Projektdokumentation

Technische Hochschule Wildau

Fachbereich Ingenieur- und Naturwissenschaften

Studiengang Automatisierungstechnik (B. Eng.)

Autonomes Fahren

Verfasser: Abraham Neme Alvarez

Gruppe: E

Dozent: Prof. Dr. Alexander Stolpmann

Erstellungsdatum: 16.06.2021

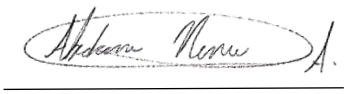
Fach: Bildverarbeitung

Dokumentenname: BV_AU18_Gruppe_E_Three_Wheeler.docx

I. Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der hier angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, Datum: Wildau, 16.06.2021



Unterschrift

Abraham Neme Alvarez

II. Abbildungsverzeichnis

III. Inhaltsverzeichnis

1. Aufgabenstellung

Mit Hilfe von OpenCV und Python ein Programm für den zur Verfügung gestellten Three-Wheeler erstellen, mit dem das Fahrzeug autonom in einer Fahrspur fährt. Das Fahrzeug sollte nicht die Fahrbahnränder überfahren und auch die Kurvenfahrt beherrschen. Zur Dokumentation gehört die Beschreibung der Software. Begründen Sie die Wahl Ihrer Lösung und beschreiben Sie Besonderheiten. Fügen Sie den kommentierten Programmcode mit bei. Zusatzpunkte gibt es, wenn die Software auf "STOP" (rot) und "GO" (grün) Schilder reagiert und entsprechend das Fahrzeug anhält oder fahren lässt.

