Ausarbeitungen zum Seminar

Konzepte, Entwurf und Bau fehlertoleranter Systeme

Wintersemester 2015/2016

27. April 2016

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik
Abt. Systemsoftware und verteilte Systeme

www.uni-oldenburg.de/svs

Inhaltsverzeichnis

A Game-Based Abstraction of Fault Tolerance to Illustrate Computer Science Workfields

Lina Spiekermann
Department für Informatik
Carl von Ossietzky Universität
Oldenburg
Email: lina.spiekermann@uni-oldenburg.de

***Abstract*—In an increasingly digitalised world computer scientists are needed in many places which makes it crucial to inspire a high number of graduates to choose such a course of studies. Yet the actual working field of a computer scientist remains rather vague and abstract to young people, although many techniques studied are adapted from everyday life strategies. This paper presents a method to illustrate basic principles of common fault tolerance in a computer science scenario via a game-based approach to lower the hurdle of engagement. More precisely a SpaceInvaders adaptation is used that can successfully simulate simple server communication with different fault tolerance levels noticeable increasing the player's score.**

I. Introduction

In computer science many techniques base on everyday life solution strategies, that people turn to instinctively without consciously realising, making them predestined to illustrate that area of expertise for potential students. One of this concepts is **fault tolerance** around which this paper is centred.

It is known throughout computer science society since Dijkstra [4], that the absence of faults in a computer program cannot be proven. However with increasing complexity and size of those the task of determining whether a program reacts as it is supposed to in every constellation imaginable has become a problem that remains theoretically determinable but practically impossible to solve due to its high dimensionality. Therefore a different approach is taken: the abandonment of the tedious path of proving correct behaviour for the cost of faults in the system to which can be reacted accordingly. A website server e.g. Google.com, Wikipedia.org which is called by millions of people every day to return its startpage information can **tolerate faults**, like its response getting lost in the network, by resending it so the user won't notice. If this behaviour can be verified all scenarios that would have caused such an error, connection failure, routing problems etc., can be handled with this one test and the dimensionality of the problem has thus been reduced.

There are many fields other than computer science where fault tolerance can be applied and people are applying it to everyday problems without being aware of it when they are leaving reminder notes, keeping an extra key or the like. The goal of this paper is to present an application that makes fault tolerance presentable to recipients unrelated to academical computer science taking a game-based approach to increase learning effectiveness as stated by Clark et al. in [1]. The game

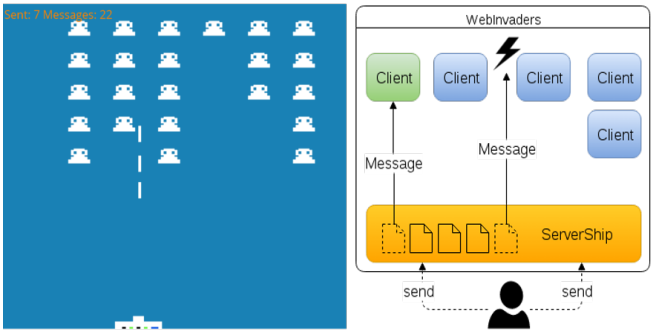


Fig. 1. Basic state: Gameplay (l) and participants (r)

used in this approach is **WebInvaders** an adaptation of the well known and easy to grasp SpaceInvaders arcade game with the objective for the ServerShip (bottom) to transmit messages to as many clients (top) as possible as can be seen in the screenshot of the gameplay in Figure 1. The player takes the role of a website server with increasing fault tolerance level resulting in higher score so the benefits of this concept can be learned. Playing the game future developers can be guided to include fault tolerance into their applications.

In the next Section II a few basic terms are outlined followed by the introduction of WebInvaders participants in Section III. After that Section IV forms the paper's main part where the implementation of the example system is described and its states are evaluated regarding their fault tolerance aspects with respect to the assessment model explained in Section IV-A. Finally the paper will be summed up in Section V.

II. Fault tolerance

This Section states some basic material concerning fault tolerance and naming conventions in particular.

A. Fault, Error and Failure

The terminology used when talking about fault tolerance often differs from publication to publication which is why it will be set at the beginning of this paper. The following set of definitions was chosen from this paper's course context.

The source of every misbehaviour is the **fault** or **defect** a system part that acts different than intended and may lead to an **error** an unintended state of the system. If such an error

prevents the system from working correctly, regardless if the system actually crashes or just produces wrong output, it has caused a **failure**: An observable misfunction of the system. An observed failure always originates in an error and an error in a fault respectively whereas not every fault causes an error and therefore may be part of any system.

B. Fault Tolerance and Redundancy

In order to make a system fault tolerant some sort of control mechanism or the like has to be included, which does not improve the intended function of the system and would have never been needed if the system would be faultless. These **redundant** parts that are only necessary in case of error/failure are the cost for the avoidance of complete and thorough verification and the assumed existence of faults.

C. Static and dynamic redundancy

There are two types of redundancy that will be focused on in this paper. They define according to Koren and Krishna [3, p.3] as follows:

Static: Redundancy will be present at any time so it can be used e.g. as a fallback when the system fails, like two computers calculating the same equations simultaneously.

Dynamic: Redundancy is only added when a failure occurs, e.g. reserve wheel in a car.

III. Participants

This Section will present the relation of the WebInvaders components shown in Figure 1 to the webserver communication participants.

ServerShip The ServerShip represents a webserver that can communicate via a simple protocol. It has a stock of messages with one for each client

Client A client represents a user who has sent a request to the webserver (blue). Naturally the user will grow more and more impatient the longer he doesn't receive a reply and therefore move faster towards the bottom.

Message A message represents the response sent by the server to a user. If a client receives a message his request is satisfied and he won't be displayed any longer (green). The message is removed from the stock after sending.

Visible Screen The background screen represents the communication channel from the server to the users. A client being visible on the screen represents him having sent a request that has not yet been answered.

Screen Border If a client reaches the bottom of the screen the game doesn't end as does the original SpaceInvaders, but the count of failures will be increased by one. The way from top to bottom of the screen represents the time a user is willing to wait for a response.

Status The current status, displayed top left on the game screen consists of two values:

Sent Number of successfully transmitted messages.

Messages Number of messages left in stock.

The game ends if the number of messages is zero.

Fig. 2. Start screen with state scores

Every client that hasn't received a message by the end of the game represents a user not getting a response from the webserver and therefore notices a failure in the system.

IV. WebInvaders

In this Section the in the context of this paper implemented WebInvaders application will be explained in more detail. WebInvaders represents a basic webserver communication scenario consisting of users sending **requests** to a webserver responding with **responses**.

It can take three states using different levels of fault tolerance each improving its predecessor in accordance with the assessment model outlined in Section IV-A:

Basic Optimistic message sending with neither fault tolerance nor error identification.

Static Static fault tolerance added by sending every message twice but no error identification either.

Dynamic Dynamic fault tolerance possible through error identification by resending a message if an acknowledgement timeout is reached.

Each of the following Sections will outline one of theses states in ascending order with a description of the general setup and scenario and an analysis of its performance according to the assessment model.

A. Assessment model

This Section shortly introduces the assessment model which is based on the following assumptions to ensure comparability:

- The number of clients during each execution of the scenario is expected to be identical.
- A client that receives a message throughout execution is a successful transfer.
- A client that doesn't, has noticed a system failure and counts as unsuccessful transfer.

In the upcoming analysis of each of the example system's states its **value** will be measured in the number of successful transfers and therefore satisfied clients that will possible access the website again and its **cost** will be measured in the number of sent messages to achieve this value representing an actual cost of energy and time necessary for the transmission. These two values are reflected in the score of WebInvaders placed beside the associated state at the start screen shown in Fig. 2.

The score increases with more successful transmissions and with a lower number of sent messages.

B. No Fault Tolerance

This Section describes the non fault tolerant communication scenario and the corresponding application contents.

1) Scenario: A simple website server scenario will form the current state's basis including the following assumptions to simplify the analysis:

- A user's request cannot get lost.
- A user does only send one request total.
- All user requests are identical. Therefore any response will be accepted by every user.
- The server sends only one response per user.

In this basic state the server receives no feedback as to whether its response was received by a user so every error increases the number of unsuccessful transfers as can be seen in Fig. 1 where three messages are left for four clients. The gameplay in this state is shown in Figure 1.

2) Analysis: Failure sources can be networking problems if a router on the way from server to user fails or the internet connection on either side disconnects as well as a long interval between request and response which will cause many users to abort the request. Such behaviour is represented by a message missing a client or a client reaching the bottom of the screen.

If the probability of one response successfully passing through the communication channel is p and the total number of requests and responses is n then the current state will be rated according to the assessment model as follows:

$$No\,fault\,tolerance:$$
$$value = \lfloor p * n \rfloor,$$
$$cost = n$$

Transfers can only be count in discrete units therefore Gaussian brackets were added to apply floor function.

This rating applies exactly to any website server scenario. During the execution of the WebInvaders representation however p is variable dependent on the decreasing number of clients present as they receive messages or leave the screen. The exact calculation of this p would be rather complex and not be performed in this paper as it depends not only on the number of clients left but also on their constellation e.g. 5 clients forming a row or a column will result in different probabilities for a hit.

Still since the initial client count and constellation remain identical for every WebInvaders execution the variable p will behave the same for every order of message reception in each state which means that the evaluation used on the webserver states can also be used to rate the corresponding WebInvaders states in relation to each other as shown in Figure 3.

C. Static Fault Tolerance

This subsection describes the realisation of the example system with fault tolerance increased.

Let $v_{state} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $v_{state}(p, n)$ calculate the value v of a simplified webserver state $state$ positively correlated to the probability p of successful transfer and n the number of sent responses. In the WebInvaders representation the value of p is dependent on the current constellation c of remaining invaders.
Let $c_n, \ldots, c_0$ be a series of client constellations with c_n being the start constellation and c_i the same as c_{i+1} minus one client and p_{c_i} the probability of successful transfer with currently formed constellation c_i, then the value of the WebInvader scenario passing through those constellations with state $state$ is:
$value = \sum_{i=0}^{n} v_{state}(p_{c_i}, 1)$.
The number of responses is set to one because every successfully sent response will change the constellation. Thus the rating order of states is preserved in WebInvaders:

$$v_{state1}(p, n) > v_{state2}(p, n)\ \forall p \in \mathbb{R},\ n \in \mathbb{N}$$
$$\Rightarrow \sum_{i=0}^{m} v_{state1}(p_{c_i}, 1) > \sum_{i=0}^{m} v_{state2}(p_{c_i}, 1)$$
$$\forall m \in \mathbb{N}, c_0, \ldots, c_m \text{ constellations}$$

Fig. 3. State relation webserver and WebInvaders

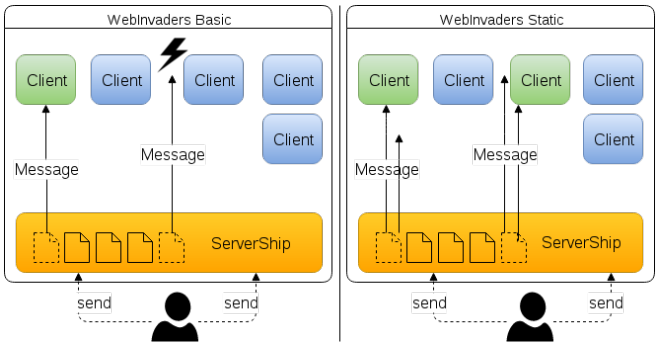


Fig. 4. No fault tolerance vs. static fault tolerance

1) Scenario: To decrease the number of unsuccessful transfers in this state, the server sends every response twice consecutively. As shown in Figure 4 if the first message got lost in the channel the second message might still reach the target client masking the previous fault from the client. This approach can therefore tolerate some lost messages for the price of redundancy when resending an already successfully transmitted message. In the WebInvaders representation the communication in static mode is realised by letting every touch of the screen cause the sending of two messages instead of one with the second message behind the first one.

2) Analysis: Error sources remain the same as in basic state leading to an evaluation of static state with value increased by $\lfloor p*(1-p)*n \rfloor$ with $(1-p)$ being the probability of a response getting lost in the channel and the doubled cost as in basic state due to the sending of twice as many messages:

$$Static\,fault\,tolerance:$$
$$value = \lfloor (p + p * (1 - p)) * n \rfloor,$$
$$cost = 2 * n$$

D. Dynamic Fault Tolerance

The dynamic state is an improved form of the previous static state with dynamic fault tolerance.

