CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

— Abteilung Systemanalyse und -optimierung —

Abschlussbericht der Projektgruppe

Marine Observation Platform for Surfaces III

Erster Gutachter: Prof. Dr. Axel Hahn
Zweiter Gutachter: Sascha Hornauer

Oldenburg, 13. Oktober 2015

Teammitglieder:

Faisal Alotaibi
Tim Baalmann
Matthias Esser
Conrad Fifelski
Thomas Hellkamp
Christelle Kamga
Paul Kröger
Simon Ortmann
Christof Schlaak
Weert Stamm
Carole Tchuenkam

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Die Entwicklung autonomer Fahrzeuge ist seit langem im automobilen Bereich populär. Doch auch im maritimen Bereich gibt es viele Anwendungsbereiche für autonome Fahrzeuge. Eigenständig Wasserwerte messen, komplexe Messungen des Umfangs von Trübheitsfahnen bei Grabarbeiten vornehmen oder sogar Fracht und Personen befördern sind nur einige der Einsatzgebiete für autonome Boote. Mit dem im Rahmen des Projekts MOPS (Marine Observation Platform for Surfaces) entwickelten Boot sollen die Möglichkeiten eines maritimen autonomen Fahrzeugs wissenschaftlich ermittelt werden. Auf seinen Fahrten soll das Boot eigenständig Hindernisse umfahren und Wasserwerte messen können. Diese Arbeit stellt die wesentlichen Konzepte des Projekts MOPS III vor und präsentiert die Ergebnisse im Vergleich zu Vorgängermodellen.

1. Einleitung

In diesem Bericht wird ein Überblick über die Projektgruppe „Marine Observation Platform for Surfaces III" (MOPS III) gegeben, die im Rahmen des Studiums an der Carl von Ossietzky Universität Oldenburg stattgefunden hat. Es wird auf die Rollen der Gruppenmitglieder und das Ziel des Projekts eingegangen, sowie die Motivation für das Projekt dargestellt. Anschließend wird die Umsetzung der Projektarbeiten beschrieben.

1.1. Allgemeines zur Projektgruppe

Die Carl von Ossietzky Universität Oldenburg bietet zahlreiche Projektgruppen für informatiknahe Studiengänge an. „Diese bestehen in der Regel aus sechs bis zwölf Teilnehmer/innen, die gemeinsam ein komplettes Projekt im Umfang von vier Modulen (jeweils zwei pro Semester) durchführen. Ziel ist es, anhand eines gegebenen Problems die vollständige Entwicklung von der Problemanalyse bis hin zur Realisierung des Systems durchzuführen" [5]. Dabei werden Studierende an die berufsnahen Arbeitsweisen wie Teamarbeit, Arbeitsteilung und Übernahme von Verantwortung herangeführt. Des Weiteren verstärken die Studierenden ihre persönlichen Fähigkeiten wie das Aufbereiten von Inhalten, das zielorientierte Argumentieren, sowie die Präsentations- und Urteilsfähigkeit [5]. Für die Mitglieder einer Projektgruppe gilt es ein Portfolio an Leistungen zu erbringen, welches für die Abschlussnote ausschlaggebend ist. Dieses Portfolio teilt sich in Individual- und Gruppenleistungen. Die grundlegend zu erbringende Leistung innerhalb einer Projektgruppe ist eine konstante, konstruktive und aktive Mitarbeit sowohl im Bereich der Konzeption als auch der Entwicklung. Zur Individualleistung zählen eine Seminararbeit und ihre Präsentation, sowie die soziale Interaktion und Problemlösungskompetenz des einzelnen Teilnehmers.

1.2. Motivation und Zielsetzung der Projektgruppe

Die Umweltbedingungen im Meer ändern sich ständig, sowohl lang- als auch kurzfristig. Um dies zu messen ist es notwendig das Marineökosystem zu beobachten. Daraus ergibt sich ein breites Forschungsfeld rund um die Meeresbiologie. Die Bewirtschaftung der See verursacht Störungen der Umwelt, denen sich Wissenschaftler auch in der heutigen technisch und elektronisch unterstützten Welt stellen müssen. Es existieren bereits Lösungen, wie z.B. Forschungsschiffe, Bojen oder Systeme, die direkt auf dem Meeresgrund platziert werden. Diese Lösungen sind allerdings teuer und nicht flexibel nutzbar.

Die Probleme dieser Lösungsansätze legen den Grundstein für das Projekt. Ziel dieser Projektgruppe ist die Weiterentwicklung eines autonomen Wasserfahrzeugs, welches in der Lage sein soll, Messungen von Wasserparametern durchzuführen. Diese Messwerte sind für das Institut für Chemie und Biologie des Meeres (ICBM) in Oldenburg interessant, daher soll es Angehörigen dieses Instituts ohne größere Probleme möglich sein, den MOPS zu

bedienen. Der vorgegebene Einsatzbereich des MOPS soll die Wesermündung sein, mit dem Ziel eines Tages auch auf Hochsee zu fahren.

1.3. Aufbau des Projektabschlussberichts

In diesem Abschnitt ist der Aufbau des weiteren Dokumentes beschrieben. Zu Beginn erfolgt eine Übersicht über den Aufbau des Projektes. Ebenso werden die zugeteilten Aufgaben und Rollen innerhalb der Projektgruppe aufgezeigt. Außerdem wird die Vorgehensweise der Projektgruppe sowie die Werkzeuge beschrieben, die im Laufe des Projekts zur Kommunikation, Entwicklung sowie Dokumentation genutzt wurden. Anschließend folgen detaillierte Darstellungen der erarbeiteten Konzepte, der durchgeführten Tätigkeiten und der erreichten Ziele, aufgeteilt in die Abschnitte Onshore-Software, Onboard-Software Architektur, Hardware und Systemsoftware Onboard, Kommunikation, Planungs- und Steuerungssoftware Onboard, Payload und Testfahrten. Zum Schluss erfolgen noch ein allgemeines Fazit und ein Ausblick.

2. Anforderungsdefinition

In diesem Abschnitt des Abschlussberichts wird auf die Anforderungsdefinition eingegangen. Sie beschreibt die gewünschte Funktionalität des Produkts (funktionale Anforderungen) und gewünschte (nichtfunktionale) Anforderungen an die Qualität (wie zum Beispiel Betriebssicherheit, Transportfähigkeit, Schonung der Umwelt, uvm.).

2.1. Funktionale Anforderungen

ID	**Name**
FA00	Autonomes / Ferngesteuertes Fahren
FA01	Einsatzzeit 8-12-Stunden
FA02	Hochseetauglichkeit
FA03	Kollisionserkennung und -vermeidung
FA04	Sammeln und Speichern von Sensordaten
FA05	Bereitstellung eines Bussystems für Sensoren
FA06	Durchführung von Bereichs- und Linienmessungen (Mission)
FA07	Plug-&-Play für Sensoren
FA08	Definition von Sperrgebieten (Kollisionsvermeidung)
FA09	Selbstständiger Abbruch der Mission bei Gefahr
FA10	Selbstständiges Zurückkehren zur Startposition
FA11	Umweltfreundlichkeit
FA12	Notabschaltung
FA13	Beleuchtung und Signalmittel
FA14	Einsatztemperaturen von $-5°C$ bis $40°C$

Tabelle 2.1.: Funktionale Anforderungen

2.2. Nichtfunktionale Anforderungen

ID	**Name**	**Eigenschaften**
NFA00	Erhöhung der Betriebssicherheit	Sicherheitskonzepte sollen vermeiden, dass Umwelt und Mensch zu Schaden kommen.

NFA01	Reduzierung des Einsatzaufwands	Schneller Zusammenbau und einfache Konfiguration des MOPS vor Missionsstart.
NFA02	Transportfähigkeit erhöhen	Einfacher Transport des MOPS.
NFA03	Preis	Die Kosten für die MOPS-Plattform soll 2000€ nicht überschreiten.
NFA04	Reengineering bzgl. Qualität und Flexibilität	Neue Konzepte sollen die Qualität und die Flexibilität des MOPS erhöhen.

Tabelle 2.2.: Nichtfunktionale Anforderungen

3. Projektorganisation

Zur Verwirklichung der Ziele von MOPS III sind ein Projektmanagement und eine Projektorganisation erforderlich. Dieses Kapitel gibt einen Einblick in die Projektorganisation sowie die Vorgehensweise der Projektgruppe MOPS III. Zunächst werden die projektinternen Aufgabenbereiche unterschieden. Nachfolgend wird der geplante Projektablauf erörtert.

3.1. Aufteilung in Kleingruppen

Nach der Anforderungserhebung werden die Anforderungen priorisiert und in vier verschiedene Entwicklergruppen aufgeteilt. Hierbei wird auf die Stärken und Vorlieben der Teilnehmer eingegangen. Jede Gruppe übernimmt einen eigenen Aufgabenbereich und bearbeitet diesen weitesgehend unabhängig von den übrigen Teilgruppen. Die folgenden Gruppen und ihre Aufgabenbereiche werden gebildet:

- **Hardware**: Bus, Steuerung, Architektur
- **Werft**: Pumpe, Licht, Hupe, Sensoren, Plane, Notaus, Inbetriebnahme auf See
- **Onshore-Software**: Kommunikation, Evaluierung, Refactoring oder Neu-Entwicklung der bestehenden Onshore-Software
- **Onboard-Software**: Kommunikation, Evaluierung, Refactoring oder Neu-Entwicklung der bestehenden Onboard-Software, Sensoransteuerung, Bahnplanung / -regelung

Um den Projektablauf zu unterstützen werden projektinterne Rollen verteilt, diese werden im Folgenden erörtert.

3.2. Projektinterne Rollen

In diesem Abschnitt sollen die Rollen vorgestellt werden, die zu Beginn des Projekts im Wintersemester 2014/2015 festgelegt wurden. Die von den Betreuern vorgeschlagenen Rollen werden nach einer Diskussion ergänzt und von der Projektgruppe verabschiedet. Das Übernehmen einer Rolle ist mit der Verantwortung für den definierten Bereich verknüpft. Desweiteren sorgen Rollen für verkürzte Kommunikationswege, da sich Gruppenmitglieder wie Betreuer direkt an entsprechende Ansprechpartner wenden können. Zusätzlich wurde die Moderation und das Protokoll der Sitzung wöchentlich gewechselt. Die jeweiligen Rollen und deren Aufgaben werden nachfolgend vorgestellt.

3.2.1. Projektmanager

Die Aufgaben des Projektmanagements zur Führung, Planung und Steuerung eines Projektes haben erheblichem Einfluss auf das Ergebnis des Projekts. Die Erstellung eines Meilensteinplans mit wichtigen Zwischenschritten, die zum Projekterfolg führen, ist eine der Aufgaben des Projektmanagements. Der Plan sollte mit Fortschreiten des Projektes überprüft und gegebenenfalls iteriert werden. Die Gruppenmitglieder gilt es zu kontrollieren und motivieren. So sollen Engpässe und kritische Aktivitäten aufgedeckt und rechtzeitig behandelt werden.

Verantwortliche: Christelle Kamga, Carole Tchuenkam.

3.2.2. Außenbeauftragter

Der Außenbeauftragte fungiert als Schnittstelle zu außenstehenden Personen und Organisationen, die nicht bereits durch andere Projektrollen abgedeckt sind. So ist er beispielsweise dafür zuständig, Anfragen zu beantworten, die die Projektgruppe über die Website erreichen.

Verantwortlicher: Paul Kröger

3.2.3. Serveradministrator (Redmine & Wiki)

Die Serveradministratoren verwalten den der Projektgruppe zur Verfügung gestellten Server. Zu ihren Aufgaben gehören die Installation, Konfiguration und kontinuierliche Pflege aller benötigten Programme, Support bei Störungen und Serverausfällen. Außerdem sind die Serveradministratoren die Ansprechpartner bei Supportanfragen, wie z.B. beim Zurücksetzen eines Passworts.

Verantwortliche: Christof Schlaak; Simon Ortmann

3.2.4. Dokumentationsbeauftragter

Der Dokumentationsbeauftragte verwaltet und strukturiert alle Dokumente, die im Laufe des Projekts entstanden sind. Er leitet Maßnahmen ein, welche die Qualität der Dokumente sicherstellen, wie das Überprüfen der Dokumente durch Reviews. Außerdem ist er der richtige Ansprechpartner bei Fragen bezüglich der Dokumentation. Er sorgt dafür, dass die Dokumentation parallel zu Planung und Entwurf organisiert wird.

Verantwortlicher: Tim Baalman

3.2.5. Webmaster

Um Informationen über das Projekt und die Projektgruppe auch außenstehenden Personen zugänglich machen zu können, gibt es die Website. Diese ist unter der Adresse http://mops.informatik.uni-oldenburg.de erreichbar. Eine stetige Aktualisierung der angebotenen Informationen gehört zu den Aufgaben des Webmasters.

Verantwortlicher: Faisal Alotaibi

3.2.6. Testbeauftragter

Der Aufgabenbereich des Testbeauftragten umfasst das Vorbereiten und Koordinieren der Testfahrten (Trockentest und Seetest), sowie das Verfassen der Testberichte und der Testdokumentation. Er leitet die analytische Besprechung nach einer Testfahrt, um Verbesserungen am Ablauf sowie am MOPS besser und schneller koordinieren zu können.

Verantwortlicher: Matthias Esser

3.2.7. SCRUM-Master

Der SCRUM-Master überwacht den SCRUM-Prozess und dessen konkrete Implementierung. Er steht dem Projektmanagement bei der Erfüllung ihrer Aufgaben zur Seite und unterstützt dabei den SCRUM-Prozess zu verstehen.

Verantwortlicher: Weert Stamm

3.2.8. ICBM Kontakt

Im Rahmen einer Abschlussarbeit am Institut für Chemie und Biologie des Meeres wird ein Konzept für einen Sensorkäfig entwickelt, welcher auch mit dem MOPS eingesetzt werden soll. Daher gibt es eine Kooperation zwischen der Projektgruppe und einem Studenten des Instituts. Der Rolleninhaber ist Ansprechpartner für den Studenten und seinen Betreuer und leitet Anfragen kurzfristig an die Projektgruppe weiter.

Verantwortlicher: Thomas Hellkamp

3.2.9. Bootsoperator

Der Bootsoperator überwacht den korrekten Aufbau des Bootes, sowie die Einhaltung von Sicherheitsmaßnahmen beim Aufbau und dem Zuwasserlassen. Er überwacht bei Seetests die Position des Bootes und stellt sicher, dass das Boot weder Menschen noch sich selber in Gefahr bringt. Bei Gefahr greift er mit der Fernbedienung ein und wendet diese ab.

Verantwortlicher: Conrad Fifelski

3.3. Projektplanung

Die Arbeit an dem Projekt MOPS muss organisiert und strukturiert werden. Es ist ein zeitlicher Ablaufplan nötig, damit der Zeitbedarf, die Ressourcen und die Kosten eingeteilt und verwaltet werden können. Die Planung dient außerdem dazu, dass frühzeitig Probleme bezüglich der Einhaltung der Fristen und der Zuweisung der Arbeitskräfte erkannt werden können.

Zu diesem Zweck wurden Meilensteine und Aufgabenpakete definiert, die den Ablauf des Projekts unterstützen. Die Redmine-Plattform hat sich für diese Strukturierung als gute Unterstützung herausgestellt. Die Abbildung **??** zeigt einen Auszug aus dem von Redmine generierten Gantt-Diagramm.

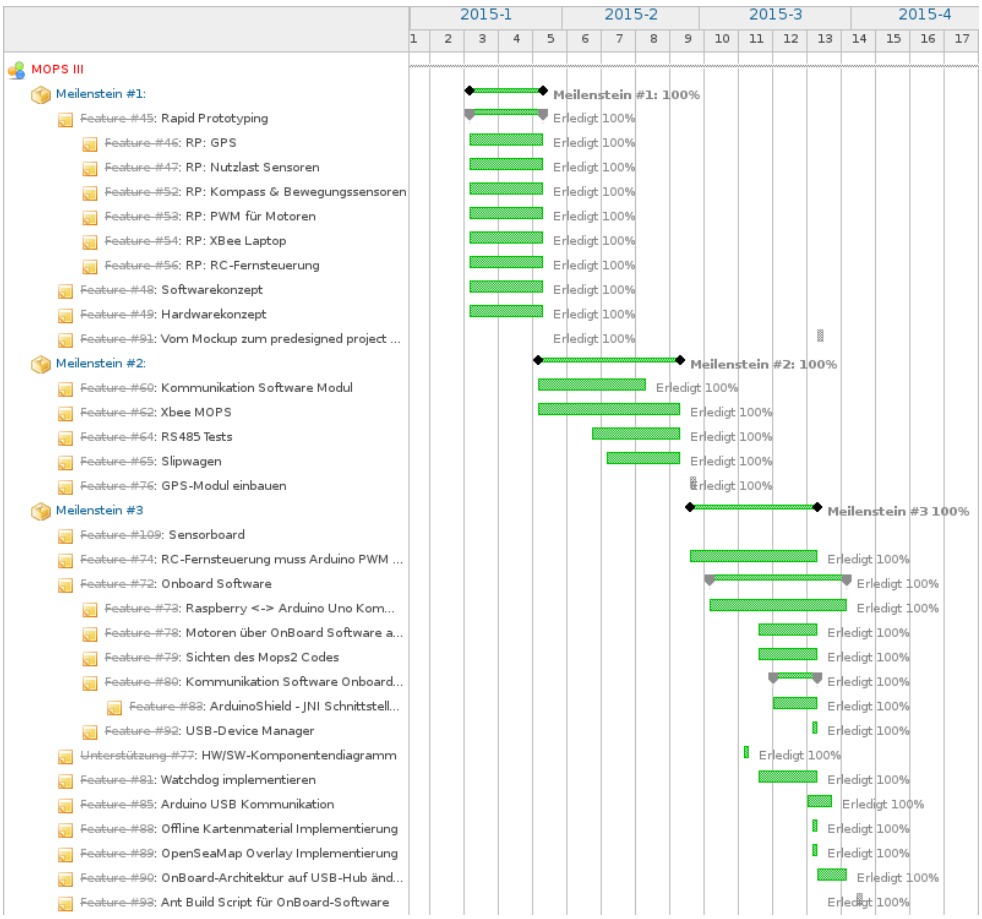

Abbildung 3.1.: Ausschnitt aus dem Gantt-Diagramm aus Redmine.

3.3.1. Meilensteine

In den folgenden Abschnitten werden die Meilensteine dieses Projekts aufgeführt. Sie haben einen zeitlichen Abstand von ein paar Wochen.

Seminarphase

Die erste Phase der Projektgruppe war die Seminarphase. Sie diente zur Einarbeitung in die bestehende Hardware und Software des MOPS II und weitere Themen, wie Projektmanagement, rechtliche Grundlagen, Umwelt, Bahnregelung und weitere Projekte.

Im Anschluss daran folgten noch Einrichtungsarbeiten am SVN, Wiki und Redmine als Arbeitsumgebung.

Meilenstein 1

Dieser Meilenstein widmet sich dem Rapid Prototyping. Es wurden schnell erste musterhafte Lösungsansätze für die Themen GPS, Kompass, XBee, Motoren, Payloadsensoren und Fernbedienung umgesetzt. Außerdem wurden neue Konzepte für die Software- und Hardware-Architektur entworfen, deren Schwerpunkt auf der Stabilität und Zuverlässigkeit einer gut erweiterbaren MOPS-Plattform lag.

Meilenstein 2

Die Vorgruppe MOPS II hatte Schwierigkeiten mit der Stabilität. Durch das Umschalten der Kommunikation von WLAN auf XBee wurde ein Absturz des Systems ausgelöst. Daher wurde die Kommunikations-Komponente der Onshore- und Onboard-Software in diesem Meilenstein komplett neu entwickelt. In diesem Zusammhang fiel auch eine Neuentwicklung der Ansteuerung der Hardware XBee-Module an. Es wurde ein Protokoll zum Ansprechen der Sensoren getestet und das GPS-Modul wurde in das Boot integriert.

Meilenstein 3

Das Ziel dieses Meilensteins ist, dass der MOPS (mithilfe einer Fernsteuerung) fahren soll. Dazu musste der Ardunio Uno für die Motorsteuerung und die Kommunikation zwischen Arduino und Raspberry eingerichtet werden. Außerdem musste die Fernbedienung inklusive Empfänger mit der Onboard-Hardware verbunden werden. Ein erstes Stabilitätskonzept wurde mithilfe eines WatchDog implementiert. Parallel dazu wurde das neue Konzept für die Onshore-Software umgesetzt und eine erste Version dieses Programms, welche bereits die Definition von Sperrgebieten unterstützt, entwickelt. Vorbereitend auf den ersten Seetest, wurde eine Regelung programmiert, damit der MOPS in der Lage ist eigenständig Wegpunkte anzufahren. Diese autonome Fahrt funktionierte ohne Bahnplanung, also ohne Kollisionsvermeidung.

Seemeilenstein 3b

Bis zu dem Seemeilenstein 3b wurde der erste Seetest vorbereitet und organisiert. Dazu musste die Elektronik auf dem Boot gegen Spritzwasser abgesichert werden. Die Steckverbindungen und ein Notaus wurden eingebaut und die Kompasskalibrierung umgesetzt.

Seemeilenstein 3c

Für den nächsten Seetest wurden entdeckte Fehler korrigiert und neue Features hinzugefügt. Erste Grundbausteine für die Integration der Messungen mit den Sensoren wurden gelegt. Die Onboard-Software wurde neu entworfen, damit sie zuverlässiger und stabiler läuft (siehe Kapitel Softwarearchitektur Onboard).

Meilenstein 4

Dieser Meilenstein schließt mit einem MOPS ab, der autonom fahren kann. Dazu gehören die Bahnplanungs- und Bahnregelungskomponenten und weitere Regler. Es kann eine Home-Position definiert werden, zu der der MOPS nach Durchführung der Mission zurückkehrt (Return-To-Home). Der Zustand des MOPS wird ausführlich in der Onshore-Software angezeigt (Monitoring). Es wurden Hardware-Komponenten angefertigt und professionell geätzt, sodass Wackelkontakte oder ähnliche Störungen und Ausfälle vermieden werden. Bis zu diesem Meilenstein wurden mehrere Seetests und Trockentests durchgeführt.

Meilenstein 5

Mit Abschluss dieses Meileinsteins war geplant, dass der MOPS die Sensordaten der Payload-Sensoren sammeln und speichern kann. Die Onshore-Software kann diese Werte anzeigen. Es können weitere Sensoren über ein Bussystem (Plug-&-Play) angeschlossen werden. Auf dem Weg zu diesem Meilenstein wurden einige Seetests durchgeführt.

Meilenstein 6

Der letzte Meilenstein ist auf das Projektende gelegt. Er schließt das Projekt mit den Abschlusspräsentationen, der Dokumentation und den letzten Seetests ab.

3.4. Projekt Roadmap

Das Projekt erstreckt sich über das Wintersemester 2014/2015 und das Sommersemester 2015.

Es beginnt mit einer Seminarphase, in welcher eine grundlegende Einarbeitung in das Thema erfolgt. Die Projektgruppenmitglieder erarbeiten eine Seminararbeit zu einem Thema, um den anderen Mitgliedern Wissen zu vermitteln. Nach der Bearbeitungsphase werden die Themen im Plenum präsentiert. Zusätzlich werden die Seminararbeiten im Wiki dokumentiert. In der folgenden Tabelle sind die Themen aufgelistet.

Verfasser	**Thema der Seminararbeit**
Tim Baalmann	Hardware
Christof Schlaak	Onboard-Software
Weert Stamm	Onshore-Software
Simon Ortmann	Sensorik
Conrad Fifelski	Rechtlicher Rahmen
Carole Tchuenkam	Umwelt
Faisal Saihan Alotaibi	Schiffssensoren
Matthias Esser	Verhalten
Paul Kröger	Bahnregelung
Christelle Kamga	Projektmanagement / Entwicklungsmanagement
Thomas Hellkamp	Weitere Projekte

Tabelle 3.1.: Verteilung der Seminararbeitsthemen

3.5. Vorgehensweise der Projektgruppe

Ein Vorgehensmodell beschreibt die Vorgehensweise der Softwareentwicklung und ist ein ausgefeilter Plan mit zeitlichen und sachlichen Vorgaben. Das Prozessmodell setzt die Reihenfolge der einzelnen Schritte mit den jeweiligen Ergebnissen fest. Es regelt welche Rolle zu welchem Zeitpunkt welche Aktivität in einem Projekt erledigt. Ein Vorgehensmodell dient zur Steuerung der Softwareentwicklung von der Konzeption bis zum Einsatz [4]. Um ein für MOPS III effizientes Vorgehensmodell auszusuchen, wird während der Seminarphase ein Vergleich zwischen zwei agilen Prozessmodellen gemacht: SCRUM und Extreme Programming (XP). Die Entscheidung fällt auf das SCRUM Modell, dessen Eigenschaften unserem Ziel besser dienen als XP. SCRUM sorgt für eine dynamische und flexible Entwicklung und Entfaltung des Projekts. Im Laufe des Projekts wird zusätzlich das Rapid-Prototyping-Verfahren angewendet. Rapid Prototyping ist ein Überbegriff von verschiedenen Verfahren zur schnellen Herstellung von Musterbauteilen ausgehend von Konstruktionsdaten (Vgl. https://de.wikipedia.org/wiki/Rapid_Prototyping). Wir testen hierbei einzelne Teile wie den Kompass, die Payloadsensoren, das GPS und die RC-Fernbedienung in kleinen Anwendungen um deren Funktionsweise zu verstehen. Dieses Verständnis hilft uns später die Komponenten miteinander zu verbinden.

Die Arbeit im Projekt wird wie folgt gestaltet. Es wird ein regelmäßiges Treffen, einmal pro Woche, festgelegt, bei dem alle Mitglieder anwesend sind. Bei jedem Treffen werden ergebnisorientierte Protokolle erstellt und es gibt einen Moderator. Die Rollen von Protokollant und Moderator werden von den Mitgliedern in alphabetischer Reihenfolge der Nachnamen wahrgenommen. Die Protokolle der Sitzungen werden auf einer Seite im Wiki eingestellt, um Entscheidungen zu archivieren und abwesende Mitglieder über den Verlauf des Projektes zu informieren. Der Moderator erstellt vor der Sitzung eine Tagesordnung um ein zügiges und effizientes Arbeiten zu ermöglichen. Die Sitzungen beginnen mit einem „Weekly SCRUM" (vergleichbar zum Daily SCRUM). Jedes Mitglied fasst kurz zusammen, was es in der vergangenen Woche gemacht hat und was für die kommende geplant ist. Danach werden die

Aufgaben und Arbeitspakete erstellt und verteilt. Das Treffen dient zudem als Schnittstelle zwischen den Betreuern des Projekts und der Projektgruppe.

Neben den regulären Sitzungen gibt es Arbeitstreffen, bei denen größere Probleme im Kollektiv gelöst werden. Kleinere Teams bearbeiten hier ihre gemeinsamen Aufgaben.

Alle Aufgaben werden in Form von Tickets im Redmine organisiert. Die Tickets werden dann dem jeweiligen Sprint zugeordnet.

3.6. Finanzielle Ressourcen

Die folgende Tabelle enthält alle Teile, die im Laufe des Projekts bestellt wurden. Hierbei handelt es sich um Bauteile, sowie Ersatzteile für ausgefallene Hardware.

Anzahl	**Produktbeschreibung**	**Stückpreis (€)**
1	Arduino (XBee) Shield für Raspberry Pi	40,00
1	RC-Fernbedienung	90,00
1	Not-Aus Vorrichtung	14,60
1	CE-Stecker für die Motoren	66,72
1	CE-Kupplung für die Motoren	84,65
1	Slipwagen	154,00
1	FTDI Breakout 3,3 V	11,95
1	Raspberry Pi 2 Model B	38,50
1	USB 2.0 7-Port HUB	34,95
2	Arduino Uno Rev. 3	23,95
1	Platine V1.0 Arduino Motor	4,66
1	Kfz-Relais	12,71
1	Relaissockel	5,38
1	Schalter	10,71
1	2-Pol Buchse, 90°	0,39
1	2-Pol Buchse, gerade	0,39
2	2-Pol Stecker	1,15
10	Kondensator	0,26
1	5V-Spannungregler	6,15
3	Kondensator	0,27
3	Kondensator	0,08
1	Induktivität	0,79
1	Schottky-Diode	0,26
2	2-Pol Buchse, 90°	0,39
2	2-Pol Stecker	1,15
2	3-Pol Buchse, 90°	0,64

2	3-Pol Stecker	1,79
1	5-Pol Buchse, 90°	1,02
1	5-Pol Stecker	2,81
3	Diode	0,00
1	Kondensator	0,00
2	p-Kanal MOSFET	0,40
3	Arduino Buchsen	1,50
1	Platine V1.0 Arduino Sensor	3,00
1	Platine V1.0 Leistungselektronik	4,00
25	Abdeckkappen 6 30 × 30, schwarz	0,50
10	Winkel-Abdeckkappen 6 60 × 60, schwarz	0,78
50	Winkel-Abdeckkappen 6 30 × 30, schwarz	0,58
1,5m	Profil 6 30 × 30 1N leicht,	11,66
Gesamtkosten		697,57

Tabelle 3.2.: Kostentabelle der Bauteile

4. Onshore-Software

Um die Missionen des MOPS zu planen, zu steuern und zu überwachen wird die Onshore-Software verwendet. Sie ermöglicht außerdem die Kalibrierung des Kompasses und die Einstellung von erweiterten Parametern, welche die Wegplanung und das Verhalten des MOPS beeinflussen.

4.1. Anwendung

4.1.1. Inbetriebname XBee Modul

Zur Kommunikation der Onshore-Software mit dem MOPS muss das XBee Modul am verwendeten Rechner angeschlossen und eingerichtet sein. Dazu wird das Sparkfun XBee Explorer USB Board mit aufgesteckter XBee benötigt. Dieses Board bietet die Möglichkeit über USB-Ports mit der XBee zu kommunizieren.

Konfiguration der XBee

Zur Konfiguration der XBee kann das Programm XCTU (http://www.digi.com/products/wireless-wired-embeded-solutions/zigbee-rf-modules/xctu) von Digi, dem Hersteller der XBee, genutzt werden. Dieses Programm gibt es bisher nur für Windows und Mac.

Windows

Unter Windows wird ein Treiber (http://www.ftdichip.com/Drivers/VCP.htm) benötigt. Das Werkzeug XCTU kann helfen, die XBee über den seriellen Port zu finden. Dies ist rein empirisch, dennoch hat es in den Tests geholfen. Die XBee sollte aus dem XCTU-Programm entfernt werden, bevor sie anderweitig verwendet wird, da keine multiplen Verbindungen möglich sind. Falls dennoch keine Verbindung zu der XBee via Serial Port möglich ist, so hilft ein Neustart des PC oft diesen Umstand zu beenden.

Für weitere Informationen gibt es ein umfangreiches Tutorial (https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu).

Um das XBee Modul mit der Onshore-Software zu nutzen, ist es notwendig im Geräte-Manager den COM-Port der XBee auf COM5 zu stellen.

Linux

Unter Linux sind in der Regel keine zusätzlichen Treiber notwendig. Es ist allerdings wichtig, dass der Benutzer, der das Programm ausführt, die benötigten Rechte hat und in den Gruppen `dialout` und `uucp` (je nach Distribution) ist. Ein Ausführen als `root` sollte in jedem Fall funktionieren.

