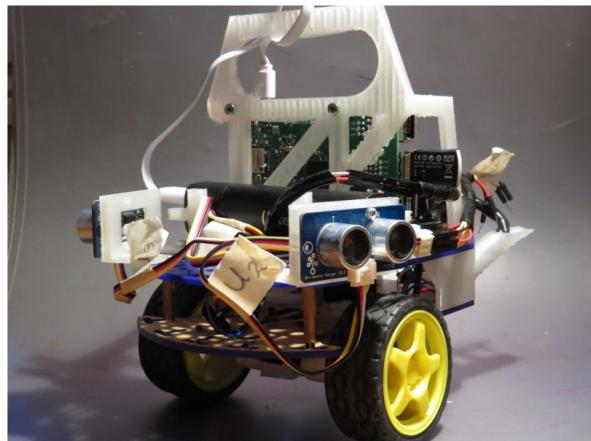


Vorlesung/Experimentelle Übung:
Programmierung mechatronischer Systeme

Hausarbeit

Raspberry Pi Roboter ("French Bulldog")



Tobias Giesecking Nils Melchert
Matrikelnummer 2883510 Matrikelnummer 2869520

Sommersemester 2016
Hannover, 15.07.2016

Dozentin: Prof. Dr.-Ing. Jessica Burgner
Betreuer: M.Sc. Ernar Amanov und M.Sc. Carolin Fellmann

Inhaltsverzeichnis

1	Einleitung	1
2	Der Roboter	1
2.1	Mobile Plattform	1
2.1.1	Kabelführung	2
2.1.2	Gehäuseteile	3
2.2	Sensorik	4
2.2.1	Liniensensor	4
2.2.2	Encoder	5
2.2.3	Ultraschallsensoren	5
2.2.4	Farbsensor	6
3	Umsetzung der Meilensteine	7
3.1	Linie folgen	7
3.2	Kreisfahrt	8
3.3	Kollisionsfreie Korridorfahrt	11
3.4	Farbfeld-Suche	15
4	Diskussion	16
5	Zusammenfassung	17
A	Abkürzungsverzeichnis	19
A	Programmcode	20

1 Einleitung

Die vorliegende Hausarbeit dokumentiert das Gruppenergebnis des Team 2 namens „French Bulldog“ aus dem Sommersemester 2016, welches während der Teilnahme an dem Kurs „Programmierung mechatronischer Systeme“ (PMS) des Lehrstuhls für Kontinuumrobotik (LKR) unter der Leitung von Frau Prof. Jessica Burgner-Kahrs erarbeitet wurde.

Das Lehrziel der Veranstaltung war es die Grundlagen des objektorientierten Programmierens in C++, den Umgang mit Nebenläufigkeiten und den für C++ charakteristischen Pointern zu vermitteln. Außerdem sollten die Studenten einen sauberen Programmierstil zur Beherrschung komplexer Programmierprojekte erlernen. Zur praktischen Untermauerung der Veranstaltung sollte ein einfacher Roboter, bestehend aus einem Einplatinencomputer und zwei Motoren sowie diverser Sensoren programmiert werden, der unterschiedliche Teilaufgaben zu bewältigen hatte. Den Studenten wurde die Möglichkeit gegeben das Zusammenspiel der vier Komponenten „mechanisches System“, „Sensorik“, „Informatik“ und „Aktorik“ eines mechatronischen Systems mit ihren Herausforderungen in der Praxis zu meistern. Parallel zur sauberen Programmierung war eine übersichtliche Kabelführung anzustreben, um vor allem in den vier über die Vorlesungszeit verteilten Prüfungssituationen Fehlfunktionen gezielt zurückverfolgen zu können. Weitere Schwerpunkte der Vorlesung waren die Nutzung der IDE Qt-Creator, das Auslegen von Reglern für die Drehzahlsteuerung der Motoren und das Implementieren eines I^2C -Buses zur Nutzung eines Farbsensors.

Die Hausarbeit beginnt mit der Vorstellung des Roboters hardwareseitig. Es werden die einzelnen Komponenten beschrieben, warum die Komponenten ausgewählt wurden und aus welchen Gründen spezifische Positionierungen am Roboter ausgewählt wurden. Im zweiten Teil werden die Programmieransätze zu den vier Meilensteinen erläutert. Unterstützend finden sich an dieser Stelle Klassendiagramme und Programmablaufpläne. Im anschließenden Teil werden aufgetretene Probleme und Herausforderungen diskutiert. Standardmäßig schließt das Dokument mit der Zusammenfassung des Erlernten und mit einem Ausblick.

2 Der Roboter

In diesem Kapitel wird der Hardware des Roboters erläutert. Zunächst wird in 2.1 ein Überblick über die verwendeten Komponenten gegeben. Es wird mit kurzer Beründung gezeigt, wo sie warum Montiert wurden. Dabei soll etwas detaillierter auf das Gehirn des Roboters, den Raspberry Pi, eingegangen werden. Dieser Teil war bei den meisten Team ähnlich, da alle das selbe Chassi verwendeten und z.B. die Motoren nur an einer definierten Stelle platziert werden konnten. Unterscheiden tun sich die Roboter in der Auswahl der Sensoren, darum wird in 2.2 auf diese detaillert eingegangen.

2.1 Mobile Plattform

Das Grundgerüst der mobilen Roboterplattform bildete das *Dagu Magician Chassis*, welches aus zwei Plastikplatten mit einer Vielzahl von Bohrungen und Nuten bestand. Es bildete die

formgebende Komponente des Roboters und besitzt die Maße 1117, 57, 5 in Zentimetern.

Tabelle 1: My caption

Komponente	Abkürzung	Anzahl
Raspberry Pi	1_Pi	1
Motor	2_Mo	2
Resolver	3_Rs	2
H-Brücke	4_HB	1
Ultraschallsensor	5_Us	2
Farbsensor	6_Fs	1
Breadboard	7_Bb	1
Blockbatterie 9V	8_B9	1
Powerbank 5V	10_Pb	1
Jumperkabel	11_Jk	k.A.
Widerstände	12_Ws	k.A.

Tabelle 2: Pinbelegungstabelle der GPIO-Pins des Raspberry Pis

wiringPi Bibl. Pin Nr.	Raspb. Pi Pin Nr.	Verbunden mit Pin	Komponente
4	16	Hallsensor 1	Resolver rechter
5	18	Hallsensor 2	Motor
8	3	SLC	
9	5	SDA	Farbsensor
12	19	SIG	Ultraschallsensor 1
13	21	SIG	Ultraschallsensor 2
15	8	Hallsensor 1	Resolver linker
16	10	Hallsensor 2	Motor
23	33	2	
24	35	1	Motor links
25	37	7	
27	36	9	
28	38	10	Motor rechts
29	40	15	

2.1.1 Kabelführung

Um für mehr Übersichtlichkeit zu sorgen (siehe Abbildung 1), haben wir unsere Kabelführung nach drei Regeln umgesetzt. Abbildung 1 zeigt die Draufsicht auf das Breadboard.

Regel Nr. 1 war, dass die Schaltungen auf dem Breadboard zweidimensional zu halten. Widerstände wurden kurz geschnitte, sodass sie auf dem Breadboard aufliegen und Verbindungen innerhalb des Breadboards wurden mit Flachsteckverbindern realisiert, die außerdem nur rechtwinklig verliefen. Die verschuf uns maximale Übersicht, da wir so sofort die Platinenabgänge von den Querverbindungen auf dem Breadboard unterscheiden konnten.

