

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Design, Medien und Information

Department Medientechnik

IT-Systeme

Prof. Dr. Torsten Edeler

Projektbericht

Raspberry Car – Team II

Jörn Kogerup

Darius Weiberg

Mirco Hülsemann

12. Februar 2018

Inhaltsverzeichnis

1 Einleitung	3
2 Projektvorstellung	3
3 Projektziel, Anforderungen und technische Rahmenbedingungen	4
4 Die Rennstrecke	4
5 Das Auto	5
6 Umsetzung	5
6.1 Kamerahalterung	5
7 Schaltplan	5
8 Code	6
8.1 main.py	7
8.1.1 import	7
8.1.2 main	8
8.1.3 line	10
8.1.4 linienfahren	11
8.1.5 lenken	13
8.1.6 checkgreen	14
8.1.7 checkblue	15
8.1.8 makevideo	16
8.2 setup.py	17
8.3 aufraeumen.py	18
9 3D-Druck Halterungen	19
9.1 Kamerahalterung	19
9.2 Ultraschallsensorhalterung	19
10 Geplante Weiterentwicklungen	21
11 Anhang	22
11.1 main.py	22

1 Einleitung

Dieser Projektbericht dient der Dokumentation des Projekts “Entwicklung eines selbstfahrenden Autos” im Modul IT-Systeme des Studiengangs Medientechnik an der HAW Hamburg im Wintersemester 2017/18.

Autonomes Fahren ist derzeit ein großes und hochaktuelles Forschungsthema vieler Universitäten, IT-Unternehmen und Automobilhersteller. Mit Hochdruck arbeiten Unternehmen wie Tesla, General Motors oder Waymo zusammen mit IT-Firmen wie Intel und Google daran, selbstfahrende Autos für die breite Masse verfügbar zu machen.

Aufgrund der aktuellen Relevanz und der Tatsache, dass ein selbstfahrendes Auto die ideale Verkörperung eines informationstechnischen Systems ist, ist dieses Thema als Projekt für die Veranstaltung “IT-Systeme” im Wintersemester 2017/2018 auserwählt worden.

Der vorliegende Abschlussbericht enthält Erläuterungen zum Projektziel, zum Projektumfeld sowie zu den gegebenen technischen Rahmenbedingungen. Es wird das technische Konzept zur Erreichung des Projektziels vorgestellt sowie die einzelnen Schritte des Entwicklungsprozesses dokumentiert und erläutert. Der gesamte Code sowie Bilder sind im Anhang des Dokuments zu finden.

2 Projektvorstellung

Im Rahmen der Vorlesung IT-Systeme (6. Semester des Studiengangs Medientechnik) sollen die Studenten/Studentinnen in kleinen Gruppen ein autonomes Modellbau-Auto entwickeln, welches in der Lage sein soll, mithilfe von Sensoren und Kameras selbstständig eine vorher definierte “Rennstrecke” abzufahren. Die Aufgaben der Studenten umfassen dabei:

- eine konzeptionelle Planung zur Umsetzung des Projekts
- die Konstruktion des Modellbau-Autos (dabei wird ein Bausatz verwendet) sowie der Rennstrecke
- die Auseinandersetzung mit der Funktionsweise der elektrischen Komponenten
- die Verschaltung der elektrischen Komponenten
- Programmierung und Implementierung des Steuer-Codes auf einem Raspberry Pi

Als zeitlicher Rahmen ist ein Semester vorgesehen. Pro Woche gibt es eine 3-stündige Einheit, in der die Studenten unter Betreuung am Projekt weiterarbeiten können. Es ist jedoch auch möglich, außerhalb dieser Einheit am Projekt weiterzuarbeiten.

Es gibt zudem zwei "Challenge"-Termine im Semester, zu welchen bestimmte Zwischenziele erreicht werden müssen. Auf diese Art wird das Vorankommen der Projekts überprüft und gewährleistet. Am Ende des Semesters erfolgt eine Generalprobe und schließlich die Projektausstellung.

3 Projektziel, Anforderungen und technische Rahmenbedingungen

Ziel des Projekts ist es, ein autonomes Auto zu entwickeln, welches mithilfe von Mikrocomputer, Kamera und Abstandssensoren zwei Runden auf einer definierten Rennstrecke selbstständig und erfolgreich absolviert. Dabei sollen die Autos einzeln starten, es befindet sich also nie mehr als ein Auto auf der Strecke. Die Gruppe, deren Auto den Kurs am schnellsten durchfährt, wird als Gewinner geehrt.

Auf dem Weg zu diesem Ziel gibt es im Laufe des Semester zudem zwei Zwischenziele, sogenannte Challenges, die erfolgreich absolviert werden müssen. Das erste Zwischenziel ist eine Geradeausfahrt von 10 Metern mit Geschwindigkeitswechseln, das zweite Zwischenziel ist das Fahren entlang einer Wand mithilfe von Abstandssensoren.

4 Die Rennstrecke

Die "Rennstrecke", die es zu meistern gilt, wird am Anfang des Semesters grob definiert: Sie soll als geschlossene Rundstrecke (Startlinie = Ziellinie) in einem Laborraum der Hochschule aufgebaut werden. Im Wesentlichen soll sie aus einer Mittellinie sowie zwei Begrenzungslinien bestehen. Der Mittellinie gilt es zu folgen, die Begrenzungslinien dürfen nicht überschritten werden. Die Strecke soll außerdem mehrere Kurven mit verschiedenem Radius enthalten sowie einen geraden Abschnitt ohne Mittellinie entlang einer Wand. In diesem Abschnitt ohne Mittellinie sollen Abstandssensoren genutzt werden, um auf der Spur zu bleiben.

Die Linien werden mit Klebeband realisiert, welches auf den Boden des Laborraums aufgeklebt wird. Als Mittellinie dient rotes Tape, als Begrenzungslinie schwarzes.

Aufgrund von Problemen mit den verfügbaren Abstandssensoren, die sich erst im Laufe des Semesters ergeben haben, ist der gerade Abschnitt entlang der Wand in der endgültigen Version nun doch mit einer Mittellinie versehen. Die Aufgabe, mithilfe von Abstandssensoren entlang einer Wand zu fahren, fällt somit - zumindest in der Generalprobe und Projektausstellung - weg.

