Abschlussbericht TURacers Projektseminar Echtzeitsysteme

Final Report of the TURacers Group for Projektseminar Echtzeitsysteme
Seminar eingereicht von
Ruixin Du, Georg Kriener, Jonas Krüger, Johannes Silberbauer, Marcel Verst
am 7. April 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachgebiet Echtzeitsysteme

Elektrotechnik und Informationstechnik (FB18)

Zweitmitglied Informatik (FB20)

Prof. Dr. rer. nat. A. Schürr
Merckstraße 25
64283 Darmstadt

www.es.tu-darmstadt.de

Gutachter: Géza Kulcsar
Betreuer: Eric Lenz

Erklärung zum Seminar

Hiermit versichere ich, das vorliegende Seminar selbstständig und ohne Hilfe Dritter angefertigt zu haben. Gedanken und Zitate, die ich aus fremden Quellen direkt oder indirekt übernommen habe, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

Ich erkläre mich damit einverstanden, dass die Arbeit auch durch das Fachgebiet Echtzeitsysteme der Öffentlichkeit zugänglich gemacht werden kann.

Darmstadt, den 7. April 2018

(R. Du, G. Kriener, J. Krüger, J. Silberbauer, M. Verst)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In diesem Bericht beschreiben wir unsere Tätigkeiten, Aufgaben und Ergebnisse im Laufe des Projektseminars des Fachbereichs IES Echtzeitsysteme. Die grundlegende Idee des Projektseminars ist es, den Studierenden eine Vertiefung in der Entwicklung echtzeitkritischer Software am Beispiel eines autonomen Fahrzeugs zu geben. Durchgeführt wird das Projektseminar in Teams zu je fünf Personen. Diese Teams können aus verschiedenen Aufgabenstellungen die wählen, die sie am meisten interessieren. Dabei zählt die Aufgabe einen hindernisfreien Rundkurs zu bewältigen als Pflichtaufgabe. Wir haben uns die Aufgaben „Rundkurs mit Hindernissen“ und „Spurhaltung“ ausgewählt. Die Aufgabenstellungen, Implementierungen und Evaluation dieser Aufgaben sind Hauptinhalte dieses Berichts.

In Kapitel 2 gehen wir zunächst auf unser Team selbst ein, zum Beispiel wie die Aufgaben verteilt wurden und den Ablauf der Meetings mit den Betreuern. Des Weiteren wird die Arbeitsumgebung, sowohl Hardware (Fahrzeug) als auch Software (ROS) beschrieben. Abschließend wird noch die Toolchain dargestellt, die notwendig ist, um ein lauffähiges Programm zu erzeugen.

In den nächsten Kapiteln 3, 4 und 5 werden die drei bereits genannten Aufgaben vorgestellt. Dabei wird jeweils zuerst die Aufgabenstellung vorgestellt und dann die erarbeitet Implementation erläutert. Abschließend folgt eine Evaluation der vorgestellten Lösung.

Zuletzt folgt ein kleines Fazit in Kapitel 6. Dort gehen wir darauf ein, was wir in diesem Projektseminar gelernt haben und wie unsere Lösungen noch hätten verbessert werden können.

2 Teamorganisation und Arbeitsumgebung

Dieses Kapitel beschreibt die Organisation im Team, die Verteilung der Aufgaben und Meetings. Außerdem wird die zur Verfügung gestellte Arbeitsumgebung vorgestellt, welche das Fahrzeug inklusive ROS Betriebssystem beinhaltet.

2.1 Aufgabenplanung

Zu Beginn des Seminars hatten wir uns zunächst mittels Gruppenchat untereinander abgesprochen, wann und wo wir uns treffen beziehungsweise wer welche Aufgaben macht. Nach einer kurzen Zeit wurde das allerdings unübersichtlich und wir mussten uns ein System heraussuchen, welches uns bei der Aufgabenverteilung und Planung unterstützt. Basierend auf Empfehlung unseres Betreuers und teilweise eigener Erfahrung im Team haben wir uns für **Trello** entschieden. Hierbei handelt es sich um eine Web-Oberfläche, welches ein Karteikastensystem implementiert. Dabei stellen die Kärtchen die einzelnen Aufgaben dar und die einzelnen Kasten sind Meilensteine, die man mit einer Aufgabe absolviert. Als Kästen haben wir uns für die Oberbegriffe **ToDo**, **Im Gange** und **Erledigt** entschieden, welche den Bearbeitungsstand einer Aufgabe symbolisieren. Eine Aufgabe wird dann je nachdem in welchem Bearbeitungsstand sie sich gerade befindet in das jeweilige Kästchen einsortiert, somit hat jedes Teammitglied die Möglichkeit, zu sehen, was aktuell gemacht wird. Weiterhin besteht die Möglichkeit, den Aufgaben Fristen und bestimmte Personen zuzuweisen. So können sehr gezielt Aufgaben im Team verteilt und Deadlines gesetzt werden.

2.2 Dateiverwaltung

Als Tool zur Dateiverwaltung haben wir uns für **BitBucket** entschieden, welches die Funktionalitäten von **Git** enthält. Git ermöglicht es Dateien, die von mehreren Benutzern gleichzeitig bearbeitet werden, in einem zentralen Repository zu speichern. Es erleichtert außerdem die Versionierung der Dateien und erhöht somit die Nachvollziehbarkeit von Änderungen.

2.3 Meetings

Zum Austausch mit dem Betreuer des Projektseminars fanden regelmäß Besprechungen statt. Dabei wurde der aktuelle Stand der Implementierung, aufgetretene Probleme und die nächtsten geplanten Aufgaben diskutiert. Bei Problemen gaben die Betreuer und Tutoren des Projektseminars immer hilfreiche Tipps und Anregungen. Des Weiteren fand in regelmäßigen Abständen ein Regelungstechniktreffen statt, bei dem verschiedene Themen und aufgetretene Probleme im Bereich der regelungstechnischen Aufgaben mit den anderen Gruppen diskutiert wurden konnten.

2.4 Fahrzeugbeschreibung

Bei dem Fahrzeug wurde vom Fachgebiet selbst aufgebaut und über die Jahre stetig erweitert und verbessert. Das Fahrzeug besteht aus dem Grundgerüst, welcher aus Unterbau, Räder, Motoren, Seitenwände, Kamerahalterung und Schutzpolster besteht.

