Fakultät Angewandte Informatik
Studiengang Bachelor Angewandte Informatik

Labyrinth Roboter

Legoroboter Programmierung in NXC

Studienarbeit im Fach:
Wahlpflichtfach Projekt
(4. Semester)

Vorgelegt von:
Felix Reithmaier
Thomas
Dominik

Prüfer:
—

Zweitprüfer:
—

Am: 06.07.2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Abbildung 1: Wände des Labyrinthes

1 Einleitung

Im Rahmen des Studiums an der Technischen Hochschule Deggendorf müssen wir im 4. Semester ein Projekt belegen, welches 4 Semesterwochenstunden beansprucht. Wir haben uns hierbei für eine Aufgabe mit dem Lego Mindstorms Roboter, welcher ein Labyrinth durchfahren muss, entschieden.

1.1 Aufgabenstellung

Folgende Aufgabenstellung wurde von Prof. Jüttner und Herr Fischer vorgegeben:

> Programmierung zweier Legoroboter, bei denen einer ein Labyrinth erkundet, einen Weg durch das Labyrinth findet, die Strecke optimiert und die optimierte Strecken an einen zweiten Roboter per Bluetooth übermittelt. Der zweite Roboter fährt auf der optimierten Strecke durch das Labyrinth.

Herr Fischer gab uns zu Beginn des Projekt eine Einführung zur NXC (Not eXactly C) Software und wie diese zu installieren ist. Ebenso wurde auf die Dokumentation von NXC [1] hingewiesen, welche uns jedoch im Verlauf des Projekts noch einige Probleme bereiten wird. Uns wurde Abbildung 1 gezeigt, an unterschiedlichen Stellen platziert werden.

1.2 Eigene Interpretation der Aufgabenstellung

Die Aufgabenstellung lässt noch Interpretationsspielraum offen. Folgende Punkte werden von der Anforderung noch nicht genau definiert:

- Der Such-Algorithmus der verwendet werden soll
- Startposition der Roboter

- Mit welchen Sensoren soll der Roboter das Labyrinth erkunden

Als passenden Such-Algorithmus wählten wir die Rechte-Hand-Methode. Dieser folgt der rechten Wand, bis der Ausgang erreicht wird.

Die Startposition ist bei uns frei wählbar, da der Eingang des Labyrinthes veränderbar ist. Der Nutzer soll zu Beginn des Programms eine Startposition eingeben können, damit der Roboter weiß, auf welcher Position er sich befindet.

Bei der Wahl der Sensoren sind wir durch durch die vorhandenen vier Sensor-Steckplätze des NXT eingeschränkt. Um Wände erkennen zu können, nutzen wir Ultraschallsensoren.

2 Methodik

Im Folgenden wird der Aufbau der beiden Roboter und die Vorgehensweiße erklärt.

2.1 Aufbau

Abbildung 2: Aufbau von Roboter 1

Abbildung 3: Aufbau von Roboter 2

Beide zu verwendende Roboter sind auf Grundlage der offiziellen Lego NXT Anleitung [2] aufgebaut. Diese eignen sich optimal um das Fahren der Roboter zu ermöglichen. Besonders hilfreich ist das kleine Stützrad, um enge Kurvenradien zu ermöglichen, was im Labyrinth hilfreich ist.

Bei Roboter 1 wird der Standardaufbau leicht verändert, um mehrere weitere Bauteile anbringen zu können. Hier befinden sich, wie in Abbildung 2 gezeigt; links, rechts und vorne am Roboter Ultraschallsensoren. Diese messen den Zentimeterabstand 0 bis 255 vom jeweiligen Sensor bis zur Labyrinthwand.

Da Roboter 2 keine Sensoren benötigt, wird der Standardaufbau, wie in Abbildung 3 zu sehen, nicht verändert.

2.2 Logik

Im Weiteren wird die Logik genauer erklärt. Dabei wird gezeigt, wie ein Weg im Labyrinth gefunden wird und wie der Algorithmus im Roboter umgesetzt wird.

2.2.1 Lösungsalgorithmus

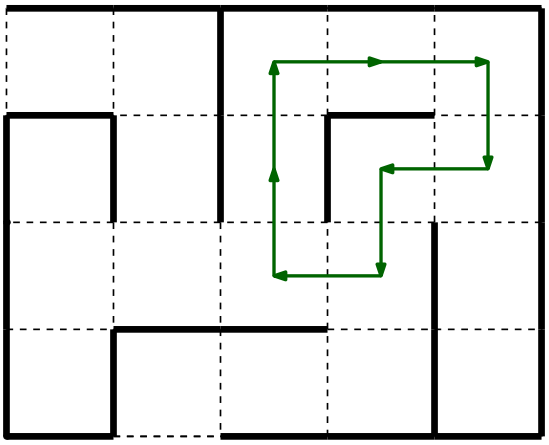


Abbildung 4: Rechte-Hand-Methode - Kreisproblem

Wie in der Einleitung bereits erwähnt, wird als Algorithmus die Rechte-Hand-Methode verwendet. Diese setzt voraus, dass das Labyrinth einen Eingang und einen Ausgang hat. Bei dieser Methode erkennt der Roboter die Wände um ihn und folgt der Wand, die sich rechts aus Sicht des Roboters befindet. Der Algorithmus kann jedoch versagen, wenn man, wie in Abbildung 4 zu sehen, im inneren des Labyrinthes startet, und sich dort logisch ein ”Kreis”befindet.

