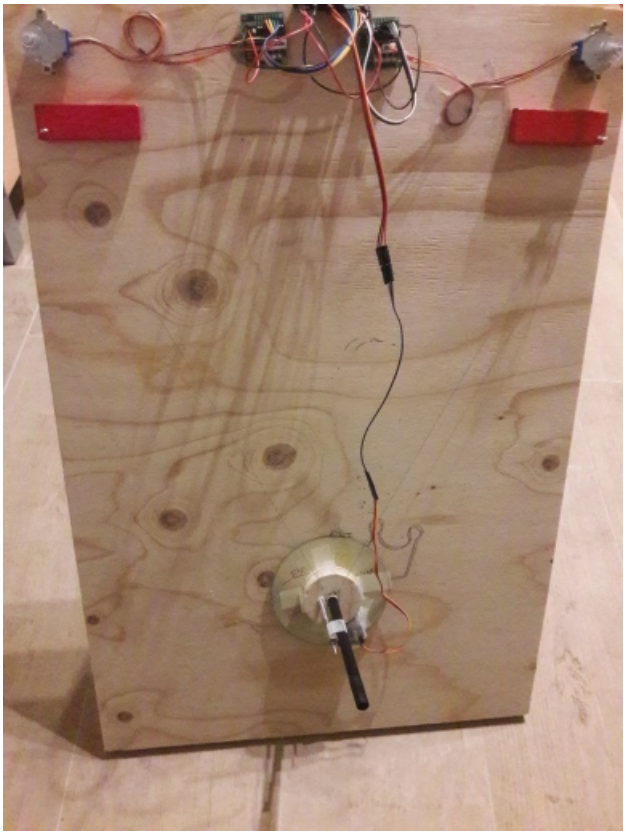
Hanging-Wall-Plotter

Projektarbeit Klasse 9

Paul Schulte

31. März 2019



Inhaltsverzeichnis

1. Idee

Die Idee hinter dem Projekt ist ein Plotter(Zeichner), der aus zwei Motoren und einem Stift, der angehoben oder abgesenkt werden kann, besteht. In G-Code umgewandelte Bilder sollen später mit diesem Plotter gezeichnet werden können. Dafür liest das Python-Programm die G-Code-Datei ein und teilt den Motoren mit, was sie zu tun haben.

2. Aufbau

2.1. Genereller Aufbau des Plotters

Die beiden Motoren sind mit Schrauben fest am Brett angeschraubt:

Abbildung 1: Motor

und mit einer Steckverbindung an eine Platine (die später vom raspberrypi genutzt wird, um die Motoren anzusteuern) angeschlossen:

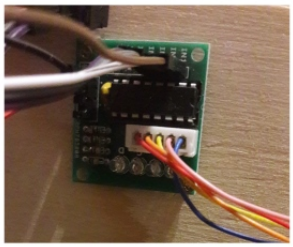


Abbildung 2: Motor-controller

Auf das Gewinde vom Motor ist eine Nähspule mit Garn gesteckt:

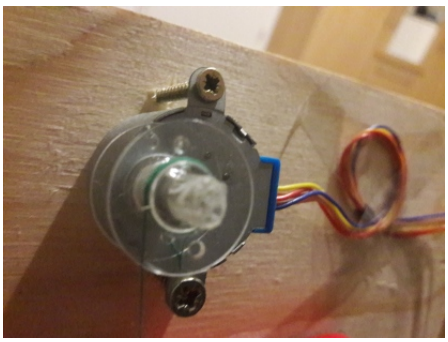


Abbildung 3: Spule

Dieser Garn läuft durch eine Schraube, die dafür sorgt, dass sich der Garn später vernünftig aufwickelt:

Abbildung 4: Spule

Das Ende des Garns ist am Stift festgebunden:

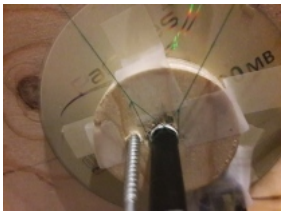


Abbildung 5: Ende des Garns

(Dieser Stift ist auf einem Holzblock, der auf einer CD aufgeklebt ist, festgemacht) Der Servo ist so an die CD angebracht, dass mit einem Zustand der Stift angehoben und mit einem anderen der Stift abgesenkt werden kann:

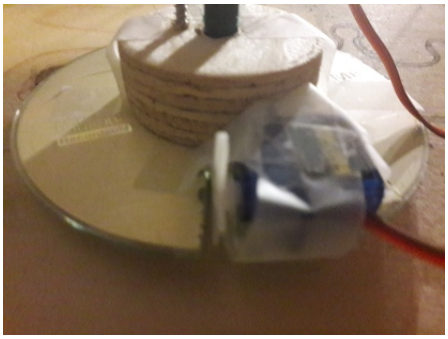


Abbildung 6: Servo

2.2. Der Schaltplan

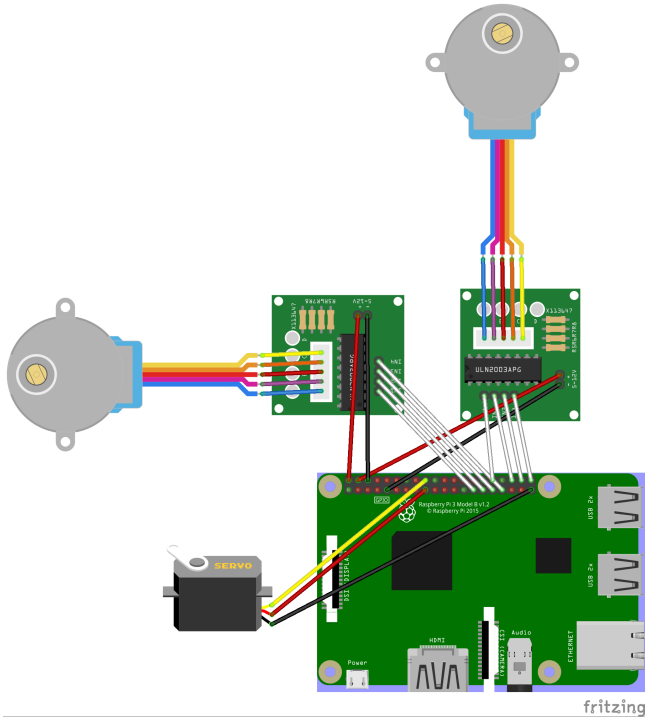


Abbildung 7: Schaltskizze

3. Das Programm

3.1. Einen Schrittmotor ansteuern

Will man einen Schrittmotor intern um 360° nach rechts(die Umsetzung sorgt dafür dass eine interne Drehung der Gradzahl a nicht gleich der äußeren Umdrehung b ist, sondern viel niedriger, aber dazu später mehr) drehen, so muss man folgenden Rhythmus benutzen (Die verschiedenen Pins stehen hierbei für die vier Magneten im Motor):

Schritt:	1	2	3	4	5	6	7	8
Pin1	x	x						x
Pin2		x	x	x				
Pin3				x	x	x		
Pin4						x	x	x

Das liegt daran, dass der Motor vier innere Magneten hat, die nacheinander angesteuert werden müssen(siehe(*Tutorial Raspberry Pi/ Schrittmotor Stepper motor*)). Damit die Umdrehungen genauer sind, benutzt man zwei Magneten gleichzeitig, um Zwischenschritte einzufügen. Deswegen gibt es auch 8 Schritte.

Damit später mehrere Motoren-Objekte erzeugt werden können, muss man das Programm als eine Klasse definieren, die später u.a. die Methoden lt(left-turn) und rt(right-turn) beinhalten soll:

```
1 import time
2 import RPi.GPIO as GPIO
3
4 class Motor:
5     def rt(self):
6         pass
7     def lt(self):
8         pass
```

bei der Erzeugung eines Motor-Objekts werden im Konstruktor die vier Anschlusspins, die auch direkt als output-Pins gesetzt und der Klasse als Attribute zugewiesen werden, übergeben:

```
1     def __init__(self,A,B,C,D):
2         GPIO.setmode(GPIO.BCM)
3         GPIO.setwarnings(False)
4
5         GPIO.setup(A, GPIO.OUT)
6         GPIO.setup(B, GPIO.OUT)
```

```
7           GPIO.setup(C, GPIO.OUT)
8           GPIO.setup(D, GPIO.OUT)
9           self.A = A
10          self.B = B
11          self.C = C
12          self.D = D
```

Damit das Wechseln z.B. von Schritt 3 zu 4 oder von 8 zu 1 später leichter von der Hand geht, wird folgende Methode definiert:

```
1       def GPIO_SETUP(self, a,b,c,d, time1, degrees):
2           GPIO.output(self.A, a)
3           GPIO.output(self.B, b)
4           GPIO.output(self.C, c)
5           GPIO.output(self.D, d)
6           time.sleep(time1 / degrees)
```

Time1 ist die übergebene Zeit, die die komplette Drehung dauern soll (siehe Abschnitt 3.2) und degrees bedeutet wie oft diese Methode in einer Umdrehung aufgerufen wird. Die letzte Zeile sagt also aus Sicht der Methode, die später GPIO_SETUP() aufruft, nichts anderes als warte bei jedem Aufruf dieser Methode so lange, dass nach degrees Aufrufen time1 umgegangen ist.

Die letzendliche Methode rt() sieht wie folgt aus:

```
1   def rt(self, deg, time):
2           full_circle = 512.0
3           degree = (full_circle/360)*deg
4           degree_save= (full_circle/360)*deg
5           self.GPIO_SETUP(0,0,0,0, time, degree_save *
                8)
6
7           while degree > 0.0:
8               self.GPIO_SETUP(1,0,0,0,time,degree_save
                    *8)#Schritt1
9               self.GPIO_SETUP(1,1,0,0,time,degree_save
                    *8)#Schritt2
10              self.GPIO_SETUP(0,1,0,0,time,degree_save
                    *8)#Schritt3
11              self.GPIO_SETUP(0,1,1,0,time,degree_save
                    *8)#Schritt4
12              self.GPIO_SETUP(0,0,1,0,time,degree_save
                    *8)#Schritt5
13              self.GPIO_SETUP(0,0,1,1,time,degree_save
                    *8)#Schritt6
```

```
14                self.GPIO_SETUP(0,0,0,1,time,degree_save
                      *8)#Schritt7
15                self.GPIO_SETUP(1,0,0,1,time,degree_save
                      *8)#Schritt8
16                degree -= 1
17            self.GPIO_SETUP(0,0,0,0, time, degree_save *
                  8)
```

Zuerst stellt sich die Frage Ẅie viele interne Umdrehungen werden benötigt, um den Motor wirklich um 360 ° (full_circle) nach rechts zu drehen?̈ Dazu guckt man auf dem zugehörigen Datasheet(siehe(*28BYJ-48 – 5V Stepper Motor Datasheet*)) zu dem Motor 28BYJ-48 nach Stride Angle(zu Deutsch: Winkel pro Schritt) und findet die Angabe 5.625 ° / 64. Multipliziert mit 8 (acht Schritte) erhält man das Ergebnis 5.625 ° / 8 (=0.703125) Winkel pro interne Umdrehung. Teilt man nun 360 durch diesen Winkel, so hat man die Anzahl der internen Umdrehungen, die für eine äußere Umdrehung des Motors benötigt werden: 360 / 0.703125 = 512 interne Umdrehungen(full_circle)

Um zu wissen, wie oft der Motor intern einmal ganz gedreht werden muss, damit der Motor sich außen um den angegebenen Parameter deg dreht, multipliziert man den Parameter deg mit dem Quotient aus der Anzahl der benötigten internen Umdrehungen für eine gesamte äußere Umdrehung(512) und der Gradanzahl für eine ganze Umdrehung(360 °). So ergibt sich z.B. für eine ganze Umdrehung(Parameter deg=360) ((512/360)*360=)512 interne Umdrehungen. degree_save soll einfach eine Aufbewahrung für die zurückzulegenden internen Umdrehungen sein, denn wenn man GPIO_SETUP die Variable degree übergeben würde, würde es nicht funktionieren, weil als nächstes eine Schleife folgt, die so lange eine interne Umdrehung macht und degree dekrementiert, bis degree <= 0.0 ist. Der Parameter time wird der Methode GPIO_SETUP() einfach immer übergeben und steht, wie bereits erwähnt, für die Zeit, die diese Umdrehung dauern soll.
Manchmal findet man die Zeile

```
1 self.GPIO_SETUP(0,0,0,0,time,degree_save*8)
```

im Programmcode. Diese Zeile sorgt einfach dafür, dass der Motor nicht überhitzt, weil ein Magnet die ganze Zeit mit Strom versorgt wird.

