Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Alexander Hoffmann

Visuelle Navigation für ein Autonomes System mit minimalen Hardwareanforderungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Hoffmann

Visuelle Navigation für ein Autonomes System mit minimalen Hardwareanforderungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 21. März 2019

Alexander Hoffmann

Thema der Arbeit

Visuelle Navigation für ein Autonomes System mit minimalen Hardwareanforderungen

Stichworte

Autonome Navigation, Visuelle Navigation, Raspberry Pi, OpenCV, Bildverarbeitung, WiringPi

Kurzzusammenfassung

Ob für Drohnen, Serviceroboter, Robotik in der Logistik oder autonome Autos im Straßenverkehr, überall wo ein System sich in einer Umgebung orientieren oder einen Weg finden muss, ist eine zuverlässige Navigation erforderlich. Insbesondere an neuen Orten, wo noch keine Infrastruktur aufgebaut ist, wie zum Beispiel durch Navigationssatelliten, ist es erforderlich, dass ein mobiles System sich zurechtfindet. Die visuelle Navigation ist eine Lösung sich in einem Gebäude, in einem Lager oder im Freien zu orientieren. Sie kann den autonomen Fahrzeugen als Unterstützung dienen oder aber auch bei kleinen Plattformen vollständig die Navigation übernehmen. Durch den Ansatz der inkrementellen Positionsbestimmung, wo die aktuelle Position aufgrund einer alten Position bestimmt wird, ist es möglich die visuelle Navigation auf Mikrocontroller mit schwacher Leistung unterzubringen. Dagegen ist die Bestimmung der Position mit Hilfe von generierten Karten der Umgebung sehr rechenintensive. Dieses Dokument beschäftigt sich mit dem Aufbau, Entwicklung und Analyse eines autonomen Systems mit visueller Navigation und möglichst kleinem Hardwareaufwand. Dafür wird der Ansatz mit der inkrementellen Positionsbestimmung verwendet, um ressourcensparende und billige Mikrocontroller verwenden zu können.

Alexander Hoffmann

Title of the paper

Visual navigation for an autonomous system with minimal hardware requirements

Keywords

autonomous navigation, visual navigation, Raspberry Pi, OpenCV, digital image processing, WiringPi

Abstract

Whether for drones, service robots, robotics in logistics or autonomous cars in traffic, wherever a system needs to orient itself in an environment or find a way, reliable navigation is required. In particular, in new places where no infrastructure is built, such as navigation satellites, it is necessary for a mobile system to find its way. Visual navigation is a solution for orienting yourself in a building, in a warehouse or outdoors. It can serve the autonomous vehicles as support or completely take over the navigation for small platforms. Through the incremental positioning approach, where the current position is determined by an old position, it is possible to accommodate visual navigation on low power microcontrollers. In contrast, determining the position using generated maps of the environment is very computationally intensive. This document deals with the construction, development and analysis of an autonomous system with visual navigation and the least amount of hardware. The incremental positioning approach is used to leverage resource-saving and cheap microcontrollers.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Einführung

Die visuelle Navigation wird verwendet, um die Position, Ausrichtung oder Fahrverhalten eines mobilen Systems zu bestimmen [vgl. 11]. Alle folgenden Aussagen beziehen sich auf den Artikel Visuelle Navigation [vgl. 11]. Die visuelle Navigation basiert rein auf der Auswertung von Kameradaten und ist unabhängig von externen Infrastrukturen. Es gibt zwei grundlegende Ansätze die visuelle Navigation zu realisieren. Die eine Möglichkeit ist das sogenannte Visual Simultaneous Localization and Mapping, kurz Visual SLAM, Verfahren und beruht auf die Bestimmung der eigenen Position innerhalb einer aus den Kamerabildern generierten Karte. Diese Möglichkeit ist sehr rechenaufwändig. Ein weniger rechenaufwändiger Ansatz heißt inkrementelle Positionsbestimmung, bei der die aktuelle Position in Bezug auf die vorausgegangene Position bestimmt und somit die Bewegungsrichtung und Lage ermittelt wird. Es gibt eine Vielzahl an Einsatzmöglichkeiten der visuellen Navigation von Drohen, Serviceroboter, Robotik in der Logistik bis hin zur Unterstützung von autonomen Autos. In den meisten Fällen wird nie eine rein visuelle Navigation eingesetzt.

Diese Bachelorarbeit stützt sich auf die inkrementelle Positionsbestimmung anhand von Landmarken. Bei den Landmarken handelt es sich um Kreise mit verschiedenen Farben. Die Navigation erfolgt durch das Fahren der Plattform von einem Kreis zum nächsten. Der Vorteil dieser Navigation ist es, dass sie extrem wenig Rechenaufwand benötigt, da es weniger komplex ist einen Kreis zu finden als die Objekte in der Umgebung zu erkennen und sie als Landmarken zu identifizieren. Auch in Bezug auf die Veränderung der Fahrumgebung ist dieses Verfahren besser als das übliche Folgen einer Linie, da es einfacher ist die Landmarken umzustellen als eine Linie neu zu legen. Die Kombination aus Kreiserkennung und das Erkennen der Farbe im Kreis macht es möglich die Kreise eindeutig als Landmarken zu identifizieren und reduziert deutlich das Erkennen von falschen Kreisen.

1.2 Ziel der Arbeit

Das Ziel dieser Bachelorarbeit ist es, ein autonomes System zu entwickeln, beruhen auf der visuellen Navigation, und dieses System dann zu analysieren und zu bewerten. Das autonome System soll mit einfachen Mitteln realisiert werden, basierend auf der Auswertung von Kameradaten. Die Anzahl der verwendeten Hardware soll möglichst klein gehalten werden und die Art der Hardware soll klein und ressourcensparend sein. Durch die fortlaufende Analyse parallel zu der Entwicklung des Systems, soll ausgewertet werden, welche Funktionen und Erweiterung eingebaut werden sollen, um das System zu verbessern. Die endgültige Analyse nach der Fertigstellung des Systems soll die geeigneten Parameter für die zuvor erstellten Funktionen festlegen und das Verhalten des Systems beschreiben.

1.3 Struktur der Arbeit

Diese Arbeit ist wie folgt aufgebaut: Kapitel 2 sind Grundlagen, in dem dem Leser die grundlegenden Hardware- und Softwarekomponenten beschrieben werden. Bei dem Kapitel 3 Autonome System handelt es sich um den Hauptteil dieser Bachelorarbeit. Dort wird das gesamte System beschrieben, die einzelnen Softwarekomponente und deren Zusammenspiel, außerdem auch die nötigen mathematischen Berechnungen. In Kapitel 4 Tests und Analyse werden die durchgeführten Tests, deren Ergebnisse und Beurteilung dieser Ergebnisse beschrieben. Zum Schluss kommt Fazit und Ausblick in Kapitel 5. Dort werden die ganzen Analysen zusammengefasst, weitere Optimierungen vorgeschlagen und ein Ausblick auf die möglichen Erweiterungen des System gegeben.

2 Grundlagen

In dem Kapitel Grundlagen werden die grundlegenden Hardware- und Softwarekomponenten vorgestellt, die bei diesem autonomen System zum Einsatz kommen.

2.1 Raspberry Pi Zero WH

Der Raspberry Pi Zero WH ist ein Einplatinencomputer der britischen Raspberry Pi Foundation und ist eine Erweiterung des Vorgängers Raspberry Pi Zero [vgl. 6]. Er ist $31,2mm$ breit, $65,0mm$ lang und hat eine Höhe von $5mm$. Er unterstützt WLAN und Bluetooth 4.0 und hat schon eine Stiftleiste. Das Herzstück ist ein ARM-Prozessor mit einem Takt von $1000MHz$. GPU-Takt beträgt $400Mhz$. Die Stromaufnahme ist von $0,5$ bis $0,7$ Watt. Er hat zwei Micro USB Ports, von denen einer ausschließlich für die Stromversorgung gedacht ist, außerdem Micro SD Kartenleser und ein HDMI Mini Anschluss.

2.2 Kamera

Bei der eingesetzten Kamera handelt es sich um den Raspberry Pi Kameramodul V2 mit 8 Megapixel [vgl. 3]. Sie hat eine CSI-2 Busschnittstelle und misst $23,86 \times 25 \times 9mm$. Die maximal unterstützte Auflösung beträgt $3280 \times 2464px$ und die maximale Bildfrequenz der Aufnahmen ist auf $30fps$ ausgelegt. Der Bildwinkel beträgt $62,2°$ horizontal und $48,8°$ vertikal.

2.3 Plattform

Bei der Plattform handelt es sich um einen Multi Chassis Tank (Rescue Platform) von DAGU [vgl. 4].

2.3.1 Motoren

Bei den Motoren handelt es sich um $48 : 1$ DC mini Gearbox Motoren DG02S von DAGU [vgl. 1]. Die Versorgungsspannung eines einzelnen Motors beträgt 3 bis 6 Volt. Leerlaufstrom ist

bei $125mA$ angelegt und die maximale Stromversorgung beträgt $170mA$. Die Motoren haben eine Drehgeschwindigkeit von $65 \pm 10rpm$ und der maximale Drehmoment ist $0,078Nm$.

2.3.2 Encoder

Die Encoder gehören zu dem Wheel Encoder Kit von DAGU [vgl. 7]. Jeder Encoder besteht aus einem 8-poligen Neodym-Magneten mit Gummi-Naben und einem Hall-Effekt-Sensor. Sie können mit einer Versorgungsspannung von 3 bis 24 Volt betrieben werden und brauchen $4mA$ an Strom. Durch den 8-poligen Magneten können 8 Schritte pro Umdrehung gezählt werden.

2.4 Motortreiber

Bei dem Motortreiber handelt es sich um einen Dual DC Motor Treiber L9110 [vgl. 2]. Die Eingangspannung beträgt $2,5$ bis 12 Volt und der Eingangsstrom ist auf $800mA$ ausgelegt. Die Maße sind $23 \times 30mm$.

2.5 OpenCV

$OpenCV$ ist eine für jeden frei zugängliche Bibliothek, die für die Bildverarbeitung und maschinelles Sehen konzipiert wurde [vgl. 5]. Die Programmbibliothek erschien erstmal in Juni 2000 und unterstützt die Programmiersprachen C, C++, Python und Java. Sie zeichnet sich durch ihre Geschwindigkeit und der Menge an Funktionalitäten aus. Die Funktionalitäten beinhalten zum Beispiel Gesichtserkennung und verschiedene effiziente Filter wie Sobel oder Gauß. $OpenCV$ wird in den Bereichen wie Objekterkennung, Gestenerkennung, Tracking, mobile Roboter und viele andere Bereiche, wo Bildverarbeitung ein wesentlicher Bestandteil des System darstellt, eingesetzt. Die verwendeten Hauptfunktionalitäten aus dieser Bibliothek in dieser Bachelorarbeit sind zum einen die Kreiserkennung und zum anderen die Farbfilter [vgl. 12]

2.6 WiringPi

$WiringPi$ ist ein extra für den Rapberry Pi erstelltes Programm, das es ermöglicht auf die GPIOs des Mikrocontrollers zuzugreifen und diese auch zu kontrollieren [vgl. 8]. Die Programmbibliothek von $wiringPi$ kann an die Programmiersprachen C, C++, Python, Java und PHP angebunden werden. Einige der Funktionen dieser Bibliothek können das Setzten eines

Signals an den Pins sein, generieren eines PWM Signals, sowohl von der Software als auch von der Hardware Seite, sofern der Minirechner das unterstützt, oder das Auslesen, ob ein Signal an den Pins anliegt. Auch ist I^2C und SPI ein Teil dieser Bibliothek. Diese Bibliothek wurde in der Arbeit verwendet, um über PWM-Signale die Motoren anzusteuern [vgl. 10] und um die Interrupts von den Encodern abzufangen [vgl. 9].

3 Autonome System

In diesem Kapitel werden das gesamte autonome System dargestellt, alle Softwarekomponente und deren Zusammenspiel. Insbesondere wird der Augenmerk auf die Bahnberechnung gelegt, deren Analyse und die dafür notwendigen mathematischen Formeln.

3.1 Gesamtüberblick

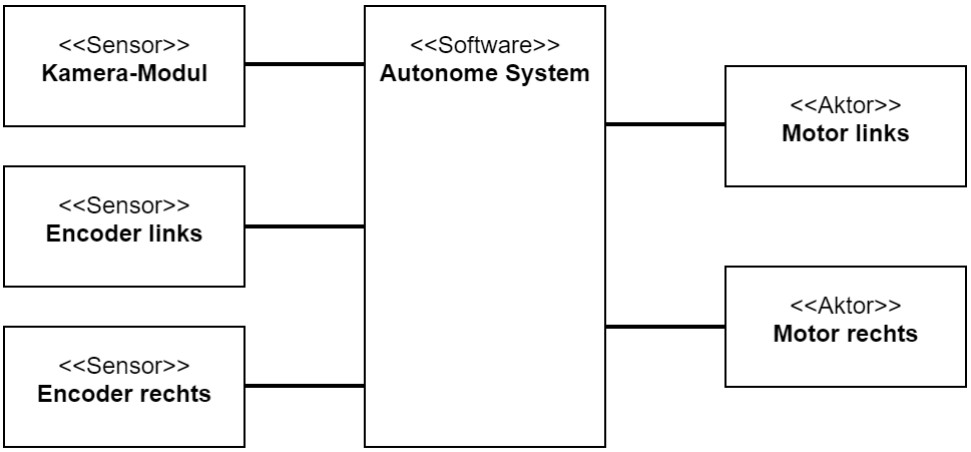


Abbildung 3.1: Übersicht über das autonome System

Die Abbildung 3.1 zeigt die Übersicht über das autonome System. Links sind die Sensoren $Kamera\text{-}Modul$, $Encoder\ links$ und $Encoder\ rechts$ zu sehen und rechts sind die beiden Motoren dargestellt $Motor\ links$ und $Motor\ rechts$.

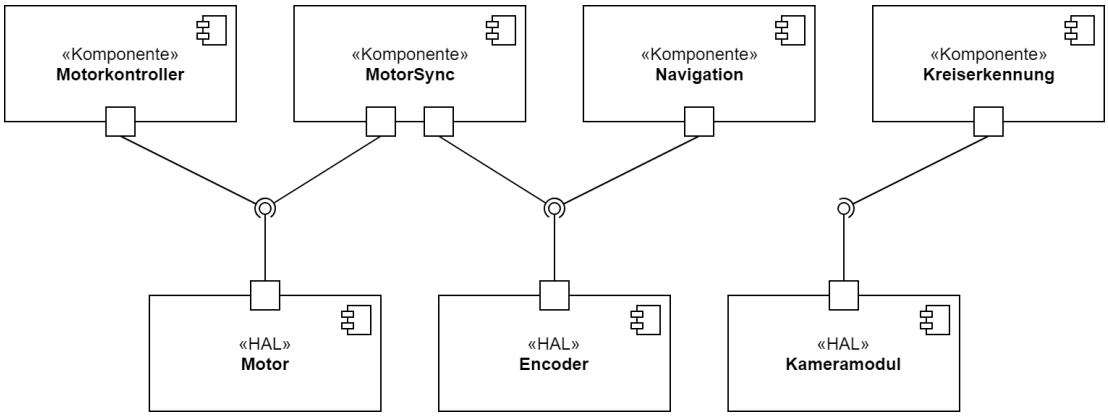


Abbildung 3.2: Die Software des autonomen Systems

Wie in Abbildung 3.2 gezeigt, besteht die Software des autonomen Systems aus sechs Hauptkomponenten: Motorkontroller, Motorsynchronisator, Navigation, Kreiserkennung und die HAL für da Kameramodul, die Motoren und die Encoder. Der Motorkontroller hat den Zugriff auf die HAL der Motoren und steuert die Plattform. Der Motorsynchronisator ist dazu da, um den rechten und den linken Motor aneinander anzupassen, sodass die beiden Motor die gleiche Umdrehungsgeschwindigkeit haben. Die Kreiserkennung hat den Zugriff auf die Kamera HAL, durch die sie die Bilder bekommt und verarbeitet. Sie sucht in den Bildern nach Kreisen mit ausgewählter Farbe und sendet die Position vom ersten Kreis an die Navigation. Die Navigation bekommt die Position eines Kreises und sorgt dafür, dass die Plattform zu diesem Kreis bis zu einem bestimmten Abstand fährt, dafür sendet sie Nachrichten an den Motorkontroller. Durch die HAL hat das System Zugriff auf die Frames, auf die Werte von den Encodern und auf die Steuerung der beiden Motoren.

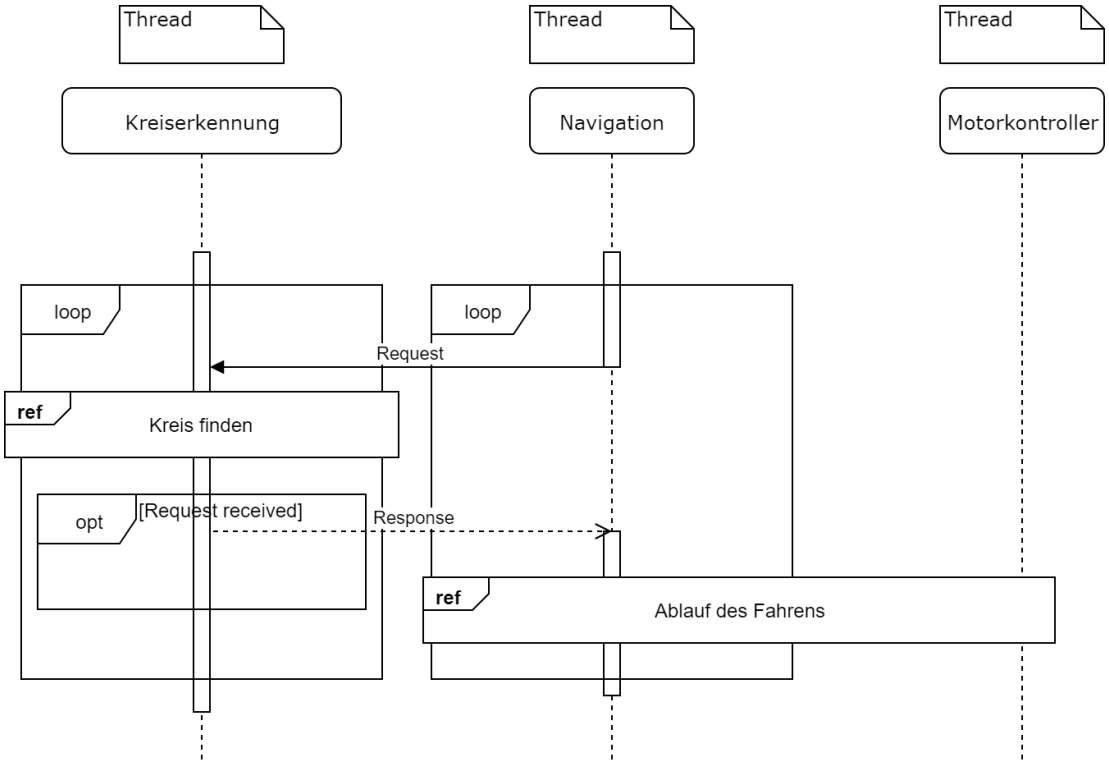


Abbildung 3.3: Kommunikation zwischen den Komponenten

Die Abbildung 3.3 zeigt den Ablauf des Nachrichtenaustausches von dem gesamten System. Die Navigation sendet eine Anfrage and die Kreiserkennung und bekommt eine Antwort zurück. In der Antwort steht die Position des Kreises, wenn er gefunden wurde. Dieser Nachrichtenaustausch ist synchron. Nach der Antwort findet eine Kommunikation zwischen der Navigation und dem Motorkontroller statt. Der Ablauf des Fahrens wird in Abschnitt 3.4 beschrieben. Der detaillierte Kommunikationsverkehr, wird in den einzelnen Kapiteln zu den Komponenten erklärt.

3.2 Motorkontroller

Der Motorkontroller bekommt Nachrichten von der Navigation. In diesen Nachrichten steht, ob er nach vorn, nach hinten fahren, nach links, nach rechts drehen oder stoppen soll. Außerdem wird noch übermittelt mit welcher Geschwindigkeit er fahren soll. Die Geschwindigkeit wird von 0% bis 100% angegeben. Der Motorkontroller steuert indirekt die Motoren, indem er die Werte für die Geschwindigkeit und das Fahrverhalten festlegt. Das Fahrverhalten ist das

Fahren nach vorn, nach hinten, das Drehen nach link, nach rechts oder das Stoppen. Dieser Verhalten wird im Zustandsdiagramm in Unterabschnitt 3.2.2 beschrieben.

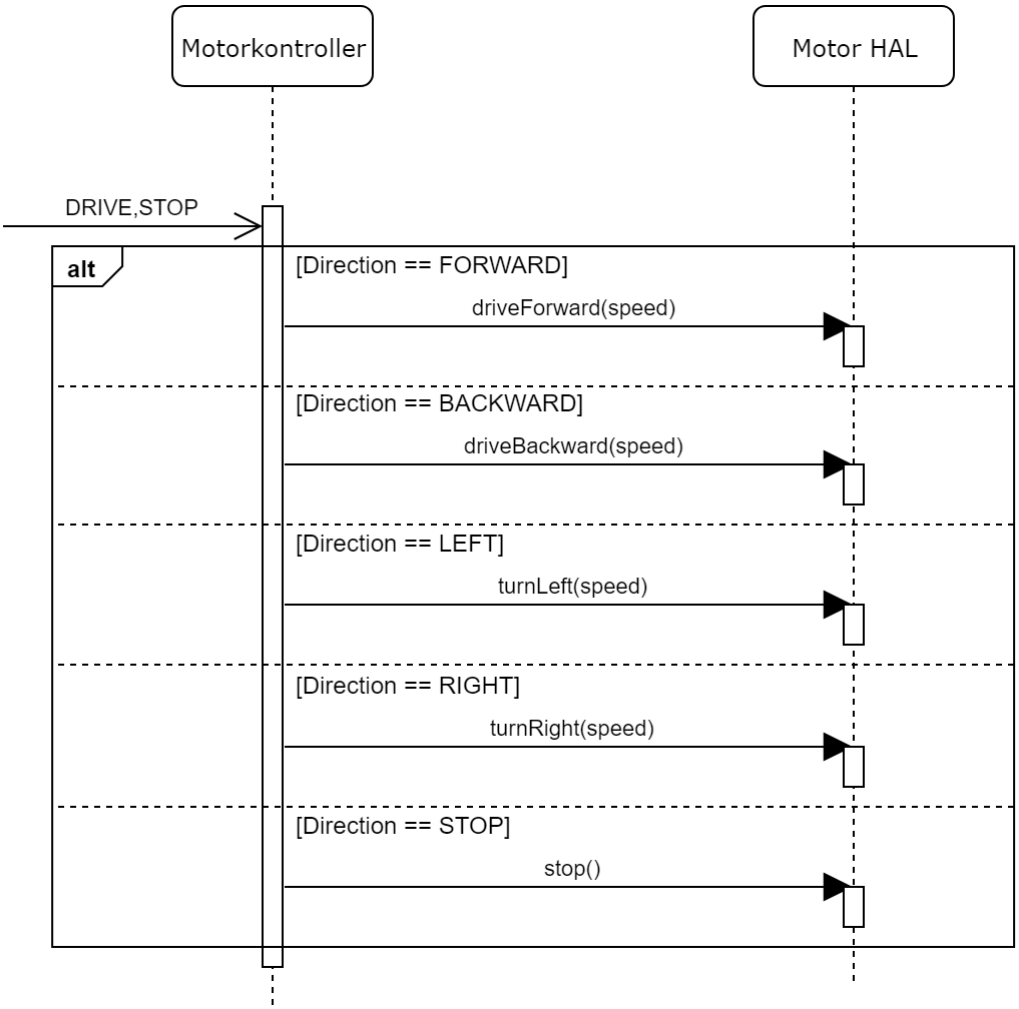


Abbildung 3.4: Ablauf im Motorkontroller

Die Abbildung 3.4 zeigt das möglicher Verhalten der Plattform. Der Motorkontroller bekommt die Nachrichten $DRIVE$ und dann $STOP$, wobei gilt:

$$DRIVE \in \{FORWARD, BACKWARD, LEFT, RIGHT, STOP\}$$

Zu dem Verhalten bekommt er auch gleichzeitig die Geschwindigkeit von 0% bis 100% übermittelt. Daraufhin ruft er die entsprechende Funktion, wie zum Beispiel $driveForward(speed)$, aus der Motor HAL auf.

