Raspberry Pi Roboter mit einer Logistikaufgabe: Entwurf, Bau und Programmierung

Maturaarbeit 2018 im Fach Informatik an der

Kantonsschule Sursee

Verfasser

Damian Kopp

St. Ottilienstrasse 35

6018 Buttisholz

Betreuer

Abdelhakim Ghezal

Oberdorfstrasse 22

6207 Nottwil

Inhalt

1 Vorwort

Die Industrialisierung 4.0 ist heutzutage bereits ein verbreiteter Begriff. Mit ihr will man möglichst viele Arbeitsschritte automatisieren, wodurch vermehrt autonom arbeitende Maschinen, sogenannte „Roboter“, eingesetzt werden. Nicht nur in der Produktionsbranche setzt man auf neue Technologien, sondern auch beispielsweise im Verkehr sollen in der Zukunft autonom fahrende Fahrzeuge eingesetzt werden. Mit geschickt programmierten Algorithmen wäre es möglich, Zeit zu sparen und Unfälle zu vermeiden. Seit Aufkommen der computergesteuerten Fahrzeuge hat mich die Thematik und die Vorstellung, eines Tages in einem Fahrzeug ohne Fahrer zur Arbeit zu fahren, fasziniert und beschäftigt. Da mich diese Thematik sehr interessiert war mir bei der Themenfindung für die Maturaarbeit klar, dass ich ein Projekt im Bereich der Robotik verwirklichen möchte. Die Idee alleine, ein autonomes Auto zu bauen, war nicht wirklich präzise. Nach reichlichen Überlegungen, Diskussionen und Nachfragen bei Familie, Freunden und Bekannten kristallisierte sich ein für mich interessantes Projekt heraus. Mein Cousin Kornel Eggerschwiler - selber Ex-Schüler der Kantonsschule Sursee und gegenwärtiger Masterstudent an der ETH Zürich - regte an, einen Roboter mit dem Raspberry Pi als "Gehirn" zu bauen. Ein Raspberry Pi ist kurz erklärt ein kleiner und kostengünstiger Computer, dessen Möglichkeiten nahezu unbegrenzt sind. Bei Kornel habe ich den Raspberry Pi zum ersten Mal gesehen, aber mir war damals noch nicht bewusst, was man alles damit machen kann. Nach Kornels Vorschlag informierte ich mich im Internet über die Möglichkeiten des Raspberry Pi und sah darin das Potenzial, das die erfolgreiche Umsetzung meiner noch unpräzisen Idee begünstigen konnte. Schon bald nach Kornels Vorschlag brachte mir mein Vater eine Idee, die genau meinen Vorstellungen für die Maturaarbeit entsprach. Er schlug mir nämlich vor, einen Roboter zu bauen der im Stande ist, Koffern nach Destinationen zu sortieren und so an Flughäfen zum Einsatz kommen könnte. Es schien mir sehr interessant, diese Idee mit dem Raspberry Pi zu kombinieren. Beide Vorschläge ergänzen sich schliesslich super. Also entschied ich mich schlussendlich dafür, ein Fahrzeug zu bauen und ein Raspberry Pi als „Gehirn“ zu programmieren und einzubauen.

2 Einleitung

Diese Arbeit beschäftigt sich mit dem Bau und der Programmierung dieses Roboters. Die Bildverarbeitung und -erkennung ist dabei immer ein wichtiger Aspekt. Das Ziel war ein fahrender Roboter, der in der Lage ist, Koffer nach Destinationen aufzuteilen und dabei stets auf einer Weglinie zu fahren. Im ersten Teil ist vor allem die Hardware und Elektronik im Vordergrund, wogegen im Hauptteil die Implementierung im Fokus liegt. Motiviert, etwas in der Thematik der Computer Vision zu lernen, wurde die Arbeit ohne grosses Vorwissen in Angriff genommen. Schlussendlich entstand eine vereinfachte Version, bei welcher der Roboter die Zielaufgabe erfüllt.

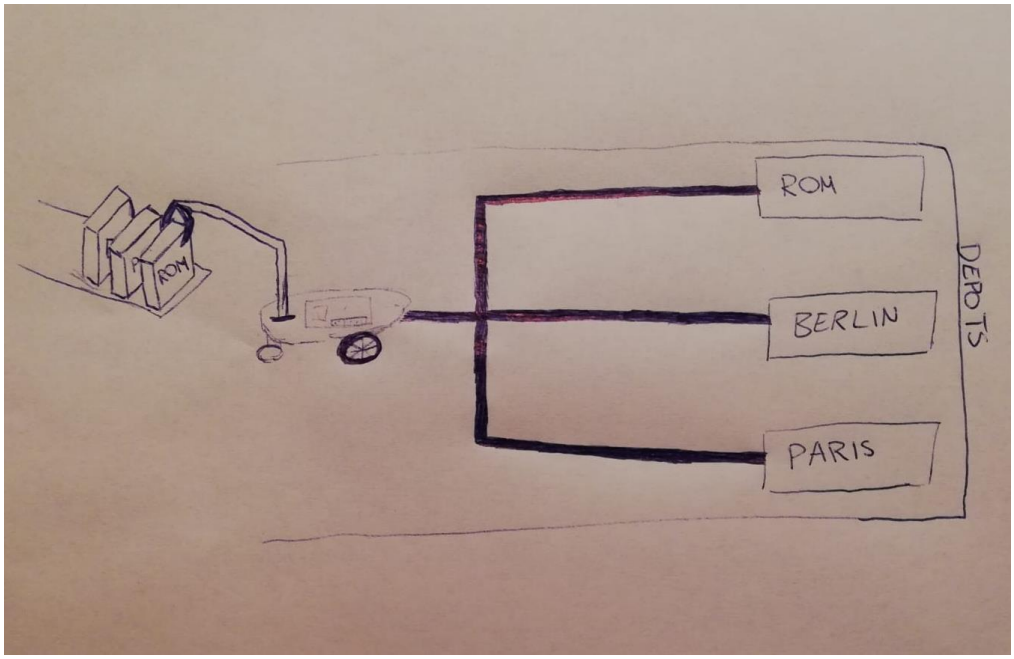


Abb. 1: anfängliche Vorstellung der ganzen Anlage

3 Einführung ins Themengebiet

3.1 Robotik

Abb. 2: Robotermodell aus Plänen von Leonardo da Vinci

Bereits im 15. Jahrhundert entstanden Pläne für Roboter, genauer gesagt für Androiden. Man hatte damals schon Vorstellungen menschenähnlicher Roboter. Es war jedoch wegen mangelndem Wissen noch nicht möglich, diese Visionen umzusetzen. Musikautomaten waren die ersten Maschinen, die automatisch funktionierten und somit einen Meilenstein in der Technik darstellten. 1966 entstand dann der erste Roboter, der auf Bildeinflüsse reagierte und sich im Raum bewegen konnte. Man kann ihn durchaus als Grundlage für die Entwicklung von autonomen Fahrzeugen bezeichnen. Der Fortschritt der Technik durch die Forschung hatte zur Folge, dass die Robotik zu einem enorm grossen Gebiet der Wissenschaft wurde. Die Robotik wurde in viele Teilgebiete unterteilt, in denen man sich spezialisiert hat. Eines dieser Teilgebiete stellen die sogenannten Nanobots dar. Sie sind winzig klein und können von blossem Auge kaum oder gar nicht erkannt werden. Wegen ihrer geringen Größe liegt ihr Einsatzgebiet grösstenteils in der Medizin, wo die Nanobots in Zukunft beispielsweise Krankheiten bekämpfen sollen. Ein weiteres Teilgebiet der Robotik, das heute und in Zukunft eine grosse Rolle spielen wird, ist die künstliche Intelligenz, auch KI genannt. Dieser Bereich befasst sich mit der Nachahmung des menschlichen Gehirns. Man geht davon aus, dass das Bewusstsein vom Gehirn kommt und somit alle Gehirnprozesse zusammen zum Bewusstsein führen. Da Gehirnprozesse gewissermassen auch nach dem Prinzip von Mikrochips ablaufen, mit Ja/Nein-Signalen, sollte es möglich sein, Robotern ein Bewusstsein zu verleihen. Genau damit befasst sich der Bereich der KI. Dafür implementieren Informatiker sogenannte neuronale Netzwerke, die nach einer Trainingsphase lernfähig sind. Die Effekte sind verblüffend und der Einsatzbereich der KI ist sehr groß. Die KI steckt zum Beispiel in Sprachsteuerungen oder kann zur Krebsdiagnostik eingesetzt werden. So kann die KI einen grossen Teil zur Automatisierung von Arbeitsprozessen beitragen.

Die Robotik deckt generell mehrere Wissenschaften ab. Dazu zählen Informatik, Elektrotechnik und Maschinenbau. Ein Teilgebiet der Robotik, in dem die Entwicklung und Erforschung momentan auf Hochtouren läuft und zu welchem man auch mein Projekt zählen kann, ist das Gebiet der autonomen Fahrzeuge. Es befasst sich mit dessen Bau und der Programmierung. Künftig soll neben der Industrie auch der Verkehr weitgehend automatisiert

werden. Dabei müssen schwierige ethnische Probleme gelöst werden, da das Fahrzeug in Gefahrensituationen heikle Entscheidungen treffen muss. Der Vorteil des Einsatzes von autonomen Autos ist beispielsweise, dass man auch bei Müdigkeit mitfahren und sogar schlafen kann. Somit gäbe es auch bedeutend weniger Unfälle. Mein Roboter ist gewissermassen ein autonomes Fahrzeug, erfüllt jedoch industrielle Logistikaufgaben.

3.2 Raspberry Pi

Raspberry Pi ist der Name eines Einplatinencomputers. Er wurde in den letzten paar Jahren immer populärer wegen seinem niedrigen Preis. Ausserdem ist dieser Computer nicht grösser als eine Kreditkarte. Seine Einsatzmöglichkeiten sind undenkbar gross. Mit kreativen Ideen kann diese Platine zu sehr interessanten Projekten führen.
Der Raspberry Pi, kurz Raspi, kann seit 2012 gekauft werden und stammt aus Grossbritannien. Die Entwickler der Raspberry Pi Foundation entwickelten diesen Mini-Computer mit dem Hintergedanken, das Gebiet der Informatik und der Elektrotechnik für Studenten, aber auch für Kinder zu vereinfachen und somit populärer zu machen. Das Programmieren sollte für möglichst viele Bevölkerungsschichten zugänglich gemacht werden, nicht nur den IT-Spezialisten. Dies ist auch der Grund, warum alle Komponenten so klein und auf nur einer Platine zu finden sind. Der Raspi beinhaltet neben einem Ein-Chip-System auch einen ARM-Mikroprozessor. Bei jedem Modell sind wieder andere Schnittstellen vorhanden.

3.3 Computer Vision

Ein digitales Bild besteht aus zahlreichen Pixeln, welche rasterartig nebeneinander liegen. Es können beispielsweise 640 Pixelspalten und 480 Pixelzeilen sein. Dann würde das Bild aus 307‘200, also 640 mal 480, Pixeln bestehen. Bei Digitalfotos besteht jedes Pixel aus drei Farbkanälen – rot, grün und blau. Jeder dieser Farbkanäle enthält eine Anzahl von möglichen Abstufungen, oft sind es 256 Abstufungen (8 Bit = 2^8). In diesem Fall gibt es von jedem Farbkanal 256 verschiedene Farbtöne. Somit erhalten bei Digitalfotos alle drei Farbkanäle eines Pixels einen gewissen Wert. Durch die Mischung dieser Farben kann ein enorm grosser Farbbereich abgedeckt werden. Folglich hat jedes einzelne Pixel eine Farbe, kombiniert aus den drei Farbkanälen.

Um ein Graustufenbild zu generieren, werden alle Pixel des farbigen Digitalfotos verändert. Beim Graustufenbild existiert allerdings nur noch ein Kanal. Der Wert dieses Kanals ergibt sich aus den vorherigen Werten der Farbkanäle.
Es gilt folgende Formel: $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$

Dabei steht Y für den neuen Wert des Graukanals. Er ist die Summe der verschieden gewichteten Farbkanäle des farbigen Digitalbildes. Diese Gewichtung entspricht der Farbempfindlichkeit des menschlichen Auges. Grundsätzlich resultiert auch mit einer anderen Gewichtung ein Graustufenbild. Diese Formel wird an jedem Pixel angewendet und bewirkt somit, dass jedem Pixel ein Wert zwischen 0 und 256 zugewiesen wird, der für die Graustufe steht.

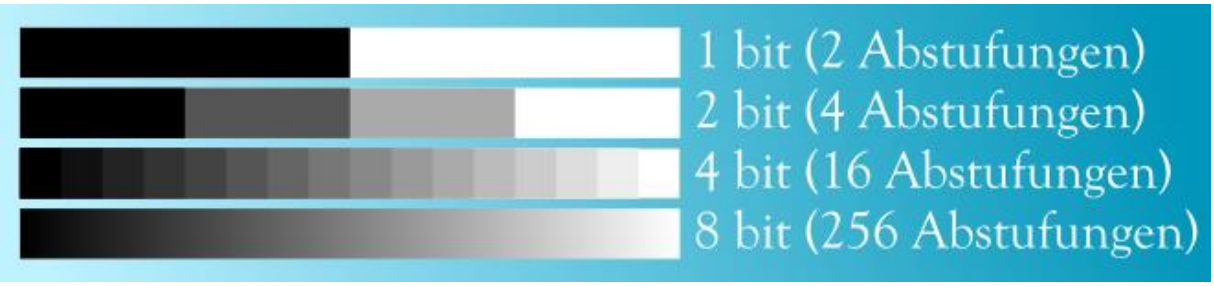
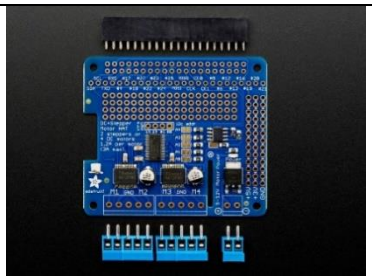
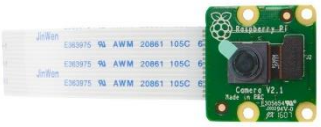
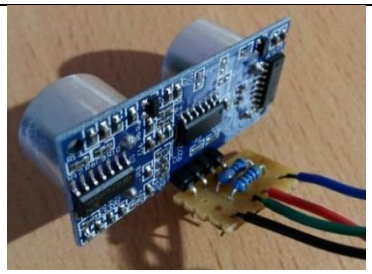
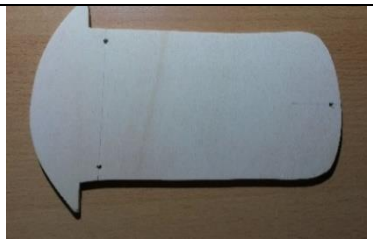
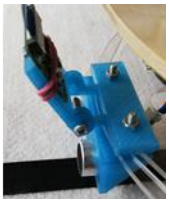


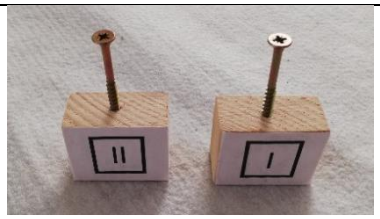
Abb. 3: Anzahl mögliche Graustufen bei verschiedenen Farbtiefen

4 Hardware

4.1 Bauteile

Beschreibung	Funktion	Bild
Roboter Chassis (play-zone.ch): Kunststoffplatte, 2 DC Motoren, 2 Steckräder, bewegliches Nylonrad, Elektronik, Befestigungsmaterial	Grundgerüst; Fahrmechanik; Fahrzeugbasis	 *Abb. 4*
Raspberry Pi 3 Model B (digitec.ch)	Computer; Grundlage für Motor-Hat, Ansteuerung der Kamera, des Distanzsensors und des Magnets; bildet mit Motor-Hat zusammen die Steuerung des Roboters	 *Abb. 5*
Motor-Hat von Adafruit (play-zone.ch)	Steuerung der DC Motoren	 *Abb. 6*
Raspberry Pi Camera (digitec.ch)	Erkennung der Weglinie und der Strichcodes	 *Abb. 7*

Distanzsensor HC-SR04 von Adafruit (play-zone.ch)	Messung der Distanz vom Roboter zum Holzblock	 *Abb. 8*
Power Bank von Varta (digitec.ch)	Energiequelle für Raspi, Motor-Hat und Elektromagnet; wieder aufladbar	 *Abb. 9*
Holzplatte	Befestigungsmöglichkeit für Raspi, Motor-Hat und Greifarm	 *Abb. 10*
Elektromagnet von Grove (distrelec.com)	Blöcke mitnehmen/aufladen	 *Abb. 11*
Kamera- und Sensorhalterung (3D-Drucker)	Befestigung von Kamera und Sensor	  *Abb. 12*

Holzblöcke mit Strichcodes und Schraube	Logistikware, dessen Bestimmungsort mit der Kamera erkannt wird	 *Abb. 13*

4.2 Unterlage

Die Bedingungen für die Unterlage waren, dass sie weiss sein musste und dass die Räder, besonders das bewegliche Nylonrad, genügend Halt hatten darauf. Als erste Unterlage kam ein Flies zum Einsatz, das teppichähnlich und sehr klebrig ist. Der Vorteil dieses Flieses ist, dass es problemlos zusammengefaltet werden kann und das Nylonrad einen guten Halt findet. Dafür resultiert aus der Textur des Flieses, dass viele kleine Kanten erkannt werden, was ein massiver Störfaktor ist. Auch haben die Räder durch die Klebrigkeit einen relativ grossen Widerstand. Die Weglinie wurde mit einem Isolierband geklebt.

