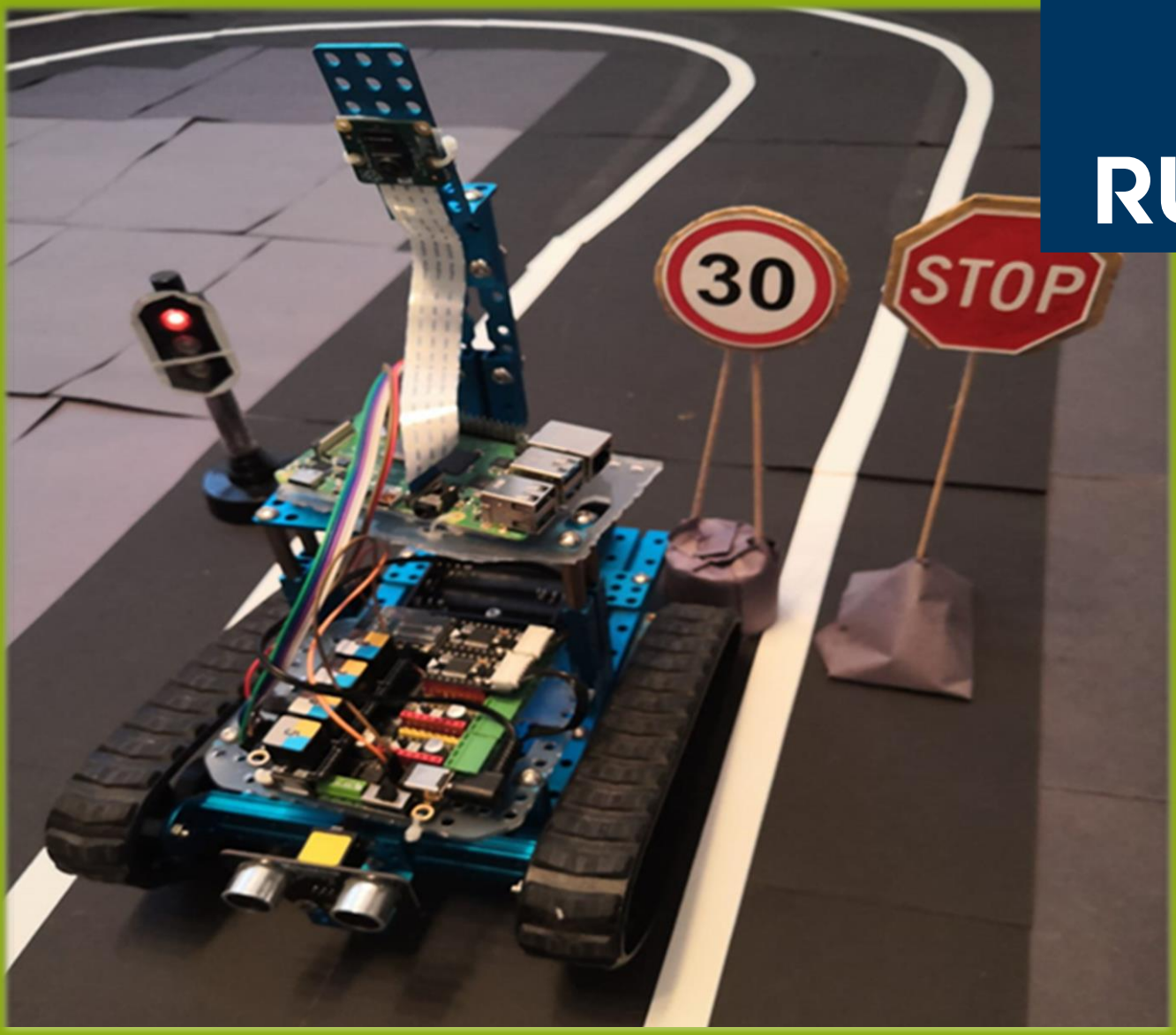

RUB

RUHR-UNIVERSITÄT BOCHUM

Sant Tokatlioglu

Computerversion-basierte Steuerung und Navigation von Lernrobotern

MASTER ARBEIT
02/2020

DIGITAL ENGINEERING
Univ.-Prof. Dr.-Ing. Detlef Gerhard

DIGITAL ENGINEERING
Univ.-Prof. Dr.-Ing. Detlef Gerhard

RUHR
UNIVERSITÄT
BOCHUM

RUB

Name **Sant Tokatlioglu**
Matr.-Nr. **1080 017 256 693**

Masterarbeit

Computerversion-basierte Steuerung und Navigation von Lernrobotern

Betreuer: Pascalis Trentsios, M.Sc.
Erstprüfer: Univ.-Prof. Dr.-Ing. Detlef Gerhard
Zweitprüfer: Dr.-Ing. Mario Wolf

Bochum, 18. Februar 2021

Abstract

Untersuchungen haben ergeben, dass Verkehrsunfälle hauptsächlich durch die Fehler der Fahrer verursacht werden. Eine große Anzahl von Menschen gefährden ihr eigenes und anderes Leben im Verkehr. Emotionale Verwirrung, Müdigkeit, Fahrlässigkeit oder Alkoholkonsum gehören zu den Hauptverursachern von Unfällen.

Ein vorübergehender Verlust der Lenkkontrolle aus diesen Gründen führt zu einer hohen Anzahl von Opfern. Menschen, die die Verkehrszeichen mit einem kurzen Augenblick der Nachlässigkeit ignorieren, gefährden nicht nur selbst, sondern auch andere Teilnehmer im Verkehr.

Ein weiteres wesentliches Ziel dieses Projektes galt der Einrichtung eines Systems das auch eingeschränkten und ältere Menschen eine Unterstützung leisten kann. Diese Menschen können sich zum Beispiel Aufgrund mangelnder Reflexe an der sich ständig ändernden Verkehrslage nicht optimal orientieren, was für sie und für andere Verkehrsteilnehmer eine große Gefahr bilden kann. Dieses System wird das Leben der Menschen erleichtern und den Verlust von Menschenleben minimieren.

Der Lernroboter ist so konstruiert, dass er die Schilder und Hindernisse in der nächsten Umgebung mit der Kamera erfasst, mit dem Abstandssensor erkennt und sich innerhalb des Fahrstreifens bewegt.

Darüber hinaus kann dieses System mit geringfügigen Änderungen im Bereich des Militär- und Pakettransports eingesetzt werden.

Danksagung

An dieser Stelle möchte ich an Herrn Wolf und Herrn Trentsios meinen Dank aussprechen, die mich beim Aufbau dieses Projekts betreut und unterstützt haben.

Mein besonderer Dank gilt meiner Lebensgefährtin Gamze Bulgen, die mir wie in allen Bereichen meines Lebens, auch bei allen Etappen meiner Diplomarbeit eine große Unterstützung geleistet hat.

Darüber hinaus möchte ich an dieser Stelle auch an meine Eltern Ani und Vrej Tokatlioglu, den ich alles zu verdanken habe und die mich stets mit großer Geduld und Verständnis umsorgt hat, meinen Bruder Sahen Tokatlioglu, der mir mit seiner Unterstützung und Motivation beharrlich zur Seite gestanden hat, sowie der Familie Bulgen, für ihre stetige Unterstützung und Hilfestellung und besonders Herrn Özcan Bulgen, der mich bei der Niederschrift meiner Diplomarbeit unterstützt hat einen herzlichen Dank richten.

Inhaltsverzeichnis

1 Einleitung

Bei der Vermeidung von Umweltverschmutzung spielen elektrische Fahrzeuge eine große Rolle, da sie keine Abgasemissionen verursachen. Außerdem reduzieren diese Fahrzeuge die Abhängigkeit von fossilen Brennstoffen und erhöhen die Bedeutung erneuerbarer Energiequellen. Autonome Fahrzeugsysteme, die sowohl zum Schutz der Umwelt als auch zum Schutz der Menschen entwickelt wurden, werden fortlaufend weiterentwickelt. Die Tatsache, dass diese Systeme leichter in Elektrofahrzeugen integriert werden können, erhöhen das Interesse der Menschen an diesen Entwicklungen.

Dieses Thema wurde ausgewählt, weil es das Leben von Menschen im Verkehr retten kann. Außerdem kann dieses System das Leben von Menschen erleichtern und erhöhte Sicherheit in Verkehr gewähren. Das Hauptziel des Projekts ist die Schaffung eines sicheren, billigen und ausrüstungsarmen Systems, das auf alle Fahrzeuge angewendet werden kann.

1.1 Aufgabenstellung

Im Rahmen des Systems wird sichergestellt, dass das Fahrzeug den Ampeln und sonstigen Verkehrszeichen folgt, ein Hindernis darin sieht und dabei die Fahrspur auf gebogenen und geraden Straßen nicht verlässt. Um ein entsprechendes System einrichten zu können, wurde folgende Frage gestellt:

Ist es möglich, das Auto mit einer einzigen Kamera und einem Abstandssensor auf der Fahrspur zu halten?

Zur Beantwortung dieser Leitfrage und Zwecks Optimierung des Fahrverhaltens des Fahrzeugs bedarf es der Beantwortung folgender Aspekte, die unter drei Überschriften wie folgt zusammengefasst werden:

- Welche Auswirkungen haben Kameraauflösung und FPS-Werte auf das System?
- Kann die Objektidentifikation effizienter gestaltet werden?
- Wie wirkt sich die Ergänzung einer PID-Regelung zum System auf die Spurverfolgung des Systems aus?

1.2 Ziel der Arbeit

Mit der sich kontinuierlich entwickelnden Technologie besteht der Hauptzweck der Fahrzeuge darin, dem Fahrer ein komfortableres und sichereres Fahrerlebnis zu bieten.

Ziel des Projekts ist es nicht, ein vollständiges autonomes Fahrzeug zu entwickeln, sondern sicherzustellen, dass sich das Fahrzeug durch eine autonome Optimierung

der Fahrstrecke zwischen den Linien, die die Fahrspuren definieren, bewegt und während der Fahrt die Verkehrsregeln einhält. Außerdem wird sichergestellt, dass das Fahrzeug sicher anhält, wenn vor dem Fahrzeug ein Objekt identifiziert wird. Um dieses System zu erstellen, werden nur die mindest erforderlichen Komponenten verwendet und übermäßige Teile werden nicht verwendet.

1.3 Test Aufbau

Eine Teststrecke wurde erforderlich, um den codierten Roboter zu testen. Die Teststrecke wird in Übereinstimmung mit Autobahnstrecken entworfen, damit der Roboter in der straßenähnlichen Strecke geprüft werden kann. (siehe Abbildung 1).

Die Teststrecke ist mit scharfen Kurven ausgestattet, um das Verhalten des Fahrzeugs unter erschwerten Straßenbedingungen zu testen. Ergänzend wurde bei Testaufbau ein variabler Abstand der Spurlinien verwendet (für mehr Informationen siehe 1.4 Testaufbau). Das Fahrverhalten des Fahrzeugs sollte mit diesen variablen Straßenbedingungen geprüft und optimiert werden. Damit sollte gewährleistet werden, dass das Fahrzeug unter allen erschwerten Bedingungen eine sichere Fahrt bietet.

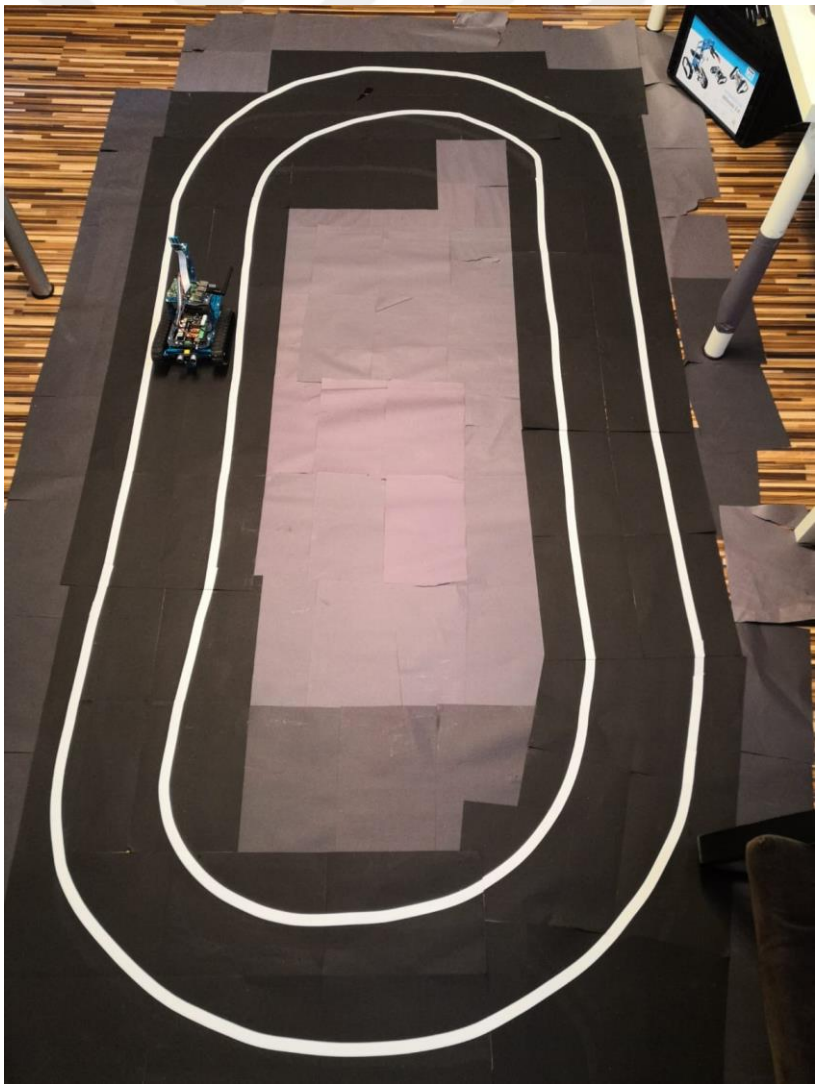

Abbildung 1 : Teststrecke

1.4 Anpassung der neuen Strecke

Nach erfolgreicher Testung der in Kapitel 1.3 gezeigten Teststrecke ist vorgesehen, dass der Roboter auf unterschiedliche Bedingungen getestet wird. Aus diesem Grund wurde eine Teststrecke bevorzugt, die an der Universität für Roboterwettbewerbe (siehe Abbildung 2) genutzt wird. Aufgrund der sehr scharfen Kurven und der sich unterscheidenden Farbe der Fahrspuren (ursprünglich weiß), die sich auf der neuen Teststrecke befinden, sind entsprechende Änderungen im Programm erforderlich, die in Abschnitt 3.3.8 ausführlich erläutert werden.

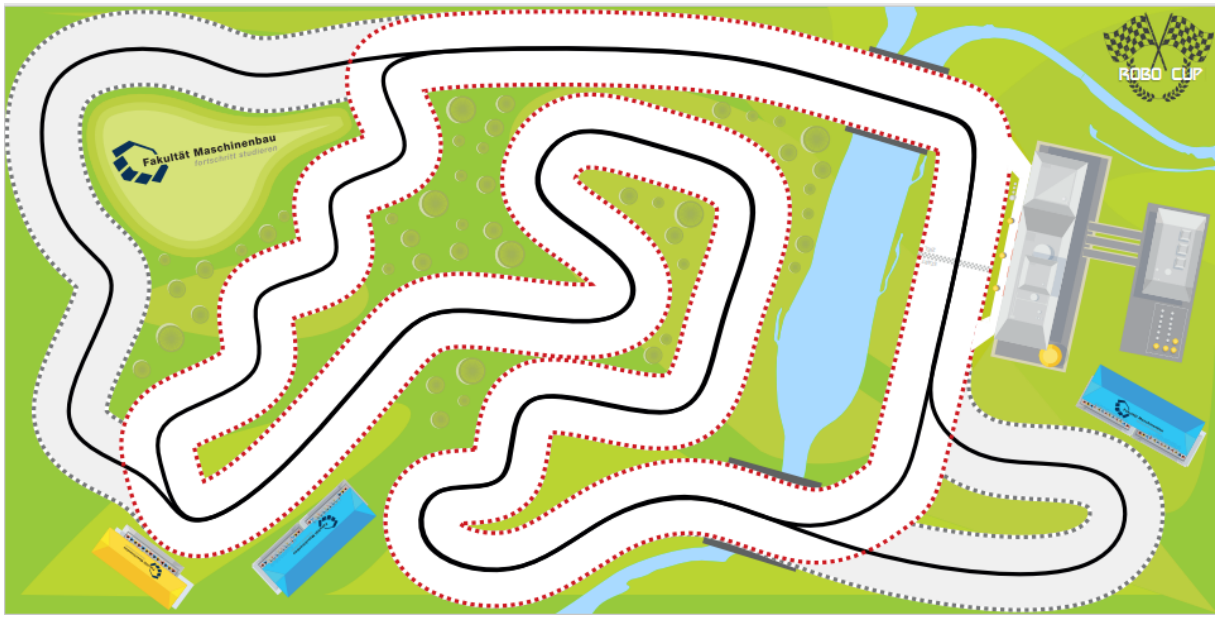

Abbildung 2 : Test Strecke an der Uni

2 Grundlagen

2.1 RGB, HSV und HSL Farbmodell

2.1.1 RGB

Die Definition von Farben in der digitalen Umgebung spielt eine große Rolle bei der Verarbeitung des Bildes. Mit Hilfe dieser Definitionen kann eine farbbasierte Trennung in dem zu bearbeitenden Bild vorgenommen werden. Das häufigste davon ist das RGB-System.

Das RGB-Farbmodell basiert auf einem kartesischen Koordinatensystem und wird als Würfel dargestellt. In diesem dreidimensionalen System können die gewünschten Farben mit den Werten der roten, grünen und blauen Farben zwischen 0 und 255 erhalten werden. Diese Werte, die von den Farben Rot, Grün und Blau übernommen werden, geben an, wie viel diese Farben in der ausgewählten Farbe enthalten sind. Auf diese Weise können alle Farben durch Mischung von RGB Farben erhalten werden.

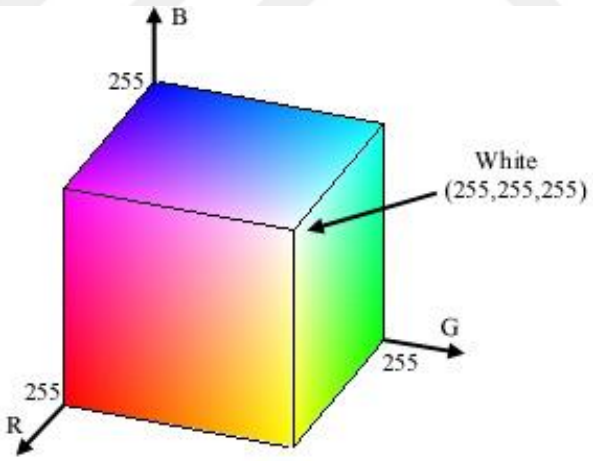

Abbildung 3 : RGB Farbmodell (MathWorks, 2021)

2.1.2 HSV – HSL

HSV und HSL sind Farbmodelle ähnlicher Typen mit höherer Effizienz als RGB. HSV symbolisiert „Farbton, Sättigung, Wert“ und HSL „Farbton, Sättigung, Helligkeit“.

Die Gründe für die Verwendung dieser beiden Farbmodelle sind dieselben. Mit Hilfe eines größeren Maßstabs ist es einfacher, Farben zu finden und diese Farbtöne besser auszudrücken. Es gibt jedoch einige Unterschiede zwischen ihnen. Der wichtigste Unterschied ist, dass der Helligkeitswert in HSL sehr unterschiedlich ist. Je näher man im HSL-Farbmodell von unten nach oben kommt, desto hellere Farben

werden erzielt. Beim HSV wird die Farbe von außen nach innen heller. (Rafael C. Gonzalez & Woods, 2004)

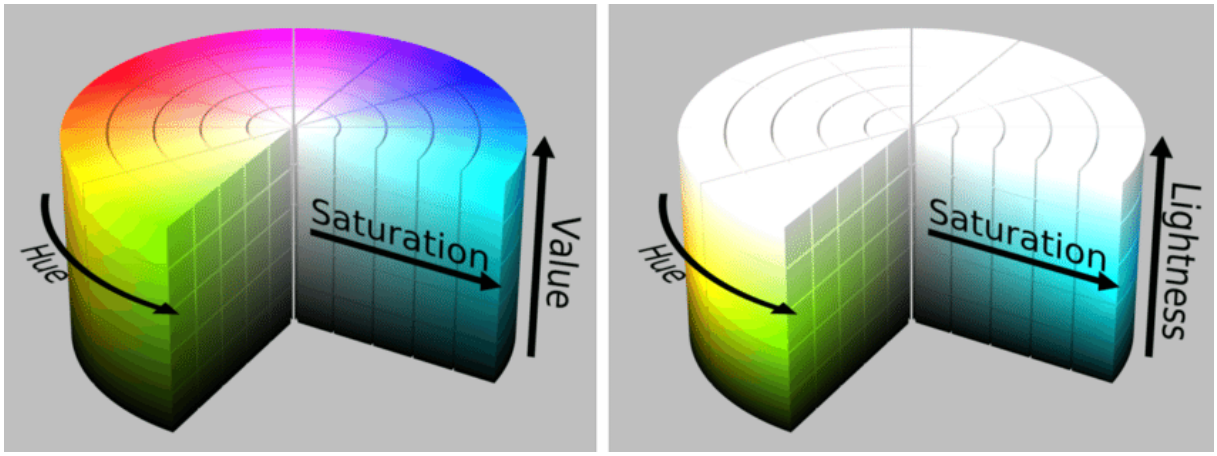

Abbildung 4 : HSV und HSL vergleich (Wikipedia, 2021)

2.2 Canny Edge Detection

Die Kantenerkennung ist der erste Schritt in vielen Computer Vision Algorithmen. Sie wird verwendet, um scharfe Diskontinuitäten in einem Bild zu identifizieren Bild, wie z. B. Änderungen in der Helligkeit oder in der Intensität aufgrund von Änderungen in der Szenenstruktur. Die Kantendetektion wurde intensiv erforscht. (Gentsos, Sotiropoulou, & Nikolaidis, 2010)

Der traditionelle Canny-Algorithmus zur Kantenerkennung ist empfindlich gegenüber Rauschen und verliert daher leicht schwache Kanten Informationen zu verlieren, wenn das Rauschen herausgefiltert wird, und seine festen Parameter zeigen eine schlechte Anpassungsfähigkeit. (Bao, Zhang, & Wu, 2005)

Die Kanteninformation eines Bildes ist eine der wichtigsten Informationen in einem Bild, die den Umriss des Ziels beschreiben kann, die relative Position innerhalb des Zielbereichs und andere wichtige Informationen. Die Kantenerkennung ist einer der wichtigsten Prozesse in der Bildverarbeitung, und die Erkennungsergebnisse haben Erkennungsergebnisse wirken sich direkt auf die Bildanalyse aus. Die traditionellen Kanten Algorithmen zur Kantenerkennung erfolgen durch die Erkennung des maximalen Wertes der ersten Ableitung oder des Nulldurchgangs der zweiten Ableitung.

