



Hochschule für Angewandte  
Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

Fakultät Technik und Informatik  
Department Informations- und Elektrotechnik

## Bachelorprojekt

---

# Automated Driving

RC Car Control with Open Source Image Processing

---

Prof. Dr.-Ing. Marc Hensel

**Projektgruppe:** Fabian Huber, Enzo Morino, Markus Trockel

**Abgabe:** 11.02.2019

## Kurzübersicht

Das Ziel des Projekts ist ein Modellauto, dass mit Kamera ,Abstandssensor und Rapsberry Pi ausgestattet, durch Fahrbahnlinienerkennung teilautonom einem Straßenverlauf folgen kann. Es ist eine praktische Umsetzung von Systemarchitektur, Sensorik und der Nutzung von den Werkzeugen der Bildverarbeitung der Library openCV in Python.

Dafür ist ein Prozess entwickelt worden, bei dem aus Sensordaten Steuerungsinformationen für den Wagen generiert werden. Die Durchführung hat gezeigt, dass die Umsetzung möglich ist, dass für eine stabile Funktionsweise weitere Entwicklungen von Funktionen nötig sind, die die zeitkritischen Anforderungen erfüllen und hardwarebedingte Fehleranfälligkeit abfangen, um so eine kontinuierliche Fahrt zu ermöglichen.

# Contents

<b>1 Einleitung</b>	<b>1</b>
<b>2 Ziel des Projekts</b>	<b>2</b>
<b>3 Software</b>	<b>2</b>
3.1 Aufbau . . . . .	3
3.2 Externe Module . . . . .	4
3.3 Eigene Module . . . . .	4
<b>4 Prinzip der Steuerung</b>	<b>5</b>
<b>5 Hardware</b>	<b>5</b>
5.1 Raspberry Pi 3 . . . . .	5
5.2 Motorcontroller . . . . .	6
5.3 Raspberry Pi Camera Module . . . . .	6
5.4 Ultraschallsensor . . . . .	7
5.5 RC Fahrzeug . . . . .	7
5.6 Aufbau . . . . .	8
<b>6 Software</b>	<b>8</b>
6.1 Einrichtung der Netzwerkverbindung . . . . .	8
6.1.1 Wifi . . . . .	8
6.1.2 Ethernet . . . . .	9
6.1.3 Login per SSH . . . . .	10
6.2 Aufbau . . . . .	11
6.3 Externe Module . . . . .	12
6.4 Eigene Module . . . . .	12
6.5 Funktionen aus der openCV Library . . . . .	13
6.5.1 Canny-Edges Filter . . . . .	13
6.5.2 Hough-Transformation . . . . .	14
<b>7 Werkzeuge der Bildverarbeitung</b>	<b>14</b>
7.1 Digitalisierung . . . . .	15
7.2 Vorverarbeitung - Der Canny-Kantenoperator . . . . .	15
7.3 Segmentierung . . . . .	15
7.4 Merkmalsextraktion zur Linienerkennung . . . . .	16
<b>8 Iterative Auswertung des Kamerabildes</b>	<b>16</b>
8.1 Verwendete Repräsentationsformen von Geraden . . . . .	17
8.2 Berechnung des Schnittpunktes der beiden Geraden . . . . .	21
<b>9 Ansteuerung von Servo und Motorcontroller</b>	<b>21</b>

# 1 Einleitung

Im Jahr 2005 ging das Projektauto Stanley der Standford University an den Start der DARPA Grand Challenge, einem Rennen autonomer Fahrzeuge über eine definierte Strecke in der Mojave-Wüste. Nach 212,76km durch unwegsames Gelände ging Stanley als Erster über die Ziellinie und das Team um Professor Sebastian Thrun war Gwinner des mit 2 Millionen Dollar dotierten Rennens.

Die Entwicklung des Autonomen Fahrens für Fahrzeuge des Straßenverkehrs ist ein aktuelles Thema, das immer mehr in die Praxis umgesetzt wird. Dabei gibt es verschiedenen Stufen des teilautonomen Fahrens. So gehört zur Stufe 2 die Funktion, dass Lenkvorgänge vom Fahrassistenten ausgeführt werden.

Durch dieses Projekts soll ein Einblick in die Materie geschaffen werden und die Umsetzbarkeit im Rahmen studentischer Möglichkeiten erfahren werden .

Die Autoren wollen mit ihrem Auto Hawey einen Startschuß setzen und mit diesem Bericht ein nachfolgendes Team inhaltlich in der Sache abholen, dass ein Einstieg in das Projekt und so ein konstruktives Fortsetzen und Verbessern möglich ist.

Fabian Huber, Enzo Morino, Markus Trockel

## 2 Ziel des Projekts

Das Projekt hat als Ziel, ein Modellauto mit Elektroantrieb durch Fahrbahnlinienerkennung eine vorgegebene Strecke fahren zu lassen. Als technische Bestandteile wird ein Raspberry Pi 3 mit dazugehöriger Kamera gewählt, dazu ein Sensor zur Abstandsmessung, um etwaige Kollisionen zu vermeiden. Implementiert wird der Code in Python unter der Nutzung der Bildverarbeitungslibrary openCV.

Das Auto soll sowohl ein 4m lange Strecke geradeaus als auch in Kurven von 90° zwischen zwei Fahrbahnlinien die Spur halten. Idealerweise ist es möglich, dass der Wagen auch mit Linienunterbrechungen weitersteuern und so eine acht-förmige Strecke abfahren kann wie sie beim Carolo-Cup, einem Wettbewerb für autonome Modellfahrzeuge, befahren wird.