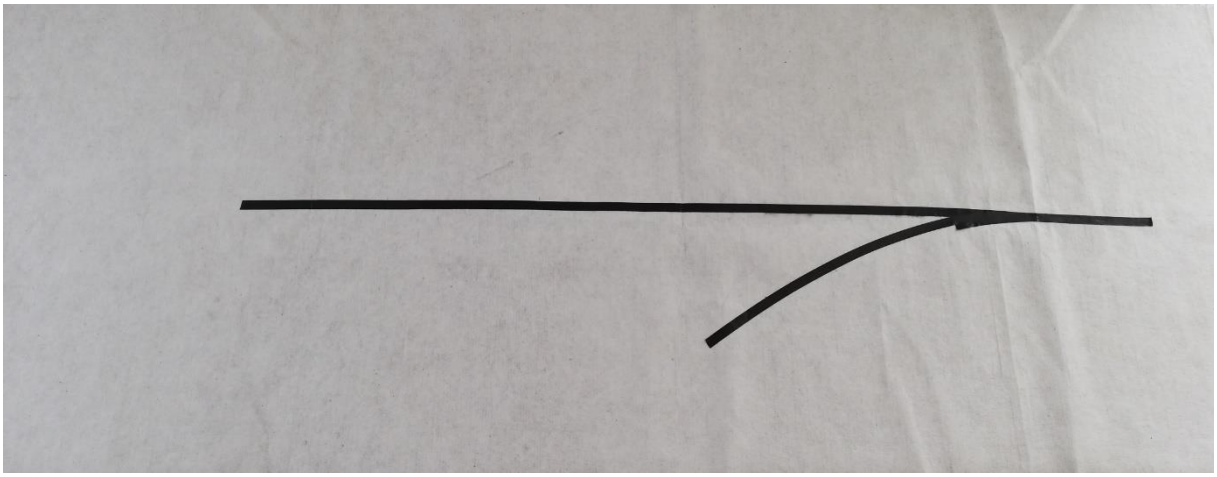


Abb. 14: Flies mit Isolierband als Weglinie

4.3 Raspberry Pi Komponenten

Für meinen Roboter verwende ich ein Raspberry Pi 3 Modell B, das aktuellste Modell. Dieses Modell besitzt eine HDMI Schnittstelle zum

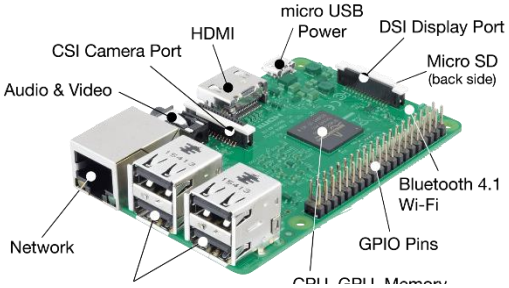


Abb. 15: Raspberry Pi 3 Modell B Komponenten

Verbinden eines Monitors. Zusätzlich sind auf der Platine vier USB Anschlüsse vorhanden. Diese sind notwendig für die Verwendung einer Tastatur und einer Maus. Ausserdem sind die USB Anschlüsse nützlich, um die Programme auf einem Memory Stick zu sichern. Wie auf Abb. 16 sichtbar sind auf der Platine 40 Pins angebracht. Diese GPIO Pins (General Purpose Input/Output) sind Schnittstellenpunkte, die frei programmierbar für Ein- und Ausgabe sind. An diese GPIO Pins kann Zubehör wie LEDs, Knöpfe, Displays etc. angeschlossen werden. Nicht alle Pins haben die gleichen Möglichkeiten, da sie Sondersteuerungen übernehmen können oder mit einer anderen Spannung laufen. Die folgende Tabelle erklärt die Abb. 16 etwas genauer.

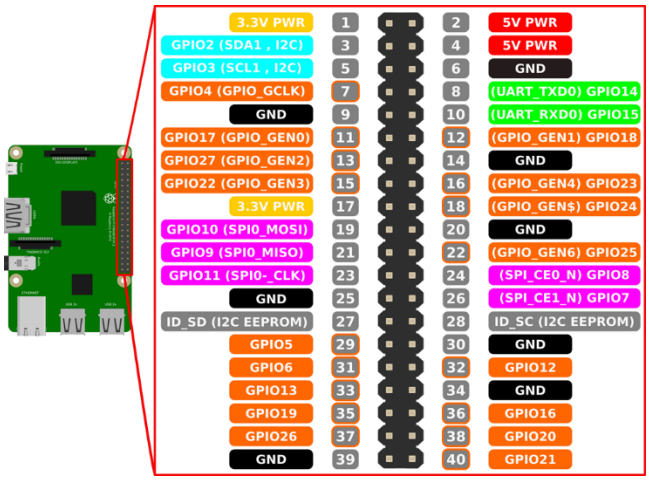


Abb. 16: GPIO Pins

Farbe	**Einsatz**
Orange	Normale Input oder Output Pins; laufen mit 3,3V (logisch 1) oder 0V (logisch 0)
Schwarz	Grounding Pins**;** 0V; für das Erden zuständig
Gelb	3,3V-Pins; können als Stromquelle dienen
Rot	5V-Pins; können als Stromquelle dienen; fliesst zu viel Strom über diese Pins, schaltet der Raspi ab
Grün	Mit dem Rx-Pin (Receiver) können Daten empfangen, mit dem Tx-Pin (Transmitter) Daten gesendet werden.
Lila, hellblau und grau	Interface Pins; dienen der Kommunikation der einzelnen Hardwarekomponenten untereinander

An einigen Stellen in der Dokumentation werden einzelne GPIO Pins mit ihren Nummern erwähnt. Dabei ist die Platznummer zwischen 1 und 40 gemeint und nicht der Name des Pins.

4.4 Ultraschallsensor

Abb. 17: Ultraschallsensor HC-SR04 von Adafruit

Der Ultraschallsensor hat vier Pins. Davon ist der Trigger Pin für das Aussenden und der Echo Pin für das Empfangen des Ultraschallsignals zuständig. Der Trigger Pin ist mit dem GPIO Pin 12 verdrahtet, der Echo Pin mit GPIO Pin 18. Beide GPIO Pins sind normale Input/Output Pins (die orangen auf der Abb. 16). Über den VCC Pin, der mit dem GPIO Pin 2 verbunden ist, wird dem Sensor mit 5V Spannung Strom geliefert. Der Grounding Pin erdet das Ganze und ist mit dem GPIO Pin 6, also einem Grounding Pin ohne Spannung verbunden.

4.5 Elektromagnet

Damit die Holzblöcke an ihren Schrauben aufgeladen und mitgeführt werden, kommt ein Elektromagnet zum Einsatz. Der Elektromagnet ist an GPIO Pin 40 angeschlossen, also an einen normalen Pin, der in diesem Fall als Output wirkt, da er Signale aussendet. Insgesamt gehen vier Drähte vom Elektromagneten aus. Der zweite Draht des Elektromagnets ist an GPIO Pin 32 angeschlossen. Dieser Pin wirkt als Input und kann abrufen, ob ein Signal auf dem Magneten vorhanden ist. Die anderen beiden Drähte (rot und schwarz) dienen dem Stromkreislauf und sind zusammen mit den Drähten der Powerbank, die dem Motor-Hat Strom liefern, beim Stromeingang des Motor-Hats eingequetscht. Der Grund dafür ist, dass der Magnet mehr Strom benötigt, als der Raspi liefern kann. Um etwas Magnetisches anzuziehen, muss Strom auf den Elektromagneten gegeben werden. Sobald der Gegenstand nicht mehr angezogen werden soll, wird die Stromzufuhr unterbrochen, was dazu führt, dass keine Spannung auf dem Magneten vorhanden ist, die den Elektromagneten magnetisch wirken lässt.

4.6 Kamera

Das Raspberry Pi Kameramodul v2 ist ein 8-Megapixel Bildsensor-Zusatzmodul für den Raspi. Die Kamera ist in der Lage, statische Bilder mit der Auflösung von 3280*2464 Pixeln zu schiessen und 1080p30, 720p60 und 640*480p90 Videos aufzunehmen. Das Flachbandkabel der Kamera wird bei der CSi-Schnittstelle, der speziellen Kameraschnittstelle des Raspberry Pi’s, angeschlossen. Für die Wegverfolgung hätte anstatt der Kamera auch einfach ein Liniensensor eingesetzt werden können. Dafür wurde aber bewusst die Kamera eingesetzt, um das Fachgebiet Computer Vision einzubeziehen und in diesem Bereich etwas zu lernen.

5 Software

5.1 Python 2

Die Programmiersprache Python erfreut sich derzeit an wachsender Popularität unter den Programmierern. Wie die Abb. 18 zeigt, gehört Python zu den beliebtesten Programmiersprachen hinter Java, C und C++. Die Bewertungen basieren nämlich auf der Anzahl der qualifizierten Informatiker weltweit, Kurse und Drittanbieter. Für die Berechnung der Bewertungen wurden gängige Suchmaschinen wie Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube und Baidu verwendet.

Programming Language	Ratings	Change
Java	16.881%	+3.92%
C	14.966%	+8.49%
C++	7.471%	+1.92%
Python	6.992%	+3.30%
Visual Basic .NET	4.762%	+2.19%
C#	3.541%	-0.65%
JavaScript	2.411%	+0.31%

Abb. 18: Ranking Programmiersprachen

Python gibt es seit 1991. Die Programmiersprache war aber noch in der Entwicklungsphase und so erschien erst 1994 die erste offizielle Version, Python 1.0. Mit fortlaufenden Erweiterungen entstand im Jahr 2000 die Version Python 2.0. Auch diese Version wurde mit der Zeit optimiert. Das letzte Update von Python 2 ist Python 2.7 und wurde 2010 veröffentlicht. Neben der Optimierung von Python 2 wurde Python 3 entwickelt. Die Version Python 3.0 wurde Ende 2008 veröffentlicht. Seitdem wird auch Python 3 weiterentwickelt und optimiert. Die Unterschiede zwischen Python 2 und Python 3 liegen mehrheitlich in

einzelnen Funktionen und Datentypen. Beispielsweise der Befehl *print* ist in Python 3 neu eine Funktion und nicht mehr ein Spezialbefehl. Des Weiteren erzeugen Funktionen wie *range()* in Python 2 statische Listen. In Python 3 werden diese Listen anders erzeugt bzw. gespeichert, was weniger Speicherplatz einnimmt. Für mein Projekt waren die Unterschiede von kleiner Bedeutung. Dennoch habe ich mich für Python 2.7 entschieden, da die Installation von OpenCV für Python 2 einfacher war und es mir nicht gelungen ist, OpenCV für Python 3 einzurichten. Und da die Version Python 2.7 die letzte von Python 2 ist, sind die Entwicklungen und Optimierungen abgeschlossen, was mir eine funktionierende und problemlose Version versicherte.
Im Wesentlichen gibt es einige Vorteile von Python gegenüber anderen Programmiersprachen, nur sehr wenige Nachteile. Die nachfolgende Tabelle zeigt die wesentlichen Vor- und Nachteile:

Vorteile	**Nachteile**
Sehr simple Sprachsyntax	2 Versionen im Einsatz; dies kann immer zu Kompatibilitätsproblemen führen
Hohe Sicherheit	Relativ langsam verglichen mit anderen Programmiersprachen
Viele Module vorhanden	
Open Source → für jeden zugängig	
Gratis Download	
Sehr vielfältig	
Grosse und aktive Community	

In Python müssen Variablen nicht einem Datentyp zugeordnet werden. Das System erkennt je nach Wert, welcher der Variable zugewiesen wird, um welchen Datentyp es sich handelt. Dies vereinfacht die Programmiersprache bereits, denn der Programmcode kann auf diese Weise gekürzt werden. Das folgende Beispiel zeigt, wie einer Variablen einen Wert zugeschrieben wird. Dabei sind Integerwerte Ganzzahlen und Floatwerte Dezimalzahlen.

```
pl = 240  #Pixellinie pl bekommt den Integerwert 240
kf = 0.18 #Korrekturfaktor kf bekommt den Floatwert 0.18
```

Code 1: Zuweisung von Werten an Variablen

Oft muss ein Wenn/Dann-Prozess gemacht werden. Das heisst, es wird beispielsweise eine Variable geprüft mittels einem Statement. Wenn dieses Statement wahr ist, wird ein Prozess gestartet. Wenn das Statement falsch ist, wird meist ein anderer oder gar kein Prozess gestartet. Um diese Entscheidungen zu treffen und Statements auf ihre Wahrheit zu prüfen, werden in der Informatik *if/else*-Statements gebraucht. Als Erstes wird das *if*-Statement geprüft. Wenn dieses falsch ist, werden die *elif*-Statements geprüft, sofern solche vorhanden sind. Nur wenn das vorherige Statement falsch ist, wird das Statement überhaupt beachtet. *Elif*-Statements funktionieren grundsätzlich gleich wie *if*-Statements, also überprüfen, ob eine Aussage stimmt. Wenn alle vorherigen Statements falsch sind, kann mit dem *else*-Befehl ein Prozess gestartet werden, welcher anders ist, als wenn ein Statement richtig wäre. Ein Beispiel für eine *if*-, *elif*- und *else*-Abfolge:

```
if bsum in range(100,600):      #wenn bsum zwischen 100 und 600 liegt
   Striche = 1                  #Variable Striche bekommt Wert 1
elif bsum in range(601,1000): #wenn bsum zwischen 601 und 1000 liegt
   Striche = 2                  #Variable Striche bekommt Wert 2
else:                           #sonst Fehlermeldung
   print("Das Foto beinhaltet zu viel oder zu wenig schwarze Pixel")
```

Code 2: if-, elif- und else-Statement

Um Prozesse mehrere Male durchlaufen zu lassen, ohne den zugehörigen Programmcode aneinanderzureihen, können Schleifen verwendet werden. Es gibt 2 Schleifenarten: *for*-Schleifen und *while*-Schleifen. Eine Schleife wird ausgeführt, wenn eine vorgegebene Bedingung erfüllt wird. *For*-Schleifen eignen sich besonders für Iterationen; das heißt, wenn beispielsweise für jedes Element einer Liste der gleiche Prozess durchlaufen werden soll. Sie können aber auch wie eine *while*-Schleife eingesetzt werden, indem eine Bedingung erfüllt werden muss, damit die Schleife ausgeführt wird. Für *for*-Schleifen gibt es den simplen Befehl *break*, mit dem die Schleife abgebrochen wird. Eine Schleife kann folgendermassen aussehen:

```
for pixel in thresh[pl,:]:
   my += 1
   if pixel == 0:
      my += 330
      break
```

Code 3: for-Schleife

Für eine bessere Strukturierung des Programms können Funktionen definiert werden. So kann mit einem Befehl gleich eine Aneinanderreihung von mehreren Befehlen ausgeführt werden. Da der Roboter immer wieder dieselben Prozesse durchlaufen muss, können viele umfangreiche Funktionen definiert werden. Dies schafft eine gute Übersicht und die Prozessabfolge kann problemlos nachvollzogen werden. Folgendes Beispiel zeigt, wie eine Funktion definiert wird:

```
def getCanny():
    olmg = takePicture()
    bflmg = cv.boxFilter(olmg, -1, (3,3))
    canny = cv.Canny(bflmg,80,180,apertureSize=3)
    return canny
```

Code 4: Definition einer Funktion

Eine Funktion kann optional auch Parameter beinhalten. Dafür wird bei *getCanny()* in der ersten Zeile der Definition der Name des Parameters zwischen die Klammern des Funktionsnamen geschrieben. Dieser Parameter kann im Aktionsteil ab Zeile 2 der Funktionsdefinition verwendet werden.