Eine weitere Änderung der anfänglichen Version der Rennstrecke ist die Einführung einer Ampel an der Start-/Ziellinie. So sollen die Autos nur bei grünem Licht starten und automatisch anhalten, wenn das Licht blau geschaltet ist. Die Ampel wird durch einen LED-Streifen auf dem Boden realisiert.

5 Das Auto

Die Komponenten, die für den Bau des Autos verwendet werden dürfen, sind vorgegeben und werden von der Hochschule bereitgestellt:

- Einfacher Modellbau-Auto-Bausatz, bestehend aus Rahmen, DC-Getriebemotoren und gummierten Rädern
- 7,2 V NiMH Akku mit einer Kapazität von 4000 mAh
- Motortreiber L298N für DC-Motoren mit Dual H-Bridge
- Raspberry Pi 3 Model B (4-Kern-CPU, 1,2 GHz, 1 GB Ram, WLAN, Bluetooth)
- USB-Webcam (640 x 480 px, 30 fps, manueller Fokus)
- Ultraschallsensor HC-SR04 (messbare Distanz: 2-300 cm, Auflösung: 3 mm, max. 50 Messungen pro Sekunde)

Alle Komponenten (mit Ausnahme des Raspberrys) wurden im Hinblick auf einen möglichst günstigen Einkaufspreis ausgewählt. Dementsprechend sind die Erwartungen an Haltbarkeit, Genauigkeit und Ausfallsicherheit der Produkte und letztendlich an die Performance des Autos nicht allzu hoch.

Neben den aufgeführten Komponenten steht den Studenten eine Werkstatt mit 3D-Drucker, Lötgerät und einer Vielzahl an Elektronik-Bauteilen (Kabel, Widerstände, Platinen ...) zur Verfügung.

Als Programmiersprache wird Python verwendet. Diese universelle Sprache zeichnet sich durch einfache und leicht verständliche Syntax, große Nutzerbasis sowie mächtige Bibliotheken aus und ist quasi Standardsprache auf dem Raspberry Pi.

6 Umsetzung

7 Schaltplan

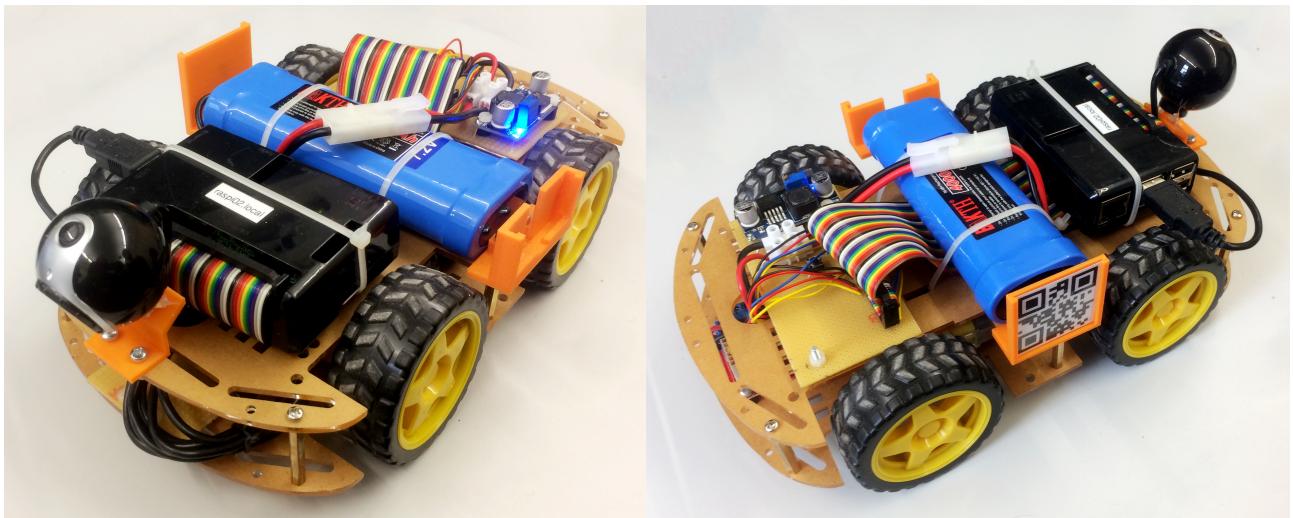


Abbildung 1: Das Auto im Finalen Aufbau

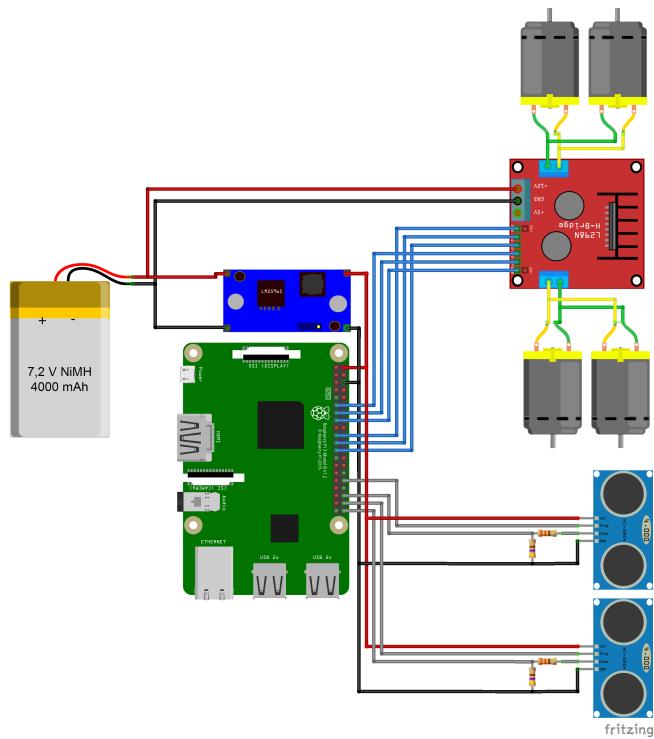


Abbildung 2: Schaltplan

8 Code

Das Programm wird über die *main.py* Datei ausgeführt. Zwei Module *setup.py* und *aufrauenmen.py* sind ausgelagert, dadurch ist die Hauptdatei übersichtlicher und die Module können einfacher von anderen Dateien mitbenutzt werden.

