SUDOMIND

Entwicklung eines Legoroboters, dessen Aufgabe darin besteht, ein Sudoku zu lösen

```
# load the input image and convert it to grayscale

image = cv2.imread(path_input)

image_to_show_vertical_lines = cv2.imread(path_input)
image_to_show_horizontal_lines = cv2.imread(path_input)
image_5_horver_lines = cv2.imread(path_input)



gray  = cv2.cvtColor(image,  cv2.COLOR_BGR2GRAY)
gray_for_gaussian  = cv2.cvtColor(image,  cv2.COLOR_BGR2GRAY)
gray_for_blurred  = cv2.cvtColor(image,  cv2.COLOR_BGR2GRAY)



imgforgaussian = cv2.medianBlur(gray_for_gaussian,5)
imagegaussian = cv2.adaptiveThreshold(imgforgaussian,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,11,2)
cv2.imwrite( "C:/temp/OpenCVTest/template-matching-ocr/imagegaussian.png", imagegaussian );

gray = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY_INV)[1]
cv2.imwrite( "C:/temp/OpenCVTest/template-matching-ocr/gray.png", gray );
```

Hrvoje Križić

Maturaarbeit

Informatik

Betreuungsperson: Rainer Steiger

3. Dezember 2019

Kantonsschule Schaffhausen

INHALTSVERZEICHNIS

"I knew that if I failed I wouldn’t regret that,
but I knew the one thing I might regret is not trying"
Jeff Bezos (Amazon Gründer)

1. EINLEITUNG

Erst vor kurzem sass ich mal wieder im Zug nach Winterthur und löste ein mittelschweres Sudoku aus einer Zeitschrift. Die Zeit vergeht wie im Flug, wenn man ein Sudoku dieser Art löst. Auf der Seite nebenan stand die Überschrift: "Schweizer Arbeitskampf 2.0; Roboter wird Gewerkschafter". Roboter haben mich schon immer fasziniert. Immer wieder stelle ich mir die Frage, wozu sie fähig sind und welche positiven, aber auch negativen Eigenschaften sie haben. Daher fragte ich mich, ob sie denn auch fähig sind, ein mittelschweres Sudoku zu lösen und falls ja, wie schwierig es ist, einen solchen Roboter zu bauen und zu programmieren.

Als Maturaarbeit möchte ich einen Roboter entwickeln, dessen Aufgabe es ist, ein beliebiges Sudoku zu lösen. Für den Bau des Roboters werde ich das Lego Mindstorms Ev3 Kit benutzen. Ich werde das Programm jedoch nicht mit der vorgegebenen Software erstellen, sondern in Python schreiben. Ziel des Projektes ist es, den Roboter so zu bauen und zu programmieren, dass man ihm ein Sudoku einfach einschieben kann und dieser das Sudoku selber löst. Dabei wird er das Sudoku zuerst im Ev3-Brick selber lösen und dann mit einem Stift die Zahlen in die leeren Felder eintragen.

2. LEGO MINDSTORM

2.1. LEJOS

Der Lego Mindstorm EV3 ist das Nachfolgemodell vom NXT. Alle Lego Mindstorms bieten die Möglichkeit an, auf einer SD-Karte ein anderes Betriebssystem zu booten. Für meine Arbeit verwendete ich das Java-basierte Betriebssystem LeJOS welches es mir ermöglichte, mit Python als Programmiersprache zu arbeiten. Ein grosser Vorteil dabei ist das Verschmelzen vom Code, welcher zuständig ist für das Bewegen des Roboters mit dem Code, welcher gebraucht wird um Berechnungen durchzuführen. Die Installation erfolgte ohne grosse Probleme, da der Ev3-Brick (Abb. 1) sofort anfängt die SD-Karte zu lesen und LeJos direkt startet.

Abb 1: Ev3-Brick: das Herz des Roboters

2.2. BAU DES ROBOTERS

2.2.1. Erster Versuch

Für den Bau des Roboters wurde keine Anleitung verwendet. Dies war anfangs nicht sehr einfach, da man nicht genau wusste, wie man den Roboter bauen sollte sodass er möglichst stabil wird. Diese Stabilität war

notwendig, damit die Zahlen des Roboters möglichst leserlich geschrieben werden können.
Schon früh fiel der Entscheid, dem Roboter beizubringen digitale Zahlen zu schreiben, da man so den Stift nur an der X- und Y- Achse bewegen muss und beide Achsen nicht gleichzeitig braucht. Für die Y-Achse wurden zwei Räder gebraucht auf denen sich nachher das Papier bewegt. So musste man den Stift nur auf der X-Achse bewegen und ich konnte mir die Y-Achse ersparen.

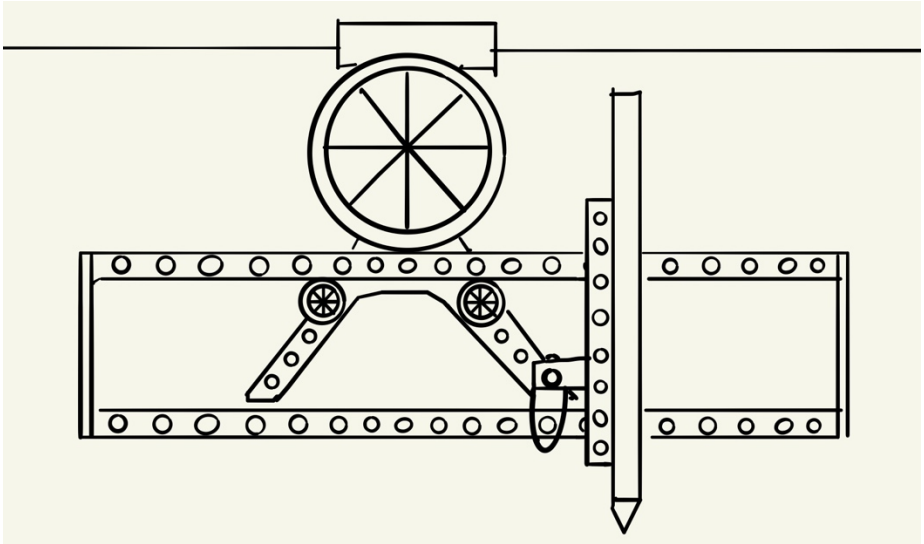


Abb 2: Umwandlung einer Drehbewegung in eine X-Achsenbewegung

Einer meiner "Highlights" war das Erstellen des Motor-Rad-Konstruktes, welches den Stift in eine horizontale Bewegung brachte. Unter dem Stifte-Halter baute ich den im EV3-Paket mitgelieferten Farbsensor ein, damit der Roboter später erkennen kann, wann sich das Blatt wo befindet. Ausserdem wurde einiges an Material benutzt, um die Auflageflächen zu bauen. An dieser Konstruktion gab es aber zwei grosse Nachteile:

1. Das Konstrukt war durch den sehr hohen Stifte-Halter sehr unstabil und konnte kaum verwendet werden, um Zahlen möglichst schön zu schreiben.

2. Die Konstruktion brauchte unnötig viel Material für die Auflageflächen und den Stifte-Halter, sodass mir kein Material mehr übriggeblieben ist für das Stabilisieren des Roboters.