Um auch die Methode lt() implementieren zu können, muss man nur die Reihenfolge der Schritte umkehren(8->1->2->...):

```
1  def lt(self, deg, time):
2          full_circle = 512.0
3          degree = (full_circle/360)*deg
4          degree_save= (full_circle/360)*deg
```

```
5            self.GPIO_SETUP(0,0,0,0, time, degree_save *
                 8)
6
7            while degree > 0.0:
8                self.GPIO_SETUP(1,0,0,1,time,degree_save
                     *8)#Schritt8
9                self.GPIO_SETUP(0,0,0,1,time,degree_save
                     *8)#Schritt7
10               self.GPIO_SETUP(0,0,1,1,time,degree_save
                     *8)#Schritt6
11               self.GPIO_SETUP(0,0,1,0,time,degree_save
                     *8)#Schritt5
12               self.GPIO_SETUP(0,1,1,0,time,degree_save
                     *8)#Schritt4
13               self.GPIO_SETUP(0,1,0,0,time,degree_save
                     *8)#Schritt3
14               self.GPIO_SETUP(1,1,0,0,time,degree_save
                     *8)#Schritt2
15               self.GPIO_SETUP(1,0,0,0,time,degree_save
                     *8)#Schritt1
16               degree -= 1
17           self.GPIO_SETUP(0,0,0,0, time, degree_save *
                 8)
```

Motor erstmals bewegen lassen

Jetzt wo die Klasse fertig ist, kann man an einem Beispiel noch einmal ihren Nutzen erklären: Zuerst importiert man die Klasse und erzeugt zwei Objekte mit den Pins, die an den Motoren angeschlossen sind(In meinem Fall: Motor1(0, 5, 6, 12) und Motor2(12, 16, 20, 21)):

```
1  import Motor_own
2
3  motor1=Motor_own.Motor(0,5,6,13)
4  motor2=Motor_own.Motor(12,16,20,21)
```

Anschließend kann man den Motor1 z.B. in 3 Sekunden um 360° nach links drehen:

```
1  motor1.lt(360, 3)
```

oder den Motor2 in z.B. 1 Sekunde um 5° nach rechts:

```
1  motor2.rt(5, 1)
```

3.2. Eine Linie zeichnen

Um später komplexere Zeichnungen realisieren zu können muss der Plotter ersteinmal lernen, wie man eine gerade Linie zeichnet. Damit der Plotter aber wirklich eine Linie zeichnen kann, muss zuerst geklärt werden, wie weit die Motoren jeweils Schnur einziehen oder ablassen müssen. Nun stellt sich die Frage wie sich die Motoren verhalten müssen, um generell von einem Punkt zu einem anderen zu kommen.

Um diese Berechnung für jeden beliebigen Punkt benutzen zu können, geht man wie folgt vor. Unter der Annahme, dass sich der Stift auf dem Punkt (0|0) befindet und sich zu einem Punkt P bewegen will:

1. X-Entfernung zum Punkt(0|0) der beiden Motoren herausfinden

   $\Rightarrow$ Da der Punkt(0|0) in der Mitte der beiden Motoren liegt:

   $$x_a = \frac{Entfernung\ zwischen\ Motoren}{2}$$
   $$x_b = \frac{Entfernung\ zwischen\ Motoren}{2}$$

2. Y-Entfernung zum Punkt(0|0) der beiden Motoren herausfinden

   $\Rightarrow$ $y_a = YEntfernung\ zum\ Ursprung$
   $y_b = YEntfernung\ zum\ Ursprung$

3. Mit dem Satz des Pythagoras erhält man folgende Formeln auf dem Punkt(0|0):

   $a_1 = \sqrt{x_a^2 + y_a^2}$

   $b_1 = \sqrt{x_b^2 + y_b^2}$

   Da aber später die Entfernung zu jedem beliebigem Punkt berechnet können werden soll, werden die Koordinaten des Punktes (0|0) folgend mit in die Formeln eingebracht:

   $$a_1 = \sqrt{(x_a + 0)^2 + (y_a - 0)^2}$$
   $$b_1 = \sqrt{(x_b - 0)^2 + (y_b - 0)^2}$$

   Dabei wird für a_1: x_a + 0 gerechnet, weil sich die X-Entfernung vom Motor1 zum Stift bei einer höheren X-Koordinate erhöht und bei einer niedrigeren als 0 verkleinert. Bei b_1 hingegen heißt es x_b - 0, da sich die X-Entfernung vom Motor2 zum Stift bei einer höheren X-Koordinate verkleinert und bei einer niedrigeren als 0 erhöht. Bei beiden ist die vertikale Entfernung vom Stift zu den Motoren

$y_{a/b}$ - 0. Das liegt daran, dass sich die Entfernung bei größer werdender Y-Koordinate verringert. Somit ist es auch logisch, dass die Entfernung sich bei kleiner werdender Y-Koordinate erhöht.

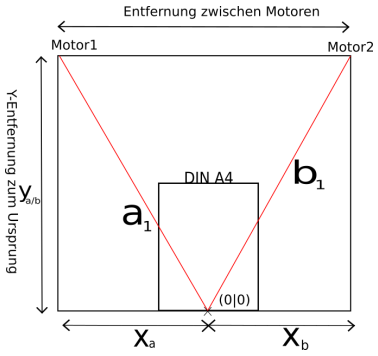


Abbildung 8: $A_1 und B_1$

4. a_2 *und* b_2 lassen sich durch die gleichen Formeln berechnen. Allerdings muss man nun anstatt (0|0) die Koordinaten des Punktes P einsetzen. Es ergibt sich

$$a_2 = \sqrt{(x_a + X_p)^2 + (y_a - Y_p)^2}$$

$$b_2 = \sqrt{(x_b - X_p)^2 + (y_b - Y_p)^2}$$

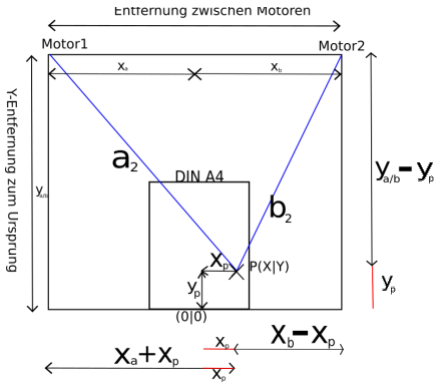


Abbildung 9: $A_2 und B_2$

=> Nun müsste man nur noch a2-a1 und b2-b1 rechnen und den Motoren mitteilen, wie sie sich zu bewegen haben.

Weil der Stift sich am Ende allerdings von jedem beliebigem Punkt zu jedem anderen beliebigem Punkt bewegen können soll und nicht nur von Punkt (0|0) zu Punkt P(X|Y), muss folgendes Konzept greifen:

a_1 *und* b_1 berechnen die Entfernung des jeweiligen Motors zur aktuellen Position und a_2 *und* b_2 berechnen die Entfernung zum Zielpunkt. Für dieses Prinzip braucht man aber zuerst eine Variable für die aktuelle X-Position und eine für die aktuelle Y-Position:

```
1        x= 0 #actual position
2        y= 0 #actual position
```

Mit diesen Variablen lassen sich dann folgende Formeln aufstellen:

$$a_1 = \sqrt{(x_a + x_{aktuell})^2 + (y_a - y_{aktuell})^2}$$
$$b_1 = \sqrt{(x_b - x_{aktuell})^2 + (y_b - y_{aktuell})^2}$$

$$a_2 = \sqrt{(x_a + x_{ziel})^2 + (y_a - y_{ziel})^2}$$
$$b_2 = \sqrt{(x_b - x_{ziel})^2 + (y_b - y_{ziel})^2}$$

Berechnung von a1 und a2, b1 und b2 im Programm

Die jeweiligen Entfernungen werden dem VPlotter-Objekt als Parameter übergeben: def __init__(self, ***distanceBetweenMotors***, ***distanceYtoOrigin***, [...]): Aber dazu später mehr... Vorerst wird deswegen davon ausgegangen, dass dem Programm die Variablen **self.xa, self.xb, x(Zielposition), y(Zielposition), self.x(aktuelle Position), self.y(aktuelle Position) und self.distanceY(Y-Entfernung zum Ursprung)** bekannt sind.

