TECHNISCHE
UNIVERSITÄT
DRESDEN

Control

James Vero Asghar

16. März 2021

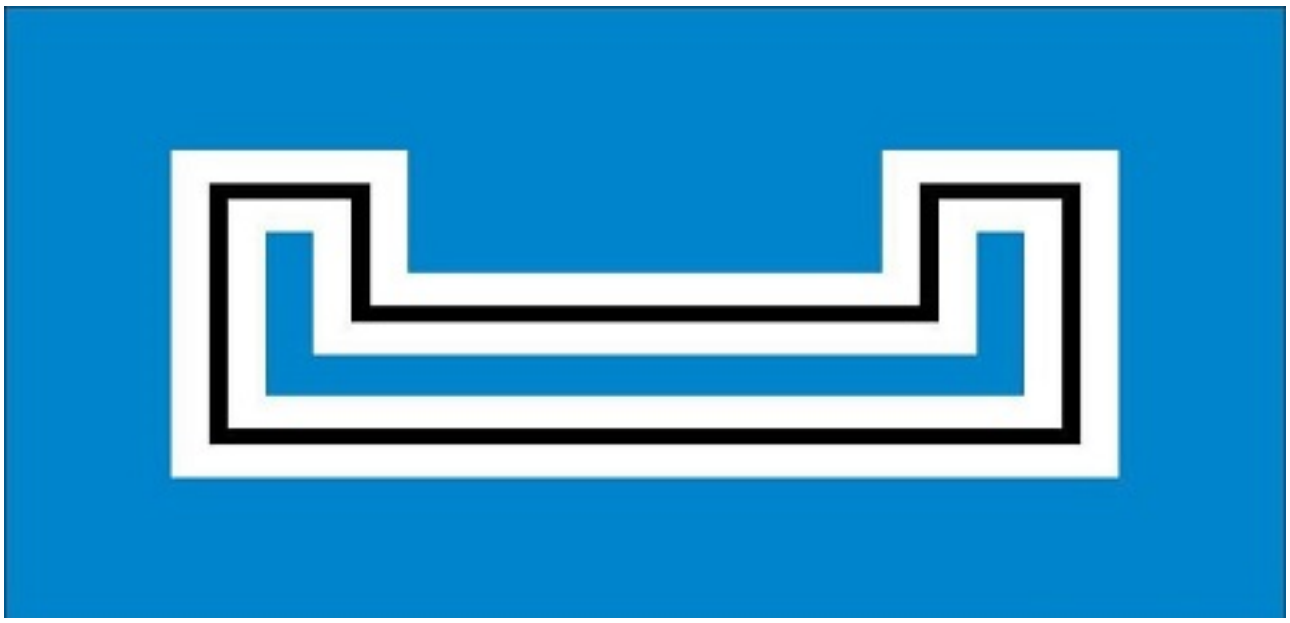

Abbildung 1: Parcour

1 Analyse der Aufgabenstellung

1.1 Motivation, Aufgabenstellung / Ziel des Semesters

In diesem Seminar muss einen Roboter einen Parcour umfahren, eine Parklücke erkennen, darin einparken und dann ausparken. Dieses Seminar ist eine kleine Version des aktuellen Technikproblems von autonomem Fahren. Der Parcour ist im Abbildung 1 zu sehen.

Das Seminarprojekt besteht aus fünf Modulen, die jeweils von einem Gruppenmitglied bearbeitet werden. Die Module sind Guidance, Control, Navigation, Perception und Human-Machine-Interface (HMI). Guidance koordiniert die andere vier Modulen, sodass der Roboter sein Ziel erreichen kann. Control steuert die Bewegung des Roboters. Navigation stellt die aktuelle Position des Roboters fest und identifiziert die Parklücke. Perception kümmert sich um die Messfehler der Sensoren und berechnet, wie präsizese sie sind. HMI erstellt eine Android-App, um ein Interface mit dem Roboter zu haben, sodass ein Benutzer dem Roboter Kommandos geben kann.

Die fünf Module arbeiten eng zusammen. Zum Beispiel muss Navigation mit Perception arbeiten, sodass der Roboter die Messfehler der Sensoren ausgleichen kann, um eine richtige aktuelle Position zu berechnen. Diese Beziehung zwischen den Modulen ist ähnlich wie bei einer reellen Anwendung, weil ein Projekt normalerweise nicht durch nur eine Person erarbeitet werden kann. Stattdessen müssen viele Aufgaben parallel abgeschlossen werden, um das Projekt abzuschließen.

Der Fokus dieser Seminararbeit sind das Control-Modul und alle Teilaufgaben, die dazu gehören.

1.2 Zusammenfassung der einzelnen Teilaufgaben

Das Control-Modul hat vier Teilaufgaben: Linieverfolgung, v/ω-Control, geregeltes Geradeausfahren und Ein-/Ausparken.

Für die Linienverfolgung muss der Roboter mit zwei Lichtsensoren einer schwarzen Linie schnellstmöglichst folgen und um die Ecken des Parcours navigieren.

Für die v/ω-Control muss eine Methode implementiert werden, sodass der Roboter mit einer konstanten Geschwindigkeit und Drehgeschwindigkeit fahren kann. Mit dieser Methode soll der Roboter einfache Kreise, Ellipsen, und für kurze Strecken gerade ausfahren können.

Die dritte Teilaufgabe ist die Verbesserung des Geradeausfahrens. Der Roboter muss nun für lange Strecken gerade ausfahren.