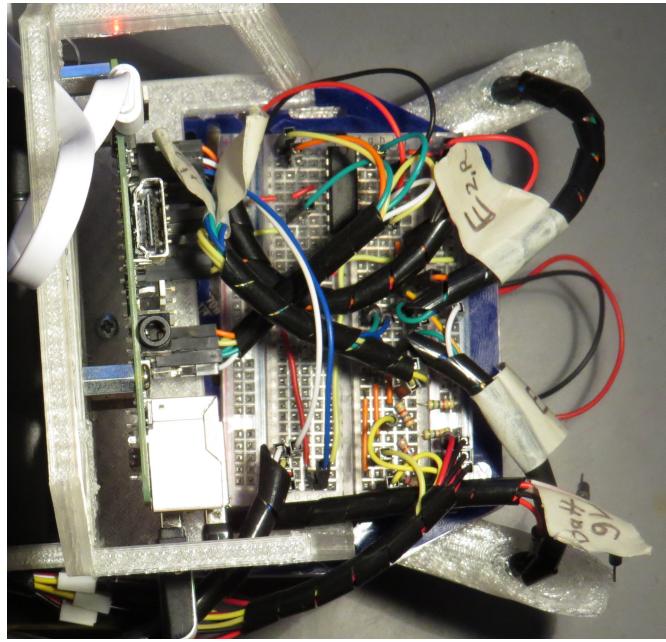


Abbildung 1: Draufsicht auf Kabelführung des Roboters

Regeln Nr. 2 war, dass von den Jumperkabeln möglichst viele Kabel mittels Spiralband (im Bild schwarz) zu Kabelsträngen zusammengefasst wurden.

Gegel Nr. 3 besagte, dass es möglichst wenig Platinenabgänge geben sollte, also stellen, an denen ein oder Mehrere Jumperkabel von der platine weg führen. Dadurch waren lediglich 6 Kabelschnittstellen zwischen Breadboard und Perifferie.

Um die Übersichtlichkeit noch weiter zu erhöhen, wurden mit Kreppband Fähnchen um die Kabelstränge geklebt. Im rechten oberen Teil der Abbildung 1 ist so ein Fähnchen besonders gut zusehen, es trägt die Aufschrift $E_{2,R}$, das bedeutet, dass dieser Kabelstrang alle Leitungen für den rechten Motor (E stand für Engine) enthält.

2.1.2 Gehäuseteile

Da bei der Gestaltung des Roboters mit den vom LKR gestellten Standardkomponenten wenig Spielraum, um sich von den anderen Teams abzugeben, wurde die den Teams angeboten, dass einzelne Komponenten mit dem 3D-Drucker des Institutes realisiert werden könnten. Wir haben von dem Angebot gebrauch gemacht. In Abbildung 2 die zwei erstellten Teile zu sehen.

Für die CAD-Zeichnungen wurden die Studentenversion von Autodesk Inventor Professional 2014 verwendet. Bild a) zeigt das Heckteil, das lediglich für die Optik entworfen wurde, wohingegen das Teil aus Bild b9, der Pi-Ständer, mehrere praktische Funktion hatte, wie in Bild c) deutlich wird. Durch das aufstellen des Pis wurde Platz auf dem Oberdeck des Chassis gespart, sodass das Powerpack dort mit einer Halterung verstaut werden konnte. Das Bauteil Pi-Ständer hält außerdem das Powerpack in Position und das Langloch im oberen Bereich dient als Griff zum Anfassen und sorgenfreiem Aufnehmen des Roboters. Da beim Anfassen des Roboters anfänglich

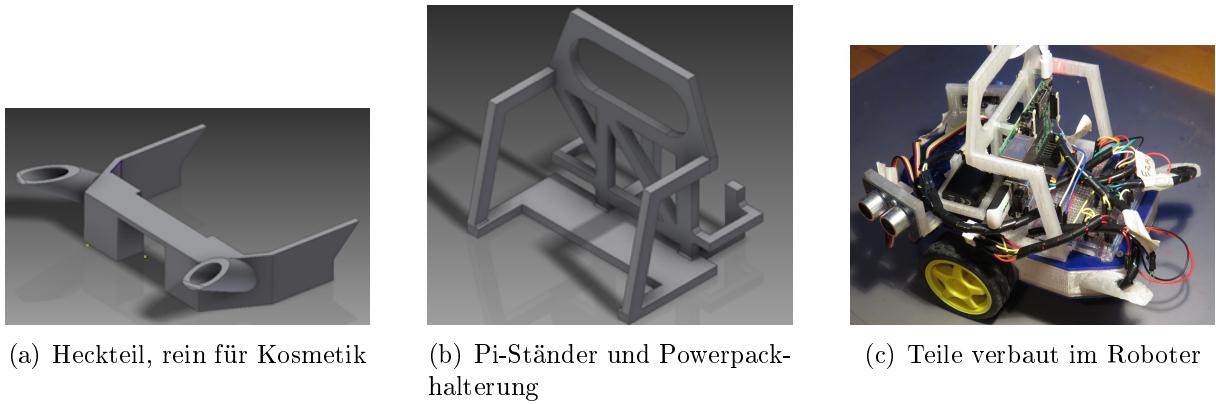


Abbildung 2: Gehäuseteile des Roboters aus dem 3D-Drucker

immer darauf geachtet werden musste, wo man ihn am besten anfasst, um keine Kabel zu lösen oder ähnliches, konnte der Roboter nun bedenkenlos hochgehoben werden.

2.2 Sensorik

Zur Bewältigung einzelner Meilensteine wurden im Laufe der Veranstaltung Programmierung mechatronischer Systeme verschiedene Sensoren implementiert. Diese wurden über ein Breadboard mit Steckverbindungen an den Raspberry Pi angeschlossen.

2.2.1 Liniensensor

Für den ersten Meilenstein (siehe Kapitel 3.1) standen Sensoren des Typs „*Grove Line Finder v1.0*“ der Firma „*Seedstudio Groove*“ bereit. Bei diesen handelt es sich um Infrarot-Sensoren, die IR-Licht über eine Infrarot-Leucht-Diode auf den Boden strahlen. Das reflektierte Licht wird über einen Fototransistor ausgewertet. Über die Intensität der Reflketion sind sie in der Lage zwischen weißem und schwarzem Untergrund zu unterscheiden. Über ein Potentiometer auf dem Sensor ist es möglich die Sensitivität des Sensors und somit einen Schwellwert des Schaltens der Transistoren anzupassen. Bei schwarzem Untergrund schaltet der Transistor und der Sensor gibt einen „*High-Pegel*“ aus. Die Spannung dieses Ausgangs entspricht der Versorgungsspannung des Sensors, welche 5 Volt beträgt und direkt vom Raspberry-Pi abgegriffen werden kann. Da die GPIO-Pins des Pi eine maximale Eingangsspannung von 3,3 Volt zulassen, wird das 5 Volt Signal des Sensors über eine Schutzschaltung auf den Raspberry-Pi gegeben. Das Ausgangssignal des Sensors wird dabei über einen Widerstand von $1k\Omega$ auf die Basis eines NPN-Transistors des Typs „*2N3904*“ gegeben (QUELLE). Am Kollektor liegen 3,3 Volt des Raspberry-Pi an und werden beim Schalten des Transistors direkt auf einen GPIO des Raspberry-Pi zurückgeführt (siehe Abbildung 3). Über 3D-gedruckte Komponenten werden die Sensoren an der Plattform montiert, um einen möglichst geringen Abstand zum Boden zu gewährleisten. Dies ist notwendig, um Störeinflüsse des Lichts so weit wie möglich zu umgehen. Um dies noch weiter zu unterstützen, wurde im Rahmen dieser Arbeit beide Liniensensoren mit Abschirmvorrichtungen in Form von dunklen Papiertrichtern versehen, um das Infrarotlicht stärker auf einen Bereich des Untergrunds zu fokussieren und weiter vor Störeinflüssen zu schützen.