In Python sieht das folgend aus:

```
1 a1= math.sqrt((self.xa+self.x)**2+(self.distanceY-
      self.y)**2)
2 b1= math.sqrt((self.xb-self.x)**2+(self.distanceY-
      self.y)**2)
3
4 a2= math.sqrt((self.xa+x)**2+(self.distanceY-y)**2)
5 b2= math.sqrt((self.xb-x)**2+(self.distanceY-y)**2)
```

Dabei darf man nicht vergessen die aktuelle Position auf das letzte Ziel zu setzen:

```
1 self.x= x
2 self.y= y
```

Bewegen des Motors

Die Strecken a2-a1 und b2-b1 werden dann später vom Motor zurückgelegt. Die Motoren diese berechnete Strecke alleine einziehen oder ablassen zu lassen, würde aber nicht funktionieren, da dann keine gerade Linie entstehen würde, was ja auch logisch ist, da bei gleicher Geschwindigkeit (Maximalgeschwindigkeit wird dem Programm ebenfalls übergeben: def __init__(self, ..., ***max_geschwindigkeit***, , [...])) und nicht gleichlangen Strecken ein Motor eher fertig wäre als der andere. Deswegen müssen beide Motoren mit einer unterschiedlichen Geschwindigkeit laufen. Um diesen Gedanken umsetzen zu können wird erstmal die längere Strecke ermittelt

```
1  if abs(a2 - a1) >= abs(b2 - b1):
2              [...]
3          else:
4              [...]
```

(hier wird die Funktion abs() benutzt, da die Strecken eventuell auch negativ sein können und dann die größere Strecke vielleicht k̈leinerwäre)

und dann die Formel für die bei einer Geschwindigkeit v für eine Strecke s benötigte Zeit zur Hand gezogen:

Zeit: $t = \frac{s}{v}$

Mit dieser Formel kann nun neben der Strecken (a2 - a1 oder b2 - b1) auch die Zeit errechnet werden, die das Bewegen beider Motoren zu einem beliebigem Punkt dauern muss (dabei müssen beide Motoren gleichzeitig anfangen und aufhören).
Damit beide Motoren gleichzeitig laufen, werden zwei Threads aus dem Paket threading benutzt:

```
1 from threading import Thread
2 [...]
3 if abs(a2 - a1) >= abs(b2 - b1):
4     t1=Thread(target=self.MotorBewegen,args=(1,a2-a1
          ,(a2-a1)/self.max_geschwindigkeit,))
5     t2=Thread(target=self.MotorBewegen,args=(2,b2-b1
          ,(a2-a1)/self.max_geschwindigkeit,))
```

```
6         t1.start()
7         t2.start()
8
9         t1.join()
10        t2.join()
11
12    else:
13        t1=Thread(target=self.MotorBewegen,args=(1,a2-a1
              ,(b2-b1)/self.max_geschwindigkeit,))
14        t2=Thread(target=self.MotorBewegen,args=(2,b2-b1
              ,(b2-b1)/self.max_geschwindigkeit,))
15        t1.start()
16        t2.start()
17
18        t1.join()
19        t2.join()
```

Die Funktion MotorBewegen(self, motor, length, time) dient hier als sogenannte Call-back Methode. Wichtig ist, dass als motor der Motor (1 oder 2), als length die zurückzulegenden Strecke und als time die Zeit, die für diese Bewegung gebraucht werden soll, übergeben wird. Deswegen erhält t1 a2-a1 als Strecke und t2 erhält b2-b1. Je nachdem welche Strecke länger ist, wird die Zeit, die für die längere Strecke bei der dem Programm als Parameter übergebenen Maximalgeschwindigkeit benötigt wird, als time genutzt. So wird für die beiden threads entweder

$$t = \frac{s}{v} => t = \frac{(a2 - a1)}{self.max\_geschwindigkeit}$$

bei einer größeren Strecke von a2-a1 und

$$t = \frac{s}{v} => t = \frac{(b2 - b1)}{self.max\_geschwindigkeit}$$

bei einer größeren Strecke von b2-b1 als Zeit übergeben.

Mit

```
1  t1.start()
2  t2.start()
```

werden die zwei Threads gestartet.

```
1  t1.join()
2  t2.join()
```

sorgt dafür, dass das Programm erst weiterläuft, wenn beide Threads abgeschlossen sind.

Die Methode MotorBewegen() sieht wie folgt aus:

```
1  def MotorBewegen(self, motor, length, time):
2      if length == 0:
3          return
4
5      if motor == 1:
6          if length >= 0:
7              self.motor1.lt(length/(25/360), abs(time
                   ))
8          else:
9              self.motor1.rt(-length/(25/360), abs(
                   time))
10
11     elif motor == 2:
12         if length >= 0:
13             self.motor2.rt(length/(25/360), abs(time
                   ))
14         else:
15             self.motor2.lt(-length/(25/360), abs(
                   time))
```

Nachdem geguckt wird, ob sich überhaupt bewegt werden muss, oder ob die Funktion schon vorzeitig abgebrochen werden kann (wenn die Länge = 0 ist und der Motor sich nicht bewegen muss), wird geprüft, um welchen Motor es sich handelt. Mit dem jeweiligen Motor (1 oder 2) passiert dann folgendes: ist die Länge größer als 0, so dreht sich Motor 1 nach links und Motor 2 nach rechts, da sie ja Strecke ablassen wollen. Ist die Länge aber negativ, so dreht sich Motor 1 nach rechts und Motor 2 nach links.
Um die in Abschnitt 3.1 definierte Methoden rt(self, deg, time) und lt(self, deg, time) benutzen zu können muss die Länge allerdings zuerst in Grad umgewandelt werden. Deswegen wurde die zurückgelegte Strecke bei einer 360° Drehung erfasst. Mit 25mm / 360° hat man die Strecke in mm, die bei einem Grad Umdrehung zurückgelegt wird. Folglich kann mit
$\frac{Laenge}{25/360}$ die Umdrehung in Grad berechnet werden. Ist die zurückzulegenden Länge negativ, gilt: $Umdrehung\ in\ Grad = \frac{-Laenge}{25/360}$

3.3. Kurven zeichnen

Um Kurven zeichnen zu können, muss diese Kurve in viele kleine gerade Strecken unterteilt werden. Diese Strecken werden dann mit den oben bereits erklärten Funktionen zurückgelegt. Die Funktion arc (engl.: Bogen) soll folgende Parameter erhalten:

1. direction

=> Die Richtung ist entweder 2(Uhrzeigersinn) oder 3 (gegen Uhrzeigersinn) (siehe Abschnitt 3.5)

2. dir_x

=> Die X-Koordinate des Ziels

3. dir_y

=> Die Y-Koordinate des Ziels

4. i

=> Die die X-Entfernung vom aktuellen Punkt zum Mittelpunkt des Kreises der Kurve

5. j

=> Die die Y-Entfernung vom aktuellen Punkt zum Mittelpunkt des Kreises der Kurve

Am Anfang der Funktion werden dann mit diesen Parametern folgende Berechnungen gemacht:

- mx= self.x + i
- my= self.y + j
- radius= math.sqrt(i ** 2 + j ** 2)

Der Methode stehen nun auch die Koordinaten des Mittelpunktes, die aus der aktuellen Position und der übergebenen Entfernung zum Mittelpunkt berechnet wurden, zur Verfügung. Mithilfe der Formel
$r^2 = i^2 + j^2 \quad => \quad r = \sqrt{i^2 + j^2}$
(Satz des Pythagoras) wurde der Radius des Kreises berechnet.

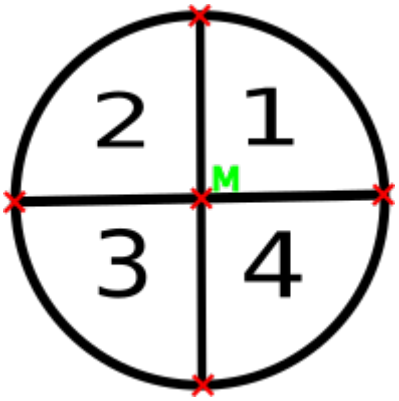


Abbildung 10: In vier Quadranten geteilter Kreis

Strategie

Für das Zeichnen von Kurven findet zuerst eine Einteilung des Kreises der Kurve in 4 Quadranten statt:

Zeichnen kann man, indem man zuerst die Formel
$r^2 = (x - m_x)^2 + (y - m_y)^2$
nach y umformt:
Quadrant 1 & und 2: $y = \sqrt{r^2 - (x - m_x)^2} + m_y$
Quadrant 3 & und 4: $y = -\sqrt{r^2 - (x - m_x)^2} + m_y$
und dann mit diesen Formeln die jeweiligen y-Koordinaten der Quadranten berechnet. Die Art und Weise diese zu berechnen unterscheidet sich aber bei den Quadranten und der Richtung:

- **Quadrant 1**
  - **Uhrzeigersinn:**

  => Die Aktuelle X-Koordinate so lange erhöhen und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm erhöhen würde) größer als der Mittelpunkt addiert mit dem Radius ist. Danach wird der aktuelle Quadrant auf 4 gesetzt.

  - **Gegen den Uhzeigersinnn:**

  => Die Aktuelle X-Koordinate so lange senken und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm senken würde) kleiner als der Mittelpunkt ist. Danach wird der aktuelle Quadrant auf 2 gesetzt.

- **Quadrant 2**
  - **Uhrzeigersinn:**

  => Die Aktuelle X-Koordinate so lange erhöhen und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm erhöhen würde) größer als der Mittelpunkt ist. Danach wird der aktuelle Quadrant auf 1 gesetzt.

  - **Gegen den Uhzeigersinnn:**

  => Die Aktuelle X-Koordinate so lange senken und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm senken würde) kleiner als der Mittelpunkt, von dem der Radius abgezogen wird, ist. Danach wird der aktuelle Quadrant auf 3 gesetzt.

- **Quadrant 3**
  - **Uhrzeigersinn:**

  => Die Aktuelle X-Koordinate so lange senken und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm senken würde) kleiner als der Mittelpunkt, von dem der Radius abgezogen wird, ist. Danach wird der aktuelle Quadrant auf 2 gesetzt.

  - **Gegen den Uhzeigersinnn:**

  => Die Aktuelle X-Koordinate so lange erhöhen und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm erhöhen würde) größer als der Mittelpunkt ist. Danach wird der aktuelle Quadrant auf 4 gesetzt.

- **Quadrant 4**
  - **Uhrzeigersinn:**

  => Die Aktuelle X-Koordinate so lange senken und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm senken würde) kleiner als der Mittelpunkt ist. Danach wird der aktuelle Quadrant auf 3 gesetzt.

  - **Gegen den Uhzeigersinnn:**

=> Die Aktuelle X-Koordinate so lange erhöhen und den Y-Wert mit der oberen Formel berechnen (mit self .GoPoint(x, y)(siehe Abschnitt 3.2) zu diesem Punkt gehen), bis die aktuelle X-Koordinate (wenn man sie jetzt noch um 1mm erhöhen würde) größer als der Mittelpunkt addiert mit dem Radius ist. Danach wird der aktuelle Quadrant auf 1 gesetzt.

Folgende Methode ermittelt in welchem Quadranten eines Kreises mit dem Mittelpunkt M(mx|my) sich der Punkt P(X|Y) befindet:

```
1  def Quadrant(self, x, y, mx, my):
2      if x >= mx:
3          if y >= my:
4              return 1   # erster Quadrant
5          elif y < my:
6              return 4   # vierter Quadrant
7      elif x < mx:
8          if y >= my:
9              return 2   # zweiter Quadrant
10         elif y < my:
11             return 3   # dritter Quadrant
```

Mit dieser Funktion lassen sich nun folgende Rechnungen anstellen:

- quadrant_start= self.Quadrant(self.x, self .y, mx, my)
- quadrant_pos= quadrant_start *#aktuelle position*
- quadrant_end= self.Quadrant(dir_x, dir_y, mx, my)

=> Der Start- und End-Quadrant werden berechnet und der aktuelle Quadrant wird auf den Start-Quadranten gesetzt.

