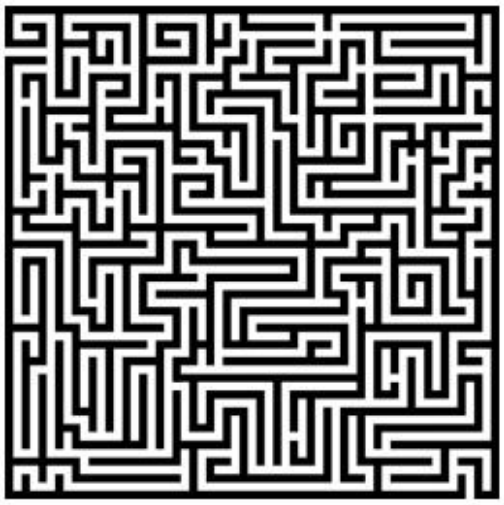
künstliche intelligenz
kognitive systeme

HOCHSCHULE
ANSBACH

# Roboter Escape Game

Projektarbeit von Tobias Saur, Niclas Hart und Jakob Kramer im Rahmen der Erstjahresprojekte im Sommersemester 2023



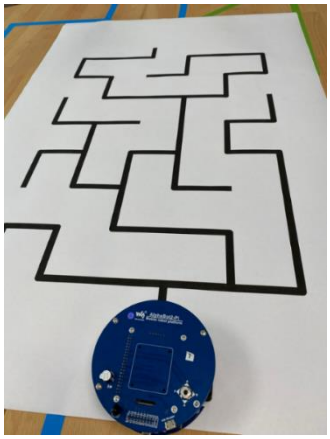
# Inhaltsverzeichnis

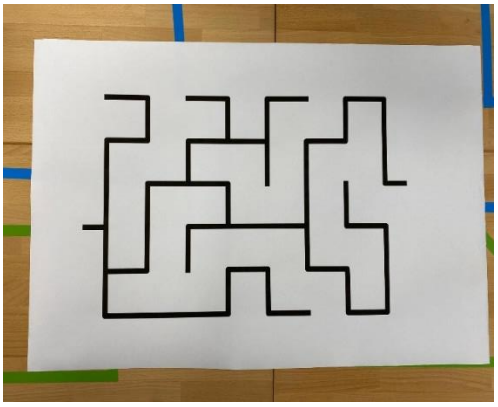
# 1. Einleitung

Die Zielstellung dieser Projektarbeit ist es, einen teilweise autonomen Roboter zu befähigen, sich in einer bekannten Umgebung mittels bereits vorhandener Sensorik zurechtzufinden. Im Rahmen dieses Erstjahresprojektes wurde der Alphabot2 verwendet, um unter Verwendung der Line-Tracking-Funktion selbständig durch das Labyrinth zu fahren und einen Ausgang zu finden.

Die essenziellen Aufgaben, die dafür bearbeitet werden müssen, sind die Implementierung beziehungsweise Verwendung der grundlegenden Fahrfunktionen, wie das Vorwärts-, Rückwärts- und das Kurvenfahren, eine geeignete Nutzung der optischen Sensoren für die Linienerkennung, sowie die generelle Konzeptionierung und Umsetzung eines Algorithmus zum Finden des Ausgangs des Labyrinths inklusive der Identifikation von Irrwegen. Das Labyrinth besteht dabei aus jeweils gleich breiten, aneinander liegenden Linien. Diese stellen zusammenhängend die Geraden, Kreuzungen, Links- oder Rechtsabbiegungen oder Sackgassen dar.

Das tatsächliche Ziel des Roboters bei der Lösung des Labyrinths ist optional und kann selbst festgelegt werden. Mögliche konkrete Zielstellungen wären die Erkundung des gesamten Labyrinths oder die Identifizierung von Ausgängen. Im Gegensatz zu herkömmlichen Labyrinthen bewegt sich der Roboter nicht zwischen zwei Mauern oder Linien, sondern die Linien selbst stellen die fahrbaren Wege dar. Dies vereinfacht die Bearbeitung deutlich, da so die bereits vom Hersteller des Alphabots mitgelieferte Linienverfolgung genutzt werden kann. Andernfalls müsste zunächst eine Möglichkeit gefunden werden, um zuverlässig zwischen zwei Umrandungen fahren zu können.

