen, ist es sinnvoll auf Laser- oder Ultraschallsysteme zurückzugreifen, welche unter dem Quadrocopter befestigt werden. Diese Systeme ermöglichen eine absolute Höhenmessung und sind in dem Szenariumfeld geeigneter einzusetzen. Durch die Verwendung solcher Sensoren wäre auch die Realisierung der autonomen Landung denkbar. Zur Automatisierung des Fluges ist ein weiterer Sensor z.B. GPS zur Bestimmung der X-Y-Position im Raum nötig.

Korrektur der Drift des Gimbals

Eine mögliche Erweiterung der Gimbal-Ansteuerung liegt in der Verwendung optionaler Sensorik. Durch das Implementieren eines Kompasses könnte die Drift der Yaw-Achse, die auf die Gyroskope zurückzuführen ist, kompensiert werden. Dazu müssten die Kompasswerte des externen Kompasses der Gimbal-Elektronik über die I2C Schnittstelle zur Verfügung gestellt werden. Dazu ist es notwendig Eingriffe in die Firmware der Gimbal-Elektronik vorzunehmen. Da die Firmware nicht Open-Source ist, muss ggf. eine eigene Firmware geschrieben werden.

Es wurde messtechnisch ermittelt, dass die Yaw-Achse in ungefähr 10 Minuten eine Drift von 180 Grad aufweist. Eine Korrektur mit der bereits integrierten Sensorik ist nicht möglich, da die Null-Position sich mit der Drift mitbewegt. Die Drift konnte durch das Fusionieren der Gyroskopwerte mit den Beschleunigungswerten der MPU6050 mit Hilfe der GUI für die Gimbal-Elektronik verringert, jedoch nicht komplett eliminiert werden.



Durch das Verwenden eines Kompasses ist es zusätzlich möglich, eine definierte horizontale Ausrichtung der Kamera anzufahren, welches die Ausrichtung auf einen bestimmten Punkt in der Umwelt ermöglicht.

Watchdog

Um ein Ausfall eines Prozessors während des Betriebes zu erkennen, sollte ein sog. Watchdog implementiert werden. Denkbar ist bspw., dass bei einem Dataminierausfall der ASG-Microblaze mit vordefinierten Sensorwerten arbeitet, um den Quadrocopter in einen kontrollierten Sinkflug zu bringen. So könnten mögliche Schäden an der Umwelt und dem Menschen minimiert werden.

Erweitertes Szenario

Neben dem in diesem Projekt umgesetzten Szenario, wird im Kapitel 1.4 noch ein erweitertes Szenario vorgestellt, welches aus zeitlichen Gründen hier nicht bearbeitet wurde. Das erweiterte Szenario ist für mögliche Folgeprojekte ein interessanter Ansatz, um auf dem aktuellen Stand aufzubauen. Gerade der Aspekt der Ermittlung der Vorzugsflugrichtung ist ein weiterer Zwischenschritt zum erfolgreichen autonomen Flug.

Leistungsstärkere Prozessorplattform

Zum aktuellen Zeitpunkt sind die beiden Kerne des AEVV Systems zu ca. 98% ausgelastet. Dabei wird für die Objekterkennung die Videoauflösung reduziert und erreicht maximal 11 fps. Mit einer leistungsstärkeren Prozessorplattform z.B dem Xilinx Zynq UltraScale+, könnte die Objekterkennung mit der Originalauflösung und mit mehreren fps arbeiten. Ebenfalls könnten aufwändigere Algorithmen zur Objekterkennung implementiert werden, um das Objekt sicherer zu erkennen.



12 Fazit

In diesem Projekt wurde erfolgreich ein Quadroptersystem umgesetzt, welches auf das Quadro XL Frame der Firma MikroKopter aufbaut. Basierend auf der Zynq 7020 Prozessorplattform der Firma Xilinx ist ein „mixed critical“ System entstanden, das aus zwei Teilsystemen besteht. Zum einen das AEVV System, welches auf dem ARM Dual Core des Zynqs ausgeführt wird und zum anderen dem ASG, das auf zwei Microblazes im FPGA-Teil der Prozessorplattform aufgeteilt ist.

Das AEVV-System ist der nicht sicherheitskritische Teil des Systems und arbeitet seine Aufgaben auf ein angepasstes Embedded-Linux ab. Diese Aufgaben bestehen darin, den Videostream der Mobius USB-Actioncam mit einer Auflösung von 720p entgegenzunehmen und auf zwei Subsysteme aufzuteilen. Das eine Subsystem besteht aus der Objekterkennung basierend auf OpenCV mit anschließender Objektverfolgung über einen 3-Achs Kameragimbal. In dem anderen Subsystem wird der Videostream an eine Telemetriestation über eine 5 GHz WLAN Verbindung weitergeleitet. Neben diesem Livestream werden ebenfalls weitere Quadroptersdaten wie Spannungs- und Sensorwerte an die Station gesendet.