Die vierte und letzte Teilaufgabe ist die Entwicklung eines Algorithmus für das Ein-/Ausparken des Roboters. Diese Aufgabe muss in enger Zussamenarbeit mit dem Guidance-Modul abgeschlossen werden. Das Guidance-Modul liefert Control die Information, wohin der Roboter fahren muss. Control implementiert einen Algorithmus, sodass der Roboter diese Information in eine Bahn übersetzt kann.

1.3 Übergabeparameter von / zu Control-Modul

Das Control-Modul ist das Herz des Roboters. Ohne das Control-Modul kann der Roboter sich nicht bewegen und es braucht im Vergleich zu den anderen Modulen die meistene Parameter. Für die Linienverfolgung braucht Control die Werte der Lichtsensoren von Perception und für v/ω-Control braucht Control die Werte der beiden Motorenencoder, sodass eine sinnvolle Regelung implementiert wird. Für die Geradeausfahrt wird die aktuelle Position gebraucht, die von Navigation berechnet wird. Das Ein-/Ausparken benötigt die Parkbahn von Guidance. Die Parkbahn ist abhängig von dem implementierten Algorithmus und kann auf entweder Zielpunkten oder Polynomen basieren.

2 Linienverfolgung

2.1 Verbesserung des Beispielprogrammes

Das Beispielprogramm bekommt die diskrete Werte 0, 1, und 2 von Perception, um zu verstehen, ob ein Lichtsensor auf eine schwarze oder weisse Linie ist. Der Roboter verändern seine Motoren zwischen zwei PWM-Werten abhängig von den diskreten Werten, sodass er immer um die Linie oscilliert.

Die Schwingungen um die schwarze Linie sind unerwünscht. Um die Schwingungen zu unterdrücken muss der Differenz zwischen den beiden PWM-Werten verkleinert werden. Die PWM-Werten dürfen auch nicht zu hoch sein oder der Roboter kann die Ecken des Parcours nicht navigieren.

2.2 Einführung eines PID-Reglers

Nun wird nur eine Regelgröße $y(t)$ betrachtet. Die ist der Differenz zwischen den linken und rechten Lichtintensitäten der Lichtsensoren.

$$\mathit{Lichtintensität}_L - \mathit{Lichtintensität}_R = \mathit{Regelgröße}\ y(t) \tag{1}$$

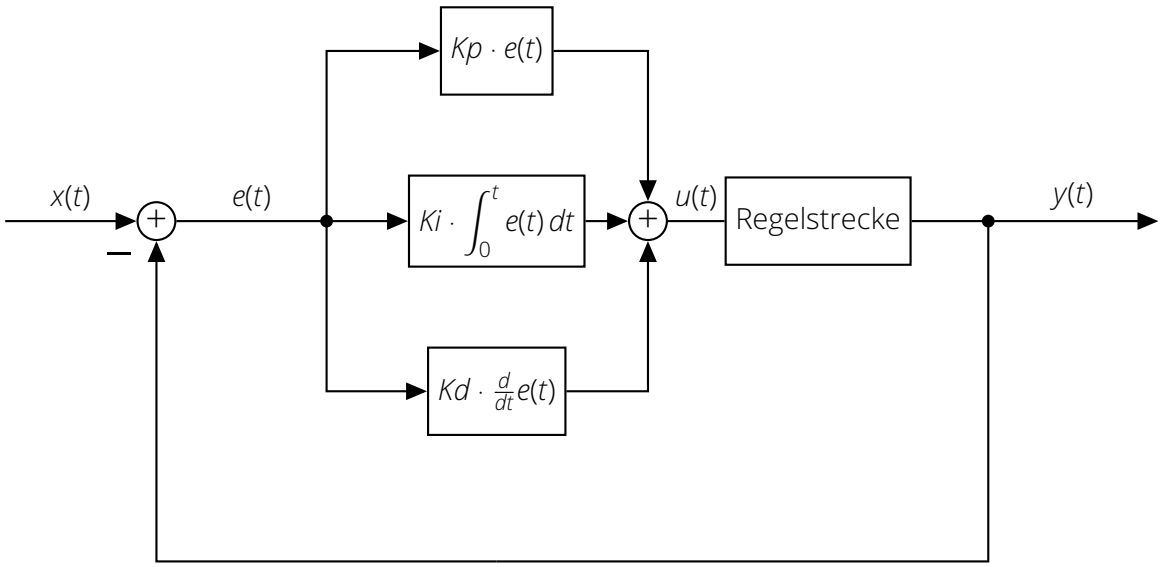

Abbildung 2: Regelkreis

Der Bereich der Regelgröße liegt zwischen –100% und 100%. Die zwei Werte bedeuten, dass bei -100% der rechte Lichtsensor auf weiss liegt und der linke Lichtsensor auf schwarz liegt und bei 100% das Gegenteil. Die Führungsgröße $x(t)$ ist 0%, weil dabei beiden Lichtsensoren auf weiss liegen. Der Regelfehler $e(t)$ ist unverändert. Es gibt nur ein Tupel von PID-Konstanten Kp, Ki, Kd und deshalb ist nur ein Regelkreis benötigt, der nochmal in Abbildung 2 zu sehen ist. Bevor die Ausgabe des Reglers direkt in den Motoren geschickt werden kann, muss zuerst das Vorzeichen von der rechten Motoreingabe umgekehrt werden. Falls die beiden Eingabe der Motoren positiv wären, führt der Regelkreis zu einer Mitkopplung auf dem rechten Motor statt Gegenkopplung. Dieses liegt an dem verändernten Bereich der Regelgröße, weil der nun negative Werte besitzt. Negative Werte bedeuten eine rechte Ablenkung, deshalb muss der rechte Motor einen positiven PWM-Wert bekommen und der linke Motor das Gegenteil, um diese Ablenkung zu kompensieren.

