

# R O B O T I K   P R A K T I K U M

Wintersemester 2017/ 2018

## Dokumentation - Gruppe 1

tin101030 - Julian Finck

tin101364 - René P. Keller

inf101301 - Lea Morschel

tin101313 - Adrian Sorge

tin101087 - Julian Weihe

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Zielsetzung	3
<b>2</b>	<b>Grundlagen</b>	<b>5</b>
2.1	Hardwareplattform	5
2.1.1	Grundkomponenten	5
2.1.2	Rechner	5
2.1.3	Aktoren	5
2.1.4	Sensoren	5
2.2	Softwareplattform	6
2.2.1	ADTF-Framework: Grundlegende Strukturen	6
<b>3</b>	<b>Implementierung</b>	<b>7</b>
3.1	Umgebung	7
3.2	Projektstruktur	7
3.2.1	Linienerkennung und -verfolgung	7
3.2.2	Abstandserkennung und -verwertung	8
3.2.3	Motoransteuerung	11
3.2.4	Fusionsschnittstelle	11
3.3	Fallstricke: Komplikationen und Fehler	12
<b>4</b>	<b>Auswertung</b>	<b>13</b>
	<b>Abbildungsverzeichnis</b>	<b>14</b>

# 1 Einführung

Das Robotik-Praktikum, wessen Ergebnis diese Dokumentation festhält, fand parallel zur und in Kombination mit der Bachelor-Vorlesung 'Einführung in die Robotik' statt, welche von Herrn Prof. Dr. Ulrich Hoffmann im Wintersemester 2017 gehalten wurde.

Ziel war die Vertiefung und Anwendung der in der Vorlesung vermittelten Inhalte, wobei der diessemestrige Schwerpunkt auf autonomen Robotern lag. Im Praktikum wurde mit einem AADC<sup>1</sup> 2017 Audi-Modell im Maßstab 1:8 gearbeitet. Es sollte zu Beginn eine nicht näher eingeschränkte Zielsetzung ausgearbeitet werden, welche auf der grundlegenden Anforderung der kollisionsfreien Fahrt aufbauen sollte.

## 1.1 Zielsetzung

Im Kontext der gegebenen Grundanforderung der Kollisionsvermeidung wurden eigene Ziele definiert. Das Auto sollte später in der Lage sein, einer zentralen blauen Fahrbahnmarkierung zu folgen, dies unter verschiedenen Lichtverhältnissen. Bei Näherkommen eines von Ultraschallsensoren detektierbaren Hindernisses soll das Modell langsamer werden und vor ihm zum Stehen kommen sowie die Fahrt erst wieder aufnehmen, wenn die Sensoren signalisieren, dass die Fahrbahn wieder frei ist.

Die zu Beginn des Praktikums ausgearbeitete Roadmap findet sich in Abbildung ??.

---

<sup>1</sup>Audi Autonomous Driving Cup[1]

## Robotik Praktikum



Abbildung 1: Zu Projektbeginn erstellte Roadmap

Die Umsetzung des Projektes wurde zu Anfang in vier Bereiche unterteilt:

1. Linienerkennung und -verwertung
  - a) Mittelpunktbestimmung
  - b) Testen unter verschiedenen Lichtintensitäten
2. Abstandserkennung und Kollisionsvermeidung
  - a) Messwertverarbeitung der Ultraschallsensoren
  - b) Messwertglättung durch Filterung (Kalman)
3. Motorsteuerung
  - a) Schnittstellendefinition zur Ansteuerung
  - b) Steuerung von Geschwindigkeit und Lenkung
4. Zusammenbringung/ Fusionsschnittstelle
  - a) Definition und Ausarbeitung des Controllers (Verhaltens-Logik)
  - b) Problembehandlung, Feinabstimmung, Testung

## 2 Grundlagen

Im Folgenden werden die Grundlagen, bestehend aus Aufbau des genutzten Modellautos, der darauf laufenden Software, sowie das genutzte Framework erläutert, welche für das Verständnis des Projektes essenziell sind.