1) Scenario: To identify if a response has successfully been transmitted to a user the server awaits an **acknowledgement** from the user confirming the response reception as shown in Figure 5. If such an acknowledgement hasn't arrived after a certain amount of time, the **timeout**, the server assumes the response to be lost and can resend it once. It is assumed that an acknowledgement cannot get lost in the channel and is always received by the server.

Thus an asynchronous communication has been turned into a synchronous communication by adding dynamic redundancy.

In the WebInvaders representation of this state the response timeout is represented by the time a message needs to move from bottom to the top of the screen. If a message reaches the top of the screen without being received, the timeout is reached, the message is set inactive (grey in Figure 5) and can be resend. Furthermore a client that receives a message turns green for one second representing the sending of an acknowledgement before he gets invisible.

2) Analysis: In this state the probability of successful transfer remains the same as in static state but the cost is decreased by $n - \lceil (1 - p) \rceil * n$ because the response is only sent twice if no acknowledgement has been received which leads to the following analysis assuming a probable number of responses reaching the first timeout:

$$Dynamic\,fault\,tolerance:$$
$$value = \lfloor (p + p * (1 - p)) * n \rfloor,$$
$$cost = n + \lceil (1 - p) * n \rceil$$

The number of responses reaching timeout can be calculated with p and this value is rounded up to satisfy a worst-case scenario.

The cost would be the identical to static state if the value of p would equal zero. This case however will not be considered as it is unnecessary to compare communication methods on a closed channel basis. p is on the contrary more likely to represent a value greater than 0.5 letting dynamic state clearly exceed static state in the matter of cost savings.

E. Implementation

WebInvaders is an Android application so it can be installed on every Android device like a tablet to present it to any audience without further setup. The implementation of WebInvaders is based on the tutorial from John Horten [2] for coding a SpaceInvaders app with some alterations to fit the specific needs of this approach:

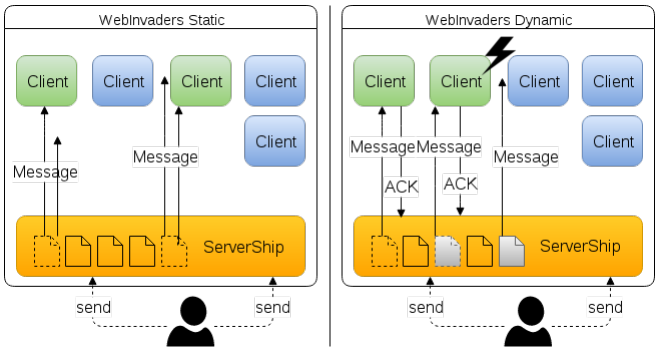


Fig. 5. Static vs. dynamic fault tolerance

The ServerShip can be moved by either touching the screen left or right of its current position and messages can be sent by touching the screen anywhere above its level. Furthermore a start screen realises the game state selection and illuminates the improvement of the score from state to state. These states are implemented as a state pattern, described by the "Gang of Four" in [5, p. 305], with three different `ToleranceStates` fitting the needs of the application and making it extendible as can be seen in Figure 6.

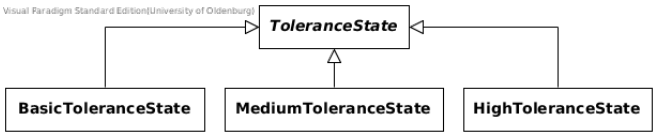


Fig. 6. WebInvaders ToleranceStates

V. Conclusion

The previous described WebInvaders application represents server communication in different stages with increasing fault tolerance and recognition and can thus be used as a means to illustrate this concept. The comparison of the different states has shown that the player should experience a considerable increase of score between the modes and therefore grasp the advantage of applying fault tolerance to a system. Its gaming nature should support the natural grasping of the strategy especially for younger users usually more familiar with computer games where the upgrading and levelling of a player entity is another constant application of fault tolerance.

The implementation of the states via the state pattern makes the application extendible and more states can easily be added to illustrate other fault tolerance principles.

References

[1] D. B. Clark, E. E. Tanner, S. Killingsworth: Digital Games, Design, and Learning: A Systematic Review and Meta-Analysis, 2014

[2] J. Horton: website: http : / / gamecodeschool . com / android / coding-a-space-invaders-game/, Juni 2015. Last access: 18.12.2015.

[3] I. Koren, C. Mani Krishna: Fault-Tolerant Systems, Elsevier, Morgan Kaufmann: 2007.

[4] Edsger W. Dijkstra: *The Humble Programmer*, ACM Turing Lecture 1972 www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns*, Addison-Wesley: 2004.

Entwicklung einer selbst-stabilisierenden Plattform auf Basis von Low-Budget-Komponenten

Christian Brunzendorf
Carl von Ossietzky Universität
Oldenburg, Deutschland
Email: christian.brunzendorf@uni-oldenburg.de

Yannick Habecker
Carl von Ossietzky Universität
Oldenburg, Deutschland
Email: yannick.habecker@uni-oldenburg.de

***Abstract*—Diese Arbeit behandelt die Implementierung einer einfachen und kostengünstigen selbst-stabilisierenden Plattform. Es werden mögliche und reale Einsatzgebiete vorgestellt. Die entwickelte Plattform wird inklusive dem zugrundeliegenden Fehlermodell und der Verifikation beschrieben.**

I. Einführung

In dieser Arbeit geht es um die Entwicklung einer einfachen und kostengünstigen selbst-stabilisierenden Plattform.

Die zunehmende Verwendung von unbemannten Luftfahrzeugen für Luftaufnahmen auch im privaten und kommerziellen Rahmen bedarf einer kostengünstigen Kamerastabilisierung um qualitativ hochwertige Bilder zu gewährleisten. Durch eine Stabilisierung, die Roll- und Neigungswinkel ausgleicht, kann die Bildqualität stark verbessert werden, so dass diese auch für Fernsehaufnahmen geeignet sind. Dafür könnte die Kamera auf einer selbst-stabilisierenden Plattform an dem Luftfahrzeug angebracht werden. Dabei übernimmt die Plattform die Aufgabe, störende Neigungs- und Rollwinkel zu erkennen und auszugleichen.

Aber auch in anderen Bereichen können selbst-stabilisierende Plattformen eingesetzt werden. In größerer Dimension kann eine solche Plattform Räume auf Schiffen stabilisieren. Dies würde es ermöglichen, beispielsweise Operationssäle auf Hochseeschiffen zu installieren, die auch bei schwerem Seegang nutzbar sind.

In dieser Arbeit werden zunächst verwandte Arbeiten vorgestellt. Im Hauptteil wird dann die im Rahmen dieser Arbeit entwickelte Plattform und das der Plattform zugrundeliegende Fehlermodell vorgestellt. Außerdem wird die Implementierung detailliert beschrieben und schließlich wird das Fehlermodell anhand des implementierten Systems mithilfe eines Tests verifiziert und Fehler, die dennoch auftreten können, in der Fehleranfälligkeitsanalyse beschrieben.

II. Verwandte Arbeiten

Bereits im 19. Jahrhundert begann die Entwicklung von ähnlichen Stabilisierungssystemen. Mithilfe eines mechanischen Gyroskops wurden Rollbewegungen auf Schiffen ausgeglichen. Zum ersten Mal eingesetzt wurde diese Technologie am Anfang des 20. Jahrhundert.

Die Verfügbarkeit von Microcontrollern, Sensorik und Aktuatorik im Niedrigpreissegment führte in den letzten Jahren zur Entwicklung von low-cost Stabilisierungssystemen.

So wurde in den vergangenen Jahren an einer selbst-stabilisierenden Plattform für Parabolantennen geforscht. Zum Empfang von Fernsehsignalen per Rundfunksatellit werden Parabolantennen an einem geographisch fixen Punkt montiert. Bei der Installation von Parabolantennen an mobilen Fahrzeugen wird das zum Problem, da eine feste Ausrichtung der Antenne auf einen Satelliten nicht mehr möglich ist. Der Einsatz einer selbst-stabilisierenden Plattform für Roll-, Neigungs- und Drehbewegungen ermöglicht die Ausgleichung aller störenden Bewegungen und damit den Empfang von Satellitensignalen. [1]

Ein weiteres Einsatzgebiet liegt in der schiffgestützten Erforschung des Klimas anhand von Wolkenbeobachtungen. Aufgrund fehlender Stabilisierung konnten in der Vergangenheit Wolken mithilfe eines schiffgestützten Radarsystems nur schwer erforscht werden. Obwohl es bereits geeignete Radarsysteme gab, konnten keine genauen Messungen über die Wolkenhöhe und die Geschwindigkeit der Wolkenpartikel gemacht werden. Erst die Entwicklung einer Stabilisierungs-Plattform konnte diese Probleme lösen und schiffgestützte Messungen ermöglichen. Um dies zu erreichen mussten neben Roll- und Neigungsbewegungen, welche das Ziel dieser Arbeit sind, auch vertikale Bewegungen stabilisiert werden. Nachdem diese Technologie erfolgreich auf Schiffen eingesetzt wurde, konnte sie auch erfolgreich in Flugzeugen eingesetzt werden und damit die Klimaforschung maßgeblich voranbringen. [2]

III. Hauptteil

Die Plattform ist leicht und tragbar, so dass der Benutzer die Lage der Plattform beeinflussen kann. Die Lage der Plattform kann durch rollen bezüglich der Längsachse und durch neigen bezüglich der Querachse beeinflusst werden. Der Regelungs-Algorithmus der Plattform hat die Aufgabe, die Parallelität in Relation zur Erdoberfläche beizubehalten.

Für die Regelung wurde das Konzept der Selbst-Stabilisierung verwendet. Vereinfacht besagt das Konzept der Selbst-Stabilisierung nach Dijstra, dass egal in welchem Zustand sich ein System befindet, es sich selbst stabilisiert, so dass es sich nach endlicher Zeit wieder in einem stabilen Zustand befindet. [4]

Das in dieser Arbeit diskutierte System soll sich innerhalb des im Folgenden beschriebenen Fehlermodells selbststabilisierend verhalten und in endlicher Zeit die Parallelität zur Erdoberfläche wiederherstellen und sich damit in einem stabilen Zustand befinden.

A. Fehlermodell

Die Plattform kann durch äußere Einflüsse aus der Ruhelage gebracht werden. Zu den äußeren Einflüssen gehört zum Beispiel ein sich bewegender Untergrund. Als Ruhelage wird die kalibrierte Ausgangsposition bezeichnet, die nach der Kalibrierung des Systems, welche nach dem Einschalten erfolgt, erreicht wird. Das System toleriert eine Abweichung von $\pm 60°$ um die Quer- und Längsachse und hält die Plattform durch eine Gegenreaktion in Waage. Bereits während der Drehbewegung hält die Plattform bis zu einer Drehgeschwindigkeit von $40°/_s$ die waagerechte Lage bei einem Fehler von höchstens $1,5°$. Bei einer Drehgeschwindigkeit von bis zu $60°/_s$ soll die Funktion mit einem maximalen Fehler von $2,1°$ gewährleistet werden. Bei keiner vorhandenen Drehbewegung hält die Plattform die waagerechte Lage mit einem Fehler von höchstens $0,5°$. Diese Anforderungen erforderten den Einsatz einer Echtzeitanalyse sowie -regelung.

B. Implementierung

1) Hardware:

Die entwickelte Plattform besteht grundlegend aus einer inneren Plattform, sowie zwei äußeren Rahmen. Die innere Plattform ist über einen Servo auf der einen Seite sowie durch eine Lagerung auf der anderen Seite befestigt, so dass die Plattform bezüglich des inneren Rahmens auf einer Achse frei durch den Servo gedreht werden kann. Analog zur Plattform ist auch der innere Rahmen am äußeren Rahmen befestigt, wobei dieser auf einer orthogonal liegenden Achse gelagert ist. Die Abbildung 1 stellt die Plattform vereinfacht dar.