2. Verwendeter Three-Wheeler

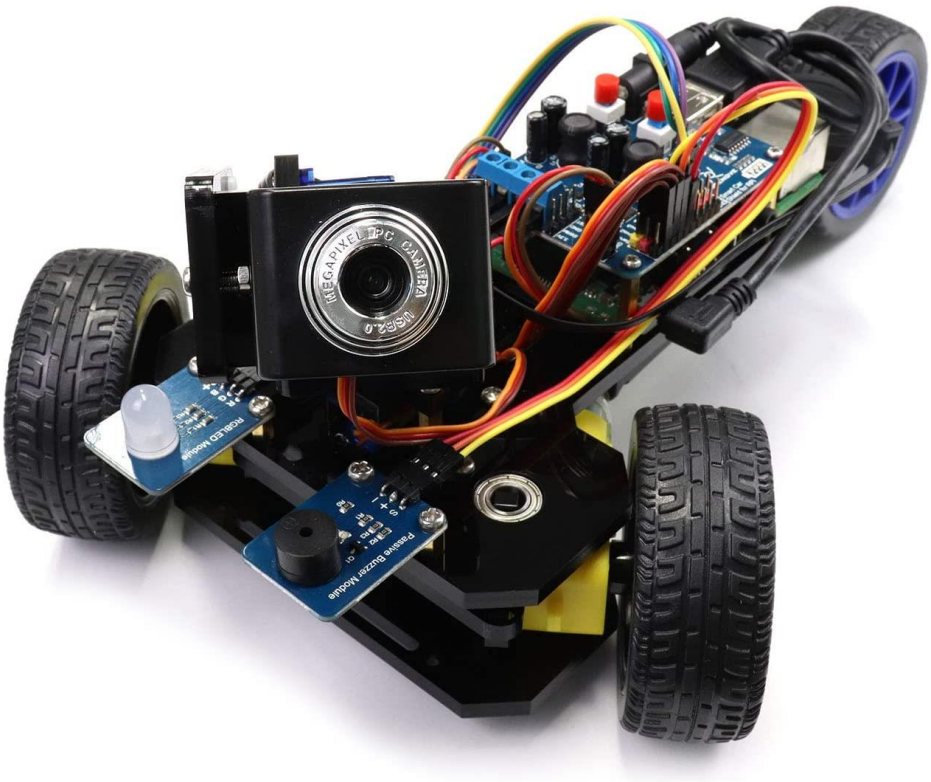


Abbildung 1 - Three Wheeler

3. Funktionen des Programms

3.1 Bibliotheken und Setup

- In der ersten 5 Zeilen werden alle verwendeten Bibliotheken importiert.
- Es wird an die RPi.GPIO-Bibliothek weitergegeben, welches Pin-Numbering-System verwendet wird.
- Folgendes wird ein Objekt aus der mDev-Datei erzeugt. Die roten, grünen und blauen LED´s werden eingeschaltet und die Position der Kamera wird eingestellt.
- Startet die Aufnahme von Bildern aus der Kamera. Einer Pixelhöhe, -breite und FPS werden vergeben und die Bilder werden in der Capture-Variablen gespeichert.

```
1  import cv2
2  import numpy as np
3  import RPi.GPIO as GPIO
4  import time
5  from mDev import *
6
7
8  GPIO.setmode(GPIO.BCM)
9
10 mdev = mDEV()
11 mdev.writeReg(mdev.CMD_IO1,1)
12 mdev.writeReg(mdev.CMD_IO2,1)
13 mdev.writeReg(mdev.CMD_IO3,1)
14 mdev.writeReg(mdev.CMD_SERVO2,1500)
15 mdev.writeReg(mdev.CMD_SERVO3,1250)
16
17
18 capture = cv2.VideoCapture(0)
19 capture.set(cv2.CAP_PROP_FPS,40)
20 capture.set(3,240)
21 capture.set(4,320)
22
23 time.sleep(3)
24
```

Abbildung 2 - Bibliotheken & Setup

3.2. Videofunktionen

3.2.1 getObjects

Diese Funktion erkennt Objekte, deren Farbe innerhalb eines voreingestellten Bereichs liegt.

- Bild wird vom BGR- in das HSV-Format geändert.
- Die inRange-Funktion zeichnet als weißen Pixeln, alle Objekte, die sich innerhalb einen Farbebereich befinden.
- Morphology, erode und dilate helfen, Störungen aus dem Bild zu entfernen.
- Das Bild wird mit Hilfe der maskingFrame-Funktion geschnitten.
- Cv2.findContours findet die Umrisse von erkennbaren Objekten im Bild.

```
48  def getObjects(frame, lH, lS, lV, hH, hS, hV):
49      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
50      LowerRegion = np.array([lH,lS,lV],np.uint8)
51      upperRegion = np.array([hH,hS,hV],np.uint8)
52      colorRange = cv2.inRange(hsv,LowerRegion,upperRegion)
53      kernel = np.ones((1,1),"uint8")
54      colorOb = cv2.morphologyEx(colorRange,cv2.MORPH_OPEN,kernel)
55      colorOb = cv2.erode(colorOb, kernel, iterations = 5)
56      colorOb = cv2.dilate(colorOb, kernel, iterations = 9)
57      croppedFrame = maskingFrame(colorOb)
58      img, contours, hierarchy = cv2.findContours(croppedFrame.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
59      return contours
60
```

Abbildung 3 - getObjects

Die getObjects-Funktion wird im Programm verwendet, um rote und grüne Objekte zu identifizieren. Wenn ein grünes Objekt erkannt wird, wird das Fahrzeug angewiesen, loszufahren. Im Gegensatz führt ein rotes Objekt zum Anhalten des Fahrzeugs.

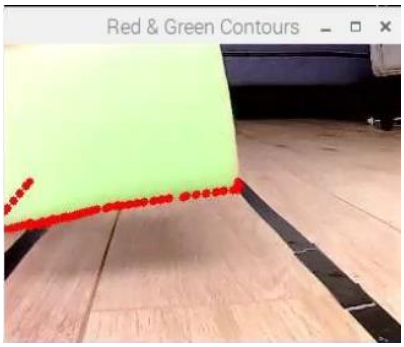


Abbildung 4 - Erkennung eines grünen Objektes

3.2.2 maskingFrame

diese Funktion grenzt den Bereich von Interesse für die Bildanalyse ab.