8.1 main.py

In *main.py* wird threading verwendet, was gleich mehrere Vorteile hat. Die Bilderaufnahme ist extrem viel schneller, wenn sie auf einem anderen thread als der restliche Algorithmus läuft und das abspeichern der Bilder für eine Videoausgabe würde ohne das threading die Rechendauer um ein vielfaches erhöhen. Außerdem können so andere Funktionen unabhängig von dem restlichen Programm problemlos ausgeführt werden.

Es laufen drei threads parallel. Der eigentliche Fahralgorithmus *linienfahren*, die Videoausgabe *makevideo* und die Ampelüberprüfung *checkblue*. *linienfahren* ruft zwei Funktionen auf, *line* zum Bildaufnehmen und *lenken* zur Motorsteuerung.

Es werden mehrere globale Variablen zwischen den einzelnen threads benutzt. Zum einen die Bildvariablen *ret* und *img*, damit nicht jede Funktion eigene Bilder aufnehmen muss und zum anderen der Linienmesswert *x* und die verstrichene Zeit *minutes*, *seconds*, die zur Videoausgabe benötigt werden.

8.1.1 import

```
1 import threading      # Modul threads
2 import cv2            # Dies ist die Bildverarbeitungsbibliothek OpenCV
3 import numpy as np    # Rechnen mit vielen Zahlen in einem Array (z. B. Bilder)
4 import math           # Modul math
5 import time           # Modul time

7 from aufraeumen import aufraeumen, losfahren, bremsen # Funktion für start/stop importieren
8 from setup import * # GPIO Setup importieren und ausführen
9
10 cap = cv2.VideoCapture(0) # Input 0
11 # Codec und VideoWriter object für Video Output
12 fourcc = cv2.VideoWriter_fourcc(*'XVID')
13 out = cv2.VideoWriter('output.avi',fourcc, 15, (640,480))
14 ret, img = cap.read()
```

..../main.py

Am Anfang der Hauptdatei werden die benötigten Module importiert, gefolgt von den beiden ausgelagerten Dateien. Von *aufraeumen.py* werden die benötigten Funktionen und von *setup.py* wird alles importiert, da diese Datei keine Funktion enthält.

In den Zeilen 10 bis 14 werden die benötigten Einstellungen für OpenCV 2 vorgenommen und ein erstes Bild aufgenommen.

8.1.2 main

```
def main():
    losfahren()
    pr.start(0) # Motor A, speed Tastverhältnis
    pl.start(0) # Motor B, speed Tastverhältnis

    run_event = threading.Event()
    run_event.set()

    th1_delay = .01 # sleep dauer der Funktion
    th2_delay = .01 # sleep dauer der Funktion
    th3_delay = .001 # sleep dauer der Funktion
    th1 = threading.Thread(target=linienfahren, args=(th1_delay, run_event)) # Funktion in
    einem neuen Thread zuordnen
    th2 = threading.Thread(target=checkblue, args=(th2_delay, run_event)) # Funktion in
    einem neuen Thread zuordnen
    th3 = threading.Thread(target=makevideo, args=(th3_delay, run_event)) # Funktion in
    einem neuen Thread zuordnen

    th3.start() # Video Thread starten

    no_green = 0
    print("Warten auf grüne Ampel")

    while no_green < 500:
        no_green = checkgreen()

    th1.start() # Linienfahren Thread starten
    th2.start() # Ampel blau Test Thread starten

    # Warten bis Strg+C gedrückt wird:
    try:
        while 1:
            time.sleep(.01)

    except KeyboardInterrupt:
        print("attempting to close threads. Max wait =", max(th1_delay, th2_delay, th3_delay))
        #
        bremsen()
        run_event.clear()
        th1.join()
        print("Thread 1 closed")
        th2.join()
        print("Thread 2 closed")
        th3.join()
        print("Thread 3 closed")
        aufräumen()
        print("Threads successfully closed")

    cap.release()
    out.release()

if __name__ == '__main__':
    main()
```

..../main.py

Das Programm wird in *main* gestartet und beendet. Als erstes werden die Motoren mit der

losfahren Funktion gestartet, allerdings noch mit einem Tastverhältnis von 0%. Anschließend wird das threading gestartet. Dafür wird *run_event*, das als *while*-Bedingung für die threads dient, gesetzt. Die delay-Angaben und die drei verwendeten threads th1, th2 und th3 werden definiert. Dabei ist *target* die Funktion die in dem neuen thread laufen soll und *args* sind die Variablen die mit übergeben werden. Die drei threads sind *linienfahren* (der eigentliche lenk-Algoritmus), *checkblue* (Test ob die Ampel blau leuchtet) und *makevideo* (die Videoausgabe).

Der erste thread (die Videoaufzeichnung) wird gestartet, gefolgt von einer *while*-Schleife die überprüft ob die Ampel grün ist. Wenn der zurückgegebene Wert der *checkgreen* Funktion größer gleich 500 ist, also ausreichend Grünanteile erkannt wurden endet die Schleife. Dadurch werden die anderen beiden threads gestartet, wodurch das Auto losfährt.

Damit das Programm wieder beendet werden kann, folgt eine *try-while* und *except* Schleife, die weiter keine andere Funktion hat. Sobald ein *KeyboardInterrupt* durch das drücken von Strg-c auftritt wird das Auto gebremst und die threads werden geschlossen. Allerdings muss dabei gewartet werden bis die *while*-Schleifen aller threads durchgelaufen sind. Anschließend werden die GPIO einstellungen bereinigt, die Videoaufnahme beendet und das Video gespeichert.

Gestartet wird die *main* Funktion durch eine *if __name__ == "__main__"* abfrage, um die Datei auch als Modul verwenden zu können.

8.1.3 line

```
1 def line(zeileNr):
2     global img, ret
3     ret, img = cap.read()
4
5     img_r = img[zeileNr, :, 2] # Alles aus zeileNr und Breite und Farbkanal 2
6     img_g = img[zeileNr, :, 1]
7     img_b = img[zeileNr, :, 0]
8
9     zeile_bin = (img_r.astype('int16') - (img_g / 2 + img_b / 2)) > 60 # Rotanteil über
10    Threshold
11
12    # Mittelpunkt berechnen und return:
13    if zeile_bin.sum() != 0:
14        x = np.arange(zeile_bin.shape[0]) # x=0,1,2 ... N-1 (N=Anzahl von Werten in zeile_bin)
15    else:
16        return None
```

..../main.py

In *line* wird ein Bild aufgenommen und, in einer Zeile, der Mittelpunkt der roten Anteile ermittelt.