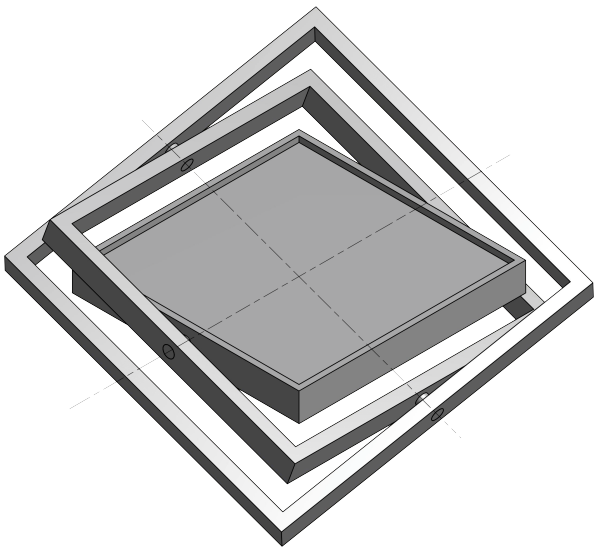


Abb. 1. Mechanischer Aufbau

Damit enthält die Plattform genau einen Servo pro gelagerter Achse. Die gesamte Aktuatorik besteht dementsprechend nur aus zwei Servos. Für die Aktuatorik wurden Servos des Typs Carson CS-6 verwendet. Dieser Servo stellt eine Stellgeschwindigkeit von $0,2\ {}^{s}/_{60\,°}$ bei einem Stellmoment von bis zu $60\ Ncm$ zur Verfügung. [12]

Die gesamte Sensorik ist auf dem 9DOF-Sensorboard untergebracht. Dieses enthält einen drei-achsialen Beschleunigungssensor, einen drei-achsialen Magnetometer, sowie ein drei-achsiales Gyroskopmodul. [8] Der eingesetzte Beschleunigungssensor (ADXL345) erkennt Beschleunigungen von bis zu $\pm 16g$ bei einer Genauigkeit von bis zu $0,004g$. [9] Das dreiachsiale Magnetometer (HMC5883L) stellt eine Genauigkeit zwischen $1\ °$ und $2\ °$ zur Verfügung. Mit diesem Sensor ist es möglich die Ausrichtung des Sensorboards zu erfassen. [10] Das Gyroskopmodul(ITG-3200) stellt Informationen über die Lage des Sensorboards im Raum zur Verfügung. Dabei werden Neigungs-, Roll- und Drehwinkel erfasst. [11] Die interne Kommunikation auf dem Sensorboard erfolgt über einen I2C-Bus.

Der I2C-Bus ist ein serieller Datenbus, der in den 1980er Jahren vom Philips Konzern entwickelt wurde. Über diesen Bus ist es möglich bis zu 128 Geräte miteinander zu verbinden. Die Daten können unidirektional versendet werden, so dass jedes angeschlossene Gerät mit jedem anderen kommunizieren kann. Als Datenleitung werden nur zwei Bus-Leitungen benötigt. [14]

Der verwendete Controller ist ein Arduino MEGA 2560. Dieser Controller stellt unter anderem eine I2C-Busschnittstelle zur Kommunikation zur Verfügung. Die Kommunikation zur Sensorik erfolgt über eine serielle Schnittstelle.

Die verwendete serielle Schnittstelle stellt eine direkte Peer-to-Peer Verbindung zur Verfügung. Dies ermöglicht eine Echtzeitauswertung der Messdaten, da keine Störungen durch andere Teilnehmer über die Verbindung zu erwarten sind. [6]

Da der Arduino Mega Controller keine ausreichende Stromstärke für den Betrieb der Aktuatorik zur Verfügung stellt, wurde eine Erweiterungsplatine für den Controller verwendet. Das Adafruit 16-channel PWM/Servo Shield ermöglicht eine externe Stromversorgung für die Aktuatorik. Die Kommunikation zwischen Controller und Erweiterungsplatine erfolgt über die I2C-Busschnittstelle. [13]

Die Ansteuerung der Aktuatorik erfolgt mittels Pulsweitenmodulation. Die Pulsweitenmodulation ermöglicht die Übertragung von Spannungssignalen über einen Digitalausgang. Der Ausgang wird dabei mit einer festen Taktfrequenz f_T getaktet, wobei Spannungen zwischen den vordefinierten Werten $U_{min} = 0$ und U_{max} übertragen werden können. Der digitale Ausgang muss dabei die Ausgangsspannung von U_{max} ermöglichen. Zur Übertragung einer Spannung wird der Digitalausgang innerhalb eines Taktzyklus nur anteilsweise aktiviert. Um bspw. eine Spannung von $U = \frac{1}{2}U_{max}$ zu übertragen ist der Digitalausgang während eines Taktzyklus nur

$$\Delta t = \frac{U}{U_{max} \cdot f_T} = \frac{1}{2 \cdot f_T} s \tag{1}$$

aktiv. Daraus ergibt sich eine durchschnittliche Ausgangsspannung, die dem Soll entspricht. [15]

2) Software:

Die Software besteht zum einen aus der bereits bestehenden Firmware des 9DOF-Sensorboards, sowie aus der in dieser Arbeit entwickelten Software für den Arduino-Mega-Controller. Im Folgenden soll zunächst auf die verwendete Fremdsoftware eingegangen werden um im Anschluss die im Rahmen dieser Arbeit entwickelte Regelung zu beschreiben.

Auf dem 9DOF-Sensorboard wurde eine spezielle Firmware (siehe [16]) installiert, die unter Verwendung des Direction Cosine Matrix Algorithmus die Lage des Sensorboards, also die Winkel bezüglich Längsachse(Rollwinkel), Querachse(Neigungswinkel) sowie Gierachse(Drehwinkel) berechnet. [17] Dem Sensorboard stehen dafür Daten der drei vorhandenen Sensoren (Beschleunigungssensor, Magnetometer und Gyroskop) zur Verfügung. Das Ziel des Direction Cosine Matrix Algorithmus ist es die Daten miteinander zur Vermischen um die Lagewinkel genau zu berechnen. So verfälschen zum Beispiel Beschleunigungen um die Gierachse die Messwerte des Gyroskops über den Neigungs- und Rollwinkel. Der Algorithmus verwendet nun den Beschleunigungssensor um diesen Fehler wieder zu kompensieren. [3]

Die in dieser Arbeit entwickelte Software für den Arduino-Mega-Controller verwendet zum einen von uns entwickelte Hilfsbibliotheken, die im folgenden beschrieben werden, sowie eingebundene Bibliotheken, auf die zunächst eingegangen wird.

Zur Kommunikation zwischen Controller und Sensorboard sowie zwischen Controller und angeschlossenen USB-Geräten (z.B. der Entwicklungsrechner) wurde die *Wire.h* Bibliothek (als Public Domain veröffentlicht) verwendet. [5] Außerdem wurde die unter der gleichen Lizenz veröffentlichte Standartbibliothek für Arduino-Controller *Arduino.h* verwendet.

Für die Ansteuerung der Servos mittels Pulsweitenmodulation wurde die *Adafruit PWM Servo Driver*-Bibliothek (veröffentlicht unter der BSD Lizenz) verwendet, diese vereinfacht die Steuerung der beiden Aktuatoren.

Die Ansteuerung der Aktuatorik erfolgt außerdem über eine als Teil dieser Arbeit entwickelten Bibliothek. Nach einer Kalibrierung ermöglicht sie die Ansteuerung eines Servos über die Angabe der Sollposition in Grad.

Das zentrale Problem bei der Steuerung war die Regelung. Um eine möglichst optimale Regelung zu erhalten wurde eine Hilfsbibliothek für die Ansteuerung eines Servos über einen Fuzzy-(PI)-Regler entwickelt. Dieser Fuzzy-Regler reagiert im Vergleich zu einem normalen PI-Regler dynamischer auf variable Fehlerabweichungen, indem der P- und I-Anteil durch die Fuzzy-Regelung bei der Steuerung angepasst wird.

Zur Verwendung der Fuzzy-Regelung wird die zu regelnde Sollgröße vorgegeben. Mithilfe der oben genannten Bibliothek zur Ansteuerung der Servos wird dann die Regelgröße an den jeweiligen Servo weitergegeben.

Auf Basis dieser Bibliotheken ermittelt das Hauptprogramm zunächst die nötigen Daten über die serielle Schnittstellenverbindung zwischen Controller und Sensorboard. Nach der Erfassung dieser Daten wird geprüft, ob ein Übertragungsfehler stattgefunden hat. Ein häufiger Übertragungsfehler ist das Vertauschen von Sensorwerten. Hierbei werden bspw. die Werte für Roll- und Neigungswinkel vertauscht. Dieser Fehler wird kompensiert, indem die neuen Messwerte mit vorherigen Werten überprüft und gegebenenfalls vertauscht werden.

Als nächstes wird über die Messwerte die Abweichung zwischen Lage der Plattform und der zur Erdoberfläche parallelen Lage ermittelt. Durch die vorhergehende Kalibrierung der Servos in Verbindung mit der Sensorik ist die Ermittlung dieser Abweichungen (in Bezug auf Roll- und Neigungswinkel) trivial. Mithilfe der nun bestimmten Werte wird nun die bereits beschriebene Fuzzy-Regelung für die Servos mit den berechneten Sollwerten aktiviert.

C. Verifikation

Aufgrund der Komplexität der eingesetzten Software und der Regelung ist eine Verifikation der Gesamtsteuerung nicht mit angemessenem Aufwand möglich. Zur Verifikation des Fehlermodells wurde aus diesem Grund ein Test durchgeführt, dessen Aufbau und Ergebnis im Folgenden beschrieben wird.

Der durchgeführte Test bestätigt die im Fehlermodell angegebenen Daten. Im Zuge des Tests sollte die Fehlertoleranz von $\pm 60\ ^\circ$ und die Gewährleistung der Funktion bei einer Drehbewegung von maximal $40\ ^\circ/s$ und $60\ ^\circ/s$ bestätigt werden. Während des Tests sollte die Plattform eine Drehbewegung von zunächst $40\ ^\circ/s$ und im Anschluss $60\ ^\circ/s$ ausgleichen. Mit den genannten Geschwindigkeiten wurden Lageänderungen bis $\pm 30\ ^\circ$ und $\pm 60\ ^\circ$ durchgeführt. Zur Reproduzierbarkeit des Tests wurden die Lagen von $\pm 30\ ^\circ$ und $\pm 60\ ^\circ$ in einer bestimmten Reihenfolge ($0\ ^\circ$, $-30\ ^\circ$, $30\ ^\circ$, $-60\ ^\circ$, $60\ ^\circ$, $0\ ^\circ$) angefahren (siehe Abbildung 2). Dabei wurde neben dem Winkel bzgl. der Längs- bzw. Querachse auch die Abweichung zur waagerechten Laage aufgezeichnet. Der Test wurde für Bewegungen um die Längs- sowie Querachse durchgeführt. Zur besseren Durchführbarkeit wurde die Drehbewegung mittels einer Manipulation der Lagesensorik simuliert. Die daraus folgende Reaktion des Systems gleicht der einer realen Reaktion in einer realen Umgebung.

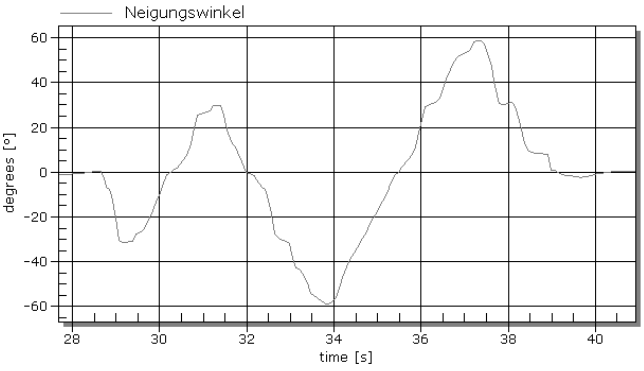


Abb. 2. Testablauf

In der Auswertung wurden die Abweichungen zur waagerechten Lage im Mittel errechnet. Die Auswertung ergab bei einer Drehgeschwindigkeit von $40\ ^\circ/s$ eine Fehlertoleranz

von $\pm 1,17\ ^\circ$ bezüglich des Neigungswinkels und $1,4\ ^\circ$ bezüglich des Rollwinkels. Bei einer Drehgeschwindigkeit von $60\ ^\circ/s$ wurden Fehlertoleranzen von $\pm 1,3\ ^\circ$ bzw $\pm 2,0\ ^\circ$ erreicht. Die Abweichungen zwischen den Fehlertoleranzen der Roll- und Neigungswinkel sind teilweise auf die geringe Messfrequenz von $16\ Hz$ zurückzuführen. Aufgrund der niedrigen Messfrequenz war die Messdatendichte stark begrenzt.