## 3 Software

### 3.1 Aufbau

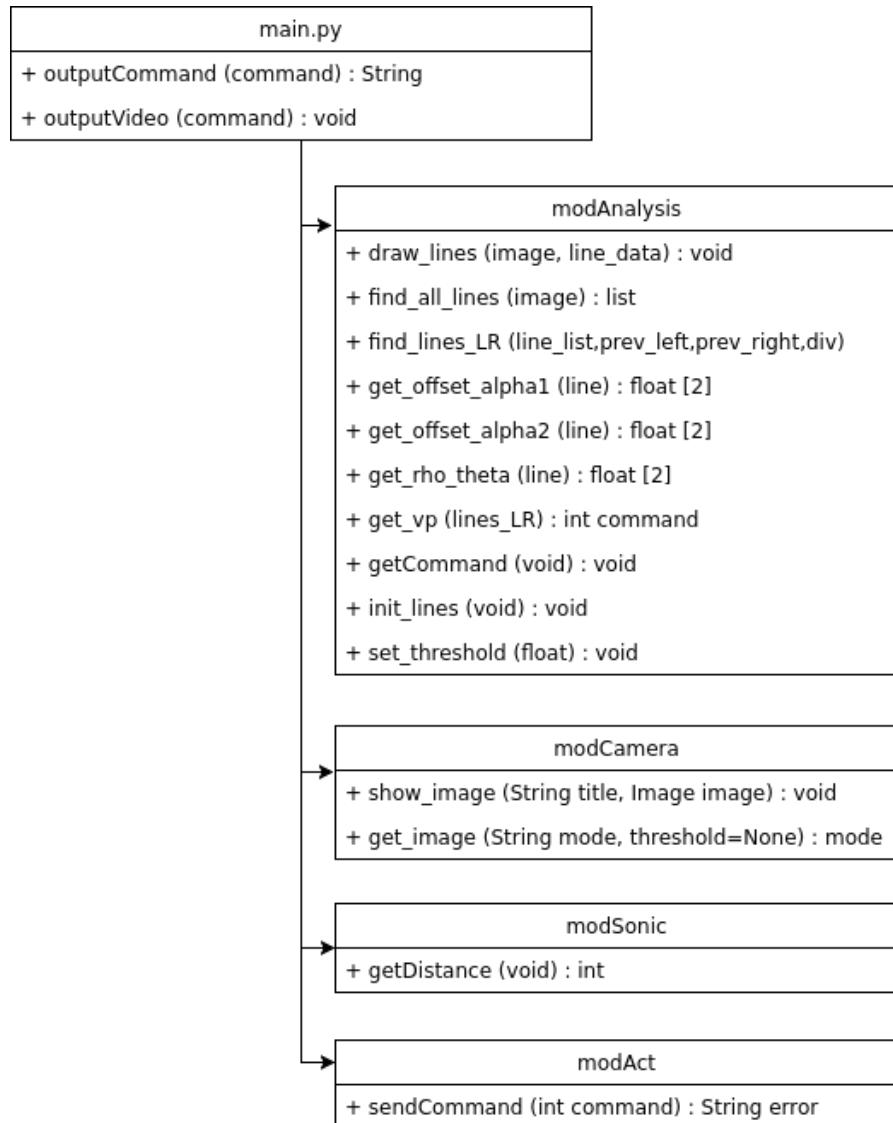


Figure 1: Aufbau des Python Codes

### 3.2 Externe Module

Name	Beschreibung
tkinter	...
Adafruit_PCA9685	Bibliothek zur Ansteuerung des Motorcontrollers
numpy	Bibliothek zur Verwendung von Matlab Funktionen
cv2	OpenCV 2 bietet Algorithmen zur Bildverarbeitung
io	...
time	...
importlib	...
argparse	...
pivideostream	...
picamera	...
threading	...
RPi.GPIO	Bibliothek zur Ansteuerung der GPIO ports des Raspberry Pi

Table 1: verwendete externe Python Module

### 3.3 Eigene Module

Name	Beschreibung
modAnalysis	Verantwortlich für die eigentliche Verarbeitung der visuellen Informationen
modAct	Verantwortlich für die Ansteuerung des Motors und der Lenkung
modCamera	Bereitet das Kamerabild für die Verarbeitung und Anzeige vor.
modSonic	Kommuniziert mit dem Ultraschallsensor und liefert Distanz zum Hindernis.

Table 2: verwendete eigene Python Module

## 4 Prinzip der Steuerung

Über ein Python-Programm, welches auf dem Raspberry Pi läuft, wird der Video-Stream der angeschlossenen Kamera iterativ ausgewertet. Es werden die beiden Fahrbahnlinien erkannt, durch Geraden angenähert und deren Fluchtpunkt berechnet. Auf Grundlage der x-Koordinate dieses Fluchtpunktes wird ein Ausgangssignal berechnet, welches über das PWM-Modul den Servo ansteuert, und damit den Lenkwinkel festlegt.

## 5 Hardware

Für die technische Umsetzung werden eine Reihe von Komponenten verwendet.

### 5.1 Raspberry Pi 3



Figure 2: Raspberry Pi 3 Model B

Das Herzstück des Projekts bildet ein Raspberry Pi 3 Model B. Ausgestattet ist dieser Mini-PC neben den üblichen Schnittstellen mit drei Elementen, die für das Vorhaben essentiell sind: Einem Wifi Modul, einem Ethernet-Port und allem einem GPIO Header mit reichlich freien Pins.

Als Betriebssystem nutzt der Raspberry Pi das hauseigene Debian Derivat Raspbian.

## 5.2 Motorcontroller

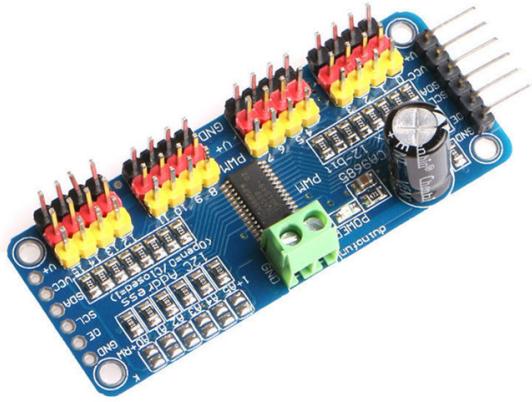


Figure 3: PCA9685 Controller Modul

Der Fahrtenregler des Motors für den Antrieb und der Servo für die Lenkung des RC Fahrzeugs werden über das PCA9685 Controller Modul mittels Pulsweitenmodulation angesprochen. Die Implementierung selbst ist glücklicherweise bereits durch eine offene Python Bibliothek (siehe Tab. 3) verfügbar.

## 5.3 Raspberry Pi Camera Module

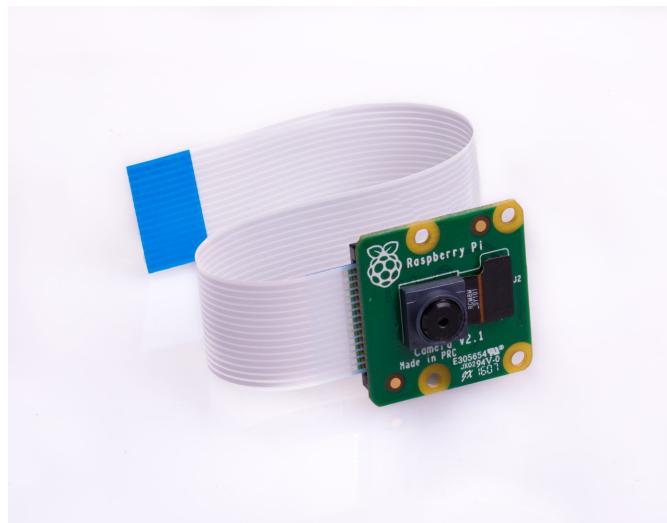


Figure 4: Camera Module V2

Das Sehen ermöglicht das eingesetzte Raspberry Pi Kameramodul, welches über einen speziell am Raspberry Pi vorhandenen Kameraanschluss. Verbaut ist ein Sony IMX219 8-Megapixel Sensor der ebenfalls direkt über vorhandene Python Bibliotheken performant und resourcenschonend ausgelesen werden kann.