Diese Methode funktioniert mit wenigen Problemen und der Roboter verfolgt die Linie mit den folgenden PID-Konstanten

$$(Kp, Ki, Kd) = (0{,}4, 0{,}2, 0{,}0565)\,.$$

Manchmal überspringt der Roboter eine Ecke und läuft der Roboter in einem Kreis, ohne die Linie wieder zu erkennen. Dieses Verhalten passiert selten, aber scheint es darauf, hauptsächlich die Konversion zwischen Gleitkommazahlen und Integerzahlen. Der Regler gibt Gleitkommazahlen aus, aber die Motoren brauchen Integerwerten. Deswegen muss der Roboter die Ausgabe des Reglers von Double zu Integer typecasten. Das Typecasten führt zu Ungenauigkeiten in den PWM-Werten, deswegen würde der Roboter zu weit ablenken und weg von der Linie fahren.

3 v/ω-Control

3.1 Steuerung

Konnte einfache Kreise am Platz machen, gibt immer eine Abweichung mit der Ellipse und Geradeausfahrt

Der Roboter muss jetzt mit einer bestimmten Geschwindigkeit v und Winkelgeschwindigkeit ω fahren. Zuerst muss die Kinematik des Roboters berechnet werden. Gegeben ist die kinematischen Matrix, die die Geschwindigkeiten der Rädern v_l, v_r zu der Geschwindigkeit des Achsmittelpunktes v_m und der Winkelgeschwindigkeit ω verbinden,

$$\begin{pmatrix} v_m \\ \omega \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 \\ 1/-d & 1/d \end{pmatrix} \begin{pmatrix} v_l \\ v_r \end{pmatrix} \tag{2}$$

mit d als die Achsenlänge. Davon ist die inverse Matrix berechnet, sodass v_l und v_r von v_m und ω ausgerechnet werden können:

$$\begin{pmatrix} v_l \\ v_r \end{pmatrix} = \begin{pmatrix} 1 & -d/2 \\ 1 & d/2 \end{pmatrix} \begin{pmatrix} v_m \\ \omega \end{pmatrix}. \tag{3}$$

Als nächstes muss die Drehzahl den beiden Rädern berechnet werden. Das ist erfolgreich durch die folgende Gleichung:

$$\omega_{l,r} = \frac{v_{l,r}}{R_{l,r}}, \tag{4}$$

mit $R_{l,r}$ als die Radien der linken und rechten Rädern.

Letzlich muss die PWM-Werte $u_{PWM,l,r}$ der Motoren berechnet werden. Obwohl der Zussamenhang zwischen die Drehzahlen $\omega_{l,r}$ und die PWM-Werte nicht linear ist, dürfte der Roboter dieser Zusammenhang als eine lineare Gleichung im Form von

$$u_{PWM,l,r} = k_{m,l,r}\,\omega_{l,r} + u_{PWM,Y-Schnittpunkt,l,r} \tag{5}$$

approximieren, weil der PWM-Wert von dem Y-Schnittpunkt linear steigt.

Die Motorenkonstanten $k_{m,l,r}$ und PWM-Y-Schnittpunkten $u_{PWM,Y-Schnittpunkt,l,r}$ werden experimentell bestimmt. Ein Program läuft auf dem Roboter, das die PWM-Werte der Motoren von 0% bis 100% mit einer Rate von $1V_{PWM}/s$ hochfährt. Jede Sekunde werden die PWM-Werte und Drehzahlen der Motoren aufgenommen. Die aufgenommenen Werte werden in einer Tabelle geschrieben und einem Diagramm dargestellt. Die Drehzahlen sind auf dem X-Achse und die PWM-Werte sind auf dem Y-Achse. Durch lineare Regression wird zwei lineare Funktionen zwischen den Drehzahlen und den PWM-Werten bestimmt. Davon sind die Steigungen der Funktionen die Motorenkonstanten und Y-Schnittpunkte sind die PWM-Y-Werte. Mit dieser Funktion ist die Kette von v_m und ω zu $u_{PWM,l,r}$ fertig.

Mit dieser Methode kann der Roboter einfache Kreise am Platz fahren, aber der Roboter kann nicht ohne Abweichungen gerade ausfahren und Ellipse fahren. Das liegt darauf, dass diese Methode nur eine Steuerung ist und nicht eine Regelung. Bei Steuerung hat die Ausgabe keinen Einfluss auf der Eingabe. Deshalb hat der Regelkreis einer Steuerung keine Gegenkopplung von Ausgabe zu Eingabe. Der Regelkreis der v/ω-Steuerung ist in Abbildung 3 zu sehen. Ohne dieser Rückführung kann der Roboter nicht wissen wie schnell der fährt.