Anschließend wird zuerst zwischen direction == 2 (im Uhrzeigersinn) und direction == 3 (gegen den Uhrzeigersinn) unterschieden:

Uhrzeigersinn

Solange der aktuelle Quadrant nicht der End-Quadrant ist, wird der Kreis weitergezeichnet. Das sieht in Programmcode wie folgt aus:

```
1  if direction == 2:
2      while quadrant_pos != quadrant_end :
3          if quadrant_pos == 1 :
4
5              arc_x= self.x + 1
6              while arc_x < mx + radius:
```

```
7                   arc_y= math.sqrt(radius**2 - (arc_x-
                        mx)**2) + my
8                   self.GoPoint(arc_x, arc_y)
9                   arc_x+= 1
10
11              arc_x= mx + radius - 0.1
12              arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
13              self.GoPoint(arc_x, arc_y)
14              quadrant_pos= 4
15
16          elif quadrant_pos == 2 :
17
18              arc_x= self.x + 1
19              while arc_x < mx:
20                  arc_y= math.sqrt(radius**2 - (arc_x-
                        mx)**2) + my
21                  self.GoPoint(arc_x, arc_y)
22                  arc_x+= 1
23
24              arc_x= mx
25              arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
26              self.GoPoint(arc_x, arc_y)
27              quadrant_pos= 1
28
29
30          elif quadrant_pos == 3:
31              arc_x= self.x - 1
32              while arc_x > mx - radius:
33                  arc_y= -math.sqrt(radius**2 - (arc_x
                        - mx)**2) + my
34                  self.GoPoint(arc_x, arc_y)
35                  arc_x-= 1
36
37              arc_x= mx - radius + 0.1
38              arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
39              self.GoPoint(arc_x, arc_y)
40              quadrant_pos= 2
41
42          elif quadrant_pos == 4:
43              arc_x= self.x - 1
44              while arc_x > mx:
```

```
45                      arc_y= -math.sqrt(radius**2 - (arc_x
                            - mx)**2) + my
46                      self.GoPoint(arc_x, arc_y)
47                      arc_x-= 1
48
49                  arc_x= mx
50                  arc_y= -math.sqrt(radius**2 - (arc_x- mx
                        )**2) + my
51                  self.GoPoint(arc_x, arc_y)
52                  quadrant_pos= 3
```

Ist der End-Quadrant erreicht, soll der Stift solange weiter zeichnen, bis er beim nächsten Schritt den Zielpunkt überschreiten würde.

```
1       if quadrant_pos == 1 or quadrant_pos == 2:
2           arc_x= self.x + 1
3           while arc_x < dir_x:
4               arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
5               self.GoPoint(arc_x, arc_y)
6               arc_x+= 1
7           self.GoPoint(dir_x, dir_y)
8
9       elif quadrant_pos == 3 or quadrant_pos == 4:
10          arc_x= self.x - 1
11          while arc_x > dir_x:
12              arc_y= -math.sqrt(radius**2 - (arc_x- mx
                    )**2) + my
13              self.GoPoint(arc_x, arc_y)
14              arc_x-= 1
15          self.GoPoint(dir_x, dir_y)
```

genauso sieht es auch gegen den Uhrzeigersinn aus. Allerdings, wie bereits erwähnt, beim Zeichnen immer in die andere Richtung:

```
1   elif direction == 3:
2       while quadrant_pos != quadrant_end :
3           if quadrant_pos == 1 :
4
5               arc_x= self.x - 1
6               while arc_x > mx:
7                   arc_y= math.sqrt(radius**2 - (arc_x-
                        mx)**2) + my
8                   self.GoPoint(arc_x, arc_y)
```

```
9                   arc_x-= 1
10
11              arc_x= mx
12              arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
13              self.GoPoint(arc_x, arc_y)
14              quadrant_pos= 2
15
16          elif quadrant_pos == 2 :
17
18              arc_x= self.x - 1
19              while arc_x > mx - radius:
20                  arc_y= math.sqrt(radius**2 - (arc_x-
                        mx)**2) + my
21                  self.GoPoint(arc_x, arc_y)
22                  arc_x-= 1
23
24              arc_x= mx - radius + 0.1
25              arc_y= math.sqrt(radius**2 - (arc_x- mx)
                    **2) + my
26              self.GoPoint(arc_x, arc_y)
27              quadrant_pos= 3
28
29
30          elif quadrant_pos == 3:
31              arc_x= self.x + 1
32              while arc_x < mx:
33                  arc_y= -math.sqrt(radius**2 - (arc_x
                        - mx)**2) + my
34                  self.GoPoint(arc_x, arc_y)
35                  arc_x+= 1
36
37              arc_x= mx
38              arc_y= -math.sqrt(radius**2 - (arc_x- mx
                    )**2) + my
39              self.GoPoint(arc_x, arc_y)
40              quadrant_pos= 4
41
42          elif quadrant_pos == 4:
43              arc_x= self.x + 1
44              while arc_x < mx + radius:
45                  arc_y= -math.sqrt(radius**2 - (arc_x
                        - mx)**2) + my
46                  self.GoPoint(arc_x, arc_y)
```

```
47                      arc_x+= 1
48
49                  arc_x= mx + radius - 0.1
50                  arc_y= math.sqrt(radius**2 - (arc_x- mx)
                        **2) + my
51                  self.GoPoint(arc_x, arc_y)
52                  quadrant_pos= 1
53
54
55
56
57          if quadrant_pos == 1 or quadrant_pos == 2:
58              arc_x= self.x - 1
59              while arc_x  > dir_x:
60                  arc_y= math.sqrt(radius**2 - (arc_x- mx)
                        **2) + my
61                  self.GoPoint(arc_x, arc_y)
62                  arc_x-= 1
63              self.GoPoint(dir_x, dir_y)
64
65          elif quadrant_pos == 3 or quadrant_pos == 4:
66              arc_x= self.x + 1
67              while arc_x < dir_x:
68                  arc_y= -math.sqrt(radius**2 - (arc_x- mx
                        )**2) + my
69                  self.GoPoint(arc_x, arc_y)
70                  arc_x+= 1
71              self.GoPoint(dir_x, dir_y)
```

3.4. Den Servo einbinden

Um den Servo zu implementieren wird wieder eine neue Klasse erstellt. Im Konstruktor wird der Servo mit dem übergebenen Pin mit Pulsweitenmodulation so angesteuert, dass der Stift angehoben wird(siehe(*Raspberry Pi Servo Motor Steuerun*)):

```
1  class Servo:
2      def __init__(self, servopin):
3          gpio.setmode(gpio.BCM)
4          gpio.setup(servopin, gpio.OUT)
5          self.p = gpio.PWM(servopin, 50)
6          self.p.start(2.5)
```

Die Funktionen

```
    def up(self):
        self.p.ChangeDutyCycle(2.5)

    def down(self):
        self.p.ChangeDutyCycle(7.5)
```

machen dem VPlotter-Objekt möglich, den Stift hochzuheben und abzusetzen. Die VPlotter Klasse erhält im Konstruktor den Pin vom Servo und erstellt ein Servo-Objekt als eigenes Attribut:

```
def __init__(self, [...], servo):
        [...]
        self.servo= Servo.Servo(servo)
```

In der Datei ReadGCode.py wird dann die Funktion

```
def TogglePen(self, richtung):
    if richtung < 0:
        self.servo.down()

    else:
        self.servo.up()
```

mit dem Z-Wert als Parameter aufgerufen.

3.5. Die G-Code-Datei auslesen

Um die G-Code-Datei auslesen zu können, wird in der neuen Datei ReadGCode.py die Methode start(file) definiert. In dieser Methode wird zuerst die als Parameter angegebene Datei file aus dem Unterverzeichnis GCode ausgelesen und in die Zeilen als Elemente in der Liste lines gespeichert.

```
a= open("GCode/" + file)
global lines
lines = a.readlines()
a.close()
```

Weiter oben in der Datei sind verschiedene Konstanten wie z.B. die Pins der Motoren vorhanden:

```
#Motor1
pin1= 0
pin2= 5
```

```
4  pin3= 6
5  pin4= 13
6
7  #Motor2
8  pin5= 12
9  pin6= 16
10 pin7= 20
11 pin8= 21
12
13 #servo
14 pinservo= 24
15
16 max_geschwindigkeit=25/6.175 # 25mm & 6.175s pro
       Umdrehung
17
18 #Origin
19 DISTANCEX= 422.5 # 42.25 cm
20 DISTANCEY= 430 # 43.0 cm
```

Mithilfe dieser Konstanten wird dann ein VPlotter-Objekt erzeugt:

```
1  plotter= VPlotter.VPlotter(DISTANCEX, DISTANCEY,
       pin1, pin2, pin3, pin4, pin5, pin6, pin7, pin8,
       max_geschwindigkeit, pinservo)
```

Anschließend wird jedes einzelne Element aus der Liste lines durchgegangen und es wird geschaut, ob es sich bei dem Anfang um ein Kommando wie z.B. G00 oder G02 handelt. Die einzelnen Parameter werden dann durch Leerzeichen getrennt und in der Liste params gespeichert:

```
1  for i in range(len(lines)):
2      line= lines[i]
3      if line.startswith("G00") or line.startswith("
           G01"):
4          params= line.split(" ")
5          if params[1].startswith("X"):
6              pass
7          elif params[1].startswith("Z"):
8              pass
9
10     elif line.startswith("G02"):
11         params= line.split(" ")
12
13     elif line.startswith("G03"):
```

```
14          params= line.split(" ")
```

3.6. Erläuterung zu den Kommandos:

- G00 & G01
  1. Ist eine X und Y Koordinate gegeben, so soll der Stift von der aktuellen Position zu der gegebenen X- und Y-Position.
     (Bsp.: G̈01 X44.902344 Y117.572270 ⇒ gehe zum Punkt(44.90 | 117.57))
  2. Ist eine Z Koordinate gegeben, so soll der Stift entweder angehoben oder abgesenkt werden.
     (Bsp.: G̈00 Z5.000000 ⇒ hebe den Stift an)
- G02

=> Der Stift soll zu der angegebenen Koordinate gehen. Dazu fährt er im Uhrzeigersinn solange um den Mittelpunkt, der durch i(X-Entfernung zur aktuellen Position) und j(Y-Entfernung zur aktuellen Position) angegebenen ist, bis er den Zielpunkt erreicht hat.
(Bsp.: G̈02 X18.342135 Y69.644791 Z-1.000000 I-0.103441 J0.000000)

- G03

=> Der Stift soll zu der angegebenen Koordinate gehen. Dazu fährt er gegen den Uhrzeigersinn solange um den Mittelpunkt, der durch i(X-Entfernung zur aktuellen Position) und j(Y-Entfernung zur aktuellen Position) angegebenen ist, bis er den Zielpunkt erreicht hat.
(Bsp.: G̈03 X15.321196 Y69.276030 Z-1.000000 I-0.618865 J-2.044972)