## 5.4 Ultraschallsensor

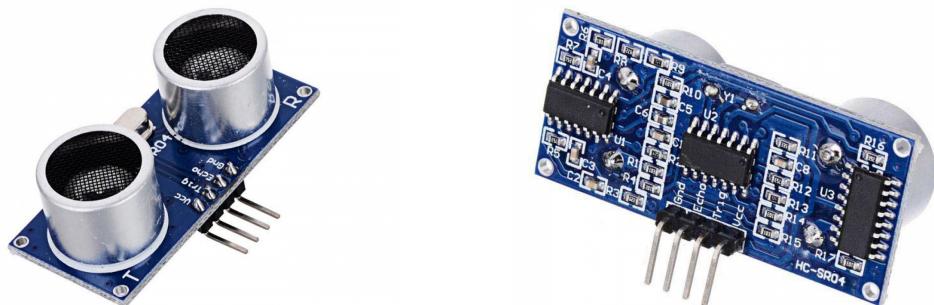


Figure 5: HC-SR04 Sensor Modul

Im Falle eines anstehenden Frontalzusammenstoßes mit einer Wand oder einem anderen größeren Hindernis, soll zusätzlich zur Kamera vorne am Fahrzeug ein Ultraschallsensor angebracht werden, der unmittelbare Hindernisse melden kann. Hier kommt der HC-SR04 zum Einsatz. Für die Abstandsmessung sendet dieser in regelmäßigen Abständen Ultraschallpulse aus und empfängt dann das rückgeworfene Echo. Die Laufzeit gibt Auskunft über den Abstand zum Hindernis. Fällt dieser Abstand unterhalb eines bestimmten Schwellwerts, so wird der Antrieb automatisch gestoppt und keine weiteren Signale verarbeitet.

## 5.5 RC Fahrzeug

Als Chassi dient ein Einsteiger RC-Modell von dem Elektronik-Versandhandel Conrad Electronic. Die mitgelieferten Module für Antrieb und Lenkung können ohne Probleme direkt mit dem verwendeten Motorcontroller verwendet werden. Einzig der Lenkservo musste zwischenzeitlich durch ein robusteres Modell ausgetauscht werden da er durch einen Fehler in der Programmierung übersteuert und damit das Getriebe unbrauchbar wurde.

Der enthaltene Akku kann mithilfe eines einfachen Voltage Regulators ebenfalls eingesetzt und zur Stromversorgung des Raspberry Pi verwendet werden.

Die Karosserie wird durch einen eigens gefertigten Acrylaufbau ersetzt, der mehr Raum für Raspberry Pi und Module bietet und zudem eine stabile Basis für den Kameramast mitbringt.

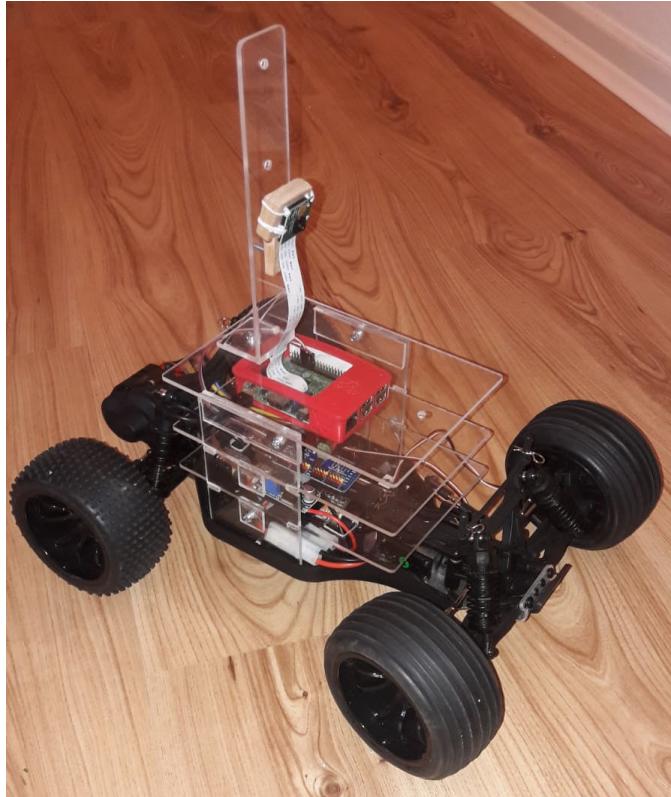


Figure 6: RC Chassis mit Aufbau

## 5.6 Aufbau

!  
! Hier fehlt noch ein Schematic welches Modul an welchen Pins hängt  
!

# 6 Software

## 6.1 Einrichtung der Netzwerkverbindung

Die Steuerungssoftware läuft vollständig auf dem Raspberry Pi. Um komfortabel am Quellcode arbeiten zu können ohne jedesmal Monitor und Tastatur anschließen zu müssen, wird der Wifi-Zugang eingerichtet und eine Backup-Verbindung über ein Ethernetkabel konfiguriert.

### 6.1.1 Wifi

Zum aktivieren der Wifi-Verbindung kann einfach das Tool `raspi-config` verwendet werden. Gestartet werden kann das Tool durch Eingabe des folgenden Befehls im Terminal-Fenster:

```
sudo raspi-config
```

Im Unterpunkt **Network Options** findet sich die Funktion **Wifi**. Im Folgenden wird der Benutzer zur Eingabe des Netzwerknamen (SSID) und dem Passwort aufgefordert.