3.2 Regelung

Bei der Einführung eines Reglers ist das Verhalten des v/ω-Steuerung viel genauerer. Der Regelkreis mit dem Regler ist in Abbildung 3 zu sehen. Aber am Anfang lief der Roboter nicht problemlos. Mit der ersten Version des Reglers fährt der Roboter ziemlich genau gerade mit

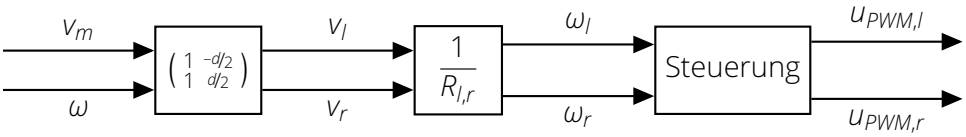

Abbildung 3: *v*/*ω*-Steuerung

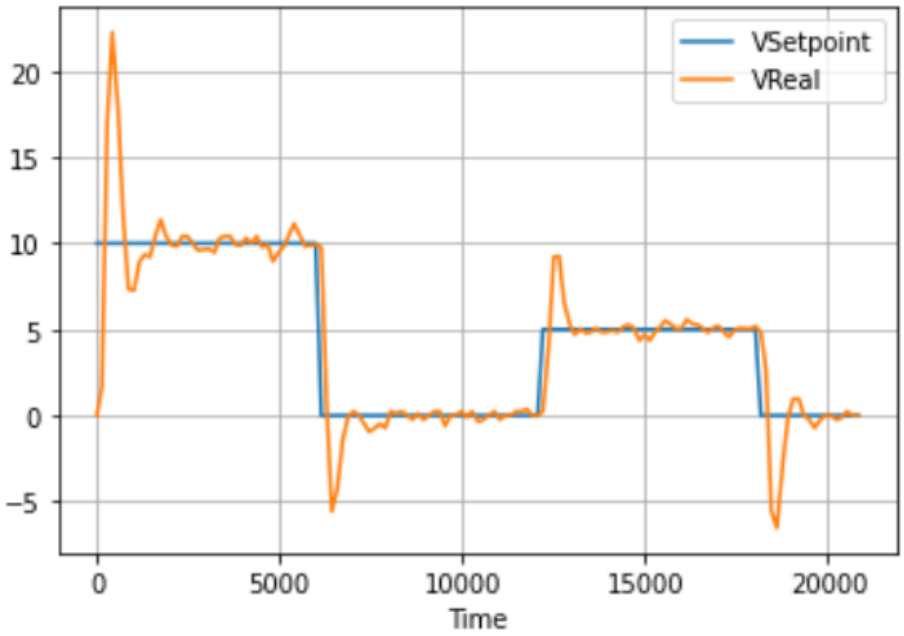

Abbildung 4: Überschwingung

einer präzisen Geschwindigkeit, aber der Roboter konnte nicht am Platz mit einer präzisen Drehgeschwindigkeit drehen. Der Roboter würde immer ein bisschen zu wenig drehen. Diese Problem lag an der Nichtlinearität des Motors. Jeder Motor hat eine bestimmte tote Zone oder in englischen “Deadzone”, die der Motor beseitigen muss, bevor er anfangen kann, zu drehen. Die tote Zone kommt von der inneren Reibung des Motors und muss bei jedem Regler berücksichtigt werden.

Eine Methode, um diese tote Zone zu kompensieren, ist durch die Implementation einer Steuerung. Weil der Roboter schon eine Steuerung hatte, konnte der Roboter gerade fahren. Aber mit nur dieser Steuerung konnte der Roboter problemlos drehen. Deswegen wird ein Arbeitspunkt zu dem Regler hinzufügt.

Die zweite Version des Reglers hat ein richtig präzise Führungsverhalten mit dem gerade Fahren und Drehen. Aber der Arbeitspunkt des Reglers liefert eine ziemlich hohe Überschwingung zu der Regelstrecke, die manchmal zwingt, der Roboter zu weit zu drehen. Die Überschwingung ist in Abbildung 4 zu sehen.

Die dritte Version des Reglers implementiert eine nichtlineare Funktion, um den Arbeitspunkt des Regler zu setzen. Bei niedriger Geschwindigkeit und Drehgeschwindigkeit wird der Arbeitspunkt hoch gesetzt, um die tote Zone zu kompensieren, und bei hoher Geschwindigkeit und Drehgeschwindigkeit wird er tiefer gesetzt. Die nichtlineare Funktion wird benötigt, weil der Arbeitspunkt plus Steuerung zusammen eine Überkompensation zu der Regelstrecke liefert. Deswegen wird der Arbeitspunkt tiefer, wenn er bei hoher Geschwindigkeit und Dreh-

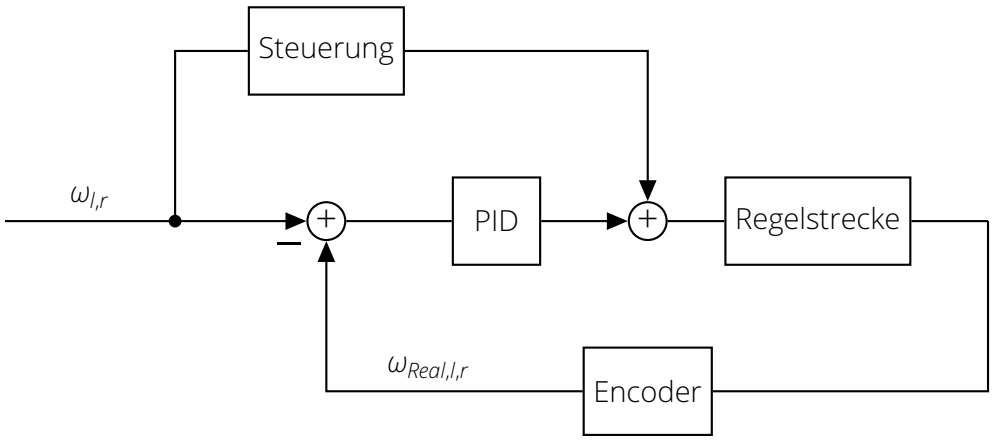

Abbildung 5: v/ω-Regelung

geschwindigkeit nicht mehr benötigt ist. Der fertige Regelkreis ist in Abbildung 5 zu sehen.

