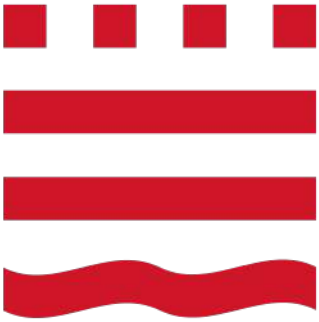


Technische Hochschule
Brandenburg
University of
Applied Sciences

Dokumentation
"Painbot"

Fachbereich Informatik und Medien
An der Technischen Hochschule Brandenburg

AMS-Projekt im WS 2018/19

Pizzabote

vorgelegt von

Hüseyin Celik
und
Janik Weideman

im Januar 2019

Betreuer: Prof. Dr.-Ing. Jochen Heinsohn
Dipl.Inform. Ingo Boersch

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Aufgabenstellung

Die Aufgabe besteht darin, einen autonomen fahrenden LEGO-Roboter zubauen. Dieser soll einen oder mehrere Fahraufträge aufnehmen können und diese erfolgreich ausführen, indem er einen Ball mit sich führt und am Ziel auf einen Reifen ablegt. Die Route soll optimal sein und in angemessener Zeit berechnet werden. Damit der Roboter das Streckennetz, dass aus einem einfachen Gitternetz besteht, erfolgreich folgen kann, sollen Sensoren am Roboter angebracht werden.

2 Bedingungen

Das Streckennetz ist ein einfaches schwarzes Gitter auf einem weißen Tisch. Das Netz besteht aus Strecken, Kreuzungen und den Lieferadressen. Kreuzungen können gesperrt sein. Es darf nur eine einzelne Pizza pro Fahrt geliefert werden. Die Lichtverhältnisse auf dem Tisch müssen Tageslicht entsprechen. Die Karte wird so kodiert bzw. dekodiert, dass eine befahrbare

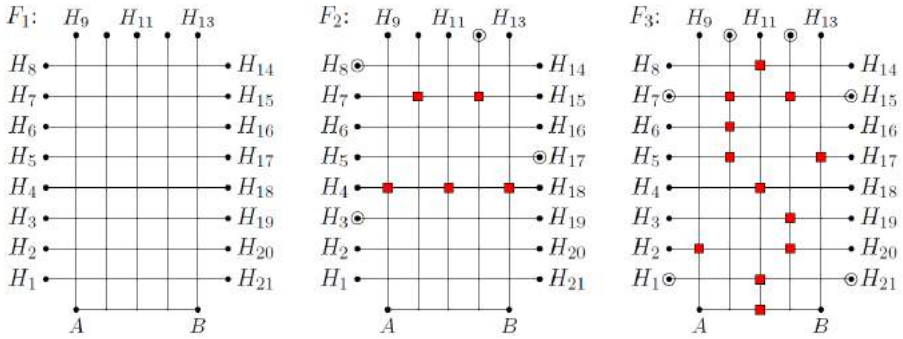


Abbildung 1: Fahrkarte [JI18]

Kreuzung „.“, eine gesperrte Kreuzung „x“ und eine Lieferadresse „F“ entsprechen. Diese werden über eine LAN-Verbindung an den Roboter übermittelt

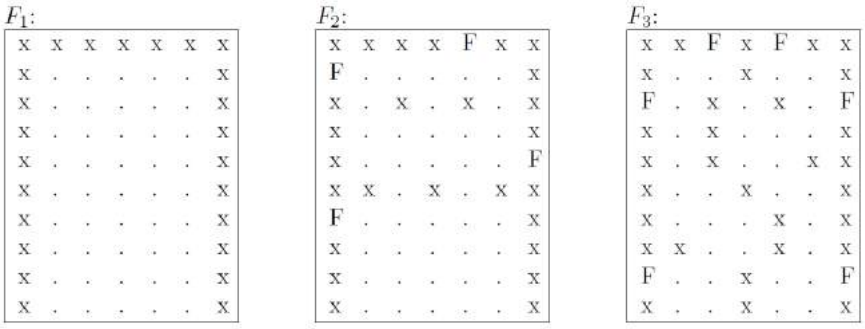
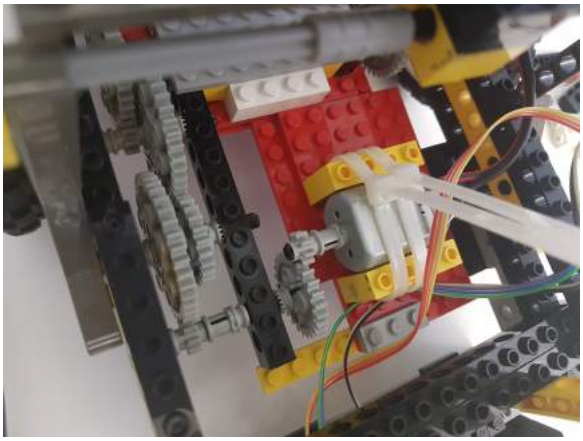


Abbildung 2: Fahrkarten Kodierung [JI18]

3 Hardware

Der überwiegende Teil des Roboters besteht aus einfachen LEGO-Bausteinen. Damit sich der Roboter bewegen kann, bestehen die beiden Getriebe aus 4 Getriebestufen. Die erste Stufe hat ein Umsetzungsverhältnis von $\frac{8}{24}$ also $\frac{1}{3}$. Die zweite und vierte Stufe haben eine Umsetzung von $\frac{8}{40}$ das entspricht $\frac{1}{5}$. Die dritte Stufe hat eine Umsetzung von $\frac{24}{40}$, was $\frac{3}{5}$ entspricht. Dadurch ergibt sich eine Gesamtumsetzung von 1 zu 125. Das bedeutet dreht sich der Motor 125 mal, drehen sich die Räder 1 mal.
Die Getriebe werden, jeweils mit einem Gleichstrommotor angetrieben. Zur Stabilisierung und zur Unterstützung der Wendigkeit des Roboters wurde im hinteren Teil ein Stützrad angebracht.

