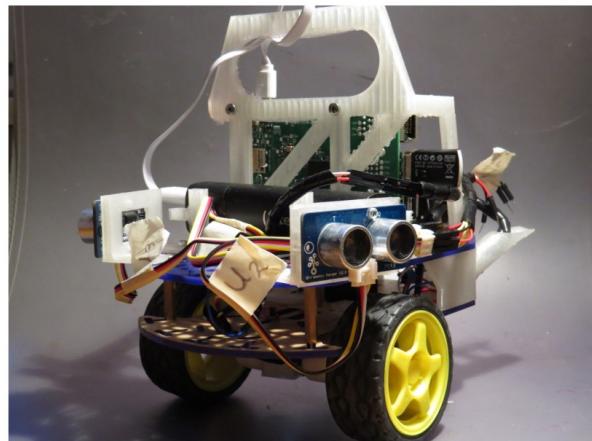


Vorlesung/Experimentelle Übung:
Programmierung mechatronischer Systeme

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
2.2	Sensorik	1
3	Umsetzung der Meilensteine	1
3.1	Linie folgen	1
3.2	Kreisfahrt	3
3.3	Kollisionsfreie Korridorfahrt	6
3.4	Farbfeld-Suche	6
4	Diskussion	6
5	Zusammenfassung	6
A	Programmcode	7

1 Einleitung

ca. 1 Seite

Einleitung zum Projekt. Motivation, Ziele die ihr mit eurem Roboter hattet, Herausforderungen...

In kursiv sind Vorgaben für eure Hausarbeit angegeben, bzw. Anregungen was in dem jeweiligen Abschnitt stehen sollte. Eure Hausarbeit sollte mindestens 13 Seiten haben, maximal 18 Seiten; der Anhang, sowie die Titelseite, Inhaltsverzeichnis und das Literaturverzeichnis werden **nicht** zur Seitenzahl gezählt. Unterstützt eure Hausarbeit visuell mit Fotos und Abbildungen, Tabellen etc.

Generell gilt: Quellen bitte angeben! [1]

2 Der Roboter

3 bis 5 Seiten

In diesem Abschnitt soll euer Roboter hardwareseitig beschrieben werden: Hardwarekomponenten, Elektronik, Schaltpläne. Welche Sensoren wurden ausgewählt und warum? Welche Eigenschaften hat der fertige Roboter? Gewicht, Maximalgeschwindigkeit,...

2.1 Mobile Plattform

2.2 Sensorik

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 einer der linken Sensor einen *Low-Pegel* zurück, befindet sich dieser über weißem Untergrund. Der Roboter ist zu weit nach rechts abgewichen. Es folgt eine Korrektur der Ausrichtung anhand einer Linksdrehung. Analog erfolgt bei einem *Low-Pegel* des rechten Sensors eine Rechtsdrehung.