Die Implementierung der dritten Version des Reglers auf dem Roboter erlaubt die präsizen Bewegungen von Kreisen und Ellipsen. Der Roboter kann auch für ziemlich weite Strecken auch gerade fahren. Manchmal dreht er sich ein bisschen zu weit, aber das ist selten.

4 Regelung der Geradeausfahrt

Die Implementation einer geregelten Geradeausfahrt eines Roboters braucht ein paar Schritte. Zuerst muss ein funktionierdes Modell des Roboters gefunden werden. Danach muss irgendeine Querabweichung berücksichtigt werden. Am Ende kann ein Regler für das System gefunden werden.

4.1 Linearisierung in X_2-Richtung

Die folgende Modellgleichung eines einachsigen Roboters wird gegeben:

$$\vec{f}(\vec{x}, \vec{u}) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \sin(x_3) & 0 \\ \cos(x_3) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6}$$

.

Wegen des nichtlinearen Forms der Modellgleichung ist es schwierig das Systemverhalten des Modells zu beobachten. Deshalb wird sie in einem Arbeitspunkt linearisiert und dann wird dieses Modells betrachtet. Im Folge ist die Herleitung des linearisierten Modells:

$$\vec{\tilde{f}}(\vec{\tilde{x}}, \vec{\tilde{u}}) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cos(x_{30})v_0 \\ 0 & 0 & -\sin(x_{30})v_0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix} + \begin{bmatrix} \sin(x_{30}) & 0 \\ \cos(x_{30}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{v} \\ \tilde{\omega} \end{bmatrix} \tag{7}$$

. Mit

$$\vec{\tilde{x}} = \begin{bmatrix} x_1 - x_{10} \\ x_2 - x_{20} \\ x_3 - x_{30} \end{bmatrix} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix}, \vec{\tilde{u}} = \begin{bmatrix} v - v_0 \\ \omega - \omega_0 \end{bmatrix} = \begin{bmatrix} \tilde{v} \\ \tilde{\omega} \end{bmatrix},$$

x_{10} und x_{20} sind die Komponenten des Vektors, wo der Roboter startet. Die können dann auf 0 gesetzt werden. x_{30} ist der Anfangswinkel des Roboters, welche hier 0° ist. Die obere Gleichung

lässt sich dann in der folgenden Gleichung vereinfachen:

$$\widetilde{f}(\overrightarrow{\widetilde{x}}, \overrightarrow{\widetilde{u}}) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} v_0 \widetilde{x_3} \\ \widetilde{v} \\ \omega \end{bmatrix} \tag{8}$$

4.2 Querabweichung

In der reellen Welt kann der Roboter nicht perfekt gerade fahren und hat einen kleinen Winkel von der X_2-Richtung. Deshalb muss die Querabweichung der Fahrt berücksichtigt werden. Die Querabweichung $\overrightarrow{e}$ ist als der Vektor zwischen dem geplanten Zielpunkt $\overrightarrow{r}$ und der jetztigen Endposition $\overrightarrow{x}$ des Roboters definiert, wie in der folgenden Gleichung gezeigt wird:

$$\overrightarrow{e} = \overrightarrow{x} - \overrightarrow{r}.$$

Der Zielpunkt $\overrightarrow{r}$ ist bei der Geradeausfahrt nur die X_2-Komponent des Positionsvectors. Die Gleichung im Vector-Form wird in der folgenden Gleichung gezeigt:

$$\begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ x_2 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix}$$

.

Die obere Gleichung lässt sich mit der folgenden eindimensionalen Gleichung vereinfachen:

$$e(t) = x_1(t). \tag{9}$$

Bei der ersten Ableitung der oberen Gleichung nach der Zeit wird der Zussamenhang zwischen der Querabweichung und dem Modell gezeigt. Der Zussamenhang ist in den folgenden Differentialgleichungen gezeigt:

$$\begin{aligned} \dot{e} &= \dot{x}_1 = v_0 \widetilde{x}_3(t) \\ \ddot{e} &= v_0 \dot{\widetilde{x}}_3(t) = v_0 \omega(t) \end{aligned} \tag{10}$$

4.3 Wurzelortskurven

Um das Systemverhalten mit verschiedenen Reglertypen zu untersuchen, muss der Differentialgleichung 10 zu einer Übertragungsfunktion transformiert werden. Mit der Laplacetransformation wird

$$P(s) = \frac{E(s)}{\Omega(s)} = \frac{v_0}{s^2} \tag{11}$$

erhaltet.

Mit einem Pythonskript wird die Strecke nun untersucht. Die

4.4 Implementation

Zur Zeit des Abschliessens dieser Seminararbeit könnte eine funktionierende Implementation der geregelten Geradeausfahrt nicht gefunden werden. Mehr Information steht im Kapitel 6.