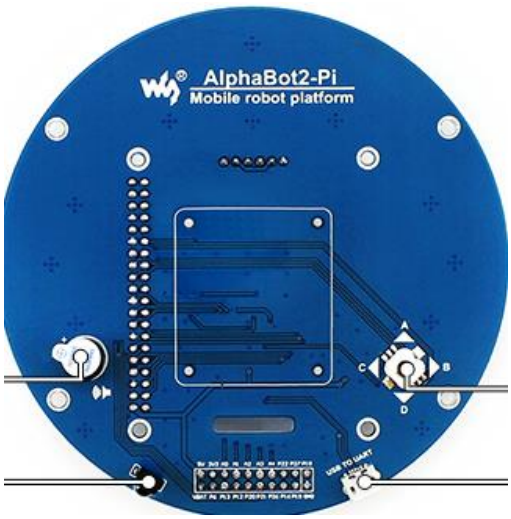




## 2. Grundlegendes zum Alphabot2 und dessen Einrichtung

Der Alphabot2 ist ein Bausatz des chinesischen Herstellers Waveshare Electronics, der es ermöglicht auf Basis eines Raspberry Pi-Minicomputers als Steuerelement mit Hilfe verschiedener Sensoren und Aktoren einen Roboter zu entwickeln, der je nach individueller Programmierung auf seine Umwelt reagiert und dem entsprechend handelt.

Die für die Bearbeitung unseres Projekts am relevantesten Teile des Roboters waren die beiden Räder des Alphabots, die durch zwei unabhängige Motoren angetrieben werden (die Geschwindigkeiten der einzelnen Motoren werden dabei per Pulsweitenmodulation gesteuert) und die fünf reflektiven Infrarotsensoren auf der Unterseite des Alphabots. Diese werden für die später noch genauer erläuterte Linienerkennung benötigt. Diese Sensoren sind in der Lage abhängig davon, ob der jeweilige Sensor zum aktuellen Zeitpunkt auf einem schwarzen / weißen steht oder ob der Alphabot gerade hochgehoben wird und sich dadurch nicht auf einem festen Untergrund befindet. Daneben wurde außerdem der Joystick auf der Oberseite verwendet, um das Starten beziehungsweise das Beenden des Programmablaufs auszulösen und die vier LEDs zur anschaulichen Darstellung der Abbiegevorgänge beim Durchfahren des Labyrinthes.

## 3. Workflow und Kommunikation mit dem Raspberry Pi

Ein großer Vorteil bei der Bearbeitung des Themas unseres Erstjahresprojektes ist es, dass die gesamte Programmierung des Alphabots auf dem Raspberry Pi in der Programmiersprache Python möglich ist. Dadurch war es möglich, ohne eine ausgeprägte Einarbeitungsphase direkt mit dem Entwickeln eigener Programme zu starten. Des Weiteren wurde frühzeitig entschieden zur Verwaltung unserer Softwareversionen Git in Kombination mit Gitlab zu verwenden. Ein Vorteil davon ist, dass die Arbeit im Team dadurch reibungslos möglich war und jeder zu jedem beliebigen Zeitpunkt Zugriff auf den aktuellen Stand hat. Weitere Vorteile sind eine bessere Nachvollziehbarkeit bei Änderungen und außerdem der Schutz vor Verlust von Dateien, die bei der Arbeit auf dem Raspberry Pi ohne Backup nicht auszuschließen sind.

Für die Kommunikation mit dem Raspberry Pi über externe Rechner wurde eine SSH-Verbindung (Secure Socket Shell) hergestellt. Dadurch war es möglich direkt über den eigenen Laptop mit Hilfe der Kommandozeile im Terminal Programme auf dem Raspberry Pi zu bearbeiten oder zu Testzwecken auszuführen, ohne dafür einen Monitor und weitere Peripheriegeräte anschließen zu müssen. Zur bequemen und unterstützten Code-Bearbeitung direkt auf dem Raspberry Pi wurde auch die VSCode-Erweiterung SSHFS verwendet, die diverse aus VSCode bekannte Funktionen wie Code-Vervollständigung und einfaches Debuggen auch direkt über die SSH-Verbindung ermöglicht.

Generell wurde bei der Arbeit an unserem Erstjahresprojekt Wert auf eine strukturierte Arbeitsteilung gelegt. Dies hat es ermöglicht sehr effizient zusammenzuarbeiten, da jedes Teammitglied auf seinem jeweiligen Fachgebiet Knowhow entwickelt hat und dadurch gemeinsame Besprechungen mit vollem Fokus auf Problemlösungen und die Zusammenführung der einzelnen Bestandteile genutzt werden konnten.

