# Projektseminar Echtzeitsysteme

**Studienarbeit eingereicht von**
**Team Matchbox**
Tobias Averhage, Clemens Chu, Dennis Hanslik, Alexander Späth
am 13. April 2016

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

Betreuer: M.Sc. Geza Kulcsar

## Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Studienarbeit 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 13. April 2016

_____

T. Averhage, C. Chu, D. Hanslik, A. Späth

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| c2c | Car-2-Car |
| ROS | Robot Operating System |

# 1 Aufgabenstellung

Bei diesem Projektseminar soll ein Team, bestehend aus vier Studenten, einem Roboterauto das „autonome Fahren“ beibringen, wobei der Begriff „autonom“ nicht abschließend definiert ist. Vor allem in diesem Semester lag der Hauptfokus auf der überarbeiteten Hardware des Autos und zu testen, ob diese lauffähig ist.

Das Seminar soll die Studenten lehren im Team ein Ziel zu verfolgen, einen Zeitplan zu erstellen und sich in die neue Plattform einzuarbeiten. Der Fachbereich Echtzeitsysteme verfolgt darüber hinaus das Ziel mit diesem Fahrzeug am Carolo-Cup[1] teilzunehmen.

Vorgegebenes Ziel für das Seminar war

- Autonomes Fahren mittels Sensorik
- Kamera-Inbetriebnahme zur Erkennung von ArUco-Markern[2]
- Eine Anwendung für die ArUco-Marker finden (z.B. durch Tore fahren)

Als weitere Optionen standen beispielsweise

- Fernsteuerung
- Car-2-Car Kommunikation
- ROS-basierte Simulation

zur Auswahl. Grundsätzlich konnten wir, nach Absprache, auch eigene Ideen umsetzen.

# 2 Aufbau des Fahrzeugs

## 2.1 Hardware

Das Fahrzeug basiert auf einem RC-Fahrzeug mit Elektromotor. Gesteuert wird es über einen Microcontroller, der seit diesem Semester einen Mini-PC zur Seite gestellt bekam. Der Microcontroller ist komplett lauffähig und kann Befehle des Mini-PCs nutzen, um Lenkung und Geschwindigkeit zu regeln. Zusätzlich verfügt der Wagen über drei Ultraschallsensoren (einer vorne und jeweils einer an den Seiten), einen Liniensensor in der Front und einen Hall-Sensor zur Ermittlung der Radumdrehungen des linken Hinterrads. Diese Signale werden über den Microcontroller an den Mini-PC weitergeleitet. Unabhängig vom Microcontroller ist noch eine Kamera an den Mini-PC angeschlossen, welche an der Front des Fahrzeugs fix montiert ist und das Vorfeld aufzeichnet.

[1] Die TU Braunschweig läd jährlich zum Carolo-Cup. Informationen finden sie unter https://wiki.ifr.ing.tu-bs.de/carolocup/carolo-cup

[2] http://www.uco.es/investiga/grupos/ava/node/26

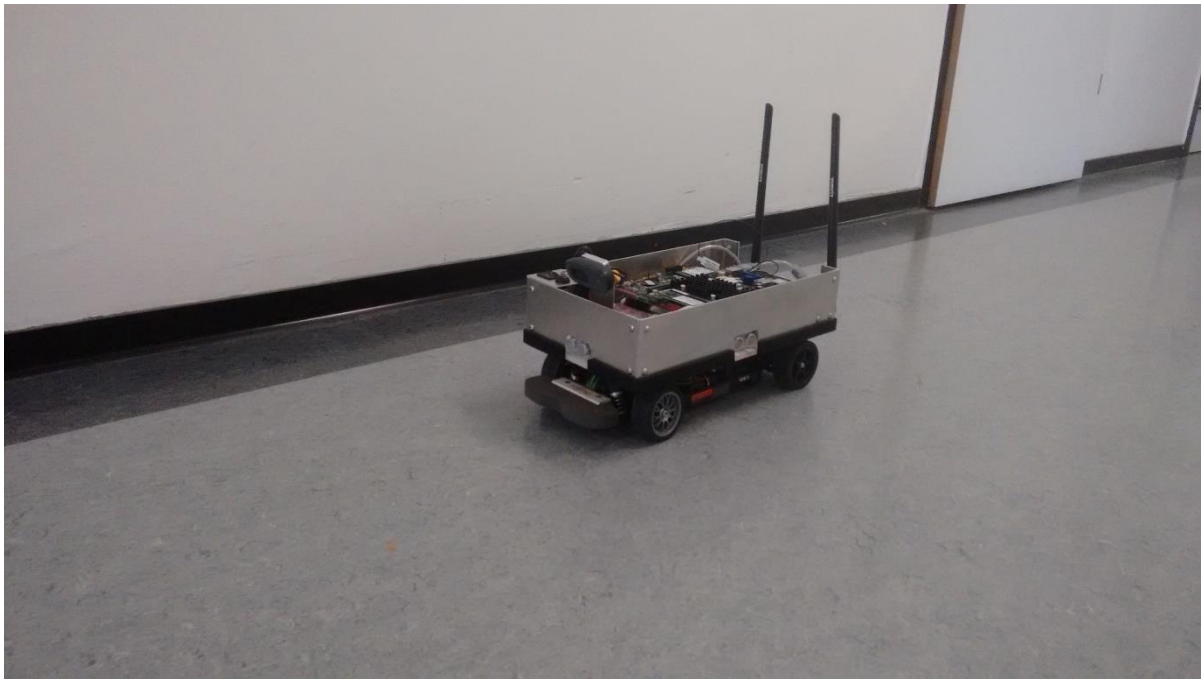

Abbildung 1: Darstellung des Fahrzeugs ohne Modifikationen

## 2.2 Software

Der Mini-PC kommt mit einem vorinstallierten Ubuntu-Linux und ROS[3]. Zusätzlich installierten wir Atom als Texteditor. Den Code lagerten wir in einem online Git-Repository.

Gegeben waren bereits zwei kleine ROS-Packages. Das erste erklärte die Kommunikation mit dem Microcontroller, das zweite die Inbetriebnahme der Kamera und das Erkennen der ArUco-Marker.

# 3 Unser Konzept

Zu Beginn des Seminars stellten sich einige Fragen, die den Verlauf des gesamten Semesters beeinflussten. Nach einer kurzen Analyse der gegebenen Hardware stand die Frage im Raum, ob wir mit den gegebenen Sensoren arbeiten konnten und welche Ziele wir damit anstreben könnten. Auch fragten wir uns, ob wir weitere Sensoren anschließen sollten um die Möglichkeiten zu erweitern.

Wir entschlossen uns die vorhandene Hardware in keinem Punkt abzuändern und auch keine weitere Hardware zu nutzen. Wir wollten die Grenzen der vorhandenen Plattform ermitteln und damit eine Grundlage für die folgenden Semester schaffen. Erst, wenn man die Grenzen der vorhandenen Plattform kennt, kann man über sinnvolle Verbesserungen nachdenken.

Wir setzten uns das Ziel durch Tore von ArUco-Markern zu fahren und einem Wandverlauf folgen zu können. Außerdem suchten wir einen sinnvollen Einsatz für Car-2-Car.

[3] http://www.ros.org

# 4 ROS

Das Robot Operating System ist ein Softwarepaket, das extra für den Betrieb von Robotern entworfen wurde. Als Kernfunktion stellt es bereits ein Nachrichtenübertragungssystem (über sogenannte „Topics“) bereit und eine starke Community entwickelt laufend neue Packages, die in das System eingebunden und genutzt werden können.

Durch den modularen Aufbau und die zentrale Nachrichtenschnittstelle kann man sich beliebig aus dem großen Fundus an Teillösungen bedienen und in seine eigenen Projekte einbinden.

Für unser Projekt nutzten wir zwar nur wenige der vorhandenen Packages und entwickelten lieber selbst, aber wir setzten trotzdem immer auf die grundsätzliche Struktur des ROS-Systems. So nutzten wir beispielsweise die Nachrichtenübertragungstechnik von ROS, statt eine eigene Paketübermittlung zu entwerfen um Car-2-Car zu realisieren.

Jedes ROS-Package läuft später als eigenständiger ROS-Node und muss sich immer zu einem ROS-Master verbinden. Diesen startet man entweder mit dem Befehl *roscore*, oder über den Befehl *roslaunch* und entsprechende Launch-Files. In den Launch-Files kann man direkt die zu startenden Nodes angeben, oder man startet sie manuell im Nachhinein.