Die Bildqualität wird durch einige Faktoren wie Rauschen und Beleuchtung im Prozess der Bildaufnahme beeinflusst. Außerdem ist der lokale Kontrast bei den Bildern, deren lokaler Kontrast bei jedem Pixel unterschiedlich ist, sind die Parameter des traditionellen Canny-Kantenerkennungs-Algorithmus sind fest und können nicht an den Kantenerkennungsprozess unter unterschiedlichen Bedingungen anpassen. (Rong, Li, & Sun, 2014)

Daher wurde der Gradientenberechnungsoperator genutzt, um nützlichere Detailkanten zu erhalten und robuster gegen Rauschen zu sein.

Damit die Kantenerkennung besser definiert werden kann und da die Kantenerkennung für Bildrauschen anfällig ist, besteht der erste Schritt darin, das Bildrauschen mit einem Gauß-Filter zu entfernen. Durch Verringern der Bildauflösung wird außerdem der verrauschte Bereich zu einem durchschnittlichen einzelnen

Pixelwert ermittelt, während gleichzeitig die Ausführung verringert wird, die für den Kantenerkennungsalgorithmus benötigt wird, da weniger Daten verarbeitet werden müssen.

Nach dem Glätten ist der nächste Schritt die Berechnung des Farbverlaufs des Bildes. Die Gradienten Berechnung führt zu der Erkennung der möglichen Kantenstärke und -richtung. Das ist ausgeführt durch eine andere Faltung mit einem Gradienten Operator (2.1.1).

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.1.1}$$

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \tag{2.1.2}$$

$$\theta = \arctan\left(G_x / G_y\right) \tag{2.1.3}$$

Nach dem obengenannten Rechenschritt (2.1.1) und (2.1.2) wird die maximale Unterdrückung nicht erreicht, die mit der Kantendicke reduziert und Lokalisierung verbessert wird.

Nach nicht maximaler Unterdrückung wird das Bild möglicherweise noch einige falsche Ausgaben enthalten. Die falschen Ausgaben, die als "Streifen" bezeichnet werden, können durch die Verwendung von Hysterisis Thresholding beseitigt werden. (Gentsos, Sotiropoulou, & Nikolaidis, 2010)

Hough Transform ist eine beliebte Technik, um jede Form zu erkennen, wenn Sie diese Form in mathematischer Form darstellen können. Es kann die Form erkennen, auch wenn sie ein wenig gebrochen oder verzerrt ist.

2.3 Hough Line Transform

Die Hough-Transformation ist eine Technik, die verwendet werden kann, um Merkmale einer bestimmten Form innerhalb eines Bildes zu isolieren. Da sie erfordert, dass die gewünschten Merkmale in einer parametrischen Form angegeben werden, wird die klassische Hough-Transformation am häufigsten für die Erkennung von regelmäßigen Kurven wie Linien, Kreisen, Ellipsen usw. verwendet. Eine verallgemeinerte Hough-Transformation kann in Anwendungen eingesetzt werden, in denen eine einfache analytische Beschreibung eines Merkmals nicht möglich ist. Aufgrund der Berechnungskomplexität des verallgemeinerten Hough-Algorithmus wird sich der Schwerpunkt dieser Diskussion auf die klassische Hough-Transformation beschränken. Trotz ihrer Domäneneinschränkung findet die klassische Hough-Transformation nach wie vor viele Anwendungen, da die meisten gefertigten Teile Merkmalsgrenzen enthalten, die durch regelmäßige Kurven beschrieben werden können. Der Hauptvorteil der Hough-Transformation ist, dass sie tolerant gegenüber Lücken in der Beschreibung von Merkmalsgrenzen ist und relativ unbeeinflusst von Bildrauschen ist.

Kantenerkennungsalhorithmen können nicht garantieren, dass alle Kanten in einem Bild erkannt werden, wenn Rauschen oder Gradienten unterhalb von Threshold bleibt.

Um die Qualität der Kantenerkennung zu maximieren, wird ein Verknüpfungsalgorithmus entwickelt, der zu aussagekräftigen Kanten oder Bereichsgrenzen zusammenbinden kann. Die Kanten zu Linien verknüpfen, ist möglich, wenn Hough Transformation implementiert wird. (Rafael C. Gonzalez & Woods, 2004)

Eine Linie kann als Gleichung (2.3.1) oder in parametrischer Form als Gleichung (2.3.2) dargestellt werden, wobei ρ der senkrechte Abstand vom Ursprung zur Linie und θ der Winkel ist gebildet durch diese senkrechte Linie und horizontale Achse, gemessen gegen den Uhrzeigersinn.

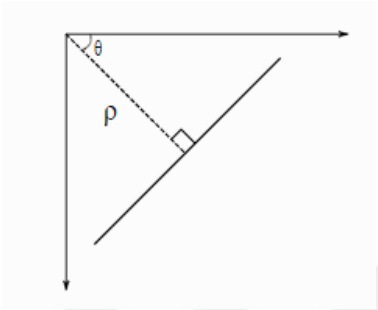

Abbildung 5 : Definition einer Linie

Wenn die Linie unter dem Ursprung verläuft, hat sie ein positives ρ und einen Winkel von weniger als 180. Wenn sie über den Ursprung verläuft, wird anstelle eines Winkels von mehr als 180 ein Winkel von weniger als 180 und ein ρ von einem negativen Winkel genommen. Jede vertikale Linie hat 0 Grad und horizontale Linien haben 90 Grad. (Mordvintsev & K, 2013)

$$y = mx + c \tag{2.3.1}$$

$$\rho = x\cos\theta + y\sin\theta \tag{2.3.2}$$

Eine Linie kann dann mit den Parametern θ und ρ in einen einzelnen Punkt im Parameterraum transformiert werden, das ist der sogenannte den Hough-Raum.

Falls statt einer Linie ein Pixel in einem Bild mit der Position (x, y) vorhanden ist, können unendlich viele Zeilen durch diesen einzelnen Pixel gehen. Unter Verwendung von Gleichung (2.3.2) können alle diese Linien in den Hough-Raum transformiert werden.

Diese Linien ergeben eine sinusförmige Kurve, die für dieses Pixel eindeutig ist. Die Verwendung derselben Methode für ein anderes Pixel führt zu einer anderen Kurve, die die erste Kurve an einem Punkt im Hough-Raum schneidet. Dieser Punkt repräsentiert die Linie im Bildraum, die durch beide Pixel verläuft. Dieses Verfahren soll für allen Pixeln auf dem Kantenerkennungsbild wiederholt werden, damit die Kanten erkannt werden können.

2.4 DC Motor Encoder

Der Gleichstrommotor ist ein kontinuierlicher Aktuator, der elektrische Energie in mechanische Energie umwandelt. Gleichstrommotoren werden häufig in vielen

industriellen Anwendungen eingesetzt, in denen große Drehzahlbereiche und zweiseitige Bewegung erforderlich sind.

Der Vorteil von Gleichstrommotoren kann die Drehzahlregelung sein, wie z. B. hohes Startdrehmoment, schnelles Ansprechverhalten, Tragbarkeit und Übereinstimmung mit vielen Arten von Regelungsverfahren. Daher werden Gleichstrommotoren in vielen Steuerungsanwendungen weit verbreitet verwendet, einschließlich Robotern, Elektrofahrzeuganwendungen, Scheibentreibern, Werkzeugmaschinen und Servoventilaktuatoren. (Ali, Mohammed, & Alwan, 2019)

Gleichstrommotor-Encoder werden zur Drehzahlregelung in Gleichstrommotoren verwendet, bei denen sich ein Anker oder Rotor mit gewickelten Drähten in einem von einem Stator erzeugten Magnetfeld dreht. Der Gleichstrommotor-Encoder bietet einen Mechanismus zu der Drehzahlmessung des Rotors und zur Rückmeldung an den Antrieb für eine präzise Drehzahlregelung.

2.5 PWM (Pulse Width Modulation)

Die Spannung, die an die Leitungen des Gleichstrommotors geliefert ist, kann durch Modulation der Versorgungsspannung mit Pulse Width Modulation (PWM) variiert werden. Das Prinzip hinter PWM ist, dass Ein- und Ausschaltung der Spannung zu einer durchschnittlichen Stromversorgung führt, die niedriger als der Nennwert der Stromversorgung liefert. Der wesentliche Vorteil ist, dass mit Digitalen Signalen Analoge Ergebnisse zu erzielen sind.

PWM eignet sich besonders für den Betrieb von Trägheitslasten wie Motoren, die von diesem diskreten Schalten nicht so leicht betroffen sind, da sie aufgrund ihrer Trägheit langsam reagieren. Die PWM-Schaltfrequenz muss hoch genug gewählt werden, um die Last nicht beeinflussen zu können. Außerdem muss die resultierende Wellenform, die von der Last wahrgenommen wird, so glatt wie möglich sein. (Böcker, 2016)

Der Prozentsatz der Zeit, bei der das Gerät eingeschaltet ist, wird als Duty Cycle genannt. Die Zeit zwischen den Impulsen wird als Periodenzeit bezeichnet. (siehe Abbildung 6)

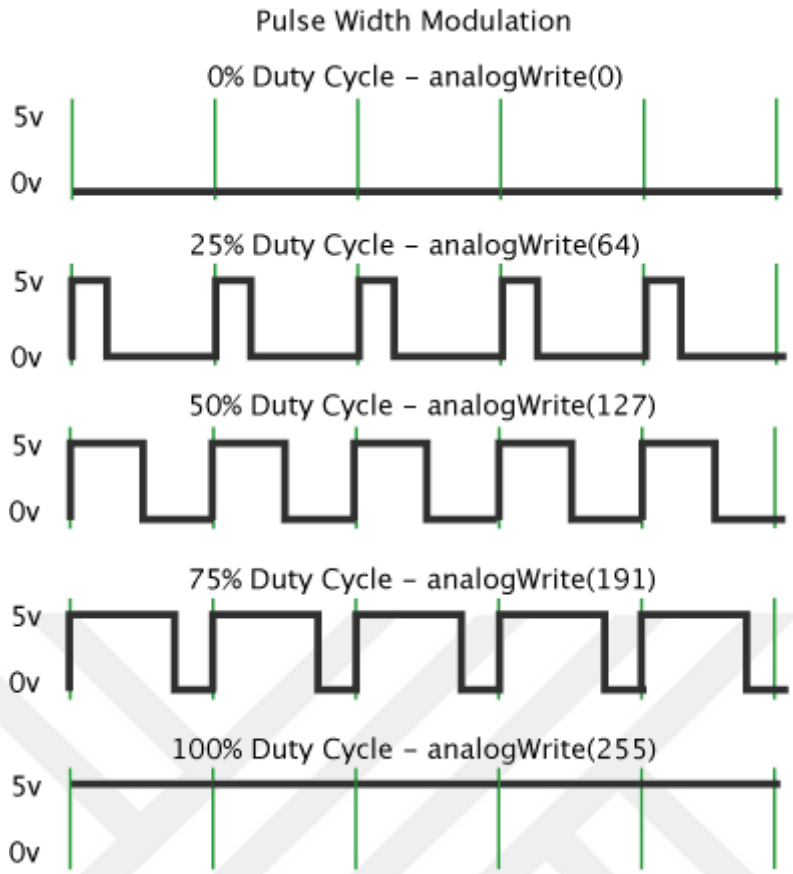

Abbildung 6 : PWM Signal (Hirzel, 2018)

2.6 Ultraschallsensor

Ultraschall-Abstandssensoren senden einen Impuls von Ultraschallwellen, die über dem von Menschen hörbaren Frequenzbereich liegen und erkennt, wann der Impuls von einem festen Objekt reflektiert wird. Die Entfernung zum Objekt „s“ kann mit der bekannten Schallgeschwindigkeit „v“ und die Zeit „t“, die benötigt wird, um den reflektierten Ultraschallimpuls vom Ultraschall-Abstandssensor zu erfassen, berechnet werden. Die Beziehung in Gleichung (2.6.1) unten wird verwendet, um die Entfernung „s“ zu bestimmen:

$$s = \frac{vt}{2} \tag{2.6.1}$$

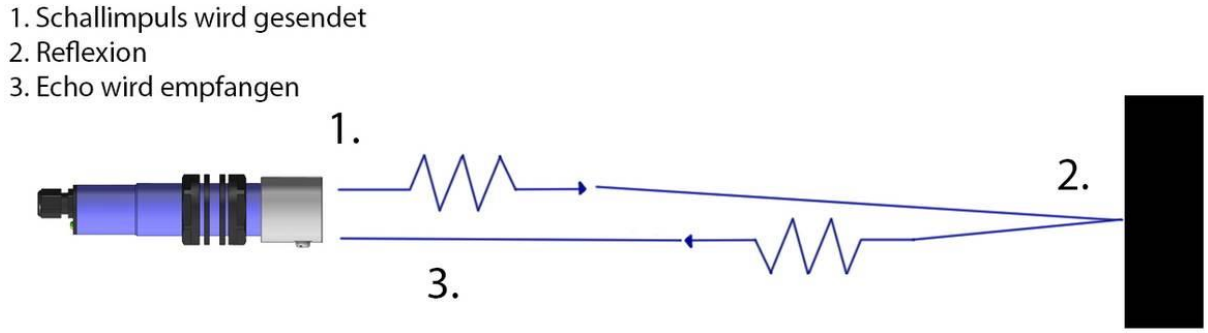

Abbildung 7: Ultraschallsensor Messprinzip (Waycon, 2020)

Die Schallgeschwindigkeit in Luft hängt vom Luftdruck und der Lufttemperatur ab. In der Nähe des Meeresspiegels und bei Raumtemperatur (circa 25 ° C) beträgt die Schallgeschwindigkeit ungefähr 343 m / s. Die Entwicklung des Demonstrators erfolgt unter der Annahme, dass diese Bedingungen zutreffen. Folglich wurde als zufriedenstellend angesehen, dass kein Temperatur Sensor vorhanden war.

2.7 Haar Cascade

Objekterkennung wird allgemein bezeichnet als eine Methode, die für entdecken und identifizieren von Objekten in einer bestimmten Klasse verantwortlich ist.

Eine Erweiterung der Objekterkennung kann durch Bildverarbeitung anhand digitaler Bilder zur Identifizierung von Objekten erreicht werden.

Normalerweise wird das Bild zuerst als eine niedrigere Stufe zur Verbesserung der Bildqualität verarbeitet. Das Rauschen wird vom Bild entfernt. Dann wird das Bild auf einer höheren Ebene verarbeitet, die Muster und Formen erkennt. Somit kann das vorher definierte Objekt im Bild gefunden werden. (Nagabhushana, 2005)

2.7.1 Erkennung durch Farbe

Eine Möglichkeit der Objekt-Erkennung besteht darin, Objekte in Bildern nach Farbe zu klassifizieren. Dieser Klassifikationstyp wird nicht überall genutzt, weil hierbei die ständige Notwendigkeit einer Neukalibrierung und Instandhaltung besteht.

2.7.2 Haar-like Erkennung

In dieser Methode wird die Erkennung von Objekten aus Bildern mit Funktionen oder bestimmten Strukturen des betreffenden Objekts gewährleistet.

Die alleinige Verarbeitung mit Bildintensitäten, die sich durch die RGB-Pixelwerte in jedem einzelnen Pixel im Bild ergeben, verlangsamt die Merkmalsberechnung auf den meisten Plattformen, weil es ziemlich rechenintensiv ist.

2.7.3 Cascade Classifier

Das System erkennt die betreffenden Objekte, indem es ein Fenster über das Bild bewegt. Jede Stufe des Klassifikators kennzeichnet den spezifischen Bereich, der durch die aktuelle Position des Fensters definiert ist, entweder positiv oder negativ.

Positiv bedeutet, dass sich ein angegebenes Objekt im Bild befindet und negativ bedeutet, dass sich das angegebene Objekt nicht im Bild befindet.

Wenn die Markierung ein negatives Ergebnis liefert, wird die Klassifizierung dieser bestimmten Region hier abgeschlossen und die Fensterposition wird zur nächsten Position verschoben.

Wenn die Kennzeichnung ein positives Ergebnis liefert, wechselt die Region zur nächsten Klassifizierungsstufe.

Der Klassifikator trifft eine endgültige Positive Entscheidung von der ersten bis zur letzten Stufe und liefert ein Ergebnis, ob das angegebene Objekt sich im Bild befindet. (Soo, 2014)

Es besteht ein Kompromiss zwischen weniger Stufen mit einer geringeren Falsch-Positiv-Rate pro Stufe oder mehr Stufen mit einer höheren Falsch-Positiv-Rate pro Stufe. Stufen mit einer niedrigeren Falsch-Positiv-Rate sind komplexer, weil sie eine

größere Anzahl von schwachen Lernern enthalten. Stufen mit einer höheren Falsch-Positiv-Rate enthalten weniger schwache Lerner. Im Allgemeinen ist es besser, eine größere Anzahl von einfachen Stufen zu haben, weil bei jeder Stufe die Gesamtfehlalarmrate exponentiell abnimmt. Wenn z. B. die Falsch-Positiv-Rate in jeder Stufe 50 % beträgt, dann ist die Gesamt-Falsch-Positiv-Rate eines Kaskadenklassifikators mit zwei Stufen 25 %. Bei drei Stufen beträgt sie 12,5 % und so weiter. Je größer jedoch die Anzahl der Stufen ist, desto mehr Trainingsdaten benötigt der Klassifikator. Außerdem steigt mit der Anzahl der Stufen die Falsch-Negativ-Rate. Diese Erhöhung führt zu einer größeren Wahrscheinlichkeit, dass eine positive Probe fälschlicherweise abgelehnt wird. Stellen Sie die Falsch-Positiv-Rate und die Anzahl der Stufen so ein, dass sich eine akzeptable Gesamt-Falsch-Positiv-Rate ergibt. Anschließend können Sie diese beiden Parameter experimentell abstimmen.

Das Training kann manchmal vorzeitig abgebrochen werden. Wenn Training nach sieben Stufen aufhört, obwohl die Anzahl der Stufen auf 15 eingestellt werden. Es ist möglich, dass die Funktion nicht genügend negative Proben erzeugen kann. Wenn die Funktion erneut ausgeführt und die Anzahl der Stufen auf sieben eingestellt wird, wird nicht das gleiche Ergebnis erhalten. Die Ergebnisse zwischen den Stufen unterscheiden sich, weil die Anzahl der positiven und negativen Proben, die für jede Stufe zu verwenden sind, für die neue Anzahl der Stufen neu berechnet wird. (The MathWorks, 2020)

2.7.4 PID Controller

Der PID-Regler ist die häufigste Form der Rückmeldung. Es ist ein wesentliches Element für Kontroll-Systeme. In der heutigen Prozesssteuerung sind mehr als 95% der Regelkreise des PID-Typen, die meisten Regelkreise sind tatsächlich PI-Regelkreise. PID-Regler sind heute in allen Bereichen zu finden, in denen die Regelung eingesetzt wird. Die Steuerungen gibt es in vielen verschiedenen Formen.

Viele ausgefeilte Steuerungsstrategien, wie z. B. die modellprädiktive Steuerung, sind ebenfalls hierarchisch organisiert. Die PID-Regelung wird auf der niedrigsten Ebene verwendet. Der multivariable Regler gibt die Sollwerte an die Regler auf der unteren Ebene weiter. Praktisch alle heute hergestellten PID-Regler basieren auf Mikroprozessoren. Dies hat die Möglichkeit gegeben, zusätzliche Funktionen wie automatische Abstimmung, Verstärkungsplanung und kontinuierliche Anpassung bereitzustellen. (Åström, 2002)

Eine typische Struktur eines PID-Steuerungssystems ist in Gleichung 2.7.4.1 dargestellt, wo zu sehen ist, dass in einem PID-Regler das Fehlersignal $e(t)$ verwendet wird, um die proportionalen, integralen und abgeleiteten Aktionen mit den resultierenden Signalen zu erzeugen gewichtet und summiert, um das Steuersignal $u(t)$ zu bilden, das auf das Systemmodell angewendet wird. Eine mathematische Beschreibung des PID-Reglers ist

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{d\, e(t)}{dt} \right] \qquad (2.7.4.1)$$

wobei $u(t)$ das Eingangssignal für das Anlagenmodell ist, ist das Fehlersignal $e(t)$ definiert als $e(t) = r(t) - y(t)$ und $r(t)$ ist das Referenzeingangssignal. (Xue, Chen, & Atherton, 2007)

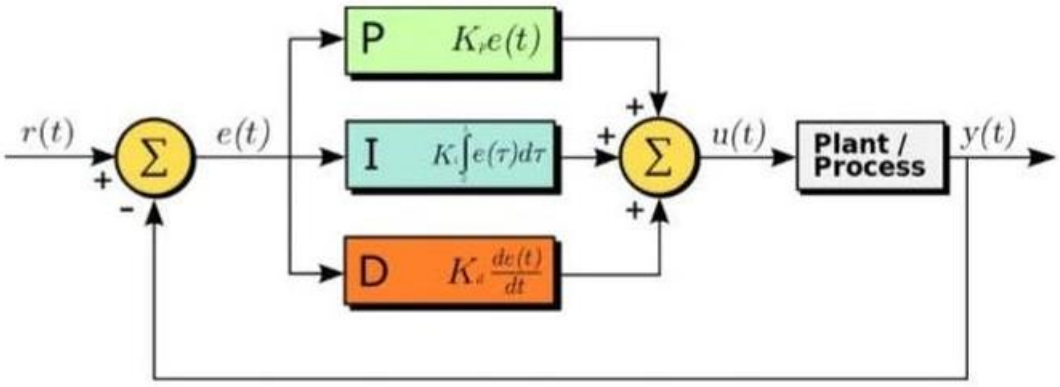

Abbildung 8 : PID Regler Konzept (Paktechpoint, 2018)

Der Proportionalteil K_p stellt die Verstärkung der Abweichung zwischen Soll- und Istwert dar. Eine große Zahl führt zu einem stabileren System und zu mehr Regelabweichung. Wenn Soll- und Istwert übereinstimmen, so wird als Stellgröße der Mittelwert zwischen minimaler und maximaler Stellgröße ausgegeben.

Der Integralteil K_i stellt die Stellgröße in Abhängigkeit von der aus dem Proportionalteil verbliebenen Abweichung periodisch nach. Eine große Zahl ergibt ein stabileres System, aber die Angleichung an den Sollwert erfolgt langsamer.