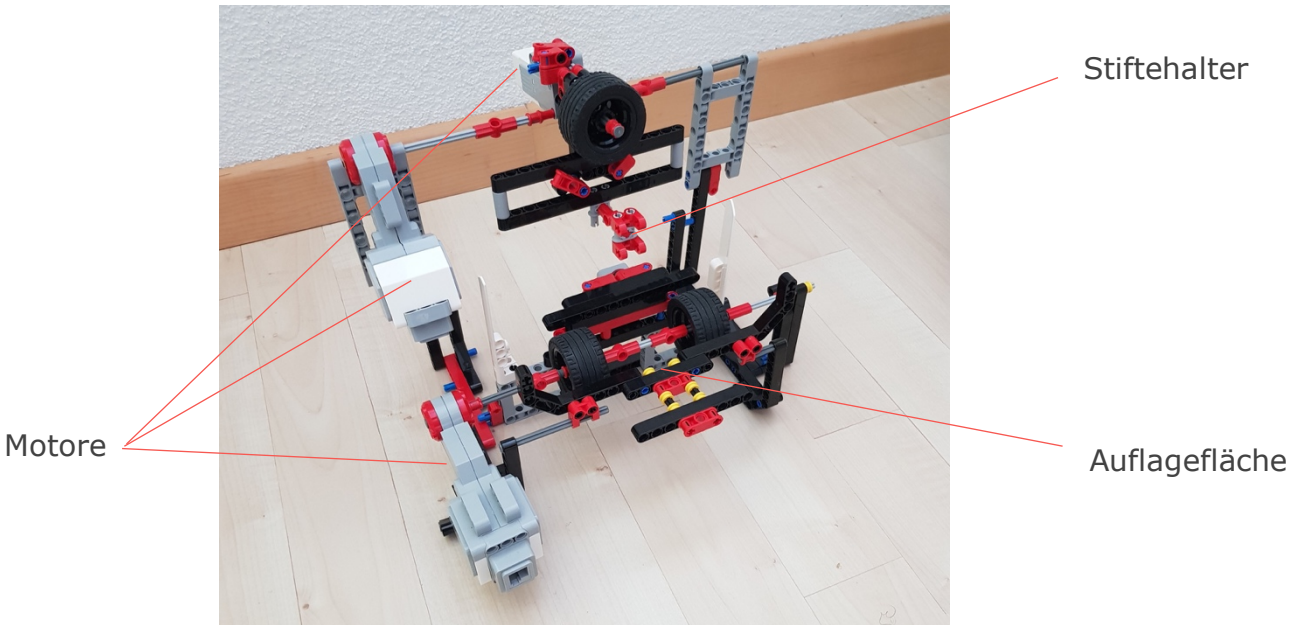


Abb 3: Erster Versuch

2.2.2. Zweiter Versuch

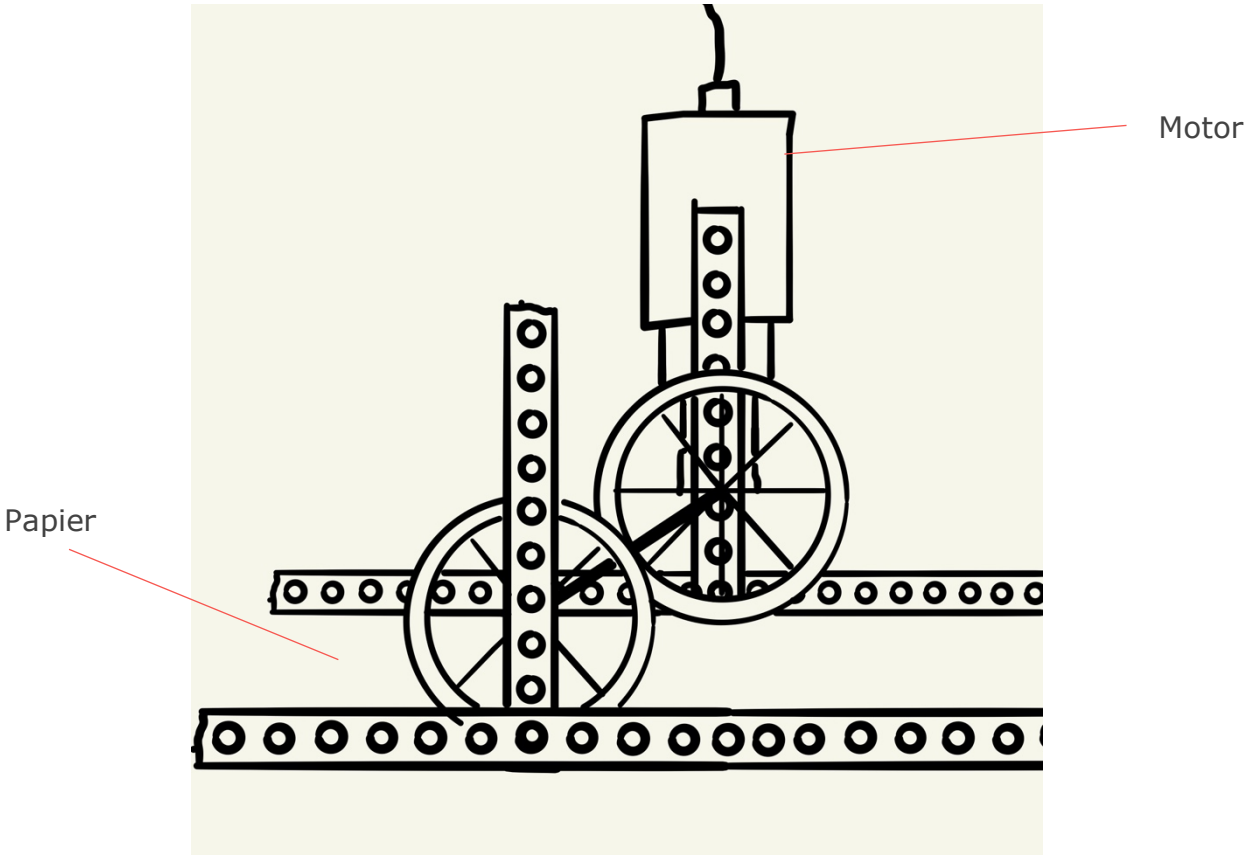


Abb 4: Y- Achsenbewegung im zweiten Versuch

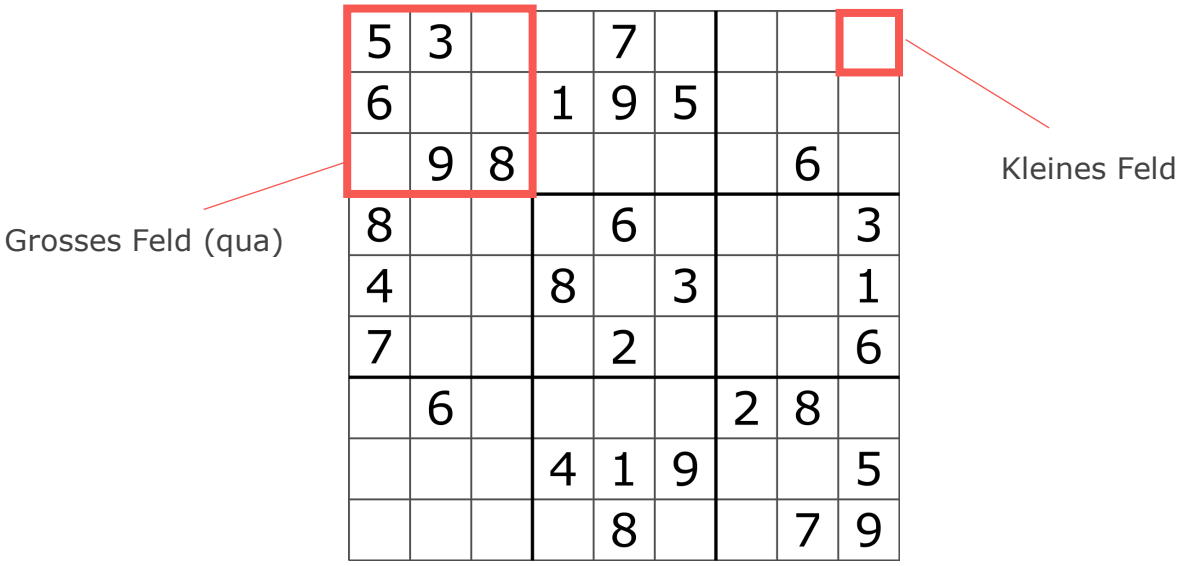
Der Entscheid fiel daher, den Roboter so umzubauen, dass die Auflagefläche der Tisch war und das Sudoku zwischen Tisch und Rädern eingeklemmt wird. So konnte man sich viel Material ersparen und den Stifte-Halter so weit nach unten setzen, dass die Stabilität so gut wie garantiert war.
Beim zweiten Versuch stellte sich heraus, dass diese Konstruktion deutlich stabiler war als die erste. Vor allem der tiefere Drehmotor (Abb. 2) sorgte für einen immensen Unterschied, da die Pfeiler, an denen der Drehmotor hängte nur noch halb so hoch waren. Ausserdem war nun ausreichend Material da, um die Stabilität der zwei vorderen Räder zu gewährleisten.

Der einzig ungünstige Punkt war die Unterlage auf der der Roboter gebaut wurde. Sie war rau und verursachte somit eine grosse Reibung. Daher wurde ab diesem Zeitpunkt ein Glastisch als Unterlage verwendet. Als Papier wurde etwas dickeres Papier mit einer glatten Rückseite verwendet. Ein weiteres Problem, welches sich herausstellte war die Verbindung vom Motor zum Ev3-Brick (siehe Abbildung 1). Die Motore waren zu weit auseinander und die Kabel für den Anschluss zu kurz. Schlussendlich musste die Konstruktion etwas umgebaut werden, sodass alle Kabel an den Ev3-Brick angeschlossen werden können.

3.SUDOKU

3.1. THEORIE