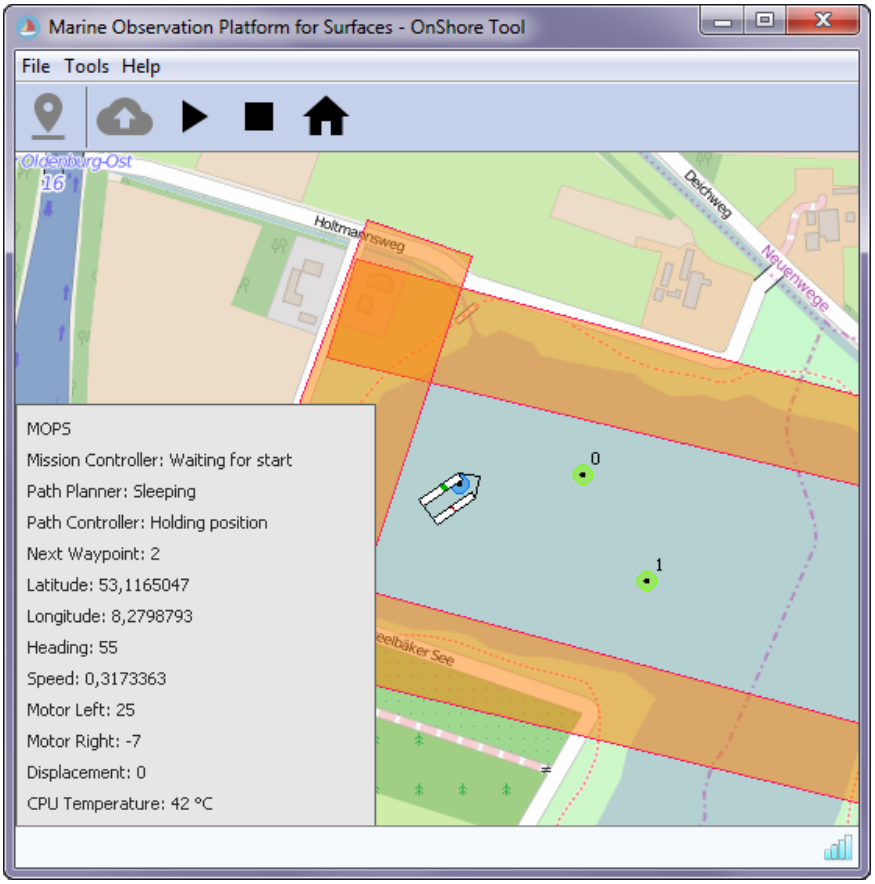

Abbildung 4.1.: Bildschirmfoto der Onshore-Software

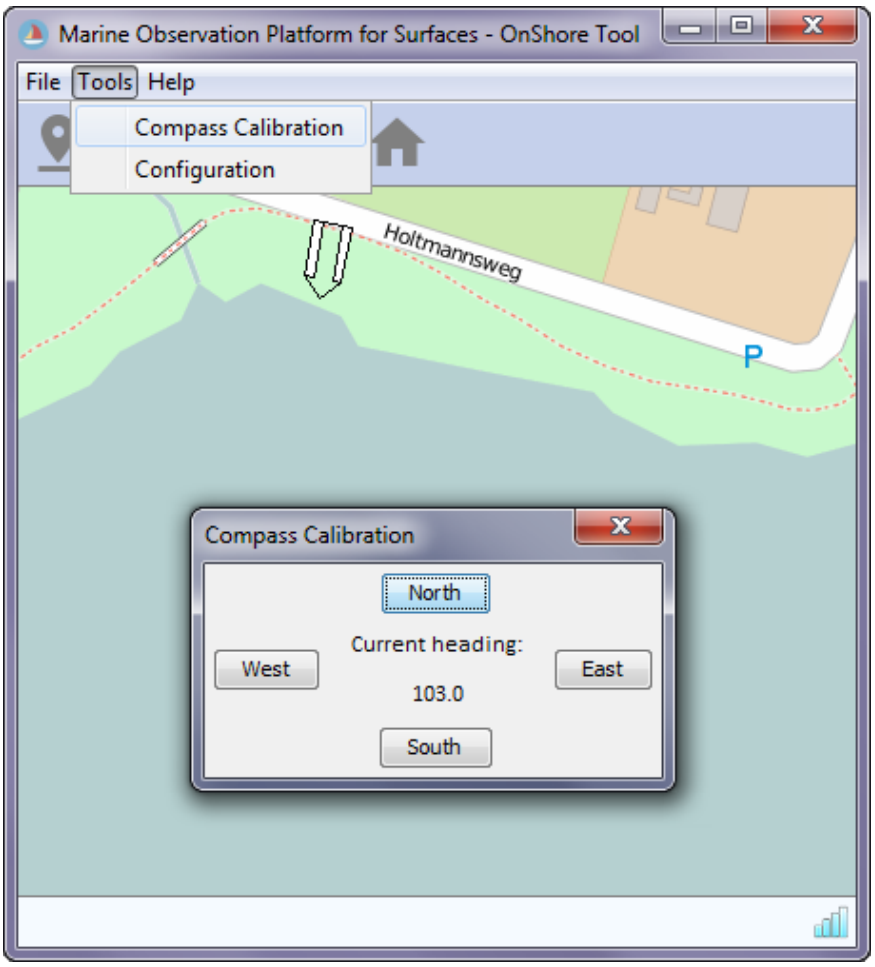

Abbildung 4.2.: Kalibrierung des Kompass in der Onshore-Software

4.1.2. Kompass Kalibrierung

Bevor eine Mission ausgeführt wird, sollte der Kompass des MOPS kalibriert werden. Dazu muss die Onboard-Software des MOPS gestartet werden und eine Verbindung zwischen MOPS und einem Rechner mit angeschlossenem XBee Modul und laufender Onshore-Software bestehen. In der Onshore-Software muss nun aus dem Menü **Tools** der Menüpunkt **Compass Calibration** ausgewählt werden.

Der MOPS muss nun mit einem externem Kompass nacheinander in alle vier Himmelsrichtungen ausgerichtet werden. Wenn Smartphones verwendet werden, ist es empfehlenswert sich nicht auf die Angaben eines einzigen zu verlassen. In der Onshore-Software muss dann im geöffneten **Compass Calibration** Dialog jeweils auf den entsprechenden Button gedrückt werden wenn der MOPS ruhend in eine Himmelsrichtung zeigt.

Beispielhaftes Vorgehen:

1. MOPS mit externem Kompass nach Norden ausrichten
2. Klick auf **North** im **Compass Calibration** Dialog
3. MOPS nach Osten ausrichten

4. Klick auf **East**
5. MOPS nach Süden ausrichten
6. Klick auf **South**
7. MOPS nach Westen ausrichten
8. Klick auf **West**

Es kann dann mit dem unter **Current Heading** angezeigten Wert überprüft werden ob die Kalibrierung fehlerfrei durchgeführt wurde. Das Heading sollte bei nach Norden ausgerichtetem MOPS ungefähr bei 360 bzw. 0 Grad liegen (Osten - 90 Grad, Süden - 180 Grad, Westen - 270 Grad). Nach erfolgreicher Kalibrierung kann der Dialog geschlossen werden. Die Kalibrierung sollte sicherheitshalber an jedem Missionstag wiederholt werden.

4.1.3. Missionsplanung

Eine Mission besteht aus Wegpunkten, Sperrzonen und einer Home-Position. Nach Übertragung und Start der Mission fährt der MOPS die Wegpunkte in der Reihenfolge ihrer Erstellung ab. Die Route wird dabei so geplant, dass die Sperrzonen umfahren werden. Bei der Festlegung der Sperrzonen gilt zu beachten, dass der MOPS beim Umfahren dieser deren Kanten tangieren kann. Es sollte daher ein ausreichender Puffer eingeplant werden. Der MOPS kehrt zur Home-Position zurück wenn bei laufender Mission der entsprechende Button in der Toolbar angeklickt wird oder wenn der letzte Wegpunkt erreicht wurde und die Mission beendet ist.

Steuerung

- **Steuerung** + **Linksklick** → Neuen Wegpunkt setzen
- **Umschalt** + **Linksklick** → Eckpunkt einer Sperrzone setzen, bei loslassen der Umschalttaste wird die Sperrzone erstellt
- **Entfernen** → Ausgewählten Wegpunkt, Sperrzone oder Home-Position löschen

Mit der linken Maustaste können Objekte (MOPS, Wegpunkt, Sperrzone oder Home-Position) ausgewählt werden, um erweiterte Informationen anzuzeigen oder sie zu löschen. Mit gedrückter linker Maustaste kann sich über die Karte bewegt werden und mit dem Mausrad die Zoomstufe verändert werden. Ein Klick auf das Verbindungssymbol in der unteren rechten Ecke lässt die Ansicht auf den MOPS zentrieren.

Toolbar

In der Toolbar befinden sich Schaltflächen um die Mission zu planen, sie zu übertragen und den MOPS während der Durchführung zu kontrollieren.

1 Die erste Schaltfläche ist aktiv wenn ein gesetzter Wegpunkt angeklickt ist. Durch Betätigung wird der ausgewählte Wegpunkt zur Home-Position der Mission umgewandelt, erkennbar daran, dass er nun in blauer Farbe dargestellt wird. Gibt es bereits eine Home-Position, wird diese ersetzt.

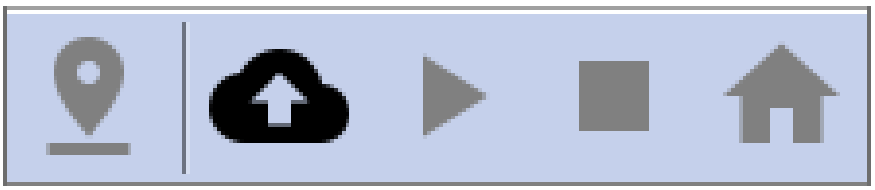

Abbildung 4.3.: Toolbar der Onshore-Software

2 Die zweite Schaltfläche löst die Übertragung der Mission an den MOPS aus. Dazu muss das XBee Modul korrekt eingerichtet, der MOPS gestartet und in Funkreichweite sein.

Die folgenden drei Schaltflächen werden aktiviert wenn eine Mission erfolgreich übertragen wurde. Sie erlauben es den MOPS während der Missionsausführung zu kontrollieren.

3 Die Schaltfläche mit dem Wiedergabe-Symbol startet die übertragene Mission, oder setzt eine pausierte Mission fort. Sobald eine Mission ausgeführt wird, verändert sich das Symbol der Schaltfläche zu einem Pausen-Symbol. Die Schaltfläche erlaubt es dann eine aktive Mission zu pausieren.

4 Die Schaltfläche mit dem Stop-Symbol bricht die laufende Mission ab und veranlasst den MOPS dazu seine Motoren abzuschalten und zu treiben.

5 Die letzte Schaltfläche, mit dem Symbol eines Hauses, bricht die Mission ebenfalls ab aber veranlasst den MOPS dazu zur Home-Position zurückzukehren und diese zu halten.

Speichern und Laden

In dem **File** Menü in der Menüleiste befinden sich weitere Optionen um die Missionsplanung zu unterstützen. Die Option **New** beginnt eine neue Missionsplanung, d.h. alle gesetzten Wegpunkte, Sperrzonen und die Home-Position werden gelöscht und gehen verloren wenn die Mission nicht zuvor gespeichert wurde. **Open** und **Save** erlauben es eine Mission (Wegpunkte, Home-Position und Sperrzonen) auf einem Datenträger abzuspeichern und wieder zu laden.
Die Optionen **Export Danger Zones** und **Import Danger Zones** ermöglichen es die definierten Sperrgebiete abzuspeichern und in anderen Missionen wiederzuverwenden.

4.1.4. Erweiterte Konfiguration

Um erweitertete Einstellungen am MOPS vorzunehmen kann in dem **Tools** Menü die **Configuration** geöffnet werden. Nach dem Öffnen des Dialogs muss zunächst auf den Button **Request Current Parameters** gedrückt werden. Bei bestehender Verbindung zum MOPS werden nun die aktuellen auf dem MOPS verwendeten Parameter übertragen und im Dialog angezeigt. Um die Handhabung zu vereinfachen sind die Parameter farblich markiert. Grün hinterlegte Parameter sind die aktuell auf dem MOPS verwendeten Werte. Orange Parameter sind zulässig, wurden aber noch nicht an den MOPS übertragen.

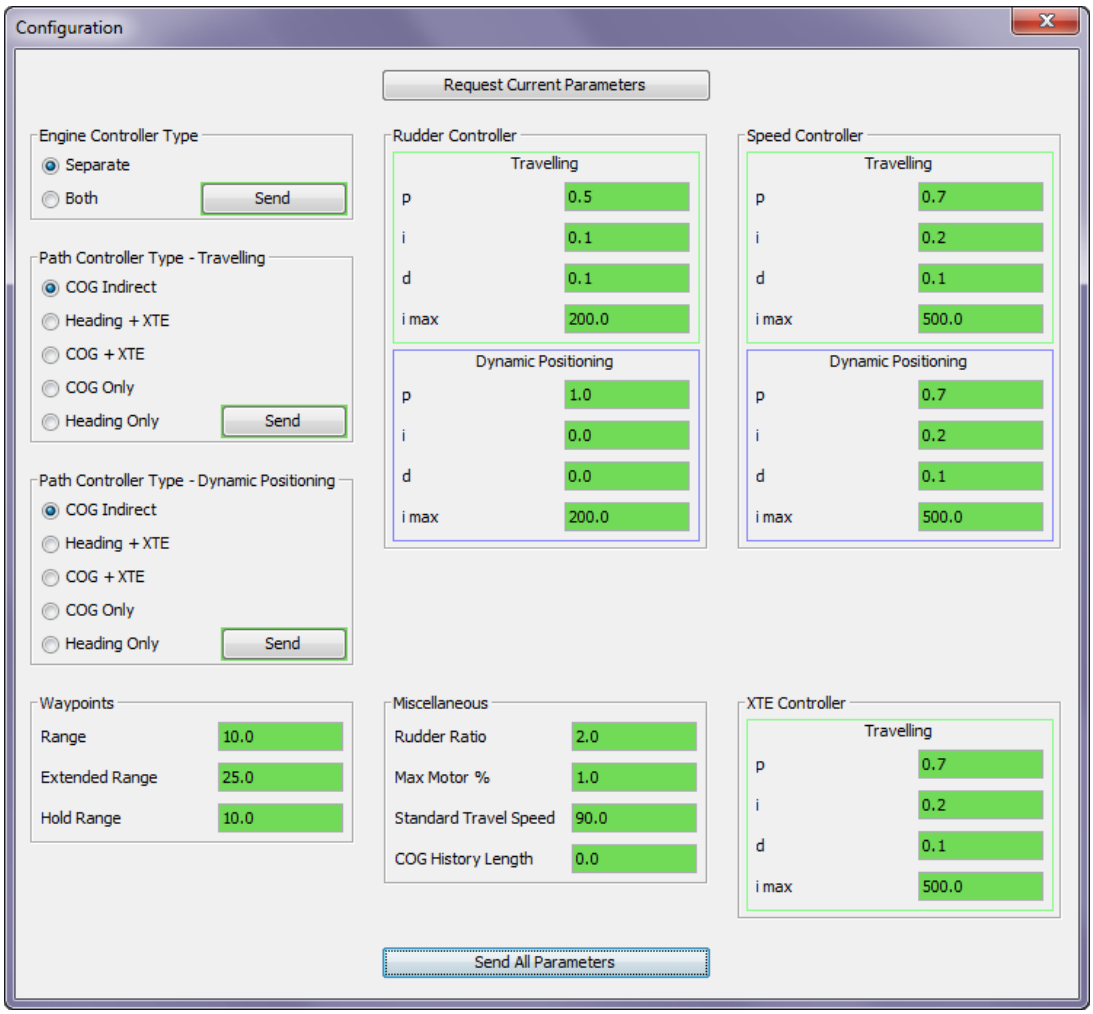

Abbildung 4.4.: Fenster fuer die Parameterkonfiguration

Rot hinterlegte Parameter sind unzulässig (z.B. ausserhalb des festgelegten Wertebereiches oder fehlerhafte Eingaben wie Strings anstatt Zahlen) und können nicht übertragen werden.

Nach einer zulässigen Änderung eines Parameters kann dieser mit der **Enter Taste** bzw. einem Klick auf den **Send** Button an den MOPS übertragen werden. Nach erfolgreicher Übertragung wechselt die Farbe des Parameters von Orange zu Grün. Ein Klick auf den **Send All Parameters** Button sendet alle zulässigen Parameter an den MOPS. Nach abgeschlossener Konfiguration kann der Dialog geschlossen werden. Die neuen Parameter werden persistent auf dem MOPS gespeichert.

4.2. Entwicklung und Hintergrund

Die bestehende Software aus den vergangenen Projektgruppen wurde zu Beginn der Projektgruppenphase evaluiert. Sie wurde größtenteils in der ersten MOPS Projektgruppe erstellt, während zur selben Zeit ein funktioneller MOPS Prototyp gebaut wurde. Sie war daher, wenn auch eine beeindruckende Leistung der ersten Projektgruppe, für die Ziele der dritten

Projektgruppe nur schwierig zu verwenden. Ungenügende Erweiterbarkeit, Fehleranfälligkeit und eine unübersichtliche Benutzeroberfläche veranlassten uns die Onshore-Software vollständig neu zu entwickeln.

4.2.1. Kompilieren

Die Software wurde auf Basis von Java 8 entwickelt, welches auf dem System installiert sein muss. Für das Kompilieren steht die Ant-Build-Datei `de.uniol.mops.onshore/build.xml` zur Verfügung. Das Default-Target kompiliert die Projekte `de.uniol.mops.onshore`, `de.uniol.mops.model`, `de.uniol.mops.communication` und `de.uniol.mops.util` und fügt die Native Libraries für Windows 64-Bit und Linux 64-Bit hinzu. Falls das Programm für 32-Bit Architekturen (Windows und Linux) kompiliert werden soll, muss das Target `main32` ausgewählt werden.

4.2.2. Architektur

Die Architektur der Onshore-Software wurde nach dem Model View Controller Prinzip modelliert. Alle Modelle sind allerdings in dem Paket `de.uniol.mops.model` zu finden, damit diese von anderen Softwaremodulen mitbenutzt werden können. Die Architektur ermöglicht eine schnellere Erweiterung des Systems und eine übersichtliche Struktur.

4.2.3. GUI und Kartenmaterial

Die grafische Oberfläche der Software wurde mit Hilfe der Java Swing API erstellt. Die Anzeige des Kartenmaterials erfolgt mit Hilfe des JXMapViewers2 (https://github.com/msteiger/jxmapviewer2 Dies ist eine nicht-kommerzielle Weiterentwicklung des eingestellten SwingX-WS Projekts und ist unter der GNU Lesser General Public License (LGPL) lizensiert. Das Kartenmaterial wird von OpenSeaMap http://www.openseamap.org bezogen, diese stellt „für den Seemann interessante nautische und touristische Information" auf einer Landkarte dar.

Die Darstellung des MOPS wird direkt auf die Karten gezeichnet. So können Informationen wie Ausrichtung, Motoransteuerung und momentane Position visuell dargestellt werden.

5. Onboard-Software Architektur

Die Onboard-Software wurde in MOPS III vollständig neustrukturiert. Die Abbildung 5.1 beschreibt den Aufbau und die Funktionsweise.

5.1. Motivation

Software, die mit vielen Sensoren und Verbindungen zu anderen Geräten arbeitet, hat oftmals das Problem, dass das Multithreading Überhand nimmt. Multithreading kann sehr schnell unübersichtlich werden. Um Daten auszutauschen muss viel zwischen den Threads synchronisiert werden und schnell kann die Komplexität so sehr ansteigen, dass Deadlocks nicht weit entfernt sind. Auch das saubere Beenden von Threads kann durch Abhängigkeiten zu anderen Threads ein Problem werden. Um genau diese Probleme zu vermeiden, wurde eine neue Architektur für die Onboard-Software entwickelt.

5.2. Vorteile

Durch diese Umstrukturierung ergeben sich einige Vorteile (gegenüber der MOPS II Version):

- Übersichtlichtkeit
  - Die Komponenten kommunizieren nicht direkt untereinander (Netzstruktur), sondern über einen zentralen zugriffsgeschützten Knoten
- Codereduzierung
- Stabilität
  - Thread-Verklemmungen können (unter Befolgung der Strukturvorgaben) nicht mehr entstehen
  - Threads können einfach beendet werden
  - Zentrale Sammlung der Daten
  - Zustand des Programms wird in ByteFile gesichert und kann wiederhergestellt werden.
- Erweiterbarkeit
  - Neue Komponenten können einfach eingehängt werden. Die Struktur dieser ist durch eine Oberklasse vorgegeben

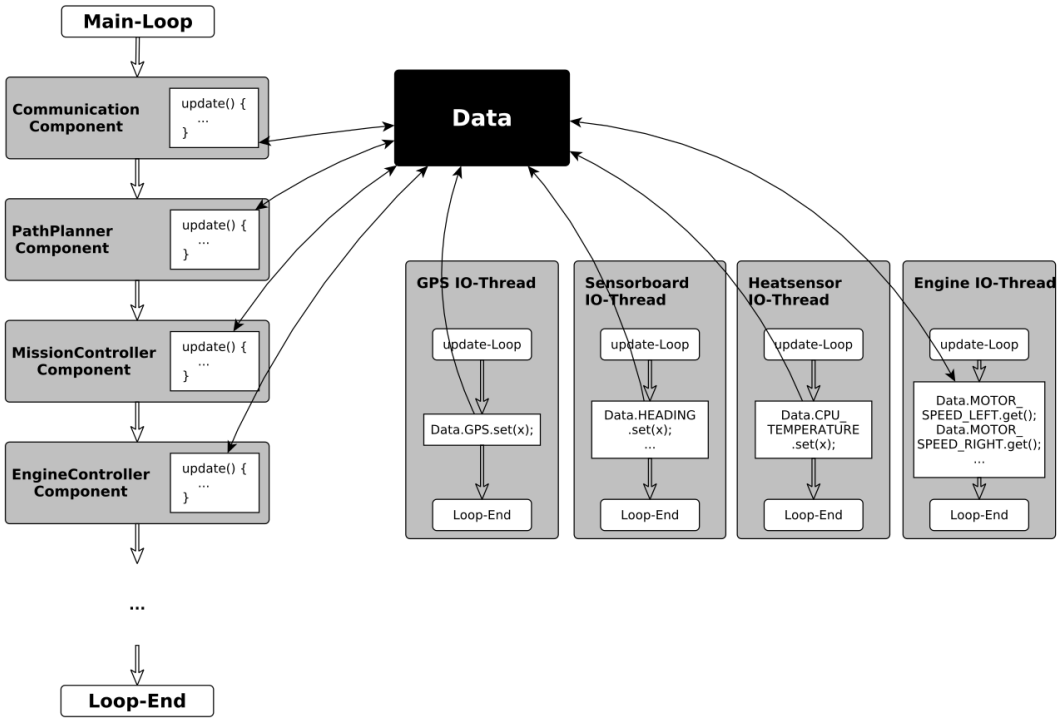

Abbildung 5.1.: Schematische Darstellung der Onboard-Software Architektur

  - bereits vorhandene Komponenten können einfach durch andere Komponenten ersetzt werden. Für die Simulation der Hardwaregeräte (GPS, Sensorboard, Motoren, ...) wurden die entsprechenden Komponenten durch simulierende Threads ersetzt.
  - Ein Life Cycle für alle Aufgaben erleichtert das Verständnis und verbessert die Leserlichkeit

- Wartbarkeit
- Performance messbar
  - Abgesehen von den IO-Threads werden alle Operationen über die Hauptschleife aufgerufen. In dieser kann dann (ähnlich wie bei einem Videospiel) die Anzahl der FPS bestimmt werden. Dadurch lässt sich die Leistungsauslastung abschätzen.

5.3. Komponenten

Die Architektur erlaubt es, Aufgaben von einzelnen Objekten, unabhängig von der restlichen Software, zu realisieren. Die Aufgabe wird dabei von einer Komponente (im Code „`Component`" genannt) bearbeitet. Die Komponente wird in die Software eingehängt und läuft im Hauptthread. Alle Komponenten werden nacheinander wiederkehrend ausgeführt, solange die Software ausgeführt wird. Sollen Daten zwischen Komponenten ausgetauscht werden, kann einfach auf die Felder der `Data` Klasse zugegriffen werden. Um eine Komponente anzulegen muss lediglich das Interface `IComponent` implementiert werden. Anschließend

kann man die Komponente in der `ComponentManager` Klasse in der `init` Methode einhängen.

5.4. IO-Threads

Für Threads, die Ein- und Ausgaben über Geräte benötigen, und damit den Hauptthread blockieren würden, gibt es die abstrakte Oberklasse `AbstractIO`. Nach der Erstellung müssen diese Klassen noch in der Klasse `IOManager` in der `init`-Methode instanziiert werden. Alle bisher benutzten seriellen Verbindungen sind über USB hergestellt. Dafür gibt es die vorgegebene Klasse `USBCommunication`.

5.4.1. Life Cycle

Den Komponenten, sowie den IO-Threads, liegt der gleiche Life Cycle zugrunde. Implementiert man das entsprechende Interface für Komponenten, oder leitet eine Klasse von AbstractIO ab, so bekommt man drei Methoden vorgegeben, die zu befüllen sind. Die `init` Methode wird ausgeführt, sobald die Komponente eingehängt wird. In ihr kann man einmalige Aktionen ausführen, die zu Beginn von Nutzen sind. Das Gegenstück zur `init` Methode ist die `destroy` Methode. Diese wird ausgeführt sobald das Objekt aus dem Manager ausgehängt oder die Software beendet wird. Dort können z.B. etwaige Verbindungen getrennt werden oder Speicher freigegeben werden. Die wichtigste Methode ist die `update` Methode. Diese Methode wird wiederkehrend aufgerufen. Hier sollte alles Wichtige, das während der Ausführung der Software laufen sollte, implementiert werden.

5.5. Data

Um Daten, sowohl zwischen den IO-Threads als auch den Komponenten auszutauschen, gibt es die Klasse `Data`. `Data` ist nach dem Shared-Memory Prinzip modelliert und erlaubt es jeder Klasse, jederzeit, aus jedem Thread auf Sie zuzugreifen. Die Felder werden alle einzelnd synchronisiert um die Gefahr in einen Deadlock zu laufen zu verringern. Dies hat weitere Vorteile, die beim Arbeiten mit Threads hilfreich sein können. Es gibt zu jeder Zeit in der Klasse Data eine gesamte Übersicht von momentanen Werten, unabhängig von den Threads, die diese Werte befüllen. Außerdem muss nicht mehr zwischen einzelnen Threads synchronisiert werden und komplizierte Multithreading Programmierung kann (fast vollständig, siehe IO-Threads) umgangen werden.

5.6. USB-Geräte-Manager

Um in den IO-Threads gezielt auf ein bestimmtes USB-Gerät zuzugreifen, ohne dass die Portbezeichnung bekannt ist, gibt es die Klasse `USBDeviceManager`. Sie untersucht das (Linux-only-) System nach angeschlossenen USB-Geräten und hält diese in einer Liste fest. Das Finden der Geräte wird mit einem Bash-Skript realisiert. Es wird in dem Java-Code aufgerufen und nimmt neben der Port-Bezeichnung (z.B. `/dev/ttyUSB0`) auch den Hersteller, die Modellbezeichnung und die ID auf. Diese Informationen sind in der Ausgabe durch ein Semikolon getrennt. Über die ID (oder falls nicht vorhanden: die Herstellerbezeichnung) kann also unabhängig vom tatsächlichen Hardware-Port das

gesuchte Gerät angesteuert werden. Die Belegung der USB-Stecker in dem USB-Hub ist damit beliebig.

Das Skript ist in dem Java Code als String einbettet, befindet sich aber auch im Ordner `tools` der Onboard-Software als Bash-Skript. Das Skript kann auf dem Raspberry Pi händisch (ohne Onboard-Software) ausgeführt werden, um die IDs von neuen angeschlossenen Geräten herauszufinden. Hier ist das Skript nocheinmal aufgeführt:

```
#!/bin/bash

for sysdevpath in $(find /sys/bus/usb/devices/usb*/ -name dev); do
(
  syspath="${sysdevpath%/dev}"
  devname="$(udevadm info -q name -p $syspath)"
  [[ "$devname" == "bus/"* ]] && continue
  eval "$(udevadm info -q property --export -p $syspath)"
  [[ -z "$ID_SERIAL" ]] && continue
  echo "/dev/$devname;$ID_SERIAL_SHORT;$ID_VENDOR;$ID_MODEL"
)
done
```

6. Hardware und Systemsoftware Onboard

In diesem Kapitel werden die Hardware-Komponenten auf dem MOPS beschrieben. Außerdem läuft auf vielen dieser Geräte eine angepasste Software, die eine bestimmte Aufgabe erfüllt.

6.1. Raspberry Pi

Der Raspberry Pi stellt die zentrale Hardware-Komponente an Bord des MOPS dar. Auf ihm läuft die Onboard-Software, welche die Ansteuerung der Geräte, die Durchführung der Mission, die Kommunikation mit dem Festland und viele weitere Aufgaben erledigt.

Die Software wurde auf Basis von Java 8 entwickelt. Zum einfachen Kompilieren kann die Ant-Build-Datei `de.uniol.mops.onboard/build.xml` benutzt werden. Es kompiliert zunächst die Quelldateien der Projekte für die Bahnplanungs-, Utility-, Model-, Communication- und Onboard-Software. Die Abhängigkeiten inklusive der nativen Bibliotheken (mittels `jarspliceplus`) der Projekte werden mit übernommen.

Nachdem die Software fertig kompiliert wurde, wird das jar-Paket mit `scp` auf den Raspberry in das Homeverzeichnis des Users 'pi' (`/home/pi`) übertragen. Dazu muss eine WLAN-Verbindung mit dem Funknetzwerk `MOPS` bestehen. Wenn die Übertragung erfolgreich war wird das Start-Skript für die Onboard-Software ausgeführt. Es wird im folgenden Abschnitt genauer beschrieben.

6.1.1. Konfiguration

In diesem Abschnitt werden die Einrichtungsschritte beschrieben, die durchgeführt wurden, um das Betriebssystem des Raspberry Pi für unsere Ansprüche zu konfigurieren.

Zugriff per Secure Shell (SSH)

Wenn eine Netzwerkverbindung zu dem Raspberry Pi besteht, kann per SSH auf das System zugegriffen werden. Die Verbindung kann über LAN-Kabel oder WLAN aufgebaut werden. Die Netzwerkkarte hat die statische IP-Adresse `192.168.1.1`. Für die kabellose Verbindung stellt der Raspberry Pi den WLAN-Access-Point `MOPS` zur Verfügung. Das WLAN-Passwort ist `uni-oldenburg-studenten`. Die Teilnehmer bekommen IP-Adressen von `192.168.42.2` bis `192.168.42.255`.

Für den Nutzer `pi` muss das Standartpasswort `raspberry` eingegeben werden. Über das Kommando `sudo su -` können Root-Rechte erworben werden.

Verbindung mit OFFIS-WLAN

Damit der Raspberry Pi Zugriff auf das Internet bekommt, kann das Shell-Skript `network/ wlan-offis.sh` (im Home-Ordner des Users pi) ausgeführt werden. Achtung: Der WLAN-Access-Point steht dann nicht mehr zur Verfügung. Das Shell-Skript `network/ wlan-access-point.sh` macht diese Einstellungen wieder rückgängig.

Onboard-Software Start Skript

Damit das Onboard-Programm mit einem Befehl gestartet werden kann, wurde ein Bash-Skript geschrieben. Es beendet alle bislang laufenden Instanzen des Programms und startet das Programm als User `root`. Dabei wird eine Logdatei mit einer eindeutigen ID und dem aktuellen Datum für das neu gestartete Programm angelegt. Das Datum kann je nach Einstellungen des Raspberrys falsch sein. Die Ausgabe auf der Console bleibt dennoch mithilfe von `tee` erhalten. Alle Nutzer, die per SSH auf dem Raspberry eingeloggt sind, werden über den Start und Stop des Programms informiert. Der Befehl `stopMops` beendet das Programm ohne ein Neues zu starten.

```
#!/bin/bash
DATE=$(date +%Y-%m-%d_%H-%M-%S)
LOGID=$(find /home/pi/log/ -maxdepth 1 -type f | wc -l)
LOGIDFILLED=$(printf "%04d" $LOGID)

sudo killall java > /dev/null 2>&1

printf "Onboardprogram started. Log-File-ID is %s" "$LOGIDFILLED" #
   | wall

#use absolute paths to make autostart work
sudo java -jar /home/pi/MOPS-Onboard-Fat.jar |
  tee "/home/pi/log/${LOGIDFILLED}_${DATE}.log" &&
  printf "Onboardprogram terminated. Log-File-ID is %s" "$LOGIDFILLED" #
   | wall &
```

Log-Dateien In dem Ordner `/home/pi/log` werden die Log-Dateien der Onboard-Software gespeichert. Der Befehl `showlastlog` zeigt den Inhalt der neuesten Log-Datei. Nach Ausführung dieses Befehls kann mit `grep` nach Schlüsselwörtern gesucht werden, z.B. `showlastlog | grep GPS`.

Autostart

Wenn der Raspberry startet wird das Skript `/etc/rc.local` ausgeführt. Es beinhaltet den Start des Onboard-Programms. Allerdings wird hierbei das Wurzelverzeichnis `/` als aktuelles Verzeichnis genommen, was zur Folge hat, dass die binäre Konfigurationsdatei `config.bin` auch in diesem Verzeichnis statt im Ordner `/home/pi` abgelegt wird.

