0.1 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 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.

0.2 main

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

6      run_event = threading.Event()
       run_event.set()
8
       th1_delay = .01   # sleep dauer der Funktion
10     th2_delay = .01   # sleep dauer der Funktion
       th3_delay = .001  # sleep dauer der Funktion
12     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
14     th3 = threading.Thread(target=makevideo, args=(th3_delay, run_event))    # Funktion in
       einem neuen Thread zuordnen

16     th3.start()  # Video Thread starten

18     no_green = 0
       print("Warten auf grüne Ampel")
20
       while no_green < 500:
22         no_green = checkgreen()

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

```
26
    # Warten bis Strg+C gedrückt wird:
28  try:
        while 1:
30          time.sleep(.01)

32  except KeyboardInterrupt:
        print("attempting to close threads. Max wait =", max(th1_delay, th2_delay, th3_delay))
      #
34      bremsen()
        run_event.clear()
36      th1.join()
        print("Thread 1 closed")
38      th2.join()
        print("Thread 2 closed")
40      th3.join()
        print("Thread 3 closed")
42      aufraeumen()
        print("Threads successfully closed")
44
        cap.release()
46      out.release()

48 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 wir 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. *target* ist 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-Algorithmus), *checkblue* (der Test ob die Ampel blau leuchtet) und *makevideo* (die Videoausgabe).

Der erste thread, die Videoaufzeichnung, wird gestartet. Dann folgt eine *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.

0.3 line

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

5      img_r = img[zeileNr, :, 2]  # Alles aus zeileNr und Breite und Farbkanal 2
       img_g = img[zeileNr, :, 1]
7      img_b = img[zeileNr, :, 0]

9      zeile_bin = (img_r.astype('int16') - (img_g / 2 + img_b / 2)) > 60 # Rotanteil über
       Threshold

11     # Mittelpunkt berechnen und return:
       if zeile_bin.sum() != 0:
13         x = np.arange(zeile_bin.shape[0])  # x=0,1,2 ... N-1 (N=Anzahl von Werten in zeile_bin
       )
           return (zeile_bin * x).sum() / zeile_bin.sum()
15     else:
           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.

0.4 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
        if mitte == ideal:
25          steer = 1
        elif mitte < ideal:
27          steer = (mitte/ideal)
            speed = steer*60+40
29          steer = steer*.9+.1
        elif mitte > ideal:
31          steer = (width-mitte)/ideal
            speed = steer*60+40
33          steer = 2-(steer*.9+.1)

35      lenken(steer, speed)

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

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 $2 - \textit{Lenkwert}$ 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:

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

$$\text{Geschwidigkeit} = \frac{\text{Messwert}}{\text{Bildmittelpunkt}} \cdot 60\% + 40\% \tag{2}$$

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 $\frac{Messwert}{10}$ Leerzeichen, dann ein Blockzeichen als Position des Messpunktes im Bild und die verbleibenden $64 - \frac{Messwert}{10}$ 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
```

0.5 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)
    elif steer < 1:
        pr.ChangeDutyCycle(((1 - steer) * speedHead + speed)) # rechte Motoren
        pl.ChangeDutyCycle(steer * speed) # linke Motoren
    elif steer > 1:
        steer = 2 - steer # steer auf 0-1 normieren
        pr.ChangeDutyCycle((steer * speed)) # rechte Motoren
        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed)) # linke Motoren
```

```
26      return
```

../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 äußeren Motoren drehen können.

$$\text{Kopfraum} = 100 - \text{Geschwindigkeit} \tag{3}$$

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.

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

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

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

0.6 checkgreen

```
1  def checkgreen():
       global img, ret
3
       # Take each frame
```

```
5       ret , img = cap.read()

7       # Convert BGR to HSV
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
9
        lower_green = np.array([45,200,200])
11      upper_green = np.array([80,255,255])

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

../main.py

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

0.7 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()
18              print("STOP!!! Rennen fertig")
                run_event.clear()
20
            time.sleep(delay)
```

../main.py

checkblue funktioniert ähnlich wie *checkblue*, 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 den *linienfahren* Funktion aufgenommen wurde. Beim starten des threads wird wartet die Funktion eine Sekunde, damit nicht die grüne Ampel fälschlicher weise als Stopsignal erkannt wird.

0.8 makevideo

```
1 def makevideo(delay, run_event):
      global img, ret, out
```

```
3       global x, minutes, seconds

5       minutes = 0
        seconds = 0
7       font = cv2.FONT_HERSHEY_SIMPLEX
        x = 320
9       capture_video = True

11      while run_event.is_set():
            if ret==True and capture_video == True:
13              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)
15              cv2.putText(img,"%.0f" % x,(int(x)-30,100), font, 1,(255,255,255),2,cv2.LINE_AA)
                out.write(img)
17              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

0.9 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
ENA = 10  # Enable Motor A
IN1 = 9   # In 1
IN2 = 11  # In 2
# Motor B
ENB = 22  # Enable Motor B
IN3 = 17  # In 3
IN4 = 27  # In 4

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

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

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

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.

0.10 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
```

```
24      GPIO.output(IN3, 1)       # Motor B Rechtslauf
        GPIO.output(IN4, 0)       # Motor B Rechtslauf
```

../aufraeumen.py

In *aufrauemen.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 *aufrauemen* stoppt das Auto und bereinigt die GPIO Einstellungen.

1 Anhang

1.1 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)

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

39     # Take each frame
       ret, img = cap.read()
41
```

```
    # 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)) # rechte Motoren
        pl.ChangeDutyCycle(steer * speed) # linke Motoren
    elif steer > 1:
        steer = 2 - steer # steer auf 0-1 normieren
        pr.ChangeDutyCycle((steer * speed)) # rechte Motoren
        pl.ChangeDutyCycle(((1 - steer) * speedHead + speed)) # linke Motoren
    return
```

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

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

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

    # Mittelpunkt berechnen und return:
    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
    )
        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()
    width = np.size(img, 1)
    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)
```

```
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():
    losfahren()
169     pr.start(0)  # Motor A, speed Tastverhältnis
    pl.start(0)  # Motor B, speed Tastverhältnis
171
    run_event = threading.Event()
173     run_event.set()

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

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

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

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

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

../main.py