Ein Sudoku ist ein Knobelspiel bestehend aus 9 grossen und 81 kleinen Feldern. Das Ziel des Spiels ist es, die Zahlen von 1 bis 9 so zu verteilen, dass keine Zahl in einer Spalte, Zeile oder einem grossen Feld doppelt vorkommt.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Abb. 5: Sudoku

3.2. METHODEN

Es gibt verschiedene Techniken ein Sudoku zu lösen. Einerseits kann man jede Zeile, Spalte und jedes grosse Feld betrachten und dann schauen, welche Zahlen fehlen und welche durch die oben genannten Bedingungen ausgeschlossen werden. Bei der zweiten Methode geht es um das Betrachten der Zahlen. Man beginnt also bei der Zahl 1 und fragt sich, wo die Zahl hineinpassen würde. So arbeitet man sich durch alle 9 Zahlen durch.

3.3. PROGRAMMIERUNG

3.3.1. Backtracking

Der bekannteste Algorithmus um ein Sudoku zu lösen ist der sogenannte Backtracking-Algorithmus. Er ist eine Kombination zwischen der ersten und zweiten Methode.

Man nehme das erste freie Feld und überprüft, welche Zahlen in diesem Feld möglich sind. Hat man die erste Zahl gefunden, geht man zum nächsten Feld und überprüft dort, welche Zahlen hineinpassen könnten. Diese Schritte wiederholt man nun bis man zu einem Feld gelangt, in welches keine der Zahlen 1 bis 9 hineinpassen. Bei diesem Fall geht man wieder zum vorherigen Feld zurück und geht zur nächsten möglichen Zahl. Falls aber die Zahl, welche man vorher hatte, die einzig mögliche war, muss das vorletzte Feld überprüft werden. Schlussendlich kommt man so zur Lösung des Sudokus.

Ein grosser Vorteil besteht bei diesem Algorithmus darin, dass jedes Sudoku, sofern eine Lösung besteht, gelöst werden kann. Dabei ist die Ausführungszeit nicht proportional zum Schwierigkeitsgrad des Sudokus.