# 4. Bearbeitung des Projekts

## 4.1 Grundlegende Fahrfunktionen

Die bereits vorhandenen Codebeispiele waren nützlich, um ein grundsätzliches Verständnis zu entwickeln, wie die Ansteuerung des Alphabots funktionieren kann. Im Anschluss wurden zusätzlich zu den Funktionen zum Vorwärtsfahren, Drehen und Anhalten weitere für das Projekt benötigte Funktionen entwickelt. Im Folgenden werden zwei dieser Implementierungen beispielhaft genauer erläutert:

Zunächst die Implementierung des Links-Abbiegens im 90° Winkel. Dabei wird zuerst die self.left()-Funktion aufgerufen, sodass sich der Roboter beginnt, sich mit der vorher festgelegten Geschwindigkeit nach links zu drehen. Nach einer selbst festgelegten Zeit wird Drehung dann wieder gestoppt.

```
81      def turnleft(self):
82          self.left()
83          time.sleep(0.17)
84          self.stop()
```

Eine generelle Optimierung zur Verbesserung der Kurvenfahrfunktionen wäre es, den Roboter sich so lange nach links oder rechts drehen zu lassen, bis der mittlere Sensor erneut die schwarze Linie detektiert. Dadurch wäre sichergestellt, dass sich der Alphabot auch tatsächlich bei jeder Abbiegung um nahezu genau 90° dreht.

Eine weitere notwendige Methode für die korrekte Erkennung von Wegpunkten ist die moveoverline()-Funktion. Das Überfahren der Wegpunkte zum Überprüfen der Sensoren ist wichtig, da diese übergebenen Werte zeigen, um welchen Wegpunkt es sich handelt. So kann beispielsweise eine X-Kreuzung vorerst aussehen wie eine T-Kreuzung und eine endgültige Entscheidung kann erst nach Überfahren der Wegpunkte getroffen werden.

In untenstehenden Code-Ausschnitt wird zunächst die self.forward()-Funktion verwendet, damit der Roboter sich geradeaus nach vorne bewegt, bevor er nach einer erneut selbst eigestellten Verzögerung wieder gestoppt wird.

```
91      def moveoverline(self):
92          self.forward()
93          time.sleep(0.35)
94          self.stop()
```

## 4.2 Die Line-Following Funktionalität (Linienfolgen)

Die Funktion des Linienfolgens ist für das Durchfahren des Labyrinths von essenzieller Bedeutung. Die hierfür grundlegenden Code-Beispiele sind werden vom Hersteller bereitgestellt und müssen dann je nach Anwendung ergänzt und mit individuellen Parametern angepasst werden.

Im Folgenden werden die relevanten Einzelschritte des Line-Followings genauer erläutert:

```
53  TR = TRSensor()
54  Ab = AlphaBot2()
55  Ab.stop()
56  print("Line follow Example")
57  time.sleep(0.5)
58  for i in range(0, 100):
59      if (i < 25 or i >= 75):
60          Ab.right()
61          Ab.setPWMA(25)
62          Ab.setPWMB(20)
63      else:
64          Ab.left()
65          Ab.setPWMA(25)
66          Ab.setPWMB(20)
67      TR.calibrate()
68  Ab.stop()
69  print(TR.calibratedMin)
70  print(TR.calibratedMax)
```

Zuerst werden die beiden die Sensoren und der Alphabot als Objekte mit den Standardwerden initialisiert. Außerdem werden die Werte für die schwarzen / weißen Bereiche durch eine kurze Drehung auf beide Seiten kalibriert und zu Debugging-Zwecken ausgegeben.

```
71  while (GPIO.input(Button) != 0):
72      position, Sensors = TR.readLine()
73      print(
74          f"{int(position):4}, {int(Sensors[0]):4}, {int(Sensors[1]):4}, {int(Sensors[2]):4}, {int(Sensors[3]):4},
75      time.sleep(0.05)
76  Ab.forward()
```