- Die vier Punkte des Bereiches werden in einem Array eingegeben.
- Ein schwarzes Bild wird erzeugt.
- Ein Weißes Trapez wird mit den Punkten auf dem schwarzen Bild erzeugt. Das normale Bild und das schwarze Bild werden zusammengeführt.

```
28
29  def maskingFrame(frame):
30      region = np.array([[20,110],[-160,320],[410,320],[280,110]], dtype=np.int32)
31      mask = np.zeros_like(frame)
32      cv2.fillPoly(mask, [region], 255)
33      maskedFrame = cv2.bitwise_and(frame, mask)
34      return maskedFrame
```

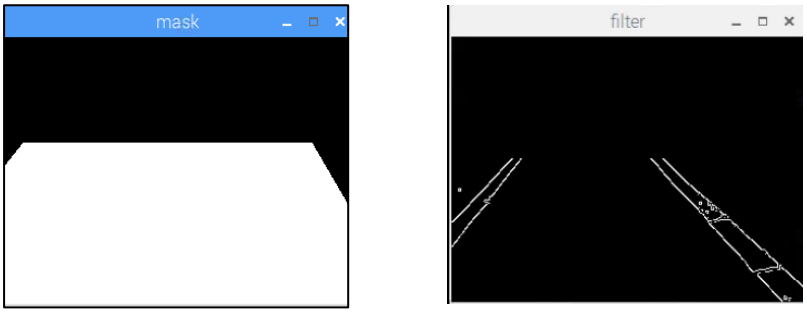


Abbildung 5 - maskingFrame

3.2.3 getLines

Die Funktion wendet eine Filterung auf die Farbintensität sowie einige Masken wie Canny an, um die Kanten der Straße zu identifizieren.

- Bild wird vom BGR- in das HSV-Format geändert.
- Die inRange-Funktion zeichnet als weißen Pixeln, alle Objekte, die sich innerhalb einen Farbebereich befinden.
- Canny findet die Kanten aller Objekten im Bild.
- Die Funktion HoughLines analysiert die Kanten und findet alle Linien im Bild.
- alle Zeilen werden dann in einer rechten und einer linken Linie zusammengerechnet.

```
35
36  def getLines(frame, lH, lS, lV, hH, hS, hV):
37      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
38      LowerRegion = np.array([lH,lS,lV],np.uint8)
39      upperRegion = np.array([hH,hS,hV],np.uint8)
40      colorRange = cv2.inRange(hsv,LowerRegion,upperRegion)
41      cannyFrame = cv2.Canny(colorRange, 50, 150)
42      croppedFrame = maskingFrame(cannyFrame)
43      cv2.imshow('filter', croppedFrame)
44      lines = cv2.HoughLinesP(croppedFrame, 2, np.pi/180, 100, np.array([]), minLineLength=50, maxLineGap=25)
45      rightLine, leftLine = average(lines)
46      return rightLine, leftLine
47
```

Abbildung 6 -getLines

3.2.4 Coordinates

Diese Funktion rechnet die Koordinaten, mit denen die Linien geplottet werden. Liefert als Ergebnis ein Array mit zwei x-Koordinaten und zwei y-Koordinaten.

```
61  def coordinates(slope, intercept):
62      y1 = 320
63      y2 = int(y1*0.45)
64      if slope == 0:
65          return np.array([0,0,0,0])
66      else:
67          x1 = int((y1-intercept)/slope)
68          x2 = int((y2-intercept)/slope)
69          return np.array([x1, y1, x2, y2])
70
```

Abbildung 7 - coordinates

3.2.5 calc_avg

Funktion, die den Durchschnittswert eines Arrays berechnet und als Ergebnis zurück liefert.

```
71  def calc_avg(values):
72      summ = 0
73      for x in values:
74          summ = summ + x
75      if len(values) == 0:
76          avg = round(summ / 1, 2)
77      else:
78          avg = round(summ / len(values), 2)
79      return avg
80
```

Abbildung 8 - calc_avg

3.2.6 average

Average erfüll die Aufgabe, ein Array, das viele Linien enthält, in zwei Linien umzuwandeln, die den Rand des Fahrwegs darstellen.