Der Differenzialteil K_d führt zu einer kurzfristigen “Überreaktion” je schneller eine Abweichung zwischen Soll- und Istwert auftritt, um schnellstmöglich einen Ausgleich zu erreichen. Hohe Werte ergeben ein stabileres System, aber es wird langsamer an den Sollwert angeglichen. (TA WIKI, 2018)

Das Verhalten, das durch die Auswahl dieser Werte beeinflusst wird, ist Überschwingen (Overshoot), statischer Fehler (Steady-State error), Einschwingzeit (Settling time) und Anstiegszeit (Rise time). Das Überschwingen wird durch Überregulieren des gewünschten Werts verursacht. Der statische Fehler (Steady-State error) ist ein konstanter Fehler, der über die Zeit bestehen bleibt. Die Einschwingzeit ist die Zeit, die die Sprungantwort eines Systems benötigt, um kontinuierlich innerhalb einer bestimmten Fehlergrenze vom gewünschten Wert zu halten. Die Anstiegszeit definiert, wie lange es dauert, bis der gewünschte Wert aus einem vorherigen stationären Zustand erreicht ist. Bei zu viel Überschwingen oder zu lange Anstiegszeit Bei Brems- und Lenkanwendungen kann es zu einem Szenario kommen, in dem ein Unfall auftritt, obwohl das System zunächst in die richtige Richtung arbeitet. Wenn ein statischer Fehler auftritt, bleibt das Fahrzeug möglicherweise ständig außerhalb der Mitte.

Ein integraler Windup tritt auf, wenn der Fehler den integralen Teil des PID- Regler Ausgangs kontinuierlich erhöht, bis die Gefahr besteht, dass der Regler Ausgang bei einer plötzlichen Änderung der Eingangsdaten zu einem erheblichen Überschwingen führt. Ein grundlegendes Verständnis darüber, wie sich eine Erhöhung der Werte K_p, $K_i$und K_d auf ein Steuerungssystem auswirkt, finden Sie in Abbildung 9.

Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Abbildung 9 : Einfluss von PID-Werten auf das System (Mike, 2018)

Die oben erläuterten Informationen werden eine sehr aktive Rolle bei der Erstellung des Projekts spielen und uns die Grundlage für die Entwicklung des Roboters liefern.

Im nächsten Abschnitt wird erklärt, was die im Roboter verwendeten Hardwares und Softwares sind und wofür sie verwendet werden und wie die oben genannten Probleme bei der Programmierung des Roboters behoben werden. Auf diese Weise werden ein besseres Verständnis des Themas und umfassende Informationen zur Programmierung vermittelt.

3 Anforderungen

Im nächsten Abschnitt werden die Anforderungen für die Erstellung des Roboters erwähnt. Diese Anforderungen können als Hardware, als Software, die Verbindungsherstellung zwischen Hardware und Software und schließlich als Kodierungssprache aufgeführt werden.

Um die bestmögliche Integration der tatsächlichen Straßenverhältnisse und das Verständnis der Ereignisse in der Umwelt sicherzustellen, wurde unter ähnlichen Bedingungen wie bei Autobahnen entwickelt. Somit sollte sichergestellt werden, dass das Fahrzeug ein besseres Verhalten während der Fahrt zeigen kann.

Für das Fahrzeug ist es wesentliche Aufgabe, alle Eingaben korrekt zu erhalten und durch die Verarbeitung dieser Daten die richtige Entscheidung zu treffen. Wenn diese Daten nicht exakt verarbeitet und die erforderlichen Ausgaben erstellt werden, kann der kleinste Fehler zu sehr großen und irreversiblen Problemen führen. Aus diesem Grund muss jede an das Fahrzeug angeschlossene Komponente harmonisch arbeiten. Die Aktionen, die das Fahrzeug ausführen muss, sind unten eingeordnet. (siehe Tabelle 1)

Tabelle 1 : Anforderungen des Systems

Nr	Funktionale Anforderungen des Systems
A01	Verbindung der Komponenten
A02	Aufladung des Betriebssystems auf RasypberryPi
A03	Notwendigen Bibliotheken herunterladen
A04	Aufladung MegaPi Firmware auf MegaPi
A05	Verbindungsherstellung zwischen MegaPi und Raspberry Pi
A06	Aufnahmen von der Kamera
A07	Auflösung und FPS definieren
A08	Region Of Interest (ROI) bestimmen
A09	Erstellung der Vogelperspektive
A10	Änderung der Farbskala
A11	Weißen Spurlinien herausfiltern
A12	Erkennung den Spurlinien
A13	Optimierung der Fahrspurerkennung
A14	Integration des Ultraschallsensors
A15	Erkennung der Verkehrsschilder

3.1 Hardwareanforderungen

Das System benötigt grundsätzlich zwei Eingänge, um eine sichere Fahrt zu gewährleisten. Dies sind die Position der Fahrspurlinien und ob sich vor dem Fahrzeug Hindernisse befinden. Die beiden Teile, die zur Bereitstellung dieser Eingänge verwendet werden, sind die Kamera und der Ultraschallsensor.

Die Kamera muss mit dem Raspberry Pi verbunden sein, damit das von der Kamera aufgenommene Bild verarbeitet, die Fahrspuren gefunden und die Verkehrszeichen in der Nähe identifiziert werden können. Auf diese Weise kann das Fahrzeug dazu gebracht werden, auf die Umgebung zu reagieren. Der Ultraschallsensor wird verwendet, um ein Hindernis vor dem Fahrzeug zu erkennen und zu verstehen, wie weit es entfernt ist. Es kann an Raspberry Pi oder MegaPi verbunden werden, um die von diesem Sensor erhaltenen Informationen anzuzeigen. Da in diesem Projekt der Ultraschallsensor der Firma Makeblock verwendet wird, ist der Sensor direkt mit dem MegaPi verbunden.

Zusätzliche Komponenten, die erforderlich sind, um das Fahrzeug in Bewegung zu setzen und auf kurvenreichen Straßen innerhalb der Fahrspur zu halten; 2 Motoren und 2 Motortreibern, die zum Antrieb von den Motoren verantwortlich sind. Zuerst wird der Motorantrieb mit dem MegaPi verbunden, dann werden die Motorverbindungen mit dem Motorantrieb verbunden und die erforderlichen Teile werden fertiggestellt. Die folgende Tabelle zeigt, was diese Teile sind und wie viele von denen verwendet werden.

Tabelle 2 : Stückliste der Hardware

Nr	Komponent	Stück
K01	Raspberry Pi 4	1
K02	Makeblock Megapi	1
K03	Raspberry Pi Kamera Module	1
K04	Makeblock Ultraschallsensor	1
K05	Makeblock Optical Encoder Motor	2
K06	Makeblock Megapi Encoder/DC Motor Driver	2

3.2 Anforderungen für die Verbindung der Hardware

Während die Komponenten mit dem MegaPi oder Raspberry Pi verbunden sind, sollten die Eingangs- und Ausgangsstifte entsprechend den Sensoren ausgewählt werden. In Anbetracht der Eigenschaften ausgewählten Komponenten liefern bessere Ergebnisse, da die speziell für die Anforderungen produziert sind.

Während der Programmierung sollte angegeben werden, an welche Pins jede Komponente verbunden ist. Daher sollte die Anzahl dieser Pins und die installierten Komponenten notiert werden. Wenn diese Definitionen bei der Programmierung falsch angegeben sind, funktioniert das Programm nicht.

Die Verbindung der Komponenten, die für den Systembetrieb erforderlich sind, wird auf der Tabelle 3 gezeigt.

Tabelle 3 : Verbindungsanforderungen der Hardware

Nr	Funktionale Anforderungen der Hardwares
V01	Raspberry Pi und Kamera Module Verbindung
V02	Raspberry Pi und MegaPi Verbindung
V03	MegaPi und DC Motor-Driver Verbindung
V04	DC Motor-Driver und Encoder Motor Verbindung
V05	Encoder Motor und Reifen Verbindung
V06	Ultraschall sensor und Megapi Verbindung

3.3 Softwareanforderungen

Raspberry Pi enthält bei der ersten Verwendung kein Betriebssystem. Dazu muss zunächst das Betriebssystem ausgewählt werden und diese Auswahl entsprechend den Anforderungen des herzustellenden Projekts getroffen werden. Es ist wichtig, das richtige Betriebssystem auszuwählen. Das in diesem Projekt ausgewählte Betriebssystem ist Raspbian.

Raspberry Pi unterstützt C, C++, Python und ähnliche Sprachen. Bei der Auswahl, der im Projekt verwendeten Programmiersprache wurde die Programmiersprache als Python ausgewählt, da sie die Mega Pi Python Sprache unterstützt. Darüber hinaus wurde die OpenCV-Bibliothek, die große Vorteile bei der Bildverarbeitung bietet, ursprünglich für die Programmiersprache C erstellt. Aber mit verschiedenen Konfigurationen kann diese Bibliothek jedoch für Python verfügbar gemacht werden. Dazu wird zuerst Python auf Raspberry Pi geladen, dann wird OpenCV über die Python-Shell heruntergeladen und Raspberry Pi wird gebrauchsfertig gemacht. Dies macht Raspberry Pi mithilfe der Python-Sprache und der OpenCV-Bibliothek programmierbar. Auf der Tabelle 4 wird gezeigt, welchen Softwares und welche Versionen davon verwendet wurden.

Tabelle 4 : Versionen von genutzten Softwares

Nr	Anforderungen der Softwares	Version
S1	Python	v2.7.16
S1.1	Open-CV Bibliothek (für Python)	v4.5.0-pre
S1.2	MegaPi Bibliothek (für Python)	v0.2.2
S1.3	Numpy Bibliothek (für Python)	v1.16.2
S1.4	Picamera Bibliothek (für Python)	v1.13
S2	Arduino IDE	v1.8.13
S2.1	Makeblock Bibliothek (für Arduino IDE)	v3.27

Um zusammenzufassen, wofür die oben genannte Software und Bibliotheken verwendet werden.

- Python ist die Hauptprogrammiersprache, auf der alle Bibliotheken installiert werden.
- Die Open-CV Bibliothek enthält Bildverarbeitungsbefehle.
- Die MegaPi-Bibliothek ermöglicht die Kommunikation zwischen Python und Megapi. Das heißt, der in Python geschriebene Befehl wird auf das MegaPi übertragen und steuert die Ein- und Ausgänge.

- Die Programmiersprache Python ist nicht für numerische Berechnungen optimiert. Um diesen Mangel auszugleichen, kommen Bibliotheken wie Numpy zum Einsatz. Diese Bibliothek vereinfacht die Verwendung von Arrays und ermöglicht mehrdimensionale Array-Operationen. Die Funktionen und Operatoren der Bibliothek sind speziell hierfür optimiert. (Luber & Litzel, 2018)
- Picamera Bibliothek ermöglicht die Bilderfassung von einer an RaspberryPi angeschlossenen Kamera.
- Mit Hilfe von Arduino IDE Software können die Kodes auf MegaPi aufgeladen werden. Dieses Programm ermöglicht, dass MegaPi von RaspberryPi gesendete Coden erhalten und verstehen kann.
- Makeblock Bibliothek wird durch der Arduino IDE auf MegaPi geladen und ermöglicht die Übersetzung von in Python geschriebenen Befehlen in eine Sprache, die vom MegaPi verstanden werden kann.

3.4 Anforderungen für die Kodierungssprache

Raspberry Pi und Mega Pi, die das Herzstück des Fahrzeugs bilden, müssen in derselben Programmiersprache zusammenarbeiten, damit nur im Raspberry Pi in Python Sprache geschriebene Kode beide Komponenten Steuern kann. Andernfalls werden beide Komponenten separat programmiert, und jedes Problem bei der Übertragung der Informationen, die sie miteinander kommunizieren müssen, führt dazu, dass das Fahrzeug nicht richtig reagiert oder sogar nicht funktioniert. Um dies zu verhindern, müssen diese zwei Komponente mit Kabeln miteinander verbunden und die direkte Kommunikation und Steuerung der beiden Geräte untereinander sichergestellt werden, indem die erforderlichen Codes auf das Mega Pi hochgeladen werden. Die für die Kommunikation der Geräte erforderlichen Schritte werden in Kapitel 4.3.2 ausführlich erläutert.

4 Methodik

4.1 Problemstellung

Um die in der Aufgabenstellung unter 1.1 geschilderten Fragen in der Praxis umzusetzen und beantworten zu können wurde im Rahmen dieses Projekts ein autonomes Fahrzeug konstruiert, welches die Verkehrszeichnen erkennt, etwaige Hindernisse im Verkehr wahrnimmt und sich innerhalb der Fahrspur bewegt.

In diesem Kapitel wird beschrieben, welche Hardware genutzt wurde, damit das konstruierte Fahrzeug die von Umfeld empfangene Daten verarbeitet, wie diese funktioniert und programmiert wurde.

4.2 Hardware

Bei der Bildung des Systems wurden nur eine Kamera und ein Ultraschallsensor verwendet. Die vollständige Hardwareübersicht und die Verbindung der einzelnen Komponenten sind in der Abbildung 10 dargestellt.

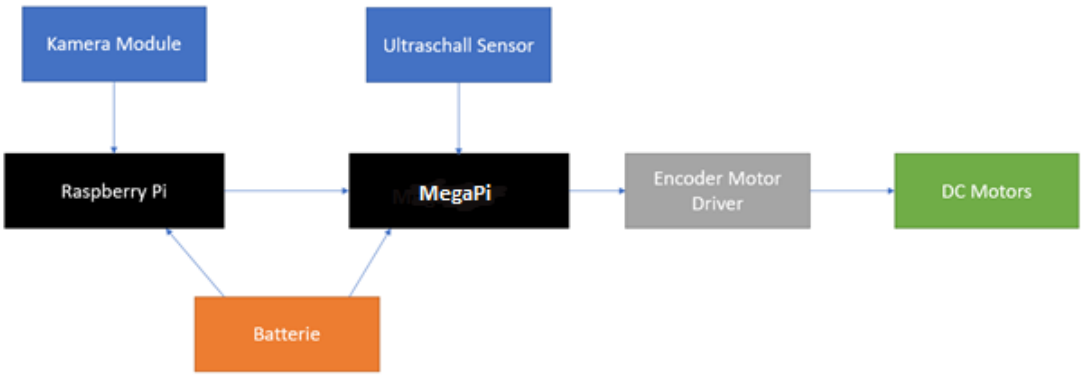

Abbildung 10 : Hardwareübersicht

4.2.1 Raspberry Pi 4

Raspberry Pi ist ein Einplatinencomputer. Aufgrund seines günstigen Preises und seiner Portabilität ist er in vielen Bereichen weit verbreitet. Wegen seiner USB- und HDMI-Eingänge bietet er sehr umfangreiche und praktische Anwendungsmöglichkeiten und hat Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz CPU.

GPIO-Pins können nicht nur als einfache softwaregesteuerte Ein- und Ausgänge verwendet werden, sondern unterstützen auch verschiedene andere dedizierte Peripherieblöcke wie I2C, UART und SPI.

Dieser Einplatinencomputer kann auch drahtlos verbunden werden, sodass das System über SSH gesteuert werden kann. Fernzugriff kann zu Raspberry Pi durch das SSH-System gewährt werden.

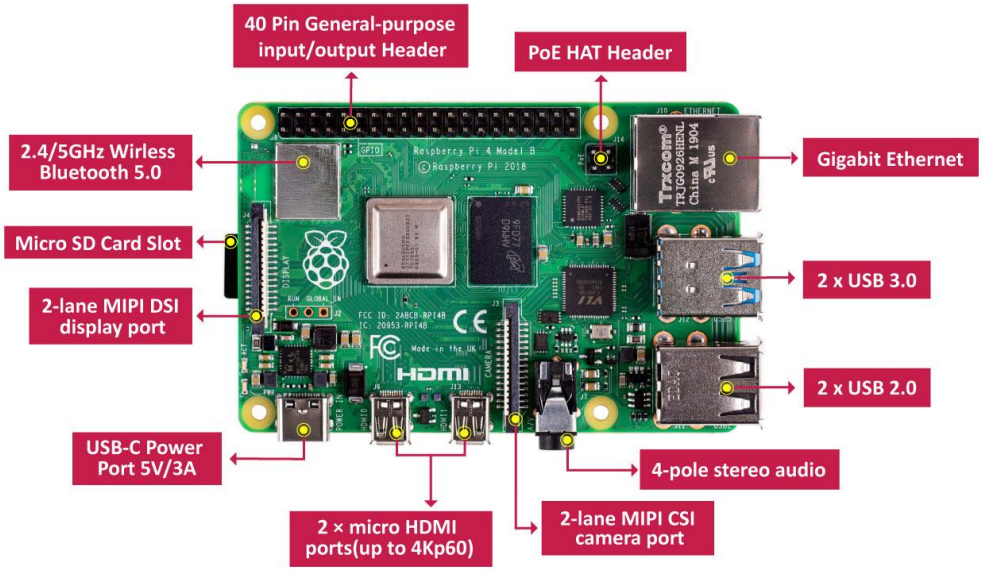

Abbildung 11 : Raspberry Pi 4 (seed, 2020)

4.2.2 Makeblock Megapi

MegaPi ist eine Hauptsteuerkarte, die auf Arduino MEGA 2560 basiert und die Programmierung mit Arduino IDE und Python unterstützt. MegaPi kann in 6 Funktionsbereiche unterteilt werden, sodass diese eine Verbindung mit verschiedenen Steckmodulen herstellen können, um Motoren und Sensoren anzutreiben und eine drahtlose Kommunikation zu realisieren. MegaPi verfügt über eine starke Motorantriebsfähigkeit, mit der 10 Servos oder 8 Gleichstrommotoren gleichzeitig angetrieben werden können.

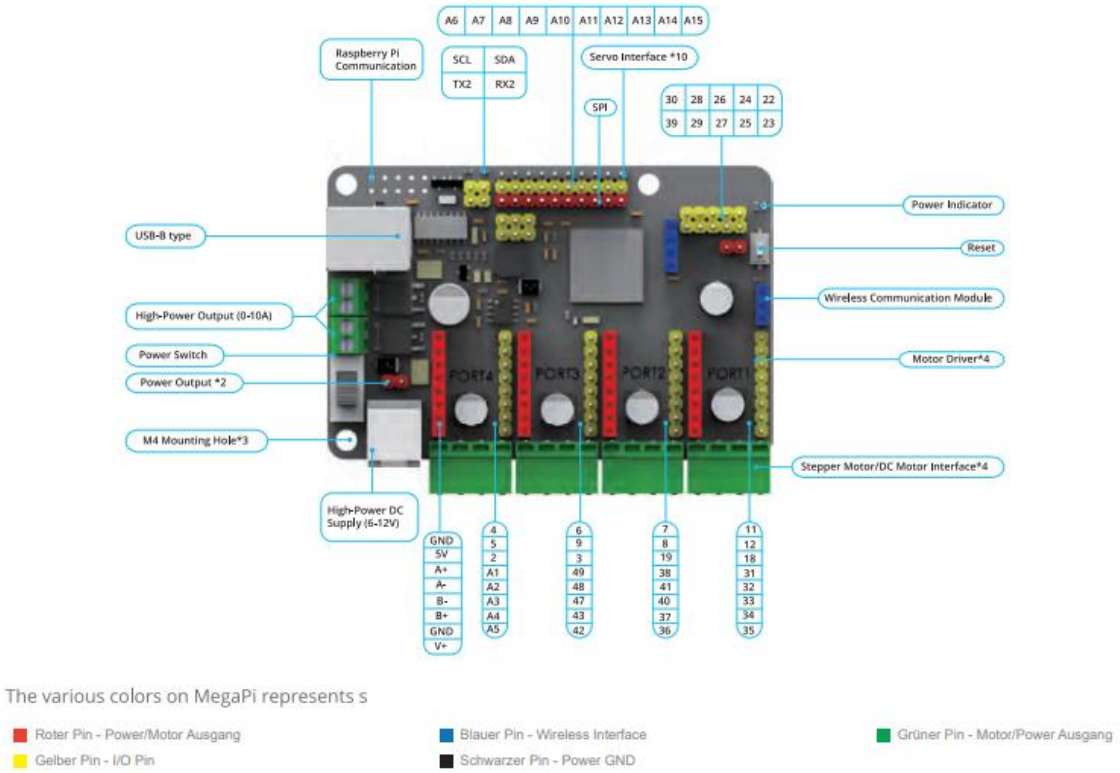

Abbildung 12: MegaPi Mainboard Aufbau (Makeblock, Makeblock, 2021)

Technische Daten

Microcontroller Chip	ATMEGA2560-16AU
Eingangsspannung	DC 6V-12V
Betriebsspannung	DC 5V
I/O Pins	43
Serielle Ports	3
I2C Schnittstelle	1
SPI Schnittstelle	1
Analoge Input Pins	15
Gleichstrom pro I/O Pin	20mA
Flash Speicher	256KB
SRAM	8KB
EEPROM	4KB
Taktfrequenz	16 MHz
Abmessungen	85*63mm

- Vier Motortreiber-Schnittstellen zum Hinzufügen von Geber- und Schrittmotor-Treibern um damit Gleichstrom-, Geber und Schrittmotoren anzusteuern.
- Eine drahtlose Kommunikationsschnittstelle Bluetooth oder 2,4Ghz Module hinzuzufügen.
- Zehn Servo Schnittstellen, die es dem Board ermöglichen, bis zu 10 Servos gleichzeitig anzusteuern.
- Zwei Hochleistungs-MOS-Treiber-Schnittstellen, die in der Lage sind, Geräte mit einem maximalen Strom von 10A und einer maximalen Leistung von 5V, 3A anzusteuern.
- Eine Raspberry-Pi Switch-Schnittstelle (erfordert manuelles Löten), um eine serielle Kommunikation von 5V bis 3.3V zu realisieren.
- Drei M4-Befestigungslöcher um die Platine mit dem Raspberry Pi zu verbinden. Schiebeschalter zur Steuerung der Stromversorgung.
- USB-B Schnittstelle zur Kommunikation und zum Herunterladen von Programmen. Der verwendete CH340G USB zu Seriell Chip, ermöglicht eine leichte und stabile Kommunikation.
- High Power Eingang mit Überspannungsschutz von 2A und Anti Reverse Messung.
- Eine Reset Taste, eine Power- und I/O Anzeige

Abbildung 13 : MegaPi Technische Daten (Makeblock, Makeblock, 2021)

4.2.3 Raspberry Pi Kamera Module V2

Zur Erkennung des Umfelds und der Fahrtrichtung des Fahrzeugs wurde ein Kameramodul v2 verwendet. Das v2-Kameramodul verfügt über einen Sony IMX219-8-Megapixel-Sensor. Mit dem Kameramodul können sowohl hochauflösende Videos als auch Standbilder aufgenommen werden. Allerdings ist anzumerken, dass hierbei eine reduzierte Auflösung vorgezogen werden muss, um die Verarbeitungszeit des Algorithmus für die Spurerkennung zu verringern. Das Kameramodul verfügt über ein Objektiv mit festem Fokus.