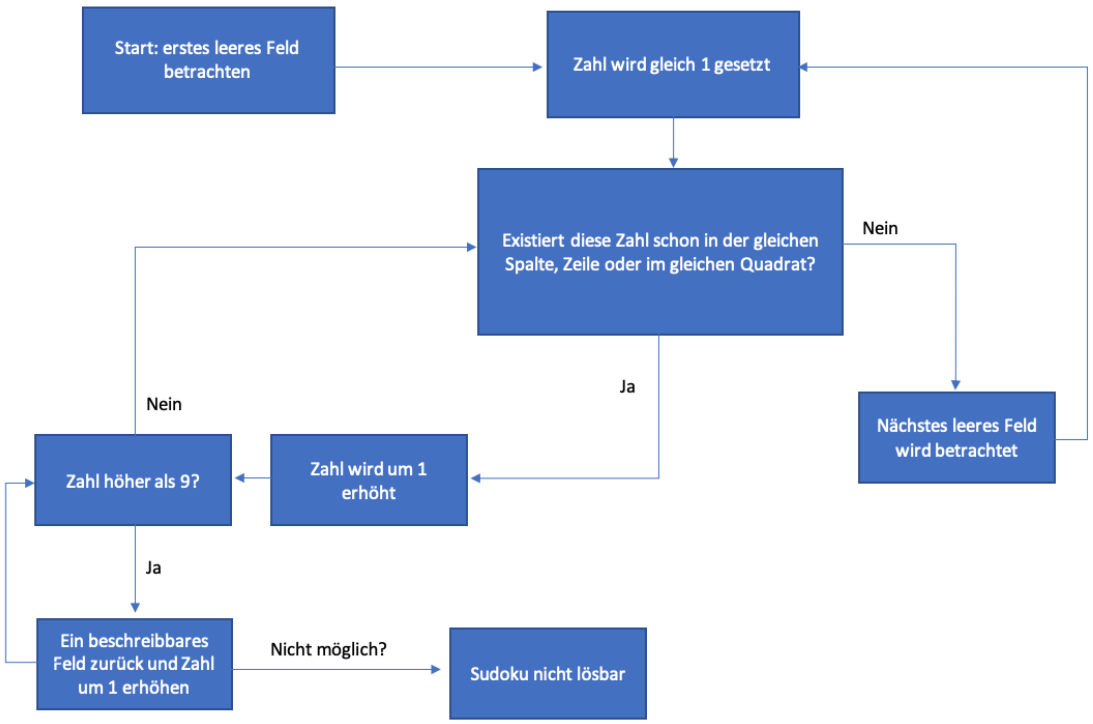


Abb. 6: Der Backtracking-Algorithmus

3.3.2. 4x4 Sudoku

Wie im Kapitel 2 schon beschrieben, stellt die Stabilität des Roboters mit dem Lego Mindstorm eine grosse Herausforderung dar. Ausserdem sind die Motoren des Lego Mindstorms sehr ungenau. Aus diesem Grund entschied ich mich, ein 4x4 Sudoku statt ein 9x9 Sudoku zu verwenden. Die Felder waren nun mehr als doppelt so gross als diejenigen des 9x9-Sudokus. Der Backtracking-Algorithmus liess sich aber ohne weiteres auf das 4x4 Sudoku übertragen.
Das Projekt wurde in zwei Teile aufgeteilt. Das erste Programm war zuständig für das Lösen des Sudokus und das zweite für das Schreiben der Zahlen. Schlussendlich verknüpfte ich die zwei Programme miteinander.

3.3.3. Sudoku-solver

Für den Sudoku-Algorithmus wurde keine Bibliothek verwendet. Das Sudoku wurde zunächst als Liste gespeichert. In diese Liste werden die Zahlen 0-4 eingefügt, wobei die Zahl 0 für ein leeres Feld steht (siehe Abb.7: sudoku). Um später den Backtracking Algorithmus auszuführen, musste man die Positionen der verschiedenen Nullen als eine neue Liste einspeichern (siehe Abb.7: unknown).

```
1.  sudoku = [0,0,0,0,
2.            0,0,0,0,
3.            0,0,0,0,
4.            0,0,0,0]
5.
6.  unknown = [i for i,val in enumerate(sudoku) if val==0]
```

Abb. 7: Die beiden Listen unknown und sudoku. Alle Felder in sudoku sind leer.

Als nächstes wurden allen Feldern die Koordinaten {c,r,q} zugewiesen. Dabei steht "c" für die Spalte, "r" für die Reihe und "q" für das 2x2 Feld in

dem sich das Feld befindet. Dabei wurden drei Listen erstellt: row (row engl. für Reihe), col (column engl. für Spalte) und qua (quadrate/square engl. Für Quadrat). Diese Listen haben je vier Elemente, da es in jedem 4x4 Sudoku vier Reihen, vier Quadrate und vier Spalten gibt. Man untersuche dies genauer am Beispiel der Liste row.

```
1.  row=list([0,0,0,0])
2.
3.  row[0] = list(sudoku[i] for i in [0,1,2,3])
4.  row[1] = list(sudoku[i] for i in [4,5,6,7])
5.  row[2] = list(sudoku[i] for i in [8,9,10,11])
6.  row[3] = list(sudoku[i] for i in [12,13,14,15])
```

Abb. 8: Die Liste row

Die Liste **row** hat zu Beginn vier "leere" Elemente. Dem ersten Element werden die ersten vier Elemente der Liste **sudoku** zugeschrieben, dem zweiten die nächsten vier und so weiter. Bei der Liste **qua** werden die Elemente 0,1,4 und 5 für das erste Quadrat benutzt. Die weiteren lassen sich nach gleichem Schema herleiten.
Falls nun eine Zahl im ersten Feld ist, so ist r=row[0], c=col[0] und q=qua[0], da sich die Zahl sowohl in der ersten Reihe, in der ersten Spalte als auch im ersten Quadrat befindet.
Sobald in einem Feld nun die Zahl 0 steht, sollte das Programm den Backtracking Algorithmus ausführen. Man definiere dafür drei neue Variablen: s, t und u, welche zählen sollten, wie viele Male die Zahl in der jeweiligen Reihe, Spalte und im Quadrat vorkommt.

```
1.  if x==0:
2.      x=1
3.      s=q.count(x)
4.      t=r.count(x)
5.      u=c.count(x)
6.      #Backtracking Algorithmus
7.      while s in [1] or t in [1] or u in [1]:
8.          x+=1
9.          s=q.count(x)
10.         t=r.count(x)
11.         u=c.count(x)
```

Abb. 9: Backtracking

Die Methode count() liefert dabei den Wert, wie häufig ein Element in einer Liste vorkommt. Wie man sieht, wurde das leere Feld durch die Variable "x" ersetzt. Die Variable "x" hat den Startwert 1. Solange also die Zahl in der

Reihe, Spalte oder im Quadrat schon vorhanden ist, muss der Wert x um 1 erhöht werden. Sobald die Zahl 5 erreicht wurde, wird der Wert wieder auf 0 gesetzt und der Wert des letzten beschreibbaren Feldes verändert. Dabei wird wieder der Wert x dieses Feldes solange um 1 erhöht, bis die Zahl weder im Quadrat, noch in der Spalte oder Reihe vorkommt.

```
1.  while x==5:
2.      sudoku[n]=0
3.      n-=1
4.      while n not in unknown or sudoku[n]==4:
5.          n-=1
6.      x=sudoku[n]
7.      aktualisieren()
8.      x+=1
9.      s=q.count(x)
10.     t=r.count(x)
11.     u=c.count(x)
12.     while s in [1] or t in [1] or u in [1]:
13.         x+=1
14.         s=q.count(x)
15.         t=r.count(x)
16.         u=c.count(x)
17.     sudoku[n]=x
```

Abb. 10: Das letzte beschreibbare Feld wird geändert.

Die Funktion aktualisieren(), welche zu Beginn definiert wurde, aktualisiert die Koordinaten r, c, q.

Im Falle, dass ein passender Wert für x gefunden wurde, wird das nächste beschreibbare Feld bearbeitet. Falls es keine beschreibbare Felder mehr gibt, ist das Sudoku gelöst.

```
1.  else:
2.      sudoku[n]=x
3.      #Solange noch nicht alle Nullen beseitigt wurden, wird das nächste Feld
        angesehen
4.      if sudoku.count(0)>0:
5.          n+=1
6.          while n not in unknown:
7.              n+=1
8.      #Alle Nullen wurden beseitigt und das Sudoku ist gelöst!
9.      else:
10.         break
```

Abb. 11: Das nächste beschreibbare Feld untersucht.

