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 aufraeumen import aufraeumen, 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 aufrauemen.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.

Abbildung 1: Position Presets

3 checkgreen

```
def checkgreen():
    global img, ret

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

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

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

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

../main.py

4 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

5 lenken

```
def lenken(steer, speed):
    if steer > 2:
        steer = 2
    elif steer < 0:
        steer = 0
    if speed > 100:
        speed = 100
    elif speed < 0:
        speed = 0

    speedHead = (100 - speed)

    if speedHead > speed:
        speedHead = speed

    if steer == 1:
        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()

5     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]

9     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
```

```
    last_mitte = mitte
    steer = 1
    startzeit = time.time()

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

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

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

        lenken(steer, speed)

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

        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), "|")

        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")
        aufraeumen()
        print("Threads successfully closed")

        cap.release()
```

```
48          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 aufraeumen import aufraeumen, 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)
```

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

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

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

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

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

# Schauen, ob Ampel blau ist
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)

# Motoren lenken
def lenken(steer, speed):
    if steer > 2:
        steer = 2
    elif steer < 0:
        steer = 0
    if speed > 100:
```

```
        speed = 100
    elif speed < 0:
        speed = 0

    speedHead = (100 - speed)

    if speedHead > speed:
        speedHead = speed

    if steer == 1:
        pr.ChangeDutyCycle(speed)  #
        pl.ChangeDutyCycle(speed)  #
    elif steer < 1:
        pr.ChangeDutyCycle(((1 - steer) * speedHead + speed))  #
        pl.ChangeDutyCycle(steer * speed)  #
    elif steer > 1:
        steer = 2 - steer
        pr.ChangeDutyCycle((steer * speed))  #
        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed))  #
    return

# Bild machen und Zeile auslesen
def line(zeileNr):
    global img, ret
    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]
    img_blue = img[zeileNr, :, 0]

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

    # Mittelpunkt berechnen:
    if zeile_bin.sum() != 0:
        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()
    else:
        return None

# Linie analysieren
def linienfahren(delay, run_event):
    global cap
    global x, minutes, seconds

    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:
139             if last_mitte > ideal:
                    mitte = 640
141             else:
                    mitte = 0
143         x = mitte

145         if mitte == ideal:
                steer = 1
147         elif mitte < ideal:
                steer = (mitte/ideal)
149             speed = steer*60+40
                steer = steer*.9+.1
151         elif mitte > ideal:
                steer = (width-mitte)/ideal
153             speed = steer*60+40
                steer = 2-(steer*.9+.1)
155
            lenken(steer, speed)
157
            hours, rem = divmod(time.time()-startzeit, 3600)
159         minutes, seconds = divmod(rem, 60)

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))
            print(" " * int(ideal/10), "|")
163
            time.sleep(delay)
165
# Programm starten
167 def main():
        global speed
169
        losfahren()
171     pr.start(0)  # Motor A, speed Tastverhältnis
```

```
    pl.start(0)  # Motor B, speed Tastverhältnis
173
    run_event = threading.Event()
175 run_event.set()

177 th1_delay = .01   # sleep dauer der Funktion
    th2_delay = .01   # sleep dauer der Funktion
179 th3_delay = .001  # sleep dauer der Funktion
    th1 = threading.Thread(target=linienfahren, args=(th1_delay, run_event)) #
    Funktion in einem neuen Thread zuordnen
181 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
183
    th3.start()  # Ampel grün Test Thread starten
185
    no_green = 0
187 print("Warten auf grüne Ampel")

189 while no_green < 500:
        no_green = checkgreen()
191
    th1.start()  # Linienfahren Thread starten
193 th2.start()  # Ampel blau Test Thread starten

195 # Warten bis Strg+C gedrückt wird:
    try:
197     while 1:
            time.sleep(.01)
199
    except KeyboardInterrupt:
201     print("attempting to close threads. Max wait =", max(th1_delay,
    th2_delay, th3_delay))  #
        bremsen()
203     run_event.clear()
        th1.join()
205     print("Thread 1 closed")
        th2.join()
207     print("Thread 2 closed")
        th3.join()
209     print("Thread 3 closed")
        aufraeumen()
211     print("Threads successfully closed")

213     cap.release()
        out.release()
```

```
215
if __name__ == '__main__':
217     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")
```

../setup.py

11 aufraeumen.py

```
import RPi.GPIO as GPIO # GPIO-Bibliothek importieren
import time             # Modul time
from setup import *

def aufraeumen():
    # Erst bremsen dann cleanup
    GPIO.output(IN1, 0)  # Bremsen
    GPIO.output(IN2, 0)  # Bremsen
    GPIO.output(IN3, 0)  # Bremsen
    GPIO.output(IN4, 0)  # Bremsen
    time.sleep(.1)
    GPIO.cleanup()       # Aufräumen
    print("GPIOs aufgeräumt")

def bremsen():
    GPIO.output(IN1, 0)  # Bremsen
    GPIO.output(IN2, 0)  # Bremsen
    GPIO.output(IN3, 0)  # Bremsen
    GPIO.output(IN4, 0)  # Bremsen

def losfahren():
    GPIO.output(IN1, 1)      # Motor A Rechtslauf
    GPIO.output(IN2, 0)      # Motor A Rechtslauf
    GPIO.output(IN3, 1)      # Motor B Rechtslauf
    GPIO.output(IN4, 0)      # Motor B Rechtslauf
```

../aufraeumen.py