Standbymodus von USB-Geräten deaktivieren

Im Internet beklagen sich viele Nutzer, dass Raspbian, das Betriebssystem auf dem Raspberry Pi, automatisch WLAN-Sticks in den Sleep-Modus versetzt. Obwohl wir den Fehler bei uns nicht bemerkt haben, haben wir zur Sicherheit diese Einstellung nach dieser Anleitung http://www.mikeslab.net/?p=178 deaktiviert. Für den Chipsatz 8192cu muss dazu die Zeile `options 8192cu rtw_power_mgnt=0` in die Datei `/etc/modprobe.d/8192cu.conf` eingetragen werden.

Automatischer Dateisystemcheck

Wenn der Raspberry unsachgemäß heruntergefahren wird, bspw. durch Unterbrechen der Stomversorgung, können Fehler im Dateisystem entstehen. Beim nächsten Boot wird das Betriebssystem versuchen diese Fehler zu korrigieren, verlangt aber für jeden gefundenen Fehler eine Bestätigung durch den Benutzer. Damit die Überprüfung des Dateisystems ohne Nutzereingabe selbstständig durchgeführt wird, muss folgende Zeile in die Datei `/etc/default/rcS` eingefügt werden:

```
# automatically repair filesystems with inconsistencies during boot
FSCKFIX=yes
```

6.1.2. CPU-Hitzesensor

Der Raspberry Pi befindet sich an Bord des MOPS in einer kleinen Box, die sich bei Sonneneinstrahlung erwärmen kann. Um festzustellen, ob der Raspberry Pi Gefahr läuft zu überhitzen, wird alle 3 Sekunden die CPU-Temperatur gemessen und in dem gemeinsamen Speicher `Data` abgelegt. Bei der nächsten Gelegenheit wird dieser Wert ans Festland übertragen und in der Onshore-Software als Wert „CPU Temperature" dargestellt.

Die Messung wird intern in Java durch das Ausführen eines Bash-Skripts (in Java als String gespeichert) realisiert. Die Ausgabe wird abgefangen und verarbeitet (vgl. USB-Geräte-Manager). Das Skript besteht aus nur einem Befehl, der Raspberry-spezifisch ist und daher auf anderen Systemen wahrscheinlich nicht funktionieren wird. Das folgende Skript befindet sich auch im Ordner `tools` der Onboard-Software:

```
#!/bin/bash

vcgencmd measure_temp
```

6.1.3. USB Hub

Da die 4 USB-Ports des Raspberry Pi 2 nicht ausreichten, wurde der UUGear 7-Port USB Hub http://www.uugear.com/product/high-speed-7-port-usb-hub-for-raspberry-pi/eingesetzt. Insgesamt stehen damit 10 Anschlüsse zu Verfügung.

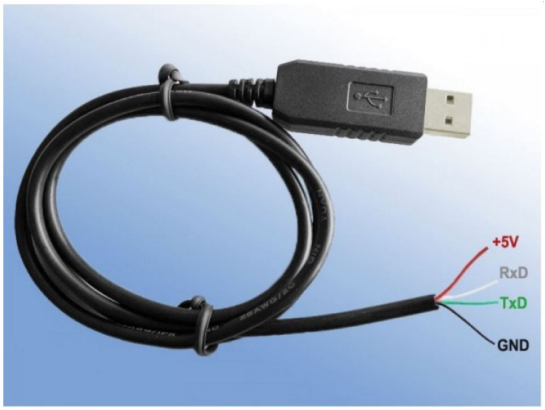

Abbildung 6.1.: USB-to-TTL Converter

6.2. Leistungselektronik

Der Sabertooth verstärkt die auf dem Motor Arduino Cape generierten PWM-Signale für die beiden Motoren.

6.2.1. Konfiguration des Sabertooth

Im Folgenden wird kurz beschrieben, wie man den Sabertooth zur Konfiguration mit der DEScribe Software verbindet. Hierbei handelt es sich um eine Zusammenfassung der mitgelieferten Quick Start Anleitung.

Zunächst wird ein USB-to-TTL Converter benötigt, der wie folgt mit dem Sabertooth verbunden werden muss:

Sabertooth	**USB-to-TTL Converter**
0V	GND (schwarze Ader)
S1	TX (grüne Ader)
S2	RX (weiße Ader)

Tabelle 6.1.: Anschlüsse des Sabertooth

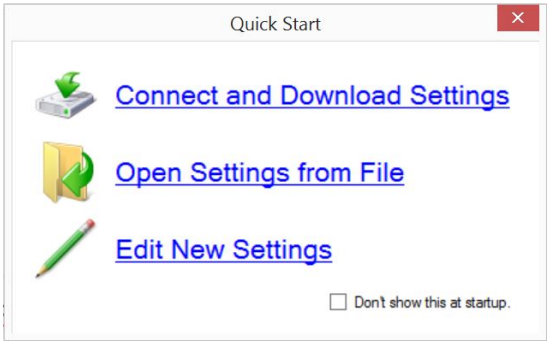

Abbildung 6.2.: DEScribe Quick Start Software

Die Belegung der Converteradern ist im Bild 6.1 verdeutlicht. Damit der Sabertooth nun auch konfiguriert werden kann, müssen die DIP-Switches noch in folgende Position gebracht werden:

- 1 und 2 OFF (Packet Serial mode)
- 3, 4, 5 und 6 ON (Adresse 128, optional aber empfohlen)

Nun kann der Sabertooth hochgefahren und der Converter an den Rechner angeschlossen werden.

Beim Start der DEScribe Software wird ein Fenster angezeigt, welches in Abbildung 6.2 zu sehen ist. Hier kann mit einem Klick auf `Connect and Download Settings` die aktuell auf dem Sabertooth befindliche Konfiguration geladen und angepasst oder mit `Open Settings from File` eine vorgefertigte Konfigurationsdatei geladen werden.

In der Konfigurationsdatei wurde das Timeout der Steuerungssignale auf 125 ms eingestellt, um die Motoren möglichst schnell abzuschalten sobald keine PWM-Signale mehr ankommen. Außerdem wurde eine lineare Rampe mit einer Breite von etwa 1 s eingestellt, mit der Sprünge des Eingangssignals auf den Motor weitergegeben werden. So können Spitzenströme begrenzt und die Motoren geschont werden.

Nach Änderung der gewünschten Einstellungen können diese nun mit `Upload Settings to Device` auf den Sabertooth geladen werden.

Bevor die Änderungen nun aber getestet werden können, sollte nicht vergessen werden die DIP-Switches wieder in die erforderliche Position zu stellen.

6.2.2. Platine

Auf die Leistungselektronik-Platine wurde mit zwei Dioden ein Verpolschutz sowohl für die Systemelektronik als auch für den Schaltstrang des Relais realisiert. Außerdem wurde der Kill-Switch zum Schalten des Relais-Schaltstrangs per Steckverbindung mit der Platine verbunden. Bei der Erstellung der Platine wurde ergänzend eine Steckverbindung für einen Schalter für die Systemelektronik vorgesehen, damit ist es möglich die Elektronik separat von den Motoren von der Spannungsversorgung zu trennen.

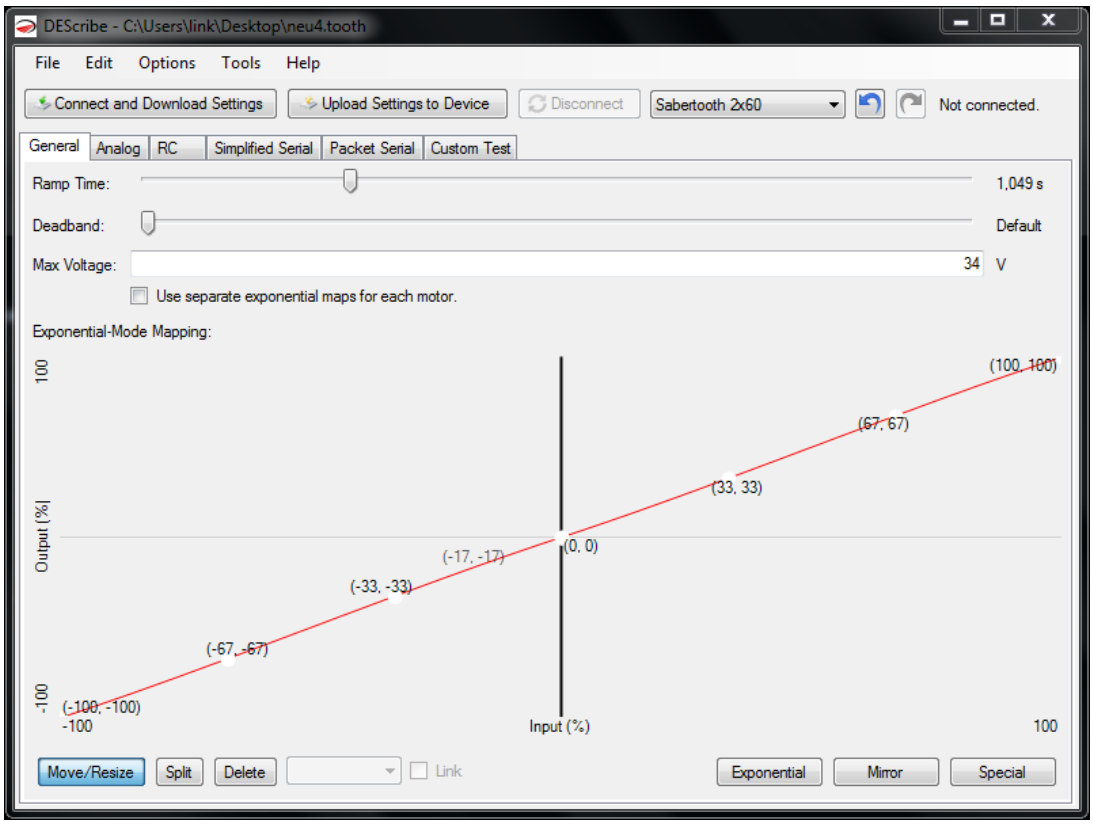

Abbildung 6.3.: DEScribe lineare Rampe

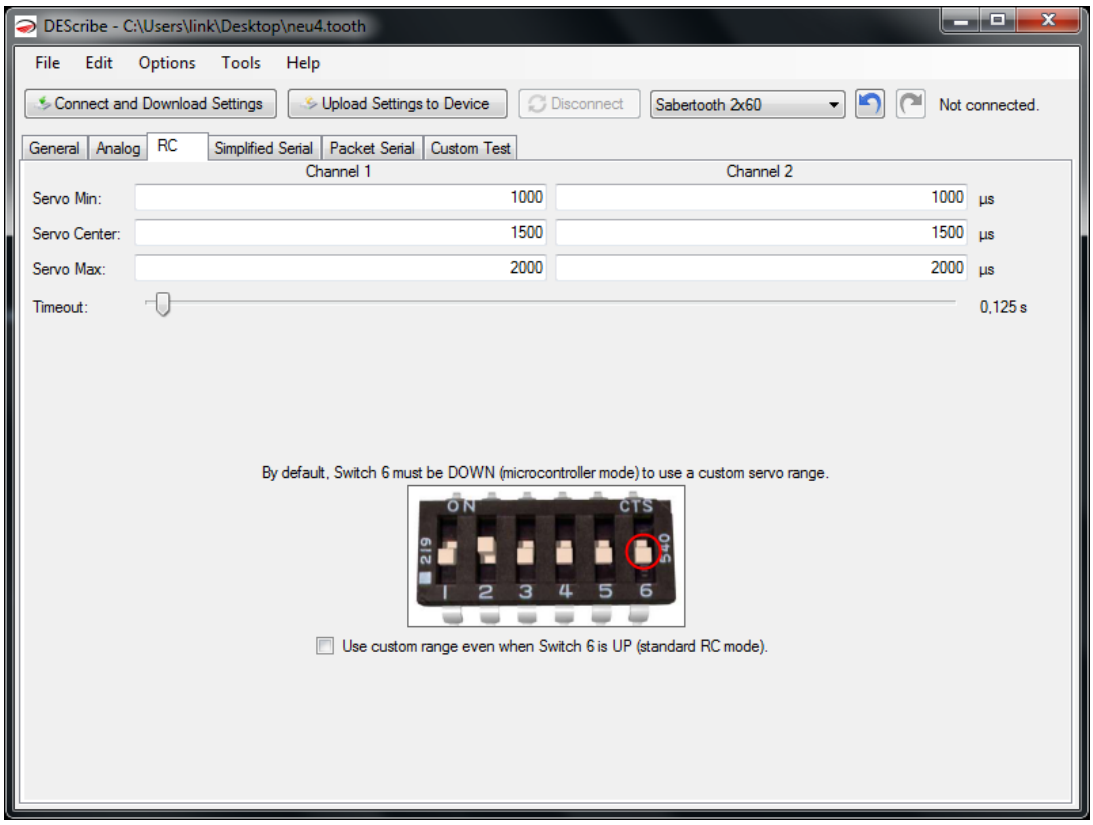

Abbildung 6.4.: DEScribe Timeout-Signal

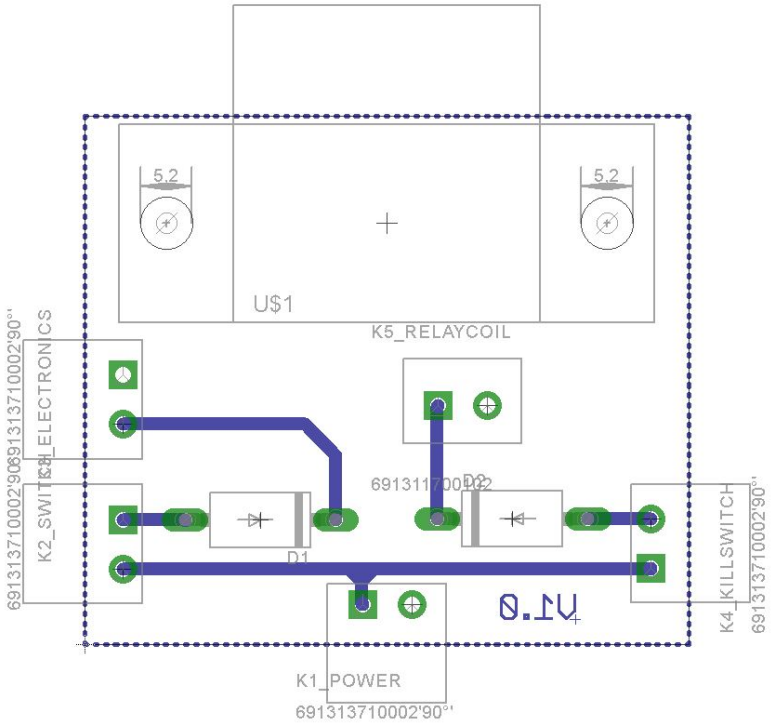

Abbildung 6.5.: Layout der Leistungselektronik-Platine

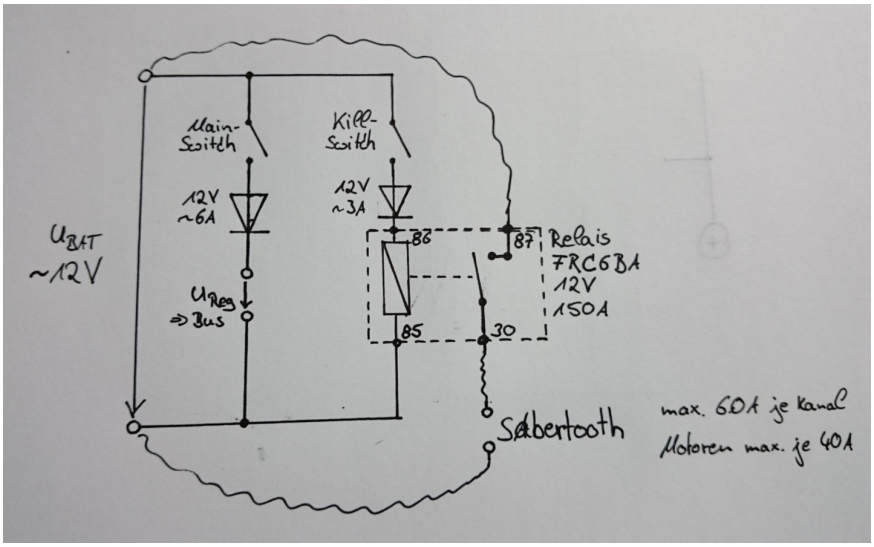

Abbildung 6.6.: Schaltungsskizze der Leistungselektronik

Des Weiteren ist auf der Platine ein Relaissockel montiert, um mit dessen Hilfe das Relais ordentlich in der Otterbox zu platzieren. Außerdem kann ein (zerstörtes) Relais somit leichter ausgetauscht werden.

Die Schaltung in Abbildung 6.6 wurde für die Leistungselektronik als Ansatz zur Verkabelung genutzt. Dabei wird das Relais als High-Side Switch genutzt, d.h. es schaltet den Sabertooth an die Versorgungsspannung. Ursprünglich geplant war es jedoch das Relais als Low-Side Switch zu verwenden (schalten gegen GND), da hierdurch bei eventueller Verpolung die Versorgungsspannung nur am (nicht geschalteten) Relais anliegen würde, wäre der Sabertooth abgesichert. Da sich der Sabertooth bei dieser Variante jedoch auch bei offenem Relais im Betriebsbereiten Zustand befand, konnte diese nicht genutzt werden. Dabei wares für uns aus Zeitgründen nicht nachvollziehbar warum der Sabertooth trotzdem versorgt wurde.

6.3. Motor Arduino

Auf diesem Arduino wurde neben der Motorsteuerung und der dafür notwendigen Kommunikation zum Raspberry Pi (RPi), mit Hilfe dieser auch die Ansteuerung der Debug-LEDs am Funkmast realisiert. Außerdem wurde auf Basis dieser Kommunikation auch ein Reset des Raspberrys bei ausbleibenden Nachrichten (Heartbeats vom Raspberry zum Arduino) mit Hilfe des Arduino WatchDog realisiert.

6.3.1. Kommunikation zum Raspberry Pi

Um die USB-Kommunikation zum RPi zu realisieren wird zunächst in der setup()-Funktion per `Serial.begin(9600)` die serielle Datenübertragung mit 9600 baud initialisiert. Dies kann auch schon vor der busyWaiting-Funktion zum Warten auf den Boot des Raspberry Pi geschehen, da der Serial Tranceiver ein separater Chip ist und somit auch die ersten Nachrichten des RPi garantiert im Buffer abgelegt werden. Zu beachten ist hier, dass für

Abbildung 6.7.: Motor Arduino mit Cape

die serielle Kommunikation die digitalen Pins 1 & 2 belegt sind. Diese können also nicht anderweitig verwendet werden.

Kommunikationsprotokoll

Das Protokoll muss zwei Arten von Daten abdecken:

- Motordaten (links und rechts) (-100% bis 100%)
- Heartbeat + LED-Debug Informationen

Für diese Informationen werden immer 3 Bytes gesendet. Das erste Byte gibt an, welche Nachricht die Debug-LEDs darstellen sollen:

- 0: keine Nachricht
- 1: XBEE Heartbeat hat gefehlt (Verbindung zum Festland unterbrochen)
- 2: Wegpunkt erreicht

Anschließend folgen 2 signed Bytes für die Prozentzahlen der Motoren (erst links, dann rechts).

Wenn der Arduino Uno über eine gewisse Zeit (z.B. 1 Sekunde) kein Heartbeat vom Raspberry bekommt, ist dieser vermutlich abgestürzt. Das wird dann durch ein bestimmtes Leuchtmuster der LEDs angezeigt und anschließend wird der Raspberry neugestartet. Diese Funktion realisiert der WatchDog des Arduinos (siehe Kapitel 6.3.4). Der WatchDog sollte danach für etwa 2 Minuten ruhen, damit der Raspberry genügend Zeit hat zu booten und die Onboard-Software zu starten.

Da laut Protokoll immer 3 Bytes übertragen werden sollen, wurde mit der if-Abfrage `Serial.available() >2` sichergestellt, dass alle 3 Bytes bereits im Buffer abgelegt sind, bevor diese ausgelesen werden. Ohne diese Abfrage ist es bereits vorgekommen, das die letzten beiden Bytes mit -1 ausgelesen wurde, da im Buffer noch keine Informationen bereitstanden. Nun werden zunächst die Bytes der Reihe nach mit `Serial.read()` ausgelesen

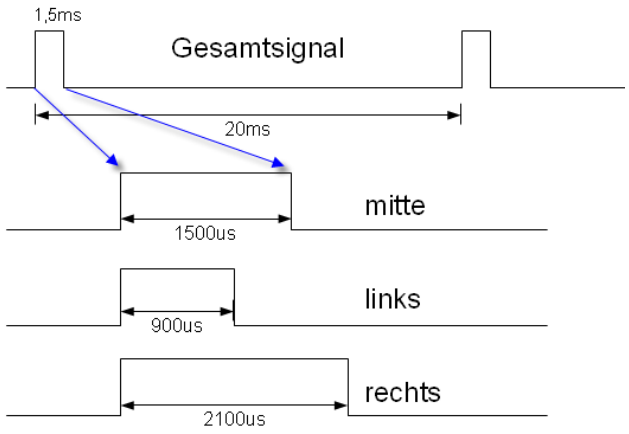

Abbildung 6.8.: RC-konformes PWM-Signal

und in entsprechende Variablen geschrieben. Darauf folgt die Auswertung des Debug-Byte, wodurch die LEDs die entsprechende Farbe anzeigen sollen. Dann werden die neuen Motorwerte an die Funktion `set_motors` übergeben, in der die entsprechenden Registerwerte für die PWM-Signale berechnet und gesetzt werden. Die genaue Verarbeitung der Bytes wird in den folgenden Kapiteln noch näher beschrieben.

6.3.2. Motorsteuerung

Zur Ansteuerung der beiden Rhino-VX-34 Motoren wird der Sabertooth 2×60 Motorregler benutzt, dieser kann durch Einstellen der Switches in verschiedenen Arbeitsmodi arbeiten, d.h. es kann auf unterschiedliche Eingabewerte reagiert werden. Hier wird der Modus 2 'R/C Input' verwendet (Switch 1 in DOWN, Switch 2 in UP Position), zusätzlich werden der 'Independent Mode' (Switch 4 in DOWN) und 'Microcontroller Mode' (Switch 6 in DOWN) gesetzt. Switch 3 und 5 befinden sich in UP Position. Genauere Wirkungsweisen der Switches können im Sabertooth Datenblatt nachgelesen werden.

Außerdem muss bei einer Neubeschaffung des Sabertooth beachtet werden, dass dieser im Werkszustand kein Timeout besitzt. D.h. sollte der Sabertooth keine gültigen Signale mehr erhalten, werden solange die letzten empfangenen Werte verwendet, bis neue gültige Werte empfangen werden. Dies könnte zur Folge haben, dass der MOPS bei einem eventuellen Ausfall des Arduinos (und damit der Steuersignale) außer Kontrolle gerät.

Um dies zu verhindern kann mit der DEScribe Software[1] der Sabertooth konfiguriert werden. Dazu wird wie in der Sabertooth Konfiguration beschrieben vorgegangen. Nach dem Upload der Einstellungen sollte der Sabertooth nach 0,125 Sekunden ohne gültige Werte in den Leerlauf wechseln.

Durch den R/C Input Mode lässt sich neben dem Mikrocontroller gleichzeitig auch eine RC-Fernbedienung anschließen. Dafür müssen die Signale allerdings gegeneinander gesperrt sein.

[1] http://www.dimensionengineering.com/software/SetupDS32.exe

Ein übliches R/C-Signal, auf das der Sabertooth eingestellt ist, sieht wie in Abbildung 6.8 dargestellt aus (Quelle: http://wiki.rc-network.de/index.php/PWM): Hierbei wird eine Pulsweitenmodulation (PWM) erzeugt, bei der die Grundperiode 20ms beträgt (50Hz) und mittels der Pulsbreite das Nutzsignal übertragen wird.

Der Arduino muss so eingestellt werden, dass er ein Nutzsignal im Bereich von [1000,2000] μs erzeugt, wobei gilt:

Duty	**Funktion**	**Bedeutung**
1000μs	-100%	Motorgeschwindigkeit maximal linksläufig (Rückwärts)
1500μs	0%	Mittelstellung (Leerlauf)
2000μs	100%	Motorgeschwindigkeit maximal rechtsläufig (Vorwärts)

Tabelle 6.2.: PWM-Signalbreite für die verschiedenen Motorgeschwindigkeiten

Für die PWM-Funktion stehen bei dem Arduino Uno zwei 8-bit und ein 16-bit Timer zur Verfügung. Bei einer 16MHz Grundfrequenz mit maximalem Prescaler von 128 werden für eine 20ms Periode 2500 Takte benötigt. Dies überschreitet die bei einem 8-bit Timer maximalen 256 Takte, daher muss hier der 16-bit Timer (mit 65536 möglichen Takten) verwendet werden. Um die Periode möglichst genau einstellen zu können, sollte der Prescaler möglichst klein gewählt werden, sodass die maximale Anzahl Takte genutzt werden. Hieraus ergibt sich ein Prescaler von 8, wodurch eine Taktrate von 0.5μs entsteht. Dadurch kann das Nutzsignal zwischen **2000 und 4000 Takten** variieren (Grundperiode **40000 Takte**).

Die Einstellung der Timer überschreitet die Arduino übliche, besonders einfache Syntax, weswegen die entsprechenden Register und Einstellungen aus dem ATmega328 Datenblatt entnommen wurden. Hier wird der sogenannte „Fast PWM Mode" verwendet, in diesem zählt der Counter bis zu einem einstellbaren Maximalwert (dieser kann in unterschiedlichen Modi durch feste Werte oder unterschiedliche Register beschrieben werden) und beginnt dann von 0 aus von vorne. Außerdem können in diesem Zählzyklus zwei voneinander unabhängige Vergleichswerte gesetzt werden. Bei erreichen dieser Werte wird ein Flankenwechsel des PWM-Signals vorgenommen. Je nachdem ob eine inverted oder non-inverted PWM eingestellt ist, wird dabei ein Wechsel von HIGH zu LOW, bzw. von LOW zu HIGH an den entsprechenden Pins durchgeführt. Diese Änderungen werden allerdings nur an die Pins übertragen, wenn diese auch als OUTPUT gesetzt sind. Am Ende des Zählzykluses wird dann wieder ein Flankenwechsel durchgeführt.

In den Registern TCCR1A und TCCR1B werden die Flankenwechsel pro Pin (COM1Ax und COM1Bx), der PWM-Modus (WGM10-WGM13) sowie der Prescaler (CS10-CS12) eingestellt. Die folgenden Werte wurden dabei von uns eingestellt:

- COM[1A1|1A0] = 0b10 (0x2) ⇒ Clear on Compare Match
- COM[1B1|1B0] wie COM1A
- WGM[13|12|11|10] = 0b1110 (0xE) ⇒ Fast PWM, ICR1 is Top Value
- CS[12|11|10] = 0b010 (0x2) ⇒ Prescaler is 8

Durch die Lage der einzelnen Bits in den Registern ergeben sich für diese die folgenden Werte. Im Code werden die hexadezimalen Werte verwendet, die bitweisen Werte werden hier mit aufgeschrieben, da diese einfacher nachzuvollziehen sind.

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 6.9.: TCCR1 Register des Arduinos

ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 6.10.: ICR1 Register des Arduinos

- `TCCR1A = 0b10100010 (0xA2)`
- `TCCR1B = 0b00011010 (0x1A)`

Da der maximale Zählwert (Top Value) sowie die Vergleichswerte (Compare Values) 16-bit groß sein können, die Register aber, wie am Beispiel ICR1 zu sehen, normalerweise nur 8-bit breit sind, wird der gleichzeitige Zugriff auf beide Werte über ein internes temporäres „High Byte Register" gewährleistet. Dadurch können die Werte für das Top Value und die Compare Values direkt in die entsprechenden Register geschrieben werden.

Für eine Grundperiode mit 50Hz werden, wie schon beschrieben, 40000 Takte benötigt, dieser Wert sollte daher auch in das Top Value geschrieben werden. Da aber der Reset des Counters erst einen Takt nach erreichen des Top Value ausgelöst wird, muss der Wert noch um 1 dekrementiert werden, um eine saubere 50Hz Frequenz zu erzeugen. Dies ist auch bei den Compare Values zu beachten. Damit gilt dann also ⇒ `ICR1 = 39999 (0x9C3F)`.

Durch die beiden 16-bit Register `OCR1A` und `OCR1B` werden in unserer Einstellung die Compare Values für die PWM-Signale an den OC1A- und OC1B-Pins (digitale Pins 9 und 10) beschrieben. Da diese jedoch die Motoren, der weiter unten beschriebenen Kommunikation mit dem Raspberry Pi entsprechend, steuern sollen, müssen die Werte variabel berechnet werden. Dazu wird der Wert für die Mittelstellung (2999) mit den überlieferten Prozentwerte aufgrechnet. Dies klappt ohne große Umrechnungen, da zu beiden Maximalwerten jeweils 1000 Schritte (500μs : 0,5μs/Takt) Unterschied besteht.

6.3.3. Debug-LED

Da der Arduino für die Motoreinstellungen bereits ständig mit dem Raspberry Pi kommuniziert, wurde diese Verbindung mitgenutzt um verschiedene Status-Informationen des Raspberry Pi zu übertragen. Mit Hilfe eines Adafruit Neopixel LED-Streifens wurde hier eine Debug-Anzeige realisiert.

Dazu wurde zunächst die `Adafruit_NeoPixel.h` Bibliothek eingebunden, um für diese LED-Streifen übliche, vorgefertigte Funktionen (bspw. zur Initialiserung) nutzen zu können. Somit konnte dann (außerhalb der Setup-Funktion o.ä.) mit dem folgenden Befehl ein LED-Streifen mit `4` LEDs, an dem angegeben `LED_PIN` (hier digitaler Pin 2) und weiteren speziellen Einstellungen für den LED-Streifen-Typen deklariert werden: `Adafruit_NeoPixel strip = Adafruit_NeoPixel(4, LED_PIN, NEO_GRB + NEO_KHZ800);`

In der Setup-Funktion wird der LED-Streifen dann mit Hilfe von `strip.begin();` aktiviert und mit `debugBlue();` wird die Farbe der LEDs auf *Blau* gesetzt. Bei der debugBlue-Funktion handelt es sich um eine selbst geschriebene Funktion, in der lediglich mit bereitgestellten Adafruit-Funktionen die Farbwerte gesetzt werden. Dabei dient die Bedeutung der Funktion vorallem dem Verständis der gesetzten Werte. Außerdem lassen sich die Funktionen so einfacher wiederverwenden oder austauschen.

Die `DebugX`-Funktionen rufen nun die folgenden Code-Zeile auf:

```
colorInstant(strip.Color(0,0,255));
```

Wobei `strip.Color(0,0,255)` die entsprechende Farbmischwerte der einzelnen RGB-LEDs setzt, in diesem Falle also bspw. keine roten, keine grünen und volle blaue Anteile, wodurch die LEDs dem Beispiel entsprechend *Blau* leuchten würden. Mit Hilfe der `colorInstant()`-Funktion wird der Farbmischwert für alle LEDs des Streifens gesetzt und übertragen.

Nachfolgend wird eine Tabelle der implementierten Farbfunktionen und ihrer bisherigen Bedeutung aufgeführt.

Farbfunktion	**Bedeutung**	**debug-Wert**
debugOff()	Setup() Arduino abgeschlossen	X
debugBlue()	Arduino wartet auf Bot des RPi	X
debugGreen()	Arduino empfängt korrekte Motorwerte	0
debugRed()	Arduino in WatchDog ISR, Neustart RPi	X
debugCyan()	MOPS hat Wegpunkt erreicht	2
debugMagenta()	nur für Testzwecke verwendet	3
debugYellow()	RPi hat XBee Signal verloren	1
debugWhite()	ungültigen debug-Wert empfangen	<0 oder >3

Tabelle 6.3.: Farbfunktionen der Debug-LEDs

6.3.4. WatchDog

Für den Fall, dass der Raspberry Pi oder der Arduino aus unbekannten Gründen abstürzen sollten, wurde eine Möglichkeit gesucht auf See einen unabhängigen Neustart durchzuführen.

