- Robotik Praktikum -
Gruppe 2

Julian Finck, tinf101030 René P. Keller, tinf101364
Lea Morschel, inf101301 Adrian Sorge, tinf101313
Julian Weihe, tinf101087

Wintersemester 2017/ 2018

Inhaltsverzeichnis

Kapitel 1

Einführung

Das Robotik-Praktikum, welches im Ergebnis diese Dokumentation festhält, fand parallel zur und in Kombination mit der Bachelor-Vorlesung 'Einführung in die Robotik' statt, welche von Herrn Professor 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[1] 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 1.1.

[1] Audi Autonomous Driving Cup: https://www.audi-autonomous-driving-cup.com/wettbewerb/ueberblick/

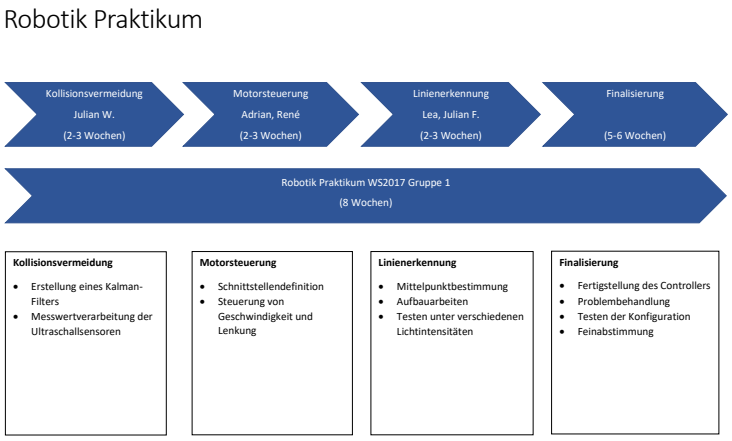

Abbildung 1.1: Zu Projektbeginn erstellte Roadmap

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

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

Kapitel 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

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

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.

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

Die Stromversorgung während der Fahrt wird durch zwei Akkus sichergestellt.

2.2 Softwareplattform

Das Modellauto verwendet die Ubuntu Version 16.04 und zur Bildschirmübertragung ist der x11vcn Serverdienst eingerichtet. Die Software EB Assist ADTF 2.14 von Elektrobit ist installiert, um die damit bereitgestellte Programmoberfläche und das Automotive Data and Time-Triggered Framework zu verwenden. Das Framework stellt Funktionen zur Kommunikation mit den im Abschnitt 'Hardwareplattform' erläuterten Sensoren und Kameras bereit. Außerdem ermöglicht es eine Programmaufteilung in Funktionsblöcken, welche parallel auf mehreren Prozessorkernen ausgeführt werden. 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 stellt die Oberfläche weitere fertige Funktionsblöcke zur Verfügung, etwa zum Anzeigen und Abspeichern von Kamerabildern und Sensordaten oder zum Abspielen selbiger. Durch Referenzen aus dem Programm heraus, können über die Benutzeroberfläche be-stimmte Parameter im Code zur Initialisierung und zur Laufzeit gesetzt werden.

Kapitel 3

Implementierung

3.1 Umgebung

Die Zielsetzung sieht vor, das Auto auf eine blaue Linie verfolgen zu lassen. Deswegen ist mit Klebeband ein Oval auf den Boden geklebt. Das verwendete Klebeband ist recht Matt, um Spiegelungen zu vermeiden. Außerdem ist das Fenster zugestellt, um die Sonneneinstrahlung zu minimieren.

TODO: Bild(er) der Umgebung

3.2 Struktur

Das Projekt ist in verschiedene Aufgabenbereiche aufgeteilt, die separat bearbeitet werden.

TODO: Bild der angepassten Roadmap

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 erge-ben sich durch Testläufe unter Realbedingungen.

TODO: Abbildung 1 und 2 (Übernehme 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 größ-ten 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 Bild-mitte 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 De-tektion 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 (übernehme ich genauso)

3.2.2 Abstandserkennung und -verwertung

Die Abstandserskennung 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 [Referenz auf USC-Modell-Bild aus Hardware-Doku-Part] implementiert.

Die Übersetzung der Mess- in Geschwindigkeitswerte wurde anhand einer linearen Funktion realisiert, die allerdings einen minimalen Input-Grenzwert zum Anstoßen der Bewegung und gleichsam einen Maximal-Geschwindigkeits-Wert fest eingebaut hat, der unter der vollen Kapazität des Motors liegt.

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 verbeifahrend 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. 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 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 Arduinokommunication 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 beherscht dieser Baustein als einziger das Protokoll, welches zur Ansteuerung der Motoren mittels der Arduinocommunication 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

Kapitel 4

Auswertung

Kapitel 5

Literaturverzeichnis

1. Im^{2}C Geschichte, https://www.i2c-bus.org/twi-bus/
2. Das Im^{2}C-Bus-System
3. http://www.g-heinrichs.de/pdv/i2c.pdf
4. http://www.robot-electronics.co.uk/i2c-tutorial
5. http://www.atmel.com/Images/8183S.pdf