

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

<b>1</b>	<b>Code</b>	<b>3</b>
1.1	main.py . . . . .	3
1.1.1	import . . . . .	3
1.1.2	main . . . . .	4
1.1.3	line . . . . .	6
1.1.4	linienfahren . . . . .	7
1.1.5	lenken . . . . .	9
1.1.6	checkgreen . . . . .	10
1.1.7	checkblue . . . . .	11
1.1.8	makevideo . . . . .	12
1.2	setup.py . . . . .	13
1.3	aufraeumen.py . . . . .	14
<b>2</b>	<b>3D-Druck Halterungen</b>	<b>15</b>
2.1	Kamerahalterung . . . . .	15
2.2	Sensorhalterung . . . . .	15
<b>3</b>	<b>Anhang</b>	<b>16</b>
3.1	main.py . . . . .	16

# 1 Code

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

## 1.1 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 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.

### 1.1.1 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 aufrauem import aufrauem, 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 *aufrauem.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.

### 1.1.2 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.

### 1.1.3 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.

#### 1.1.4 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
■          |
```

### 1.1.5 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

### 1.1.6 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.

### 1.1.7 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.

### 1.1.8 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.: 1).

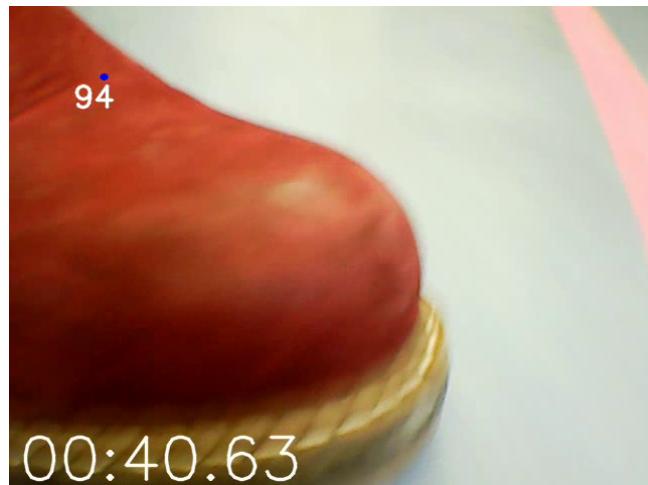


Abbildung 1: Kamerabild mit Zeitanzeige und Messpunkt, abgelenkt von einem roten Schuh

## 1.2 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 und wird am Anfang von *main.py* aufgerufen. Verwendet wird der BCM Modus. Für die PWM wurde eine Frequenz von 73 Hz gewählt.

## 1.3 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.

## **2 3D-Druck Halterungen**

### **2.1 Kamerahalterung**



Abbildung 2: Kamerahalterung

### **2.2 Sensorhalterung**

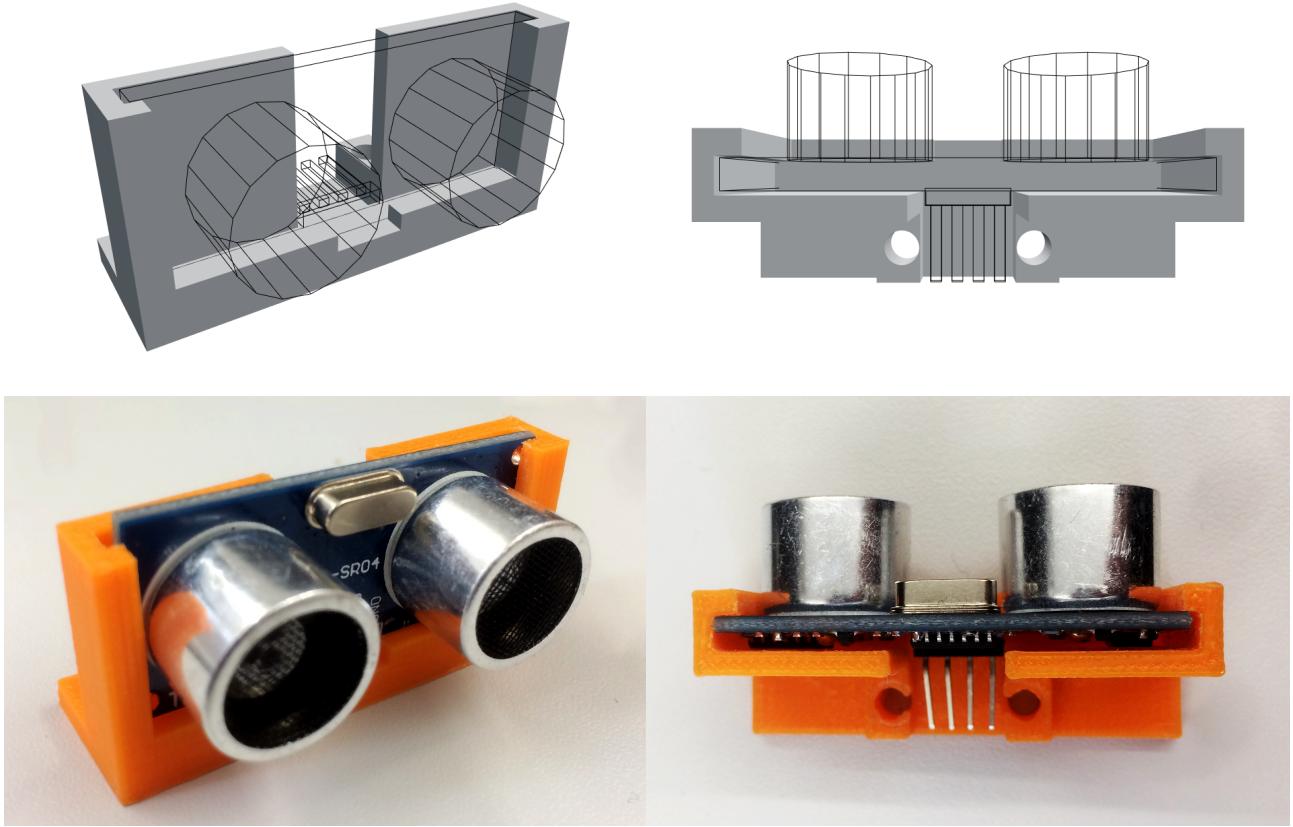


Abbildung 3: Sensorhalterung 3D-Modell und 3D-Druck

### 3 Anhang

#### 3.1 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
8 from setup import * # GPIO Setup importieren und ausführen

9 cap = cv2.VideoCapture(0) # Input 0
10 # Codec und VideoWriter object für Video Output
11 fourcc = cv2.VideoWriter_fourcc(*'XVID')
12 out = cv2.VideoWriter('output.avi',fourcc, 15, (640,480))
13 ret, img = cap.read()

15 # Video erstellen
16 def makevideo(delay, run_event):
17     global img, ret, out
18     global x, minutes, seconds
19
20     minutes = 0
21     seconds = 0