Die zwei Microblazes ergeben zusammen das sicherheitskritische System. Auf einem Microblaze wird die Kommunikation mit der Peripherie wie z.B. die Fernsteuerung oder die Spannungsüberwachung sowie die Sensordatenfusion ausgeführt. Die Flugalgorithmen werden auf dem zweiten Microblaze abgearbeitet und die Motorstellwerte gesetzt. Diese Avionikalgorithmen sind modellgetrieben mit Hilfe des Tools CAMEL-View entstanden. Aufgrund diverser Komplikationen bei der Portierung des generierten Sourcecodes auf die reale Hardware, wurden die Flugalgorithmen nachträglich manuell implementiert. Die Kommunikation zwischen den Prozessoren werden über BRAMs realisiert, die so ausgelegt sind, dass diese sicherheitskritischen Anforderungen genügen. Die Sensordatenfusion sowie die Flugregelung werden mit einer Frequenz von 500 Hz ausgeführt, um ein möglichst stabiles und agiles Flugverhalten zu erreichen.

Über die MC-20 Fernsteuerung von Graupner kann zum einen der Quadropters jederzeit positioniert und zum anderen kann mit Hilfe verschiedener Schalter auf der Fernbedienung mit dem Flugobjekt interagiert werden. Z.B. kann über einen Schalter zwischen unterschiedlichen Kamera-/Gimbalmodi gewechselt werden. Dazu wurde der sog. „Searching Mode“ implementiert, der mit der Kamera (unter Zuhilfenahme des Gimbals) das Quadroptersumfeld auf den Ball absucht. Beim Erkennen des Objekts wird in den „Tracking Mode“ gewechselt, der den Ball immer im Mittelpunkt des Bildes hält. Das gesuchte Objekt wird mit Hilfe eines Schwellwertverfahrens erkannt. Daraus folgt, dass der Ball eine eindeutige Farbe innerhalb des Videoframes haben muss, um ihn sicher zu detektieren.

Neben der erfolgreichen technischen Umsetzung ist an dieser Stelle der besonders gute



Zusammenhalt zwischen den Projektgruppenmitglieder hervorzuheben. Während der gesamten Projektphase kam es zu keiner nennenswerten Auseinandersetzung auf zwischenmenschlicher Ebene. Aufgrund der heterogenen Gruppeneinteilung konnten größtenteils alle Arbeitspakete innerhalb des dafür vorgesehenen Zeitraumes abgeschlossen werden, sodass ein Quadrokopter entstanden ist, der alle an ihn gestellten Anforderungen erfüllt. Durch die Einhaltung der wöchentlichen Gruppensitzungen war es möglich, zeitnah auf Probleme einzugehen und somit beispielsweise kurzfristig die CAMEL-View Taskforce zu gründen, um sich noch intensiver mit dem Tool auseinander zu setzen.

Abschließend ist festzuhalten das innerhalb der Projektgruppe das Konzept des modellbasierten Testens unterschätzt wurde. Hier hätten direkt zu Beginn des Projekts mehr Arbeitsstunden investiert werden müssen, sodass frühzeitiger Probleme erkennbar gewesen wären. Des Weiteren ist es an einigen Stellen, aufgrund fehlender Schnittstellenbeschreibungen bzw. Übersicht des Gesamtkonzeptes, zu internen Kommunikationsproblemen in der Software gekommen. Dieser Umstand führte zu einer leichten, jedoch vermeidbaren Verzögerung bei der Implementierung einiger Softwarekomponenten. Ebenfalls war es eine Herausforderung ein genügend genaues physikalisches Modell des Quadrokopters zu entwerfen, das nicht zu viele Abstraktionen eines realen Quadrokopters und seiner Umgebung enthält.



Literaturverzeichnis

- [Aac] RTWH Aachen. Barometrische Höhenformel. Website. <http://web.physik.rwth-aachen.de/~fluegge/Vorlesung/PhysIpub/Exscript/8Kapitel/II6Kapitel.html> Letzter Zugriff: 11.06.2014.
- [ABB09] J.H. Anderson, S.K. Baruah, and B.B. Brandenburg. Multicore operating-system support for mixed criticality. In Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, 2009. San Francisco.
- [Abs] AbsInt. ait worst-case execution time analyzers. Website. <http://www.absint.com/profile.htm> Letzter Zugriff: 5.03.2015.
- [And] Sven Andersson. Fpga design from scratch. part 89. Website. <http://svenand.blogdrive.com/archive/150.html> Letzter Zugriff: 21.03.2015.
- [Art] Xavi Artigas. Gstreamer sdk documentation - basic tutorial 5: Gui toolkit integration. Website. <http://docs.gstreamer.com/display/GstSDK/Basic+tutorial+5%3A+GUI+toolkit+integration> Letzter Zugriff: 22.03.2015.
- [Aud91] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Real-Time Systems Research Group, Dept. of Computer Science, University of York, November 1991.
- [BBB⁺09] James Barhorst, Todd Belote, Dr Pam Binns, Jon Hoffman, Dr James Pournicka, Dr Prakash Sarathy, Dr John Scoredos, Peter Stanfill, Dr Douglas Stuart, and Russell Urzi. A research agenda for mixed-criticality systems. Technical report, Boeing, Lockheed Martin, Honeywell, AFRL, Northrop Grumman, 2009.
- [BD14] Alan Burns and Robert I. Davis. Mixed criticality systems - a review. University of York, UK - Department of Computer Science, Juli 2014.
- [blu] Bluetooth 3.0+hs. Website. <http://www.elektronik-kompodium.de/sites/kom/1405151.htm> Letzter Zugriff: 13.07.2014.
- [Bos09] Bosch. *Datasheet BMP085 - Digital pressure sensor*, 2009. <https://www.sparkfun.com/datasheets/Components/General/BST-BMP085-DS000-05.pdf> Letzter Zugriff: 30.09.2014.
- [Bre13] Dipl.-Ing. (FH) W.-D. Bretschneider. Versuch: Bildverarbeitung. Website, November 2013. <https://www.htw-dresden.de/fileadmin/userfiles/et/Labore/Prozessmesstechnik/Bildverarbeitung.pdf> Letzter Zugriff: 24.06.2014.
- [Buc] Jörg Buchholz. Regelungstechnik und flugregler. Technical report, HS Bremen.