Dieser Teil des Codes liest kontinuierlich die Sensorsignale aus und gibt sie auf der Konsole aus, solange der zentrale Joystick-Button nicht gedrückt wird. Anschließend beginnt der Alphabot sich vorwärtszubewegen.

```
78  while True:
79      position, Sensors = TR.readLine()
80      print(
81          f"{int(position):4}, {int(Sensors[0]):3}, {int(Sensors[1]):3}, {int(Sensors[2]):3}, {int(Sensors[3]):3},
82      if (Sensors[0] > 900 and Sensors[1] > 900 and Sensors[2] > 900 and Sensors[3] > 900 and Sensors[4] > 900):
83          Ab.setPWMA(0)
84          Ab.setPWMB(0)
85      else:
86          # The "proportional" term should be 0 when we are on the line.
87          proportional = position - 2000
88
89          # Compute the derivative (change) and integral (sum) of the position.
90          derivative = proportional - last_proportional
91          integral += proportional


105         power_difference = proportional/30 + integral/100000 + derivative*0
```

In diesem Teil wird die eigentliche Linienfolge-Logik ausgeführt. Dabei wird mit Hilfe der TR.readLine() Funktion laufend ein Wert für die relative Position des Roboters zur Linie berechnet. Zu Beginn wird außerdem geprüft, ob sich der Roboter gerade auf keinem festen Untergrund befindet (alle Sensoren liefern Werte über 900 zurück). In diesen Fall wird die Geschwindigkeit der beiden Motoren auf 0 gesetzt, sodass der Roboter stoppt. Ist dies nicht der Fall wird mit Hilfe der Codezeile 105 die benötige Regelung als „power_difference“ berechnet. Dabei haben drei Variablen Einfluss: Zuerst die Variable „proportional“, die in Zeile 87 in obigen Codeauszug berechnet wird und die Abweichung von der Idealposition 2000 berechnet. Außerdem wird die Variable „integral“ in Codezeile 91 durch Summierung der zuvor bestimmten „proportional“-Werte bei jedem Schleifendurchlauf aktualisiert. Der Einfluss des Faktors „derivative“, der die Veränderung des proportionalen Werts im Vergleich zum letzten gespeicherten Wert vergleicht, wurde nach Tests der Performance des Line-Trackings nicht einbezogen.

Die im Anschluss in Codezeile 105 verwendeten Koeffizienten sollten je nach Anwendungsfall individuell bestimmt werden, um ein möglichst gutes Verfolgen der Linie zu ermöglichen.

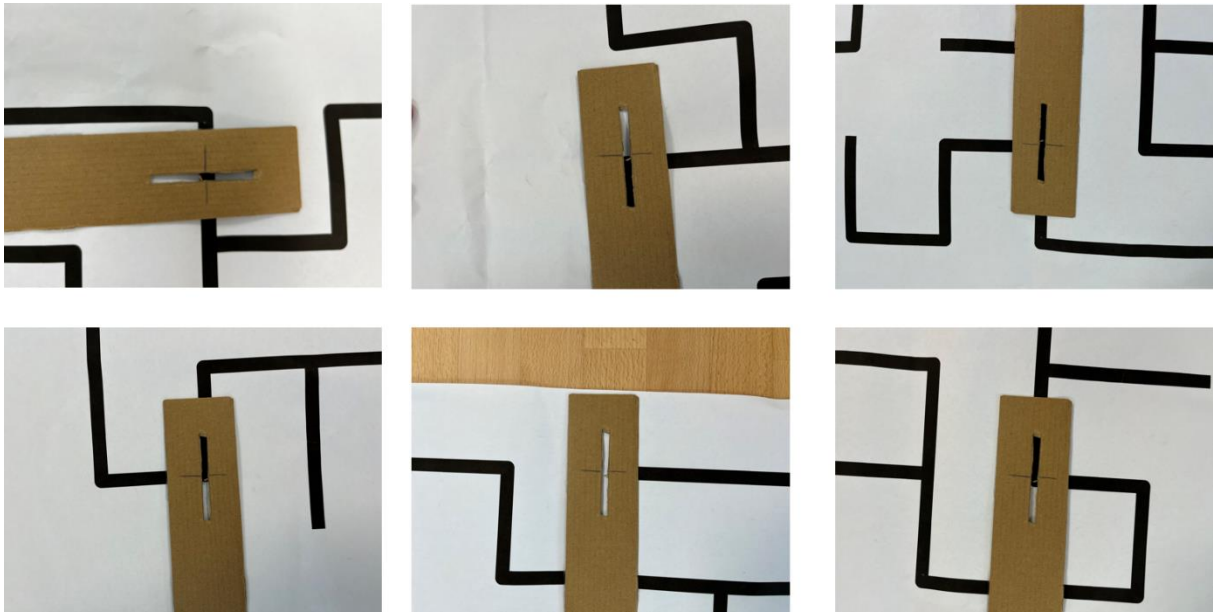
```
107         if (power_difference > maximum):
108             power_difference = maximum
109         if (power_difference < - maximum):
110             power_difference = - maximum
111         print(position, power_difference)
112         if (power_difference < 0):
113             Ab.setPWMA(maximum + power_difference)
114             Ab.setPWMB(maximum)
115         else:
116             Ab.setPWMA(maximum)
117             Ab.setPWMB(maximum - power_difference)
118
```