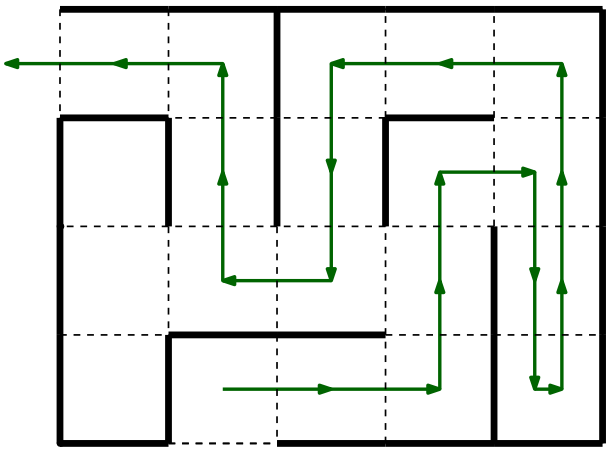
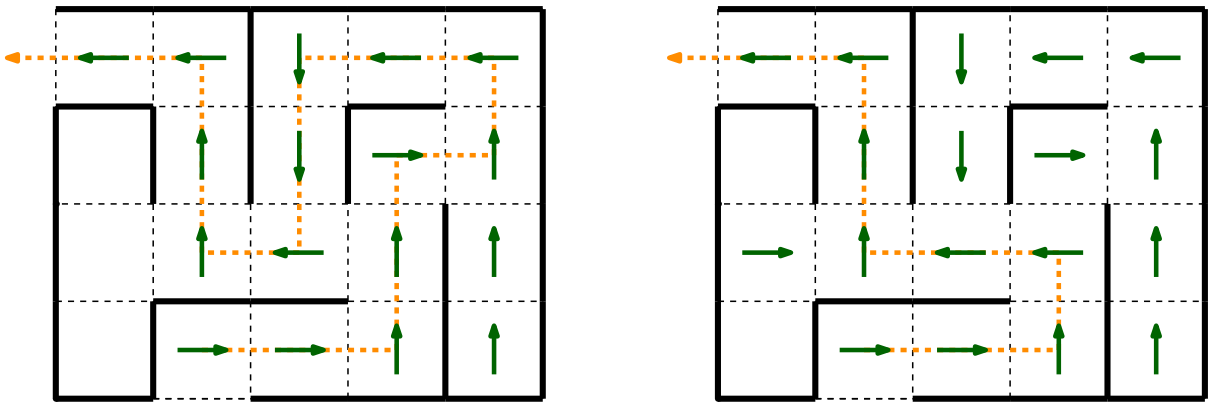


Abbildung 5: Rechte-Hand-Methode - Startposition außen

Um dieses Problem zu vermeiden startet unser Roboter an einer Randposition, da die Außenwand zusammenhängend ist. So kann keine Situation entstehen, bei der der Roboter in einer Endlosschleife gefangen ist. Ein Beispiel wird in Abbildung 5 gezeigt.



(a) ohne Optimierung (b) mit Optimierung

Abbildung 6: Finale Rechte-Hand-Methode

Auch wenn der Algorithmus bereits immer eine Lösung findet, kann dieser noch optimiert werden. Wie in der Abbildung 5 rechts unten zu sehen, gibt es Positionen, die mehrmals durchfahren werden. Um dies zu vermeiden speichert der Roboter Pfeile im Feld, die die Richtung des letzten Durchquerens darstellen. (siehe Abbildung 6)

Zusätzlich werden bei unserem Algorithmus manche Wege optimiert. Hierzu prüft der Roboter bei offenen Wänden, ob ein Pfeil im Nachbarfeld zu ihm ”hingedreht”werden kann. Dies in der Abbildung 6b zu sehen.