Auf dem Prozessor des Fahrzeugs läuft ein Ubuntu Betriebssystem, auf welchem die Grundpakete des Robot Operating System (ROS) installiert und eingerichtet wurden. Das Fahrzeug verfügt über verschiedenen Sensoren:

- Ultraschall: Front, Links, Rechts
- Gyroskop (x,y,z)
- Inertial Measurement Unit (IMU)
- Hall-Sensor

2.5 Toolchain für das Fahrzeug

Um aus dem Sourcecode nun ein lauffähiges Programm zu erzeugen sind verschiedene Schritte notwendig, welche hier beschrieben werden. In der folgenden Auflistung werden alle notwendigen Schritte beschrieben:

1. **Programmieren**
2. **Kompilieren**
3. **UC-Bridge starten**
4. **Programm starten**

Diesen langwierigen Prozess konnte vereinfacht werden, indem eine Launchfile erstellt wurde. Dieses führt diese Schritte hintereinander aus, sodass nach einer Codeänderung nur noch ein einzelnes Launchfile ausgeführen werden muss.

3 Wall-Follower

Im Folgenden wird die Umsetzung des Wall-Followers beschrieben. Dabei wird zuerst die Aufgabenstellung und anschließend die Implementierung beschrieben. Am Schluss werden noch die Probleme dargestellt, die während der Implementierung aufgetreten sind und wie diese Probleme gelöst wurden.

3.1 Aufgabenstellung

Wir beginnen mit dem Wall-Follower, da dies unsere Hauptaufgabe war und unsere anderen beiden Aufgaben den Wall-Follower als Grundlage nutzen. Das Fahrzeug soll in der Lage sein, einer Wand zu folgen. Es sollte also im Optimalfall immer einen festen Abstand zur Wand haben und diesen auch während der Fahrt halten. Weiter soll das Fahrzeug in der Lage sein, um eine Ecke zu fahren. Es muss also überlegt werden, wie eine Kurve erkannt und mit möglichst kleinem Radius umfahren werden kann.

3.2 Implementierung

Grundsätzlich gibt es zwei Möglichkeiten, einen Wall-Follower zu implementieren. Die *sparsame, aber ungenauere* erste Möglichkeit ist es, sich ausschließlich auf die Abstandssensoren im Fahrzeug zu verlassen. Die *rechenintensivere, aber genauere* zweite Möglichkeit ist die zusätzliche Unterstützung durch die Kinect-Kamera.

Wir haben uns für die erste Methode entschieden, da diese viel sparsamer mit den Ressourcen umgeht, welche eventuell an anderer Stelle benötigt werden könnten. Dabei werden grundlegend zwei Modi verwendet, welche in Grafik 3.1 zu sehen sind. Im Folgenden werden diese beiden Modi und deren Implementierung näher beschrieben. Auch wird darauf eingegangen, wie zwischen den beiden Modi gewechselt wird.

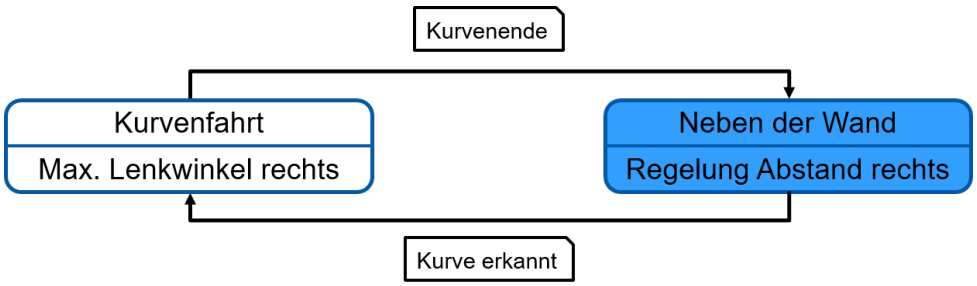

Abbildung 3.1: Entscheidungsvorgang zwischen Wand-Modus und Kurven-Modus.

3.2.1 PD-Regler

Bei dem PD-Regler handelt es sich um einen Regler, welcher Differenzen von gemessenen Abständen zu vorgegebenen Werten berechnet und versucht, diese Differenzen so gering wie möglich zu halten, indem der Lenkwinkel des Fahrzeugs beeinflusst wird. Außerdem ist der Lenkwinkel des Fahrzeugs ist abhängig von der Änderung dieser Differenz. Zur Berechnung des Lenkwinkels wurde folgende Formel genutzt:

$$s_{out} = -(pk * err + pd * \frac{err - lastErr}{current\_t - last\_t})$$

In dieser Formel bezeichnet s_{out} den Lenkwinkel, welcher maximal zwischen −1000 und +1000 liegen kann. Die mechanischen Komponenten erlauben aber nur die Maximalwinkel −800 beziehungsweise +800. Ein Lenkwert von 0 bedeutet geradeaus zu fahren, negative Werte stehen für die Lenkung nach rechts und positive Werte für die Lenkung nach links. Bei dem Wall-Follower genügt es, ihn nur für eine Seite zu implementieren. Wir haben uns dafür entschieden, nur die rechte Wand zu betrachten, es kann also auch nur Kurven nach rechts geben. Weiter bezeichnet err den aktuell gemessenen Fehler zwischen Abstandsvorgabe und tatsächlichem Abstand zur Wand. In $lastErr$ wird der Fehler aus dem letzten Schleifendurchlauf gespeichert. Der Fehler wird dann noch mittels $current\_t$ und $last\_t$ auf die Zeit genormt.

Das wichtigste in der Formel sind die beiden Parameter pk und pd. Diese erlauben es dem Anwender, die Gewichtung eines Fehlers anzupassen. Mit pk setzt man die Gewichtung des aktuellen Fehlers und mit pd die Gewichtung der Differenz zwischen aktuellem und letztem Fehler genormt auf die Zeit. Je nachdem wie groß die Parameter gewählt werden, reagiert die Lenkung stärker oder schwächer auf eine Abweichung des Abstandes. Durch Testläufe wurden folgende Parameter ermittelt:

$$pk = 800$$
$$pd = 80$$

Um Unfälle oder Beschädigungen an der Hardware zu vermeiden, wird außerdem in jedem Schleifendurchlauf nach Berechnung der Lenkwerte überprüft, ob diese die Minimal- beziehungsweise Maximalwerte von −800 und +800 unter- oder überschreiten. Sollte dies der Fall sein, wird der Wert auf den maximal erlaubten Wert gesetzt.

Damit das Fahrzeug nicht frontal gegen die Wand fährt, wird in jedem Schleifendurchlauf der Frontsensor überprüft. Sollte dieser einen vorgegeben Wert unterschreiten, wird der Motorwert auf 0 gesetzt, um das Fahrzeug zu stoppen. Genauso wird auch mit den linken und rechten Abstandssensoren umgegangen.

3.2.2 Kurvenmodus

Der PD-Regler, wie in Abschnitt 3.2.1 beschrieben, funktioniert einwandfrei, solange sich das Fahrzeug neben einer Wand befindet. Sobald aber eine Rechtskurve kommt, gibt es Probleme. Der berechnete Lenkwinkel erreicht durch die Formel nicht immer den Maximalwert, was zu einem zu großen Kurvenradius führt. Die engeren Kurven der Teststrecke können so nicht erfolgreich umfahren werden.