## 4.1 ROS-Topics

Zur Kommunikation zwischen unseren Nodes setzen wir konsequent auf ROS-Topics. Diese Nachrichtenpakete werden allen, mit dem Master verbundenen, Nodes zur Verfügung gestellt und können von ihnen genutzt werden. Nodes können sowohl Daten über Topics versenden, als auch empfangen.

Man kann auch über das Terminal Topics schreiben. Der Befehl

*rostopic pub speed std_msgs/Int32 3*

sendet beispielsweise den Integer-Wert 3 über das Topic „speed“. Des Weiteren kann man Namespaces verwenden um seine Topics zu ordnen. Dazu muss man zwischen absoluten und relativen Topic-Bezeichnern unterscheiden. Das Topic „speed“ ist nicht zwingend dasselbe wie „/speed“, könnte aber dasselbe wie „/MyNamespace/speed“ sein. Mit dem Befehl *rostopic list* erhält man eine Auflistung aller aktuell registrierten Topics. Den Namespace eines Terminals wechselt man mit *export ROS_NAMESPACE=“MyNamespace“*.

Man kann sich im Terminal auch den Datentransfer über ein Topic ansehen. Dazu genügt beispielsweise der Befehl

*rostopic echo speed*

um sich alle übertragenen Geschwindigkeitswünsche anzeigen zu lassen.

Es folgt eine Auflistung der durch uns genutzten Topics, deren Aufbau und Bedeutung.

## 4.2 speed

Ein relatives Topic, das von car_handler empfangen wird und dazu dient die Geschwindigkeit des Fahrzeugs zu beeinflussen. Dieser Int32-Wert muss zwischen -10 und 10 liegen und wird dem Microcontroller übergeben und stellt die entsprechende Geschwindigkeitsstufe ein.

## 4.3 steering

Ein relatives Topic, das von car_handler empfangen wird und dazu dient die Lenkung des Fahrzeugs zu beeinflussen. Dieser Int32-Wert muss zwischen -50 und 50 liegen und wird dem Microcontroller übergeben. Der Wert -50 stellt dabei einen Vollausschlag der Lenkung nach links und ein Wert von +50 einen Vollausschlag nach rechts dar.

## 4.4 topSpeed

Ein relatives Topic, das von signs versendet und von car_handler empfangen wird. Es wird nur versendet, wenn ein Verkehrsschild mit einer Geschwindigkeitsbegrenzung gesehen wurde. Dieses Topic stellt nicht die einzustellende Geschwindigkeit dar, sondern den Maximalwert. Es muss also ständig von car_handler geprüft werden, welcher Wert einzustellen ist. Im Prinzip nutzt er dazu min(speed, topSpeed).

## 4.5 speed_actual

Ein relatives Topic, das von car_handler versendet wird und die wirklich einstellte Geschwindigkeit darstellt ( min(speed, topSpeed) ). Nodes die mit der Geschwindigkeit des Fahrzeugs Berechnungen anstellen müssen daher nicht auf „speed“ und „topSpeed“ horchen und selbst den vermutlich eingestellten Wert ermitteln, sondern erfahren durch diesen Int32-Wert was wirklich an den Microcontroller gesendet wurde.

## 4.6 arucoPos

Ein relatives Topic, das vom Package arucopos versendet und von signs und gates verwendet wird. Es besteht aus einer selbst definierten Struktur (arucopos\msg\vArucoPos.msg) und enthält neben der ID auch die X- und Y-Entfernung aller bisher erkannten Marker in cm zum Wagenmittelpunkt.

## 4.7 Kp, Ki, Kd

Diese drei relativen Topics (Int32) dienen dem Austausch von P-, I- und D-Werten für den PID-Regler. Sie werden nur genutzt um den PID zu justieren, indem man z.B. über das Terminal Werte setzt.

## 4.8 direction

Ein relatives Topic, das von signs genutzt wird um wall_follower über die gewünschte Abbiegerichtung zu unterrichten. Es wird versendet, sobald das Fahrzeug eine Kreuzung (oder Engstelle) erreicht hat. Es besteht aus einem Int32, der über die arucopos\include\arucopos\myMarker.h definiert wurde. Es werden nur die IDs AR_SIGN_DIR_L, AR_SIGN_DIR_R, AR_SIGN_DIR_S, AR_SIGN_NARROW_MIN und AR_SIGN_NARROW_END versendet.

## 4.9 c2cReqPerm

Ein relatives Topic, das von signs genutzt wird um c2c dazu aufzufordern die nächste Gefahrenstelle zu sperren. Es besteht aus einer selbst definierten Struktur (c2c\msg\reqperm.msg) und enthält die ID des Markers und die gewünschte Abbiegerichtung.

## 4.10 c2cPerm

Ein relatives Topic, das von c2c genutzt wird um die Sperrung der Gefahrenstelle zu bestätigen. Es wird in signs genutzt um die Geschwindigkeit wieder herauf zu setzten und das Befahren der Gefahrenstelle einzuleiten. Es enthält dieselben Daten wie das zuvor gesendete „c2cReqPerm“.

## 4.11 c2cCrPassed

Ein relatives Topic, das von signs oder wall_follower gesendet wird um das Ende der Gefahrenstelle an c2c zu übermitteln. Daraufhin wird c2c die Gefahrenstelle wieder freigeben. Es wird nur „true“ übertragen. Bei dieser Message ist der Inhalt uninteressant. Es zählt lediglich, dass sie empfangen wird.

## 4.12 globalCall

Das einzige absolute Topic wird von c2c versendet und soll von anderen Fahrzeugen empfangen werden. Es besteht aus der selbst definierten Struktur c2c\msg\enterDangerZone.msg und enthält Informationen über die ID des Markers, die gewünschte Abbiegerichtung und den Status der Sperrung. Beim Status steht 0 für „ich gebe frei“, 1 für „ich möchte gerne sperren“ und 2 für „ich habe gesperrt“.

# 5 ArUco-Marker

ArUco-Marker, die entfernt an QR-Codes erinnern, bestehen aus 7 mal 7 „Pixeln“. Da die äußeren Reihen/Spalten immer schwarz gefärbt sind, bleiben nur die inneren 5 mal 5 Pixel zur Darstellung

des Markers. Die Marker sind immer eindeutig, auch bei Drehungen. Es gibt insgesamt 1024 verschiedene Marker.

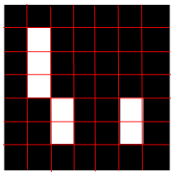

Abbildung 2: Aufbau eines ArUco-Marker (Quelle: http://www.uco.es )

Zur Erkennung wird, neben dem Kamerabild, das ROS-Package aruco verwendet. Mit einigen wenigen Zeilen Code wird eine Liste der derzeit im Sichtbereich erkannten Marker geliefert – jeder mit seiner eindeutigen ID. Wenn zu einem Zeitpunkt zwei oder mehr Marker mit derselben ID im Sichtbereich sind, dann wird nur einer der Marker erkannt.

Die Erkennung der Marker hängt zum Teil von ihrer Größe ab. Zum einen müssen sie komplett im Bild sein, zum anderen dürfen sie aber auch nicht zu groß oder zu klein sein.

Über die Methode setMinMaxSize() der Klasse MarkerDetector kann man die relative Größe der zu erkennenden Marker einschränken (wir wählten setMinMaxSize(0.01, 0.5) ).

Des Weiteren entschieden wir uns dazu Marker in einer Größe von circa 3 mal 3 cm zu verwenden. Sie sind, mit den vorgenommenen Änderungen an setMinMaxSize(), auf über einem Meter erkennbar und bleiben lange erkannt, selbst wenn sie sich zum Bildrand bewegen. Größere Marker würden früher den Bildrand überschreiten und dann nicht mehr erkannt werden.

Die Entscheidung für kleine Marker hat aber natürlich nicht nur Vorteile. Bei der geringen Auflösung der Kamera (wir wählten 640 mal 480 Pixel) ist die Erkennung bei über einem Meter fast nur im Stillstand möglich. Auch führen umgeknickte Ecken oder Teilschatten dazu, dass die Marker nicht mehr erkannt werden. Auch bei starken Lichtreflektionen auf dem Flurboden werden Marker oft nicht erkannt.

Bedingt durch diese Schwierigkeiten setzten wir darauf die Position der einmal erkannten Marker gut berechnen zu können. Dadurch konnten wir bei den kleinen Markern bleiben die oft im Bild sind, mussten sie aber nicht ständig sehen/erkennen.

