Abschlussbericht Projektseminar Echtzeitsysteme

Team Herbie
Projektseminar eingereicht von
Andreas Diefenbach, Maurice Rohr, Hikmet Baran, Darko Pavlovic, Tobias Glätzner
am 8. 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: Geza Kulscar
Betreuer: Geza Kulscar

Erklärung zum Projektseminar

Hiermit versichere ich, das vorliegende Projektseminar 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 8. April 2018

(Maurice Rohr, Andreas Diefenbach, Hikmet Baran, Darko Pavlovic, Tobias Glätzner)

Inhaltsverzeichnis

1 Einleitung

Bei der Entwicklung von Fahrzeugen sind heutzutage autonome Fahrassistenzsysteme von großer Bedeutung. Das liegt vor allem an der immer leistungsstärkeren Hardwaretechnik, welches es ermöglicht komplexe Aufgaben zuverlässig und in Echtzeit zu automatisieren.

Einen kleinen Einblick und Einstieg bietet daher das Projektseminar Echtzeitsysteme in diesem wichtigen Bereich der Kraftfahrzeugentwicklung. Dieses wird jedes Wintersemester an der TU-Darmstadt angeboten und richtet sich an Studenten der Fachrichtungen Informationssystemtechnik, Informatik und Elektro- und Informationstechnik, sowie Automatisierungstechnik und Mechatronik.

In einer 5er Gruppe werden dabei im Verlauf des Semesters mindestens 3 Themen bearbeitet und anschließend die Ergebnisse bei einer Präsentation und Live Demo präsentiert.

Bei den Themen handelt es sich zunächst um den Rundkurs ohne Hindernisse, welches von jeder Gruppe behandelt werden muss. Für die anderen Themen im Projektseminar können die Studenten kreativ sein und sich selbst Themen aussuchen, wobei sie ebenfalls auch eigene Themenvorschläge einbringen können.

Als Team Herbie haben wir uns mit folgenden Themen beschäftigt:

- Rundkurs ohne Hindernisse
- Visuelle Odometrie
- Verkehrsschilderkennung
- Rundkurs mit Hindernissen

Nach einem Überblick über die Projektorganisation, werden darauffolgend die aufgeführten Themen hinsichtlich der Zielsetzung und Implementierung beschrieben. Nach jedem Thema werden die während der Implementierung auftretenden Probleme diskutiert und ein eventueller Ausblick vorgeschlagen.

2 Projektorganisation

Da das Projekt von fünf Teammitgliedern bearbeitet wird und mehrere verschiedene Aufgabenstellungen bearbeitet werden sollen, mussten zunächst einige Formalien geklärt werden. Dies betrifft vor allem die Verwaltung der Quellcode-Dateien und das Aufbauen von Kommunikationskanälen und Planungswerkzeugen.

Code-Verwaltung

Für die strukturierte Verwaltung des Quellcodes standen mehrere Plattformen zur Debatte. Die naheliegendste Option war natürlich github. Davon entfernten wir uns jedoch recht schnell, da der recht geringe Umfang der Quellcodes die Mächtigkeit von github in unseren Augen einfach nicht erforderlich war. Zudem wurde meist direkt am Auto gearbeitet, wodurch eine Synchronisation des aktuellen Stand für den Arbeitsprozess nicht zwingend nötig war. Deshalb entschieden wir uns dafür, neben dem Auto selbst nur Dropbox zu nutzen. Dies nutzten wir als zentrale Datensicherung, nicht nur für den Programmcode, sondern auch für Vortagsfolien, Messergebnisse und ähnliches. So konnte nach Abschluss eines Arbeitspakets die Daten gesichert werden.

Kommunikation und Planung

Neben den Treffen mit dem Betreuer des Seminars, welche im zweiwöchigen Turnus anstanden, bauten wir uns eine teaminterne Kommunikationstruktur auf. Um einen guten Austausch über aktuelle Themen und Fortschritte zu gewährleisten, wurden zu Beginn des Projekt wöchentliche Treffen vereinbart, zu denen alle Teammitglieder erscheinen sollten. Dies sollte sicherstellen, dass jeder über den Fortschritt der Arbeitspakete informiert ist und gegebenenfalls Ideen und Anregungen aus der Gruppe mit in die Arbeit einfließen konnten.
Für die Planung und Dokumentation der Arbeitsschritte entschieden wir uns die Plattform Trello zu nutzen. Damit konnten wir für die einzelnen Aufgabenstellungen eine virtuelle Pinnwand einrichten, bei der wir uns Meilensteine und Deadlines setzten und unseren Fortschritt evaluieren konnten. Die nachfolgende Abbildung zeigt diese Pinnwand nach Abschluss der Implementierungsphase.

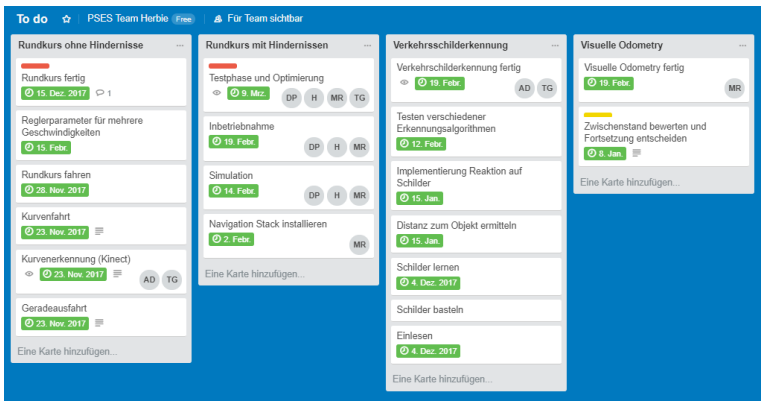

Abbildung 2.1: Trello-Board von Team Herbie

3 Rundkurs ohne Hindernisse

Diese Aufgabe stellt die für alle Gruppen verpflichtende Aufgabe dar. Es soll ein Programm entworfen werden, mit dem das Auto einen Rundkurs, bestehend aus vier 90 Grad Kurven, bewältigen kann.

3.1 Zielsetzung

Unsere individuelle Zielsetzung bei dieser Aufgabe ist es, den Rundkurs zuverlässig und möglichst schnell zu befahren. Zuverlässig soll in diesem Fall bedeuten, dass sowohl die Startposition in gewissen Grenzen irrelevant sein soll, als auch die Anzahl der gefahrenen Runden keinen Einfluss auf die Funktionalität haben soll. Zudem soll es in Hinblick auf die weiteren Teilaufgaben möglich sein, den Rundkurs mit verschiedenen Fahrgeschwindigkeit absolvieren zu können.

3.2 Implementierung

3.2.1 Reglerentwurf

Das Ackermannmodell

Um einen Reglerentwurf durchführen zu können ist zunächst ein mathematisches Modell des Fahrzeugs notwendig. Dieses soll, zum Einen, das dynamische Verhalten des Fahrzeugs möglichst realitätsnah widerspiegeln und zum Anderen eine Struktur besitzen, die nicht zu Komplex ist und somit den Reglerentwurf nicht unnötig zu erschweren. Das Ackermannmodell ist ein solches Modell. Dieses fasst die Räder der Front- bzw. der Vorderachse zu jeweils einem Rad zusammen, wodurch ein Einspurmodell entsteht. Damit die Annahme eines Ackermannmodells jedoch überhaupt stattfinden kann, muss das „Schieben"der Räder ausgeschlossen werden können [1]. Da die Beschleunigung des Fahrzeugs relativ gering ist und die Reibung zwischen dem Boden und den Rädern relativ hoch, ist die Voraussetzung für das Anwenden des Ackermannmodells offensichtlich erfüllt.