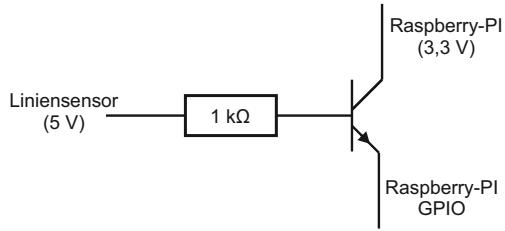


Abbildung 3: Schaltung der Transistor-Schutzschaltung des Liniensensors

2.2.2 Encoder

Zur Erfassung der Motorgeschwindigkeiten wurden Encoder oder auch Drehgeber verwendet. Diese bestehen aus zwei Hallsensoren und einer rotierenden Magnetscheibe mit 6 Polpaaren (siehe Abbildung 4). Nach dem Prinzip eines stromdurchflossenen Leiters in einem Magnetfeld ergibt sich beim Durchlauf des Hallsensors eine Spannung die proportional zum Produkt aus magnetischer Feldstärke und Stromstärke des Leiters ist. Die sich bei einer rotierenden Magnetscheibe einstellenden Spannungssprünge, werden als Encoder-Ticks bezeichnet und sind Grundlage zur Drehzahlberechnung. Bei einer vollständigen Umdrehung der Magnetscheibe ergeben sich bei zwei Hallsensoren somit 12 verwendbare Impulse. Der Spannungspegel entspricht dabei der Versorgungsspannung der Hallsensoren von 3,3 Volt, welche direkt vom Raspberry-Pi gespeist wird. Da der Encoder nicht direkt an die Motorwelle angeschlossen ist, sondern sich auf der Abtriebsseite eines Getriebes mit einer Getriebeübersetzung von 120:1 befindet, vervielfältigt sich diese Anzahl zu 1440 Impulsen bei einer Radumdrehung. Die Auflösung ist somit ausreichend hoch für eine genaue und schnell abgetastete Drehzahlerfassung, die für regelungstechnische Zwecke eingesetzt werden kann. Während der Fahrt werden die Impulse des Encoders kontinuierlich über einen separaten Thread ausgelesen, um ein Fehler während schnellen Impulsmessung zu vermeiden.

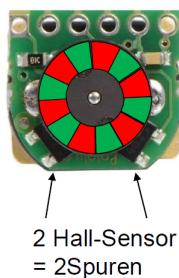


Abbildung 4: Aufbau des verwendeten Drehgebers basierend auf dem Hall-Effekt QUELLE!

2.2.3 Ultraschallsensoren

Zur Distanzmessung wurden im Rahmen des dritten Meilensteins (Vergleiche Kapitel 3.3) und der „Roboter-Challange“ Ultraschallsensoren vom Typ „*Grove Ultrasonic Finder*“ der Firma „*Seedstudio Groove*“ bereitgestellt. Sie dienen der berührungslosen Abstandsmessung und können bei einer Auflösung von 1cm einen Bereich von 3 – 400cm messen (QUELLE!). Bei der Abstands-

messung wird ein Ultraschallsignal mit einer Frequenz von 40kHz ausgesandt. Dabei wird die Zeit Δt gemessen, die das Echo an der Wand benötigt, um wieder zum Sensor zurückzukehren. Mit der gemessenen Zeit und der konstanten Schallgeschwindigkeit unter thermodynamischen Grundbedingungen von $v_{sonic} = 343.2$ ergibt sich eine Abstand d nach Gleichung 1. Dabei wird durch zwei geteilt, da die halbe Distanz berechnet werden soll, die der Schall zurücklegt.

$$d = v_{sonic} \cdot \frac{\Delta t}{2} \quad (1)$$

Um die Abstandsmessung zu starten muss ein „*High-Pegel*“ über $10\mu\text{s}$ auf einen Pin des Sensors gegeben werden. Nach 8 internen 40kHz -Impulsen des Ultraschallsensors wird dann auf eine Antwort des Sensors in Form eines „*High-Pegels*“ gewartet. Die gemessene Zeit des Pegels entspricht dann der in Gleichung 1 verwendeten Zeit Δt über die sich der Abstand berechnen lässt. Das Senden und Empfangen der Daten läuft dabei über den selben Pin des Sensors. Im Programmcode ist es dementsprechend beim Ansprechen des Sensors notwendig den GPIO des Raspberry-Pi öfters zwischen Eingang und Ausgang umzukonfigurieren. Da der Sensor mit einer Spannung von 5 Volt versorgt wird, und diese auch als Pegel ausgibt, ist aus dem gleichen Grund wie beim Liniensensor eine Schutzschaltung zu implementieren. Da diese jedoch sowohl Eingangs- als auch Ausgangssignale verarbeiten können muss, wird eine Spannungsteiler- anstelle einer Transistorschutzschaltung verwendet. Für weitere Informationen sei an dieser Stelle auf QUELLE! verwiesen.

2.2.4 Farbsensor

3 Umsetzung der Meilensteine

3.1 Linie folgen

Im Rahmen des ersten Meilensteins, gilt es einer weißen Linie auf schwarzem Untergrund zu folgen. Der Linienverlauf beinhaltet dabei sowohl Geradeausverläufe, als auch Links- und Rechtsabbiegungen in Form von S-Kurven und 90-Grad-Abbiegungen. Zwei der in Kapitel 2.2 beschriebenen digitalen Infrarot-Spursensoren werden zur Ermittlung der Position des Roboters entlang der besagten Linie montiert und angeschlossen. Die Schnittstelle zum Auslesen der digitalen Signale der Sensoren liefert eine Klasse *Liniensor*, welche eine Methode zur Erfassung digitaler Pegel bereitstellt. Sie liest einen Anschlusspin des RaspberryPi aus und detektiert, ob sich der Roboter über weißem oder schwarzem Untergrund befindet. Für jeden Sensor wird ein Objekt des Typs *Liniensor* in der Klasse *Mobileplatform* erstellt.

Zur Linienfahrt ergeben sich drei Ansätze, die auf der Orientierung besagter Spursensoren basieren: Das Verfolgen einer Linienkante, Platzierung beider Sensoren auf der Linie und Positionierung rechts und links neben der Linie. Im Rahmen dieser Arbeit wurde der letzte Ansatz gewählt, da er sich bei der Implementierung als robuster erwiesen hat und sich zugleich ein ruhigeres Fahrverhalten einstellte.

Basierend auf der zuvor genannten Lösungsstrategie, ergibt sich die Implementierung des Algorithmus. Ein Programmablaufplan dessen ist in Abbildung ... zu sehen. Er hat die Aufgabe die Sensoren während der Fahrt stets links und rechts neben der weißen Linie zu halten. Der Aufruf des Algorithmus erfolgt in der Klasse *Mobileplatform* über eine ungetaktete While-Schleife, sodass kontinuierlich der Status beider Sensoren abgefragt wird. Detektieren die Liniensensoren beide schwarzen Untergrund, befinden sich die Sensoren in ihrer Soll-Position und der Roboter fährt geradeaus. Ist gibt der linke Sensor einen *Low-Pegel* zurück, befindet sich dieser über weißem Untergrund. Der Roboter ist zu weit nach rechts abgewichen. Es folgt eine Korrektor der Ausrichtung anhand einer Linksdrehung. Analog Erfolgt bei einem *Low-Pegel* des rechten Sensors eine Rechtsdrehung.

Die Bewältigung der Aufgabe lief mit dem gewählten Lösungsansatz problemlos. Dies ist unter anderem aber auch auf die stets nahezu konstante Breite der weißen Linie zurückzuführen. Sobald der Abstand der Sensoren geringer ist, als die besagte Breite, versagt die gewählte Lösungsstrategie. Unter dieser Bedingung ist erfahrungsgemäß der Ansatz zur Orientierung an Linienkanten zu wählen.