Um die Netzwerkeinstellungen händisch hinzuzufügen reicht es in der Regel, die Datei `/etc/wpa_supplicant/wpa_supplicant.conf` mit einem Editor zu öffnen (Achtung: Root-Rechte erforderlich) und folgenden Eintrag am Ende der Datei anzufügen:

```
1   network={  
2     ssid="NameDesNetzwerks"  
3     psk="Passwort"  
4 }
```

Nach der Änderung der Einstellung kann der Netzwerk-Service neu gestartet werden:

```
sudo service networking restart
```

### 6.1.2 Ethernet

Gelegentlich kann es vorkommen, dass sich der Raspberry Pi nicht automatisch mit dem eingerichteten Wifi verbindet, zum Beispiel wenn das WLAN Netzwerk nicht verfügbar ist. Um sich in dieser Situation trotzdem mit dem Gerät verbinden zu können, wird eine Backup-Verbindung über Ethernet-Kabel eingerichtet. Damit kann sich der Benutzer bei Bedarf per Kabel mit seinem Laptop direkt mit dem Raspberry Pi verbinden.

Hierfür kann mit einem Editor die Konfigurationsdatei `/etc/network/interfaces` geöffnet werden. Für das entsprechende Ethernet-Gerät, in diesem Fall `eth0`, wird somit eine statische IP Adresse vergeben, über die dann künftig ebenfalls eine SSH-Verbindung aufgebaut werden kann.

```
1   iface eth0 inet static  
2     address 192.168.1.10  
3     netmask 255.255.255.0  
4     gateway 192.168.1.1
```

### 6.1.3 Login per SSH

Nachdem das Netzwerk eingerichtet ist und eine Verbindung zum Wifi Netzwerk steht, wird zum Login über das Netzwerk die neue IP Adresse des Raspberry Pi benötigt. Um diese herauszufinden gibt es zahlreiche Möglichkeiten. Eine davon ist direkt am Raspberry Pi im Terminal das Kommando `ifconfig` einzugeben und zu bestätigen. In der Ausgabe findet sich unter der Bezeichnung des Wifi-Geräts, meist `wlan0`, ein Eintrag `inet addr`, gefolgt von einer IP Adresse. Mit Hilfe dieser Adresse lässt sich nun von jedem Computer im selben Wifi Netzwerk eine Verbindung zum Raspberry Pi herstellen (die 'x' stehen hier für die eigentliche IP Adresse):

```
ssh -X pi@xxx.xxx.xxx.xxx
```

Standard Benutzername ist hier 'pi' und das Passwort nach dem gefragt wird ist 'raspberry'. Der Parameter `-X` ermöglicht die grafische Ausgabe des Programms ebenfalls an die lokale Ausgabe des entfernten Laptops weiterzuleiten.

## 6.2 Aufbau

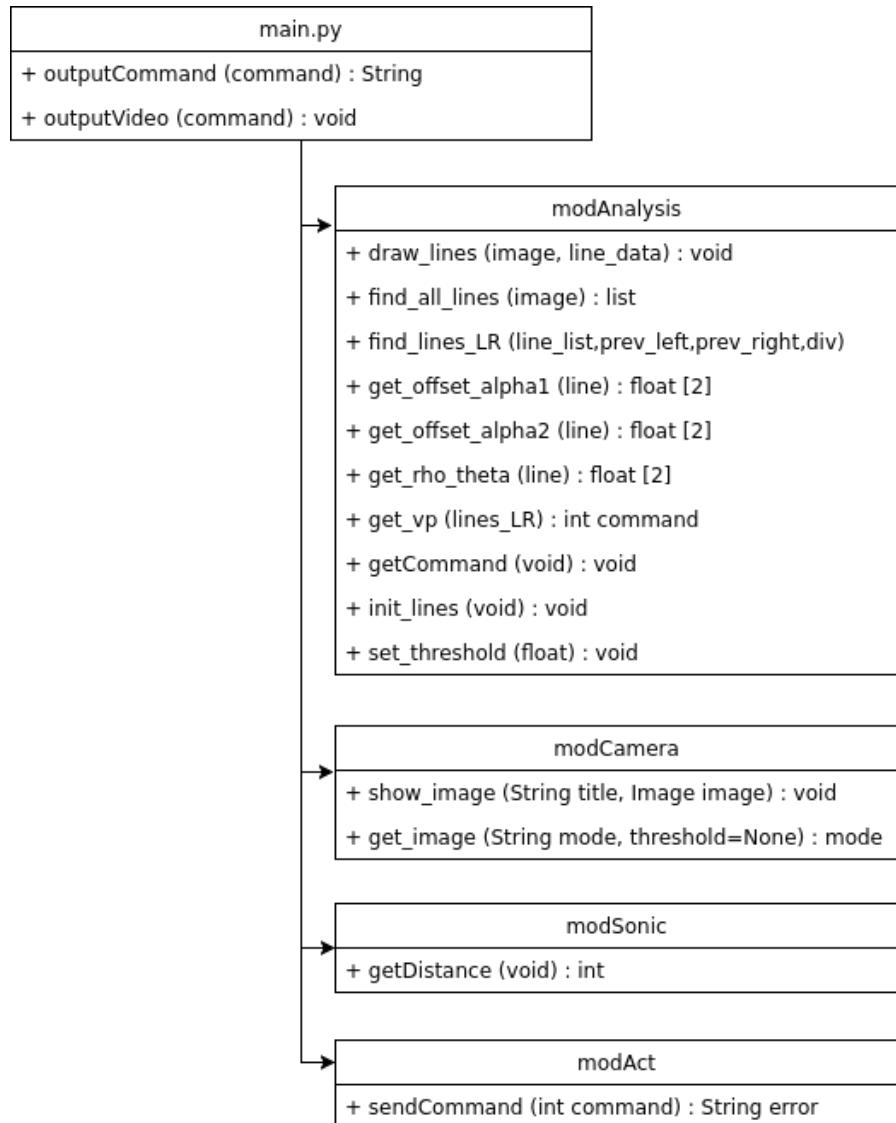


Figure 7: Aufbau des Python Codes

## 6.3 Externe Module

Name	Beschreibung
Adafruit_PCA9685	Bibliothek zur Ansteuerung des Motorcontrollers
numpy	Bibliothek zur Verwendung von Matlab Funktionen
cv2	OpenCV 2 bietet Algorithmen zur Bildverarbeitung
io	Bibliothek zur Behandlung und Verarbeitung unterschiedlicher I/O Streams
time	Bibliothek zur Behandlung und Umwandlung von Zeitstempeln
importlib	Ermöglicht die Benutzung der 'import' Funktion
argparse	Ermöglicht die einfache Behandlung von übergebenen Argumenten
pivideostream	Teilpacket der Bibliothek <code>imutils</code>
picamera	Bibliothek zu direkten Interaktion mit dem Pi Camera Modul
threading	Stellt High Level Funktionen zur Parallelverarbeitung mit Threading zur Verfügung
RPi.GPIO	Bibliothek zur Ansteuerung der GPIO ports des Raspberry Pi

Table 3: verwendete externe Python Module

## 6.4 Eigene Module

Name	Beschreibung
modAnalysis	Verantwortlich für die eigentliche Verarbeitung der visuellen Informationen.
modAct	Verantwortlich für die Ansteuerung des Motors und der Lenkung.
modCamera	Bereitet das Kamerabild für die Verarbeitung und Anzeige vor.
modSonic	Kommuniziert mit dem Ultraschallsensor und liefert Distanz zum Hindernis.