2.2.2 Roboterlogik

Algorithmus 1: Roboterlogik

$Position \leftarrow Startposition$ {Startposition wird vom Nutzer eingegeben}
while $Position$ ist in $Feld$ **do**
  Drehen: nach Entscheidungstabelle
  $Feld[Position] \leftarrow aktuelleDrehrichtung$
  Fahren: vorwärts
  $Position \leftarrow neuePosition$
end while

Um diesen Algorithmus als Roboter umsetzen zu können, benutzen wir eine Schleife, die einen Code so lange wiederholt, bis der Roboter einen Ausgang gefunden hat. Die Startposition muss vom Nutzer eingegeben werden und wird dann als aktuelle Position gesetzt. Der Algorithmus 1 beschreift diese Vorgehensweiße. Es wird auch klar, dass der Roboter einen Ausgang gefunden hat, falls sich seine eigene Position sich nicht mehr im Feld befindet.

Um zu wissen, in welche Richtung sich der Roboter drehen muss, nutzen wir die Ultraschallsensoren. Diese messen den Abstand zu den Wänden. Falls diese eine bestimmte Mindestdistanz betragen, kann davon ausgegangen werden dass sich in diese Richtung keine Wand befindet. Die jeweilige Drehrichtung kann aus der Tabelle 1 entnommen werden.

Ultraschallsensor			
links	**vorne**	**rechts**	**Aktion**
-	-	keine Wand	nach Rechts drehen
-	keine Wand	Wand	Keine Drehung
keine Wand	Wand	Wand	nach Links drehen
Wand	Wand	Wand	2x nach Rechts drehen

Tabelle 1: Entscheidungstabelle

Algorithmus 2: umliegende Pfeile drehen

if Keine Wand bei $Sensor_{rechts}$ **then**
  Pfeil rechts vom Roboter setzen {zeigt zum Roboter}
end if
if Keine Wand bei $Sensor_{vorne}$ **then**
  Pfeil vorm Roboter setzen {zeigt zum Roboter}
end if
if Keine Wand bei $Sensor_{links}$ **then**
  Pfeil links vom Roboter setzen {zeigt zum Roboter}
end if

Zusätzlich wird der Algorithmus 2 verwendet, um den Weg wie in Abbildung 6b zu optimieren. Dabei wird an jeder neuen Position zuerst ermittelt, ob sich Wände um den Roboter befinden. Ist dies nicht der Fall, so setzt der Roboter einen Pfeil am jeweiligen Nachbarfeld, der zum Roboter zeigt.

3 Ergebnis

Nun wird ein Test durchgeführt. Das Labyrinth wird so aufgebaut, wie es in den Beispielen aus Unterunterabschnitt 2.2.1 gezeigt wurde. Der finale Aufbau ist in Abbildung 7 zu sehen.

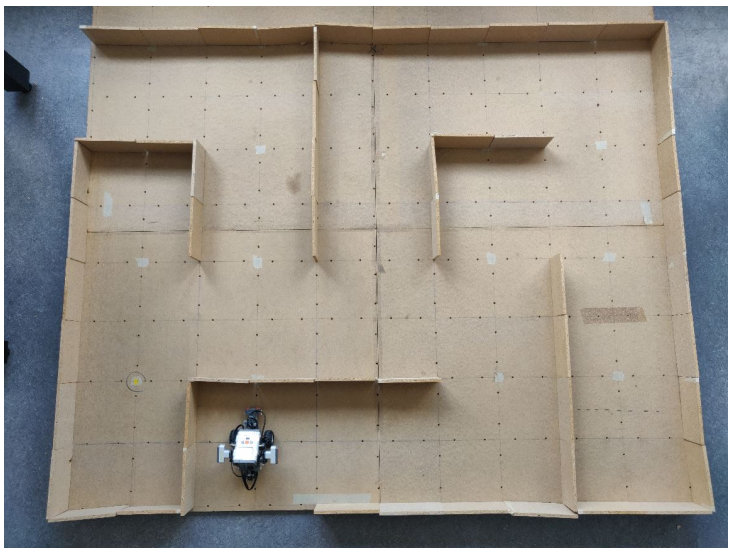


Abbildung 7: Labyrinth Beispielaufbau Draufsicht

Nach Durchlaufen des ersten Programms wird überprüft, ob dieser alle Felder richtig abspeichert. Alle Ergebnisse werden vor dem Übertragen per Bluetooth am Display ausgegeben. Das Ausgegebene Feld-Array wird in Tabelle 2 gezeigt.

4	4	3	4	4
0	1	3	2	1
2	1	4	4	1
0	2	2	1	1

Tabelle 2: Richtungen im Array

Richtung	Zahl im Feld
Nichts	0
Oben	1
Rechts	2
Unten	3
Links	4
Code Ende	5

Tabelle 3: Richtungen

Nun kann mittels der Tabelle 3 der abgespeicherte Weg in identifiziert werden. Achtet man in Tabelle 2 auf die orange eingefärbte Startposition und auf die blau eingefärbten Felder, so wird klar, dass der Roboter 1 die Richtigen korrekt ermittelt hat.