Zudem hatten hohe Geschwindigkeiten aufgrund der Massenträgheit zur Folge, dass der Roboter besonders bei 90-Grad-Abbiegungen nicht rechtzeitig abbremst und die Linie verlässt. Durch die Detektion schwarzen Untergrunds beider Sensoren, betrachtet der Algorithmus den Roboter in seiner Soll-Position und lässt ihn geradeaus fahren. Um diesem Versagen des Algorithmus entgegenzuwirken, wurde die Fahrtgeschwindigkeit angepasst.

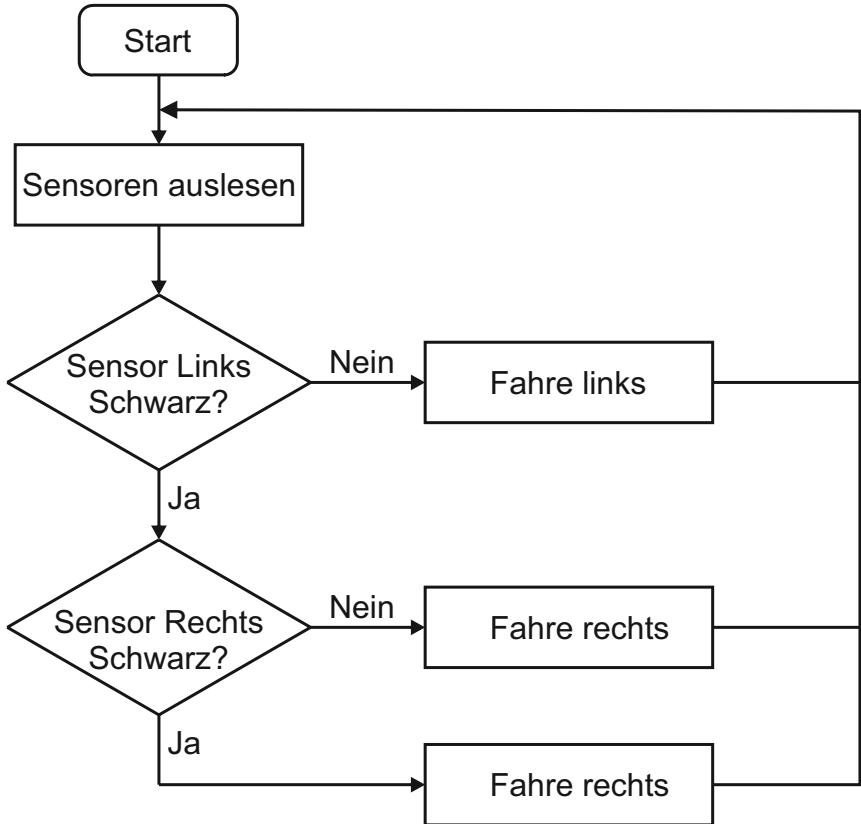


Abbildung 5: Programmablaufplan der Linienfahrt

3.2 Kreisfahrt

Die Kreisfahrt war Bestandteil des zweiten Meilensteins. Unter Vorgabe eines Radius im Bereich von 15-35 Zentimeter, galt es den Roboter einen Kreis fahren zu lassen. Dieser durfte einen Toleranzbereich von $\pm 2\text{cm}$ nicht verlassen. Zur Überprüfung wurde ein Stift am Roboter befestigt, welcher auf einer Kreisschablone die gefahrene Strecke des Roboters aufzeichnet. Zwei der in Kapitel 2.2 beschriebenen Encoder wurden angeschlossen und eine Klasse *Encoder* in *Dcmotor* integriert (Siehe Abbildung ...). Des weiteren wurde eine Regelung der Geschwindigkeit in der Klasse *Dcmotor* implementiert. Der Algorithmus zur Kreisfahrt wurde schließlich in eine Funktion geschrieben, die die Berechnung der Geschwindigkeiten beider Gleichstrommotoren realisiert.

Über zwei Hall-Sensoren und ein Getriebe mit einer Übersetzung von 120:1 [1] ergeben sich pro Radumdrehung 1440 Signalflanken des Drehgebers. Über einen Zeitraum von $\Delta T = 0.1\text{s}$ werden Impulse k gezählt und nach Gleichung 2 unter Berücksichtigung des Raddurchmessers von $d = 0,063\text{m}$ zu einer Geschwindigkeit in SI-Basiseinheiten umgerechnet.

$$v = \frac{kd\pi}{1440\Delta T} \quad (2)$$

Da der Signalpegel des Drehgebers im Zeitraum von 0,1 Sekunden ausreichend hoch ist, um eine Geschwindigkeit zu berechnen, kann Messrauschen durch ungenaue Fertigung der Enco-

der vernachlässigt werden. Demzufolge führt die Mittelwertbildung mehrerer Drehzahlsignale zu keiner Verbesserung in der Regelung.

Zur Regelung wird ein PID-Regler mit einer Taktrate von $10Hz$ implementiert. Somit wird bei jeder Erfassung der aktuellen Geschwindigkeit eine Änderung dieser durch den Regler vorgenommen. Abbildung 6 zeigt ein Blockschaltbild des verwendeten PID-Reglers.

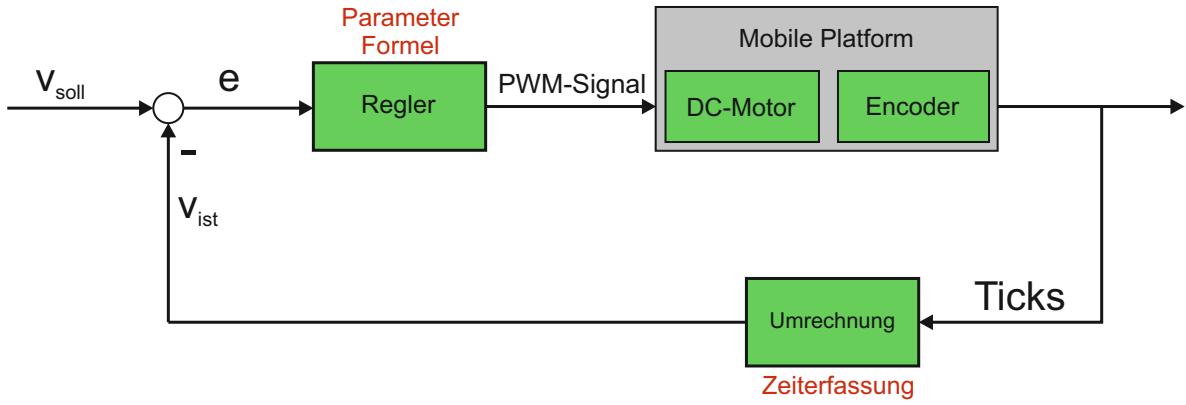


Abbildung 6: Blockschaltbild eines PID-Reglers

Die Eingangsgröße des Reglers ergibt sich aus der Differenz und somit dem Fehler e aus Ist-Geschwindigkeit v_{ist} und Soll-Geschwindigkeit v_{soll} . Diese Differenz wird nach Formel 3 durch einen proportionalen, integralen und differentiellen Anteil so verstärkt, dass sich die Regelabweichung verkleinert und im optimalen Fall verschwindet. Der proportionale Anteil verstärkt dabei den Fehler mit einem konstanten Faktor K_P . Da es sich um einen zeit diskreten Regler handelt, bildet sich der integrale Anteil nicht durch eine Integration der Regelabweichung über die Zeit, sondern durch die Multiplikation der Summe aller Fehler mit der Konstanten K_I . Analog dazu wird der differenzielle Anteil nicht nach der Zeit abgeleitet sondern ergibt sich aus dem Produkt der Differenz des Fehlers mit dem Fehler des vergangenen Abtastschrittes und dem Faktor K_D .

$$y[n] = K_P \cdot e[n] + K_I \cdot \sum_{m=0}^n e[m] \cdot \Delta T + K_D \cdot \frac{e[n] - e[n-1]}{\Delta T} \quad (3)$$

Die Verstärkungen der Regler werden über *Live – Tuning*, das heißt im laufenden Betrieb des Roboters ermittelt. Eine Auslegung der Parameter über ein lineares Systemmodell zweiter Ordnung ist nur schwer möglich, da das Übertragungsverhalten von Geschwindigkeit und PWM-Signal nichtlinear ist. Dieses Verhalten ist in der Kennlinie von Abbildung 7 zu entnehmen. Des Weiteren wurden für beide Motoren verschiedene Parameter ermittelt, da sie bei gleichem Eingangssignal unterschiedliche Drehzahlen aufweisen.

