Fachbereich Informatik und Medien

AMS-Projekt im WS 2017/18 Pizzabote Dokumentation „Kabelbinder-Jack“

Vorgelegt von: Annick Michelle Sokeng
Dominik Schwamm
Jan Bremges

am 21.01.2018

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

Inhaltsverzeichnis

1 Aufgabenstellung

Die Aufgabe besteht darin, dass der Roboter einen Auftrag erhält, eine oder mehrere Pizzen auszuliefern. Das Streckennetz ist ein einfaches Gitter, in dem es allerdings zu Störungen und damit unbefahrbaren Kreuzungen kommen kann. Die aktuelle Karte der befahrbaren Wege wird dem Roboter kurz vor dem Start zur Verfügung gestellt.

1.1 Lieferadressen, Karte und Fahrauftrag

Das Streckennetz (Abb. 1 links) besteht aus Strecken, Kreuzungen und den Lieferadressen H1 bis H21. Einige Kreuzungen können gesperrt sein, wie bei F2 und F3 in Abb. 1. Die in der Pizzeria frisch gebackene Pizzen werden an den Startpunkten A bzw. B eingeladen. Da die Pizzen immer frisch sein sollen, darf nur einzeln transportiert werden. Daher muss der Roboter nach jeder Lieferung zum Startpunkt zurück fahren und neu beladen werden.

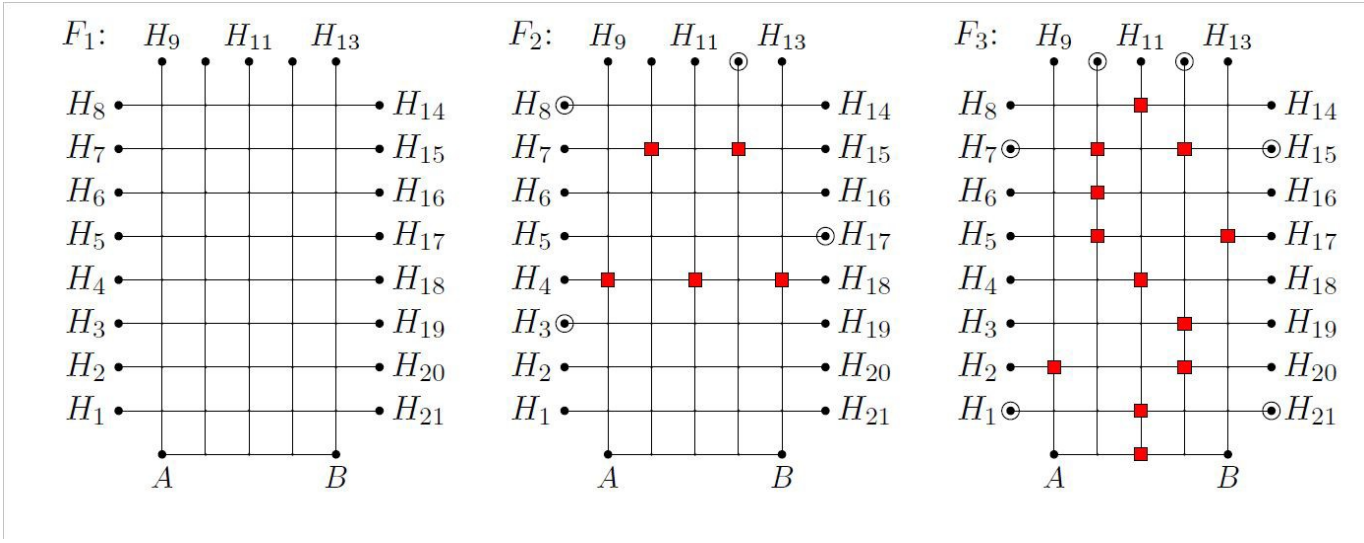

Abbildung 1: Fahraufträge: Fahrauftrag F1 - alle Kreuzungen sind befahrbar, keine Lieferung, Fahrauftrag F2 - fünf Kreuzungen sind gesperrt (rotes Rechteck), vier Lieferadressen warten (Kreis), F3 - zwölf Kreuzungen sind gesperrt, sechs Lieferungen warten.