Somit wird anschließend der Weg im Programm virtuell durchlaufen, um die zu übergebende Zeichenkette ermitteln zu können. Diese endet mit der Zahl 5, damit der zweite Roboter weiß, dass er am Ende angekommen ist. Folgendes überträgt der Roboter an den zweiten Roboter:

”2214411445”

Der zweite Roboter muss dann manuell auf der Startposition vom ersten Roboter platziert werden. Anschließend fährt dieser die Richtungen der Zeichenkette ab. Der optimierte Weg ist dann auch ersichtlich kürzer als der des ersten Roboters.

4 Fazit

Im Schluss soll noch auf Probleme während des Arbeitens und auf Verbesserungsmöglichkeiten eingegangen werden.

4.1 Probleme

Probleme ergaben sich beim Programmieren des Roboters. Nicht nur die Mechanik des Roboters, sondern auch die Programmiersprache NXC hat uns oft Sorgen bereitet

4.1.1 90-Grad Drehungen

Im Programmcode sind einige Funktionen gegeben, die eine Drehung des Roboters ermöglichen. Jedoch kann keine dieser sicherstellen, dass der Roboter genau 90-Grad Kurven fahren kann. Auch wenn die Abweichung bei den Kurven nur sehr minimal ist, summieren sich die Fehler und beim Durchfahren des Labyrinths muss die Drehrichtung des Roboters immer wieder manuell angepasst werden.

4.1.2 Double Array

Ein weiteres großes Problem trat beim Speichern vom Double-Array auf. Hier gab der Roboter immer seltsame Werte zurück, wenn man auf diese zugreifen wollte. Letztendlich mussten wir das zweidimensionales Array in einem eindimensionalen Arrays speichern, da ein zweidimensionales nicht funktionierte. Auf den Fehler wurden wir erst durch eine Eintrag im Stackoverflow Forum aufmerksam. [3] Hier schrieb ein Nutzer, dass zweidimensionale Arrays in NXC nicht existieren. Was jedoch sehr seltsam ist, da diese laut offizieller Dokumentation [1] funktionieren müssten.

4.2 Verbesserungsmöglichkeiten

Die vorher erwähnten Probleme bei den 90-Grad Drehungen könnten verbessert werden. Eine Möglichkeit hierbei wäre das Einbauen eines Gyrosensors. Dabei könnte man während des Drehen des Roboters kontrollieren, ob dieser die Drehung perfekt ausgeführt hat, und falls nicht, diese anpassen.

Literatur

[1] Danny Benedettelli. *NXC Documentation*. Accessed: 1-6-2022. URL: http://bricxcc.sourceforge.net/nbc/nxcdoc/NXCGuide.pdf.

[2] LEGO. *NXT-Bauanleitng*. Accessed: 5-6-2022. URL: https://www.lego.com/cdn/product-assets/product.bi.core.pdf/4556207.pdf.

[3] NsJn. *NXC StrToNum always returns 0*. Accessed: 2-7-2022. URL: https://stackoverflow.com/questions/25733616/nxc-strtonum-always-returns-0.