### 2.1 Hardwareplattform

Das Audimodell bietet verschiedene Hardwarekomponenten, um das autonome Fahren zu verwirklichen.

TODO: Bild vom Audi gesamt

#### 2.1.1 Grundkomponenten

Die Stromversorgung während der Fahrt wird durch zwei Akkus sichergestellt.

#### 2.1.2 Rechner

Herzstück des Audi ist das *GIGABYTE GA-Z170N-WIFI miniITX* Mainboard. Unterstützt wird es von einer *NVIDIA GeForce GTX 1050Ti*. TODO: Bild vom Audi, vor allem mit der Intel

#### 2.1.3 Aktoren

Angetrieben wird das Modell von vier *Hacker SKALAR 10 21.5 Brushless Motor 1/10*. Die Lenkung funktioniert über *Absima 'ACS1615SG' Combat Series* Servos.

#### 2.1.4 Sensoren

Die Linienerkennung arbeitet mit der Intelkamera, die mit einer anderen Position und einem anderen Winkel benutzt wird, als im Basismodell. Zum Vergleich siehe Abbild 1 und 2. Grund ist der verbesserte Blick auf den Nahbereich und das mittig ausgerichtete Bild.

TODO: Bild von den Sensoren und Arduinos

Zur Kollisionsvermeidung werden die verbauten Ultrasonic Sensoren verwendet. Ansteuern lassen sich diese über Arduinos, die mit dem Mainboard verbunden sind. Der Audi besitzt insgesamt zehn Stück, von denen fünf, in unterschiedlichen Winkeln, nach vorne, jeweils einer

zu jeder Seite und drei, wieder in unterschiedlichen Winkeln, nach hinten ausgerichtet sind. Die drei hinteren Sensoren bekommen nicht so viel Gewichtung, da das Auto in unserem Projekt nicht rückwärts fahren soll.

## 2.2 Softwareplattform

Das Modellauto verwendet die Ubuntu Version 16.04 und zur Bildschirmübertragung ist der x11vnc Serverdienst eingerichtet. Die Software EB Assist ADTF 2.14 von Elektrobit ist installiert, um die damit bereitgestellte Programmoberfläche und das ADTF-Framework <sup>2</sup> zu verwenden.

### 2.2.1 ADTF-Framework: Grundlegende Strukturen

Das Framework stellt Funktionen zur Kommunikation mit den im Abschnitt 'Hardwareplattform' erläuterten Sensoren und Kameras bereit. Außerdem ermöglicht es eine Programmaufteilung in Funktionsblöcke, *Filter* genannt, welche parallel auf mehreren Prozessorkernen ausgeführt werden können. Die Kommunikation zwischen diesen Blöcken wird über Nachrichtenqueues und Priorisierung vom Framework sichergestellt. Die Ausführung der Funktionsblöcke wird einstellbar periodisch eventgesteuert ausgelöst und beginnt meist mit dem Anstoß zum Auslesen der Sensoren, welche dann das Event über die Queues an andere Funktionsblöcke weitergeben.

Von der Benutzeroberfläche aus lassen sich die Funktionsblöcke grafisch miteinander verbinden, welches die Kommunikation zwischen den Blöcken repräsentiert, und die Ausführung der Funktionen starten. Des Weiteren gewährt die Oberfläche Zugriff auf fertige Funktionsblöcke, etwa zum Anzeigen und Abspeichern von Kamerabildern und Sensordaten oder zum Abspielen selbiger. Durch Referenzen aus dem Programm heraus können über die Benutzeroberfläche bestimmte Parameter zur Initialisierung und zur Laufzeit gesetzt werden.

---

<sup>2</sup>Automotive Data and Time-Triggered Framework[2]

## 3 Implementierung

### 3.1 Umgebung

Die Zielsetzung sieht vor, das Auto eine blaue Linie verfolgen zu lassen. Zu Testzwecken wurde daher auf dem Boden eine große, ovalförmige Linienstruktur mit mattblauem Klebeband befestigt, deren Maße etwa 1,5m x 3m beträgt. Zur Vermeidung von störender Spiegelung wurde zudem das Fenster zugestellt, um die Sonneneinstrahlung zu minimieren.