2 Aufbau des Roboters

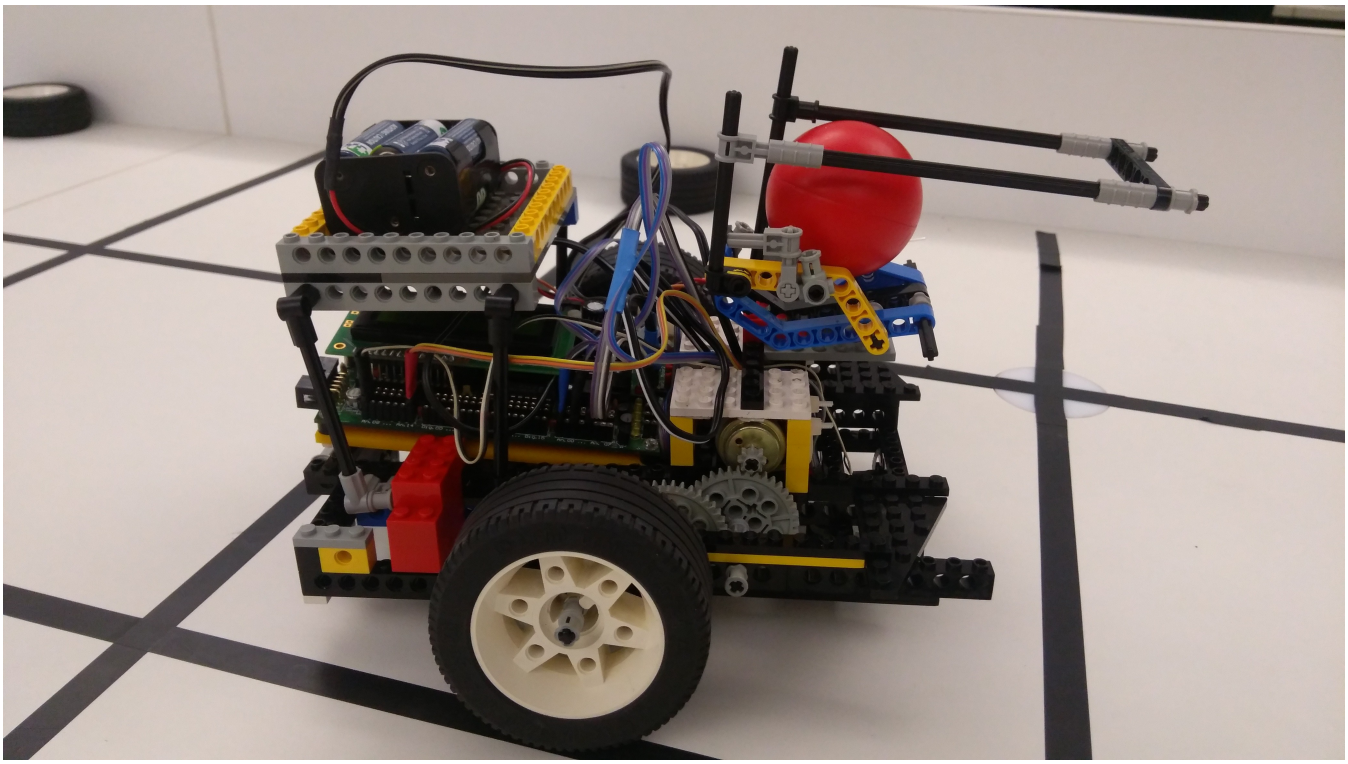

Abbildung 2: Unser „Kabelbinder-Jack“

2.1 Hardware

Der Roboter wurde mithilfe folgender Komponenten gebaut: Sensoren, Aktoren, Mikroprozessor und Lego. Aufgrund von erhöhtem Kabelbindereinsatz erhielt er im Laufe der Konstruktionsphase den Namen „Kabelbinder-Jack“. Kabelbinder-Jack besitzt auf der vorderen Seite jeweils einen Motor rechts und links. Jeder Motor ist mit einem 125er Getriebe aus Zahnrädern verbunden, welche die Kraft dann auf das Getriebe der jeweiligen Seite übertragen. Über dem linken Motor ist ein Servo-Motor befestigt welcher die Abwurfvorrichtung für den Ball hoch und runter fahren kann. An der Vorderseite befindet sich ein Taster, welcher aber nur beim Ziel ausgelöst wird, das heißt, selbst wenn der Taster während der Fahrt gedrückt wird, wird der Ball nicht abgeworfen. Hinter dem Taster befinden sich die drei Optokoppler, alle werden zum Linienfolgen und für die Erkennung der Kreuzungen verwendet. An der Unterseite befindet sich noch ein Lichtsensor, welcher den Roboter nur starten lässt, wenn das Startlicht an geht. Das Aksen Board befindet sich hinter den Motoren und ist das Herzelement des Roboters, da es alle Teile ansteuert. Über dem Board befindet sich noch eine Art Dach, um den Akku abzulegen. Dieser dient auch als Gewichtsausgleich für die schwere Vorderseite. Damit der Roboter nicht mit der hinteren Hälfte am Boden streift, ist noch ein kleines Rad an der Rückseite befestigt.