In der Abbildung 3.2.1 ist eine Skizze des Ackermannmodells mit den entsprechenden Größen dargestellt. Dabei beschreibt φ_l den Lenkwinkel, φ_k den Kurswinkel und l den Abstand zwischen Hinter- und Vorderachse. Weiterhin ist v die Geschwindigkeit an der ungelenkten Hinterachse und l_H der Referenzpunkt zur Hinterachse von dem aus der Abstand zur Wand gemessen wird. Dieser wird hier genau mittig zwischen die beiden Achsen gelegt. Die Koordinatenrichtungen lassen sich ebenfalls der Skizze entnehmen.

Nach einer Linearisierung der Modellgleichungen, sowie der Annahme eines kleinen Lenkwinkels, so dass $tan\varphi \approx \varphi$ gilt, ergibt ergibt sich nach [1] das lineare Zustandsraummodell des Fahrzeugs:

$$\begin{bmatrix} \dot{y} \\ \dot{\varphi}_k \end{bmatrix} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \varphi_k \end{bmatrix} + \begin{bmatrix} v \cdot \frac{l_H}{l} \\ \frac{v}{l} \end{bmatrix} \cdot \varphi_l$$

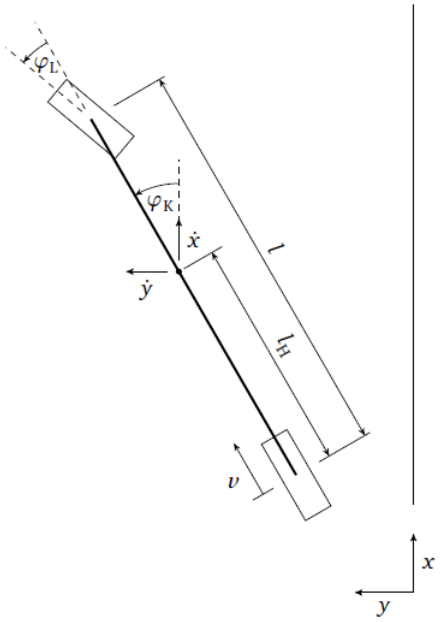

Abbildung 3.1: Graphische Darstellung des Ackermannmodells [1]

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} y \\ \varphi_k \end{bmatrix}$$

Das vorliegende Zustandsraummodell beschreibt den kinematischen Zusammenhang zwischen dem Abstand zur Wand y und des Kurswinkels φ_K bei einer konstanten Geschwindigkeit v der Hinterachse. Der Ausgang des Systems ist wie gewünscht der Abstand zur Wand, wobei der Lenkwinkel die Eingangsgröße des Systems beschreibt.

Im Weiteren wird für den Reglerentwurf von der Übertragungsfunktion des obigen Systems im Laplacebereich:

$$G_s(s) = \frac{v \cdot l_H}{l} \cdot \frac{\frac{v}{l_H + s}}{s^2}$$

ausgegangen.

Simulation und Entwurf des PD-Reglers

Anhand des doppelten Pols in Null in der Übertragungsfunktion der Strecke, lässt sich sofort auf die Instabilität der Strecke schließen. Dies lässt sich durch Simulation bestätigen. Da sich der Einsatz eines Reglers mit I-Anteil zusätzlich destabilisierend auf den späteren Regelkreis auswirkt, soll hier zunächst der Ansatz mit einem PD-Regler gemacht werden.
Mithilfe des Wurzelortskurvenverfahrens, sowie einer Simulation der Sprungantwort des PD-geregelten Systems ergibt sich für die Übertragungsfunktion des PD-Reglers:

$$G_R(s) = 0.06871 \cdot (1 + 2s). \tag{1}$$

Somit lauten die theoretischen Reglerparameter $K_P = 0.06871$ und $K_D = 0.13742$. Die Sprungantwort des geregelten Systems ist in Abbildung 3.2.1 dargestellt.

Der Sprungantwort des Regelkreises lässt sich ein Überschwingen, sowie eine rela-

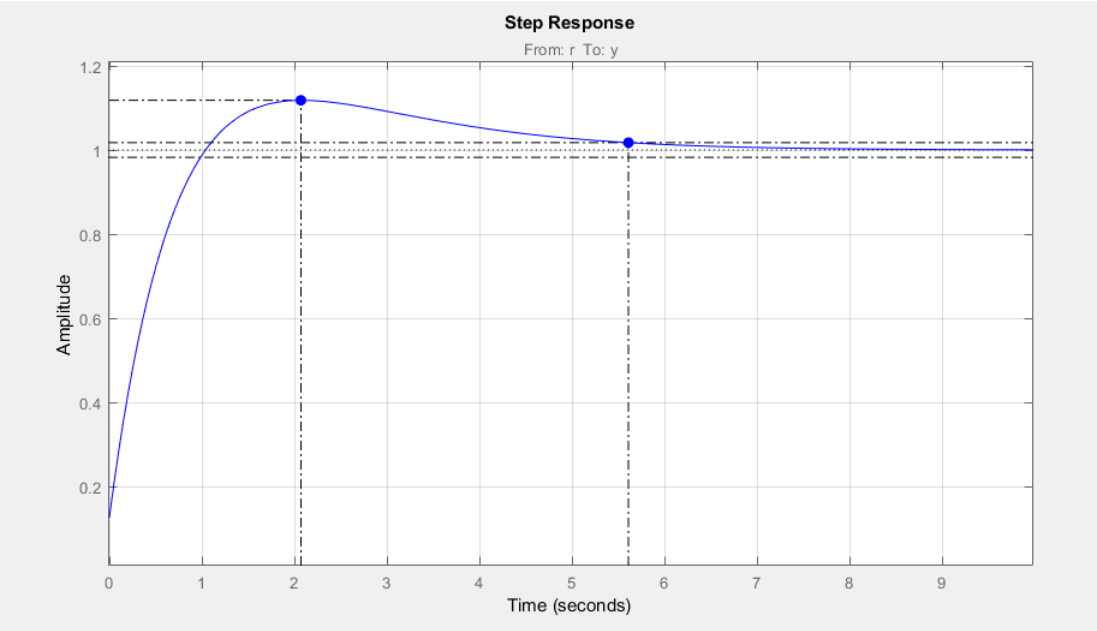

Abbildung 3.2: Sprungantwort des geregelten Systems

tiv lange Ausregelzeit von ungefähr 5, 5 Sekunden entnehmen. Das Überschwingen des Systems lässt sich durch einen PD-Regler nicht beseitigen. Dieser Umstand spielt allerdings für die vorliegende Anwendung keine große Rolle. Von Interesse ist jedoch an dieser Stelle die Ausregelzeit des Systems, die in diesem Fall verhältnismäßig lang ist. Dieser Umstand lässt sich mit der relativ gering zur Verfügung stehenden Stellgröße begründen. Eine weitere Verschiebung der Pole des Regelkreises nach links, mit dem Ziel die Ausregelzeit zu verkürzen, würde zum Einen die Stellgrößenbeschränkung verletzen und zum Anderen zum Verlassen des Linearitätsbereiches des Systems führen. Im zweiten Fall wäre die Instabilität des Regelkreises die Folge, was sich auch experimentell bestätigt hat.