- [Bü13] R. Büchi, editor. *Faszination Quadrocopter*. Verlag für Technik und Handwerk neue Medien GmbH, 2013.
- [Can] Canon. Bildstabilisator is. Website. http://www.canon.de/for_home/product_finder/cameras/ef_lenses/image_stabilizer.asp Letzter Zugriff: 22.06.2014.
- [cf] cafe future.net. Pizza-lieferservice per drohne. Website. <http://www.cafe-future.net/news/pages/show.php?id=30839> Letzter Zugriff: 30.07.2014.
- [Com14] RoboCup Technical Committee. Robocup standard platform league (nao) rule book. Website, Juni 2014. <http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2014.pdf> Letzter Zugriff: 16.07.2014.
- [Cor08] D. Cordes. Schleifenanalyse für einen wcet-optimierenden compiler basierend auf abstrakter interpretation und polylib. Diplom thesis, TU Dortmund, April 2008.
- [CR07] B. Zengler C. Rupp, S. Queins, editor. *UML 2 Glasklar*. Carl Hanser Verlag München, 2007.
- [dlr] Beschreibung der barometrischen höhenstufe. Website. http://www.dlr.rlp.de/Internet/global/themen.nsf/se_quick/7678BA051452BBFBC1256F250047DCC6?OpenDocument Letzter Zugriff: 30.08.2014.
- [dog] Can you see the dog? Website. http://www.mindfake.com/illusion_22.html Letzter Zugriff: 24.06.2014.
- [DRRG10] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems Journal*, 2010.
- [EC112] Report from the workshop on mixed criticality systems. European Commission, Information Society and Media Directorate-General Unit G3/Computing Systems Research Objective, 2012.
- [Elea] Trenz Electronic. Trenz electronic. Website. <http://www.trenz-electronic.de/de/startseite.html> Letzter Zugriff: 02.10.2014.
- [Eleb] Trenz Electronic. Trenz electronic wiki - te0720 user manual. Website. <https://wiki.trenz-electronic.de/display/TE0720/TE0720+User+Manual> Letzter Zugriff: 05.04.2015.
- [elec] Basecam electronics. Basecam electronics. Website. <http://www.basecamelectronics.com> Letzter Zugriff: 29.09.2014.

- [Eled] Robotics Electronics. Using the i2c bus. Website. http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html Letzter Zugriff: 02.09.2014.
- [FBE⁺13] P. Fallavollita, M. Balsi, S. Esposito, M.G. Melis, M. Milanese, and L. Zappino. Uas for archaeology. new perspectives on aerial documentation. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(2):131–135, 2013.
- [FPL12] Uwe Kiencke Fernando Puente León. *Messtechnik*. Springer Vieweg, 9 edition, 2012.
- [FPL13] Uwe Kiencke Fernando Puente León. *Numerische Mathematik: Eine beispielorientierte Einführung*. Carl Hanser Verlag GmbH & Co. KG, 5 edition, 2013.
- [Fra] Boost Framework. Boost framework. Website. <http://http://www.boost.org/> Letzter Zugriff: 17.01.2015.
- [GAC⁺] Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, Manil Dev Gomony, Sven Goossens, Martijn Koedam, Yonghui Li, Davit Mirzoyan, Anca Molnos, Ashkan Beyranvand Nejad, Andrew Nelson, and Shubhendu Sinha. Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow. Eindhoven University of Technology, Delft University of Technology.
- [Gim] Serial protocol specification v.2.4.x. Website. http://www.basecamelectronics.com/files/SimpleBGC_2_4_Serial_Protocol_Specification.pdf Letzter Zugriff: 16.03.2015.
- [Gmba] Aapekteins GmbH. Slowmotion-quadrocopter filmaufnahmen. Website. <http://www.aspekteins.com/slowmotion-quadrocopter-filmaufnahmen> Letzter Zugriff: 30.07.2014.
- [Gmbb] Bildungshaus Schulbuchverlage Westermann Schroedel Diesterweg Schöningh Winklers GmbH. Pole und magnetfeld der erde. Website. <http://www.diercke.de/kartenansicht.xtp?artId=978-3-14-100700-8&stichwort=Rotationsachse&fs=1> Letzter Zugriff: 31.07.2014.
- [Gmbc] Bose GmbH. Bose soundlink mini bluetooth speaker. Website. <http://www.bose.de/DE/de/home-and-personal-audio/bluetooth-ipod-speakers/bluetooth-speakers/soundlink-mini-bluetooth-speaker/> Letzter Zugriff: 31.07.2014.
- [Gmbd] Reichelt Elektronik GmbH. Zigbee xbee d. Website. <http://www.reichelt.de/?ARTICLE=108350> Letzter Zugriff: 31.07.2014.