Abbildung 3: Getriebe



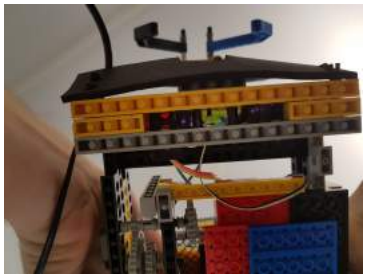
(a) Gleichstrommotor

(b) Stützrad

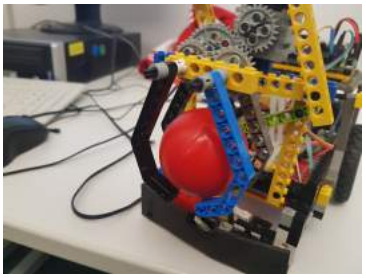
Abbildung 4: Gleichstrommotor und Stützrad

Damit der Roboter den Linien auf dem Boden zuverlässig folgen kann, sind vorn zwei Optokoppler angebracht. Sie werden dazu genutzt, um eine farbliche Veränderung von weiß nach schwarz beziehungsweise. von schwarz auf weiß zu registrieren. Dadurch sind wir softwareseitig in der Lage auf die Veränderung zu reagieren. Die Optokoppler befinden sich mittig am Roboter und haben einen Abstand der minimal größer ist, als die Linienbreite. Um den kleinen Plastikball, der in unserem Beispiel eine Pizza darstellt, zum schwarzen Reifen liefern und ablegen zu können wurde vorn am Roboter ein Plastikgreifarm aus LEGO-Bausteinen angebracht. Dieser wird von einem Gleichstrommotor angetrieben. Dadurch ist der Greifarm in der Lage sich zu öffnen und zu schließen. Damit der Greifarm den Kontakt mit dem Reifen registriert kann ist am Greifarm ein Tastsensor angebracht. Sobald dieser einen Anschlag registriert, fängt der Motor an sich in Bewegung zusetzen und öffnet dadurch den Greifarm

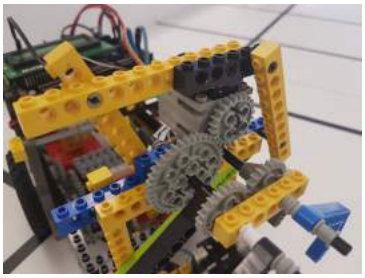
und die Pizza fällt auf den Reifen. Dadurch gilt die Pizza als geliefert.



(a) Sensoren



(b) Greifer



(c) Greifer mit Motor und Zahnrädern

Abbildung 5: Optokoppler, Greifer, Motor und Zahnräder

Gesteuert und programmiert wird auf dem AKSEN-Board. Dieser wurde 2003 an der Technischen Hochschule Brandenburg entwickelt und besitzt einen Mikrocontroller der 8051-Familie, eine serielle Schnittstelle, welches dazu genutzt wird das lauffähige Programm zu übertragen. Außerdem besitzt es Motor-Ports, die mit den Servomotoren verbunden sind und angesteuert werden. Die Digital-Ports werden mit den Optokopplern verbunden. Die Dip-Schalter auf dem Board werden dazu genutzt, um vor antritt der Fahrt die genaue Startposition des Roboters ans AKSEN-Board weiter zu leiten. Die Stromversorgung des AKSEN-Boards wird anhand von $5 * 1,2V$ NiMH-Mignon Akkupacks mit 2500 mAh sichergestellt. Das Akkupack ist im hinteren Teil des LEGO-Roboters befestigt, sodass eine sichere Stromversorgung während der Fahrt gesichert ist. Da bei diesem Roboter das meiste Selbst zusammengebaut ist, mussten für die Befestigung von einigen Teilen wie zum Beispiel den Sensoren Kabelbinder und Klettverschlüsse verwendet werden.

(a) Akku

(b) AKSEN-Board [Fac05]