Besonders gut gelungen ist uns das Gesamtgewicht des Roboters, da er trotz 125er Getriebe noch ziemlich schnell fährt. Auch die Abwurfvorrichtung ist gut, da der Ball fast immer im Ziel landet und

dort auch liegen bleibt. Außerdem ist er sehr wendig da die Räder sich bei Kurven entgegengesetzt drehen.

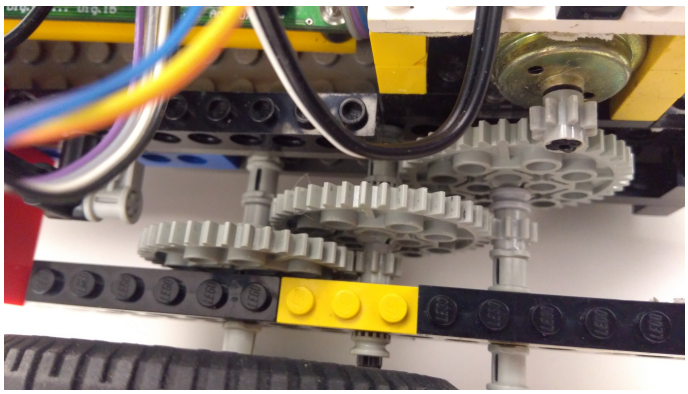

Abbildung 3: Das 125er Getriebe

2.1.1 Aktorik

Folgende Aktoren wurden bei dem Aufbau genutzt:

- 2 * Elektromotoren: die zwei Motoren setzen den Roboter über das Getriebe in Bewegung
- 1 * Servo-Motor

Wichtig bei dem Aufbau war das Gewicht des Roboters so gering wie möglich zu halten. Der Roboter verfügt auch über einen Greifer (aus Lego gebaut), der dazu dient die Pizzen an den Lieferadressen abzuladen.

2.1.2 Sensorik

Für die Sensorik wurden folgende Sensoren benutzt:

- 1 * Fotosensor (unter dem Roboter befestigt): dieser Sensor wurde benutzt, um das Startsignal der Startleuchten zu erkennen und so die Fahrt beginnen zu können.
- 3 * Optokoppler
- 1 * Schalter

Die Optokoppler bestehen aus einer Infrarot-LED, sowie einer infrarotempfindlichen Photodiode.

2.1.3 Dip-Schalter

Das AKSEN-Board verfügt über 4 Dip-Schalter wovon wir 2 genutzt haben um das richtige Startfeld einzustellen.

2.1.4 Lego

Für den Bau des Roboters wurde genug Lego zur Verfügung gestellt um sich frei zu entfalten. Mithilfe von Zahnrädern, Achsen und verschiedenen Reifen war eine gute Grundlage für den Roboterbau gegeben.

2.1.5 Akkumulatoren

Zur Verfügung standen NiMH-Mignon Akkupacks bzw. 5 * 1,2 V und 2500 mAh. Pro Team gab es zwei Stück, die Abends aufgeladen werden sollten.