# 6 Car-2-Car

Die Verbindung zweier Roboter funktioniert in ROS über verschiedene Wege. ROS-Topics sind grundsätzlich so ausgelegt, dass sie bereits über ein Netzwerk verteilt werden können.

## 6.1 ROS-Standard

Von Haus aus können ROS-Nodes nur mit Nodes kommunizieren, die mit demselben ROS-Master verbunden sind. Das bedeutet, dass nur einer der Roboter einen ROS-Master startet und die Nodes

auf dem zweiten (und jedem weiteren) Roboter über das Netzwerk eine Verbindung zu dem Master aufbauen müssen. Sobald die Verbindung einmal unterbrochen wird verlieren alle externen Nodes ihre Verbindung und stellen diese auch nicht automatisch wieder her. Alle externen Nodes müssen neu gestartet werden und ihre Verbindung zum Master neu aufbauen. Des Weiteren werden alle Topics über das Netzwerk verteilt. Bei hohem Datenaufkommen, beispielsweise durch Kamerabilder vieler Roboter, könnte das dazu führen, dass Nachrichten verzögert übermittelt werden.

## 6.2 Historie

Eine Recherche im Internet ergab, dass dieses Problem bereits oft in Angriff genommen wurde, aber nie zu einem Ergebnis führte. Bis 2013 wurden immer die Packages Master Sync, Foreign Relay und Fkie Multimaster empfohlen[4]. Jedes dieser Pakete hat Einschränkungen und scheint die Testphase nie abgeschlossen zu haben. Nach 2013 findet man keine Hinweise mehr auf weitere Entwicklungen der empfohlenen Pakete und für die aktuelle ROS-Version stehen sie nicht zur Verfügung.

## 6.3 ROCON

Inzwischen wurde ROCON[5] entwickelt. Es wurde als Multi-Master-System konzipiert, so dass jeder Roboter eigenständig arbeiten kann, selbst wenn die Verbindung unterbrochen wird. Es wird ein so genanntes Concert gestartet. Dazu startet ein Roboter einen Hub und jeder Roboter ein Gateway. Die Gateways verbinden sich automatisch zum Hub und stellen damit die Grundlage für die Übermittlung der Topics.

Da ROCON nicht jedes Topic über das Netzwerk published, müssen als letzter Schritt noch die gewünschten Topics ausgewählt werden.

[4] Beispiel für einen Multi-Robot Aufbau: http://www.j-robotics.com/turtlebot-multi-robot-system/

[5] Robotics in Concert (ROCON): http://wiki.ros.org/rocon

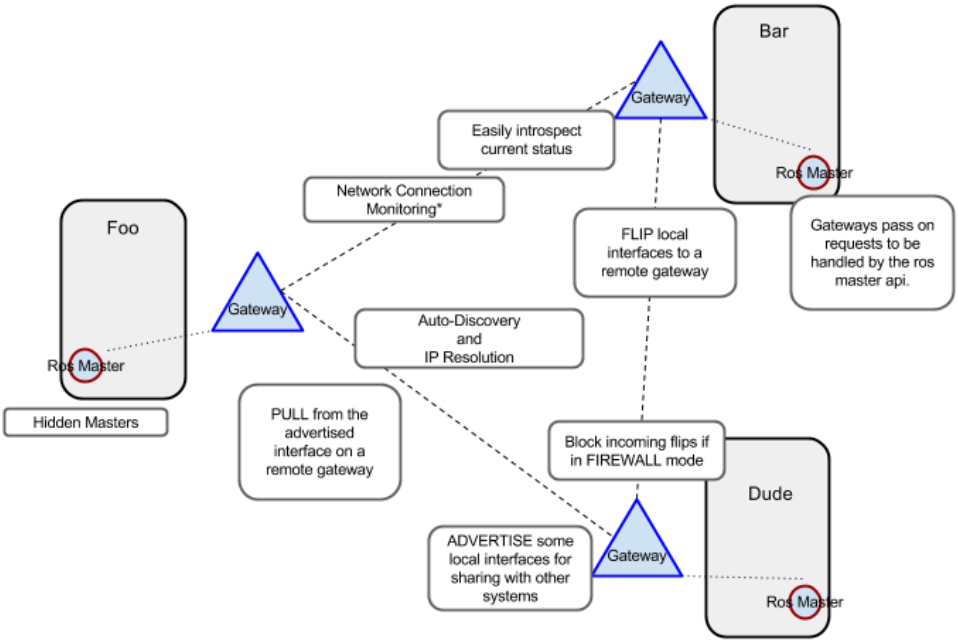


Abbildung 3: Verbindung der Master über Gateways (Quelle: http://ros.org )

## 6.4 Entscheidung

ROCON stellt das beste Konzept dar und wäre unsere Wunschvorstellung für Car-2-Car gewesen. Leider konnten wir die Verbindung der Gateways nicht standfest aufbauen. Sofort nach dem Start stand die Verbindung kurz und wurde dann immer unterbrochen. Da wir den Fehler nicht finden oder eingrenzen konnten mussten wir auf die ROS-Standardlösung zurückgreifen.

Grundsätzlich ist der Code aber unabhängig von der Art der Verbindung zwischen den Fahrzeugen. Sollte eine Möglichkeit gefunden werden die Fahrzeuge mit ROCON zu verbinden, dann würde der Code auch über diese Schnittstelle ohne Änderungen weiter funktionieren.

# 7 Odometrie

Die Odometrie für unser Fahrzeug ist prinzipiell in zwei größere Unterabschnitte aufgeteilt. Der erste Unterabschnitt behandelt die „direkte“ Odometrie, also die Auswertung von empfangenen Kamerabildern, und die Verwertung der daraus gewonnenen Daten. Dazu musste eine Methode entwickelt werden, um zuverlässig und genau die von der Kamera gewonnenen Daten in eine Form zu bringen, die das Entfernungs- und Richtungsverhältnis der erkannten Weltobjekte in Bezug zum Fahrzeug darstellt. Der zweite Unterabschnitt befasst sich mit der „indirekten“ Odometrie. Hierbei geht es hauptsächlich darum die Veränderung in dem Verhältnis zwischen bereits erkannten Weltobjekten und dem Fahrzeug auch weiterhin Realitätsgetreu zu aktualisieren, selbst wenn diese das Blickfeld der auf dem Fahrzeug montierten Kamera verlassen. Mit diesen beiden Methoden ist es dem Fahrzeug dann möglich, zu jedem Zeitpunkt die räumliche Lage von relevanten Weltobjekten mit hoher Genauigkeit an die relevanten Module weiterzureichen.

## 7.1 Grundlegende Annahmen

Um die Methodik der Odometrie etwas zu vereinfachen, wurden zum Anfang einige Annahmen über den Zustand der Umwelt des Fahrzeuges getroffen. Die erste und wohl wichtigste Annahme die getroffen wurde ist, dass alle Weltobjekte, die für die Odometrie des Fahrzeuges relevant sind, durch ArUco-Marker dargestellt werden. Die Erkennung von Wänden durch die am Fahrzeug angebrachten Ultraschallsensoren wird hierbei außer Acht gelassen, da diese in der Implementierung im Code von einem separaten Modul gehandhabt wird. Des Weiteren wird davon ausgegangen, dass die Umgebung des Fahrzeugs eine annährend Zweidimensionale Umgebung ist. Zuletzt wird noch davon ausgegangen, dass die ArUco Marker, die die Umwelt für das Fahrzeug erkenntlich machen, von einer einheitlichen Beschaffenheit sind. Insbesondere ist es für die Odometrie wichtig, dass eine einheitliche Größe und eine Einheitliche Aufstellung der Marker vorhanden ist.

Da diese Annahmen keinen größeren Mehraufwand in der Gestaltung der spezifischen Implementation der ArUco-Marker darstellten, und die besagten annahmen komfortabel innerhalb der Rahmenbedingungen der Aufgabenstellung lagen, aber die der Odometrie zugrundeliegenden Mathematik signifikant vereinfacht, wurden sämtliche odometrische Methoden in ihren Implementationen auf die durch diese Annahmen erschaffenen Eigenschaften ausgelegt.