Table 4: verwendete eigene Python Module

## 6.5 Funktionen aus der openCV Library

OpenCV ist eine open-source Bibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen (siehe auch [opencv.org](http://opencv.org)). Sie ist u.a. für die Programmiersprache Python geschrieben und beinhaltet Funktionen, die für die Fahrbahnlinienerkennung eingesetzt werden.

### 6.5.1 Canny-Edges Filter

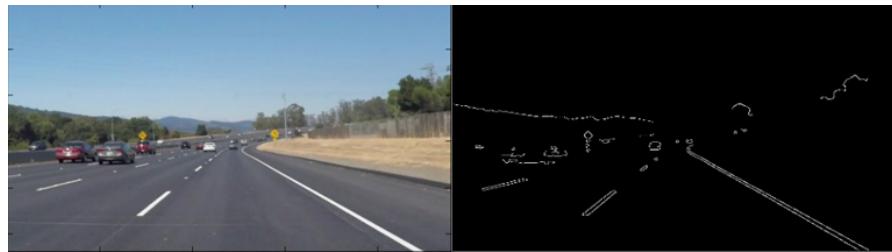


Figure 8: Beispiel: Fahrbahnbild und Canny-Edges-Filter

Der Canny-Edges-Filter wird im Kamera-Modul angewendet und über

<sup>1</sup> `openCV.Canny("image-file", int lowerThreshold, int upperThreshold)`

aufgerufen. Die Funktion liefert ein binäres Bild, das die Umrisse von Bildobjekten in weiß auf schwarzem Hintergrund darstellt (siehe Abb.9).

Nach einem Rauschfilter, der im Hintergrund läuft, wird der Gradient für jedes Pixel bestimmt. Liegt ein Gradient unterhalb des *untererThresholds*, wird ein Pixel schwarz im Zielbild, ist der Gradient oberhalb des *upperThresholds*, wird ein weißes Pixel ins Zielbild geschrieben. Gradientwerte im Zwischenbereich werden als schwarz geschrieben, es sei denn, sie sind mit Gradienten im Bild verbunden, die oberhalb des Thresholds liegen.

Die Einstellungen des Thresholds sind ein sensibler Bereich, da hier eine Vorentscheidung getroffen wird, welche Bildinformationen erhalten bleiben bzw. verworfen werden. Idealerweise würden hier nur die Umrisse der Fahrbahnlinien ausgewählt werden.

### 6.5.2 Hough-Transformation

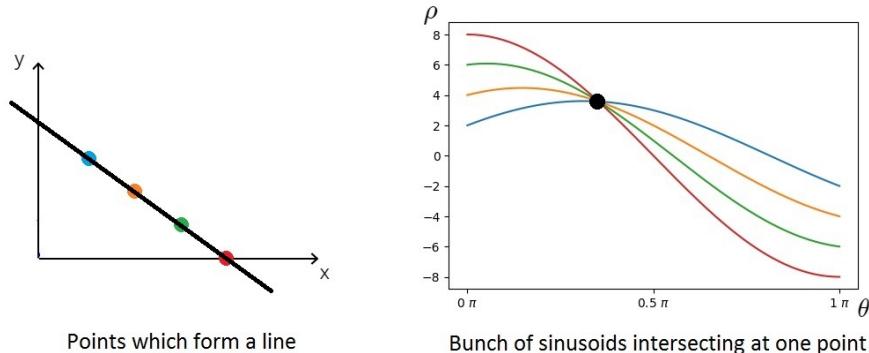


Figure 9: Beispiel: Darstellung einer Geraden kartesisch und im Hough-Raum

Die Houghtransformation wird mit `opencv.HoughLines(image, rho, theta, threshold)` aufgerufen und gibt einen Vektor zurück mit allen im Bild erkannten Linien, die jeweils mit dem Wertpaar  $(\rho, \theta)$  beschrieben werden. Für  $\rho$  und  $\theta$  wird durch den Wert ein Toleranzwert beschrieben. "threshold" bezeichnet einen Wert, der die Mindestanzahl von Punkten bestimmt, die vorliegen müssen, damit eine Grade als beschrieben gilt.

Die Funktion wird zweimal angewendet: zum Einen beim Start zur Initialisierung, daß im gegeben Kamerabild die Fahrbahnlinien erkannt werden. Hierbei wird mit einem größeren Toleranzbereich gearbeitet, um eine Erkennung sicherzustellen. Zum anderen wird für jedes Bild des Kamerastreams die Linienerkennung gemacht, und mit einem strengeren Toleranzbereich mit den aus dem Vorbild erkannten Linienpaar verglichen. Dadurch soll verhindert werden, dass weitere erkannte Linien, die nicht Teil der Fahrbahn sind, fehlerhaft in die erkannten Linien mit einbezogen werden.

Die so noch erkannten Linien werden anhand von  $\rho$  und  $\theta$  entweder der linken oder der rechten Fahrbahnlinie zugeordnet, und für beide Seite eine Linie gemittelt und als neue Orientierungslinien zur Richtungsbestimmung genutzt.

## 7 Werkzeuge der Bildverarbeitung

Die folgende Zusammenfassung der Thematik basiert auf dem Buch "Digitale Bildverarbeitung" von Burger, Burge (?), das als Standardliteratur zu empfehlen ist. Eine zusammenfassende Wiedergabe der für das Projekt relevanter Themen ist im Folgenden nachzulesen.

Ein Bestandteil der Bildverarbeitung ist die Bildanalyse, bei der sinnvolle Informationen aus Bildern extrahiert werden. Genauer ist der Themenbereich *Computer Vision* gemeint, bei dem es um das Mechanisieren von Sehvorgängen des Menschen in der dreidimensionalen Welt geht.