- Die Funktion bekommt ein Array von Linien als Parameter.
- Jede Linie wird in vier Punkte geteilt und mit der Funktion polyfit wird die Neigung der Linie berechnet.
- Alle Linien mit positiver Neigung werden als rechte Linien bezeichnet. Linien mit negativer Neigung werden als linke Linien berücksichtigt.
- Die Funktionen calc_avg und coordinates werden aufgerufen, so dass am Ende aus allen rechten und linken Linien nur zwei Linien bestehen.

```
81
82  def average(lines):
83      rightslopes = []
84      rightintercepts = []
85      leftslopes = []
86      leftintercepts = []
87      rightLine = []
88      leftLine = []
89      lastL = []
90      if lines is None:
91          lines = lastL
92      if lines is not None:
93          for line in lines:
94              lastL = line
95              x1, y1, x2, y2 = line.reshape(4)
96              par = np.polyfit((x1,x2),(y1,y2), 1)
97              slope = par[0]
98              intercept = par[1]
99              if slope > 0:
100                 rightslopes.append(slope)
101                 rightintercepts.append(intercept)
102             if slope < 0:
103                 leftslopes.append(slope)
104                 leftintercepts.append(intercept)
105         rightSlAv = calc_avg(rightslopes)
106         rightInAv = calc_avg(rightintercepts)
107         leftSlAv = calc_avg(leftslopes)
108         leftInAv = calc_avg(leftintercepts)
109         rightLine = coordinates(rightSlAv, rightInAv)
110         leftLine = coordinates(leftSlAv, leftInAv)
111     return rightLine, leftLine
112
```

Abbildung 9 - average

Abbildung 10 - gemittelte Linien

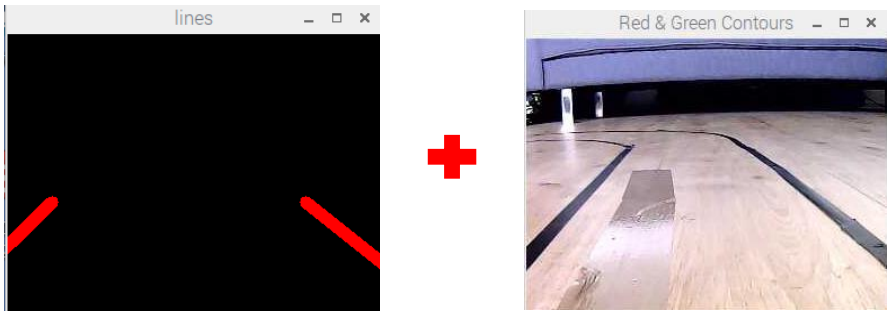
3.2.7 mergeFrameAndLines

Die Funktion erhält als Parameter, die mit der Funktion Durchschnitt berechneten Linien und das originale Bild.

- Die Linien werden auf einem schwarzen Bild mit der cv2.line-Funktion geplottet.
- addWeighted fügt die beiden Bilder zu einem zusammen.

```
112
113 def mergeFrameAndLines(frame, lines):
114     linesOnFrame = np.zeros_like(frame)
115     if lines.size < 3 :
116         merged = cv2.addWeighted(frame, 0.8, linesOnFrame, 1, 1)
117         return merged
118     if lines.size > 3:
119         for line in lines:
120             x1, y1, x2, y2 = line.reshape(4)
121             cv2.line(linesOnFrame, (x1,y1), (x2,y2), (0,0,255), 10)
122         merged = cv2.addWeighted(frame, 0.8, linesOnFrame, 1, 1)
123         return merged
124
```