Wenn nun die Funktion im Programmcode aufgerufen wird, muss für jeden definierten Parameter ein Wert eingetragen werden.

```
def getCanny(minVal, maxVal):      #Funktion getCanny() mit gewählten Parametern wird definiert
    olmg = takePicture()
    bflmg = cv.boxFilter(olmg, -1, (3,3))
    canny = cv.Canny(bflmg,minVal,maxVal,apertureSize=3) #gewählte Parameter in Canny() eingesetzt
    return canny  #canny wird zurückgegeben
```

Code 5: Definition einer Funktion mit Parametern

```
canny = getCanny(80,180)  #minVal = 80, maxVal = 180
```

Code 6: Zuweisung eines Wertes an eine Variable mittels Aufruf einer Funktion, die Parameter beinhaltet

Funktionen müssen jedoch nicht direkt im gleichen Python-File definiert werden, wie der Programmcode. Es können auch Module importiert werden, was manchmal unumgänglich ist. Module können einzelne Funktionen und Befehle oder gar ganze Klassen mit Funktionen beinhalten und schaffen Übersicht. Sie werden am Anfang des Programms importiert, dann können die dazugehörigen Befehle verwendet werden. OpenCV beispielsweise muss bei

jedem Programm importiert werden, bei dem man OpenCV benötigt, ansonsten sind die Befehle fremd und es gibt unzählige Fehlermeldungen.

```
import time      #importiert Modul time
import RPi.GPIO as GPIO      #importiert Modul RPi.GPIO mit einfacherem Namen GPIO
import cv2 as cv
import numpy as np
from picamera.array import PiRGBArray    #importiert Funktion PiRGBArray des Moduls picamera.array
from picamera import PiCamera
import Robot
```

Code 7: Importieren von Modulen

Alle in der Dokumentation erwähnten Befehle gehen von den Modulen aus, wie sie in Code 7 importiert wurden.

5.2 Raspbian

Auf die SD Karte, das Speichermedium des Raspberry Pi‘s, wurde nach dem Formatieren der Download-Manager Noobs installiert, mit dem man ganz einfach ein Betriebssystem installieren kann. Noobs kann ganz einfach und komplett gratis von der Webseite von Raspberry Pi heruntergeladen werden. Mittels Noobs kann aus einigen vorbereiteten Betriebssystemen ausgewählt werden. Ein extra für den Raspi entwickeltes Betriebssystem ist Raspbian. Es handelt sich dabei um ein Linux Betriebssystem, das auf Debian basiert. Aus Gründen der einfacheren Handhabung sind Python und Entwicklungsumgebungen bei Raspbian bereits vorinstalliert, was das Einrichten des Raspberry Pi's für Anfänger erleichtert. Generell war die Absicht der Raspberry Pi Foundation, mit diesem Betriebssystem das Programmieren für ein möglichst großes Publikum schmackhaft zu machen und viele Leute zum Programmieren zu animieren. Für die Navigation auf Raspbian gibt es zwei Möglichkeiten. Es ist eine grafische Desktopumgebung vorhanden. Am Anfang und besonders für Linux Neulinge ist das Navigieren auf der grafischen Desktopumgebung wohl einfacher. Sie ist ähnlich wie Windows aufgebaut: Zuerst taucht ein Sperrbildschirm für das Login auf. Danach wird ein Startbildschirm mit Taskleiste usw. angezeigt. Ein Dokument beispielsweise kann aber auch über das Terminal, eine Konsolenumgebung, geöffnet oder bearbeitet werden. Viele Aktionen können über das Terminal schneller und einfacher ausgeführt werden. Ein kleiner Nachteil mit der Arbeit via Terminal sind die Kommandozeilen, welche für die Zurechtfindung und Navigation erlernt werden müssen.

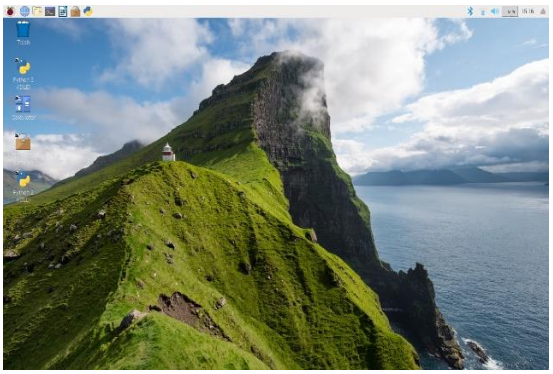


Abb. 19: Desktop

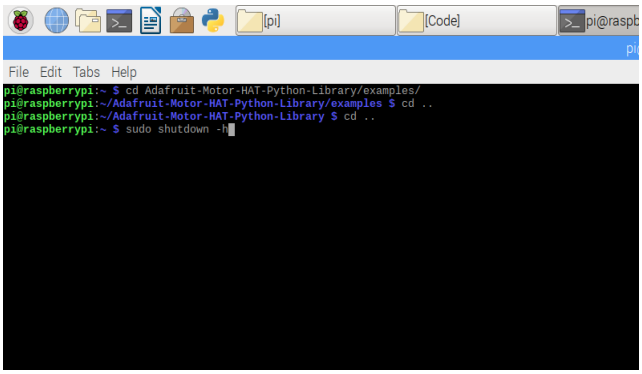


Abb. 20: Terminal

5.3 IDLE

Um zu programmieren braucht man auch eine Entwicklungsumgebung, auch IDE genannt (Integrated Development Environment). In diesem Fall heisst diese Software IDLE. Es handelt sich dabei um die Standartumgebung von Python. Sie wurde auch in Python programmiert und funktioniert für Windows, Unix und Mac OS X. In IDLE gibt es neben dem Editorfenster auch ein Shellfenster, wo Befehle ausprobiert, aber nicht fest abgespeichert werden können. Um einen Algorithmus zu implementieren, der später vom Roboter ausgeführt werden soll, muss im Editorfenster programmiert werden. Das Shellfenster dient dabei eher dazu, einen Befehl beispielsweise auf die Korrektheit der Syntax zu prüfen. IDLE ist auf Raspbian schon standardmässig installiert, weshalb man sich nach der Installation von Raspbian nicht mehr um die IDE kümmern muss, sofern diese einem zufriedenstellt.

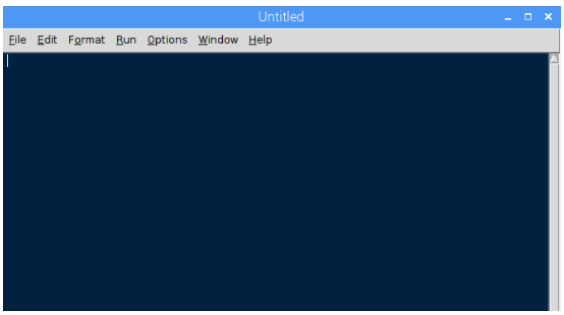


Abb. 21: Editorfenster

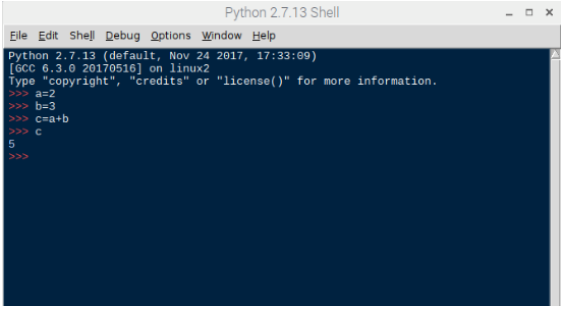


Abb. 22: Shellfenster

5.4 OpenCV

Um ein Bild möglichst schnell bearbeiten zu können, habe ich mich dazu entschieden, mit OpenCV zu arbeiten. Es handelt sich dabei um eine Computer Vision Bibliothek, die gratis heruntergeladen werden kann. Die Bibliothek ist open Source, das heißt, sie ist für jeden frei zugänglich und jeder kann darauf zugreifen. Mit Computer Vision wird die Bildverarbeitung

bezeichnet. Mit der Benützung von OpenCV können Prozesse mit einfachen Algorithmen implementiert werden, die ohne diese Bibliothek sehr komplex zu programmieren wären und nicht im Rahmen einer Maturaarbeit liegen würden. Für das bessere Verständnis von OpenCV habe ich nach dem Download erstmals einfache Programme geschrieben, die beispielsweise ein bereits gespeichertes Bild einlesen und in Form eines Graustufenbildes ausgeben. In dieser Experimentierphase wurden Filter ausgetestet und verschiedene Operationen an Pixeln ausprobiert. Auf einzelne Pixel kann mit OpenCV schnell zugegriffen werden, was eine wichtige Rolle bei der Bilderkennung spielt, um beispielsweise die schwarze Weglinie zu finden.

Einige häufige OpenCV Befehle:

Befehl	**Aktion**
cv.imread(„bild.jpg“, bild)	Die Bilddatei „bild.jpg“ wird eingelesen
cv.imwrite(Zielort, img)	Das Bild img wird am Zielort abgespeichert
cv.cvtColor(img, cv.COLOR_BGR2GRAY)	Das Bild img wird in ein Graustufenbild umgewandelt
cv.boxFilter(img, -1, (3,3))	Das Bild img wird mit einem Boxfilter der Grösse 3*3 und der Bildtiefe -1 (Standartwert) gefiltert.
cv.threshold(img, 100, 255, cv.THRESH_BINARY)	Das Bild img wird in ein Threshold-Bild mit Schwellwert 100 umgewandelt
cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_NONE)	Im Threshold-Bild thresh werden Konturen gesucht und mit zugehöriger Hierarchie zurückgegeben

5.5 Putty

Wie kann man auf den Raspi zugreifen, ohne ihn an einem Monitor und einer Tastatur anschliessen zu müssen? Das war eine bedeutende Frage in Bezug auf meinen Roboter, denn er muss sich schliesslich frei im Raum bewegen können. Trotzdem muss das Programm gestartet werden können, damit der Roboter seine Arbeit beginnen kann. Die Lösung dafür ist Secure Shell (SSH). Es handelt sich dabei um eine Fernsteuerung, die von einem Computer aus betrieben werden kann. Dabei muss sich der Raspi und der Computer im gleichen Netzwerk befinden.

Mithilfe der Software Putty gelingt es ganz einfach, mit dem Computer auf den Raspberry Pi

zuzugreifen. Die Software muss auf einem Computer heruntergeladen werden, in meinem Fall ist es ein Laptop, der mit Windows läuft. Ausserdem muss SSH auf dem Raspi erlaubt werden. Dies muss auf den Einstellungen des Raspberry Pi's vorgenommen werden. Erst nach dieser Einstellung ist einen SSH-Verbindung mit dem Raspi möglich. Dafür wird Putty gestartet und beim obersten Eingabefeld der Host Name eingetragen oder die IP Adresse vom Raspberry Pi. Nun kann bereits mit ‚open' die SSH-Verbindung aufgebaut werden. Wie immer muss man sich zuerst einloggen, damit man den vollen Fernzugriff auf den Raspi hat und ihn über den Computer verwalten kann.

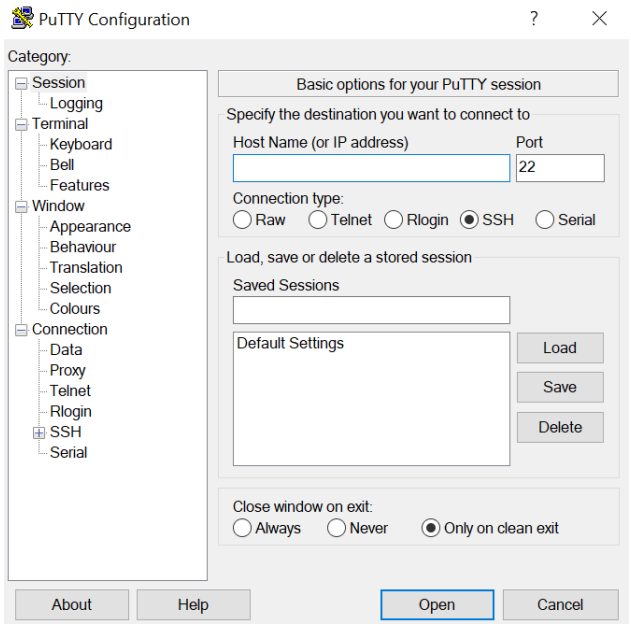


Abb. 23: Putty

6 Vorgehensweise

Nach dem Festlegen meines Themas konnte ich zusammen mit meinem Betreuer schon bald den Projektvertrag abschließen. Darauf musste möglichst schnell das notwendige Material ausgewählt und besorgt werden, um möglichst speditiv und umgehend nach Bewilligung des Projektes starten zu können. Als Erstes stand die Wahl eines Chassis an. Hierfür habe ich mich mehrheitlich auf playzone.ch umgesehen. Schnell konnten meine Möglichkeiten eingeschränkt werden, da ein dreirädriges Gefährt ausreichend für mein Projekt schien. Somit musste ich nur zwei Motoren ansteuern, anstatt vier, was das einfache Wenden an Ort und Stelle ermöglicht und die Steuerung durchaus erleichtern sollte. Aus den wenigen dreirädrigen Bausätzen habe ich mich dann für dasjenige entschieden, welches sich von der Form besser eignete, um das Gefährt weiter auszubauen.
Bei derselben Bestellung war das Motor-Hat und eine SD Karte für den Raspberry Pi dabei. Mit dem Kauf eines Raspberry Pi bei Digitec.ch waren die wichtigsten Komponenten griffbereit. Den Strom für den Raspi lieferte zunächst ein Micro USB Kabel.
Das Chassis war in kurzer Zeit zusammengebaut. Lediglich ein Schraubenzieher war nötig für den Zusammenbau. Die zwei großen Räder konnten ganz simpel auf die Motoren gesteckt werden. Das hintere kleine Rad wurde mit Schrauben befestigt. Die Drähte der Motoren konnten dann bereits beim nächsten Schritt angelötet werden.

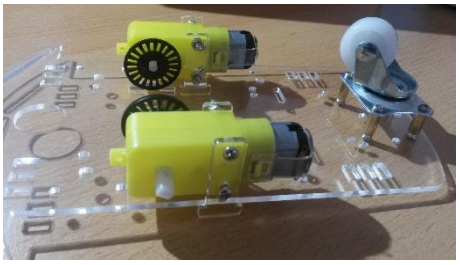


Abb. 24: Chassis von unten ohne Steckräder

Abb. 25: Chassis zusammengebaut von oben

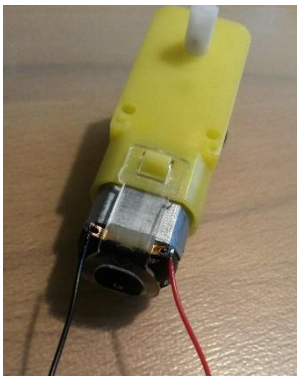


Abb. 26: DC Motor mit angelöteten Drähten

Besonders in der Anfangsphase dieser Maturaarbeit war die Arbeit oft mehrspurig. Neben dem Bau und der Hardware des Roboters musste auch die Software installiert werden. Der Raspberry Pi brauchte zuerst ein Betriebssystem auf der SD Karte, damit er überhaupt angeschaltet werden konnte.

Damit sich der Roboter später frei im Raum bewegen konnte, war eine mobile Stromquelle Pflicht. Um nicht etliche Batterien zu leeren wurde entschieden, eine Powerbank als Stromquelle einzusetzen. Einer der beiden 5V-Ausgang liefert Strom mit 2.4A an den Raspi und somit auch an den Distanzsensor. Der andere 5V-Ausgang mit 1A versorgt das Motor-Hat und den Elektromagnet mit Strom. Als Befestigung der Powerbank dient ein Klettstreifen. Die Powerbank kann somit praktisch auf die Kunststoffplatte gesetzt und weggenommen werden.

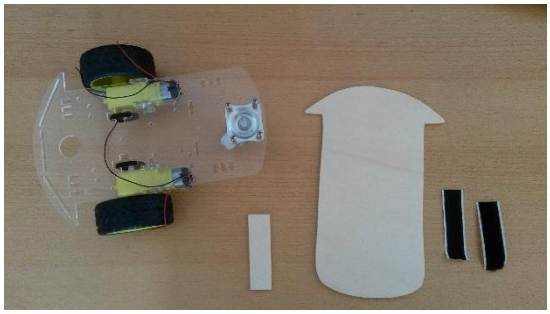


Abb. 27: Chassis mit ersten Erweiterungskomponenten

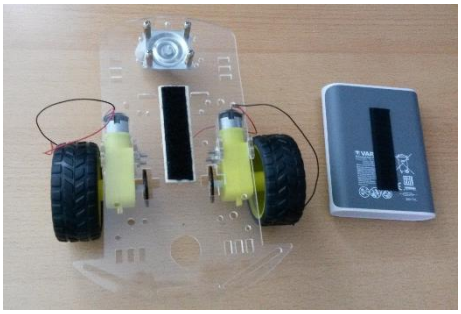


Abb. 28: Chassis und Powerbank mit Klettstreifen

Das Chassis wurde mit einer Holzplatte erweitert, die auf der Powerbank für die Befestigung von Komponenten eine weitere Fläche bietet. Das Motor-Hat wurde auf den Raspi gesteckt und vorerst nur auf die Holzplatte gelegt und noch nicht festgeschraubt. Die Drähte der Motoren konnten anschliessend am Motor-Hat angeschlossen werden. In diesem Zustand war der Roboter bereits fahrfähig. Nur noch eine Softwarekomponente von Adafruit für das Motor-Hat musste installiert werden, sodass die Motoren getestet werden konnten. Dieser Test hat gezeigt, dass die Motoren entgegengesetzt laufen. Nachdem also die Drähte des einen Motors umgekehrt angeschlossen wurden, funktionierten beide Motoren wunschgemäss.