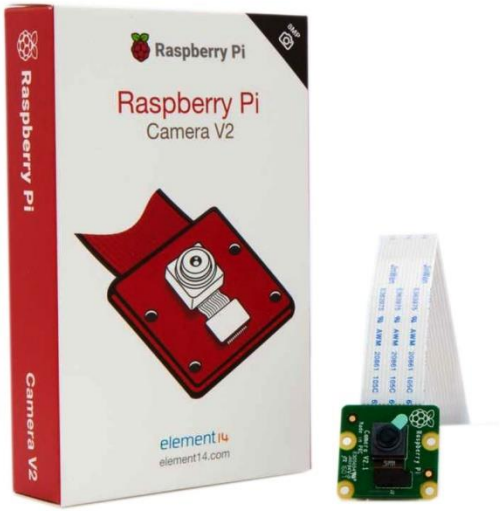

Abbildung 14 : Raspberry Pi Kamera (Direnc, 2021)

Die technischen Daten zur Raspberry Pi Kamera Module V2 wird in der Abbildung 15 gezeigt.

	Camera Module v1	Camera Module v2	HQ Camera
Net price	$25	$25	$50
Size	Around 25 × 24 × 9 mm		38 x 38 x 18.4mm (excluding lens)
Weight	3g	3g	
Still resolution	5 Megapixels	8 Megapixels	12.3 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90
Linux integration	V4L2 driver available	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available	
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX477
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4056 x 3040 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)
Pixel size	1.4 µm × 1.4 µm	1.12 µm x 1.12 µm	1.55 µm x 1.55 µm

Abbildung 15 : Technische Daten von Kamera Module (RaspberryPi, 2021)

4.2.4 Makeblock – 180 Optical Encoder Motor

Der optische Encodermotor 180 ist mit einem optischen Encoder ausgestattet, mit dem der Motor hochgenau steuern kann. Die einzigartige Struktur macht diesen Motor mit verschiedenen Teilen kompatibel. Aufgrund der speziellen Materialien erzeugt dieser Motor während der Fahrt weniger Geräusche und verspricht ein großes Ausgangsdrehmoment.

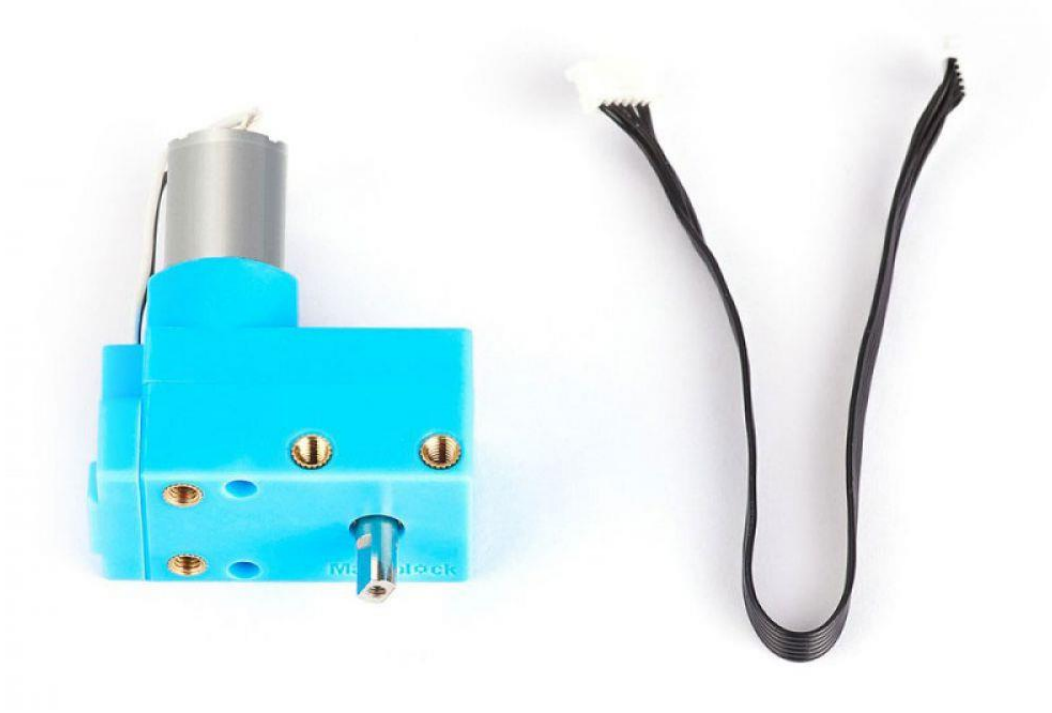

Abbildung 16: Optical Encoder Motor (Conrad, 2021)

Eigenschaften

- Flexible Installation: Kompatibel mit der Makeblock-Plattform mit flexibler Installation. Direktantrieb bei Kettenrädern, Zahnräder, Synchronriemen, Gummireifen und Ketten usw.;
- Optischer Encoder: präzise Steuerung mit PID- und PWM-Berechnung
- Leistung: Im Vergleich zu anderen Motoren der gleichen Stufe verfügt dieser Gebermotor über ein hohes Drehmoment, ein hohes Untersetzungsverhältnis und eine hohe Drehzahl.
- Geringeres Geräusch: Ausgestattet mit einem Getriebegehäuse aus POM-Materialien, das weniger abrasiv ist und weniger Geräusche verursacht.
- Lange Betriebszeit: Mit einer speziellen Stahl-Abtriebswelle kann der Motor ein hohes Drehmoment effizient übertragen, Abrieb reduzieren und die Lebensdauer verlängern.

4.2.5 Makeblock – MegaPi Pro Encoder/DC Motor Driver

Dieses Modul kann einfach durch 2 × 8 Pin-Einsetzen auf MegaPi Pro und MegaPi installiert werden.

Eigenschaften

- Unterstützt Motoren mit Betriebsspannungen von 6 ~ 12 V.

- Liefert einen Betriebsstrom von bis zu 3 A (Spitzenstrom von bis zu 5,5 A) bei eine 12-V-Stromversorgung.
- Das Modul verfügt über einen Überspannungs-, Überstrom- und Übertemperaturschutz, um die Gebrauchssicherheit umfassend zu gewährleisten.
- Bietet farbenfrohe Steckdosen für Männer und Frauen, um ein falsches Einsetzen zu verhindern.
- Das Modul ist klein und kann leicht ausgetauscht werden. (MakeBlock, 2019)

Abbildung 17 :MegaPi Pro Encoder/DC Motor Driver (Makeblock, Makeblock, 2019)

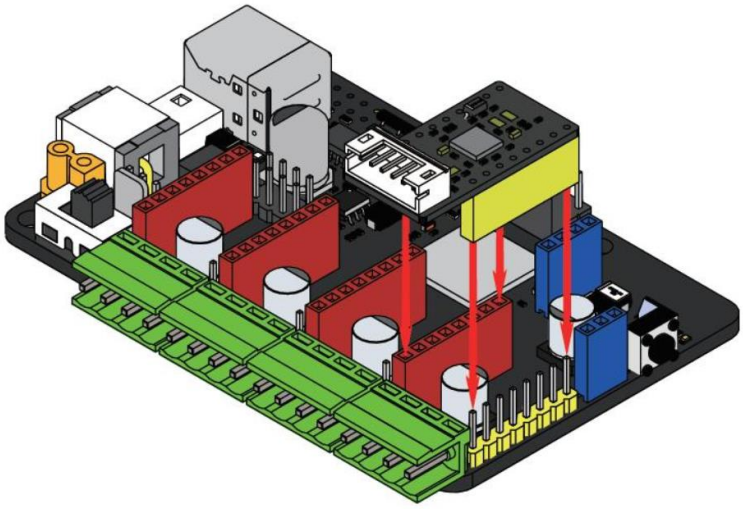

Abbildung 18 : Installation des Motordrivers (MakeBlock, 2019)

4.3 Software

4.3.1 Betriebssystem

Raspberry Pi bietet viele Betriebssystem Optionen, wie Raspbian, NOOBS, Ubuntu MATE usw.

4.3.1.1 Raspbian

Als offizielles Betriebssystem bietet Raspberry Pi die Linux-Distribution Raspbian. Es bringt die beste und aktuellste Hardware-Unterstützung mit. Außerdem bietet dieses Betriebssystem viele nützliche und notwendige Programme, damit die Benutzer nichts falsch machen können.

4.3.1.2 NOOBS

NOOBS ist kein richtiges Betriebssystem, sondern ein Bootmanager, der eine einfache Anwendungsmöglichkeit bietet und als eine Linux-Distribution auf die SD-Speicherkarte installiert werden kann. Aber dieser Bootmanager ist nicht für den dauerhaften Einsatz geeignet, weil die Benutzer mit vielen Problemen konfrontiert werden können.

4.3.1.3 Ubuntu MATE

Ubuntu MATE basiert auf der beliebten Linux-Distribution Ubuntu in Kombination mit dem MATE-Desktop. Diese Ubuntu- Version wird die gute Wahl für die Nutzer sein, die Raspberry Pi mit Desktop-Schnittstelle verwenden möchten.

Bei diesem Projekt wird Raspbian Buster mit Desktop-Version angewandt. Dafür wird eine SD-Card gebraucht, um das Betriebssystem zu installieren. Das Betriebssystem wird durch PC über SD-Card installiert und wird somit operationsbereit für die Raspberry Pi, die direkt angesteckt werden kann. (Elektronik Kompendium, 2020)

4.3.2 Raspberry Pi und MegaPi Kommunikation

Um das Fahrzeug zu bewegen, muss die Verbindung, die über Kabel oder USB hergestellt werden soll, korrekt bereitgestellt werden. Anstatt eine Verbindung über USB herzustellen, kann auch eine Verbindung durch Kabel über die vom Megapi bereitgestellten Ports hergestellt werden.

Durch die Verwendung der USB-Verbindung können weitere Probleme auftreten. Um diese Probleme zu lösen, müssen die für Raspberry-Pi spezifischen Systembefehle geändert werden. Ein Fehler bei dieser Codeänderung kann dazu führen, dass das System zusammenbricht oder sogar das gesamte System von Anfang an geladen werden muss. Daher wäre es logischer, die auf dem MegaPi verfügbaren speziellen Ports zu verwenden. Auf diese Weise kann die Verbindung, die zwischen Raspberry-Pi und MegaPi reibungsloser eingerichtet werden.

Welche Ports miteinander verbunden werden müssen, zeigen die gelben Quadrate die Abbildung 19 angegeben sind. Diese Verbindung kann auch hergestellt werden, indem der spezielle Stecker, der neben dem MegaPi liegt, mit dem MegaPi verlötet wird.

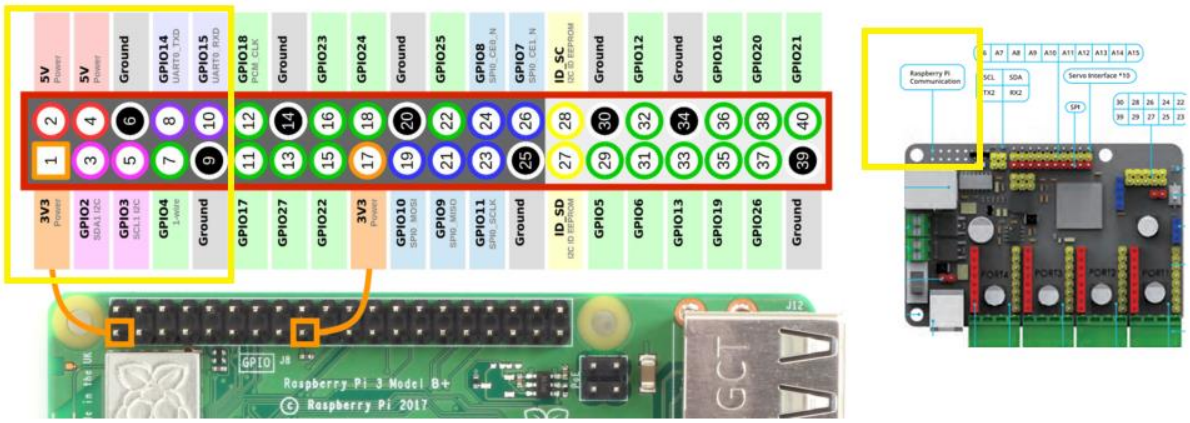

Abbildung 19: Raspberry-MegaPi Kommunikation

Nachdem die Verbindung hergestellt wurde, wird MegaPi durch USB-Kabel mit dem Computer verbunden und der MegaPi_firmware-Code wird mithilfe der Arduino IDE hochgeladen. (MakeBlock, 2020)

Auf diese Weise werden die Codes, die in Raspbery Pi mit der Python-Sprache geschrieben wird, vom MegaPi gelesen und ausgeführt. Andernfalls versteht der MegaPi nicht, wofür diese Codes genutzt wird, selbst wenn die in Python geschriebenen Codes auf das MegaPi übertragen werden.

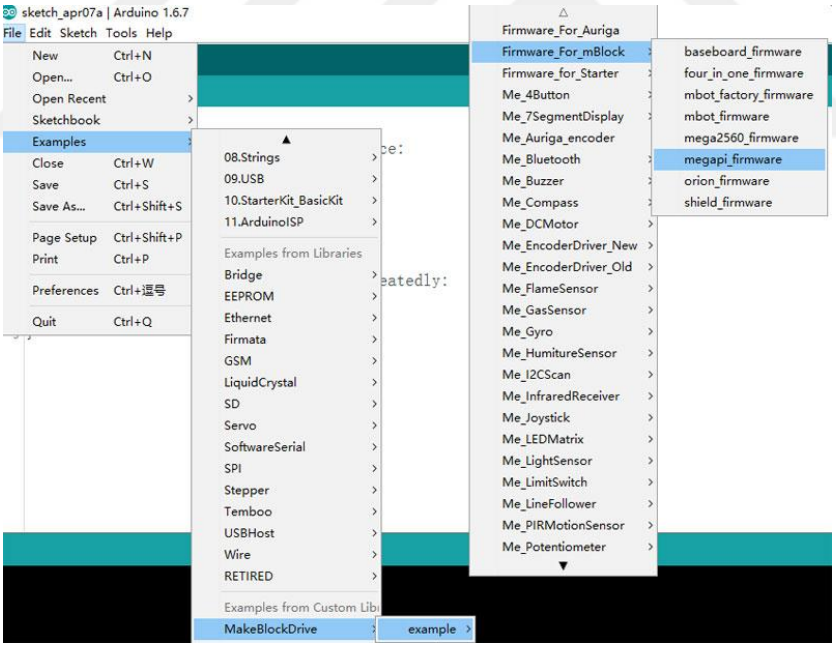

Abbildung 20:MegaPi Software (MakeBlock, 2020)

Die Verbindung von Kabeln spielt hier eine sehr wichtige Rolle. Die Kabel dürfen nicht falsch angeschlossen werden. Wenn die Kabel falsch angeschlossen sind, kann es zum Verbrennen der elektronischen Teile von MegaPi oder RaspberryPi kommen. Bei diesem Fall diese Komponente können als Defekt definiert können und nicht weiter genutzt werden.

Nach allen durchgeführten Operationen wird das Ergebnis Abbildung 21 gezeigt.

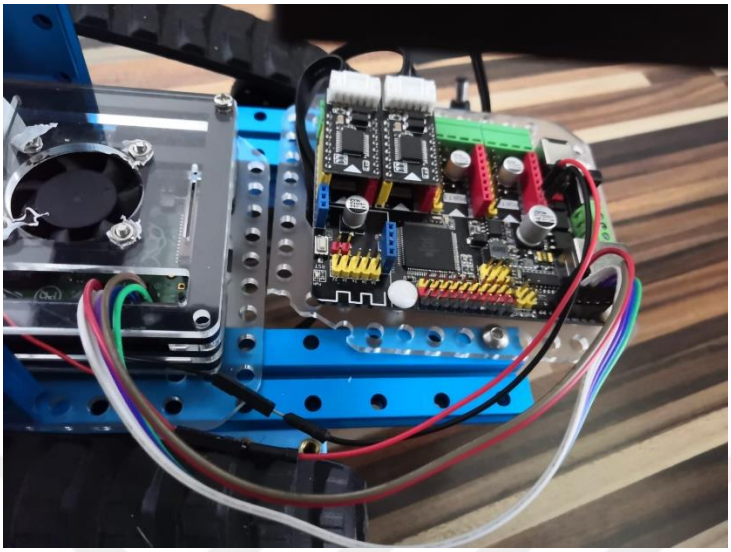

Abbildung 21: richtige Verbindung zwischen RaspberryPi und MegaPi

4.3.3 Image Processing

4.3.3.1 Region of Interest

Das Verarbeiten des gesamten Bilds des Streifens und das Aufdecken der Streifenlinien bedeutet viel Verarbeitung. Je mehr Operationen ausgeführt werden, desto mehr Last wird auf den Prozessor gelegt. Um diese Last zu verringern, wird nur ein bestimmter Teil des Streifens fokussiert und die erforderlichen Maßnahmen werden nur für diesen Bereich ergriffen. Andernfalls wird der Prozessor zu stark belastet und verlangsamt, da zu viele Vorgänge ausgeführt werden müsse, um die erlangten Daten verarbeiten zu können.

Zunächst wird der beim Bild erforderliche Bereich mit den roten Linien markiert, um die sichtbaren Streifen optimal zu verarbeiten. Dafür wird folgender Kode genutzt. (siehe Quellcode 1)

Abbildung 22 : Region of Interest

Quellecode 1: Perspective Transformation

```
<python
Source = np.float32([[0, 240], [30, 150], [370, 150], [400, 240]])
Destination = np.float32([[0, 240], [0, 0], [400, 0], [400, 240]])
While True:
    # Schaffung von Region of Interest
    colour = (0, 0, 255)
    frameP = cv2.line(frameP, (0, 240), (30, 150), colour, 2)
    frameP = cv2.line(frameP, (30, 150), (370, 150), colour, 2)
    frameP = cv2.line(frameP, (370, 150), (400, 240), colour, 2)
    frameP = cv2.line(frameP, (400, 240), (0, 240), colour, 2)
   # Winkel Veränderung
    Matrix = cv2.getPerspectiveTransform(Source, Destination)
    framePers = cv2.warpPerspective(frameP, Matrix, (400, 240))
>
```

Durch Anwendung des obigen Codes kann der markierte Bereich senkrecht zum Bildschirm angezeigt werden (Vogelperspektive) und die Koordinaten der Linien, die in fortgeschrittenen Stufen erhalten werden sollen, werden ohne Probleme erhalten.

Das auf diese Weise erhaltene Bild wird von den Umgebungsfaktoren getrennt und ist in dieser Form für eine perfekte Erkennung verwendbar geworden. (siehe Abbildung 23)

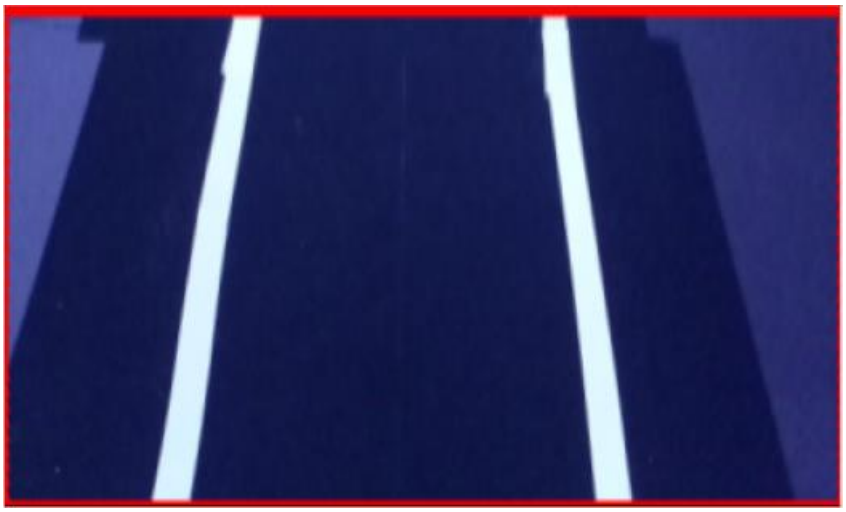

Abbildung 23 : Perspective Transformation

4.3.3.2 Farbenskala Veränderung

Das Bild, das zum Definieren des Streifens verwendet wird, wurde in der Abbildung 23 erhalten. Dieses erhaltene Bild enthält viele irrelevante Informationen, die verarbeitet werden müssen. Durch die alleinige Erkennung der weißen Linien werden sämtliche irrelevante Informationen beseitigt.

Um die weißen Farben des Streifens stärker hervorzuheben, muss das Bild mit einer anderen Farbtonskala gefiltert werden.

Es gibt verschiedene Methoden, die für diesen Farbveränderungsprozess verwendet werden können. Zwei davon sind HSV- und HLS-Farbänderungen.

Es ist sehr wichtig, bei diesen beiden Farbänderungen (siehe Abbildung 24) den Maßstab des gewünschten Farbbereichs korrekt einzugeben, da es sonst sehr schwierig ist, die gewünschte Farbe vom gesamten Bild zu trennen. Die (lower- und upper_white) Werte sollten geändert werden, bis nur noch die gewünschte Farbe auf dem Bildschirm angezeigt wird. Andernfalls werden andere Farben, die im Bild erkannt werden, Probleme beim Auffinden und Verfolgen der Linien verursachen.

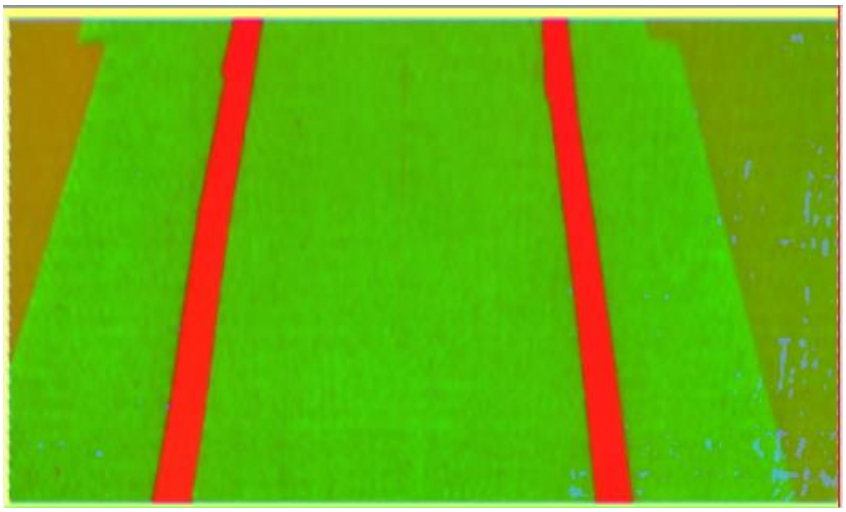

Abbildung 24: HSV-Transform

Der Code, der verwendet werden soll, um die Streifen im Bild von den umgebenden Farben zu unterscheiden und sie besser zu bestimmen, ist unten angegeben. (siehe Quellecode 2)

Quellecode 2 : Farbe Veränderung

```
<python
hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)


    lower_white = np.array([0, 0, 212], dtype="uint8")
    upper_white = np.array([180, 40, 255], dtype="uint8")
    colourmask = cv2.inRange(hsv, lower_white, upper_white)
>
```

Upper_white und Lower_white sind die Werte der unteren und oberen Grenze der weißen Farbe, die vom Bild getrennt werden. Diese Skala wird verwendet, um die weißen Linien auf dem Bild zu verdeutlichen.