3.2.1 Motorsynchronisator

An den Motoren sind Encoder angebaut. Die Interrupts, die von den Encoder ausgelöst werden, werden im Programm verarbeitet. Die Geschwindigkeit, die an den Motorkontroller von der Navigation übergeben wird, wird von ihm in ein Intervall zwischen zwei Interrupt umgerechnet. Ist der tatsächliche Zeitabschnitt zwischen zwei Interrupts kleiner, wird weniger Gas gegeben, ist er größer, wird mehr Gas gegeben. Dadurch sorgt der Motorsynchronisator, dass die beiden Motoren die gleiche Drehzahl haben.

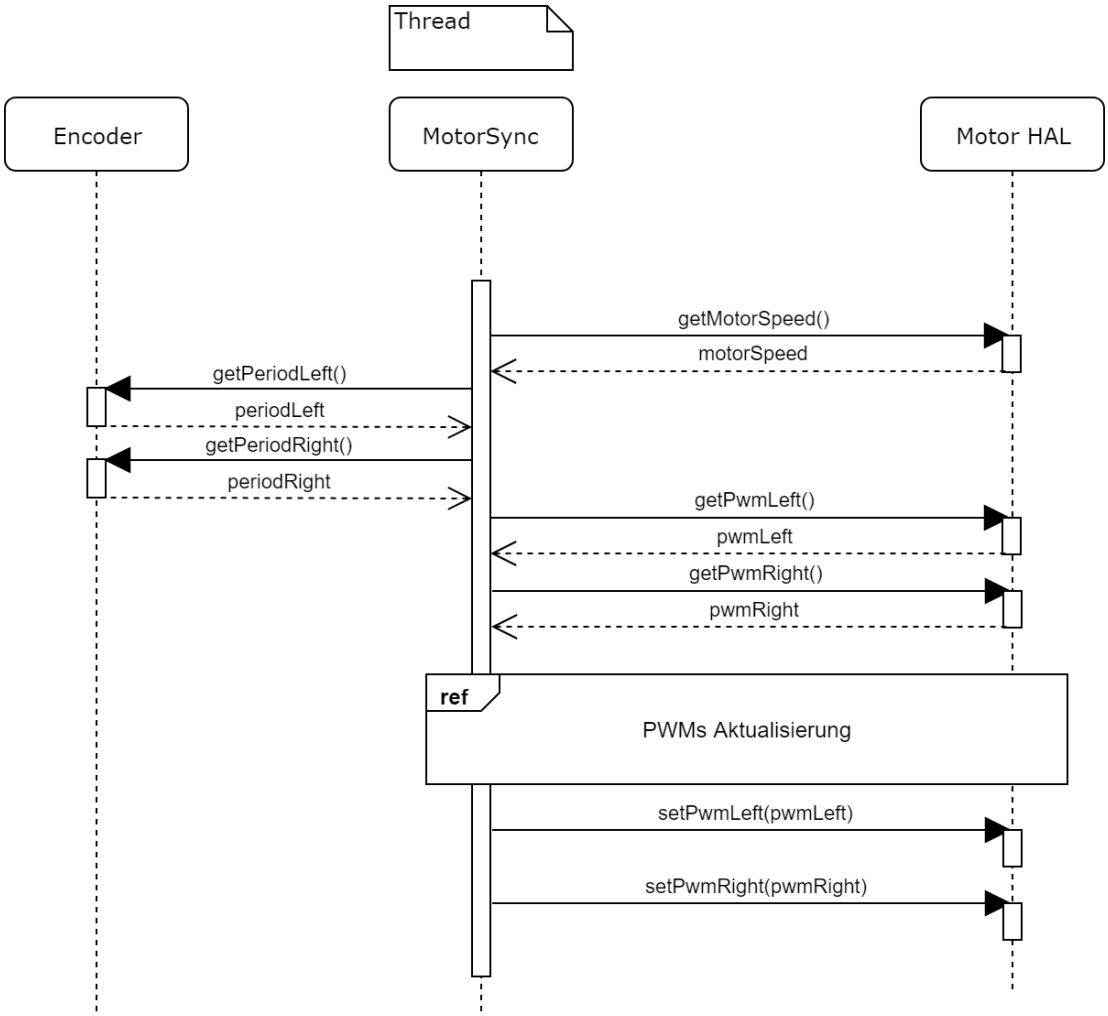


Abbildung 3.5: Ablauf des Motorsynchronisators

Die Abbildung 3.5 zeigt die Aufgabe des Motorsynchronisators. Zuerst holt er sich die vom Motorkontroller festgelegte Geschwindigkeit $motorSpeed$. Diese Geschwindigkeit entspricht

dem gewünschten Zeitintervall zwischen zwei Interrupts bei den Encodern. Danach holt er sich die tatsächlichen Zeitintervalle vom linken und rechten Encoder. Zum Schluss nimmt der Motorsynchronisator die aktuellen festgelegten PWM-Signale für den linken und rechten Motor, aktualisiert sie, entsprechend der gewünschten und tatsächlichen Periodenbreite und setzt diese wieder in der Motor HAL.

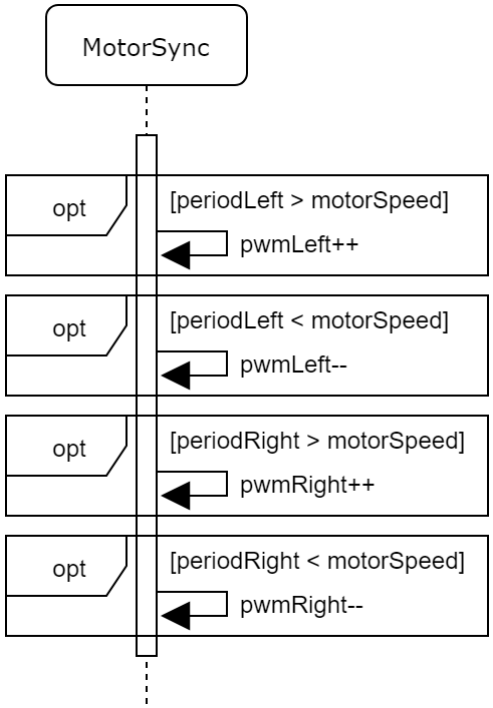


Abbildung 3.6: PWMs Aktualisierung

Die Abbildung 3.6 zeigt, wie der Motorsynchronisator die PWMs aktualisiert. Er vergrößert bzw. verkleinert den Wert der PWM bis der tatsächlicher Zeitintervall aus den Encodern dem gewünschten Zeitintervall $motorSpeed$ entspricht. Das macht er für die beiden Motoren.

3.2.2 Motor HAL

Die Motor HAL stellt die Möglichkeit die Motoren zu steuert bereit. Sie gibt die Funktionen für das Drehen der Motoren in eine und in die anderer Richtung. Die HAL erzeugt an den Pins vom Raspberry Pi PWM Signale. Je breiter der PWM Signal, desto schneller dreht sich der Motor. Für den Zugriff auf die Pins benutzt die HAL die WiringPi Bibliothek und darin die Funktion der Software PWM. Durch den Motorkontroller berechnet die HAL die gewünschte

Geschwindigkeit in Millisekunden und der Motorsynchronisator sorgt dann dafür, dass die Motoren auf diese Geschwindigkeit gebracht werden.

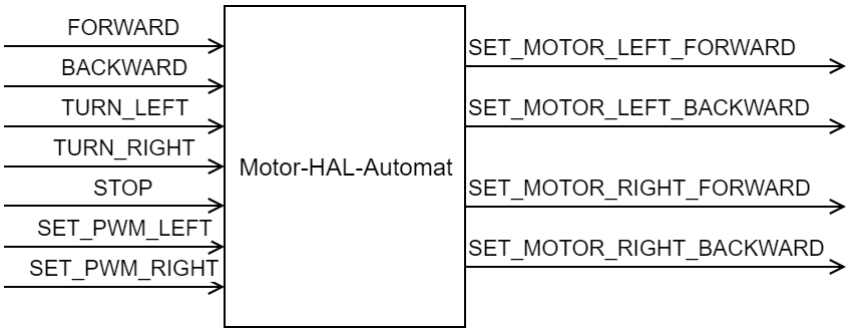


Abbildung 3.7: Eingehenden und ausgehenden Signale des Motor-HAL-Automaten

Die Abbildung 3.7 zeigt die eingehenden und ausgehenden Signale des Automaten in der HAL für die Motoren. Die Signale *FORWARD*, *BACKWARD*, *TURN_LEFT*, *TURN_RIGHT* und *STOP* mit der Geschwindigkeit in Prozent bekommt der Automat von dem Motorkontroller und die Signale *SET_PWM_LEFT* und *SET_PWM_RIGHT* mit dem Wert der PWM bekommt er von dem Motorsynchronisator. Die ausgehenden Signale *SET_MOTOR_LEFT_FORWARD*, *SET_MOTOR_LEFT_BACKWARD*, *SET_MOTOR_RIGHT_FORWARD* und *SET_MOTOR_RIGHT_BACKWARD* bewirken, dass sich der jeweilige Motor mit zuvor übergebenden PWM in eine oder in die andere Richtung dreht.

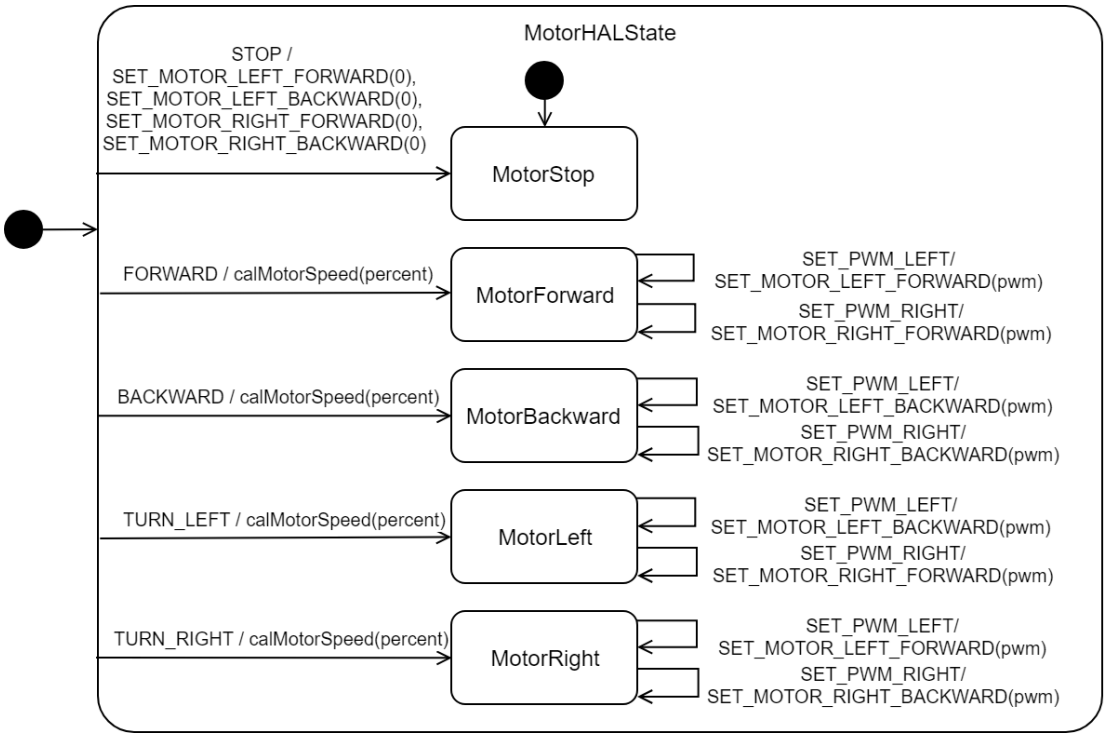


Abbildung 3.8: Motor-HAL-Automat

Die Abbildung 3.8 zeigt den Motor-HAL-Automaten. Nur durch die Signale von dem Motorkontroller kann der Automat seinen Zustand wechseln. Zusätzlich wenn die Plattform fahren soll, wird die Geschwindigkeit berechnet, wie die folgende Formel beschreibt.

$$p* = \begin{cases} 0 & \text{für } p < 0 \\ p & \text{für } 0 \leq p \leq P_{MAX} \\ P_{MAX} & \text{für } P_{MAX} < p \end{cases}$$

$$s = \frac{S_{MAX} \cdot P_{MAX}}{p*}$$

Die Variable p ist die Geschwindigkeit in Prozent. Sie wird überprüft, ob sie zwischen 0% und 100% liegt. Die Variablen P_{MAX} steht für 100% und S_{MAX} ist die maximale Periodendauer zwischen zwei Interrupts des Encoders und wurde auf $20ms$ festgelegt. Die gewünschte Geschwindigkeit s, auch eine Periodendauer, ist umgekehrt proportional zu der Geschwindigkeit in Prozent. Je höher der Prozentwert, desto kleiner ist die Periodendauer und desto schneller dreht sich der Motor.

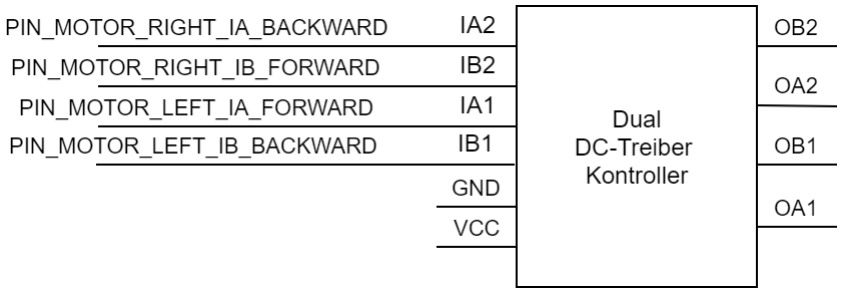


Abbildung 3.9: Die Pins des Motortreibers

Die Abbildung 3.9 zeigt, dass die Motor HAL, für die Steuerung der Motoren, vier Pins des Motortreibers verwendet. Zwei ausgehende Pins für den linken Motor *PIN_MOTOR_LEFT_IA_FORWARD*, *PIN_MOTOR_LEFT_IB_BACKWARD* sind an $IA1$, $IB1$ angeschlossen und zwei ausgehende Pins für den rechten Motor *PIN_MOTOR_RIGHT_IA_BACKWARD*, *PIN_MOTOR_RIGHT_IB_FORWARD* sind an $IA2$, $IB2$ angeschlossen. Die folgenden Zeilen beschreiben das Verhalten der Motoren für alle möglichen Kombinationen an Signalen, die an den Pins $IA1$, $IB1$, $IA2$ und $IB2$ anliegen.

$$
\begin{aligned}
IA1 = 0 \quad &\wedge \quad IB1 = 0 \quad &&\Rightarrow Der\ linke\ Motor\ stoppt \\
IA1 = 0 \quad &\wedge \quad IB1 = PWM \quad &&\Rightarrow Der\ linke\ Motor\ fährt\ rückwärts \\
IA1 = PWM \quad &\wedge \quad IB1 = 0 \quad &&\Rightarrow Der\ linke\ Motor\ fährt\ vorwärts \\
IA1 = PWM \quad &\wedge \quad IB1 = PWM \quad &&\Rightarrow Der\ linke\ Motor\ stoppt
\end{aligned}
$$

$$
\begin{aligned}
IA2 = 0 \quad &\wedge \quad IB2 = 0 \quad &&\Rightarrow Der\ rechte\ Motor\ stoppt \\
IA2 = 0 \quad &\wedge \quad IB2 = PWM \quad &&\Rightarrow Der\ rechte\ Motor\ fährt\ vorwärts \\
IA2 = PWM \quad &\wedge \quad IB2 = 0 \quad &&\Rightarrow Der\ rechte\ Motor\ fährt\ rückwärts \\
IA2 = PWM \quad &\wedge \quad IB2 = PWM \quad &&\Rightarrow Der\ rechte\ Motor\ stoppt
\end{aligned}
$$

Da die Motoren gespiegelt in der Plattform eingebaut sind, muss sich der eine Motor in Uhrzeigersinn und der andere gegen den Uhrzeigersinn drehen, damit die Plattform vorwärts bzw. rückwärts fährt. Deshalb muss auch zum Beispiel bei dem linken Motor an Pin $IA1$ und bei dem rechten Motor an Pin $IB2$ ein PWM-Signal anliegen und die anderen beiden Null sein, damit sich die Plattform vorwärts bewegt.

$IA1$	$IB1$	$IA2$	$IB2$	Verhalten
0	0	0	0	Die Plattform stoppt
PWM	0	0	PWM	Die Plattform fährt vorwärts
0	PWM	PWM	0	Die Plattform fährt rückwärts
PWM	0	PWM	0	Die Plattform dreht sich in Uhrzeigersinn
0	PWM	0	PWM	Die Plattform dreht sich gegen den Uhrzeigersinn

Tabelle 3.1: Implementierter Plattformverhalten in der Motor HAL

Die Tabelle 3.1 zeigt das Verhalten der Plattform, das in der HAL vom Motor realisiert wurde.

3.2.3 Encoder HAL

Es werden zwei Interrupt Service Routinen für zwei an den beiden Motoren angebauten Encoder konfiguriert. Sowohl bei einer fallenden, als auch bei einer steigenden Flanke wird eine Interrupt geworfen. Tritt ein Interrupt auf wird ein Zeitintervall abgespeichert, der sich aus dem aktuellen Zeitstempel minus dem letzten Zeitstempel berechnen lässt. Der letzte Zeitstempel wird dann aktualisiert für den nächsten Interrupt. Die folgende Formel beschreiben dieses Verhalten.

$$s = T_{NOW} - T_{LAST}$$
$$T_{LAST} = T_{NOW}$$

Die Variable s ist der Zeitintervall und T ist ein Zeitstempel. Für den Fall, dass sich der Rad nicht dreht und deshalb s nicht aktualisiert wird, muss beim Abrufen von s überprüft werden, ob die Differenz zwischen dem letzten und dem neuen Zeitstempel größer ist als der Zeitintervall s. In diesem Fall wird diese Differenz zurückgeliefert, wie die folgenden Zeilen verdeutlichen.

$$s_{new} = T_{NOW} - T_{LAST}$$
$$s* = \begin{cases} s_{new} & \text{für } s < s_{new} \\ s & \text{für } s_{new} \leq s \end{cases}$$

Dieses Zeitintervall wird von dem Motorsynchronisator verwendet und gilt als die Geschwindigkeit. Zusätzlich wird bei jedem Interrupt der für den jeweiligen Encoder gehöriger Zähler inkrementiert. Diese Zähler können zurückgesetzt werden und werden benutzt um festzustellen, wie viele Umdrehungen ein Motor durchgeführt hat. Bei einer vollständigen Drehung

eines Rades werden acht Interrupts geworfen. Die Geschwindigkeiten und die Zähler können dann über die entsprechenden Methoden ausgelesen werden.

3.3 Kreiserkennung

Die Kreiserkennung benutzt die Funktionen von der Bibliothek $OpenCV$, um die Aufnahmen, die sie von der HAL bekommt, zu bearbeiten. Zuerst sucht sie Kreise im ganzen Bild, dafür macht sie das Bild grau und wendet ein Gauß-Filter an, um das Rauschen zu glätten. Nachdem die Kreise gefunden wurden, schaut sie nach bei welchem Kreis im inneren blaue ggf. rote Farbe ist. Dieser Kreis ist dann der gesuchte Kreis. Dann sendet die Kreiserkennung die Informationen über den Kreis, also Position und Radius, an die Navigation. Davor sendet die Navigation der Kreiserkennung, dass diese Information gesendet werden können. Die Navigation gibt auch an, in welchem Fall eine Nachricht gesendet wird, zum Beispiel wenn ein Kreis gefunden wurde, wenn kein Kreis gefunden wurden oder beides. Wenn die Kreiserkennung einen Kreis gefunden hat, sucht sie beim nächsten Bild in dem Bereich, wo sie ihn zuletzt gefunden hat. Dieser Region Of Interest, kurz ROI, dehnt sich dynamisch in die Richtung, in die sich der Kreis bewegt, damit bei der nächsten Suche der Kreis den ROI möglichst nicht verlässt. Hat sie den Kreis verloren, sucht sie wieder die ganze Bild ab.

3.3.1 Kameramodul HAL

In der Initialisierung wird die Größe der Frames festgelegt. Die $OpenCV$ bieten zwei Methoden um ein Frame rauszuholen, eine für das Greifen $grab$ und eine für das Abrufen $retrieve$ des Frames. Das Programm greift den nächsten in der Warteschlange wartenden Frame und dann wird durch das Abrufen des Frames, er dem Programm zur Verfügung gestellt. Da es immer nur der nächste in dem Puffer gespeicherter Frame genommen und der aktuelle Frame gebraucht wird, wird zuerst eine bestimmte Anzahl an Frames aus dem Puffer herausgegriffen und dann der letzte zurückgeliefert. Dies hat zur Folge, dass immer das letzte aufgenommene Bild herausgenommen wird, auf den dann die Operationen ausgeführt werden. Bei den Frames handelt es sich um Farbbilder in einem RGB-Farbschema.

3.3.2 ROI

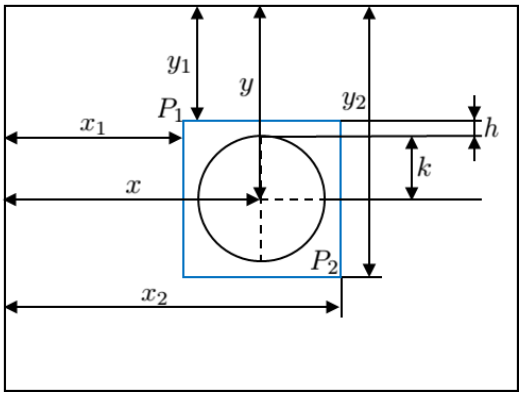
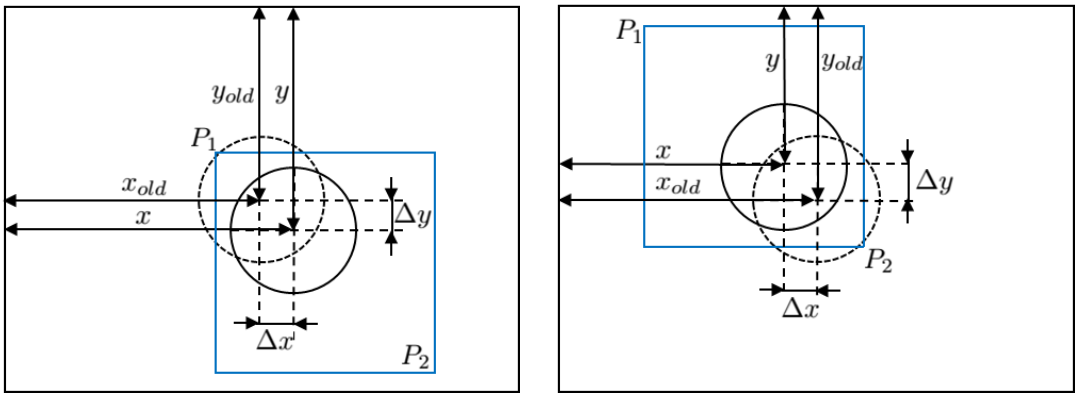


Abbildung 3.10: Region of Interest (ROI) nach dem Finden eines Kreises

Die Abbildung 3.10 zeigt den ROI nach dem ein Kreis gefunden wurde. Wurde noch kein Kreis gefunden, entspricht der ROI der Größe des Frames. Alle Operationen für das Aufspüren eines Kreisen werden nur in dem ROI durchgeführt. Die Variablen x_1, y_1, x_2 und y_2 bilden die Punkte P_1 und P_2, die den ROI aufspannen. Es gibt eine Funktion, bei der ein Frame als Input, ein Frame als Output, und die zwei Punkte übergeben werden. Diese Funktion schneidet den Bereich, den die zwei Punkte umspannen, aus dem Input-Frame heraus und überschreibt damit den Output-Frame. Dieser Output-Frame wird dann weiterverarbeitet. Zur Berechnung von $P_1(x_1, y_2)$ und $P_2(x_2, y_2)$ wird die Position des Kreises x, y, der Radius k und ein Offset h berücksichtigt.

$$x_1 = x - k - h$$
$$y_1 = y - k - h$$
$$x_2 = x + k + h$$
$$y_2 = y + k + j$$

Die oberen Berechnungen zeigen die Situation, wenn der Kreis zum ersten Mal gefunden wurde. Wenn der Kreis schon gefunden wurden und sich bewegt, werden zu den Berechnungen noch die Position des Kreisen, an der er sich zuvor befand, berücksichtigt.