Anhang A Roboterlogik 1 Code

Listing 1: Roboter 1 - Weg finden und per Bluetooth übertragen

```
1                       /*
2       Code fur den ersten Roboter, der das Labyrinth erkundet und dann
            einen Weg uber Bluetooth an den zweiten Roboter schickt.
3   */
4
5   //Anschlusse
6   #define MOTOREN OUT_BC
7
8   #define SENSOR_RECHTS IN_4
9   #define SENSOR_VORNE IN_1
10  #define SENSOR_LINKS IN_3
11
12  //Settings
13  #define MINDESTABSTAND 22
14  #define POWER 40
15  #define DISTANCE 360*2.2
16  #define BLUETOOTH_CONNECTION 1
17  #define OUTBOX 5
18  //Defines fur Lesbarkeit.
19  #define MAXIMAL_SIZE 9
20  #define NICHTS 0
21  #define OBEN 1
22  #define RECHTS 2
23  #define UNTEN 3
24  #define LINKS 4
25  #define EndOfLine 5
26  #define RECHTS_DREHEN 100
27  #define LINKS_DREHEN -100
28
29  //Vars
30  float sensorRechts;
31  float sensorVorne;
32  float sensorLinks;
33  string befehlskette;
34  int feld[MAXIMAL_SIZE*MAXIMAL_SIZE];
35  int richtung = OBEN; //Startrichtung
36  int posY = 0; //Momentane Y-Position vom Roboter.
37  int posX; //Momentane X-Position vom Roboter.
38  int startpos;
39  int SIZE_X;
40  int SIZE_Y;
41
42
43  //Fuktion Headers.
44  void printFeld();
45  int AbfrageNachStart();
46  void Displayausgabe();
47  void Setup();
48  void Drehen(int drehwert);
49  void DrehenMitFeld(int drehwert);
50  int PfeilDrehen(int nachRechts, int anzahlDerUmdrehungen);
51  void Fahren();
52  int FahrenMitFeld();
```

```
53  void VirtuellenPfeilAnwenden(int virtuellerPfeil);
54  void Zyklus();
55  void BefehlsketteErstellen();
56  void BluetoothCheck(int connection);
57  void ErgbenisUbertragen();
58
59
60  int AbfrageNachStart() {
61      //Abfrage fuer Feld Groesse (SIZE_X, SIZE_Y)
62
63      ResetScreen();
64      TextOut(16, LCD_LINE1 , "Feld X Groesse", false);
65      TextOut(16, LCD_LINE2 , "auswaehlen", false);
66
67      SIZE_X = 5;
68      while(true){
69          NumOut(48, LCD_LINE5, SIZE_X);
70
71          //INPUT-Eingabe
72          if (ButtonPressed(BTNRIGHT,false) && SIZE_X < MAXIMAL_SIZE){
73              SIZE_X += 1;
74              Wait(400);
75          }else if (ButtonPressed(BTNLEFT, false) && SIZE_X > 1){
76              SIZE_X -= 1;
77              Wait(400);
78          }else if (ButtonPressed(BTNCENTER, false)){
79              break;
80          }
81      }
82      Wait(400);
83
84      ResetScreen();
85      TextOut(16, LCD_LINE1 , "Feld Y Groesse", false);
86      TextOut(16, LCD_LINE2 , "auswaehlen", false);
87
88      SIZE_Y = 4;
89      while(true){
90          NumOut(48, LCD_LINE5, SIZE_Y);
91
92          //INPUT-Eingabe
93          if (ButtonPressed(BTNRIGHT,false) && SIZE_Y < MAXIMAL_SIZE){
94              SIZE_Y += 1;
95              Wait(400);
96          }else if (ButtonPressed(BTNLEFT, false) && SIZE_Y > 1){
97              SIZE_Y -= 1;
98              Wait(400);
99          }else if (ButtonPressed(BTNCENTER, false)){
100             break;
101         }
102     }
103     Wait(400);
104
105     //Abfrage fuer den Startpunkt (0-4)
106
107     ResetScreen();
108     int startfeld = 0;
109     TextOut(16, LCD_LINE1 , "Startpunkt (X-Seite , Y=0)", false);
```

```
110     TextOut(16, LCD_LINE2 , "auswaehlen", false);
111
112     while(true){
113         NumOut(48, LCD_LINE5, startfeld);
114
115         //INPUT-Eingabe
116         if (ButtonPressed(BTNRIGHT,false) && startfeld < SIZE_X-1){
117             startfeld += 1;
118             Wait(400);
119         }else if (ButtonPressed(BTNLEFT, false) && startfeld > 0){
120             startfeld -= 1;
121             Wait(400);
122         }else if (ButtonPressed(BTNCENTER, false)){
123             break;
124         }
125     }
126
127     //Ausgabe Bestatigung und CountDown.
128     TextOut(16, LCD_LINE2, "Vielen Dank", true);
129     TextOut(16, LCD_LINE3, "Starte in..", false);
130     for(int i=3; i>=0; i--){
131             NumOut(48, LCD_LINE5 , i);
132             if(i > 0){
133                 PlayTone(440, 550);
134                 Wait(1000);
135             }else{
136                 PlayTone(880, 1000);
137                 Wait(1100);