- [Gon13] David Gonzalez. „integration of mixed-criticality subsystems on multi-core processors“ – industrial application of multipartes. IK4 IKERLAN, HiPEAC Workshop, 2013.
- [Goo] Google. Google c++ testing framework. Website. <https://code.google.com/p/googletest/>.
- [GoP] GoPro Inc., Kalifornien. *Datasheet GoPro Hero 3+ Silver Edition*. <http://de.gopro.com/cameras/hd-hero3-silver-edition> Letzter Zugriff: 09.07.2014.
- [Gro] M. Groer. kleine-fotoschule. http://www.kleine-fotoschule.de/dateien/bilder/Bildstabilisierung/Bildstabilisierung_Body.jpg Letzter Zugriff: 10.06.2014.
- [Gru] Kim Gruettner. Modelling, simulation and analysis of mixed-criticality systems under consideration of extra-functional properties. DAC 2014.
- [Hea12] J. Herz and et. al., editors. *Mobile Roboter - Eine Einführung aus Sicht der Informatik*. Springerverlag, 2012.
- [Hei14] Andreas Hein. Vorlesungsfolien zur robotik, März 2014.
- [HGE⁺14] Domenik Helms, Kim Grüttner, Reef Eilers, Malte Metzdorf, Kai Hylla, Frank Poppen, and Wolfgang Nebel. Considering variation and aging in a full chip design methodology at system level. Carl von Ossietzky Universität Oldenburg, OFFIS - Institute for Information Technology, Juni 2014.
- [Hin] Powerparts UAV Armin Hinz. 3-achs smart gopro bl gimbal dys. Website. <http://powerparts-uav.de/produkte/brushless-gimbal/gopro-gimbal/44/3-achsen-smart-gopro-bl-gimbal-dys> Letzter Zugriff: 31.07.2014.
- [HMS07] E. Hering, R. Martin, and M. Stohrer. *Physik für Ingenieure*. Springerverlag, 10 edition, 2007.
- [Hob] Hobbyking. Hobbyking. Website. <http://www.hobbyking.com/hobbyking/store/catalog/52136.jpg> Letzter Zugriff: 10.10.2014.
- [HS09] Mike Hinchey Hesham Shokry. Model-based verification of embedded software, 2009.
- [hSF12] Prof. Dr-Ing. habil. S. Fatikow. Microrobotics 2, 2012. 7. Edition.
- [IL07] Sang H. Son Insup Lee, Joseph Y-T. Leung, editor. *Handbook of Real-Time and Embedded Systems*. Chapman and Hall, 2007.
- [Inca] GoPro Inc. Gopro hero3 white edition. Website. <http://de.gopro.com/cameras/hd-hero3-white-edition> Letzter Zugriff: 31.07.2014.

- [Incb] Xilinx Inc. Xilinx wiki - build device tree blob. Website. <http://www.wiki.xilinx.com/Build+Device+Tree+Blob> Letzter Zugriff: 10.04.2015.
- [Incc] Xilinx Inc. Zynq-7000 all programmable soc technical reference manual - ug585 (v1.10) february 23, 2015. Website. http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf Letzter Zugriff: 28.03.2015.
- [Ind] Helico Aerospace Industries. Auto-follow drone for gopro camera. Website. <https://www.kickstarter.com/projects/airdog/airdog-worlds-first-auto-follow-action-sports-dron> Letzter Zugriff: 30.07.2014.
- [Inv] InvenSense Inc., Sunnyvale/USA. *MPU-9150 Product Specification Revision 4.3*. http://www.invensense.com/mems/gyro/documents/PS-MPU-9150A-00v4_3.pdf Letzter Zugriff: 31.07.2014.
- [JK13] Anna-Maria von Pippich Juerg Kramer, editor. *Von den natuerlichen Zahlen zu den Quaternionen*. Springer Verlag Wiesbaden, 2013.
- [JS13] Thomas Zurawka Jörg Schäuuffele. *Automotive Software Engineering*. Springer Vieweg, 5 edition, 2013.
- [KG] Micro-Epsilon Messtechnik GmbH & Co. KG. Kompakte wärmebildkamera für industrielle anwendungen. Website. http://www.micro-epsilon.de/temperature-sensors/thermoIMAGER/thermoIMAGER_160/index.html Letzter Zugriff: 31.07.2014.
- [LCO] LCOV. Linux test project's kernel code coverage results. Website. <http://ltp.sourceforge.net/test/coverage/> Letzter Zugriff: 5.03.2015.
- [Lei] J.G. Leishman. The bréguet-richet quad-rotor helicopter of 1907. Website. <http://aero.umd.edu/~leishman/Aero/Breguet.pdf> Letzter Zugriff: 20.06.2014.
- [LL04] Christian Große Lordemann and Martin Lambers. Objekterkennung in bilddaten. Website, 2003/2004. <https://www.math.uni-muenster.de/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe9/Thema09-ObjekterkennungInBilddaten-ChristianGrosseLordemann-MartinLambers.pdf> Letzter Zugriff: 24.06.2014.
- [LLC] Novotm LLC. Novotm llc. Website. <http://www.novotm.com> Letzter Zugriff: 02.10.2014.
- [Lufa] Luftverkehrsgesetz. Website. http://www.gesetze-im-internet.de/luftvg/___1.html Letzter Zugriff: 16.07.2014.