Bei der Implementierung des Reglers muss berücksichtigt werden, dass beim Entwurf im Laplacebereich gearbeitet wurde, und der vorliegende Regler somit zunächst für ein zeitkontinuierliches System ausgelegt wurde. Wird jedoch eine hinreichend kleine Abtastzeit angenommen, so liefert das oben beschriebene Verfahren eine gute Näherungslösung für den zeitdiskreten Fall.
Weiterhin müssen die durch die Simulation ermittelten Reglerparameter auf die entsprechenden Integerwerte für die Programmierung umgerechnet werden. Dabei muss sowohl die Abtastzeit von 25 Hertz als auch die Kennlinie der Lenkung berücksichtigt werden. Da experimentell ein weitgehend lineares Verhalten der Lenkung festgestellt wurde, insbesondere für kleine Lenkwinkel φ_l, ergeben sich die Werte der Parameter für den digitalen PD-Regler:

$$K_{P,digital} = 160$$

$$K_{D,digital} = 8000$$

Die Stabilität des hier beschriebenen Reglers konnte - wenn auch für keine hohen Geschwindigkeiten - experimentell bestätigt werden. Nichtsdestotrotz liegt das theoretische Ergebnis relativ gut in der Größenordnung des mittels Trial-and-Error optimierten PD-Reglers ($K_P = 275$ und $K_D = 12000$), welcher auch für hohe Geschwindigkeiten die Regelstrecke sehr zuverlässig regelt. Weiterhin hat sich die destabilisierende Wirkung des I-Anteils im Regler auch experimentell bestätigt.

An dieser Stelle sei lediglich noch angemerkt, dass ein beobachterbasierter Zustandsreglerentwurf eine sehr interessante Alternative zu dem herkömmlichen PD-Regler darstellt. Insbesondere für den Fall einer verrauschten Messung des Kurswinkel φ_k, wie es in dieser Anwendung der Fall ist, bietet sich der Einsatz eines Kalmanfilters an. Durch Simulation konnte mit einem Zustandsregler, für den Fall unverrauschter Zustandsgrößen, insgesamt ein gutes Regelergebnis, ohne Überschwingen, erzielt werden.

3.2.2 Arbeitspakete

Die Problemstellung erlaubt unterteilbare Arbeitspakete, in denen die einzelnen Aspekte gelöst werden und anschließend miteinander verknüpft werden können.

Geradeausfahrt

Das Befahren der geraden Strecken des Rundkurses stellt einen zentralen Aspekt der Aufgabenstellung dar. Dabei bietet es sich an die Beschaffenheit der Umgebung auszunutzen. Mit Hilfe der Ultraschallsensoren an den Seiten des Modellautos soll der Abstand zur kurveninneren Wand konstant gehalten werden. Dies sorgt für eine stabile Geradeausfahrt.

Kurvenfahrt

Bei der Kurvenfahrt kann durch die Gegebenheiten des Kurses ebenfalls eine Vereinfachung vorgenommen werden. Da es keine Variation in der Kurvengestalt gibt und auch die Richtung immer die gleiche ist, kann jedes Mal durch eine Lenkwinkelvorgabe und die Information über den aktuellen Drehwinkel eine Drehung um 90 Grad realisiert werden.

Kurvenerkennung

Über die bereits genutzten Ultraschallsensoren kann keine rechtzeitige Erkennung einer Kurve umgesetzt werden. Daher muss hier auf die Kamera zurückgegriffen werden. Dafür kann das Tiefenbild genutzt werden, mit dem eine Ecke erkannt werden kann.

3.2.3 Programmstruktur

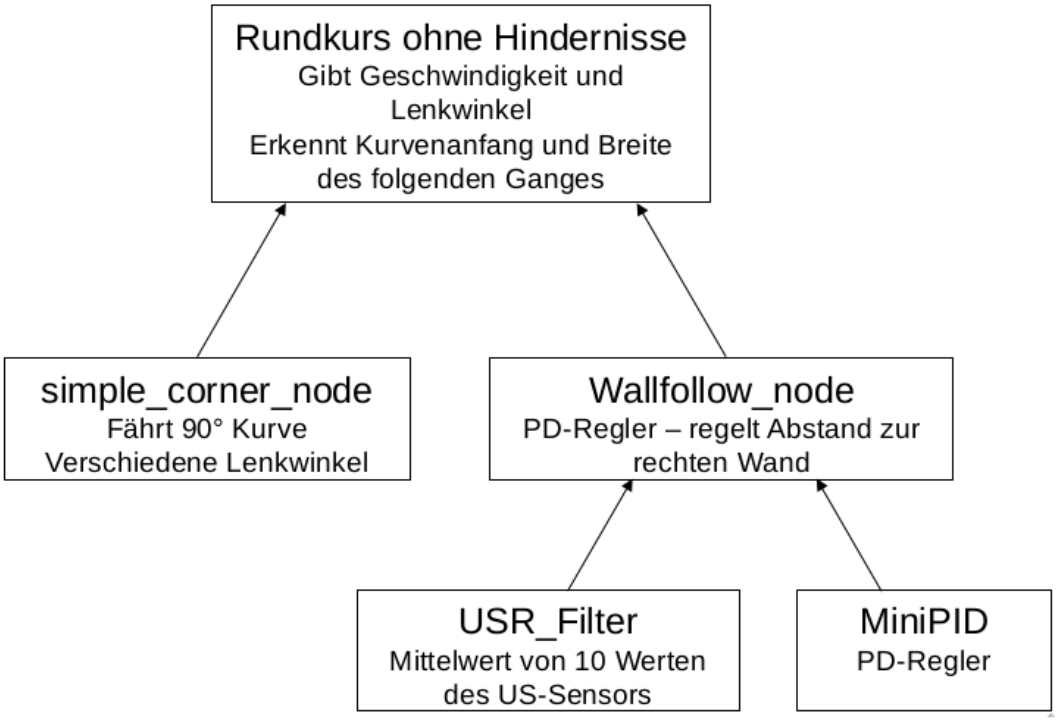

Abbildung 3.3: Programmstruktur Rundkurs ohne Hindernisse

USR_Filter

Da der Ultraschallsensor hin und wieder Werte leifert, die durch Messfehler unbrauchbar sind, werden die Werte mit einem einfachen Mittelwertfilter geglättet. Dieser berechnet aus den letzten 10 Sensorwerten den Mittelwert. So werden kritische Messfehler kompensiert, sodass die Regelung wesentlich besser funktioniert.

wallfollow_node

Das Halten eines konstanten Abstands zur Wand wird mit Hilfe eines PD-Reglers realisiert, welcher den vom Ultraschallsensor geliferten Wert als Regelgröße nutzt. Die Parameter für P- und D-Anteil wurden experimentell bestimmt. Diese Node gibt den errechneten Lenkwinkel an seinen Publisher weiter.