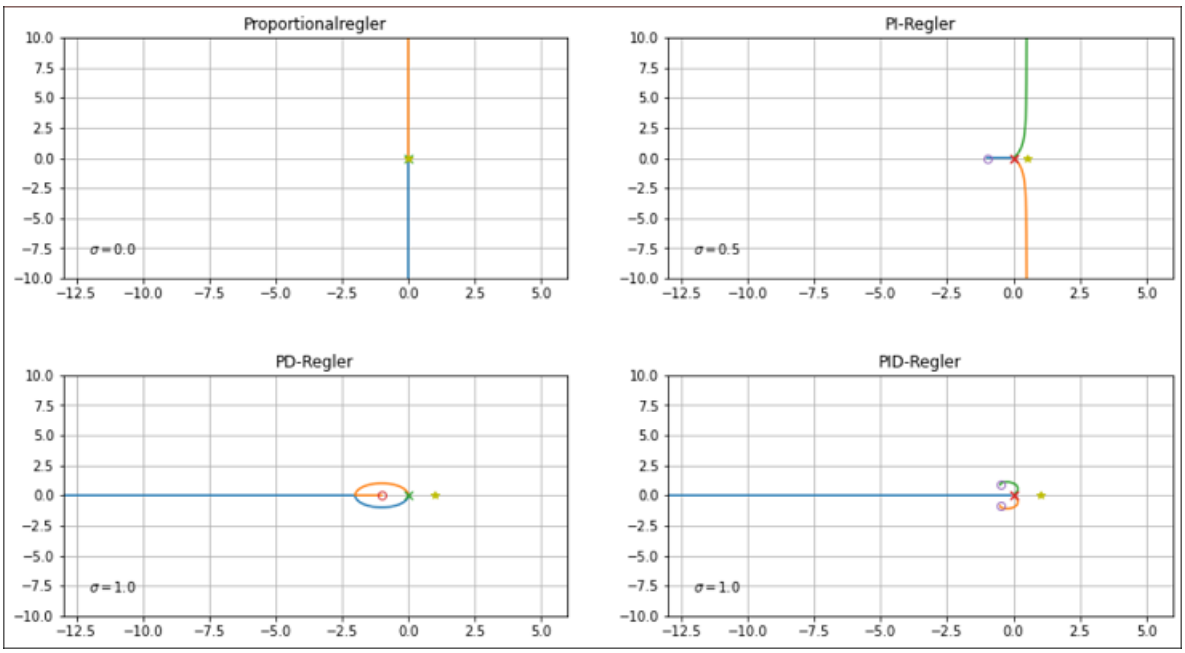

Abbildung 6: Wurzelortskurven

5 Ein- und Ausparken

5.1 Theoretische Vorbetrachtung

Nun wird ein Ein-/Ausparkalgorithmus implementiert. Das heißt, dass der Roboter von der Linien zu dem Mittelpunkt einer Parklücke fährt und dann mit einem bestimmten Kommando wieder zur Linie fährt. Es gibt ein paar Methoden, um dieses Ziel zu implementieren. Die sind ein Drehen-Fahren-Drehen-Sequenzen, eine ein dimensionale Trajektorie und eine zwei dimensionale Trajektorie.

Als erstes kann der Roboter nach der Anerkennung der Parklücke stoppen, dann schrittweise drehen und fahren, bis der Roboter am Zielpunkt angekommen ist. Diese Methode ist einerseits einfach zu implementieren, andererseits ist es langsam und braucht einen zusätzlichen Regler, weil der Roboter wissen muss, wie weit es drehen und fahren muss.

Die zweite Methode für das Algorithmus ist eine ein dimensionale Trajektorie. Das heißt, dass der Roboter ein Polynom von seinem Startpunkt zum Endpunkt erstellt. Der Roboter stellt eine Richtung in seiner X1-X2-Ebene als die unabhängige Variable des Polynoms und die andere Richtung als die abhängige Variable des Polynoms fest. Im Folge ist die Herleitung der Koeffizienten:

$$
\begin{aligned}
y(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\
\dot{y}(x) &= 3a_3x^2 + 2a_2x + a_1 \\
\ddot{y}(x) &= 6a_3x + 2a_2
\end{aligned}
$$

mit

$$
a_3 = -\frac{2}{x_f^3}(y_1 - y_0), a_2 = \frac{3}{x_f^2}(y_1 - y_0), a_1 = 0, a_0 = y_0.
$$

Diese Gleichung wird mit der Annahme gelöst, dass die Geschwindigkeit der Bahn am Anfang und am Ende 0 ist.

Die dritte Methode dafür ist eine zweidimensionale Trajektorie. Der Roboter erstellt eine Bahn zwischen seinem Startpunkt und dem Zielpunkt. Die Bahn ist eine Kombination aus zwei Funktionen $x_1(t)$ und $x_2(t)$. Die zwei Funktionen sind parametrisierte Polynomen dritter Grades, die abgeleitet werden können, um die Geschwindigkeit und Drehgeschwindigkeit zu berechnen. Der Anfangspunkt der Herleitung ist die Modellgleichung 6. Im Folge ist die restliche Herleitung der Geschwindigkeiten:

$$\begin{bmatrix} \sin(x_3) & \cos(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \sin(x_3) & \cos(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sin(x_3) & 0 \\ \cos(x_3) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$\begin{bmatrix} \sin(x_3) & \cos(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \sin^2(x_3) + \cos^2(x_3) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}.$$

Mit $\sin^2(\alpha)+\cos^2(\alpha) = 1$ lässt sich die obere Gleichung zur folgenden Gleichung vereinfachen:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \sin(x_3) & \cos(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix}. \tag{12}$$

x_3 lässt sich aus dem Winkel von der $\dot{X}_2$-Richtung berechnen. Mit den folgenden Gleichungen

$$x_3 = \arctan(\frac{\dot{x}_2}{\dot{x}_1})$$
$$\frac{d}{dt}x_3 = \frac{d}{dt}\arctan(\frac{\dot{x}_2}{\dot{x}_1})$$

kann ω bestimmt werden. Der Roboter verwendet die folgende Gleichung, um die richtige Drehgeschwindigkeit zu berechnen.

$$\dot{x}_3 = \frac{\dot{x}_1\ddot{x}_2 - \dot{x}_2\ddot{x}_1}{\dot{x}_1^2 + \dot{x}_2^2} = \omega \tag{13}$$

Diese Methode hat die gleiche Vorteile als die eindimensionale Version, aber sie vermindert die Nachteile davon. In dem eindimensionalen Fall muss der Roboter die Parcours Koordinatensystem mit dem Roboters Koordinatensystem übereinstimmen. Danach muss es sich entscheiden, welche Variable unabhängig und abhängig ist, und dann kann der Roboter der Polynom erstellen. Der Roboter berechnet dann die richtige Geschwindigkeit und Drehgeschwindigkeit. In dem zweidimensionalen Fall werden nur die Geschwindigkeiten berechnet. Der zweidimensionale Fall vereinfacht das Prozess bei vielen Schritten, aber diese Methode braucht zwei Polynomen im Gegensatz zu einem.