Abbildung 6: Akku, AKSEN-Board

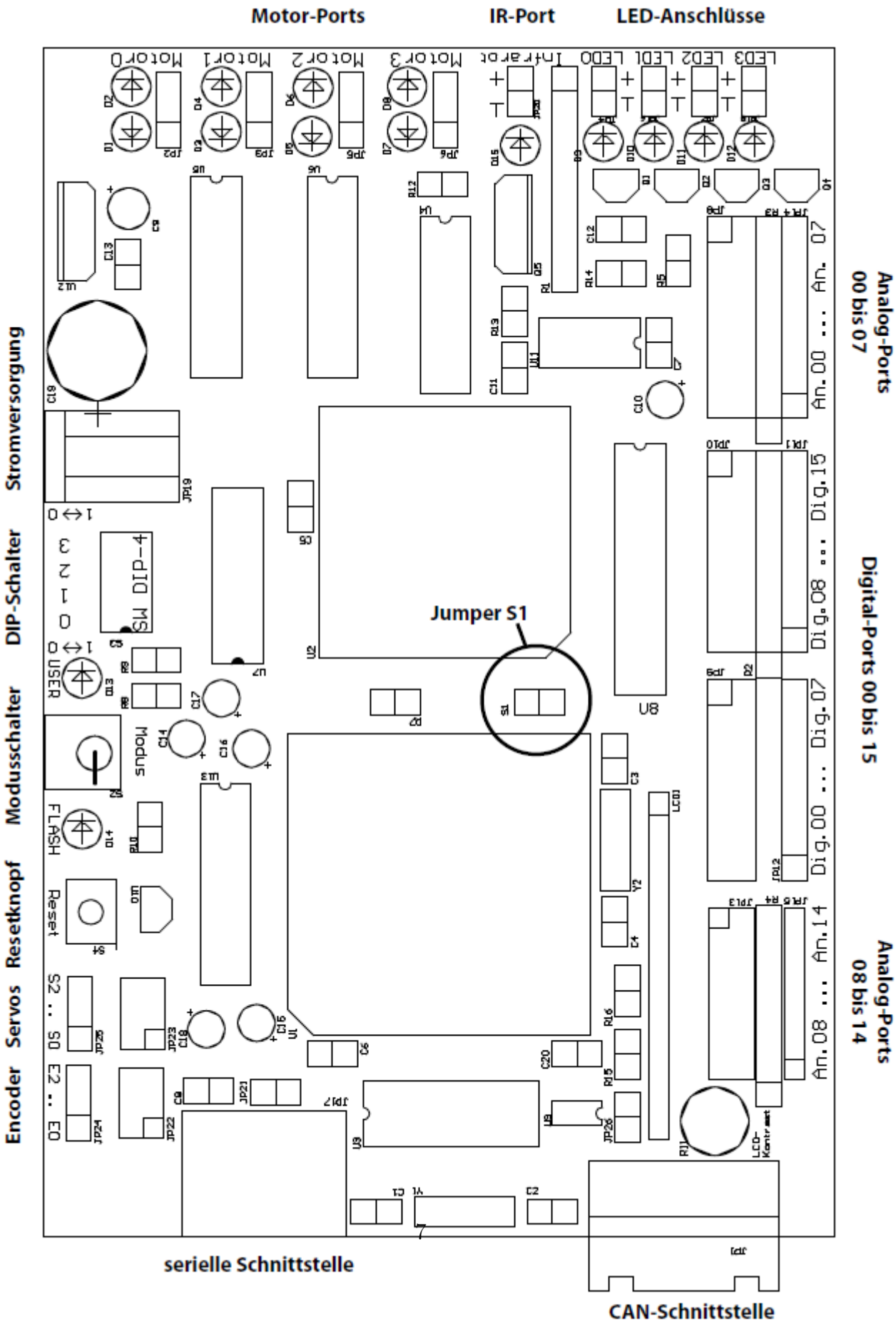
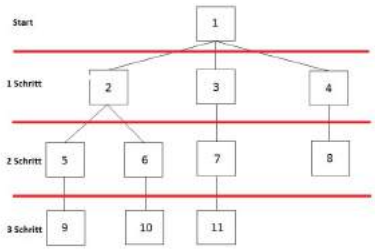


Abbildung 7: Schemazeichnung des AKSEN-Boards [Fac05]

4 Lösungsvarianten der Wegfindung

4.1 Breitensuche

Die Breitensuche ist eine uninformierte Suche zum Durchsuchen von Knoten in einem Graphen. Durch Expansion der einzelnen Suchknoten in die Breite findet die Breitensuche immer einen optimalen Weg. Aus diesem Grund eignet sich prinzipiell das Suchverfahren für unser Projekt. Jedoch haben uns die Zweifel an der möglichen langen Laufzeit als auch am unvorhersehbaren hohen Speicherplatzverbrauch, der beim AKSEN-Board knapp bemessen ist, davon abgehalten dieses Verfahren zu nutzen.



(a) Breitensuche grafische Darstellung [Kar15]

```
function breitensuche()
   agenda := [start()];
   while agenda ≠ []
      aktknoten := delete_first(agenda);
      if ziel?(aktknoten) then return aktknoten;
      agenda := append(agenda , expand(aktKnoten)) ;
   return „keine Loesung gefunden".
```

(b) Breitensuche Pseudocode [BHS07]

Abbildung 8: Breitensuche

4.2 A*-Suche

Die A*-Suche ist im Gegensatz zur Breitensuche eine informierte Suche. Dabei untersucht der Algorithmus zuerst immer die Knoten, die am wahrscheinlichsten am schnellsten zum Ziel führen. Dafür benötigt die Suche eine Funktion f, die wie folgt definiert ist: $f(x) = g(x) + h(x)$, wobei $g(x)$ die Kosten vom Startknoten zum aktuellen Knoten x angibt und $h(x)$ die geschätzten Kosten vom aktuellen Knoten x zum Zielknoten. Hierbei ist es wichtig zu beachten, dass die Funktion $h(x)$ die echte Distanz nicht überschätzen darf.

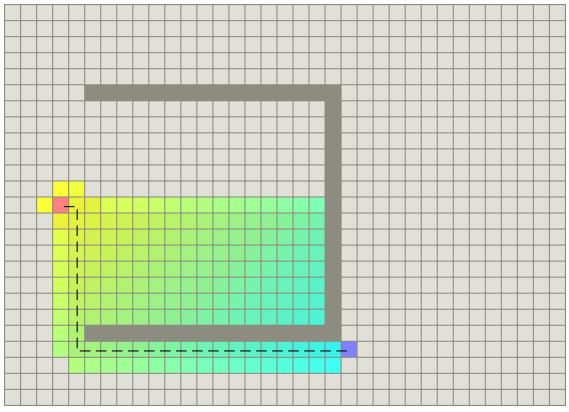


Abbildung 9: A*-Suche grafische Darstellung [Bri19]

Der A*-Algorithmus findet ähnlich wie die Breitensuche einen optimalen Weg. Jedoch unterscheidet sie sich insofern, dass sie gerichtet zum Ziel expandiert und dadurch in der Regel von der Laufzeit schneller und vom Speicherplatzverbrauch deutlich günstiger ist, als die Breitensuche. Trotz dieser Vorteile, eignet sich die A*-Suche für unsere Aufgabenstellung leider nicht, da wir uns auf einem Gitternetz bewegen, welches in alle Richtungen jeweils die Kosten von 1 besitzen und die Lage der Startpunkte sind so ge-

wählt, dass der A*-Algorithmus in unserem Fall nahezu identisch zur Breitensuche agieren würde und uns dadurch keine nennenswerten Vorteile bringen würde.