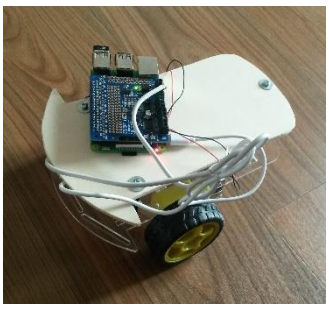


Abb. 29: Erster Prototyp des Roboters

Der Raspi mit dem Motor-Hat wurde zentral auf die Holzplatte geschraubt. Danach musste OpenCV installiert werden. Dies erforderte mehrere Versuche, denn es konnte rasch ein Fehler passieren bei dieser Download-Menge von Ordnern und Files. Auch diese Installation wurde getestet. Dafür wurden *.jpg* Bilder mit OpenCV eingelesen und ausgegeben.

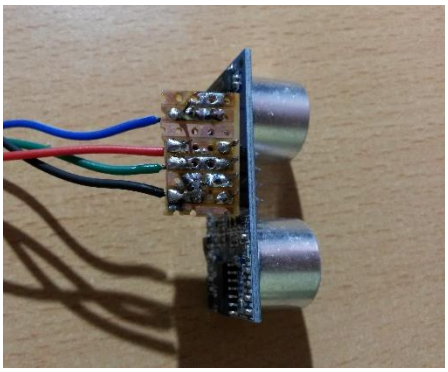


Abb. 30: Ultraschallsensor an Breadboard gelötet

Einmal mehr musste der Lötkolben erhitzt werden, um einige Drähte an den Distanzsensor zu löten. Die Drähte konnten nicht direkt an den Sensor gelötet werden. Es bedurfte eines Breadboards, eines Kunststoffplättchens mit Löchern, welche durch Kupferreihen horizontal verbunden sind. Der Grund für das Breadboard ist folgender: Der Sensor wird mit 5V gespeist und der Echo Pin wird auch auf 5V gesetzt, wenn das Signal empfangen wird. Der Raspi aber ist für 3.3 V an den GPIO Pins ausgelegt und deswegen wird mit zwei Wiederständen ein Spannungsteiler erstellt. Dadurch kann das Echo Signal auf einen tolerierbaren Spannungswert vermindert werden. Hier wurden nun die Drähte und Widerstände gemäss Schaltplan angelötet. Der Sensor passte mit seinen Metall-Anschlüssen problemlos in die Löcher und konnte mit Lötzinn befestigt werden. Somit mussten später nur noch die Drahtenden an die richtigen GPIO Pins gelötet werden. Zuerst musste aber eine Halterung dafür entworfen und erstellt werden. Dafür habe ich zusammen mit meinem Onkel und meinem Cousin einen Entwurf einer möglichen Halterung erstellt. Dasselbe machten wir auch für die Kamera. So konnten die Pläne der beiden Halterungen im 3D-Drucker eingegeben und gedruckt werden. Der Plan der Kamerahalterung war erst ein Prototyp, da ausgetestet werden musste, ob es überhaupt möglich war, eine solche Halterung herzustellen. Es war nämlich nicht sicher, wie gut die beiden Teile zueinander passten und aneinander befestigt werden konnten. Nachdem der Druck erfolgreich war und die beiden Teile gut aneinandergesteckt werden konnten, wurde die Kamerahalterung mit einigen Optimierungen an den anderen Löchern für die Befestigung am Roboter nochmal gedruckt.

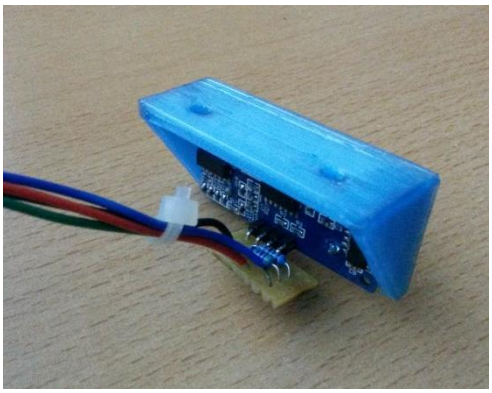


Abb. 31: Ultraschallsensor in der Halterung

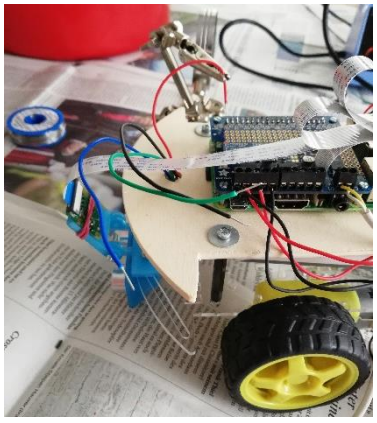


Abb. 32: Anlöten der Drähte des Sensors

Die beiden Halterungen für den Sensor und die Kamera wurden dann am selben Ort an der Kunststoffplatte mit Schrauben und Muttern befestigt. Nach Anlöten der Drähte des Sensors an die Pins folgten weitere Tests mit dem Distanzsensor. Dafür wurde ein einfaches Programm zur Messung und Ausgabe der Distanz zum Sensor geschrieben. Es stellte sich heraus, dass der Sensor ausserordentlich genau mass und er problemlos funktionierte. Die Kamera konnte an ihrer Halterung mit lediglich einem Gummiband befestigt werden. Zunächst verkürzte ich auch das lange Kameraband mit einem kleinen Gummiband. Nach einigen Fotoaufnahmen fiel allerdings auf, dass das Kameraband nicht so kompakt sein durfte, weil es sonst zu Bildrauschen kam. Meine Lösung waren dicke Klebstreifen, die an das Kameraband geklebt wurden. Dies ermöglichte eine kompakte Montage des Kamerabands ohne Bildrauschen und Störungen bei Fotoaufnahmen.

Abb. 33: Kamerabild mit Gummiband zur Verkürzung des Kamerabands

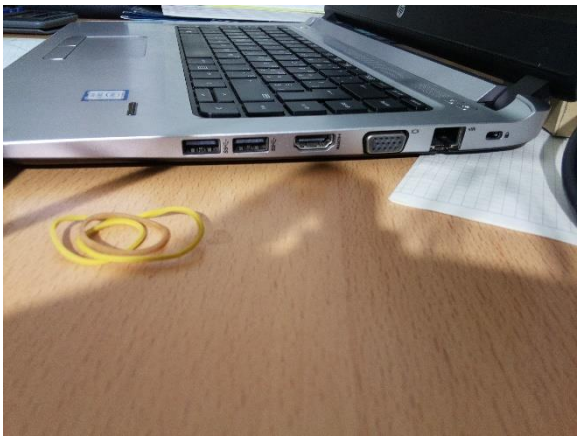


Abb. 34: Kamerabild mit Klebstreifen zur Verkürzung des Kamerabands

Bereits einige einfache Versionen konnten nun schon programmiert werden. Zuerst musste man sich darauf fokussieren, dass der Roboter die schwarze Weglinie erkannte und dieser entlangfahren konnte. Vorläufig war die Weglinie bloss eine einfache Linie ohne Kurve oder Abzweigung. Trotzdem musste man auch dafür schon viel Zeit investieren, damit der Korrekturfaktor und die Geschwindigkeit des Roboters stimmten, um sich auf der Linie zu halten und am Ende wieder zu drehen. Neben dem Optimieren der Parameter im Programmcode konnte bereits eine erste Kofferreihe hergestellt werden. Bei einer Holzleiste kleine Quader abgeschnitten, Strichcodes ausgedruckt und aufgeklebt; und schon waren die ersten Modellkoffer gemacht.

Abb. 35: Holzblock

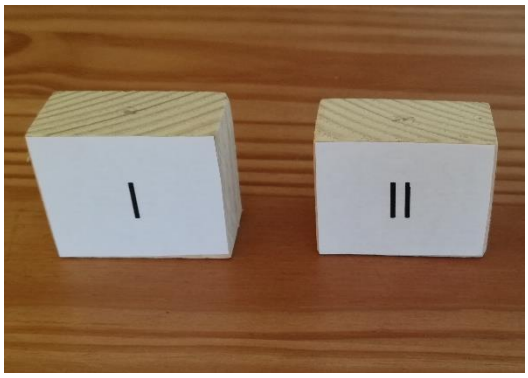


Abb. 36: Holzblöcke mit Strichcodes

Bei weiteren Codeexperimenten wurde getestet, wie gut die Strichcodes erkannt werden. Verschiedene Methoden wurden ausprobiert und verglichen. Wie schon oben erwähnt war die Arbeit oft mehrspurig. So musste sowohl das Fahren optimiert und auf Abzweigungen erweitert werden, als auch die Unterlage, die Weglinie und der Roboter erweitert und optimiert werden. Ein Elektromagnet wurde bestellt. Für den Einbau des Magnets konnte aus Holz eine Halterung hergestellt werden. Ein letztes Mal wurde dann der Lötkolben gebraucht, um die Drähte des Magnets an GPIO Pins zu löten. Nach dieser Montage war der Roboter fertig gebaut.

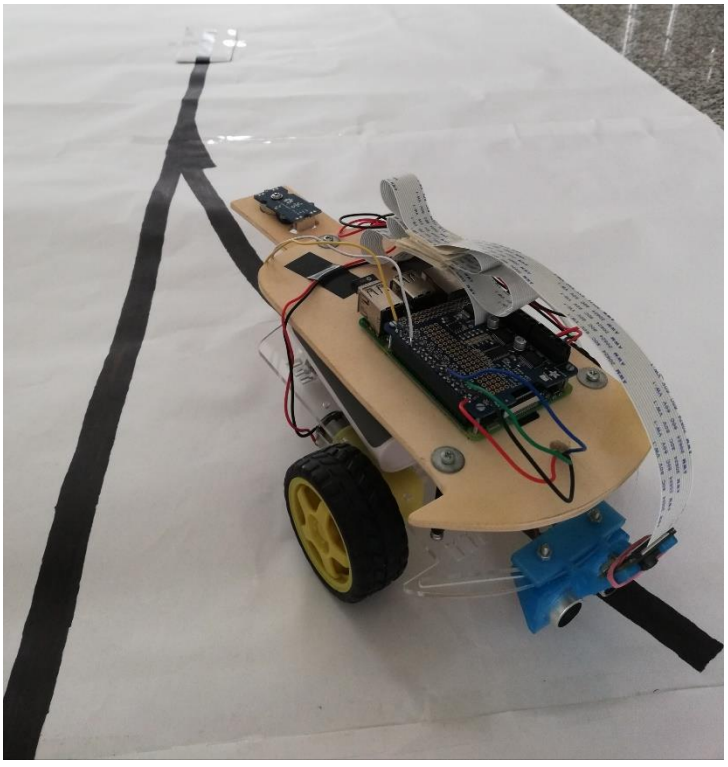


Abb. 37: Fertig gebauter Roboter

7 Implementierung

7.1 Motoren ansteuern

Mit den Python-Files von Adafruit zum Testen der Motoren funktionierten die DC Motoren einwandfrei. Es fiel auf, dass in diesen Files immer eine Klasse namens *Robot* importiert wird. Diese Klasse ist ebenfalls im selben Ordner als Python-File zu finden. Sie beinhaltet die Funktionen und Befehle, um die Motoren anzusteuern, war gut verständlich und wurde deshalb auch in meinen Versionen verwendet. Es muss allerdings erwähnt werden, dass die Klasse noch umgeschrieben und angepasst werden musste, da der Roboter in meinem Projekt streng genommen immer rückwärtsfährt. Um später Verwirrungen im Programmcode zu verhindern, musste also in der Klasse *Robot forward()* und *backward()* vertauscht werden. Folgendes Beispiel zeigt, wie so eine Funktion der Klasse Robot definiert ist.

```
def forward(self, speed, seconds=None): #Funktion forward() mit Parametern speed und evt. seconds
    self._left_speed(speed)    #Geschwindigkeit bekommt Wert von speed, muss zwischen 0-255 liegen
    self._right_speed(speed)
    self._left.run(Adafruit_MotorHAT.BACKWARD) #Drehrichtung beider Motoren wird festgelegt
    self._right.run(Adafruit_MotorHAT.BACKWARD)
    # Falls für seconds eine Zahl eingegeben wird, fahre so lange und stoppe danach
    if seconds is not None:
        time.sleep(seconds)
        self.stop()
```

Code 8: Definition der Steuerungsfunktion forward() in der Klasse Robot

Am Beispiel von *forward()* erkennt man, dass neben den Parametern auch *self* vorkommen muss. *Self* steht für das Objekt, das in meinen Versionen *robot* heisst und den Roboter repräsentiert. Die Funktion *forward()* setzt zuerst die Geschwindigkeit. Beide Motoren bekommen in diesem Fall die gleiche Geschwindigkeit. Sie hat den Wert, der für *speed* gewählt wird. *Speed* steht für die Anzahl Umdrehungen pro Minute. Dann wird die Drehrichtung für beide Motoren festgelegt. Für *forward()* ist die Drehrichtung beider Motoren *BACKWARD*, da ich den Roboter, wie schon oben erklärt, umgekehrt fahrend verwende, als dies die Klasse *Robot* vorsieht. Der Parameter *seconds* ist optional. Der Prozess wird ohne Zeitbegrenzung ausgeführt, falls für *seconds* kein Wert eingegeben wird. Wenn *seconds* dagegen einen Wert zugeschrieben bekommt, läuft der Prozess – in diesem Fall *forward()* – so viele Sekunden, wie für *seconds* gewählt wurde und stoppt anschliessend. Neben *forward()* enthält die Klasse *Robot* noch einige weitere Funktionen zur Steuerung des Roboters.

Befehl	**Erklärung**
forward(speed, opt. seconds)	Fährt vorwärts
backward(speed, opt. seconds)	Fährt rückwärts
right(speed, opt. seconds)	Dreht nach rechts (Uhrzeigersinn)
left(speed, opt. seconds)	Dreht nach links (Gegenuhrzeigersinn)
more_to_right(speed, trim, opt. seconds)	Fährt vorwärts und ein wenig nach rechts
more_to_left(speed, trim, opt. seconds)	Fährt vorwärts und ein wenig nach links

Bei den Funktionen *more_to_right()* und *more_to_left()* kommt ein Parameter dazu; *trim*. Dieser bestimmt, wie stark der Roboter eine Kurve nach rechts bzw. nach links fährt. Die beiden Funktionen werden im Programmcode eingesetzt, damit der Roboter bei Abweichungen von der Weglinie korrigieren kann und gleichzeitig vorwärtskommt. Der Wert von *trim* beeinflusst die Geschwindigkeit der einzelnen Motoren.

```
def more_to_right(self, speed, trim, seconds=None):
    self._left_speed(speed-(trim*3/4))
    self._right_speed(speed+(trim/4))
    self._left.run(Adafruit_MotorHAT.BACKWARD)
    self._right.run(Adafruit_MotorHAT.BACKWARD)

    if seconds is not None:
        time.sleep(seconds)
        self.stop()
```

Code 9: Definition der Steuerungsfunktion more_to_right() in der Klasse Robot

In der Klasse *Robot* werden auch Befehle verwendet, die von anderen Klassen importiert sind. Um die Drehrichtung der Motoren festzulegen, werden z.B. Befehle der Klasse *Adafruit_MotorHAT* verwendet.