Der Funktion wird beim aufrufen die Zeile (*zeileNr*) übergeben, die auf den Rotwert analysiert werden soll. Damit wird es ermöglicht die selbe Funktion zum analysieren unterschiedlicher Zeilen zu verwenden. Das wird in der finalen Version aber nicht benutzt.

In Zeile 12 bis 16 wird der Mittelpunkt berechnet und zurückgegeben. Falls kein Rotanteil über dem Schwellwert liegt, wird *None* zurückgegeben.

8.1.4 linienfahren

```
1 def linienfahren(delay, run_event):
2     global cap
3     global x, minutes, seconds
4
5     ret, img = cap.read()
6     width = np.size(img, 1)
7     ideal = width/2
8     mitte = ideal
9     last_mitte = mitte
10    steer = 1
11    startzeit = time.time()
12
13    while run_event.is_set():
14        last_mitte = mitte
15        mitte = line(70)
16
17        if mitte is None:
18            if last_mitte > ideal:
19                mitte = 640
20            else:
21                mitte = 0
22        x = mitte
23
24        if mitte == ideal:
25            steer = 1
26        elif mitte < ideal:
27            steer = (mitte/ideal)
28            speed = steer*60+40
29            steer = steer*.9+.1
30        elif mitte > ideal:
31            steer = (width-mitte)/ideal
32            speed = steer*60+40
33            steer = 2-(steer*.9+.1)
34
35        lenken(steer, speed)
36
37        hours, rem = divmod(time.time()-startzeit, 3600)
38        minutes, seconds = divmod(rem, 60)
39
40        print(" " * int(mitte/10), "■", " " * int(64 - mitte/10), "x = %.1f" % mitte, ";steer = %.1f" % steer, ";speed = %.1f" % speed, ";time = {:0>2}:{:05.2f}".format(int(minutes), seconds))
41        print(" " * int(ideal/10), "|")
42
43        time.sleep(delay)
```

.. /main.py

linienfahren ruft die Funktion *line* auf und berechnet die Lenkung und Geschwindigkeit aus dem zurückgegebenen Wert.

Am Anfang der Funktion wird ein Testbild aufgenommen und die für die Berechnung benötigten Variablen erstellt. Danach folgt eine *while* Schleife, die bis zum Beenden des threading durchläuft. Falls keine rote Linie im Bild war, oder in seltenen Fällen, wenn die Linie nicht erkannt wurde, gibt *line* *None* zurück. In diesem Fall wird, in den Zeilen 17 bis 21, überprüft ob die Linie zuletzt rechts oder links im Bild war. Der Messwert *mitte* wird dementsprechend

auf das Minimum (0) oder das Maximum (640) gesetzt. Anschließend wird der Wert für die Videoausgabe abgespeichert (Zeile 22).

In den Zeilen 24 bis 33 wird der Wert für die Lenkung und Geschwindigkeit berechnet. Unterschieden wird zwischen drei Fällen. Ist der Messwert in der Mitte des Bildes wird geradeaus gelenkt, andernfalls wird die Lenkung und die Geschwindigkeit berechnet. Ist der Messwert größer als der Bildmittelpunkt wird $2 - \text{Lenkwert}$ gerechnet. Es ergibt sich ein Lenkwert zwischen 0 und 1 für eine Linkskurve und 1 bis 2 für eine Rechtskurve. Das hat den Vorteil, dass der *lenken* Funktion nicht zusätzlich eine Richtungsangabe übergeben werden muss. Anschließend wird die *lenken* Funktion mit den Lenk- und Geschwindigkeits-Variablen aufgerufen .

Der Wert für die Lenkung und die Geschwindigkeit wird aus dem Messwert und dem Bildmittelpunkt berechnet:

$$\text{Lenkung} = \frac{\text{Messwert}}{\text{Bildmittelpunkt}} \cdot 90\% + 10\% \quad (1)$$

$$\text{Geschwindigkeit} = \frac{\text{Messwert}}{\text{Bildmittelpunkt}} \cdot 60\% + 40\% \quad (2)$$

Damit die Lenkung nicht zu stark ist, wird der Wert kleiner gewichtet (-90%) und angehoben (+10%). So startet der Wert nicht bei 0% sondern bei 10%. Die Geschwindigkeit wird gleich berechnet, allerdings mit (-60%) gewichtet und (+40%) angehoben. Es wird also mit 60% bis 100% Geschwindigkeit gefahren.

Für die Text- und Videoausgabe wird die verstrichene Zeit berechnet. Anschließend folgt in Zeile 40/41 eine Textausgabe über die Konsole. Dabei werden $\frac{\text{Messwert}}{10}$ Leerzeichen, dann ein Blockzeichen als Position des Messpunktes im Bild und die verbleibenden $64 - \frac{\text{Messwert}}{10}$ Leerzeichen geschrieben. Dann folgen noch Angaben zu Messpunkt, Lenkwert, Geschwindigkeit und Zeit. In der nächsten Zeile wird der Mittelpunkt mit einem Strich markiert. Es ergibt sich eine Textausgabe die zum Beispiel so aussieht:

```
■          x = 245.0 ;steer = 0.8 ;speed = 85.9 ;time = 00:00.00
■          |
■          x = 265.0 ;steer = 0.8 ;speed = 89.7 ;time = 00:00.04
■          |
■          x = 284.0 ;steer = 0.9 ;speed = 93.2 ;time = 00:00.08
■          |
■          x = 292.0 ;steer = 0.9 ;speed = 94.8 ;time = 00:00.12
■          |
■          x = 304.0 ;steer = 1.0 ;speed = 97.0 ;time = 00:00.16
■          |
■          x = 312.0 ;steer = 1.0 ;speed = 98.5 ;time = 00:00.20
■          |
■          x = 325.0 ;steer = 1.0 ;speed = 99.1 ;time = 00:00.24
■          |
```

8.1.5 lenken

```

1 def lenken(steer, speed):
2     if steer > 2:
3         steer = 2
4     elif steer < 0:
5         steer = 0
6     if speed > 100:
7         speed = 100
8     elif speed < 0:
9         speed = 0
10
11    speedHead = (100 - speed)
12
13    if speedHead > speed:
14        speedHead = speed
15
16    if steer == 1:
17        pr.ChangeDutyCycle(speed) # rechte Motoren
18        pl.ChangeDutyCycle(speed) # linke Motoren
19    elif steer < 1:
20        pr.ChangeDutyCycle(((1 - steer) * speedHead + speed))
21        pl.ChangeDutyCycle(steer * speed)
22    elif steer > 1:
23        steer = 2 - steer # steer auf 0-1 normieren
24        pr.ChangeDutyCycle((steer * speed))
25        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed))
26

```

..../main.py

Die *lenken* Funktion steuert das Tastverhältnis der Motoren. Übergeben werden zwei Parameter für die Lenkgewichtung und die Geschwindigkeit. Am Anfang der Funktion werden einige Abfragen zur Fehlerverhütung vorgenommen, damit das Tastverhältnis nicht unter 0% oder über 100% liegt, dass würde sonst zu einem Absturz führen.