Die Steuerung des Wagens soll durch Informationen aus den Kamerabildern er-

folgen: Durch Erkennen der Fahrbahnlinien wird der Lenkungsservo geregelt, der das Auto innerhalb der Spur zwischen den Fahrpahnliniien hält.  
Dieser Prozess lässt sich in Einzelschritten beschreiben mit:

1. *Digitalisierung* der dreidimensionalen Welt mit Hilfe der Kamera
2. *Vorverarbeitung* (Bildverbesserung bzw. Anpassung an den Zweck) durch Umwandlung in ein binäres Canny-Edges-Bild
3. *Segmentierung* durch Vorauswahl des Bildausschnittes mit den relevante Informationen
4. *Merkmalsextraktion* zur Linienerkennung durch die Hough-Transformation
5. *Parametrisierung* als mathematische Beschreibung der Fahrbahnlinien zur weiteren Informationsverarbeitung

## 7.1 Digitalisierung

Der von der Kamera aufgenommen Bilder-Stream liegt in digitaler Form vor. Dabei lassen sich benötigte Parameter einteilen. Um ein optimales Bild zu erhalten sind die Beleuchtungsumstände zu beachten, da hierdurch die Differenzierung von Linien im Bild erheblich beeinflusst wird.

Anzumerken ist hier, dass in der Bildverarbeitung der Ursprung des Koordinatensystems bzw. der x- und y-Achse in der linken oberen Ecke des Bildes definiert ist.

## 7.2 Vorverarbeitung - Der Canny-Kantenoperator

Zur späteren Erkennung der Linien werden die dafür relevanten Informationen aus dem Bild gefiltert: sogenannte Bildkannten. Bildkanten sind Übergangsstellen, wo ein hoher Grauwertsprung von einem Pixel zum Nachbarpixel vorliegt, wie z. B. bei einem weißen Farbahnstreifen auf dunkler Fahrbahn, hier werden zwei Kannten analysiert. Kanten werden dem entsprechend an diversen Stellen im Bild detektiert, deshalb sind die Parameter entsprechend zu wählen.

Die Canny-Edges-Funktion ist ein bewährter Algorithmus zur Kantenerkennung, da drei Ziel gleichzeitig erreicht werden: ein zuverlässiges Detektieren vorhandener Kanten, die Position der Kante präzise zu bestimmen, und Farbsprünge, die nicht als Kante interpretiert werden sollen, auszulassen. Diese Funktion ist in der OpenCV-Library enthalten.

## 7.3 Segmentierung

[BILD Kamera mit Fahrbahnlinien] Der für die Erkennung der Fahrbahnlinien relevante Bereich liegt in einem unteren Dreieck des Kamerabildes. Dieser Teil wird ausgewählt, bzw. davon ausserhalb liegende Bereiche werden nicht berücksichtigt.

## 7.4 Merkmalsextraktion zur Linienerkennung

Das Canny-Kantenbild wird mit Hilfe der Hough-Transformation aus der openCV-Library in den Hough-Raum transformiert, wo dann die Erkennung der Linien erfolgt.

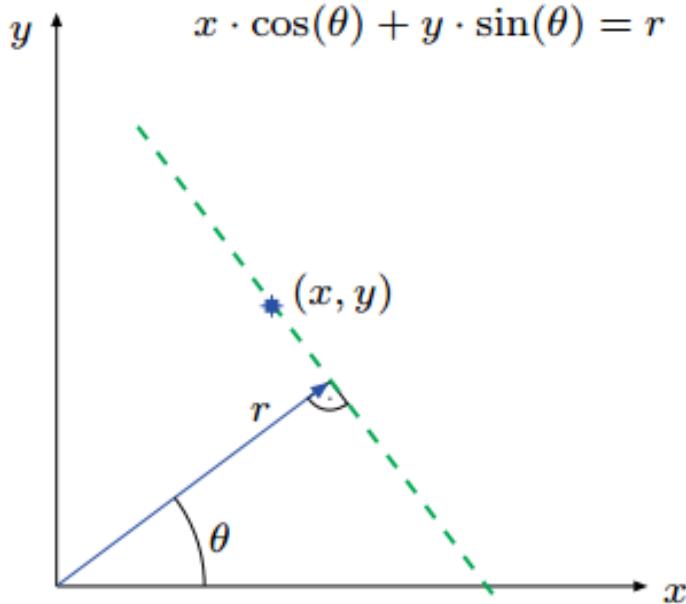


Figure 10: Beschreibung einer Geraden durch Winkel und Abstand vom Ursprung

Eine Gerade kann mathematisch sowohl mit  $y = m \cdot x + b$  als auch mit  $r(\theta) = x \cdot \cos(\theta) + y \cdot \sin(\theta)$  beschrieben werden. Dabei ist in der zweiten Schreibweise  $\theta$  der Winkel des Radius  $r$  zum Ursprung, an dessen Ende senkrecht die beschriebene Gerade verläuft. Zu Bedenken ist, dass, wie schon erwähnt, der Koordinatenursprung des Bildes oben links definiert ist.

Das rechenaufwendige Verfahren der Hough-Transformation generiert für jeden Kantenbildpunkt des Canny-Edges-Bildes Bildpunkte im Hough-Raum, dessen Achsen ein  $\theta / r$  - Koordinatensystem bilden - eine Gerade wird also durch einen Punkt dargestellt. Teilstücke einer Geraden im kartesischen System sind als Punkthaufungen im Hough-Raum erkennbar. Liegt eine Anzahl von Punkten oberhalb eines zu definierenden Schwellwertes, wird eine Linie als erkannt gewertet, und die Hough-Transformationsfunktion gibt ein Wertpaar  $\theta, r$  zurück.

## 8 Iterative Auswertung des Kamerabildes

Das Bild der Kamera werden iterativ ausgewertet um die beiden Fahrbahnlinien zu erkennen und daraus ein Ausgangssignal für die Steuerung zu generieren. Der gewählte Ansatz war es, die Fahrbahnlinien durch zwei Geraden anzunähern. Die Funktion HoughLines wird verwendet um die Fahrbahnlinien zu erkennen und die zugehörigen Geraden zu berechnen. Da die Funktion HoughLines in den meisten Fällen nicht nur die beiden Fahrbahnlinien erkennt, sondern alle möglichen Ger-

aden, bestand die erste Aufgabe darin, nur die Geraden herauszufiltern, welche die Fahrbahnlinien repräsentieren, und alle anderen Geraden zu verwerfen. Im Programm wird dies erreicht, indem alle Geraden, die durch die Funktion HoughLines gefunden wurden mit dem Geradenpaar aus dem vorherigen Programmdurchlauf verglichen werden, und nur ähnliche Geraden beibehalten werden.