Als Ergebnis der angewandten Prozesse bleiben auf dem Bild nur die weißen Streifen erhalten. (siehe Abbildung 25)

Abbildung 25: Colour Mask

Das Streifenbild, das nach dem Entfernen der unerwünschten Angaben erhaltet wird, kann jetzt für Edge Detection genutzt werden. Dafür wird folgender Kode genutzt. (siehe Quellecode 3)

Quellecode 3 : Canny Edge Kode

```
<python
edges = cv2.Canny(colourmask, threshold1, threshold2)

# threshold1 : Dies ist der hohe Schwellenwert des Intensitätsgradienten.
# threshold2 : Dies ist der untere Schwellenwert des Intensitätsgradienten.
>
```

Das Resultat wird in der Abbildung 26 gezeigt. Mit Hilfe von Canny edge Detection wird das Bild bereitgestellt, damit das Programm mit höchster Effizienz die Linien-erkennung verwirklichen kann.

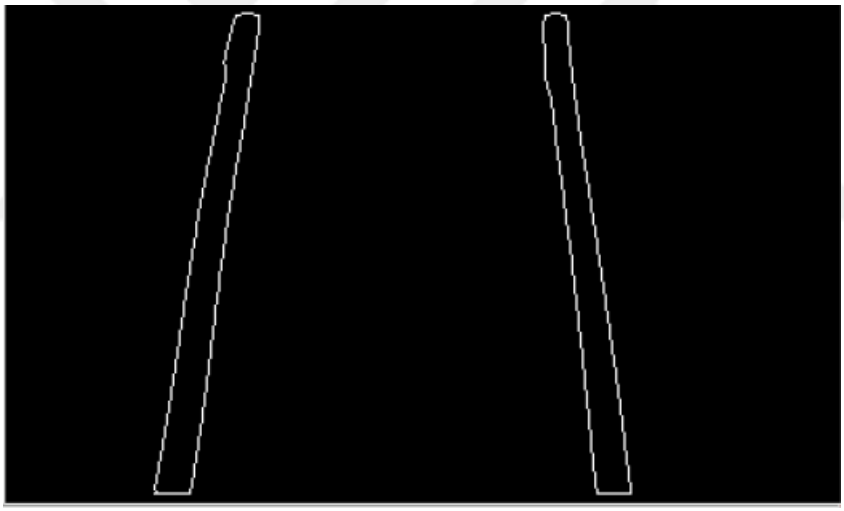

Abbildung 26 : Canny edge detection

4.3.4 Linien erkennen

Nachdem eine glatte Streifenansicht auf dem Bildschirm erhalten wird, können die Positionen der Linien auf dem Bildschirm mit dem Befehl HoughLines bestimmt werden. Dazu können zwei Befehle genutzt werden; cv2.HoughLines und cv2.HoughLinesP. Der Unterschied wird in Abbildung 27 gezeigt.

Abbildung 27 : Unterschied zwischen Houghlines und HoughlinesP (Kacmajor, 2017)

Das Suffix P von HoughLinesP steht hier für probabilistisch. Ein wichtiger Vorteil von HoughLinesP ist, dass eine effiziente Implementierung möglich ist und der Befehl die erkannten Zeilen in Form von ′(x_0, y_0, x_1, y_1)‘ angibt, was die genaue Koordinierung der Linien effizienter gestalten lässt. (Kacmajor, 2017)

Der Befehl HoughlinesP wird verwendet, sodass die genauen Koordinaten der Punkte auf dem Bildschirm ermittelt werden können. Dies vereinfacht auch die spätere Verwendung.

Quellecode 4 : HoughLinesP

```
<python
rho = 1
theta = np.pi / 180
min_threshold = 15

#Linien Erkennung
cv2.HoughLinesP(edges, rho, theta, min_threshold,
                                    np.array([]),
minLineLength=50, maxLineGap=150)
>
```

- Rho : Entfernungsauflösung des Akkumulators in Pixel.
- Theta: Winkelauflösung des Akkumulators im Bogenmaß.
- Threshold: Es werden nur die Zeilen zurückgegeben, die genügend Angaben erhalten
- minLineLength: Minimale Zeilenlänge. Liniensegmente, die kürzer sind, werden abgelehnt.

- maxLineGap: Maximal zulässige Lücke zwischen Liniensegmenten, um sie als einzelne Linie zu behandeln. (Matas, Galambos, & Kittler, 2000)

Mit Hilfe von HoughLinesP werden Koordinaten von den Punkten „$(x_0,\ y_0,\ x_1\ , y_1\ )$“ erhaltet. Die erhaltenen Koordinaten werden verwendet, um die durchschnittliche Steigung (m_d) und den durchschnittlichen Schnittpunkt $(x_{sch,drh})$ zu finden. Die Gleichung (3.3.4.1) wird die Linie ausgedrückt, die durch die zwei Punkte $(x_0,\ y_0)$ und $(x_1\ , y_1\ )$ durchgeht. Steigung (m) von Linie wird mit Gleichung (3.3.4.2) erreicht.

„c“ kann leicht aus der Gleichung (3.3.4.1) erhalten werden. Dann wird x_{sch} durch Einstellung $y = 0$ in Gleichung (1) berechnet. Da mehr als eine einzelne Linie an der linken und rechten Seite erkannt werden, kann eine glatteste Linie durch die durchschnittliche Steigung m_d und den durchschnittlichen Schnittpunkt $x_{sch,d}$ definiert werden. „N“ steht für die Anzahl der Linien, die im Bild erkannt werden.

$$y = mx + c \tag{3.3.4.1}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.3.4.2}$$

$$x_{sch} = \frac{c}{m} \tag{3.3.4.3}$$

$$(x_{sch,d}) = \frac{1}{N} \sum_{i=1}^{N} int, i \tag{3.3.4.4}$$

$$(m_d) = \frac{1}{N} \sum_{i=1}^{N} m_i \tag{3.3.4.5}$$

Nachdem alle vorher bezeichneten Werte gefunden wurden, sollten die Linien in rechte und linke Streifen getrennt werden. Dafür gibt’s mehrere Methoden. Zwei davon werden zusammen verwendet, um ein genaueres Ergebnis zu erzielen.

1. Die rechte Spur hat $x_2 > x_1$ und y2 > y1, was eine positive Steigung ergibt. Daher werden alle Linien mit positiver Steigung als Punkte auf der rechten Spur betrachtet. Im Gegenteil linke Spur hat x1 < x2 and y2 < y1, was eine negative Steigung ergibt.
2. Linke Spur sollte sich links vom Mittelpunkt des Bildes befinden und rechte Spur sollte rechts vom Mittelpunkt des Bildes bleiben.

Mit Hilfe von den Berechnungen wird die korrekte Definition der Streifenlinien bereitgestellt. (Siehe Abbildung 28)

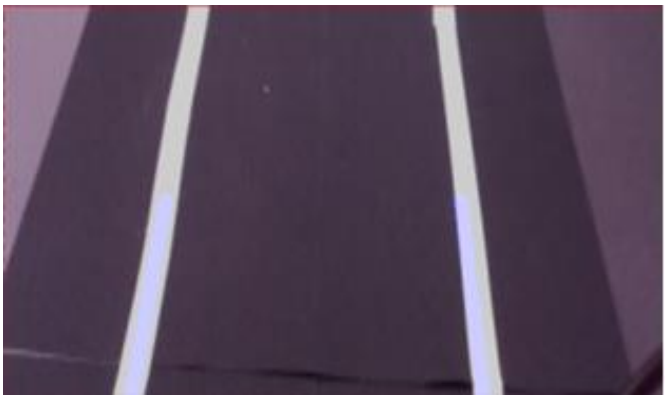

Abbildung 28 : Definierung der Linien

Mit diesen Definitionen kann ein optimiertes Liniendefinitionssystem eingerichtet werden. Durch die Verarbeitung der Linienkoordinaten werden die notwendigen Vorkenntnisse erlangt, um das Auto auf der Spur zu halten.

4.3.5 Optimierung der Spuren

Das Ziel dieses Projekts ist, das Fahrzeug in der Mitte der Fahrspur zu halten. Dafür werden einige Berechnungen gebraucht, die auf die vorher erhaltenen Koordinaten von Spuren basieren.

Mit Hilfe von Gleichung (3.3.5.1) wird Mittelpunkt der Spuren berechnet. Gleichung (3.3.5.2) liefert Mittelpunkt des Bildes. Der als Fehler definierte Wert ist die Differenz dieser beiden Werte (siehe Gleichung (3.3.5.3))

$$x_c = \frac{(x_{rechts} - x_{links})}{2} + x_{links} \tag{3.3.5.1}$$

$$Bild_c = (Bild_{Breite})/2 \tag{3.3.5.2}$$

$$Fehler = Bild_c - x_c \tag{3.3.5.3}$$

Wenn der Fehler Wert größer als null ist, dann soll das Auto sich nach rechts bewegen und im Gegenteil wenn der Fehler kleiner als null ist, dann soll das Auto sich nach links orientieren.

Die Ergebnisse, die sich aus den erworbenen Informationen ergeben, werden auf dem unteren Bild (Abbildung 24) gezeigt. Der blaue Punkt markiert den Mittelpunkt der Streifen und die vertikale rote Linie zeigt den Mittelpunkt des Bilds an. Der aus den Berechnungen resultierende Fehler und die Richtung, in die das Autofahren soll, werden auf dem Bildschirm angezeigt.

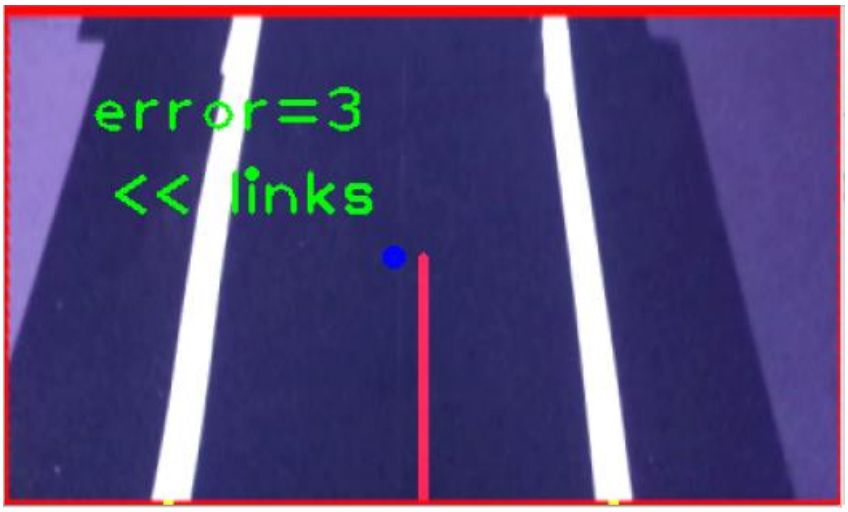

Abbildung 29 : Anzeige des gefundenen Fehlers

4.3.6 Ultraschall Sensor

Für die Verwendung des Abstandssensors sollten die Befehle auf der offiziellen Bibliothek von Makeblock verwendet werden. Mit diesem Befehl kann jedes Hindernis vor dem Auto erkannt werden. (MrHezhisheng, 2019)

Quellecode 5 : Ultraschall Sensor Kode

```
<python
def Ultraschall(v):
    global distance
    #print("distance:"+str(v)+" cm")
    distance = v
while 1:
bot.ultrasonicSensorRead(6,Ultraschall)
    if distance < 15:
        bot.encoderMotorRun(1,0); #sol motor
        bot.encoderMotorRun(2,-0);#sag motor (-)
>
```

Wenn sich ein Hindernis näher als 15 cm vor dem Fahrzeug befindet, wird die Leistung in den Motoren des Fahrzeugs mit Hilfe Quellecode 5 automatisch abgeschaltet und somit wird ein Aufschlag des Fahrzeugs auf ein Hindernis verhindert.

4.3.7 Haar Cascade

Haar-Kaskaden-Klassifikatoren sind ein effektiver Weg zur Objekterkennung. Die Objekterkennung ist eine Computertechnologie in Bezug auf Computer Vision, Bildverarbeitung und Deep Learning, die sich mit der Erkennung von Instanzen, den

Objekten in Bildern und Videos befasst. Dafür werden positive und negative Bilder benutzt.

- Positive Bilder - Diese Bilder enthalten die Objekte, die durch den Klassifikator identifiziert werden sollen.
- Negative Bilder - Diese Bilder enthalten keine Objekte, die identifiziert werden sollen.

Es empfiehlt sich, positive und negative Bilder aus verschiedenen Winkeln (siehe Abbildung 30 und 31) aufzunehmen, um eine gute Wahrnehmung der Objektserkennung zu erzielen. Sonst müssen die Bilder eine gute Auflösung haben. Nach der Aufnahme der Bilder sollte das zu definierende Objekt nicht in die Negativproben aufgenommen werden. Andernfalls kann keine gute Objektbeschreibung vorgenommen werden.

Abbildung 30 : Stoppschild mit den unterschiedlichen Winkeln

Abbildung 31 : Tempolimit Schild mit den unterschiedlichen Winkeln

Nachdem die positive Probe entnommen wurde, wird das Bild geschnitten und andere Objekte im Bild werden entfernt. Auf diese Weise werden Probleme vermieden, die durch die Identifizierung anderer Objekte im Bild entstehen können. (siehe Abbildung 32)

Abbildung 32 : Positiv und negativ Proben von Stoppschild

Je höher die Anzahl der negativen und positiven Bilder ist, desto besser kann maschinelles Lernen erreicht werden. Wenn mehr als ein Objekt eingeführt werden soll und diese Objekte einander ähnlich sind, können diese Objekte schwer zu identifizieren sein. Um dieses Problem zu beheben, können Bilder von Objekten vor dem maschinellen Lernen zwischen negativen Bildern anderer Objekte platziert werden. Auf diese Weise werden die Fehler minimiert. (siehe Abbildung 33)

Abbildung 33 : ähnliche Verkehrszeichen

Es ist sehr wahrscheinlich, dass diese Verkehrszeichen miteinander verwechselt werden. Diesbezüglich ist Vorsicht geboten, da sonst möglicherweise maschinelles Lernen erneut durchgeführt werden muss.

Die Anzahl der Bilder sollte erhöht werden, wenn keine ausreichende Effizienz erreicht wird. Das Training kann mit 1000 negativen 50 positiven Bildern beginnen. Wenn keine ausreichende Effizienz erreicht werden kann, sollte dieser Vorgang durch Ändern der Anzahl der Bilder wiederholt werden. Da dieser Vorgang lange dauert, ist es besser, von Anfang an mit mehr Bildern zu beginnen.

Ein Sonderfall gilt auch für das rote Licht. Das größte Problem beim Unterrichten von Rotlicht besteht darin, dass sowohl grünes als auch rotes Licht vom Prozessor nicht unterschieden werden können. Denn dieser Vorgang wird normalerweise mit einem Bild mit Grautönen durchgeführt, um den Prozessor nicht zu belasten. Das Scannen von Farbbildern erfordert viel Strom und dies reduziert die Leistung des Prozessors, sodass die Spezifikationen des verwendeten Prozessors bei diesem Prozess eine sehr wichtige Rolle spielen. Wenn dies nicht getan würde, würde es Probleme bei der Identifizierung geben, da die Farben nicht bemerkt bzw. unterschieden werden können. Diese Methode kann auch verwendet werden, um zwei sehr ähnliche Objekte zu trennen. Auf diese Weise ist es möglich, die Effizienz zu steigern, indem nur zu negativen Bildern einige weitere Bilder hinzugefügt werden.

Als Ergebnis der Experimente wurde das beste Ergebnis erzielt, indem das Grünlichtbild zwischen die Negativbilder gelegt wurde. Auf diese Weise wird sichergestellt, dass sich das grüne und das rote Licht nicht gegenseitig stören.

Abbildung 34 : positiv und negativ Bildern für rotes Licht

Die durch maschinelles Lernen erhaltene Datei mit der Erweiterung .xml kann mithilfe von Python und OpenCV verwendet werden.

Zunächst wird die Datei mit der Erweiterung .xml aufgerufen (mit Hilfe von `cv2.CascadeClassifier`) und die erforderliche Variablenzuweisung vorgenommen. Dann wird das Bild ausgewählt, in dem das Objekt gefunden werden soll, und eine Suche nach dem Bild wird gestartet (Mit Hilfe von `cascade.detectMultiScale)`. siehe Quellcode 6 für weitere Information.

Quellecode 6 : Objekterkennung Kode

```
<python
stop_cascade=
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Stop_cascade
.xml')
stops = stop_cascade.detectMultiScale(red_sign_frame, 1.1, 5)
# red_sign_frame = der Rahmen für die Erkennung
# ScaleFactor = 1.1
# min Neighbours = 5     >
```

Nachdem die Quellecodenummer 6 verwendet wurde, kann nicht genauer erkannt werden, ob die sichtbaren Objekte gefunden wurden. Um dies zu verstehen, kann das Objekt auf dem Bildschirm mit einem zusätzlichen Code in einem Rahmen dargestellt werden. Mit diesem Code kann die Position des Objekts auf dem Bildschirm angezeigt und ein Text darauf geschrieben werden, um anzuzeigen, welches Objekt sich wo befindet. (siehe Quellecode 7)

Quellecode 7 : Anzeige gefundener Objekte

```
<python
for (x,y,w,h) in stops:
        cv2.rectangle(gray,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(gray, "StopSign", (x, y + 30), font, 1,
(0,0,255), 1)
        cv2.putText(gray, "Stop sign detected!!", (50, 50),
font, 2, (0, 255, 0), 2)
>
```

Die Anzahl der Bilder sollte erhöht werden, wenn keine ausreichende Effizienz erreicht wird. Das Training kann mit 1000 negativen 50 positiven Bildern beginnen. Wenn keine ausreichende Effizienz erreicht werden kann, sollte dieser Vorgang durch Ändern der Anzahl der Bilder wiederholt werden. Da dieser Vorgang lange dauert, ist es besser, von Anfang an mit mehr Bildern zu beginnen.

Ein weiterer Faktor, der die Trainingsgeschwindigkeit beeinflusst, sind die ausgewählten Optionen für die Objekteinführung. Die Abmessungen des einzuführenden Objekts sollten so klein wie möglich eingegeben werden. Dabei soll die Größe des geschnittenen Bildes berücksichtigt werden. Wenn die Größe des einzuführenden Objekts zu groß eingegeben wird, dauert der Lernprozess länger. Es ist notwendig, den optimalen Wert zu finden, um diese Zeit zu verkürzen.

Die in diesem Projekt verwendeten negativen und positiven Bildnummern lauten wie folgt. (siehe Tabelle 5)

Tabelle 5 : Anzahl der Bilder

Verkehrsschild	Anzahl positiver Bilder	Anzahl negativer Bilder
Stoppschild	50	550
Rotlicht	50	650
30 Tempo Limit	50	600

Es gibt 3 Arten von Unterrichtsformen, die bei der Objekterkennung verwendet werden können: HAAR, LBP und HOG. Jedes davon erfüllt unterschiedliche Anforderungen. Um sie aufzulisten;

- HOG wird nur verwendet, wenn für die Objekterkennung OpenCV 3.1 oder höhere Version genutzt wird.
- HAAR-Klassifikatoren sind sehr genau und effizient, benötigen jedoch viel mehr Zeit zum Trainieren.

- LBP-Klassifikatoren wird verwendet, wenn viele Beispielsbilder zur Verfügung gestellt werden können. LBP-Klassifikatoren hingegen sind weniger genau, trainieren jedoch viel schneller und erkennen fast dreimal schneller.

Die folgende Darstellung (siehe Tabelle) wurde erstellt, um alle hier ablaufenden Prozesse zusammenzufassen und das Verständnis zu erleichtern.

Tabelle 6 : Schritte für die Objekterkennung

Schritte für die Objekterkennung
Auswahl des definierenden Objekts
Erstellung der positiven Bilder des Objekts
Bearbeitung der positiven Bilder
Erstellung der negativen Bilder des Objekts
Erstellung von negativen und positiven Bilddateien
Bestimmen der Abmessungen der positiven Bildern
Auswahl HAAR oder LBP Form
Erstellung der XML-Datei
XML-Datei testen
Verwendung der XML-Datei zur Objekterkennung

Nachdem die Objekte korrekt definiert wurden, muss ein weiteres Problem behoben werden. Obwohl dieses Problem nicht zu groß ist, kann es das Fahrzeug daran hindern, ordnungsgemäß zu funktionieren und die Regeln einzuhalten.

Was vom Fahrzeug erwartet wird, ist in einer bestimmten Entfernung anzuhalten, nachdem die Zeichen erkannt wurden. Nach dem ersten geschriebenen Code hält das Fahrzeug jedoch an, wenn es das Stoppschild sieht. Es kann verschiedene Lösungsmethoden für dieses Problem geben.

Die Größe des wahrgenommenen Schildes gibt auch den Abstand des Schildes an, da das Schild kleiner ist, wenn es weit entfernt ist, und größer, wenn es näher ist.

Wenn sich das Fahrzeug in einer bestimmten Entfernung befindet, kann der Wert von w im Programm bestimmt werden. Der Wert von w gibt die Breite des erkannten Objekts an. Auf diese Weise kann die Entfernung in cm festgestellt werden.