Insgesamt betrachtet, bestätigt der Test das Fehlermodell.

D. Fehleranfälligkeitsanalyse

Im Folgenden soll beschrieben werden, welche Fehler zu einer Fehlfunktion des Systems führen.

Aufgrund der relativ hohen Berechnungszeit des 9DOF-Sensorboards von $20\ ms$ liegen die Messdaten nur mit einer Messfrequenz von $50\ Hz$ zur Verfügung. Im Worst-Case sind damit die Messdaten, die in der Regelung verarbeitet werden, bis zu $20\ ms$ alt. Außerdem steht für die Pulsweitenmodulation für die eingesetzte Aktuatorik nur eine Taktfrequenz von $f_T = 60\ Hz$ zur Verfügung. Damit beträgt die Zeitspanne, die für die Übertragung eines einzelnen Wertes über die Pulsweitenmodulation verwendet werden kann, $t = \pm 16,67\ ms$. Zusätzlich beansprucht die Aktuatorik eine Stellzeit von $0,2s$ für eine Stellgrößenänderung von $60\ ^\circ$.

Die eingesetzte Aktuatorik ermöglicht außerdem nur eine relativ ungenaue Ansteuerung. Aus den Kalibrierungsdaten des Systems geht hervor, dass die Ansteuerung der Servos mittels Pulsweitenmodulation nur mit einer maximalen Genauigkeit von $\pm 0,2\ ^\circ$ möglich ist. Aufgrund dieser Ungenauigkeit in Verbindung mit den hohen Verzögerungszeiten ist die Funktion der Echtzeitregelung bei einer Drehgeschwindigkeit, die $60\ ^\circ/s$ übersteigt, nicht mehr garantiert.

IV. Zukünftige Arbeiten

In dieser Arbeit ging es ausschließlich um Bewegungen um die Quer- und Längsachse. Dies schränkt den Anwendungsbereich ein, so dass eine Weiterentwicklung des bisherigen Prototyps sinnvoll erscheint.

Der nächste Schritt wäre die Ausgleichung von Bewegungen um die Gierachse. Dafür stellt das verwendete 9DOF-Sensorboard bereits alle Informationen bereit. Um dies zu implementieren kann ein weiterer Rahmen in Betracht gezogen werden. Das hätte aber den Nachteil, dass die für die Stabilisierung notwendige Mechanik in Relation zur stabilisierten Plattform sehr groß wäre. Eine bessere Möglichkeit bietet eine Art Arm, wie in [1] verwendet, da dieser wesentlich weniger Platz braucht.

Mithilfe eines solchen Arms könnten auch Beschleunigungen ausgeglichen werden, indem durch eine Lageveränderung die Schwerkraft als Gegenkraft genutzt wird. Dafür müsste dann nur die Software implementiert werden, da Änderungen am System nicht mehr notwendig wären.

Zuletzt könnten noch störende Vibrationen mithilfe von Stoßdämpfern eliminiert werden. Eine Möglichkeit wäre es, dafür die Plattform mittels geeigneten Stoßdämpfern von dem restlichen System zu entkoppeln.

V. Zusammenfassung

Die im Rahmen dieser Arbeit entwickelte Plattform spiegelt das Konzept der Selbst-Stabilisierung in einem begrenzten Rahmen wieder. Das Ziel der eingesetzten Regelung ist es, die Plattform in einer waagerechten Lage zu halten, auch wenn die Lage der Plattform durch äußere Einflüsse verändert wird.

Obwohl das System nur in einem kleinen Team mit einem geringen Budget entwickelt wurde, konnte ein funktionsfähiger Prototyp vorgestellt werden.

Literaturverzeichnis

[1] Said Leghmizi und Sheng Liu, International Journal of Computer Science & Engineering Survey (Ausgabe 2 No. 3, 2011): A survey of fuzzy control for stabilized platforms.

[2] Ken Moran, Sergio Pezoa, Chris Fairall, Chris Williams, Tom Ayers, Alan Brewer, Simon P. de Szoeke und Virendra Ghate, Boundary-Layer Meteorol (Ausgabe 143, 2012) S. 3-24: A Motion-Stabilized W-Band Radar for Shipboard Observations of Marine Boundary-Layer Clouds

[3] William Premerlani und Paul Bizard, *Direction Cosine Matrix IMU: Theory*, Draft: 17.05.2009

[4] Edsger W. Dijkstra, Commun. ACM 17 (Ausgabe 11, 1974) S. 643–644: EWD 426: Self-stabilizing systems in spite of distributed control

[5] Arduino LLC, Wire Library Reference, https://www.arduino.cc/en/Reference/Wire (09.01.2016)

[6] Arduino LLC, Serial Reference, https://www.arduino.cc/en/Reference/Serial (10.01.2016)

[7] Adafruit, Adafruit PWM Servo Driver Library, https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library (09.01.2016)

[8] SparkFun Electronics Inc, SparkFun 9 Degrees Of Freedom - Sensor stick, https://www.sparkfun.com/products/10724 (09.01.2016)

[9] SparkFun Electronics Inc, 3-Axis Digital Accelerometer ADXL345, https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf (15.01.2016)

[10] Honeywell International Inc, 3-Axis Digital Compass HMC5883L, http://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf (15.01.2016)

[11] InvenSense Inc, 3-Axis Gyroscope ITG-3200, https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf (15.01.2016)

[12] DICKIE-TAMIYA Modellbau GmbH & Co. KG, Carson Servo CS-6 http://www.carson-modelsport.com/de/produkte/fernsteueranlagen/servos/produktdetails.htm?sArtNr=500502041 (15.01.2016)

[13] Adafruit, Adafruit 16-channel PWM/Servo Shield Overview, https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield/overview (16.01.2016)

[14] NXP Semiconductors N.V., I2C-bus specification and user manual, http://www.nxp.com/documents/user_manual/UM10204.pdf (16.01.2016)

[15] Springer Vieweg Verlag, Joachim Specovius, Grundkurs Leistungselektronik (6. Auflage, 2013) S. 266ff: 15.3 Die Pulsweitenmodulation (PWM)

[16] Peter Bartz, Razor 9DOF AHRS Firmware, https://github.com/ptrbrtz/razor-9dof-ahrs (16.01.2016)

[17] Peter Bartz, Razor 9DOF AHRS Firmware - Mathematical background and firmware internals, https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial (16.01.2016)

Fehlertoleranz im Modellauto

Michael Beering, Universität Oldenburg

***Abstract*—Die Ausarbeitung zeigt anhand eines Modellfahrzeugs die Umsetzung verschiedener Fehlertoleranzkonzepte. Hierbei wurden ein Spurhalteassistent und ein Mechanismus zur Vermeidung von Kollisionen mit Vorfahrzeugen implementiert und mit dem Modellfahrzeug getestet. Der Artikel erläutert das Fehlermodell, die eingesetzten Fehlertoleranzkonzepte, beschreibt den entwickelten Prototypen sowie die Software und analysiert, welche Fehler dennoch zum Versagen der Assistenten führen.**

I. Motivation

Die Fahrzeugtechnologie forscht seit einigen Jahren intensiv in der Entwicklung von Fahrassistenzsystemen. Hiermit wird die Sicherheit der Fahrzeuginsassen erhöht und der Fahrkomfort verbessert. Zur Bewältigung komplexer Fahrsituationen müssen die Assistenzsysteme in der Lage sein, mit Fehlern verschiedenster Herkunft umzugehen. Beispielsweise sollte ein Bussystem zur Signalübertragung ein gewisses Maß an auftretenden Fehlern erkennen und im Idealfall korrigieren können. Außerdem sollten Assistenzsysteme einen Ausfall von Sensoren tolerieren können oder bei einem vollständigen Versagen der Funktionalität entsprechende Notfallmaßnahmen einleiten. Um die Zuverlässigkeit und Verfügbarkeit eines Systems im Auto zu erhöhen, spielen Konzepte der Fehlertoleranz eine bedeutende Rolle. Zur Erkennung von Übertragungsfehlern verwendet der CAN-Bus, welcher die Kommunikation zwischen den meisten Steuergeräten realisiert und in der Automotivdomäne industrieller Standard ist, beispielsweise das CRC-Verfahren.

II. Idee

Das Ziel dieser Arbeit besteht darin, am Beispiel eines Modellautos Konzepte der Fehlertoleranz zu veranschaulichen. Das Fahrzeug soll im Wesentlichen zwei Nutzfunktionen erbringen: Die erste besteht darin, dass das Fahrzeug in der Lage sein soll einer Linie automatisch zu folgen. Auf diese Weise soll ein Spurhalteassistent für das Modellauto realisiert werden. Ein weiteres Ziel ist die Entwicklung eines Assistenzsystems, welches Kollisionen in Form von Auffahrunfällen vermeidet. Hierfür soll das Vorfahrzeug erkannt und die Geschwindigkeit entsprechend des vorfahrenden anpasst werden.

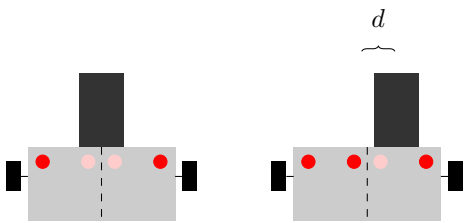
Beim Erbringen dieser beiden Nutzfunktionen können verschiedene Fehler auftreten. Das System soll derart ausgelegt werden, dass diese auftretenden Fehler komplett oder partiell kompensiert werden können.

III. Fehlermodell

In Systemen wie dem Modellauto, welches mit zwei einfachen Assistenzsystemen ausgestattet ist, treten verschiedene Arten von Fehlern auf. Üblicherweise äußern sich diese Fehler dadurch, dass ein Teil eines Systemzustandes von seinem Sollwert abweicht. Die Folgen, die ein Fehler auf das Gesamtsystem hat, können einen unterschiedlichen Grad erreichen und ihren Höhepunkt im vollständigen Versagen des Systems finden. Die auftretenden Fehler, die daraus resultierenden Folgen und die Ursachen werden in einem Fehlermodell zusammengefasst. Der folgende Abschnitt entwickelt ein Fehlermodell für das Modellfahrzeug und geht auf die Kompensation der auftretenden Fehler ein, sodass die Nutzfunktionen der Fahrassistenzsysteme weiter erbracht werden können.

A. Die Fahrbahnmitte wird verlassen

Die zentrale Funktion eines Spurhalteassistenten besteht darin, das Fahrzeug näherungsweise auf der Fahrbahnmitte zu halten. Veranschaulicht wird dieser ideale Zustand in Abbildung 1(a). Die Fahrbahn ist in diesem Fall durch eine durchgezogene Linie modelliert worden. Verlässt das Fahrzeug die Spur, wie in Abbildung 1(b) dargestellt, ist eine Abweichung vom Sollwert aufgetreten und es liegt somit ein Fehler vor. Die Abweichung ist in der Abbildung durch das Zeichen d beschrieben. Das Verlassen der Fahrbahn kann über vier Sensoren festgestellt werden. Diese sind in der Abbildung durch rote LEDs gekennzeichnet. Sind die zwei LEDs in der Mitte des Fahrzeugs aus, ist eine schwarze Linie erkannt worden und das Fahrzeug befindet sich näherungsweise in der Mitte der Fahrspur. Wird die schwarze Linie nicht erkannt, so leuchten die LEDs. Im Fall von Abbildung 1(b) hat das Fahrzeug die Spur nach links verlassen und die zweite LED leuchtet entsprechend rot.



(a) Fahrzeugmittelpunkt ist exakt in Mitte der Fahrspur.

(b) Fahrzeugmittelpunkt hat die Fahrspur verlassen.