Beim Start des Programm werden für die beiden Geraden feste Werte vorgegeben, die dann korrigiert und an die erkannten Linien angepasst werden. In Abbildung 11 ist dies bildlich dargestellt. Die roten Linien sind zu Beginn des Programms fest vorgegeben. Die blauen Linien sind das Ergebnis der Korrektur durch das Programm.

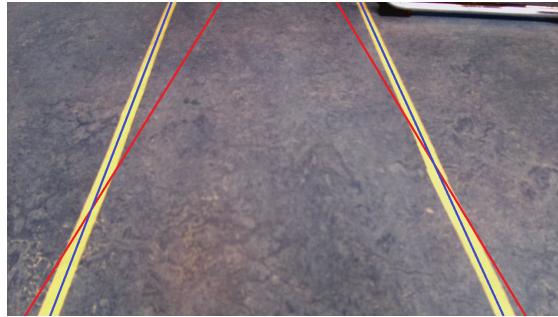


Figure 11: Beispiel für die erste Form der Geradenrepräsentation

Der Vergleich der Geraden wird durchgeführt, indem um die beiden Geraden aus dem letzten Durchlauf ein Toleranz-Fenster gelegt wird, und von allen gefundenen Geraden nur diejenigen herausgefiltert werden, die innerhalb dieses Toleranz-Fensters liegen. Von diesen Geraden wird dann der Durchschnitt berechnet, sodass nur noch zwei Geraden für die beiden Fahrbahnlinien übrig bleiben.

Eine Schwierigkeit besteht darin die Geraden so zu repräsentieren, dass diese gut miteinander vergleichbar sind. Bei einer geringen Änderung einer Geraden von einem zum nächsten Zyklus sollen sich deren Parameter dabei auch nur geringfügig ändern. Ansonsten werden ähnliche Geraden beim Filtern verworfen, da deren Parameter nicht im Toleranzfenster liegen. Im Programm wurden daher je nach Anwendungsfall verschiedene Repräsentationsformen verwendet, welche im folgenden beschrieben werden.

## 8.1 Verwendete Repräsentationsformen von Geraden

Die Funktion HoughLines gibt als Ergebnis eine Liste von Geraden aus, die durch die beiden Parameter  $\rho$  und  $\theta$  beschrieben werden wird.  $\theta$  ist der Winkel zwischen der Normalen, die senkrecht auf der Geraden steht, und der x-Achse. Der Winkel  $\rho$  ist die Länge dieser Normalen, d.h. der Abstand zwischen der Geraden und dem Ursprung des Koordinatensystems. Der Winkel  $\theta$  ist immer positiv,  $\rho$  kann, je nach Lage der Geraden positiv oder negativ sein. Abbildung 12 zeigt ein Beispiel für die Repräsentation zweier Fahrbahnlinien in der  $\rho$ - $\theta$ -Darstellung, für den Fall,

dass der Wagen relativ mittig und gerade auf der Fahrbahn steht. Das Rechteck symbolisiert die Ränder des Kamerabildes. Der Koordinatenursprung liegt in der oberen linken Ecke.  $\theta_R$  und  $\rho_R$  sind die Parameter der rechten Fahrbahnlinie,  $\theta_L$  und  $\rho_L$  die der Linken.

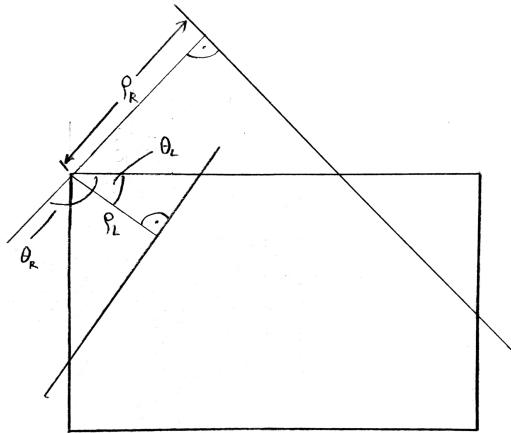


Figure 12: Beispiel für die erste Form der Geradenrepräsentation

Diese Form der Geradenrepräsentation jedoch problematisch, wenn es darum geht, zwei Geraden miteinander zu vergleichen. Ein Beispiel ist in Abbildung 13 dargestellt. Hier beträgt die Änderung der Winkel  $\theta_R$  und  $\theta_L$  der beiden Fahrbahnlinien  $\Delta\theta = 5^\circ$ , obwohl die Position der rechten Fahrbahnlinie sich viel mehr verändert hat, als die der linken.

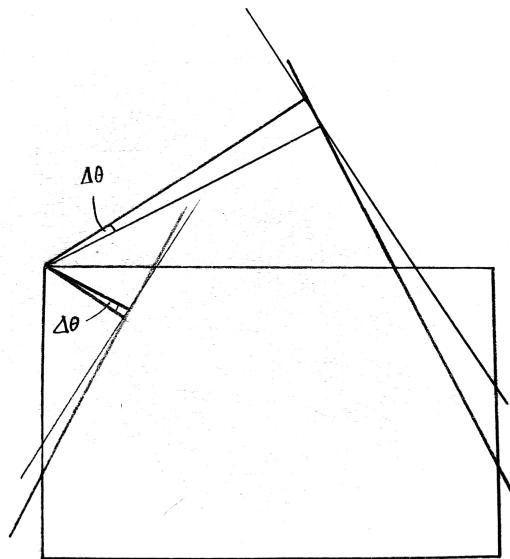


Figure 13: Problematischer Fall für diese Form der Geradenrepräsentation

Um dieses Problem zu beheben wurden die Geraden mithilfe der Funktion `get_offset_alpha1` in eine andere Form umgerechnet, bei der diese durch den Offset

$\omega$  auf der y-Achse und den Winkel zur x-Achse beschrieben werden. Die Umrechnung von der  $\rho\text{-}\theta$ -Form erfolgt folgendermaßen:

Für Winkel  $\theta < 90^\circ$  (linke Fahrbahnlinie):

$$\alpha = 90^\circ - \theta\omega = -\frac{\rho}{\cos 90^\circ - \theta}$$

Für Winkel  $\theta > 90^\circ$  (rechte Fahrbahnlinie):

$$\alpha = \theta - 90^\circ\omega = -\frac{\rho}{\cos \theta - 90^\circ}$$

Abbildung 14 veranschaulicht die Repräsentation der beiden Fahrbahnlinien in dieser Form.

