

1 import

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

7 from aufräumen import aufräumen, losfahren, bremsen # Funktion für
    KeyboardInterrupt importieren
from setup import *     # GPIO Setup importieren und ausführen

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

../main.py

Am Anfang der Hauptdatei werden die benötigten Module importiert, gefolgt von den beiden ausgelagerten Dateien. Von aufräumen.py werden nur die benötigten Funktionen importiert, 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.

2 makevideo

```
def makevideo(delay, run_event):
2     global img, ret, out
    global x, minutes, seconds

4
    minutes = 0
6     seconds = 0
    font = cv2.FONT_HERSHEY_SIMPLEX
8     x = 320
    capture_video = True

10
    while run_event.is_set():
12         if ret==True and capture_video == True:
            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, 2,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(img, "%0f" % x,(int(x)-30,100), font, 1,(255,255,255),2,
cv2.LINE_AA)
16             out.write(img)
```

```
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

3 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    return no_green
```

../main.py

4 checkblue

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

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

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

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

20      time.sleep(delay)
```

../main.py

5 lenken

```
1 def lenken(steer, speed):
   if steer > 2:
3       steer = 2
   elif steer < 0:
5       steer = 0
   if speed > 100:
7       speed = 100
   elif speed < 0:
9       speed = 0

11      speedHead = (100 - speed)

13      if speedHead > speed:
```

```

    speedHead = speed

15
    if steer == 1:
17        pr.ChangeDutyCycle(speed) #
        pl.ChangeDutyCycle(speed) #
19    elif steer < 1:
        pr.ChangeDutyCycle(((1 - steer) * speedHead + speed)) #
21        pl.ChangeDutyCycle(steer * speed) #
    elif steer > 1:
23        steer = 2 - steer
        pr.ChangeDutyCycle((steer * speed)) #
25        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed)) #
    return

```

../main.py

6 line

```

1 def line(zeileNr):
    global img, ret
3    ret, img = cap.read()

    img_red = img[zeileNr, :, 2] # Alles aus der Dimension Höhe und Breite
    (:,:) und den Farbkanal 2
    img_green = img[zeileNr, :, 1]
7    img_blue = img[zeileNr, :, 0]

    zeile_bin = (img_red.astype('int16') - (img_green / 2 + img_blue / 2)) > 60

11    # Mittelpunkt berechnen:
    if zeile_bin.sum() != 0:
13        x = np.arange(zeile_bin.shape[0]) # x=0,1,2 ... N-1 (N=Anzahl von
        Werten in zeile400_bin)
        return (zeile_bin * x).sum() / zeile_bin.sum()
15    else:
        return None

```

../main.py

7 linienfahren

```

1 def linienfahren(delay, run_event):
    global cap
3    global x, minutes, seconds

```

```

5     ret, img = cap.read()
    width = np.size(img, 1)
7     ideal = width/2
    mitte = ideal
9     last_mitte = mitte
    steer = 1
11    startzeit = time.time()

13    while run_event.is_set():
        last_mitte = mitte
15        mitte = line(70)

17        if mitte is None:
            if last_mitte > ideal:
19                mitte = 640
            else:
21                mitte = 0
        x = mitte

23
25        if mitte == ideal:
            steer = 1
27        elif mitte < ideal:
            steer = (mitte/ideal)
            speed = steer*60+40
            steer = steer*.9+.1
29        elif mitte > ideal:
            steer = (width-mitte)/ideal
            speed = steer*60+40
31            steer = 2-(steer*.9+.1)

33
35    lenken(steer, speed)

37    hours, rem = divmod(time.time()-startzeit, 3600)
    minutes, seconds = divmod(rem, 60)
39

41    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))
    print(" " * int(ideal/10), "|")

43    time.sleep(delay)

```

../main.py

8 main

```

def main():
    global speed

    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() # Ampel grün Test 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")

```

```

44         aufräumen()
         print("Threads successfully closed")

46
48         cap.release()
         out.release()

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

```

../main.py

9 main.py

```

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

7 from aufräumen import aufräumen, losfahren, bremsen # Funktion für
    KeyboardInterrupt importieren
from setup import *    # GPIO Setup importieren und ausführen

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

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

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

27     while run_event.is_set():
        if ret==True and capture_video == True:
29             cv2.line(img,(int(x)-1,70),(int(x)+1,70),(255,0,0),5)
            cv2.putText(img, '{:0>2}:{:05.2f}'.format(int(minutes),seconds)
,(10,470), font, 2,(255,255,255),2,cv2.LINE_AA)

```

```

31         cv2.putText(img, "%.0f" % x, (int(x) - 30, 100), font, 1, (255, 255, 255), 2,
cv2.LINE_AA)
        out.write(img)
33         time.sleep(delay)

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

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

41

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

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

47

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

51

53 # Schauen, ob Ampel blau ist
def checkblue(delay, run_event):
    global img, ret
55     time.sleep(1)

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

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

63

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

73         time.sleep(delay)