7.2 GPIO Pins aktivieren/deaktivieren

Auf Raspbian ist die GPIO Bibliothek bereits vorinstalliert. Um die GPIO Pins ansteuern zu können, muss im Programmcode das entsprechende Modul importiert werden. Das Modul heisst *RPi.GPIO* und wird im Programmcode als *GPIO* importiert.

```
import RPi.GPIO as GPIO
```

Code 10: Importieren des GPIO Moduls

Des Weiteren werden den einzelnen Pins bzw. ihren Nummern Namen zugeteilt, um im späteren Verlauf des Programmcodes einen einzelnen Pin aufrufen zu können und nicht seine Nummer merken zu müssen. Logischerweise müssen die Nummern mit der Hardware übereinstimmen. Eine Namenszuweisung ist jedoch nicht zwingend. Die Pins können auch direkt mit ihren Nummern aufgerufen werden. Der Trigger Pin des Ultraschallsensors kann beispielsweise nicht eine andere Nummer als 18 bekommen, wenn er mit dem GPIO Pin 18 verlötet ist.

```
#Pin Nummer Modus (1-40)
GPIO.setmode(GPIO.BCM)

#Sensor Pins zuweisen
GPIO_TRIGGER = 18
GPIO_ECHO = 24
GPIO_MAGNET = 21

#festlegen, ob Input - oder Output Pin
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
GPIO.setup(GPIO_MAGNET, GPIO.OUT)
```

Code 11: Initialisierung der verwendeten Pins

Nun können die Pins an beliebigen Orten im Programmcode ein- oder ausgeschaltet werden. Am Beispiel des Magnet-Testprogramms wird ersichtlich, wie so ein einzelner Pin gesteuert werden kann. In diesem Fall lautet der Befehl zum Einschalten des Pins *GPIO.output(GPIO_MAGNET, True)*. Um den Pin auszuschalten, wird *False* anstelle von *True* gesetzt.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO_MAGNET = 21

GPIO.setup(GPIO_MAGNET, GPIO.OUT)

try:
    while True:
        time.sleep(4.0)
        GPIO.output(GPIO_MAGNET, True)
        time.sleep(5.0)
        GPIO.output(GPIO_MAGNET, False)

except KeyboardInterrupt:
    print("Messung vom User gestoppt")
    GPIO.cleanup()
```

Code 12: Steuerung eines Pins

7.3 Weglinie zentrieren

Eine grundlegende Aufgabe, damit der Roboter der schwarzen Weglinie folgen kann, ist die Erkennung dieser Weglinie. In diesem Fall werden zwei Koordinaten verwendet, wobei nur die eine stetig verändert wird und die andere konstant bleibt. Letztere Koordinate ist die Anzahl Pixel der halben Bildbreite und beträgt 340 Pixel. Die andere Koordinate steht für die Mitte der Weglinie bezüglich des Bildursprungs, der in der linken oberen Ecke des Bildes liegt, und muss berechnet werden. Dafür wird das aktuellste Bild verwendet. Zunächst wird es mit einigen Filtern und Operationen umgewandelt. Für die Erkennung der Weglinie wird dann ein Canny-Bild analysiert. Anfänglich war anstelle des Canny-Bildes ein Threshold-Bild der Ausgangspunkt für die Bildanalyse. Die Bildanalyse mit dem Threshold-Bild funktioniert grundsätzlich gleich wie beim Canny-Bild. Im Threshold-Bild sind die Kantenpixel der Weglinie jedoch schwarz und im Canny-Bild weiss. Daher wird hier die Bildanalyse mit Canny erklärt, da diese auch in der Schlussversion verwendet wurde. Beim Bild, welches analysiert werden soll, wird je nach Fahrtrichtung und Abzweigungsziel die linke oder die rechte Kante der Weglinie gesucht. Dafür wird eine ausgewählte Pixellinie (Variable *pl*) iteriert und bei dieser Iteration das erste weisse Pixel gesucht. Wenn der Roboter der linken Kante der schwarzen Weglinie folgen soll, muss die linke Kante und somit im Canny-Bild das erste weisse Pixel der Iterationszeile gefunden werden. Die Iterationsrichtung ist rechts, die Pixel werden also einzeln von links nach rechts ausgelesen. Bei jedem Pixel wird eins zur Koordinatenvariablen addiert und geprüft, ob das Pixel weiss ist, also den Wert 255 hat. Falls das Pixel den Wert 0 hat und somit schwarz ist, wird die Iteration fortgesetzt und der gleiche Prozess am nächsten Pixel vorgenommen. Erst wenn ein weisses Pixel gefunden wird, bricht die Iteration ab.

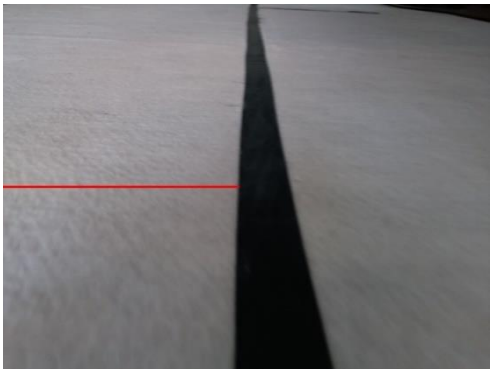


Abb. 38: Kantensuche links visualisiert

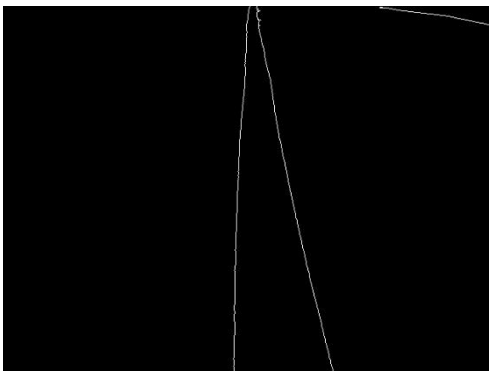


Abb. 39: Canny-Bild der Bildanalyse

Dadurch, dass jedes Pixel mittels der Koordinatenvariabel gezählt wurde, weiss man nach der Iteration, wo bezüglich des Bildursprungs die Kante liegt. Damit sich jedoch der Roboter

nicht nach der Kante der Weglinie ausrichtet, sondern nach der Mitte, muss zur Koordinate der Kante noch die halbe Wegbreite addiert werden. Die Wegbreite bei der Iterationszeile wurde durch Abmessen am Bildschirm und einfache Rechnungen berechnet und beträgt 90 +- 5 Pixel. Die Wegmitte wird also ermittelt, indem die zu verfolgende Kante durch Iteration gefunden wird und 40 zur Kantenkoordinate addiert, falls die linke Kante verfolgt wird, oder subtrahiert wird, falls die rechte Kante verfolgt wird. Die 40 Pixel entsprechen ungefähr der Hälfte der Weglinie, wenn diese sich in der Mitte des Bildes befindet. Somit ist die Wegmitte, dessen x-Koordinate in den Programmen als *my* bezeichnet wird, berechnet.

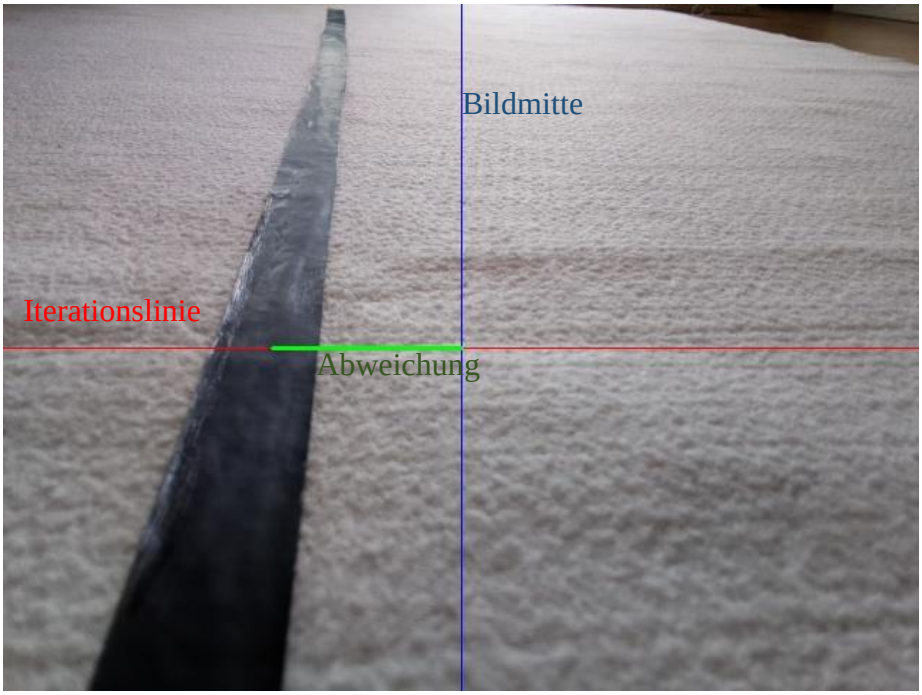


Abb. 40: Abweichungsberechnung visualisiert

Die Abweichung wird folgendermassen berechnet:

$$Abweichung = 320 - my$$

Der Parameter *trim* wird berechnet, indem die Abweichung anschliessend mit dem Korrekturfaktor (Variable *kf*) multipliziert wird. Somit verändert sich *trim* nach jedem Fahrprozess, muss angepasst und während dem nächsten Fahrprozess für die Korrektur eingesetzt werden.

7.4 Abstandsmessung

Um die Distanz zu messen, wird vom Trigger Pin ausgehend ein Signal, der Ultraschall, ausgesendet. Das Signal prallt am Gegenstand ab und wird zurückgeworfen. Der Echo Pin merkt, wenn das Signal zurück ist. Erst wenn das Signal zurück beim Echo Pin ist, hört der

Trigger Pin auf, Signal zu senden. Nun wird nicht, wie ich anfänglich angenommen habe, die Differenz zwischen Absendemoment und Empfangsmoment berechnet, sondern die Länge des Signals. Diese wird mit der Schallgeschwindigkeit multipliziert und anschliessend durch zwei dividiert, was zur Distanz zum nächsten Gegenstand frontal führt. Für die Distanzmessung wurde eine Funktion erstellt.