4.3 Jump Point Search

Die Jump-Point-Suche ist eine Optimierung des A*-Algorithmuses für Gitternetze mit Gleichen Kostenlängen zu allen Nachbarknoten und findet immer einen optimalen Weg, sofern ein Weg vorhanden ist. Der Algorithmus reduziert die Symmetrien vom Startknoten zum Zielknoten. Dabei werden Knotenpunkte eliminiert, bei denen ausgegangen werden kann, dass diese schon von anderen Nachbarknoten zu gleichen Kosten erreicht werden kann. Das Konzept ist höchst interessant und sollte für andere Aufgabenstellungen sehr gut geeignet sein, jedoch haben wir uns bewusst dagegen entschieden, diesen Algorithmus in diesem Projekt anzuwenden, da wir zum einen der Meinung sind, das der Algorithmus zu Komplex für diese Aufgabe ist und zum anderen die Zeit nicht ausreichend ist, den Algorithmus vollständig zu verstehen und diesen dann Programmiertechnisch in einer angemessen Zeit umzusetzen. Für genaue Informationen zum Algorithmus empfehlen wir das Paper von [HB08]

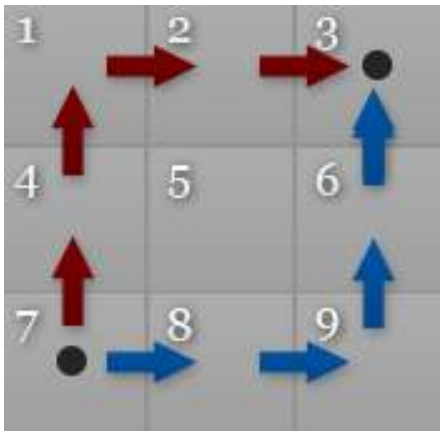


Abbildung 10: Symmetrien in der A*-Suche [Pod13]

5 Software und finaler Lösungsweg

5.1 Implementierung der Wegfindung

Die Software ist in der Programmiersprache C geschrieben. Sie enthält mehrere kleinere Funktionen, die jeweils einen Teil der Aufgabenstellung behandelt und löst. Die Teilaufgaben bestehen aus: einen optimalen Pfad finden, Pfad zum Ziel planen, eine Linie geradeaus folgen, Links und Rechts abbiegen, Greifarm öffnen und schließen. Zunächst erhält der Roboter “Painbot” die Wegbeschreibung aus dem fa-import-Header. Dieser beschreibt alle 70 Kreuzungen auf dem Spielfeld. Die Beschreibung konvertiert er dann in ein zweidimensionales char-Array. Dabei werden die nicht passierbaren Kreuzung zu einer 70, die passierbaren zu einer 71 und die Ziele zu einer 72. Anschließend liest der Roboter seine Startposition aus. Und fügt sie dem Muster durch eine 0 hinzu.
Im nächsten Schritt werden im zweidimensionalen Array die Nachbarn (oben, unten, links und rechts) von 0 mit einer 1 ausgefüllt. Bedingung ist dabei, dass die Kreuzung frei ist, d.h. eine 71 an dieser Stelle definiert ist. Darauffolgend sorgt der Algorithmus dafür, dass die Nachbarn der 1 im Array mit einer 2 gefühlt werden unter der gleichen Bedingung der freien Kreuzung. Der Algorithmus füllt das Array nach diesem Prinzip weiter bis zur Zahl 69. Als nächstes sucht das Programm nach den Nachbarn von 72 (Zielorte) im Array. Dabei findet

```
//Kovertierung 1D zu 2D Muster
for (x = 0; x <= 10; x++)
{
    for (y = 0; y < 7; y++)
    {
        if (Muster1D[i] == 'x')
            Muster2D[x][y] = 70;
        else if (Muster1D[i] == '.')
            Muster2D[x][y] = 71;
        else if (Muster1D[i] == 'F')
            Muster2D[x][y] = 72;
        i++;

    }
}
```

(a) Konvertierung der Eingabe

```
//Startpunkt
if(dif()==1)
    Muster2D[9][1] = 0;
else
    Muster2D[9][5] = 0;
```