Damit das Auto in den Kurven nicht ungewollt langsamer fährt, wird ausgerechnet wie viel schneller sich die äußereren Motoren drehen können.

$$\text{Kopfraum} = 100 - \text{Geschwindigkeit} \quad (3)$$

Ist der Kopfraum größer als der Geschwindigkeitswert, wird der Kopfraum gleich der Geschwindigkeit gesetzt.

Die Tastverhältnisse der Motoren werden gleich dem Geschwindigkeitswert gesetzt, wenn der Lenkwert *steer* = 1 ist. Sonst wird unterschieden ob der Lenkwert größer oder kleiner als 1 ist, um in die entsprechende Richtung zu lenken.

$$\text{Innere Motoren: } \text{Tastverhältnis} = \text{Lenkwert} \cdot \text{Geschwindigkeit} \quad (4)$$

$$\text{Äußere Motoren: } \text{Tastverhältnis} = (1 - \text{Lenkwert}) \cdot \text{Kopfraum} + \text{Geschwindigkeit} \quad (5)$$

Die Kombination aus dem Lenkwert, Geschwindigkeitswert und Kopfraum ermöglicht ein stufenloses kurven fahren, ohne das an Geschwindigkeit verloren wird. Dabei ist es egal worauf die Werte basieren, es wird immer gleich sanft gelenkt.

Tabelle 1: Beispielwerte für die Motorensteuerung

Lenken	Geschwindigkeit	$T_{\text{innen}} [\%]$	$T_{\text{außen}} [\%]$
0,2	20	2	36
	60	12	92
	80	16	96
0,8	20	16	24
	60	48	68
	80	64	84

8.1.6 checkgreen

```

1 def checkgreen():
2     global img, ret
3
4     # Take each frame
5     ret, img = cap.read()
6
7     # Convert BGR to HSV
8     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
9
10    lower_green = np.array([45,200,200])
11    upper_green = np.array([80,255,255])
12
13    mask = cv2.inRange(hsv, lower_green, upper_green)
14    no_green = cv2.countNonZero(mask)
15
16    return no_green

```

..../main.py

Die *checkgreen* Funktion Analysiert ein ganzes Bild der Webcam auf ihren Grünanteil und gibt die Anzahl zurück.

8.1.7 checkblue

```
def checkblue(delay, run_event):
    global img, ret
    time.sleep(1)

    while run_event.is_set():
        # Convert BGR to HSV
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        lower_blue = np.array([90,100,255])
        upper_blue = np.array([100,255,255])

        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        no_blue = cv2.countNonZero(mask)
        if no_blue > 500:
            print("Blaue Ampel, warte 1,5 sekunden...")
            time.sleep(1.5)
            bremsen()
            print("STOP!!! Rennen fertig")
            run_event.clear()

    time.sleep(delay)
```

..../main.py

checkblue funktioniert ähnlich wie *checkgreen*, allerdings läuft die Funktion in Dauerschleife und stoppt das Auto 1,5 Sekunden nachdem die blaue Ampel erkannt wurde. Die Funktion nimmt dabei kein eigenes Bild auf, sondern verwendet das Bild, dass von der *linienfahren* Funktion aufgenommen wurde. Beim starten des threads wartet die Funktion eine Sekunde, damit die grüne Ampel nicht fälschlicher weise als Stoppsignal erkannt wird.

8.1.8 makevideo

```
1 def makevideo(delay, run_event):
2     global img, ret, out
3     global x, minutes, seconds
4
5     minutes = 0
6     seconds = 0
7     font = cv2.FONT_HERSHEY_SIMPLEX
8     x = 320
9     capture_video = True
10
11    while run_event.is_set():
12        if ret==True and capture_video == True:
13            cv2.line(img,(int(x)-1,70),(int(x)+1,70),(255,0,0),5)
14            cv2.putText(img,'{:0>2}:{:05.2f}'.format(int(minutes),seconds),(10,470),font,
15            2,(255,255,255),2,cv2.LINE_AA)
16            cv2.putText(img,"%0.f" % x,(int(x)-30,100),font,1,(255,255,255),2,cv2.LINE_AA)
17            out.write(img)
18            time.sleep(delay)
```

..../main.py

Die Aufgabe der *makevideo* Funktion ist es die von der Kamera aufgenommenen Bilder als ein Video zu exportieren. Da das Video nur zur Fehleranalyse und Veranschaulichung dient, wird es vernachlässigt ob das Video in Echtzeit läuft. Dafür wird die aktuelle Fahrzeit, ab dem erkennen der grünen Ampel, unten links im Bild eingeblendet. Außerdem wird der zu dem Bild gehörende Messpunkt und dessen Wert eingezeichnet.

Das Video in Zusammenhang mit den eingeblendeten Werten ermöglicht eine deutlich bessere Fehleranalyse als eine Textausgabe über die Konsole (Abb.: 3).