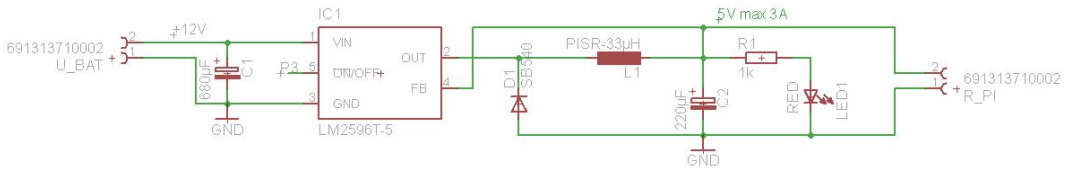

Abbildung 6.11.: Grundbeschaltung des 5-Volt-Spannungsreglers

Dazu wurde der beim Arduino verbaute WatchDog verwendet, indem nach Ablauf des Watchdog-Timers zunächst eine Interruptserviceroutine(ISR) den auf dem Cape verbauten Spannungsregler des RPi ausschaltet und beim zweiten Ablauf (dies ist vom μController so vorgegeben, wenn man ISR und Reset verwenden möchte) des Timers den Reset des Arduinos ausführt.

Dazu wird zunächst mit `#include <avr/wdt.h>` die WatchDog Library eingebunden, damit die nötigen Befehle bekannt sind. Nun sollte zunächst am Anfang der setup()-Funktion per `wdt_disable()` der WatchDog deaktiviert werden, damit kein unbeabsichtiger Timerablauf die ordnungsgemäße Initialisierung verhindert. Nachdem alle anderen verwendeten Funktionen aktiviert wurden, wird nun per `delay(120000)` für ~2 Minuten (120000 ist in Millisekunden angegeben, d.h. es sollten genau 2 Minuten sein, bei Messungen wurden jedoch Zeiten um etwa 2min 30sec festgestellt) der Programmablauf gestoppt. So wird sichergegangen, dass der Raspberry Pi genug Zeit zum booten hat (da dieser sonst immer wieder einen Reset durchführen würde (aufgrund ausbleibender USB-Nachrichten)). Als letzte Maßnahme der setup()-Funktion wird nun per `wdt_enable(WDTO_4S)` der WatchDog mit einem Timerzyklus von 4 Sekunden aktiviert. Außerdem wird durch den Befehl `WDTCSR = (1 «WDIE)` der Interrupt des WatchDog enabled, dadurch kann eine ISR geschrieben werden, die nach dem ersten Ablauf des Timers ausgeführt wird. In der ISR wird per `digitalWrite(3, HIGH)` der Pegel an Pin 3 hochgezogen, womit der Spannungsregler für den Raspberry Pi abgeschaltet wird. Bei jedem Aufruf der setup()-Funktion wird dieser Wert initial auf LOW gesetzt.

Nun muss noch an der entsprechende Stelle der USB-Kommunikation mit dem Raspberry Pi per `wdt_reset()` der WatchDog-Timer zurückgesetzt werden, damit dieser nur bei ausbleibenden Nachrichten auslöst.

6.3.5. Motor Arduino Cape

Auf diesem Cape wurden neben der Spannungsregelung von 12V der Auto-Batterie auf 5V für die Systemkomponenten Raspberry Pi und USB Hub, außerdem hauptsächlich eine Schaltung zur Trennung der Arduino PWM-Signale für die Motorregelung realisiert. Neben diesen Hauptaufgaben sind hier die erforderlichen Steckanschlüsse für die Debug-LEDs, den Raspberry Pi, den Sabertooth Motorregler und den RC-Receiver untergebracht.

Für die Spannungsregelung wurde aus dem Datenblatt des Spannungsreglers LM2596T die vorhandene Grundbeschaltung übernommen. Die entwickelte Schaltung für die PWM-Signale wird im folgenden Kapitel genauer erklärt.

Die folgenden Pins des Arduinos sind mit den beschriebenen Funktionen belegt:

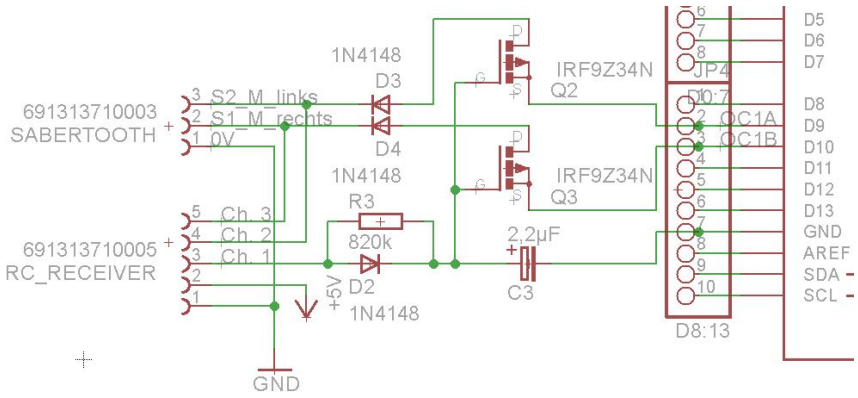

Abbildung 6.12.: MOSFET-Schaltung der PWM-Signale

- digitaler Pin 2 ⇒ Steuerung der Debug-LEDs
- digitaler Pin 3 ⇒ !On/OFF Schalten des RPi-Spannungsreglers
- digitaler Pin 9 ⇒ PWM Signal Motor links
- digitaler Pin 10 ⇒ PWM Signal Motor rechts
- Vin ⇒ 12V Versorgung des Arduino Uno
- 5V ⇒ Versorgung für RC-Receiver und Debug-LEDs

Motorschaltung

Die Motoren sollen sowohl durch den Arduino, als auch im Notfall von der RC-Fernbedienung gesteuert werden können. Dazu wurde eine Schaltung entworfen, die die zur Steuerung erforderlichen PWM-Signale des Arduino wegsperren, sobald die RC-Ferbedienung angeschaltet wird. Dazu wurde je ein p-Kanal MOSFET vorgesehen, diese lassen die PWM-Signale durch, solange die Fernbedienung ausgeschaltet ist. Die Gates werden dann von der Fernbedienung gesteuert, sodass die MOSFETs sperren.

Da die Fernbedienung jedoch nur PWM-Signale ausgibt, muss dieses erst zu einem konstanten Signal umgewandelt werden. Dazu wird ein extra PWM-Signal (keines der Motorsteuerung) dazu genutzt um einen 2,2μF-Kondensator aufzuladen, dieser Wert wurde so berechnet, dass er innerhalb von 1500μs (PWM-Puls) voll aufgeladen ist und sich mit Hilfe des 820k-Wiederstandes erst nach knapp 9s wieder entlädt. Dies führt dazu das solange die Fernbedienung angeschaltet bleibt ein konstantes Signal an den Gates der MOSFETs anliegt. Durch die Diode D2 wird sichergestellt, dass sich der Kondensator schneller auflädt, als entlädt. Die Dioden D3 und D4 verhindern Störeinflüsse der PWM-Signale der RC-Fernbedienung an den Drain-Pins der MOSFETs.

Da der RC-Receiver nach anlegen der Spannungsversorgung zunächst konstante 3,3V-Signale auf den meisten seiner Signalpins ausgibt, wird durch die Schaltung ebenfalls verhindert, dass der Arduino sofort Zugriff auf die Motorsteuerung hat. Erst nach einmaligem Ein- und Ausschalten der Fernbedinung liegen an den von uns benutzten Kanälen ~0V an und die MOSFETs lassen die Arduinosignale durch. Dieses Verhalten wurde von uns als

Abbildung 6.13.: GPS-Mouse

Sicherheitsfunktion genutzt, die es möglich macht den MOPS zunächst per Fernbedienung ein Stück von der Anlegestelle wegzufahren.

6.4. GPS

Der verwendete GPS-Empfänger GM-65 mit dem Adopt SkytraQ Venus8 Chipsatz ist am Bug des Bootes angebracht und über ein 2m langes USB-Kabel mit dem Raspberry verbunden. Der Empfänger ist wasserdicht und mit Magneten an der Unterseite ausgestattet, daher erfüllt der Empfänger unsere Anforderungen.

Technische Daten:

- Empfindlichkeit -160dBm
- 167 parallele Satelliten-Kanäle
- Startzeiten (Cold/Warm/Hot Start: 29/25/1 Sek)
- Kompatibel mit Windows XP / Vista / Windows 7 / 8 / 8.1/ Linux / Mac.
- Achtung! Keine Funktion mit Android-Betriebssystem!
- Protokoll: NMEA-0183

NMEA Die NMEA (National Marine Electronics Association) setzt sich für den Fortschritt der Marine-Elektronikindustrie ohne finanziellen Hintergedanken ein. Es handelt sich dabei um eine Vereinigung von Herstellern, Vertreibern und Ausbildungsinstitutionen.

Der von der NMEA festgelegte Standart 0183 ist für die Kommunikation zwischen GPS-Empfänger und PCs gedacht. Er besteht aus einer RS422-Schnittstelle und einer Definition von Datensätzen.

Anhand dieser standardisierten NMEA-Datensätze, können die Daten jedes GPS-Geräts mit einem beliebigen Navigationsprogramm auf dem PC ausgewertet werden. Im Folgenden ist ein Datensatz des NMEA Protokolls dargestellt. Der oberste GPGGA-Datensatz ist einer der wichtigeren Datensätze, da dort die Zeit (173755.001), Position (Latitude:5312.2222,N Longitude:00759.1459,E), Höhe (43.3,M) sowie die Genauigkeit der Messung (0.9) enthalten ist.

Abbildung 6.14.: Razor 9DOF Sensorboard

```
$GPGGA,173755.001,5312.2222,N,00759.1459,E,1,08,0.9,5.2,M,43.3,M,,0000*5A
$GPGSA,A,3,15,13,28,24,18,17,22,12,,,,,1.7,0.9,1.4*3E
$GPGSV,3,1,11,15,71,240,46,13,51,147,49,24,41,272,43,28,40,055,38*72
$GPGSV,3,2,11,17,30,108,40,18,27,298,42,22,10,329,33,12,08,215,26*79
$GPGSV,3,3,11,11,04,033,,04,02,022,,30,02,085,*47
$GPRMC,173755.001,A,5312.2222,N,00759.1459,E,000.0,001.2,010215,,,A*6E
$GPVTG,001.2,T,,M,000.0,N,000.0,K,A*0E
```

Detaillierte Informationen finden sich auf folgender Website:

http://www.gpsinformation.org/dale/nmea.htm

6.5. Kompass

Das 9DoF Razor IMU Board ist eine inertiale Messeinheit mit 9 Freiheitsgraden zur Messung des Drehverhaltens (Gyroscope) und Beschleunigung (Accelerometer) in allen drei Raumrichtungen. Zusätzlich besitzt das Board einen 3-Achsen-Magnetfeld-Sensor, mit welchem die aktuelle Ausrichtung des MOPS bestimmt wird. Mit dem Gyroscope wird der Wellengang bestimmt und in der Onshore-Software als „displacement"-Wert dargestellt. Falls dieser Wert zu hoch ist, besteht Gefahr für den MOPS und er muss zurückgeholt werden.

Das Auslesen der Werte erfolgt über eine USB-Schnittstelle. Der Java Code ist an diesem Beipiel https://github.com/ptrbrtz/razor-9dof-ahrs angelehnt. Bei der Berechnung der Kompasswerte werden zuerst die alten Messwerte im Puffer gelöscht. Dabei wird darauf geachtet, dass mindestens 12 Byte erhalten bleiben, die eine vollständige Menge von Messwerten darstellen. Aus diesen Bytes werden die Werte für Yaw, Pitch und Roll bestimmt. Der Kompass hat eine gravierende nicht-lineare Ungenauigkeit, die herausgerechnet werden muss. Dazu kann er kalibriert werden. Die Kalibrierung speichert die Werte, die der Kompass ausgibt, wenn er nach Norden, Osten, Süden und Westen ausgerichtet ist. Der korrigierte Wert wird mithilfe der Kalibrierungswerte und des aktuellen Kompasswerts interpoliert.

Wenn beispielsweise der Kompass im Norden einen Yaw-Wert von 20° und im Osten 60° hat und ein Rohwert von 40° gemessen wurde, dann wird der Winkelwert zu 45° interpoliert.

7. Kommunikation

7.1. Motivation

Um Daten von dem MOPS zu erhalten und ihm Kommandos zukommen zu lassen, ist es wichtig, dass eine stabile und weitreichende Kommunikation vorhanden ist. Die Gruppe MOPS II hatte sich für zwei Kommunikationsmöglichkeiten entschieden: WLAN und XBee. Kommunikation über WLAN hat den Vorteil der gesicherten Datenübertragung über bekannte Protokolle, wie TCP, jedoch den Nachteil der Reichweite (ca. 150m). Um die Reichweite zu verbessern, wurde von MOPS II zusätzlich die XBee, ein Funkmodul das Peer-to-Peer Daten austauschen kann, hinzugezogen. Da Probleme beim Übergang zwischen WLAN und XBee entstanden sind, und in der alten Version noch zu viele Daten übertragen wurden, ist die Kommunikation oftmals abgebrochen und musste daher überarbeitet werden. Die Gruppe MOPS III entschied sich vollständig auf WLAN zu verzichten, um den fehleranfälligen Übergang zwischen den beiden Kommunikationsarten zu vermeiden. Die Kommunikation erfolgt daher nun ausschließlich über XBee.

7.2. Protokoll

Durch ausführliche Betrachtung des XBee Nachrichtenprotokolls, welches von Werk vorgegeben ist, konnte ein massiver Overhead festgestellt werden. Daher wurde ein eigenes Protokoll zur Datenübertragung implementiert, das sicherstellen kann, dass die Nachrichten mit geringen Datengrößen und damit auch geringer Übertragungszeit übermittelt werden. Trotzdessen ist sichergestellt, dass die Nachrichten korrekt am Ziel ankommen. Nachfolgend ein kurzer Überblick über das Protokoll:

- Byte 1 ist immer die ID der Nachricht
- Byte 2 ist die Länge der Nachricht in Bytes
- Byte 3..N sind der Inhalt der Nachricht
- Byte N+1 und N+2 sind die Prüfsumme als CRC16

Auf `acknowledgements` wurde wegen der Dauer der Übertragungen weitestgehend verzichtet, dennoch ist der Ablauf dem Request-Reponse Modell nachempfunden. Das heisst, dass eine Seite eine Nachricht schickt, während die andere Seite nur auf Nachrichten antwortet. Falls keine semantisch wertvolle Antwort auf eine Nachricht existiert, wird ein `OK` zurückgeschickt. Die Onshore-Software übernimmt die Aufgabe des Sendens der Nachrichten. Mit den Antworten der Onboard-Seite kann abgeschätzt werden ob die Funkverbindung noch besteht. Zu diesem Zweck wurde ein Herzschlag (im Code `HeartBeat`) implementiert, der in einem kurzen Intervall versendet wird. Die Antwort auf den Herzschlag gibt Auskunft über den momentanen Zustand des MOPS und enthält unter anderem seine aktuellen Koordinaten sowie weitere Daten.

Bei jedem Empfangen der Nachrichten wird die Nachricht auf Vollständigkeit überprüft. Falls noch nicht alle Bytes eingetroffen sind, wird für eine festgelegt Zeit gewartet. Wird diese Zeit überschritten gilt die Verbindung als getrennt. Zudem wird beim Empfangen die Prüfsumme verglichen. Falls diese Überprüfung fehlschlägt, wird ein Ereignis ausgelöst in dem je nach Wichtigkeit der Nachricht auch ein Resend Command versandt werden kann.

7.3. Ereignisgesteuert

Das System ist weitestgehend ereignisgesteuert implementiert. Ereignisse, im Quelltext `event` genannt, werden ähnlich wie das Observer Pattern ausgeführt, mit dem entscheidenden Unterschied, dass Ereignisse nicht nur für Objekte gelten, sondern jedes tatsächliche Ereignis einen Methodenaufruf erzeugen kann. Damit werden ungewünschte Aufrufe vermieden und eine breitere Differenzierung zwischen Aktionen möglich.

7.4. Versenden großer Datenmengen

Um große Datenmengen von der Onshore-Software zum MOPS zu übertragen wurde ein weiteres System entwickelt. Dieses System wandelt die gegebenen Daten in einen `ByteBuffer` um und teilt diesen in mehrere Pakete auf. Zunächst wird allerdings die Prüfsumme über diesen Buffer berechnet und in einer initialen Nachricht zusammen mit der Länge des Buffers übertragen. So kann die Vollständigkeit beim Empfänger überprüft werden. Nachdem das letzte Paket übertragen wurde, wird zusätzlich mit einer weiteren Nachricht das Ende des Datentransfers angezeigt.

7.5. Implementierung

Die gesamte Kommunikation ist in ein eigenes Paket `de.uniol.mops.communication` ausgelagert. So kann die Implementierung auf beiden Systemen (Onshore- und Onboard-Software) laufen. Die Systeme müssen in der Regel nur über einen simplen Methodenaufruf Nachrichten versenden. Das Empfangen ist ebenso einfach, da Nachrichten mit Ereignissen verteilt werden. Es wird also nur eine Methode implementiert, in der die Nachrichten direkt als Java Objekt überreicht werden. Die Hauptklasse für die Kommunikation ist die Klasse `XBee`. Es gibt jedoch für jedes System noch eine eigene Helferklasse, die z.B. bei großen Teilen des Datentransfers behilflich ist. Für die Onshore-Software übernimmt diese Helferklasse auch die Funktion für das Einreihen von Nachrichten.

Das XBee-Modul ist über Treiberklassen in die Klasse `XBee` eingehängt. Diese Treiberklassen ermöglichen einen schnellen Wechsel von über USB angesteuerten XBees zu Xbees, die über eine andere Hardwarecomponente, wie z.B. ein Raspberry Pi Shield, angesteuert werden. So können auch andere Anschlussmöglichkeiten ohne großen Aufwand implementiert werden.

Die Implementierung ist auf mehrere Threads aufgeteilt und thread safe programmiert. Beim abarbeiten der Ereignisaufrufe sollte jedoch beachtet werden, dass diese in einem anderen Thread ablaufen, als die Onshore-Software oder die Onboard-Software.

Die Nachrichten, im Quelltext auch `Command` genannt, sind von einer gemeinsamen Basisklasse abgeleitet. Dies vereinfacht das Hinzufügen von einem neuen `Command`.

8. Payload

Zum Untersuchen von Gewässern muss der MOPS mit Sensoren bestückt werden. Hierfür muss der MOPS eine Schnittstelle bereitstellen, über welche die Sensoren zum Messen aufgefordert werden, wenn in der Mission ein entsprechender Missionspunkt erreicht wurde. Die Sensoren gehören nicht zum MOPS. Sie wurden verwendet um die MOPS-Plattform zu testen.

8.1. Designentscheidungen

Die Projektgruppe MOPS II hat bereits eine Schnittstelle erstellt, mit welcher die Sensoren über SDI-12 angesteuert werden konnten. Das Programm, welches auf dem Raspberry die Kommunikation verwaltet hat, wurde jedoch in Python geschrieben. Zur weiteren Vereinfachung der Plattform haben wir für diese Aufgabe einen neuen Quelltext in Java implementiert. Zu Beginn der Projektgruppe wurde eine Integration einer RS-485 Schnittstelle in Erwägung gezogen. In der Rapid-Prototyping-Phase wurde versucht, unter Zuhilfename einer Arduino-Bibliothek, die Sensoren auch über die RS-485 Adern mit dem Modbus-Protokoll anzusprechen. Es konnte jedoch keine Antwort der Sensoren empfangen werden. Da zu diesem Zeitpunkt die einfacherere SDI-12 Schnittstelle nicht stabil funktioniert hat wurde vom neuen Feature der RS-485 Schnittstelle abgesehen und stattdessen die SDI-12 Schnittstelle verbessert. Inzugedessen ist unter anderem das im folgenden beschriebene Cape, sowie die weiter unten beschriebene Software entstanden.

8.2. Sensor Arduino Cape

Auf diesem Cape wurde die Grundbeschaltung für die SDI-12 Verbindung realisiert. Dabei wurde darauf geachtet, dass der Abschlusswiderstand an die Länge der Bus-

Abbildung 8.1.: Sensor-Arduino mit Cape

Leitung angepasst werden kann. Dazu ist der Widerstand auf zwei Pinheadern montiert.

Der Widerstand kann so einfach ausgetauscht werden, je nachdem was an den Bus angeschlossen ist. In der SDI-12 Spezifikation ist ein Widerstand von 200 kΩ angegeben. Ist nur ein Sensor angeschlossen und der zweite Anschluss offen hat sich jedoch ein Widerstand von <100 kΩ bewährt (39 kΩ bzw. 68 kΩ). Außerdem ist eine 3-polige Steckverbindung auf dem Cape untergebracht, mit der dieses an den NMEA-Bus angeschlossen wird. Der Arduino wird mit Hilfe des Capes im Normalfall ebenfalls direkt mit 12V versorgt, in unserem Fall funktioniert der Spannungsregler auf dem verwendeten Arduino nicht mehr ordnungsgemäß, sodass eine Versorgung über USB nötig ist. Da der Arduino allerdings nur in Kombination mit dem Raspberry Pi und einer bestehenden USB-Verbindung genutzt wird, sollte dies kein Problem darstellen.

Die platzierten Pinheader wurden größtenteils aus Stabilitätsgründen in dieser Variante und Anzahl verwendet.

8.3. SDI-12

SDI-12 (Serial Digital Interface at 1200 Baud) ist ein serielles Protokoll für die Kommunikation zwischen einem Master und bis zu 62 Sensoren. Die bei der Implementierung verwendeten SDI-12 Befehle sind:

„a!" → Ping

- Sensor mit Adresse 'a' antwortet mit a<CR><LF>

„aI!" → Identifikationsanfrage

- Sensor antwortet mit einer Identifikationsnachricht, die unter anderem eine Sensor ID, den Namen des Herstellers

„aM!" → Aufforderung zum Messen

- Sensor antwortet mit einer Zeitangabe, wann der Sensor die Messung abgeschlossen haben wird, eventuell gefolgt von einem a<CR><LF>, falls der Sensor die Messung abgeschlossen hat bevor die von ihm angegebene Zeit abgelaufen ist.

„AD0!" → Aufforderung zum Senden von Daten

- Der Sensor sendet die gemessenen Daten

Weitere Befehle können der SDI-12 Dokumentation entnommen werden.

8.4. Sensor Arduino

Die SDI-12 Schnittstelle wird mithilfe einer SDI-12 Library für den Arduino Uno realisiert. Die Kommunikation zwischen Raspberry und dem Sensor Arduino besteht nur aus den Befehlen, die an die SDI-12 Schnittstelle weitergegeben werden sollen. Der Arduino speichert alle einkommenden Zeichen zwischen, bis er ein '!' empfängt. Dann wird die Nachricht an die Sensoren übertragen. In die entgegengesetzte Richtung funktioniert die Kommunikation ähnlich. Der Arduino empfängt die Zeichen vom SDI-12 Bus und überträgt die Nachricht an den Raspberry, sobald er einen Zeilenumbruch empfängt.

Abbildung 8.2.: C4E: Conductivity / Salinity - Sensor

8.5. Sensor Code

Beim Start der Onboard-Software stellt die Software eine USB-Verbindung zum Sensor Arduino her und ping die Sensoradressen 0-9 an, um festzustellen welche Sensoren angeschlossen sind. Wenn ein Sensor antwortet, wird zusätzlich seine Identifikation angefordert und gespeichert. Wenn gemessen werden soll, setzt die Missionsplanung ein Flag, woraufhin nacheinander mit allen Sensoren entsprechend des SDI-12 Protokolls kommuniziert wird um die gemessenen Werte zu erhalten. Sie werden erst zum Messen und nach der vom Sensoren übermittelten Zeit zum Übertragen der Daten aufgefordert. Die daraufhin vom Sensoren empfangenen Daten werden gespeichert.

8.6. Sensoren

Bei den Sensoren, die zum Testen verwendet wurden handelt es sich um einen C4E: Conductivity / Salinity[1] zum Messen des Salzgehalts und der Leitfähigkeit des Wassers, der in Abbildung 8.2 abgebildet ist.

Der andere Sensor Nephelometric Turbidity[2] gibt die Trübheit des Wassers an. Es ist in Abbildung 8.3 dargestellt.

Beide Sensoren unterstützen sowohl das SDI-12, als auch das RS-485- / Modbus- Protokoll. Sie wurden von der vorrausgehenden Projektgruppe in einen schützenden Käfig integriert.

[1] `http://www.ponsel-web.com/cbx/_ftp/datasheetc4euk.pdf`
[2] `http://www.ponsel-web.com/cbx/_ftp/datasheetdigisensntuuk.pdf`

Abbildung 8.3.: Nephelometric Turbidity

9. Planungs- und Steuerungssoftware Onboard

9.1. Aufbau

Der wesentliche Zweck der Onboard-Software ist, den MOPS an im Missionsplan angegebene Messpunkte zu steuern, dort die entsprechenden Messungen vorzunehmen, und den MOPS zurück zum Ausgangsort zu steuern. Diese Aufgaben fassen wir unter dem Oberbegriff „Planungs- und Steuerungsaufgaben" zusammen. Dabei wird von der hardwarenahen Schicht weitestgehend abstrahiert. Die Planungs- und Steuerungsaufgaben lassen sich in die nachfolgenden Teilaufgaben gliedern. Überwachungsaufgaben, zum Beispiel zur Kollisionsverhütung, werden dabei vorerst nicht berücksichtigt.

1. Globale Kontrolle und Verwaltung der Mission
2. Berechnung eines kollisionsfreien Pfads zum nächsten Zielpunkt
3. Anfahren von Wegpunkten
4. Steuerung der Messelektronik am Ziel

Die Kontrolle und Verwaltung der Mission beinhaltet die Berechnung des nächsten Schrittes bei der Ausführung der Mission. Ein solcher Schritt kann eine Teilaufgabe wie das Anfahren eines Wegpunktes oder das Ausführen einer Messaktion sein. Die Berechnungen basieren auf dem aktuellen Systemzustand, dem Fortschritt der Mission sowie etwaigen Benutzereingaben.

Die identifizierten Teilaufgaben lassen sich als einzelne Komponenten der Onboard-Softwarearchitektur implementieren. Dabei kann das Anfahren von Wegpunkten wiederum in zwei Teilkomponenten zerlegt werden:

- Berechnen von Konfigurationen für einen abstrakten Motor und ein abstraktes Ruder
- Berechnen von tatsächlichen Steuergrößen für die vorhandenen Aktoren aus den abstrakten Steuergrößen

Diese zusätzliche Abstraktionsschicht erlaubt eine bessere Wiederverwendung der Software beim Wechsel der Hardware-Plattform. Wir bezeichnen die resultierenden Komponenten im Folgenden dabei wie folgt:

1. Mission controller
2. Path planner
3. Für das Anfahren von Wegpunkten:
   a) Path controller
   b) Engine controller
4. Measurement controller

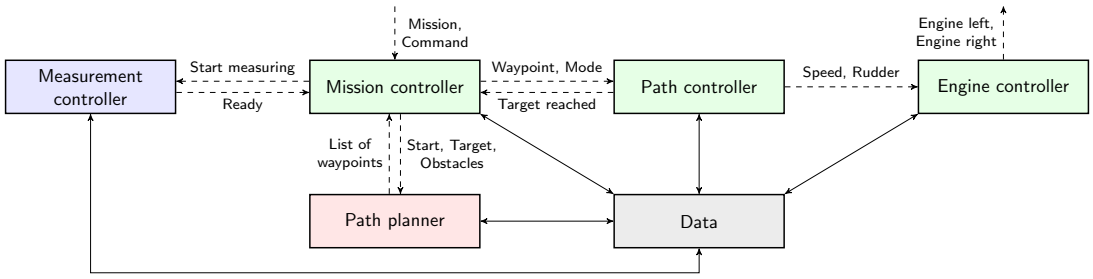

Abbildung 9.1.: Komponenten der Planungs- und Steuerungssoftware

Dabei sind die Missionskontrolle, die abstrakte Pfadregelung und die Berechnung von konkreten Motorkonfigurationen Kernaufgaben, die kontinuierlich ausgeführt werden müssen. Nur so kann gewährleistet werden, dass sich das Boot stets an der gewünschten Position befindet und auf Benutzereingaben zügig reagiert.

Die Pfadplanung und das Kontrollieren und Steuern der Messelektronik wiederum sind Vorgänge, die längere Zeit in Anspruch nehmen können. Auch während dieser Zeit muss die Position des MOPS geregelt und auf Benutzereingaben reagiert werden. Daher müssen Pfadplanung und Messaktionen parallel zu den Kernaufgaben ausgeführt werden. Daraus folgt, dass die Kernaufgaben als `Component` implementiert werden, während die Pfadplanung und die Ausführung von Messaktionen im Wesentlichen als `AbstractIO` realisiert werden.

Die Kommunikation der Teilaufgaben untereinander erfolgt ausnahmslos über `Data`. Abbildung 9.1 zeigt eine abstrakte Übersicht der einzelnen Komponenten und der Kommunikation untereinander. Gleichfarbige Komponenten werden in einem gemeinsamen Thread ausgeführt, wobei „Data" den gemeinsamen Datenbereich repräsentiert. Komponenten, die nicht direkt in die Missionsausführung involviert sind (beispielsweise die Kommunikation mit der Onshore-Software oder das Auslesen von GPS-Position oder Kompass) wurden der gewählten Abstraktionsebene entsprechend ausgelassen. Durchgezogene Pfeile stellen einen tatsächlichen Datenfluss dar. Gestrichelte Pfeile repräsentieren einen logischen Datenfluss. So erhält „Mission controller" beispielsweise von der Onhore-Software Missionspläne und Commands wie „Start", „Stop", „Pause" oder „Return To Home". Die Komponente „Path planner" hingegen erhält eine Startposition, eine Zielposition sowie Hindernisinformationen und gibt eine Liste von Wegpunkten als Pfad zum Ziel zurück.

Die folgenden Abschnitte beschreiben die Komponenten der Planungs- und Steuerungssoftware. Die Komponente „Measurement controller" bildet dabei eine Ausnahme. Sie ist zusammen mit der Beschreibung der zugehörigen Hardware und des entsprechenden Bussystems im Abschnitt 8 auf Seite 46 beschrieben.

9.2. Mathematisches

9.2.1. Problemstellung

Für die Pfadregelung und die Bahnplanung des Bootes sind unter anderem Winkel, Distanzen und Schnittpunkte zu berechnen. Dabei bewegt sich der MOPS im Koordinatensystem der Erde. Hierbei kommen Positionsangaben in Form von Längen- und Breitengrad zum

Einsatz. Daraus ergibt sich scheinbar ein zweidimensionaler Raum. Tatsächlich trifft das jedoch nicht zu; die Erde ist bekanntlich (zumindest in idealisierter Darstellung) eine Kugel. In letzter Konsequenz ergibt sich unter anderem, dass der Abstand zwischen zwei Längengraden nicht überall gleich ist. In Polnähe geht dieser Abstand gegen 0, am Äquator beträgt der Abstand zwischen zwei ganzzahligen benachbarten Längengraden 60 Bogenminuten, also 60 Seemeilen und somit etwa 111.12 Kilometer. Außerdem sind die „Ränder" des Koordinatensystems direkt miteinander verbunden. Diese Faktoren sind zu berücksichtigen, wenn Winkel und Distanzen berechnet werden.

Wir betrachten folgendes Beispiel, um eine Idee über die Größenordnungen der Abweichungen der Abstände zwischen zwei Längengraden zu geben. Oldenburg befindet sich in etwa an der Position $(53°N, 8°E)$. Die höchste gemessene Geschwindigkeit des Bootes über Grund beträgt etwa $8\frac{\text{km}}{\text{h}}$. Die gewünschte Einsatzdauer soll 8 bis 12 Stunden betragen. Da das Boot zurückkehren muss, kann es sich also maximal 6 Stunden und damit 48 Kilometer weit vom Ausgangsort entfernen. Verschiebt man die genannte Koordinate entlang $8°E$ um 48 Kilometer nach Norden, erreicht man etwa 53.25.54° nördlicher Breite. Der Abstand zwischen $(53°N, 8°E)$ und $(53°N, 9°E)$ (das ist etwa Bremen / Achim) beträgt circa 66.92km. Der Abstand zwischen $(53.25.54°N, 8°E)$ und $(53.25.54°N, 9°E)$ beträgt etwa 66.25km. Die Differenz beläuft sich also auf circa 670m. Startet man bei $(83°N, 8°E)$, also in arktischen Regionen, führt dasselbe Verfahren zu einer Differenz von etwa 830m bei einem Abstand von 12.72km bzw. 13.55km. Würde also beispielsweise für ein spezielles Einsatzgebiet ein bestimmter Abstand ganzzahliger Längengrade festgelegt, so könnten vermutlich Fehler im Meterbereich entstehen. Dies muss nicht kritisch sein, insbesondere nicht für die Bestimmung eines kürzesten Weges. Bei der Konstruktion freier Pfade können jedoch – je nach Art und Weise der Konstruktion – theoretisch Probleme auftreten. Dies gilt insbesondere, wenn bei der Berechnung von Ausweichmanövern Sicherheitsabstände automatisch berücksichtigt und berechnet werden sollen.