(b) Startpunkt setzen

Abbildung 11: Konvertierung der Eingabe und Startpunkt festlegen

er die kleinsten Nachbarn aller Zielorte heraus und speichert diesen Zielort dann als das Ziel mit der kleinsten Distanz zum Roboter. Dabei speichert er auch die erste Wegbeschreibung als Himmelsrichtung zum Roboter.
Nun folgt der Algorithmus den Zahlen rückwärts Nachbar für Nachbar bis zur 0. Dabei werden die Himmelsrichtungen Nord(N), Ost(O), Süd(S), West(W) gespeichert. Zu beachten ist hierbei, dass die Himmelsrichtungen entgegengesetzt angeordnet sind, da der Weg rückwärts abgegangen wird. Am Ende der Wegbeschreibung im char steht eine 0 damit das Ende der Kette bekannt ist.
Im nächsten Schritt wird die Himmelsrichtung in eine Wegbeschreibung umgewandelt. Die Himmelsrichtungen Norden, Osten, Süden und Westen müssen in die Wegbeschreibungen Rechts, Links oder Geradeaus umgewandelt werden. Hilfe im Arbeitsprozess war dabei eine Uhr, um die Wegbeschreibung geben zu können. Die Betrachtung der Himmelsrichtungen erfolgt wie auf einem Kompass. Bewegt sich die Ausrichtung mit Hilfe des Kompasses im Uhrzeigersinn, bedeutet das übersetzt die Wegbeschreibung nach rechts. Dementsprechend erfolgt mit einer Ausrichtung entgegensetzt des Uhrzeigesinns die Wegbeschreibung nach links. Wenn sich die Kompassausrichtung nicht ändert, ist die Wegbeschreibung "geradeaus". Diese Umwandlung lässt sich am besten durch ein Array bewältigen. Beispielsweise ist der Roboter "Painbot"bisher nach Norden gefahren und muss an der nächsten Kreuzung nach Osten. Im Array ist nach Kompass-Orientierung Norden als Ausgangspunkt eingestellt. Es erfolgt eine Betrachtung der eigenen Position und der Nachbarn. Das Programm stellt dabei fest, dass es im Array +1 springen muss. Die daraus resultierende Wegbeschreibung ist nach rechts zu fahren Abbildung 14.

```
//Füllen mit Schritte bis
for (i = 0; i < 70; i++)
{
    for (x = 1; x < 10; x++)
    {
        for (y = 1; y < 6; y++)
        {
            if (Muster2D[x][y] == 71)
            {
                if (Muster2D[x + 1][y] == i)
                    Muster2D[x][y] = i + 1;
                else if (Muster2D[x][y + 1] == i)
                    Muster2D[x][y] = i + 1;
                else if (Muster2D[x - 1][y] == i)
                    Muster2D[x][y] = i + 1;
                else if (Muster2D[x][y - 1] == i)
                    Muster2D[x][y] = i + 1;
            }
        }
    }
}
```

(a) Karte fluten

```
for (x = 1; x < 9; x++)
{
    for (y = 1; y < 6; y++)
    {
        if (Muster2D[x][y] < i&&Muster2D[x][y] > 0)
        {
            if (Muster2D[x + 1][y] == -12)
            {
                xs = x; ys = y; i = Muster2D[x][y];
                hweg[k] = 'S';
            }
            else if (Muster2D[x][y + 1] == 72)
            {
                xs = x; ys = y; i = Muster2D[x][y];
                hweg[k] = 'O';
            }
            else if (Muster2D[x - 1][y] == 72)
            {
                xs = x; ys = y;  i = Muster2D[x][y];
                hweg[k] = 'N';
            }
            else if (Muster2D[x][y - 1] == 72)
            {
                xs = x; ys = y; i = Muster2D[x][y];
                hweg[k] = 'W';
            }
        }
    }
}
```

(b) Wegsuche

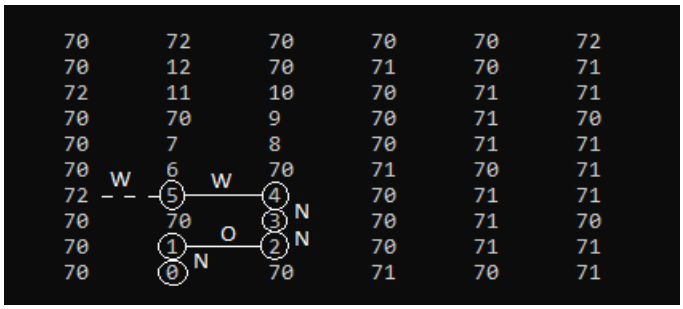
Abbildung 12: Karte fluten und optimalen Wegsuchen

5.2 Implementierung der Fahrt

Damit der Roboter der Linie folgt, werden die Außensensoren abgefragt. Sobald ein Sensor andere Signale wahrnimmt, wird in die entgegengesetzte Richtung korrigiert. Die "followLine"bricht ab, wenn sich beide Sensoren gleichzeitig auf der schwarzen Linie befinden.
Beim Rechtsfahren wird der rechte Motor gestoppt während der linke Motor die Geschwindigkeit beibehält. Nach einer Sekunde prüft der Roboter immer wieder den linken Sensor. Dieser schlägt an, wenn sich die Linie unter ihm befindet. In diesem Fall stoppt der Roboter den linken Motor und führt die "followLine"Methode aus. Beim Linksfahren erfolgt der gleiche Prozess in entgegengesetzt.
Beim Abliefern führt der Roboter auch erst die Methode “Follow-Line” aus bis der TastSensor anschlägt. Danach öffnet er den Greifer mit Hilfe seines Servo-Motors. Nun gibt er das Kommando, seine Wegbeschreibung umzukehren. Anschließend fährt er rückwärts und dreht sich durch den Befehl “turn-Right” um 180 Grad. Nun fährt er wieder zurück.