Abbildung 11 - mergeFrameAndLines



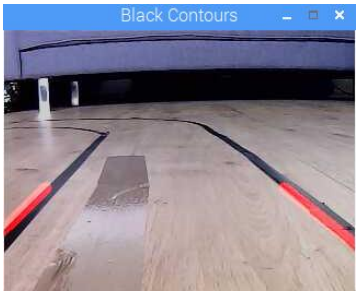


Abbildung 12 - zusammengefügte Bilder

3.3 Bewegungsfunktionen

3.3.1 Motorensteuerung

Die folgenden Funktionen werden zur Bewegungssteuerung des Fahrzeugs verwendet

- Der Servo1 steuert die Richtung (links oder rechts), in die sich das Fahrzeug bewegt. ein Wert von 1000 bedeutet, dass die Reifen des Fahrzeugs nach rechts zeigen. 1500 bedeutet die Reifen zeigen nach vorne und 2000 nach links
- DIR1 und DIR2 kontrollieren, ob das Fahrzeug vorwärts (1) oder rückwärts (0) fährt.
- PWM1 und PWM2 regulieren die Geschwindigkeit des Fahrzeuges

```
128
129 def forward():
130     global camHor
131     mdev.writeReg(mdev.CMD_SERVO1,1500)
132     mdev.writeReg(mdev.CMD_DIR1,1)
133     mdev.writeReg(mdev.CMD_DIR2,1)
134     mdev.writeReg(mdev.CMD_PWM1,speed)
135     mdev.writeReg(mdev.CMD_PWM2,speed)
136
137 def goRight():
138     mdev.writeReg(mdev.CMD_SERVO1,1350)
139     mdev.writeReg(mdev.CMD_DIR1,1)
140     mdev.writeReg(mdev.CMD_DIR2,1)
141     mdev.writeReg(mdev.CMD_PWM1,speed)
142     mdev.writeReg(mdev.CMD_PWM2,speed)
143
144 def goLeft():
145     mdev.writeReg(mdev.CMD_SERVO1,1650)
146     mdev.writeReg(mdev.CMD_DIR1,1)
147     mdev.writeReg(mdev.CMD_DIR2,1)
148     mdev.writeReg(mdev.CMD_PWM1,speed)
149     mdev.writeReg(mdev.CMD_PWM2,speed)
150
151 def stop():
152     mdev.writeReg(mdev.CMD_SERVO1,1500)
153     mdev.writeReg(mdev.CMD_DIR1,1)
154     mdev.writeReg(mdev.CMD_DIR2,1)
155     mdev.writeReg(mdev.CMD_PWM1,0)
156     mdev.writeReg(mdev.CMD_PWM2,0)
157
158 def rightTurn():
159     time.sleep(0.1)
160     mdev.writeReg(mdev.CMD_SERVO1,1150)
161     mdev.writeReg(mdev.CMD_DIR1,1)
162     mdev.writeReg(mdev.CMD_DIR2,1)
163     mdev.writeReg(mdev.CMD_PWM1,speed)
164     mdev.writeReg(mdev.CMD_PWM2,speed)
165
166 def leftTurn():
167     time.sleep(0.1)
168     mdev.writeReg(mdev.CMD_SERVO1,1850)
169     mdev.writeReg(mdev.CMD_DIR1,1)
170     mdev.writeReg(mdev.CMD_DIR2,1)
171     mdev.writeReg(mdev.CMD_PWM1,speed)
172     mdev.writeReg(mdev.CMD_PWM2,speed)
173
```

Abbildung 13 - Steuerungsfunktionen

3.3.2 getDistance

Funktion zur Entfernungsberechnung mit HC-SR04 Sensor. Berechnet die Entfernung anhand der Signaldauer und der Schallgeschwindigkeit.

```
174 def getDistance(trig,echo):
175     GPIO.setup(trig, GPIO.OUT)
176     GPIO.setup(echo, GPIO.IN)
177
178     GPIO.output(trig, False)
179     time.sleep(0.0002)
180
181     GPIO.output(trig, True)
182     time.sleep(0.00001)
183     GPIO.output(trig, False)
184
185     while GPIO.input(echo) == 0:
186         signal0 = time.time()
187
188     while GPIO.input(echo) == 1:
189         signal1 = time.time()
190
191     signalDuration = signal1 - signal0
192     distance = signalDuration * 17150
193     distance = round(distance, 2)
194     GPIO.cleanup
195     return distance
196
```