(a) Bewegung nach rechts unten

(b) Bewegung nach links oben

Abbildung 3.11: Dynamische Erweiterung des ROI in die Bewegungsrichtung

Die Abbildung 3.11 zeigt, wie der ROI sich dynamisch in die Richtung erweitert, in die sich der Kreis bewegt. Für den Fall, dass sich der Kreis ungefähr konstant bewegt, sollte durch die dynamische ROI Erweiterung sichergestellt werden, dass der Kreis bei dem nächsten Frame die ROI nicht verlässt.

$$x_1 = \begin{cases} x - k - h + \Delta x & \text{für } x < x_{old} \\ x - k - h & \text{für } x_{old} \leq x \end{cases} \tag{3.1}$$

$$y_1 = \begin{cases} y - k - h + \Delta y & \text{für } y < y_{old} \\ y - k - h & \text{für } y_{old} \leq y \end{cases} \tag{3.2}$$

$$x_2 = \begin{cases} x + k + h & \text{für } x < x_{old} \\ x + k + h + \Delta x & \text{für } x_{old} \leq x \end{cases} \tag{3.3}$$

$$y_2 = \begin{cases} y + k + h & \text{für } y < y_{old} \\ y + k + h + \Delta y & \text{für } y_{old} \leq y \end{cases} \tag{3.4}$$

Bewegt sich der Kreis nach links, dann gilt $x < x_{old}$ für Gleichungen (3.1) und (3.3), deshalb erweitert sich der ROI zusätzlich um Δx nach links für P_1. Bewegt sich der Kreis nach recht, dann erweitert sich der ROI nach rechts zusätzlich um Δx für P_2. War der Kreis vorher unten, erweitert sich der ROI nach oben für P_1 um Δy, weil für die Gleichungen (3.2) und (3.4)

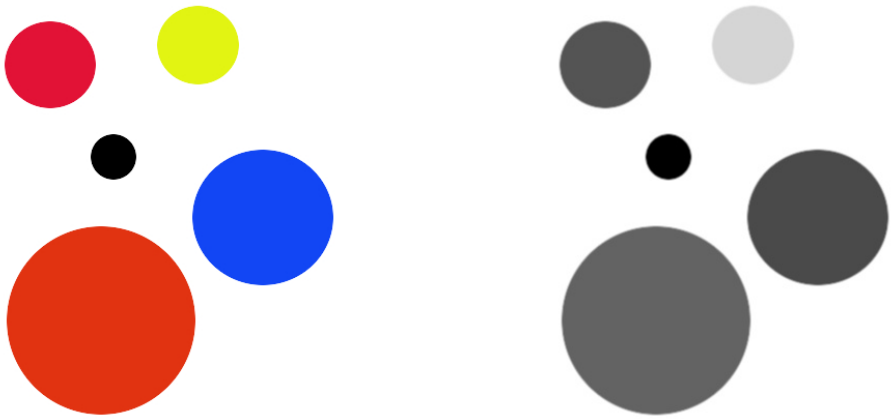
$y < y_{old}$ gilt. War der Kreis vorher oben, erweitert sich der ROI nach unten für P_2 um Δy. Für die Berechnung von Δx und Δy wird folgende Formeln benutzt.

$$\Delta x = x - x_{old}$$
$$\Delta y = y - y_{old}$$

Zusätzlich wird berücksichtigt, ob x_1, y_1, x_2 und y_2 sich innerhalb des Frames befinden, um nicht auf Pixel zuzugreifen, die gar nicht existieren. Ist x_1 oder x_2 kleine als Null wird x_1 oder x_2 auf Null gesetzt. Sind sie größer als die Framebreite, werden sie auf Framebreite minus eins gesetzt, was der letzte Index in x-Richtung darstellt. Das Gleiche gilt für y_1 und y_2, wobei hier die Framehöhe genommen wird und nicht die Framebreite. Einen Region Of Interests zu definieren spart Zeit, die die Operationen benötigen, um einen Kreis zu finden, weil nicht im ganzen Frame gesucht wird.

3.3.3 Finde Kreise

Um einen Kreis mit einer bestimmten Farbe zu finden (siehe Abb. 3.3), müssen eine Reihe an Filterungen und Operationen durchgeführt werden. Im Folgendem werden diese Operationen die Reihe nach aufgelistet. Alle Operationen stammen aus der $OpenCV$ Bibliothek.



(a) Original (b) Nach Konvertierung und Gauß-Filter

Quelle: https://solarianprogrammer.com/2015/05/08/detect-red-circles-image-using-opencv/

Abbildung 3.12: Konvertierung zu Schwarz-Weiß-Bild mit Glättung

Wie in Abbildung 3.12 gezeigt wird, wird der aufgenommener Frame (siehe Abb. 3.12a) in ein Schwarz-Weiß-Bild umgewandelt und um das Rauschen aus diesem umgewandelten Frame rauszubekommen, wird ein Gauß-Filter auf diesen angewandt. Das resultierende Bild ist in Abbildung 3.12b dargestellt. Im folgenden Codeabschnitt sind die beiden Funktionen für die Abbildung 3.12 aufgezeigt.

```
1 cv::cvtColor(src, gray, cv::COLOR_BGR2GRAY);
2 cv::GaussianBlur(gray, gray, cv::Size(3, 3), 0, 0);
```

Die Variablen src und $gray$ sind von Type $cv :: Mat$, wobei src der aufgenommene Frame ist. Die erste Funktion bekommen von links nach rechts den Input-Array, Output-Array und den Code, der besagt welches Farbschema zu welchem Farbschema konvertiert werden muss. Die zweite Funktion bekommt auch den Input- und Output-Array und dann die Größe der Faltungsmatrix, mit der Breite und Höhe. Die beiden hinteren Werte stehen für die Standartabweichungen in x- und y-Richtung.

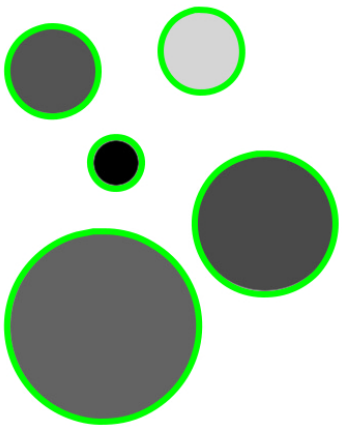
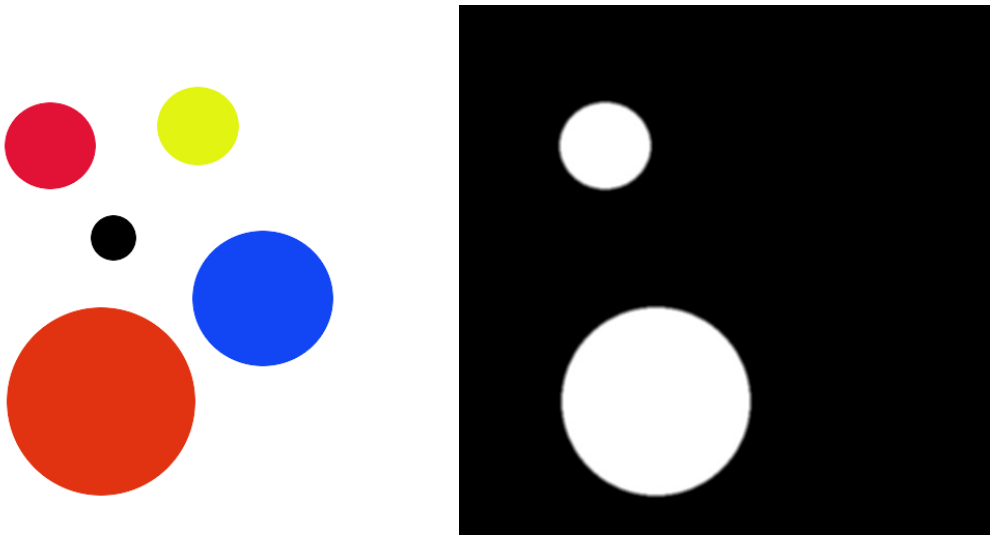


Abbildung 3.13: Gefundene Kreise in dem Schwarz-Weiß-Bild

Die Abbildung 3.13 zeigt die gefundenen Kreise in dem Frame aus Abbildung 3.12b. Um die Kreise zu finden, wird die Hough-Transformation für Kreise aus der $OpenCV$ Bibliothek verwendet, wie folgender Codeabschnitt zeigt.

```
1 std::vector<cv::Vec3f> circles;
2 cv::HoughCircles(gray, circles, CV_HOUGH_GRADIENT, 1, gray.rows / 8, 150,
      50, MIN_RADIUS, MAX_RADIUS);
```

Die zweite Variable $circles$ ist eine Liste von Drei-Elementen-Arrays $cv :: Vec3f$. In diese Liste werden die gefundenen Kreise abgespeichert. Die Elemente der Arrays sind Fließkommazahlen, von denen die ersten beiden die Position des Kreises in dem Frame, in x- und y-Richtung, beschreiben und der dritte den Radius des Kreises darstellt. Alle Elemente haben die Einheit Pixel. Die Variablen MIN_RADIUS und MAX_RADIUS beschreiben den Bereich, in dem der Radius eines Kreise liegen muss, um gefunden zu werde. Unter MIN_RADIUS und über MAX_RADIUS sucht die Funktion keine Kreise. Die Variable $CV\_HOUGH\_GRADIENT$ gibt die Methode der Kreiserkennung an, die die Kreise anhand der Neigungen der Kanten identifiziert. Der vierte Parameter in der Funktion beschreibt das umgekehrter Verhältnis der Auflösung der sogenannten Akkumulatormatrix, die von der Funktion erstellt wird, zum Frame. In dieser Matrix werden die möglichen Kreismittelpunkte von allen gefundenen Kanten eingetragen bzw. aufsummiert, wenn die Mittelpunkte sich überlagern. Bei den Stellen in der Matrix, die einen gewissen Wert übersteigen, handelt es sich mit einer gewissen Wahrscheinlichkeit um die Position des Mittelpunktes eines Kreises. Der fünfte Parameter gibt den minimalen Abstand, in diesem Fall $gray.rows/8$, der zwischen den Mittelpunkten der Kreise liegen muss. Die möglichen Mittelpunkte, die unter diesem Abstand zu einem gefundenen Kreismittelpunk liegen, werden ignoriert. Das Argument 150 an der sechsten Stelle wird von dem internen Canny-Algorithmus benutzt. Dieser Algorithmus findet Kanten und das Argument ist der maximale Schwellenwert für diese Kanten. Der minimaler Schwellenwert ist halb so groß. Der siebte Parameter ist der Schwellenwert für die Kreismittelpunkte in der Akkumulatormatrix, die dann als Kreismittelpunkte gezählt werden. Da nach einem Kreis mit einer bestimmten Farbe gesucht wird, muss noch in dem ursprünglichen Frame nach dieser Farbe gefiltert werden.



(a) Original (b) Gefiltert nach roter Farbe

Abbildung 3.14: Filterung nach roter Farbe

Die Abbildung 3.14 zeigt das Ergebnis des Farbfilters. In diesem Fall wird nach der roten Farbe gefiltert (siehe Abb. 3.14b). Der folgende Codeabschnitt zeigt die dazugehörige Funktion.

```
1  cv::Mat hsv_image;
2  cv::cvtColor(bgr_image, hsv_image, cv::COLOR_BGR2HSV);
3
4  cv::Mat lower_red_hue_range;
5  cv::Mat upper_red_hue_range;
6  cv::inRange(hsv_image, cv::Scalar(H_MIN_RED_L, S_MIN_RED_L, V_MIN_RED_L), cv
       ::Scalar(H_MAX_RED_L, S_MAX_RED_L, V_MAX_RED_L), lower_red_hue_range);
7  cv::inRange(hsv_image, cv::Scalar(H_MIN_RED_U, S_MIN_RED_U, V_MIN_RED_U), cv
       ::Scalar(H_MAX_RED_U, S_MAX_RED_U, V_MAX_RED_U), upper_red_hue_range);
8
9  cv::Mat red_hue_image;
10 cv::addWeighted(lower_red_hue_range, 1.0, upper_red_hue_range, 1.0, 0.0, dst
       );
```

Zuerst wird der unbearbeitete Frame von RGB-Farbraum in HSV-Farbraum umgewandelt, wie in der Zeile 2 dargestellt. Der HSV-Farbraum beschreibt die Pixel wie folgt: Das H steht für *hue* und gibt den Farbwert eines Pixels in Grad an, dabei steht 0° für Rot, 120° für Grün und 240° für Blau. Die Farben haben einen bestimmten Bereich innerhalb der 360°, der diesen Farben zugeordnet ist. Im Falle der roten Farbe liegen die Werte ungefähr im Bereiche von 0° bis 20° und im Bereich von 320° bis 360°. Werden breitere Bereiche gewählt, werden Pixel, die nicht zu der roten Farbe gehören fälschlicherweise als diese erkannt. Werden schmalere

Bereiche gewählt, werden Pixel, die doch zu der roten Farbe zählen, nicht als diese erkannt. Das S im HSV-Farbraum steht für $saturation$ und bedeutet die Sättigung des Pixels und somit den Grau-Anteil. 0% bedeutet der Pixel ist neutralgrau und bei 100% Sättigung hat man die reine Farbe. Der letzte Wert in HSV ist $value$ und beschreibt den Hellwert mit 0% für dunkel und 100% für volle Helligkeit. Nach dem Umwandeln des Frames in HSV Schema kann eine entsprechende Funktion, wie in Zeile 6 oder 7, den Frame-Array durchgehen und die Pixel, bei denen die HSV Werte in einem ausgewählten Bereich liegen, ausfindig machen. Dafür werden zwei Argumente von Typ $cv :: Scalar$ übergeben. Das sind Vektoren aus vier Elementen. In den ersten drei Elementen stehen H, S und V, wie zuvor beschrieben, und der letzte ist automatisch Null. Der erste Vektor ist der untere Schwellenwert und der zweite der obere. Diese Funktion liefert ein neues schwarzes Frame zurück mit weißen Stellen, an denen die Pixel innerhalb der Schwellenwerte lagen. Da die rote Farbe bei dem HSV-Farbraum in zwei Bereichen, innerhalb der $360°$, liegt, am Anfang und am Ende, werden zwei Funktionen $cv :: inRange$ aufgerufen und zwei Frames zurückgeliefert. Die beiden entstandenen Frames werden dann, wie in Zeile 10, zu einem Frame zusammengefasst (siehe Abb. 3.14b).

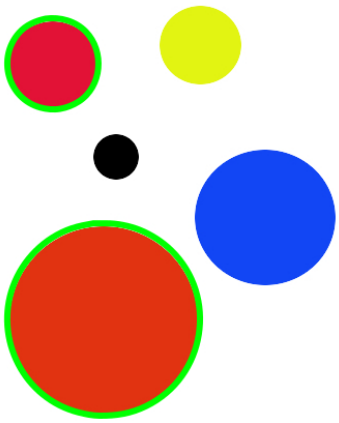


Abbildung 3.15: Gefundenen roten Kreise

Um jetzt die roten Kreise zu identifizieren, muss durch alle gefundenen Kreise iteriert werden. Für jeden Kreis wird geschaut, ob die Position ihres Mittelpunktes, in dem Frame aus der Abbildung 3.14b, im weißen Bereich liegt. Liegt der Mittelpunkt im weißen Bereich, wird dieser Kreis als ein roter Kreis angenommen. Die Abbildung 3.15 veranschaulicht die gefundenen roten Kreise.

3.4 Navigation

Die Navigation ist das Herzstück des Systems, die darüber entscheidet, wie die Plattform agiert. Bei der Navigation gibt es zwei Programme, die man auswählen kann. Eines sorgt dafür, dass sich die Plattform auf ein Kreis ausrichtet und ein anderes sogt dafür, dass die Plattform von einem Kreis zum anderen fährt.

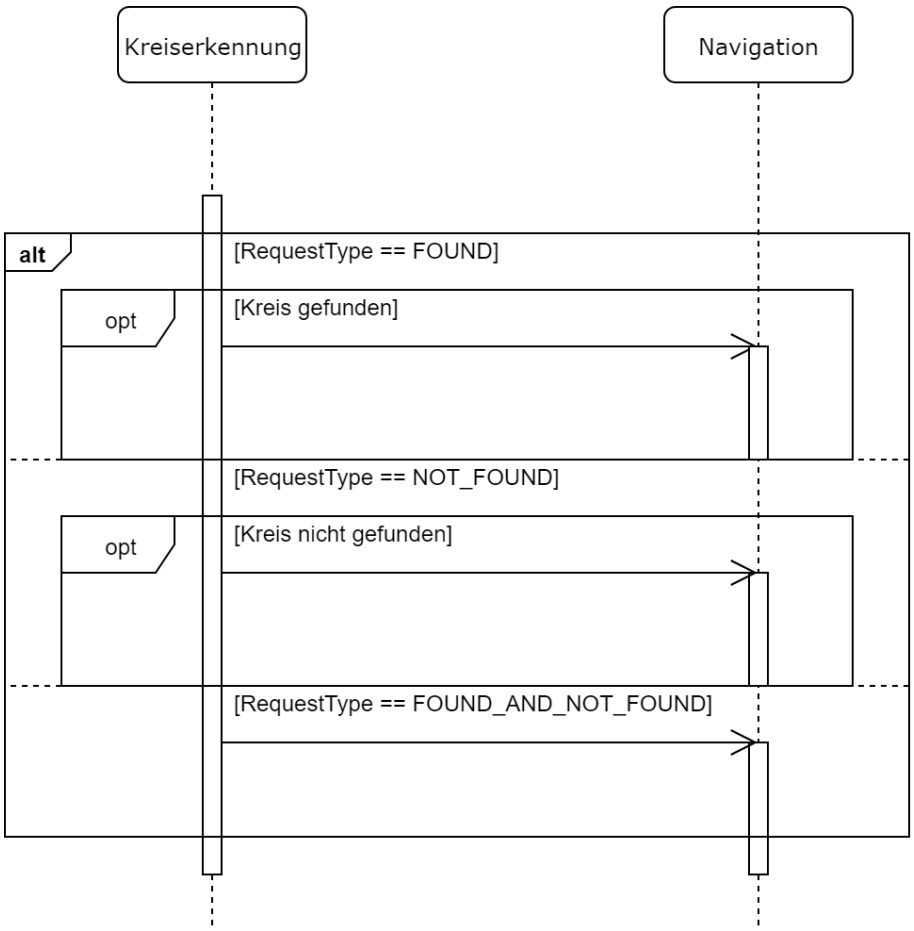


Abbildung 3.16: Die Antwort von der Kreiserkennung

Die Abbildung 3.16 zeigt, dass die Navigation drei verschiedene Antwortmöglichkeiten von der Kreiserkennung erwarten kann. Diese werden in den folgenden Unterabschnitten näher beschrieben.

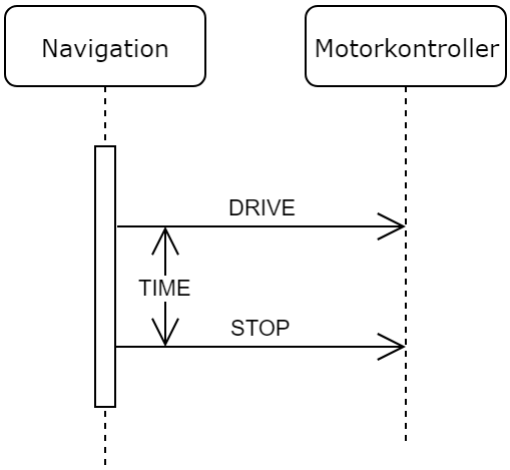


Abbildung 3.17: Struktur der Kommunikation mit dem Motorkontroller

Die Abbildung 3.17 zeigt, wie die Kommunikation mit dem Motorkontroller aufgebaut ist. Das Signal $DRIVE$ ist ein Element der Menge aus $FORWARD$, $BACKWARD$, $LEFT$, $RIGHT$ und $STOP$. Jedes Mal wird für eine bestimmte Zeit eine Aktion ausgeführt und danach gestoppt. Die Berechnung dieser Zeit wird in Unterabschnitt 3.4.3 beschrieben.

3.4.1 Auf ein Kreis ausrichten

Die Navigation sendet der Kreiserkennung, dass sie bereit ist zu empfangen und dass die Kreiserkennung ihr Nachrichten senden soll, wenn ein Kreis gefunden wurde. Wenn sie die Position des Kreises bekommt, sorgt sie dafür, dass sich die Plattform auf den Kreis ausrichtet.

Track-Navigation-Automat

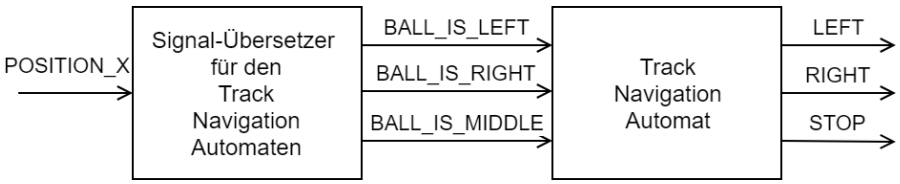


Abbildung 3.18: Eingehende und ausgehende Signale des Track-Navigation-Automaten

Die Abbildung 3.18 zeigt die eingehenden und ausgehenden Signale des Automaten für das Ausrichten auf einen Kreis. Das Signal $POSITION\_X$ bekommt die Navigation von der Kreiserkennung. Dieses Signal zeigt die Position des Kreises auf dem aufgenommenen Frame.

```
getTransition() {
  if (POSITION_X > (CENTER_X + TOLERANCE)) {
    return BALL_IS_RIGHT;
  }
  if (POSITION_X < (CENTER_X - TOLERANCE)) {
    return BALL_IS_LEFT;
  }
  return BALL_IS_MIDDLE;
}
```

Der obere Pseudocode zeigt die Übersetzung des Signals $POSITION\_X$ für den Track-Navigation-Automaten (siehe Abb. 3.18). Die Variable $CENTER\_X$ ist die Position in der Mitte des Frames. Ist die übergebene Position größer als $CENTER\_X$, mit eine gewissen Toleranz, ist der Kreis rechts und $BALL\_IS\_RIGHT$ wird zurückgeliefert. Das Gleiche gilt umgekehrt mit $BALL\_IS\_LEFT$, wenn der Kreis links ist. $BALL\_IS\_MIDDLE$ wird somit zurückgeliefert, wenn sich der Kreis im Zentrum $\pm TOLERANZ$ befindet. Alle Werte sind in Pixel dargestellt.

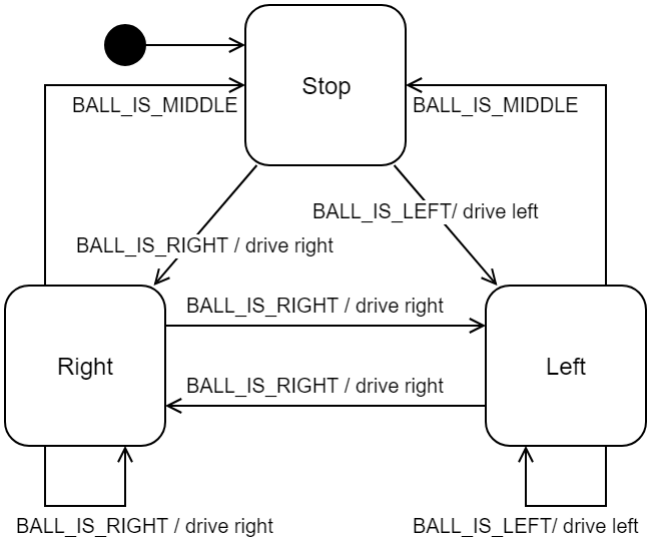


Abbildung 3.19: Automat für das Ausrichten auf ein Kreis

Die Zustandsmaschine für das einfache Verfolgen eines Kreises mit einer Farbe besteht nur aus drei Zuständen $Right$, $Left$ und $Stop$, wie in Abbildung 3.19 dargestellt. Die Transitionen beziehen sich nur darauf, ob sich der Kreis links, rechts oder mit eine gewissen Toleranz in der Mitte befindet. Von $Right$ führt die Transition $BALL\_IS\_RIGHT$ auf sich selber und führt auch dazu, dass sich die Plattform für eine bestimmte Zeit nach rechts dreht. Die Transition

$BALL\_IS\_LEFT$ überführt den Automaten von $Right$ zu $Left$ und dreht die Plattform nach links. In den Zuständen $Left$ und $Stop$ bewirken die beiden genannten Transitionen das gleiche Verhalten. Die letzte Transition $BALL\_IS\_MIDDLE$ überführen den Automaten in den Zustand $Stop$.