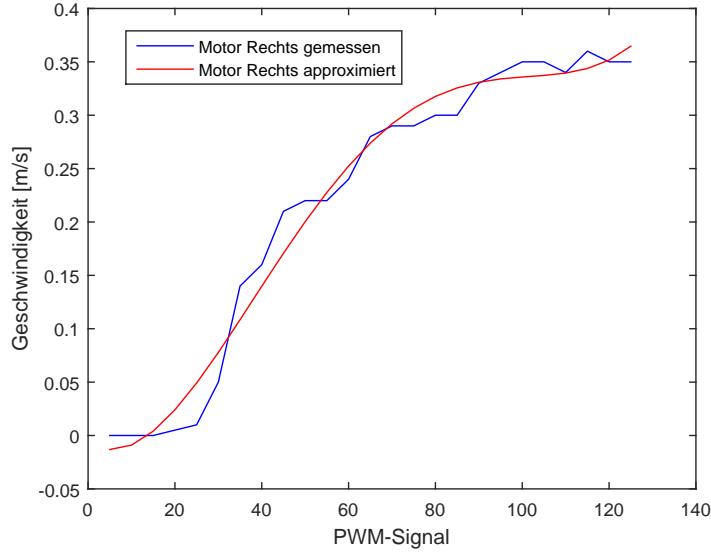


Abbildung 7: Nichtlineare Kennlinie der Drehzahl des rechten Motors

Zur Realisierung der Kreisfahrt im Uhrzeigersinn wurde die Klasse *Mobileplatform* um eine Methode erweitert, die eine Funktion zur Berechnung der einzelnen Radgeschwindigkeiten bereitstellt. Die Basisgeschwindigkeit wird dabei zwischen beiden Rädern angenommen (Abbildung 8), deren Abstand ($b = 11\text{cm}$) beträgt. Da die Distanz von Mittelpunkt zu Stifthalter ($s = 12\text{cm}$), welcher am hinteren Ende des Roboters befestigt wurde, nicht vernachlässigbar ist, fließt dessen Position in die Berechnung der Geschwindigkeiten mit ein. Aus Abbildung 8, die eine Rotation des Roboters um einen Punkt P darstellt, folgen die Gleichungen 11, 5 und 6 zur Berechnung der Soll-Geschwindigkeiten.

$$r_{soll} = \sqrt{r^2 - s^2} \quad (4)$$

$$v_{Links} = v \cdot \frac{r + \frac{b}{2}}{r} \quad (5)$$

$$v_{Rechts} = v \cdot \frac{r - \frac{b}{2}}{r} \quad (6)$$

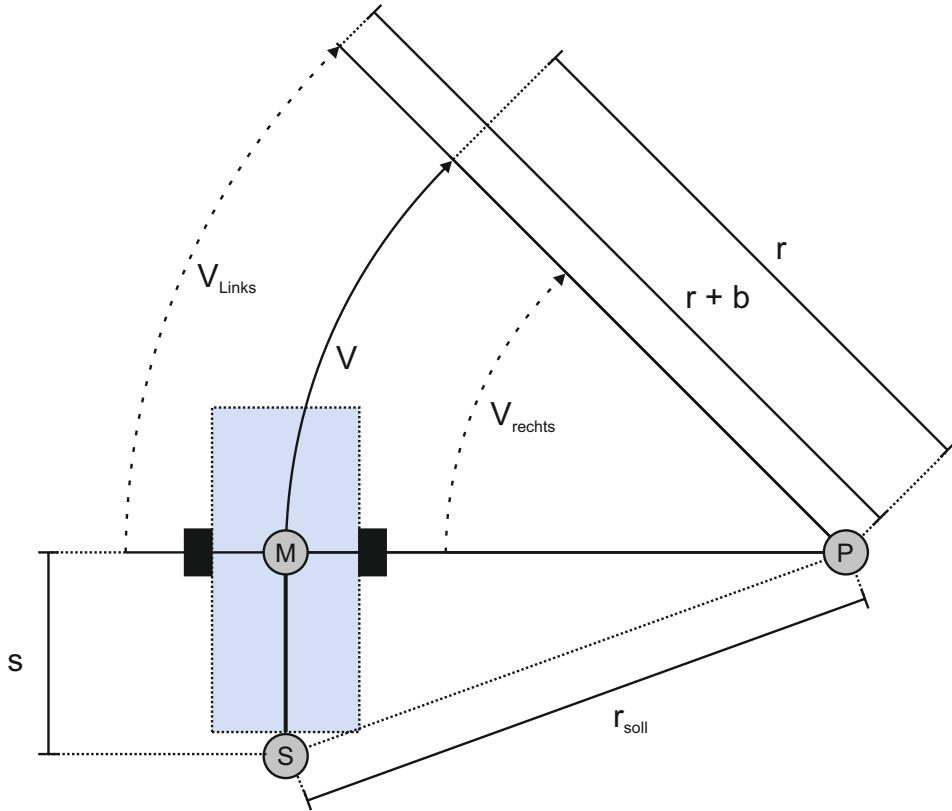


Abbildung 8: Nichtlineare Kennlinie der Drehzahl des rechten Motors

Bei der Abnahme des Meilensteins wurden Radien von 23 und 30 cm abgefahren. Dies lief bei $r_{soll} = 23\text{cm}$ problemlos. Nur beim Radius von 30 cm gab es kleine Überschreitungen der Toleranzgrenze von $\pm 2\text{cm}$. Da die Signale der Encoder ausreichend hochauflösend sind, ist dieser Fehler hauptsächlich auf die Einschwingvorgänge des Reglers und Messunsicherheiten der Geometrien zurückzuführen.

In den weiteren Aufgabenteilen wird die Geschwindigkeitsregelung beibehalten, da sie die Laufruhe des Roboters unterstützt und auch dabei hilft Hindernisse zu überwinden.

3.3 Kollisionsfreie Korridorfahrt

Herausforderung des 3. Meilensteins war eine autonome, kollisionsfreie Fahrt durch ein Labyrinth mit einem konstanten Wandabstand von 35 cm. Die Rechts- und Linksabbiegungen der Strecke verlaufen dabei alle mit einem Winkel von 90 Grad. Als kollisionfrei wurde im Kontext der Vorlesung als grundsätzlich kontaktfrei von Objekten in der Umgebung definiert, also war auch keine streifen von Objekten zulässig. Die Berührung der zur Auswahl gestellten Tastensensoren mit den Wänden wurde jedoch nicht als Kollision gewertet. Da mit den Ultraschallsensoren eine quasi kontinuierliche Abstandsmessung möglich war und mit den Tastern nur ein definierter Abstand detektiert werden konnte, fiel die Wahl auf die Ultraschallsensoren zur Navigation der Plattform. Zur Messung des absoluten Abstands von Roboter zu Wand wurden somit zwei der besagten Sensoren (siehe Kapitel 2.2) angeschlossen und über einer Klasse *Ultrasonic* angesprochen. Die Sensoren werden nacheinander ausgelesen um Interferenzen und somit Fehler bei der

Distanzmessung zu vermeiden. Des weiteren wurden die Ultraschallsensoren nicht direkt seitlich an der Plattform angebracht, sondern mit einem leichten Winkelversatz von $\alpha = 30^\circ$ nach vorne montiert. Abbildung 9 zeigt, dass dies ein Vorausschauen des Roboters und somit die rechtzeitige Erkennung von Abbiegungen bewirkt.