simple_corner_node

Diese Node realisiert die Kurvenfahrt. Zu Beginn wird mit Hilfe der Daten aus der Odometrie die aktuelle Orientierung bestimmt. Dann wird ein Lenkwinkel vorgegeben, solange bis die derzeitige Orientierung um 90 Grad zugenommen hat. Ist die Drehung abgeschlossen, sendet die Node ein finish-Signal. Um eine gewisse Robustheit gegenüber von Messungenauigkeiten der Sensoren zu erreichen, liegt der Schwellwert in dieser Implementierung bei 86 Grad. Zur bessern Kurvenfahrt wird die Kurve zusätz-

lich in zwei Teile geteilt - in einen Teil mit normalen Lenkwinkel und in einen Teil mit großem Lenkwinkel. Dadurch ergibt sich ein stabilerer Übergang zur Geradeausfahrt am Ausgang der Kurve.

rundkurs_node

Diese Node ist das Herzstück des Programms. Sie führt alle Informationen aus den übrigen Nodes zusammen und beinhaltet die Kurvenerkennung. Die Kurvenerkennung erfolgt über die Daten des depthimage-to-laserscan Pakets, welches aus dem Tiefenbild ein Array von Entfernungswerten zur Verfügung stellt. Durch den konstanten Abstand zur Wand während der Geradeausfahrt überschreitet die äußeren Werte des Arrays nie einen kritischen Wert. Gelangt das Auto nun an eine Ecke, wird dieser Abstand schlagartig größer. Dieser Effekt wird als Erkennung für eine nahende Kurve genutzt. Dazu betrachten wir die ersten 11 Werte des Arrays, sind alle Werte größer als 2.9 Meter wird eine Kurve erkannt. Um an den zwei Glaswänden vorbeifahren zu können brauchen wir 11 Werte des Arrays, eine Glaswand macht etwa 6 Werte des Arrays.
Nun kommt der Automat zum Einsatz, welcher das Umschalten zwischen Geradeausfahrt und Kurvenfahrt übernimmt. Wird eine Kurve erkannt, so wird ein start-Signal gesetzt, welches die Berechnungen simple_corner_node startet und die Geschwindigkeits- und Lenkwerte dieser Node an die UC-Bridge weitergeleitet. Meldet die Kurvenfahrt das finish-Signal, scahaltet der Automat auf Geradeausfahrt um. Das bedeutet, dass Geschwindigkeit und Lenkung wieder von wallfollow_node an das Fahrwerk weitergeleitet werden. Nach diesem Umschalten wird wegen der verwendeten Methode zur Kurvenerkennung das erneute Erkennen einer Kurven um 0.7 Sekunden deaktiviert, da das versehentliche Erkennen einer Kurven direkt am Kurvenausgang in der Praxis oft Probleme bereitet hat.

3.3 Probleme

Unsere Kurvenerkennung erkennt eine Kurve zuverlässig und früh genug um optimal die Kurve fahren zu können. Jedoch gibt es ein paar Probleme, zum Beispiel sollte das Auto zu weit nach links auf der Geraden auslenken, wird der gemessene Abstand zur Wand größer. Dadurch wird eine Kurve erkannt und das Auto fährt gegen die Wand. Um dieses Probelm zu lösen wird ein stabiler Regler benötigt, sodass das Auto nicht so stark den Abstand zur Wand korrigieren muss. Ein anderes Problem gibt es bei der Fahrt um die Kurve. Nachdem die Kurve fertig gefahren ist, soll das finish-Signal kommen. Es kann aber passieren das durch Rauschen beim Gyroskop das Signal schon früher als gewollt kommt. Auch dann wird fälschlicherweise eine Kurve erkannt und das Auto fährt sofort nachdem es mit der gewollten Kurve fertig ist noch eine. Deswegen haben wir einen Timer für die Kurvenerkennung eingebaut, das Auto darf keine erneute Kurve innerhalb von 0.7 Sekunden fahren nachdem es eine Kurve vollendet hat.

4 Visuelle Odometrie

Aufgrund der teilweise sehr ungenauen Selbstlokalisierung des Autos mittels des Odometrie-Pakets von PSES, liegt der Gedanke nahe auch die Kamera zur Lokalisierung zu verwenden, was vor allem bei der Navigation für den Rundkurs mit Hindernissen starke Vorteile bringt. Da die Kinekt als RGB-D-Kamera gewisse Vorzüge mitbringt, wurde diese für diese Aufgabe gewählt. Prinzipiell standen noch weitere Kameras mit weiterem Blickwinkel zur Auswahl mit der Option Stereo zu verwenden. Der kleine Blickwinkel ist für diese Vermessungsaufgabe ein Nachteil der Kinekt, ein Vorteil ist, dass man sich durch das Tiefenbild Schätzungen der Disparität sparen kann. Die folgenden Verfahren sind Mono-Verfahren, welche feature-basiert oder zumindest sparse arbeiten.

4.1 Mono-Verfahren

Zunächst sollen die Grundzüge des von uns implementierten Verfahren erläutert werden.

Allgemein lässt sich die Bewegung eines Punktes im Raum (genauer eines Starrkörpers) aus der Sicht eines gewählten Koordinatensystems anhand einer Rotation $\mathbf{R}$ und einer Translation $\mathbf{T}$ beschreiben:

$$\mathbf{X}_2 = \mathbf{R}\mathbf{X}_1 + \mathbf{T}. \tag{2}$$

Mit dem Zusammenhang zwischen den Koordinaten eines projizierten Punktes auf der Bildebene $\mathbf{x}$ und dem 3D-Punkt $\mathbf{X}$ aus Sicht des Kamerakoordinatensystems ergibt sich

$$\lambda\mathbf{x} = \Pi_0\mathbf{X}. \tag{3}$$

λ stellt den Abstand zu Kamera dar, welcher durch die Projektion mit Π_0 verloren geht. Die Projektion ergibt sich unter anderem aus der Kalibrationsmatrix der Kamera, welche für die Kinekt über ein ROS-Topic abrufen lässt. Damit ergibt sich die *Epipolareinschränkung*:

$$\mathbf{x}_2^T\hat{\mathbf{T}}\mathbf{R}\mathbf{x}_1 = 0. \tag{4}$$

$\mathbf{E} = \hat{\mathbf{T}}\mathbf{R}$ wird Essentialmatrix genannt. Diese Matrix lässt sich in OpenCV einfach mit dem 3- beziehungsweise 8-Punkt-Algorithmus ermitteln, womit im Anschluss die Bewegung bis auf einen Faktor berechnet werden kann. Dazu sollte eine größere Anzahl an Features auf den Bildern, welche die Kinekt liefert detektiert und über mehrere Bilder getrackt werden, sodass sich die Koordinaten $\mathbf{x}_1, \mathbf{x}_2$ ergeben. Dies wurde hier mittels des FAST-Detectors und einer Brute-Force Matching Methode getan. Im Idealfall ergibt sich so eine korrekte Rotationsmatrix und ein Translationsvektor, welcher bis auf einen