3.4.2 Das Fahren zwischen zwei Kreisen

Die Navigation sendet der Kreiserkennung, dass sie bereit ist zu empfangen und dass sie sowohl wissen will, wenn ein Kreis gefunden wurde, als auch wenn keins gefunden wurde. Wenn nach dem Starten des Programms noch kein Kreis gefunden wurde, sorgt die Navigation dafür, dass sich die Plattform etappenweise nach links dreht, bis der rote Kreis auftaucht. Nach einer Rotation stoppt die Plattform, wenn sie kein Kreis findet, solange bis dieser nicht erscheint. Wurde ein Kreis gefunden und er befindet sich nicht im Fokus, richtet die Navigation die Plattform so aus, dass der Kreis in der Mitte steht. Dann fährt sie auf den Kreis zu. Das Ganze wiederholt sich solange, bis die Plattform bis zu einem gewissen Abstand zum Kreis gefahren ist. Danach wird der Kreis mit der anderen Farbe gesucht.

Navigation-Automat

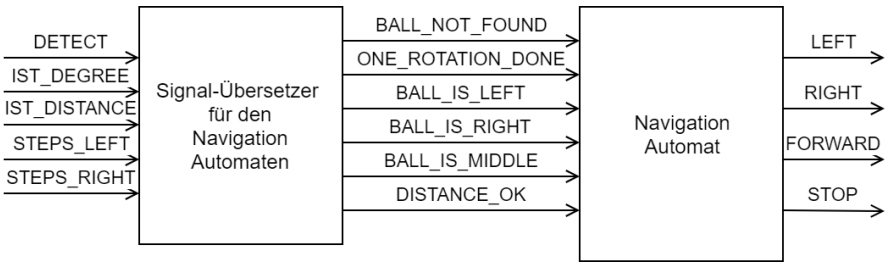


Abbildung 3.20: Eingehende und ausgehende Signale des Navigation-Automaten

Die Abbildung 3.20 zeigt die eingehenden und ausgehenden Signale des Automaten für das Fahren zwischen zwei Kreisen. Aus der Position und Radius des Kreises werden die Signale $IST\_DISTANCE$ und IST_DEGREE berechnet (siehe Abschnitt 3.4.3). Die Position und Radius bekommt die Navigation von der Kreiserkennung. Das Signal $DETECT$ bekommt die Navigation direkt von der Kreiserkennung, dieses gibt an, ob der Kreis gefunden wurde. Die beiden Signale $STEPS\_LEFT$ und $STEPS\_RIGHT$ liefert der Encoder.

```
getTransition() {
  Transition transition;
  if (not DETECT) {
    transition = BALL_NOT_FOUND;
    if((STEPS_LEFT + STEPS_RIGHT) > STEPS_FOR_ONE_ROTATION){
      transition = ONE_ROTATION_DONE;
    }
  } else {
    if (IST_DEGREE <= -WINKEL_TOLERANZ) {
      transition = BALL_IS_LEFT;
    } else if (IST_DEGREE >= WINKEL_TOLERANZ) {
      transition = BALL_IS_RIGHT;
    } else if (IST_DISTANCE <= (MIN_ABSTAND + ABSTAND_TOLERANZ)){
      transition = DISTANCE_OK;
    } else {
      transition = BALL_IS_MIDDLE;
    }
    reset(STEPS_LEFT,STEPS_RIGHT);
  }
  return transition;
}
```

Der obere Pseudocode zeigt die Übersetzung der Signale in die Signale für den Navigation-Automaten (siehe Abb. 3.20). Wenn der Kreis nicht gefunden wurde, wird $BALL\_NOT\_FOUND$ zurückgeliefert, es sei denn die Plattform hat eine Rotation gemacht, was in der fünften Zeile abgefragt wird, dann bekommt der Automat das Signal $ONE\_ROTATION\_DONE$. Wenn der Kreis gefunden wurde, liefert der Code, ob der Kreis links $BALL\_IS\_LEFT$, rechts $BALL\_IS\_RIGHT$ oder mittig $BALL\_IS\_MIDDLE$ gefunden wurde. Außerdem wenn der Abstand zum Kreis den gewünschten Abstand erreicht hat, wird $DISTANCE\_OK$ zurückgeliefert. Zusätzlich werden $STEPS\_LEFT$ und $STEPS\_RIGHT$ zurückgesetzt, um bei dem nächsten Verlust des Kreises von vorne anfangen zu zählen.

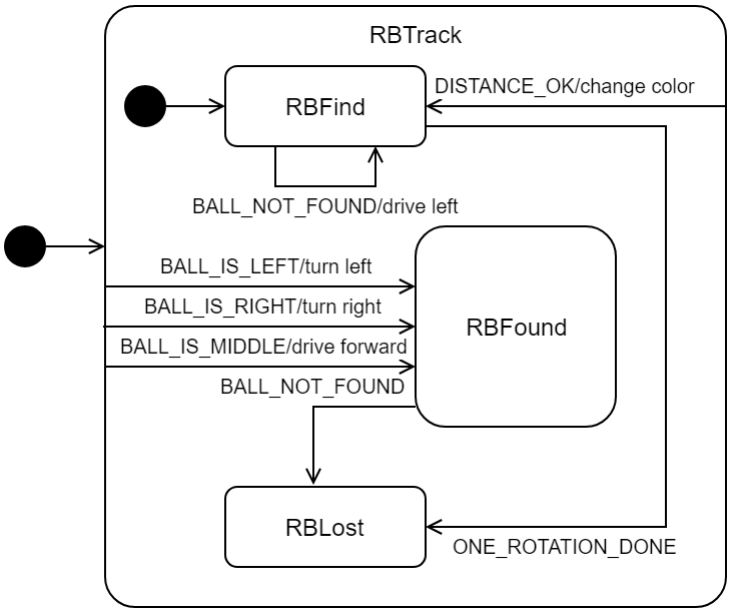


Abbildung 3.21: Automat für das Fahren zwischen zwei Kreisen

Um zwischen zwei Kreisen hin und her zu fahren, benutzt die Navigation einen einfachen hierarchischen Zustandsautomaten, wie in Abbildung 3.21 dargestellt. Dieser Automat besteht aus vier Zuständen $RBTrack$, $RBFind$, $RBFound$ und $RBLost$. Die drei letzteren Zustände befinden sich im Zustand $RBTrack$. Der Anfangszustand ist der $RBFind$. Von diesem Zustand führt eine Transition $ONE\_ROTATION\_DONE$ zum Zustand $RBLost$ und ein Transition $BALL\_NOT\_FOUND$ führt auf sich selbst. Solange kein Kreis gefunden wurde, führt die letztere Transition dazu, dass sich die Plattform um eine gewisse Zeit dreht. Wurde mit Hilfe der Encoder registriert, dass eine Rotation gemacht wurde, wird der Automat in den Zustand $RBLost$ überführt. Der übergeordneter Zustand $RBTrack$ reagiert auf vier Transitionen. Drei der Transitionen $BALL\_IS\_RIGHT$, $BALL\_IS\_LEFT$ und $BALL\_IS\_MIDDLE$ überführen den Automaten in den Zustand $RBFound$ und sorgen dafür, dass die Plattform zu einem Kreis fährt. Die letzte Transition $DISTANCE\_OK$ führt zum Zustand $RBFind$, wechselt die Farbe des gesuchten Kreises und bedeutet, dass die Plattform bis zu einem gewissen Abstand an den Kreis herangefahren ist. Ist der Automat im Zustand $RBFind$ fängt alles von vorne an, aber mit einem andere Zielobjekt.

```
void Regler::drive(Direction direction, int speed, double time) {
  static HQueue<MotorMessage*>* queue = (HQueue<MotorMessage*>*)
      HQueueFactory::getQueue(MOTOR_QUEUE_ID);
  static MotorMessage* message = NULL;
  if(time > 0) {
    message = new MotorMessage(direction, speed);
    queue->enqueue(message);
    usleep(time*1E6);
    message = new MotorMessage(STOP, 0);
    queue->enqueue(message);
  } else if(direction == STOP) {
    message = new MotorMessage(STOP, 0);
    queue->enqueue(message);
  }
}
```

Der obere Codeabschnitt zeigt die Funktion, die bei den Transitionen, außer $DISTANCE\_OK$, ausgeführt wird. $Direction$ sind die Signale $LEFT$, $RIGHT$, $FORWARD$ und $STOP$ für den Motorkontroller, die Variable *speed* ist die Geschwindigkeit, wobei hier immer die volle Geschwindigkeit gewählt wird und $time$ ist die Zeit in Sekunden, wie lange er fahren soll. Das Verhalten dieses Codeabschnittes und die Berechnung der Zeit wird in dem Abschnitt Bahnberechnung (siehe Abschnitt 3.4.3) beschrieben. Bei der Transition $DISTANCE\_OK$ wird folgende Code ausgeführt.

```
  switch (bTContent_->color_) {
  case HColor::RED:
    bTContent_->color_ = HColor::BLUE;
    break;
  case HColor::BLUE:
    bTContent_->color_ = HColor::RED;
    break;
  }
```

Dieser Codeabschnitt sorgt dafür, dass wenn vorher nach einem roten Kreis gesucht wurde, jetzt nach einem blauen Kreis gesucht wird und umgekehrt.

3.4.3 Bahnberechnung

Für die Zeiten, wie lange die Plattform drehen oder fahren soll, werden die Gleichungen eines Regelkreises mit Proportional-Elementen verwendet [vgl. 13, Seite 31]. Beim Ausrichten der Plattform auf ein Kreis benutzt die Navigation eine Winkel-Zeit-Funktion. Bei dem Winkel handelt es sich um den Winkel zwischen dem Fokus der Kamera und dem Mittelpunkt des

Kreises. Steigt dieser Winkel, steigt auch die Zeit, wie lange er drehen soll. Ähnlich gilt es für das Fahren zu einem Kreis. Für das Fahren gibt es eine Weg-Zeit-Funktion. Der Weg ist der Abstand zwischen der Kamera und dem Kreis minus dem gewünschten Abstand zum Kreis, also der Weg, den er fahren soll. Auch hier steigt die Zeit je weiter die Plattform von dem Kreis entfernt ist.

Regelstrecke für Drehen

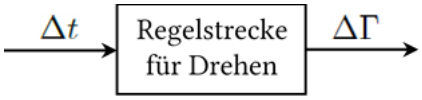


Abbildung 3.22: Blockschaltbild der Regelstrecke für das Drehen

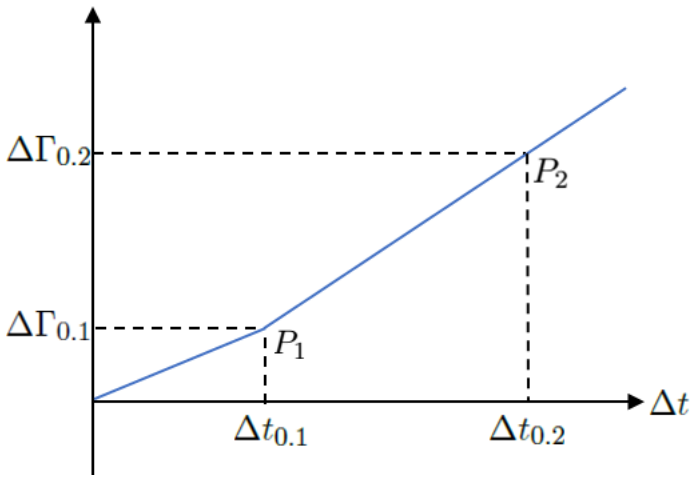


Abbildung 3.23: Regelstrecke für das Drehen

Die Abbildung 3.23 zeigt den Funktionsgraphen der Regelstrecke für das Drehen der Plattform. Die Variable $\Delta\Gamma$ ist die Winkeländerung in einem bestimmten Zeitabschnitt Δt. Ob sich die Plattform nach links oder nach rechts dreht, entscheiden die Automaten (siehe Abb. 3.19 und 3.21). Das folgende Bild verdeutlicht das Verhalten aus Abbildung 3.23.

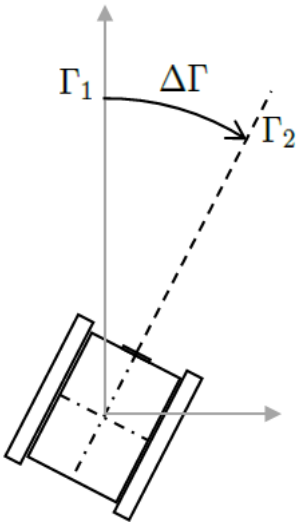


Abbildung 3.24: Drehstrecke der Plattform

Die Abbildung 3.24 zeigt den Winkel $\Delta\Gamma$, der in einer bestimmten Zeit Δt zurückgelegt wird. Der Winkel entsteht aus der Differenz der Winkel im Koordinatensystem $\Delta\Gamma = \Gamma_2 - \Gamma_1$ und die Zeit aus der Zeitdifferenz $\Delta t = t_2 - t_1$, wobei in diesem Fall gilt $\Gamma_1 = 0°$ und $t_1 = 0s$. Die folgende Funktion beschreibt die Abbildung 3.23.

$$\Delta\Gamma(\Delta t) = \begin{cases} \omega_1 \cdot \Delta t & \text{für } 0 \leq \Delta t < \Delta t_{0.1} \\ \omega_2 \cdot \Delta t + S_2 & \text{für } \Delta t_{0.1} \leq \Delta t \end{cases} \tag{3.5}$$

In der Gleichung 3.5 sind die Variablen ω_1 und ω_2 die Winkelgeschwindigkeiten, wie schnell sich die Plattform dreht , Δt ist der Zeitabschnitt, wie lange die Plattform drehen soll, und S_2 wird benötigt, um den Graphen zwischen P_1 und P_2 vollständig zu beschreiben. Um die Winkelgeschwindigkeiten auszurechnen, müssen Messungen gemacht werden. Dafür werden zwei Zeiten $\Delta t_{0.1} = 0,1s$ und $\Delta t_{0.2} = 0,2s$ gewählt (siehe Abb. 3.23), für die jeweils 10 Mal gemessen wird, wie weit sich die Plattform gedreht hat. Danach werden aus den gemessenen Werten die Erwartungswerte mit der Formel

$$E(\Delta\Gamma) = \frac{1}{N} \sum_{i=0}^{N} \Delta\Gamma_i \tag{3.6}$$

und die Standartabweichungen mit der Formel

$$s(\Delta\Gamma) = \sqrt[2]{\frac{1}{N}\sum_{i=0}^{N}(\Delta\Gamma_i - E(\Delta\Gamma))^2} \tag{3.7}$$

berechnet.

Versuche:	1	2	3	4	5	6	7	8	9	10	
$0,1s$	$17°$	$17°$	$21°$	$16°$	$20°$	$15°$	$15°$	$15°$	$12°$	$17°$	$\Delta\Gamma_{0.1}$
$0,2s$	$40°$	$43°$	$41°$	$35°$	$40°$	$40°$	$43°$	$42°$	$39°$	$42°$	$\Delta\Gamma_{0.2}$

Tabelle 3.2: Gemessene Werte für Drehen

Aus den gemessenen Werten (siehe Tabelle 3.2) können die Erwartungswerte und die Standartabweichungen berechnet werden. Der Erwartungswert E ist die Summe aller Messwerte durch die Anzahl der Messungen (siehe Gleichung (3.6)) und die Standartabweichung ist die zweite Wurzel aus der Summe aller Messwerte minus Erwartungswert ins Quadrat durch die Anzahl der Messungen (siehe Gleichung (3.7)).

$$E_{0.1}(\Delta\Gamma_{0.1}) = \frac{1}{N}\sum_{i=0}^{N}\Delta\Gamma_{i_{0.1}} = \frac{1}{10}\sum_{i=0}^{10}\Delta\Gamma_{i_{0.1}} = \frac{1}{10}\cdot(17° + 17° + \ldots + 17°) = 16,5°$$

$$s_{0.1}(\Delta\Gamma_{0.1}) = \sqrt[2]{\frac{1}{N}\sum_{i=0}^{N}(\Delta\Gamma_{i_{0.1}} - E_{0.1}(\Delta\Gamma_{0.1}))^2} = \sqrt[2]{\frac{1}{10}\sum_{i=0}^{10}(\Delta\Gamma_{i_{0.1}} - E_{0.1}(\Delta\Gamma_{0.1}))^2}$$
$$= \sqrt[2]{\frac{1}{10}\cdot((17° - 16,5°)^2 + \ldots + (17° - 16,5°)^2)} = 2,46°$$

$$E_{0.2}(\Delta\Gamma_{0.2}) = \frac{1}{N}\sum_{i=0}^{N}\Delta\Gamma_{i_{0.2}} = \frac{1}{10}\sum_{i=0}^{10}\Delta\Gamma_{i_{0.2}} = \frac{1}{10}\cdot(40° + 43° + \ldots + 42°) = 40,5°$$

$$s_{0.2}(\Delta\Gamma_{0.2}) = \sqrt[2]{\frac{1}{N}\sum_{i=0}^{N}(\Delta\Gamma_{i_{0.2}} - E_{0.2}(\Delta\Gamma_{0.2}))^2} = \sqrt[2]{\frac{1}{10}\sum_{i=0}^{10}(\Delta\Gamma_{i_{0.2}} - E_{0.2}(\Delta\Gamma_{0.2}))^2}$$
$$= \sqrt[2]{\frac{1}{10}\cdot((40° - 40,5°)^2 + \ldots + (42° - 40,5°)^2)} = 2,25°$$

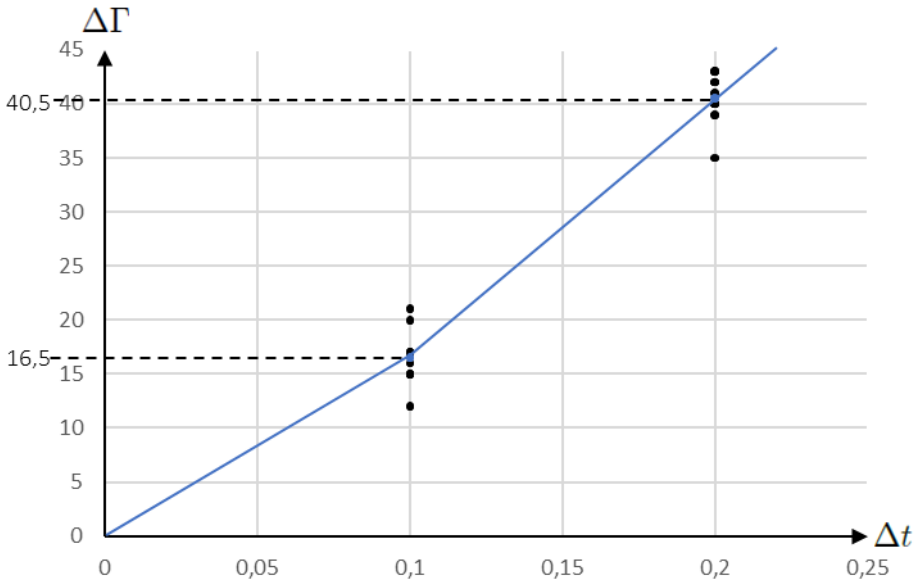


Abbildung 3.25: Gemessene Werte für Drehen

Formeln für ω_1, ω_2 und S_2:

$$\omega_1 = \frac{\Delta\Gamma_{0.1}}{\Delta t_{0.1}} \tag{3.8}$$

$$\omega_2 = \frac{\Delta\Gamma_{0.2} - \Delta\Gamma_{0.1}}{\Delta t_{0.2} - \Delta t_{0.1}} \tag{3.9}$$

$$S_2 = \Delta\Gamma_{0.1} - (\omega_2 \cdot \Delta t_{0.1}) \tag{3.10}$$

Für $\Delta\Gamma_{0.1}$ und $\Delta\Gamma_{0.2}$ werden deren Erwartungswerte genommen, sodass gilt:

$$\Delta\Gamma_{0.1} = E_{0.1}(\Delta\Gamma_{0.1}) = 16,5°$$

$$und$$

$$\Delta\Gamma_{0.2} = E_{0.2}(\Delta\Gamma_{0.2}) = 40,5°$$

Berechnungen für ω_1, ω_2 und S_2:

$$\omega_1 = \frac{\Delta\Gamma_{0.1}}{\Delta t_{0.1}} = \frac{E_{0.1}(\Delta\Gamma_{0.1})}{\Delta t_{0.1}} = \frac{16,5°}{0,1s} = 165°\frac{1}{s}$$
$$\omega_2 = \frac{\Delta\Gamma_{0.2} - \Delta\Gamma_{0.1}}{\Delta t_{0.2} - \Delta t_{0.1}} = \frac{E_{0.2}(\Delta\Gamma_{0.2}) - E_{0.1}(\Delta\Gamma_{0.1})}{\Delta t_{0.2} - \Delta t_{0.1}} = \frac{40,5° - 16,5°}{0,2s - 0,1s} = 240°\frac{1}{s}$$
$$S_2 = \Delta\Gamma_{0.1} - (\omega_2 \cdot \Delta t_{0.1}) = E_{0.1}(\Delta\Gamma_{0.1}) - (\omega_2 \cdot \Delta t_{0.1}) = 16,5° - (240°\frac{1}{s} \cdot 0,1s) = -7,5°$$

Für die Regelstrecke in Gleichung 3.5 ergibt sich folgende Formel:

$$\Delta\Gamma(\Delta t) = \begin{cases} 165°\frac{1}{s} \cdot \Delta t & \text{für } 0 \leq \Delta t < \Delta 0,1s \\ 240°\frac{1}{s} \cdot \Delta t - 7,5° & \text{für } 0,1s \leq \Delta t \end{cases} \tag{3.11}$$

Diese Formel gilt nur annähernd für die Geraden zwischen 0 und P_1 und zwischen P_1 und P_2 (siehe Abb. 3.23).