Als letzter Schritt erfolgt nun die tatsächliche Anpassung der Drehgeschwindigkeit der einzelnen Räder. Dabei wird die im vorherigen Teil berechnete „power_difference“ verwendet, um zu entscheiden welches der beiden Räder nun langsamer gedreht werden muss, damit der Alphabot möglichst zentral auf der Linie weiterfährt.

Insgesamt kann man festhalten, dass die Linienverfolgung ein zentrales Element für die Erreichung der Zielstellung ist, da diese Funktionalität immer wieder zum Befahren der geradlinigen Teilstücke des Labyrinths benötigt wird. Eine starr programmiertes Geradeausfahren wie es mit der reinen Ab.forward()-Funktion ohne Regelung der Motordrehgeschwindigkeiten anhand des Line-Trackings denkbar wäre, führt zu unbefriedigenden Ergebnissen und wäre nicht ausreichend um zuverlässig auf der Linie zu bleiben.

## 4.3 Erkennung von Wegpunkten

Die Wegpunkterkennung spielt im Hinblick auf das erfolgreiche Meistern des Labyrinths eine wichtige Rolle. Wie bereits zuvor erwähnt, sind fünf Infrarot-Sensoren auf der Unterseite des Alphabots verbaut. Diese werden für die Erkennung von Wegpunkten, ähnlich wie beim Line-Tracking genutzt. Es werden alle fünf Sensorwerte eingelesen, in einer Liste abgespeichert und im Terminal ausgegeben. Hierbei war sehr gut zu erkennen, welcher Sensor sich gerade auf der schwarzen Linie befindet und welcher auf weißem Hintergrund. Man konnte feststellen, dass die Sensoren, die über der schwarzen Linie liegen, in den meisten Fällen einen Wert von 100 bis 1000. Mit Hilfe der folgenden Abbildung kann verstanden werden, wie der Alphabot durch seine Sensoren die Linien wahrnimmt und welche Einzelfälle dementsprechend genauer betrachtet werden müssen.



Beim grundsätzlichen Ablauf des Gesamtprogramms bewegt sich der Roboter durch das Line-Following entlang der Linie und wertet stetig die gemessenen Sensorwerte aus. Während jedes Schleifendurchlaufs werden dabei durch die Funktion erkennen() die Sensorwerte ausgewertet. Die nachfolgende Abbildung definiert diese Funktion.

```
def erkennen(Sensors, higher):
    poss = [False, False, False] #Geradeaus, links, rechts
    if Sensors[2] >= higher or Sensors[1] >= higher or Sensors[3] >= higher:
        poss[0] = True

    if Sensors[0] >= higher:
        poss[1] = True

    if Sensors[4] >= higher:
        poss[2] = True
    return poss
```

Sensors ist hierbei die vorhin erwähnte Liste bestehend aus den 5 eingelesenen Sensorwerten. Der Parameter „higher“ ist ein Schwellwert, der verwendet wird, um festzustellen, ob der Alphabot sich gerade mit dem jeweiligen Sensor über der schwarzen Linie oder dem schwarzen Hintergrund befindet. In unserem Fall haben wir diesen Wert auf 100 gesetzt, um die bestmögliche Erfolgsrate unseres Roboters im Labyrinth erzielen zu können.