3.3.4. Motore

Im nächsten Abschnitt wird erklärt, wie man die Motore des Lego Mindstorms Ev3 in der Programmiersprache Python programmiert. Als Compiler wurde für das Programmieren die Software "TigerJython" benutzt. Dieses Programm ermöglicht es dem Programmierer, über Bluetooth und LeJos auf den Roboter zugreifen zu können.
Zu Beginn müssen die Motore dem Roboter zugewiesen werden.

```
1.  from ev3robot import *
2.  import time
3.  robot = LegoRobot()
4.  gear = Gear()
5.  robot.addPart(gear)
6.  motord = Motor(MotorPort.D)
7.  motorc = Motor(MotorPort.C)
8.  robot.addPart(motord)
9.  robot.addPart(motorc)
```

Abb. 12: Motore dem Roboter zuordnen

Die Funktion Gear() wird für das Fahren eines Auto-Roboters verwendet. Diese Funktion verbindet die beiden Motore A und B die im SUDOMIND für die Bewegung an der Y-Achse gebraucht werden. Am Anschluss C wird der Motor angeschlossen, der zuständig ist für das Auf- und Abfahren des Stiftes. Am Port D wird zuletzt der Motor für die X-Achsenbewegung angeschlossen.
Wie aus der Abbildung 9 ersichtlich ist, wurden die zwei Bibliotheken **ev3robot** und **time** benutzt. Die Bibliothek **ev3robot** wird verwendet, um die Verbindung zu LeJos und somit zum Roboter herzustellen. Die Bibliothek **time** wird später vor allem für die Funktion time.delay([Sekunden]) verwendet. Mit dieser Funktion lässt sich der Motor so lange bewegen bis die angegebene Zeit abgelaufen ist.
Die Zahlen werden nun wie folgt geschrieben:

```
1.  def number1():
2.      #nach oben
3.      gear.setSpeed(5)
4.      gear.forward()
5.      time.sleep(0.4)
6.      gear.stop()
7.      penDown()
8.
9.      #nach unten
10.     gear.backward()
11.     time.sleep(0.4)
12.     gear.stop()
```

```
13.
14.     #nach unten
15.     gear.backward()
16.     time.sleep(0.4)
17.     gear.stop()
18.     penUp()
19.
20.     #nach oben
21.     gear.forward()
22.     time.sleep(0.4)
23.     gear.stop()
```

Abb. 13: Bewegung des Roboters am Beispiel der Nummer 1

Zunächst wird die Geschwindigkeit des Motors definiert. **setspeed(Geschwindigkeit)** setzt die Motorgeschwindigkeit in Grad/Sekunden. Danach wird die Funktion **gear.forward()** bzw. **gear.backward()** aufgerufen. Diese zwei Funktionen ermöglichen die Bewegung des Motors. Sie bewegen sich konstant mit der Geschwindigkeit von **setspeed()** bis die Zeit in der Funktion **time.sleep()** abgelaufen ist und der Motor mit **gear.stop()** zum Stehen gebracht wird.
Zwei wichtige Funktionen welche noch vorher definiert wurden, waren die Funktionen **penUp()** und **penDown().** Sie ermöglichen die Auf- und Ab-Bewegung des Stiftes.

```
1.  def penUp():
2.      motorc.setSpeed(2)
3.      motorc.forward()
4.      time.sleep(stifthalter)
5.      motorc.stop()
6.  def penDown():
7.      motorc.setSpeed(2)
8.      motorc.backward()
9.      time.sleep(stifthalter)
10.     motorc.stop()
```

Abb. 14: Die Funktionen penUp() und penDown()

4. ERKENNUNG DES SUDOKUS

4.1. BIBLIOTHEKEN

Für die Programmierung des Erkennungsprogramms benutzte ich zwei wichtige Bibliotheken.

4.1.1. OpenCV

Eine davon ist die Bibliothek OpenCV. Das «CV» im Namen steht für «Computer Vision» und beschreibt die Bibliothek schon sehr gut. Sie umfasst Algorithmen für das «maschinelle Sehen». Einige Beispiele hierfür sind etwa Gesichtserkennungen, künstliche neuronale Netze und Gestenerkennung. Die Bibliothek wird später den grössten Teil des Codes in Anspruch nehmen.

4.1.2. Tesseract

Tesseract ist eine Bibliothek mit den Algorithmen zur Texterkennung. Der grösste Vorteil dieser Bibliothek ist, dass sie jede Schriftart in jeder Sprache erkennen kann. Ein weiterer grosser Vorteil dieser Bibliothek ist, dass sie beispielsweise mit Handschriften trainiert werden kann. Im Code kommt die Bibliothek nur einmal zum Einsatz, nämlich dann, wann die fertigen Bildausschnitte in Zahlen umgewandelt werden.

4.2. AUSFILTRIEREN

Zunächst wird ein Bild aufgenommen. Dazu wird eine Kamera an einem USB-Port mit der Funktion **VideoCapture()** aufgenommen und gespeichert. Um ein Sudoku zu erkennen, muss das Programm zunächst erkennen, wo sich das Sudoku im Bild befindet. Im folgenden Beispiel sind Störelemente in der Form eines Fingers oder einer Seitenkante vorhanden. Diese sollten ausgefiltert werden.

Das Bild wird als erstes von farbig zu schwarz-weiss transferiert. Das entstandene schwarz-weiss-Bild wird anschliessend *geblurrt*. Das heisst, es wird durch die Funktion medianBlur() von allen kleinen Pünktchen befreit. Dies ist zu vergleichen mit einer Audio-Aufnahme bei der das Rauschen entfernt wird.

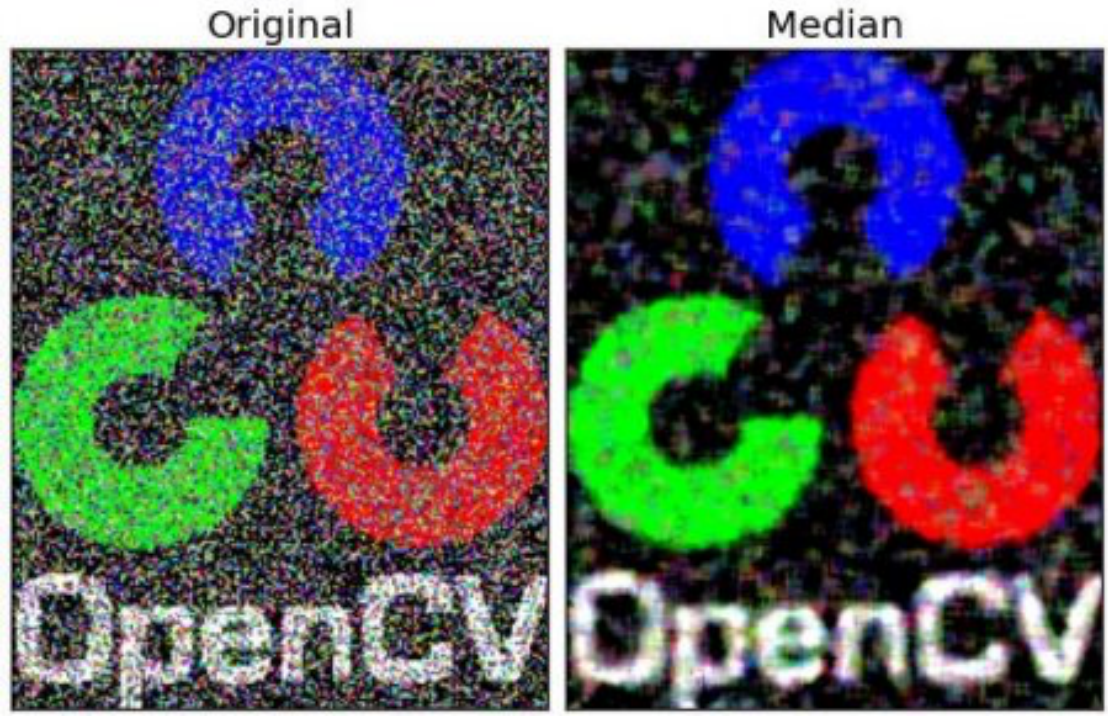
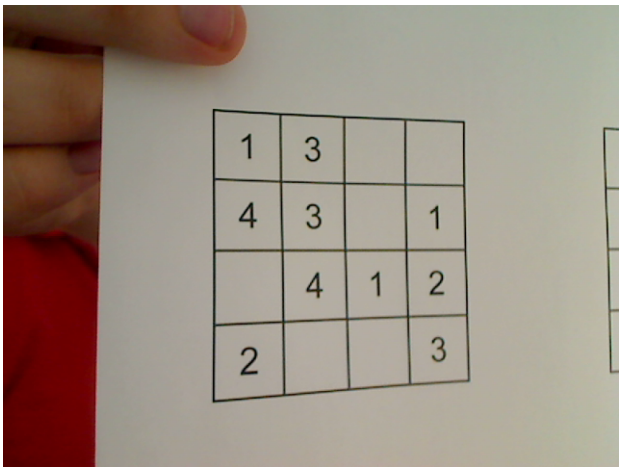