Abb. 1. Fahrzeug und dessen Verhalten auf einer geraden Fahrspur.

Für das Auftreten des beschriebenen Fehlers sind verschiedene Ursachen verantwortlich. Die erste Ursache besteht darin, dass die Mechanik, welche die Vorderachse des Fahrzeugs ausrichtet, Fertigungstoleranzen unterliegt. Dies hat zur Folge, dass bei Neutralstellung der Vorderachse, welche eine Vorwärtsfahrt herbeiführen sollte, immer ein leichtes Spiel der Räder nach rechts bzw. links existiert. Das beschriebene Spiel lässt sich auf einer langen geraden Strecke deutlich wahrnehmen. Der minimale Winkelver-

satz der Vorderachse zur Idealposition führt dazu, dass das Fahrzeug bei Geradeausfahrt langsam die Fahrspur nach links oder rechts verlässt und der Sollwert wird wie in Abbildung 1(b) verlassen. Wird der hier auftretende Fehler nicht kompensiert, führt das dazu, dass die Fahrbahn vollständig verlassen und das Fahrzeug mit Gegenständen kollidiert.

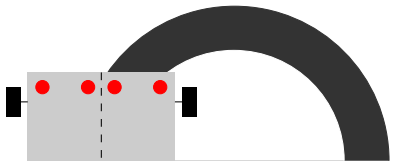
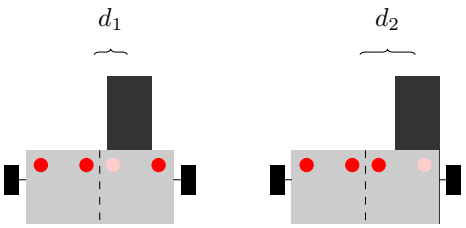


Abb. 2. Fahrzeug und dessen Verhalten in einer Kurve.

Eine weitere Ursache für das Verlassen der Mitte der Fahrspur ist eine Änderung der Straßengeometrie. Abbildung 2 veranschaulicht die Problematik. Durchfährt das Fahrzeug eine Kurve, so wird die Fahrbahn direkt verlassen. Um dies zu verhindern, müssen geeignete Maßnahmen getroffen werden.

Dazu kommt das Auftreten von Schlupfeffekten. Diese sind auf die Reibung zwischen Reifen und Fahrbahn zurückzuführen. Da die Fahrbahn aus aufgeklebtem Isolierband besteht, sind an vielen Stellen, an denen ein gerader Streckenabschnitt vorliegen soll, leichte Krümmungen sichtbar. Dies führt ebenfalls zum Verlassen der Fahrbahn.

Um eine Toleranz gegen das Verlassen der Fahrspur zu erhalten, ist ein Gegenlenken je Richtung der Überfahrt des Fahrstreifens notwendig. Es hat sich folgendes einfaches Modell bewährt.



(a) Verlassen der Fahrspur am ersten Sensor detektiert.

(b) Verlassen der Fahrspur am zweiten Sensor detektiert.

Abb. 3. Fahrzeug verlässt Fahrspur.

Abbildung 3(a) zeigt, dass die Fahrbahn um den Abstand d_1 verlassen wurde. Optokoppler 2 hat dies aufgrund des Helligkeitsunterschieds erkannt und die entsprechende LED wurde eingeschaltet. Ist dies der Fall, so wird das Lenkrad um einen kleinen Wert, z.B. $\phi = 5°$, nach rechts eingeschlagen und somit in Richtung des Fahrstreifens gelenkt. Wird die Fahrbahnmitte so nicht erreicht, wird sich der Zustand in Abbildung 3(b) einstellen. Offensichtlich muss stärker gegengelenkt werden und der maximale Lenkwinkel von ca. $\phi_{max} = 20°$ wird eingeschlagen. Dieses sehr einfache Verhalten lieferte dennoch gute Ergebnisse.

B. Ein Vorfahrzeug erscheint

Adaptive Cruise Control ist ein Assistenzsystem, welches einem Fahrzeug ermöglicht ein vorausfahrendes Fahrzeug zu erkennen und die eigene Geschwindigkeit gemäß der Geschwindigkeit des Vorfahrzeuges anzupassen. Auf diese Weise werden Zusammenstöße vermieden und der Fahrkomfort erhöht. Das im Modellauto implementierte Verfahren orientiert sich an diesem System und hat folgende Vorgehensweise: Wird kein Vorfahrzeug erkannt, so fährt das Fahrzeug mit einer vom Benutzer gewählten Geschwindigkeit. Wird jedoch ein Vorfahrzeug erkannt, so wird die Geschwindigkeit angepasst. In diesem Kontext ist der fehlerfreie Zustand wie folgt definiert. Abbildung 4 zeigt die Situation, in der ein Vorfahrzeug erscheint. Ist der Abstand zu diesem größer als der Abstand d_{min}, so liegt ein fehlerfreier Zustand vor. Wird der Abstand geringer, so wird durch das Vorfahrzeug ein Fehler verursacht, der sich im Unterschreiten der minimalen Distanz äußert. Die Ursache des Fehlers liegt daher in dem in Erscheinung tretenden Vorfahrzeug.

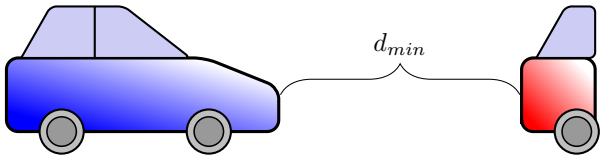


Abb. 4. Schema eines Fahrzeugs mit Vorfahrzeug.

Um den Fehler des Unterschreitens des Sollabstandes tolerieren zu können, ist es sinnvoll, dass das Fahrzeug seine Geschwindigkeit der Geschwindigkeit des Vorfahrzeugs anpasst. Da die Geschwindigkeit des Vorfahrzeugs nicht bekannt ist, kann diese approximiert werden. Dazu wird ein Ultraschallsensor an der Frontseite des Fahrzeugs angebracht. Die Funktion des Sensors beruht auf der Reflexion von Ultraschallwellen an einem Gegenstand (in diesem Fall am Vorfahrzeug). Aus dem reflektierten Signal kann ein Abstand bestimmt werden. Abbildung 5 modelliert die Idee zur Bestimmung der Geschwindigkeit des Vorfahrzeugs.

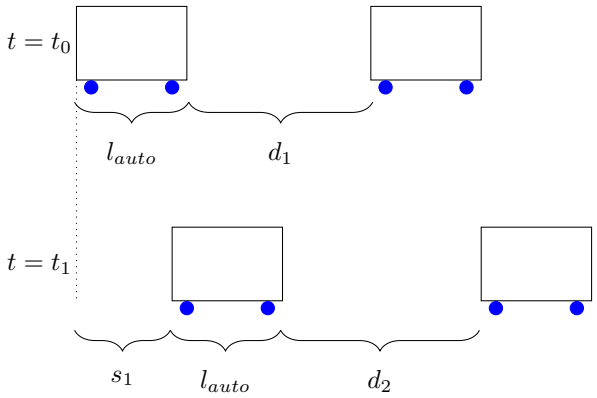


Abb. 5. Berechnung der Relativgeschwindigkeit zwischen zwei Fahrzeugen.

Da die Geschwindigkeit des Vorfahrzeugs nicht direkt berechnet werden kann, ist ein Umweg über die Relativgeschwindigkeit notwenig. Die Relativgeschwindigkeit beschreibt in diesem Kontext die Geschwindigkeit, die Fahrzeug F_2 relativ zu Fahrzeug F_1 aufweist. Den Zusam-

menhang zwischen den Geschwindigkeiten zeigt Formel 1.

$$v_{rel} = v_1 - v_2 \tag{1}$$

Ist die Relativgeschwindigkeit positiv, so fährt Fahrzeug F_1 schneller. Ist sie hingegen negativ, so fährt Fahrzeug F_2 schneller. Durch Umstellen der Formel lässt sich leicht die Geschwindigkeit des Vorfahrzeugs bestimmen. Gleichung 2 zeigt den Zusammenhang.

$$v_2 = v_1 - v_{rel} \tag{2}$$

Die Relativgeschwindigkeit lässt sich nun mit Abbildung 5 bestimmen. Zum Zeitpunkt $t = t_0$ fährt Fahrzeug F_1 mit der Geschwindigkeit v_1 hinter Fahrzeug F_2. Über den Ultraschallsensor lässt sich außerdem der Abstand d_1 ermitteln.

$$v_{rel} = \frac{d_2 - d_1}{t_1 - t_0} \tag{3}$$

Wiederholt man die Abstandsmessung zum Zeitpunkt $t = t_2$, so hat Fahrzeug F_1 den Weg s_1 zurückgelegt. Darüber hinaus kann der Abstand d_2 durch eine weitere Messung ermittelt werden. Einsetzen in Formel 3 liefert die entsprechende Relativgeschwindigkeit von Fahrzeug F_2 zu F_1, sodass hiermit auf die Geschwindigkeit v_2 geschlossen werden kann. Jetzt kann die Geschwindigkeit des Fahrzeugs F_1 angepasst werden.

Ist die Geschwindigkeit entsprechend angepasst, wird dies zwar ein Auffahren auf das Vorfahrzeug verhindern, allerdings wird unter Umständen noch nicht der Sollabstand eingehalten und der Fehler wird noch nicht toleriert. Hierzu wird die Geschwindigkeit solange eine Geschwindigkeitsstufe tiefer gesetzt, bis der korrekte Abstand gemessen wurde.

C. *Der Sensor fällt aus*

Der letzte mögliche Fehler besteht in einem Verlust der Fahrspur, welcher auf einem Ausfall der Sensoren, welche die Fahrbahn detektieren, beruht. Der durch den Ausfall auftretende Fehler ist gravierend und führt zu einem vollständigen Versagen des Spurhalteassistenten. Die Ursache für einen Sensorausfall kann in einem Defekt der Sensoren selbst, defekten Kabeln oder schlechten Lötstellen liegen.

Um diesen Fehler zu kompensieren lässt sich ein zweites Sensorboard hinzufügen, welches parallel oder im Bedarfsfall eingesetzt werden kann. Da keine weiteren Sensoren für das Projekt verfügbar waren, wurde dieser Ansatz nicht umgesetzt.

IV. Prototyp

A. *Hardware*

Der entwickelte Prototyp basiert auf dem in Abbildung 6 dargestellten Modellbauchassis. Markierung 1 zeigt den Motor des Modellfahrzeugs. Die Abtriebswelle des Motors ist über ein Getriebe mit den Rädern des Fahrzeugs verbunden. Angesteuert wird der Motor über einen Geschwindigkeitsregler. Der Regler erwartet als Eingabe pulsweitenmodulierte Signale mit einer Pulsfrequenz von $1kHz$, wobei über die Länge des Impulses die Geschwindigkeit des Fahrzeugs eingestellt werden kann. Der Regler ermöglicht sowohl eine Vorwärtsfahrt als auch eine Rückwärtsfahrt, wobei für diese Arbeit die Funktion der Rückwärtsfahrt deaktiviert wurde, da sie nicht benötigt wird. Die Stromversorgung des Geschwindigkeitsreglers und des Motors lässt sich direkt über den rechts befestigten Akku realisieren. Dieser liefert eine Ausgangsspannung von $7.2V$, welche der Versorgungsspannung des Motors entspricht.

Die Lenkung des Fahrzeugs zeigt Markierung 2. Diese besteht aus einem Servomotor, der die Lenkmechanik ansteuert. Der Servo erwartet als Eingabe ebenfalls pulsweitenmodulierte Signale, allerdings ist hierbei eine Pulsfrequenz von $50Hz$ zu verwenden. Über die Länge des Impulses lässt sich die Drehung des Servomotors einstellen, die durch die Lenkmechanik die Vorderreifen in Stellung bringt.