Um dies genau zu berechnen, wurde zunächst festgestellt, bei welchem Wert „w“ in einem Abstand von 60 cm liegt. (siehe Abbildung 35)

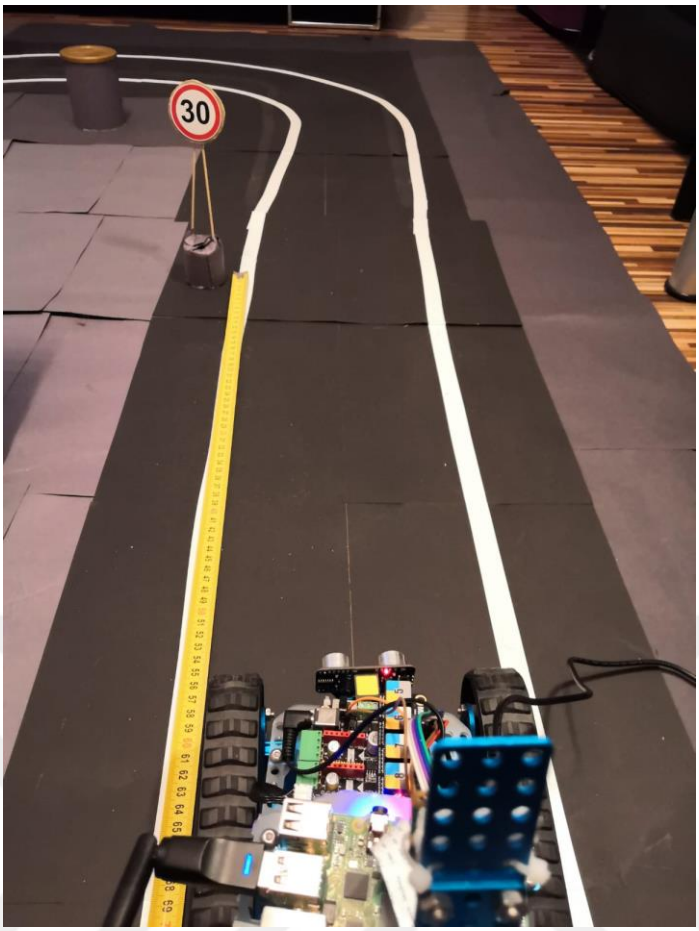

Abbildung 35 : 60cm Entfernung vom definierten Objekt

Als Ergebnis der Platzierung des Objekts 60 cm vor dem Fahrzeug wurde ein Wert von „w“ 50 gefunden. (siehe Abbildung 36) Eine Verbindung zwischen diesen beiden Werten kann hergestellt werden, indem der Wert von w erhalten und mit den Werten in der anderen Entfernung verglichen wird. Auf diese Weise kann festgestellt werden, wie weit das Fahrzeug in cm vom Objekt entfernt ist.

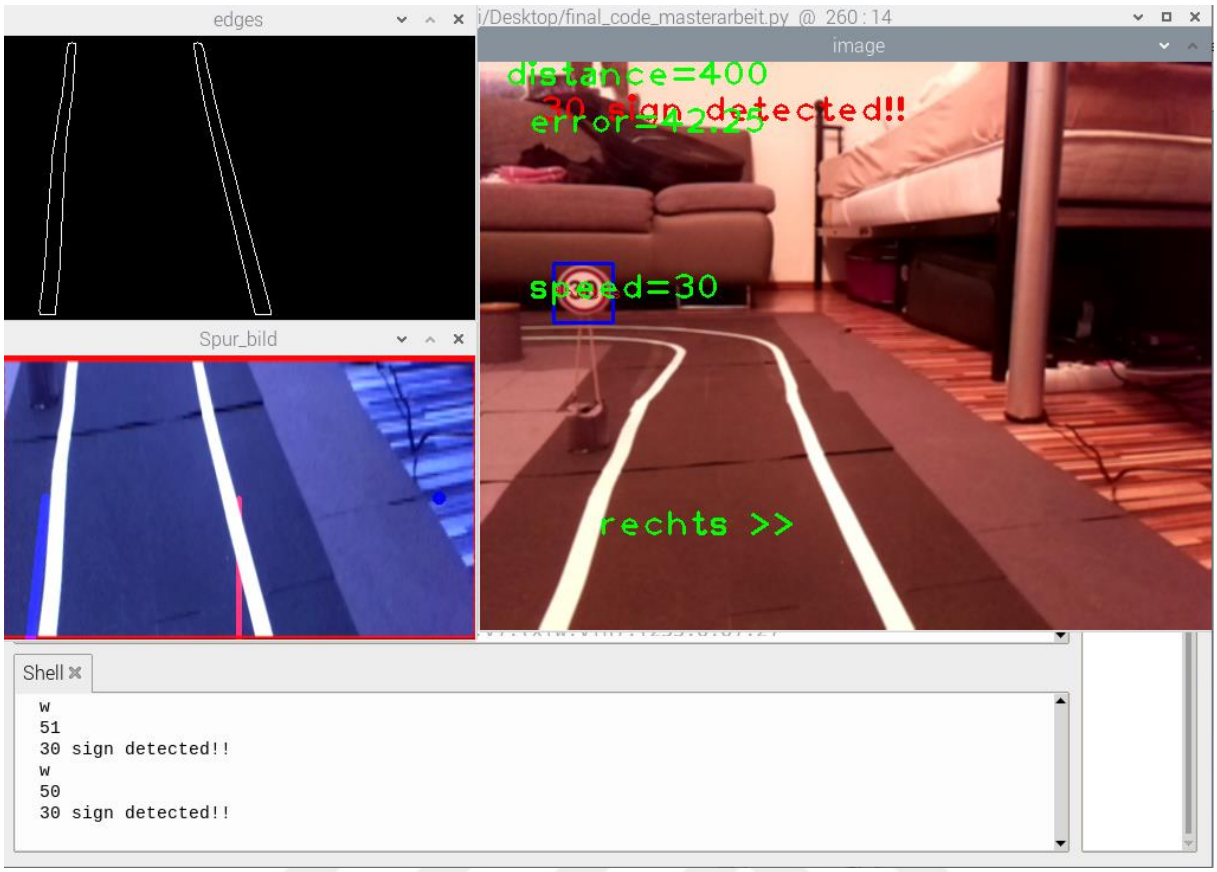

Abbildung 36 : Erhalten des w-Wertes für 60cm Entfernung

Das Fahrzeug wurde gestartet, indem das Fahrzeug 20 cm vom Objekt entfernt aufgestellt wurde, und es wurde versucht zu erhalten, dass der Wert von „w“ aufgrund der Entfernung höher sein würde. (siehe Abbildung 37)

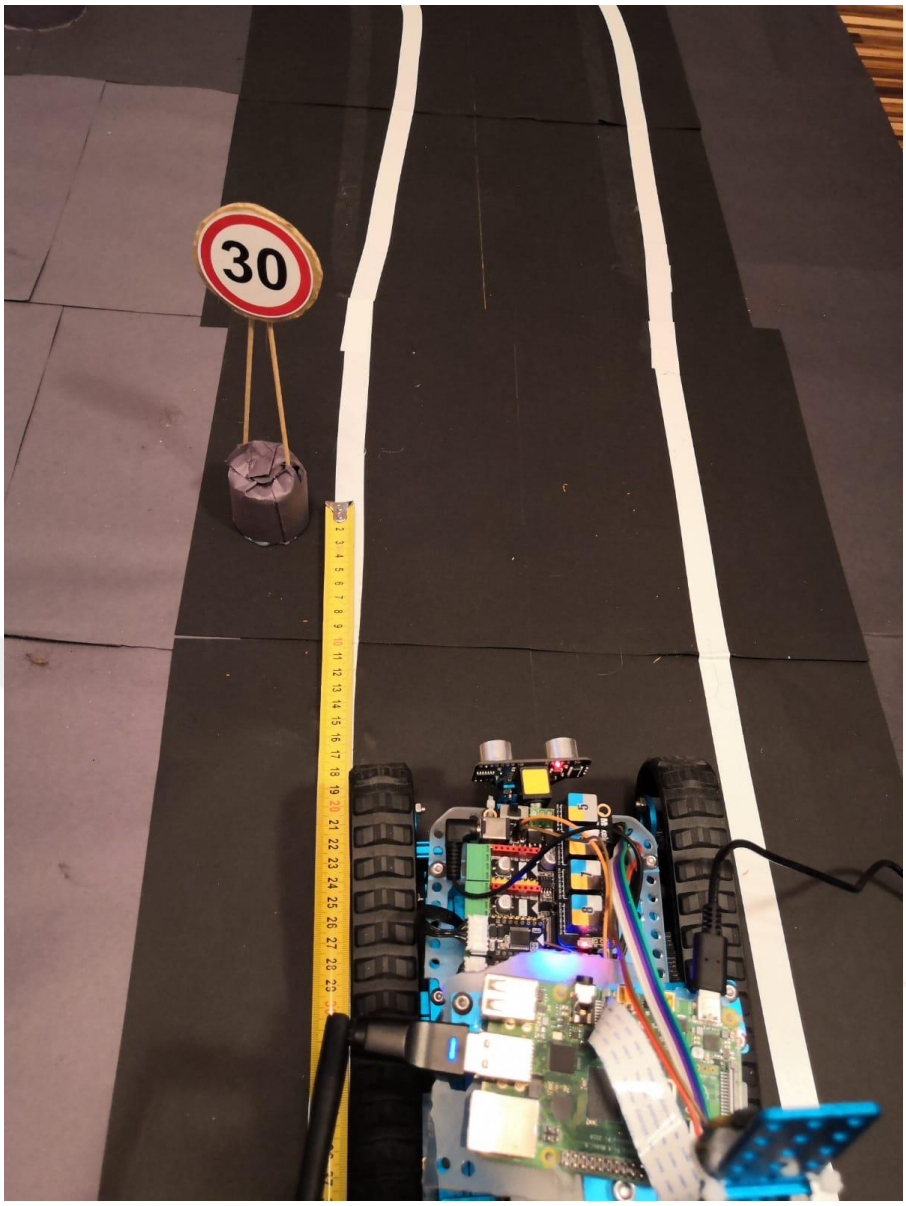

Abbildung 37 : 20 cm Entfernung vom definierten Objekt

Durch Platzieren des Fahrzeugs 20 cm vom Objekt entfernt ergibt sich ein w-Wert von 100. (siehe Abbildung 38)

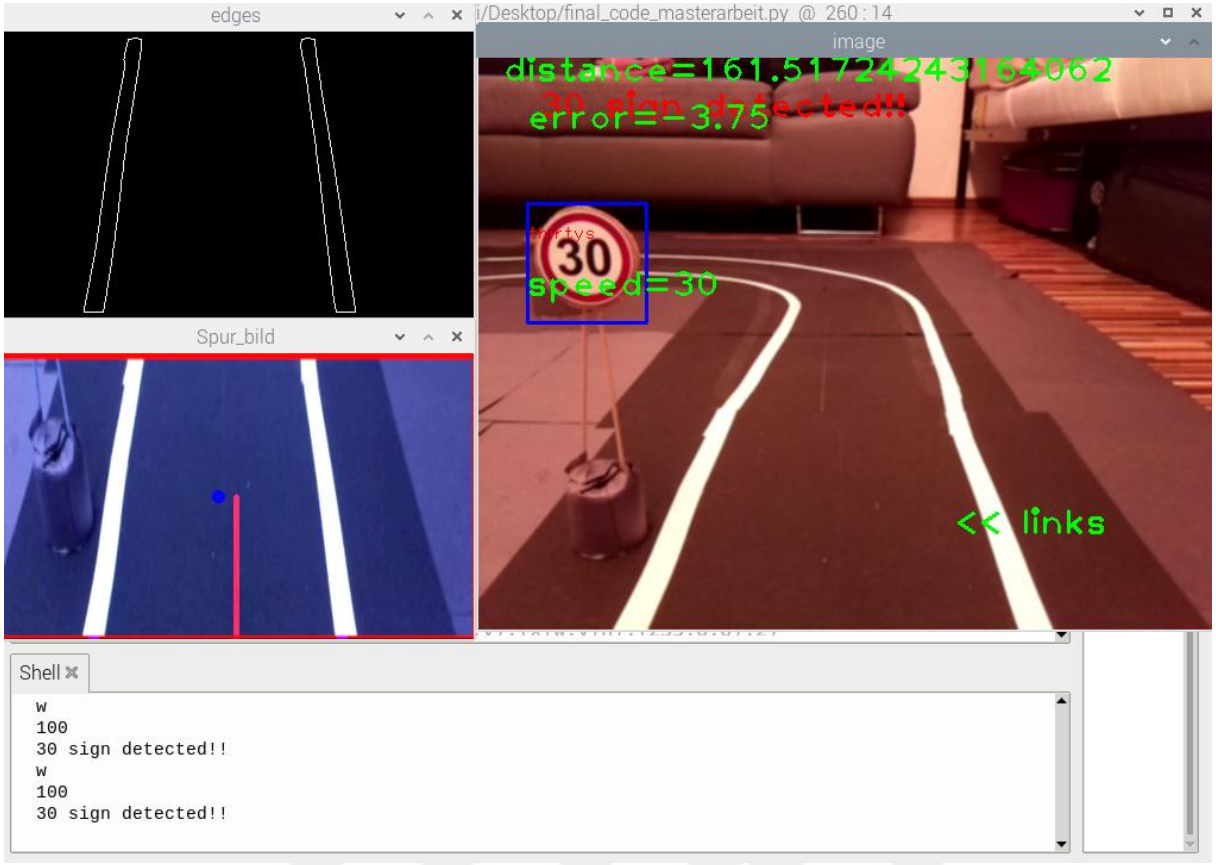

Abbildung 38 : Erhalten des w-Wertes für 20cm Entfernung

Die erhaltenen Informationen sind wie folgt (siehe Tabelle 7)

Tabelle 7 : W-Werten für unterschiedlichen Entfernungen

Entfernung	W-Werte
60cm	50
20cm	100

Mit Hilfe dieser Werte kann berechnet werden, wie viele cm der W-Wert beträg, oder es kann eine Gleichung zwischen zwei Variablen entsprechend erstellt werden.

Da in diesem Projekt jedoch festgestellt wird, dass das Fahrzeug die erforderlichen Befehle ausführt, wenn es 20 cm von den Verkehrszeichen entfernt ist, wird der Wert von w als 100 akzeptiert. In diesem Fall sind keine Berechnungen erforderlich.

Dieser ausgewählte Wert wird im Python-Code mit if-Bedingung (`if w > 100`) verwendet. Das Fahrzeug führt Befehle nur aus, wenn die Breite des Verkehrsschilds größer als der definierte W-Wert ist. (siehe Quellcode 8)

Quellecode 8 : Bestimmen der Entfernug des Verkehrsshildes

```
<python
    for (x,y,w,h) in thirtys:
        print("w:" + str(w))
        cv2.rectangle(sign_30,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(sign_30, "thirtys", (x, y + 30), font, 1,
(0,0,255), 1)
        cv2.putText(image, "30 sign detected!!", (50, 50),
font, 2, (0, 0, 255), 2)
        if w > 100 and thirtys is not None:
            speed= 30
            print("30 sign detected!!")>
```

4.3.8 Was muss bei Verwendung einer neuen Teststrecke geändert werden?

Falls die Farbe der Streifen auf der Teststrecke als eine andere Farbe anstelle von weiß definiert wird, müssen die Werte, die als Upper- und Lower_white definiert sind, im Befehl „cv2.inRange“ geändert werden. Diese Werte werden je nach Farbe variiert und die Wirkung von Licht aus der Umgebung muss dabei auch berücksichtigt werden, damit das Fahrzeug sich ordnungsgemäß auf der neuen Teststrecke bewegen kann. (siehe Quellecode 2)

Andernfalls kann der vorhandene Code Probleme verursachen, wenn die Linie nicht nur farblich unterschiedlich, sondern auch gestrichelt ist. Um dieses Problem zu lösen, sollten die Parameter min_threshold, minLineLength und maxLineGap im Befehl cv2.HoughLinesP in Quellecode 4 geändert werden. Diesen Parameter werden verwendet, um die Länge der zu erfassenden Linien und die Verbindung der gestrichelten Linien zu bestimmen. Infolge dieser Prozesse ändert sich der Code wie Quellcode 8.

Quellecode 9 : Anwendung des Codes mit roter Spur

```
<python
# Änderungen für Quellcode 2
lower_red = np.array([16, 160, 137], dtype="uint8")
upper_red = np.array([179, 255, 255], dtype="uint8")
colourmask = cv2.inRange(hsv, lower_red, upper_red)
# Änderungen für Quellcode 4
cv2.HoughLinesP(edges,
                rho,
                theta,
                min_threshold,
np.array([]), minLineLength=20, maxLineGap=200)
>
```

4.3.9 PID System

Es ist schwierig, ein Auto lange Zeit entlang einer geraden Linie zu bewegen. Dies ist für kurze Strecken möglich, jedoch nicht für lange Strecken. Es gibt viele Gründe für diese Situation. Einige dieser Gründe sind eine unebene Fahrfläche, die variierende Raddurchmesser der Fahrzeuge und unterschiedlicher Luftdruck an den Reifen und ähnliche Gründe. Am wichtigsten ist, dass die beiden Motoren nicht mit der gleichen Drehzahl drehen. Dies kann auf einen Fehler in der Motorproduktion oder auf eine Verformung des Materials zurückgeführt werden. Obwohl diese Faktoren sehr kleine Unterschiede aufweisen, verursachen sie im Laufe der Zeit oder mit der Verlängerung der Entfernung sehr große und offensichtliche Fehler. (O'Hanlon, 2020)

Um dieses Problem zu lösen, kann das Fahrzeug durch Vergleichen der Eingangs- und Ausgangswerte stabiler gemacht werden. Dafür kann PID-Kontrolle verwendet werden.

Quellecode 10 : PID Kontroller

```
< python
kp = 0.15
kd = 0.1
ki = 0.003
    lastError = 0
    pid_output = kp*error + ki*sumError + kd*lastError
    lastError = error
    sumError = lastError + error
>
```

Um das Fahrzeug auf geraden und kurvenreichen Straßen glatter abbiegen zu lassen, wird Quellcode 8 verwendet. Fahrzeuge, die ohne PID-Steuerung betrieben werden, sind unkontrollierter und neigen dazu, die Straße zu verlassen.

Die gesamte Hardware und Software wird vollständig verwendet, um sicherzustellen, dass sich das Fahrzeug einwandfrei auf der Straße bewegt und sich an die Umgebungsvariablen anpasst. Im nächsten Abschnitt werden die aufgetretenen Probleme und Lösungen näher erläutert.

5 Ergebnisse

5.1 Auflösung und FPS-Werte des Kamerawinkels

Image Processing wird hauptsächlich mit Python und OpenCV Bibliothek durchgeführt. Jedes aufgenommene Bild durchläuft mehrere verschiedene Bildverarbeitungsschritte, um den aktuellen Standort des Fahrzeugs auf der Fahrspur zu bestimmen. Wie in Abbildung 28 dargestellt wird, liefert 640x 480 Auflösung wegen der Hardware Beschränkung nur bis 90 FPS. Nach einigen Versuchen konnte festgestellt werden, dass Bilder vom Kameramodul v2 mit einer Auflösung von 640x480 Pixeln kontinuierlich mit 40 FPS aufgenommen werden mussten, damit ein ausreichender Gesichtswinkel erreicht werden konnte. (siehe Abbildung 39).

Resolution	Aspect Ratio	Framerates	Video	Image	FoV
2592x1944	4:3	1-15fps	x	x	Full
1296x972	4:3	1-42fps	x		Full
1296x730	16:9	1-49fps	x		Full
640x480	4:3	42.1-60fps	x		Full
640x480	4:3	60.1-90fps	x		Full
1920x1080	16:9	1-30fps	x		Partial

Abbildung 39: PiKamera Auflösung/ FPS (Hughes, 2013)

Eine niedrige Auflösung (640x480 statt 1296x730) und 40 FPS wurde gewählt, um die Belastung der erforderlichen Verarbeitung der Bilder zu verringern, während immer noch genug Details bereitgestellt wurden, um die Fahrspurmarkierungen zu erkennen.

Abbildung 40: Unterschied zwischen 40FPS und 60 FPS

Wenn der Blickwinkel eng ist, werden die Linien an der rechten oder linken Seite der Kurve nicht gesehen. Aus diesem Grund kann das Fahrzeug nicht wie gewünscht sich auf der Fahrspur halten. Daher ist es notwendig, den Blickwinkel wie oben beschrieben zu erweitern.

5.2 Optimierung der Objekterkennung

Viele Operationen werden auf dem Bildschirm ausgeführt, die zur Objektidentifikation verwendet werden. Nachdem der Code geschrieben wurde, wird jeder Pixel im Bild untersucht und es wird versucht diesen mit den zuvor gelernten Objekten zu vergleichen. Dieser Prozess ist für den Prozessor sehr arbeitsaufwändig. Um die Belastung des Prozessors zu verringern, wird das zu beschriftende Bild im Allgemeinen durch Konvertieren in Grautöne verwendet. In den Experimenten wurde jedoch festgestellt, dass dies nicht ausreicht. Ferner wurde festgestellt, dass die Empfindlichkeit gegenüber der Umgebung aufgrund der durchgeführten Operationen dennoch sehr hoch ist. Denn die ständige Bewegung des Autos und die kontinuierliche Verarbeitung der Bilder in der Umgebung nehmen die irrelevanten Bilder manchmal als vorher definierte Objekte wahr. Um dies zu verhindern, werden die vorherrschenden Farben auf den Schildern getrennt.

Abbildung 41 : Rote Maske für Objekterkennung

Abbildung 42 : weiße Maske für Objekterkennung

Da die zur Objektidentifizierung zu verwendenden Verkehrszeichen normalerweise rote und weiße Farben enthalten, trennt die HSV-Farbskala die weißen und roten Farben von anderen Farben und stellt sicher, dass nur rot-weiße Farbtöne auf dem Bildschirm angezeigt werden, was zu einem Erscheinungsbild führt, das nicht von der Umgebung beeinflusst wird.

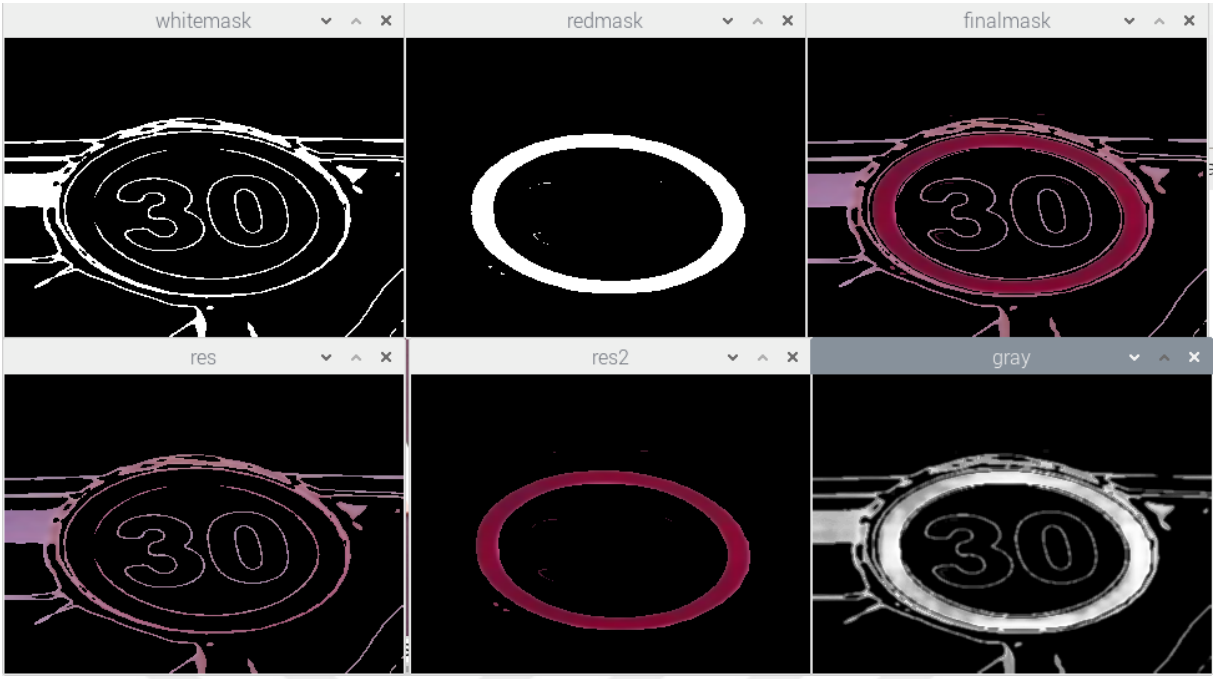

Abbildung 43 : rot-weißes Maskenergebnis

Das Bild, das für die Objektdefinition verwendet wird, entsteht somit wie in Abbildung 35 gezeigt. Mithilfe dieses Bildes wird sichergestellt, dass nur die benötigten und notwendigen Objekte erkannt werden.