Dies führte zu der Überlegung, zwei Zustände einzuführen, in denen sich das Fahrzeug befinden kann. Der erste Zustand ist „Neben der Wand“ und der zweite Zustand „Kurvenfahrt“. Befindet sich das Fahrzeug im Zustand „Neben der Wand“, so wird der PD-Regler verwendet, wie in Abschnitt 3.2.1 beschrieben. Implementiert werden musste nun der zweite Zustand „Kurvenfahrt“.

Die Idee ist es, den Beginn einer Kurve mithilfe des Abstandssensors zu erkennen und dann den Lenkwinkel direkt auf den Maximalwert −800 zu setzen, damit die Kurve mit minimalem Radius durchfahren wird. Dieser Maximalwinkel soll dann für einen 90-Grad Winkel lang eingehalten werden. Nach befahren der 90 Grad soll wieder automatisch auf den PD-Regler umgeschaltet werden, da dann die Kurve zu Ende ist und sich das Fahrzeug wieder normal neben einer Wand befindet.

Sobald der Wert des rechten Ultraschallsensors einen Schwellwert überschreitet, wird vom PD-Regler in den Kurvenmodus gewechselt. Das Fahrzeug fährt nun so lange mit maximalem Lenkwinkel, bis eine Drehung von 90 Grad erreicht wurde. Zum Messen der Drehung wird das Gyroskop des Fahrzeugs, speziell der dort vorhandenen Yaw-Wert, verwendet. Dieser gibt einen Zahlenwert zwischen $-\pi$ und $+\pi$ zurück, deckt also die vollen 360 Grad ab. Bei Kurveneintritt wird der aktuellen Yaw-Wert gespeichert und in jedem Schleifendurchlauf die Differenz zwischen dem aktuellen Yaw-Wert und dem gespeicherten berechnet, als in den Kurvenmodus gewechselt wurde. Überschreitet die Differenz nun einen Wert von $\frac{\pi}{2} = 1.57$, hat sich das Fahrzeug um 90 Grad gedreht hat. Die Kurve ist dann komplett durchfahren und es kann wieder auf den PD-Regler umgeschaltet werden. Um Berechnungszeit und der Latenz der Sensoren entgegen zu kommen, wurde der Wert 1.57 auf 1.4 reduziert. Durch Testfahrten hat sich bestätigt, dass dieser Wert zu guten Ergebnissen führt.

Mit diesem Modell können wir nun verlässlich sowohl der Wand folgen, als auch mit einem minimalen Radius um die Kurve fahren und das ganz ohne Bildverarbeitungstechnik.

3.3 Evaluation und Probleme

Probleme bei der Implementierung hatten wir anfangs mit den Sensorwerten. Wir zeichneten diese auf und stellten fest, dass es vereinzelt sehr hohe Peaks auftraten, die keinen Sinn ergaben, da sich das Fahrzeug immer neben der Wand befand. Wir überlegten diese Werte herauszufiltern, um ein einheitliches Bild zu erhalten, haben uns dann aber dagegen entschieden, da sich das kontraproduktiv auf die Kurvenerkennung hätte auswirken können. Durch Testfahrten stellten wir außerdem fest, dass sich die fehlerhaften Messungen kaum auf das Fahrverhalten auswirken, da diese nur in einer sehr kurzen Zeit aktiv sind und danach dann wieder für eine lange Zeit korrekte Werte geliefert werden.

Schwierigkeiten hatten wir auch, den Yaw-Wert zu erhalten. Dieser ist nicht direkt abzulesen. Nach langer Recherche haben wir herausgefunden, dass dieser mit Hilfe des Odometry-Paket empfangen werden muss. Der Wert muss dann in ein Quarternion umgewandelt werden, woraus dann eine 3x3 Matrix erstellt werden muss. Mit dieser Matrix hat man nun die Möglichkeit, mittels $mat.getEulerYPR(yaw, pitch, roll)$ Funktion, den Yaw-Wert zu berechnen und zu verwenden.

Das letzte Problem befasst mit dem Überlauf des Yaw-Wertes. Wir haben nach mehreren Testfahrten festgestellt, dass nicht immer vom Kurvenmodus in den PD-Regler zurückgeschaltet wurde, was an einem Überlauf des Yaw-Wertes lag. Unterschreitet man den Wert $-\pi$, geht es direkt bei $+\pi$ weiter, wo der Wert dann weiter abnimmt. Überschreitet man den Wert $+\pi$, geht es direkt bei $-\pi$ weiter, wo der Wert dann weiter zunimmt. Das gleiche Schema findet man im Einheitskreis. Diesen Überlauf fingen wir ab, indem bei Kurveneintritt überprüft wird, ob mit den dazugerechneten Wert $\frac{\pi}{2}$ ein Überlauf stattfinden wird oder nicht. Wenn einer stattfinden wird, wird die Differenz, mit der geprüft wird, ob 90 Grad gefahren wurden, entsprechend angepasst. Damit kann das Fahrzeug aus jeder Ursprungslage zurück in den PD-Regler Modus wechseln.

4 Spurhaltung

In diesem Kapitel wird unsere zweite Aufgabe im Laufe des Projektseminars beschrieben, nämlich die Spurhaltung. Ziel dieser ist es, dass das Fahrzeug mithilfe einer Kamera den Bereich vor sich aufnimmt, die dortigen Spurmakierungen erkennt und sich beim Fahren nach diesen richtet. Dies ist ein erster Schritt in Richtung autonomes Fahren.

Zur Bewältigung dieser Aufgabe wird einerseits eine Bildverarbeitung zur Positionsbestimmung und andererseits eine Regelung zur Steuerung der Lenkung und der Geschwindigkeit benötigt. Wichtig hierbei ist, dass beide Komponenten in gleichem Maße zuverlässig funktionieren müssen um ein robustes Gesamtsystem zu erhalten. Im Folgenden wollen wir die eben genannten Komponenten, sowie die zur Implementierung genutzten Bibliotheken, genauer beschreiben. Anschließend möchten wir unsere Testergebnisse vorstellen, sowie einen Überblick über mögliche zukünftige Arbeiten geben.

4.1 Bildverarbeitung

Die Aufgabe der Bildverarbeitung ist es das Fahrzeug auf Basis des Kamerabildes relativ zur Fahrspur zu lokalisieren. Dies geschieht in zwei Schritten, der Spurerkennung und der Lokalisierung relativ zur erkannten Spur. Weiterhin wurde hier zur Verbesserung des Verhaltens ein Kalmanfilter eingesetzt. Der folgende Abschnitt beschreibt die jeweils verwendete Lösungsstrategie.