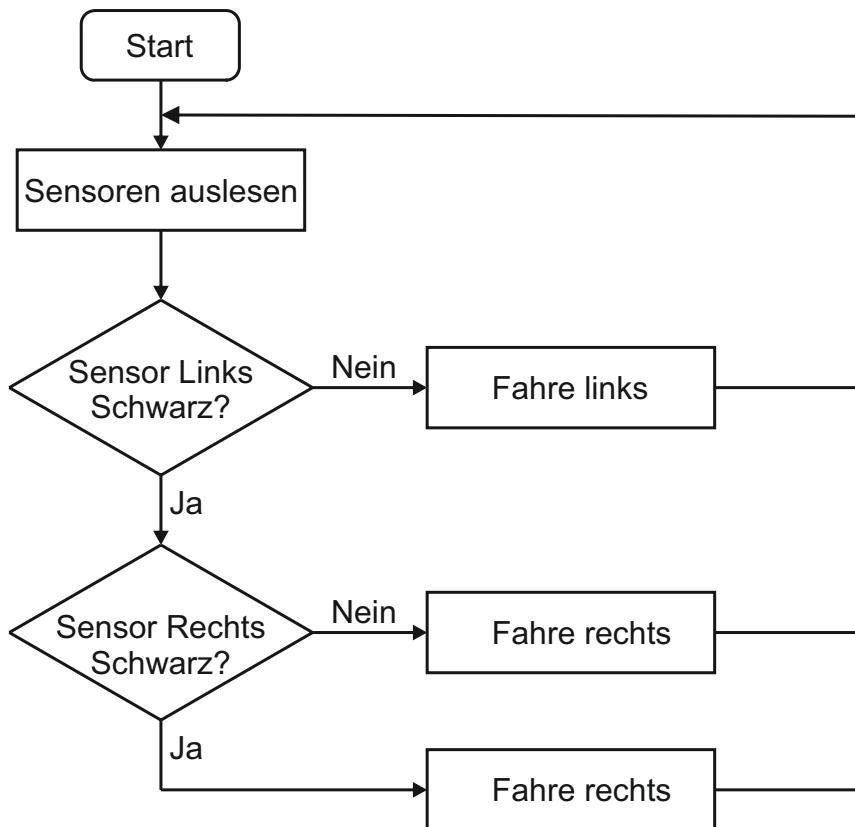


Abbildung 1: Programmablaufplan der Linienfahrt

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.

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 1 unter Berücksichtigung des Raddurchmessers von $d = 0,063\text{m}$ zu einer Geschwindigkeit in SI-Basiseinheiten umgerechnet.

$$v = \frac{kd\pi}{1440\Delta T} \quad (1)$$

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 Encoder vernachlässigt werden. Demzufolge führt die Mittelwertbildung mehrerer Drehzahlssignale 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 ... zeigt ein Blockschaltbild des verwendeten PID-Reglers.

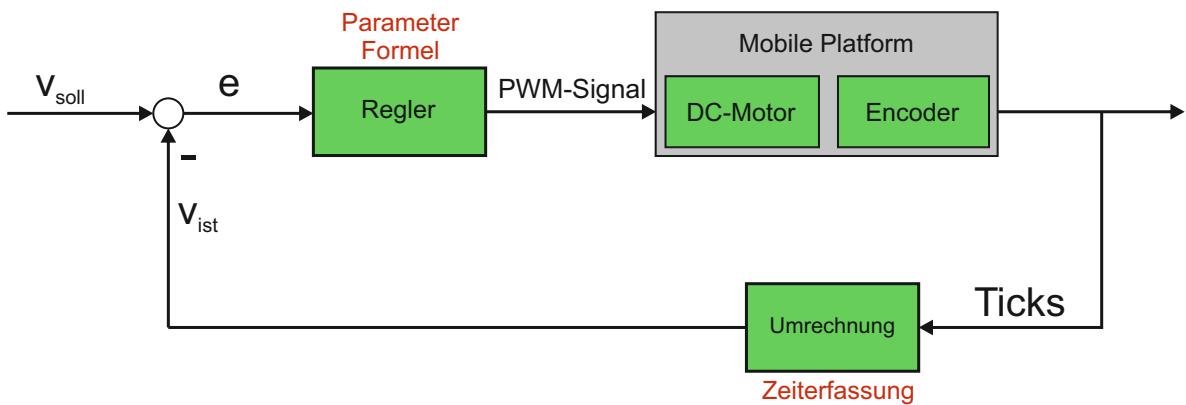


Abbildung 2: Blockschaltbild eines PID-Reglers

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 2 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 der 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 differentielle 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 .

$$y[n] = K_P \cdot e[n] + K_I \cdot \sum_{m=0}^n e[m] \cdot \Delta T + K_D \cdot \frac{e[n] - e[n-1]}{\Delta T} \quad (2)$$

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 3 zu entnehmen. Des Weiteren wurden für beide Motoren verschiedene Parameter ermittelt, da sie bei gleichem Eingangssignal unterschiedliche Drehzahlen aufweisen.

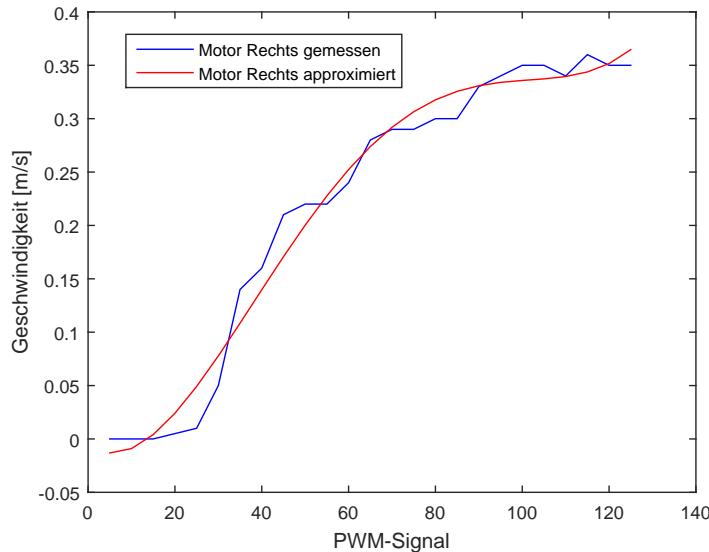


Abbildung 3: Nichtlineare Kennlinie der Drehzahl des rechten Motors

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 4), 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 4, die eine Rotation des Roboters um einen Punkt P darstellt, folgen die Gleichungen 3, 4 und 5 zur Berechnung der Soll-Geschwindigkeiten.