Abb. 15: Die Funktionen medianBlur() am Beispiel des OpenCV Logos

Das geblurrte schwarz-weiss-Bild wird nun nochmals so gefiltert, dass nur noch die zwei Farben schwarz und weiss vorhanden sind. Ein übliches schwarz-weiss-Bild hingegen ist ein Bild, dessen Farben in verschiedene Graustufen eingeteilt werden. Im nächsten Schritt müssen diese Graustufen

je nach Präferenz als Weiss oder Schwarz definiert werden. Dies geschieht durch das *adaptive thresholding*. Im adaptiven thresholding können Parameter gesetzt werden, welche die Grenzen von Weiss zu Schwarz festlegen. Diese wurden in diesem Beispiel experimentell gefunden. Um nun noch genauer die Linien vom Blatt abzutrennen wird in diesem Programm zusätzlich die *Canny Edge Detection* verwendet. Diese besteht aus fünf Algorithmen, welche ein Bild ergeben, bei dem jede Kante jedes Objektes erkennt wird. Ausserdem werden die Farben vertauscht, da die Bibliothek Tesseract später die Zahlen Weiss auf Schwarz besser erkennen kann.



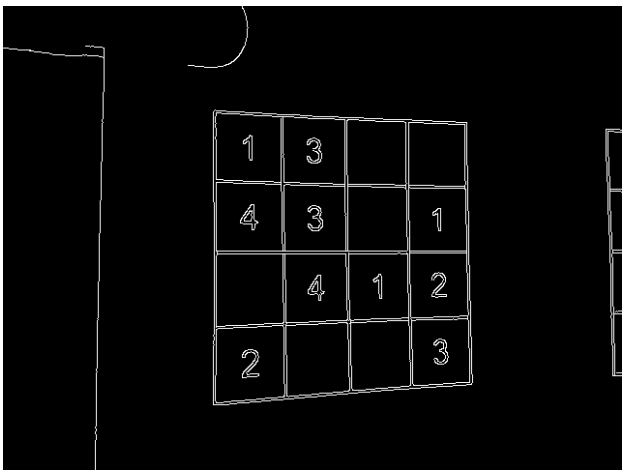


Abb. 16 und 17: Das Bild zu Beginn und nach dem Ausfiltrieren

Wie man nicht schwer erkennen kann, sind links und rechts immer noch Linien vorhanden, welchen nicht zum Sudoku gehören. Im folgenden Kapitel wird beschrieben, wie die Ecken jedes einzelnen Feldes aufgesucht werden.

4.3. FELDERERKENNUNG

Die Idee hinter der Feldererkennung ist es, zunächst die obere linke Ecke zu finden, dann alle horizontalen und vertikalen Linien aufzusuchen und schlussendlich diese miteinander zu vergleichen und die Schnittpunkte daraus zu bilden.

4.3.1. Sudoku erkennen

Als erstes wird die obere linke Ecke gefunden. Dies erfolgt durch das Vergleichen von verschiedenen Flächen. Man findet durch die Konturen alle möglichen Felder die es gibt. Da die Felder ineinander verschachtelt sind, muss zunächst untersucht werden, in welcher Ebene sich das Sudoku befindet. Dies ist jedoch ganz einfach, da die Zahlen die innersten Flächen definieren. Diese Flächen sind von zwei Konturen begrenzt. Danach kommt die innere Kontur des Sudokus und dann die äussere. Die erste Ebene ist die '0'-te Ebene. Wir befinden uns also beim Rand des Sudokus auf der Ebene 3. Anschliessend wird die Fläche des Sudokus mit einem experimentell gefundenen Wert verglichen. Ist die Fläche in dieser Spannweite, so muss es sich um das Sudokufeld handeln. Da wir nun das Rechteck haben, welches wir gesucht haben, können wir die obere linke Ecke daraus schliessen.

4.3.2. horizontale und vertikale Linien

Als nächstes werden alle horizontalen und vertikalen Linien gesucht. Die obere linke Ecke des Bildes (wichtig: **nicht** des Sudokus) wird mit den Koordinaten {0,0} beschrieben. Von dort aus wird eine Senkrechte auf die nächste erkennbare Linie gelegt. Nun werden folgende Eigenschaften überprüft:

1. Falls die Linie kürzer als ein bestimmter Wert ist, wird die Linie nicht gezählt.
2. Falls die Linie weniger als einige Pixel von einer anderen Linie entfernt ist, wird diese nicht gezählt (der Abstand zwischen dem Ordinatenursprung und der Linie wird als r bezeichnet).
3. Falls der Polarwinkel θ nicht in einem bestimmten Intervall liegt, so ist die Linie «zu schräg» und sie wird nicht gezählt.
4. Der Polarwinkel bestimmt, ob die Linie eine horizontale oder eine vertikale Linie ist.

Der Punkt 2 ist sehr wichtig, da die Canny Edge Detection aus einer Linie zwei macht, da sie es als Fläche und nicht als Linie betrachtet.