4.1.1 Spurerkennung

Zur Erkennung der Fahrspur im Kamerabild wurde ein Schwellenwertverfahren basierend auf dem Bild-Gradienten implementiert, dessen Ablauf in Abbildung 4.1 dargestellt ist und im folgenden beschrieben wird.

Durch die Beschaffenheit der Spurmakierungen erschien es plausibel, dass sich diese durch hohe Kontrastunterschiede zu ihrer Umgebung auszeichnen. Diese Intuition hat sich in der Praxis als eine hinreichend gute Methodik erwiesen und wurde hier durch eine Schwellwertbildung des Bild-Gradienten realisiert (Abbildung 4.1, Schritt 2/3). Das so erhaltene Binärbild sollte idealerweise nur noch an den Stellen der Spurmakierung eins und sonst null sein. Das Binärbild wird anschließend durch eine perspektivische Transformation in ein Bild aus der Vogelperspektive transformiert (Abbildung 4.1, Schritt 4). Dieses erleichtert die weitere Verarbeitung, da hier nun parallele Linien nicht mehr zusammen laufen, wie es in der Perspektive der Kamera der Fall ist. Die zur Transformation verwendete Matrix ist in Gleichung 1 dargestellt.

$$m = \begin{pmatrix} 500 & -118.833504 & -20022.0859 \\ 0 & 2.70307922 & 34148.5703 \\ 0 & -0.990279198 & 353.982605 \end{pmatrix} \tag{1}$$

Der letzte Schritt zur Erkennung der Spur ist die Anpassung des Spurmodells auf das transformierte Binärbild. Als Spurmodell wurde hier ein Polynom zweiten Grades ge-

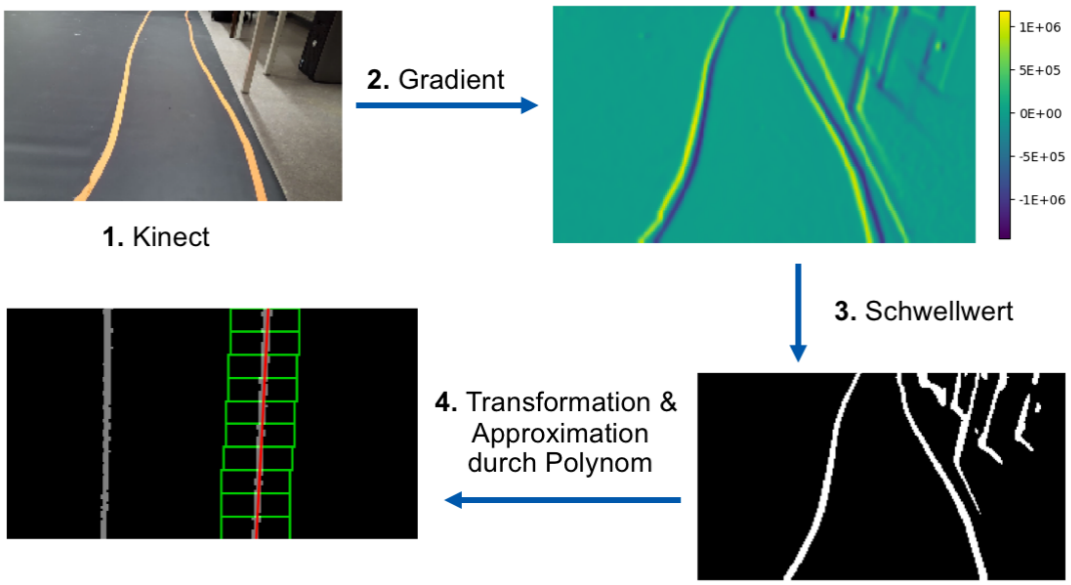

Abbildung 4.1: Gradientenberechnung, Schwellwertermittlung, Transformation und Approximation der Lane durch dein Polynom.

wählt, da sich damit einfache Kurven leicht repräsentieren lassen. Zu diesem Zeitpunkt haben wir uns auf die Erkennung der rechten Spur beschränkt.

Die Anpassung der Parameter dieses Polynoms geschieht nun in zwei Schritten. Zunächst wird mit dem in [Sun17] implementierten Verfahren eine Menge an Punkten auf der rechten Spur extrahiert. Anschließend werden die Parameter des Polynoms so optimiert, dass die Summe der quadratischen Abstände des resultierenden Polynoms zu diesen Punkten, minimiert wird. Das so erhaltene Polynom repräsentiert nun die rechte Spur.

4.1.2 Lokalisierung

Die Lokalisierung des Fahrzeuges zu dieser Spur ist in Abbildung 4.2 dargestellt. Wie man dort erkennen kann, erhält man die folgende Größen, die die Position des Fahrzeugs in der Spur beschreiben:

- x: Lateraler Abstand des Fahrzeugmittelpunktes zum Spurmittelpunkt.
- ϕ: Orientierung des Fahrzeuges relativ zur erkannten Spur.

4.1.3 Kalmanfilter

Die so erhaltene Lokalisierung ist für die weitere Verarbeitung noch zu stark verrauscht. Desweiteren kann in manchen Situationen gar keine Spur detektiert werden. Um dieses Verhalten zu verbessern wurde hier ein Kalman-Filter auf Basis des Ackermann-Modells entworfen.

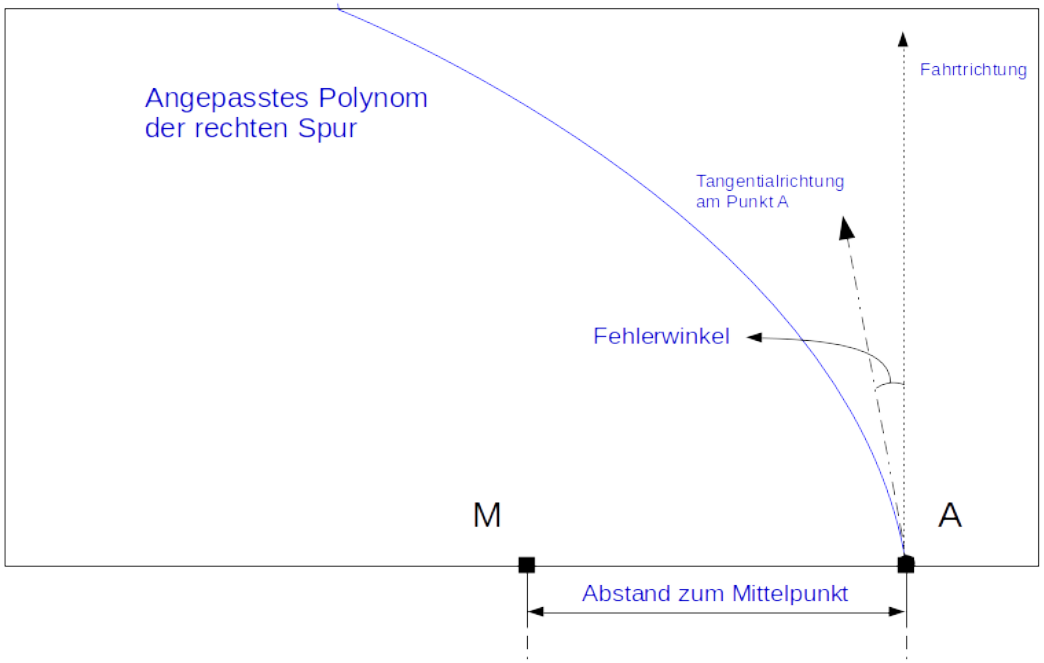

Abbildung 4.2: Darstellung des Abstands vom Anfangspunkt der Lane zum Mittelpunkt des Fahrzeuges und der Tangentialrichtung der Lane.