9.2.2. Aktuelle Implementation

Die aktuelle Implementation der Planungs- und Steuerungssoftware rechnet daher vorerst mit Orthodromen. Eine Orthodrome ist ein Teilstück eines Großkreises. Ein Großkreis ist der Schnitt einer Kugeloberfläche mit einer Ebene, die den Kreismittelpunkt beinhaltet. Der kürzeste Weg zwischen zwei Punkten entlang einer Kugeloberfläche ist eine solche Orthodrome [3]. Die implementierten Formeln, wie zum Beispiel die Semiversus-Formel zur Berechnung der Entfernung zweier Punkte auf einem Großkreis, entstammen im Wesentlichen [1]. Der klare Nachteil dieser Formeln ist die hohe Anzahl an trigonometrischen Funktionen, die in jeder Formel zu berechnen sind. Dies erhöht den Rechenzeit. Der Vorteil ist jedoch, dass kürzeste Wege und Winkel bis auf Rundungs- und Modellfehler (die Erde ist ja keine perfekte Kugel) korrekt berechnet werden, Längen- und Breitengrade direkt als Eingabe benutzt werden können, und die Formeln robust gegen Randfälle sind. Alle im vorigen Abschnitt beschriebenen Probleme des Koordinatensystems der Erde werden also automatisch kompensiert, während beispielsweise bei einer Übertragung in den normalen zweidimensionalen Raum diese Faktoren explizit berücksichtigt werden müssten.

MOPS II nutzte bereits Berechnungen mit Orthodromen zur Pfadregelung (vgl. Programmcode des MOPS II). Dies gab Anlass zu prüfen, ob Orthodrome trotz ihrer Nachteile in MOPS III sinnvoll benutzt werden können. Ausschlaggebend für den Einsatz von Orthodromen waren die folgenden Kriterien:

- Die Rechenzeit zeigte sich in Experimenten für den Einsatzzweck bereits mehr als ausreichend: Bei der Ausführung der Main-Loop wird die Pfadregelung ausgeführt, die Abstände und Winkel berechnen muss. Für die gesamte Main-Loop waren Zykluszeiten im Zehntelsekundenbereich bei moderater Rechenlast machbar. Dies ist für die relativ träge Dynamik des MOPS III und seiner Umwelt ausreichend kurz (im Gegensatz zur Avionik beispielsweise). Die Rechenzeit der Pfadplanung wird grundsätzlich nicht als kritisch angesehen, solange keine zeitkritischen Ausweichmanöver geplant werden. Es kann während der Berechnung die aktuelle Position gehalten werden. Die Laufzeit der Bahnplanung wird in Abschnitt 9.6.5 auf Seite 74 betrachtet.
- Zu Projektbeginn wurden verschiedenste Einsatzszenarien des MOPS skizziert. Unter anderem wurden beispielsweise auch Einsatzgebiete in Skandinavien benannt. Solche Szenarien erfordern unbedingt eine Berücksichtigung der am Anfang des Abschnitts beschriebenen Probleme. Eine Festlegung auf ein reduziertes Einsatzgebiet (z. B. Nordseeküste) und somit beispielsweise des Abstands zwischen Längengraden erscheint daher wenig sinnvoll. Eine automatische Kompensierung der Probleme ist daher hilfreich.
- Zuverlässigkeit und Stabilität waren die wesentlichen Kriterien bei der Entwicklung des MOPS III. Unter diesem Aspekt wurde versucht, möglichst wenige unnötige Optimierungen durchzuführen. Die Laufzeit ist für die aktuellen Aufgaben zunächst ausreichend. Sie sollte also nicht zu Lasten der Genauigkeit der Berechnungen optimiert werden.

Die geographischen Berechnungen sind in der Klasse `de.uniol.mops.util.GeoCalc` des Projektes `de.uniol.mops.pathplanning` gebündelt implementiert. Eine eigene Implementation wurde potentiellen externen Bibliotheken vorgezogen, da auf diese Weise eine eigene Schnittstelle zu den Berechnungen definiert werden kann. Bei Änderungen am Berechnungsprinzip können die entsprechenden Funktionen neu implementiert werden, ohne die Schnittstelle, also den Zugriff auf die Funktionen, ändern zu müssen. Ferner bestehen bei einer Eigenimplementation keinerlei Abhängigkeiten in Bezug auf Unterstützung der Bibliothek durch die Plattform oder die verwendete Java-Version.

9.2.3. Optimierungsansätze

Das Hinzufügen weiterer Komponenten zur Main-Loop kann jedoch eine Laufzeitoptimierung erforderlich machen, wenn deutlich mehr Berechnungen durchgeführt werden müssen, ohne dass die Zykluszeit erhöht werden soll. Zudem können Erweiterungen wie die Planung von Ausweichmanövern oder eine Selbstüberwachung der Position (siehe Abschnitt 9.7) strengere zeitkritische Anforderungen an die Berechnungen stellen. Wir stellen daher an dieser Stelle einige Ansätze zur Laufzeitoptimierung vor. Diese sollten ursprünglich detailliert betrachtet werden. Durch den zusätzlichen Aufwand der Neuimplementationen im Zuge der Neugestaltung der Software-Architektur konnte diese Betrachtung jedoch nicht mehr realisiert werden. Daher kann keine Bewertung der Verfahren gegeben werden. Die Auflistung ist eher als Ausgangspunkt für eine Suche nach geeigneten schnelleren Verfahren zu sehen, falls dies erforderlich sein sollte:

- Rechnen mit Loxodromen anstelle von Orthodromen. Loxodrome werden in üblichen, auf Mercator-Projektion basierenden Karten als Geraden dargestellt. Die Differenz der Längen einer Orthodrome und einer Loxodrome, also ein möglicher Fehler bei Längenberechnungen, dürfte im Einsatzradius des MOPS vergleichsweise gering sein. Nähere Informationen zu Loxodromen sind unter anderem unter [2, 1] zu finden.

- Die Anzahl der zu berechnenden trigonometrischen Funktionen kann mitunter durch Transformation von Koordinaten in 3D-Vektoren reduziert werden. Diese Transformation (und eine eventuell nicht notwendigerweise durchzuführende Rücktransformation) erfordert selbst die Berechnung trigonometrischer Funktionen. Allerdings sind Schnittpunkte von Großkreisen danach lediglich Kreuzprodukte von Vektoren. Auch hierzu sind weitere Informationen unter [1] zu finden.
- Die geographischen Koordinaten könnten in ein zweidimensionales kartesisches Koordinatensystem projiziert werden, wobei der Abstand zwischen Längengraden durch eine Verschiebung des Punktes entlang des Breitengrades normiert werden könnte. Die Länge der Verschiebung kann mit Hilfe der Kenntnis des Breitengrades – ebenfalls unter Verwendung einiger trigonometrischer Funktionen – bestimmt werden. Zudem könnten der Ursprung des Koordinatensystems so normiert werden, sodass eventuelle Problembereiche wie der Übergang am Meridian nicht auftreten (beispielsweise durch Drehung des Koordinatensystems um 180°. Ferner könnte überprüft werden, ob es ausreicht, die Länge der Verschiebung entlang des Breitengrads einmalig zu Beginn einer Mission zu berechnen und im Folgenden stets denselben Verschiebungsfaktor anzuwenden.

9.3. Engine controller

Modularität war ein wesentlicher Aspekt bei der Entwicklung der Onboard-Software. Der Hintergrund ist – unter anderem – auch die Austauschbarkeit des Bootes in Teilen oder im Ganzen. Bei einem solchen Tausch könnten nicht nur Sensorbauteile wie GPS oder Kompass geändert werden. Insbesondere kann nach einem Austausch von Rumpf und / oder Aktuatoren ein völlig anderes Konzept zur Steuerung des MOPS vorliegen. Es könnte beispielsweise ein Rumpf mit Ruder und womöglich nur einem Motor eingesetzt werden, oder ein Rumpf mit Pod-Antrieben. Diese Antriebsarten sind offensichtlich völlig unterschiedlich anzusteuern.

Um eine Austauschbarkeit auf dieser Ebene des Systems zu gewährleisten, wurde eine Abstraktionsebene zwischen Bahnregelung und Aktuatorsteuerung realisiert. Das bedeutet, dass die Bahnregelung Stellwerte für ein virtuelles Ruder und einen virtuellen Motor berechnet, die von der Komponente „Engine controller" in konkrete Stellgrößen umgerechnet werden. Bei einem kleineren Hardware-Tausch muss womöglich nur die Komponente zur Übertragung der Stellwerte an die Aktuatoren getauscht werden. Ändert sich jedoch das Steuerungskonzept essentiell, so kann dieses in der Regel durch den Tausch der Komponente „Engine controller" angepasst werden, ohne die eigentliche Bahnregelung anpassen zu müssen.

Die Komponente „Engine controller" erhält mittels `Data` von der Komponente „Path controller" die Steuergrößen „Speed" und „Rudder" für ein abstraktes Ruder und eine abstrakte Maschine für den Vortrieb. Daraus werden die Konfigurationen der beiden tatsächlich vorhandenen Motoren berechnet, die aus jeweils einer prozentual angegebenen Drehzahl bestehen. Diese können auf zwei verschiedene Arten berechnet werden, die im Folgenden in Anlehnung an die jeweils implementierte Berechnungsvorschrift als „Both" und „Separate" werden:

- **„Both"** Multipliziert den „Rudder"-Wert r mit einem Verstärkungsfaktor f, addiert (subtrahiert) ihn zum (vom) „Speed"-Wert s und ergibt die Steuergrößen e_{outer} (e_{inner}) für den kurvenäußeren (kurveninneren) Motor. Das Resultat wird anschließend auf

das Intervall $[-100, 100]$ beschränkt.

$$e_{\text{outer}} = \max(s + f \cdot r, 100) \tag{9.1a}$$
$$e_{\text{inner}} = \min(s - f \cdot r, -100) \tag{9.1b}$$

- **„Separate"** Multipliziert den „Rudder"-Wert r mit einem Verstärkungsfaktor f und addiert ihn zum „Speed"-Wert s um die Steuergröße e_{outer} für den kurvenäußeren Motor zu erhalten. Ist $e_{\text{outer}} > 100$, so wird die Differenz zwischen e_{outer} und 100 vom „Speed"-Wert s subtrahiert, um die Steuergröße e_{inner} für den kurveninneren Motor zu berücksichtigen. Beide Resultate werden auf das Intervall $[-100, 100]$ beschränkt.

$$e_{\text{outer}} = \max(s + f \cdot r, 100) \tag{9.2a}$$
$$e_{\text{inner}} = \min(s - \max(s + f \cdot r - 100, 0), -100) \tag{9.2b}$$

Die Komponente „Engine controller" ist zur Laufzeit über die Onshore-Software konfigurierbar. Es lässt sich die Variante auswählen sowie der Verstärkungsfaktor ändern. Der Verstärkungsfaktor wird in der Konfiguration als `Rudder Ratio` bezeichnet.

Tests haben gezeigt, dass beide Varianten den MOPS auf dem Tweelbäker See zuverlässig steuern können. Bei der Konfiguration des Verstärkungsfaktors ist zu berücksichtigen, dass durch die Berechnungsvorschrift in der Variante „Both" die Drehzahldifferenz zwischen den Motoren automatisch doppelt so groß ist wie in der Variante „Separate".

Ein weiterer Unterschied besteht darin, dass in der Variante „Both" die Drehzahldifferenz bei großen Geschwindigkeiten kleiner wird. Beträgt beispielsweise der Wert „Speed" bereits 90 und ist „Rudder" gleich 30, so wird für den kurvenäußeren Motor die Drehzahl 100% berechnet, und für den kurveninneren Motor 60%. Beträgt „Speed" anfangs jedoch nur 60, so ergeben sich die Drehzahlwerte 90% und 30%. Im ersten Fall beträgt die Differenz 40 Prozentpunkte, im zweiten Fall 60 Punkte. Dadurch entsteht bei größeren Geschwindigkeiten zunächst eine geringere Drehrate, die gegebenenfalls durch die mit PID-Reglern implementierte Bahnregelung verstärkt wird, falls die Drehrate nicht ausreicht, um den Fehler zu korrigieren. So kann selbst bei einem „Speed"-Wert von 100 zusammen mit einem Verstärkungsfaktor $f \geq 2.0$ Drehzahlwerte 100% und -100% erreicht und so auf der Stelle gedreht werden.

Ein Vorteil dieser verringerten Drehrate ist in der zurückgelegten Strecke begründet: Die als Komponente implementierte Regelung berechnet einmal pro Zyklus der Main-Loop neue Werte für „Speed" und „Rudder". Für die Dauer einer Zykluszeit fährt der MOPS mit dieser Drehrate. Zugleich haben sich größere Zykluszeiten von etwa einer Sekunde als im Allgemeinen für den Einsatzzweck ausreichend gezeigt. Bei hohen Geschwindigkeiten und solchen Zykluszeiten ist daher die mit dieser Drehrate zurückgelegte Strecke mitunter deutlich größer als bei kleineren Geschwindigkeiten. Dies entspricht einer längeren Drehzeit bei kleinerer Geschwindigkeit. Dadurch kann ein Überschwingen entstehen. Durch die verringerte Drehrate wird bei hohen Geschwindigkeiten zunächst „vorsichtiger" gesteuert und die Gefahr des Überschwingens zum Teil kompensiert. Daher setzten wir in der Regel die Variante „Both" ein.

Die Grundlagen der beiden Umrechnungsvarianten „Both" und „Separate" entstammen MOPS II (vgl. Code). Aufgrund der neuen Software-Architektur des MOPS III musste die Umrechnung neu implementiert werden. Dabei wurde nicht nur entsprechend der Architektur modularisiert, sondern auch innerhalb der Umrechnungskomponente, sodass eine dynamische Auswahl der Variante zur Laufzeit möglich ist. Dies erleichtert Tests und erlaubt eine schnelle Umkonfiguration für den Fall, dass der MOPS durch die aktuelle Konfiguration oder Implementationsfehler in einer Variante nicht in der Lage ist, einen Weg zu verfolgen. Ferner ist es nun möglich, neue Varianten zu implementieren und der Software hinzuzufügen (oder zu entfernen), ohne den Code der eigentlichen Umrechnungskomponente oder der Onshore-Konfigurationssoftware ändern zu müssen. Zudem wurden Fehler in der Variante „Separate" der Implementation des MOPS II behoben.

9.4. Path controller

Es gibt im Wesentlichen zwei verschiedene Gründe, den MOPS fahren zu lassen: Entweder soll eine Zielposition angefahren werden, oder aber das Boot soll eine bestimmte Position nicht verlassen. Daraus ergibt sich eine Sollbewegung, beispielsweise entlang einer Geraden auf dem Weg zum Ziel oder Stillstand wenn das Ziel erreicht ist. Durch äußere Einflüsse wie Wind oder Strömung wird die tatsächliche Bewegung und damit die Position des Bootes beeinflusst und Abweichungen von der Sollbewegung entstehen.

Die Aufgabe der Komponente „Path controller" ist die abstrakte Pfadregelung des MOPS III: Sie berechnet basierend auf Sollbewegung und Istbewegung Steuergrößen für Aktoren des Bootes, die eine Abweichung von Soll und Ist kompensieren. Wir bezeichnen die Pfadregelung als abstrakt, da sie von den technischen Details der Regelstrecke abstrahiert: Sie berechnet die Steuergrößen r für ein abstraktes Ruder, im Folgenden auch „Rudder" genannt, und s für eine abstrakte Maschine für den Vortrieb, im Folgenden auch „Speed" genannt. Dies sind die Eingangsgrößen für die nachgelagerte Komponente „Engine controller".

9.4.1. Allgemeines zur Pfadregelung

Bevor wir die verschiedenen Arten den Pfad bzw. die Bewegung des MOPS zu regeln betrachten, klären wir an dieser Stelle einige grundlegende Begriffe, die in Abbildung 9.2 gezeigt werden. Die Bezugsrichtung für Winkelangaben ist dabei 0° (Nord).

- **Desired path:** Das ist die Gerade vom Startpunkt zur Zielposition. Diese Gerade ist der Sollpfad.
- **Drift:** Bewegung des Bootes durch äußere Einflüsse wie Wind und / oder Strömung. Ohne Motoren und Ruder würde das Boot im Wesentlichen Richtung und Geschwindigkeit der Drift annehmen.
- **Course Over Ground (COG):** Das ist die tatsächliche Bewegung des Bootes über Grund. Ein Boot schwimmt in der Regel auf dem Wasser. Motor und Ruder erzeugen folglich einen Bewegungsvektor durch das Wasser. Wind erzeugt ebenfalls ein Bewegung des Bootes durch das Wasser, wobei dieser Vektor im Allgemeinen eine andere Richtung hat als der durch Motor und Ruder erzeugte Vektor. Durch Strömung kann sich die das Boot umgebende Wassermenge wiederum über die

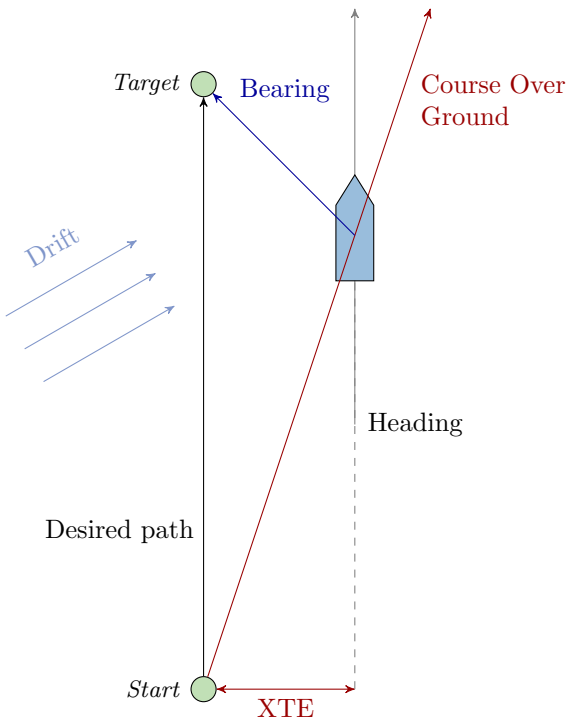

Abbildung 9.2.: Grundlegende Begriffe

Erdoberfläche (den Grund des Gewässers) bewegen. Die Summe dieser Bewegungsvektoren ergibt den Kurs des Bootes über Grund. Der COG wird als Winkel zwischen dem Bewegungsvektor über Grund und der Bezugsrichtung gemessen.

- **Heading:** Das ist die Ausrichtung der Längsachse des Bootes in Fahrtrichtung. Das Heading wird als Winkel zwischen der Längsachse und der Bezugsrichtung.
- **Bearing:** Das ist vom Boot aus gesehen die Richtung, in der das Ziel liegt. Sie wird ebenfalls als Winkel zwischen der Geraden vom Boot zum Ziel und der Bezugsrichtung angegeben.
- **Cross Track Error (XTE):** Das ist der kürzeste Abstand des Bootes zum gewünschten Pfad, also die Länge der Strecke von einem Punkt auf dem gewünschten Pfad zur aktuellen Position, wobei die Strecke orthogonal zum gewünschten Pfad ist.

Abbildung 9.3 zeigt schematisch, wie die Pfadregelung des MOPS arbeitet. Aus der Bahnplanung geht die Zielkoordinate in die Regelung ein. Mittels der Sensoren des Schiffes werden beispielsweise die aktuelle Position, das Heading oder der Kurs über Grund bestimmt und zusammen mit der Zielposition die entsprechende Abweichung vom Soll berechnet. Aus dieser Abweichung werden dann unter Berücksichtigung der Kenngrößen des Bootes, die sich in der Reglerkonfiguration widerspiegeln, neue Steuergrößen für Geschwindigkeit und Ruder berechnet. Diese werden an die in diesem Fall durch die Komponente „Engine controller" repräsentierten Aktoren Motor und Ruder weitergeben. Motor und Ruder verändern dann wiederum die Bewegung des Bootes. Die Komponente „Path controller" umfasst die graue Box der Darstellung, während die oben beschriebene Komponente „Engine controller" den ersten Teil des Übergangs vom Regleralgorithmus zu den Aktoren realisiert (der zweite

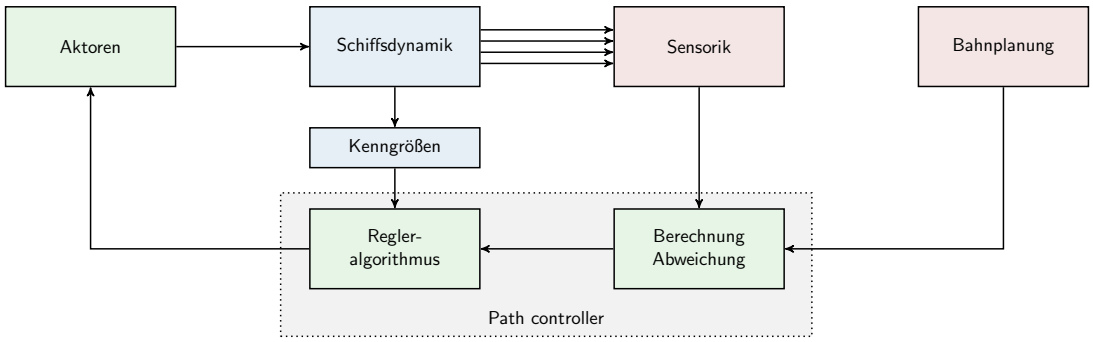

Abbildung 9.3.: Regelkreis

Teil besteht in einer separaten Komponente zur Übertragung der Konfiguration an die Motoren).

9.4.2. PID-Regler

Im nachfolgenden Abschnitt werden die Varianten zur Pfadregelung in MOPS III beschrieben. Alle Varianten haben gemeinsam, dass jede geregelte Größe mittels eines sogenannten PID-Reglers geregelt wird. Diese Regler kommen zum Einsatz, da sie leicht zu implementieren und daher wenig anfällig gegen Programmierfehler sind. PID-Regler bestehen aus drei Anteilen, dem Proportionalanteil P, dem Integralanteil I und dem Differentialanteil D. Sei $e(t)$ der Fehler zum Zeitpunkt t, der die Eingabe des PID-Reglers ist. Dann berechnen sich die einzelnen Anteile wie folgt:

$$P(t) = K_P \cdot e(t) \tag{9.3a}$$

$$I(t) = K_I \cdot \int_0^t e(t)\,\mathrm{d}t \tag{9.3b}$$

$$D(t) = K_D \cdot \dot{e(t)} \tag{9.3c}$$

Dabei sind die Faktoren K_i mit $i \in \{P, I, D\}$ konstante Faktoren zur Konfiguration des Reglers. Die Implementation des PID-Reglers des MOPS III erlaubt die Veränderung dieser Faktoren zur Laufzeit über die Onshore-Software. Die Herleitung geeigneter Faktoren ist nicht trivial. Da kein mathematisches Modell der Regelstrecke existiert, können sie nur empirisch bestimmt werden. Dies ist wiederum möglich, da ein instabiler Zustand der Bahnregelung in freiem Gewässer kein nennenswertes Sicherheitsrisiko darstellt.

Der P-Anteil des Reglers reagiert auf die Größe des Regelfehlers (der Abweichung vom Soll): Ist beispielsweise die Abweichung des Headings vom Soll groß, erfolgt eine große Korrektur der Steuergröße „Rudder". Der I-Anteil reagiert ebenfalls auf die Größe des Fehlers, jedoch über einen Zeitraum hinweg. Dieser Anteil ist insbesondere dazu geeignet, konstante Abweichungen auszugleichen. Das kann beispielsweise ein falsch eingestelltes Ruder sein, bei dem in Neutralstellung nicht geradeaus gefahren wird, oder eine stetige seitliche Drift, bei der durch den P-Anteil allein je nach Güte der Konfiguration ein mehr oder weniger ausgeprägtes „Schlingern" entstehen kann: Ein Abweichung des Headings wird

durch den P-Anteil korrigiert, der dann wieder gegen 0 geht, um die anschließend erneut auftretende Abweichung abermals zu korrigieren. Während eines solchen „Schlingerns" wächst der I-Anteil solange, bis der er groß genug ist, um die kontinuierliche Drift oder das falsch eingestellte Ruder zu kompensieren. Der D-Anteil wiederum reagiert überhaupt nicht auf die Größe des Fehlers, sonder auf die Änderungsrate des Fehlers: Wächst beispielsweise eine Abweichung des Headings sehr schnell, so dreht sich das Boot vermutlich sehr schnell. Daraus folgt, dass eine große Beschleunigung in die entgegengesetzte Drehrichtung sinnvoll sein kann und das Ruder sollte stark gegensteuern.

Es ist an dieser Stelle anzumerken, dass der I-Anteil mit einer sogenannten Windup-Kontrolle auszustatten ist. In Situationen, in denen die Regelung einem Fehler nicht entgegenwirken kann, sorgt die Windup-Kontrolle dafür, dass das Integral des I-Anteils nicht beliebig wächst. Eine solche Situation wäre im Falle des MOPS III beispielsweise die Übernahme der Kontrolle per RC-Fernsteuerung. Diese Übernahme ist für die Software des MOPS völlig transparent. Wird der Regelungsfehler nicht mit der Fernbedienung ausgeglichen, beispielsweise weil ein manuelles Ausweichmanöver gefahren wird, so könnte der I-Anteil ohne Windup-Kontrolle nach dem Ausschalten übersteuern und bzw. dauerhaft einseitige Werte annehmen (der Integralwert kann nicht wieder abbaut werden). Allgemein finden sich in der Literatur zu dieser Problematik verschiedene Lösungsansätze. Die Implementation des MOPS III setzt auf ein sehr einfaches aber robustes Verfahren: Die Größe des Betrags des Integrals des I-Anteils wird beschränkt. Diese Begrenzung kann ebenfalls zur Laufzeit durch die Onshore-Software geändert werden.

Die resultierende Steuergröße *PID* berechnet sich dann wie folgt:

$$PID\,(t) = P\,(t) + I\,(t) + D\,(t) \tag{9.4}$$

Die Berechnung erfolgt zeitdiskretisiert in jedem Zyklus der Main-Loop.

9.4.3. Regelungsvarianten

Es gilt für die Pfadregelung des MOPS III allgemein, dass sowohl die Geschwindigkeit des Bootes als auch die Fahrtrichtung kontrolliert und geregelt werden müssen. Für die Geschwindigkeitsregelung wird die Distanz zwischen der aktuellen Position und der Zielposition berechnet und als Fehler $e\,(t)$ an einen PID-Regler weitergeleitet. Dieser berechnet dann die Steuergröße $\mathrm{PID\,(t)} = s\,(t)$ für die abstrakte Maschine für den Vortrieb, die als „Speed" an die Komponente „Engine controller" weitergegeben wird.

Es gibt verschiedene Arten, die Fahrtrichtung des Bootes zu regeln. Sie unterscheiden sich im Wesentlichen in der beobachteten Größe. Die folgenden Varianten wurden für die Regelung des MOPS III realisiert. Alle Varianten berechnen mittels eines PID-Reglers die Steuergröße $\mathrm{PID\,(t)} = r\,(t)$ für das abstrakte Ruder, die als „Rudder an die Komponente „Engine controller" weitergegeben wird. Dabei ist anzumerken, dass bei Winkeldifferenzen stets die kleinere Differenz betrachtet wird: Heading und Bearing sind zwei Winkel zur Bezugsrichtung. Als Differenz bezeichnen wir den kleineren Winkel zwischen den beiden durch die Winkel Heading und Bearing beschriebenen vom Boot ausgehenden Halbgeraden.

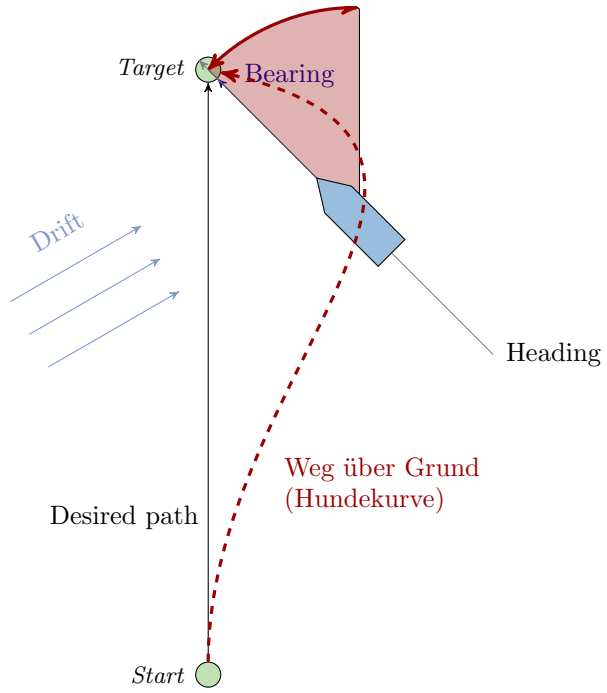

Abbildung 9.4.: Regelung Heading mit Hundekurve

Regelung des Headings

In dieser Variante wird die Differenz zwischen Heading und Bearing berechnet und als Regelungsfehler an einen PID-Regler weitergegeben. Diese Variante hat den Nachteil, dass eine einmal aufgetretene Abweichung vom gewünschten Pfad nicht korrigiert wird. Dadurch kann eine sogenannte Hundekurve entstehen, die unter Umständen den gewünschten Pfad so weit verlassen kann, dass Sicherheitsabstände zu Hindernissen unterschritten werden und Unfälle entstehen können. Abbildung 9.4 zeigt eine solche Hundekurve sowie den Fehler im Vergleich zum Heading aus Abbildung 9.2 (rot schattierter Winkel).

Direkte Regelung des COG

Eine Variante, die Hundekurven zu vermeiden bzw. deutlich zu verringern, ist die Regelung des Kurses über Grund, der in der Implementation des MOPS III anhand einer frei und zur Laufzeit konfigurierbaren Anzahl an gemessenen Positionen berechnet wird. Eine größere Anzahl an Punkten ist robuster gegen „Ausreißer" und Ungenauigkeiten bei der Positionsbestimmung, benötigt aber längere Zeit, bis echte Kursänderungen sich bemerkbar machen. Das Soll ist abermals das Bearing, also die Peilung zum Ziel. Diese Variante berücksichtigt konstante Drift durch sogenanntes „Vorhalten": Solange eine Abweichung des COG (beispielsweise) nach Steuerbord vorliegt, wird nach Backbord gegengesteuert. Dadurch wird „links am Ziel vorbei" gesteuert. Die Summe der Bewegungsvektoren des Bootsantriebs und der Drift ergibt dann im Idealfall den gewünschten COG direkt auf das Ziel zu. Abbildung 9.2 zeigt den gemessenen Differenzwinkel (rot schattiert) und einen möglichen resultierenden Vorhaltewinkel (grün schattiert), der dafür sorgen könnte, dass

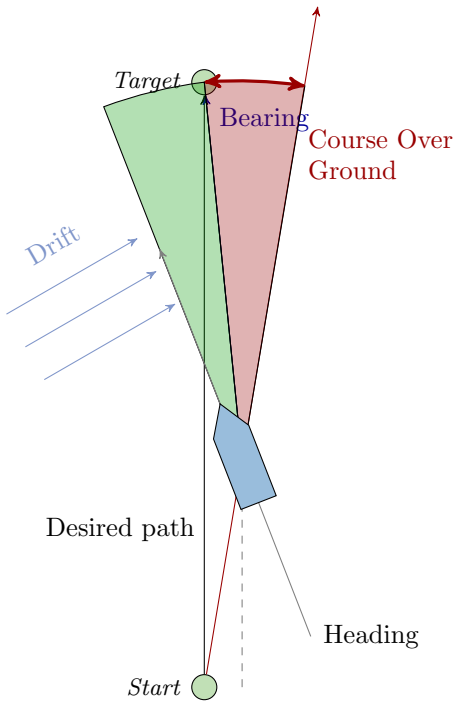

Abbildung 9.5.: Regelung COG mit Vorhaltewinkel

das Boot im weiteren Verlauf einen COG aufweist, der dem gezeigten Bearing entspricht.