6 Vorschläge

Einer unserer Verbesserungsvorschläge wäre es, dass die LEGO-Roboter schon fertig gebaut uns übergeben werden beziehungsweise wir die gleiche Aufgabe oder eine ähnliche Aufgabe mithilfe von richtigen Robotern wie zum Beispiel den Pioniers bearbeiten. Das bauen der LEGO-Roboter hat unserer Meinung nach einen zu hohen Anteil an der Aufgabe in An-



(a) Grafische Darstellung: Gestrichelt Weg ein Schritt vorher erfolgt

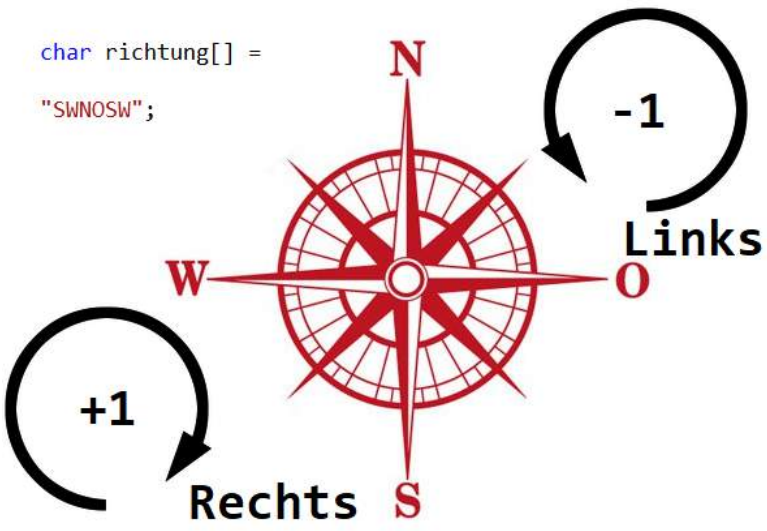
```
k++;
for (; i >= 0; i--) {
    if (Muster2D[xs - 1][ys] == i - 1) {
        xs--;
        hweg[k] = 'S';
        k++;
    }
    else if (Muster2D[xs + 1][ys] == i - 1) {
        xs++;
        hweg[k] = 'N';
        k++;
    }
    else if (Muster2D[xs][ys - 1] == i - 1) {
        ys--;
        hweg[k] = 'O';
        k++;
    }
    else if (Muster2D[xs][ys + 1] == i - 1) {
        ys++;
        hweg[k] = 'W';
        k++;
    }

}
hweg[k] = 0;
```

(b) Wegfindung

Abbildung 13: Wegfindung im Potentialfeld

spruch genommen. Gerade die Tatsache, dass vieles mit Kabelbindern oder Klebeband gelöst werden musste, hat dazu geführt, dass zu häufig die Sensoren sich während der Fahrten in irgendeiner Weise bewegt haben. Dadurch stimmte häufig das erwartete Ergebnis nicht mit dem echten Ergebnis überein, obwohl die Programmierung richtig war. Nachdem der Roboter wieder neu zusammengebaut wurde, hat das Ergebnis der Programmierung wie erwartet erfolgreich funktioniert. Außerdem finden wir es schade, dass keine gemeinsame Erforschung und Erarbeitung von Algorithmen zustande kam. Wir glauben, dass wenn man gemeinsam neue Lösungsansätze bearbeitet, wie zum Beispiel den Jump Point Search Algorithmus der Horizont der Studierenden Erweitert werden könnte und vieles davon Früchte für zukünftige Forschung beziehungsweise Vorteile für die zukünftige Arbeit in der Arbeitswelt hervorbringen könnte.



(a) Kompass

```
x = 0;
y = 2;


while (k > 0) {
    k--;
    if (hweg[k] == richtung[y]) {
        rweg[x] = 'G';
    }
    else if (hweg[k] == richtung[y + 1]) {
        rweg[x] = 'R';
        y++;
        if (y == 5)
            y = 1;
    }
    else if (hweg[k] == richtung[y - 1]) {
        rweg[x] = 'L';
        y--;
        if (y == 0)
            y = 4;
    }

    x++;
}
rweg[x] = 0;
```

(b) Implementierung

Abbildung 14: Grafische Darstellung der Orientierung

7 Quelltext

```
1  //Autoren: Janik Weidemann und Hüseyin Celik
2
3  //Standard-Include-Files
4  #include <stdio.h>
5  #include <regc515c.h>
6  #include <stdint.h>
7  #include "bool.h"
8
9  //Diese Include-Datei macht alle Funktionen der
10 //AkSen-Bibliothek bekannt.
11 //Unbedingt einbinden!
12 #include <stub.h>
13 #define FA1
14 #include"fa.h"
15
16 //refreshing and output string on display
17 void outputString(char* string);
18 void start();
19 void followLine();
20 void rfollowLine();
21 void geradeaus();
22 void turnRight();
23 void turnLeft();
24 void Abliefern();
```

```
void followLine(){
        motor_richtung(1,0);
        motor_richtung(2,0);
    while(!(analog(optoSensorRechts)>wert1 && analog(optoSensorLinks)>wert2)&&digital_in(0)==1){
        if(analog(6)>wert1)
        {//Rechts
            motor_pwm(2, standardSpeed);
            motor_pwm(1, 5);
        }
        else if(analog(7)>wert2)
        {// Links
            motor_pwm(1, standardSpeed);
            motor_pwm(2, 5);
        }
        else
        {//Mitte
            motor_pwm(1, standardSpeed);
            motor_pwm(2, standardSpeed);
        }
    }
}
```

Abbildung 15: Implementierung followLine

```
void turnRight(){

    motor_richtung(motorRechts,0);
    motor_richtung(motorLinks,0);

    motor_pwm(motorRechts, 0);
    motor_pwm(motorLinks, standardSpeed);

    sleep(1000);
    while(analog(optoSensorLinks)<wert2);
    motor_pwm(motorLinks, 0);
    followLine();


}
```