```

```

23 font = cv2.FONT_HERSHEY_SIMPLEX
24 x = 320
25 capture_video = True
26
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, "%0f" % x,( int(x)-30,100), font , 1,(255,255,255),2,cv2.LINE_AA)
33         out.write(img)
34         time.sleep(delay)
35
36 # Schauen, ob Ampel grün ist
37 def checkgreen():
38     global img, ret
39
40     # Take each frame
41     ret , img = cap.read()
42
43     # Convert BGR to HSV
44     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
45
46     lower_green = np.array([45,200,200])
47     upper_green = np.array([80,255,255])
48
49     mask = cv2.inRange(hsv, lower_green, upper_green)
50     no_green = cv2.countNonZero(mask)
51
52     return no_green
53
54 # Schauen, ob Ampel blau ist
55 def checkblue(delay , run_event):
56     global img, ret
57     time.sleep(1)
58
59     while run_event.is_set():
60         # Convert BGR to HSV
61         hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
62
63         lower_blue = np.array([90,100,255])
64         upper_blue = np.array([100,255,255])
65
66         mask = cv2.inRange(hsv, lower_blue, upper_blue)
67         no_blue = cv2.countNonZero(mask)
68         if no_blue > 500:
69             print("Blaue Ampel, warte 1,5 sekunden...")
70             time.sleep(1.5)
71             bremsen()
72             print("STOP!!! Rennen fertig")
73             run_event.clear()
74
75         time.sleep(delay)
76
77 # Motoren lenken
78 def lenken(steer , speed):
79     if steer > 2:
80         steer = 2
81     elif steer < 0:
82         steer = 0
83     if speed > 100:
84