Regler für Drehen

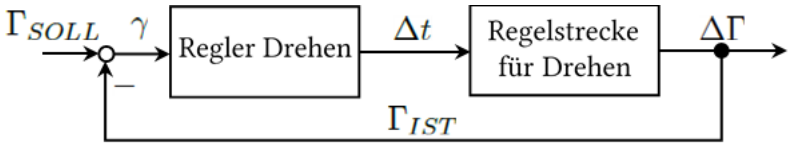


Abbildung 3.26: Blockschaltbild eines Regelkreises für das Drehen

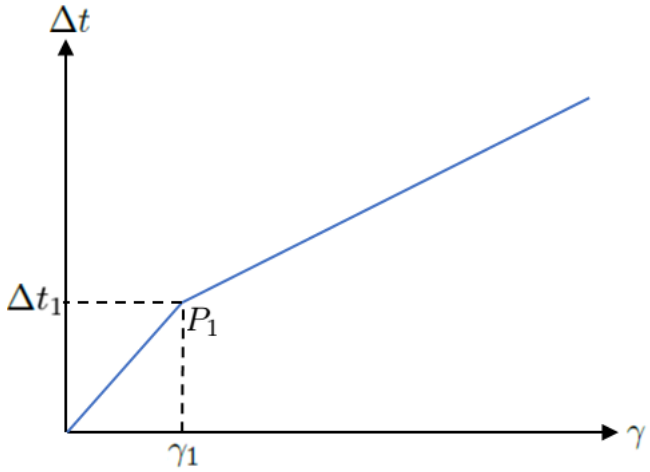


Abbildung 3.27: Proportionalregler für das Drehen der Plattform

Die Abbildung 3.27 zeigt den Funktionsgraphen der Winkel-Zeit-Funktion. Die Variable γ ist die Regelabweichung und wird aus der Differenz des Ist-Wertes Γ_{IST}, in diesem Fall der Winkel zwischen Kreismitte und dem Fokus der Kamera, und des Soll-Wertes Γ_{SOLL}, in diesem Fall $0°$, gebildet (siehe Abb. 3.26). Zusätzlich sollte γ positiv sein, also handelt es sich um den Betrag der Differenz. Die Variable γ_1 ist der Winkel, der zwei Funktionen trennt, da der Regler durch zwei Geraden beschrieben wird. Die Variable Δt_1 gibt die Zeit für den Winkel γ_1 an. Die folgende Funktion beschreibt diesen Graphen.

$$\Delta t(\gamma) = \begin{cases} P_{\gamma 1} \cdot \gamma & \text{für } \gamma < \gamma_1 \\ P_{\gamma 2} \cdot \gamma + S_{\gamma 2} & \text{für } \gamma \leq \gamma \end{cases} \tag{3.12}$$

$$\gamma = |\Gamma_{SOLL} - \Gamma_{IST}| \tag{3.13}$$

Die Variablen $P_{\gamma 1}$, $P_{\gamma 2}$, in der Gleichung 3.12 sind die Proportionalitätsfaktoren der Funktion und $S_{\gamma 2}$ ist der Startwert. Der Faktor $P_{\gamma 1}$ gilt nur bis γ_1 und ab γ_1 beschreiben $P_{\gamma 2}$ und $S_{\gamma 2}$ die Funktion. Wenn für die Variable γ_1 der Erwartungswert aus $\Delta\Gamma_{0.1}$ genommen wird (siehe Abb. 3.25), können die Variablen in der Gleichung 3.12 aus der Gleichung 3.11 abgeleitet werden. Rechnung für $\gamma < \gamma_1$, also $\gamma < \Delta\Gamma_{0.1}$:

$$\begin{aligned} \Delta\Gamma(\Delta t) &= 165° \frac{1}{s} \cdot \Delta t \\ \gamma &= 165° \frac{1}{s} \cdot \Delta t \quad | : 165° \frac{1}{s} \\ \frac{\gamma}{165° \frac{1}{s}} &= \Delta t \\ \frac{1s}{165°} \cdot \gamma &= \Delta t \end{aligned}$$

Rechnung für $\gamma_1 \leq \gamma$, also $\Delta\Gamma_{0.1} \leq \gamma$:

$$\begin{aligned}
\Delta\Gamma(\Delta t) &= 240° \frac{1}{s} \cdot \Delta t - 7,5° \\
\gamma &= 240° \frac{1}{s} \cdot \Delta t - 7,5° \quad | + 7,5° | : 240° \frac{1}{s} \\
\frac{\gamma + 7,5°}{240° \frac{1}{s}} &= \Delta t \\
\frac{\gamma}{240° \frac{1}{s}} + \frac{7,5°}{240° \frac{1}{s}} &= \Delta t \\
\frac{1s}{240°} \cdot \gamma + \frac{7,5}{240} s &= \Delta t
\end{aligned}$$

Aus den beiden oberen Rechnungen ergibt sich für $P_{\gamma 1}$, $P_{\gamma 2}$ und $S_{\gamma 2}$ dann

$$\begin{aligned}
P_{\gamma 1} &= \frac{1s}{165°} = 6,0606 \cdot 10^{-3} \frac{s}{°} \\
P_{\gamma 2} &= \frac{1s}{240°} = 4,1667 \cdot 10^{-3} \frac{s}{°} \\
S_{\gamma 2} &= \frac{7,5}{240} s = 31,25 \cdot 10^{-3} s
\end{aligned}$$

Für die Gleichung 3.12 folgt daraus:

$$\Delta t(\gamma) = \begin{cases} 6,0606 \cdot 10^{-3} \frac{s}{°} \cdot \gamma & \text{für } \gamma < 16,5° \\ 4,1667 \cdot 10^{-3} \frac{s}{°} \cdot \gamma + 31,25 \cdot 10^{-3} s & \text{für } 16,5° \leq \gamma \end{cases} \tag{3.14}$$

Regelstrecke für Fahren

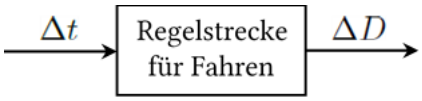


Abbildung 3.28: Blockschaltbild der Regelstrecke für das Fahren

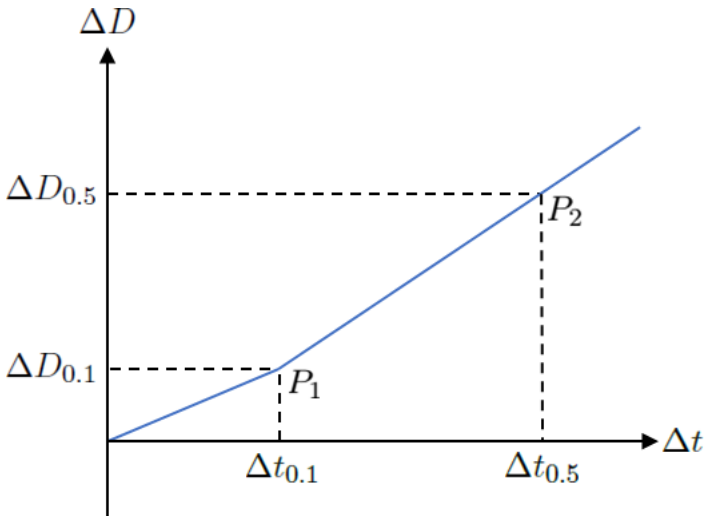


Abbildung 3.29: Weg-Zeit-Funktion für das Vorwärtsfahren der Plattform

Die Abbildung 3.29 zeigt den Funktionsgraphen für das Fahren der Plattform. Die Variable ΔD ist die Distanzänderung in einem bestimmten Zeitabschnitt Δt. Das folgende Bild verdeutlicht das Verhalten aus Abbildung 3.29.

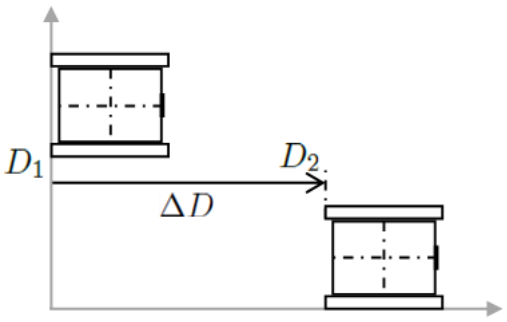


Abbildung 3.30: Fahrstrecke der Plattform

Die Abbildung 3.30 zeigt die Distanz ΔD, die in einer bestimmten Zeit Δt zurückgelegt wird. Die Distanz entsteht aus der Differenz der Distanzen im Koordinatensystem $\Delta D = D_2 - D_1$ und die Zeit aus der Zeitdifferenz $\Delta t = t_2 - t_1$, wobei in diesem Fall gilt $D_1 = 0cm$ und $t_1 = 0s$. Die folgende Funktion beschreibt die Abbildung 3.29.

$$\Delta D(\Delta t) = \begin{cases} v_1 \cdot \Delta t & \text{für } 0 \leq \Delta t < \Delta t_{0.1} \\ v_2 \cdot \Delta t + S_2 & \text{für } \Delta t_{0.1} \leq \Delta t \end{cases} \tag{3.15}$$

In der Gleichung 3.15 sind die Variablen v_1, v_2 die Geschwindigkeiten, wie schnell die Plattform fährt, Δt ist die Zeit, wie lange die Plattform fahren soll, und S_2 wird benötigt, um den Graphen zwischen P_1 und P_2 vollständig zu beschreiben. Um die Geschwindigkeiten auszurechnen, müssen Messungen gemacht werden. Dafür werden zwei Zeiten $\Delta t_{0.1} = 0,1$ und $\Delta t_{0.5} = 0,5s$ gewählt (siehe Abbildung 3.29), für die jeweils 10 Mal gemessen wird, wie weit die Plattform gefahren ist. Danach werden aus den gemessenen Werten die Erwartungswerte mit der Formel

$$E(\Delta D) = \frac{1}{N} \sum_{i=0}^{N} \Delta D_i \tag{3.16}$$

und die Standartabweichungen mit der Formel

$$s(\Delta D) = \sqrt[2]{\frac{1}{N} \sum_{i=0}^{N} (\Delta D_i - E(\Delta D))^2} \tag{3.17}$$

berechnet.

Versuche:	1	2	3	4	5	
$0,1s$	$3,3cm$	$3,2cm$	$3,1cm$	$3,5cm$	$3,4cm$	$\Delta D_{0.1}$
$0,5s$	$26cm$	$27,2cm$	$27cm$	$27,1cm$	$27,3cm$	$\Delta D_{0.5}$

Tabelle 3.3: Gemessene Werte für Fahren. Versuche 1 bis 5

Versuche:	6	7	8	9	10	
$0,1s$	$3cm$	$2,7cm$	$3cm$	$3,4cm$	$2,5cm$	$\Delta D_{0.1}$
$0,5s$	$27,4cm$	$27,4cm$	$27,5cm$	$27,5cm$	$27,6cm$	$\Delta D_{0.5}$

Tabelle 3.4: Gemessene Werte für Fahren. Versuche 6 bis 10

Aus den gemessenen Werten (siehe Tabellen 3.3 und 3.4) können die Erwartungswerte und die Standartabweichungen berechnet werden.

$$E_{0.1}(\Delta D_{0.1}) = \frac{1}{N}\sum_{i=0}^{N}\Delta D_{i_{0.1}} = \frac{1}{10}\sum_{i=0}^{10}\Delta D_{i_{0.1}} = \frac{1}{10}\cdot(3,3cm + \ldots + 2,5cm) = 3,11cm$$

$$s_{0.1}(\Delta D_{0.1}) = \sqrt[2]{\frac{1}{N}\sum_{i=0}^{N}(\Delta D_{i_{0.1}} - E_{0.1}(\Delta D_{0.1}))^2} = \sqrt[2]{\frac{1}{10}\sum_{i=0}^{10}(\Delta D_{i_{0.1}} - E_{0.1}(\Delta D_{0.1}))^2}$$
$$= \sqrt[2]{\frac{1}{10}\cdot((3,3cm - 3,11cm)^2 + \ldots + (2,5cm - 3,11cm)^2)} = 0,305cm$$

$$E_{0.5}(\Delta D_{0.5}) = \frac{1}{N}\sum_{i=0}^{N}\Delta D_{i_{0.5}} = \frac{1}{10}\sum_{i=0}^{10}\Delta D_{i_{0.5}} = \frac{1}{10}\cdot(26cm + \ldots + 27,6cm) = 27,2cm$$

$$s_{0.5}(\Delta D_{0.5}) = \sqrt[2]{\frac{1}{N}\sum_{i=0}^{N}(\Delta D_{i_{0.5}} - E_{0.5}(\Delta D_{0.5}))^2} = \sqrt[2]{\frac{1}{10}\sum_{i=0}^{10}(\Delta D_{i_{0.5}} - E_{0.5}(\Delta D_{0.5}))^2}$$
$$= \sqrt[2]{\frac{1}{10}\cdot((26cm - 27,2cm)^2 + \ldots + (27,6cm - 27,2cm)^2)} = 0,438cm$$

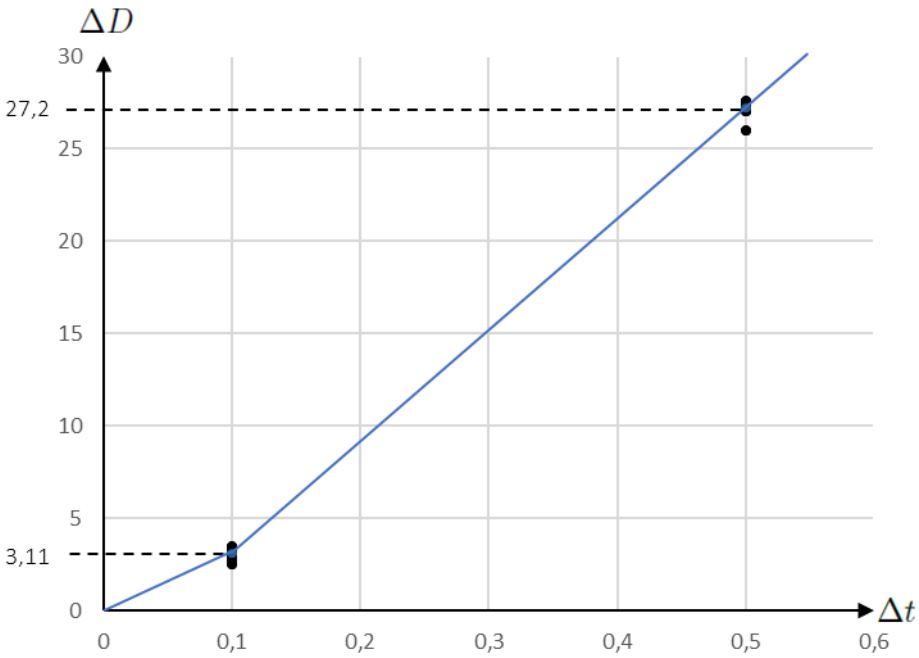


Abbildung 3.31: Gemessene Werte für Fahren

Formeln für v_1, v_2 und S_2:

$$v_1 = \frac{\Delta D_{0.1}}{\Delta t_{0.1}} \tag{3.18}$$

$$v_2 = \frac{\Delta D_{0.5} - \Delta D_{0.1}}{\Delta t_{0.5} - \Delta t_{0.1}} \tag{3.19}$$

$$S_2 = \Delta D_{0.5} - (v_2 \cdot \Delta t_{0.5}) \tag{3.20}$$

Für $\Delta D_{0.1}$ und $\Delta D_{0.5}$ werden deren Erwartungswerte genommen, sodass gilt:

$$\Delta D_{0.1} = E_{0.1}(\Delta D_{0.1}) = 3,11cm$$
$$und$$
$$\Delta D_{0.5} = E_{0.5}(\Delta D_{0.5}) = 27,2cm$$

Berechnungen für v_1, v_2 und S_2:

$$v_1 = \frac{\Delta D_{0.1}}{\Delta t_{0.1}} = \frac{E_{0.1}(\Delta D_{0.1})}{\Delta t_{0.1}} = \frac{3,11cm}{0,1s} = 31,1\frac{cm}{s}$$

$$v_2 = \frac{\Delta D_{0.5} - \Delta D_{0.1}}{\Delta t_{0.5} - \Delta t_{0.1}} = \frac{E_{0.5}(\Delta D_{0.5}) - E_{0.1}(\Delta D_{0.1})}{\Delta t_{0.5} - \Delta t_{0.1}} = \frac{27,2cm - 3,11cm}{0,5s - 0,1s} = 60,225\frac{cm}{s}$$

$$S_2 = \Delta D_{0.1} - (v_2 \cdot \Delta t_{0.1}) = E_{0.1}(\Delta D_{0.1}) - (v_2 \cdot \Delta t_{0.1}) = 3,11cm - (60,225\frac{cm}{s} \cdot 0,1s)$$
$$= -2,9125cm$$

Für die Regelstrecke in Gleichung 3.15 ergibt sich folgende Formel:

$$\Delta D(\Delta t) = \begin{cases} 31,1\frac{cm}{s} \cdot \Delta t & \text{für } 0 \leq \Delta t < 0,1s \\ 60,225\frac{cm}{s} \cdot \Delta t - 2,9125cm & \text{für } 0,1 \leq \Delta t \end{cases} \tag{3.21}$$

Diese Formel gilt nur annähernd für die Gerade zwischen 0 und P_1 und für die Gerade zwischen P_1 uns P_2 (siehe Abb. 3.29).

Regler für Fahren

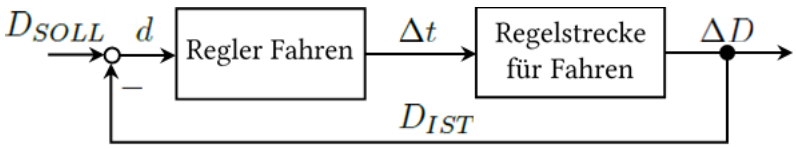


Abbildung 3.32: Blockschaltbild eines Regelkreises für das Fahren

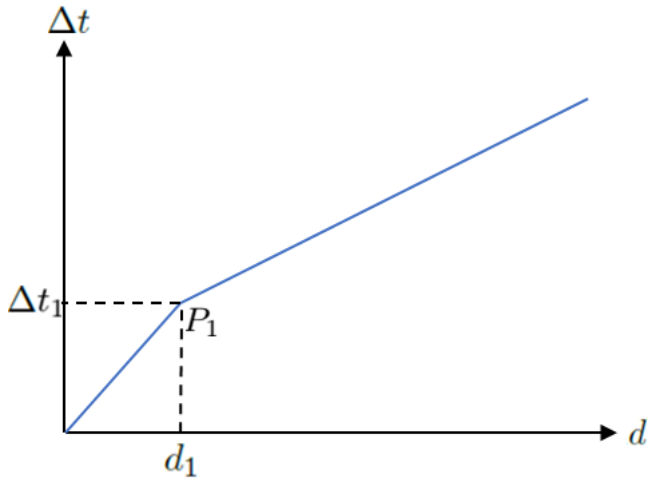


Abbildung 3.33: Proportionalregler für das Vorwärtsfahren der Plattform

Die Abbildung 3.33 zeigt den Funktionsgraphen für die Weg-Zeit-Funktion. Die Variable d ist die Regelabweichung und wird aus der Differenz des Ist-Wertes D_{IST}, in diesem Fall der Abstand zum Kreis, und des Soll-Wertes D_{SOLL}, in diesem Fall den minimalen Abstand, bei dem die Plattform stoppen soll, gebildet (siehe Abb. 3.32). Bei dem d handelt es sich um den Betrag der Differenz. Die Variable d_1 trennt die zwei Funktionen, die den Regler beschreiben. Die Variable t_1 ist die Zeit für den Abstand d_1. Die folgende Funktion beschreibt diesen Graphen.

$$\Delta t(d) = \begin{cases} P_{d1} \cdot d & \text{für } d < d_1 \\ P_{d2} \cdot d + S_{d2} & \text{für } d_1 \leq d \end{cases} \tag{3.22}$$

$$d = |D_{SOLL} - D_{IST}| \tag{3.23}$$

Die Variablen P_{d1}, P_{d2} und S_{d2} haben die gleiche Bedeutung wie $P_{\gamma 1}$, $P_{\gamma 2}$ und $S_{\gamma 2}$ aus der Gleichung 3.12, außer dass sie sich auf den Abstand beziehen. Wenn für die Variable d_1 der Erwartungswert aus $\Delta D_{0.1}$ genommen wird (siehe Abb. 3.31), können die Variablen in der

Gleichung 3.22 aus der Gleichung 3.21 abgeleitet werden.
Rechnung für $d < d_1$, also $d < \Delta D_{0.1}$:

$$\begin{aligned}
\Delta D(\Delta t) &= 31,1\frac{cm}{s} \cdot \Delta t \\
d &= 31,1\frac{cm}{s} \cdot \Delta t \quad | : 31,1\frac{cm}{s} \\
\frac{d}{31,1\frac{cm}{s}} &= \Delta t \\
\frac{1s}{31,1cm} \cdot d &= \Delta t
\end{aligned}$$

Rechnung für $d_1 \leq d$, also $D_{0.1} \leq d$:

$$\begin{aligned}
\Delta D(\Delta t) &= 60,225\frac{cm}{s} \cdot \Delta t - 2,9125cm \\
d &= 60,225\frac{cm}{s} \cdot \Delta t - 2,9125cm \quad | + 2,9125cm \, | : 60,225\frac{cm}{s} \\
\frac{d + 2,9125cm}{60,225\frac{cm}{s}} &= \Delta t \\
\frac{d}{60,225\frac{cm}{s}} + \frac{2,9125cm}{60,225\frac{cm}{s}} &= \Delta t \\
\frac{1s}{60,225cm} \cdot d + \frac{2,9125}{60,225}s &= \Delta t
\end{aligned}$$

Für P_{d1}, P_{d2} und S_{d2} ergeben sich dann

$$\begin{aligned}
P_{d1} &= \frac{1s}{31,1cm} = 32,154 \cdot 10^{-3}\frac{s}{cm} \\
P_{d2} &= \frac{1s}{60,225cm} = 16,604 \cdot 10^{-3}\frac{s}{cm} \\
S_{d2} &= \frac{2,9125}{60,225}s = 48,360 \cdot 10^{-3}s
\end{aligned}$$

Für die Gleichung 3.22 folgt daraus:

$$\Delta t(d) = \begin{cases} 32,154 \cdot 10^{-3}\frac{s}{cm} \cdot d & \text{für } d < 3,11cm \\ 16,604 \cdot 10^{-3}\frac{s}{cm} \cdot d + 48,360 \cdot 10^{-3}s & \text{für } 3,11cm \leq d \end{cases} \tag{3.24}$$

Code

Da Gleichung 3.12 und Gleichung 3.22 die gleichen Funktionalitäten aufweisen, können sie zusammengefasst werden. Der folgende Codeabschnitt zeigt die Implementierung dieser Gleichungen.

```
1 double Regler::time(double soll, double ist, double prop_faktor_1, double
      prop_1_grenze, double prop_faktor_2, double startwert_2) {
2         double delta = abs(soll - ist);
3         if (delta < prop_1_grenze) {
4                 return prop_faktor_1 * delta;
5         }
6         return (prop_faktor_2 * delta + startwert_2);
7 }
```

Sollte für das Drehen der Plattform die Zeit berechnet werden, bekommt die Methode Γ_{SOLL}, Γ_{IST}, $P_{\gamma 1}$, γ_1, $P_{\gamma 2}$ und $S_{\gamma 2}$. Sollte die Zeit für das Fahren zu einem Kreis berechnet werden, werden die Variablen D_{SOLL}, D_{IST}, P_{d1}, d_1, P_{d2} und S_{d2} übergeben.

Winkel und Abstand berechnen

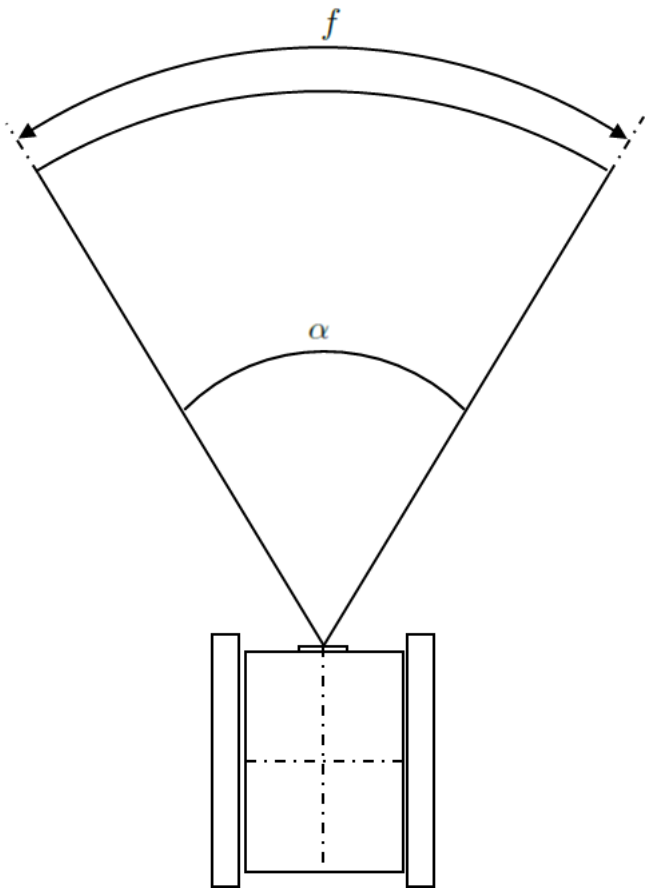


Abbildung 3.34: Bildwinkel vom Raspberry Pi Kamera Modul V2.1

Die Abbildung 3.34 zeigt den Bildwinkel α des Raspberry Pi Kameramoduls. In der Spezifikation des Kameramoduls steht, dass der Winkel $62,2°$ umfasst. Die Variable f beschreibt die Framebreite in Pixel. Diese Breite ist konfigurierbar und wurde auf $640px$ eingestellt.