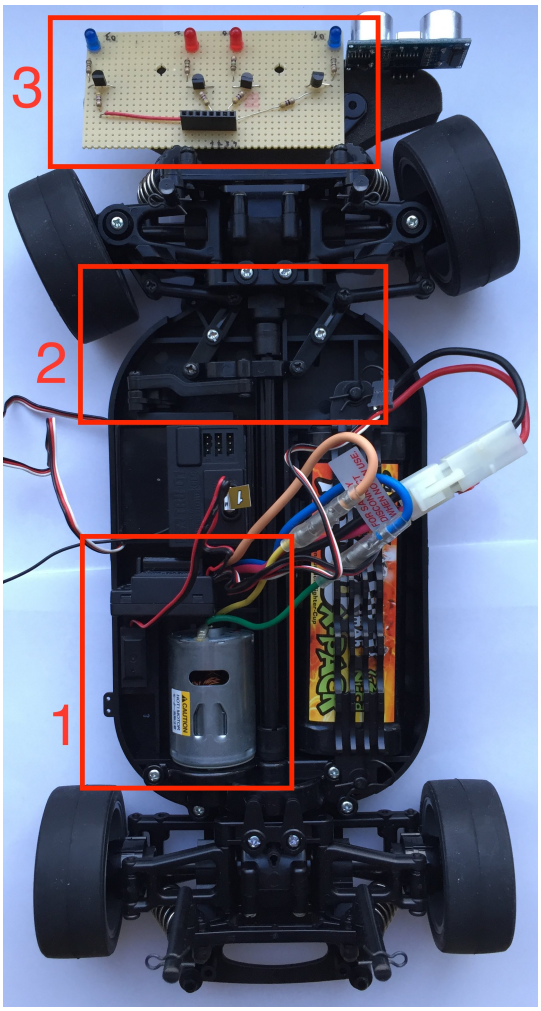


Abb. 6. Ansicht des Modellautos ohne Montageplatte.

Abbildung 7 zeigt die Montageplatte des Fahrzeugs. Bei dieser handelt es sich um einen Zuschnitt aus Dachpappe, der über Kabelbinder in der Mitte des Fahrzeugs befestigt worden ist. Der Vorteil dieser Platte besteht darin, dass elektronische Bauteile leicht darauf befestigt werden können.

Bei dem mit Nummer 1 markierten Bauteil auf der Montageplatte handelt es sich um einen Schaltregler, welcher eine zu hohe Eingangsspannung auf einen eingestellten Ausgangswert regelt. Der Regler erhält eine Eingangsspannung von $7.2V$ von dem daneben befestigen Akku. Da die übrigen elektronischen Bauteile des Fahrzeugs eine Versorgungsspan-

nung von $5V$ benötigen, wird über den Schaltregler eine entsprechende $5V$ Spannung ausgegeben. Die Verteilung der Energieversorgung wird über die mit 2 markierte Platine realisiert. Diese ist mit dem Ausgang des Schaltreglers verbunden und ermöglicht über eine entsprechende Stiftleiste die Verbindung zu anderen Bauteilen.

Markierung Nummer 3 zeigt den eingesetzten Mikrocontroller, der sämtliche Steuerungsaufgaben an Bord des Fahrzeugs übernimmt. Es handelt sich dabei um ein STM32F4 Discovery Board. Das Board verfügt über einen 32-Bit ARM Cortex-M4F Kern, 1 MB Flash und 192 KB RAM Speicher. Des Weiteren enthält es einige LEDs um Hinweise auszugeben. Die Stromversorgung erfolgt per USB oder über den zuvor beschriebenen Schaltregler. Zur Ansteuerung des Geschwindigkeitsreglers und des Servomotors für die Lenkung lassen sich über das Board pulsweitenmodulierte Signale mit verschiedenen Pulsfrequenzen erzeugen. Diese werden auf einen Ausgangspin des Boards gelegt und können über die entsprechenden Steuerleitungen mit den Bauteilen verbunden werden.

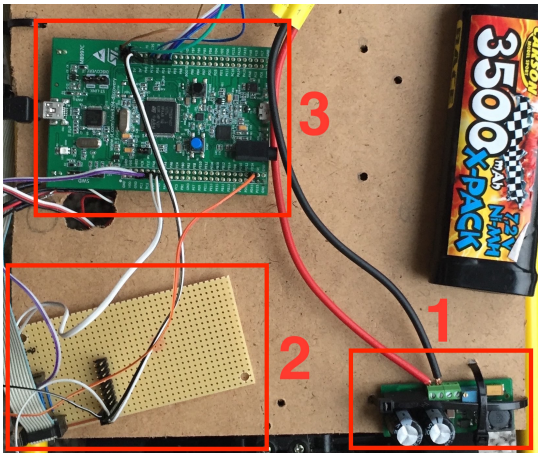
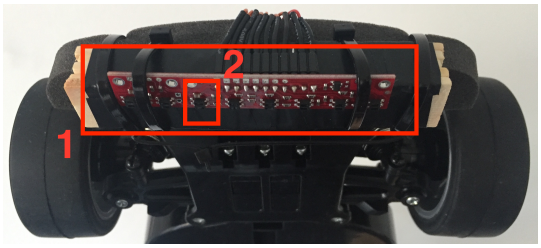


Abb. 7. Ansicht des Modellautos mit Montageplatte.

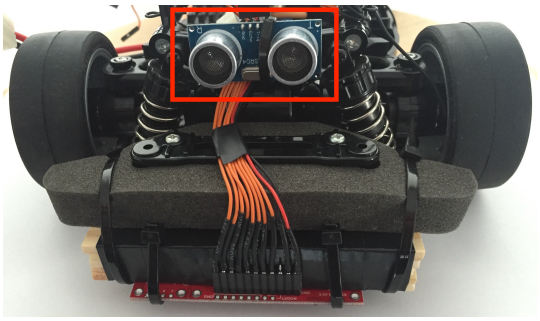
Abbildung 8(a) zeigt die an der Frontseite des Fahrzeugs verbauten Reflexoptokoppler (Markierung 1), mit denen eine Linie auf dem Boden erkannt werden kann. Bei einem Reflexoptokoppler handelt es sich um ein Bauteil, welches aus einer Infrarot-LED und einem Fototransistor besteht. Strahlt die LED eine Fläche an, so wird diese einen gewissen Anteil des Lichts reflektieren. Dieser Anteil wird vom Fototransistor detektiert und ein Rückschluss auf die Helligkeit der angestrahlten Fläche ist möglich. Ist die Fläche schwarz, wird wenig Licht reflektiert, ist sie hingegen weiß, so wird deutlich mehr Licht zum Fototransistor reflektiert. Auf dieser Basis können sehr einfach weiße Fahrbahnmarkierungen auf einem schwarzen Untergrund erkannt werden. Insgesamt sind acht Reflexoptokoppler am Fahrzeug verbaut. Markierung 2 zeigt einen einzelnen Optokoppler auf der Plantine. Versorgt werden die Optokoppler von der $5V$ Spannung.

Um die Werte, welche die Reflexoptokoppler liefern, auszulesen, ist es notwendig, diese über einen im Mikrocontroller integrierten Analog-Digital-Wandler auszulesen. Hierfür sind für die vier Sensoren vier Pins am Board notwendig, sodass die entsprechenden Werte später in der Software verwendet werden können.

Bei dem in Abbildung 8(b) dargestellten Bauteil, welches an der Frontseite des Fahrzeugs befestigt ist, handelt es sich um den Ultraschallsensor. Versorgt wird auch dieses Bauteil mit einer $5V$ Versorgungsspannung. Gelesen werden die Werte des Sensors über einen Analog-Digital-Wandler.



(a) Fahrzeugunterseite mit Reflexoptokoppler.



(b) Ultraschall-Abstandssensor.

Abb. 8. Eingebaute Reflexoptokoppler zur Fahrbahndetektion und Ultraschall-Abstandssensor

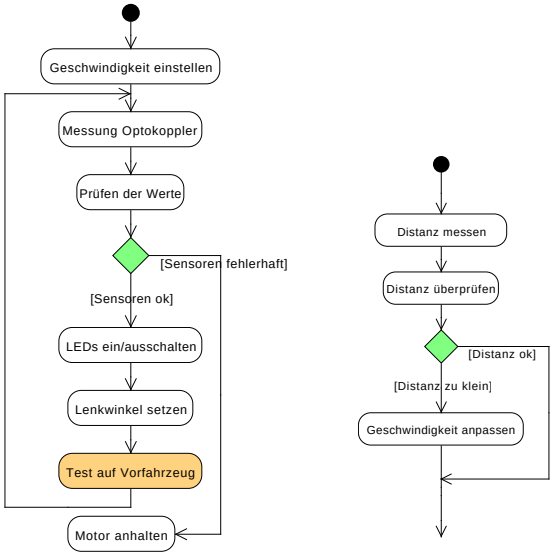
Als optische Ausgabe, ob eine Fahrnbahnmarkierung erkannt wurde oder nicht, dienen die in Abbildung 6 Markierung 3 dargestellten LEDs. Diese sind auf einer Platine angebracht und können über NPN-Transistoren über den Mikrokontroller geschaltet werden. Versorgt werden die LEDs mit $5V$ und sind mit entsprechenden Vorwiderständen ausgestattet, um ein Durchbrennen zu verhindern. Die äußeren LEDs entsprechen den äußeren Optokoppler auf der Platine und die inneren LEDs den zwei Optokoppler in der Mitte.

B. Software

Die entwickelte Software ist auf dem Mikrocontroller in C implementiert und enthält im Wesentlichen zwei Komponenten, welche in Abbildung 9 als Ablaufdiagramm dargestellt sind. Die erste Komponente realisiert das Halten der Spur des Modellautos. Als erstes wird eine Basisgeschwindigkeit eingestellt. Darauffolgend werden die Reflexoptokoppler abgefragt und ihre entsprechenden Helligkeitswerte gespeichert. Diese Werte werden im Anschluss geprüft, sodass ein Defekt der Sensoren ausgeschlossen werden kann. Liegt ein Defekt der Senoren vor, wird der Motor abgeschaltet und das Fahrzeug stoppt. Sind die Werte in Ordnung, so werden als nächstes die LEDs an der Front des Fahrzeugs geschaltet, um eine Auskunft darüber zu liefern, welche Sensoren eine Linie erkannt haben. Je nachdem welche Sensoren die Linie

detektiert haben, wird der Lenkwinkel eingestellt und der Vorgang wird bei der Messung der Optokoppler wiederholt.

Der zweite Teil der Software besteht darin, das Auffahren auf ein Vorfahrzeug zu verhindern. Im Gesamtablauf wird diese Prüfung in dem orange hinterlegten Kästchen in Abbildung 9(a) durchgeführt. Als erstes wird gemäß 9(b) die Distanz zum Vorfahrzeug bestimmt. Beim Überprüfen der Distanz wird entschieden, ob der Abstand groß genug ist oder nicht. Ist der Abstand größer als der Schwellwert von $d_{min} = 50cm$, bleibt die Geschwindigkeit unverändert. Andernfalls wird die Geschwindigkeit nach dem im Fehlermodell beschriebenen Verfahren angepasst. Ist der Vorgang beendet, wird die nächste Messung der Optokoppler durchgeführt.



(a) Ablauf des Spurhalteassistenten.

(b) Ablauf für den Test auf ein Vorfahrzeug.

Abb. 9. Ablaufdiagramm der Softwarekomponenten.

V. Experimente

Der folgende Abschnitt beschreibt einige Experimente, die durchgeführt wurden, um die entwickelten Konzepte zu testen. Die Probleme, die beim Halten der Spur zu einem Versagen des Systems führten, haben ihre Hauptursache in einer zu hoch eingestellten Geschwindigkeit des Fahrzeugs und an zu kleinen Kurvenradien. Tabelle I zeigt eine Übersicht über verschiedene Kurvenradien, die mit verschiedenen Geschwindigkeiten durchfahren wurden. Für jede Konfiguration wurden zehn Fahrten durchgeführt. Die letzte Spalte zeigt, wie viele Fahrten davon erfolgreich durchgeführt wurden ohne dass das System versagte.

Die Ergebnisse zeigen, dass ein Kurvenradius von $r_{min} = 70cm$ bei einer Geschwindigkeit von $v_{max} = 0.7\frac{m}{s}$ nicht durchfahren werden kann. Wird die Geschwindigkeit reduziert, werden deutlich bessere Resultate erzielt, sodass bei $v = 0.3\frac{m}{s}$ 9 von 10 Fahrten erfolgreich verlaufen. Wird die Geschwindigkeit erhöht, so muss auch der Kurvenradius erhöht werden, damit das System nicht versagt. Bei einer Geschwindigkeit von $v = 0.7$ gelingt bei einem Kurvenradius von $r = 100cm$ nur eine Durchfahrt. Die auftretenden Probleme äußeren sich vor allem dadurch, dass das Fahrzeug beginnt um die Fahrbahnmarkierung zu schwingen, wobei die Amplitude schnell so groß wurde, dass die Spur vollständig verloren wurde.