Abbildung 3: Kamerabild mit Zeitanzeige und Messpunkt, abgelenkt von einem roten Schuh

8.2 setup.py

```
1 import RPi.GPIO as GPIO # GPIO-Bibliothek importieren
2
3 GPIO.setmode(GPIO.BCM) # Verwende BCM-Pinnummern
4
5 # GPIO für Motoren
6 # Motor A
7 ENA = 10 # Enable Motor A
8 IN1 = 9 # In 1
9 IN2 = 11 # In 2
10 # Motor B
11 ENB = 22 # Enable Motor B
12 IN3 = 17 # In 3
13 IN4 = 27 # In 4
14
15 # GPIOs als Ausgang setzen
16 GPIO.setup(ENA, GPIO.OUT)
17 GPIO.setup(IN1, GPIO.OUT)
18 GPIO.setup(IN2, GPIO.OUT)
19 GPIO.setup(ENB, GPIO.OUT)
20 GPIO.setup(IN3, GPIO.OUT)
21 GPIO.setup(IN4, GPIO.OUT)
22
23 # PWM für Motor A und B
24 pr = GPIO.PWM(ENA, 73) # Motor A, Frequenz = 73 Hz
25 pl = GPIO.PWM(ENB, 73) # Motor B, Frequenz = 73 Hz
26
27 GPIO.output(IN1, 0) # Bremsen
28 GPIO.output(IN2, 0) # Bremsen
29 GPIO.output(IN3, 0) # Bremsen
30 GPIO.output(IN4, 0) # Bremsen
31
32 print("GPIO-Setup erfolgreich")
```

..../setup.py

In *setup.py* werden die GPIOs definiert, der Skript wird am Anfang von *main.py* aufgerufen. Verwendet wird der BCM Modus. Für die PWM wurde eine Frequenz von 73 Hz gewählt. Dadurch dass das GPIO-Setup ausgelagert war konnten alle Python Skripte während der ganzen Entwicklungsphase problemlos verwendet werden. Wenn die GPIOs sich ändern, muss dies nur in dieser einen Datei eingetragen werden.

8.3 aufraeumen.py

```
1 import RPi.GPIO as GPIO # GPIO-Bibliothek importieren
2 import time           # Modul time
3 from setup import *
4
5 def aufraeumen():
6     # Erst bremsen dann cleanup
7     GPIO.output(IN1, 0) # Bremsen
8     GPIO.output(IN2, 0) # Bremsen
9     GPIO.output(IN3, 0) # Bremsen
10    GPIO.output(IN4, 0) # Bremsen
11    time.sleep(.1)
12    GPIO.cleanup()      # Aufräumen
13    print("GPIOs aufgeräumt")
14
15 def bremsen():
16     GPIO.output(IN1, 0) # Bremsen
17     GPIO.output(IN2, 0) # Bremsen
18     GPIO.output(IN3, 0) # Bremsen
19     GPIO.output(IN4, 0) # Bremsen
20
21 def losfahren():
22     GPIO.output(IN1, 1)      # Motor A Rechtslauf
23     GPIO.output(IN2, 0)      # Motor A Rechtslauf
24     GPIO.output(IN3, 1)      # Motor B Rechtslauf
25     GPIO.output(IN4, 0)      # Motor B Rechtslauf
```

..../aufraeumen.py

In *aufraeumen.py* sind drei Funktionen ausgelagert. *losfahren* setzt die vier GPIOs der Motoren auf die entsprechenden Werte zum Vorwärtsfahren, *bremsen* setzt die GPIOs, zum stoppen des Autos, auf null und *aufraeumen* stoppt das Auto und bereinigt die GPIO Einstellungen.

9 3D-Druck Halterungen

9.1 Kamerahalterung

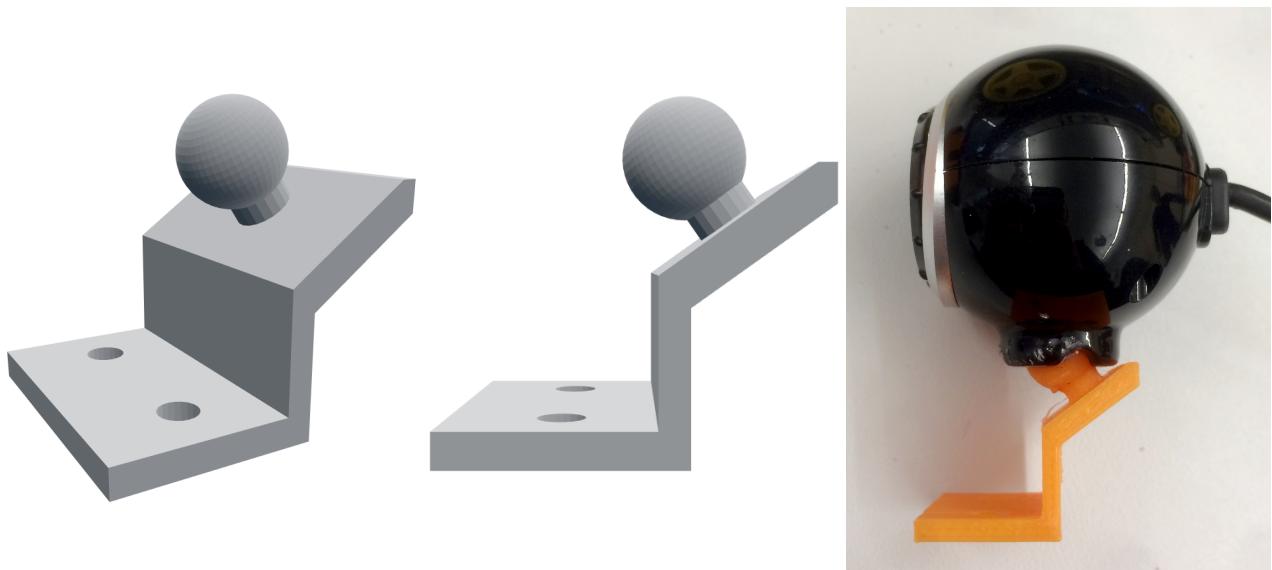


Abbildung 4: Kamerahalterung 3D-Modell und fertiger 3D-Druck

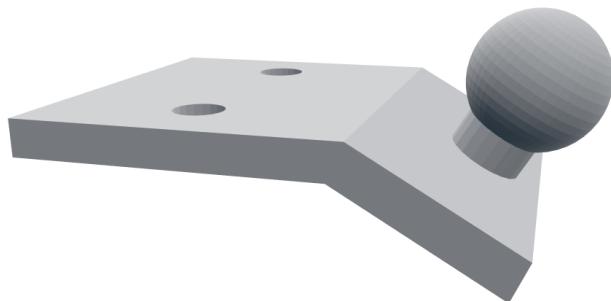


Abbildung 5: 3D-Modell der ersten Kamerahalterung

9.2 Ultraschallsensorhalterung

Für den Ultraschallsensor HC-SR04 wurde eine Halterung entworfen die es ermöglicht den Sensor sowohl aufrecht als auch Kopfüber an dem Auto zu befestigen. Außerdem sollte der Sensor, wie alle Teile des Autos einfach abnehmbar sein, falls Änderungen vorgenommen werden müssen. Entworfen wurde das Modell in Blender 2.79, mit zwei Löchern für Schrauben, Platz für weitere Bohrungen und einen Schlitz für die Steckverbindung.