Dabei wurden die Kovarianzmatrizen des Mess- und Prozessrauschens experimentell bestimmt:

$$Q = \begin{bmatrix} 500 & 0 \\ 0 & 0,5 \end{bmatrix} \quad R = \begin{bmatrix} 100 & 0 \\ 0 & 0,0141 \end{bmatrix}$$

4.2 Regelung

Zur Regelung müssen der Lenkwinkel und die Geschwindigkeit des Fahrzeuges vorgegeben werden. Den Lenkwinkel wird mittels Stanley-Reglers [TMD+06], welcher folgende Formel nutzt, ermittelt:

$$\delta(t) = \phi(t) + arctan(\frac{kx(t)}{u(t)}) \tag{2}$$

Hier ist $\delta(t)$ der Lenkwinkel, $x(t)$ und $\phi(t)$ die aus der Bildverarbeitung gewonnene Position des Fahrzeugs, $u(t)$ die Geschwindigkeit des Fahrzeugs und k ein Verstärkungsparameter der hier experimentell zu $0,14$ bestimmt wurde. Zur Umrechnung des hier als Winkel bestimmten Lenkwinkels in das an das Fahrzeug übergebene "Lenkungs-Level"wurde ein proportionaler Zusammenhang, welcher über die Konstante $1600/\pi$ abgebildet wurde, angenommen.

Die Geschwindigkeit des Fahrzeuges wird proportional zur Ausrichtung des Fahrzeugs relativ zur Spur berechnet. Das bedeutet, je größer der Unterschied der Fahrzeugrichtung zur Spurrichtung ist, desto langsamer fährt es. Grund dafür ist, dass das Fahrzeug durch das langsamere Fahren mehr Rechenzeit für eine bestimmte Strecke bekommt und somit auch den Lenkwinkel und die Geschwindigkeit des Fahrzeuges genauer berechnen kann um nicht über die Spur hinaus zu fahren.

4.3 Implementierung

Zur Implementierung wurde die Programmiersprache *Python* anstelle von *C++* genutzt, wobei für die Bildverarbeitung verschieden Funktionen aus der *OpenCV* Bibliothek, wie z.B. *sobel()* oder *warpperspective()*, verwendet wurden. Auch für den Kalman-Filter wurde eine Implementierung aus *OpenCV* genutzt. Für die Handhabung der Bilder wurde die Python-Bibliothek *numpy* genutzt.

4.4 Evaluation und zukünftige Arbeit

Als Ergebnis der Implementierung haben wir nun ein Fahrzeug, welches verlässlich die Spur einhalten kann und dabei seinen Lenkwinkel und seine Geschwindigkeit mit Hilfe von Bildern aus der Kamera ermitteln kann. Bei der Demonstration ließen wir das Fahrzeug ohne Probleme für ein Paar Runden fahren.

Die Teststrecke haben wir uns selbst aufgebaut. Dafür haben wir aus verschiedenen Teilen eines Tanzbodenbelags eine große Fläche zusammengebaut. Auf diese Fläche haben wir mit einem roten Tape die Fahrspuren markiert. Der Abstand zwischen der linken und der rechten Fahrbahnbegrenzung beträgt in unserer Demonstration in etwa 40 cm. Zum Start wird das Fahrzeug innerhalb der Begrenzungen positioniert und das Programm mittels selbst erstelltem Launch-File gestartet. Danach sollte das Fahrzeug relativ sicher seine Runden drehen. Zur Demonstration haben wir einen Rundkurs genutzt, natürlich kann man aber aus dem Rundkurs auch eine abstraktere Bahn bauen, welche verschiedene Kurven besitzt. Dabei sollte man aber beachten, den Kurvenradius nicht zu klein zu wählen, da das Fahrzeug aufgrund des eingeschränkten Lenkwinkels zu enge Kurven nicht fahren kann und dann unweigerlich aus der Fahrspur kommt.

Für zukünftige Arbeiten kann vor allem noch die Bildverarbeitung verbessert werden, was z.B. auf der Basis eines Machine-Learning Ansatzes erfolgen könnte. Weiterhin ist es naheliegend, dass die Regelung bzw. Lokalisierung verbessert werden könnte, wenn man das linearisierte Ackermann-Modell durch ein anderes ersetzt, da hier besonders bei großen Lenkwinkeln Fehler gemacht werden.

5 Rundkurs mit Hindernissen

Dieses Kapitel beinhaltet die dritte Aufgabe, nämlich den Rundkurs mit Hindernissen. Wie in den vorherigen Kapiteln wird dabei zuerst die Aufgabenstellung vorgestellt und das Aufgabenziel definiert. Anschließend wird der erarbeitete Lösungsansatz und die verschiedenen Schritte der Implementierung erläutert. Am Ende des Kapitels werden Probleme, welche bei der Implementierung aufgetreten sind dargestellt und eine Evaluierung der Implementierung durchgeführt.

5.1 Aufgabenstellung

Zur weiteren Unterstützung zum autonomen Fahren benötigt ein Fahrzeug die Fähigkeit, Hindernisse zu erkennen und ihnen Auszuweichen, um eine Kollision zu verhindern. Diese Fähigkeit soll in dieser Aufgabe implementiert werden. Das Ziel dieser Aufgabe ist es, Hindernisse zu erkennen, zu lokalisieren und ihnen anschließend auszuweichen. Um dies zu bewerkstelligen, muss das Fahrzeug Abstände zu Objekten oder allgemein der Umgebung messen können, woraus dann die Dimensionen von Hindernissen abgeleitet werden können.

5.2 Implementierung

Der folgende Abschnitt beschreibt die erarbeitete Vorgehensweise zur Implementierung dieser Aufgabe. Dabei erfolgt eine Unterteilung in die verschiedenen Teilaufgaben Hinderniserkennung, Hindernislokalisierung und Pfadplanung sowie die Pfadregelung.