- [Lufb] Luftverkehrsordnung. Website. http://www.gesetze-im-internet.de/luftvo/___16.html Letzter Zugriff: 16.07.2014.
- [Lun13] Jan Lunze. *Regelungstechnik 1*. Springer Vieweg, 9 edition, 2013.
- [LW12] H. Lutz and W. Wendt. *Taschenbuch der Regelungstechnik*. Verlag Harri Deutsch, 9 edition, 2012.
- [Mika] Modellbauservo ansteuerung. Website. <http://www.spiegel.de/auto/aktuell/google-auto-unterwegs-im-selbstfahrenden-auto-a-969532.html> Letzter Zugriff: 30.06.2014.
- [Mikb] MikroKopter. MikroKopter - quadroKopter xl. Website. <http://wiki.mikroKopter.de/MK-QuadroXL#include.2Fassembling.2Fpowerboard-QuadroXL2.Stromverteiler> Letzter Zugriff: 02.10.2014.
- [Mikc] MikroKopter. Mk3638 & mk3538. Website. <http://wiki.mikroKopter.de/MK3538> Letzter Zugriff: 02.10.2014.
- [Mikd] MikroKopter. Propeller. Website. http://wiki.mikroKopter.de/Propeller#Grunds.2BAOQ-tzliche_.2BANw-berlegungen Letzter Zugriff: 02.10.2014.
- [Mob] Mobius. Mobius. Website. http://imgs.inkfrog.com/pix/kdataonline999on/Mobius_Camera-Nr_4GB-D.jpg Letzter Zugriff: 10.10.2014.
- [MP02] Andreas Spillner Martin Pol, Tim Koomen. *Management und Optimierung des Testprozesses*. Dpunkt.Verlag, 2 edition, 2002.
- [NKLL10] Yamini Nimmagadda, Karthik Kumar, Yung-Hsiang Lu, and C.S. Georg Lee. Real-time moving object recognition and tracking using computation offloading. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2449–2455. IEEE, 2010.
- [Nol06] W. Nolting, editor. *Theoretische Physik 1*. Springerverlag, 2006.
- [ope] opencv.org. Opencv. Website. <http://opencv.org> Letzter Zugriff: 31.07.2014.
- [Orl11] Peter F. Orlowski. *Praktische Regeltechnik*. Springer Verlag, 9 edition, 2011.
- [Phi04] H.-W. Philippsen, editor. *Einstieg in die Regelungstechnik*. Hanser Verlag, 2004.
- [Pli] Peter Plischka. Apmcopter mikroKopter quadroKopter hexakopter. Website. <http://plischka.at/index.html> Letzter Zugriff: 31.07.2014.



- [Plu13] Werner Pluta/Golem.de. Saarbrücker drohne fliegt mit drei rotoren. Website, April 2013. <http://www.golem.de/news/tricopter-saarbruecker-drohne-fliegt-mit-drei-rotoren-1304-98694.html> Letzter Zugriff: 31.07.2014.
- [pp2] /powerparts-uavWebsite. http://powerparts-uav.de/media/image/thumbnail/3829_dimg1_720x600.jpg Letzter Zugriff: 11.06.2014.
- [Pre] Staff/Associated Press. Dhl joins amazon in testing drone delivery. Website. http://www.chicoer.com/ci_24685805/dhl-joins-amazon-testing-drone-delivery Letzter Zugriff: 31.07.2014.
- [Pul05] Tim Puls. Diplomarbeit: Validierung komplementärer sensorsysteme für die autonome innenraumnavigation von kleinstflugzeugen mit anwendung auf einen 4-rotor-helikopter. Technical report, Universität Oldenburg, 2005.
- [Pul11] T. Puls. *Lokalisations- und Regelungsverfahren für einen 4-Rotor-Helikopter*. Dissertation, Carl von Ossietzky Universität Oldenburg, Juni 2011.
- [Quaa] Rotation zwischen zwei quaternionen. Website. <http://help.autodesk.com/cloudhelp/2015/ENU/Maya/images/GUID-D6410250-2B26-4B80-B810-C2D9AAE79F9E.png> Letzter Zugriff: 23.02.2015.
- [Quab] LLC Quadrocopter. Mk basicset quadrocopter xl. Website. http://www.quadrocopter.com/MK-Basicset-QuadroKopter-XL_p_283.html Letzter Zugriff: 31.07.2014.
- [rc2] img.rcmaster. http://img.rcmaster.net/41995_0.jpeg Letzter Zugriff: 11.06.2014.
- [rc214] Website, Juli 2014. <http://www.robocup.org/> Letzter Zugriff: 07.07.2014.
- [Ret14] Prof. Dr. Achim Rettberg. Realzeitbetriebssysteme vorlesung. Carl von Ossietzky Universität Oldenburg, 2014.
- [rob] Sprungantworten verschiedener regler. Website. www.roboternetz.de/wiki/uploads/Main/vergleich.gif Letzter Zugriff: 21.05.2014.
- [rq] rc quadrocopter.de. Zookal und flirtey beginnen mit der auslieferung per quadrocopter. Website. <http://rc-quadrocopter.de/zookal-und-flirtey-beginnen-mit-auslieferung-per-quadrocopter> Letzter Zugriff: 30.07.2014.