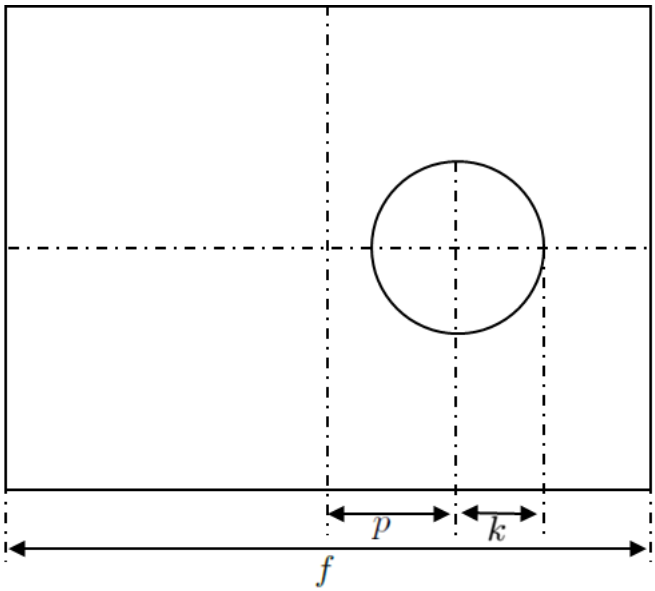


Abbildung 3.35: Frame mit einem Kreis

In der Abbildung 3.35 wird ein Frame mit einem Kreis dargestellt. Darin ist p der Abstand des Kreismittelpunktes von der Framemitte, k ist der Radius und f wieder die Framebreite. Alle drei Variablen werden in Pixel dargestellt.

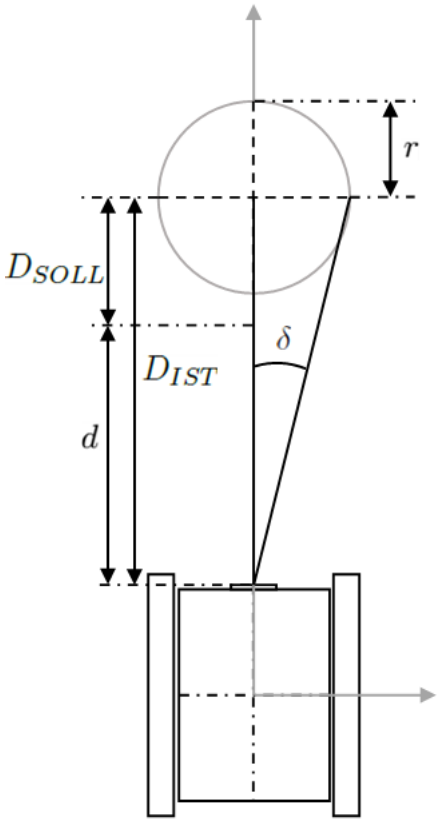


Abbildung 3.36: Abstand zum Kreis

Die Abbildung 3.36 zeigt die Situation wenn sich die Plattform auf den Kreis ausgerichtet hat. Die Variable d ist die Regelabweichung, D_{IST} ist der gesuchte Abstand zum Kreis, D_{SOLL} ist der ausgewählte Abstand, δ ist der Winkel zwischen Kreismitte und dem Rand des Kreises und r ist der tatsächlicher Radius des Kreises. Außer δ werden alle Variablen in Zentimeter dargestellt. Der Ist-Abstand wird mit den folgenden Formeln berechnet:

$$\delta = \frac{\alpha}{f} \cdot k \tag{3.25}$$

$$D_{IST} = \frac{r}{\tan \delta} \tag{3.26}$$

Zuerst wird Radius k (siehe Abb. 3.35) aus dem Frame in Winkel δ umgerechnet (siehe Gleichung (3.25)), dann wird, da der tatsächlicher Radius des Kreises bekannt ist, mit Hilfe der Tangensfunktion der Abstand D_{IST} ausgerechnet (siehe Gleichung (3.26)). Ist D_{IST} bekannt, wird d ausgerechnet und dieser wird der Weg-Zeit-Funktion aus Gleichung 3.24 übergeben, um die Zeit herauszufinden, wie lange die Plattform vorwärts fahren soll.

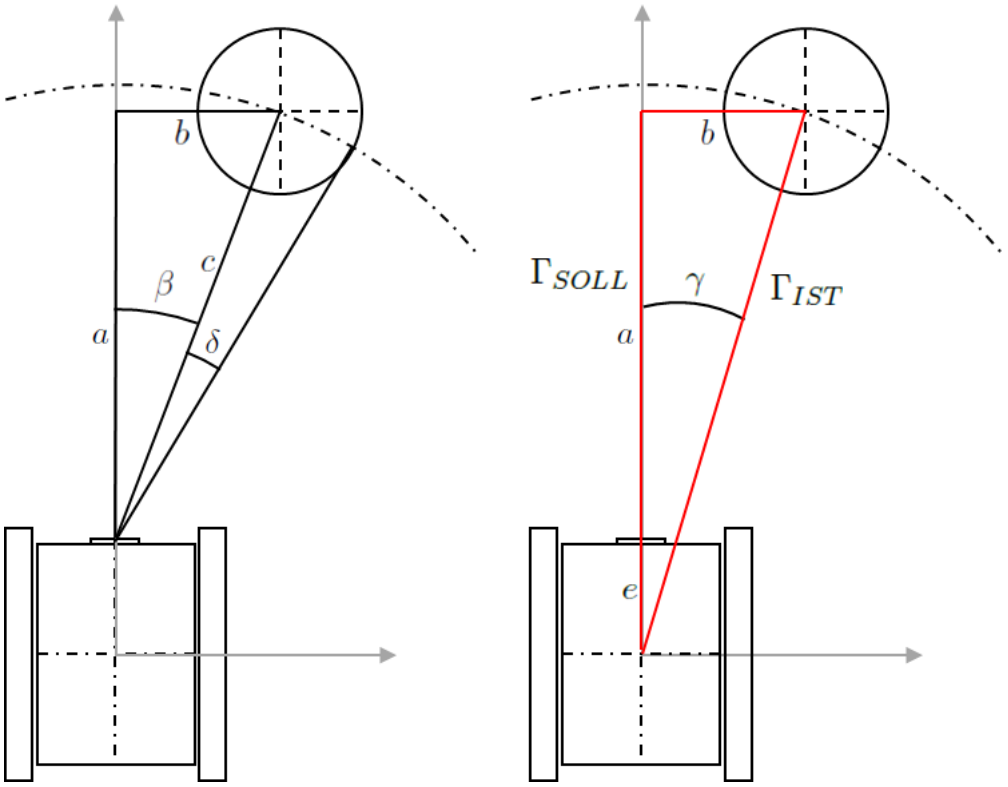


Abbildung 3.37: Position des Kreisen von der Plattform

Die Abbildung 3.37 verbildlicht die Situation wenn der Kreis im Bildfeld der Kamera ist. Die Variable γ ist der Winkel, um den sich die Plattform drehen soll, um sich auf den Kreis auszurichten, $beta$ ist der Winkel zwischen der Blickrichtung der Kamera und dem Mittelpunkt des Kreises von der Kamera aus gesehen, δ ist der Winkel zwischen dem Kreismittelpunkt und dem Kreisrand auch von der Kamera aus gesehen und die Variable c ist der Abstand zum Kreis. Die Variablen a und b werden mit Hilfe des Kosinus bzw. Sinus aus β und c berechnet. Dann wird aus ihnen und der Variable e mit dem Tangens am Ende γ ausgerechnet, wie im roten rechtwinkligen Dreieck dargestellt (siehe Abb. 3.37 rechts).

Die Variablen α, f (siehe Abb. 3.34), der Abstand p, der Radius k (siehe Abb. 3.35), der tatsächlicher Radius r des Kreises und der Abstand e sind bekannt.

$$\beta = \frac{\alpha}{f} \cdot p \tag{3.27}$$

$$\delta = \frac{\alpha}{f} \cdot k \tag{3.28}$$

$$c = \frac{r}{\tan \delta} \tag{3.29}$$

$$b = \sin \beta \cdot c; \tag{3.30}$$

$$a = \cos \beta \cdot c \tag{3.31}$$

$$\gamma = \arctan(\frac{b}{a+e}) \tag{3.32}$$

Die Gleichungen (3.27) bis (3.32) zeigen die schrittweise Berechnung von γ. Zuerst werden p und k in die Winkel β und δ umgerechnet. Mit dem Winkel δ wird der Abstand c zum Kreis berechnet (siehe Gleichung (3.29)), was die Hypotenuse darstellt. Mit Hilfe des Sinus wird aus c und β die Gegenkathete b berechnet und mit Hilfe des Kosinus die Ankathete a. Am Ende wird Tangens angewendet und aus b, a und e die Regelabweichung γ ausgerechnet. Dieser Winkel wird dann der Winkel-Zeit-Funktion (siehe Gleichung (3.14)) übergeben, um die Zeit zu bestimmen, wie lange die Plattform drehen soll.

```
1 double Hilfsfunktionen::pixelToDegree(int pixel) {
2     return (FRAME_WIDTH_DEGREE/FRAME_WIDTH)*pixel;
3 }
4
5 double Hilfsfunktionen::distanceToCircle(int radiusP) {
6     double delta = pixelToDegree(radiusP);
7     return RADIUS/tan(degreeToRadian(delta));
8 }
```

Der obere Codeabschnitt zeigt die Berechnungen des Winkels δ und des Abstandes D_{IST} (siehe Abb. 3.36) oder die Berechnungen der Winkel β, δ und des Abstandes c (siehe Abb. 3.37).

```
double Hilfsfunktionen::getGamma(double radiusP, double positionP) {
    double beta = pixelToDegree(positionP);
    double c = distanceToCircle(radiusP);
    double b = oppositeSide(beta, hypotenuse);
    double a = adjacentSide(beta, hypotenuse);
    return radianToDegree(atan(b/(a + B)));
}

double Hilfsfunktionen::adjacentSide(double angle, double hypotenuse) {
    return cos(degreeToRadian(angle))*hypotenuse;
}

double Hilfsfunktionen::oppositeSide(double angle, double hypotenuse) {
    return sin(degreeToRadian(angle)) * hypotenuse;
}
```

Der obere Codeabschnitt zeigt die Berechnung der Regelabweichung γ, wie in den Gleichungen (3.27) bis (3.32) beschrieben.

Da sich die Plattform nicht vollständig auf ein Kreis ausrichten kann, sondern unter einem Toleranzwinkel, muss das Verhalten aus Abbildung 3.36 erweitert und die Formeln dementsprechend angepasst werden.

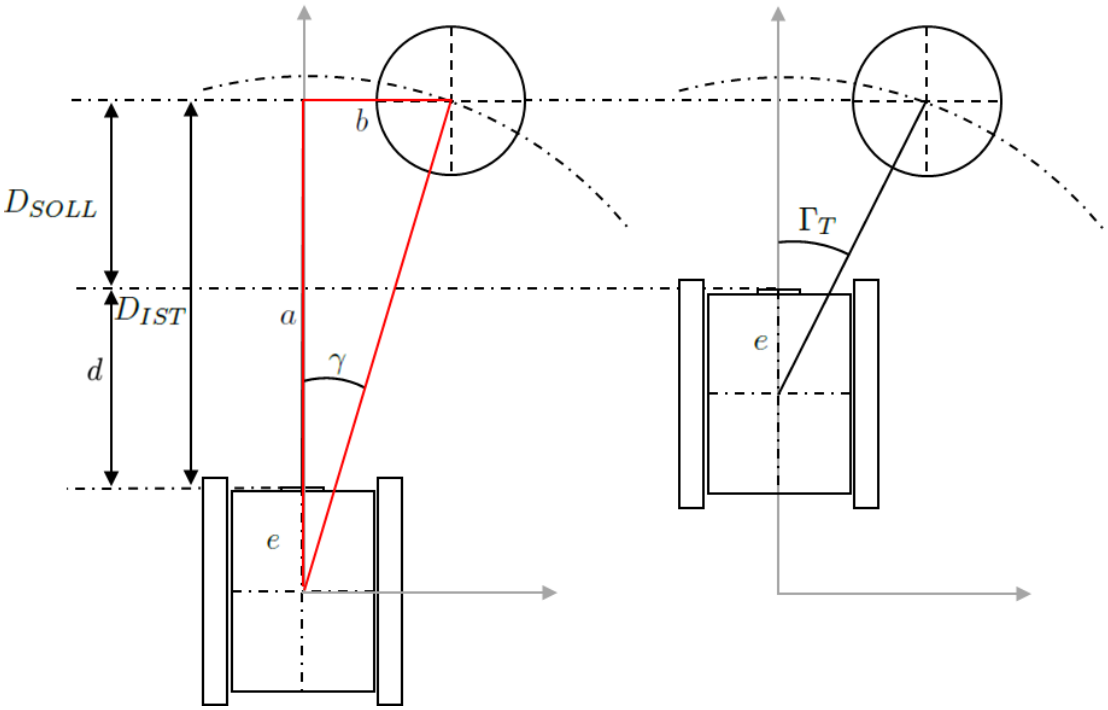


Abbildung 3.38: Tatsächliche Situation beim Ausrichten auf ein Kreis

Die Abbildung 3.38 zeigt, wie sich die Plattform tatsächlich verhält, wenn sie sich auf ein Kreis ausgerichtet hat, sodass gilt $\gamma < \Gamma_T$. Dadurch müssen die Variablen D_{SOLL} und D_{IST} anders berechnet werden als aus Abbildung 3.36. D_{SOLL} entspricht dann dem Abstand, bei dem der Winkel γ den Toleranzwinkel Γ_T nicht überschreiten würde. Dadurch fährt die Plattform nicht direkt zum Kreis, sondern muss auf dem Weg dorthin stoppen und korrigieren. Die Formeln für die Berechnungen von D_{IST} und D_{SOLL} sind die Gleichen wie aus Gleichungen (3.27) bis (3.32). D_{IST} entspricht der Variablen a (siehe Gleichung (3.31)) und wenn b berechnet wurde (siehe Gleichung (3.30)) wird dann D_{SOLL} mit der folgenden Formel berechnet.

$$D_{SOLL} = \frac{b}{\tan \Gamma_T} - e \tag{3.33}$$

Die Gleichung 3.33 ist eine Umformung der Gleichung 3.32. Der folgender Codeabschnitt zeigt die Berechnungen von D_{SOLL} und D_{IST}.

```
double Hilfsfunktionen::sollDistance(double radiusP, double positionP) {
        double beta = pixelToDegree(abs(positionP));
        double c = distanceToCircle(radiusP);
        double b = oppositeSide(beta, c);
        return (b/tan(degreeToRadian(WINKEL_TORERANZ))) - B;
}

double Hilfsfunktionen::istDistance(double radiusP, double positionP) {
        double beta = pixelToDegree(abs(positionP));
        double c = distanceToCircle(radiusP);
        return adjacentSide(beta, c);
}
```

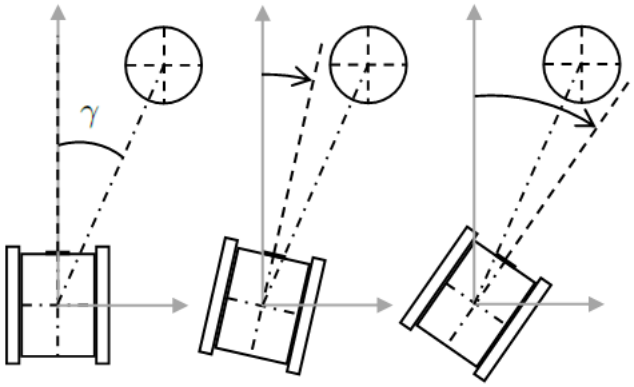


Abbildung 3.39: Drehen der Plattform

Die Abbildung 3.39 zeigt den Drehvorgang der Plattform, wobei links die Ausgangsposition ist. Werden die Variable $P_{\gamma 1}$, $P_{\gamma 2}$, $S_{\gamma 2}$ (siehe Gleichung (3.12)) zu klein gewählt, dreht sich die Plattform zu kurz und erreicht nicht die gewünschte Position, wie das Bild in der Mitte es verdeutlicht. Werden $P_{\gamma 1}$, $P_{\gamma 2}$, $S_{\gamma 2}$ zu groß gewählt, dreht sich die Plattform zu lang und verpasst die gewünschte Position, wie rechts in der Abbildung.

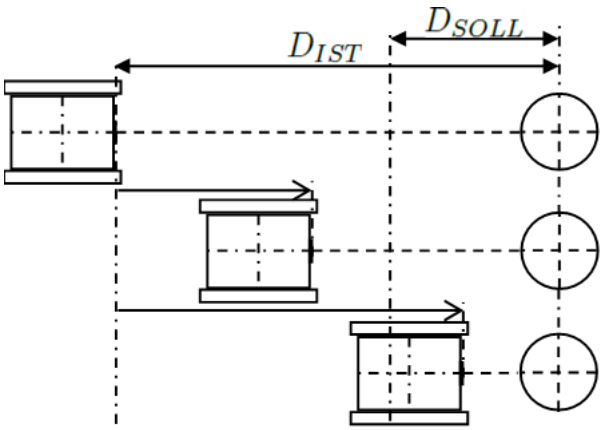


Abbildung 3.40: Anfahren der Plattform

Die Abbildung 3.40 zeigt den Vorgang, bei dem die Plattform zu einem Kreis fährt. Die Zeichnung oben zeigt die Ausgangsposition. Die Variable D_{SOLL} ist der gewünschte Abstand vom Kreis, bei dem die Plattform stoppen soll. Auch hier ist es notwendig die Variablen P_{d1}, P_{d2} und S_{d2} aus Gleichung 3.22 nicht zu klein, sonst fährt die Plattform zu wenig, wie die mittlere Zeichnung verdeutlicht, oder nicht zu groß zu wählen, sonst, wie das untere Bild zeigt, fährt die Plattform über den gewünschten Abstand hinaus.

3.5 Kommunikation

Die Kommunikation zwischen den Komponenten erfolgt über Queues $HQueue$. Jede Queue hat eine eindeutige Identifikationsnummer. Die Kreiserkennung hat zwei Queues, eine für das Empfangen der Nachrichten von der Navigation und eine für das Senden der Nachrichten an die Navigation. Die Navigation hat drei Queues, eine für das Senden der Nachrichten an die Kreiserkennung, eine für das Empfange der Nachrichten von der Kreiserkennung und die letzte für das Senden der Nachrichten an den Motorkontroller. Der Motorkontrolle hat nur eine Queue fürs Empfangen. Eine Faktory erstellt oder liefert schon vorhandene Queues.

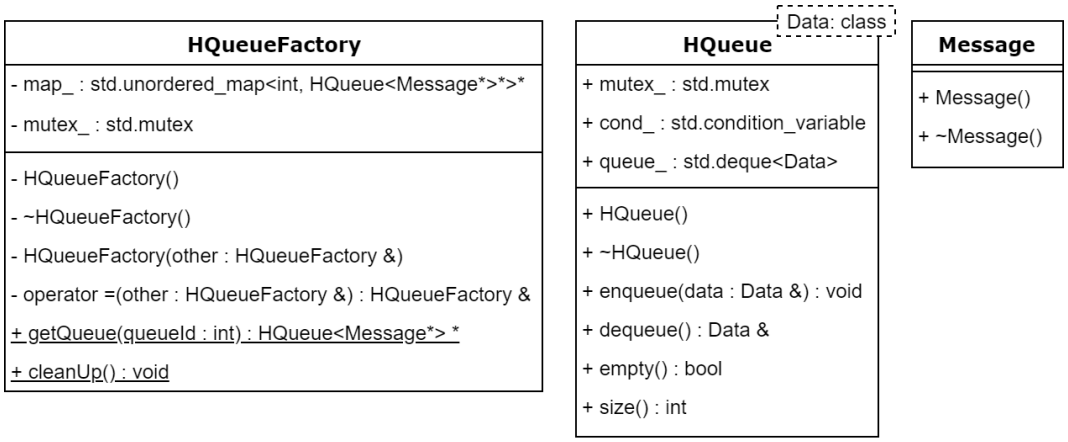


Abbildung 3.41: Fabrikmethode für die Queues

In Abbildung 3.41 ist aufgeführt, wie die Queues erstellt werden. Die $HQueueFactory$ bekommt eine *Queue-Id* als Schlüssen und schaut in seiner *HashMap*, ob zu diesem Schlüssel schon ein Queue hinterlegt ist und gibt diese dann zurück. Wenn er keine Queue findet, erstellt er eine neue und trägt sie in die *HashMap* mit dem dazugehörigen Schlüssel *Queue-Id* ein. Das Erstellen und Zurückliefern der Queues wird durch ein *Mutex* geschützt, um nicht versehentlich zwei verschieden Queues für ein und der selben *Queue-Id* zu erstellen. Bei der $HQueue$ wurde die Queue $deque$ der Standardbibliothek verwendet. Die Queue $HQueue$ hat die Funktionen $enqueue$ und $dequeue$, bei denen in die $deque$ etwas reingelegt $push\_back$ oder aus ihr etwas herausgenommen $front$, pop_front wird. Ein *Mutex* schützt die Operationen an der $deque$ und eine Bedingungsvariable sorgt dafür, dass die $dequeue$ blockierend ist, sodass ein *Thread* warten muss bis in die Queue etwas reingelegt wird. Alle Nachrichten in den Queues sind von Type $Message$ und die eigentlichen Nachrichten müssen dann nur von $Message$ erben. Wenn eine $HQueue$ aus der $HQueueFactory$ herausgenommen wird, wird sie auf die Queue mit dem zu erwarteten Type von Nachrichten gecastet, wie folgender Codeabschnitt zeigt.

```
1 (HQueue<TrackerMessage*>*) HQueueFactory::getQueue(TRACKER_QUEUE_ID);
```

Alle Nachrichten und die Queues werden auf dem HEAP hinterlegt.

3.5.1 Nachrichtentypen

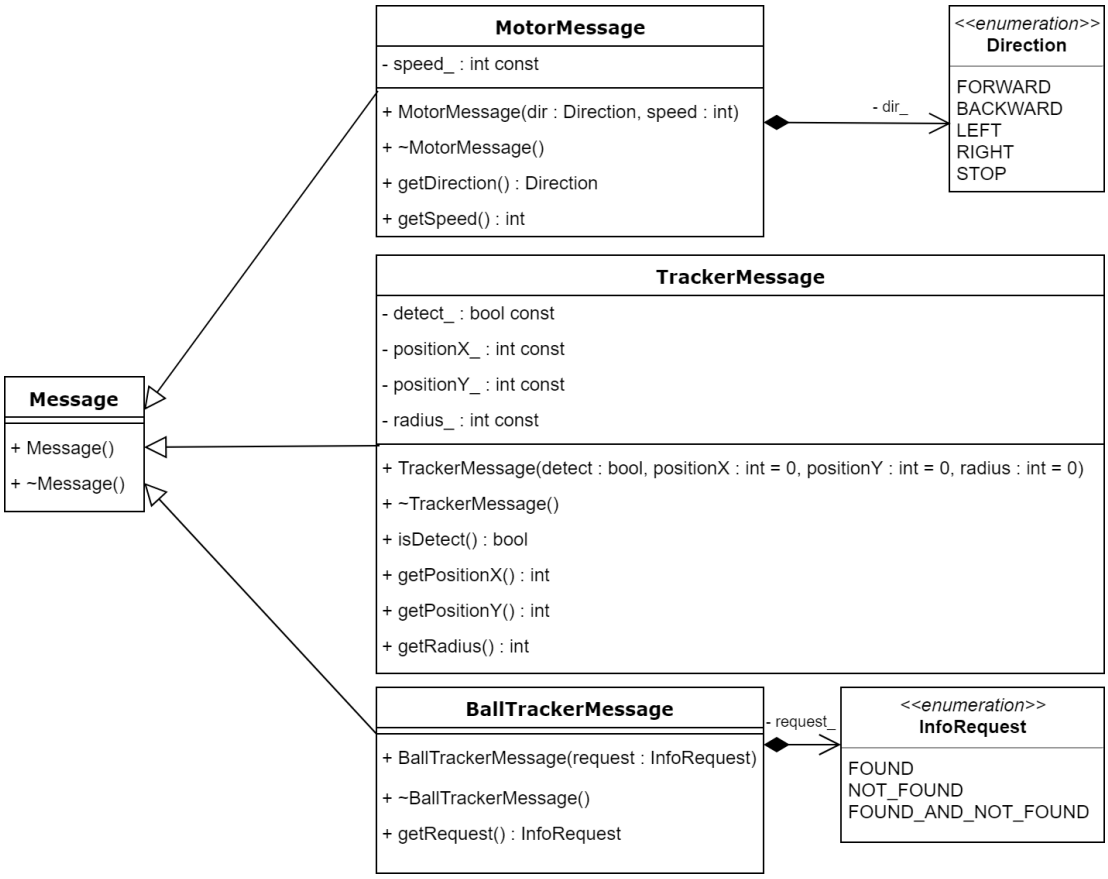


Abbildung 3.42: Drei verschiedene Nachrichtentypen

Die Abbildung 3.42 zeigt die drei verschiedenen Nachrichten $BallTrackerMessage$, $TrackerMessage$ und $MotorMessage$. $MotorMessage$ wird an den Motorkontroller gesendet und hat zwei Werte. Ein Wert gibt die Richtung an, wie vorwärts, rückwärts, links und rechts, aber auch, ob der Motor stoppen soll. Der andere Wert gibt die Geschwindigkeit an. $TackerMessage$ wird an die Navigation gesendet und beschreibt Position des Kreises und sein Radius, als auch, ob ein Kreis gefunden wurde. Und in $BallTrackerMessage$ steht die Art der Anfrage an die Kreiserkennung. Diese Anfragen sind: Sende mir eine Nachricht, wenn du ein Kreis gefunden hast, sende mir eine Nachricht, wenn du kein Kreis gefunden hast oder sende mir in beiden Fällen eine Nachricht. Das hat zufolge, dass die Navigation keine Nachrichten bekommt, wenn sie noch beschäftigt ist.