75 # Motoren lenken
def lenken(steer, speed):

```



```

77     if steer > 2:
78         steer = 2
79     elif steer < 0:
80         steer = 0
81     if speed > 100:
82         speed = 100
83     elif speed < 0:
84         speed = 0
85
86     speedHead = (100 - speed)
87
88     if speedHead > speed:
89         speedHead = speed
90
91     if steer == 1:
92         pr.ChangeDutyCycle(speed) #
93         pl.ChangeDutyCycle(speed) #
94     elif steer < 1:
95         pr.ChangeDutyCycle(((1 - steer) * speedHead + speed)) #
96         pl.ChangeDutyCycle(steer * speed) #
97     elif steer > 1:
98         steer = 2 - steer
99         pr.ChangeDutyCycle((steer * speed)) #
100        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed)) #
101    return
102
103 # Bild machen und Zeile auslesen
104 def line(zeileNr):
105     global img, ret
106     ret, img = cap.read()
107
108     img_red = img[zeileNr, :, 2] # Alles aus der Dimension Höhe und Breite
109     # (:,:) und den Farbkanal 2
110     img_green = img[zeileNr, :, 1]
111     img_blue = img[zeileNr, :, 0]
112
113     zeile_bin = (img_red.astype('int16') - (img_green / 2 + img_blue / 2)) > 60
114
115     # Mittelpunkt berechnen:
116     if zeile_bin.sum() != 0:
117         x = np.arange(zeile_bin.shape[0]) # x=0,1,2 ... N-1 (N=Anzahl von
118         # Werten in zeile400_bin)
119         return (zeile_bin * x).sum() / zeile_bin.sum()
120     else:
121         return None
122
123 # Linie analysieren

```

```

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
# Programm starten

```

```

167 def main():
168     global speed
169
170     losfahren()
171     pr.start(0) # Motor A, speed Tastverhältnis
172     pl.start(0) # Motor B, speed Tastverhältnis
173
174     run_event = threading.Event()
175     run_event.set()
176
177     th1_delay = .01 # sleep dauer der Funktion
178     th2_delay = .01 # sleep dauer der Funktion
179     th3_delay = .001 # sleep dauer der Funktion
180     th1 = threading.Thread(target=linienfahren, args=(th1_delay, run_event)) #
181     Funktion in einem neuen Thread zuordnen
182     th2 = threading.Thread(target=checkblue, args=(th2_delay, run_event)) #
183     Funktion in einem neuen Thread zuordnen
184     th3 = threading.Thread(target=makevideo, args=(th3_delay, run_event)) #
185     Funktion in einem neuen Thread zuordnen
186
187     th3.start() # Ampel grün Test Thread starten
188
189     no_green = 0
190     print("Warten auf grüne Ampel")
191
192     while no_green < 500:
193         no_green = checkgreen()
194
195     th1.start() # Linienfahren Thread starten
196     th2.start() # Ampel blau Test Thread starten
197
198     # Warten bis Strg+C gedrückt wird:
199     try:
200         while 1:
201             time.sleep(.01)
202
203     except KeyboardInterrupt:
204         print("attempting to close threads. Max wait =", max(th1_delay,
205 th2_delay, th3_delay)) #
206         bremsen()
207         run_event.clear()
208         th1.join()
209         print("Thread 1 closed")
210         th2.join()
211         print("Thread 2 closed")
212         th3.join()
213         print("Thread 3 closed")

```

```

211         aufräumen()
        print("Threads successfully closed")

213         cap.release()
        out.release()

215
217 if __name__ == '__main__':
    main()

```

../main.py

10 setup.py

```

1 import RPi.GPIO as GPIO # GPIO-Bibliothek importieren

3 GPIO.setmode(GPIO.BCM) # Verwende BCM-Pinnummern

5 # GPIO für Motoren
# Motor A
7 ENA = 10 # Enable Motor A
  IN1 = 9  # In 1
9  IN2 = 11 # In 2
# Motor B
11 ENB = 22 # Enable Motor B
  IN3 = 17 # In 3
13 IN4 = 27 # In 4

15 # GPIOs als Ausgang setzen
  GPIO.setup(ENA, GPIO.OUT)
17  GPIO.setup(IN1, GPIO.OUT)
  GPIO.setup(IN2, GPIO.OUT)
19  GPIO.setup(ENB, GPIO.OUT)
  GPIO.setup(IN3, GPIO.OUT)
21  GPIO.setup(IN4, GPIO.OUT)

23 # PWM für Motor A und B
  pr = GPIO.PWM(ENA, 73) # Motor A, Frequenz = 73 Hz
25  pl = GPIO.PWM(ENB, 73) # Motor B, Frequenz = 73 Hz

27  GPIO.output(IN1, 0) # Bremsen
  GPIO.output(IN2, 0) # Bremsen
29  GPIO.output(IN3, 0) # Bremsen
  GPIO.output(IN4, 0) # Bremsen

31
print('GPIO-Setup erfolgreich')

```

11 aufräumen.py

```
import RPi.GPIO as GPIO # GPIO-Bibliothek importieren
2 import time            # Modul time
from setup import *
4
def aufräumen():
6     # Erst bremsen dann cleanup
    GPIO.output(IN1, 0) # Bremsen
8     GPIO.output(IN2, 0) # Bremsen
    GPIO.output(IN3, 0) # Bremsen
10    GPIO.output(IN4, 0) # Bremsen
    time.sleep(.1)
12    GPIO.cleanup()      # Aufräumen
    print("GPIOs aufgeräumt")
14
def bremsen():
16    GPIO.output(IN1, 0) # Bremsen
    GPIO.output(IN2, 0) # Bremsen
18    GPIO.output(IN3, 0) # Bremsen
    GPIO.output(IN4, 0) # Bremsen
20
def losfahren():
22    GPIO.output(IN1, 1)      # Motor A Rechtslauf
    GPIO.output(IN2, 0)      # Motor A Rechtslauf
24    GPIO.output(IN3, 1)      # Motor B Rechtslauf
    GPIO.output(IN4, 0)      # Motor B Rechtslauf
```