```

```

    speed = 100
83  elif speed < 0:
     speed = 0
85
speedHead = (100 - speed)
87
if speedHead > speed:
    speedHead = speed
89
91 if steer == 1:
    pr.ChangeDutyCycle(speed) # rechte Motoren
93  pl.ChangeDutyCycle(speed) # linke Motoren
95  elif steer < 1:
    pr.ChangeDutyCycle(((1 - steer) * speedHead + speed))
    pl.ChangeDutyCycle(steer * speed)
97  elif steer > 1:
    steer = 2 - steer # steer auf 0-1 normieren
99  pr.ChangeDutyCycle((steer * speed))
    pl.ChangeDutyCycle(((1 - steer) * speedHead + speed))
101 return
103 # Bild machen und Zeile auslesen
def line(zeileNr):
105  global img, ret
    ret, img = cap.read()
107
img_r = img[zeileNr, :, 2] # Alles aus zeileNr und Breite und Farbkanal 2
109 img_g = img[zeileNr, :, 1]
img_b = img[zeileNr, :, 0]
111
zeile_bin = (img_r.astype('int16') - (img_g / 2 + img_b / 2)) > 60 # Rotanteil über
Threshold
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:
    return None
119
121 # Linie analysieren
def linienfahren(delay, run_event):
123  global cap
    global x, minutes, seconds
125
ret, img = cap.read()
127 width = np.size(img, 1)
ideal = width/2
129 mitte = ideal
last_mitte = mitte
131 steer = 1
startzeit = time.time()
133
while run_event.is_set():
135  last_mitte = mitte
    mitte = line(70)
137
if mitte is None:
    if last_mitte > ideal:
139

```

```

141         mitte = 640
142     else:
143         mitte = 0
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),
162     seconds))
163     print(" " * int(ideal/10), "|")
164
165     time.sleep(delay)
166
167 # Programm starten
168 def main():
169     losfahren()
170     pr.start(0) # Motor A, speed Tastverhältnis
171     pl.start(0) # Motor B, speed Tastverhältnis
172
173     run_event = threading.Event()
174     run_event.set()
175
176     th1_delay = .01 # sleep dauer der Funktion
177     th2_delay = .01 # sleep dauer der Funktion
178     th3_delay = .001 # sleep dauer der Funktion
179     th1 = threading.Thread(target=linienfahren, args=(th1_delay, run_event)) # Funktion in
180     einem neuen Thread zuordnen
181     th2 = threading.Thread(target=checkblue, args=(th2_delay, run_event)) # Funktion in
182     einem neuen Thread zuordnen
183     th3 = threading.Thread(target=makevideo, args=(th3_delay, run_event)) # Funktion in
184     einem neuen Thread zuordnen
185
186     th3.start() # Video Thread starten
187
188     no_green = 0
189     print("Warten auf grüne Ampel")
190
191     while no_green < 500:
192         no_green = checkgreen()
193
194     th1.start() # Linienfahren Thread starten
195     th2.start() # Ampel blau Test Thread starten
196
197     # Warten bis Strg+C gedrückt wird:
198     try:

```

```

195     while 1:
196         time.sleep(.01)
197
198     except KeyboardInterrupt:
199         print("attempting to close threads. Max wait =", max(th1_delay, th2_delay, th3_delay))
200         #
201         bremsen()
202         run_event.clear()
203         th1.join()
204         print("Thread 1 closed")
205         th2.join()
206         print("Thread 2 closed")
207         th3.join()
208         print("Thread 3 closed")
209         aufraeumen()
210         print("Threads successfully closed")
211
212         cap.release()
213         out.release()
214
215 if __name__ == '__main__':
216     main()

```

..../main.py