Es ist wichtig, dass die Polynomen für beide Trajektorie dritte Grad oder höher sind. Wenn ein Polynom dritter Grades verwendet wird, kann die zweite Ableitung des Polynomes auf Null bei dem Startpunkt und Endpunkt der Bahn gesetzt werden. Diese Randbedingungen erlaubt das Lösen des Gleichungssystems und daraus wird die Koeffizienten des Polynoms berechnet.

5.2 Implementierung

Um die zweidimensionale Trajektorie zu implementieren, muss eine Bahnplanalgorithmus gewählt werden. Es gibt viele Variante, die implementiert werden können, aber hier wird eine sogennante Bézier-Kurve verwendet. Die Koeffizienten der zwei Polynomen unterscheiden sich von

dem eindimensionalen Fall, aber die Polynomen sind noch dritter Grad. Die Gleichung der Bézier-Kurve hat den folgenden Form:

$$\overrightarrow{B}(s) = (1-s)^3\overrightarrow{P_0} + 3(1-s)^2 s\overrightarrow{P_1} + 3(1-s)s^2\overrightarrow{P_2} + s^3\overrightarrow{P_3}, s \in [0,1].$$

Mit

$$\overrightarrow{P_i} = \begin{bmatrix} x_{1i} \\ x_{2i} \end{bmatrix}, i \in [0,3]$$

Eine Bézier-Kurve dritter Grades braucht normalerweise vier Kontrollpunkte. Aber es lässt sich vereinfachen, wenn die vier Kontrollpunkte in einem Viereck sich befinden. Damit können alle Koeffizienten der zwei Polynomen mit nur zwei Kontrollpunkte gefunden werden.

$$\begin{aligned} x_1(s) &= 4(x_1 - x_0)s^3 + 6(x_0 - x_1)s^2 + 3(x_1 - x_0)s + x_0 \\ \dot{x}_1(s) &= 12(x_1 - x_0)s^2 + 12(x_0 - x_1)s^2 + 3(x_1 - x_0) \\ \ddot{x}_1(s) &= 24(x_1 - x_0)s + 12(x_0 - x_1) \end{aligned}$$

$$\begin{aligned} x_2(s) &= 2(y_1 - y_0)s^3 + 3(y_0 - y_1)s^2 + y_1 \\ \dot{x}_2(s) &= 6(y_1 - y_0)s^2 + 6(y_0 - y_1)s \\ \ddot{x}_2(s) &= 12(y_1 - y_0)s + 6(y_0 - y_1) \end{aligned}$$

Die oberen Gleichungen sind nicht einstellbar nach der Zeit t. Sie arbeiten nur mit dem Parameter s, wenn die Kurve ungeändert implementiert wird, ist die Bewegung zu schnell für den Roboter. Deshalb muss eine Funtkion gefunden werden, sodass Parameter s einstellbar nach der Zeit s ist. Mit der linearen Gleichung:

$$s(t) = \frac{t}{T}$$

wird eine lineare Beziehung der Parameter s und die Zeit t erstellt. T ist die Zeitdauer der Bewegung. Wegen dem Kettenregel müssen auch T in den Ableitungen von $x_1(s(t))$ und $x_2(s(t))$ berücksichtigt werden. Im Folge sind die geänderten Gleichungen:

$$\begin{aligned} x_1(s(t)) &= 4(x_1 - x_0)s^3 + 6(x_0 - x_1)s^2 + 3(x_1 - x_0)s + x_0 \\ \dot{x}_1(s(t)) &= \frac{1}{T}(12(x_1 - x_0)s^2 + 12(x_0 - x_1)s^2 + 3(x_1 - x_0)) \\ \ddot{x}_1(s(t)) &= \frac{1}{T^2}(24(x_1 - x_0)s + 12(x_0 - x_1)) \end{aligned}$$

$$\begin{aligned} x_2(s(t)) &= 2(y_1 - y_0)s^3 + 3(y_0 - y_1)s^2 + y_1 \\ \dot{x}_2(s(t)) &= \frac{1}{T}(6(y_1 - y_0)s^2 + 6(y_0 - y_1)s) \\ \ddot{x}_2(s(t)) &= \frac{1}{T^2}(12(y_1 - y_0)s + 6(y_0 - y_1)). \end{aligned}$$

Mit der Implementation dieser Gleichungen fährt der Roboter erfolgreich die geplante Kurve in der Zeitdauer T.

6 Anmerkungen und Verbesserungsmöglichkeiten

Ich wollte in diesem Kapitel nur ein paar Probleme und Lösungen von dem Projekt schreiben, sodass ich ein besseres Bild des ganzen Prozesses präsentieren kann.