TODO: Bild(er) der Umgebung

### 3.2 Projektstruktur

Das Projekt wurde zu Beginn in verschiedene Aufgabenbereiche aufgeteilt, die separat bearbeitet wurden (siehe Unterabschnitt 1.1). Die vier großen Bereiche sind im Folgenden näher beschrieben.

1. Linienerkennung und -verfolgung
2. Abstandserkennung und -verwertung
3. Motorsteuerung
4. Fusionsschnittstelle

#### 3.2.1 Linienerkennung und -verfolgung

Damit das Auto der blauen Fahrbahnlinie folgen kann, wird die Linie im Kamerabild detektiert. Anschließend wird die Position der Linie ausgewertet um daraus Steuersignale für die Lenkung zu gewinnen. Diese Aufgaben werden von zwei Filtern gelöst:

**BlueImgFilter** Dieser Filter markiert die Fahrbahnlinie und greift dafür auf die OpenCv Bibliothek zurück. Weil die Kamera hauptsächlich den (hellgrauen) Boden unmittelbar vor dem Auto sieht, kann man da-von ausgehen, dass die Fahrbahnlinie im relevanten Suchbereich der einzig blaue Bildinhalt ist.

Zunächst wird das Eingangsbild mit OpenCv vom RGB- in den HSV-Farbraum transformiert, um die Festlegung von Ober- und Untergrenzen für Blauwerte zu vereinfachen. Mittels

der OpenCv-Funktion `inRange` werden anschließend alle nicht blauen Bildinhalte herausgefiltert und ein Binär-bild zurückgegeben, welches alle blauen Flächen markiert. Die endgültigen Farbwertgrenzen ergeben sich durch Testläufe unter Realbedingungen.

TODO: Abbildung 1 und 2 (Übernahme ich aus deiner Doku Julian)

**OneLineDetect** Dieser Filter erhält das Binärbild und ermittelt die Position der blauen Linie relativ zur Position des Autos. Dafür wird in einer festgelegten Bildzeile nahe des Unteren Bildrandes nach der größten Anzahl weißer, zusammenhängender Pixel gesucht. Durch die Aufbereitung des Bildes durch den `blueImgFilter` kann davon ausgegangen werden, dass diese Pixel zur blauen Linie gehören.

Anschließend wird die Mitte des weißen Pixelabschnittes berechnet und deren Abstand zur Bildmitte ausgegeben. Dieser Abstand wird auf einen Bereich zwischen -100 und 100 normiert. Über die Filterproperties lässt sich eine Mindestbreite für die Anzahl weißer Pixel festlegen um die Detektion noch robuster zu gestalten. Sind im aktuell zu bearbeitenden Bild keine oder zu wenig weiße Pixel am Stück, so wird statt einer Zahl zwischen -100 und 100 der Wert -101 ausgegeben.

Für Debugzwecke wird die detektierte Linienmitte mit einem roten Kreis markiert. Dieser Ansatz setzt voraus, dass das Kameraobjektiv auf der Mittelachse des Modelautos montiert ist.

TODO: Abbildung 3 (übernahme ich genauso)

### 3.2.2 Abstandserkennung und -verwertung

Die Abstandserkennung ist eine der wichtigsten grundlegenden Anforderungen an einen autonomen Roboter zur Ermöglichung von gefahrloser Navigation. In diesem Projekt wird sie auf Basis von Messwerten der zehn im Auto verbauten Ultraschallsensoren [TODO: Referenz auf USC-Modell-Bild aus Hardware-Doku-Part] implementiert.