```
def distanz():
    GPIO.output(GPIO_TRIGGER, True) #Signal senden
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False) #Signal nicht mehr senden
    StartZeit = time.time()
    StopZeit = time.time()
    while GPIO.input(GPIO_ECHO) == 0:
        StartZeit = time.time()
    while GPIO.input(GPIO_ECHO) == 1: #Ultraschall zurück
        StopZeit = time.time()
    TimeElapsed = StopZeit - StartZeit
    distanz = (TimeElapsed * 34300) / 2
    return distanz
```

Code 13: Definition der Funktion distanz(), welche die Distanz misst

7.5 Stricherkennung

Als erste Methode zur Strichcodeerkennung wurde Image Thresholding eingesetzt. Für die Umwandlung des Bildes wird ein Bild der Koffer aufgenommen, in ein Graustufenbild umgewandelt, mittels Boxfilter gefiltert und schliesslich in ein Threshold-Bild umgewandelt. Nach diesem Vorgang wurde die Summe aller schwarzen Pixel ermittelt. So konnte je nach Menge schwarzer Pixel festgestellt werden, wie viele Striche sich auf dem Koffer befinden. Die Bereiche der Anzahl schwarzen Pixel, die für die einzelnen Anzahlen von Strichen zutreffend waren, mussten vor der Implementierung separat berechnet werden. Diese Methode war zwar einfach zu implementieren, hatte aber den großen Nachteil, dass der Abstand zum Koffer nur kleine Abweichungen haben durfte, damit die Anzahl der schwarzen Pixel noch im Bereich zur korrekten Erkennung der Anzahl Striche lag. Ausserdem musste immer die Anzahl schwarzer Pixel ermittelt werden, bevor man die Erkennung implementieren konnte. Dies war sehr mühsam bei Veränderungen des Strichcodes oder des Abstands zum Koffer.

Eine bessere Methode war die Konturenerkennung. Dazu wird ebenfalls ein Threshold- oder Canny- Bild benötigt. Eine Kontur lässt sich als eine Kurve erklären, die alle nebeneinanderliegenden Punkte mit gleicher Farbe oder Intensität verbindet. Um den Bereich einzuschränken, der auf Striche untersucht werden soll, wurde um jeden Strichcode ein schwarzes Viereck gezeichnet. Was bei der Funktion *cv.findContours()* zurückgegeben wird, sind nicht nur die Konturen selbst, sondern auch eine Hierarchie. Diese beinhaltet Informationen, welche Kontur innerhalb welcher anderen Kontur liegt. Das vereinfachte die Stricherkennung sehr, da nämlich die Striche Konturen innerhalb des schwarzen Vierecks sind. Für die Implementierung bedeutet das, dass bloss geprüft werden muss, ob es Konturen gibt im Bild, welche Unterkonturen haben, also in der Hierarchie tiefer gestellte Konturen. Wenn eine Kontur gefunden wird, die diese Bedingung erfüllt, kann die Anzahl Unterkonturen ermittelt werden. Somit ist die Anzahl Striche innerhalb des schwarzen Vierecks gefunden. Der grosse Vorteil dieser Methode ist, dass sie sehr tolerant ist bezüglich des Kofferabstands und sie somit eine kleine Fehlerquote hat. Zudem kann der Algorithmus schnell und einfach angepasst werden bei einer grösseren Anzahl Striche. Anstatt Striche können jegliche andere Zeichen eingesetzt werden. Beispielsweise können Wörter mit verschiedener Anzahl Buchstaben eingesetzt werden und die Koffern können durch die Anzahl Buchstaben erkannt und voneinander unterschieden werden.

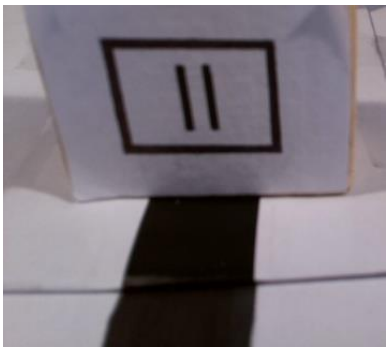


Abb. 41: Bild eines Strichcodes

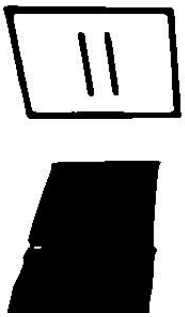


Abb. 42: Threshold-Bild des Strichcodes

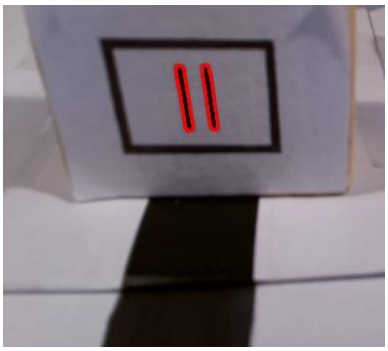


Abb. 43: Visualisierung der Stricherkennung

7.6 Thresholding

Um ein Threshold-Bild zu generieren, wird ein Graustufenbild benötigt. Im Image Thresholding ist das Graustufenbild der Input und ein Threshold-Bild der Output. Thresholding bewirkt, dass alle Pixel mit dem Wert oberhalb eines ausgewählten Schwellwertes, des Thresholds, weiss und alle Pixel mit einem tieferen Wert schwarz gemacht werden. Die folgenden Abbildungen zeigen die Wirkung von Threshold.

Abb. 44: Beispielbild eines Weisskopfseeadlers

Abb. 45: schwarzweiss-Bild des Weisskopfseeadlers

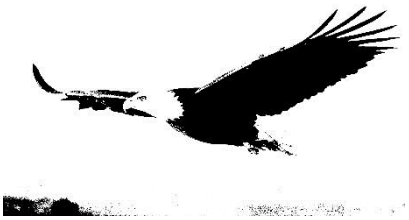


Abb. 46: Threshold-Bild des Weisskopfseeadlers mit Schwellwert = 110

7.7 Canny Edge Detection

Abb. 47: Beispielbild eines Tigers

Abb. 48: Canny-Bild des Tigers mit minVal = 80 und maxVal = 180

Der Canny-Algorithmus wurde 1986 von John F. Canny entwickelt. Canny Edge Detection wird auch schlaue Kantenerkennung genannt. Somit hat der Name des Algorithmus neben dem Erfinder eine zweite zutreffende Bedeutung. Der erste Schritt im Algorithmus ist das Unterdrücken von Rauschen im Bild, da die Kantenerkennung darauf anfällig ist. Als Filter wird ein 5*5 Gaussfilter eingesetzt. Danach wird eine erste Ableitung in horizontaler Richtung und eine in vertikaler Richtung gemacht und die Gradienten-Richtung ermittelt. Diese steht immer senkrecht zu den Kanten. Der nächste Schritt ist ein vollständiger Scan des Bildes. Bei jedem Pixel wird untersucht, ob es sich dabei um ein lokales Maximum handelt oder ob es nicht mehr beachtet werden muss. Dafür wird die Intensitätsänderung zum Nachbarpixel in Gradienten-Richtung mittels Ableitung berechnet. Ist ein Pixel ein lokales Maximum, liegt die grösste Intensitätsänderung der Umgebung vor. In diesem Fall wird das Pixel auf 1 gestellt, ansonsten auf 0. Damit schlussendlich zusammenhängende Kanten

zurückgegeben werden, müssen in diesem Schritt die Intensitätsgradienten untersucht werden. Alle Intensitätsgradienten mit einem Wert über *maxVal* sind sicher Kanten, alle unter *minVal* sind sicher keine Kanten. Bei den Werten dazwischen ist es abhängig davon, ob das Pixel ein Nachbarpixel von einem „sicher Kante"-Pixel ist. In diesem Fall wird das Pixel auch als Kante betrachtet. Ansonsten gehört es nicht zur Kante.

In der Abbildung ist erkennbar, dass neben Pixel A auch Pixel C zur Kante gehört, da es mit Pixel A verbunden ist. Da Pixel B nicht mit Pixel A verbunden ist, gehört es nicht zur Kante. Vereinfacht gesagt werden also Pixel mit ihren Nachbarpixeln verglichen und wenn ihr Intensitätsunterschied einen Schwellwert überschreitet, gehört das untersuchte Pixel zur Kante. Wenn dieser Unterschied unter einem Schwellwert liegt, gehört das Pixel nicht zur Kante. Alle Pixel, die Intensitätsunterschiede zwischen diesen Schwellwerten haben und mit einem Pixel verbunden sind, das zur Kante gehört, werden auch als Kantenpixel betrachtet.

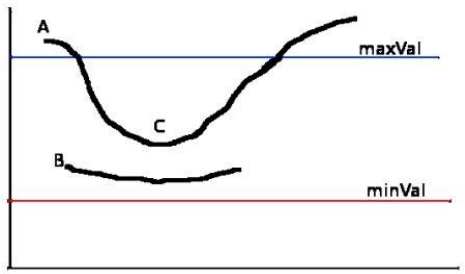


Abb. 49: Grafik einer angewandten Canny Edge Detection

7.8 Boxfilter

Ein Boxfilter filtert ein Bild systematisch durch und glättet es. Die Grösse des Kernels, also der Box, kann frei bestimmt werden, darf aber nicht grösser als die Bildbreite oder die Bildhöhe sein. In meinem Projekt wird meist ein 3*3 Boxfilter eingesetzt. Das heisst, die Box besteht aus 9 Pixeln. Der Filtervorgang eines einzelnen Pixels ist der folgende. Das Pixel hat 8 Nachbarpixel. Die Pixelwerte aller 9 Pixel werden addiert und mit 9 dividiert. Das zu filternde Pixel in der Mitte der Box bekommt nun diesen Durchschnittwert der 9 Pixel zugewiesen. Dieser Vorgang wird bei jedem Pixel ausgeführt und hat zur Folge, dass bei jedem Pixel der Einfluss der Nachbarpixel miteinbezogen und korrigiert wird. Die Box wird systematisch von Pixel zu Pixel verschoben und der Filter angewendet. Kanten und Konturen würden verschmiert werden, wenn der Boxfilter einen grösseren Kernel hätte.

8 Resultate

8.1 Linienverfolgung

Die Fähigkeit, einer geraden Linie nachzufahren, war eine wichtige Grundvoraussetzung, damit der Roboter später auch einer Abzweigung folgen konnte. Für die einfache Linienverfolgung müssen alle Parameter möglichst optimal gesetzt werden. Die grösste Herausforderung war dabei, herauszufinden, wie stark der Roboter eine Abweichung korrigieren soll. Die Korrektur wird dabei relativ zur Abweichung zur Linienmitte berechnet und ausgeführt. Für die Korrektur wird das eine Rad mehr und das andere weniger angetrieben. Um von der Abweichung zum Korrekturwert zu kommen, wird mit einem Korrekturfaktor multipliziert. Dieser Faktor war anfänglich zu klein und bewirkte somit zu wenig Korrektur. Mit vielen Stunden experimentieren konnte ein Korrekturfaktor gefunden werden, der gut funktioniert. Dieser hat in der Schlussversion den Wert 0.18. Trotz dem ziemlich optimalen Wert kann der Roboter nicht sehr schnell Abweichungen korrigieren. Dies führt dazu, dass der Roboter Abweichungen wellenartig korrigiert und er einige Zeit braucht, bis er zentriert und parallel zur Weglinie fährt. Bis zu diesem Zeitpunkt wurde Thresholding eingesetzt und noch nicht Canny Edge Detection. Es wurden bloss einzelne Stellen untersucht, damit erkannt wurde, ob die Weglinie vollständig im Bild lag oder nur eine Kante. Dies war allerdings sehr ungenau und es musste eine bessere Lösung her. Der Einsatz von Canny Edge Detection änderte dies massgeblich. Die Implementierung vereinfachte sich einerseits um ein Vielfaches und die Erkennung wurde sehr viel zuverlässiger als mit Thresholding.

8.2 Kreuzungsabzweigung