## 7.2 Vermessung der Fahrzeugeigenschaften

Es war zur genauen Kalibrieren der Odometrie ebenfalls nötig einige Messungen des Fahrzeugverhaltens zu unternehmen. Die erste Eigenschaft die vermessen wurde war der Wenderadius des Fahrzeugs. Dies war enorm wichtig für die indirekte Odometrie, da bei einer Kurvenfahrt sich sowohl die Orientierung als auch die Position des Fahrzeugs verändert. Dies verursacht, dass ArUco-Marker aus dem Blickfeld der Kamera verloren gehen, und sich das Entfernungs- und Richtungsverhältnis zwischen dem Fahrzeug und den Markern verändert. Um eine Formel für die Beziehung zwischen dem eingestellten Wert für SteeringVal und dem tatsächlich vorhandenen Lenkwinkel zu erstellen, wurden für einen Satz an verschiedenen Lenkwinkeln Messungen bezüglich des tatsachlichen Wendekreises angestellt, und mit Hilfe von einer Analysesoftware eine Formel für den Bezug zwischen den Werten erstellt.

Die Methodik für die Ermittlung des Wenderadius gestaltete sich wie folgt: Der Lenkwinkel des Fahrzeugs wurde zuerst auf einen bestimmten Wert eingestellt. Als nächstes ließ man das Fahrzeug einen Bogen von 90° mit einer Rechtskurve abfahren. Nachdem das Fahrzeug diese Strecke zurückgelegt hatte, wurde die Distanz zwischen der Ursprungsposition des rechten Hinterreifens und der Endposition des rechten Hinterreifens gemessen. Es wurde bei der weiteren Berechnung nun davon ausgegangen, dass der Weg den das Fahrzeug zurückgelegt hat ein Segment eines Kreises darstellt. Unter dieser Annahme kann man nun mit der Distanz zwischen ursprünglicher Radposition, und neuer Radposition, und dem Lenkradius gesehen von der Ursprungsposition und der Endposition ein gleichschenkliges rechtwinkliges Dreieck aufstellen. In dieser Form ließ sich nun die Beziehung zwischen der zurückgelegten Diagonale D und dem Lenkradius LR in die Formel des Pythagoras einsetzen:

$$D^2 = LR^2 + LR^2$$

Da es bei dieser Messung jedoch darum ging den Lenkradius des Fahrzeugs zu ermitteln, wurde diese Formel so umgestellt, dass sich ein Wert für LR berechnen lies:

$$LR = \sqrt{\frac{D^2}{2}}$$

Die gesamte Messreihe ist in folgender Tabelle zu sehen. Die berechneten Werte für den Lenkradius wurden ebenfalls mit in der Tabelle aufgeführt. Die Spanne der eingestellten SteeringVal's wurde nach oben begrenzt durch den maximalen physikalisch erreichbaren Stellwinkel. Nach unten wurden die Werte dadurch begrenzt, dass der verfügbare Testraum nicht ausreichte um einen vollen Bogen von 90° abzufahren.

Tabelle 1: Wenderadius

| **Steering Wert** | **Diagonale Distanz (cm)** | **Errechneter Wenderadius (cm)** |
|---|---|---|
| **50** | 105 | 74,2 |
| **45** | 120 | 84,9 |
| **40** | 134 | 94,8 |
| **35** | 151 | 106,8 |
| **30** | 183 | 129,4 |
| **25** | 233 | 164,8 |
| **20** | 311 | 219,9 |
| **15** | 434 | 306,9 |
| **10** | 694 | 490,7 |

Als nächstes musste eine mathematische Beziehung zwischen den beiden aufgestellt werden. Dies war nötig um den jeweiligen Lenkradius für die restlichen SteeringVal's zu interpolieren. Durch eine Reduzierung auf eine mathematische Formel wurden ebenfalls eventuell vorhandene Messfehler bei den einzelnen Werten geglättet. Um die besagte Formel zu erstellen wurden die einzelnen Datenpunkte nun in ein Graphen- und Analyseprogram eingegeben.

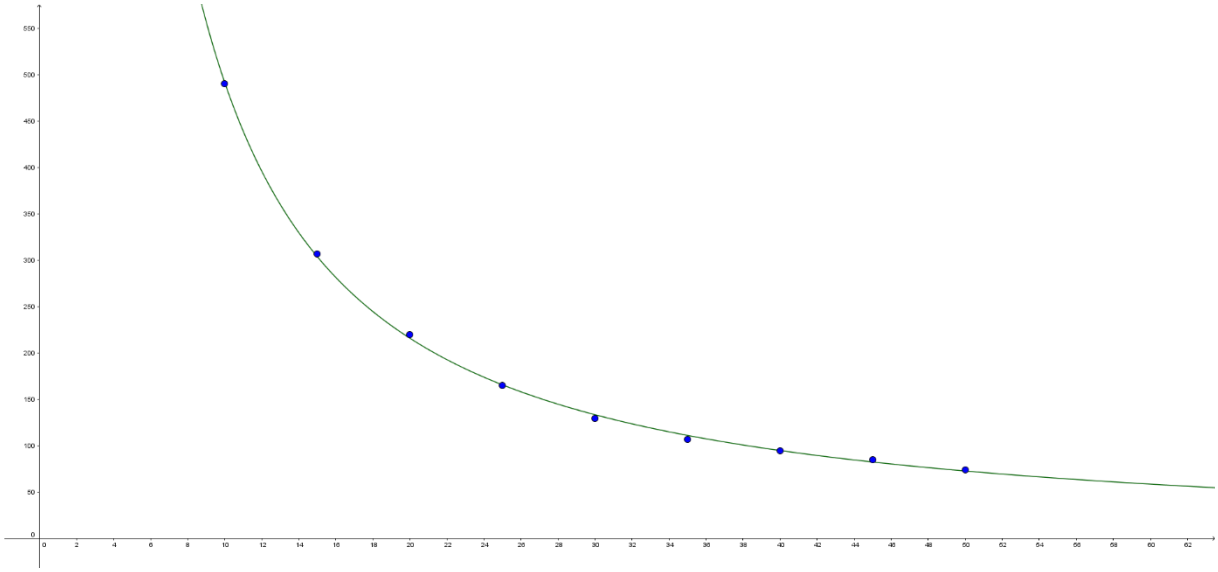

Abbildung 4: Graph Wenderadius

Allein von der Verteilung der Punkte ließ sich das Feld der möglichen Funktionen auf eine Potenzfunktion mit einem negativen Exponenten verkleinern. Und in der Tat ergab die Interpolation mit besagter Potenzfunktion ein Ergebnis mit einer sehr hohen Genauigkeit. Die Funktion die für den eingestellten Wert von SteeringVal den dazugehörigen tatsächlichen Lenkradius in cm berechnet gestaltete sich nun wie folgt:

$$LR = 7582{,}255 * SteeringVal^{-1{,}187622}$$

Als nächstes wurde leider festgestellt, dass der am Fahrzeug angebrachte Hall-Sensor nur bedingt brauchbare Werte für die tatsächliche Geschwindigkeit des Fahrzeuges lieferte. Daher wurde beschlossen stattdessen ebenfalls die Beziehung zwischen der Fahrzeug-Geschwindigkeit und dem dafür eingestellten Wert per Messung zu ermitteln. Die Methodik hierfür gestaltete sich im Vergleich relativ einfach: Es wurde ein bestimmter Wert für die Geschwindigkeit eingestellt. Bei diesem Wert ließ man das Fahrzeug eine abgemessene Strecke mehrmals abfahren, und stoppte jedes Mal die Zeit, die dafür gebraucht wurde. Die Zeitwerte wurden dann gemittelt, und daraus eine Geschwindigkeit errechnet. Nachdem dieser Prozess für jede Geschwindigkeitsstufe durchgeführt wurde ergaben sich folgende Ergebnisse:

Tabelle 2: Geschwindigkeit

| Speed Level | Zeit (s) | Berechnete Geschwindigkeit (m/s) |
|---|---|---|
| **1** | 37,8 | 0,103 |
| **2** | 23,4 | 0,167 |
| **3** | 13,1 | 0,298 |
| **4** | 8,3 | 0,453 |
| **5** | 6,3 | 0,619 |
| **6** | 4,4 | 0,886 |
| **7** | 3,2 | 1,218 |
| **8** | 2,4 | 1,625 |

## 7.3 Vermessung von Markern aus dem Kamerabild

Die erste eigentliche Aufgabe der Odometrie ist es die durch die Kamera erkannten Marker räumlich zu platzieren. Hierzu musste eine Methode entwickelt werden, die aus den von der Kamera gelieferten Informationen, d.h. in diesem Fall einer Pixel-Position innerhalb des Bildes, einen physikalischen Abstand in Relation zum Fahrzeug gewinnt. Der hierfür verwendete Ansatz beruht wieder auf dem Satz des Pythagoras. Es wird hier die Annahme getroffen, dass Pixel an verschiedenen Stellen des gelieferten Kamerabildes zwar verschiedene tatsächliche Entfernungen oder Größen unterschlagen, jeder Pixel aber jedoch einen Gleichgroßen räumlichen Winkel abdeckt. Zusätzlich musste noch die Entfernung des Fahrzeugs zu der unteren Kante des von der Kamera abgedeckten Blickfeldes gemessen werden. Mit Hilfe dieser Informationen ist es nämlich möglich sowohl die Distanz eines Objekts vor dem Fahrzeug, als auch dessen Versetzung links oder rechts der Mittellinie des Fahrzeuges zu berechnen.