- [Sch] Sören Schreiner. Modellbasierter entwurf, validierung und verifizierung der sicherheitskritischen software eines quadropters. Technical report, Universität Oldenburg.
- [Sch06] A. Schauer. Nie mehr verwackelte fotos, 2006. http://www.chip.de/artikel/Die-besten-Digicams-mit-Bildstabilisator_18872397.html Letzter Zugriff: 12.06.2014.
- [Sch10] Sören Schreiner. Bachelorthesis - modellbasierter entwurf, validierung und verifizierung der sicherheitskritischen software eines quadropters. Technical report, OFFIS e. V., 2010.
- [Sch14] Thomas Schulz. Testfahrt in google self-driving car: Dieses auto kommt ohne sie aus. Website, Mai 2014. <http://www.spiegel.de/auto/aktuell/google-auto-unterwegs-im-selbstfahrenden-auto-a-969532.html> Letzter Zugriff: 24.06.2014.
- [SE] Conrad Electronic SE. Led-streifen flexibel. Website. <http://www.conrad.de/ce/de/product/154977/LED-Streifen-flexibel-selbstklebend-12-VDC-540-mm-RGB> Letzter Zugriff: 31.07.2014.
- [SL05] A. Spillner and T. Linz. *Basiswissen Softwaretest*. dpunkt.verlag, 3 edition, 2005.
- [SPE] SPECTAIR. Flugroboter für vielfältige einsatzbereiche. Website. <http://www.spectair.de/anwendungen> Letzter Zugriff: 30.07.2014.
- [STMa] STMicroelectronics. *Datasheet L6234*.
- [STMb] STMicroelectronics. Using lsm303dlh for a tilt compensated electronic compass. Website. <https://www.pololu.com/file/0J434/LSM303DLH-compass-app-note.pdf> Letzter Zugriff: 13.04.2015.
- [Str] J. Strickland. What is a gimbal – and what does it have to do with nasa? Website. <http://science.howstuffworks.com/gimbal.htm> Letzter Zugriff: 11.06.2014.
- [SZ06] J. Schäuffele and T. Zurawka. *Automotive Software Engineering*. Springerverlag, 3 edition, 2006.
- [SZ11] Manfred Reuter Serge Zacher. *Regelungstechnik für Ingenieure*. Vieweg+Teubner Verlag, 13 edition, 2011.
- [Teb] Tido Tebben. Funktionsweise am beispiel eines quadropters in +-formation. Website. <http://www.qc-copter.de/wiki/index.php?title=Funktionsweise> Letzter Zugriff: 31.07.2014.

- [TOG⁺] Salvador Trujillo, Roman Obermaisser, Kim Grüttner, Francisco J. Cazorla, and Jon Perez. European project cluster on mixed-criticality systems. Technical report, IK4-IKERLAN, University of Siegen, OFFIS - Institute for Information Technology, Barcelona Supercomputing Center and IIIA-CSIC.
- [Unia] Grundlagen videokompression. Website. <http://www.hki.uni-koeln.de/sites/all/files/courses/842/video.pdf> Letzter Zugriff: 18.06.2014.
- [Unib] Servosteuerung. Website. http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.050/vorlesungen/sose09/lrob/servos.pdf Letzter Zugriff: 11.06.2014.
- [Ves07] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. Honeywell Labs, 2007.
- [W.] Olli W. Imu data fusing: Complementary, kalman, and mahony filter. Website. <http://www.olliw.eu/2013/imu-data-fusing/#refRM05> Letzter Zugriff: 26.10.2014.
- [Wei08] S. Weinzierl, editor. *Handbuch der Audiotechnik*. Springer-Verlag, 2008.
- [Wie13] Markus Wieghaus. *Entwicklung eines drahtlosen Sensornetzwerks zum Einsatz im Rasterelektronenmikroskop*. Bachelorarbeit, Carl von Ossietzky Universität Oldenburg, 2013.
- [wlaa] Ieee 802.11 / wlan-grundlagen. Website. <http://www.elektronik-kompodium.de/sites/net/0610051.htm> Letzter Zugriff: 13.07.2014.
- [wlab] Ieee 802.11ac / gigabit-wlan. Website. <http://www.elektronik-kompodium.de/sites/net/1602101.htm> Letzter Zugriff: 13.07.2014.
- [wlac] Ieee 802.11ad - wireless gigabit (wigig). Website. <http://www.elektronik-kompodium.de/sites/net/1407241.htm> Letzter Zugriff: 13.07.2014.
- [wlad] Ieee 802.11b / wlan mit 11 mbit. Website. <http://www.elektronik-kompodium.de/sites/net/0907031.htm> Letzter Zugriff: 13.07.2014.
- [wlae] Wlan-topologie. Website. <http://www.elektronik-kompodium.de/sites/net/0907071.htm> Letzter Zugriff: 13.07.2014.
- [ZN09] Justyna Zander-Nowicka. Model-based testing of real-time embedded systems in the automotive domain, 2009.