Der 3D-Druck ging problemlos und die Halterung erfüllte die Anforderungen, es stellte sich aber heraus, dass die Maße der Sensoren teilweise zu stark voneinander abweichen, deswegen wurde das Modell noch einmal mit mehr Spielraum überarbeitet.

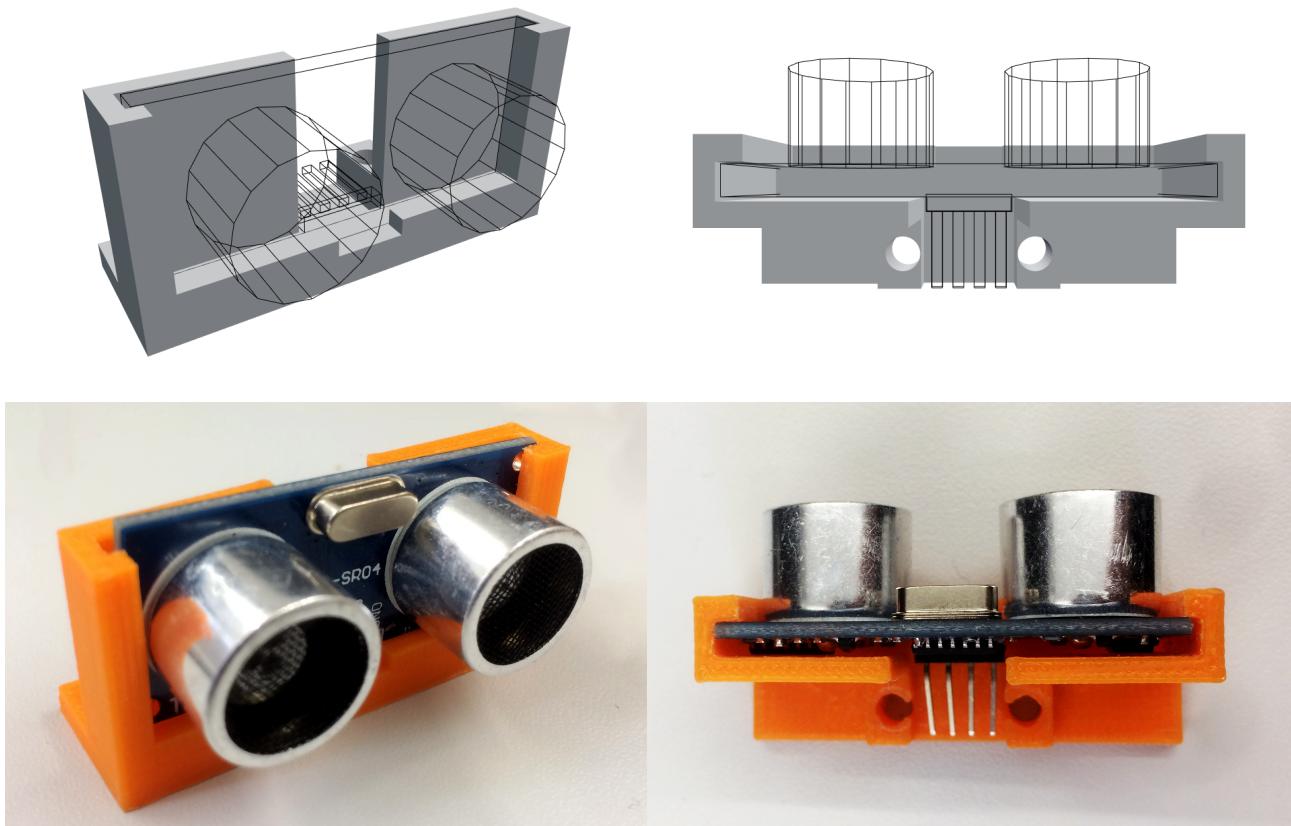


Abbildung 6: Ultraschallsensorhalterung 3D-Modell und fertiger 3D-Druck

10 Geplante Weiterentwicklungen

Es sind diverse Weiterentwicklungen an dem Auto geplant, die bisher nicht verwirklicht wurden. Ein Plan ist es die Videoausgabe deutlich auszubauen. So können mehr Messwerte angezeigt werden, wie zum Beispiel die Lenk- und Geschwindigkeitswerte oder die Drehzahl der Motoren und damit die Zurückgelegte Strecke. Optimal wäre die Ausgabe eines Livestreams, was allerdings nur möglich ist, wenn dadurch der Fahralgorithmus nicht verlangsamt wird. Zudem war eine Anzeige der aktuellen Messwerte über LEDs in Entwicklung.

Der Aufbau der Bauteile auf dem Auto soll noch kompakter werden, in dem Platzsparender gelötet wird. Dadurch wären alle Platinen auf der unteren Platte platziert, so dass nur noch der Raspberry Pi, der Akku und die Kamera zu sehen sind.