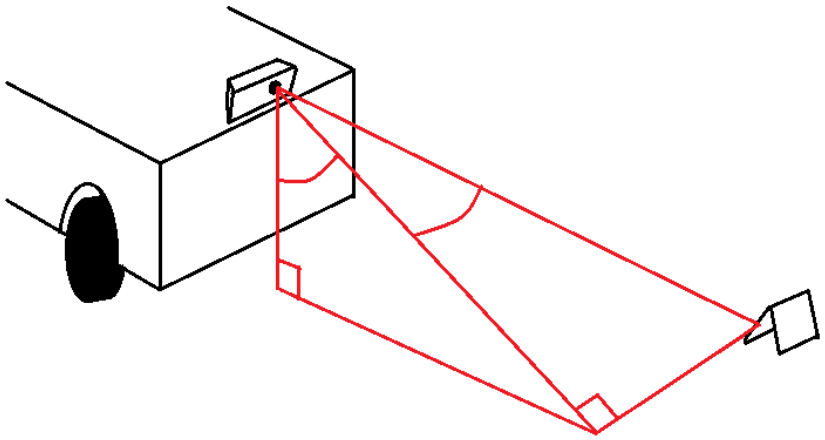

Abbildung 5: Blickfeld des Fahrzeugs

Hierzu war es allerdings nötig, wie bereits erwähnt, den Winkel der durch einen Pixel abgedeckt wird zu wissen. Der erste Gedanke war hier, in den Hersteller-Informationen der Kamera die besagten Informationen ausfindig zu machen. Nach längerer Suche zeigte sich jedoch, dass diese Informationen nicht ohne weiteres auffindbar waren. Daher wurde auch hier wieder eine Messung bezüglich der Werte durchgeführt. Dazu wurden in drei verschiedenen Entfernungen vor dem Fahrzeug jeweils drei Marker aufgestellt. Einer am äußeren rechten Rand des Blickfeldes, einer mittig im Blickfeld, und einer am äußeren linken Rand des Blickfeldes. Dabei wurde für jeden Marker die X und Y Koordinate des Pixels (mittlere Koordinate des Markers) notiert.

Tabelle 3: Entfernung

| **Entfernung (cm)** | | **Pixel** | |
|---|---|---|---|
| **Vor** | Links/Rechts | X | Y |
| **30** | 0 | 348 | 370 |
| **30** | 12L | 50 | 375 |
| **30** | 11R | 592 | 369 |
| **60** | 0 | 332 | 195 |
| **60** | 24L | 29 | 202 |
| **60** | 22,5R | 609 | 193 |
| **90** | 0 | 318 | 125 |
| **90** | 35L | 23 | 134 |
| **90** | 35R | 621 | 119 |

Mit diesen Messungen war es nun möglich den von einem Pixel unterschlagenen Winkel zu berechnen. Dazu wurden die Y Pixel-Koordinaten der Marker mit derselben Entfernung vor dem Fahrzeug zusammengerechnet um einen Mittelwert zu errechnen. Dann wurde wieder für jeden berechneten Mittelwert mit Hilfe der Höhe der Kamera über dem Boden eine Dreiecksbeziehung nach Pythagoras aufgestellt. Da man nun für drei verschiedene Werte sowohl die Pixel Koordinaten, als auch die Winkel von dem Marker zur Kamera in Relation zu der vertikalen hatte, konnte man zwischen zwei verschiedenen Markern den Pixel- und Winkelunterschied berechnen. Durch Kombination dieser beiden Werte ergab sich nun ein durchschnittliches Verhältnis von 0,0803° pro Pixel.

## 7.4 Positionsaktualisierung ohne Kamerabild

In Abwesenheit eines Kamerabildes von einem bereits erkannten Objekt wird es jedoch bedeutend schwerer dessen Position zu aktualisieren, wenn sich das Fahrzeug bewegt. Bei Fahren ohne Lenkeinschlag reicht es noch nach jedem Zeitabschnitt die Y-Koordinate proportional zur Geschwindigkeit herabzusetzen. Sobald sich das Fahrzeug jedoch mit einem eingeschlagenen Lenk-Winkel bewegt, gestaltet sich der Prozess bedeutend aufwändiger. Der Grund hierfür ist, dass sich bei der Kur-

venfahrt nicht nur die Position, sondern auch die Richtung des Fahrzeuges verändert. Damit ändert sich natürlich auch die relative Position und Richtung aller Objekte. Die Neuberechnung der Positionen wurde in mehrere logische Unterabschnitte aufgeteilt. Diese werden anhand des hypothetischen Punkts P illustriert:

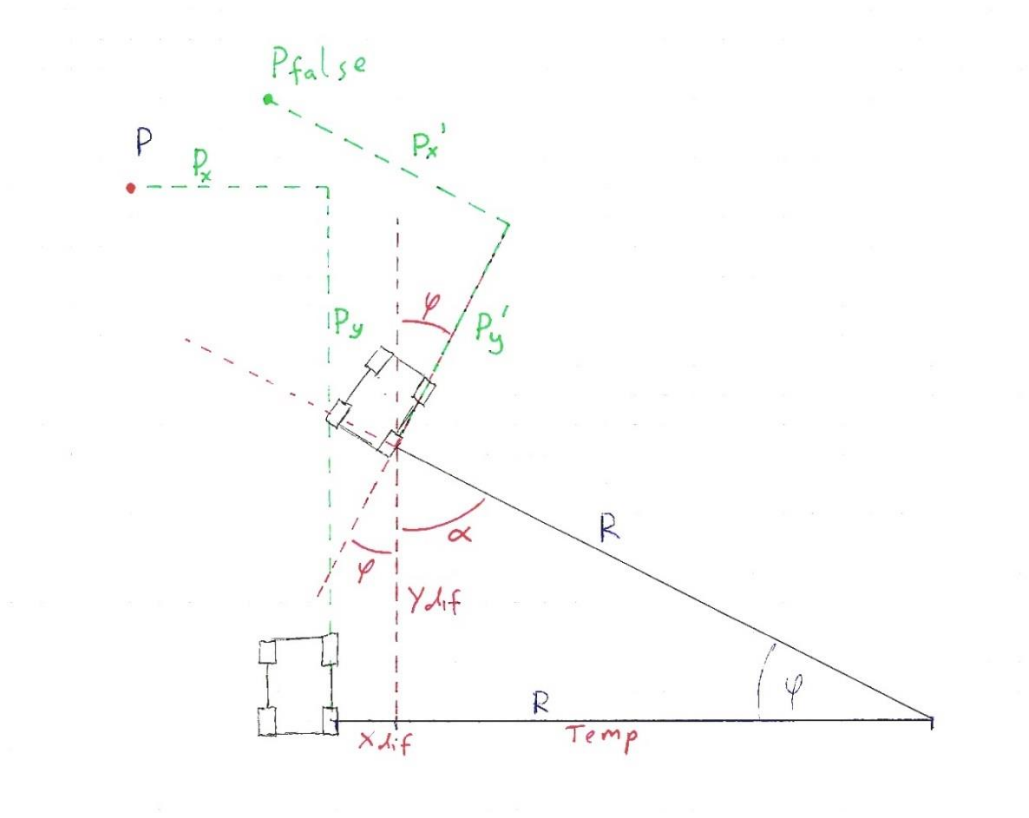


Abbildung 6: Umrechnung von Koordinaten

Vor der Positionsveränderung des Fahrzeugs hat der Punkt P die Entfernung Y vor dem Fahrzeug und Z Links der Mittellinie des Fahrzeugs. Nun durchfährt das Fahrzeug ein Segment des Wendekreises mit dem Radius R das durch den Winkel φ eingeschlossen wird. Um die weitere Berechnung zu vereinfachen werden die relativen Koordinaten des Punkts P auf das innere Rad umgerechnet. Es ergeben sich $P_y$ und $P_x$.