Die eingehenden Abstandswerte liegen im Bereich von 0 bis 400 cm. Diese werden unterschiedlich gewichtet, dann der geringste Abstandswert in eine Geschwindigkeit umgerechnet. Die Übersetzung eines Abstands- in einen Geschwindigkeitswert wurde anhand einer linearen Funktion realisiert, die einen minimalen Input-Grenzwert zum Anstoßen der Bewegung beinhaltet und gleichsam einen maximalen Input-Wert, ab dem die volle Geschwindigkeit zurückgegeben wird (siehe Abbildung 2). Dies dient dazu, einen gewissen Mindestabstand beim Fah-



ren nicht zu unterschreiten, sowie zum Ignorieren von Hindernissen, die weiter als ein gewisser Maximalwert von den Sensoren entfernt aufgefasst werden. Letzterer wurde statisch auf 100 cm gesetzt. Im Abstands-Wertebereich von 40 bis 100 gilt somit die lineare Funktion  $y = x$ .

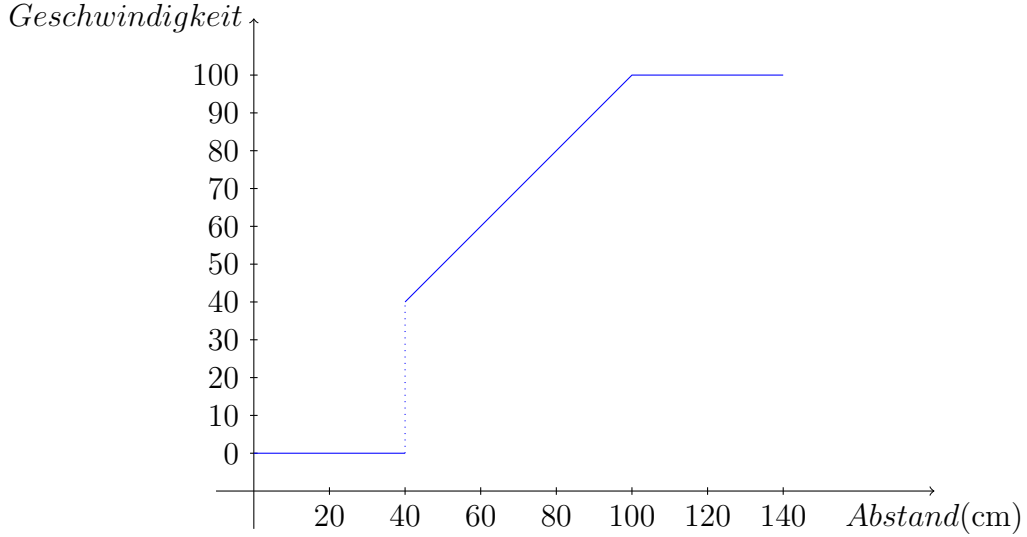


Abbildung 2: Übersetzung von gemessenem Abstand in Geschwindigkeit

Die statisch festgelegten Mindestabstandswerte für die einzelnen Sensoren führen allerdings zu der unerwünschten Eigenschaft, dass situationsunabhängig die Gewichtung der Sensorwerte konstant bleibt, die vor allem für ein Geradeaus-Fahren mit gewisser Geschwindigkeit optimal ist. Das ist beispielsweise in Kurven problematisch, wo das Hauptaugenmerk nicht mehr geradeaus liegen sollte, da die Geschwindigkeit durch vorbeifahrend frontal erkannte Hindernisse unnötig gedrosselt werden würde. Daher wurde eine dynamische Anpassung des relevanten Blickkegels realisiert. Dazu wird über den Parameter des aktuell eingeschlagenen Lenkwinkels jeder Messwert zunächst gemäß einer Gaußkurve gewichtet, bevor der geringste Abstandswert in eine Geschwindigkeit übersetzt wird.

$$weightedSensorVal = 5 - 4 * \exp\left(-3 * 2^{sensorAngle - steeringAngle}\right) \quad (1)$$

Konkret sieht man in Abbildung [x] die fixen Ablesungspunkte der einzelnen Sensoren auf der x-Achse, welche den Sensor-Winkeln in Blickrichtung des Autos angelehnt wurden. Der entsprechende Punkt auf der Gauss-Kurve gibt den jeweiligen Gewichtungs-Faktor an, wobei ein größerer Wert eine größere Unempfindlichkeit bedeutet. Gemäß Lenkwinkel wird nun die