$$r_{soll} = \sqrt{r^2 - s^2} \quad (3)$$

$$v_{Links} = v \cdot \frac{r + \frac{b}{2}}{r} \quad (4)$$

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

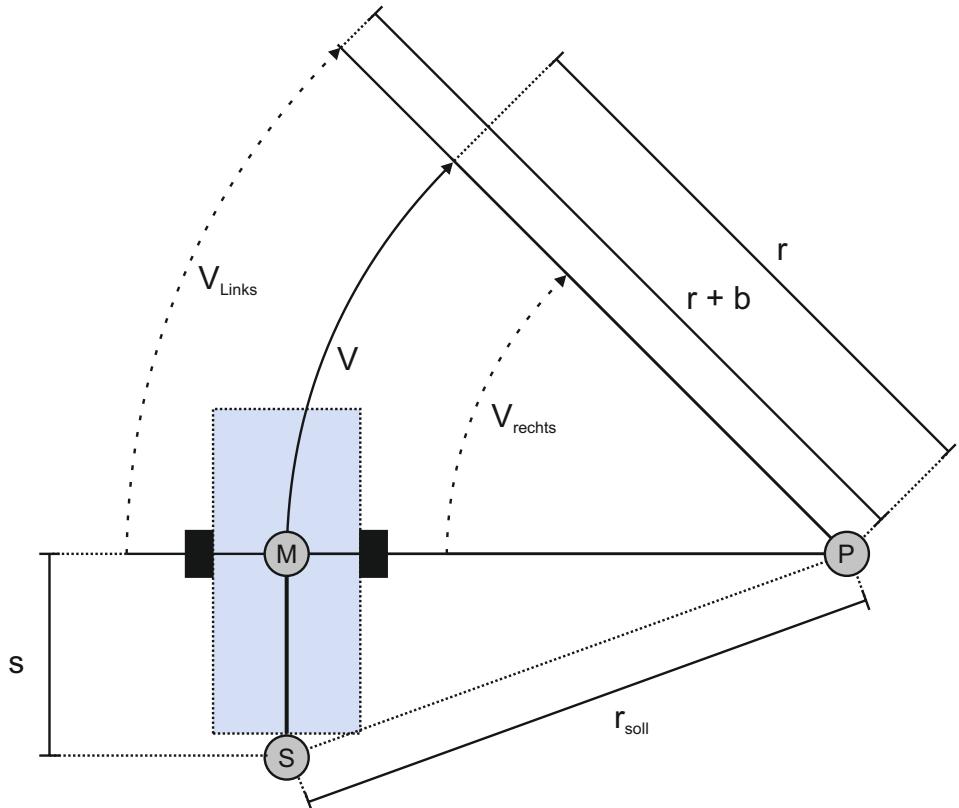


Abbildung 4: Nichtlineare Kennlinie der Drehzahl des rechten Motors

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

Aufgabe des 3. Meilensteins war eine 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. Zur Messung des Abstands von Roboter zu Wand wurden zwei Ultraschallsensoren (siehe Kapitel 2.2) angeschlossen und in einer Klasse *Ultrasonic* angeprochen. Über einen separaten Thread werden die Sensoren nacheinander

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 Farbfelder in den Farben Rot, Grün und Blau ausgelegt. Diese 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 ist, wird die Farbe Rot nach im Verhältnis zu Grün und Blau identifiziert. Die dabei benutzten Schwellenwerte wurden dabei empirisch ermittelt.

4 Diskussion

0.5 bis 1 Seite Probleme mit Hardware/Software, Herausforderungen die euch begegnet sind, Probleme im Team, Warum hat euer Roboter in der Challenge wie abgeschnitten? Gibt es offene Fragen? Wenn ihr jetzt noch weiter daran arbeiten könntet, was würdet ihr als nächstes machen?

5 Zusammenfassung

0.5 Seite

A Programmcode

Literatur

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