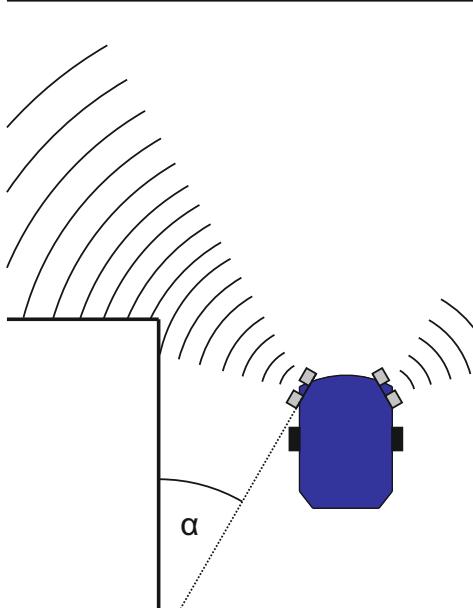


Abbildung 9: Anbringung der Ultraschallsensoren im Hinblick auf Kurvenverhalten

Das UML-Diagramm in Abbildung ... veranschaulicht, den Zusammenhang der Klasse *Mobileplatform* und *Ultrasonic*. Es ist ersichtlich, dass *Mobileplatform* vollständigen zugriff auf Funktionalitäten der Ultraschallsensoren erhält. Somit eignet sich diese Klasse gut für die Implementierung des Algorithmus zur Korridorfahrt, der die Erkennung von Hindernissen gewährleisten soll. Zur Regelung des Roboters während der Fahrt wurden im Rahmen dieser Arbeit drei verschiedene Ansätze mir unterschiedlicher Anordnung der Ultraschallsensoren erprobt. Diese verschiedenen Strategien werden im Folgenden erläutert und durch Abbildung 10 unterstützt.

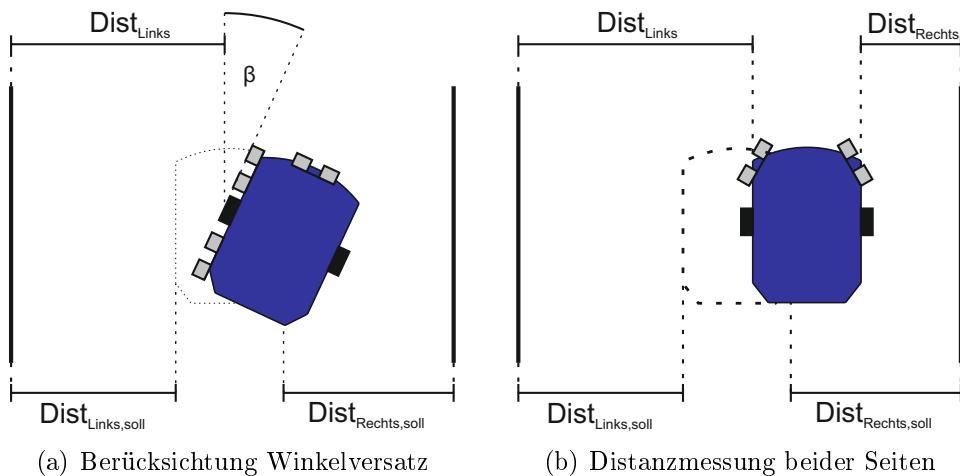


Abbildung 10: Position des Roboters im Korridor

In Abbildung 10(a) sind zwei Sensoren seitlich am Roboter montiert und einer in Fahrtricht-

tung. Der Ansatz des Algorithmus beruht auf einer Orientierung an der Linken Wand des Korridors. Dabei werden wird sowohl der Winkelversatz der seitlich montierten Sensoren, als auch der Abstand zur linken Wand während Fahrt des Roboters berücksichtigt und gegebenenfalls korrigiert. Der vordere Sensor detektiert, ob sich ein Hindernis in Fahrtrichtung befindet und ein Abbiegevorgang ausgeführt werden muss. Aufgrund von Schwierigkeiten der Implementierung eines Kaskadereglers mit Winkel- Abstandskorrektur wurde dieser Ansatz aufgrund von Zeitmangel verworfen. Zudem hätte ein Abbiegevorgang implementiert werden müssen, was als „Hard Coding“ gilt und im Vergleich zum Regleransatz unelegant und nicht wiederverwendbar ist.

Der zweite und dritte erprobte Ansatz wurde mit der zuvor beschriebenen winkelversetzten Sensorkonfiguration ausgeführt. Beide Ansätze beruhen darauf eine Soll-Distanz zu beiden Wänden einzuhalten und den Roboter somit mittig im Korridor zu halten (Abbildung 10).

Dies wurde im Rahmen des zweiten Ansatzes dadurch realisiert, das Verhältnis aus $Dist_{Links}$ und $Dist_{Rechts}$ mit der Initialgeschwindigkeit des jeweiligen Motors zu multiplizieren. Somit ergeben sich nach Gleichung 7 und 8 für beide Räder neue Soll-Geschwindigkeiten. Diese Geschwindigkeiten sind dabei auf einen Bereich zwischen 0,05 und 0.4 Meter pro Sekunde begrenzt.

$$v_{Rechts} = v \cdot \frac{Dist_{Links}}{Dist_{Rechts}} \quad (7)$$

$$v_{Links} = v \cdot \frac{Dist_{Rechts}}{Dist_{Links}} \quad (8)$$

Da sich bei Abbiegungen ein entsprechend großes Verhältnis einstellt, ist dieser Ansatz auch für Abbiegevorgänge geeignet.

Als dritte Option, wurde eine Abstandsregelung mittels zweier P-Regler implementiert. Diese setzte sich im aufgrund ihrer Robustheit und sich einstellenden Fahrtruhe gegen die zuvor genannten Verfahren durch. Da die Ultraschallsensoren, aufgrund ihrer Positionierung nicht den direkten Abstand zur Wand messen, wurde zur Bestimmung der Soll-Größen eine Referenzmessung durchgeführt, bei der der Roboter mittig im Korridor platziert wurde. Somit ergab sich bei mittiger Fahrt eine minimale Regelabweichung e . Ein Blockdiagramm des verwendeten P-Reglers ist in Abbildung 11 zu sehen.

Die Ermittlung der proportionalen Regelverstärkung erfolgte empirisch. Dabei musste ein Kompromiss aus steiler Kurveneinlenkung und ruhiger Geradeausfahrt gefunden werden. Abbildung ... zeigt, dass sich bei geringen Verstärkungen der Regeldifferenz zwar ein ruhiges Fahrverhalten auf gerader Strecke ergibt, das Einlenken in Kurven jedoch durch zu träge ist und zu Kollisionen führen kann. Analog dazu ergibt sich bei großer Fehlerverstärkung eine ungleichmäßige, zur Seite schwingende Geradeausfahrt, die bis zum Kontakt mit den Seitenwänden führen dann. In Kurven agiert der Roboter hingegen hoch dynamisch (Abbildung 12).