5.2.1 Hinderniserkennung

Wir haben uns dafür entschieden selbst ein Verfahren zum Erkennen von Hindernissen zu implementieren. Dabei wird das Tiefenbild der Kinect-Kamera verwendet. Unser erster Ansatz, zur Hindernisserkennung basiert auf dem Verfahren der Background-Subtraction [Elg15]. Dabei wurde angenommen, dass der Unterschied zwischen dem aktuellen Tiefenbild und einem Referenzbild den Hindernissen entspricht. Diese Annahme war jedoch nicht richtig, da sich die Kamera zu stark bewegt und somit der Winkel der Kamera zum Boden nicht als statisch angenommen werden kann. Das hat zur Folge, dass die Differenzen zum Referenzbild in einigen Bildbereichen stark variieren und hier eine sehr hohe Gefahr für fehlerhafte Detektionen besteht, d.h. es werden Hindernisse erkannt, die nicht existieren.

Das Vorgehen wurde daher wie folgt geändert. Abbildung 5.1 zeigt die verschiedenen Schritte der Bildverarbeitung für die Hinderniserkennung. Das aktuelle Tiefenbild wird nun vom vorherigen Tiefenbild subtrahiert, um Abstandveränderungen relativ zum Fahrzeug zu messen. Danach werden fehlerhafte Pixel aus dem Tiefenbild gefiltert, um Störungen zu minimieren. Mit Hilfe einer Maskierung werden nur noch die relevanten Bildbereiche betrachtet. Ränder des Bildes sind tendenziell stark fehlerbehaftet und werden daher nicht verwendet. Bereiche in der oberen rechten und linken Ecke des Bildes sind für die Hinderniserkennung ebenfalls irrelevant, da für ein Hindernis an

diesen Positionen kein Ausweichmanöver erforderlich ist.

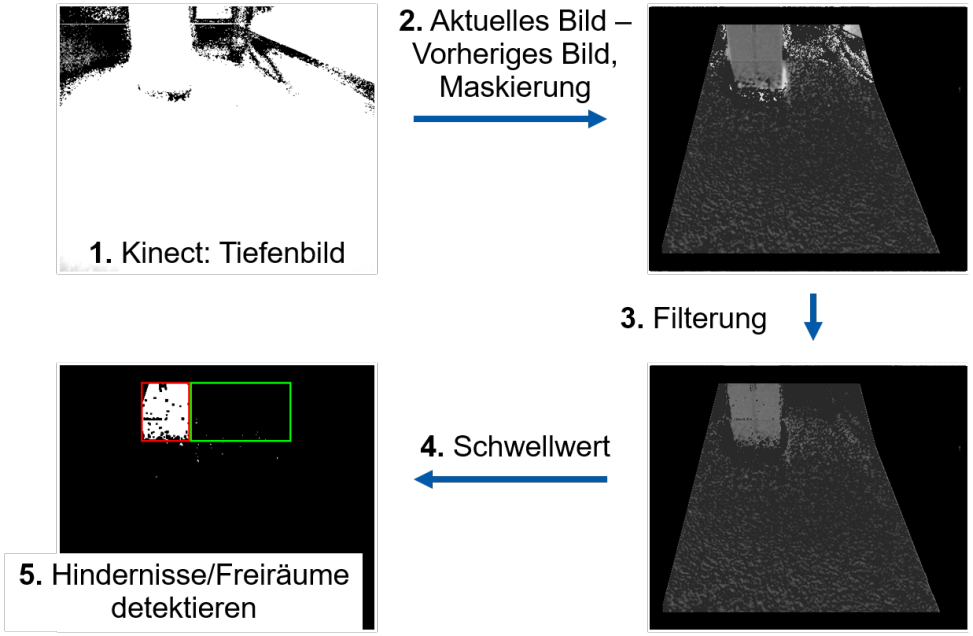

Abbildung 5.1: Schritte der Bildverarbeitung zur Hinderniserkennung.

Nach der Berechnung des Differenzbildes wird ein Binärbild erzeugt. Hierbei werden alle Pixel mit einem Wert größer oder gleich dem Schwellwert weißund alle übrigen Pixel schwarz markiert. Im nächsten Schritt wird dieses Binärbild dazu genutzt, um die Konturen der Hindernisse zu erkennen. Damit ein Hindernis als solches erkannt wird, muss es eine bestimmte Größe aufweisen. Für ein Szenario in der echten Welt müssten natürlich auch kleinere Objekte als Hindernisse betrachtet werden. Für unser Szenario war es jedoch ausreichend, nur die größeren Objekte zu betrachten, da im Hindernisparkour auch nur große Hindernisse in Form von Boxen benutzt werden. Durch die kleinere Anzahl an möglichen Hindernissen wird das Risiko von false Positives minimiert.

Im nächsten Schritt wird für jedes potentielle Hindernis die durchschnittliche Entfernung zum Fahrzeug ermittelt. Diese kann direkt aus den Pixelwerten des ursprünglichen Tiefenbildes berechnet werden. Anschließend wird nur das Hindernis mit der geringsten Entfernung zum Fahrzeug betrachtet. Ist dieses Hindernis noch zu weit entfernt, wird es auch ignoriert, da Berechnungen für einen neuen Kurs bei einem weit entfernten Hindernis ungenau sind. Rechts und links vom Hindernis werden mögliche Fahrtkorridore markiert und die Abstände zu diesen berechnet.

5.2.2 Hindernislokalisierung und Pfadplanung

Der folgende Abschnitt erläuert das Vorgehen zur Lokalisierung der Hindernisse und er daraus folgenden Pfadplanung für das Ausweichmanöver. Abbildung 5.2 zeigt die gegebenen geometrischen Beziehungen und Größen, die bei der Lokalisierung berücksichtigt werden, und die Formel zur Berechnung. Im ersten Schritt wird der Abstand des

Hindernisses zur rechten Wand ermittelt. Hierbei wird zunächst im ersten Summanden der Formel die Position der rechten Kante des Hindernisses bezüglich der Kameramitte berechnet. Anschließend wird mit Hilfe des seitlichen Abstandes des Fahrzeugs zur rechten Wand, der Fahrzeugbreite und des Abstandes der Kameramitte zur Fahrzeugmitte der Abstand des Hindernisses zur rechten Wand ermittelt. Diese Berechnung wird für die linke Kante des Hindernisses wiederholt.