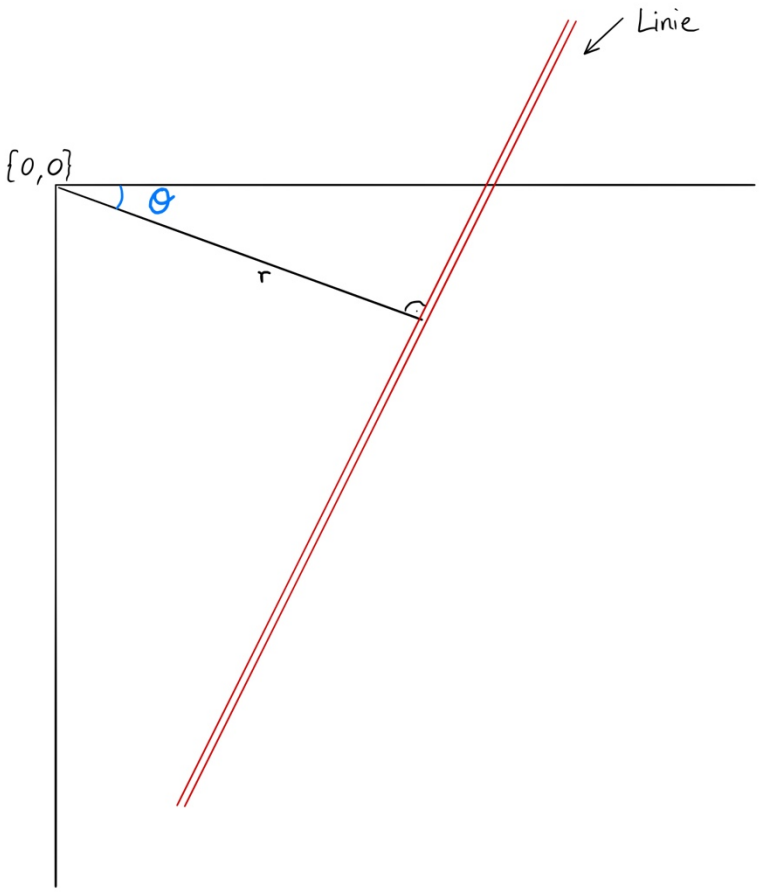


Abb. 18: Linienerkennung

Die Linien, welche alle Eigenschaften erfüllen, werden in Arrays gespeichert. Wie man in der nächsten Abbildung erkennen kann, hat die Funktion noch weitere Linien erkannt, welche diese Eigenschaften ebenfalls erfüllen, aber nicht zum Sudoku gehören. Diese müssen aussortiert werden.

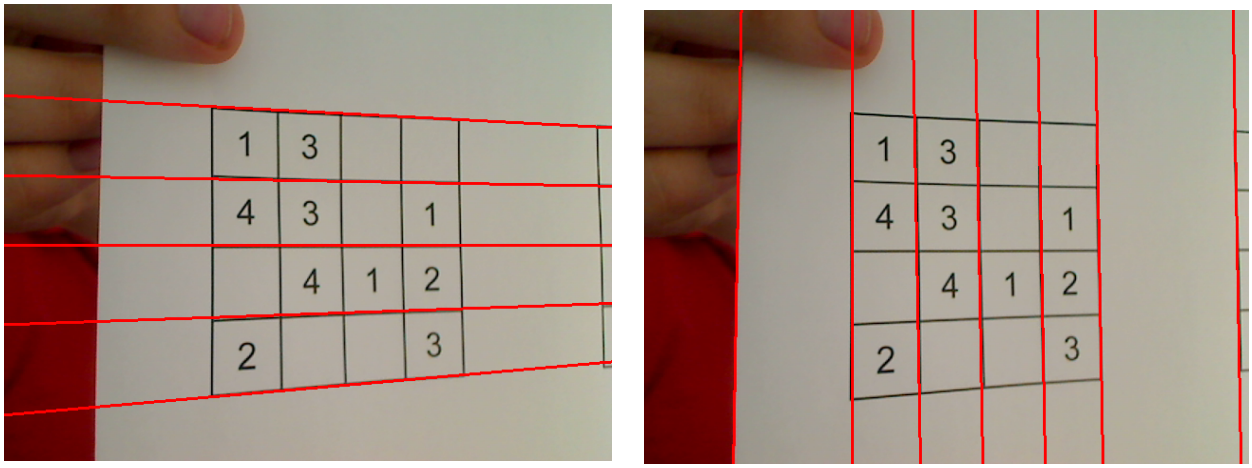


Abb. 19: die vertikalen und horizontalen Linien

4.3.3. Zusammensetzen

Das Finden der richtigen fünf Linien erfolgt durch den Vergleich der Koordinaten der Geraden mit den Koordinaten der oberen linken Ecke des Sudokus. Man weiss, dass die oberste horizontale und die erste vertikale Linie des Sudokus sehr nah an der oberen linken Ecke vorbeiführen. So findet man die erste gerade Linie. Die nächsten vier Geraden gehören dann ebenfalls zum Sudoku. Die Schnittpunkte der fünf vertikalen und horizontalen Geraden bilden die Ecken jedes Feldes. Somit ist die Feldererkennung abgeschlossen.

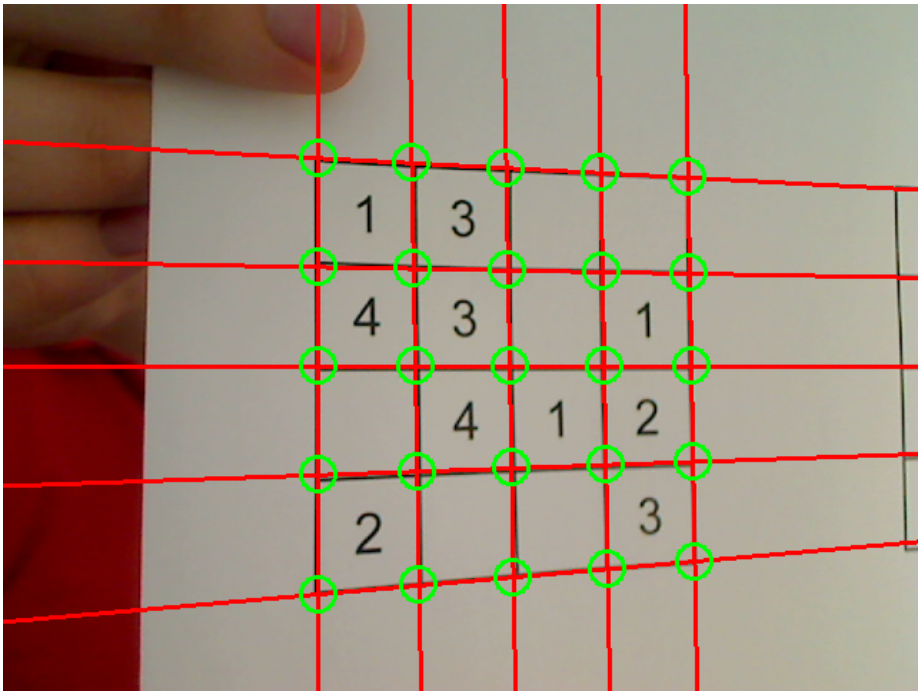


Abb. 19: die abgeschlossene Feldererkennung

4.4. ZAHLENERKENNUNG

Als nächstes müssen aus allen Feldern die Zahlen ausgelesen werden. Dazu betrachtet man ein etwas kleineres Feld, damit die Zahlen etwas grösser erscheinen. Die Zahlen können nun nach und nach mit der Tesseract Bibliothek verglichen und ausgewertet werden. Als Referenz wurden die Zahlen der Schriftart «Arial» benutzt. Wird in einem Feld keine Zahl erkannt, so setzt man den Wert auf 0. Die Zahlen werden schlussendlich in eine Textdatei verpackt und können ab nun weiterverwendet werden für den Backtracking Algorithmus.

5.PROBLEME UND ERFAHRUNGEN

Während des Baus und der Programmierung sind einige Probleme aufgetreten. Diese möchte ich im nächsten Kapitel genauer erläutern.