A.3 Debug-Pinzuordnung

Tabelle A.1: Debug-Pinzuordnung zwischen der Verteilerplatine (VP) und dem Xilinx Zynq 7020

Pin VP	Belegung	Pin Zynq	Bank Zynq	Mapping
J1-A18	J3.1	L16P	Bank 35	-
J1-A19	J3.2	L16N	Bank 35	-
J1-A20	J3.3	L12N	Bank 34	-
J1-A21	J3.4	L12P	Bank 34	-
J1-A22	J3.5	L5N	Bank 34	-
J1-A23	J3.6	L5P	Bank 34	-
J1-A24	J3.7	L18P	Bank 34	-
J1-A25	J3.8	L18N	Bank 34	-
J1-A26	J3.9	L10P	Bank 34	-
J1-A27	J3.10	L10N	Bank 34	-
J1-A28	J3.11	L21P	Bank 34	-
J1-A29	J3.12	L21N	Bank 34	-
J1-A30	J3.13	L17P	Bank 34	-
J1-A31	J3.14	L17N	Bank 34	-
J2-C2	JP1.8	AB1	Bank 13	ASG
J2-C3	JP1.7	AB2	Bank 13	ASG
J2-C4	JP1.6	AB4	Bank 13	AEVV
J2-C5	JP1.5	AB5	Bank 13	AEVV
J2-C6	JP1.4	AB6	Bank 13	Dataminer
J2-C7	JP1.3	AB7	Bank 13	Dataminer
J2-C8	JP1.2	U4	Bank 13	Dataminer
J2-C9	JP1.1	T4	Bank 13	Dataminer
J2-B27	JP2.6	X9 (CPLD)	-	CPLD
J2-B28	JP2.3	X10 (CPLD)	-	CPLD
J2-B29	JP2.2	X12 (CPLD)	-	CPLD
J2-B30	JP2.1	X13 (CPLD)	-	CPLD
J2-C26	JP2.8	X2 (CPLD)	-	CPLD
J2-C27	JP2.7	X3 (CPLD)	-	CPLD
J2-C28	JP2.5	X6 (CPLD)	-	CPLD
J2-C29	JP2.4	X7 (CPLD)	-	CPLD

A.4 Ausführungszeiten Dataminertasks

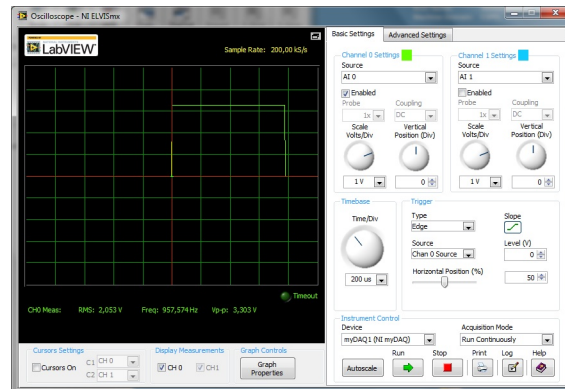


Abbildung A.7: τ_1 Maximale Ausführungszeit der Aktualisierung der Lagewerte

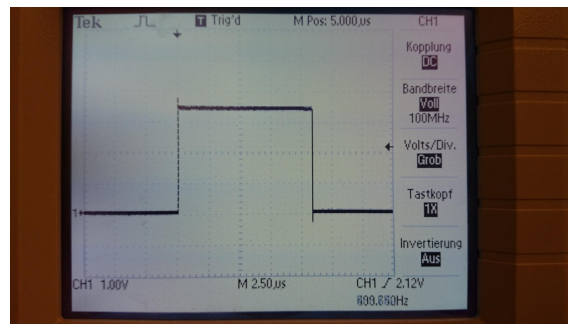


Abbildung A.8: τ_1 Maximale Ausführungszeit um die Lagewerte in das BRAM zu schreiben