Indirekte Regelung des COG

Ein Nachteil der Regelung des Kurses über Grund ist eine größere Zeitverzögerung bei der Reaktion auf Änderungen der Ruderkonfiguration: Je nachdem wie viele Punkte zur Bestimmung des COG gesammelt werden, ist eine Kursänderung tatsächlich erst mit deutlichem Zeitverzug messbar. Dies erschwert die Regelung der Fahrtrichtung: Solange die Abweichung des COG nicht geringer wird, wird der Ruderwinkel nicht reduziert. Die Drehrate wird folglich weiter erhöht. Bei Benutzung von P- und I-Anteilen kann der Ruderwinkel gegebenenfalls sogar vergrößert werden, obwohl sich der Fehler bereits reduziert. Es droht das klassische, aus der Regelungstechnik allgemein bekannte „Aufschwingen" und damit Instabilität des Systems. Eine sorgfältige Konfiguration der PID-Regler und eine möglichst geringe Anzahl an Punkten zur Berechnung des COG sind erforderlich.

Die indirekte Regelung des COG soll diesem Problem ebenfalls entgegenwirken, indem versucht wird, einen Vorhaltewinkel zu antizipieren. Diese Variante nutzt ebenfalls die Differenz zwischen COG und Bearing. Nun werden zwei Annahmen getroffen:

1. Der aktuelle Kurs über Grund ist entstanden, während das Heading auf das Ziel ausgerichtet war.
2. Der Bewegungsvektor, der aus der Spiegelung der Abweichung des Kurses über Grund am Bearing resultiert, kompensiert die Abweichung des COG vom Bearing.

Unter diesen Annahmen wird das Resultat der Spiegelung als Soll für das Heading verwendet. Die Abweichung des aktuellen Headings und des Sollheadings wird dann als Regelungsfehler für den entsprechenden PID-Regler genutzt. Dadurch wird bei einer verzögerten Reaktion des COG auf Ruderbewegungen nicht beliebig gegengesteuert, sondern während dieser Verzögerung „nur" weiterhin ein gegebenenfalls zu großer Vorhaltewinkel gesteuert.

Einer der größten Nachteile dieser Variante ist sicherlich in den Annahmen zu finden. Insbesondere bei Drehungen zu Beginn des Anfahrens einer Position können Abweichungen des COG vom Bearing entstehen, deren Spiegelung ein Heading ergäbe, das vom Ziel wegführte, würde man es einregeln: Angenommen die Peilung zum Ziel ist 0° und der Kurs über Grund ist 100°. Dann ergäbe sich eine Spiegelung von 260°. Ein solches Heading wäre bei jeder Drift kontraproduktiv. Aus diesem Grund muss die indirekte Regelung des COG auf relativ geringe Abweichungen beschränkt werden. Ist beträgt Abweichung des COG in der aktuellen Implementation mehr als 35°, so erfolgt eine einfache Regelung des Heading ohne Antizipation eines Vorhaltewinkels.

Regelung des XTE

Die zuvor beschriebenen Varianten der Regelung der Fahrtrichtung sorgen im Allgemeinen dafür, dass das Ziel nicht verfehlt wird. Die COG-Varianten versuchen zudem, die seitliche Abweichung vom Idealpfad, den Cross Track Error (XTE), zu begrenzen. Einmal entstandene seitliche Abweichungen können jedoch nicht korrigiert werden. Hier muss darauf vertraut werden, dass durch äußere Einflüsse eine Kursabweichung entsteht, die den XTE reduziert. In der Komponente „Path controller" wurde daher ein PID-Regler realisiert, der zusätzlich zu einer der beschriebenen Regelungsvarianten den XTE beobachtet und zu minimieren versucht. Das Soll ist dabei 0, der Regelfehler entspricht der seitlichen Abweichung XTE in Metern. Dieser Regler berechnet ebenfalls eine Steuergröße „Rudder", die dann zur derjenigen Steuergröße hinzuaddiert wird, die aus der Regelung der Fahrtrichtung per Heading oder COG stammt.

9.4.4. Betriebsmodi

Der MOPS muss mitunter eine bestimmte Position für längere Zeit halten können. Diese Funktion lässt sich auf das Anfahren eines Ziels reduzieren. Das Anfahren einer Zielposition und das Halten einer Position können jedoch unterschiedliche Anforderungen an die Pfadregelung stellen. So ist beispielsweise denkbar, dass die Ausrichtung der Längsachse des Bootes auf das Ziel beim Halten einer Position vorsichtiger geschehen soll als beim Anfahren einer Position: Durch die große Nähe zur Position können geringere Änderungen an der Position des Bootes schneller zu größeren Änderungen der Peilung zum Ziel, d. h. des Winkels zwischen Längsachse des Bootes und der Geraden vom Boot zum Ziel, führen. Dadurch können schneller größere Abweichungen von der Sollausrichtung auftreten und damit werden größere Korrekturen nötig werden. Daher werden die folgenden grundsätzlichen Betriebsmodi der Komponente „Path controller" definiert:

1. **Disabled:** Das Boot soll nicht fahren, Motor und Ruder sind in Neutralstellung zu setzen.
2. **Travelling:** Das Boot soll eine Zielposition anfahren.

3. **Holding Position**: Teilweise auch als „Dynamic Positioning" bezeichnet. Das Boot soll eine Position nicht verlassen.

Die Implementation des MOPS III erlaubt die Konfiguration der PID-Regler für Geschwindigkeit und Fahrtrichtung separat für jeden Modus (außer „Disabled").

9.4.5. Definition eines Ziels

Um eine Zielposition anzufahren oder zu halten ist zunächst das Ziel zu definieren. Es genügt nicht, eine geographische Koordinate anzugeben. Die Auflösung der Koordinaten ist deutlich höher als die Genauigkeit mit der der MOPS gesteuert werden kann. Ferner ist die Positionsbestimmung selbst zu ungenau, um exakt festzustellen zu können, ob das Boot eine bestimmte Position erreicht hat. Zudem müsste die Zielposition in genau dem Zeitpunkt eingenommen sein, in dem die Position abgefragt wird – trotz eventueller kleiner Sprünge durch Ungenauigkeiten der Positionsbestimmung und Drift.

Rundet man Längen- und Breitenangabe einer Koordinate auf einige wenige Nachkommastellen, so bezeichnet eine Koordinate nicht mehr einen kleinen Punkt sondern eine größere Fläche. Eine solche Fläche ist leichter zu erreichen, da die zuvor genannten Nachteile sich hierbei nicht entsprechend auswirken können. Allerdings kann das Runden ebenfalls zu einem Fehlverhalten führen: Die Differenz zwischen zwei Längengraden variiert mit Breitengrad der Koordinate. Somit variiert beim Runden auch die Größe der Fläche, die die Koordinate beschreibt.

Wir definieren daher explizit einen Radius in Metern, der die Fläche um eine Position angibt, die als Ziel gilt. Dadurch ist die tatsächliche Flächengröße des Ziels frei wählbar und die maximale Abweichung vom tatsächlichen Ziel direkt kontrollierbar. Die Gründe zur Unterscheidung der Betriebsmodi „Travelling" und „Holding Position" erfordern eine möglichst separate Konfiguration der Komponente je Modus. Daher wird für jeden Modus außer „Disabled" ein eigener solcher Radius definiert. Diese Radien können ebenfalls zur Laufzeit über die Onshore-Software angepasst werden.

- R_{T} für den Modus „Travelling"
- R_{H} für den Modus „Holding Position"

Weiterhin ist eine Regelung der Geschwindigkeit nur in der Nähe des Ziels erforderlich und sinnvoll. Hier sollte die Geschwindigkeit gedrosselt werden, um nicht über das Ziel hinaus zu fahren. Ist der MOPS jedoch noch weit vom Ziel entfernt, kann mit maximaler Geschwindigkeit gefahren werden, ohne die Regelung zu benutzen. Der Radius R_{S}, in dem die Geschwindigkeitsregelung aktiv sein soll, wird ebenfalls in Metern angegeben und ist wie alle anderen Parameter zur Laufzeit änderbar. Dieser Radius dient gleichzeitig als zusätzliche Windup-Kontrolle.

9.4.6. Reglerkombinationen

So wie alle Reglerparameter lassen sich auch die Varianten der Fahrtrichtungsregelung zur Laufzeit je Betriebsmodus über die Onshore-Software auswählen. Die folgenden Varianten sind aktuell möglich:

- Heading
- Heading + XTE

- COG direkt
- COG direkt + XTE
- COG indirekt

Analog zur Komponente „Engine controller" können auch hier aufgrund des modularen Aufbaus der Implementation leicht neue Kombinationen oder Reglerarten erstellt werden. Diese können ebenfalls zum Framework hinzugefügt werden und sind dann über die Onshore-Software auswählbar, ohne dass diese explizit angepasst werden muss. Das Erstellen der Reglervariante und das Hinzufügen der erzeugten Klasse zu einer Liste von Reglervarianten genügt. Dabei ist die aktuelle Implementation so angelegt, dass möglichst viele Komponenten für andere Reglervarianten wiederbenutzt werden können.

9.4.7. Vergleich zu MOPS II und Ergebnisse

In MOPS II wurden bereits auch verschiedene Varianten zur Pfadregelung eingesetzt. Es existierten beispielsweise die Regelung des Headings, des COG (direkt), und auch eine XTE-Regelung ist implementiert worden. Ferner kamen ebenfalls PID-Regler zum Einsatz (vgl. Code und [7]). Durch die Änderung der Architektur der Onboard-Software war es jedoch notwendig, die Regelungssoftware vollständig neu zu implementieren. Dabei wurden verschiedenste Verbesserungen und Optimierungen vorgenommen, von denen an dieser Stelle nur einige exemplarisch benannt werden können. Die deutlich verstärkte Modularisierung der Bestandteile der Regelungsimplementation ist neben der Konfigurierbarkeit zur Laufzeit sicherlich eine der auffälligsten Änderungen. Im Gegensatz zum MOPS II-Code gibt es keine nennenswerten Wiederholungen im Code und die Erweiterbarkeit und Wartbarkeit ist deutlich verbessert.

Obwohl es uns nicht möglich war, die Onboard-Software des MOPS II in Betrieb zu nehmen und zu testen, wies die Reglerimplementationen anscheinend Fehler auf, die instabiles Verhalten wenigstens begünstigt haben dürften. Dies betrifft die Implementation der Regelung des COG und des XTE. Diese Fehler wurden behoben. Zudem wurde für die Regelung des XTE nicht mehr eine manuelle proportionale Berechnung der Steuergröße benutzt, sondern ein eigener PID-Regler (der allerdings wiederum so konfiguriert werden kann, dass er ausschließlich proportional zum Fehler regelt). Dies hat den Vorteil der besseren Konfigurierbarkeit der XTE-Regelung und der Wiederbenutzung einer vorhandenen und zuverlässigen Komponente.

Ferner wurde die indirekte Regelung des COG vollständig neu entworfen und implementiert, diese war in MOPS II nicht vorhanden. Aus MOPS II übernommen wurde die Radienkonzepte zur Definition des Ziels sowie die Reduktion des Haltens einer Position auf das Anfahren dieser Position mit besonderen Regelparametern.

Ergebnisse zur Pfadregelung aus MOPS II liegen uns nicht vor. Testfahrten des MOPS III haben gezeigt, dass sich alle implementierten Reglervarianten so konfigurieren lassen, dass ein relativ genaue und gerade Fahrt möglich ist. Eine Beispielkonfiguration ist in Abschnitt A.7 auf Seite 84 aufgeführt. Im Wesentlichen erscheinen Kombinationen mit Regelung von XTE und COG tatsächlich zu einer etwas genaueren Bahn zu führen. Sie sind aber aus den oben genannten Gründen schwieriger zu konfigurieren. Es ist jedoch klar anzumerken, dass die Testfahrten ausschließlich auf dem Tweelbäker See stattfanden. Hier sind die äußeren Einflüsse, insbesondere die Strömung, nicht sehr ausgeprägt. Testfahrten in Umgebungen mit stärkeren Einflüssen wie beispielsweise dem Jadebusen oder der

Wesermündung erscheinen dringend nötig, um die Zuverlässigkeit der Regelung auch unter solchen Bedingungen zu prüfen.

Das Halten einer Position ist nicht trivial. Für die Einsatzszenarien des MOPS III erscheint das Verweilen in einem Umkreis um die Position ausreichend, da das Vermessen von Wassereigenschaften nicht zwingend zentimetergenau an einer Position erfolgen muss bzw. kann. Insofern erscheint das aus MOPS II übernommene Konzept der Reduktion dieses Problems auf das Anfahren einer Position mit bestimmten Parametern als sinnvoll. Dies ist eine gute Vereinfachung des Problems, die keine zusätzliche Implementation erfordert. Tests haben gezeigt, dass der MOPS den definierten Halteradius gut einhalten kann (tatsächlich muss dieser ja verlassen werden, damit die Regelung aktiv wird): Der Radius wurde in der Regel um nicht mehr als zwei Meter verlassen, in Ausnahmefällen entfernte sich der MOPS um bis zu 3 Meter. Auch hier ist allerdings anzumerken, dass keine Tests bei starkem Wind oder starker Strömung stattgefunden haben. Die Wahl des Heading Reglers ohne XTE erscheint für das Halten einer Position mindestens als ausreichend, wenn nicht gar sinnvoll.

Tatsächlich erscheint das Halten einer Position mit einer gewissen Drift besser zu funktionieren als ohne: Mit nur geringer Drift verlässt das Boot irgendwann den einzuhaltenden Kreis und wird durch die Regelung – dann im Allgemeinen entgegen der Drift – zurückgeführt. Ist die Drift gering, treibt das Boot nach Abschalten der Motoren deutlich länger in die angesteuerte Richtung und verlässt dann den einzuhaltenden Radius auf der anderen Seite. Ist jedoch eine gewisse Driftstärke vorhanden, so wirkt diese dem Weitertreiben entgegen. Die Folge ist, dass das Boot sich am Rand des einzuhaltenden Kreises relativ stabil einregeln kann. Eine Reglerkonfiguration, die sowohl mit sehr geringer als auch mit stärkerer Drift die Position stabil einregeln kann, konnte nicht ermittelt werden. Dennoch erscheint die aktuelle Regelung der Position als genau genug, zumal das Halten der Position vermutlich in der Regel mit in das Wasser hinabgelassenen Payload-Sensoren erfolgen wird. Diese bieten eine zusätzliche Angriffsfläche für Strömung und können so die Drift erhöhen, sodass eine gute Positionsregelung mit Drift als wertvoller erscheint als eine, die ausschließlich mit geringer Drift funktioniert.

Soll das Halten der Position dennoch verbessert werden, so kann versucht werden, das Verlassen des einzuhaltenden Kreises zu antizipieren: Das Bearing kann ebenso berechnet werden wie die Geschwindigkeit über Grund gemessen werden kann (per GPS). Ein optimierter Regler zum Halten einer Position könnte versuchen, dem aus diesen beiden Werten resultierenden Bewegungsvektor entgegenzuwirken, indem er versucht, das Heading auf das Bearing einzuregeln und die Geschwindigkeit über Grund auf 0. Dabei muss diese Form der Regelung auch innerhalb des einzuhaltenden Kreises aktiv sein. Vermutlich ergeben sich jedoch in der Nähe der Zielkoordinate Probleme durch ständige Drehmanöver durch das Einregeln des Headings. Eine Kombination aus einem (kleinen) Radius, in dem die Regelung nicht aktiv ist, und der Regelung der Geschwindigkeit über Grund kann sinnvoll sein.

9.5. Mission controller

9.5.1. Allgemeines zum Missionskontrolle

Der „Mission controller" ist eine Komponente, die basierend auf Missionsplan und Benutzereingaben wie „Start", „Stop", „Pause" oder „Return To Home" die weiteren Komponenten direkt oder indirekt steuert. So setzt sie beispielsweise den Betriebsmodus der

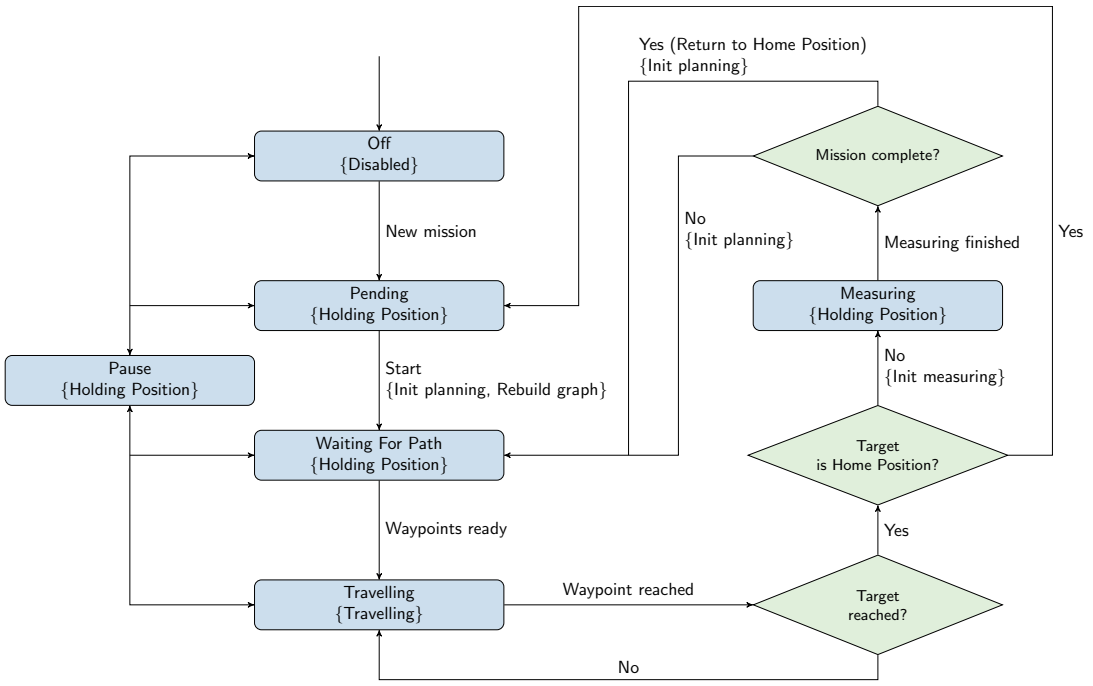

Abbildung 9.6.: Ablaufschema einer Mission

Komponente „Path controller" und bestimmt die zu haltende oder anzufahrende Position. Außerdem werden in separaten Threads implementierte Komponenten wie die Bahnplanung oder die Payload-Messung gestartet. Die Kommunikation erfolgt dabei ausschließlich über `Data`.

Die Unterscheidung der Begriffe „Zielposition" (auch „Ziel" oder „Zielpunkt" genannt) und „Wegpunkt" ist im Folgenden essentiell: Die Zielposition ist das Ziel einer Fahrt oder die Position, die zu halten ist. Wegpunkte sind Zwischenpositionen, die auf dem Weg zum Ziel angesteuert werden, um Hindernisse zu umfahren. An Wegpunkten erfolgt somit niemals eine Messung mit den Payload-Sensoren. Ziele werden in Messpunkte und Home-Positions unterschieden: Erstere sind die für die Mission wesentlichen Positionen; hier soll eine Messung vorgenommen werden. Eine Home-Position dient ausschließlich „organisatorischen" Aufgaben: Hierher soll das Boot im Gefahrenfall oder nach Missionende zurückkehren. Hier wird also ebenfalls keine Messung vorgenommen.

9.5.2. Zustandsautomat

Die Komponente „Mission controller" ist im Wesentlichen als Zustandsautomat implementiert. Abbildung 9.6 zeigt ein Schema dieses Automaten, das den Ablauf einer Mission beschreibt. Die Zustände sind als blaue Rechtecke dargestellt. Der Name des Zustands ist in der ersten Zeile aufgeführt. In der zweiten Zeile ist in geschweiften Klammern der Modus der Komponente „Path control" dargestellt, der vom „Mission controller" ausgelöst wird und für die Dauer des Aufenthalts im Zustand gelten soll. Grüne Rauten bezeichnen Entscheidungen im Ablauf. Aus Zuständen ausgehende Übergänge sind mit eingehenden Signalen beschriftet, aus Entscheidungen ausgehende Übergänge mit den entsprechenden Fällen. In geschweifte Klammern gefasste Übergangsbeschriftungen bezeichnen nur beim Betreten des Zustands ausgelöste Events.

Aus Gründen der Leserlichkeit wurde die Abbildung vereinfacht. So wurden beispielsweise Übergänge, die auf Benutzereingaben wie „Stop", „Return To Home" oder das Übertragen einer neuen Mission zur Laufzeit einer anderen Mission basieren, ausgelassen. Auf die Eingabe „Stop" wechselt der Automat in den Zustand „Off", in dem die Motoren ausgeschaltet sind. Auf die Eingabe „Return To Home" wechselt das System in den Zustand „Waiting For Path", wobei als nächstes Ziel die Home-Position gesetzt wird. Wird eine neue Mission übertragen, während eine Mission ausgeführt wird, erfolgt ein Wechsel in den Zustand „Pending", in dem auf den Start der neuen Mission gewartet wird.

Initial befindet sich das System also im Zustand „Off". Durch das Hochladen einer Mission wechselt es in den Zustand „Pending", in dem auf das Start-Signal gewartet wird. In diesem Zustand wird der „Path controller" in den Modus „Holding Position" versetzt, in dem die Position gehalten werden soll, die das Boot zum Zeitpunkt des Betretens des Zustands hatte.

Wird das Start-Signal durch Benutzereingabe an der Onshore-Software gesendet, wechselt das System in den Zustand „Waiting For Path". Dabei wird zunächst das Signal „Init planning" an die als `AbstractIO` realisierte und somit parallel arbeitende Komponente „Path planner" gesendet. Ferner wird, da eine neue Mission vorliegt, das Signal „Rebuild graph" gesendet. Dieses Signal veranlasst den völligen Neuaufbau des Planungsgraphen von „Path planner" und wird nur dann gesendet, wenn eine neue Mission vorliegt. Der „Path planner" beginnt dann, den Weg zum ersten Zielpunkt der Mission zu berechnen. Wie im Zustand „Pending" wird auch während des Verweilens in diesem Zustand die Komponente „Path controller" angewiesen, die aktuelle Position zu halten.

Sendet „Path planner" das Signal „Waypoints ready", ist die Bahnplanung beendet und die benötigten Wegpunkte zum Ziel liegen vor. Die Komponente „Mission controller" wechselt in den Zustand „Travelling". Sie wählt dabei den ersten Wegpunkt aus und setzt ihn als nächste anzufahrende Position. Dann wird die Komponente „Path controller" in den Modus „Travelling" versetzt. Kann jedoch kein Pfad gefunden werden, wechselt „Mission controller" in den Zustand „Pending" und versucht, die aktuelle Position zu halten.

Voraussetzung für den Wechsel nach „Travelling" ist, dass ein gefundener Pfad valide ist. Bei der Initiierung der Bahnplanung wird eine Prüfzahl generiert und an die Planung weitergeleitet. Ist die Planung abgeschlossen, wird die Prüfzahl an „Mission controller" zurückgegeben. Der Pfad ist dann valide, wenn die letzte generierte Prüfzahl mit der von „Path controller" zurückgegebenen übereinstimmt. Dieses Verfahren ist nötig, um einen Effekt der Nebenläufigkeit zu kompensieren: Während die Planung läuft, könnte eine neue Mission gesendet werden. Die aktuelle Implementation erlaubt nicht das (technisch saubere) aktive Beenden einer Planung. Wird also dann die Planung beendet, würde ohne diese Prüfzahl „Mission controller" beginnen, den auf Grundlage der alten Mission Pfad abzufahren. Gleichzeitig könnte die Bahnplanung nun mit der Planung des Pfades zu einem Ziel der neuen Mission beginnen und dann während des Abfahrens des alten Pfades den neuen Pfad zurückgeben. Dadurch könnten beim Wegpunktwechsel unkontrolliert Pfade entstehen, die durch Hindernisse hindurch führen. Die leicht zu implementierende Prüfzahl erlaubt es, Pfade, die auf alten, ungültigen Missionen basieren, zu verwerfen und dieses Problem zu lösen.

Ist der Wegpunkt erreicht, sendet die Komponente „Path controller" das entsprechende Signal. Ist der angefahrene Wegpunkt noch nicht das Ziel der Fahrt, so wird der nächste Wegpunkt als anzufahrende Position gesetzt. Die Komponenten „Mission controller" und

„Path controller" verbleiben im Zustand „Travelling". Ist der erreichte Wegpunkt jedoch zugleich das Ziel der Fahrt, so wird überprüft, ob die Fahrt eine Fahrt zur Home-Position war. Falls ja, so wechselt das System in den Zustand „Pending". Der „Path controller" wird abermals angewiesen, die beim Betreten des Zustands eingenommene Position zu halten. In diesem Zustand wartet der MOPS dann auf weitere Missionen oder das Ausschalten. War das Ziel jedoch keine Home-Position, so wechselt der „Mission controller" in den Zustand „Measuring". In diesem Zustand wird ebenfalls die beim Betreten des Zustands eingenommene Position versucht zu halten. Ferner wird beim Wechseln nach „Measuring" der Komponente zur Steuerung der Payload-Sensoren das Signal „Init measuring" gesendet.

Ist die Messung abgeschlossen, sendet das Messsystem ein Signal „Measuring finished". Gibt es keinen weiteren Zielpunkt in der Mission, wird die Home-Position als nächstes anzufahrendes Ziel ausgewählt. Sonst wird der nächste Messpunkt in der Liste der Messpunkte des Missionsplans als nächstes Ziel gewählt. Komponente „Mission controller" wechselt in den Zustand „Waiting For Path", wobei die aktuelle Position gehalten werden soll und die Bahnplanung initiiert wird.

Aus den Zuständen „Off", „Pending", „Waiting For Path" und „Travelling" heraus ist es möglich, in den Zustand „Pause" zu wechseln, wenn der Benutzer über die Onshore-Software das entsprechende Signal sendet. Dabei werden die aktuellen Zustände bzw. Modi der Controller, der aktuelle und der letzte Wegpunkt (letzterer wird für die XTE-Regelung benötigt) gespeichert und der „Path controller" wird angewiesen, die aktuelle Position zu halten. Wird das Start-Signal gesendet, werden die gespeicherten Zustände wiederhergestellt.

9.5.3. Designentscheidungen zur Übergangsrelation

Aus dem Zustand „Measuring" ist aktuell kein Wechsel in den Zustand „Pause" aktiviert. Grundsätzlich sind alle Benutzereingaben in diesem Modus derzeit deaktiviert. Das hat mehrere Gründe. Einerseits ist aufgrund technischer Schwierigkeiten die Benutzung der Payload-Sensoren aktuell nicht möglich. Somit sendet die entsprechende Komponente direkt nach dem Signal „Init measuring" triviallerweise „Measuring finished" zurück. Eine Reaktion auf eine Benutzereingabe wird also höchstens um zwei Zykluszeiten verzögert. Andererseits birgt das Reagieren auf Benutzereingaben gewisse Risiken: Es muss dafür Sorge getragen werden, dass eventuell von der Measuring-Komponente gesendete Signale, die zeitlich nach der Reaktion auf eine Benutzereingabe eintreffen, nicht einen nachfolgenden Messvorgang beeinflussen. Wird beispielsweise eine neue Mission gesendet und wechselt der „Mission controller" in den Modus „Pending", und wird danach das Signal „Measuring finished" gesendet, so wird beim nächsten Betreten des Zustands „Measuring" wiederum eine Messung initiiert. Gleichzeitig könnte noch das alte Signal „Measuring finished" vorliegen, welches das Boot zum Weiterfahren veranlassen könnte, obwohl noch eine Messung läuft. In diesem Beispiel müsste beim Betreten des Zustands „Measuring" das Signal „Measuring finished" zurückgesetzt werden. Das Zurücksetzen von Signalen zwischen parallelen Komponenten ist im Allgemeinen nicht trivial. Es muss stets beachtet werden, wann ein Signal zurückgesetzt werden darf und wann nicht. Da Zuverlässigkeit für MOPS III im Vordergrund stand, wurde entschieden, für diesen Zustand vorerst auf eine Reaktion auf Benutzereingaben zu verzichten. Dadurch entsteht gegebenenfalls eine nur sehr gering verzögerte Reaktion auf Benutzereingaben, aber potentielle Fehlerquellen werden reduziert.

Zudem ist das Reagieren auf Benutzereingaben während des Messens nur dann sinnvoll, wenn die entsprechende Reaktion auch an die Messvorrichtung weitergegeben werden kann. Das Boot sollte keinesfalls nicht genau geplante Fahrmanöver mit im Wasser befindlichen Sensoren durchführen. Es könnte beispielsweise dabei in Regionen vordringen, in denen die Sensoren auf Hindernisse oder Grund treffen. Aktuell ist es nicht möglich, die Sensoren aus dem Wasser zu ziehen. Daher sollten Messungen ohnehin sicherheitshalber noch nicht während des Ausführens einer autonomen Mission durchgeführt werden, sondern beispielsweise nur bei trivialen Missionen bei denen Start, Ziel und Home Position der aktuellen Position des Bootes entsprechen. Können die Sensoren automatisch aus dem Wasser gezogen werden, kann entsprechend auf Benutzereingaben reagiert werden, die Fahrmanöver zur Folge haben. Bei der Aktivierung des entsprechenden Codes im „Mission controller" kann und sollte dann ein entsprechendes Signal an die Measuring-Komponente implementiert werden, wobei das Fahrmanöver erst ausgeführt werden sollte, wenn die Sensoren vollständig aus dem Wasser gezogen wurden. Derzeit existiert allerdings noch keine solche Signal-Schnittstelle, sodass im „Measuring"-Zustand sicherheitshalber auf die Reaktion auf Benutzereingaben verzichtet wurde, um keine Funktionalität zu bieten, die ohne Erweiterung (Anweisung an die Measuring-Komponente, die Sensoren vorzeitig zu bergen) zu einem Sicherheitsrisiko werden kann.

Grundsätzlich wird nicht in jedem Zustand auf jedes Signal reagiert: Die Übergangsrelation des implementierten Automaten ist nicht explizit total definiert. So wird im Zustand „Pause" beispielsweise nicht auf das Signal „New mission" reagiert. Während „Pause" soll keine andere Aktion erfolgen. Kehrt das System aus dem „Pause"-Zustand zurück, liegt das Signal „New mission" noch vor. Nun wird direkt darauf reagiert. So kann die Verarbeitung bestimmter Signale an die entsprechenden Zustände gekapselt werden.

9.5.4. Signalhierarchie

Bei jeder Ausführung der Main-Loop entscheidet die Komponente „Mission controller" also basierend auf dem aktuellen Zustand und den gesetzten Signalen den Folgezustand des Systems. Da stets mehrere Signale gleichzeitig gesetzt sein können, berücksichtigt die Entscheidung die folgende, absteigende Hierarchie („Stop" ist das am höchsten priorisierte Signal):

1. Signal „Stop"
2. Signal „Return To Home"
3. Signal „Pause"
4. Signal „New mission"
5. Signal „Waypoints ready", Signal „Start", Signal „Measuring finished"

9.5.5. Vergleich zu MOPS II

Die Implementation der Komponente „Mission controller" als Automat mit weitestgehend an Zustände gekapselte Verarbeitung von Signalen hat sich als vergleichsweise gut überschaubare und wartbare Lösung erwiesen. Die Umsetzung der Missionskontrolle in MOPS II war deutlich unübersichtlicher implementiert und durch das verwendete Thread- und

Eventsystem fehleranfällig (vgl. Code). Unter anderem aus diesem Grund ist eine Herauslösung aus der alten Software und Anpassung an die neue Software-Architektur gescheitert. Der „Mission controller" wurde im Zuge der Umsetzung der in Abbildung 9.1 gezeigten hierarchischen Modulstruktur der Planungs- und Steuerungssoftware neu definiert und implementiert. Bei einer Erweiterung der Komponente ist die Einhaltung eines gültigen Zustands besonders zu beachten: Wie im Beispiel des „Measuring"-Zustands muss klar definiert sein, zu welchem Zeitpunkt welche Signale zurückgesetzt werden dürfen, müssen oder können. Dieses Problem ist durch Nebenläufigkeiten nicht unbedingt trivial. Die beschriebene Kapselung der Signalverarbeitung an Zustände hat sich zusammen mit einer klaren Signalhierarchie in diesem Kontext als deutlich hilfreich und eine der wesentlichen Verbesserungen der Missionskontrolle gegenüber MOPS II herausgestellt.