(a) Rechtsabbiegen

```
void turnLeft(){

    motor_richtung(motorRechts,0);
    motor_richtung(motorLinks,0);

    motor_pwm(motorRechts, standardSpeed);
    motor_pwm(motorLinks, 0);

    sleep(1000);
    while(analog(optoSensorRechts)<wert1);
    motor_pwm(motorRechts, 0);
    followLine();
}
```

(b) Linksabbiegen

Abbildung 16: Implementierung der Methoden turnRight und turnLeft

```
25
26 //Hauptprogrammroutine
27 #define wert1 100
28 #define wert2 100
29 #define wertlicht1 30
30 #define wertlicht2 30
31 #define slepi 150
32 #define optoSensorRechts 6
33 #define optoSensorLinks 7
34 #define motorRechts 1
35 #define motorLinks 2
36 #define motorVorwaerts 0
37 #define motorRueckwaerts 1
38 #define standardSpeed 10
39
```

```
void Abliefern(){

    followLine();
    motor_pwm(motorRechts, 0);
    motor_pwm(motorLinks, 0);

    servo_arc(2,90);
    umkehren();

    sleep(900);

    motor_richtung(motorRechts,1);
    motor_richtung(motorLinks,1);
    motor_pwm(motorRechts, standardSpeed);
    motor_pwm(motorLinks, standardSpeed);

    sleep(200);

    motor_pwm(motorRechts, 0);
    motor_pwm(motorLinks, 0);

    servo_arc(2,-90);
    turnRight();

}
```

Abbildung 17: Implementierung der Methode abliefern

```
40      char Muster2D[10][7];
41      char x, y, xs=0, ys=0, k;
42      char hweg[20];
43      char rweg[]="HHHHHHHH";
44      char i = 0;
45      char Muster1D[] =
            "xFxxxFxx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.x";
46      char richtung[] = "SWNOSW";
47      char Muster2D[10][7];
48      char count=0;
49
50  void outputString(char* string){
51     lcd_cls();
52     lcd_puts(string);
53  }
54
55  void start(){
56     motor_richtung(1, 0);
57     motor_richtung(2, 0);
58     while((analog(optoSensorRechts)>wertlicht1 &&
           analog(optoSensorLinks)>wertlicht1)){}
59     followLine();
60  }
61
```

```
62  void followLine(){
63       motor_richtung(1,0);
64       motor_richtung(2,0);
65     while(!(analog(optoSensorRechts)>wert1 &&
           analog(optoSensorLinks)>wert2)&&digital_in(0)==1){
66       if(analog(6)>wert1)
67       {//Rechts
68          motor_pwm(2, standardSpeed);
69          motor_pwm(1, 5);
70       }
71       else if(analog(7)>wert2)
72       {// Links
73          motor_pwm(1, standardSpeed);
74          motor_pwm(2, 5);
75       }
76       else
77       {//Mitte
78          motor_pwm(1, standardSpeed);
79          motor_pwm(2, standardSpeed);
80       }
81     }
82  }
83
84  void umdrehen(){
85     motor_richtung(motorRechts,0);
86     motor_richtung(motorLinks,0);
87     motor_pwm(motorRechts, 0);
88     motor_pwm(motorLinks, standardSpeed);
89     Wegfindung();
90     sleep(1000);
91     while(analog(optoSensorLinks)<wert2);
92     motor_pwm(motorLinks, 0);
93     count++;
94     followLine();
95  }
96
97  void geradeaus(){
98     motor_pwm(1, 8);
99     motor_pwm(2, 8);
100    sleep(700);
101    motor_pwm(1, 0);
102    motor_pwm(2, 0);
103    followLine();
104 }
105
106 void turnRight(){
```

```
107     motor_richtung(motorRechts,0);
108     motor_richtung(motorLinks,0);
109     motor_pwm(motorRechts, 0);
110     motor_pwm(motorLinks, standardSpeed);
111     sleep(1000);
112     while(analog(optoSensorLinks)<wert2);
113     motor_pwm(motorLinks, 0);
114     followLine();
115 }
116 
117 void turnLeft(){
118     motor_richtung(motorRechts,0);
119     motor_richtung(motorLinks,0);
120     motor_pwm(motorRechts, standardSpeed);
121     motor_pwm(motorLinks, 0);
122     sleep(1000);
123     while(analog(optoSensorRechts)<wert1);
124     motor_pwm(motorRechts, 0);
125     followLine();
126 }
127 
128 void abliefern(){
129     followLine();
130     motor_pwm(motorRechts, 0);
131     motor_pwm(motorLinks, 0);
132     servo_arc(2,90);
133     umkehren();
134     sleep(900);
135     motor_richtung(motorRechts,1);
136     motor_richtung(motorLinks,1);
137     motor_pwm(motorRechts, standardSpeed);
138     motor_pwm(motorLinks, standardSpeed);
139     sleep(200);
140     motor_pwm(motorRechts, 0);
141     motor_pwm(motorLinks, 0);
142     servo_arc(2,-90);
143     turnRight();
144 }
145 
146 void testOutputRightSensor(){
147     lcd_cls();
148     lcd_puts("RightS: ");
149     lcd_ubyte(analog(optoSensorRechts));
150     sleep(700);
151 }
152 
```

```
153 void testOutputLeftSensor(){
154    lcd_cls();
155    lcd_puts("LeftS: ");
156    lcd_ubyte(analog(optoSensorLinks));
157    sleep(700);
158 }
159 
160 void Wegfindung(){
161    for (x = 0; x <= 10; x++)
162    {
163       for (y = 0; y<7; y++)
164       {
165          if (Muster1D[i] == 'x')
166             Muster2D[x][y] = 70;
167          else if (Muster1D[i] == '.')
168             Muster2D[x][y] = 71;
169          else if (Muster1D[i] == 'F')
170             Muster2D[x][y] = 72;
171          else if (Muster1D[i] == '#')
172             Muster2D[x][y] = 73;
173          i++;
174       }
175    }
176    if(dif()==1)
177    Muster2D[9][1] = 1;
178    else
179    Muster2D[9][5] = 1;
180    for (i = 0; i<70; i++)
181    {
182       for (x = 1; x<10; x++)
183       {
184          for (y = 1; y<6; y++)
185          {
186             if (Muster2D[x][y] == 71)
187             {
188                if (Muster2D[x + 1][y] == i)
189                   Muster2D[x][y] = i + 1;
190                else if (Muster2D[x][y + 1] == i)
191                   Muster2D[x][y] = i + 1;
192                else if (Muster2D[x - 1][y] == i)
193                   Muster2D[x][y] = i + 1;
194                else if (Muster2D[x][y - 1] == i)
195                   Muster2D[x][y] = i + 1;
196             }
197          }
198       }
```

```
199    }
200    k = 0;
201    //Für Außen
202    for (x = 1; x<9; x++)
203    {
204       for (y = 1; y < 6; y++)
205       {
206          if (Muster2D[x][y] < i&&Muster2D[x][y] > 0)
207          {
208             if (Muster2D[x + 1][y] == 72)
209             {
210                xs = x; ys = y; i = Muster2D[x][y];
211                hweg[k] = 'S';
212             }
213             else if (Muster2D[x][y + 1] == 72)
214             {
215                xs = x; ys = y; i = Muster2D[x][y];
216                hweg[k] = 'O';
217             }
218             else if (Muster2D[x - 1][y] == 72)
219             {
220                xs = x; ys = y; i = Muster2D[x][y];
221                hweg[k] = 'N';
222             }
223             else if (Muster2D[x][y - 1] == 72)
224             {
225                xs = x; ys = y; i = Muster2D[x][y];
226                hweg[k] = 'W';
227             }
228          }
229       }
230    }
231    k++;
232    for (; i >= 0; i--){
233       if (Muster2D[xs - 1][ys] == i - 1){
234          xs--;
235          hweg[k] = 'S';
236          k++;
237       }
238       else if (Muster2D[xs + 1][ys] == i - 1){
239          xs++;
240          hweg[k] = 'N';
241          k++;
242       }
243       else if (Muster2D[xs][ys-1] == i - 1){
244          ys--;
```

```
245         hweg[k] = 'O';
246         k++;
247      }
248      else if (Muster2D[xs][ys+1] == i - 1){
249         ys++;
250         hweg[k] = 'W';
251         k++;
252      }
253   }
254   hweg[k] = 0;
255   x = 0;
256   y = 2;
257   while (k>0){
258      k--;
259      if (hweg[k] == richtung[y]){
260         rweg[x] = 'G';
261      }
262      else if (hweg[k] == richtung[y+1]){
263         rweg[x] = 'R';
264         y++;
265         if (y == 5)
266            y = 1;
267      }
268      else if (hweg[k] == richtung[y - 1]){
269         rweg[x] = 'L';
270         y--;
271         if (y == 0)
272            y = 4;
273      }
274      x++;
275   }
276   rweg[x] = 0;
277 }
278
279 void AksenMain(void){
280   start();
281   if(count==3)
282      break;
283   {
284   if(rweg[i]==71)
285   {
286   geradeaus();
287   i++;
288   }
289   else if(rweg[i]==82){
290   turnRight();
```

```
291     i++;
292     }
293     else if(rweg[i]=='U'){
294     umdrehen();
295     i++;
296     }
297     else if(rweg[i]==76){
298     turnLeft();
299     i++;
300     }
301     else if(rweg[i]==76){
302     abliefern();
303     }
304     else if(rweg[i]==48){
305     break;
306     }
307     }
308     // Die folgende Endlosschleife "erzeugt" das Programmende
309     while(true)
310 }
```