Jetzt müssen nur noch die Parameter an die Funktionen von plotter übergeben werden.

```
1  for i in range(len(lines)):
2      line= lines[i]
3      if line.startswith("G00") or line.startswith("
           G01"):
4          params= line.split(" ")
5          if params[1].startswith("X"):
6              plotter.GoPoint(float(params[1][1:]),
                   float(params[2][1:]))
7          elif params[1].startswith("Z"):
8              plotter.TogglePen(float(params[1][1:]))
9
```

```
10      elif line.startswith("G02"):
11          params= line.split(" ")
12          plotter.arc(2, float(params[1][1:]), float(
                params[2][1:]), float(params[4][1:]),
                float(params[5][1:]))
13
14      elif line.startswith("G03"):
15          params= line.split(" ")
16          plotter.arc(3, float(params[1][1:]), float(
                params[2][1:]), float(params[4][1:]),
                float(params[5][1:]))
```

Am Ende wird dann die Funktion start(file) mit dem Parameter sys.argv[1] aufgerufen (Dazu muss das Paket sys importiert werden: import sys). Das sorgt dafür, dass das Programm später aus der Kommandozeile ausgeführt werden kann:

pi@raspberrypi:~/Desktop $ python3 ReadGCode.py Datei.ngc

Die Datei Datei.ngc wird dann gezeichnet

3.7. Ausgaben des Programms

Mit dem Aufruf

pi@raspberrypi:~/Desktop $ python3 ReadGCode.py car_0001.ngc

Wird die Datei car_0001.ngc gezeichnet. Nach dem Ende des Programms sieht das Blatt auf dem Plotter wie folgt aus:

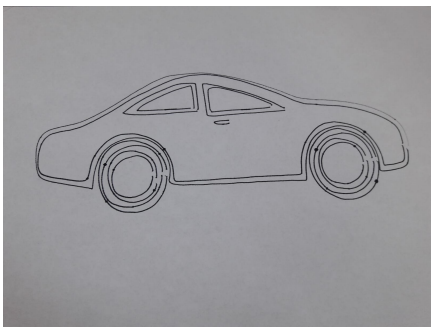


Abbildung 11: Gezeichnetes Auto

Anhang

A. GCode

car_0001.ngc

```
%
(Header)
(Generated by gcodetools from Inkscape.)
(Using default header. To add your own header create file "header" in the output dir.)
M3
(Header end.)
G21 (All units in mm)

(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X52.234375 Y164.748050

G01 Z-1.000000 F100.0(Penetrate)
G03 X42.813117 Y163.995250 Z-1.000000 I-2.098847 J-33.062733 F400.000000
G03 X37.472656 Y161.271480 Z-1.000000 I2.690845 J-11.873277
G03 X29.707521 Y149.481913 Z-1.000000 I17.587051 J-20.035633
G03 X26.119141 Y126.373050 Z-1.000000 I69.614695 J-22.642913
G02 X25.479236 Y122.976177 Z-1.000000 I-9.027208 J-0.058160
G02 X22.476562 Y117.218750 Z-1.000000 I-29.658323 J11.806017
G03 X7.466831 Y81.517296 Z-1.000000 I88.082035 J-58.037708
G03 X9.792969 Y58.685550 Z-1.000000 I36.209482 J-7.845284
G03 X13.062338 Y52.677540 Z-1.000000 I38.219224 J16.904146
G03 X21.388672 Y40.998050 Z-1.000000 I153.631523 J100.716572
G02 X31.584159 Y25.155713 Z-1.000000 I-86.196306 J-66.674327
G02 X35.011719 Y15.164060 Z-1.000000 I-30.296049 J-15.976552
G03 X37.166055 Y10.127578 Z-1.000000 I11.902554 J2.112278
G03 X40.634766 Y7.498050 Z-1.000000 I6.149934 J4.509993
G03 X45.232546 Y6.529270 Z-1.000000 I5.469954 J14.565317
G03 X51.067032 Y6.715443 Z-1.000000 I1.861997 J33.163628
G03 X56.653370 Y8.050111 Z-1.000000 I-3.002582 J24.925809
G03 X58.355469 Y9.632810 Z-1.000000 I-1.021151 J2.804791
G03 X58.689380 Y10.592210 Z-1.000000 I-5.339626 J2.396220
G03 X61.160156 Y20.705080 Z-1.000000 I-600.071395 J151.967673
G01 X63.546875 Y30.845700 Z-1.000000
G01 X60.103516 Y30.828100 Z-1.000000
G02 X48.460760 Y33.218409 Z-1.000000 I-0.142003 J28.858352
G02 X41.880859 Y38.746070 Z-1.000000 I6.012327 J13.836879
G02 X40.543637 Y50.549138 Z-1.000000 I10.935784 J7.216248
G02 X49.916016 Y60.244120 Z-1.000000 I15.814278 J-5.910308
G02 X55.790205 Y61.656019 Z-1.000000 I7.436847 J-18.015223
G02 X61.367188 Y61.166000 Z-1.000000 I1.343003 J-16.696348
G03 X62.221098 Y61.611048 Z-1.000000 I0.181282 J0.693893
G03 X63.173828 Y66.046860 Z-1.000000 I-14.951998 J5.531633
G03 X63.431231 Y71.307943 Z-1.000000 I-132.005862 J9.095330
G03 X63.949219 Y92.806620 Z-1.000000 I-2197.014101 J63.690355
G02 X64.381471 Y114.309364 Z-1.000000 I11905.445618 J-228.569410
```

```
G02 X64.457031 Y116.998030 Z-1.000000 I189.553149 J-3.981681
G01 X64.552731 Y119.687480 Z-1.000000
G01 X60.990231 Y119.669880 Z-1.000000
G02 X48.155454 Y122.219123 Z-1.000000 I-0.161082 J32.773507
G02 X41.623044 Y127.790980 Z-1.000000 I5.456249 J13.012077
G02 X39.592357 Y134.599096 Z-1.000000 I11.499558 J7.136933
G02 X41.285153 Y141.728480 Z-1.000000 I14.386938 J0.349636
G02 X48.017780 Y148.332642 Z-1.000000 I13.873848 J-7.409848
G02 X58.871091 Y150.689410 Z-1.000000 I9.902178 J-19.432141
G01 X62.804685 Y150.517540 Z-1.000000
G01 X62.433591 Y155.607380 Z-1.000000
G03 X60.709510 Y161.934912 Z-1.000000 I-17.046415 J-1.246039
G03 X58.382809 Y163.804650 Z-1.000000 I-3.176230 J-1.569957
G03 X55.282922 Y164.420934 Z-1.000000 I-8.260807 J-33.447284
G03 X52.234372 Y164.748010 Z-1.000000 I-5.282733 J-34.867569
G01 X52.234375 Y164.748050 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X49.722656 Y162.902340

G01 Z-1.000000 F100.0(Penetrate)
G02 X55.770385 Y162.615341 Z-1.000000 I0.284506 J-57.868011 F400.000000
G02 X57.859375 Y162.087890 Z-1.000000 I-0.729697 J-7.290490
G02 X59.201558 Y160.761734 Z-1.000000 I-0.967501 J-2.321474
G02 X60.111328 Y156.865230 Z-1.000000 I-10.405006 J-4.483858
G01 X60.339844 Y152.486330 Z-1.000000
G01 X57.064453 Y152.468730 Z-1.000000
G03 X43.541625 Y147.545100 Z-1.000000 I0.104777 J-21.319918
G03 X37.673828 Y136.351540 Z-1.000000 I10.508443 J-12.643414
G03 X42.180818 Y124.827605 Z-1.000000 I13.850799 J-1.226283
G03 X57.708984 Y118.171860 Z-1.000000 I17.418740 J19.196897
G01 X62.544922 Y117.818340 Z-1.000000
G01 X62.449222 Y116.060530 Z-1.000000
G03 X62.377504 Y114.301313 Z-1.000000 I63.453423 J-3.467884
G03 X61.716800 Y88.835920 Z-1.000000 I13379.019731 J-359.862094
G02 X61.037328 Y63.398517 Z-1.000000 I-7405.517401 J185.084930
G02 X60.992191 Y63.333960 Z-1.000000 I-0.071648 J0.002037
G02 X60.900887 Y63.314869 Z-1.000000 I-0.096706 J0.234622
G02 X56.648441 Y63.242160 Z-1.000000 I-10.797155 J507.091950
G03 X45.034694 Y59.396056 Z-1.000000 I0.259526 J-20.241233
G03 X38.398441 Y50.025360 Z-1.000000 I10.458942 J-14.442159
G03 X41.312458 Y37.108296 Z-1.000000 I12.996013 J-3.855398
G03 X57.009769 Y29.318330 Z-1.000000 I17.657499 J15.870505
G02 X60.439683 Y28.500917 Z-1.000000 I-0.965723 J-11.656998
G02 X60.794925 Y27.783170 Z-1.000000 I-0.226674 J-0.558976
G03 X60.483698 Y26.584108 Z-1.000000 I48.192553 J-13.148709
G03 X58.560550 Y18.712860 Z-1.000000 I2044.824741 J-503.773817
G02 X56.693398 Y12.079460 Z-1.000000 I-91.334124 J22.129003
G02 X55.444658 Y9.937674 Z-1.000000 I-5.486219 J1.763744
```

```
G02 X53.426422 Y8.799162 Z-1.000000 I-2.770886 J2.553825
G02 X48.902347 Y8.234350 Z-1.000000 I-5.927990 J29.081417
G02 X39.724990 Y10.502023 Z-1.000000 I-0.783087 J16.535205
G02 X36.992191 Y15.242160 Z-1.000000 I2.811579 J4.778770
G03 X34.949708 Y23.501500 Z-1.000000 I-18.234359 J-0.127026
G03 X23.296878 Y41.681610 Z-1.000000 I-104.240197 J-53.989819
G02 X11.761336 Y60.143918 Z-1.000000 I82.987157 J64.686635
G02 X8.503909 Y71.931610 Z-1.000000 I29.827457 J14.586487
G02 X10.709324 Y89.080935 Z-1.000000 I40.885264 J3.458598
G02 X24.167972 Y116.185520 Z-1.000000 I118.473154 J-41.933554
G03 X27.385709 Y122.496970 Z-1.000000 I-26.934912 J17.708068
G03 X28.138675 Y126.687470 Z-1.000000 I-10.908298 J4.122945
G02 X29.916015 Y141.276632 Z-1.000000 I57.881861 J0.351326
G02 X35.324222 Y155.003880 Z-1.000000 I56.553678 J-14.351819
G02 X38.271663 Y159.239471 Z-1.000000 I21.402888 J-11.750408
G02 X40.740237 Y161.255830 Z-1.000000 I7.035551 J-6.094153
G02 X43.751809 Y162.345059 Z-1.000000 I4.759727 J-8.452100
G02 X49.722659 Y162.902310 Z-1.000000 I6.138037 J-33.501058
G01 X49.722656 Y162.902340 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X57.035156 Y149.636720

G01 Z-1.000000 F100.0(Penetrate)
G03 X51.484665 Y148.663856 Z-1.000000 I0.576203 J-19.607486 F400.000000
G03 X46.537109 Y146.087890 Z-1.000000 I5.387945 J-16.387685
G03 X42.254098 Y141.104277 Z-1.000000 I9.084126 J-12.139326
G03 X40.869141 Y135.500000 Z-1.000000 I10.316812 J-5.522809
G03 X41.997240 Y130.423667 Z-1.000000 I12.373450 J0.086203
G03 X45.253906 Y125.947270 Z-1.000000 I12.750980 J5.853744
G03 X59.390781 Y121.104375 Z-1.000000 I12.590942 J13.699274
G03 X71.695312 Y128.152340 Z-1.000000 I-1.391911 J16.694814
G03 X73.617115 Y131.266866 Z-1.000000 I-24.627973 J17.346752
G03 X74.119141 Y132.671880 Z-1.000000 I-4.431920 J2.375767
G03 X73.245291 Y140.530905 Z-1.000000 I-12.914186 J2.542158
G03 X67.917969 Y146.714840 Z-1.000000 I-12.828727 J-5.665010
G03 X62.643204 Y149.003335 Z-1.000000 I-9.958530 J-15.730308
G03 X57.035156 Y149.636720 Z-1.000000 I-5.021712 J-19.318912
G01 X57.035156 Y149.636720 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X56.746094 Y147.894530
```

```
G01 Z-1.000000 F100.0(Penetrate)
G02 X62.726904 Y147.050991 Z-1.000000 I0.661630 J-16.933115 F400.000000
G02 X68.333984 Y143.707030 Z-1.000000 I-4.727509 J-14.299875
G02 X70.644213 Y141.002382 Z-1.000000 I-11.358168 J-12.040790
G02 X71.750000 Y138.705080 Z-1.000000 I-7.014148 J-4.790985
G02 X71.833548 Y132.979193 Z-1.000000 I-9.788632 J-3.006381
G02 X68.691406 Y127.601560 Z-1.000000 I-11.679013 J3.217237
G02 X58.198453 Y123.020776 Z-1.000000 I-11.144768 J11.220504
G02 X47.248047 Y126.716800 Z-1.000000 I-0.664211 J16.101794
G02 X43.900888 Y131.190398 Z-1.000000 I7.620267 J9.190484
G02 X43.013672 Y136.564450 Z-1.000000 I10.416520 J4.479951
G02 X47.518437 Y144.663587 Z-1.000000 I11.113533 J-0.879030
G02 X56.746094 Y147.894530 Z-1.000000 I9.878239 J-13.419821
G01 X56.746094 Y147.894530 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X56.146484 Y145.646480

G01 Z-1.000000 F100.0(Penetrate)
G03 X53.674522 Y145.197211 Z-1.000000 I1.294368 J-14.147097 F400.000000
G03 X51.271484 Y144.277340 Z-1.000000 I3.512498 J-12.774661
G03 X45.830637 Y135.117367 Z-1.000000 I4.394134 J-8.805897
G03 X51.912109 Y126.468750 Z-1.000000 I9.686378 J0.348726
G03 X62.145516 Y126.030873 Z-1.000000 I5.738965 J14.323647
G03 X68.791016 Y131.607420 Z-1.000000 I-3.224507 J10.590556
G03 X69.678197 Y135.610283 Z-1.000000 I-7.332015 J3.724790
G03 X68.511719 Y139.859380 Z-1.000000 I-9.492074 J-0.321137
G03 X63.542947 Y144.427216 Z-1.000000 I-9.253921 J-5.079805
G03 X56.146484 Y145.646480 Z-1.000000 I-6.040581 J-13.599843
G01 X56.146484 Y145.646480 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X58.158203 Y143.951170

G01 Z-1.000000 F100.0(Penetrate)
G02 X62.575778 Y142.799447 Z-1.000000 I-0.518587 J-11.037041 F400.000000
G02 X66.250000 Y139.492190 Z-1.000000 I-4.143417 J-8.297741
G02 X67.529396 Y136.986548 Z-1.000000 I-12.363844 J-7.892511
G02 X67.808594 Y135.546880 Z-1.000000 I-3.717692 J-1.467886
G02 X67.549077 Y134.103509 Z-1.000000 I-3.989604 J-0.027689
G02 X66.304688 Y131.585940 Z-1.000000 I-13.649698 J5.180477
```

```
G02 X54.824477 Y126.652479 Z-1.000000 I-9.272902 J5.754120
G02 X47.609375 Y135.447270 Z-1.000000 I1.829400 J8.857781
G02 X50.994651 Y141.918613 Z-1.000000 I7.774488 J0.054146
G02 X58.158203 Y143.951170 Z-1.000000 I6.613718 J-9.669510
G01 X58.158203 Y143.951170 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X28.847656 Y114.136720

G01 Z-1.000000 F100.0(Penetrate)
G01 X27.187500 Y112.429690 Z-1.000000 F400.000000
G03 X22.410526 Y106.154185 Z-1.000000 I23.383833 J-22.755886
G03 X16.375000 Y93.955080 Z-1.000000 I83.921596 J-49.112936
G03 X11.352832 Y79.340928 Z-1.000000 I122.507365 J-50.269791
G03 X11.847656 Y77.755860 Z-1.000000 I1.469788 J-0.410934
G03 X12.139902 Y77.659422 Z-1.000000 I0.268875 J0.323766
G03 X20.623047 Y78.244140 Z-1.000000 I-17.812467 J320.254320
G01 X29.142578 Y78.945310 Z-1.000000
G01 X28.996094 Y96.541020 Z-1.000000
G01 X28.847656 Y114.136720 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X26.628906 Y108.183590

G01 Z-1.000000 F100.0(Penetrate)
G02 X26.775048 Y108.042149 Z-1.000000 I0.000709 J-0.145488 F400.000000
G02 X27.013672 Y94.322270 Z-1.000000 I-657.956739 J-18.305571
G02 X26.542616 Y81.001920 Z-1.000000 I-157.538802 J-1.097364
G02 X25.931641 Y80.414060 Z-1.000000 I-0.635911 J0.049487
G03 X24.754649 Y80.347801 Z-1.000000 I1.335712 J-34.213813
G03 X19.537109 Y79.943360 Z-1.000000 I49.114793 J-667.468277
G02 X14.748720 Y79.984970 Z-1.000000 I-2.163537 J26.564616
G02 X14.314453 Y80.460940 Z-1.000000 I0.047442 J0.479379
G02 X15.040927 Y84.988245 Z-1.000000 I13.868092 J0.096596
G02 X19.931641 Y97.117190 Z-1.000000 I106.703440 J-35.975166
G02 X25.544073 Y107.569785 Z-1.000000 I104.449288 J-49.350085
G02 X26.628906 Y108.183590 Z-1.000000 I1.088820 J-0.658806
G01 X26.628906 Y108.183590 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)
```

```
(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X33.449219 Y88.978520

G01 Z-1.000000 F100.0(Penetrate)
G03 X32.879326 Y88.538706 Z-1.000000 I0.002986 J-0.592999 F400.000000
G03 X32.464844 Y85.296880 Z-1.000000 I11.805929 J-3.156849
G03 X32.923722 Y82.057203 Z-1.000000 I12.269015 J0.085485
G03 X33.500000 Y81.623050 Z-1.000000 I0.573279 J0.161408
G03 X34.070084 Y82.063123 Z-1.000000 I-0.002987 J0.593158
G03 X34.484375 Y85.306640 Z-1.000000 I-11.823500 J3.158419
G03 X34.025339 Y88.544598 Z-1.000000 I-12.251323 J-0.085320
G03 X33.449219 Y88.978520 Z-1.000000 I-0.573121 J-0.161516
G01 X33.449219 Y88.978520 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X27.314453 Y75.160160

G01 Z-1.000000 F100.0(Penetrate)
G02 X26.123869 Y74.964186 Z-1.000000 I-1.839707 J7.462112 F400.000000
G02 X18.902344 Y74.447270 Z-1.000000 I-23.334965 J275.296412
G03 X11.874471 Y73.827653 Z-1.000000 I6.958189 J-119.087675
G03 X11.667969 Y73.593750 Z-1.000000 I0.027427 J-0.232321
G03 X12.610573 Y68.052230 Z-1.000000 I17.475750 J0.121670
G03 X17.867188 Y55.494140 Z-1.000000 I97.033039 J33.237263
G03 X22.114439 Y48.434810 Z-1.000000 I46.824985 J23.364938
G03 X26.699219 Y43.134770 Z-1.000000 I30.786753 J21.998926
G01 X28.730469 Y41.185550 Z-1.000000
G01 X28.611328 Y58.318360 Z-1.000000
G03 X27.923090 Y74.720043 Z-1.000000 I-234.750567 J-1.635184
G03 X27.314453 Y75.160160 Z-1.000000 I-0.490796 J-0.037821
G01 X27.314453 Y75.160160 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X26.488281 Y73.101560

G01 Z-1.000000 F100.0(Penetrate)
G01 X26.580081 Y59.734380 Z-1.000000 F400.000000
G02 X26.505202 Y48.720620 Z-1.000000 I-400.100393 J-2.786969
```

```
G02 X26.049441 Y47.355970 Z-1.000000 I-2.419929 J0.049767
G02 X25.235727 Y47.359786 Z-1.000000 I-0.405496 J0.292002
G02 X21.650394 Y52.890620 Z-1.000000 I68.045105 J48.037374
G02 X17.330421 Y61.652398 Z-1.000000 I54.264089 J32.200640
G02 X14.697269 Y70.271480 Z-1.000000 I49.814127 J19.930104
G01 X14.257815 Y72.308590 Z-1.000000
G01 X17.435550 Y72.689450 Z-1.000000
G02 X20.619540 Y72.983293 Z-1.000000 I6.915502 J-57.536869
G02 X23.550784 Y73.085940 Z-1.000000 I3.208939 J-49.731922
G01 X26.488281 Y73.101560 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X57.988281 Y60.287110

G01 Z-1.000000 F100.0(Penetrate)
G03 X52.144719 Y59.352815 Z-1.000000 I0.073670 J-19.202233 F400.000000
G03 X46.972656 Y56.607420 Z-1.000000 I5.189342 J-16.020763
G03 X42.794193 Y51.697106 Z-1.000000 I9.838034 J-12.604741
G03 X41.503906 Y46.646480 Z-1.000000 I9.166171 J-5.031816
G03 X42.571886 Y41.212224 Z-1.000000 I14.516107 J0.030740
G03 X45.441406 Y36.957030 Z-1.000000 I11.549835 J4.693576
G03 X50.844174 Y33.548222 Z-1.000000 I10.804674 J11.138853
G03 X58.089844 Y32.261720 Z-1.000000 I7.161363 J19.285988
G03 X68.956497 Y35.526847 Z-1.000000 I-0.081355 J19.985949
G03 X74.589844 Y43.064450 Z-1.000000 I-7.531323 J11.502540
G03 X74.348008 Y50.414260 Z-1.000000 I-11.011806 J3.316554
G03 X69.484375 Y56.695310 Z-1.000000 I-12.563826 J-4.705038
G03 X63.866328 Y59.428001 Z-1.000000 I-11.350384 J-16.193547
G03 X57.988281 Y60.287110 Z-1.000000 I-5.801217 J-19.153657
G01 X57.988281 Y60.287110 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X57.363281 Y58.472660

G01 Z-1.000000 F100.0(Penetrate)
G02 X66.498666 Y56.359417 Z-1.000000 I0.545870 J-18.442649 F400.000000
G02 X72.130859 Y49.892580 Z-1.000000 I-5.519631 J-10.493294
G02 X72.850155 Y45.915000 Z-1.000000 I-11.393157 J-4.114139
G02 X72.212889 Y42.634770 Z-1.000000 I-8.280343 J-0.093355
G02 X63.619054 Y34.679109 Z-1.000000 I-12.995762 J5.418804
G02 X50.681639 Y35.548830 Z-1.000000 I-5.395832 J16.394268
G02 X45.665877 Y39.678908 Z-1.000000 I6.054881 J12.464061
```

```
G02 X43.652342 Y44.884770 Z-1.000000 I8.175366 J6.154418
G02 X44.315337 Y50.134260 Z-1.000000 I12.154366 J1.131551
G02 X47.056639 Y54.384770 Z-1.000000 I10.500353 J-3.762807
G02 X51.838684 Y57.397069 Z-1.000000 I9.557638 J-9.870901
G02 X57.363279 Y58.472660 Z-1.000000 I6.038618 J-16.290485
G01 X57.363281 Y58.472660 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X60.931641 Y56.105470

G01 Z-1.000000 F100.0(Penetrate)
G03 X52.638046 Y55.433730 Z-1.000000 I-2.839026 J-16.482197 F400.000000
G03 X47.279297 Y50.941410 Z-1.000000 I3.190652 J-9.248332
G03 X45.959741 Y45.958019 Z-1.000000 I9.145287 J-5.087987
G03 X47.187497 Y41.628910 Z-1.000000 I7.965444 J-0.079615
G03 X53.771445 Y36.596007 Z-1.000000 I10.070988 J6.351725
G03 X62.937497 Y36.804690 Z-1.000000 I4.268148 J13.934821
G03 X66.787325 Y38.971222 Z-1.000000 I-4.391851 J12.307866
G03 X69.214841 Y41.906250 Z-1.000000 I-6.126271 J7.538338
G03 X69.923127 Y48.332882 Z-1.000000 I-7.600129 J4.089965
G03 X65.757809 Y54.197270 Z-1.000000 I-10.282330 J-2.891813
G03 X63.505500 Y55.395259 Z-1.000000 I-6.054531 J-8.666724
G03 X60.931638 Y56.105470 Z-1.000000 I-4.874661 J-12.647114
G01 X60.931641 Y56.105470 Z-1.000000
G00 Z5.000000

(End cutting path id: path34)


(Start cutting path id: path34)
(Change tool to Default tool)

G00 Z5.000000
G00 X57.111328 Y54.451170

G01 Z-1.000000 F100.0(Penetrate)
G02 X63.410786 Y53.323120 Z-1.000000 I0.956207 J-12.813482 F400.000000
G02 X67.123047 Y49.873050 Z-1.000000 I-3.350468 J-7.327313
G02 X67.959417 Y44.951878 Z-1.000000 I-6.679492 J-3.666860
G02 X65.568359 Y40.562500 Z-1.000000 I-7.531953 J1.256997
G02 X54.186323 Y38.183530 Z-1.000000 I-7.709412 J8.467400
G02 X48.224609 Y46.435550 Z-1.000000 I2.814869 J8.313162
G02 X48.837022 Y49.209179 Z-1.000000 I6.385611 J0.044491
G02 X50.773438 Y51.828120 Z-1.000000 I7.595168 J-3.590430
G02 X53.582343 Y53.611854 Z-1.000000 I6.500095 J-7.132408
G02 X57.111328 Y54.451170 Z-1.000000 I4.381384 J-10.583317
G01 X57.111328 Y54.451170 Z-1.000000
G00 Z5.000000
```

```
(End cutting path id: path34)


(Footer)
M5
G00 X0.0000 Y0.0000
M2
(Using default footer. To add your own footer create file "footer" in the output dir.)
(end)
%
```