Der erste Versuch einer Abzweigung war eine Kreuzung. Mein Gedanke war, dass ich somit neben der Iteration für die Fahrt zwei weitere Iterationen für die Erkennung der Abzweigung machen kann, je eine für links und rechts der Weglinie. Dafür wurden also zwei weitere Iterationslinien benötigt, die vertikal verliefen. Diese durften nicht ganz am Bildrand liegen, da dort durch die Filtervorgänge Informationen verloren gehen. Als die Kreuzungsabzweigungen getestet wurden, wurde jedoch immer noch Thresholding eingesetzt zur Linienerkennung und die Idee, Canny Edge Detection zu verwenden, war noch nicht da. Damit man sicher sein konnte, dass eine Abzweigung vorhanden war und nicht der

Abb. 50: Kreuzung aus Sicht des Roboters

Hauptweg, war die Idee, nicht nur eine, sondern beide Kanten der Abzweigung zu suchen. Dies führte dazu, dass fast keine Abzweigung erkannt wurde. Wenn nur eine Kante der Abzweigung gesucht wurde, war das Problem, dass zu viele Abzweigungen erkannt wurden, die gar keine Abzweigungen waren, sondern wahrscheinlich Hauptweg. Zudem müsste sich der Roboter bei der Kreuzung möglichst genau auf der Abzweigung um 90° drehen, damit er sich weiter an der Weglinie orientieren könnte. Damit der Roboter immer so genau fährt, dass dies funktioniert, müsste sehr viel mehr Zeit aufgewendet werden als zur Verfügung stand. Man sieht auf der Abbildung auch, dass die Weglinie an einigen Stellen weisser scheint als das Flies. Somit war es für den Roboter unmöglich, an diesen Stellen den Weg im Threshold-Bild zu finden. Das ist eine weitere Schwachstelle dieser Version, welche die Fehlerquote vergrössert. Um all diesen schwachen Punkten entgegenzuwirken wurde die Idee aufgenommen, die fliessende Verzweigung auszuprobieren und dies auf einer anderen

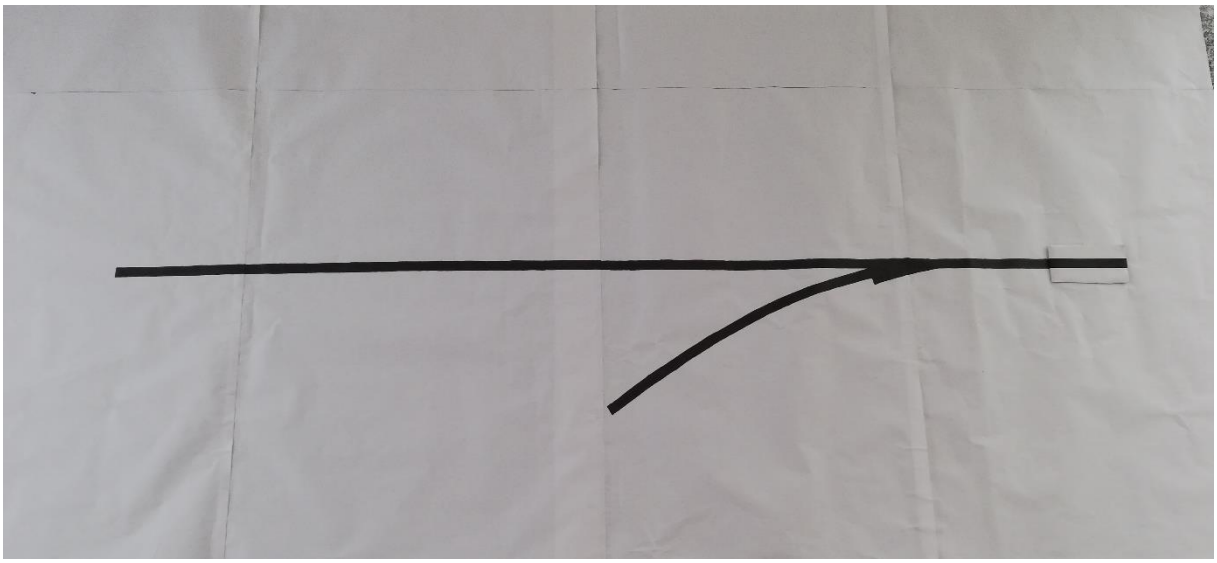


Abb. 51: Papier mit aufgemalter Weglinie und Startdepot aus Karton

Unterlage.

Damit das Problem mit der Spiegelung gelöst wird, ist es idealer, die Weglinie direkt auf die Unterlage zu malen. Dafür wurde der ganze Weg mit einem schwarzen Edding auf zusammengeklebte Flipchartpapiere gemalt. Zudem ist das Papier weisser und weniger texturiert als das Flies, erzeugt also weniger Fehler. Beim Zusammenrollen des Papiers muss dringend darauf geachtet werden, dass keine Falten entstehen. Das Papier ist allgemein nicht ganz so eben wie das Flies, eignet sich dennoch besser. Nach der Umstellung auf Papier mussten die Prozesszeiten des Roboters angepasst, genauer gesagt verringert werden, da das Papier weniger Widerstand liefert.

8.3 Umstellung auf Canny

Da das Erkennen einer Abzweigung mit Thresholding sehr mühsam und stark fehlerhaft war, musste eine andere Methode eingesetzt werden. Die Canny Edge Detection eignete sich dazu sehr gut, da nur Kanten weiss und alles andere schwarz zurückgegeben wird. Somit kann die Kante simpel durch eine Iteration gefunden werden, indem das erste weisse Pixel einer Pixellinie gesucht wird. Diese Methode vereinfachte die Implementierung sehr und war ein wichtiger Schritt zum Erreichen der Zielversion.

8.4 Verzweigung

Mit der Iteration im Canny-Bild kann nun bei einer Verzweigung immer einer Kante gefolgt werden. Wenn der Roboter links abbiegen soll, kann er der linken Kante folgen bis zur Wende. Bei der Rückfahrt fährt er der rechten Kante nach. Der Einsatz der Verzweigung in Kombination mit Canny ist viel zuverlässiger als die Kreuzungsabzweigung, bei der Canny nicht ein sehr grosser Vorteil ist. Damit der Roboter der linken Kante von Anfang an problemlos folgen kann, ist die Weglinie bereits vor der Verzweigung leicht verdickt und nach links gerichtet. Da bei diesen Versionen alles zusammengefügt wurde und die Schlussversion im Visier war, wird bei diesen Versionen auch der Elektromagnet eingesetzt. Dieser wird aktiviert, wenn der zugehörige GPIO Pin 40 eingeschaltet wird, also auf logisch 1 gesetzt wird.

8.4.1 Fliessende Fahrt

Bis anhin fuhr der Roboter immer in kleinen Schritten. Nach jedem Fahrprozess erfolgte eine Bildanalyse zum Ermitteln des Abstands und somit der Korrekturstärke im folgenden Fahrprozess. Die unterbrochene Fahrt funktionierte in den vorherigen Versionen nicht schlecht, war aber nicht so schön. Optimaler wäre eine ununterbrochene Fahrt des Roboters mit fortlaufenden Bildanalysen und entsprechenden Wegkorrekturen. Die Fahrtgeschwindigkeit muss dafür stark reduziert werden, damit das Bild nicht zu verschwommen und die Analyse ausreichend schnell ist. Dennoch muss die Geschwindigkeit insbesondere nach Wendungen genug hoch sein, dass der Roboter nicht zu wenig Schubkraft hat. All das musste berücksichtigt werden und war eine Herausforderung. Bei der Fahrtgeschwindigkeit, bei der der Roboter noch problemlos und ohne Hänger fährt, ist vermutlich die Bildanalyse zu langsam. Der Roboter kann somit zu wenig schnell auf Abweichungen bezüglich der Weglinie reagieren und verliert diese aus der Sicht. Um die Ursache genau benennen zu können, wären noch weitere zeitintensive Untersuchungen nötig. Diese Version hat somit eine sehr grosse Fehlerquote. Der Roboter schafft es meist nur beim

ersten Durchlauf, der Abzweigung zu folgen, da er oft bei der Rückfahrt oder beim Geradeausfahren die Weglinie aus der Sicht verliert.
Eine parallel zum Fahrprozess regelmässige Distanz-Messung und eine entsprechende Reaktion wäre optimal, jedoch sehr komplex.

8.4.2 Unterbrochene Fahrt

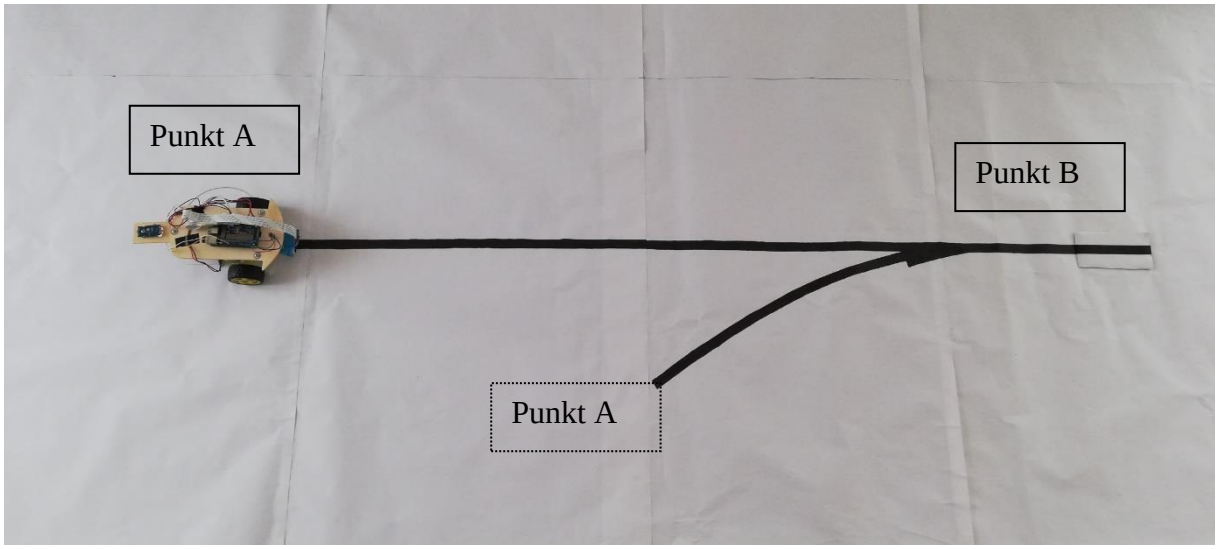


Abb. 52: Anlage mit Prozessstandorten

Da bisher die besten Resultate erzielt wurden, wenn Fahrprozess und Bildanalyse getrennt laufen, wurde wieder auf die unterbrochene Fahrt umgestellt. Für die finale Version entschied ich mich bezüglich Fahrvariante für die unterbrochene Fahrt und für Verzweigung statt Kreuzung. Der Roboter startet beim Punkt A und fährt in Richtung Holzblöcke. Dabei orientiert er sich an der linken Wegkante. Bei jeder Bildanalyse wird auch der Abstand vom Roboter zum Koffer frontal gemessen. Wenn der Koffer nahe ist, verlangsamt sich der Roboter und macht keine Wegkorrektur mehr, sondern misst nur noch den Abstand. So kann er ziemlich genau bei einem Abstand von 9.5cm anhalten und ein Bild machen (Punkt B). Nun wird mittels einer Bildanalyse die Anzahl Striche ermittelt. Daraufhin wird Strom auf den Elektromagneten gegeben und der Roboter dreht sich um 180°. Folglich wird der Holzblock an der magnetischen Schraube angezogen. Nach der Strichcodeanalyse wurde je nach Anzahl Striche die linke oder die rechte Abzweigung zum Verfolgen festgelegt. Im Falle von einem Strich biegt der Roboter rechts ab. Somit orientiert sich der Roboter bis zur nächsten Wende an der rechten Wegkante. Bei zwei Strichen verfolgt der Roboter die linke Wegkante und gelangt so zur linken Abzweigung. Während der ganzen Fahrt läuft die Wegkorrektur mittels Bildanalyse logischerweise wieder. Am Ende der Abzweigung findet er auf dem Canny-Bild keine Kante mehr und wendet. Danach wird der Holzblock abgeladen, in

dem der Elektromagnet ausgeschaltet wird. Diese Stelle stellt wieder Punkt A dar. Wo vorher die linke Kante verfolgt wurde, wird nach dieser Wende die rechte Kante verfolgt und umgekehrt. Nun macht sich der Roboter wieder auf den Weg in Richtung nächster Koffer und der ganze Prozess beginnt von vorne. Der ganze Ablauf des Algorithmus ist auf der Abb. 51 ersichtlich.

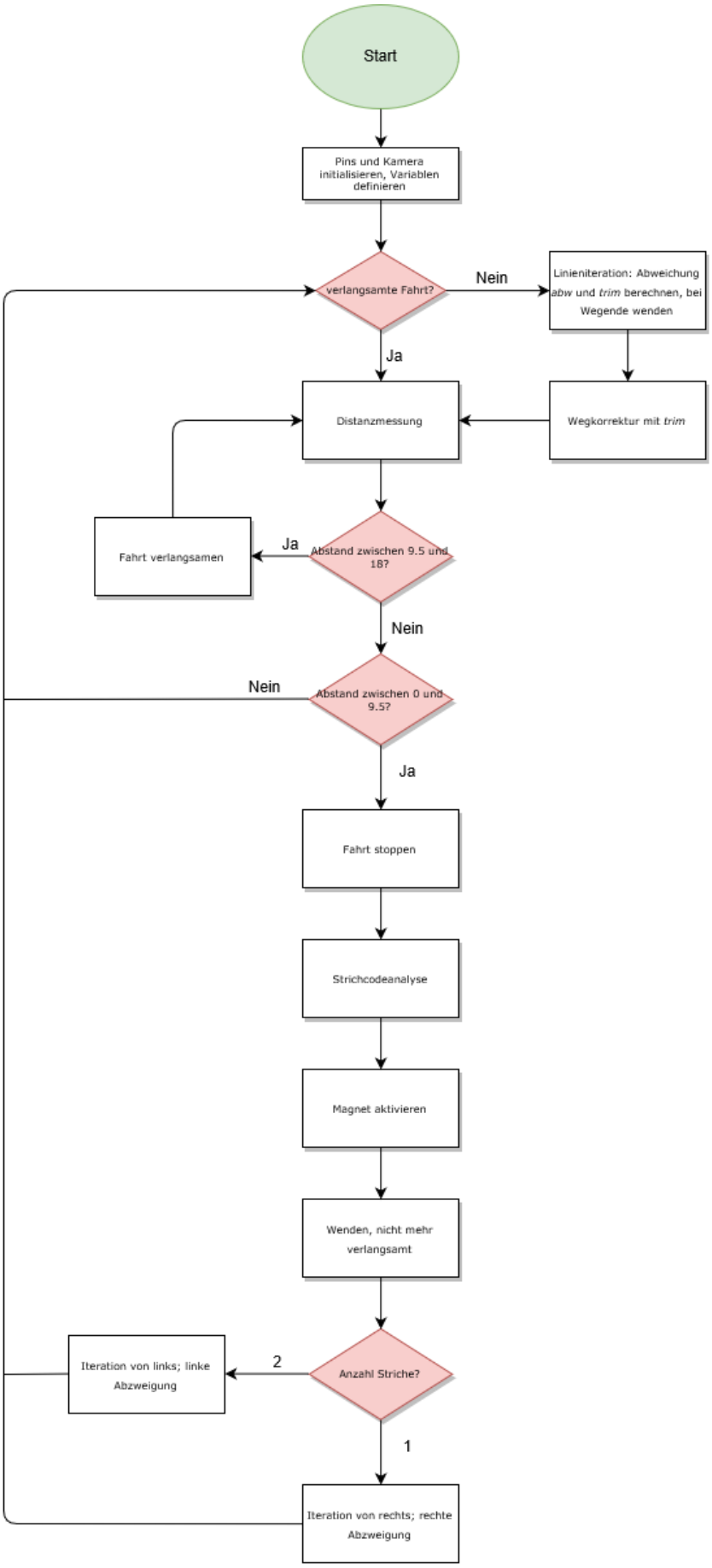


Abb. 53: Schlussalgorithmus

9 Diskussion

9.1 Wegkorrektur

Die Korrekturgrösse des Roboters wird zwar nach jedem Fahrprozess neu berechnet, die Bildanalyse für die Korrektur erfolgt jedoch für die Iterationslinie im Bild, die etwa 10cm vor dem Roboter liegt. Demzufolge korrigiert der Roboter im Grunde genommen immer zu früh. Er korrigiert nicht seinen Standort, sondern eine weiter vorne liegende Koordinate im Bild. Es ist die Koordinate (Bildmitte / Iterationslinie), also (320 / 240), die zum Ermitteln der Abweichung und schliesslich der Korrekturgrösse verwendet wird. Deshalb müsste eine Verzögerung zwischen Iterationslinie und der Radachse des Roboters einbezogen werden. Dafür müsste gemessen werden, wie lange der Roboter braucht, um von seinem Standort zum Korrekturstandort zu gelangen. Erst nach dieser Verzögerung müsste der Roboter die zuvor berechnete Abweichung korrigieren.

Des Weiteren wird zur Korrektur von Wegabweichungen in allen Versionen nur die Abweichung zur Wegmitte berücksichtigt. Damit wird dann die Korrekturgrösse linear berechnet. Wenn zur Berechnung der Korrekturgrösse zusätzlich der Winkel des Roboters zur Weglinie einbezogen würde, könnte eine deutlich genauere und effizientere Korrektur erzielt werden. Im Wissen, dass durch Einbezug von Winkelberechnungen ein noch optimaleres Resultat erzielt werden könnte und aus Komplexitätsgründen, entschied ich mich für die nun vorliegende Lösung dieser Thematik. Der Einbezug des Winkels fordert mehr Wissen im Gebiet der Regelungstechnik, das sich auf fortgeschrittenes Hochschulniveau beläuft. Für eine Optimierung müssten in einer nächsten Arbeit genaue Experimente und Messungen gemacht werden, um einen Algorithmus zu entwickeln, in dem der Winkel einbezogen wird.

9.2 Einsatz zweier Kameras