5.3 PID Optimierung

Der Fehlerwert, der als der Abstandsunterschied zwischen der Bildmitte und dem Mittelpunkt der Streifen definiert wird, wird verwendet, um das Fahrzeug auf der Spur zu halten. Eine zu starke Änderung dieses Wertes in den Kurven erschwert dem Fahrzeug die Einhaltung der Fahrspur.

Kontrollversuche mit Rohdaten zeigten, dass das Fahrzeug nicht effektiv genug ist. Aus diesem Grund wurde die Anwendung der PID-Kontrolle als angemessen befunden. Durch Änderung der K_p, $K_d$und K_i -Werte kann das Fahrzeug besser an die Variablen auf der Straße angepasst werden. Wenn die PID-Werte nicht ausreichend optimiert werden, kann es dem System mehr Schaden als Nutzen zufügen. Da es zum Beispiel außer Kontrolle geraten kann.

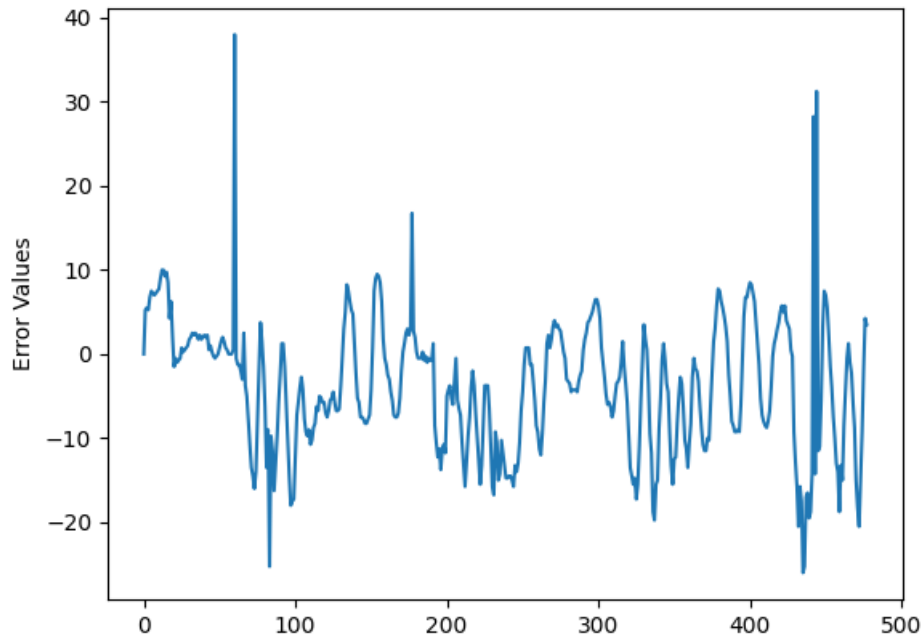

Abbildung 44 : ohne PID

Die Werte, die während der PID-Optimierung bestimmt werden sollten, sind K_p, K_d und K_i-Werte. Wenn einer dieser Werte hoch oder niedrig ist, treten Probleme bei der Stabilisierung des Fahrzeugs auf.

Grundsätzlich kann erklärt werden, was die Änderung dieser Werte. Die sind folgendes;

- K_p beschleunigt das Erhöhen des Werts von dem System auf den Sollwert.
- K_d versucht zu verhindern, dass das System den gewünschten Wert überschreitet.
- K_i verhindert, dass das System nach Erreichen des gewünschten Punktes einen dauerhaften Fehler hinterlässt.

Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Abbildung 45 : PID-Werte und ihre Einflüsse (Mike, 2018)

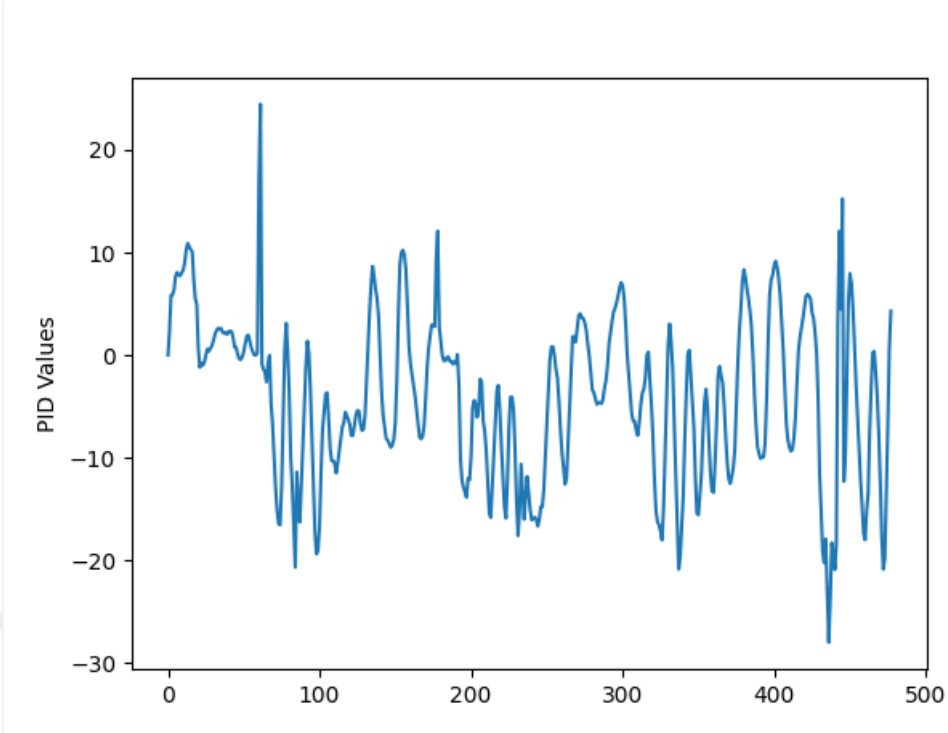

Abbildung 46 : PID (mit K_p:0.07 K_i: 0.05 K_d:0.02)

In Anbetracht der erzielten Ergebnisse wird im nächsten Abschnitt erläutert, ob die Anforderungen erfüllt sind.

6 Verifizierung

In diesem Abschnitt wird erläutert, ob die zuvor ermittelten Anforderungen erfüllt wurden und wenn ja, inwieweit diese Erwartung erfüllt wurde. Ob die in Abschnitt 3 genannten Anforderungen erfüllt wurden oder nicht, wurde mit Hilfe der Sterne angezeigt.

Anforderungen der Hardwares

Dieser Abschnitt erklärt die Verifizierung der funktionalen Anforderungen der Hardwares. Der Erfüllungsgrad der funktionalen Anforderungen der Hardwares ist in Tabelle 8 dargestellt und wie folgendes ausgeführt.

Raspberry Pi Kamera Module (K03) wird aus der Fahrtrichtung des Fahrzeugs erhaltene Daten zu Raspberry Pi (K01) gesendet. MegaPi wird von Ultraschallsensor(K04) erhaltene Informationen zu MegaPi (K02) geliefert. Damit alle von Umgebung erhaltene Daten in Raspberry Pi gesammelt werden. Die verarbeiteten Daten werden über RaspberryPi (K01) und MegaPi (K02) an den Makeblock MegaPi Encoder(K06) übertragen, der mit dem MegaPi verbunden ist. Auf diese Weise können die an die Räder angeschlossenen Makeblock Optical Encoder Motoren (K05), die die letzte Stufe darstellen, angetrieben werden. Dadurch wird die Bewegung des Fahrzeugs sichergestellt.

Tabelle 8 : Erfüllung der Anforderungen des Hardwares

Nr	Komponent	Stück	Erfüllung
K01	Raspberry Pi 4	1	★★★
K02	Makeblock Megapi	1	★★★
K03	Raspberry Pi Kamera Module	1	★★★
K04	Makeblock Ultraschallsensor	1	★★★
K05	Makeblock Optical Encoder Motor	2	★★★
K06	Makeblock Megapi Encoder/DC Motor Driver	2	★★★

Funktionale Anforderungen der Hardwares

Dieser Abschnitt erklärt die Verifizierung der funktionalen Anforderungen der Hardwares. Der Erfüllungsgrad der funktionalen Anforderungen der Hardwares ist in Tabelle 9 dargestellt und wie folgendes ausgeführt.

Rasberry Pi und Kamera Module Verbindung (V01) spielt eine große Rolle. Mit Hilfe dieser Verbindung ist es möglich, die für die Bewegung des Fahrzeugs erforderlichen Fahrspuren zu finden und die Verkehrsschilder zu bestimmen. Wenn bei dieser Verbindung Probleme auftreten oder diese aus irgendeinem Grund unterbrochen wird, wird das Fahrzeug keine Funktionen haben. RaspberryPi und MegaPi Verbindung (V02) kann durch das Kabel gewährleistet werden. Ohne diese Verbindung können aus der Bildverarbeitung abgerufene Informationen nicht mehr in Aktionen

umgewandelt werden. MegaPi und DC Motor-Driver Verbindung (V03) ist die an der einfachsten herzustellenden Verbindung und ermöglicht die Einstellung, wie viel Leistung innerhalb der Daten vom MegaPi auf den Motor übertragen wird. DC-Motor Driver und Encoder Motor Verbindung(V04) macht die elektrische Regelung, die das Einfrieren des Motors gewährleistet und den Motor laufen lässt. Ultraschall Sensor und MegaPi Verbindung(V06) erkennt, wie weit ein Hindernis in Fahrtrichtung des Fahrzeugs vom Fahrzeug entfernt ist.

Tabelle 9 : Erfüllungsgrad der funktionalen Anforderungen der Hardwares

Nr	Funktionale Anforderungen der Hardwares	Erfüllung
V01	Raspberry Pi und Kamera Module Verbindung	★★★
V02	Raspberry Pi und MegaPi Verbindung	★★★
V03	MegaPi und DC Motor-Driver Verbindung	★★★
V04	DC Motor-Driver und Encoder Motor Verbindung	★★★
V05	Encoder Motor und Reifen Verbindung	★★★
V06	Ultraschall sensor und Megapi Verbindung	★★★

Funktionale Anforderungen des Systems

Dieser Abschnitt erklärt die Verifizierung der funktionalen Anforderungen der Hardwares. Der Erfüllungsgrad der funktionalen Anforderungen der Softwares ist in Tabelle 10 dargestellt und wie folgendes ausgeführt.

Zunächst muss die Verbindung zwischen den Komponenten(A01) hergestellt werden. Um notwendige Bibliotheken herunterladen(A03) zu können, wird zuerst Betriebssystem auf RaspberryPi (A02) aufgeladen werden und es muss in SD-Karte geladen werden. Durch Einstecken der SD-Karte wird RaspberryPi einsatzbereit. Anschließend sollte durch Herstellen einer Verbindung zum Computer mit MegaPi USB die MegaPi-Firmware(A04) mithilfe der Arduino IDE geladen werden. Nach diesen Prozessen kann jedoch die Verbindung zwischen RaspberryPi und MegaPi hergestellt(A05) werden. Dann wird die Kamera an den speziellen Eingang von Raspberry Pi angeschlossen und das Bild wird erhalten(A06). Die Einstellungen für Auflösung und FPS(A07) werden angepasst, um den Blickwinkel und die Bildqualität der Kamera zu bestimmen. Das resultierende Bild fokussiert auf das Segment mit den Streifen und der ROI(A08) wird bestimmt. Diese Ansicht wird aufrecht gestellt, um eine Vogelperspektive(A09) zu ermöglichen. Die Farbskala muss geändert(A10) werden, damit die Farben in diesem Bild getrennt werden. Durch Bestimmen der unteren und oberen Werte der weißen Farbe(A11) wird diese Farbe vom Bild getrennt. Weiße Streifenlinien sind alles, was im Bild verbleibt. Um diese Linien verwenden zu können, müssen jedoch die Koordinaten der angegebenen Linien(A12) ermittelt werden. Durch die Optimierung der Fahrspurerkennung(A13) kann das Fahrzeug diese Linien während der Fahrt mit weniger Fehlern begegnen. Durch die Integration des Ultraschallsensors(A14) wird sichergestellt, dass Hindernisse im Fahrzeugweg gefunden werden und eine sichere Bremsung erfolgt. Mit Hilfe des im RaspberryPi enthaltenen Programms, mit dem die von der Kamera aufgenommenen Bilder verarbeitet, werden die Verkehrszeichen erkannt(A15) und es wird gewährleistet, das das Fahrzeug sich gemäß den definierten Zeichen bewegt.

Tabelle 10 : Erfüllung der funktionalen Anforderungen des Systems

Nr	Funktionale Anforderungen des Systems	Erfüllung
A01	Verbindung der Komponenten	★★★
A02	Aufladung des Betriebssystems auf RasypberryPi	★★★
A03	Notwendigen Bibliotheken herunterladen	★★★
A04	Aufladung Megapi firmware auf MegaPi	★★★
A05	Verbindungsherstellung zwischen Megapi und Raspberry Pi	★★★
A06	Aufnahmen von der Kamera	★★☆
A07	Auflösung und FPS definieren	★★★
A08	Region of interest(ROI) bestimmen	★★★
A09	Erstellung der Vogelperspective	★★★
A10	Änderung der Farbskala	★★★
A11	Weißen Spurlinien herusfiltern	★★★
A12	Erkennung den Spurlinien	★★★
A13	Optimierung der Fahrspurerkennung	★★★
A14	Integration des Ultraschallsensors	★★★
A15	Erkennung der Verkehrschilder	★★☆

Erfüllung der funktionalen Anforderungen der Sofwares

Dieser Abschnitt erklärt die Verifizierung der funktionalen Anforderungen der Sofwares. Der Erfüllungsgrad der funktionalen Anforderungen der Softwares ist in Tabelle 11 dargestellt und wie folgt ausgeführt.

Tabelle 11 : Erfüllung der funktionalen Anforderungen der Softwares

Nr	Funktionale Anforderungen der Softwares	Version	Erfüllung
S1	Python	v2.7.16	★★★
S1.1	Open-CV Bibliothek (für Python)	v4.5.0-pre	★★★
S1.2	Megapi Bibliothek (für Python)	v0.2.2	★★★
S1.3	Numpy Bibliothek (für Python)	v1.16.2	★★★
S1.4	Picamera Bibliothek (für Python)	v1.13	★★★
S2	Arduino IDE	v1.8.13	★★★
S2.1	Makeblock Bibliothek (für Arduino IDE)	v3.27	★★★

Es muss eine Programmiersprache ausgewählt werden, um das gesamte System zu steuern. Es kann nicht mehr als eine Programmiersprache ausgewählt werden. Bei der Auswahl der Programmiersprache muss zunächst festgelegt werden, welche Sprache die zu verwendenden Komponenten unterstützen.

C++ wird für solche Projekte häufiger verwendet. Dies liegt daran, dass die OpenCV-Bibliothek (S1.1) ursprünglich für die Sprache C++ geschrieben wurde. Da das in diesem Projekt verwendete MegaPi-Steuerelement jedoch von Python bereitgestellt wird und OpenCv mit einigen Änderungen für die Python- Programmiersprache verwendet werden kann, wurde die Python- Programmiersprache (S1) ausgewählt.

Nach dem Herunterladen von Python muss die OpenCV-Bibliothek aus dem Python-Programm heruntergeladen werden. Die OpenCV-Bibliothek bietet mit vorgefertigten Bildverarbeitungscodes einen großen Komfort. Außerdem muss die MegaPi-Bibliothek (S1.2) für Python heruntergeladen werden, um die in der Python-Sprache in RasperryPi geschriebenen Befehle über Megapi zu übertragen. Wenn diese Bibliothek nicht heruntergeladen wird, starten die Motoren des Fahrzeugs nicht.

Picamera Bibliothek ermöglicht den Bildern, die aus Kamera erhalten werden, mit den spezifische Eigenschaften aufzunehmen. Die Bilder werden innerhalb der gewünschten Kriterien von der Kamera aufgenommen. Außer der Picamera Bibliothek können auch andere Befehle verwendet werden, um Bilder von der Kamera aufzunehmen. Wenn diese Befehle verwendet werden, können die Bilder jedoch verzerrt sein. (Siehe Abbildung 47)

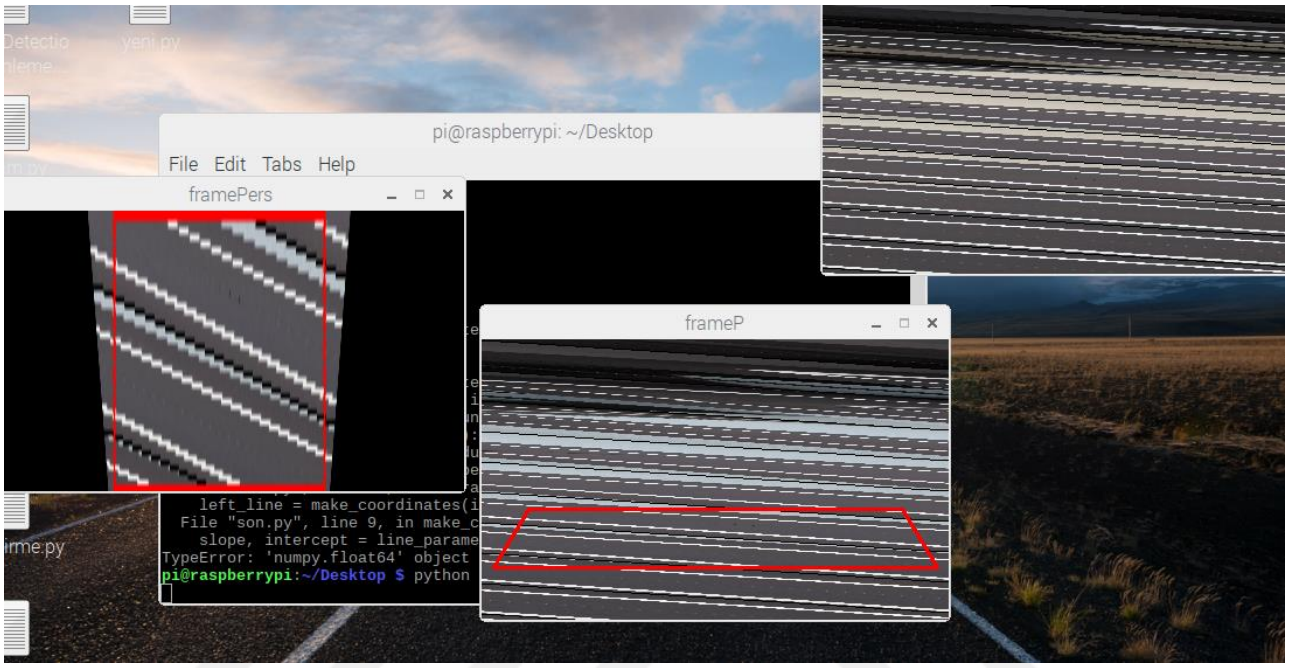

Abbildung 47 : Defekte Aufnahme

Numpy-Bibliothek wird zum Erstellen von Arrays verwendet. Dank dieser Bibliothek kann mehr als ein Wert in einer Variablen gespeichert werden und diese Nummern können in der angegebenen Reihenfolge gespeichert werden.

7 Zusammenfassung

Mit der Verwendung von Raspberry-Pi und Raspberry-Pi Kamera Modulen, die eine wichtige Rolle bei der Erkennung der Fahrspur des Fahrzeugs spielen, wird die Fahrtrichtung permanent beobachtet. Die von der Kamera ermittelten Daten werden unmittelbar an Raspberry Pi übermittelt. Die Erkennung der Fahrspurlinien und der Verkehrszeichen ermöglichen dem Fahrzeug eine autonome Fortsetzung der Fahrt zwischen den Fahrspurlinien. Dabei wird gewährleistet, dass das Fahrzeug die Verkehrszeichen berücksichtigt und sich an die Verkehrsregeln hält. Zusätzlich können mit dem Abstandssensor die Hindernisse vor dem Fahrzeug erkannt werden. Auf diese Weise wurde ein sicheres Fahrverhalten des Fahrzeugs gefördert.

Zur Testung des einwandfreien Betriebs dieses Systems wurde eine Teststrecke mit einem schwarzen Hintergrund und weißen Linien erstellt, damit der Eindruck einer „echten“ Straße vermittelt werden konnte, auf der sich das Fahrzeug vorwärtsbewegen soll. Zusätzlich wurde die Teststrecke mit Ampeln und Verkehrsschildern ausgestattet. Das Fahrzeug erkennt zunächst die Fahrspuren, die umliegenden Verkehrszeichen und entscheidet was zu tun ist, um sich auf der Strecke fortzubewegen. Wenn in der Zwischenzeit vor dem Fahrzeug ein Hindernis auftaucht, wird die Stromversorgung des Motors automatisch unterbrochen. Das Fahrzeug kommt somit zum Stillstand, was in Folge einen eventuellen Aufprall vermeidet.

Die Leitfrage des Projektes, ob es möglich ist, das Fahrzeug mit nur einer Kamera und einem Abstandssensor auf der Fahrspur zu halten, kann -auch wenn mit Begrenzungen- nach den durchgeführten Versuchen mit einem „Ja“ beantwortet werden.

Mittels einer harmonischen Integration von entsprechender Hardware und Software ist es durchaus möglich ein autonomes Fahrzeug zu entwickeln, welches sich mit Hilfe von PID-Regler in einer Teststrecke innerhalb von Fahrtstreifen den Erwartungen entsprechend bewegen kann. Zusätzlich berücksichtigt das Fahrzeug mit der Kamera die Verkehrsregeln, erkennt mit dem Abstandssensor etwaige Objekte auf der Fahrspur und reagiert entsprechend.

Die in der Aufgabestellung gestellten Fragen können wie folgt beantwortet werden.

Welche Auswirkungen haben Kameraauflösung und FPS-Werte auf das System?