B. Quellcode

ReadGCode.py

```
1  # python3 ReadGCode.py <file>
2  import VPlotter
3  import sys
4
5  #Motor1
6  pin1= 0
7  pin2= 5
8  pin3= 6
9  pin4= 13
10
11 #Motor2
12 pin5= 12
13 pin6= 16
14 pin7= 20
15 pin8= 21
16
17 #servo
18 pinservo= 24
19
20 max_geschwindigkeit=25/6.175       # 25mm  &  6.175 s  pro Umdrehung
21
22 #Origin
23 DISTANCEX= 422.5 # 42.25 cm
24 DISTANCEY= 430 # 43.0 cm
25
26
27 def start(file):
28
29     a= open("GCode/" + file)
30     global lines
31     lines = a.readlines()
32     a.close()
33
34     plotter=  VPlotter.VPlotter(DISTANCEX, DISTANCEY, pin1, pin2, pin3,
           pin4, pin5, pin6, pin7, pin8, max_geschwindigkeit, pinservo)
35     plotter.GoOrigin()
36
```

```
37      for i in range(len(lines)):
38          line= lines[i]
39          if line.startswith("G00") or line.startswith("G01"):
40              #print("Line" + str(i + 1) + ": " + lines[i])
41              params= line.split(" ")
42              if params[1].startswith("X"):
43                  plotter.GoPoint(float(params[1][1:]), float(params
                        [2][1:]))
44              elif params[1].startswith("Z"):
45                  plotter.TogglePen(float(params[1][1:]))
46
47          elif line.startswith("G02"):
48              params= line.split(" ")
49              plotter.arc(2, float(params[1][1:]), float(params[2][1:]),
                    float(params[4][1:]), float(params[5][1:]))
50
51          elif line.startswith("G03"):
52              params= line.split(" ")
53              plotter.arc(3, float(params[1][1:]), float(params[2][1:]),
                    float(params[4][1:]), float(params[5][1:]))
54
55
56
57  start(sys.argv[1])
58
59
60
61
62
63
64
65
66
67  '''
68  for line in lines:
69          #if line.startswith("G00"):
70          print("Line " + str(lines.index(line)) + ": " + line)
71  '''
```

Motor_own.py

```
1   import time
2   import RPi.GPIO as GPIO
3
4
5   '''
6   A = 7
7   B = 11
8   C = 13
9   D = 15
10
11
12
13          1   2   3   4   5   6   7   8
```

```
14
15  Pin1   x   x                    x
16  Pin2       x   x   x
17  Pin3           x   x   x
18  Pin4               x   x   x
19
20
21  '''
22  class Motor:
23      A = 0
24      B = 5
25      C = 6
26      D = 13
27
28      def __init__(self,A,B,C,D):
29          GPIO.setmode(GPIO.BCM)
30          GPIO.setwarnings(False)
31
32          GPIO.setup(A, GPIO.OUT)
33          GPIO.setup(B, GPIO.OUT)
34          GPIO.setup(C, GPIO.OUT)
35          GPIO.setup(D, GPIO.OUT)
36          self.A = A
37          self.B = B
38          self.C = C
39          self.D = D
40
41      def __del__(self):
42          self.GPIO_SETUP(0,0,0,0, 1, 512 * 8)
43
44
45
46      def GPIO_SETUP(self, a,b,c,d,time1, degrees):   # degrees = Anzahl
            der Aufrufe von GPIO_SETUP in einem Aufruf von rt() oder lt()
47          GPIO.output(self.A, a)
48          GPIO.output(self.B, b)
49          GPIO.output(self.C, c)
50          GPIO.output(self.D, d)
51          time.sleep(time1 / degrees)
52
53      def rt(self, deg, time):
54
55          '''
56          siehe "http://robocraft.ru/files/datasheet/28BYJ-48.pdf"
57          unter Stride Angle: 5.625° /64 => 0.087890625  (Umdrehung in °
                bei einem Schritt)
58          0.087890625 * 8 Schritte = 0.703125 für eine interne Umdrehung
59          360 / 0.703125 = 512.0 Schritte für eine Umdrehung
60          '''
61          full_circle = 512.0
62          degree = (full_circle/360)*deg     #CHANGE
63          degree_save= (full_circle/360)*deg    #CHANGE
64          self.GPIO_SETUP(0,0,0,0, time, degree_save * 8)
65
```

```
66          while degree > 0.0:
67              self.GPIO_SETUP(1,0,0,0, time, degree_save * 8)
68              self.GPIO_SETUP(1,1,0,0, time, degree_save * 8)
69              self.GPIO_SETUP(0,1,0,0, time, degree_save * 8)
70              self.GPIO_SETUP(0,1,1,0, time, degree_save * 8)
71              self.GPIO_SETUP(0,0,1,0, time, degree_save * 8)
72              self.GPIO_SETUP(0,0,1,1, time, degree_save * 8)
73              self.GPIO_SETUP(0,0,0,1, time, degree_save * 8)
74              self.GPIO_SETUP(1,0,0,1, time, degree_save * 8)
75              degree -= 1
76          self.GPIO_SETUP(0,0,0,0, time, degree_save * 8)
77
78      def lt(self, deg, time):
79
80          full_circle = 512.0
81          degree = (full_circle/360)*deg        #CHANGE
82          degree_save= (full_circle/360)*deg    #CHANGE
83          self.GPIO_SETUP(0,0,0,0, time, degree_save * 8)
84
85          while degree > 0.0:
86              self.GPIO_SETUP(1,0,0,1, time, degree_save * 8)
87              self.GPIO_SETUP(0,0,0,1, time, degree_save * 8)
88              self.GPIO_SETUP(0,0,1,1, time, degree_save * 8)
89              self.GPIO_SETUP(0,0,1,0, time, degree_save * 8)
90              self.GPIO_SETUP(0,1,1,0, time, degree_save * 8)
91              self.GPIO_SETUP(0,1,0,0, time, degree_save * 8)
92              self.GPIO_SETUP(1,1,0,0, time, degree_save * 8)
93              self.GPIO_SETUP(1,0,0,0, time, degree_save * 8)
94              degree -= 1
95          self.GPIO_SETUP(0,0,0,0, time, degree_save * 8)
96
97  #MAIN ############################
```