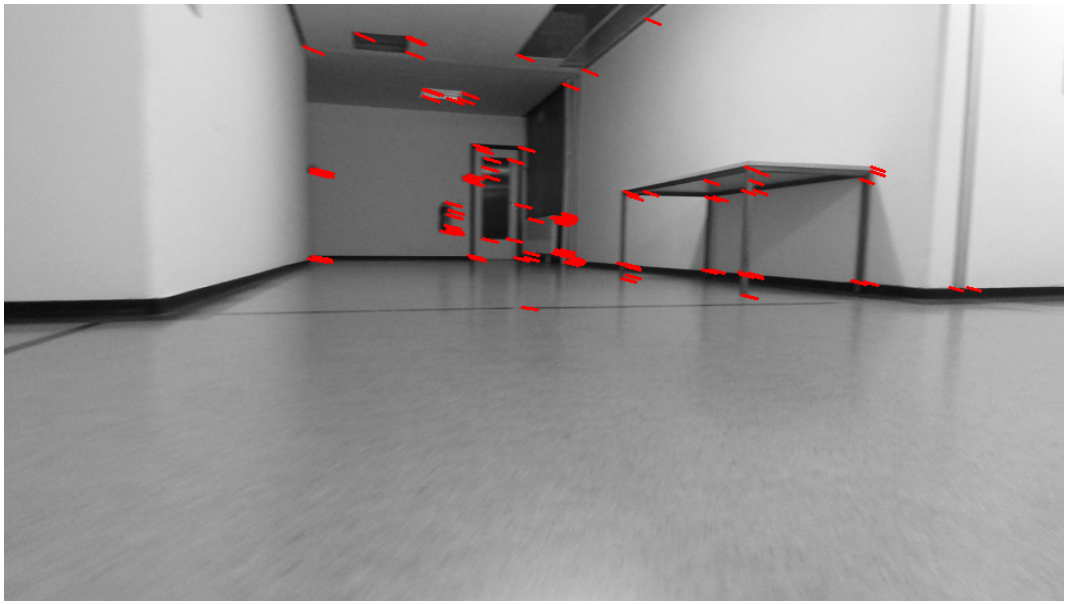

Abbildung 4.1: gemessener Optischer Fluss bei der Rundkursfahrt

Faktor stimmt. Mittels des Tiefenbildes kann der absolute Wert ermittelt werden. Alternativ kann auch die Geschwindigkeit verwendet werden, welche am Hinterrad per Hall-Sensor gemessen wird. Die einzelnen Teilbewegungen zwischen zwei Bildern lassen sich im Anschluss integrieren und stellen die Position und Orientierung im Raum dar. Ein entscheidendes Problem, was sich bei dieser Vorgehensweise ergeben hat war, dass in den schlecht beleuchteten merkmalsarmen Gängen einerseits nur wenige Features gefunden wurden, andererseits beim Tracking vor allem um Kurven nur wenige übrig blieben. Dies lässt sich gut in Abbildung 4.1 erkennen, in welcher der optische Fluss in rot markiert ist. Der optische Fluss beschreibt die Bewegung der Features auf dem Bild. Somit stellte sich das Problem, dass für eine präzise Schätzung nur stark ausreißerbehaftete Eingangsdaten zur Verfügung standen und keine vernünftigen Relativposen geschätzt werden konnten. Da wir an den äußeren Bedingungen nichts ändern wollten, haben wir diesen Ansatz nicht weiter verfolgt.

4.2 Alternativen

Als mögliche fertige RGB-D-Implementierung wurde zudem eine Klasse in OpenCV gefunden, welche eine Visuelle Odometrie bereitstellt [2]. Allerdings ist es nicht gelungen sinnvolle Ergebnisse damit zu erzielen, auch wenn eine fertige Implemetierung wünschenswert wäre.

Als weitere Alternative wurde die Direct-Sparse-Odometry (DSO) von Forschern der TU München [3], welche nicht mit Features, sondern direkt mit Bildpunkten arbeitet, ermittelt. Das besondere dabei ist, dass auch Kanten und bereits sanfte Grauwertänderungen auf weißen Wänden zur Bestimmung der Pose verwendet werden können. Die Qualität des Ergebnisses kann in Abbildung 4.2 bestaunt werden. Hier ist ein Ausschnitt einer Fahrt beim Fachgebiet ES zu sehen, wobei eine Punktwolke über alle aufgenom-

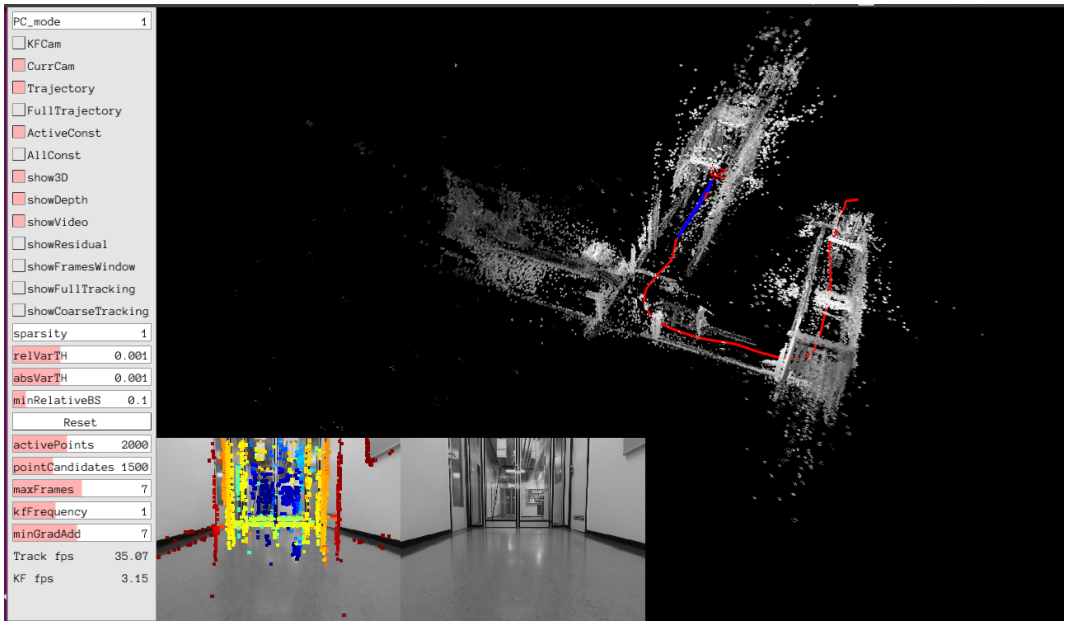

Abbildung 4.2: Beispiel der Kartierung und Posenbestimmung mittels DSO im HBI

menen Bilder hinweg erstellt wurde. In Rot ist der zurückgelegte Pfad eingezeichnet. Am Ende ist in Form einer Kamera die abschließende Orientierung eingezeichnet. Offenbar kann so eine recht gute Pose ermittelt werden. Mit ca 14 fps funktioniert dies sogar in Echtzeit. Problematisch wird dabei die weitere Verwendung des Ergebnisses der Schätzung, da nur wenig Rechenzeit für andere Aufgaben verbleibt. Aufgrund des Zeitdrucks und der Priorität auf dem Rundkurs mit Hindernissen wurde auf die Integration in unser System und ROS verzichtet. Allerdings ist dies ein vielversprechender Ansatz, da bei den merkmalsarmen Gängen, nur eine geringe Anzahl stabiler Features detektiert und getrackt werden kann.