4 Tests und Analyse

In diesem Kapitel werden die durchgeführten Tests beschrieben. Die Ergebnisse dieser Tests werden dann analysiert und beurteilt.

4.1 Versuchsaufbau

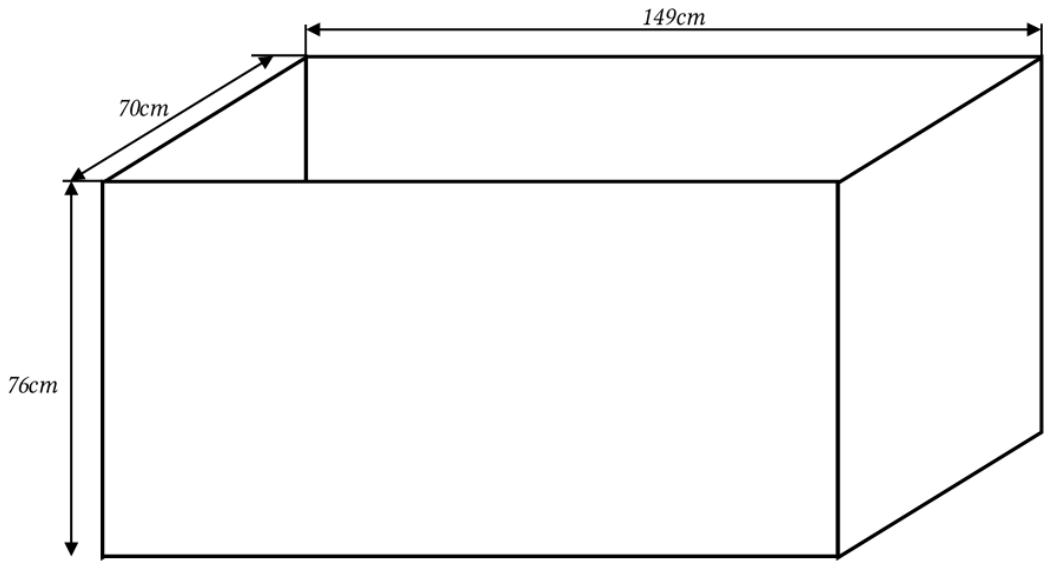


Abbildung 4.1: Testumgebung

Die Abbildung 4.1 zeigt die Testumgebung, in der sich die Plattform bewegt. Es handelt sich um einen Kasten aus Karton mit monotoner Farbe. Die Höhe, Breite und Länge wurden so gewählt, dass das Kameramodul der Plattform nicht über den Kasten schauen kann. Der Zweck dieser Umgebung ist es, das Erkennen von falschen Kreisen durch die Gegenstände in dem Raum zu eliminieren.

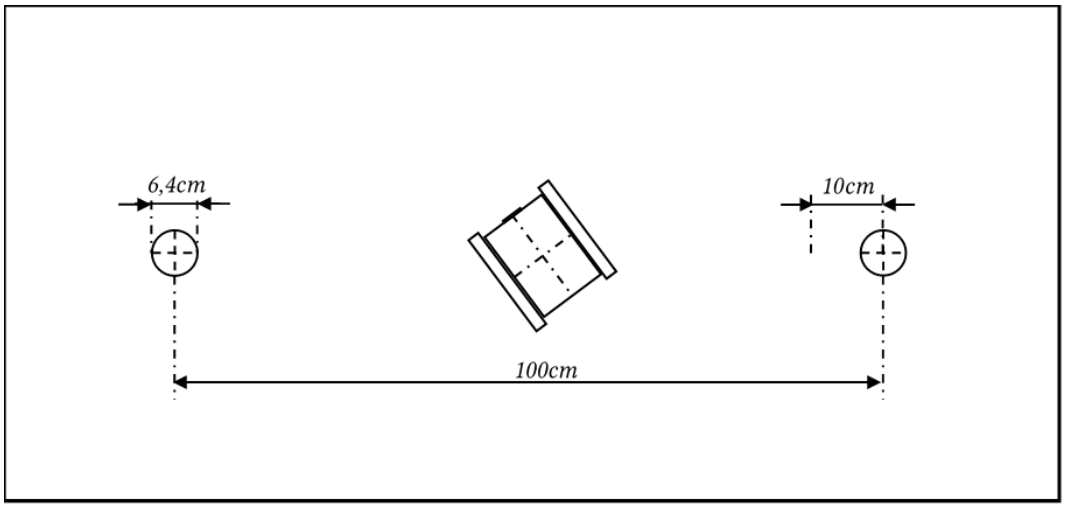


Abbildung 4.2: Testaufbau

In Abbildung 4.2 ist der Aufbau im Inneren der Testumgebung dargestellt. Dieser Aufbau wird bei allen Tests verwendet. Es wurden zwei Schilder mit jeweils einem Kreis gegenübergestellt. Einer der Kreise ist rot und der andere ist blau. Der Durchmesser der Kreise beträgt $6,4cm$. Die Kreise befinden sich in einer Entfernung von einem Meter zueinander, auf der gleichen Höhe wie die Kamera an der Plattform. Die $10cm$ ist der Mindestabstand, bei dem die Plattform stoppen soll.

4.2 Tests

Test 1	Regler Drehen	$\Delta t(\gamma) = \begin{cases} 6,0606 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma & \text{für } \gamma < 16,5^\circ \\ 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s & \text{für } 16,5^\circ \leq \gamma \end{cases}$
	Regler Fahren	$\Delta t(d) = \begin{cases} 32,154 \cdot 10^{-3} \frac{s}{cm} \cdot d & \text{für } d < 3,11cm \\ 16,604 \cdot 10^{-3} \frac{s}{cm} \cdot d + 48,36 \cdot 10^{-3} s & \text{für } 3,11cm \leq d \end{cases}$
	Beschreibung	Für den ersten Test werden die Regler-Funktion für das Drehen und Fahren, wie in Unterabschnitt 3.4.3 beschrieben, verwendet.
Test 2	Regler Drehen	$\Delta t(\gamma) = 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s$
	Regler Fahren	$\Delta t(d) = 16,604 \cdot 10^{-3} \frac{s}{cm} \cdot d + 48,36 \cdot 10^{-3} s$
	Beschreibung	Wie in Test 1, aber nur jeweils die zweite Funktion ohne die Fallunterscheidung.
Test 3	Regler Drehen	$\Delta t(\gamma) = 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s$
	Regler Fahren	$\Delta t(d) = 16,604 \cdot 10^{-3} \frac{s}{cm} \cdot d + 38,69 \cdot 10^{-3} s$
	Beschreibung	Wie in Test 2, aber der Startwert der Regler-Funktion für das Fahren ist 20% weniger.
Test 4	Regler Drehen	$\Delta t(\gamma) = 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s$
	Regler Fahren	$\Delta t(d) = 16,272 \cdot 10^{-3} \frac{s}{cm} \cdot d + 48,36 \cdot 10^{-3} s$
	Beschreibung	Wie in Test 2, aber die Steigung der Regler-Funktion für das Fahren ist 2% weniger.
Test 5	Regler Drehen	$\Delta t(\gamma) = 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s$
	Regler Fahren	$\Delta t(d) = 15,773 \cdot 10^{-3} \frac{s}{cm} \cdot d + 48,36 \cdot 10^{-3} s$
	Beschreibung	Wie in Test 2, aber die Steigung der Regler-Funktion für das Fahren ist 5% weniger.
Test 6	Regler Drehen	$\Delta t(\gamma) = 4,1667 \cdot 10^{-3} \frac{s}{\circ} \cdot \gamma + 31,25 \cdot 10^{-3} s$
	Regler Fahren	$\Delta t(d) = 14,944 \cdot 10^{-3} \frac{s}{cm} \cdot d + 48,36 \cdot 10^{-3} s$
	Beschreibung	Wie in Test 2, aber die Steigung der Regler-Funktion für das Fahren ist 10% weniger.

Tabelle 4.1: Regler-Funktionen

Die Tabelle 4.1 zeigt, welche Funktionen bei den jeweiligen Tests zum Einsatz kommen. Die Steigungen und Startwerte der Funktionen sind die Parameter, die für die Tests verändert werden. Die Veränderungen dieser Parameter wurden empirisch ermittelt. Als Ausgangsfunktionen dienen die Gleichung 3.14 und die Gleichung 3.24. Die Annahme ist, dass die Plattform

stabiler fährt, wenn sie kürzere Strecken zurücklegt, also kürzere Zeit fährt. Dafür müssen die Parameter aus den Ausgangsfunktionen reduziert werden, wenn die Plattform die Kreise verliert. Um die Funktionen zu vereinfachen, wird die Fallunterscheidung, außer in Test 1, herausgenommen und jeweils die Funktion aus dem zweiten Fall betrachtet.

4.3 Durchführung

Die Plattform wird zwischen den Kreisen platziert und das Programm gestartet. Sie fährt von einem Kreis zum anderen. Es werden zwei Phasen des Fahrens gemessen. Die erste Phase befindet sich zwischen der Situation, wo die Plattform den Mindestabstand erreicht hat, kurz MAE, und der Situation, wo die Plattform den nächsten Kreis gefunden hat, kurz NKG. MAE tritt ein, wenn die Plattform bis zu einem bestimmten Abstand zu einem Kreis hingefahren ist. NKG tritt ein, wenn nach MAE die Plattform den nächsten Kreis registriert, nachdem sie sich gedreht hat. Die zweite Phase befinden sich dann zwischen NKG und MAE. Der Mindestabstand wurden auf $10cm$ festgelegt. Besonders wird die Zeit betrachten, die sich aus den beiden Phasen ergibt. Die beiden Phasen zusammen werden als eine Runde bezeichnet. Die Runde geht von MAE bis NKG und dann von NKG bis MAE, somit beschreibt sie das Fahren von einem Kreis zum nächsten. Zu den Zeiten werden auch die Anzahl der Runden in den einzelnen Durchläufen festgehalten. Ein Durchlauf ist das Hin- und Herfahren der Plattform bis sie den nächsten Kreis verliert oder 10 Runden gefahren ist. Zehn Runden sind somit das Abbruchkriterium und ein Durchlauf gilt als erfolgreich, wenn die Plattform diese zehn Runden fährt.
Die Plattform verliert den Kreis dadurch, dass sie nach dem Hinfahren zu einem Kreis zu stark nach rechts abdriftet wird und der Kreis außerhalb des Blickfelds der Kamera geht oder so weit an den Rand des Blickfelds, dass der Kreis nicht mehr als ein Kreis erkannt wird. Das Abdriften der Plattform geschieht zufällig aufgrund eines Defekts im rechten Motor. Nach 50 Durchläufen ist ein Test beendet und ein neuer wird durchgeführt.

4.4 Ergebnisse und Beurteilung

Test 1

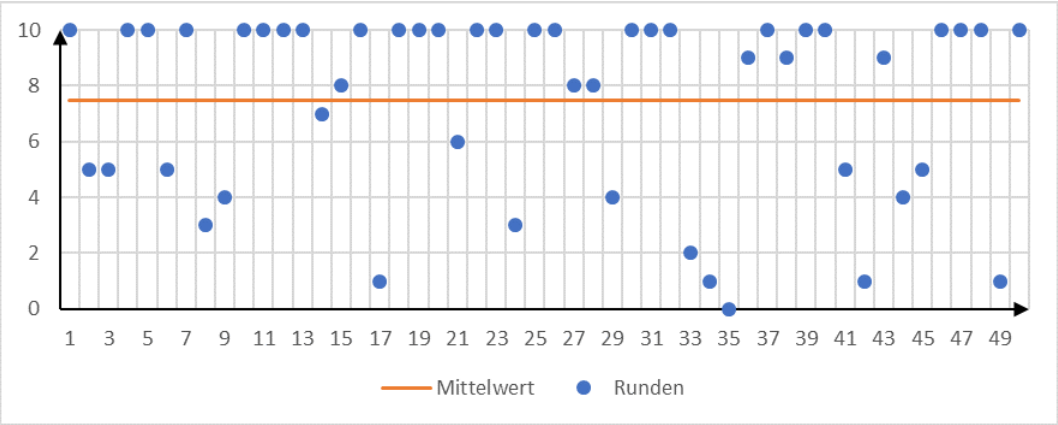


Abbildung 4.3: Anzahl der Runden bei 50 Durchläufen für Test 1

Die Abbildung 4.3 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Eine Runde bedeutet einmal von einem Kreis zum anderen zu fahren. Wie zu erkennen ist, haben nur 26 Durchläufe die 10 Runden erreicht. Das heißt nur 52% der Durchläufe waren erfolgreich. Der Mittelwert bei diesem Test beträgt 7, 46 Runden.

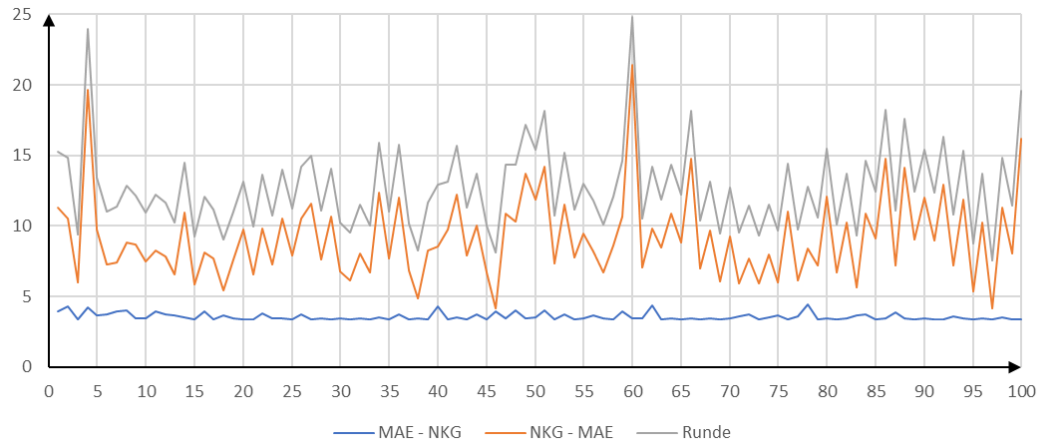


Abbildung 4.4: Benötigte Zeiten für die Runden für Test 1

Die Abbildung 4.4 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $12,73s$ mit einer Standardabweichung von $3,009s$. Wie zu erkennen ist, schwankt die Zeit zwischen NKG bis MAE sehr stark, wobei die Zeit zwischen MAE bis NKG fast kaum schwankt. Der Grund dafür ist, dass das Hinfahren zu einem Kreis viel komplexer ist als das Finden des nächsten Kreises. Das Hinfahren zu einem Kreis ist sehr viel mit Bahnkorrekturen, Auswertungen von Messdaten und Berechnungen verbunden und das Aufspüren des nächsten Kreises hat nur einen festen Wert um den sich die Plattform ständig dreht bis sie feststellt, dass ein Kreis aufgetaucht ist.

Test 2

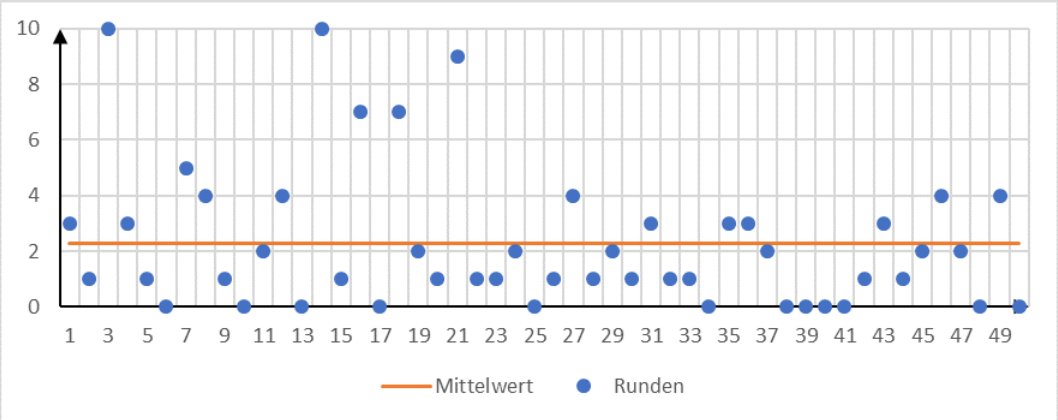


Abbildung 4.5: Anzahl der Runden bei 50 Durchläufen für Test 2

Die Abbildung 4.5 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Wie zu erkennen ist, haben nur zwei Durchläufe die 10 Runden erreicht. Das sind 4% der 50 Durchläufen und $92,31\%$ weniger als im Test 1. Der Mittelwert bei diesem Test beträgt $2,28$ Runden.

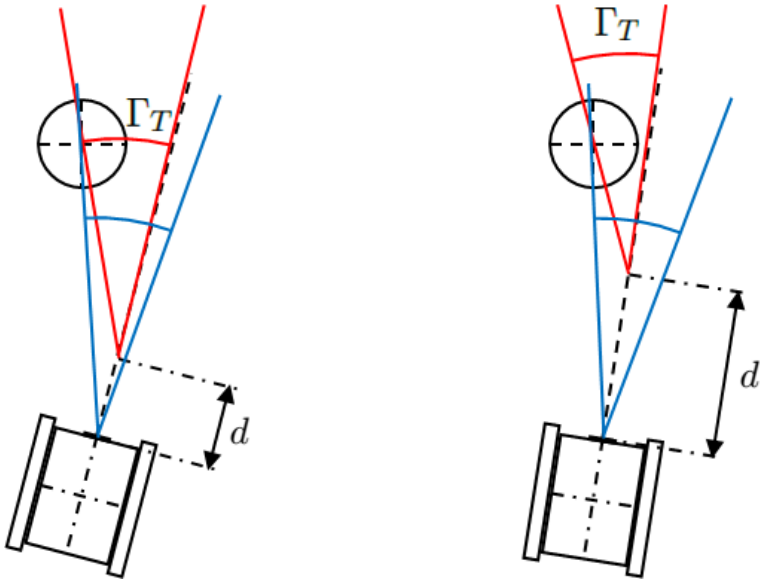


Abbildung 4.6: Fahrstrecke bei großen und kleinen Winkeln

Die Abbildung 4.6 beschreibt, wie weit die Plattform vorwärts fährt, nachdem der Kreis sich innerhalb des Toleranzwinkels befindet. Die roten und blauen Winkel entsprächen dem Toleranzwinkel und sind identisch. Der Toleranzwinkel ist der Winkel, bei dem die Plattform als auf den Kreis ausgerichtet gilt und vorwärts fahren darf. Dieser Winkel wurde auf $2,5°$ gesetzt. Der Unterschied bei den Fahrstrecken besteht darin, dass sich die Plattform einmal näher an der Grenze des Toleranzwinkels befinden (siehe Abb. 4.6 links, blau) und einmal weiter weg von ihm (siehe Abb. 4.6 rechts, blau). Die Stecke d beschreibt die Strecke, bei der der Winkel, der nach dem Fahren entstehen würde, größer als der Toleranzwinkel wird, was der rote Winkel mit Γ_T in Abbildung 4.6 verdeutlicht. Durch die Beschaffenheit der Motoren, fährt die Plattform nicht direkt geradeaus. Der rechte Motor hat einen kleinen Defekt und dreht ab und zu durch. Dadurch kommt es vor, dass die Plattform nach rechts abdriftet. Je weiter die Plattform fährt, desto größer kann die Ablenkung der Fahrstrecke sein. Dadurch kann die Plattform zu weit nach rechts fahren und den Kreis verlieren. Wie in den Formeln zur Test 1 und Test 2 zu erkennen ist, dreht sich die Plattform in Test 1, für kleine Winken, weniger als in Test 2. Durch diese kleinere Drehungen kommt die Plattform näher an der Grenze des Toleranzwinkels, wenn sie ihn unterschreitet und fährt eine kürzere Strecke vorwärts, wie in Abbildung 4.6 links zu erkennen ist. Bei Test 2 macht die Plattform größere Drehungen und ist häufig weiter von der Grenze des Toleranzwinkels entfernt, wenn sie in unterschreitet, dadurch fährt sie weiter (siehe Abb. 4.6 rechts), kann stärker abgelenkt werden und verliert somit häufiger den Kreis.

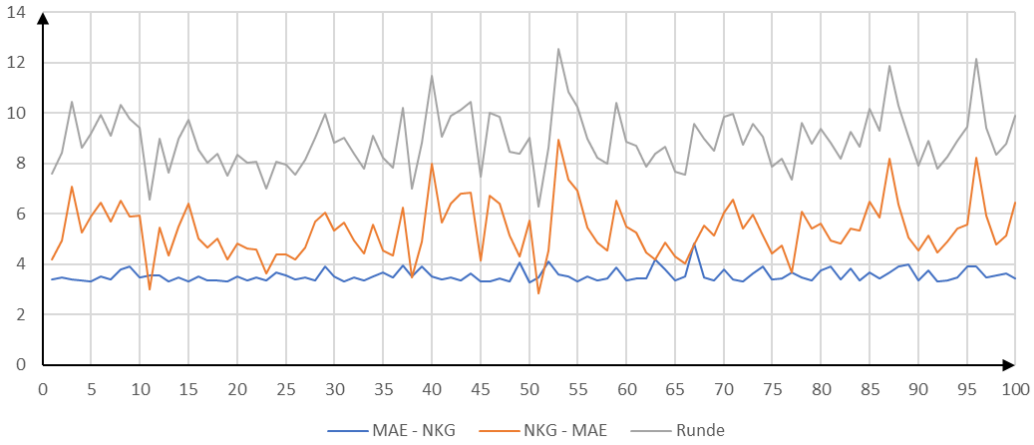


Abbildung 4.7: Benötigte Zeiten für die Runden für Test 2

Die Abbildung 4.7 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $8,92s$ mit einer Standardabweichung von $1,122s$. Durchschnittlich hat sie für die Runden $29,91\%$ weniger gebraucht als in Test 1. Auch dies ist damit zu erklären, dass die Plattform in Test 1 für kleine Winkel, weniger dreht als in Test 2. Dadurch braucht sie in Test 1 deutlich länger um in den Toleranzwinkel zu kommen. Das Gleiche gilt auch für das Hinfahren auf ein Kreis, wo sie in Test 1 für kleine Abstände kürzere Strecken fährt.

Test 3

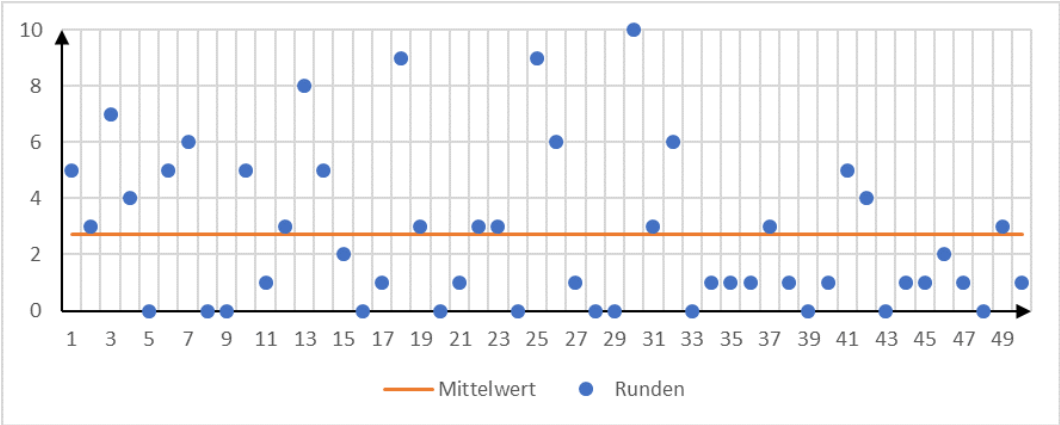


Abbildung 4.8: Anzahl der Runden bei 50 Durchläufen für Test 3

Die Abbildung 4.8 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Eine Runde bedeutet einmal von einem Kreis zum anderen zu fahren. Es hat nur ein Durchlauf die 10 Runden erreicht. Der Mittelwert bei diesem Test beträgt 2, 7 Runden. Wie zu erkennen ist, hat das Absenken des Funktionsgraphen in Test 3 in Vergleich zu Test 2 um 20% keine nennenswerte Veränderungen in Bezug auf die Anzahl der Runden. Zwar ist der Mittelwert höher als in Test 2, dafür ist die Anzahl der erfolgreichen Durchläufen weniger, was in Anbetracht des zufälligen Abdriftens der Plattform kein Aussagekraft hat.