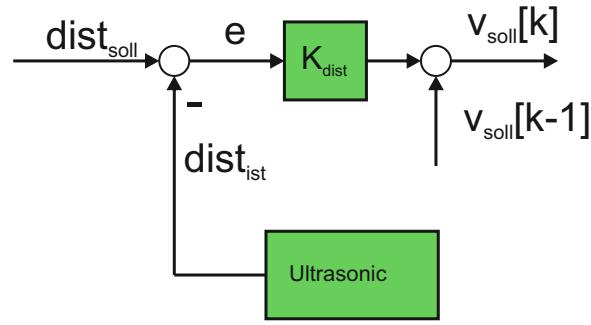


Abbildung 11: Anbringung der Ultraschallsensoren im Hinblick auf Kurvenverhalten

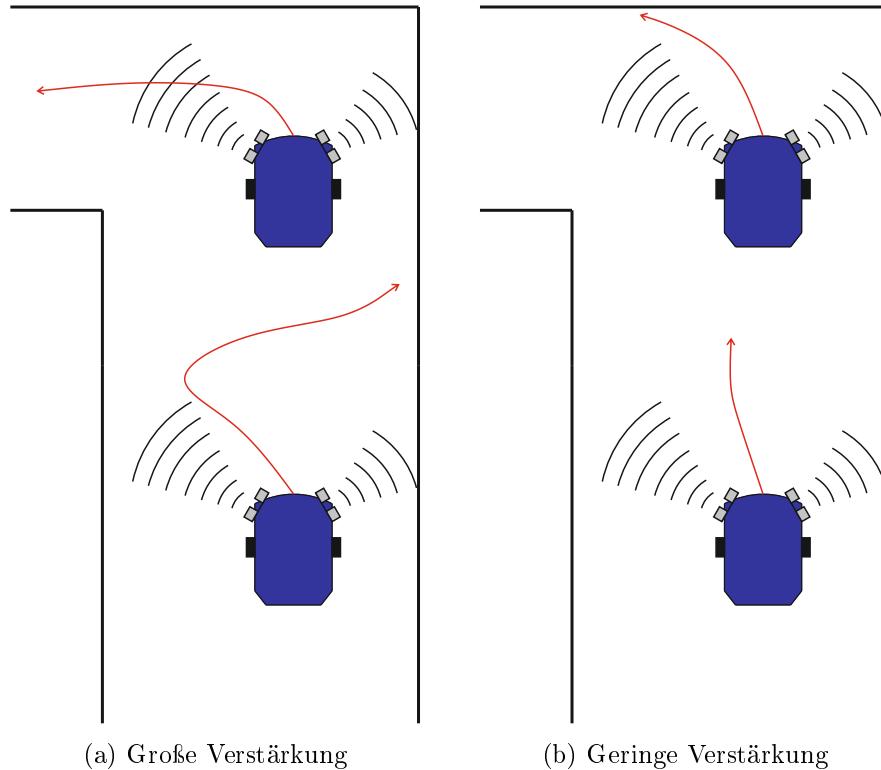


Abbildung 12: Verhalten des proportionalen Verstärkungsfaktors

Ist die Verstärkung zu groß, lenkt der Roboter in einer Linksabbiegung gegebenenfalls zu schnell ein und es ergibt sich eine weitere Situation, die den Algorithmus versagen lässt (Abbildung 13). Durch eine große gemessene Distanz des linken Sensors, detektiert der Roboter eine weitere Linksabbiegung und dreht im Labyrinth um oder bleibt im schlechtesten Fall an der Abbiegung stecken.

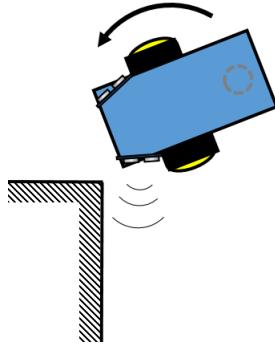


Abbildung 13: Ultraschallsensor detektiert Innenwand nicht mehr

3.4 Farbfeld-Suche

Beim letzten Meilensteins, der auch als "Roboter-Challenge" bezeichnet wird, handelt es sich um eine Erweiterung der kollisionsfreien Korridorfahrt. Die abzufahrende Strecke wird vergrößert und um weitere Abbiegungen erweitert, welche nicht mehr nur um 90 Grad, sondern auch in einem beliebigen Winkel abknicken können. Zudem werden im Labyrinth Hindernisse in Form einer Wippe und einer Bodenwelle verteilt und Farbfelder in den Farben Rot, Grün und Blau ausgelegt. Die Hindernisse sind ohne weitere Veränderung des Algorithmus zu bewältigen, was auf den Geschwindigkeitsregler zurückzuführen ist, der einer Verringerung der Geschwindigkeit auf einer Steigung entgegenwirkt. Die Farbfelder werden über einen RGB-Farbsensor des Typs "FLORA Color Sensor" detektiert. Über eine Klasse *Colorsensor*, die in der *Mobileplatform* implementiert ist, können die einzelnen RGB-Farben über 16-Bit-Integer ausgelesen werden. Der Parkour ist dann erfolgreich abgeschlossen, wenn der Roboter ohne Kollisionen über die Hindernisse fährt und schließlich auf dem roten Farbfeld stehen bleibt. Da für die Parkourfahrt nur das rote Farbfeld von Bedeutung war, wurde die Farbe Rot im Verhältnis zu Grün und Blau identifiziert. Der Schwellenwerte wurde empirisch ermittelt.

Um Störeinflüsse auf die Farbmessung zu filtern, die auf alle drei Farbdioden näherungsweise gleichermaßen wirken, wird der Mittelwert aller Messergebnisse aus einem Messzyklus gebildet (Gl. 6) und anschließend von dem Messwert, den der Sensor lieferte subtrahiert (Gl. 7).

$$average = \frac{redVal_{Sensor} + greenVal_{Sensor} + blueVal_{Sensor}}{3} \quad (9)$$

$$redVal_{filtered} = redVal_{Sensor} - average \quad (10)$$

Um das rote Farbfeld robust zu erkennen und nicht mit den weißen Flächen auf der Wippe und dem Bump zu verwechseln, welche ebenfalls den gesamten Rotanteil des Lichts reflektieren, hat sich bei Versuchen herausgestellt, dass es nicht allein reichte einen bestimmten Grenzwert des gefilterten roten Farbwertes $redVal_{filtered}$ zu betrachten. Es war notwendig den Farbwert

des Rotanteils mit dem des Blauanteils in Relation zu setzen. Letzenendes setzte sich folgende Ungleichung mit dem dazugehörigem Schwellwert bei einer robusten Rotfelderkennung durch:

$$220 > redVal_{filtered} - blueVal_{filtered} \quad (11)$$

4 Diskussion

In diesem Abschnitt werden die vier lehrreichsten Hindernisse, die zu überwinden waren, in chronologischer Reihenfolge erörtert. Darauf folgend werden einige Ansätze angesprochen, die bei einer Weiterentwicklung verfolgt worden wären.

Der Wunsch unseres Teams besonders gut bei der Vorlesung abzuschneiden und unsere Experimentierfreudigkeit haben dazu geführt, dass stellenweise zu komplexe Lösungsansätze für die Erfüllung der Aufgaben gewählt oder unbewusst Raum für weitere Fehlerquellen geschaffen wurde. Unsere erstes Hindernis bestand aus der Inbetriebnahme der Liniensensoren. Die selbst hergestellte Teststrecke für die Linienfahrt verwendete als Untergrund eine handelsübliche schwarze Pappe, deren einziges Kaufkriterium war, sie müsse eben schwarz sein. Da die Pappe verhältnismäßig stark spiegelte war der Kontrast im infraroten Lichtspektrum nicht groß genug um eine robuste schwarz-weiß-Unterscheidung gewährleisten zu können.