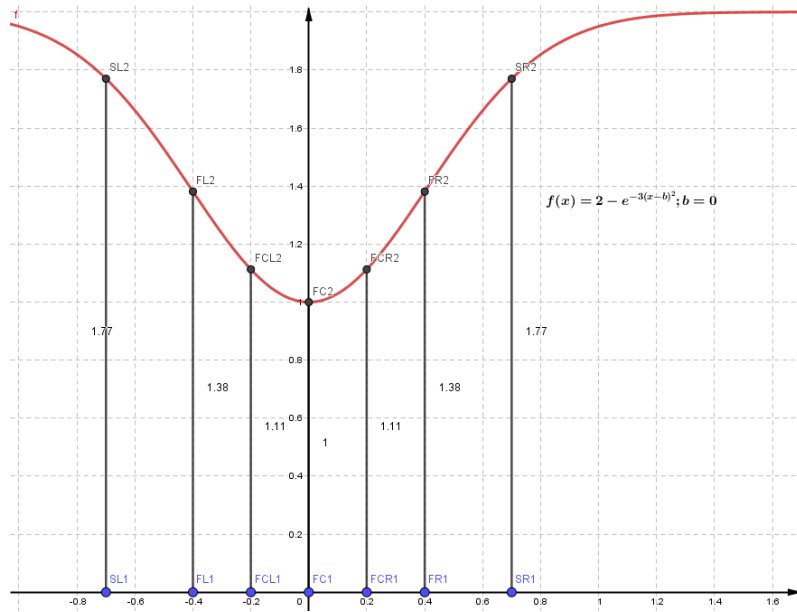


Abbildung 3: Gewichtung der Abstands-Messwerte beim Geradeaus-Fahren

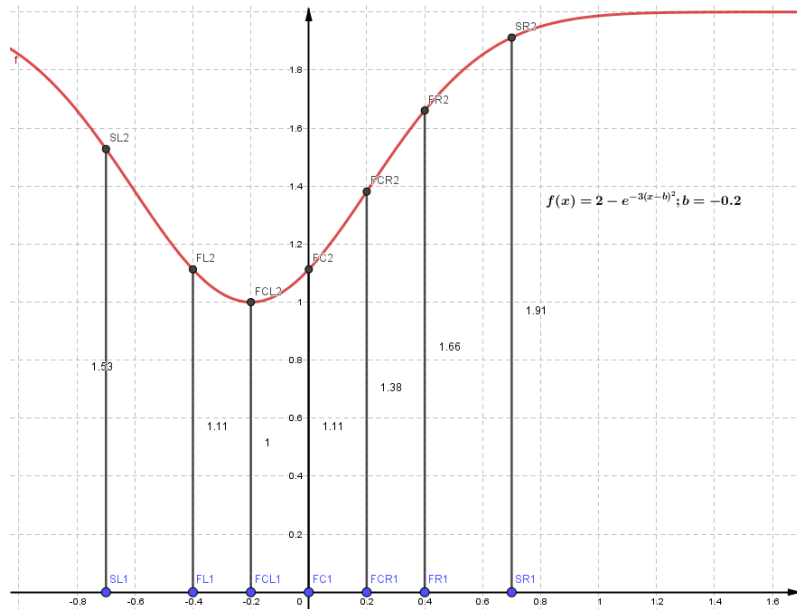


Abbildung 4: Gewichtung der Abstands-Messwerte beim Kurve-Fahren

Kurve auf der x-Achse verschoben, wodurch die Gewichtung sich auf die entsprechend seitlich liegenden Sensoren verschiebt.

### 3.2.3 Motoransteuerung

Die Hauptaufgaben der Motorsteuerung sind das Reagieren auf Flags, wie beispielsweise das Emergencybreak-Flag, sowie die Motorsteuerungsdaten, welche sie von dem Controller bekommt und an die Arduinokommunikation weitersendet.