138             }
139     }
140
141     Wait(1000);
142     ResetScreen();
143     return startfeld;
144 }
145
146 void Displayausgabe() {
147     ResetScreen();
148
149     TextOut(1, LCD_LINE1 , "Sensor Links:", false);
150     TextOut(1, LCD_LINE3 , "Sensor Vorne:", false);
151     TextOut(1, LCD_LINE5 , "Sensor Rechts:", false);
152     TextOut(1, LCD_LINE7 , "Position:", false);
153
154     NumOut(1, LCD_LINE2 , SensorUS(SENSOR_LINKS));
155     NumOut(1, LCD_LINE4 , SensorUS(SENSOR_VORNE));
156     NumOut(1, LCD_LINE6 , SensorUS(SENSOR_RECHTS));
157     NumOut(1, LCD_LINE8 , posY);
158     NumOut(10, LCD_LINE8 , posX);
159
160     NumOut(40, LCD_LINE8 , richtung);
161
162 }
163
164 void Setup() {
165     //Settup fur drehen und Kordinatennetz.
166
```

```
167     //Sensoren setzen.
168     SetSensorLowspeed(SENSOR_VORNE);
169     SetSensorLowspeed(SENSOR_RECHTS);
170     SetSensorLowspeed(SENSOR_LINKS);
171
172     //innit Kordinatennetz, Abfrage nach Startposition.
173     for (int x = 0; x < SIZE_X; x++){
174         for (int y = 0; y < SIZE_Y; y++) {
175             feld[x+SIZE_X*y] = NICHTS;
176         }
177     }
178     posX = AbfrageNachStart();
179     startpos = posX;
180 }
181
182 void Drehen(int drehwert) {
183     //90 Grad Drehung nach rechts durchfuhren.
184
185     //Drehen
186     RotateMotorEx(MOTOREN, POWER, 175, drehwert, true, true);
187 }
188
189 void DrehenMitFeld(int drehwert) {
190     //Drehung ausfuhren und Richtung anpassen.
191
192     Drehen(drehwert);
193
194     //Richtung anpassen.
195     richtung = PfeilDrehen((drehwert==RECHTS_DREHEN), 1);
196 }
197
198
199
200 void Fahren() {
201     //Ein Tile nach vorne fahren.
202
203     //Vorwartsfahren
204     RotateMotorEx(MOTOREN, POWER, DISTANCE, 0, true, true);
205 }
206
207 int FahrenMitFeld() {
208     //Nach vorne Fahren, neue Position merken und in Feld eintragen.
209
210     //In Feld eintragen.
211     feld[posX+posY*SIZE_X] = richtung;
212
213     Fahren();
214
215     //Position Akualisieren.
216     switch (richtung) {
217         case OBEN:
218             posY++;
219         break;
220         case RECHTS:
221             posX++;
222         break;
223         case LINKS:
```

```
224             posX--;
225         break;
226         case UNTEN:
227             posY--;
228         break;
229         default:
230         break;
231     }
232 }
233
234 void VirtuellenPfeilAnwenden(int virtuellerPfeil) {
235     //Looperkennung muss erreichbare felder Markieren, diese
        funktion ist fur die Markierung da.
236
237     //In die entgegengesetze Richtung von virtueller Pfeil gehen und
        dann in dieser Zelle einzeichnen.
238     int pfeilPosX = posX;
239     int pfeilPosY = posY;
240
241     switch (virtuellerPfeil) {
242         case OBEN:
243             pfeilPosY--;
244         break;
245         case LINKS:
246              pfeilPosX++;
247         break;
248         case UNTEN:
249              pfeilPosY++;
250         break;
251         case RECHTS:
252              pfeilPosX--;
253         break;
254         default:
255     }
256
257     if(!(pfeilPosX > (SIZE_X-1) || pfeilPosX < 0 || pfeilPosY > (
        SIZE_Y-1) || pfeilPosY < 0)){
258           feld[pfeilPosX+(pfeilPosY*SIZE_X)] = virtuellerPfeil;
259     }
260 }
261
262
263 void Zyklus() {
264     //Zyklus fur ein Tile(Logik).
265
266     //Sensoren auslesen.
267     float sensorRechts = SensorUS(SENSOR_RECHTS);
268     float sensorVorne = SensorUS(SENSOR_VORNE);
269     float sensorLinks = SensorUS(SENSOR_LINKS);
270
271
272     //Freie benachbarte Zellen markieren,
273     //Wenn frei, entgegengesetzten Peil erstellen und dann eintragen
        .
274     if (sensorRechts > MINDESTABSTAND) {
275         //Rechts markieren.
```

```
276             VirtuellenPfeilAnwenden(PfeilDrehen(0, 1)); //PfeilDrehen(
                    int nachRechts, int anzahlDerUmdrehungen)
277         }
278         if (sensorLinks > MINDESTABSTAND) {
279             //Links markieren.
280             VirtuellenPfeilAnwenden(PfeilDrehen(1, 1));
281         }
282         if (sensorVorne > MINDESTABSTAND) {