Als nächstes wird die X- und Y-Verschiebung des Bezugsrades berechnet. Die Werte $X_{dif}$ und $Y_{dif}$ die hier gesucht werden, lassen sich dadurch ermitteln, dass die Distanz R und der Winkel φ bekannt sind. Das durch die Anfangsposition, Ausgangsposition und den Mittelpunkt des Wendekreises aufgespannte Dreieck wird nun geteilt. Durch die Verwendung mehrerer Umformungen wird der Winkel α bestimmt. Nun benutzt man die Trigonometrischen Identitäten um $Y_{dif}$ und Temp zu bestimmen, und verwendet Temp um $X_{dif}$ zu berechnen.

Als nächste werden $X_{dif}$ und $Y_{dif}$ verwendet um $P_y$` und $P_x$` zu berechnen. Diese neuen Positionen beachten jedoch noch nicht das sich bei dem Durchfahren der Kurve auch die Orientierung des Fahrzeuges geändert Hat. Daher deuten diese Werte noch auf den Punkt $P_{false}$. Um dies zu korrigieren muss der Punkt P noch um φ Grad entgegen der Fahrtrichtung rotiert werden. Um dies zu erreichen

wird erst eine Koordinatentransformation auf Polarkoordinaten durchgeführt. Dies dient Vereinfachungszwecken, da man nun den Winkel φ direkt auf den durch die Polarkoordinaten gegebenen Winkel aufrechnen Kann.

Damit wurde die neue Position des Punkts P relativ zum Fahrzeug korrekt berechnet. Nun muss nur noch die Koordinatentransformation rückgängig gemacht werden, und die Koordinaten wieder relativ zur Mitte des Fahrzeugs normiert werden. Dieser Prozess wird anschließend für jedes gespeicherte Objekt wiederholt, und somit die gesamte Weltkarte aktualisiert.

# 8 ROS-Packages

## 8.1 car_handler

Das Package car_handler stellt die Schnittstelle zum Microcontroller dar. Hier werden die Sensordaten vom Fahrzeug an die anderen ROS-Nodes verschickt und im Gegenzug die Nachrichten der anderen Nodes zu Geschwindigkeit und Lenkeinschlag an den Microcontroller weitergeleitet.

Vor dem Versenden werden die Ultraschallsignale geprüft und gepuffert. Die Prüfung ist notwendig, weil zu viele falsche Werte geliefert werden. Nachdem wir uns einen Überblick über die möglichen Werte verschafft hatten, beschlossen wir alle Werte über 200 cm zu ignorieren. Die Sensoren lieferten nie Werte über 200, die korrekt gewesen wären. Die falschen Werte befinden sich sogar meist im vierstelligen Bereich. Der zusätzliche Puffer speichert die letzten sechs Werte. Der Durchschnitt, ohne den kleinsten und größten Wert, wird an die anderen Nodes gemeldet. Dieses Vorgehen verringert den Einfluss von Fehlern im Bereich bis 200 cm.

Zur Steuerung der Geschwindigkeit leitet der car_handler nicht einfach den Wert des speed-Topics an den Microcontroller weiter, sondern horcht auch auf das Topic topSpeed, welches eine vorgegebene Höchstgeschwindigkeit darstellt. Damit andere Nodes nicht speed und topSpeed verfolgen müssen um die wirklich eingestellte Geschwindigkeit zu ermitteln, sendet car_handler über das Topic speed_actual die wirklich eingestellte Geschwindigkeit, die auch an den Microcontroller gesendet wurde.

## 8.2 wall_follower

Grundlage für das Fahren im Flur war für uns die Orientierung an der Wand. Dazu muss das Fahrzeug über die seitlichen Ultraschallsensoren die Entfernung zur Wand ermitteln und diesen Abstand möglichst konstant halten. Um das zu erreichen wurde ein PID-Regler[6] implementiert und empirisch mit Werten bestückt. Unsere Tests zeigten, dass ein Satz an Werten nicht für alle Geschwindigkeiten genutzt werden kann. Der nächste Schritt war also, eine Liste mit vordefinierten Werten anzulegen, so dass bei Geschwindigkeitsänderungen die neuen Werte geladen und das Fahrzeug weiter geregelt werden kann.

Mit einem P-Wert von 7 und einem D-Wert von 6 erzielten wir bei den Geschwindigkeitsstufen 1 bis 4 gute Ergebnisse. Erst bei Stufe 5 mussten wir P auf 3 verändern. Für höhere Geschwindigkeiten

[6] https://de.wikipedia.org/wiki/Regler#PID-Regler

und Rückwärtsfahren wurde keine Werte ermittelt, da wir dafür in unseren Szenarien keine Verwendung hatten. Der I-Wert kann immer bei 0 gehalten werden, so dass es sich eigentlich nur um einen PD-Regler handelt.

Der wall_follower dient aber nicht nur dem „an der Wand entlang fahren“. Er muss auch Kreuzungen und Engstellen meistern können. Zu diesem Zweck horcht er auf das Topic „direction“, welches von signs übertragen wird. Hier wird mitgeteilt, wie sich das Fahrzeug an der nächsten Kreuzung verhalten soll (abbiegen oder geradeaus) oder ob wegen einer Engstelle die Regelung angepasst werden muss.

Signs sendet direkt vor der Kreuzung oder Engstelle den entsprechend definierten Code aus arucopos\include\arucopos\myMarker.h. Also beispielsweise AR_SIGN_DIR_L zum links abbiegen oder AR_SIGN_NARROW_MIN, wenn eine Engstelle zu meistern ist. Außer dem rechts abbiegen muss dazu in allen Fällen der PID-Regler angepasst werden. So muss beim Geradeausfahren über eine Kreuzung der Regler abgeschaltet werden, sobald das Fahrzeug sich auf der Kreuzung befindet. Erst wenn das Fahrzeug feststellt, dass sich neben ihm wieder eine Wand befindet (die Wand kann rechts, oder links, oder auf beiden Seiten verschwinden!) wird der Regler wieder eingeschaltet und die Fahrt normal fortgesetzt.

Beim Linksabbiegen orientiert sich das Fahrzeug an der linken Wand. In dem Moment, wenn die Richtungsanweisung von signs kommt, wird der Abstand zur linken Wand ermittelt und ab diesem Zeitpunkt auf diesen Wert geregelt bis die Kurve durchfahren ist. Mangels vernünftiger Alternativen griffen wir hier darauf zurück die Zeit vorzugeben nach der wieder auf die rechte Wand umgeschaltet wird.

Engstellen sind so definiert, dass sowohl rechts als auch links ein Hindernis den Weg einengt und das Fahrzeug dazu zwingt sich in der Mitte der Straße zu bewegen. Hier wird der Regler so geschaltet, dass er versucht den Abstand auf der linken und rechten Seite gleich zu halten bis das Signal für das Ende der Engstelle kommt. Da das Fahrzeug selbst das Ende nicht feststellen kann, ist dies die bislang einzige Ausnahme in der wall_follower nicht selbstständig wieder zurückschaltet, sondern auf ein externes Signal wartet.

Wenn die Gefahrenstelle gemeistert ist und die Regelung wieder im Normalbetrieb arbeitet, sendet wall_follower über das Topic „c2cCrPassed“ das Signal zur Freigabe der Engstelle. C2c wird nun die Kreuzung oder Engstelle für das nächste Fahrzeug freigeben.

## 8.3 arucopos

Durch die Entscheidung ArUco-Marker sowohl für die Tordurchfahrt, als auch für die Verkehrszeichen zu nutzen wurde ein zentraler Knoten notwendig, der die Erkennung und Berechnung der Marker-Positionen übernimmt. Arucopos erledigt diese Aufgabe und sendet eine Liste mit allen jemals erkannten Markern, die noch nicht hinter das Fahrzeug zurückgefallen sind. Zur Übermittlung wurden im Ordner arucopos\msg zwei Nachrichtentypen definiert. Einmal eine Struktur für eine Markerposition (sArucoPos.msg) und einmal ein Vektor aus diesen Strukturen (vArucoPos.msg). Details zur Berechnung finden sie im Abschnitt Odometrie.

Es war anfangs nicht klar, wie gut diese Berechnung funktionieren würde. Vor allem vor dem Hintergrund, dass die ermittelten Daten für Lenkung und Geschwindigkeit Schwankungen und Mess-

fehlern unterlagen war dieses Teilprojekt ein großes Wagnis. Aber das Endergebnis spricht für sich und die viele Arbeit hat sich gelohnt. Die anderen Knoten können sich blind auf die Entfernungsangaben verlassen und das Fahrzeug präzise manövrieren.