Wir haben uns entschlossen in diesem Baustein auf Flags zu reagieren, um eine schnellere Reaktion auf kritische Ereignisse, wie zum Beispiel das plötzliche Auftauchen eines Hindernisses zu ermöglichen. Dies erreichen wir, indem wir die Flags direkt aus den Blöcken bekommen, welche die Sensordaten verarbeiten und das Signal nicht erst durch verschiedene Verarbeitungsschritte durchgehen muss. Die Emergencybreak wird erst wieder aufgehoben, sobald ein entsprechendes Signal von den Sensoren kommt.

Außerdem beherrscht dieser Baustein als einziger das Protokoll, welches zur Ansteuerung der Motoren mittels der Arduinokommunikation verwendet wird. So kann intern ein Protokoll verwendet werden, dass sich deutlich besser lesen und im Falle von Verarbeitungsfehlern debuggen lässt. Da der Motorcontroller eine gewisse Anzahl an Nullen benötigt, um aktiviert zu werden, werden die ersten 150 Geschwindigkeitsdaten auf Null gesetzt. Nach diesen 150 Nullen können dann Geschwindigkeiten an die Motoren gesendet werden. Um eine Geschwindigkeit zu verhindern, die zu schnell für die Ultraschallsensoren ist, haben wir uns entschieden eine maximale Geschwindigkeit festzulegen(momentan 12%). Sollte ein Wert, der größer als dieser ist, an die Motorsteuerung gesendet werden, so wird dieser ignoriert und ein Fehler in der Konsole ausgegeben.

**Umgebung** Das Auto wurde programmiert, um eine blaue Linie zu verfolgen. Es wurde ein großes blaues Oval auf den Boden geklebt. Das verwendete Klebeband ist recht Matt, um Spiegelungen zu vermeiden, jedoch wurde in der Testung die Sonneneinstrahlung minimiert.

### 3.2.4 Fusionsschnittstelle

Interprozesskommunikation

Da die Programmierung in verschiedenen Funktionsblöcken erfolgte, musste eine einheitliche

Kommunikation sichergestellt werden. Hierzu wurde eine separate Funktionssammlung erstellt, die von allen Funktionsblöcken inkludiert wurde. Sie beinhaltet Definitionen über den Nachrichtentyp und die Zusammensetzung strukturierter Datentypen. Da der vom ADTF gestellte Ablauf zum Senden oder Empfangen eines bestimmten Nachrichtentyps zur Codeverdopplung in den Funktionsblöcken führen würde, wurden die Funktionen zusammengefasst und ausgelagert. Damit ließ sich auch einheitlich sicherstellen, dass Funktionsblöcke beim Empfangen die Nachricht erst kopierten, bevor deren Inhalt verändert wurde.

### **3.3 Fallstricke: Komplikationen und Fehler**

## 4 Auswertung

# Abbildungsverzeichnis

1	Zu Projektbeginn erstellte Roadmap . . . . .	4
2	Übersetzung von gemessenem Abstand in Geschwindigkeit . . . . .	9
3	Gewichtung der Abstands-Messwerte beim Geradeaus-Fahren . . . . .	10
4	Gewichtung der Abstands-Messwerte beim Kurve-Fahren . . . . .	10

## Literatur

- [1] *AADC-Website*, Audi AG: <https://www.audi-autonomous-driving-cup.com/wettbewerb/ueberblick/>
- [2] *ADTF-Website*, Elektrobit: <https://www.elektrobit.com/products/eb-assist/adtf/>
- [3] *Im<sup>2</sup>C-Geschichte* <https://www.i2c-bus.org/twi-bus/>
- [4] E. Eube, G. Heinrichs, U. Ihlefeldt, *Im<sup>2</sup>C-Bussystem*, Fortbildungsveranstaltung in Mönchengladbach, 23.08.2008: <http://www.g-heinrichs.de/pdv/i2c.pdf>
- [5] *Im<sup>2</sup>C-Bussystem*, Robot Electronics: <http://www.robot-electronics.co.uk/i2c-tutorial>