283             //Vorne markieren.
284             VirtuellenPfeilAnwenden(PfeilDrehen(1, 2));
285         }
286 
287         printFeld();
288 
289         //Fahren und speichern.
290         if (sensorRechts > MINDESTABSTAND) {
291             //Rechts abbiegen.
292             DrehenMitFeld(RECHTS_DREHEN);
293             Wait(400);
294             FahrenMitFeld();
295         } else if (sensorVorne > MINDESTABSTAND) {
296             //Gerade aus fahren.
297             FahrenMitFeld();
298         } else if (sensorLinks > MINDESTABSTAND) {
299             //Links abbiegen.
300             DrehenMitFeld(LINKS_DREHEN);
301             Wait(400);
302             FahrenMitFeld();
303         } else {
304             //Umdrehen
305             DrehenMitFeld(RECHTS_DREHEN);
306             DrehenMitFeld(RECHTS_DREHEN);
307             Wait(400);
308             FahrenMitFeld();
309         }
310 
311         printFeld();
312     }
313 
314     int PfeilDrehen(int nachRechts, int anzahlDerUmdrehungen) {
315         //Richtung anpassen.
316         int variable = richtung;
317         for(int umdrehung = 0; umdrehung < anzahlDerUmdrehungen;
                umdrehung++) {
318             if (nachRechts) {
319                 if (variable < LINKS) {
320                     variable++;
321                 } else {
322                     variable = OBEN;
323                 }
324             } else {
325                 if (variable > OBEN) {
326                     variable--;
327                 } else {
328                     variable = LINKS;
329                 }
330             }
```

```
331     }
332
333     return variable;
334 }
335
336 void BefehlsketteErstellen() {
337     //Befehlskette aus Feld erstellen fur zweiten Roboter.
338
339     //Tempurare Vars.
340     int virtualPosX = startpos;
341     int virtualPosY = 0;
342
343     while (!(virtualPosX > (SIZE_X-1) || virtualPosX < 0 ||
        virtualPosY > (SIZE_Y-1) || virtualPosY < 0)) {
344         //Richtung auslesen.
345
346         befehlskette += NumToStr(feld[virtualPosX+SIZE_X*virtualPosY
            ]);
347
348         ResetScreen();
349         TextOut(1, LCD_LINE1, "Position x, y:");
350         NumOut(1, LCD_LINE2, virtualPosX);
351         NumOut(20, LCD_LINE2, virtualPosY);
352
353         Wait(500);
354
355         //Position Akualisieren.
356         switch (feld[virtualPosX+SIZE_X*virtualPosY]) {
357             case OBEN:
358                 virtualPosY++;
359             break;
360             case RECHTS:
361                 virtualPosX++;
362             break;
363             case LINKS:
364                 virtualPosX--;
365             break;
366             case UNTEN:
367                 virtualPosY--;
368             break;
369             default:
370             break;
371         }
372     }
373
374     befehlskette += NumToStr(EndOfLine);
375 }
376
377 void BluetoothCheck(int connection) {
378     //Testet die Bluetoothverbingung.
379     if (!BluetoothStatus(connection) == NO_ERR) {
380         TextOut(5, LCD_LINE2, "Error");
381         Wait(1000);
382         Stop(true);
383     }
384 }
385
```

```
386 void ErgbenisUbertragen() {
387     //Erstellt die Befehlskette und ubertragt diese.
388 
389     printFeld();
390 
391     BefehlsketteErstellen();
392 
393     Wait(1000);
394 
395     ResetScreen();
396     TextOut(5, LCD_LINE2, befehlskette);
397     Wait(1000);
398 
399     BluetoothCheck(BLUETOOTH_CONNECTION);
400 
401     SendRemoteString(BLUETOOTH_CONNECTION, OUTBOX, befehlskette);
402     PlayTone(440, 550);
403     Wait(1000);
404 }
405 
406 void printFeld(){
407 
408     ResetScreen();
409 
410     for (int x = 0; x < SIZE_X; x++){
411         for (int y = 0; y < SIZE_Y; y++) {
412             NumOut(10*x, 10*y ,feld[x+SIZE_X*y]);
413         }
414     }
415 
416     Wait(400);
417 }
418 
419 task main() {
420     //Setup
421     Setup();
422 
423     //Weg finden.
424     while(!(posX > (SIZE_X-1) || posX < 0 || posY > (SIZE_Y-1) ||
        posY < 0)) {
425 
426         Zyklus();
427     }
428 
429     //Ubertragen
430     ErgbenisUbertragen();
431 }
```