Die Ergebnisse aus der Tabelle geben nur einen minimalen Einblick in die Problematik und haben aufgrund der wenigen betrachteten Radien und Geschwindigkeiten nur geringe Aussagekraft. An dieser Stelle ist eine ausführlichere Untersuchung notwendig.

VI. Fehleranfälligkeitsanalyse

Der folgende Abschnitt gibt einen Überblick über auftretende Fehler, die trotz der Fehlertoleranzmaßnahmen zu einem Versagen des Systems führen. Es werden hierbei die Fehlerarten aus dem Fehlermodell nacheinander betrachtet.

A. Die Fahrbahnmitte wird verlassen

Das Verlassen der Fahrbahnmitte mit dem entsprechenden Fehlerkorrekturmechanismus funktioniert bei kleinen Geschwindigkeiten sehr gut, was sich durch Experimente bestätigt. Erhöht man die Geschwindigkeit des Fahrzeugs fällt deutlich auf, dass das Fahrzeug beginnt um die Fahrbahnmarkierungen zu schwingen, was im schlimmsten Fall zu einem Verlassen der Fahrspur führt. Die Zuverlässigkeit des Spurhalteassistenten nimmt bei steigender Geschwindigkeit deutlich ab. Dazu kommt, dass die Kurvengeometrie bestimmte Vorgaben erfüllen muss. Ein Kurvenradius bei niedrigen Geschwindigkeiten von $r_k = 100cm$ kann gut durchfahren werden. Wird der Kurvenradius kleiner, kommt das Fahrzeug schnell von der Fahrbahn ab oder kann aufgrund der Lenkmechanik den Kurvenradius nicht durchfahren. Der Radius einer Kurve ist in diesem Zusammenhang gemäß Abbildung 10 definiert.

B. Ein Vorfahrzeug erscheint

Das Erkennen eines Vorfahrzeugs und die damit verbundene Geschwindigkeitsanpassung ist ebenfalls nur zu einem gewissen Grad tolerierbar. Wird die Geschwindigkeitsdifferenz zwischen beiden Fahrzeugen zu groß, führt dies dazu, dass das Vorfahrzeug häufig tangiert wird. Der Grund dafür besteht darin, dass das Modellfahrzeug keine Bremsfunktion enthält

Radius [cm]	Geschwindigkeit [$\frac{m}{s}$]	Erfolgreiche Fahrten
70	0.3	10/10
70	0.5	8/10
70	0.7	0/10
80	0.3	10/10
80	0.5	9/10
80	0.7	0/10
100	0.3	10/10
100	0.5	10/10
100	0.7	1/10

Table I
Erfolgreiche Durchfahrten in Abhängigkeit von Kurvenradius und Geschwindigkeit des Fahrzeugs.

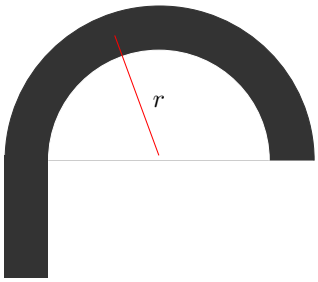


Abb. 10. Definition des Kurvenradius.

und der Korrekturalgorithmus das Fahrzeug im schlimmsten Fall ausrollen lässt.

Ein weiteres Problem in Bezug auf die Ultraschallsensoren, die zur Abstandsmessung eingesetzt werden, besteht darin, dass diese starr auf dem Modellfahrzeug montiert sind. Dies führt dazu, dass in Kurven keine korrekte Erkennung gewährleistet ist. Der Ultraschallsensor misst auf einer Linie direkt vor dem Fahrzeug. Befindet sich dieses nun in einer Kurve, so entgeht dem Ultraschallsensor unter Umständen das vorausfahrende Fahrzeug, da es aufgrund der Kurve nicht direkt vor ihm ist. Je kleiner der Kurvenradius, desto gravierender wird das Problem.

VII. Zusammenfassung

In dieser Arbeit wurde ein fehlertolerantes System am Beispiel eines Modellautos implementiert. Die auftretenden Fehler äußern sich im Verlassen des Mittelpunktes der Fahrbahn und dem nicht Einhalten eines Sollabstandes zu einem Vorfahrzeug. Die Ursachen, die diese Fehler erzeugen, wurden benannt und Gegenmaßnahmen, die zur Tolerierung dieser Fehler bis zu einem gewissen Maß führen, wurden vorgestellt. Die durchgeführten Experimente geben einen Eindruck von der Zuverlässigkeit der eingeführten Konzepte, konnten aufgrund knapper Zeit allerdings nicht exzessiv getestet oder sogar verifiziert werden.

Literatur

[1] Grahs, T. und Sonar, T., *Angewandte Mathematik, Modellbildung und Informatik*, Braunschweig/Wiesbaden: Vieweg+Teubner Verlagg, 2001.
[2] *STM32F4 Discovery Board*, Discovery kit with STM32F407VG MCU, ST Microelectronics, Januar 2016.
[3] Theel O., *Vorlesungsfolien zu Fehlertoleranz in verteilten Systemen*, Wintersemester 2015/16.
[4] *Tymiya Electronic Speed Controller*, TEU-101BK, Tamiya, Januar 2016.
[5] Winner, H. und Hakuli, S. und Wolf, G., *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*, Wiesbaden: Vieweg+Teubner Verlag, 2011.

A fault tolerant traffic routing model

Connor Fibich, Paul Kröger

***Abstract*—In our seminar work, we develop a road network model with four cars. The main goal is that the cars can always reach their randomly determined destination in that network. We use a redundant infrastructure and techniques taken from the field of computer networks to make our system robust against faults. Furthermore, we identify and evaluate failure situations that can occur despite our countermeasures.**

I. Introduction

Computer controlled traffic and autonomous vehicles are nowadays important issues: Railroads use a computerised rail network and several companies are working on autonomous cars. Such systems are obviously very complex. This entails an increase of the risk of making mistakes during development. The result could be a failure of a component while driving. The system environment, which cannot be controlled by the systems themselves, provides additional sources of failures: Falling trees, for example, can damage overhead contact lines or block routes. Thus, it is difficult or impossible to eliminate all errors in advance.

Railroads and autonomous vehicles shall reach their destinations anyway. In addition, the safety of passengers and cargo shall not be compromised in case of failures. There is therefore the need for the systems to be able to compensate for errors at run time.

In our seminar project, we developed a model of an autonomous transportation system: In a model road network, small cars try to reach randomly set destinations in order to deliver imaginary goods. Our goal is that the cars always reach their destination with high level on safety even if some components fail. We identified several error sources and implemented techniques to make our system robust against these errors. Some of these techniques are adopted from the domain of computer networks since we identified some similarities of this domain to our system.

In Section II, we describe our system in detail. Our error model is given in Section III while Section IV presents how we try to handle errors. Section V comprises an error forecast describing failures which still can occur. We complete this seminar paper with a short conclusion in Section **??**.

II. System description

We developed a model of a transportation system comprising a road network, shown in Figure 1, and a set of robot vehicles. Each robot vehicle shall be able to reach an arbitrary destination in the road network. Possible destinations are defined in the following.

The road network is build from a set of square modules. This allows to create different networks. Wires are embedded into the surfaces of the modules. The cars use a magnet to detect these wires. They thereby can follow the wires when moving forward. In other words, the wires form the lanes of our road network. This technology is taken from the Faller Car System [1]. Each road has only one lane which is open in both directions.

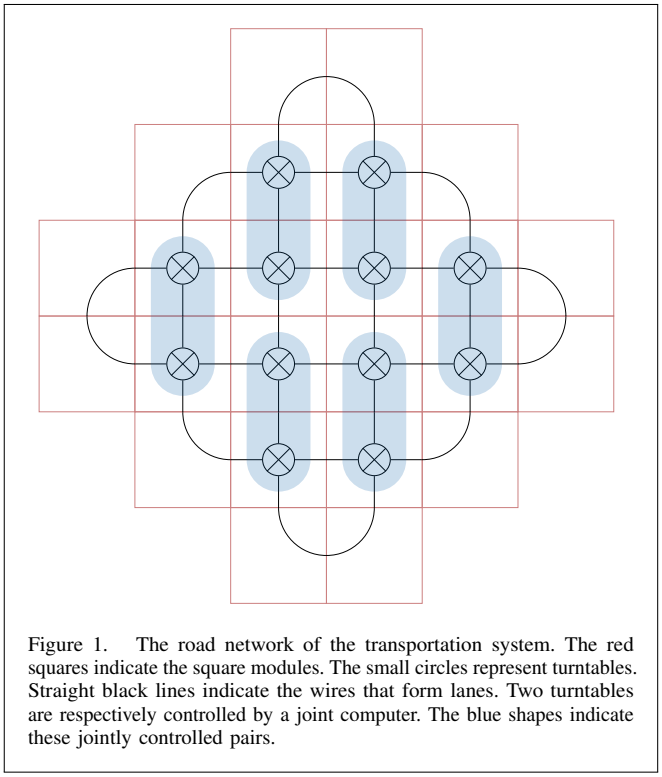


Figure 1. The road network of the transportation system. The red squares indicate the square modules. The small circles represent turntables. Straight black lines indicate the wires that form lanes. Two turntables are respectively controlled by a joint computer. The blue shapes indicate these jointly controlled pairs.

We constructed two kinds of modules: Modules with an intersection, and modules without an intersection. We decided that all intersection modules shall provide intersections with four junctions. Furthermore, all junctions of an intersection shall allow cars to turn in each junction, even the junction the car came from[1]. Intersections of the original Faller Car System are build with a kind of gate for the wire. Using these gates to build intersections with the desired properties would lead to a complex construction. In addition, the turning radius of our cars is relatively large. A complex intersection would therefore require too much space for our purpose.

For this reason, we constructed intersections as turntables that turn cars in the direction in which they have to go. The

[1] We employed the original axles of the Faller Car System to build our robot cars. With this steering technique it is not possible to drive backwards. We therefore need to turn the car in the junction it came from if it has to go back.

turntables are turned by stepper motors. We selected motors that are shipped pairwise and with a control board that can be mounted on the well known Raspberry Pi [2]. Each control board can control at most two motors. Our turntables are thus controlled in pairs.

There is one lane from the mid of each border of an intersection module to the turntable. Modules without intersections provide a curve from the mid of a border to the mid of an adjacent border. With these modules, we can construct road networks with straight lanes, quarter-, half- or three-quarter circles as curves, and intersections with at most four junctions.

The robot cars are equipped with a radio connection and a reed contact. The reed contact detects magnets beside the lanes. Since the cars have fixed initial positions, this allows to localise them. The radio connection is used for communication with a traffic control center.

The cars try to reach randomly chosen destinations. Possible destinations are the turntables. Choosing a destination requires knowledge about available destinations as well as planning routes requires knowledge about available paths. Our modular design allows to arrange several road networks. However, we did not implement an automatic detection of the actual setup. The user has to provide the required information about available destinations and paths in a configuration file.

This configuration file is given to a central computer, called the traffic control center. This traffic control center chooses the destinations of the cars. Furthermore, it calculates the routes and determines whether a car is allowed to enter a segment or not. With this structure, we do not have to distribute information about available paths or destinations or taken lanes or turntables over several computers or cars.

III. Error model

A. Possible impairments

We identified several errors that can occur in our system and lead to a partial or total failure. In the following, we specify these errors.

Deadlock: Assume there are two cars and a lane that is part of each path between the current positions of the cars and their destination. Furthermore, assume the two cars are positioned on the two turntables at the ends of that lane in order to pass it. If no car can yield, a deadlock arises.

Failure of a car: A car can fail. A possible reason for such a failure could be be an empty battery or a mechanical problem. In this case, the car will permanently stop its operation.

Car reboot: The controller boards of the cars are equipped with a watchdog that reboots the car if a problem is detected. Furthermore, the car is rebooted to re-establish the radio connection between the car and the traffic control center if that connection was interrupted.