Unser nächstes Hindernis war beispielhaft für einen zu komplexen Lösungsansatz: Bei der Reglerauslegung der Motoren für die Kreisfahrt im zweiten Meilenstein haben wir versucht die *Simulink Control Design Toolbox* zur Regelparametrierung zu nutzen. Es wurde eine Methode zur Aufzeichnung einer Identifikationsfahrt der Servomotoren erstellt und anhand dieser Daten das System in Simulink simuliert sowie die Regelparameter bestimmt. Bedauerlicherweise war die Performance nicht ausreichend, da der Roboter nicht gerade aus fuhr und das Fahrverhalten unruhig war. Letzteres lies sich an einem Schwingen der Motorgeäusche erkennen. Nach längerem experimentieren und aus Zeitmangel wurden die Parameter letztenendes heuristisch bestimmt. Hierfür wurden Textboxen im GUI implementiert, um Lifetuning vornehmen zu können. Dies ersparte das vielfache Ändern des Programmcodes, das damit einhergehende neu Kompilieren und ausführen.

Nummer drei und vier der Hindernisse ereigneten sich beim dritten Meilenstein, der Kollisionsfreien Fahrt durch einen Korridor. Aus Beobachtung der Challenge im Vorjahr und durch Erfahrungswerte von der Linienfahrt im ersten Meilenstein entstand bereits früh der Gedanke einen neuen Ansatz für die Korridorfahrt zu erproben: Da in der Roboter-Challenge die Zeit für das Durchfahren des Korridors wichtig sein würde, sollte sich unser Roboter besonders flüssig durch den Korridor bewegen. Uns fiel auf, dass viele Roboter im Zickzack von Wand zu Wand fuhren. Die Idee war, sich mit zwei seitlich angebrachten Ultraschallsensoren parallel in einem definiertem Abstand zur Wand zu bewegen. Über Triangulation sollte eine Winkel- und Abstandsregelung implementiert werden. Dieser Ansatz wurde später verworfen, weil eine stabile Regelung aufgrund der hohen Ungenauigkeiten in der Abstandsmessung nicht gelang. Nicht bedacht wurde, dass zusätzlich zu der Messtolleranz von einem 1 cm (in selbstversuchen bestätigt

bei Abständen von 5 - 20 cm) nicht genau festgelegt war von welchem Punkt in der Ferne der Abstand gemessen wurde. So kann ein Objekt auch detektiert werden, wenn die Mittellinie des Ultraschallsensores einige Zentimeter an dem Objekt vorfährt. Unser vierter Hindernis waren Probleme bei der Abstandsregelung für die Korridorfahrt. Grundsätzlich haben wir versucht die Korridorfahrt mit softcoding zu lösen, also möglichst wenig vordefinierte Fälle zu programmieren (z.B. eine 45° oder 90° Drehung), da die Korridorfahrt jedoch eine verhältnismäßig geringe Komplexität aufwies, wären ein paar mehr Fallunterscheidungen im Quellcode sinnig gewesen, um vor allem die Grundgeschwindigkeit beim Durchfahren des Korridors höher als 15 cm/s anzusetzen.

In der endgültigen Challenge belegte unser Roboter Rang 4 in der Gesamtwertung. Auffallend war, dass die Bestzeit unseres Roboters mit 1 min und 14 sek knapp 30 sek langsamer war als die Zeiten der Erst- und Zweitplatzierten. Unser Roboter bewegte sich flüssig und ohne Zwischenstops durch den Korridor jedoch war die Grundgeschwindigkeit mit 15 cm/s im Schnitt offensichtlich deutlich unter der Durchschnittsgeschwindigkeit der Gewinnerteams. Da das Gewinnerteam die Ultraschallsensoren in der selben Art und Weise an ihrem Roboter befestigt hat, wie wir, sollte für weitere Verbesserungen der Positionsregler besser ausgelegt werden. Des Weiteren könnte die Idee eines der anderen Teams aufgenommen werden und eine Vorsteuerung implementiert werden, um die PWM-Drehzahl-Kennlinie der Motoren zu Linearisieren und somit die Regelung zu verbessern. Außerdem könnte man noch mehr den Kompromiss eingehen und mehr hardcoding für die Korridorfahrt verwenden, um zumindest abschnittsweise schneller fahren zu können. Zum Beispiel das Nutzen von zwei unterschiedlichen pGain-Werten je nach dem ob der Roboter gerade eine Kurvenfahrt durchführt oder eine gerade Passage überwindet. Ein weiterer Ansatz wäre den Error bei der Berechnung der neuen Radgeschwindigkeit quadratisch einfließen zu lassen.

5 Zusammenfassung

In dieser Hausarbeit wurde ein einfaches mechatronisches System in Form einer manövrierfähigen Roboterplattform mit günstiger Hardware zusammengefügt und programmiert. Beginnend mit der Implementierung einer Fernsteuerung wurde der Programmcode mit Methoden zur Verfolgung einer Linie ergänzt. Als nächstes wurden Encoder angeschlossen, mit deren Einsatz Regler programmiert wurden, die eine Drehzahlsteuerung der Motoren ermöglichen, welche durch das Abfahren eines definierten Kreises bewertet wurden. Im Anschluss daran wurde die Roboterplattform mit Ultraschallsensoren ausgestattet, um autonome Navigationsaufgaben in einer Korridorfahrt meistern zu können. Die Einbindung eines Farbsensors über einen $I^2C - Bus$ bildete die letzte Hardwareergänzung für die abschließende Roboter-Challenge. Durch die Wiederverwendbarkeit von Methoden und Klassen wurde eine sukzessive Weiterentwicklung des Programmcodes ermöglicht.

Die Bearbeitung der Teilaufgaben verliefen einzeln sowie im Team zügig und zielorientiert. Detaillierte Absprachen zu den Aufgabenzuweisungen unter den Teammitgliedern ermöglichten die Einhaltung der Meilensteine. Die frühzeitige Bearbeitung der Aufgaben garantierte zum

Ende der Fristen hin Kapazitäten um unvorhersehbaren Schwierigkeiten zu meistern. Obwohl der Umfang des endgültigen Programmcodes für routinierte Informatiker überschaubar war, bot er uns zahlreiche Möglichkeiten um die Notwendigkeit eines sauberen und konsequenten Programmierstils zu begreifen und unsere Fähigkeiten im Bereich *Software Engineering* zu vertiefen. Zusätzlich konnten wir theoretische Grundlagen der Reglerauslegung anwenden, elektrische Komponenten Verkabeln und dabei ein praktisches Grundverständnis für Schutzschaltungen und Signalglättung bekommen.

Der Raspberrie Pi 3 lieferte für das Projekt eine zufriedenstellende Performance. Obwohl er über prägnant weniger Rechenleitung verfügt als ein standardmäßiger Personalcomputer verfügt und obwohl der Pi mit bis zu 6 Threads simultan abarbeiten musste, blieb das Fahrverhalten dynamisch und die GUI reagierte in der Regel instantan auf Benutzereingaben.

A Abkürzungsverzeichnis

- PMS - Programmierung mechatronischer Systeme
- LKR - Lehrstuhl für Kontinuumsrobotik
- C++ - eine Programmiersprache
- IDE - Integrierte Entwicklungsumgebung (aus dem Englischen: *integrated development environment*)
- Qt - das Programm Qt-Creator
- Pi - Raspberry Pi, verwendeter Einplatinencomputer
- I^2C - Kommunikationsstandard für einen seriellen Datenbus
- GPIO - General purpose input output Pins
- GUI - Graphical User Interface
- PAP - Programmablaufplan nach DIN 66001
- SSH - Netzwerkprotokoll (Secure Shell)
- RGB - Rot Gelb Blau
- PWM - Pulsweitenmodulation

A Programmcode

Literatur

- [1] Klaus Dembowski: *Raspberry Pi - Das Handbuch*. Springer Vieweg, ISBN 978-3-658-03166-4, 2013.