## 8.4 signs

Aus der Liste von Markern, die arucopos versendet, werden die herausgefiltert, die für die Verkehrsregelung relevant sind. Grundlage dafür ist die Header-Datei „myMarker.h“, welche zum Paket arucopos gehört.

Erkannte Verkehrszeichen führen nicht sofort zu einer Reaktion. Sie werden verfolgt, bis das Fahrzeug sie erreicht hat. Erst dann werden sie behandelt und gegebenenfalls eine Nachricht an andere Nodes gesendet. Dazu wird, parallel zur Marker-Liste die von arucopos kommt, eine Liste mit behandelten (publizierten) Markern angelegt. Es wird sofort nach der Behandlung des Verkehrszeichens die ID des Markers im Vektor pubSigns gespeichert. Aus dieser Liste werden die Marker erst entfernt, wenn arucopos sie nicht mehr führt (sprich: sie sind hinter das Auto zurückgefallen). Erst ab diesem Zeitpunkt kann das selbe Verkehrszeichen neu behandelt werden.

Die Behandlung an sich hängt vom Verkehrszeichen ab. Schilder für Höchstgeschwindigkeit werden zum Beispiel ohne Umwege direkt an den car_handler kommuniziert. Genutzt wird das Setzen der Höchstgeschwindigkeit auch, um vor Kreuzungen das Tempo zu drosseln und gegebenenfalls stehen zu bleiben.

Bei Kreuzungen und Engstellen hingegen muss eine Car-2-Car-Absprache eingeleitet werden, bevor weitere Maßnahmen getroffen werden können. Die Struktur sC2CProg speichert in dem Fall alle Details zu einer aktiven Absprache. Es kann zu jedem Zeitpunkt nur eine Absprache laufen, was aber auch nicht stört, da zwei Kreuzungen und/oder Engstellen sich niemals überschneiden. c2cInit() teilt dem c2c-Node mit, dass eine Absprache mit anderen Fahrzeugen getroffen werden soll und übergibt ihm alle notwendigen Details zum aktuellen Standort und der geplanten Fahrtrichtung (Topic c2cReqPerm). Erst wenn die Erlaubnis zum Befahren der Gefahrenstelle vorliegt (Topic c2cPerm), wird die Höchstgeschwindigkeit wieder auf den Ausgangswert gesetzt und eine Nachricht an wall_follower gesendet um ihn über die Kreuzung/Engstelle zu informieren. Bei Kreuzungen wird nun wall_follower gemäß den Vorgaben die Kreuzung überqueren und dann das Ende der Kreuzung mitteilen und c2c die Freigabe ermöglichen. Im Fall von Engstellen läuft die Kommunikation etwas anders. Das Ende einer Engstelle kann vom wall_follower nicht ermittelt werden. Hier wurde ein spezielles Verkehrszeichen eingeführt um das Problem zu beheben. Daher sendet in dem Fall signs das Topic c2cCrPassed um die Gefahrenstelle freizugegeben.

## 8.5 c2c

Die Car-2-Car-Kommunikation wird nur benötigt, wenn Gefahrenstellen befahren werden sollen. Damit hier keine Unfälle passieren, erfolgt vorab eine Absprache der Fahrzeuge, so dass jeweils nur eines die Gefahrenstelle befährt und alle anderen warten. Die IDs der Marker wurden so gewählt, dass eine Engstelle immer eindeutig identifizierbar ist. Des Weiteren werden die Marker für Kreu-

zungen immer im Uhrzeigersinn angeordnet, so dass alle Fahrzeuge die Position der anderen bestimmen können und auch selbst erkennen, ob sie Vorfahrt haben.

Dazu sendet ein Fahrzeug A, sobald es eine Gefahrenstelle befahren möchte, das Topic globalCall mit den Details zu seiner Gefahrenstelle und seiner gewünschten Fahrtrichtung. Zur Absprache wurde die Struktur enterDangerZone entwickelt, die neben der Marker-ID der Gefahrenstelle und der Fahrtrichtung auch den aktuellen Status des Fahrzeugs beinhaltet. So steht Status 0 für „Freigaben der Engstelle“, Status 1 für „Ich fordere die Engstelle an“ und Status 2 für „Ich habe die Engstelle für mich gesperrt“.

Das Fahrzeug sendet also als erstes eine Nachricht mit Status 1. Sollte sich innerhalb eines gesetzten Zeitlimits kein anderes Fahrzeug melden, so sendet es eine Nachricht mit Status 2 und befährt die Gefahrenstelle.

Wenn innerhalb des Zeitlimits ein anderes Fahrzeug (Fahrzeug B) ebenfalls dieselbe Kreuzung befahren möchte (ebenfalls Status 1), dann wird noch einmal eine Nachricht mit Status 1 gesendet (Fahrzeug B hört erst jetzt zu) und prüft sofort, wer von beiden Vorfahrt hätte. Fahrzeug A erkennt, dass es Vorfahrt gewähren muss und sendet Status 0. Intern wird ein Timer gestartet, um reagieren zu können falls B die Gefahrenstelle nie freigibt. Fahrzeug B erkennt während dessen, dass es Vorfahrt hat und sendet Status 2. Nach Durchfahren der Gefahrenstelle gibt B sie wieder frei und sendet Status 0. Fahrzeug A sendet nun erneut Status 1 und wartet wieder, falls inzwischen neue Fahrzeuge eingetroffen sind.

Da zum Zeitpunkt der Fertigstellung zwar zwei Fahrzeuge zum Testen zur Verfügung standen, aber bei beiden die Ultraschallsensoren nicht funktionsfähig waren, mussten wir den Test im Labor durchführen. Dazu stellen wir zwei Fahrzeuge auf Böcke, damit wir die Reifen frei drehen lassen konnten. Die beiden Fahrzeuge wurden mit einem Crossover-Kabel verbunden um Störungen des Netzwerks auszuschließen. Im realen Einsatz hätten wir auf ein Ad-hoc-Netzwerk gesetzt, da die Verbindung über Eduroam in den Fluren des Gebäudes nicht stabil genug ist. Den beiden Fahrzeugen wurden dann die Marker für Kreuzungen gezeigt und die Reaktion beobachtet.

## 8.6 gates

Zum Durchfahren von ArUco-Marker-Toren muss man zuerst einmal Tore definieren. Wir entschieden, dass ein Tor aus zwei Markern besteht. Einer für den linken und einer für den rechten Pfosten. Um die Erkennungsrate zu erhöhen wurden sie halb aufrecht hingestellt. Die IDs wurden so gewählt, dass ein Marker mit gerader ID immer den linken Pfosten und ein Marker mit ungerader ID immer den rechten Pfosten darstellt. So benötigt das Fahrzeug lediglich einen der Marker um zu wissen wie es an ihm vorbeifahren muss. Des Weiteren bilden eine gerade und die nächst höhere ID immer ein Tor, welches im Szenario einmalig sein muss.

Es kann passieren, und das ist leider sehr häufig der Fall, dass der eingeschränkte Sichtbereich der Kamera nur einen Pfosten erkennt. Die Position des anderen wird dann „geraten“. Dazu nimmt gates an, dass der zweite Pfosten dieselbe Entfernung hat und 25 cm zur jeweiligen Seite vorschoben ist. Weniger als 25 cm sollten hier nie eingestellt werden. Bei einer Spurbreite von 18,5 cm plus etwa anderthalb cm für den Marker ist sonst der Sicherheitsabstand zu gering. Grundsätzlich könnte man auch einen sehr großen Abstand wählen und das Fahrzeug damit zwingen stark zu lenken und den zweiten Pfosten in den Sichtbereich zu bekommen. Das wilde Lenken und die Gefahr, dass der Mar-

ker inzwischen zu nah am Fahrzeug ist um ihn zu erkennen führten dazu, dass wir uns für die Variante mit dem geringen Abstand entschieden.

Für die Logik hatten wir zwischenzeitlich geplant eine sauber S-Kurve zu fahren und ein Tor möglichst Ideal zu durchfahren. Der eingeschränkte Sichtbereich der Kamera und die Tatsache, dass aufgestellte Marker nur in einem eng begrenzten Bereich von der Seite erkannt werden können, zeigten uns aber, dass wir keine Szenarien haben sollten in denen eine S-Kurve einen Vorteil bringt. Vielmehr sehen unsere Szenarien so aus, dass die Tore dicht hinter einander stehen und immer nur eine leichte Änderung der Fahrtrichtung mit sich bringen. Aus diesem Grund konnten wir für die Routenfindung auf die einfachst mögliche Logik zurückgreifen und das Fahrzeug auf die Mitte des Tores zielen lassen. Viel wichtiger war die möglichst korrekte Positionsberechnung und Verfolgung von aus dem Blick verlorenen Markern, was arucopos sehr gut erledigt.