5 Verkehrsschilderkennung

Bei dieser Aufgabe muss das Auto eine Auswahl von Verkehrsschildern erkennen und korrekt auf diese reagieren.

5.1 Zielsetzung

Das Auto soll den Rundkurs ohne Hindernisse fahren und dabei auf unsere Verkehrsschilder reagieren. Hierbei liegt der Fokus auf dem korrekten Erkennen der Schilder und nicht auf der Rundenzeit. Daher soll das Auto langsamer fahren als es könnte.

5.2 Implementierung

5.2.1 Erkennungssoftware

Wir haben uns dazu entschieden das ROS-Paket Find Object 2d zu benutzen [4]. Es vergleicht Features von Objekten in seiner Datenbank mit den Features des Kamerabildes. Wird ein Objekt erkannt gibt es die ID des Objektes und seine Position relativ zur Position an. Die Position wird bestimmt, indem die x,y und z Werte aus dem Tiefenbild entnommen werden. Aus den x,y und z Werten wird über eine Transformation der Output erzeugt. Da wir uns jedoch nur für die z-Werte des erkannten Objektes interessieren haben wir einen zusätzlichen Publisher eingebaut, der die z-Werte ausgibt.
Find Object 2d hat eine optionale GUI, mit welcher man Objekte aus dem Kamerabild auswählen und verschiedene Parameter einstellen kann.
Experimentell haben wir herausgefunden, dass GFTT als Detector und SIFT als Descriptor die stabilste Erkennung bei annehmbarer Latenz liefert. Diese beiden Algorithmen werden im Folgenden vorgestellt.

5.2.2 GFTT Detector

Dieser Detektor, eigentlich Shi-Tomasi Corner Detector genannt, ist prinzipiell eine Modifikation des Harris Corner Detector und wurde von J. Shi und C. Tomasi entwickelt. Zunächst verläuft die Berechnung exakt wie bei Harris [5].
Als erstes wird mit einer Window-Funktion die Intensitätsdifferenz der Stelle (x,y) in alle Richtungen berechnet, was letztendlich in einer Funktion

$$E(u,v) \approx \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix}$$

zusammengeführt wird und eine Matrix

$$M = \sum_{x,y} w(x,y) \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$

enthält. Deren Eigenwerte, werden anschließend zur Berechnung in der Ergebnisfunktion

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

verwendet, um zu bestimmen, ob das betrachtete Fenster einer Ecke enthält oder nicht. Genau hier findet sich die entscheidende Modifikation: die Ergebnisfunktion ergibt sich nun zu

$$R = min(\lambda_1, \lambda_2).$$

OpenCV bietet die Funktion cv2.goodFeaturesToTrack(), deshalb auch der Name GFTT Detector, in der genau dieser Algorithmus verwendet wird. Die Funktion findet die N stärksten Ecken in Abhängigkeit von Qualitätsparametern. Zunächst werden alle detektierten Ecken nach ihrer Qualität sortiert. Diese werden dann absteigend durchlaufen und es werden nacheinander die Nachbarn entfernt, die in einer vorher definierten Umgebung liegen. Das Ergebnis sind dann die N stärksten verbleibenden Ecken [6].

5.2.3 SIFT Descriptor

Um die gefundenen Ecken bzw. Schlüsselpunkte vergleichen zu können, braucht jeder Schlüsselpunkt eine mathematische Repräsentation. Dazu werden bei Scale-invariant feature transform (SIFT) um einen detektierten Schlüsselpunkt 16 Blöcke der Größe 4 Pixel x 4 Pixel – insgesamt 16x16 - gelegt. Für jeden dieser Blöcke wird ein 8 Bit Orientierungshistogramm erstellt. Ein Orientierungshistogramm wird aus der Richtung und der Größe der Gradienten in einem Block generiert. Anschließend wird es noch mit Hilfe einer Gaußschen Funktion über der Hälfte der Länge eines Blockes gewichtet. Der Deskriptor ist nun ein Vektor bestehend aus jedem einzelnen Histogramm – insgesamt 128 Bit. Auf diesen Vektor werden verschiedene Filter angewandt und er wird mehrmals normalisiert, damit SIFT robust gegen Drehungen, unterschiedliche Skalierungen, Beleuchtungsvariationen und Bildrauschen wird.

5.2.4 Programmstruktur

In diesem Abschnitt soll auf die Implementierung bzw. deren Struktur eingegangen werden.

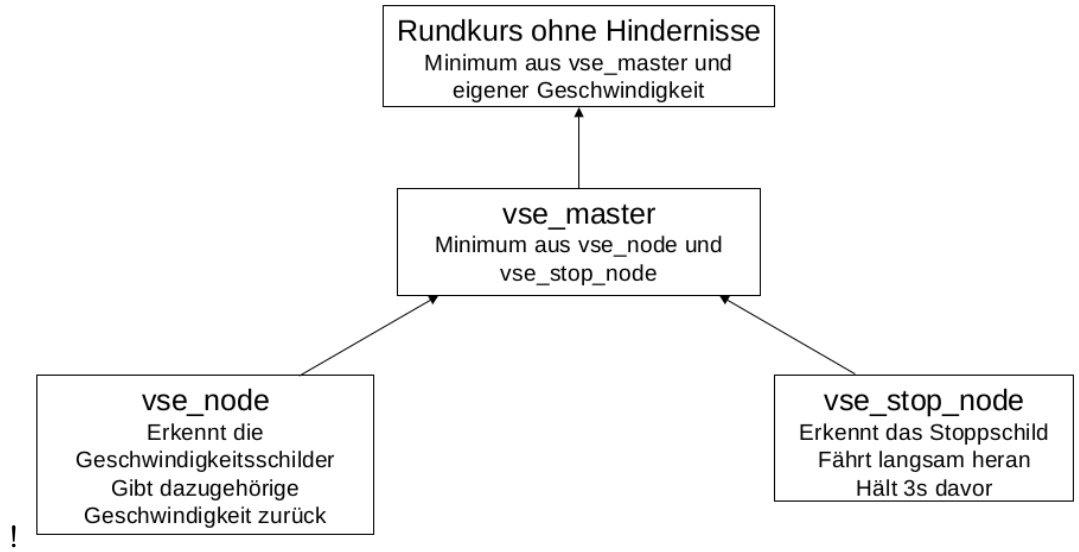

Abbildung 5.1: Programmstruktur Verkehrsschilderkennung

VSE_Node

Diese Node behandelt die Erkennung von Geschwindigkeitsschildern. Dafür werden die bekannten IDs der detektierten Schilder als Zustandsübergang für die Ausgabegeschwindigkeit verwendet. Zu Beginn wird die Geschwindigkeit auf den Maximalwert(1000) gesetzt, sodass das Auto mit der Geschwindigkeit aus dem Rundkurs zu fahren beginnt. Sobald ein Schild erkannt wird, wird die Geschwindigkeit auf den entsprechenden Wert geändert.

VSE_Stop_Node

Diese Node behandelt die Erkennung des Stoppschildes. Wird kein Stoppschild erkannt gibt VSE_Stop_Node die maximale Geschwindigkeit(1000) aus. Wird das Stoppschild in 5 Metern erkannt, fährt das Auto erst langsam(200) auf es zu. Sobald es einen Abstand von 2 Metern hat bleibt es 3 Sekunden stehen. Im Anschluss fährt es mit seiner ursprünglichen Geschwindigkeit weiter.