Kamera Auflösung und FPS Werte haben direkte Auswirkungen auf das System und beeinflussen den Blickwinkel der Kamera. Je mehr Daten verarbeitet werden müssen, desto mehr wird das System belastet. Dies verzögert den Verarbeitungsprozess der Daten.

Als Ergebnis der Experimente wurden die Auflösungs- und FPS-Werte als 640x480 und 40 FPS bestimmt. Diese ermittelten Werte ermöglichen einen ausreichenden Kamerawinkel und bieten eine optimale Bilderkennung.

Kann die Objektidentifikation effizienter gestaltet werden?

Die durchgeführten Tests haben gezeigt, dass bei der Erkennung der Verkehrsschilder Lücken entstehen, sodass diese fallweise vom System nicht erkannt wurden. Eine genauere Untersuchung hat gezeigt, dass mangelnde Leistung des Computers und unzählige aufwändige Informationen, die sich auf dem Erkennungsbild befinden, dieses Problem verursachen.

Um das festgestellte Problem zu lösen, ist es notwendig, die überschüssigen Informationen im Bild zu reduzieren. Die Lösung besteht darin, die Farben im Bild zu filtern. Da Verkehrsschilder im Allgemeinen rote und weiße Farben haben, bietet das Trennen dieser Farben vom Bild eine gute Möglichkeit für eine effektivere Objekterkennung. Dadurch wurde eine große Zahl von Daten auf das nötigste reduziert, was sich in weiterfolge positiv auf die Datenverarbeitung auswirkt.

Wie wirkt sich die Ergänzung einer PID-Regelung zum System auf die Spurverfolgung des Systems aus?

Mit der Ergänzung des PID-Reglers zu den Codes wurde die Bestimmung der Werte von K_p, K_i und K_d erwirkt, was sich auf den Fahrstil des Fahrzeugs durch Reduzierung der Vibrationen und bessere Reaktion bei Kurven durchaus positiv ausgewirkt hat.

Nach dieser kurzen Zusammenfassung werden Empfehlungen für zukünftige Studien im nächsten Abschnitt erläutert.

8 Ausblick

Die Leistung des Fahrzeugs auf geraden Straßen und in scharfen Kurven wurde durch die PID-Anwendung verbessert. In der etablierten Teststrecke wurden verschiedene Experimente durchgeführt und es wurde festgestellt, dass das installierte System von der Lichtstärke beeinflusst wird. Mangelnde Lichtintensität in der Umgebung erschweren die Erkennung der Fahrspuren und Verkehrsschilder.

Zur weiteren Entwicklung des Systems kann ein Computer mit einer höheren Prozessorgeschwindigkeit verwendet werden. Dadurch können die von der Kamera erlangten Daten auf den Bildern schneller und intensiver verarbeitet werden. Dies würden die Wahrnehmung und Erkennung der Umgebung des Fahrzeugs fördern. Auf diese Weise können mehr Details untersucht und immer effizientere Informationen über die Umgebung mit mehr FPS erhalten werden.

Darüber hinaus sorgt der Abstandssensor direkt vor dem Fahrzeug dafür, dass das Fahrzeug nur vor Hindernissen bzw. sonstigen Gefahren von vorne geschützt wird. Was rund um das Fahrzeug um 360 Grad passiert, kann nur mit zusätzlichen Kameras oder Abstandssensoren ermittelt werden. Auf diese Weise kann sichergestellt werden, dass das Fahrzeug alles, was um sich herum ist verfolgen und erkennen kann. Dies gewährleistet eine sichere und verlässliche Fahrt.

Die Leistung des Fahrzeugs in scharfen Kurven wurde durch die PID-Anwendung verbessert. In Kurven mit einem Winkel von mehr als 75 Grad bewegt sich das Fahrzeug jedoch leicht außerhalb der Fahrspur. Eine Kamera mit einem größeren Winkel kann verwendet werden, um dies zu überwinden. Auf diese Weise kann in sehr scharfen Kurven eine bessere Effizienz erzielt werden.

Dieses System kann ausgiebig genutzt werden. Um es in verschiedenen Bereichen verwenden zu können, müssen die erforderlichen Änderungen entsprechend der Farbe der Fahrspuren vorgenommen werden, auf der das Fahrzeug fahren wird, wie in Kapitel 4.3.8 gezeigt. Auf diese Weise kann das System für den Transport von Produkten in den Lagern, die Übertragung von militärischem Material, den Transport von Menschen mit einer Behinderung und ähnliche Situationen verwendet werden.

Literaturverzeichnis

AashalKamdar. (25. 11 2019). *GeeksforGeeks*. Von https://www.geeksforgeeks.org/python-haar-cascades-for-object-detection/ abgerufen

Ali, A. M., Mohammed, A. H., & Alwan, H. M. (2019). Tuning PID Controllers for DC Motor by Using Microcomputer. *International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 1* , 202-206.

Åström, K. J. (2002). PID Control. In *Control System Design* (S. 216-250).

Bao, P., Zhang, L., & Wu, X. (9. September 2005). Canny Edge Detection Enhancement. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, S. 1485-1490.

Böcker, P. D.-I. (2016). *Geregelte Drehstromantriebe.* Paderborn: Universität Paderborn.

Conrad. (Januar 2021). *Conrad*. Von https://www.conrad.com/p/makeblock-motor-180-optical-encoder-motor-2228479 abgerufen

Direnc. (Januar 2021). *Direnc*. Von https://www.direnc.net/raspberry-pi-kamera-modulu-v2-en abgerufen

Elektronik Kompendium. (17. 09 2020). Von https://www.elektronik-kompendium.de/sites/raspberry-pi/2109051.htm abgerufen

Gentsos, C., Sotiropoulou, C.-L., & Nikolaidis, S. (2010). Real-Time Canny Edge Detection Parallel. *IEEE International Conference on Electronics, Circuits and Systems* (S. 499-502). Thessaloniki, Greece : IEEE.

Hirzel, T. (05. Februar 2018). *Arduino*. Von https://www.arduino.cc/en/Tutorial/Foundations/PWM abgerufen

Hughes, D. (2013). *PiCamera*. Von https://picamera.readthedocs.io/en/release-1.3/fov.html#under-the-hood abgerufen

Kacmajor, T. (05. Januar 2017). *tomaszkacmajor.pl*. Von https://tomaszkacmajor.pl/index.php/2017/06/05/hough-lines-transform-explained/ abgerufen

Luber, S., & Litzel, N. (24. Juli 2018). *Bigdata Insider*. Von https://www.bigdata-insider.de/was-ist-numpy-a-735687/#:~:text=NumPy%20vereinfacht%20die%20Verwendung%20von,Bibliothek%20sind%20speziell%20hierfür%20optimiert. abgerufen

Makeblock. (24. Januar 2019). *Makeblock*. Von https://www.makeblock.com/project/megapi-pro-encoder-dc-motor-driver abgerufen

MakeBlock. (24. 01 2019). *MakeBlock*. Von https://www.makeblock.com/project/megapi-pro-encoder-dc-motor-driver abgerufen

MakeBlock. (15. 01 2020). Von http://learn.makeblock.com/python-for-megapi/ abgerufen

Makeblock. (Januar 2021). *Makeblock*. Von http://learn.makeblock.com/en/megapi/ abgerufen

Matas, J., Galambos, C., & Kittler, J. (2000). Computer Vision and Image Understanding. In *Robust detection of lines using the progressive probabilistic hough transform* (S. 119–137). Elsevier Inc.

MathWorks, T. (Januar 2021). *The MathWorks*. Von http://matlab.izmiran.ru/help/toolbox/images/color4.html abgerufen

Mike. (Mai 2018). *Instrumentation Forum*. Von https://instrumentationforum.com/t/pid-controller-manual-tuning/4043 abgerufen

Mordvintsev, A., & K, A. (2013). *Open-CV*. Von https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html abgerufen

MrHezhisheng. (18. Februar 2019). *Github*. Von https://github.com/Makeblock-official/PythonForMegaPi/tree/master/examples abgerufen

Nagabhushana, S. (2005). *Computer Vision and Image Processing.* New Age International.

O'Hanlon, M. (2020). *raspberrypi.org*. Von https://projects.raspberrypi.org/en/projects/robotPID abgerufen

Paktechpoint. (2018). Von https://paktechpoint.com/pid-control-basics-paktechpoint/ abgerufen

Prack, D. N. (19. 08 2020). *ADAC*. Von ADAC: https://www.adac.de/news/bilanz-verkehrstote/ abgerufen

Rafael C. Gonzalez, & Woods, R. (2004). *Digital Image Processing.* New Jersey: Prentice Hall.

RaspberryPi. (2021). Von https://www.raspberrypi.org/documentation/hardware/camera/ abgerufen

Rong, W., Li, Z., & Sun, W. Z. (2014). *An Improved Canny Edge Detection Algorithm.* Tianjin, China: IEEE.

seed. (2020). Von https://www.seeedstudio.com/Raspberry-Pi-4-Computer-Model-B-4GB-p-4077.html abgerufen

Soo, S. (2014). *Object detection using Haar-cascade Classifier.* University of Tartu: Institute of Computer Science, University of Tartu, 1-12.

TA WIKI. (29. August 2018). Von https://wiki.ta.co.at/PID-Regelung_(Funktion)#:~:text=PID-Werte,-Der%20Proportionalteil%20P&text=Der%20Integralteil%20I%20stellt%20die,Proportionalteil%20verbliebenen%20Abweichung%20periodisch%20nach.&text=Weicht%20der%20Ist-%20vom%20Sollwert,Stellgröße%2 abgerufen

The MathWorks, I. (26. 10 2020). *MathWorks*. Von https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html abgerufen

Waycon. (2020). Von https://www.waycon.de/produkte/ultraschallsensoren/messprinzip-ultraschallsensoren/ abgerufen

Wikipedia. (15. Januar 2021). Von https://en.wikipedia.org/wiki/HSL_and_HSV#Use_in_image_analysis abgerufen

Xue, D., Chen, Y., & Atherton, D. (2007). PID Controller Design. In *Linear Feedback Control* (S. 183-235).

Zhang, Y. J. (2007). Image Engineering(II)Image Ananlysis. *Tsinghua University press*, S. 75-85.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

PID Kontroller..... Proportional-integral-derivative controller

DC Motor..... Direct Current motor

PWM ..... Pulse-Width Modulation

GPIO..... General Purpose input/output

K_p ..... Proportional gain

K_d ..... Derivative gain

K_i ..... Integral gain

SSH ..... Secure Shell

XML ..... Extensible Markup Language

FPS..... Frame Per Second

UART ..... Universal Asynchronous Reception and Transmission

I2C ..... Inter-integrated-circuit

SPI ..... Serial Peripheral interface

Anhang

```
import cv2
import numpy as np
import math
import sys
import time
from picamera.array import PiRGBArray
from picamera import PiCamera
from megapi import *
distance=123
def Ultraschall(v):
    global distance
    #print("distance:"+str(v)+" cm")
    distance = v

bot = MegaPi()
bot.start()
Source = np.float32([[0, 240], [30, 150], [370, 150], [400, 240]])
Destination = np.float32([[0, 240], [0, 0], [400, 0], [400, 240]])

#stop_cascade                                                    =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Stop_cascade.xml')
stop_cascade                                                    =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Stop_cascade.xml')
thirty_cascade                                                  =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Thirty_cascade.xml')
fifty_cascade                                                   =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Fifty_cascade.xml')
stop_light_cascade                                              =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/Stop_light_cascade.xml')
redlight_cascade                                                =
cv2.CascadeClassifier('/home/pi/Desktop/cascade/lastredlight_cascade.xml')
font = cv2.FONT_HERSHEY_PLAIN

def Perspective(frame):
    frameP = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    Scalar = (0, 0, 255)
    frameP = cv2.line(frameP, (0, 240), (30, 150), Scalar, 2)
    frameP = cv2.line(frameP, (30, 150), (370, 150), Scalar, 2)
    frameP = cv2.line(frameP, (370, 150), (400, 240), Scalar, 2)
    frameP = cv2.line(frameP, (400, 240), (0, 240), Scalar, 2)
    #cv2.imshow("frameP", frameP)
    Matrix = cv2.getPerspectiveTransform(Source, Destination)
    framePers = cv2.warpPerspective(frameP, Matrix, (400, 240))
    #cv2.imshow("framePers", framePers)

    return framePers

def detect_edges(frame):
    # filter for blue lane lines
    hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)
```

```
    #cv2.imshow("HSV",hsv)    # (179, 0, 240) (180, 16, 255)
    lower_white = np.array([0, 0, 222], dtype="uint8") #0, 0, 212]  hls0, 200, 0
    upper_white = np.array([180, 40, 255], dtype="uint8") #(133,255,255)180, 40, 255
hls 255,255,255
    colourmask = cv2.inRange(hsv, lower_white, upper_white)
    colourmask = cv2.GaussianBlur(colourmask, (5, 5), 0)
    #cv2.imshow("colourmask",colourmask)
    # detect edges
    edges = cv2.Canny(colourmask, 50, 100)
    cv2.imshow("edges", edges)

    return edges

def detect_line_segments(edges):
    rho = 1
    theta = np.pi / 180
    min_threshold = 15

    #critical performance
    line_segments = cv2.HoughLinesP(edges, rho, theta, min_threshold,
                                    np.array([]), minLineLength=50, maxLineGap=150)

    return line_segments

def average_slope_intercept(frame, line_segments):
    lane_lines = []

    if line_segments is None:
        return lane_lines

    height, width, _ = frame.shape
    left_fit = []
    right_fit = []

    boundary = 1 / 2
    left_region_boundary = width * (1 - boundary)
    right_region_boundary = width * boundary

    for line_segment in line_segments:
        for x1, y1, x2, y2 in line_segment:
            if x1 == x2:
                continue

            fit = np.polyfit((x1, x2), (y1, y2), 1)
            slope = (y2 - y1) / (x2 - x1)
            intercept = y1 - (slope * x1)


            if x1 < left_region_boundary and x2 < left_region_boundary:
                    left_fit.append((slope, intercept))
```

```
            elif x1 > right_region_boundary and x2 > right_region_boundary:
                right_fit.append((slope, intercept))

    left_fit_average = np.average(left_fit, axis=0)
    if len(left_fit) > 0:
        lane_lines.append(make_points(frame, left_fit_average))

    right_fit_average = np.average(right_fit, axis=0)
    if len(right_fit) > 0:
        lane_lines.append(make_points(frame, right_fit_average))

    return lane_lines

def make_points(frame, line):
    height, width, _ = frame.shape

    slope, intercept = line

    y1 = height  # bottom of the frame
    y2 = int(y1 / 2)  # make points from middle of the frame down

    if slope == 0:
        slope = 0.1

    x1 = int((y1 - intercept) / slope)
    x2 = int((y2 - intercept) / slope)

    return [[x1, y1, x2, y2]]


def display_lines(frame, lines, line_color=(255, 0, 0), line_width=7):
    line_image = np.zeros_like(frame)

    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image, (x1, y1), (x2, y2), line_color, line_width)

    line_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    #cv2.imshow("line_image", line_image)
    return line_image


def getting_center_of_lane(frame, lane_lines):
    height, width, _ = frame.shape
    lane_center = width/2

    if len(lane_lines) == 2:
        left_x = lane_lines[0][0][2]
        right_x = lane_lines[1][0][2]
        lane_center = (right_x - left_x)/2 + left_x
```

```
    elif len(lane_lines) == 1:
        x1, _, x2, _ = lane_lines[0][0]
        if x1 < width/2:
            mid = (x1 + x2)/2
            lane_center = width - mid
        elif x1 > width/2:
            mid = (x1 + x2)/2
            lane_center = mid

    elif len(lane_lines) == 0:
        lane_center = (width/2)

    return lane_center

def find_error(frame ,lane_center, line_color=(0, 0, 255), line_width=3):
    Spur_bild = np.zeros_like(frame)
    height, width, _ = frame.shape

    x1 = int(width / 2)
    y1 = int(height)
    x2 = int(width / 2)
    y2 = int(height / 2)

    lane_center = int(lane_center)
    font = cv2.FONT_HERSHEY_PLAIN
    cv2.line(Spur_bild, (x1, y1), (x2, y2), line_color, line_width)
    cv2.circle(frame, (lane_center, y2), radius=3, color=(255, 0, 0), thickness=3)
    Spur_bild = cv2.addWeighted(frame, 0.8, Spur_bild, 1, 1)
    global error
    error = lane_center - x2
    #print(error)

    return Spur_bild


error_value = []
pid_value = []
cumError = 0
sumError= 0
kp = 0.15
kd = 0.1
ki = 0.003
speed = 50
previousTime =0
lastError = 0
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(640, 480))
```

```
# allow the camera to warmup
time.sleep(0.1)
# capture frames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    # grab the raw NumPy array representing the image, then initialize the timestamp
    # and occupied/unoccupied text
    image = frame.array
    # show the frame
    #cv2.imshow("ganze", image)
    frame = cv2.resize(image, (400,240), interpolation = cv2.INTER_AREA)

    image = cv2.GaussianBlur(image,(3,3),cv2.BORDER_DEFAULT)
    sign_30 = image [100:230, 0:180] #[100:300, 440:640] [80:240, 500:640]
    sign_stop = image [180:300, 0:200]
    #sign = cv2.resize(sign, (360, 240))
    gray_30 = cv2.cvtColor(sign_30, cv2.COLOR_BGR2GRAY)
    gray_30 = cv2.equalizeHist(gray_30)
    gray_stop = cv2.cvtColor(sign_stop, cv2.COLOR_BGR2GRAY)
    gray_stop = cv2.equalizeHist(gray_stop)
    rawCapture.truncate(0)
    font = cv2.FONT_HERSHEY_PLAIN


    framePers = Perspective(frame)
    edges = detect_edges(framePers)
    line_segments = detect_line_segments(edges)  # roi
    lane_lines = average_slope_intercept(framePers, line_segments)
    line_image = display_lines(framePers, lane_lines)
    lane_center = getting_center_of_lane(framePers, lane_lines)
    Spur_bild = find_error(line_image, lane_center)
    #mashine learning
    stops = stop_cascade.detectMultiScale(gray_30, 1.1, 5)
    thirtys = thirty_cascade.detectMultiScale(gray_30, 1.1, 5)
    #fiftys = fifty_cascade.detectMultiScale(gray_30, 1.1, 5)
    redlights = redlight_cascade.detectMultiScale(gray_stop, 1.1, 5)

    for (x,y,w,h) in stops:
        cv2.rectangle(sign_stop,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(sign_stop, "StopSign", (x, y + 30), font, 1, (0,0,255), 1)
        cv2.putText(image, "Stop sign detected!!", (50, 50), font, 2, (0, 0, 255), 2)
        if w > 80 and stops is not None:
            #print('stop sign detected')
            bot.encoderMotorRun(1,(0)); #sol motor
            bot.encoderMotorRun(2,-(0));#sag motor (-)
            time.sleep(3)
            bot.encoderMotorRun(1,(speed)); #sol motor
            bot.encoderMotorRun(2,-(speed));#sag motor (-)
            time.sleep(3)
    for (x,y,w,h) in thirtys:
        #print("w")
```

```
        #print(w)
        cv2.rectangle(sign_30,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(sign_30, "thirtys", (x, y + 30), font, 1, (0,0,255), 1)
        cv2.putText(image, "30 sign detected!!", (50, 50), font, 2, (0, 0, 255), 2)
        if w > 80 and thirtys is not None:
            speed= 30
            #print("30 sign detected!!")

    for (x,y,w,h) in fiftys:
        cv2.rectangle(sign_30,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(sign_30, "fiftys", (x, y + 30), font, 1, (0,0,255), 1)
        cv2.putText(frame, "50 sign detected!!", (50, 50), font, 2, (0, 255, 0), 2)
        if thirtys is not None:
            print("50 sign detected!!")
            speed= 50

    for (x,y,w,h) in redlights:
        cv2.rectangle(sign_stop,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(sign_stop, "red light", (x, y + 30), font, 1, (0,0,255), 1)
        cv2.putText(image, "red light detected!!", (50, 100), font, 2, (0, 0, 255), 2)
        if redlights is not None:
            print("red light detected!!")
            bot.encoderMotorRun(1,0); #sol motor
            bot.encoderMotorRun(2,-0);#sag motor (-)
            sleep(2)
    #Ultraschall
    bot.ultrasonicSensorRead(6,Ultraschall)
    #print (distance)
    distance = int(distance)

    hata1 = (error/4)
    hata = abs(hata1)
    hata = int(hata)


    pid_output = kp*error + ki*sumError + kd*lastError    #PID output

    lastError = error
    sumError = lastError + error
    # save variables
    error_value.__iadd__([hata1])
    pid_value.__iadd__([pid_output])

    pidout = abs(pid_output)
    pidout = int(pidout)

    high = speed + pidout
    low = speed - pidout

    if distance < 15:
        cv2.putText(image, "Object detected!!", (50, 100), font, 2, (0, 0, 255), 2)
```

```
        bot.encoderMotorRun(1,0); #sol motor
        bot.encoderMotorRun(2,-0);#sag motor (-)

    elif error > 0:  # rects abbiegen

        cv2.putText(image, "error=" + str(hata1) , (40, 60), font, 2, (0, 255, 0), 2)

        cv2.putText(image, "rechts >>", (100, 400), font, 2, (0, 255, 0), 2)
        bot.encoderMotorRun(1, (high));
        bot.encoderMotorRun(2, -(low));

    elif error < 0:  # links abbiegen
        cv2.putText(image, "error=" + str(hata1)  , (40, 60), font, 2, (0, 255, 0), 2)

        cv2.putText(image, "<< links", (400, 400), font, 2, (0, 255, 0), 2)
        bot.encoderMotorRun(1, (low))
        bot.encoderMotorRun(2, -(high))

    elif error == 0 :  # geradeaus
        cv2.putText(image, "in lane", (250, 400), font, 2, (0, 255, 0), 2)
        bot.encoderMotorRun(1, (low))
        bot.encoderMotorRun(2, -(high))



    cv2.putText(image, "speed=" + str(speed) , (40, 200), font, 2, (0, 255, 0), 2)
    Spur_bild = cv2.addWeighted(Spur_bild, 0.8, Spur_bild, 1, 1)
    cv2.putText(image, "distance=" + str(distance)  , (20, 20), font, 2, (0, 255, 0), 2)
    cv2.imshow("Spur_bild", Spur_bild)
    cv2.imshow("sign_stop", sign_stop)
    cv2.imshow("sign_30", sign_30)
    cv2.imshow("image", image)



    #print("error_value:" + str(error_value))
    #print("pid_value:" + str(pid_value))

    #plt.plot(error_value)
    #plt.ylabel('Error Values')
    #plt.show()

    key = cv2.waitKey(1)
    if key == 27:
        bot.encoderMotorRun(1,0); #left motor
        bot.encoderMotorRun(2,-0);#right motor (-)
        break

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

19.02.2021
Datum


Unterschrift