Faulty rotation: The turntables are made of wooden panels. Changes in humidity cause a change in the shape of the turntables. This can increase the friction when turning such that the rotation cannot be completed. Since our stepper motors are open-loop controlled, this friction might lead to a loss of steps. There are also some manufacturing tolerances in mounting of the turntable on the motor axle. Both, the manufacturing tolerances and lost steps can cause an inaccurate positioning of a turntable after a turn. An offset of the wire at the transition between turntable and module can be the result.

Failure of traffic control center: The computer serving as traffic control center or the traffic control software can fail.

Failure of Raspberry Pi: A Raspberry Pi that controls stepper motors can also fail due to some reason.

Other electrical failures: Several other components can fail, e. g. network devices that connect the Raspberry Pis and the traffic control center or the general power supply.

Faulty initial turntable position: Since the cars are steered by following a wire, it is crucial that the position of the wires in the turntable align with the wires in the module surface. An inaccurate starting position of the turntable can cause a misalignment of the wires. This error can be caused by careless placing of the turntables on the motor spindle or mechanical influences during transport.

B. Functional consequences

We identified some possible faults and errors. In general, these impairments can result in situations in which cars cannot reach their destination. In this section, we divide the faults and errors into three groups of resulting misfunctions.

Permanently blocked segment: When a deadlock occurs, the involved cars are permanently stopped. Thus, they cannot reach their destination. The segments that are currently occupied by the involved cars can not be passed by other cars. Depending on the algorithm that prevents cars from colliding, two lanes, two turntables, or one lane and one turntable can be blocked. Blocked turntables are not reachable. In addition, depending on the collision avoidance algorithm, cars may enter a lane that leads to a blocked turntable. Cars approaching a blocked turntable are not able to pass that turntable, effectively blocking the occupied lane. Depending on the structure of the road network, a blocked lane or turntable reduces the number of available paths in the road network. This can lead to further deadlock situations. If the blocked segment is part of the only connection between two sub-networks A and B, destinations in B cannot be reached from A and vice versa.

The most of the other identified impairments have the same consequences as a deadlock: The segment owned by a failed car is blocked and the car cannot reach its destination. A failure of the traffic control center or the occurrence of another electrical failure will also let the car wait at the end of the current segment: If for instance a central network component like a router or switch fails, no message can be transmitted between any system component. If a single connection to a Raspberry Pi fails, the turntable will not get a command that it has to turn nor it can send a message that the turn is complete and the car can drive on.

Arbitrary trajectory: A faulty rotation and a faulty initial turntable position can both result in an arbitrary trajectory of the car: If the position of the wires in the turntable and the wires in the module are not aligned, the car might lose the track. A car that lost the track moves in an unpredictable trajectory until it runs out of space of the module surface or it finds another track. Driving without a track can lead to collisions. If the

car finds another track, it drives without a correct positioning and can also collide with other cars. If the car does not detect another magnet, the segment assigned to the car can be blocked permanently. In any case, the car will probably not reach its destination.

Note that this situation occurs almost only when a car leaves a turntable: The wires of the turntables are layed straight and through the center of the turntable. Furthermore, the cars approach the turntable on a straight line that passes the center of the turntable. When a car enters a turntable, it therefore runs over the gap between turntable and module surface with both wheels of an axle at the same time. Thus, the steering of the car is not impaired by passing this gap. The car will continue moving straight ahead, approaching the center of the turntable and regains contact to the wire. Passing a wire that is almost transverse to the driving direction has no influence on the steering. The car therefore follows the wire that has almost the same direction than the car. This should be the correct wire.

The only case in which the car can not find the correct wire occurs if the center of the turntable is misplaced but the wire to follow is parallel to the moving direction of the car. In this case, the magnet of the car is either strong enough to find the wire anyway, or the steering is without any influence and the car still moves straight ahead. The latter case is not a problem if the car has not to be turned. If the car has to be turned, the situation mentioned above can occur: The car does not meet the wire of the modules surface after leaving a turntable.

Another source of an arbitrary trajectory is the reboot of a car: In this case, the car loses its information about its position, destination and the next turntable. The initial values are restored. The car might request routing information from the wrong routing table.

C. Focus of our error handling

The handling of all identified faults and errors is beyond our means in this seminar. Arbitrary trajectories can cause physical damage. Thus, we focused first on faults leading to this misfunction.

The computers and network devices we employed for our system seem to be quite reliable. Furthermore, we assume that the software we developed is also quite reliable. Then, there are three remaining faults that can block segments: deadlock situations, failures of cars, and car reboots. Focussing on these errors seems to make the most sense.

IV. Error handling

In the previous section, we identified some errors and the resulting misfunctions some errors and the resulting misfunctions. Furthermore, we named some of these errors we concentrated on. This section describes how we handled these errors.

A. Arbitrary trajectories

Arbitrary trajectories can be a result of faulty rotations or a bad initial position of the turntable. We therefore have to turn the open-loop control of the turntables into a closed-loop control. This could be achieved by using a rotary encoder giving accurate positional feedback. Such an encoder needs to be placed at the axles. This would increase the complexity of the modules. Each turntable is already equipped with four magnets that are placed in a square with an equal distance to the rotation axis. These magnets could be used to provide the needed positional feedback. A reed contact is mounted on the substructure of the turntable such that the turntable is in the correct position when a magnet is positioned above the contact. The turntable rotates until the correct position is reached.

Magnet fields have a certain range. With some experiments we were able to determine the difference between the turntables position when the contact triggers and the target position.

In cases of the turntables temporarily being stuck, the solution above is accurate enough to compensate a faulty rotation. In cases where the turntable is permanently stuck, the stepping motor is stopped when no magnet passes the reed contact within a certain amount of time. Furthermore, all lanes leading to that turntable are marked as blocked such that other cars are forced to drive around that turntable.

The bad initial position of the turntable is solved using the rotation technique above: After the system is started, each turntable rotates until it reaches a valid position.

In order to prevent arbitrary trajectories arising from a car reboot, the current position, the destination and the next turntable are requested from the traffic control center when a connection is re-established.

B. Blocked segments

We identified several faults leading to blocked segments and listed three of them to focus on. We implemented two key concepts to address these problems: We added redundancy to our road network and we used a technique to determine the routes of our cars through the network that respects blocked routes dynamically.

We saw some similarities between our road network and computer networks: Turntables can be seen as network routers, lanes as network connections and robot cars can represent data packets. Due to these similarities, we decided to adopt some internet routing techniques [3].

The general procedure is as follows: When the traffic control center has chosen a destination, it is send to the corresponding car. Furthermore, the car gets the information which turntable will be the next on its path. Now, the car starts approaching its destination. When it reaches a turntable, it sends a message to that turntable requesting access. Once access is granted, it moves up to the center of the turntable. Now the cars sends its destination to the turntable which then checks its routing table. The routing table holds the information which neighbouring turntable is the next on the shortest path to every possible destination. This information is used to rotate the car to the correct exit of the turntable.

The routing tables are computed immediately after the system is started. For this purpose, we used the widely known Dijkstra algorithm, first published in [4] and shown in Algorithm 1. Given a graph G and a source node s, this algorithm computes for all nodes n of the graph the shortest path from s to n. This

path can determined by following the predecessor fields from the destination to the source.

Algorithm 1 Dijkstra algorithm

```
procedure DIJKSTRA(Graph G, source node s)

    for all nodes n of G do
        dist[n] ← ∞            ▷ Initial distance to node n
        pred[n] ← NULL▷ Initial predecessor on path to n
    end for

    dist[s] ← 0
    U ← set of nodes of G

    while Q ≠ ∅ do
        x ← u ∈ U where ∀u' ∈ U : dist[u] ≤ dist[u']
        Q ← Q \ {u}
        for all neighbours y of x do
            ℓ ← length of edge (x, y)
            if ℓ + dist[x] < dist[y] then
                dist[y] ← ℓ
                pred[y] ← x
            end if
        end for
    end while

end procedure
```

The routing tables are computed by the traffic control center and send to the corresponding Raspberry Pis. The are updated with respect to the current traffic situation if a car A tries to enter a lane that is currently marked as owned by a car B and one of the following conditions is fulfilled:

- The direction of car A is not the same as the direction of car B. Car B will be marked as a reason for updating the routing tables.
- A is waited longer than a certain amount of time. In this case, it has to be assumed that B failed.

When the routing tables are updated, the car A follows the new path.

If a car marked as a reason for updating the routing tables leaves a segment, the marking will be removed and the routing tables are updated again.

This procedure allows in general to make use of alternative routes if a lane is blocked and avoids trivial deadlock situations arising from two oncoming cars.

Deadlock: Our main concept against deadlocks is redundancy. Adding as many routes as possible to our road network increases the probability that an alternative route is available and can be chosen by the routing algorithm. Figure 1 depicts our road network. Each turntable is connected to four lanes. Since each car can block at most one lane, a deadlock situation arising from two oncoming cars can not occur if there are at most four cars.

The routing algorithm itself can provoke a deadlock-like situation: Assume a car A needs to take an alternative route because of an oncoming car B. On its alternative route, there can be another oncoming car C. Taking another alternative route, possible but not necessarily the first one, there can be again an oncoming car. This can be car B or a car D. With unfortunate chosen targets and more than two cars, it is possible that A cannot reach its destination.

Destinations, however, are chosen by random experiments. Thus, it is expected that the probability of the occurrence of this situation converges to zero for long-term runs of the system. Note that it is not possible that two cars can block each other by oncoming and choosing alternative routes: When the routing tables are re-computed, one of the two cars is located on a turntable and the other one is located on a lane adjacent to that turntable. The first car follows an alternative route while the latter one just continues following its path.

Failure of a car: We were not able to implement techniques that prevent a car from failing if a component of the car fails. If a car A fails, it blocks a lane or a turntable. Assume a car B waits for A to leave the currently owned segment. As described above, if B waits longer than a certain amount of time, A is considered as being failed and the routing tables are updated. If A blocks a turntable, the lanes connected to that turntable are also marked as blocked in order to prevent other cars from driving into lanes that cannot be left.

Aside from the failed car and cars already waiting for the grant to enter a turntable with a failed car, the routing procedure ensures together with the redundant layout of the road network that other cars can still reach their destination if not too many segments are blocked.

A critical situation arises, when three or more turntables are blocked by failed cars. Then, it is possible that the road network is divided into two not connected sub-networks. Then, it is possible that a destination is not reachable. A similar situation can arise with two blocked turntables and two blocked lanes. If there are not more than four cars on the road network, no running car can be prevented from reaching its destination – all cars are failed yet. Besides the fact that permanent blocked turntables are trivially unreachable, we were not able to construct a case with less than three failed cars in which additional turntables became unreachable.

Since we have four cars, we cannot completely prevent that a situation can arise where a fully functional car cannot reach its destination. In order to ensure that this car at least can partially fulfil its function, unreachable destinations are discarded. Thus, the car can at least drive to turntables the reachable subnet.

V. Error forecast and conclusion

As we already discussed before, the structural redundancy allows four cars to use the road network without deadlock classical deadlock situations. If we confine our system to at most four cars, no fully functional car is prevented from reaching its destination as long as not three turntables are blocked and no fully functional car is waiting for the grant to enter such a blocked turntable.

We did not handle several possible impairments. The failure of the traffic control center or a central network device would bring the system to a halt: No car will receive any grant message and thus wait forever at the end of the current segment.

If a Raspberry Pi fails, the controlled turntables do not operate any longer. Cars could enter these turntables but will never leave them. The turntables will be marked as blocked. Thus, the cars on the turntables and the connected lanes could be blocked. Both, the failure of a Raspberry Pi and the traffic control center could handled by introducing additional redundant computers or decentralising.

We handled the most relevant error of our system. The systems function cannot be fulfilled in some cases. However, the remaining system failures seem to be unlikely.

REFERENCES

[1] “Faller Car System,” http://www.car-system-digital.de, accessed: 2016-01-20.

[2] “Raspberry Pi,” http://www.raspberrypi.org, accessed: 2016-01-20.

[3] S. Halabi and D. McPherson, *Internet Routing Architectures, Second Edition*, 2nd ed. Cisco Systems, 2000.

[4] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390