9.6. Path planner

9.6.1. Allgemeines zur Pfadplanung

Die Komponente „Path planner" ist für die Bahnplanung zuständig. Als Bahn oder Pfad bezeichnen wir dabei den Weg von einer Startposition zu einem Zielposition. Sie ist nicht wie die zuvor beschriebenen Komponenten als `Component` realisiert, sondern als `AbstractIO`. Dies erlaubt die Ausführung parallel zur Main-Loop. Dadurch wird diese, die zeitkritische Aufgaben wie die Pfadregelung ausführt, auch bei längeren Laufzeiten nicht blockiert. Die Kommunikation mit „Mission controller" erfolgt auch hier über `Data`.

Der wesentliche Zweck der Bahnplanung ist das Umfahren von Hindernissen. Hindernisse können feste Objekte wie Ufer, Inseln, Leuchttürme, Untiefen oder aus sonstigen Gründen zu vermeidende Bereiche sein. Aber auch bewegliche Objekte wie andere Schiffe können als Hindernis betrachtet werden. Daher ist die Bahnplanung onboard-seitig implementiert, um auf dynamisch auftretende Hindernisse reagieren zu können. Ferner wurde zu Projektbeginn vom ICBM ein mögliches Einsatzszenario geschildert, bei dem das Boot während des Messens im Wasser treiben soll. Dadurch zum Zeitpunkt der Planung der Mission der Startpunkt des Pfades zum nächsten Messpunkt nicht zwingend bekannt. Daher kann die Bahnplanung im Allgemeinen (auch wenn es Ausnahmen gibt) erst direkt vor dem Anfahren des nächsten Ziels erfolgen. Auch dadurch ist eine onboard-seitige Lösung notwendig.

Grundsätzlich sind bei der Bahnplanung diverse Aspekte zu berücksichtigen. Es geht nicht nur darum, eine Reihenfolge von Wegpunkten zu finden, die im Idealfall kein Hindernis schneiden. Die Schiffsdynamik erfordert auch Überlegungen, wie ein Punkt angefahren wird. Als Beispiel mag ein Punkt gelten, an dem nach Steuerbord abgebogen werden soll. Hinter diesem Punkt befinde sich ein Hindernis, beispielsweise eine Kaimauer. Wird dieser Punkt angefahren ohne zu bremsen, so kann durch den entstehenden Wendekreis eine Kollision mit der Kaimauer womöglich nicht vermieden werden, obwohl der berechnete Pfad an sich kein Hindernis schneidet. Durch die Bauart des MOPS lässt sich diese Art von Problemen leicht lösen: Das Boot lässt sich auf der Stelle drehen. Wir können also Wegpunkte anfahren, dort halten, in Richtung des nächsten Wegpunktes drehen, und dann weiterfahren. Dadurch entsteht nicht unbedingt eine flüssige Fahrt, allerdings ist das Prinzip immun gegen die oben geschilderte Art von Problemen. Da einfache Systeme leichter zuverlässig zu realisieren sind, haben wir uns für das geschilderte Prinzip des

Anfahrens von Wegpunkten entschieden und verzichten auf komplexe Algorithmen zur Berechnung von kollsionsfreien Kurven durch Wegpunkte.

Es gibt verschiedenste Verfahren zur Bahnplanung. Unter der Berücksichtigung der Art der zu lösenden Aufgabe, nämlich ein Fahrzeug entlang von geraden Teilstücken einer nicht glatten Trajektorie zu einem Ziel zu führen, erscheint die Suche eines kürzesten Weges in einem Graphen intuitiv als gutes Verfahren. Das graphische Skizzieren des Problems erzeugt bereits einen solchen Graphen. Zudem ist die Suche von kürzesten Wegen in Graphen mit bereits relativ simplen Algorithmen effizient zu bewältigen und leicht zu verstehen. Daher kommt ein solches Suchverfahren für die Bahnplanung des MOPS III zum Einsatz. Aktuell verfügt der MOPS über keine Möglichkeiten, zuverlässig dynamische Hindernisse erkennen zu können. Daher basiert die Bahnplanung ausschließlich auf den mit dem Missionsplan zur Verfügung gestellten Hindernisinformationen. Die Hindernisse werden dort in Form von Polygonen beschrieben, die dabei keinen besonderen Einschränkungen unterliegen. Die Polygone selbst werden als Liste von Punkten definiert.

Da später die Ränder der Polygone in die Pfadsuche einbezogen werden und ihre Ecken als Wegpunkte dienen, sollten sie mit einem entsprechenden Sicherheitsabstand um Hindernisse herum definiert werden. Dabei ist auch zu beachten, dass beim Entlangfahren an der Kante eines Polygons nicht exakt gerade gefahren werden kann. Abweichungen vom Pfad können ein Stück weit in das Hindernis hinein führen. Es wurden zwar Funktionen zur Vergrößerung von Polygonen implementiert um diesen Sicherheitsabstand implizit zu gewährleisten, allerdings wurde später von der Benutzung dieser Funktionen Abstand genommen, da dadurch möglicherweise aus für den Benutzer nicht offensichtlichen Gründen scheinbar freie Wege nicht genutzt werden können. Ferner erscheint es sinnvoller, den Benutzer die Sicherheitsabstände manuell zeichnen zu lassen. So könnte beispielsweise der benötigte Sicherheitsabstand auf der Luv-Seite eines Hindernisses größer gewählt werden als auf der Lee-Seite. Eine implizite Einrechnung eines Sicherheitsabstands könnte solche Faktoren nicht ohne Weiteres berücksichtigen und durch Überapproximation des Abstands unnötig große Areale für das Boot sperren.

9.6.2. Graphkonstruktion

Die Planung wird durch die Komponente „Mission controller" über das bereits beschriebene Signal „Init planning" (im Code als `MISSION_PLANNER_START` realisiert) ausgelöst. Als erster Schritt wird dabei ein als Adjazenzliste implementierter, kantengewichteter, ungerichteter Graph erzeugt, der die möglichen Pfade repräsentiert. Von diesem Graphen wird allerdings zunächst nur der Kern erzeugt, der für alle Fahrmanöver der aktuellen Mission wiederverwendet werden kann. Dieser Kern beinhaltet nur die Hindernisse. Diese werden als Knoten dem Graphen hinzugefügt. Dann wird versucht, zwischen allen paarweise verschiedenen Knoten eine Kante zu erzeugen. Als Kantengewicht wird die Länge des Wegs, den die Kante repräsentiert, verwendet. Eine Kante ist dann zulässig, wenn sie kein Hindernis schneidet. Das Berühren des Rands eines Polygons gilt dabei nicht als Schneiden eines Hindernisses. Sonst könnte nicht am Rand eines Hindernisses entlang gefahren werden.

Die Erzeugung dieses Graphen (bzw. Kernstücks des Graphen) ist der rechenintensivste Teil der Bahnplanung, da hierbei viele Randfälle betrachtet werden müssen. Beispielsweise darf das Berühren eines Eckpunktes eines Polygons nicht als Schneiden eines Hindernisses gelten. Andererseits darf eine Diagonale eines Rechtecks keine Kante des Graphen sein. Daher wird dieses Kernstück des Graphen für jede Planungsoperation der Mission wiederverwendet.

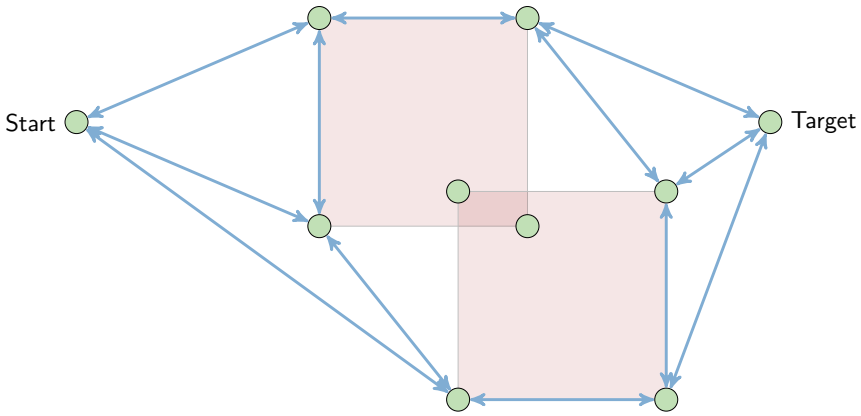

Abbildung 9.7.: Beispiel eines Planungsgraphen

Die vollständige Neuplanung des Graphen erfolgt nur dann, wenn in „Path planner” noch kein Graph vorliegt oder durch „Mission controller” das Signal „Rebuild graph” gesendet wird.

Für jede Planungsoperation werden dann Startpunkt und Zielpunkt zum Graphen hinzugefügt und diejenigen Kanten angelegt, die die freien Pfade von und zu diesen Punkten repräsentieren. Nun existiert ein Graph, dessen Knoten Wegpunkte repräsentieren und dessen gewichtete Kanten die freien Wege und ihre Längen in der durch die Polygone des Missionsplans definierten Umwelt darstellen. Auf diesem Graphen kann ein Algorithmus zur Suche von kürzesten Pfaden ausgeführt werden. Abbildung 9.7 zeigt einen solchen Graphen. Die roten Flächen repräsentieren Hindernisse, die blauen Pfeile die Kanten des Graphen.

Nach Abschluss der Suche können Start und Ziel wieder aus dem Graphen entfernt werden. Dies ist nicht zwingend nötig, es entstehen in späteren Planungsoperationen keine unerlaubten Kanten, falls dies nicht geschieht. Möglicherweise ist es sogar effizienter, die Knoten und Kanten im Graphen zu belassen, sofern ein guter Suchalgorithmus eingesetzt wird: Der Aufwand zum Entfernen der Knoten ist dann größer als der Aufwand, der durch zusätzliche Exploration bei der Pfadsuche entsteht. Tatsächlich entstünde beim Hinzufügen von weiteren Start- und Zielpunkten kein zusätzlicher Aufwand: Start- und Zielpunkte werden ausschließlich mit Ecken von Polygonen verbunden, da die Verbindung mit einem einzelnen Start- oder Zielpunkt keinen zusätzlichen kürzeren Weg erzeugen kann. Zu Testzwecken ist das Entfernen von Start und Ziel aus dem Graphen nach der Pfadsuche implementiert. Durch die Darstellung des Graphen als Adjazenzlisten kann ein Teil der Löschoperationen sehr effizient ausgeführt werden: Die Nachfolgerlisten von Start- und Ziel, die sich im Regel am Ende der Hauptliste befinden, können direkt gelöscht werden. Dabei bieten sie zuvor eine Lookup-Table, in welchen der Nachfolgerlisten Start und Ziel als Nachfolger zu löschen sind. Dieses Verfahren ist vermutlich effizienter als den Graphen über tiefe Kopien zurückzusetzen, insbesondere bei Großen Graphen, in denen Start und Ziel nur mit einer kleineren Teilmenge der Knotenmenge verbunden sind.

Aktuell ist das Löschen von Start und Ziel aktiv. Es konnten bisher keine wesentlichen Laufzeitunterschiede zum Verfahren ohne Löschen festgestellt werden.

9.6.3. Suchalgorithmus

Es existieren diverse gute Suchalgorithmen, die auf dem erzeugten Graphen einen kürzesten Weg effizient finden. Weit verbreitet sind unter anderem der A*-Algorithmus und seine verschiedenen Varianten. Sie sind in der Lage, den kürzesten Weg in einem Graphen mit positiven Kantengewichten effizient zu finden. Gleichzeitig ist insbesondere der A*-Algorithmus selbst relativ einfach zu implementieren und dadurch gut wartbar und wenig fehleranfällig. Dies entspricht dem allgemeinen Projektziel, ein hoch zuverlässiges System zu realisieren. Daher kommt eine Variante des A*-Algorithmus für die Bahnplanung des MOPS III zum Einsatz, der gewichtete A*-Algorithmus ARA* [8].

Der gewichtete A*-Algorithmus ist wie der A*-Algorithmus ein informierter Suchalgorithmus, der mittels einer Schätzfunktion $h(x)$ die Länge des Restwegs vom aktuell explorierten Knoten x zum Ziel abschätzt. Dabei darf die Schätzfunktion die Länge des Restwegs nie über-, wohl aber unterschätzen. Für MOPS III berechnet die Schätzfunktion die Länge der kürzesten Orthodrome vom aktuellen Knoten zum Ziel.

Die Idee des Algorithmus lässt sich wie folgt grob skizzieren: Zu beginn wird der Startknoten mit den Kosten 0 in eine Warteliste eingetragen. Danach wird wie folgt verfahren bis die Warteliste leer ist (dann existiert kein Pfad) oder der Zielknoten erreicht wurde (dann wurde der kürzeste Pfad gefunden): Es wird der Knoten mit den geringsten Kosten aus der Warteliste ausgewählt und entfernt. Nun werden alle Nachfolger dieses Knotens exploriert. Das bedeutet, sie werden in die Warteliste eingetragen, falls sie noch nicht darin enthalten sind oder die neu berechneten Kosten geringer sind als die in der Warteliste vermerkten Kosten. Die neu berechneten Kosten betragen dabei die Kosten des soeben aus der Warteliste entfernten Knotens zuzüglich der Kosten der Kante zum Nachfolgerknoten und des Wertes der Schätzfunktion für den Nachfolgerknoten.

Die Erweiterung zum gewichteten Algorithmus besteht darin, die Schätzfunktion um einen positiven Faktor nicht kleiner 1 zu erweitern, also zu $w \cdot h(x)$ mit $w \geq 1$. Dadurch wird dem Algorithmus erlaubt, Wege zu finden, nicht optimal sind. Der Faktor definiert eine obere Schranke für die Abweichung vom Optimum: Mit $w = 1.1$ sind beispielsweise Wege erlaubt, die bis zu 10% länger sind als der kürzeste Weg. Dadurch ist es in manchen Szenarien möglich, die Laufzeit des Algorithmus deutlich zu verkürzen. Gibt es beispielsweise sehr viele Pfade zum Ziel, und sind viele diese Pfade und ihre Teilpfade in der Länge ähnlich, dann können Situationen entstehen, in denen der Algorithmus mitunter sehr viel Explorationsarbeit leisten muss. Der Suchvorgang kann dann gegebenenfalls durch die Gewichtung abgekürzt werden.

Die detaillierte und exakte Beschreibung des Algorithmus findet sich in [8]. Tests haben ergeben, dass die Benutzung einer gewichteten Schätzfunktion für die Szenarien am Tweelbäker See keinen Nutzen haben. In der Regel gibt es nicht genügend Pfade mit entsprechenden Kriterien, sodass ein solcher Nutzen entstehen kann. Die Implementation des Gewichts hat jedoch keinerlei nennenswerte Nachteile. Sie ist daher erhalten geblieben, wobei das Gewicht auf 1.0 fixiert wurde. Damit arbeitet der Algorithmus wie der ursprüngliche A*-Algorithmus. Die gewichtete Schätzfunktion kann jedoch bei bestimmten Erweiterungen der Bahnplanung nützlich sein. Der Aufbau von Suchgraphen, die beispielsweise eine komplexere Auswahl aus verschiedensten Fahrmanöver an den Eckpunkten einer Trajektorie berücksichtigen, könnte eine solche Erweiterung sein.

9.6.4. Pfadkonstruktion

Wenn der Suchalgorithmus den Zielknoten gefunden hat, muss noch der Pfad konstruiert werden. Jeder Knoten des Graphen beinhaltet eine Referenz auf den Vorgängerknoten, von dem aus der Suchalgorithmus den jeweiligen Knoten exploriert hat. Die Konstruktion des Pfades besteht also darin, ausgehend vom Zielknoten den Vorgängerknoten auszuwählen, bis der Startknoten erreicht ist. Die ausgewählten Knoten werden umkehrt sortiert (damit der Startknoten der erste Wegpunkt ist) in einer Liste als Wegpunktliste in `Data` hinterlegt.

9.6.5. Ergebnisse

In der Software des MOPS II finden sich Code-Fragmente einer Bahnplanung zur Umgehung von Hindernissen. Die Hindernisse bestehen dabei offensichtlich ausschließlich aus Quadraten. Zudem ist die Planung innerhalb der Onshore-Software realisiert. Diese Planung konnte jedoch nie getestet werden: Jeglicher Versuch dazu führte zu einem Abbruch mit Exceptions. Es muss angenommen werden, dass diese Funktionalität nicht vollständig in MOPS II umgesetzt wurde, obwohl ihre Existenz in einer Tabelle des Abschlussberichts der Projektgruppe zumindest angedeutet wird. Außerdem erfüllt eine onshore-seitige Planung nicht die in Abschnitt 9.6 beschriebenen Anforderungen.

Daher kann die Bahnplanung des MOPS III als neues Feature gegenüber MOPS II betrachtet werden. Zudem können beliebige Polygone behandelt werden, nicht nur Quadrate. Außerdem erfolgt die Planung onboard-seitig, sodass zur Laufzeit auftretende Ereignisse wie dynamische Hindernisse oder erst zur Laufzeit bekannte Startpunkte berücksichtigt werden können.

Insgesamt hat sich die implementierte Bahnplanung als einfach aber zuverlässig erwiesen. Durch die in Abschnitt 9.2 beschriebene Rechnung mit Orthodromen ergeben sich mitunter längere, aber moderate Laufzeiten. In einem Szenario mit mehreren überlappenden und komplexen Polygonen, nichttrivialen Pfaden und einem Graphen mit circa 90 Knoten ergab sich auf der MOPS-Hardware bei der initialen Planung mit Graphkonstruktion eine Laufzeit von ca. 60 Sekunden. Die weiteren Planungsphasen, bei denen nur Start- und Ziel hinzugefügt und entfernt wurden, dauerten etwa 5 Sekunden.

Für den aktuellen Entwicklungsstand des MOPS sind dies durchaus ausreichende Laufzeiten. Dies gilt insbesondere, da aktuell die Position während der Planungsphase gehalten wird. Dies kann theoretisch belieblig lange fortgesetzt werden. Für zeitkritische Planungen, wie sie möglicherweise bei der Planung von Ausweichmanövern entstehen können, sollten die Abschnitt 9.2 aufgeführten oder eventuell bekannte andere Optimierungsansätze für `GeoCalc` geprüft werden, um Planungszeiten falls erforderlich zu reduzieren. Eine nur auf die Planung bezogene Optimierung könnte beispielsweise auf die Vorausberechnung der möglichen Kanten und ihrer Gewichte verzichten und diese während der Planungsphase explorieren. Dadurch könnte ggf. die Anzahl der Kanten, die teuer auf Kollisionsfreiheit untersucht werden müssen, unter Ausnutzung der Vorteile des A*-Algorithmus (tendentiell partielle Exploration des Graphen mit Bevorzugung vielversprechender Kanten) reduziert werden.

Prinzipiell können weitere Hindernispolygone problemlos zu einem bestehenden Graphen hinzugefügt werden. Dies ist für dynamisch auftretende Hindernisse relevant. Das Entfernen des dynamischen Hindernisses ist dann ebenfalls erforderlich, falls das Areal abermals passiert werden muss. Dies kann mit ähnlichen Mitteln geschehen, wie das Entfernen von

Start- und Zielpunkt nach einer Planungsphase. Es ist jedoch zu bedenken, dass durch neue Hindernisse bisher kollisionsfreie Wege versperrt werden können. Die betroffenen Kanten müssen entsprechend markiert oder entfernt und nach Entfernen des Hindernisses wiederhergestellt werden.

Im Kontext dynamischer Hindernisse bleibt zu prüfen, ob andere Suchalgorithmen besser funktionieren. So existiert beispielsweise eine Variante des A*-Algorithmus, der D*-Algorithmus, der auf sich dynamisch verändernde Umgebungen ausgelegt ist.

9.7. Erweiterungsansätze

Die Planungs- und Steuerungssoftware bietet durch ihre hochgradig modulare Strukture gute Ansatzpunkte für Modelchecking mit parallelen Automaten. Die Überprüfung verschiedener Eigenschaften mittels eines solchen Verfahrens war für MOPS III vorgesehen, konnte jedoch zeitlich nicht mehr realisiert werden. Insofern besteht der erste Erweiterungsansatz nicht in einer funktionale Erweiterung, sondern in der Anregung, eine solche Überprüfung von kritischen Eigenschaften zu realisieren. Dadurch kann mitunter die Existenz schwer zu entdeckender Fehler wie beispielsweise beim Rücksetzen von Signalen zwischen parallelen Komponenten aufdeckt werden.

Neben einer aktiven Verhütung von Kollisionen mit dynamischen Hindernissen empfiehlt es sich, einen kontinuierlichen Sanity-Check für die Position zu implementieren. Aktuell wird ein kollisionsfreier Pfad geplant. Durch Treiben während eines Messvorgangs (sofern eine solche Funktionalität implementiert wird), starke Strömung oder Fahren mit der RC-Fernsteuerung kann das Boot jedoch mit Hindernissen kollidieren oder in eine Position geraten, von der aus der Weg zum nächsten Wegpunkt nicht mehr kollisionsfrei ist. Als Beispiel mag ein V-förmiges Gewässer dienen. Das Boot soll vom Fuß des V in den rechten Flügel fahren. Wird es mit der Fernsteuerung, deren Benutzung für die Software völlig transparent ist, in den linken Flügel gefahren, so wird nach Abschalten der Fernsteuerung versucht, auf geradem Wege zum nächsten Wegpunkt im rechten Flügel zu fahren. Dieser führt dann jedoch durch das keilförmige Hindernis. Ein kontinuierlicher Sanity-Check, der fortwährend überprüft, ob sich das Boot in der Nähe eines Hindernisses befindet, könnte eine solche Gefahrensituation erkennen und beispielsweise eine Neuplanung auslösen. Hierfür ist jedoch vermutlich eine Laufzeitoptimierung einiger Funktionen aus `GeoCalc` nötig, um diese zeitkritische Prüfung effizient durchführen zu können.

10. Testfahrten

Im Rahmen dieser Projektgruppe wurden insgesamt fünf Tests am Tweelbäker See durchgeführt. Nach dem Implementieren neuer Funktionen und dem erfolgreichem Test im OFFIS oder auf dem Parkplatz (Trockentest), kann ein neuer Seetest geplant werden. Im Folgenden sind die wichtigsten Planungsaspekte für einen Seetest aufgeführt.

Wichtige Planungspunkte:

- Zeitpunkt und Ort
- Team
- Anhänger reservieren (AZO)
- Batterie frühzeitig aufladen

Wichtiges Equipment:

- Notebook mit aktueller Software (+ Ersatz-Notebook)
- Fernbedienung inkl. Batterien
- Kabeltrommel
- Tisch und Stühle
- Monitor inkl. Kabel
- Tastatur
- 3-fach Steckdose
- Multimeter
- Werkzeug

Da es für uns die Möglichkeit gibt, das Clubhaus des Oldenburger Yacht-Clubs e.V. am Tweelbäker See in Oldenburg zu benutzen, werden dort alle Tests der Projektgruppe durchgeführt. Für jeden Seetest wird in der Projektgruppenbesprechung ein Planungsprotokoll erstellt, in welchem die Ziele beschrieben sind.

Bei den Seetests werden neue Funktionen unter realen Bedingungen getestet. Da die Seetests einen hohen Planungs und Arbeitsaufwand haben, ist es wichtig, dass aus jedem Bereich (Onshore-Software, Onboard-Software, Hardware, Werft) mindestens eine Person anwesend ist. So können kleinere Fehler schnell behoben und viel Zeit gespart werden.

Bei den Seetests, können wir kleine Fehler in der Software sowie Hardware feststellen und beheben. So erhöht sich die Betriebssicherheit, Stabilität und Zuverlässigkeit des MOPS III mit jedem Seetest, so dass das Hauptziel der Projektgruppe, eine stabile und erweiterbare Plattform zu erstellen, erfüllt ist und auch bei Langzeittests keine Komplikationen auftreten.

Abbildung 10.1.: Matze bei der Arbeit

11. Schluss

11.1. Zusammenfassung

Die Architektur der Software auf dem MOPS wurde auf ein komponentenbasiertes Modell umgestellt. Um Erweiterbarkeit und Leserlichkeit des Quelltextes zu erhöhen teilen sich alle Komponenten den gleichen Aufbau und Lebenskreislauf. Um die Gefahr von Deadlocks zu eleminieren wurde das Konzept des Shared Memory implementiert. Durch die Abarbeitung von nichtblockierenden Komponenten nacheinander in einem Thread kann eine grobe Abschätzung der Auslastung der CPU gegeben werden.

Die vorherige Projektgruppe hatte neben dem Raspberry Pi ein Beaglebone für die Onboard-Software verwendet und hatte damit viele Stabilitätsprobleme. Daher wurde die Architektur vollständig neu entworfen und auf einen Raspberry Pi 2 mit Arduino Unos für die Motor- und Sensorenansteuerung vereinfacht. Die ehemalige LAN-Verbindung an Bord zwischen BeagleBone und Raspberry fällt dadurch weg. Die Ansteuerung der Hardwarekomponenten, wie zum Beispiel GPS und Kompass, erfolgt über USB. Weitere Geräte sind einfach per Plug-&-Play hinzuzufügen.

Mit Hilfe der Fernbedienung ist es nun möglich, den MOPS unabhängig vom Status der Software zu steuern und gegebenenfalls aus Gefahrenzonen zu manövrieren. Außerdem wurde mit dem WatchDog des Motor Arduinos sichergestellt, dass das System bei einem Absturz automatisch neu gestartet wird.

Die Kommunikation erfolgt im Gegensatz zur vorherigen Projektgruppe ausschließlich über XBee. Zudem wurde das XBee eigene Protokoll durch ein selbstentwickeltes schlankes, schnelles und sicheres Protokoll ersetzt. Der Transfer von großen Datenmengen ist ebenso über XBee möglich. Der zugrundeliegende Quelltext ist ereignisgesteuert, thread-safe und leicht erweiterbar. Ein Verbindungsabbruch durch umschalten von WLAN Verbindung zu XBee Verbindung kann nicht mehr auftreten.

Für die Planungs- und Steuerungssoftware wurde ein neues modulares und hierarchisches System realisiert, welches den gesamten Prozess des Abarbeitens einer Mission umfasst. Alle Bestandteile der Software wurden entsprechend der neuen Software-Architektur und der verbesserten Modularisierung von Missionskontrolle und Pfadregelung neu implementiert. Einige der Konzepte wurden aus MOPS II übernommen, wesentliche Aspekte jedoch verbessert. Mit der onboard-seitigen Bahnplanung wurde zudem eine völlig neue Funktionalität eingeführt. Dabei hat sich die neue Planungs- und Steuerungssoftware als zuverlässig und gut wartbar gezeigt. Zudem bietet sie gute Ansätze für die Überprüfung von Eigenschaften mittels automatenbasiertem Modelchecking.

11.2. Fazit

Die gelungenen Testfahrten des MOPS III belegen den Erfolg der Projektgruppe. Die Anforderungen der Sicherheit, Stabilität und Zuverlässigkeit sind erfüllt und die Projekt-

gruppenarbeit von MOPS II wurde sinnvoll erweitert. Sämtliche Hard- und Softwarelösungen laufen stabil und der MOPS verhält sich zu jeder Zeit vorhersagbar. Ein schnelles Eingreifen, bei etwaigen Gefahrensituationen ist zu jedem Zeitpunkt möglich. Durch intuitive Nutzeroberflächen können sogar Laien die Software bedienen.

11.3. Ausblick

Trotz des zufriedenstellenden Ergebnisses des Projekts konnten einige enthusiastische Anforderungen nicht vollständig verwirklicht werden. Aufgrund von Zeit- und Kostengründen konnte keine Echtzeitbehandlung von Hindernissen stattfinden. Auch ein geplanter Kran oder eine Winde am Boot zum Absenken der Sensoren kann erst in Zukunft realisiert werden. Die Hochseetauglichkeit kann weiterhin durch die Installation einer Plane oder Ähnlichem in MOPS IV umgesetzt werden. Alternative Stromquellen, wie Solarzellen, sind bereits im Gespräch und können, sobald sich Sponsoren gefunden haben, die vorhandene Autobatterie unterstützen. Allgemein kann das Boot weiterhin durch Komponenten erweitert werden, um strenge nichtfunktionale Anforderungen an Umweltschutz und Energieeffizienz zu realisieren.

Für die einfache Erweiterbarkeit wurde stets ein Plug-&-Play Konzept eingesetzt. So können neue Software-Komponenten und Hardware-Geräte aber auch andere Sensoren einfach in die MOPS-Plattform integriert werden. Die Simulation kann beispielsweise durch eine umfangreichere externe Software-Komponente ausgetauscht werden, falls ein realistischeres Modell benötigt wird. An den an Bord befindlichen USB-HUB können zum Beispiel genauere GPS-Empfänger oder Distanzmessgeräte und Kameras für Kollisionserkennung angeschlossen werden. Softwareseitig wird auf ähnliche Weise eine Komponente eingehängt, die beispielsweise das Kamerabild ausliest und analysiert.

Die MOPS-Plattform ist so organisiert, dass sie einfach auf ein anderes Boot oder Schiff portiert werden kann. Falls andere Motoren oder sogar ein Ruder oder Pod-Antriebe eingesetzt werden sollen, können die Bahnregelungs- und Motorregelungskomponenten dementsprechend angepasst oder ausgetauscht werden.

Die dritte Iteration der Projektgruppe MOPS stellt eine solide und zuverlässige Platform bereit, auf der die folgende Projektgruppe MOPS IV erweiterte Funktionalitäten und Sicherheitskonzepte realisieren kann.

A. Betriebsbereitschaft herstellen

Hier soll kurz erklärt werden, wie man den MOPS in den betriebsbereiten Zustand versetzt, d.h. welche Einstellungen vorgenommen und welche Verbindungen gesteckt werden müssen. Dabei wird dem Verlauf der Spannungsversorgung entsprechend vorgegangen, d.h. von der Leistungselektronik (12V), über die Arduino-Box (12V $\rightarrow$ 5V), zum Raspberry Pi (5V). Danach wird die Systemelektronik angeschaltet, der MOPS eventuell kalibriert und zu Wasser gelassen.

Für den (unwahrscheinlichen) Fall das die Alu-Rahmenkonstruktion des MOPS noch zusammengebaut werden muss, sollte sich hier an der Anleitung in der Dokumentation der Projektgruppe MOPS II gehalten werden (Kapitel 7.4 „MOPS Aufbau", Seite 84, hier).

Außerdem kann die Belegung der Stecker/Kabel im Notfall in Kapitel B nachgesehen werden.

In der folgenden Anleitung wird davon ausgegangen, das sich bereits alle Komponenten (bspw. die Motoren) im bzw. am MOPS befinden.

A.1. Leistungselektronik-Box

1. Zunächst ist zu prüfen, ob die Switches der Box deaktiviert sind, dazu muss sich der Main-Switch in OFF-Position befinden und die Kappe des Kill-Switch muss abgezogen sein [vgl. Bild links oben].
2. Danach muss geprüft werden, ob die Stecker der Platine mit den entsprechenden Buchsen verbunden sind, dazu wurden Stecker und Buchse jeweils mit einem Etikett versehen. Außerdem sollte ein Relais in dem entsprechenden Sockel stecken [vgl. Bild links unten].
3. Nun ist zu prüfen, ob der Sabertooth ordnungsgemäß angeschlossen ist und die DIP-Switches in der richtigen Position stehen (siehe *Steckverbindung Sabertooth*). Außerdem sollte der Sabertooth im Zweifelsfall mit der notwendigen Konfiguration versehen werden (siehe *Konfiguration Sabertooth*).
4. Weiterhin sollte geprüft werden, ob der Stecker für den NMEA-Bus ordnungsgemäß angebracht ist. Nach befolgen der bisherigen Schritte sollte es in der Box nun wie im rechten Bildausschnitt aussehen.
5. Nun kann auch der Motorstecker geschlossen werden.