VSE_Master_Node

Diese Node ist für die Zusammenführung der Geschwindigkeiten aus VSE_Node und VSE_Stop_Node zuständig. Sie gibt immer das Minimum der beiden Nodes an den Rundkurs weiter.

5.3 Probleme und Ausblick

Wir hatten große Probleme mit der Leistung der gegebenen Hardware. Für die Objekterkennung müssen wir das Kamerabild in QHD nutzen, da das Bild in HD eine Latenz von mehr als 2 Sekunden hat - mit QHD haben wir eine Latenz von etwa einer Sekunde. Das beeinflusst die Reichweite und Stabilität unserer Erkennung. Für die Objekterkennung ist ein häufiges Iterieren durch die Bildmatrix von Nöten, welches durch eine CPU nur schlecht parallelisiert werden kann und in Kombination mit alter Hardware ein massives Performanzproblem hervorruft. Durch den Einbau einer Grafikkarte lässt sich dieses Problem lösen. Zusätzlich bietet sich dadurch die Möglichkeit eine leistungsfähigere Objekterkennungssoftware, die in CUDA oder OpenGL geschrieben ist und die parallelisierbarkeit der Aufgabe besser ausnutzt, zu verwenden. Beispiele hiefür sind „Vision for Robotic“ der TU Wien und „You only look once“ welches auf dem neuronalen Netzwerk „Darknet“ aufbaut.

6 Rundkurs mit Hindernissen

6.1 Zielsetzung

Das Ziel bei dieser Aufgabe ist ein Hindernissparcour zu durchfahren. Die Lösung soll für eine beliebige Anordnung der Hindernisse funktionieren. Dabei soll die Position des Fahrzeugs möglichst genau erfolgen und die Berechnung der Trajektorien, sowie Steuerung des Fahzeugs optimal sein. Des Weiteren soll die Performanz der Lösung weitesgehend unempfindlich gegenüber Lichtverhältnissen sein.

6.2 Implementierung

Für den Rundkurs mit Hindernissen wurde das *Navigation Stack* verwendet. Dieser ist schematisch in Abbildung 6.2 dargestellt.
Anhand der Karte des Fachgebiets, die uns zur Verfügung gestellt wurde, konnte die *globale Costmap* erstellt werden. Für die Erstellung der *lokalen Costmap* wurde der Laserscan der Kinect verwendet. Um ungültigen Messungen der Kinect vorzubeugen, wurde zusätzlich ein Filter programmiert welcher die *inf* und *nan* Werte auf den maximalen Wert umrechnet. Dadurch sollte verhindert werden, dass der TEB-Local Planner ungültige Werte erhält. Außerdem wurde das *kinect_filter* aus dem Paket *pses_kinect_utilities* verwendet um Störeinflüsse wie z.B. Lichtreflexionen zu unterdrücken.
Die Vorgabe der globalen Ziele erfolgte mittels der graphischen Benutzeroberfläche RViz. Dabei wurde ein verteiltes System konfiguriert und das Fahrzeug als *rosmaster* und das Laptop als *slave* deklariert. Dadurch war das Empfangen von Topics als auch die Vorgabe von Zielen möglich. Das Szenario ist in Abbildung 6.1 dargestellt. In der Abbildung ist deutlich die lokale Costmap zu erkennen, in der auch das Hindernis dargestellt wird, das anhand des Laserscans registriert wurde. Für die Lokalisierung des Fahrzeugs wurde der AMCL-Algorithmus verwendet.
Um das bestmögliche Ergebnis aus dem Navigation Stack zu herauszuholen ist es jedoch notwendig, die entsprechenden Parameter für die globale und lokale Costmap, als auch die Parameter des AMCL auf das entsprechende Fahrzeug möglichst gut anzupassen. An dieser Stelle ist der Anwender auf das Trial-and-Error Prinzip angewiesen.

6.2.1 TEB - Local Planner

Nachdem die Konfiguration des Base Local Planners daran scheiterte, dass dieser Trajektorien so plante, dass die die Ecken des Rundkurses geschnitten wurden, wechselten wir zum Timed-Elastic-Band (TEB) Local Planner. Die Aufgabe des Local Planners ist Trajektorien auf einer lokalen Karte so zu planen, dass Hindernisse im direkten Umfeld berücksichtigt werden und gleichzeitig dem globalen Ziel näher zu kommen. Dieser plant zufällige Trajektorien bis zu einem gewissen Planungshorizont und bewertet diese anhand einer Kostenfunktion. Die Trajektorie mit dem besten Gütewert wird ausgewählt. Für die Kostenfunktion entscheidend ist auch die Costmap. Über Parameter lassen sich somit die Relevanz des Abstands von Hindernissen, die benötigte Zeit sowie Beschleunigungen und Stellgrößen einstellen. Zudem lassen sich die Forwärtssimulationsdauer

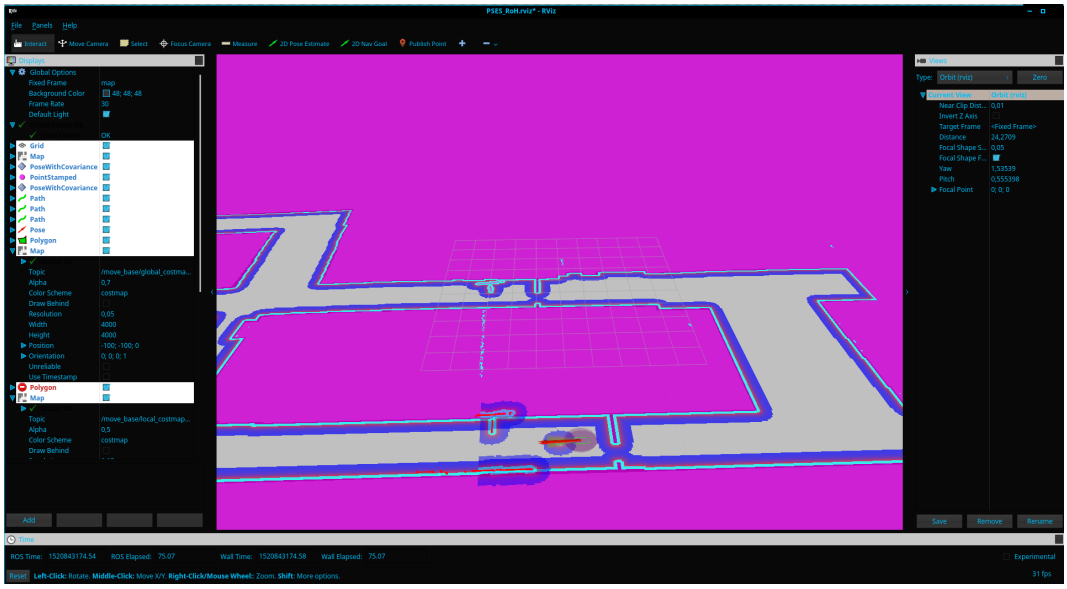

Abbildung 6.1: Programmstruktur Move Base

sowie der Fußabdruck des Autos, als auch Geschwindikeits- und Lenkwinkelbegrenzungen vorgeben. Die gesamte Parameterauswahl ist im ROS-Wiki gut dokumentiert. Ausgegeben werden dabei eine Zielgeschwindigkeit und eine Rotationsgeschwindigkeit, welche sich direkt in die Lenkwinkel des Autos umrechnen lässt.