Abbildung 14 - getDistance

Abbildung 15 - HC-SR04 Sensor

4. Globale Variablen

Globale Variablen erlauben den Programmierer, Variablen sowohl innerhalb von Funktionen als auch außerhalb zu verwenden. Sie machen es auch einfach, Änderungen an verschiedenen Funktionen vorzunehmen.

```
200
201 camVer = 1250
202 camHor = 1500
203 run = True
204 go = False
205 collision = False
206 speed = 190
207
```

Abbildung 16 - globale Variablen

5. Hauptschleife

5.1 Video und Entfernung

Die erste Hälfte der Hauptschleife nimmt das Bild und verarbeitet es, um Linien und Konturen zu identifizieren. Ebenso wird die Funktion getDistance aufgerufen, um zu verhindern, dass das Fahrzeug kollidiert.

- Wenn sich ein Objekt weniger als 25 Zentimeter vor dem Fahrzeug befindet, wird das Fahrzeug angehalten.
- Wird ein grünes Objekt detektiert, wird die Variable „go“ auf true gesetzt und das Fahrzeug kann vorwärtsfahren. Das Gegenteil eintritt mit einem roten Objetk.
- Es werden grüne und blaue Linien eingezeichnet, die die Bereiche markieren, in denen die Navigationslogik arbeitet.

```
while run:
    _,vFrame = capture.read()
    goFrame = vFrame.copy()
    rightLine, leftLine = getLines(vFrame, 0, 0, 0, 179, 255, 115)
    blackLines = np.array([rightLine, leftLine], dtype="object")
    redContours = getObjects(vFrame, 0, 200, 30, 10, 255, 255)
    greenContours = getObjects(vFrame,25, 52, 72, 102, 255, 255)

    dist = getDistance(23,24)
    print("frontal distance:  ", dist)
    if dist < 25:
        collision = True
    elif dist > 25:
        collision = False

    xr, xl = 0, 0

    if go == False:
        for con in greenContours:
            if cv2.contourArea(con) > 300:
                go = True
                cv2.drawContours(goFrame, con, -1, (0,0,255), 5)
    if go == True:
        for con in redContours:
            if cv2.contourArea(con) > 300:
                go = False
                cv2.drawContours(goFrame, con, -1, (0,255,0), 5)

    mergedFrame = mergeFrameAndLines(vFrame, blackLines)
    cv2.line(mergedFrame, (45,140), (45,200), (0,255,30), 5)
    cv2.line(mergedFrame, (125,140), (125,200), (0,255,30), 5)
    cv2.line(mergedFrame, (210,140), (210,200), (50,200,0), 5)
    cv2.line(mergedFrame, (290,140), (290,200), (50,200,0), 5)

    if rightLine is not None:
        xr = blackLines[0][2]
    if leftLine is not None:
        xl = blackLines[1][2]
    cv2.line(mergedFrame, (xr,140), (xr,200), (255,50,0), 5)
    cv2.line(mergedFrame, (xl,140), (xl,200), (255,50,0), 5)
```

Abbildung 17 - Hauptschleife Teil1

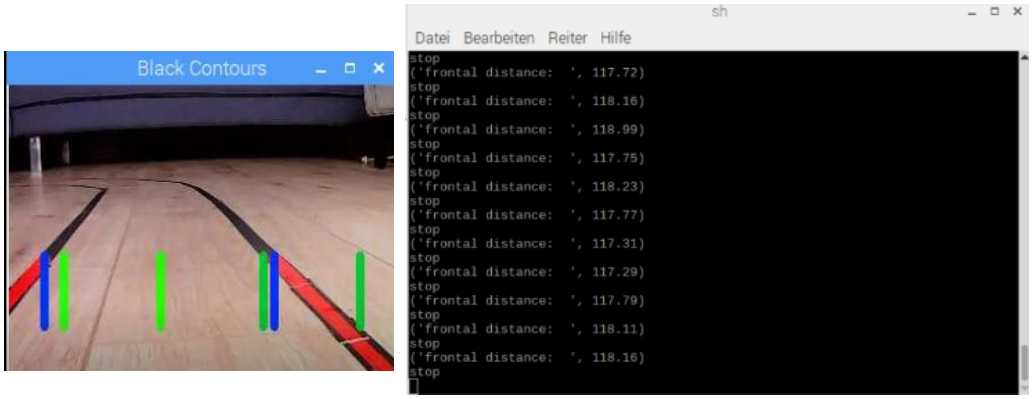


Abbildung 18 - Fahrzeug im Stillstand

5.2 Logik der Bewegung

In diesem Teil der Schleife sind alle Bedingungen vorhanden, die es dem Fahrzeug ermöglichen, während der Fahrt Entscheidungen zu treffen.