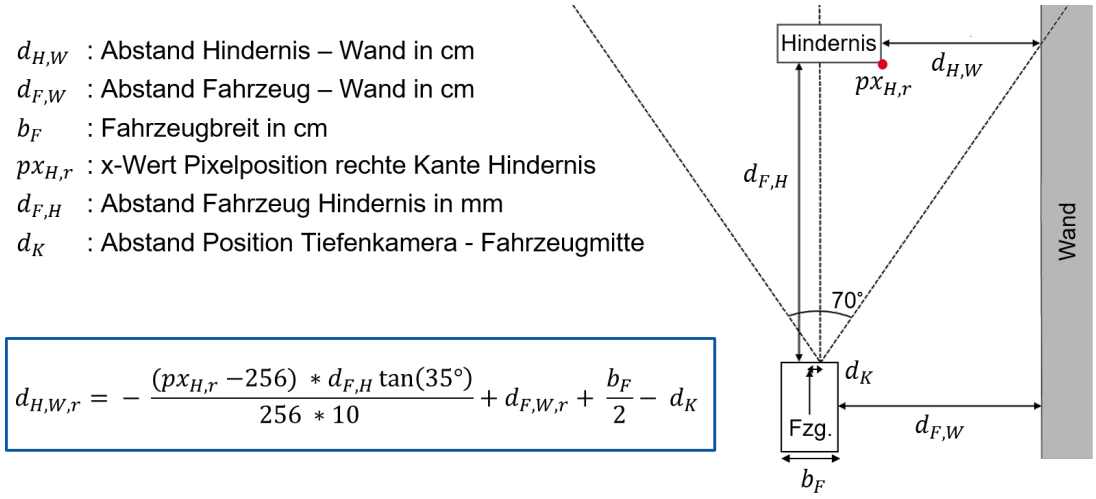

Abbildung 5.2: Lokalisierung der Hindernisse.

Im letzten Schritt werden nun die berechneten Abstände zur Planung des Ausweichpfades verwendet. Dabei gibt es folgende drei Möglichkeiten:

1) Raum rechts neben Hindernis frei und Abstand Hindernis - rechte Wand größer als Schwellwert: Ausweichen rechts

2) Raum rechts neben Hindernis nicht frei oder zu klein und Raum links neben Hindernis frei: Ausweichen links

3) Rechte und linke Seite unpassierbar: Kein Ausweichen

Bei einem Ausweichmanöver zur rechten Seite wurde der Pfad so geplant, dass das Fahrzeug mittig zwischen dem Hindernis und der rechten Wand fährt. Bei der Vorbeifahrt links an einem Hindernis, soll das Fahrzeug das Hindernis mit einem festen Abstand passieren. Für den dritten Fall, dass sowohl die rechte als auch die linke Seite des Hindernisses unpassierbar sind, wurde angenommen, dass das erkannte Hindernis eine Wand am Ende des Ganges ist. In diesem Fall soll folglich kein Ausweichmanöver eingeleitet werden.

5.2.3 Pfadregelung

Die Grundlage für die Pfadregelung dieser Aufgabe bildet der Regelalgorithmus aus der Implementierung für den Rundkurs ohne Hindernisse (PD-Regler mit Kurvenmodus)

aus Abschnitt 3. Der Zustand des PD-Reglers wurde hier jedoch um mehrere Unterzustände erweitert. Abbildung 5.3 zeigt den erweiterten Zustandsautomaten für die Pfadregelung.

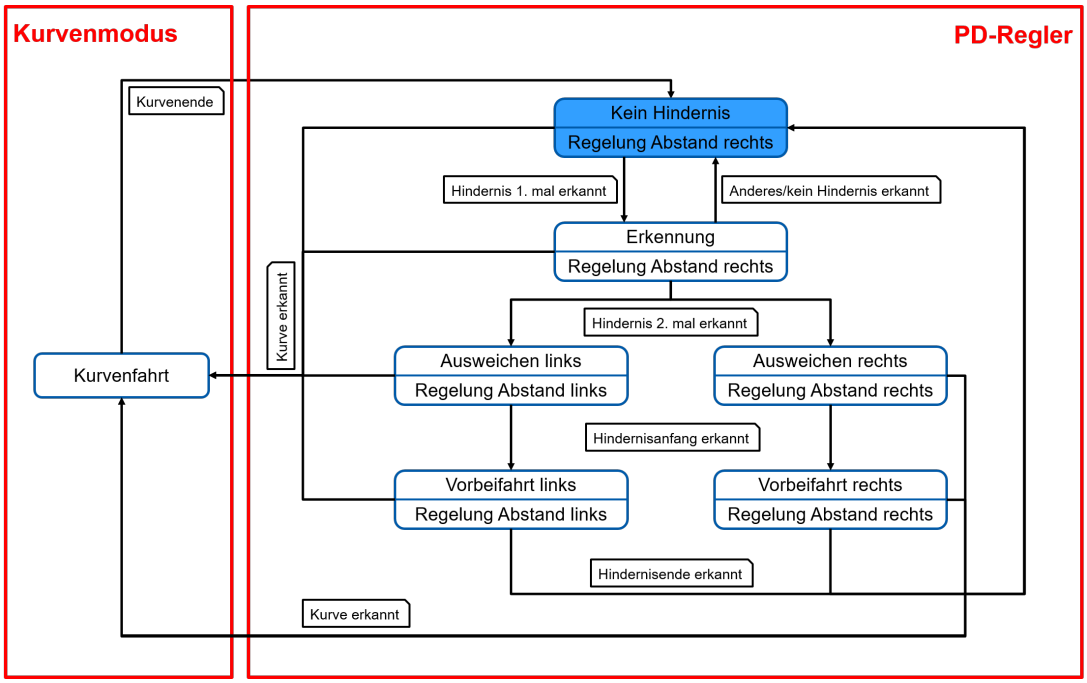

Abbildung 5.3: Vollständiger Zustandsautomat des Fahrzeugs für den Hindernisparkour.

Der Initialzustand ist der Zustand **Kein Hindernis**. Sobald ein Hindernis erkannt wird, folgt der Wechsel in den Zustand **Erkennung**. In den beiden Zuständen **Kein Hindernis** und **Erkennung** erfolgt eine Abstandsregelung zur rechten Wand. Verwendet wird dabei der bereits aus Abschnitt 3.2.1 bekannte PD-Regler. Wird in diesem Zustand das gleiche Hindernis erneut erkannt, folgt der Wechsel in einen der beiden Zustände **Ausweichen links** oder **Ausweichen rechts**. Dieser Zwischenzustand ist notwendig, um die Wahrscheinlichkeit einer fehlerhaften Detektion, die in einzelnen Bildern auftreten kann, zu verringern. Falls es sich um eine fehlerhafte Hinderniserkennung handelte, d.h. das Hindernis im nachfolgenden Bild nicht bestätigt wurde, folgt der Sprung zurück in den vorherigen Zustand **Erkennung**.