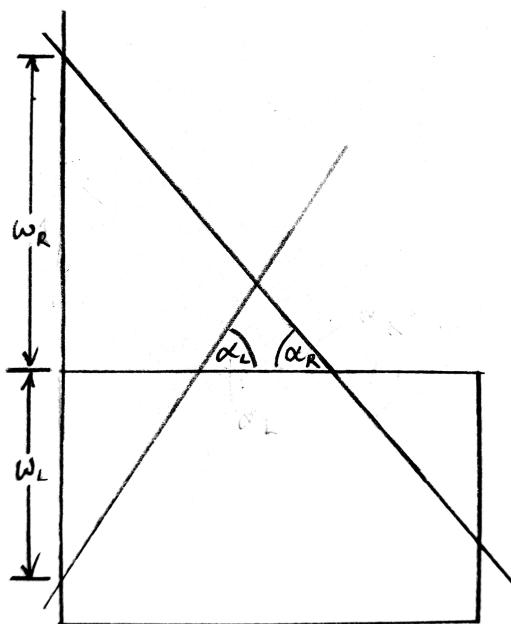


Figure 14: Beispiel für die erste Form der Geradenrepräsentation

Bei dieser Form ergibt sich jedoch das Problem, dass sich bei geringfügiger Veränderung der Geraden deren Parameter in bestimmten Situationen stark ändern. Dies ist besonders für die rechte Fahrbahnlinie der Fall, wie sich in Abbildung 15 erkennen lässt. Hier sieht man, dass sich der Offset  $\omega$  stark ändert, auch wenn sich der Winkel der Geraden nur leicht ändert.

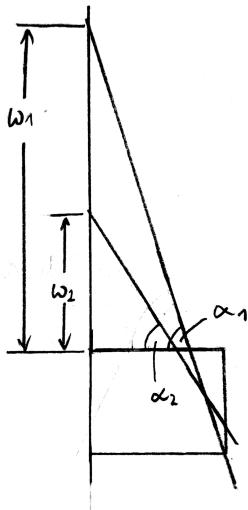


Figure 15: Beispiel für die erste Form der Geradenrepräsentation

Daher wurden für die beiden Geraden zwei verschiedene Repräsentationsformen gewählt. Für die Linke gerade wird der Offset von der linken oberen Ecke gemessen, für die rechte der Offset von der rechten oberen Ecke. Die Winkel werden wie gehabt zur x-Achse gemessen. In dieser Form lassen sich die Geraden am besten vergleichen. Die Umformung von der  $\rho$ - $\theta$ -Form geschieht in der Funktion `get_offset_alpha2` durch folgende Berechnungen:

$$\alpha = 90^\circ - \theta$$

Für Winkel  $\theta < 90^\circ$  (linke Fahrbahnlinie):

$$\omega = -\frac{\rho}{\cos 90^\circ - \theta}$$

Für Winkel  $\theta > 90^\circ$  (rechte Fahrbahnlinie):

$$\omega = \frac{B + \frac{\rho}{\cos \theta - 90^\circ}}{\tan 90^\circ - \theta}$$

Mit der Bildbreite B.

Abbildung 16 veranschaulicht diese Repräsentationsform.

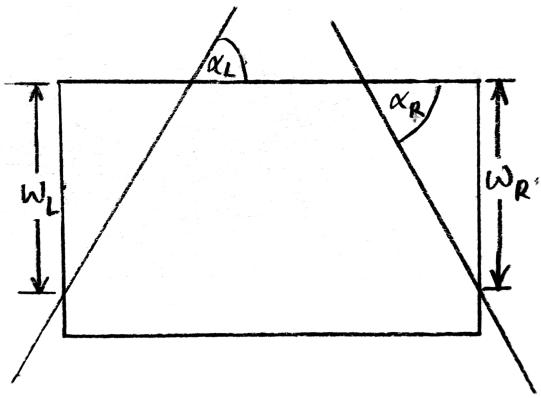


Figure 16: Beispiel für die erste Form der Geradenrepräsentation

## 8.2 Berechnung des Schnittpunktes der beiden Geraden

Der Lenkwinkel wird mithilfe der x-Koordinate des Fluchtpunktes, also des Schnittpunktes der beiden Geraden berechnet. Die Abweichung dieses x-Wertes vom Mittelpunkt kann als Regeldifferenz verwendet werden.

Die x-Koordinate des Fluchtpunktes (engl. vanishing point) wird in der Funktion get\_vp mit folgender Formel berechnet:

$$vp_x = \frac{\omega_R - \omega_L}{\tan \alpha_L + \tan \alpha_R}$$

Um die Abweichung vom x-Wert der Bildmitte zu erhalten, wird von diesem Wert die Hälfte der Bildbreite B abgezogen:

$$Dx = vp_x - \frac{B}{2}$$

## 9 Ansteuerung von Servo und Motorcontroller

Motorcontroller und Servo werden mithilfe eines Adafruit PCA9685 16-Kanal Servo Treibers angesteuert. Dieser nimmt Befehle des Raspberry Pi's über I<sup>2</sup>C entgegen und wandelt sie in ein PWM-Signal um. Das PWM Signal hat eine Periodendauer von 20ms, die Einschaltzeitdauer  $T_{on}$  liegt zwischen 1ms und 2ms.

Der Servotreiber hat eine eigene Bibliothek aus der am wichtigsten die Funktion set\_pwm(kanal, on,off) ist. Als Kanal wurde für den Servo 0 gewählt, für den Motorcontroller 1. Der Einschaltzeitpunkt "on" wird immer auf 0 gesetzt. Für den

Auschaltzeitpunkt kann theoretisch eine Zahl zwischen 0 und 4095 gesetzt werden. Da beim Servo das On-Signal zwischen einer und zwei Millisekunden lang sein muss ergibt sich der theoretische Wert folgendermaßen:

$$off = T_{on} \cdot \frac{20ms}{4096}$$

$$off_{min} = 1ms \cdot \frac{20ms}{4096} = 204,8$$

$$off_{max} = 2ms \cdot \frac{20ms}{4096} = 409,6$$

Für das verwendete Auto wurde der Min- und Maxwert empirisch ermittelt indem die geschaut wurde, bei welchen Werten die Räder bis zum Anschlag ausgelenkt sind. Es wurden folgende Werte ermittelt:

Vollausschlag rechts: 272

Vollausschlag links: 342

Mittelposition: 307

Für den Vollausschlag nach rechts und links wurden die Werte 297 und 317 Verwendet, damit nicht zu scharfe Kurven gefahren werden.

Ebenso wurden für den Motor die Werte für den off-Parameter durch Ausprobieren ermittelt, indem geschaut wurde, bei welchen Werten das Auto mit angemessener Geschwindigkeit fährt. Hier wurden ermittelt:

Vorwärts: 320

Rückwärts: 295

Stop: 307