Anschließend werden verschiedene Bedingungen überprüft, um festzustellen, ob bestimmte Sensoren des Alphabots einen Wert über oder gleich dem Schwellwert haben:

Wenn der Sensorwert des Sensors[2], also des mittleren Sensors, größer oder gleich dem Schwellwert „higher“ ist oder der Sensor[1], also der Sensor links neben dem mittleren Sensor, größer oder gleich „higher“ ist oder der Sensor[3], sprich der Sensor rechts neben mittleren Sensor, einen höheren Wert im Vergleich zum festgelegten Schwellwert besitzt, so wird der erste Wert einer neu erstellten Liste „poss“ auf den Wert „True“ gesetzt. Die Liste „poss“ beschreibt hierbei die verschiedenen Arten an Möglichkeiten, die dem Alphabot im Labyrinth zum Abbiegen zur Verfügung stehen: [geradeaus, links, rechts].

Danach wird die vorher definierte erkennen()-Funktion im Gesamtprogrammablauf angewendet.

```
higher = 100    # Schwellwert für schwarze Linie

poss = erkennen(Sensors, higher)

# Falls abbiegen (links oder rechts) möglich ist
if poss[1] == True or poss[2] == True:

    # Falls abbiegen (links und rechts) möglich
    if poss[1] == True and poss[2] == True:
        Ab.stop()
        time.sleep(0.5)
        Ab.moveoverline()
        _, new_values = TR.readLine()
        if new_values[2] > higher:
            poss[0] = True
        else:
            poss[0] = False
```

Es wird überprüft, ob an der 2. oder 3. Stelle der Liste „poss“ ein Wegpunkt erkannt wurde. In diesem Fall, ob die Möglichkeit links oder rechts abzubiegen gegeben ist. Falls diese Bedingung erfüllt ist, wird zuerst der Alphabot mit der Funktion Ab.stop() angehalten, mit time.sleep() kurz pausiert und dann mit der zuvor beschriebenen moveoverline()-Funktion ein Stück über den erkannten Wegpunkt hinausgefahren ,um zu prüfen, ob ein Geradeausfahren auch möglich ist.

Dafür werden die Sensorwerte erneut eingelesen und in der neuen Variable „new_values“ abgespeichert. Im Folgenden wird überprüft, ob der mittlere Sensorwert der neuen Variable größer als der vorherige Schwellwert „higher“ ist. Wenn dies der Fall ist, wird die erste Position der Liste „poss“ auf den Wert „True“ gesetzt, was wie vorhin erwähnt, bedeutet, dass der Alphabot die Möglichkeit besitzt geradeaus zu fahren. Andernfalls wird dieser Wert auf „False“ gesetzt.

Im weiteren Verlauf des Codes wurden auch die anderen Entscheidungsmöglichkeiten betrachtet und entsprechende Reaktionen implementiert.

## 4.4 Strategie bei der Reaktion auf Wegpunkte

In Betracht auf die Reaktion auf die erkannten Wegpunkte, haben wurde grundlegend festgelegt, den Alphabot im Fall einer T-Kreuzung nach links fahren zu lassen und im Fall, dass das Geradeausfahren möglich ist auch grundsätzlich dies zu tun.

Hierbei ist das Problem aufgetreten, dass der Roboter sich nur selten tatsächlich beim gleichen Schleifendurchlauf der Wegpunkterkennung mit den beiden außen liegenden Sensoren auf der schwarzen Linie befindet. Dadurch konnten beispielsweise T-Kreuzungen nur unzuverlässig detektiert werden. Ein möglicher Lösungsansatz hierfür wäre es, während sich der Alphabot über die Linie bewegt, laufend die Sensoren auszuwerten und nicht nur nachdem dieses Manöver ausgeschlossen ist. Dieses Problem war bei der Beobachtung des Roboters dadurch zu erkennen, dass der Alphabot dazu geneigt hat, an T-Kreuzungen nach rechts abzubiegen obwohl eigentlich das Abbiegen nach Links beabsichtigt wäre.

Insgesamt kann man bezogen auf die Strategie feststellen, dass hierbei noch Potential im Hinblick auf eine intelligentere Entscheidungsfindung beim Durchfahren des Labyrinthes besteht. Denkbar wäre zum Beispiel tatsächliche Suchalgorithmen zu verwenden, die den kürzesten Weg zur Lösung des Labyrinths finden.