In diesem Seminar sind bei Control vier Probleme rausgekommen. Interessanterweise hat jede Teilaufgabe ein großes Problem, das gelöst werden muss. Die Probleme waren die Implementation des Reglers auf der Linienverfolgung, ein Missverständnis bei dem Unterschied von Steureung und Regelung der v/ω-Control, die ganze Positionsregelung und die Berechnung von v/ω aus der Parkbahn.

Für die Linienverfolgung wurde am Anfang ein andere Regler implementiert. Zwei Regelgrößen wurden $y_1(t)$ und $y_2(t)$ betrachtet. Dabei handelt es sich um die Lichtintensitäten von beiden Lichtsensoren, die zwischen 0% (Schwarz) und 100% (Weiss) angegeben wurden. Zwei Führungsgrößen $x_1(t)$ und $x_2(t)$von 100% wurden eingeführt, sodass sich zwei Regelfehler $e_1(t)$ und $e_2(t)$ ergeben:

$$100\%\ (\mathit{Weiss}) - \mathit{Lichtintensität}_{L,R} = \mathit{Regelfehler}\ e_{1,2}(t).$$

Den zwei Regelfehler wurden die zwei PID-Regler zugeführt, sodass die Regler die zwei Stellgrößen $u_1(t)$ und $u_2(t)$ für die beiden Motoren ausgegeben haben. Dies sind die PWM-Abweichungen $u_{PWM,1}$ und $u_{PWM,2}$, die zwischen 0% und 100% liegen. Die Stellgrößen schwanken um einen Arbeitspunkt $u_{PWM,AP}$ von 37%. Der Regler sollte die PWM-Werte so ändern, dass der Roboter die schwarze Linie verfolgt hat. Deshalb musste der Regler mit verschiedenen Werten für Kp, Ki, Kd justiert werden. Wegen der zwei Regelgrößen war es nötig, zwei Tupel von PID-Konstanten zu justieren. Aber wegen der Gleichheit der Motoren könnten beide Regler die gleichen Konstanten haben. Deswegen hatten bei dieser Implementation beide Regelgrößen den gleichen Regelkreis.

Diese Methode war leider fehlerhaft. Die Annahme, dass beide Motoren das Gleiche sind, war falsch. Im Kapitel 3 wurde es dann gezeigt, dass die beide Motoren unterschiedliche Motorkonstante k_m hatten. Deswegen mussten die PID-Regler zwei Tupel von Konstanten haben. Es war zu schwierig zwei Regler gleichzeitig zu justieren, deshalb habe ich zu der anderen Methode, als ich im Kapitel 2 geschrieben habe.

Bei dem v/ω-Control konnte ich die Steuerung relativ schnell implementieren. Das Problem war die Regelung. Als ich am Anfang die Regelung implementiert habe, war ich überhaupt nicht sicher, warum ich den Regler nicht justieren konnte. Ich habe Monate darauf gearbeitet. Die Lösung kam, als ich einfach alles zu meiner Gruppe geklärt habe. Anscheidend habe ich immer den Reglerausgang direkt in der Steuerungsgleichung geschickt, statt der Regelstrecke (die Motoren). Dieses Problem lag einfach auf dem Grund, dass ich damals die Implementierung eines Regelkreises auf Hardware nicht verstanden habe. Als ich dieses Missverständnis korrigiert habe, konnte ich dann auf der anderen Versionen der Regler weiterarbeiten, wie ich im Kapitel 3 geschrieben habe.

Nachdem ich dieses Problem gelöst habe, konnte ich schnell den Regler justieren. Dies lag auf dem Grund, dass ich ein Pythonskript geschrieben habe, dass ein Diagramm aus den Werten des Monitorprogramms dargestellt hat. Das war einfach die beste Entscheidung, die ich seit dem Anfang des Projekts getroffen hat. Es hat die Zeit, jede Teilaufgabe zu debuggen, verkleinert.

Es gab zwei Probleme bei der Positionsregelung. Das erste Problem war einfach eine Zeitbegrenzung, die nur an mich liegt. Ich hatte schon Flüge nach den USA gebucht, deshalb hatte ich nur zwei Tage zu testen. Falls ich die Positionsregelung nicht implementieren könnte, muss ich dann aufhören, weil meine Partner hatten, noch viel zu testen. Das zweite Problem war die Blockade, die ich in der Zeit nicht beseitigen konnte. Wenn der Roboter einen Zielpunkt bekommt, der nicht kollinear mit der X_2-Richtung des Roboters ist, dreht er sich, bis er damit kollinear ist. Das erste Problem ist, dass er dann manchmal um diese Richtung oscilliert. Das zweite Problem ist, wenn eine Grenze implementiert ist, sodass er nicht mehr darum oscilliert, fängt er manchmal an, gerade zu fahren und manchmal nicht. Wenn der Roboter fährt, fährt er dann nicht geradelinig und immer in einem großen Bogen. In den zwei Tagen konnte ich dieses Problem nicht lösen und musste ich der Roboter an meiner Gruppe übergeben.

Ehrlich zu sagen, gab es nicht ein großes Problem. Aber ich hatte ein Problem bei der Berechnung von ω aus der Modellgleichung. Ich dachte, dass die Gleichung

$$x_3 = \arctan(\frac{x_2}{x_1})$$

richtig war. Aber die echte Gleichung wie vorhin geschrieben ist

$$x_3 = \arctan(\frac{\dot{x}_2}{\dot{x}_1}).$$

Als ich dieses Problem gelöst habe, konnte ich den richtigen Wert von ω finden und das Ein-/Ausparken testen. Dabei musste ich nur den richtigen Zeitdauer der Bewegung finden, was nicht wirklich schwierig war.