6.2.2 Global Planner

Der Global Planner, welches im Paket Move Base enthalten ist, hat die Aufgabe die Trajektorie für ein gewünschtes Ziel auf einer globalen Karte zu erstellen. Diese wird dann dem Local Planner zur Verfügung gestellt. Im Move Base Paket existieren verschiedene Global Planner, welche durch den Parameter „base global planner„ausgewählt werden.

Es stehen dabei folgende drei Global Planner als Plug-In zur Verfügung

- Navfn (default): Ein Raster basierter Global Planner, welches eine Navigationsfunktion zur Planung nutzt.
- Global Planner: Ein schneller und flexibler Ersatz zum Navfn.
- Carrot Planner: Ein einfacher Planner welches so nah wie möglich zum Zielpunkt plant selbst wenn sich das ziel in einem Hindernis befindet.

Für unsere Umsetzung haben wir das Navfn Plug-In verwendet. Dabei wird als erstes aus einer globalen Karte, die der Map Server bereitstellt, die Global Costmap erstellt. Der Global Planner greift dann auf die Costmap zurück und berechnet mittels einer Kostenfunktion die Trajektorie zu einem vorgegebenen Ziel. Hierfür stehen für die Optimierungs Parameter für den Global Planer und für die Costmap zur Verfügung, welches ausführlich in der Ros-Wiki dokumentiert ist.

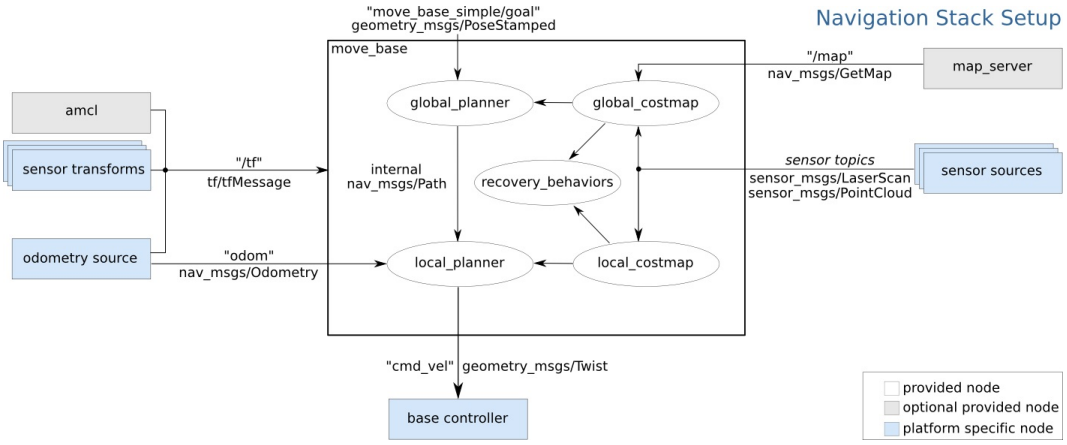

Abbildung 6.2: Schematische Darstellung des Navigation Stack [7]

6.2.3 Adaptive Monte Carlo Localization - AMCL

AMCL ist das Paket das für die Lokalisierung des Fahzeugs verwendet wurde. Es verwendet den Laserscan und die Odometrie des Fahrzeugs und schätzt über einen probabilistischen Ansatz seine Position. Es existiert eine Reihe von Parametern, die zur Optimierung der Positionsbestimmung verwendet werden können. Beispielsweise kann über die Parameter die Kovarianz der Anfangsposition des Fahrzeugs oder Annahmen über das verwendete Lasermodell gemacht werden. Alle Parameter sind detailliert in der ROS Wiki beschrieben und sollten für jedes Fahrzeug individuell angepasst werden, weshalb an dieser Stelle nicht detaillierter auf die einzelnen Parameter eingegangen wird. Ein wichtiger Punkt ist jedoch der Umstand, dass sich das Rückwärtsfahren des Fahrzeugs sehr negativ auf die Lokalisierung auswirkt. Deshalb sollte dieses im Local Planner über die entsprechenden Parameter möglichst stark bestraft werden.

6.3 Probleme

Die Hauptschwierigkeit beim Rundkurs mit Hindernissen war die Qualität des Laserscans und generell die Robustheit der Lösung über das Navigation Stack. Es konnte beobachtet werden, dass die Qualität des Laserscans nicht unwesentlich von den Lichtverhältnissen abhängt. So haben beispielsweise die Reflexionen vom Boden, verursacht durch die Raumbeleuchtung, dazu geführt, dass in der lokalen Costmap nicht existente Hindernisse erscheinen und dadurch die Trajektorienplanung stark erschweren. So ein Fall lässt sich in Abbildung 6.1 beobachten. Der kinect_filter konnte dieses Problem allerdings deutlich reduzieren. Die Robustheit des Navigation Stack hängt stark von der Parameteroptimierung ab und ist dieser gegenüber sehr empfindlich. Eine optimale Einstellung der Parameter ist daher ein Trial-and-Error Prozess.

7 Fazit

Insgesamt hat das Projekt einen guten Einblick in die Entwicklung einer Lösung aus dem Bereich der Bildverarbeitung und des autonomen Fahrens geboten. Insbesondere war die Arbeit mit einem Echtzeitbetriebssystem wie ROS für die meisten Studenten neu aber auch gleichzeitig sehr interessant. Kombiniert mit einer richtigen Teamarbeit und einer gelungenen Projektorganisation wird der Eindruck von der Lösung einer ingenieurstechnischen Problemstellung im Team sehr gut vermittelt. Um jedoch zuerst mit der tatsächlichen Implementierung der Aufgaben zu beginnen, sollte zunächst der Umgang mit ROS erlernt werden. Dabei hat sich die ROS Wiki, sowie die zahlreichen Tutorials, als nützlich erwiesen. Nichtsdestotrotz würden sich am Anfang des Projektseminars weitere Workshops sehr gut anbieten um die Einarbeitungszeit zu verkürzen.
Des Weiteren ist unsere Gruppe mit der Betreuungsrelation während der Veranstaltung sehr zufrieden. Zum Einen sind die regelmäßigen Meetings mit dem Betreuer als auch die Tipps und Ratschläge von weiteren Betreuern als sehr positiv zu bewerten.

Ein möglicher Verbesserungsvorschlag wäre der Einbau einer Grafikkarte, sodass auch anspruchsvollere Themen aus dem Bereich der Bildbearbeitung behandelt werden können.

Abbildungsverzeichnis

Literatur

[1] Eric Lenz. Handout zum projektseminar echtzeitsysteme.

[2] https://docs.opencv.org/3.4.1/d0/d60/classcv_1_1rgbd_1_1rgbdodometry.html.

[3] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 2018.

[4] http://wiki.ros.org/find_object_2d.

[5] https://docs.opencv.org/3.3.0/dc/d0d/tutorial_py_features_harris.html.

[6] https://docs.opencv.org/3.3.0/d4/d8c/tutorial_py_shi_tomasi.html.

[7] http://wiki.ros.org/navigation/tutorials/robotsetup.