5.1. LEGO

Im Sommer 2019 bestellte ich mir das Ev3 Mindstormkit von Lego. Zu Beginn hatte ich keine Schwierigkeiten mich mit dem Brick und den Programmen vertraut zu machen. Nach einigen Monaten bemerkte ich jedoch, dass sich mein Brick nicht mehr richtig einschalten liess. Nach der Abgabe des Bricks an Lego wurde mir gesagt, dass die Reparatur des Bricks vier bis fünf Wochen dauern würde. Zum Glück konnte mir Flori Kunert seinen Brick für diese Zeitspanne ausleihen, damit ich weiterhin am Projekt arbeiten konnte. Mein neuer Brick wurde wie versprochen einige Wochen später geliefert.
Ein weiteres Problem hatte ich beim Bauen des Roboters. Da ich, wie im Kapitel 2 schon erwähnt, keine Anleitungen oder Vorlagen benutzte und zuvor nie mit einem Lego Mindstormkit gearbeitet habe, brauchte es einige Zeit bis der erste Versuch gelungen war. Die grössten Schwierigkeiten hatte ich mit der Stabilität des Roboters. Der Drehmotor war zu schwer für die Konstruktion. Mit dem zweiten Versuch konnte ich dann den Drehmotor durch zwei sogenannte «Frames» stabilisieren.

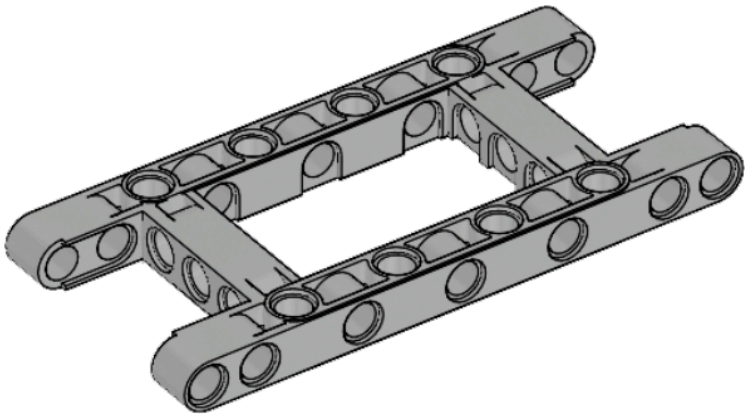


Abb. 20: Der Legobaustein «Frame»

5.2. BACKTRACKING ALGORITHMUS

Wie schon beim Bau des Roboters, habe ich bei der Programmierung des Backtracking Algorithmus entschieden, keine Anleitung zu benutzen. Diese Entscheidung führte dazu, dass ich einige Stunden für den Backtracking Algorithmus benötigte, denn dieser war deutlich schwieriger als erwartet. Die grösste Herausforderung war die Schleife für die Änderung des letzten beschreibbaren Feldes (siehe Abb. 10), da bei dieser Schleife viele Ausnahmen beachtet werden mussten.

5.3. MOTORE

Ein weiteres Problem war die Programmierung der Motore am Ev3 Roboter. Die Motore des Ev3 sind sehr ungenau. Dies führt dazu, dass man nicht einfach mit den Gesetzen der Physik arbeiten kann, sondern in erster Linie durch das Experimentieren. Zum einen spielt die Unterlage eine grosse Rolle. Weiter kommen die Reibung des Stiftes und die Reibung der Räder auf dem Papier dazu. Da der Drehmotor dahinter nicht gestützt war, war das Konstrukt anfangs etwas schräg, bevor ich die «Frames» eingebaut habe.

5.4. ERKENNUNG-EV3

Das einzig ungelöste Problem blieb die Verbindung zwischen dem Programm zur Erkennung des Sudokus und dem Programm zur Steuerung des Roboters. Das Problem liegt darin, dass sich das Programm zur Steuerung auf dem Ev3 befindet und das Programm zur Erkennung auf dem Laptop. Somit kann das Steuerungsprogramm nicht auf die Daten, die das Erkennungsprogramm generiert hat, zugreifen. Man müsste eine Client-Server-Verbindung aufbauen, die ich mit meinem Macbook als Client und meinem Roboter als Server noch nicht erzeugen konnte.

5.5. AUSSICHT

Neben dem, noch zu lösenden Problem, möchte ich in Zukunft weitere Möglichkeiten meines Sudokus ausnutzen. Ein Beispiel hierfür wäre ein Roboter zum Lösen von magischen Quadraten oder ein Roboter der wie ein Tic-Tac-Toe-Gegner funktioniert. Zu allererst möchte ich aber versuchen, den Roboter mit einem zweiten Kit so auszubauen, dass er auch ein 9x9 Sudoku lösen kann. Ausserdem möchte ich die Erkennungssoftware auf Handschriften trainieren lassen. Wie schon in Kapitel 4.1.2 erklärt, ist dies mit der **tesseract** Bibliothek möglich.

6.DANKSAGUNG

Ich möchte mich bei allen Menschen, die mich bei meiner Arbeit unterstützt haben herzlich bedanken.
Als erstes möchte ich mich bei Rainer Steiger, meiner Betreuungsperson bedanken. Ohne ihn wäre die Maturaarbeit bei Weitem nicht das, was sie nun ist.

Weiter möchte ich mich bei meiner Familie für ihre tatkräftige Unterstützung bedanken, vor allem bei meinem Vater der mich bei der Lösung des Erkennungsproblems unterstützt hat.

Zu guter Letzt möchte ich mich bei allen Personen bedanken, die einen kleinen, aber sehr wichtigen Beitrag zu meiner Maturaarbeit geleistet haben, insbesondere Flori Kunert, der mir seinen Brick zur Verfügung gestellt hat.

7.QUELLEN

7.1. ABBILDUNGEN

Abb. 1:
https://sh-s7-live-s.legocdn.com/is/image/LEGO/45500?scl=1.7&op_sharpen=1

Abb. 5:
https://upload.wikimedia.org/wikipedia/commons/thumb/e/e0/Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg/800px-Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg.png

Abb. 15:

https://docs.opencv.org/master/median.jpg

Abb. 20:

https://le-www-live-s.legocdn.com/sc/media/files/element-surveys/ev3/45544_element_survey-e40199b2a68b9ce583e2ad99844ce865.pdf

Alle weiteren Abbildungen sind eigene Abbildungen oder Ausschnitte aus dem Programmiercode.

8. LITERATURVERZEICHNIS

Geeks for geeks. (kein Datum). Abgerufen am 12. Oktober 2019 von https://www.geeksforgeeks.org/backtracking-algorithms/

Moser, O. (6. Juni 2017). *Codecentric*. Abgerufen am 27. Oktober 2019 von https://blog.codecentric.de/2017/06/einfuehrung-in-computer-vision-mit-opencv-und-python/

Roos, F. (12. April 2019). *JAXenter*. Abgerufen am 12. Oktober 2019 von https://jaxenter.de/perl-code-tesseract-ocr-82139

Sotek, O. (15. März 2019). *Heise*. Abgerufen am 28. Oktober 2019 von https://www.heise.de/ratgeber/Texterkennung-mit-Tesseract-und-Python-4334232.html

Wikipedia. (kein Datum). Abgerufen am 19. Oktober 2019 von https://de.wikipedia.org/wiki/Sudoku

Wikipedia. (kein Datum). Abgerufen am 18. September 2019 von https://de.wikipedia.org/wiki/Backtracking

9.ANHANG

Video des SUDOMIND

OpenCV Erkennungsprogramm

SUDOMIND-Programm