Ideal wäre es, wenn man zusätzlich zur bereits montierten Kamera eine weitere Kamera verwenden würde. So könnte die eine Kamera senkrecht nach unten gerichtet werden und sich auf die Korrektur von Wegabweichungen beschränken. Dann wäre wohl auch die Spiegelung kein Problem mehr. Die zweite Kamera müsste dann horizontal nach vorne gerichtet werden und würde nur bei der Strichcodeanalyse eingesetzt werden. Dann müsste aber zusätzlich ein Multikameraadapter eingesetzt werden und auch programmtechnisch wäre es komplexer. Um es möglichst simpel zu halten, beschränkte ich mich jedoch auf eine einzelne Kamera. So lernte ich auch, wie mit einer Kamera mehrere Aufgaben erledigt werden können. Ich konnte mich somit bereits ein wenig in das Gebiet von Computer Vision versetzen. Anstelle der

Kamera für die Linienverfolgung könnte auch ein Lichtsensor eingesetzt werden, wie es bei vielen Bausatzrobotern üblich ist. Das gewählte Setup ist allerdings näher am selbstfahrenden Auto, was die ganze Arbeit interessanter macht.

9.3 Rad

Anstatt dem kleinen, beweglichen Nylonrad des Roboters könnte auch eine frei bewegliche Kugel verwendet werden. Das Nylonrad stand beispielsweise nach einer Wende immer fast senkrecht zur Weglinie. Die Weiterfahrt muss dann immer noch leicht gegen die Wenderichtung korrigiert werden, damit der Roboter nicht in die Wenderichtung weiterfährt. Mit einer Kugel hätte man dieses Problem nicht und müsste keine Korrekturen nach Wendungen vornehmen.

9.4 Belichtung

Es ist schwierig, eine optimale Belichtung der ganzen Anlage zu erreichen. Dazu wurden neben dem Sonnenlicht auch Pultlampen eingesetzt. Diese waren allerdings verschieden stark, was es erschwerte, den Bereich des Koffer-Startdepots optimal und mit möglichst wenig Schatten zu belichten. Um die Belichtung gleichmässiger zu machen, müsste beispielsweise ein Set von mehreren kleinen Scheinwerfern oder vielen LED-Lampen zur Beleuchtung dienen. Die Beleuchtung abseits des Startdepots ist erfreulicherweise sehr unsensibel, da der Kontrast zwischen Weglinie und Unterlage sehr gross ist. Bei kleinen Schattenlinien beim Startdepot wird dieser Schatten im Threshold-Bild unterteilt, da nicht der ganze Schatten den Schwellwert des Thresholdings unterbietet, sondern einige Pixel des Schattens zu weissen Pixeln im Threshold-Bild werden. Somit werden in der Strichcodeanalyse viele Konturen erkannt, welche zum Teil innerhalb anderer Konturen liegen, was zur Folge hat, dass die Anzahl Striche nicht richtig erkannt wird. Grössere Schattenstellen sind kein Problem, solange sie nicht stark unterteilt werden im Threshold-Bild. Ein weiteres Problem war die Spiegelung, welche vor allem vom Sonnenlicht erzeugt wurde. Der Roboter sollte entweder in einem gut beleuchteten aber von direktem Sonnenlicht abgeschirmten Raum gestartet werden. Oder die Unterlage muss entsprechend möglichst wenig Licht spiegeln.

9.5 Kamerafokus

Auf Fotos der Raspberry Pi Kamera fällt auf, dass die Ferne viel schärfer ist als die Nähe. Das liegt daran, dass der Fokus auf eher weite Distanzen eingestellt ist. Für den Einsatz der Kamera in meinem Projekt ist dieser Fokus nicht optimal. Er ist allerdings ziemlich mühsam

einzustellen, da an der kleinen Linse gedreht werden muss.

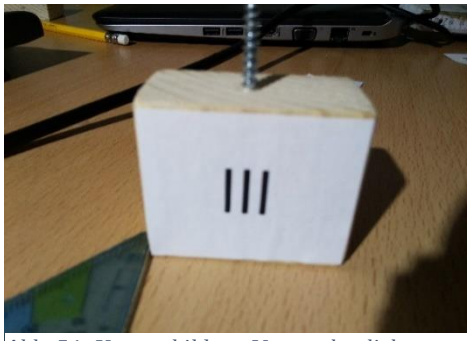


Abb. 54: Kamerabild zur Veranschaulichung des Kamerafokus'

9.6 Persönliches Schlussfazit

Insgesamt fand ich mein Projekt sehr interessant. Mein Ziel bei der Themensuche war, ein Thema zu nehmen, dass mich interessiert, bei dem ich auch viel lernen kann und Arbeiten in verschiedenen technischen Teilbereichen bietet. Das ist mir gelungen. Weil mein Projekt ziemlich komplex war und ich beispielsweise die Programmiersprache komplett neu lernen musste, war die Arbeit auch sehr zeitintensiv. Oft musste stundenlang experimentiert werden, um zu verstehen, was noch optimiert werden kann. Deshalb opferte ich auch ein Grossteil der Sommerferien für die Maturaarbeit. Ich fing früh an, konkret an der Maturaarbeit zu arbeiten und Material zu besorgen. Dies zahlte sich schlussendlich aus, da auch in der Endphase nicht viel Zeit übrigblieb. Ursprünglich sollte die Schlussversion mehrere Abzweigungen enthalten. Dies lag aber aus Zeit- und Komplexitätsgründen nicht mehr drin. Während der Maturaarbeit lernte und verbesserte ich viele Fertigkeiten, z.B. Löten. Neben den elektrotechnischen Aspekten verbesserte ich beim Implementieren auch das genaue und fehlerfreie Arbeiten und das logische Denken. Das vermehrte Auftreten von Fehlermeldungen und die „Ungehorsamkeit" des Roboters brachte mich beim Programmieren oft an den Rand der Verzweiflung. Rückblickend bin ich froh, mich für die Maturaarbeit in diesem Themenbereich entschieden zu haben, erfreute ich mich doch während der ganzen Arbeit eines grossen Interessens und Motivation. Ich bin stolz auf meine Arbeit.

10 Quellen

10.1 Bücher

Weigend, M. (2016). *Raspberry Pi programmieren mit Python*

Schmidt, M. (2014). *Raspberry Pi*

Vrisk, S. R. (2016). *Autonomer Roboter mit Raspberry Pi 2*

10.2 Internetquellen [Stand 11.10.2018]

Raspberry Pi

https://www.raspberrypi.org/

Terminalbefehle

https://praxistipps.chip.de/raspberry-pi-die-wichtigsten-befehle-auf-einen-blick_45521
https://maker-tutorials.com/35-raspberry-pi-linux-cli-terminal-befehle/

SSH einrichten

https://tutorials-raspberrypi.de/raspberry-pi-ssh-windows-zugriff-putty/

Python Bibliothek

https://docs.python.org/3/library/

OpenCV Download

https://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/
https://docs.opencv.org/master/d7/d9f/tutorial_linux_install.html

OpenCV Bibliothek

https://docs.opencv.org/master/index.html

Numpy Bibliothek

http://scipy.github.io/old-wiki/pages/Tentative_NumPy_Tutorial#The_Basics

Kamera

https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/
https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/6

Bildaufbau

https://de.wikipedia.org/wiki/Farbtiefe_(Computergrafik)

Threshold

https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

Canny Edge Detection

https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

Ultraschallsensor Hardware

https://developer-blog.net/ultraschall-sensor-hc-sr04/

Ultraschallsensor Software

https://tutorials-raspberrypi.de/entfernung-messen-mit-ultraschallsensor-hc-sr04/

10.3 Abbildungen

Die restlichen Abbildungen wurden mit der Raspberry Pi Kamera oder der Handykamera geschossen, mit IDLE generiert oder sind Screenshots.

11 Danksagung

Zum Schluss meiner Arbeit bedanke ich mich herzlich bei folgenden Personen für die grossartige Unterstützung:

Kornel Eggerschwiler

- für die Idee, eine Arbeit mit Raspberry Pi zu gestalten
- für die vielen hilfreichen Ratschläge zu Vorgehensweisen und die Hilfe bei der Fehlersuche
- für den Druck der Halterungen

Walter Eggerschwiler

- für die Hilfe beim Erstellen der Pläne und beim Löten

Markus Kopp

- für die finale Idee der autonomen Gepäcksortierung
- die Bereitstellung des Werkraums mit Werkzeug, Holz etc.
- seine Unterstützung in der Anfertigung verschiedener Teile.

Herr Ghezal

- für die Betreuung während der ganzen Arbeit

Allen, die mit Gesprächen, aufmunternden und anerkennenden Worten und Gesten zur Motivation und zum Gelingen des Projektes beigetragen haben.

12 Anhang

12.1 Deklaration

„Ich erkläre hiermit,

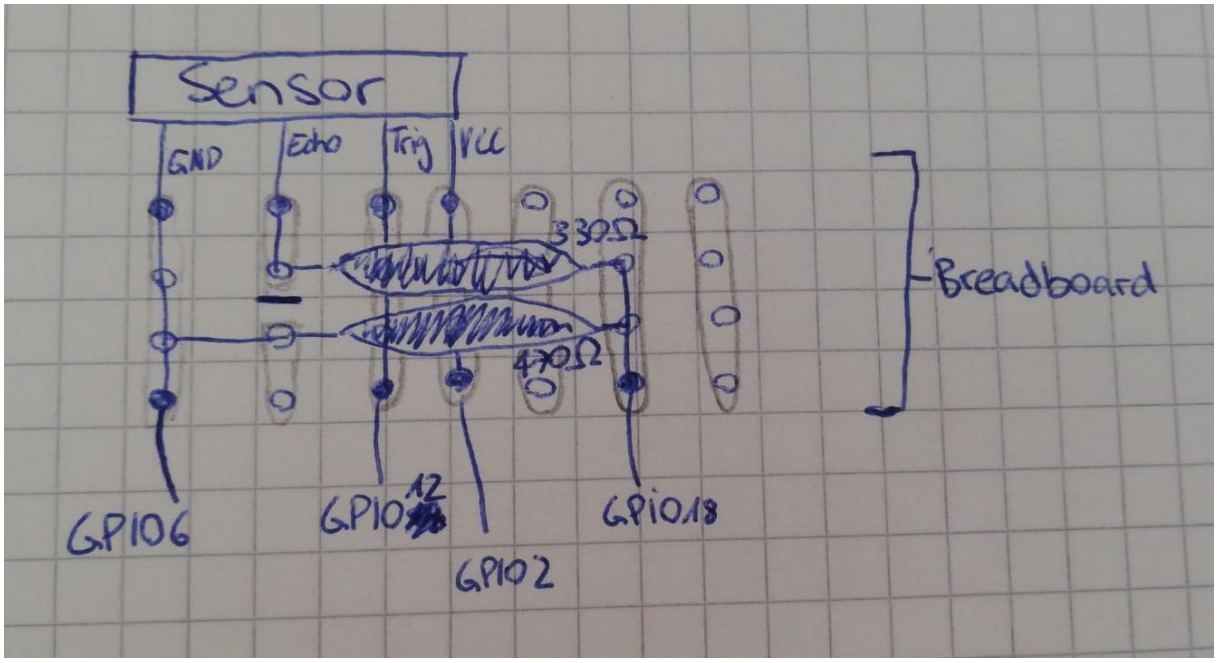
- dass ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Quellen verfasst habe,
- dass ich auf eine eventuelle Mithilfe Dritter in der Arbeit ausdrücklich hinweise,
- dass ich vorgängig die Schulleitung und die betreuende Lehrperson informiere, wenn ich diese Maturaarbeit, bzw. Teile oder Zusammenfassungen davon veröffentlichen werde, oder Kopien dieser Arbeit zur weiteren Verbreitung an Dritte aushändigen werde.“

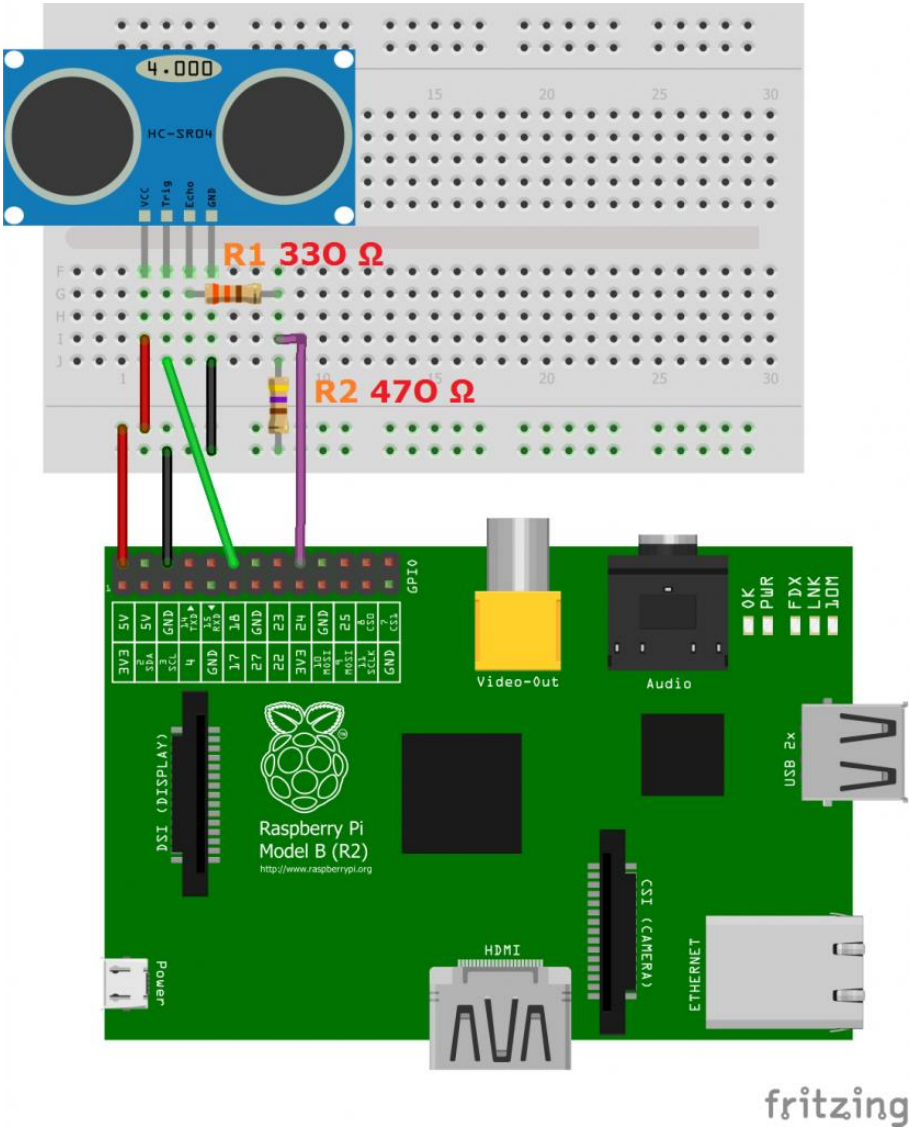
Ort: Buttisholz

Datum: 11. Oktober 2018

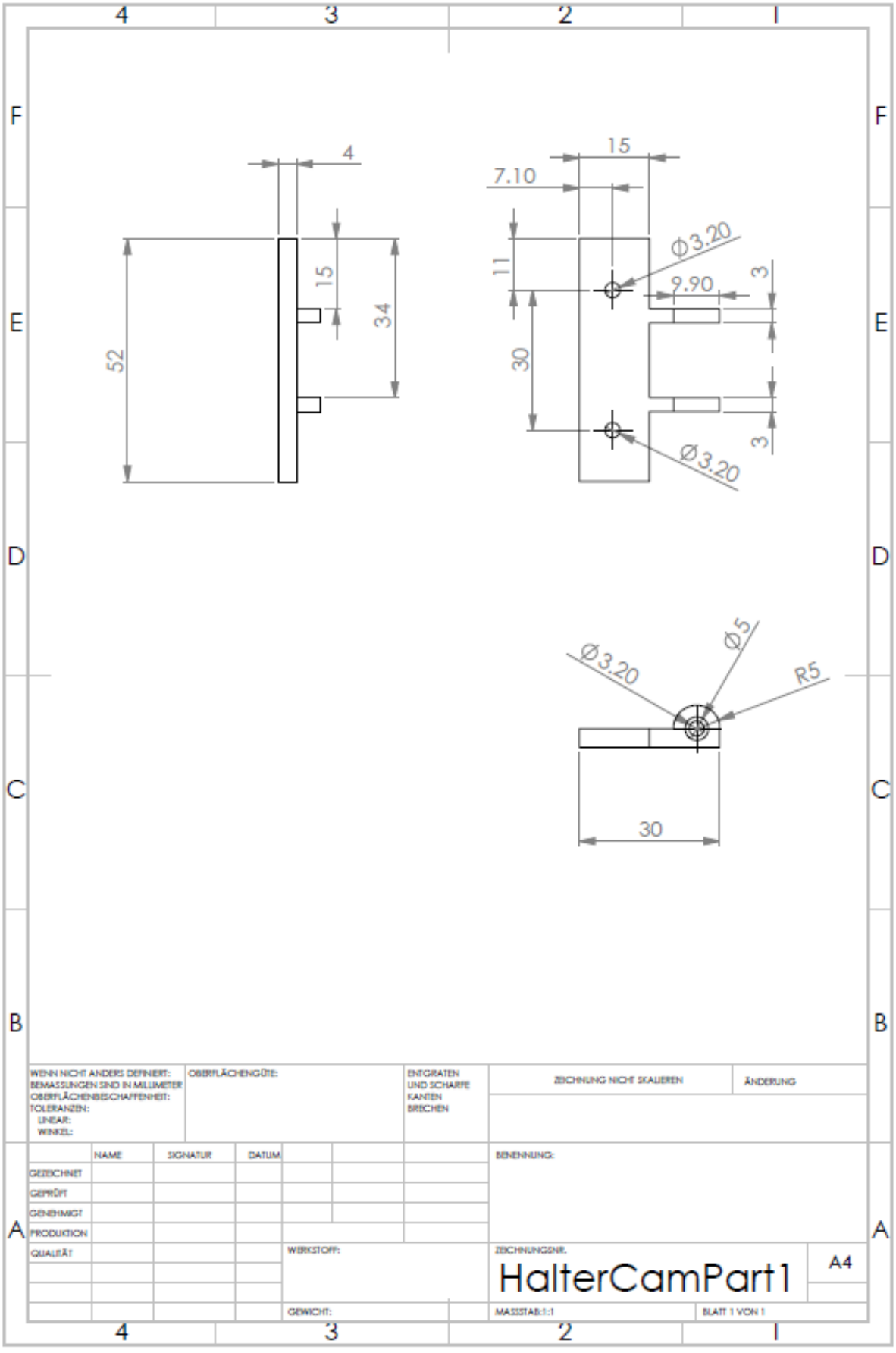
Unterschrift:

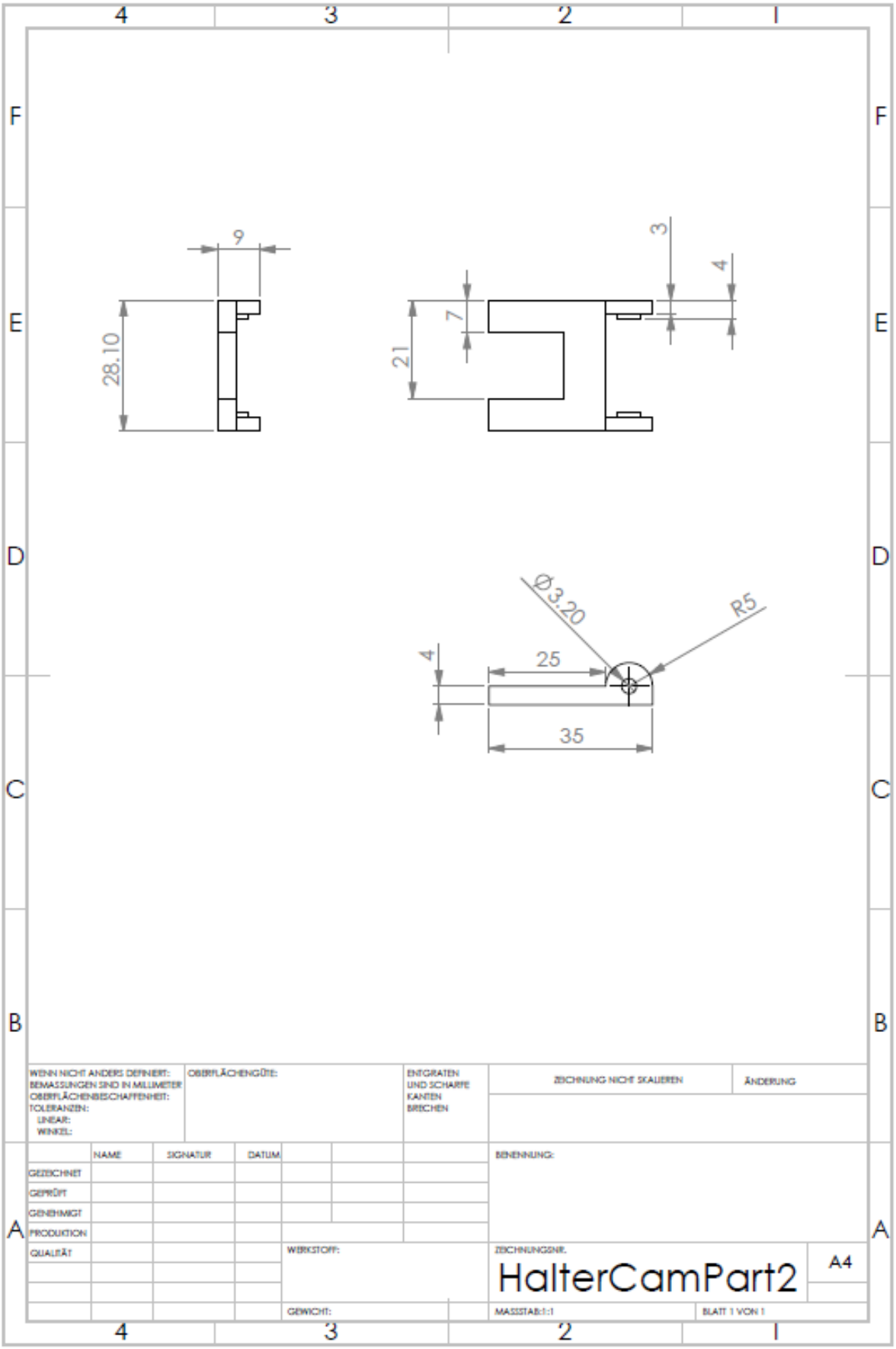
12.2 Schaltplan des Ultraschallsensors

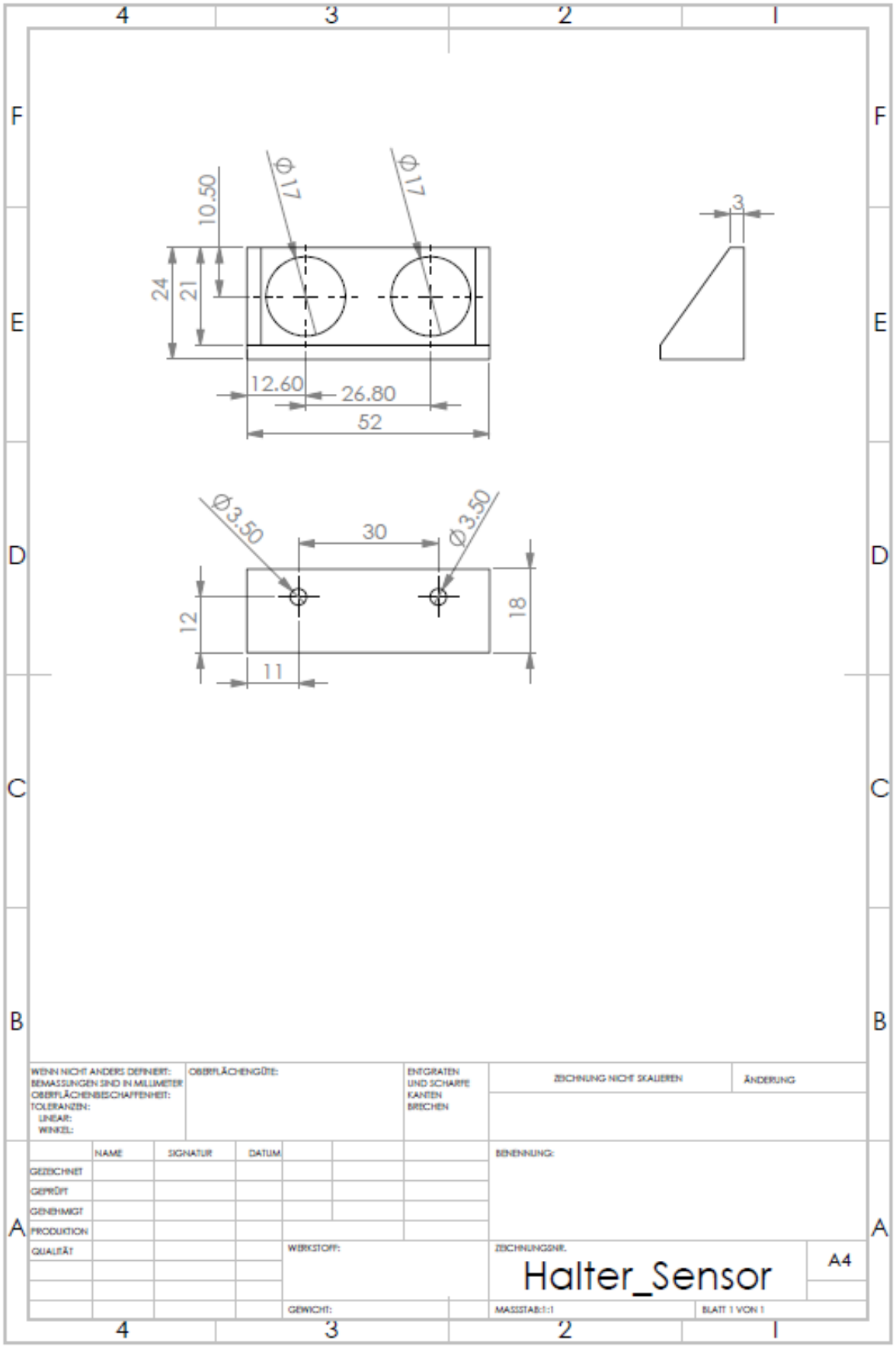




12.3 Pläne der Halterungen




4
3
2
1
F
E
D
C
B
A
9
28.10
3
4
7
21
Ø3.20
R5
4
25
35
WENN NICHT ANDERS DEFINIERT:
BEMASSUNGEN SIND IN MILLIMETER
OBERFLÄCHENBESCHAFFENHEIT:
TOLERANZEN:
LINEAR:
WINKEL:
OBERFLÄCHENGÜTE:
ENTGRATEN UND SCHARFE KANTEN BRECHEN
ZEICHNUNG NICHT SKALIEREN
ÄNDERUNG
NAME
SIGNATUR
DATUM
GEZEICHNET
GEPRÜFT
GENEHMIGT
PRODUKTION
QUALITÄT
BENENNUNG:
WERKSTOFF:
ZEICHNUNGSNR.
HalterCamPart2
A4
GEWICHT:
MASSSTAB:1:1
BLATT 1 VON 1


4
3
2
1
F
E
D
C
B
A
Ø17
Ø17
10.50
24
21
12.60
26.80
52
3
Ø3.50
30
Ø3.50
12
18
11
WENN NICHT ANDERS DEFINIERT:
BEMASSUNGEN SIND IN MILLIMETER
OBERFLÄCHENBESCHAFFENHEIT:
TOLERANZEN:
LINEAR:
WINKEL:
OBERFLÄCHENGÜTE:
ENTGRATEN UND SCHARFE KANTEN BRECHEN
ZEICHNUNG NICHT SKALIEREN
ÄNDERUNG
NAME
SIGNATUR
DATUM
GEZEICHNET
GEPRÜFT
GENEHMIGT
PRODUKTION
QUALITÄT
BENENNUNG:
WERKSTOFF:
ZEICHNUNGSNR.
Halter_Sensor
A4
GEWICHT:
MASSSTAB:1:1
BLATT 1 VON 1

12.4 Programmcode

Alle Python-Files, Auswertungsbilder und IDLE sind auf der beigelegten CD bzw. dem USB Stick zu finden.