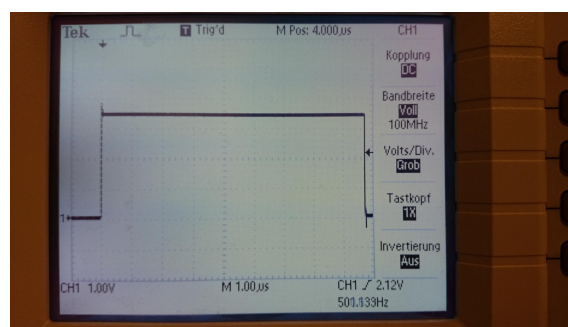


Abbildung A.9: τ_2 Maximale Ausführungszeit um die Temperaturen auszulesen

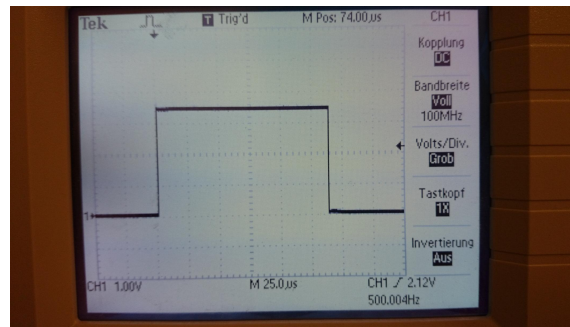


Abbildung A.10: τ_3 Maximale Ausführungszeit um die Spannungen zu ermitteln

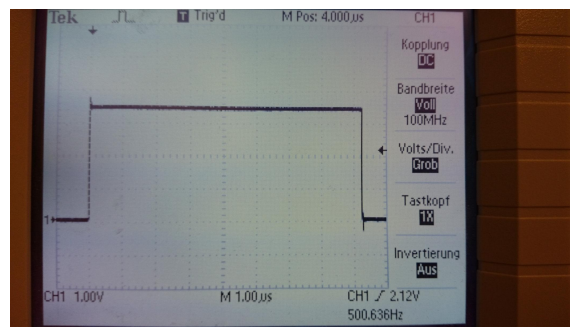


Abbildung A.11: τ_4 Maximale Ausführungszeit um das PPM-Signal zu ermitteln

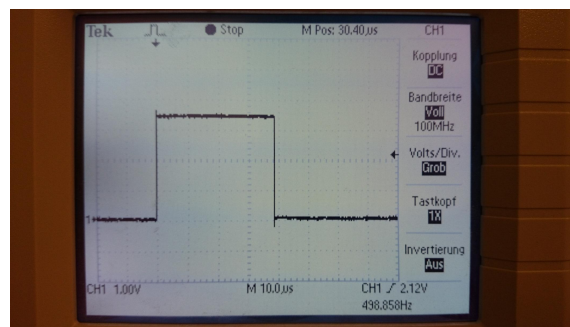


Abbildung A.12: τ_5 Maximale Ausführungszeit für das Schreiben der Daten in das GAM-Daten Frame

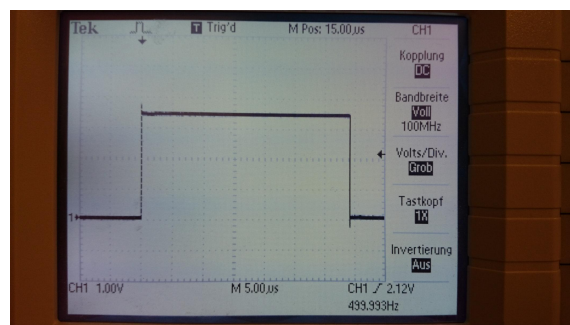


Abbildung A.13: τ_6 Maximale Ausführungszeit für das Schreiben der Peripheriedaten in das BRAM

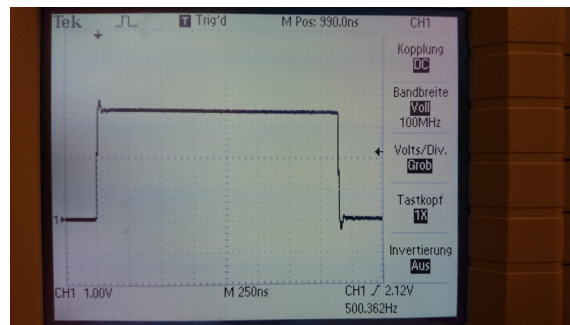


Abbildung A.14: τ_7 Maximale Ausführungszeit für das Setzen des akustischem Signals im GAM-Daten Frame

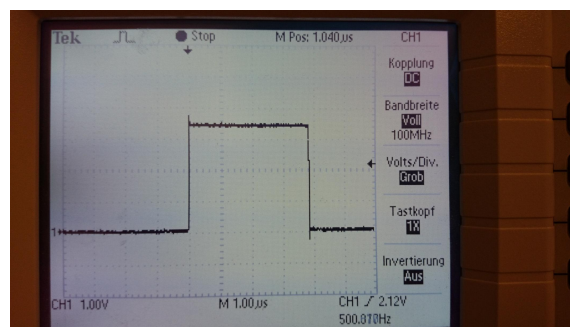


Abbildung A.15: τ_8 Maximale Ausführungszeit für das Senden des GAM-Frames

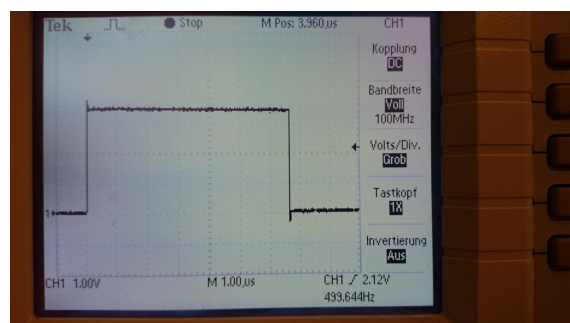


Abbildung A.16: τ_9 Maximale Ausführungszeit für das Ein-/Ausschalten der Peripherie