11 Anhang

11.1 main.py

```
1 import threading      # Modul threads
2 import cv2           # Dies ist die Bildverarbeitungsbibliothek OpenCV
3 import numpy as np   # Rechnen mit vielen Zahlen in einem Array (z. B. Bilder)
4 import math          # Modul math
5 import time          # Modul time

7 from aufraeumen import aufraeumen, losfahren, bremsen # Funktion für start/stop importieren
from setup import * # GPIO Setup importieren und ausführen
9
10 cap = cv2.VideoCapture(0) # Input 0
# Codec und VideoWriter object für Video Output
11 fourcc = cv2.VideoWriter_fourcc(*'XVID')
12 out = cv2.VideoWriter('output.avi',fourcc, 15, (640,480))
ret, img = cap.read()

15 # Video erstellen
16 def makevideo(delay, run_event):
    global img, ret, out
    global x, minutes, seconds

21     minutes = 0
22     seconds = 0
23     font = cv2.FONT_HERSHEY_SIMPLEX
24     x = 320
25     capture_video = True

27     while run_event.is_set():
28         if ret==True and capture_video == True:
29             cv2.line(img,(int(x)-1,70),(int(x)+1,70),(255,0,0),5)
30             cv2.putText(img, '{:0>2}:{:05.2f}'.format(int(minutes),seconds),(10,470), font,
31             2,(255,255,255),2,cv2.LINE_AA)
32             cv2.putText(img, "%0.f" % x,(int(x)-30,100), font, 1,(255,255,255),2,cv2.LINE_AA)
33             out.write(img)
34             time.sleep(delay)

35 # Schauen, ob Ampel grün ist
36 def checkgreen():
37     global img, ret

39     # Take each frame
40     ret, img = cap.read()

41     # Convert BGR to HSV
42     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

45     lower_green = np.array([45,200,200])
46     upper_green = np.array([80,255,255])

47     mask = cv2.inRange(hsv, lower_green, upper_green)
48     no_green = cv2.countNonZero(mask)
49     return no_green

51 # Schauen, ob Ampel blau ist
52 def checkblue(delay, run_event):
    global img, ret
```

```

55     time.sleep(1)

57     while run_event.is_set():
58         # Convert BGR to HSV
59         hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

61         lower_blue = np.array([90,100,255])
62         upper_blue = np.array([100,255,255])

63         mask = cv2.inRange(hsv, lower_blue, upper_blue)
64         no_blue = cv2.countNonZero(mask)
65         if no_blue > 500:
66             print("Blaue Ampel, warte 1,5 sekunden...")
67             time.sleep(1.5)
68             bremsen()
69             print("STOP!!! Rennen fertig")
70             run_event.clear()

72         time.sleep(delay)

74 # Motoren lenken
75 def lenken(steer, speed):
76     if steer > 2:
77         steer = 2
78     elif steer < 0:
79         steer = 0
80     if speed > 100:
81         speed = 100
82     elif speed < 0:
83         speed = 0

85     speedHead = (100 - speed)
86
87     if speedHead > speed:
88         speedHead = speed

89     if steer == 1:
90         pr.ChangeDutyCycle(speed) # rechte Motoren
91         pl.ChangeDutyCycle(speed) # linke Motoren
92     elif steer < 1:
93         pr.ChangeDutyCycle(((1 - steer) * speedHead + speed))
94         pl.ChangeDutyCycle(steer * speed)
95     elif steer > 1:
96         steer = 2 - steer # steer auf 0-1 normieren
97         pr.ChangeDutyCycle((steer * speed))
98         pl.ChangeDutyCycle(((1 - steer) * speedHead + speed))
99
100    return

102 # Bild machen und Zeile auslesen
103 def line(zeileNr):
104     global img, ret
105     ret, img = cap.read()

106     img_r = img[zeileNr, :, 2] # Alles aus zeileNr und Breite und Farbkanal 2
107     img_g = img[zeileNr, :, 1]
108     img_b = img[zeileNr, :, 0]

109     zeile_bin = (img_r.astype('int16') - (img_g / 2 + img_b / 2)) > 60 # Rotanteil über
110     Threshold
111
112
113

```

```

# Mittelpunkt berechnen und return:
115 if zeile_bin.sum() != 0:
    x = np.arange(zeile_bin.shape[0]) # x=0,1,2 ... N-1 (N=Anzahl von Werten in zeile_bin)
)
117     return (zeile_bin * x).sum() / zeile_bin.sum()
else:
119     return None

121 # Linie analysieren
122 def linienfahren(delay, run_event):
123     global cap
124     global x, minutes, seconds
125
126     ret, img = cap.read()
127     width = np.size(img, 1)
128     ideal = width/2
129     mitte = ideal
130     last_mitte = mitte
131     steer = 1
132     startzeit = time.time()
133
134     while run_event.is_set():
135         last_mitte = mitte
136         mitte = line(70)
137
138         if mitte is None:
139             if last_mitte > ideal:
140                 mitte = 640
141             else:
142                 mitte = 0
143         x = mitte
144
145         if mitte == ideal:
146             steer = 1
147         elif mitte < ideal:
148             steer = (mitte/ideal)
149             speed = steer*60+40
150             steer = steer*.9+.1
151         elif mitte > ideal:
152             steer = (width-mitte)/ideal
153             speed = steer*60+40
154             steer = 2-(steer*.9+.1)
155
156         lenken(steer, speed)
157
158         hours, rem = divmod(time.time()-startzeit, 3600)
159         minutes, seconds = divmod(rem, 60)
160
161         print(" " * int(mitte/10), "■", " " * int(64 - mitte/10), "x = %.1f" % mitte, ";steer = %.1f" % steer, ";speed = %.1f" % speed, ";time = {:0>2}:{:05.2f}".format(int(minutes), seconds))
162         print(" " * int(ideal/10), "|")
163
164         time.sleep(delay)
165
166 # Programm starten
167 def main():
168     losfahren()
169     pr.start(0) # Motor A, speed Tastverhältnis
170     pl.start(0) # Motor B, speed Tastverhältnis

```

```

171
172     run_event = threading.Event()
173     run_event.set()

175     th1_delay = .01    # sleep dauer der Funktion
176     th2_delay = .01    # sleep dauer der Funktion
177     th3_delay = .001   # sleep dauer der Funktion
178     th1 = threading.Thread(target=linienfahren, args=(th1_delay, run_event)) # Funktion in
179     # einem neuen Thread zuordnen
180     th2 = threading.Thread(target=checkblue, args=(th2_delay, run_event))      # Funktion in
181     # einem neuen Thread zuordnen
182     th3 = threading.Thread(target=makevideo, args=(th3_delay, run_event))      # Funktion in
183     # einem neuen Thread zuordnen

184
185     th3.start()  # Video Thread starten
186
187     no_green = 0
188     print("Warten auf grüne Ampel")
189
190     while no_green < 500:
191         no_green = checkgreen()
192
193     th1.start()  # Linienfahren Thread starten
194     th2.start()  # Ampel blau Test Thread starten
195
196     # Warten bis Strg+C gedrückt wird:
197     try:
198         while 1:
199             time.sleep(.01)
200
201     except KeyboardInterrupt:
202         print("attempting to close threads. Max wait =", max(th1_delay, th2_delay, th3_delay))
203         #
204         bremsen()
205         run_event.clear()
206         th1.join()
207         print("Thread 1 closed")
208         th2.join()
209         print("Thread 2 closed")
210         th3.join()
211         print("Thread 3 closed")
212         aufraeumen()
213         print("Threads successfully closed")
214
215         cap.release()
216         out.release()

```

..../main.py