8 Literatur

[BHS07] Boersch, Ingo ; Heinsohn, Jochen ; Socher, Rolf: *Wissensverarbeitung: Eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure*. Elsevier, Spektrum Akad. Verlag, 2007

[Bri19] Brilliant.org: *A* Search*. Online. `https://brilliant.org/wiki/a-star-search/`. Version: Januar 2019. – Besucht am: 17.01.2019

[Fac05] Fachhochschule Brandenburg, Fachbereich Informatik und Medien (Hrsg.): *Nutzerhandbuch AKSEN-Board*. Magdeburger Str. 50, 14770 Brandenburg an der Havel: Fachhochschule Brandenburg, Fachbereich Informatik und Medien, 2005. `http://ots.th-brandenburg.de/downloads/aksen/handbuch.pdf`. – Besucht am: 17.01.2019

[HB08] Harabor, Daniel ; Botea, Adi: Hierarchical path planning for multi-size agents in heterogeneous environments. In: *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium on* IEEE, 2008, S. 258–265

[JI18] Jochen, Heinsohn ; Ingo, Boersch: *AMS-Projekt im WS 2017/18 Pizzabote*. Online, Oktober 2018

[Kar15] Karllebolla: *Breitensuche*. Online. `https://github.com/antbots/AntBot/wiki/Breitensuche`. Version: März 2015. – Besucht am: 17.01.2019

[Pod13] Podhraski, Tomislav: *How to Speed Up A* Pathfinding With the Jump Point Search Algorithm*. Online. `https://gamedevelopment.tutsplus.com/tutorials/how-to-speed-up-a-pathfinding-with-the-jump-point-search-algorithm--gamedev-5818`. Version: März 2013. – Besucht am 17.01.2019