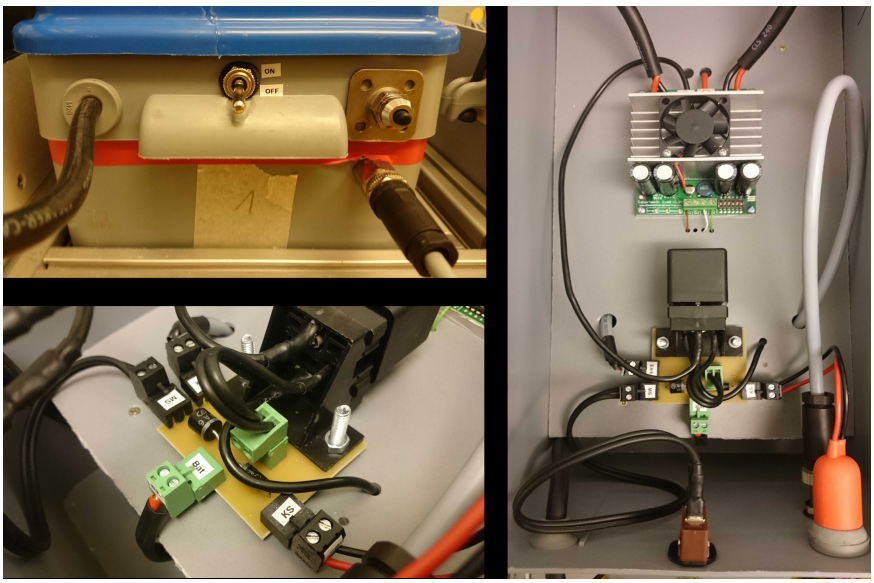

Abbildung A.1.: Leistungselektronik

Abbildung A.2.: Arduino-Box

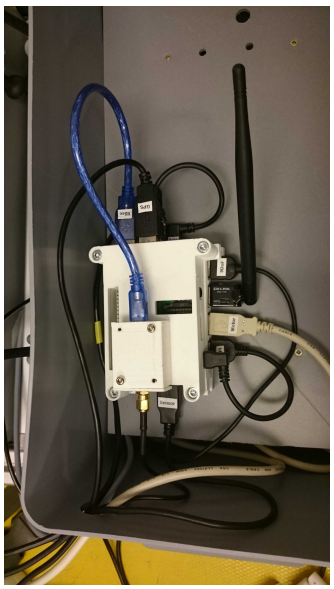

Abbildung A.3.: Raspberry-Box

A.2. Arduino-Box

1. Es ist zunächst zu prüfen, ob der NMEA-Bus ordnungsgemäß verbunden ist.
2. Weiterhin sollten beim Sensor Arduino die Spannungsversorgung (*BAT*) und der USB-Stecker für den Raspberry Pi gesteckt sein. Der Arduino sollte im Zweifelsfall mit der Software unter *„MOPS3-SVN"/trunk/Software/ArduinoUno/ Arduino_Payload/ Arduino_Payload.ino* bespielt werden.
3. Beim Motor Arduino sind die Stecker für die 12V-Spannungsversorgung (*BAT*), die 5V-Spannungsversorgung für den HUB/RPi (*HUB*), die Debug-LEDs (*LEDs*), die Motor-Steuersignale (*Motor*), die Steuersignale der RC-Fernbedienung (*RC-Receiver*) sowie der USB-Stecker für den Raspberry Pi zu prüfen. Der Arduino sollte im Zweifelsfall mit der Software unter *„MOPS3-SVN"/trunk/Software/ ArduinoUno/ Arduino_Motor/Arduino_Motor.ino* bespielt werden.
4. Außerdem sollte geprüft werden, ob der RC-Receiver ordnungsgemäß verbunden ist.

A.3. Raspberry-Box

1. Es ist zunächst zu prüfen, ob die Spannungsversorgung (*BAT*) im Power-Port des HUBs steckt.
2. Als nächstes ist zu prüfen, ob beim Raspberry Pi am Power-Port eine USB-Verbindung zum HUB besteht. Außerdem sollte noch eine Verbindung von einem RPi USB-Port zum Master-Port des HUBs bestehen.
3. Nun ist zu prüfen, ob die folgenden USB-Stecker in Ports des HUBs oder RPi stecken:
   - *GPS*

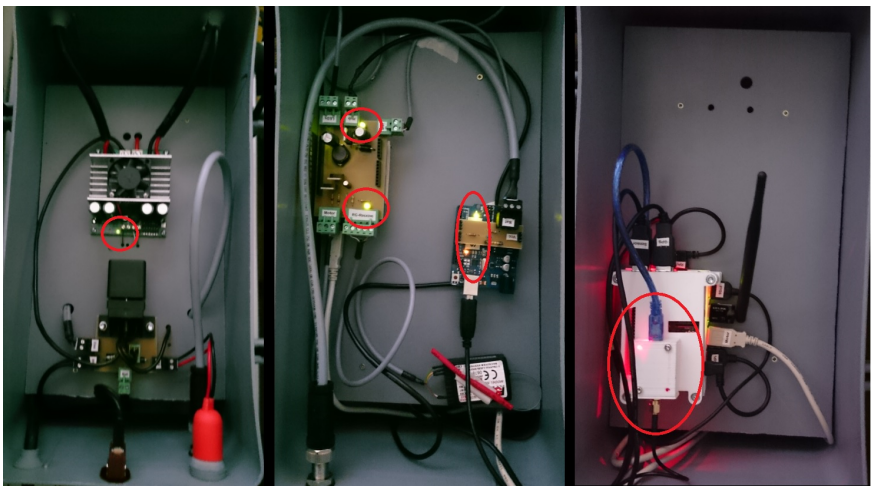

Abbildung A.4.: Kontrollleuchten der Hardware-Komponenten in den Otterboxen

- *XBee*
- *9DoF*
- *WLAN*
- *Motor*
- *Sensor*

4. Im Zweifelsfall sollte die Konfiguration des Raspberry durchgeführt werden, dazu wie in *Raspberry Pi Konfiguration* beschrieben vorgehen.

A.4. Anschalten

1. Wurden die bisherigen Schritte pflichtbewusst befolgt, kann nun ohne Bedenken die Batterie angeschlossen werden, sowie die Systemelektronik mit Hilfe des Main-Switch angeschaltet werden.
2. Nun sollten die Status-LEDs innerhalb der Boxen überprüft werden, diese sollten wie in nebenstehendem Bild (rote Kreise) aussehen. Außerdem sollten die Debug-LEDs zunächst Blau und nach ungefähr 2-3 Minuten Grün leuchten.

A.5. Kalibrierung

Die Kalibrierung ist persistent. Falls dennoch eine Neukalibrierung erwünscht ist:

1. MOPS Richtung Norden ausrichten und in Onshore-Software im Kalibrierungsdialog den `Nord`-Button drücken
2. MOPS Richtung Osten ausrichten und in Onshore-Software im Kalibrierungsdialog den `East`-Button drücken

3. MOPS Richtung Süden ausrichten und in Onshore-Software im Kalibrierungsdialog den `South`-Button drücken
4. MOPS Richtung Westen ausrichten und in Onshore-Software im Kalibrierungsdialog den `West`-Button drücken

A.6. Stapellauf

1. Für den Stapellauf ist zunächst die Stellung der Motorengriffe zu überprüfen, diese geben an wie viel Leistung den Motoren zur Verfügung steht, hierbei ist Position 5 (volle Leistung) zu wählen. Die Griffe müssen (mehr oder weniger gut) gegen verdrehen gesichert werden, in dem sie mit Hilfe eines Aluprofils an den beiden Manschetten verbunden werden.
2. Die Motoren müssen zum Transport mit dem Slipwagen zunächst hochgeklappt sein und sollten im Zweifelsfall auch erst im Wasser wieder abgesenkt werden. Je nach dem wie zu Wasser gelassen wird, könnte es sonst zu Beschädigungen an den Motoren kommen.
3. Beim zu Wasser ist man stark von den Gegenbenheiten vor Ort abhängig. Am einfachsten ist die Prozedur, wenn am Einsatzort eine Slipanlage vorhanden ist, dann kann der MOPS relativ einfach mit dem Slipwagen ins Wasser gefahren werden. Bei genügend Tiefe löst der MOPS sich aus dem Slipwagen und wäre somit für die weiteren Schritte bereit. Sollte keine Slipanlage vorhanden sein, kann der MOPS auch ohne weiteres ins Wasser getragen werden, hierbei ist es allerdings unbedingt darauf zu achten, dass es zu keinen Beschädigungen kommt.
4. Befindet der MOPS sich nun im Wasser, müssen zunächst die Motoren in ihre Einsatzposition gebracht werden. Außerdem muss jetzt die Motorregelung noch mit Hilfe des Kill-Switch aktiviert werden, dazu einfach die Kappe aufstecken.
5. Damit der MOPS nun auch manövrierfähig wird, **muss** die RC-Fernbedienung nun zumindest einmal **AN und wieder AUS** geschaltet werden. Es bietet sich jedoch an, den MOPS zunächst ferngesteuert ein Stück von der Landungszone zu entfernen, um mögliche Gefahren zu vermeiden.

A.7. Einstellung Onshore-Software

Die folgenden Werte geben einen Anhaltspunkt für eine brauchbare Software-Konfiguration des MOPS. Dabei wird zwischen dem realen Test und einem Simulationslauf unterschieden. Es ist zu berücksichtigen, dass je nach Witterung (Wind, Strömung), Einsatzort (Tweelbäker See, Wesermündung, Jadebusen, ...) und Rüstzustand (Tiefgang, Eintauchtiefe der Nutzlast-Sensoren bzw. Anbringung der Sensoren vorne oder hinten) wenigstens die Parameter der Bahnregelung des MOPS anzupassen sind.

A.7.1. MOPS Konfiguration

Parameter	Zweck	Wert
Engine Controller Type	Art der Umrechnung von Speed / Rudder in Motorkonfigurationen	Both
Path Controller Type - Travelling	Art der Bahnregelung während der Fahrt zum Ziel	COG + XTE
Path Controller Type - Dynamic Positioning	Art der Bahnregelung während des Haltens einer Position	Heading Only
Waypoints: Range	Radius um das Ziel, in dem das Ziel während der Fahrt zum Ziel als erreicht gilt (in Metern)	5.0
Waypoints: Extended Range	Radius um das Ziel, in dem die Geschwindigkeitsregelung aktiv ist (in Metern)	10.0
Waypoints: Hold Range	Radium um die Position, in dem die Position während des Haltens der Position als erreicht gilt (in Metern)	5.0
Rudder Controller Travelling: P	P-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	0.7
Rudder Controller Travelling: I	I-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	0.0
Rudder Controller Travelling: D	D-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	0.0
Rudder Controller Travelling: I-MAX	Windup control des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	200.0
Rudder Controller Dynamic Positioning: P	P-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	0.7
Rudder Controller Dynamic Positioning: I	I-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	0.0
Rudder Controller Dynamic Positioning: D	D-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	0.0
Rudder Controller Dynamic Positioning: I-MAX	Windup control des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	200.0
Speed Controller Travelling: P	P-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	0.8

Speed Controller Travelling: I	I-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	0.2
Speed Controller Travelling: D	D-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	0.1
Speed Controller Travelling: I-MAX	Windup control des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	500.0
Speed Controller Dynamic Positioning: P	P-Wert des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	0.8
Speed Controller Dynamic Positioning: I	I-Wert des PID-Reglers für die Geschwindigkeitsregler während des Haltens einer Position	0.2
Speed Controller Dynamic Positioning: D	D-Wert des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	0.1
Speed Controller Dynamic Positioning: I-MAX	Windup control des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	500.0
XTE Controller: P	P-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.3
XTE Controller: I	I-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.0
XTE Controller: D	D-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.0
XTE-Controller: I-MAX	Windup control des PID-Reglers für die Regelung des Cross Track Errors	500.0
Miscellaneous: Rudder Ratio	Verstärkungsfaktor für Ruder-Wert	2.0
Miscellaneous: Max Motor %	Faktor zur Begrenzung Motordrehzahl (in Prozent), aktuell ungenutzt	1.0
Miscellaneous: Standard Travel Speed	Standardgeschwindigkeit bei voller Fahrt (Drehzahl in Prozent)	90.0
Miscellaneous: COG History Length	Anzahl der gespeicherten Positionen über die der Kurs über Grund berechnet wird (Vermeidung von „Ausreißern” im GPS-Signal)	1.0

Tabelle A.1.: Konfigurationswerte für den Tweelbäkersee

A.7.2. Simulationswerte

Die folgenden Werte sind eine Beispielkonfiguration, die für Simulationsläufe genutzt werden kann:

Parameter	Zweck	Wert
Engine Controller Type	Art der Umrechnung von Speed / Rudder in Motorkonfigurationen	Both
Path Controller Type - Travelling	Art der Bahnregelung während der Fahrt zum Ziel	Heading + XTE
Path Controller Type - Dynamic Positioning	Art der Bahnregelung während des Haltens einer Position	Heading Only
Waypoints: Range	Radius um das Ziel, in dem das Ziel während der Fahrt zum Ziel als erreicht gilt (in Metern)	5.0
Waypoints: Extended Range	Radius um das Ziel, in dem die Geschwindigkeitsregelung aktiv ist (in Metern)	15.0
Waypoints: Hold Range	Radium um die Position, in dem die Position während des Haltens der Position als erreicht gilt (in Metern)	5.0
Rudder Controller Travelling: P	P-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	1.0
Rudder Controller Travelling: I	I-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	0.0
Rudder Controller Travelling: D	D-Wert des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	0.0
Rudder Controller Travelling: I-MAX	Windup control des PID-Reglers für die Ruder-Regelung während der Fahrt zum Ziel	200.0
Rudder Controller Dynamic Positioning: P	P-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	1.0
Rudder Controller Dynamic Positioning: I	I-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	0.0
Rudder Controller Dynamic Positioning: D	D-Wert des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	0.0
Rudder Controller Dynamic Positioning: I-MAX	Windup control des PID-Reglers für die Ruder-Regelung während des Haltens einer Position	200.0
Speed Controller Travelling: P	P-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	2.0

Speed Controller Travelling: I	I-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	0.3
Speed Controller Travelling: D	D-Wert des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	0.0
Speed Controller Travelling: I-MAX	Windup control des PID-Reglers für die Geschwindigkeitsregelung während der Fahrt zum Ziel	500.0
Speed Controller Dynamic Positioning: P	P-Wert des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	0.9
Speed Controller Dynamic Positioning: I	I-Wert des PID-Reglers für die Geschwindigkeitsregler während des Haltens einer Position	0.2
Speed Controller Dynamic Positioning: D	D-Wert des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	0.1
Speed Controller Dynamic Positioning: I-MAX	Windup control des PID-Reglers für die Geschwindigkeitsregelung während des Haltens einer Position	500.0
XTE Controller: P	P-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.3
XTE Controller: I	I-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.0
XTE Controller: D	D-Wert des PID-Reglers für die Regelung des Cross Track Errors	0.0
XTE-Controller: I-MAX	Windup control des PID-Reglers für die Regelung des Cross Track Errors	200.0
Miscellaneous: Rudder Ratio	Verstärkungsfaktor für Ruder-Wert	2.0
Miscellaneous: Max Motor %	Faktor zur Begrenzung Motordrehzahl (in Prozent), aktuell ungenutzt	1.0
Miscellaneous: Standard Travel Speed	Standardgeschwindigkeit bei voller Fahrt (Drehzahl in Prozent)	90.0
Miscellaneous: COG History Length	Anzahl der gespeicherten Positionen über die der Kurs über Grund berechnet wird (Vermeidung von „Ausreißern” im GPS-Signal)	1.0

Tabelle A.2.: Konfigurationswerte für die Simulation

Bitte beachten:

- Werden Reglerkombinationen mit XTE ausgewählt, werden zwei separate Steuer-

größen für das Ruder berechnet und anschließend addiert. Dadurch werden die Steuergrößen potentiell schneller groß als bei Kombinationen ohne XTE. Dies ist bei der PID-Regler-Konfiguration zu berücksichtigen.

- Bei der Variante „Both" der Komponente „Engine controller" ist die Drehzahldifferenz der Motoren durch Ruderbewegungen prinzipiell doppelt so groß wie in der Variante „Separate", da die Steuergröße „Rudder" auf jeden Motor angewendet wird. In der Variante „Separate" wird sie zunächst nur auf den kurvenäußeren Motor angewendet. Wird dabei das zulässige Intervall für den Motor über- oder unterschritten, wird der Rest auf den kurveninneren Motor angewendet. Beträgt „Speed" also 80 und „Rudder" 30, so wird für den kurvenäußeren Motor der Steuerwert 100 und für den kurveninneren Motor der Wert 70 berechnet. Details können dem entsprechenden Abschnitt im Kapitel „Planungs- und Steuerungssoftware" des Abschlussberichts entnommen werden.
- In der Variante „Both" sind Drehzahldifferenzen bei hohen Geschwindigkeiten initial kleiner als bei niedrigen Geschwindigkeiten. Dies ist dadurch bedingt, dass die Steuergrößen auf das Intervall $[-100, 100]$ beschränkt werden. Reicht die Drehzahldifferenz nicht aus, so wird nachgeregelt. Mit einer „Rudder Ratio" von mindestens 2.0 ist das Drehen auf der Stelle auch bei einem „Speed"-Wert von 100 möglich.
- Mangels mathematischem Modell der Regelstrecke sind die Reglerparameter in empirischen bzw. Faustformelverfahren zu bestimmen. Die vorliegenden Parameter wurden bestimmt, indem zunächst ausschließlich P-Anteile verwendet wurden. Diese wurden soweit erhöht, bis ein instabiles Verhalten kurz bevor stand. Danach wurde mit den weiteren Parametern systematisch nachkonfiguriert, bis das Fahrverhalten sich dem gewünschten Verhalten annäherte. Unter `https://de.wikipedia.org/wiki/Faustformelverfahren_%28Automatisierungstechnik%29`findet sich ein Einblick in Faustformelverfahren.

A.8. Missionsplanung

Nach Abarbeitung der oberen Punkte ist der MOPS nun Betriebsbereit und in der Onshore-Software kann eine Mission geplant werden.

B. Belegung der Steckverbindungen

Hier sollen die Belegungen der Steckverbindungen im MOPS beschrieben werden, um eventuelle spätere Reparaturen oder Veränderungen zu erleichtern. Dazu werden neben den direkten Pin-Belegungen der Stecker, möglichst auch die Aderfarben der jeweiligen Kabel beschrieben.

Die Belegung der Stecker ist generell von links nach rechts aufsteigend nummeriert.

B.1. Steckverbindung Batterie

Das Kabel wird Batterie-Seitig von zwei Klemmen abgeschlossen, in der jeweiligen Farbe der Ader. Hierüber wird die gesamte Spannungsversorgung realisiert.

2-polig, 12V-Versorungsspannung

Ader	Funktion
Rot	Pluspol
Schwarz	GND

Tabelle B.1.: Steckverbindung Batterie

B.2. Steckverbindung Motoren - CE-Stecker/-Kupplung

Ausgelegt sind Stecker und Kupplung für 3-phasen Wechselspannung mit einem maximalen Strom pro Ader von 63A. Außerdem ist die geschlossene Steckverbindung mit der Schutzart IP67 versehen, womit sie Staubdicht und gegen zeitweiliges Untertauchen geschützt ist. Die Belegung der Steckerpins ist in nebenstehendem Bild dargestellt. Dabei ist der Pluskontakt jeweils mit einer roten Ader, der Minuskontakt mit einer schwarzen Ader verbunden.

B.3. Steckverbindungen Sabertooth

Der Sabertooth erledigt die Motorregelung, dementsprechend sind hier neben der Spannungsversorgung auch die Motoren und die Steuersignale des Motor Arduino angeschlossen.

Steckerpin	Ader	Funktion
1	Braun	GND
2	Weiß	S1 Steuersignal

3	Grün	S2 Steuersignal

Tabelle B.2.: Aderbelegung Sabertooth

B.4. Steckverbindungen der Leistungselektronikplatine

Im Folgenden werden kurz die Belegungen der Steckverbindungen der Leistungselektronikplatine dargestellt.

B.4.1. Versorgung

Mit dieser Steckverbindung wird die 12V-Versorgung der Platine besorgt. Über die Platine werden sowohl die Systemelektronik sowie der Schaltstrang des Relais versorgt.

2-polig, 12V-Versorgungsspannung

Steckerpin	**Ader**	**Funktion**
1	Rot	Pluspol
2	Schwarz	GND

Tabelle B.3.: Aderbelegung Stromversorgung

B.4.2. Main- und Kill-Switch

Bei Main- und Kill-Switch ist die Belegung der Steckerverbindung uninteressant, da die Funktion der Schalter auch bei „Verpolung" nicht verändert wird.

B.4.3. Elektronik

Über diese Steckverbindung wird die Versorgungsspannung an den NMEA-Stecker weitergeleitet. Dieser Zweig ist über den Main-Switch schaltbar und mit einer Diode als Verpolschutz versehen.

2-polig, 12V-Versorgung

Steckerpin	**Ader**	**Funktion**
1	Schwarz	GND
2	Rot	Pluspol

Tabelle B.4.: Aderbelegung Elektronik

Abbildung B.1.: Batterieklemmen

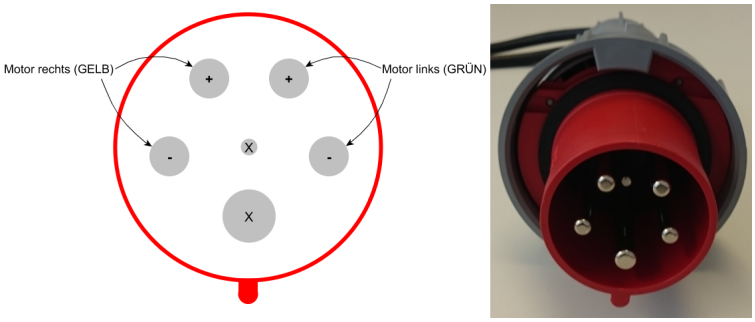

Abbildung B.2.: CE-Stecker Motor

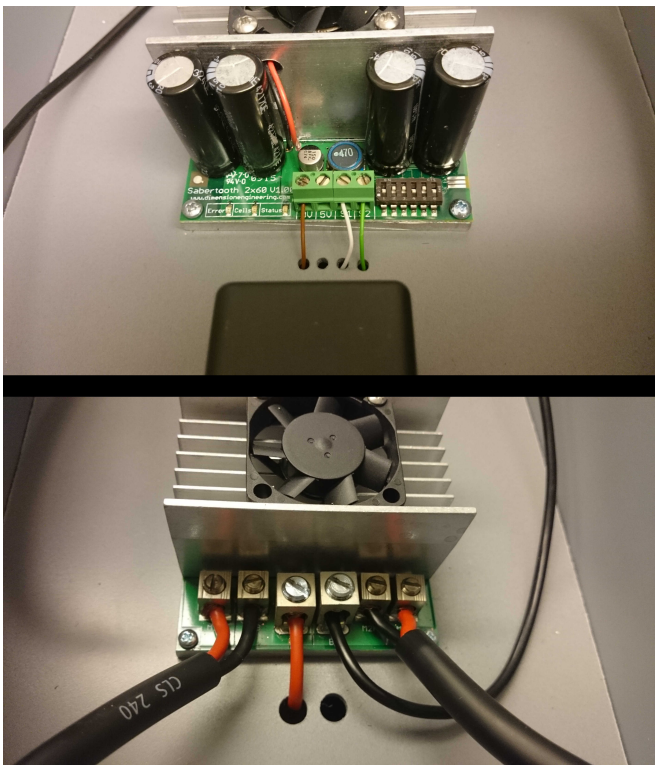

Abbildung B.3.: Steckverbindung Sabertooth

Abbildung B.4.: 2-Pol Versorgung

Abbildung B.5.: 2-Pol Elektronik

Abbildung B.6.: 2-Pol Relais

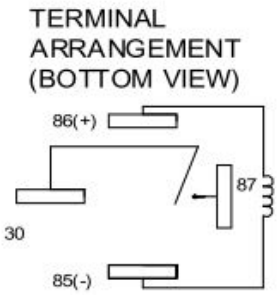

Abbildung B.7.: Relaisbelegung

B.4.4. Relais-Spule

Bei der Belegung der Steckverbindung ist darauf zu achten, dass die Spule in der richtigen Richtung durchflossen wird, sodass das Relais auch schalten kann. Dazu folgende Belegung beachten:

Steckerpin	**Funktion**
1	GND
2	Pluspol

Tabelle B.5.: Relais-Spule

B.4.5. Steckverbindungen des Relais

Beim Relaissockel sind momentan die Verbindungen zur Relaisspule (linker und rechter Slot) sowie der geschaltete Zweig des Relais (oberer und unterer Slot) mit der Platine bzw. dem System verbunden. Das Relais kann in den Sockel eingesteckt werden. Dabei ist folgenden Belegung zu beachten:

Slot	**Pinausrichtung**	**Pin#**	**Funktion**
oben	senkrecht	30	Sabertooth(+)
links	senkrecht	85	Schaltspule(-)
rechts	senkrecht	86	Schaltspule(+)
unten	waagerecht	87	12V-Versorgung

Tabelle B.6.: Steckverbindungen des Relais

Abbildung B.8.: NMEA-Bus

B.5. Steckverbindung NMEA-Bus

Der NMEA-Bus wird verwendet um die 12V-Versorgungsspannung für die Sensoren sowie die anderen Otterboxen bereitzustellen. Außerdem soll es möglich sein hiermit sowohl SDI-12 als auch RS485 zur Datenübertragung realisiert werden. Derzeit wird nur die SDI-12 Variante verwendet. Die Nummerierung der Pins ist hier der Steckerbeschreibung entnommen. Folgende Belegung ist dabei zu beachten.

5-polig, 12V-Versorgung & Datenübertragung

Steckerpin	**Ader**	**Funktion**
1	Shield	
2	Rot	Pluspol
3	Schwarz	GND
4	Weiß	RS485
5	Blau	SDI-12/RS485

Tabelle B.7.: Steckverbindung NMEA-Bus

Abbildung B.9.: Steckverbindung der Sensor-Arduinoplatine

B.6. Steckverbindung der Sensor-Arduinoplatine

Der Stecker ist an den NMEA-Bus angeschlossen und dient zur Versorgung sowie der Datenübertragung per SDI-12. Außerdem wird hier die Parallelschaltung der Versorgung für den Motor Arduino realisiert. Dazu sind momentan folgende Adern mit dem Stecker verbunden:

3-polig, 12V-Versorgung

Steckerpin	**Funktion**	**Ader (NMEA)**	**Ader (Motor Arduino)**
1	Pluspol	Rot	Braun
2	GND	Schwarz	Weiß
3	SDI-12	Blau	

Tabelle B.8.: Steckverbindung der Sensor-Arduinoplatine

Abbildung B.10.: Versorgung Motor-Arduinoplatine

B.7. Steckverbindungen der Motor-Arduinoplatine

Im Folgenden werden kurz die Belegungen der Steckverbindungen der Motor-Arduinoplatine dargestellt.

B.7.1. Versorgung (Bat)

Der Stecker ist an den Versorgungsstecker des Sensor Arduino angeschlossen, allerdings ohne Datenleitung.

2-polig, 12V-Versorgungsspannung

Steckerpin	**Ader**	**Funktion**
1	Weiß	GND
2	Braun	Pluspol

Tabelle B.9.: Steckverbindungen der Motor-Arduinoplatine

B.7.2. HUB/RPi

Versorgung des HUBs und Rapsberry Pi mit 5V, das Kabel verbindet einen 2-poligen Stecker vom Arduino mit einem microUSB-Stecker für HUB/RPi.

2-polig, 5V-Versorgungsspannung

Steckerpin	**Ader**	**Funktion**
1	Schwarz	GND
2	Rot	Pluspol

Tabelle B.10.: HUB/RPi

Abbildung B.11.: Stecker für Debug-LED & Sabertooth

B.7.3. Debug-LED & Sabertooth

Da die Debug-LEDs und der Sabertooth den gleichen Stecker besitzen und die Belegung der Adern auch gleich ist, werden die Verbindungen hier in einem Kapitel beschrieben. Dabei wird bei den LEDs die 5V Versorgung sowie eine Datenleitung gebraucht und für die Steuersignale des Sabertooth eine GND-Leitung sowie 2 PWM-Signalleitungen. Folgende Belegung ist einzuhalten.

3-polig, 5V-Versorgung

Steckerpin	**Ader**	**Funktion (LEDs)**	**Funktion (Sabertooth)**
1	Braun	GND	GND
2	Weiß	Pluspol	PWM-Signal S1
3	Grün	Datenleitung	PWM-Signal S2

Tabelle B.11.: Debug-LED & Sabertooth

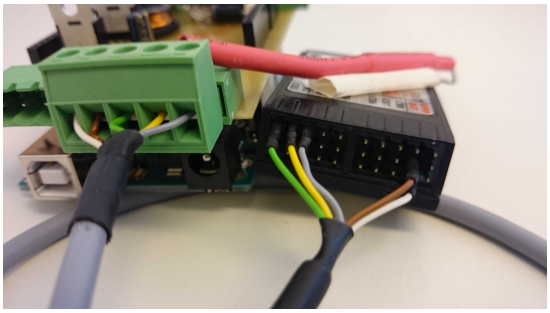

Abbildung B.12.: RC-Receiver

B.7.4. RC-Receiver

Mit dieser Steckverbindung wird sowohl der RC-Receiver mit 5V versorgt als auch die PWM-Signale des Receivers zur Steuerung des Motors mit der Fernbedienung übertragen.

5-polig, 5V-Versorgungsspannung

Steckerpin	Ader	Funktion
1	Weiß	GND
2	Braun	Pluspol
3	Grün	PWM-Signal MOSFETs
4	Gelb	PWM-Signal Motor links
5	Grau	PWM-Signal Motor rechts

Tabelle B.12.: Aderbelegung RC-Receiver

B.8. Steckverbindungen am HUB/Raspberry Pi

Der HUB wird von der Spannungsregelung auf dem Motor Arduino versorgt, dieser speist den Raspberry Pi über den Power-microUSB-Port. Außerdem ist der Raspberry Pi per weiterer USB-Verbindung mit dem Master-Port des HUBs verbunden, damit der HUB die Nachrichten an den RPi weiterleitet.

Die weiteren USB-Verbindungen können an beliebige Ports des HUBs angeschlossen werden, da die von den Geräten verwendeten Ports automatisch vom USB-Device-Manager erkannt werden.

C. Software-Struktur

Die gesamte Software des MOPS III Projekts ist in mehrere `Java Projects` aufgeteilt. Da zwei verschiedene Programme, die Onboard-Software und die Onshore-Software, implementiert wurden, wurden `Java Projects` angelegt, die von beiden Programmen benutzt werden können. Diese beiden Programme werden im Dokument vorgstellt, daher wird im Folgenden eine Übersicht der anderen `Java Projects` im einzelnen vorgestellt.

C.1. de.uniol.mops.model

Dieses `Java Project` bietet eine gemeinsame Basis für Modelle. Solche Modelle sind z.B. eine Mission, die von der Onshore-Software entwickelt wurde, zur Onboard-Software übertragen wurde und von der Bahnplanung bearbeitet wurde.

C.2. de.uniol.mops.util

Um kleine unabhängige Java Systeme überall zu verwenden wird dieses `Java Project` benutzt. Unter den wichtigen Systemen befindet sich das Paket `de.uniol.mops.util.io`. Dieses Paket beinhaltet Klassen um das speichern und laden und generelles umwandeln von Java Objekten in Bytes zu erleichtern. Zudem wird in einem Unterpaket auch die Anbindung von Hardwaregeräten über USB vereinfacht. Ein Ereignissystem, dass von der Kommunikation benutzt wird, findet sich in `de.uniol.mops.util.event`. Mathematische Klassen und Helferklassen, Checksummenhelfer, ein eigenes und schlankes Log System, sowie Methoden um automatisch das Betriebssystem zu erkennen finden sich jeweils in eigenen kleinen Paketen.

Literaturverzeichnis

[1] Chris Veness: *Calculate distance, bearing and more between Latitude/Longitude points.* `http://www.movable-type.co.uk/scripts/latlong.html`. Zuletzt abgerufen: 2015-10-11.

[2] Wikipedia: *Loxodrome.* `https://de.wikipedia.org/wiki/Loxodrome`. Zuletzt abgerufen: 2015-10-11.

[3] Wikipedia: *Orthodrome.* `https://de.wikipedia.org/wiki/Orthodrome`. Zuletzt abgerufen: 2015-10-11.

[4] Rolf Isernhagen, Hartmut Helmke, Frank Höppne: *Einführung in die Softwareentwicklung*, Hanser Fachbuchverlag, 2007.

[5] Dietrich Boles: *Leitfaden zur Durchführung von Projektgruppen*, Dietrich Boles, 2006.

[6] Björn Borgmann, et al.: *Abschlussdokumentation der Projektgruppe MOPS 2012/2013*, 2013.

[7] Hauke Evers, et al.: *Abschlussdokumentation im Rahmen des Projekts Marine Observation Platform for Surfaces 2*, 2014.

[8] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun: *ARA*: Anytime A* with provable bounds on sub-optimality.* In: Advances in Neural Information Processing Systems, 2003