Gates empfängt die Marker-Liste von arucopos und filtert alle Tor-Marker heraus. Diese müssen dann in einer internen Liste gespeichert werden, weil sie zu Toren zusammengefasst und gegebenenfalls um geratene Pfosten ergänzt werden müssen. Da für die Tordurchfahrt außer arucopos und car_handler keine weiteren Nodes notwendig sind, sendet gates seine Steuerbefehle direkt an car_handler. Selbst das Tempo wird hier bestimmt. Sobald ein Tor-Marker erkannt wird, wird die vordefinierte Geschwindigkeit eingestellt. Wenn das letzte Tor durchfahren ist, dann wird die Geschwindigkeit automatisch wieder auf 0 gestellt.

## 8.7 Webinterface

### 8.7.1 Implementierung

Das Webinterface besteht aus zwei grundlegenden Komponenten: Eine Webseite als Client und eine Python-Anwendung als Server. Die Webseite kommuniziert mit der Python-Anwendung über HTTP, um ROS Topics zu veröffentlichen und um alle verfügbaren ROS Topics aus einer Datenbank zu laden. Die Python-Anwendung stellt einen Webserver, Datenbankzugriffe und eine Anbindung zu den ROS Topics zur Verfügung. Wenn nun ein Benutzer mittels dem Webinterface ein ROS Topic veröffentlicht, wird dieses per HTTP an die Python-Anwendung gesandt, welche wiederrum über ein Interface zum ROS Master letztendlich das ROS Topic veröffentlicht. Auf Client sowie Server kommen diverse Frameworks zum Einsatz. Im Client wird das Javascript-Framework AngularJS verwendet. Dies ermöglicht eine einfache Programmierung der Webseite seitens des Client. Im Server wird das Microframework Flask verwendet. Es bietet zahlreiche Funktionen wie einen Webserver, Datenbankanbindung, Laden von Konfigurationsdaten, Vorlagen und weitere.

### 8.7.2 Funktion

Die Funktionen des Webinterfaces belaufen sich auf folgende: ROS Topic veröffentlichen, neues ROS Topic zum Webinterface hinzufügen, Fernsteuerung für mobile Geräte mit Gyroskop.

Ein ROS Topic kann auf verschiedene Arten veröffentlicht werden. Hier ist es möglich einen Schieber zu aktivieren mit dem man den gewünschten Wert einstellen kann. Jederzeit ist auch die manuelle Eingabe eines numerischen Wertes möglich. Dabei kann gewählt werden, ob der Wert direkt

veröffentlich werden soll oder erst auf Wunsch des Benutzers. Möchte man ein neues ROS Topic zum Webinterface hinzufügen, so wird dieses automatisch in einer Datenbank gespeichert. Zuletzt gibt es noch eine Möglichkeit den Lenkwinkel und die Geschwindigkeit über das Gyroskop eines mobilen Gerätes zu steuern. Hierzu werden die benötigten Daten über ein HTML5 Objekt erfasst.

### 8.7.3 Verbesserungsmöglichkeiten

Das Webinterface kann derzeit nur ROS Topics veröffentlichen, sinnvoll wäre es ebenfalls ROS Topics zu empfangen. Mit diesen empfangenen Daten kann man nun Graphen erstellen, die helfen das System besser zu verstehen. Ebenfalls denkbar wäre eine Aufnahme der aktuellen Werte inklusive eines Kommentars. Dies kann hilfreich sein um z.B. einen PID-Regler manuell einzustellen.

## 8.8 sinnvolle Kombinationsmöglichkeiten

Da car_handler die Schnittstelle zum Microcontroller darstellt, ist ein Betrieb des Fahrzeugs ohne dieses Package unmöglich. Aus diesem Grund enthält dieses Package auch die Launch-Files, welche die sinnvollsten Kombinationen der Packages automatisch starten.

car_handler.launch startet lediglich den car_handler und dient Testzwecken und als Grundlage für das Webinterface. Hier können die Sensoren vom Fahrzeug überprüft werden und Tests durchgeführt werden, ohne dass andere Nodes stören könnten.

this_car.launch startet die Nodes uvc_camera, car_handler, signs, arucopos, wall_follower und c2c. Diese Sammlung ist für die Verkehrssimulation gedacht. Sobald man eine Wunschgeschwindigkeit an das Fahrzeug sendet, wird es an der Wand entlangfahren und auf Verkehrsschilder reagieren. An Gefahrenstellen wird die Car-2-Car Kommunikation gestartet und, falls andere Fahrzeuge unterwegs sind, eine Absprache getroffen.

gates.launch startet die Nodes uvc_camera, car_handler, arucopos und gates. Das Fahrzeug wird, sobald es Tor-Marker erkennt, losfahren und die Tore durchfahren. Nach Durchfahrt des letzten erkannten Tores wird das Fahrzeug automatisch anhalten.

# 9 Verbesserungsvorschläge

Das Thema „Wie könnte man die Plattform verbessern/weiter ausbauen“ muss differenziert betrachtet werden.

Mit dem Blick auf das Projektseminar und bei gleichen Anforderungen für die Studenten (sprich: Fahrt durch den Flur) könnte man über weitere Ultraschallsensoren nachdenken. Jeweils zwei pro Fahrzeugseite würden es erlauben den Winkel des Fahrzeugs zur Wand zu ermitteln und damit eine deutlich bessere Steuerung ermöglichen. Der Liniensensor wurde von uns nicht genutzt, weil wir keinen Einsatz dafür finden konnten. Den Hall-Sensor hingegen hätten wir gerne genutzt um die Geschwindigkeit zu ermitteln, er lieferte aber keine zuverlässigen Daten. Die Kamera ist ausreichend, stellt nur in ihrem eingeschränkten Sichtbereich eine Einschränkung dar. Wenn man das Szenario daran anpasst, dann kann man gut damit leben. Die Leistung des Mini-PCs und des

Microcontroller und auch die Kommunikation zwischen den beiden waren für unser Projekt mehr als ausreichend.

Wenn man aber das Ziel Carolo-Cup verfolgt, dann müsste man das Auto in eine andere Richtung entwickeln. Man müsste deutlich mehr Fokus auf die Kamera und Objekterkennung legen. Die Ultraschallsensoren werden dagegen kaum noch einen Einsatzzweck haben, da im Cup keine Wände sind denen man folgen könnte. Auch der Liniensensor ist keine Option. Man muss zwar zum Beispiel vor einer Linie an einem Stopp-Schild halten, aber die Linie muss sich VOR dem Auto befinden. Außerdem könnte man sich überlegen den Flur oder eine freie Halle mit den Fahrstreifen des Cups zu versehen um die Erkennung der Fahrspuren über die Kamera zu testen.

## 10 Fazit

Das Fahrzeug bietet eine gute Grundlage für eigene Projekte. Schlussendlich sind wir froh, dass das Projektseminar nun endlich zu Ende ist. Nicht, weil es keinen Spaß gemacht hätte, sondern weil man beliebig viel Zeit investieren kann und trotzdem nicht „fertig“ ist. Wir konnten einen Großteil unserer Ziele erreichen. Am Ende wurde es etwas knapp und wir mussten ein Teilprojekt abbrechen, das die Nodes alle verbunden hätte und für einen fließenden Übergang von Verkehrssimulation, Tordurchfahrt und Fernbedienung gesorgt hätte. Trotzdem sind wir mit dem Ergebnis zufrieden und glauben eine gute Basis geschaffen zu haben.

Ist die Plattform für das Projektseminar geeignet? Sicher. Man kann verschiedene Szenarien antesten und sich grundsätzlich mit ROS und Sensorik beschäftigen. Natürlich ist die Plattform nicht perfekt, aber gibt es das überhaupt? Für einige Szenarien ist es sicher besser geeignet als für andere, aber grundsätzlich kann man fast alles realisieren, auch wenn man sich manchmal etwas zu helfen wissen muss.

Würden wir es als Basis für den Carolo-Cup einsetzen? Schwierig zu beantworten. Grundsätzlich benötigt man nur das Chassis mit Elektromotor und Akku. Für alle andere Anbauteile müsste man sich das Regelwerk und die Fahrszenarien ansehen und gezielt darauf hin entwickeln. Also: Ja, man könnte das Chassis nutzen, müsste aber beispielsweise die Kamera deutlich niedriger anbringen um den Nahbereich des Fahrzeugs erkennen zu können.