Sobald der Anfang des Hindernisse mit Hilfe der seitlichen Ultraschallsensoren detektiert wurde, erfolgt der Sprung in den Zustand **Vorbeifahrt links**, bzw. **Vorbeifahrt rechts**. Wird das Ende des Hindernisses erkannt, wechselt das Fahrzeug wieder in den Initialzustand und folgt der Wand in einem festgelegten Abstand. Aus allen vorher genannten Zuständen, wird in den Zustand **Kurvenfahrt** gewechselt, sobald eine Kurve erkannt wurde. Die Kurvenerkennung und die Kurvenfahrt erfolgen analog zum Rundkurs ohne Hindernisse. Zusätzlich zur Abstandsregelung erfolgt eine Anpassung der Geschwindigkeit abhängig vom Fahrzeugzustand. Um die Kurvenfahrt und das Auswei-

chen von Hindernissen zu verbessern, erfolgt in diesen Zuständen eine Verringerung der Fahrzeuggeschwindigkeit.

5.3 Evaluation und Probleme

Im Rahmen dieser Aufgabe wurde ermöglicht, dass das Fahrzeug Hindernisse mit Hilfe des Tiefenbildes der Kinect erkennt und deren Abmessungen bestimmt. Des weiteren werden die Räume rechts und links neben dem Hindernisse für potenzielle Ausweichmanöver analysiert. Basierend auf diesen Informationen erfolgt die Pfadplanung zum Ausweichen der Hindernisse.

Bei der Demonstration hat dies bei den kleineren Hindernissen zu Beginn des Rundkurses gut funktioniert. Schwierigkeiten hatte das Fahrzeug allerdings bei den breiteren Hindernissen. Die Gründe dafür sind zum einen die eingebaute Begrenzung des Lenkwinkels für Ausweichmanöver und der durch den Kameraöffnungswinkel eingeschreänkte Sichtbereich im Bereich nahe vor dem Fahrzeug. Die Begrenzung des Lenkwinkels verhindert starke Ausweichmanöver speziell zur linken Fahrzeugseite. Diese war aber notwendig, um den Kurswinkel des Fahrzeugs gegenüber der Wand zu begrenzen. Denn zu große Winkel zwischen Fahrzeug und Wand führten zu Fehlern bei der Hinderniserkennung und -lokalisierung. Abhilfe für das Problem des begrenzten Sichtbereichs würde hierbei eine Kamera mit einem größeren Öffnungswinkel schaffen.

6 Fazit

Als Schlusswort geben wir noch ein Fazit über das Projektseminar, sowohl über unsere persönlichen Erfahrungen als auch die Erfahrungen mit der Organisation des Seminars.

6.1 Teamarbeit

In diesem Seminar hatten wir die Möglichkeit, ein größeres Projekt in einem von Anfang an festgelegten Team zu realisieren. Dazu gehörte die eigenständige Organisation innerhalb des Teams, die Aufgabenverteilung und Kommunikation untereinander.

Diese Punkte haben bei uns sehr gut funktioniert, wodurch wir einen guten Einstieg in die Teamarbeit bekommen haben. Auch sind wir sicherer im Umgang mit Werkzeugen zur Softwareentwicklung und zur Organisation des Teams geworden. Unseren Wissensstand konnten wir insbesondere in der Softwareentwicklung mit ROS Bibliotheken in Kombination mit C++ und Matlab erweitern. Aber auch im Umgang mit dem Softwareverwaltungssystem Git und dem Aufgabenplaner Trello sind wir sicherer geworden.

Zu Beginn hatten wir für die Kommunikation im Team die App Slack benutzt, welche wir mit Trello kombiniert hatten. Allerdings war uns die Einarbeitung in Slack zu aufwendig, beziehungsweise der Mehrwert wäre geringer gewesen als der notwendige Aufwand. Die Aufgabenplanung erfolgte bei den regelmäßigen Teamtreffen und wurden dort direkt in Trello eingetragen haben. Die Kommunikation außerhalb der Uni funktionierte sehr gut mittels einer WhatsApp Gruppe.

Unserer Meinung nach ist eine App wie Slack erst bei größeren Teams sinnvoll, welche sich nicht regelmäßig persönlich treffen können. Allerdings sollte dann eine gemeinsame Einführung für alle Teammitglieder erfolgen.

6.2 Organisation

Mit der Ausstattung während des Seminars waren wir sehr zufrieden. Durch einen Transponder hatten wir jederzeit Zugang zum Lab, in welchem sich die Fahrzeuge befanden. Auch die Ausstattung des Fahrzeugs an sich war gut, kann aber noch weiter verbessert werden. Denkbar wäre die Montage von zusätzlichen Sensoren, etwa weiteren Ultraschallsensoren an den Seiten und Rückseite des Fahrzeugs. Dies würde beispielsweise den Einparkvorgang, welcher von einer anderen Gruppe implementiert wurde, und auch die Lokalisierung des Fahrzeugs (Winkel zur Wand) erheblich erleichtern.

Auch die Verwendung einer anderen Kamera mit einem größeren Blickwinkel könnte in Betrachtung gezogen werden. Zusätzlich könnte, falls möglich der Unterbau der Kamera modifiziert werden, um die Neigung der Kamera ändern und feststellen zu können. Für unsere Spur- und Hinderniserkennung mussten wir die Kamera nach unten neigen. Zwecks fehlender Befestigungsmöglichkeit haben wir mit einem Klebeband improvi-

siert, was aber dennoch zu kleinen Wacklern im Bild geführt hat.

Der Einstieg in die Software gestaltete sich etwas kompliziert. Nach etwa vier Wochen hatten wir aber einen guten Überblick über die Möglichkeiten und auch Grenzen des Systems bekommen. Eventuell kann hier mit einer zusätlichen Einführungsveranstaltung die Einarbeitungszeit verringert werden und somit den Teams mehr Zeit zur Verfügung gestellt werden, um sich mit den eigentlichen Aufgaben auseinanderzusetzen.

6.3 Zusammenfassung

Alles in Allem können wir dieses Seminar jedem Student, welcher sich für die Entwicklung eines Echtzeitsystems interessiert, empfehlen. Durch die sehr gute Betreuung, die interessante Aufgabenstellung und die selbständige Teamarbeit hat das Seminar sehr viel Spaß gemacht.

TEAM TURACERS

Literatur

[Elg15] Elgammal, Ahmed: *Background subtraction: theory and practice*. San Rafael, California : Morgan & Claypool Publishers, 2015

[Sun17] Sung, Jou-ching: *Advanced Lane Detection Using Computer Vision*. https://github.com/georgesung/advanced_lane_detection. Version: 2017. – [Online; Aufgerufen am 15.01.2018]

[TMD+06] Thrun, Sebastian ; Montemerlo, Mike ; Dahlkamp, Hendrik ; Stavens, David ; Aron, Andrei ; Diebel, James ; Fong, Philip ; Gale, John ; Halpenny, Morgan ; Hoffmann, Gabriel u. a.: Stanley: The robot that won the DARPA Grand Challenge. In: *Journal of field Robotics* 23 (2006), Nr. 9, S. 661–692