- Sind die blaue Linien innerhalb der voreingestellten grünen Bereiche, dann fährt der Wagen gerade aus.
- Werden die Linien nach links verschoben, bewegt sich dann das Fahrzeug auch nach Links
- Werden die Linien nach rechts verschoben, bewegt sich das Fahrzeug nach rechts.
- Falls go auf False oder collision auf True gesetzt werden, kommt das Fahrzeug zum Haltzustand.
- Sollte die „Q“-Taste gedrückt werden, wird dann die Schleife verlassen und das Programm beendet.

```
253     if go == True and collision == False:
254         print(xl, " xl   ", xr, " xr")
255         if (xl > 50 and xl < 120) and (xr > 215 and xr < 285):
256             forward()
257             print("going Forward")
258         if (xl > 120 and xr > 285) and (xr == 0 and xl > 0 and xl < 165):
259             goRight()
260             print("adjusting Right")
261         if (xr == 0 and xl > 165):
262             rightTurn()
263             print("turning Right")
264         if (xl > 0 and xl < 50 and xr < 215) or (xl == 0 and xr > 165):
265             goLeft()
266             print("adjusting Left")
267         if (xl == 0 and xr > 0 and xr < 165):
268             leftTurn()
269             print("turning Left")
270     else:
271         stop()
272         print("stop")
273
274     cv2.imshow("Red & Green Contours", goFrame)
275     cv2.imshow("Black Contours", mergedFrame)
276     if cv2.waitKey(1) & 0xFF == ord('q'):
277         break
278
279 run = False
280 stop()
281 capture.release()
282 cv2.destroyAllWindows()
```

Abbildung 19 - Hauptschleife Teil2

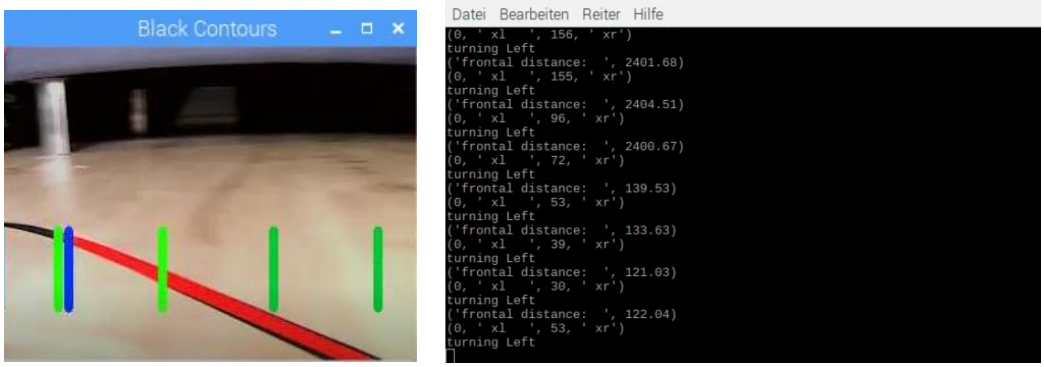


Abbildung 20 - Fahrzeug biegt links ab

6. Schluss

Es wurde beschlossen, die Funktion cv2.HoughLines anstelle von cv2.findContours zu verwenden. So wurde mit Linien anstelle von Konturen gearbeitet, und war es möglich, die Steigung der Linien zu berechnen, wodurch das Programm fast fehlerfrei erkennen kann, ob es sich um die rechte oder linke Straßenseite handelt. Die Trennung der Daten in linke und rechte Werte ermöglichte es, vereinfachte Bedingungen für die Fahrzeugnavigation zu schreiben.