Anhang B Roboterlogik 2 Code

Listing 2: Roboter 2 - Weg per Bluetooth erhalten und Labyrinth durchfahren

```
1  /*
2      Code fur den zweiten Roboter, der das Labyrinth mit den Daten
           vom ersten Roboter durchfahrt.
3  */
4
5  //Anschlusse
6  #define MOTOREN OUT_BC
7  //Settings
8  #define POWER 40
9  #define DISTANCE 360*2.2
10 #define INBOX 5
11 //Defines fur Lesbarkeit.
12 #define OBEN 1
13 #define RECHTS 2
14 #define UNTEN 3
15 #define LINKS 4
16 #define EndOfLine 5
17 #define RECHTS_DREHEN 100
18 #define LINKS_DREHEN -100
19 #define CONVERT_ASCII(x) (x-48)
20
21 //Vars
22 string befehlskette = "";
23 int richtung = OBEN; //Startrichtung
24
25 //Funktion Headers.
26 void Drehen(int drehwert);
27 void DrehenMitFeld(int drehwert);
28 int PfeilDrehen(int nachRechts, int anzahlDerUmdrehungen);
29 void Richtunganpassen(int sollRichtung);
30 void Fahren();
31 void BluetoothCheck(int connection);
32 void ErgbenisEmpfangen();
33
34
35 void Drehen(int drehwert) {
36     //90 Grad Drehung nach rechts durchfuhren.
37
38     //Drehen
39     RotateMotorEx(MOTOREN, POWER, 175, drehwert, true, true);
40 }
41
42 void DrehenMitFeld(int drehwert) {
43     //Drehung ausfuhren und Richtung anpassen.
44
45     Drehen(drehwert);
46     Wait(500);
47
48     //Richtung anpassen.
49     richtung = PfeilDrehen((drehwert==RECHTS_DREHEN), 1);
50 }
51
52 int PfeilDrehen(int nachRechts, int anzahlDerUmdrehungen) {
```

```
53      //Richtung anpassen.
54      int variable = richtung;
55      for(int umdrehung = 0; umdrehung < anzahlDerUmdrehungen;
           umdrehung++) {
56          if (nachRechts) {
57              if (variable < LINKS) {
58                  variable++;
59              } else {
60                  variable = OBEN;
61              }
62          } else {
63              if (variable > OBEN) {
64                  variable--;
65              } else {
66                  variable = LINKS;
67              }
68          }
69      }
70      return variable;
71  }
72
73  void Richtunganpassen(int sollRichtung) {
74      //Passt die Richtung an die angegebene Richtung an.
75
76      //Richtungen durchchecken mit virtellerPfeil und dann
           dementsprechend Drehen.
77      int virtuellerPfeil;
78      if (richtung == sollRichtung) {
79          //Gerade aus.
80          return;
81      }
82
83      if(PfeilDrehen(1, 1) == sollRichtung) {
84          //Nach rechts.
85          DrehenMitFeld(RECHTS_DREHEN);
86          return;
87      }
88
89
90      if(PfeilDrehen(1, 2) == sollRichtung) {
91          //Nach hinten.
92          DrehenMitFeld(RECHTS_DREHEN);
93          DrehenMitFeld(RECHTS_DREHEN);
94          return;
95      }
96
97      //Nach links.
98      DrehenMitFeld(LINKS_DREHEN);
99  }
100
101 void Fahren() {
102     //Ein Tile nach vorne fahren.
103
104     //Vorwartsfahren
105     RotateMotorEx(MOTOREN, POWER, DISTANCE, 0, true, true);
106 }
107
```

```
108 void BluetoothCheck(int connection) {
109     //Testet die Bluetoothverbindung.
110     if (!BluetoothStatus(connection) == NO_ERR) {
111         TextOut(5, LCD_LINE2, "Error");
112         Wait(1000);
113         Stop(true);
114     }
115 }
116
117 void ErgbenisEmpfangen() {
118     //Empfangt Befehlskette.
119     while(!strcmp(befehlskette, "")){
120         BluetoothCheck(0);
121         ReceiveRemoteString(INBOX, true, befehlskette);
122     }
123     ResetScreen();
124     TextOut(5, LCD_LINE2, befehlskette);
125     Wait(400);
126     PlayTone(880, 1000);
127     Wait(1100);
128 }
129
130 void Displayausgabe(int i) {
131     ResetScreen();
132
133     TextOut(10, LCD_LINE1 , "i:");
134     NumOut(10, LCD_LINE2 , i);
135
136     TextOut(10, LCD_LINE3 , "befehlskette[i]:");
137     NumOut(10, LCD_LINE4, CONVERT_ASCII(befehlskette[i]));
138
139     TextOut(10, LCD_LINE7 , "richtung:");
140     NumOut(10, LCD_LINE8 , richtung);
141
142 }
143
144 task main() {
145     ErgbenisEmpfangen();
146
147     //Befehlskette abfahren.
148     int i = 0;
149     while(CONVERT_ASCII(befehlskette[i]) != EndOfLine) {
150         Displayausgabe(i);
151
152         Richtunganpassen(CONVERT_ASCII(befehlskette[i]));
153         Fahren();
154         i++;
155     }
156     PlayTone(880, 1000);
157     Wait(1100);
158 }
```

Anhang C Arbeitsaufteilung

- Dominik Dillinger
  - Überlegen der Algorithmen
  - Entwickeln von neuen Code Konzepten
  - Codefunktionen:
    - DrehenMitFeld
    - PfeilDrehen
    - FahrenMitFeld
    - Zyklus
    - BefehlsketteErstellen
    - BluetoothCheck
    - ErgbenisUbertragen
- Thomas Hochgesand
  - Testen des Codes
  - Fehlersuche
  - Anpassung des Codes
  - Codefunktionen:
    - Setup
    - AbfrageNachStart
    - Drehen
    - Fahren
- Felix Reithmaier
  - Testen des Codes
  - Fehlersuche
  - Schreiben der Dokumentation
  - Codefunktionen:
    - printFeld
    - Displayausgabe
    - VirtuellenPfeilAnwenden