## 4.5 Erkennung von Zyklen im Labyrinth

Ein interessanter Bereich bei der Lösung von Labyrinthen stellt auch das Erkennen von Zyklen / Sequenzen im Labyrinth dar. Zyklen stellen dabei Bereiche im Labyrinth dar, die einfach beschrieben einen Kreis im Labyrinth darstellen, aus dem der Roboter rechtzeitig ausbrechen muss, um nicht dauerhaft darin festzustecken. Dabei wurden bereits grundlegende Überlegungen angestellt, um solche Zyklen im Labyrinth zu erkennen und korrekt darauf zu reagieren:

Die Idee dabei ist es bei jeder Abbiegung die jeweilige Entscheidung (Links oder Rechts) laufend in einer Liste abzuspeichern. Diese Liste kann anschließend unter Hilfe bereits zu diesen Zwecken bekannten Algorithmen analysiert werden, um solche Zyklen zu erkennen. Um diese Funktionalität in unserer Umsetzung verwenden zu können, wäre es allerdings zuerst notwendig, die Erkennung der Wegpunkte noch robuster zu gestalten, sodass die Zyklen-Erkennung auch stets die tatsächlich korrekten Wegpunkt als Input erhält.

## 5. Fazit und mögliche Verbesserungen

Im Rahmen dieses Projekts haben wir erfolgreich den Alphabot2 auf Raspberry Pi-Basis beigebracht, durch ein Labyrinth zu fahren. Dafür wurden zunächst grundlegende Funktionen genutzt, die es dem Roboter ermöglichen, einer Linie zu folgen. Durch die Verwendung von Sensoren zur Linienverfolgung auf dem Untergrund und der Steuerung der Motoren konnten wir eine selbständige Navigation des Roboters durch das Labyrinth erreichen.

Das Projekt hat uns wertvolle Einblicke in Hard- und Software von autonom fahrenden Fahrzeugen gegeben. Dennoch gibt es selbstverständlich noch Raum für Anpassungen und Verbesserungen, die das Verhalten des Roboters beim Durchfahren des Labyrinths noch weiter optimieren könnten. Ein Aspekt, den wir während des gesamten Verlaufs unseres Projektes immer wieder feststellen konnten, war der Unterschied zwischen der theoretischen Programmierung in einer idealen Umgebung im Vergleich zu den tatsächlichen physikalischen Rahmenbedingungen, die in der realen Welt auf den Roboter einwirken. So mussten wir an verschiedenen Stellen von unsren theoretischen Lösungsansätzen abweichen, um das tatsächlich vorgegebene Ziel zu erreichen.

Während der Bearbeitung sind wir außerdem auf einige Verbesserungsansätze gestoßen, die das Verhalten des Roboters im Labyrinth weiter verbessern könnten oder zu noch fortgeschritteneren Ergebnissen führen würden.

Ein Bereich, der definitiv ausbaufähig ist, ist das in der aktuellen Version starr vorprogrammierte Abbiegen an Links- oder Rechtskurven. Um dies zu verbessern könnte der Roboter während der Drehung laufend die Werte der Sensoren abrufen,

solange bis er sich mit seinem zentral platzierten Sensor wieder mittig auf der Linie befindet. Anschließend könnte wieder mit dem Linienfolgen bis zur nächsten Kreuzung oder Abbiegung fortgefahren werden. So wäre sichergestellt, dass sich der Roboter immer im genau passenden Winkel dreht.

Weiterhin wäre es denkbar die allgemeine Robustheit des Roboters auch beim Durchfahren unbekannter Labyrinthe zu erhöhen. Dabei könnte zum Beispiel angestrebt werden, dass der Alphabot auch bei variablen Linienstärken das Labyrinth durchfahren kann oder im Falle von Kreuzungen und Abbiegungen, welche nicht in einem 90° Winkel angeordnet. Möglich wäre auch hinsichtlich eines möglichst schnellen und damit effizienten Lösens des Labyrinths durch den Roboter zu optimieren.

Auch ein Einsatz von Algorithmen des Machine-Learnings, wie beispielsweise des Reinforcement-Learnings, wäre durchaus denkbar. Hierbei würde der Roboter sich nun selbständig anhand einer Optimierungsfunktion und eigener Versuche an eine Lösungsstrategie herantasten.

Insgesamt hat dieses Projekt gezeigt, dass der Alphabot2 auf Raspberry Pi-Basis eine solide und einsteigerfreundliche Plattform für die Entwicklung autonomer Fahrzeuge bietet. Mithilfe eigener Anpassungen und Erweiterungen der Funktionalität kann der Alphabot2 zu einem vielseitigen und lernfähigen Roboter werden, der zur Lösung verschiedenster Problemstellungen verwendet werden kann.