VPlotter.py

```
1  import math
2  from threading import Thread
3  import Motor_own
4  import time
5  import Servo
6
7  class VPlotter:
8
9      x= 0 #actual position
10     y= 0 #actual position
11     distanceX= 0
12     distanceY= 0
13     max_geschwindigkeit= 0
14
15
```

```
16      def __init__(self, distanceBetweenMotors, distanceYtoOrigin, pin1,
            pin2, pin3, pin4, pin5, pin6, pin7, pin8, geschw, servo):
17          self.distanceX= distanceBetweenMotors
18          self.distanceY= distanceYtoOrigin
19          self.motor1 = Motor_own.Motor(pin1, pin2, pin3, pin4)
20          self.motor2 = Motor_own.Motor(pin5, pin6, pin7, pin8)
21          self.xa= self.distanceX / 2
22          self.xb= self.distanceX / 2
23          self.max_geschwindigkeit= geschw
24          self.x= 0
25          self.y= 0
26
27          self.servo= Servo.Servo(servo)
28
29
30      def GoOrigin(self):
31          pass
32
33      def TogglePen(self, richtung):
34          if richtung < 0:
35              print("Stift unten")   # später Stift herunterfahren
36              self.servo.down()
37
38          else:
39              print("Stift oben") # später Stift hochfahren
40              self.servo.up()
41
42      def GoPoint(self, x, y):
43
44          a1= math.sqrt((self.xa + self.x) ** 2 + (self.distanceY - self.y)
                ** 2)
45          b1= math.sqrt((self.xb - self.x) ** 2 + (self.distanceY - self.y)
                ** 2)
46
47          self.x= x
48          self.y= y
49          print("X: " + str(self.x), "Y: " + str(self.y))
50
51          a2= math.sqrt((self.xa + x) ** 2 + (self.distanceY - y) ** 2)
52          b2= math.sqrt((self.xb - x) ** 2 + (self.distanceY - y) ** 2)
53
54          print("A ",a2-a1)
55          print("B ", b2-b1)
56
57          if abs(a2 - a1) >= abs(b2 - b1):
58              t1 = Thread(target = self.MotorBewegen, args=(1, a2 - a1, (a2
                    - a1) / self.max_geschwindigkeit,))
59              t2 = Thread(target = self.MotorBewegen, args=(2, b2 - b1, (a2
                    - a1) / self.max_geschwindigkeit,))
60              t1.start()
61              t2.start()
```

```
62
63              t1.join() # look, if thread is finished
64              t2.join() # look, if thread is finished
65
66              #time.sleep(abs((a2 - a1) / self.max_geschwindigkeit) + 3)
                    #3 ERSETZEN undzwar gucken, ob thread vorbei
67
68
69          else:
70              t1 = Thread(target = self.MotorBewegen, args=(1, a2 - a1, (b2
                    - b1) / self.max_geschwindigkeit,))
71              t2 = Thread(target = self.MotorBewegen, args=(2, b2 - b1, (b2
                    - b1) / self.max_geschwindigkeit,))
72              t1.start()
73              t2.start()
74
75              t1.join() # look, if thread is finished
76              t2.join() # look, if thread is finished
77
78              #time.sleep(abs((b2 - b1) / self.max_geschwindigkeit) + 3)
                    #3 ERSETZEN undzwar gucken, ob thread vorbei
79
80
81
82      def MotorBewegen(self, motor, length, time):
83          if length == 0:
84              return
85          if motor == 1:
86              if length >= 0:
87                  self.motor1.lt(length/(25/360), abs(time))    #25mm pro
                        360° also: 25/360 mm pro grad
88              else:
89                  self.motor1.rt(-length/(25/360), abs(time))
90
91          elif motor == 2:
92              if length >= 0:
93                  self.motor2.rt(length/(25/360), abs(time))
94              else:
95                  self.motor2.lt(-length/(25/360), abs(time))
96
97
98
99
100
101
102
103
104
105     def Quadrant(self, x, y, mx, my):
106         if x >= mx:
107             if y >= my:
108                 return 1  # erster Quadrant
```

```
109             elif y < my:
110                 return 4  # vierter Quadrant
111         elif x < mx:
112             if y >= my:
113                 return 2  # zweiter Quadrant
114             elif y < my:
115                 return 3  # dritter Quadrant
116
117
118
119     def arc(self, direction, dir_x, dir_y, i, j):
120             mx= self.x + i
121             my= self.y + j
122             radius= math.sqrt(i ** 2 + j ** 2)
123             print("Radius:", radius)
124
125             #for x in range(self.x, dir_x, 0.1):
126
127             if direction == 2:
128                 print("G02")
129                 print("X AND Y", self.x, self.y)
130
131
132                 quadrant_start= self.Quadrant(self.x, self.y, mx, my)
133                 quadrant_pos= quadrant_start # aktuelle position
134                 quadrant_end= self.Quadrant(dir_x, dir_y, mx, my)
135
136                 while quadrant_pos != quadrant_end :            #geht bis
                        zum Anfang des letzen Quadrantens
137                     if quadrant_pos == 1 :
138
139                         arc_x= self.x + 1
140                         while arc_x < mx + radius:
141                             arc_y= math.sqrt(radius**2 - (arc_x- mx)**2)
                                + my
142                             self.GoPoint(arc_x, arc_y)
143                             arc_x+= 1
144
145                         arc_x= mx + radius - 0.1
146                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
147                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                            WIRKLICH BIS ZUM ENDE DES ERSTEN QUADRANTEN
148                         quadrant_pos= 4
149
150                     elif quadrant_pos == 2 :
151
152                         arc_x= self.x + 1
153                         while arc_x < mx:
154                             arc_y= math.sqrt(radius**2 - (arc_x- mx)**2)
                                + my
155                             self.GoPoint(arc_x, arc_y)
156                             arc_x+= 1
```

```
157
158                         arc_x= mx
159                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
160                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                                WIRKLICH BIS ZUM ENDE DES ZWEITEN QUADRANTEN
161                         quadrant_pos= 1
162
163
164                     elif quadrant_pos == 3:
165                         arc_x= self.x - 1
166                         while arc_x > mx - radius:
167                             arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2)
                                    + my
168                             self.GoPoint(arc_x, arc_y)
169                             arc_x-= 1
170
171                         arc_x= mx - radius + 0.1
172                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
173                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                                WIRKLICH BIS ZUM ENDE DES DRITTEN QUADRANTEN
174                         quadrant_pos= 2
175
176                     elif quadrant_pos == 4:
177                         arc_x= self.x - 1
178                         while arc_x > mx:
179                             arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2)
                                    + my
180                             self.GoPoint(arc_x, arc_y)
181                             arc_x-= 1
182
183                         arc_x= mx
184                         arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2) +
                                my
185                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                                WIRKLICH BIS ZUM ENDE DES VIERTEN QUADRANTEN
186                         quadrant_pos= 3
187
188
189
190
191                 if quadrant_pos == 1 or quadrant_pos == 2:
192                     arc_x= self.x + 1
193                     while arc_x < dir_x:
194                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
195                         self.GoPoint(arc_x, arc_y)
196                         arc_x+= 1
197                     self.GoPoint(dir_x, dir_y)                  #GEHE AUCH
                            WIRKLICH BIS ZUM PUNKT
198
199                 elif quadrant_pos == 3 or quadrant_pos == 4:
200                     arc_x= self.x - 1
201                     while arc_x > dir_x:
202                         arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2) +
```

```
                    my
203                 self.GoPoint(arc_x, arc_y)
204                 arc_x-= 1
205             self.GoPoint(dir_x, dir_y)              #GEHE AUCH
                    WIRKLICH BIS ZUM PUNKT
206
207
208
209
210
211         elif direction == 3:
212             print("G03")
213             print("X AND Y", self.x, self.y)
214
215
216             quadrant_start= self.Quadrant(self.x, self.y, mx, my)
217             quadrant_pos= quadrant_start # aktuelle position
218             quadrant_end= self.Quadrant(dir_x, dir_y, mx, my)
219
220             while quadrant_pos != quadrant_end :            #geht bis
                    zum Anfang des letzen Quadrantens
221                 if quadrant_pos == 1 :
222
223                     arc_x= self.x - 1
224                     while arc_x > mx:
225                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2)
                                + my
226                         self.GoPoint(arc_x, arc_y)
227                         arc_x-= 1
228
229                     arc_x= mx
230                     arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
231                     self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                            WIRKLICH BIS ZUM ENDE DES ERSTEN QUADRANTEN
232                     quadrant_pos= 2
233
234                 elif quadrant_pos == 2 :
235
236                     arc_x= self.x - 1
237                     while arc_x > mx - radius:
238                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2)
                                + my
239                         self.GoPoint(arc_x, arc_y)
240                         arc_x-= 1
241
242                     arc_x= mx - radius + 0.1
243                     arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
244                     self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                            WIRKLICH BIS ZUM ENDE DES ZWEITEN QUADRANTEN
245                     quadrant_pos= 3
246
247
248                 elif quadrant_pos == 3:
```

```
249                         arc_x= self.x + 1
250                         while arc_x < mx:
251                             arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2)
                                    + my
252                             self.GoPoint(arc_x, arc_y)
253                             arc_x+= 1
254
255                         arc_x= mx
256                         arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2) +
                                my
257                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                                WIRKLICH BIS ZUM ENDE DES DRITTEN QUADRANTEN
258                         quadrant_pos= 4
259
260                     elif quadrant_pos == 4:
261                         arc_x= self.x + 1
262                         while arc_x < mx + radius:
263                             arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2)
                                    + my
264                             self.GoPoint(arc_x, arc_y)
265                             arc_x+= 1
266
267                         arc_x= mx + radius - 0.1
268                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
269                         self.GoPoint(arc_x, arc_y)              #GEHE AUCH
                                WIRKLICH BIS ZUM ENDE DES VIERTEN QUADRANTEN
270                         quadrant_pos= 1
271
272
273
274
275                 if quadrant_pos == 1 or quadrant_pos == 2:
276                     arc_x= self.x - 1
277                     while arc_x  > dir_x:
278                         arc_y= math.sqrt(radius**2 - (arc_x- mx)**2) + my
279                         self.GoPoint(arc_x, arc_y)
280                         arc_x-= 1
281                     self.GoPoint(dir_x, dir_y)                  #GEHE AUCH
                            WIRKLICH BIS ZUM PUNKT
282
283                 elif quadrant_pos == 3 or quadrant_pos == 4:
284                     arc_x= self.x + 1
285                     while arc_x < dir_x:
286                         arc_y= -math.sqrt(radius**2 - (arc_x- mx)**2) +
                                my
287                         self.GoPoint(arc_x, arc_y)
288                         arc_x+= 1
289                     self.GoPoint(dir_x, dir_y)                  #GEHE AUCH
                            WIRKLICH BIS ZUM PUNKT
```

Servo.py

```
1 import RPi.GPIO as gpio
```

```
2  import time
3
4  class Servo:
5      def __init__(self, servopin):
6          gpio.setmode(gpio.BCM)
7          gpio.setup(servopin, gpio.OUT)
8          self.p = gpio.PWM(servopin, 50)
9          self.p.start(2.5)
10
11     def up(self):
12         self.p.ChangeDutyCycle(2.5)
13
14     def down(self):
15         self.p.ChangeDutyCycle(7.5)
16
17
18
19     '''
20         p.ChangeDutyCycle(7.5) # nach links
21         time.sleep(1)
22         p.ChangeDutyCycle(12.5) # nach rechts
23         time.sleep(1)
24         p.ChangeDutyCycle(2.5) # nach vorne
25         time.sleep(1) '''
```

Literaturverzeichnis

Kiatronics. *28BYJ-48 – 5V Stepper Motor Datasheet*. `http://robocraft.ru/files/datasheet/28BYJ-48.pdf`.

raspberrypi, tutorials. *Raspberry Pi Servo Motor Steuerun*. `https://tutorials-raspberrypi.de/raspberry-pi-servo-motor-steuerung/`.

Tests, Technik und Reviews. *Tutorial Raspberry Pi] Schrittmotor Stepper motor*. `https://www.youtube.com/watch?v=4fHL6BpJrC4&list=PLCG2pkhu23HU-S4DkapfWdNXNwHw6gPev`.