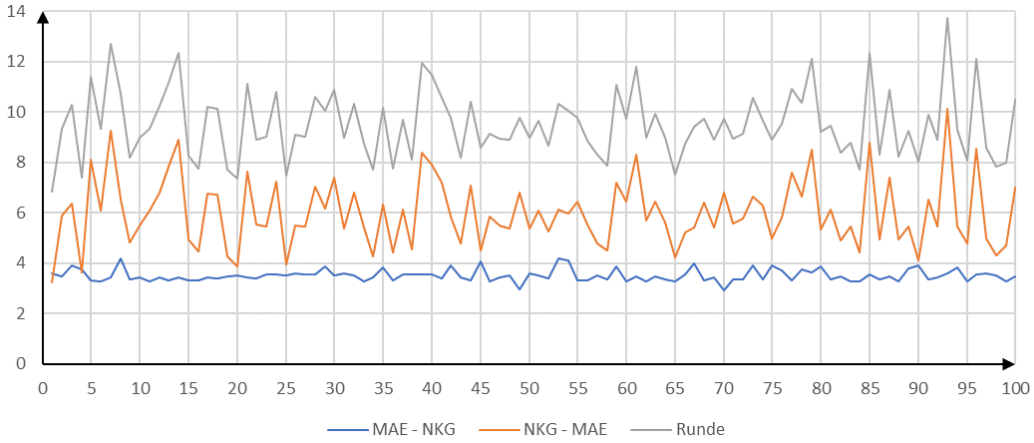


Abbildung 4.9: Benötigte Zeiten für die Runden für Test 3

Die Abbildung 4.9 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $9,508s$ mit einer Standardabweichung von $1,344s$. Durchschnittlich hat sie für die Runden $6,59\%$ länger gebraucht als in Test 2.

Test 4

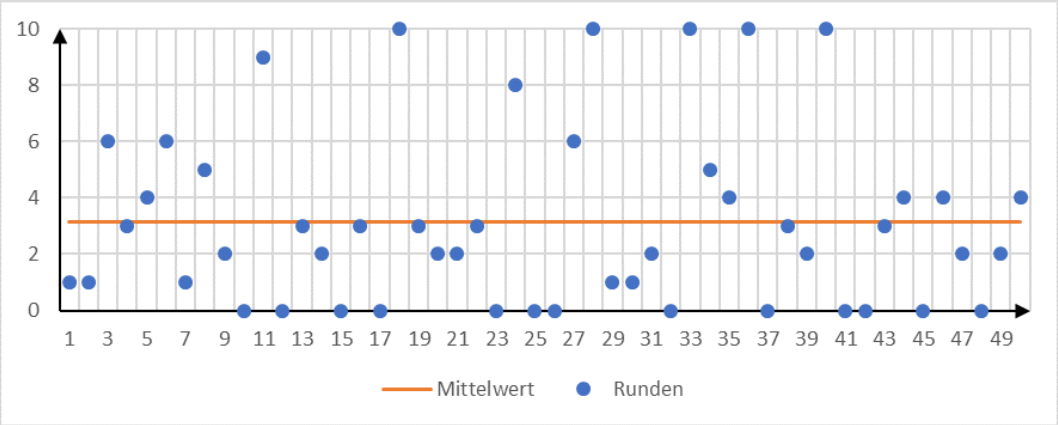


Abbildung 4.10: Anzahl der Runden bei 50 Durchläufen für Test 4

Die Abbildung 4.10 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Eine Runde bedeutet einmal von einem Kreis zum anderen zu fahren. Die Plattform war in der Lage bei 5 von 50 Durchläufen die 10 Runden zu erreichen. Der Mittelwert bei diesem Test beträgt 3, 14 Runden. Das ist mehr als in Test 2 und 3, aber trotzdem zu wenig, als dass dieser Test als Erfolg gelten kann.

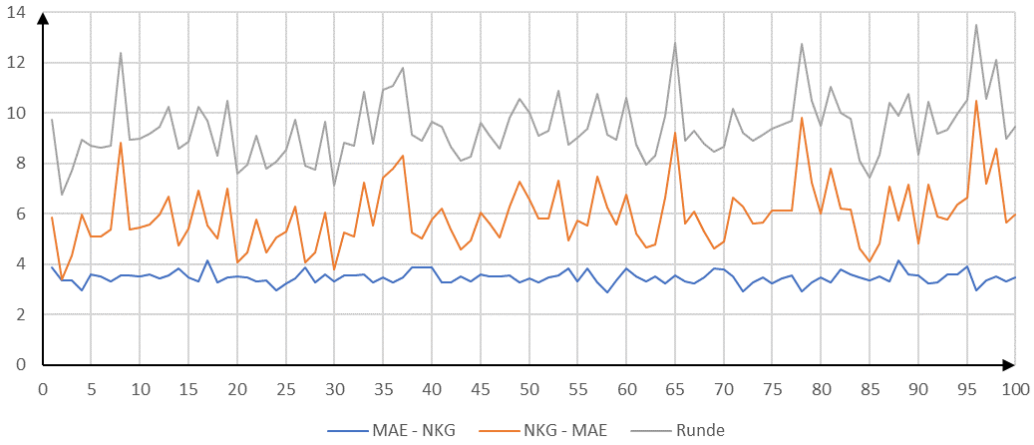


Abbildung 4.11: Benötigte Zeiten für die Runden für Test 4

Die Abbildung 4.11 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $9,413s$ mit einer Standartabweichung von $1,219s$. Die Plattform hat im Durchschnitt 1% weniger als im Test 3 gebraucht, um von einem Kreis zum anderen zu fahren, also eine Runde zu absolvieren. Dieses Verhalten ist damit zu erklären, dass die Plattform zwar öfters vorwärts fährt als in Test 3, aber dafür kürzere Strecken fährt, nicht so stark abdriftet und dadurch weniger drehen muss, um unter dem Toleranzwinkel zu kommen.

Test 5

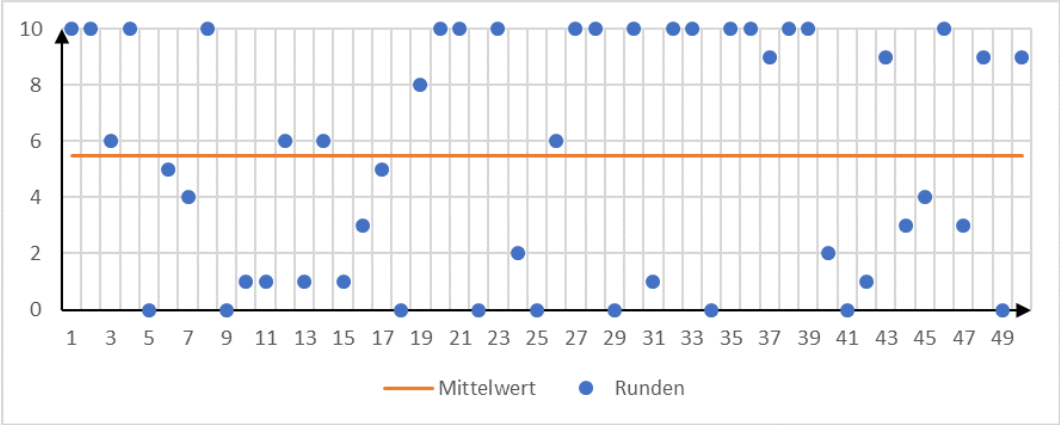


Abbildung 4.12: Anzahl der Runden bei 50 Durchläufen für Test 5

Die Abbildung 4.12 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Eine Runde bedeutet einmal von einem Kreis zum anderen zu fahren. Von 50 Durchläufen haben 17 die 10 Runden erreicht, das sind 3, 4 mal mehr als im Test 4, aber 34, 62% weniger als es in Test 1 der Fall war. Der Mittelwert bei diesem Test beträgt 5, 5 Runden, das sind 11% der gesamten Durchläufen. Dieser Test ist schlechter ausgefallen als Test 1, bei dem knapp die Hälfte der Durchläufe, die 10 Runden geschafft haben.

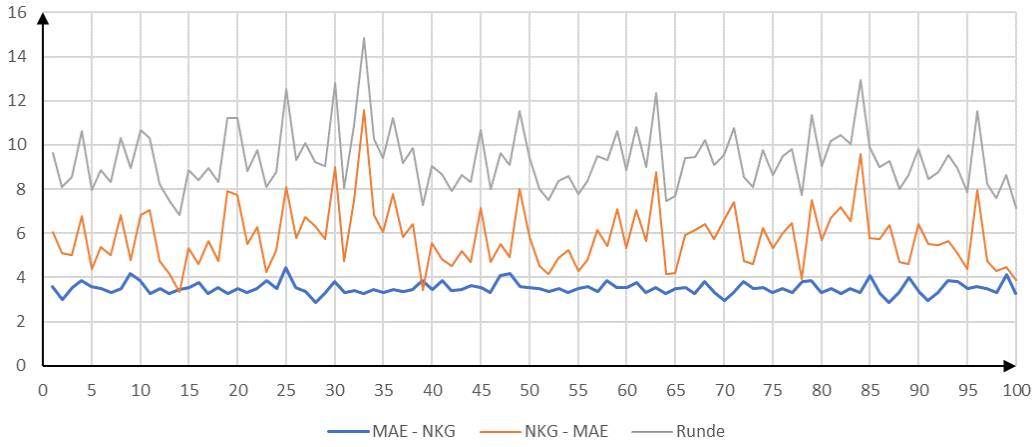


Abbildung 4.13: Benötigte Zeiten für die Runden für Test 5

Die Abbildung 4.13 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $9,325s$ mit einer Standardabweichung von $1,387s$. Das ist wieder knapp 1% weniger als in dem Test davor, also Test 4. Die Erklärung für das Verhalten ist der Selber, wie in Test 4.

Test 6

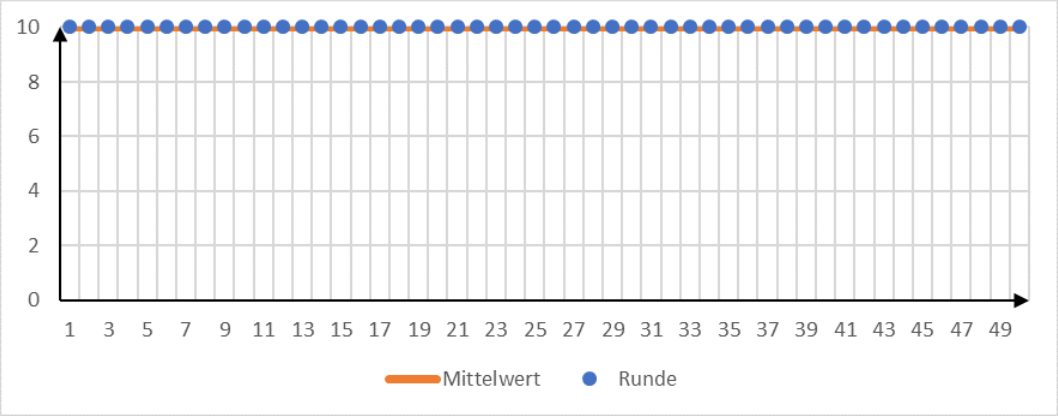


Abbildung 4.14: Anzahl der Runden bei 50 Durchläufen für Test 6

Die Abbildung 4.14 zeigt einen Graphen mit der Anzahl der Runden bei 50 Durchläufen. Zehn Runden waren das Abbruchkriterium. Eine Runde bedeutet einmal von einem Kreis zum anderen zu fahren. Bei jedem der 50 Durchläufen wurde die 10 Runden erreicht. Dadurch sind die Funktionen, die für diesen Test gewählt wurden, von allen anderen Tests am Besten geeignet, um das Abdriften und somit das verlieren des Kreises zu korrigieren.

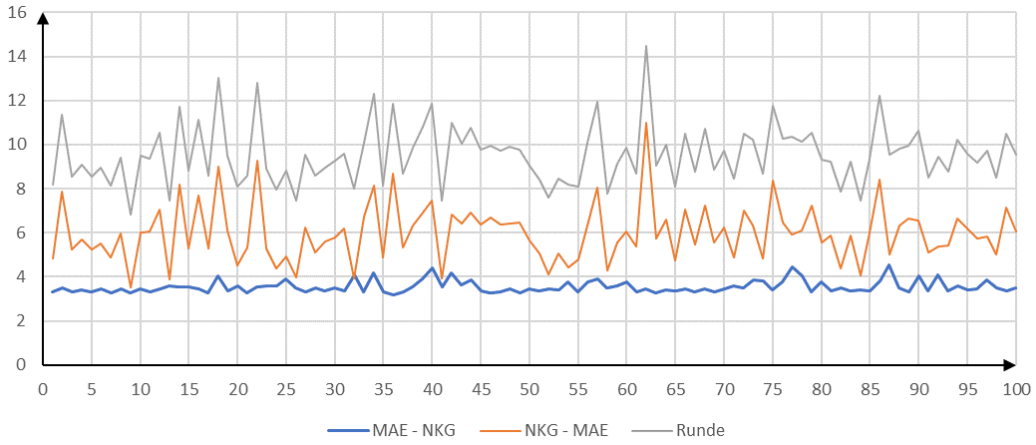


Abbildung 4.15: Benötigte Zeiten für die Runden für Test 6

Die Abbildung 4.15 zeigt die Zeiten in Sekunden an, wie lange die Plattform für die Runden gebraucht hat. Es wurden 100 Runden aufgenommen. In den aufgenommenen Messungen wird unterschieden zwischen Mindestabstand erreicht (MAE) bis nächster Kreis gefunden (NKG), nächster Kreis gefunden bis Mindestabstand erreicht und Runde. Der Mittelwert für die Runden beträgt $9,545s$ mit einer Standardabweichung von $1,355s$. In diesem Fall hat die Plattform länger gebraucht als in Test 2 bis Test 5, wobei der Unterschied zu Test 3 $0,39\%$ beträgt. Das ist damit zu erklären, dass die Tatsache, dass er bei jedem Fahrschritt vorwärts, kürzere Strecken zurücklegt, weniger abdriftet und dadurch weniger drehen muss, die Häufigkeit der Fahrschritte vorwärts nicht mehr kompensieren kann und somit das Fahren zum Kreis länger dauert.

Vergleich der Tests

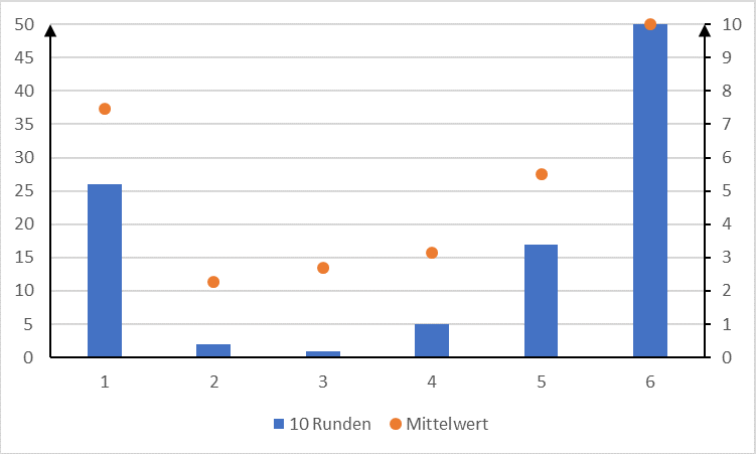


Abbildung 4.16: Vergleich der Test für die Runden

In Abbildung 4.16 ist die Anzahl der Durchläufe, bei denen die 10 Runden erreicht wurde, und die durchschnittliche Anzahl der erreichten Runden für die einzelnen Tests dargestellt. Die untere Achse zeigt die Nummer des Tests. Die linke Skala stellt die Anzahl der Durchläufe dar, zu der die blauen Balken gehören. Die rechte Skala stellt die Mittelwerte der erreichten Runden in den 50 Durchläufen dar, wobei die 10, da 10 Runden das Abbruchkriterium war, den größte Wert darstellt. Es ist zu erkennen, dass die Mittelwerte und die Anzahl der Durchläufe, die die 10 Runden erreicht haben, abgesehen von Test 1, steigen. Test 1 bis 3 müssen gesondert betrachtet werden, weil Test 4 bis 6 zusammenhängen, da in diesen Tests die Steigung der Funktion verändert wurden. Es wird einleuchtend, dass die Veränderung der Funktion für das Vorwärtsfahren zufolge hat, dass je kürzer die Plattform vorwärts fährt, desto kleiner ist die Ablenkung in die Fahrtrichtung und desto weniger die Wahrscheinlichkeit, dass der Kreis aus dem Blickfeld verschwindet und verloren geht. Sind also die Werte für die Fahr-Funktion klein, fährt die Plattform häufig und kürzere Strecken und der Kreis bleibt im Blickfeld.

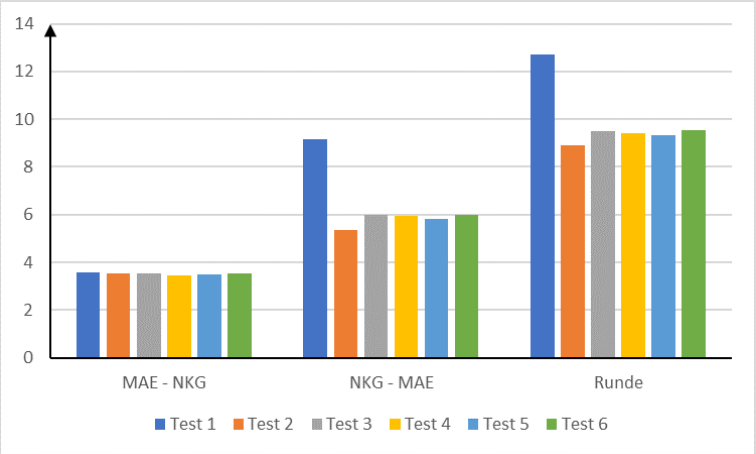


Abbildung 4.17: Durchschnitt der Zeiten bei den Fahrabschnitten für die sechs Tests

Die Abbildung 4.17 zeigt, wie die einzelnen Tests durchschnittlich abschneiden in Bezug auf die Fahrt zwischen Mindestabstand erreicht (MAE) und nächster Kreis gefunden (NKG) (links), nächster Kreis gefunden und Mindestabstand erreicht (mittig) und die gesamte Runde (rechts). Besonders auffallend ist der Balken für Test 1. Das Verhalten, das den ersten Balken erklärt ist unter den Punkten Test 1 und Test 2 beschrieben worden. Für MAE bis NKG bewegen sich die Zeiten von $3,469s$ bis $3,573s$, das entspricht einem Differenzbereich von $0,104s$, also 104 Millisekunden. Das kommt dadurch, weil der Wert für das Drehen um den nächsten Kreis zu finden nicht verändert wird. Für das Hinfahren zu einem Mindestabstand (siehe Abb. 4.17 mittig) sieht es ein wenig anders aus. Wenn der Test 1 nicht betrachtet werden soll, bewegen sich die Zeiten von $5,365s$ bis $5,996s$, was einen Differenz von 631 Millisekunde entspricht.

5 Fazit und Ausblick

In diesem Kapitel werden die Ergebnisse der Analyse zusammengefasst und eine abschließende Beurteilung abgegeben. Außerdem werden Optimierungen vorgeschlagen und Erweiterungen in Hinblick auf die Zukunft vorgestellt.

5.1 Fazit

Es wurde festgestellt, dass das Einstellen der Parameter, wie in Test 6, das Verlieren des Kreises verhindert. Da die Abweichungen der Zeiten bei den Tests 2 bis 6 sich in Millisekundenbereichen befinden und die Plattform keine harten Deadlines hat, müssen bei dem Einstellen der Parameter die Fahrzeiten nicht berücksichtigen werden. Bei der Wahl der Parameter sollte nur darauf geachtet werden, dass die Plattform die Kreise nicht aus ihrem Sichtfeld verliert. Obwohl die Plattform mit den geeigneten Parameter die Kreise nicht verliert, können trotzdem noch Sicherheitsmechanismen eingebaut werde, falls sie die Kreise doch verliert. Einer dieser Mechanismen könnte das Schauen nach links und rechts sein, dabei sollte darauf geachtet werden, dass die Kreise nur nach einer bestimmten Zeit als verloren gelten. Die visuelle Navigation anhand von Kreisen hat sich als eine relative einfache Realisierung der Navigation herausgestellt, besonders in Bezug auf das visuelle Aufspüren des Zielobjektes und das Verändern der Fahrumgebung. Die Fahrumgebung kann sehr einfach durch das Umstellen der Landmarken verändert werden.

5.2 Ausblick

Obwohl in dieser Arbeit nur das Fahren der Plattform zwischen zwei Kreisen mit einer von der Größe her nicht wirklich kleinen Plattform betrachtet wurde, gibt es viele Möglichkeiten für Optimierungen und Erweiterungen. Das System kann einfach um mehr Farben für die Kreise erweitert werden. Dadurch kann das Fahren eines bestimmten Weges realisiert werden, wo die Plattform von einem Kreis zum nächsten diesen dann zurücklegt. Diese Kreise dienen dadurch als Landmarken in einer unbekannten Umgebung und können fast beliebig platziert werden. Da das System Kugeln von Kreisen nicht unterscheiden kann, können auch diese verwendet

werden. Das hat den Vorteil, dass die fehlerhafte Berechnungen der Abstände, aufgrund des Blickwinkels auf den Kreis, wegfällt. Durch die kleine Größe des Mikrocontrollers und die einfachen Hardwareanforderungen, kann das System, je nach Anwendungsfall, auf große oder kleine Plattformen untergebracht werden.

Literaturverzeichnis

[1] DC mini Gearbox Motoren DG02S. https://cdn.sparkfun.com/datasheets/Robotics/DG02S.pdf. Stand(01.03.2019).

[2] Dual DC Motor Treiber L9110. https://www.roboter-bausatz.de/media/pdf/8b/81/46/L9110_Datasheet59880680d9819.pdf. Stand(01.03.2019).

[3] Kameramodul V2. https://www.raspberrypi.org/documentation/hardware/camera/.

[4] Multi Chassis Tank (Rescue Platform). https://cdn.sparkfun.com/datasheets/Robotics/multi-chassis%20rescue%202001.pdf. Stand(01.03.2019).

[5] OpenCV - Open Source Computer Vision Library. https://opencv.org/.

[6] Raspberry Pi Zero. https://www.raspberrypi.org/products/raspberry-pi-zero/.

[7] Simple Encoder Kit. https://cdn.sparkfun.com/datasheets/Robotics/multi-chassis%20encoder001.pdf. Stand(01.03.2019).

[8] WiringPi. http://wiringpi.com/.

[9] WiringPi Interrupts. http://wiringpi.com/reference/priority-interrupts-and-threads/.

[10] WiringPi Software PWM Library. http://wiringpi.com/reference/software-pwm-library/.

[11] Dr. David Offenberg. Visuelle Navigation. https://www.int.fraunhofer.de/content/dam/int/de/documents/EST/EST%200515%20Visuelle%20Navigation.pdf, 2015. Stand(01.03.2019).

[12] Gary Bradski und Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.

[13] Prof. Dr.-Ing. Holger Lutz und Prof. Dr.-Ing. Wolfgang Wendt. *Taschenbuch der Regelungstechnik: mit MATLAB und Simulink*. Wissenschaftlicher Verlag Harri Deutsch, Frankfurt am Main, 2012.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 21. März 2019 Alexander Hoffmann