2.1.6 Mikroprozessor-Board

Der Roboter wird durch ein AKSEN-Board gesteurt. Es handelt sich um ein 8-bit Mikroprozessor C515 von Siemens mit seriellen Schnittstellen, 16 analogen und 16 digitalen Sensorports, diverse Elektronik zur Steuerung von Motoren, Servos, LED, LCD...

2.2 Software

Die Software wurde mithilfe der Programmiersprache C entwickelt. Das Programm enthält viele Funktionen, welche jeweils eine bestimmte Teilaufgabe der Gesamtaufgabe erfüllen z.B. Linienfolgen, Kreuzungen erkennen sowie die Fahrstrecke planen.

Zuerst wird das vorgegebene Streckennetz eingelesen und mithilfe von planuebernehmen() in ein Integer-Array übertragen. Ein „x“ wird dabei zu „100“, ein „F“ zu einer „99“ und ein „.“ zu einer „0“. Mithilfe von starteinstellen() wird das Startfeld ausgelesen welches über die Dip-Schalter eingestellt wurde. KB-Jacks wichtigste Funktion ist fahrendlich() denn diese enthält nun alle anderen relevanten Funktionen zur Abwicklung seiner Fahraufträge. Mit planerstellen() wird jeder noch nicht belegten Kreuzung eine Zahl zugewiesen die ihrer Entfernung zum Zielfeld entspricht. Sind Kreuzungen nicht erreichbar da sie komplett versperrt sind oder die Zielkarte in der Mitte getrennt ist, bleiben diese Kreuzungen mit „0“ belegt. Anschließend erstellt routeplanen() eine Befehlskette für KB-Jack, die den Weg zum Ziel, seinen Abladealgorithmus sowie seinen Rückweg enthält. Danach wird mithilfe einer While-Schleife wiederholt kreuzung() aufgerufen. Erkennt KB-Jack hier eine Kreuzung, führt er den

nächsten Befehl seiner Befehlsliste aus:

„L“ → Links abbiegen
„R“ → Rechts abbiegen
„G“ → Geradeaus über Kreuzung fahren
„A“ → Links abbiegen und Pizza liefern, anschließend 90 Grad Drehung nach links
„D“ → Rechts abbiegen und Pizza liefern, anschließend 90 Grad Drehung nach rechts
„W“ → Geradeaus über Kreuzung fahren und Pizza liefern, anschließend 180 Grad Drehung
„S“ → 180 Grad Drehung und neu beladen lassen

Hat KB-Jack einen Auftrag ausgeführt und ist wieder am Startfeld angelangt werden alle Variablen auf ihren Startwert zurückgesetzt und er beginnt mit der Berechnung für den nächsten Auftrag. Hat er alle Aufträge erfüllt oder ist die Wettbewerbszeit von 2 Minuten abgelaufen schaltet er die Motoren ab und bleibt stehen.

3 Lösungswege

3.1 Bau des Roboters

Unsere erste Aufgabe bestand darin ein Getriebe zusammenzubauen und eine Ablage für das AKSEN-Board zu schaffen. Wir haben uns für ein 125er Getriebe entschieden und fortan versucht die wöchentlich geforderten Ziele zu erreichen. Nachdem der fahrbare Untersatz fertig war sollte der Roboter als nächstes einer schwarzen Linie folgen können. Dazu montierten wir an der Vorderseite 3 Optokoppler die dafür sorgen sollten dieses Ziel zu erreichen. Parallel dazu wurde auch begonnen zu programmieren und unsere erste Funktion linienfolgen() entstand. Als nächstes sollten Kurven gefahren werden und sobald dies möglich war sollte der Roboter so instruiert werden, dass er auf dem Streckennetz eine „8“ fährt. Auch dieses Ziel konnte nach einigen Versuchen was den Abbiegevorgang betrifft schnell implementiert werden. Für den Akku bauten wir aus Platz- und Gewichtsgründen eine Art Dach über dem AKSEN-Board, wo wir den Akku nun ablegen konnten. Nun folgte ein größerer Programmierteil denn es ging nun darum einen optimalen Weg zum Ziel zu finden und eine passende Befehlskette zu erstellen. Leider bietet die Ausführung auf dem AKSEN-Board selbst wenig Möglichkeiten Fehler zu finden. Diese Aufgabe nahm wohl den größten Teil der Zeit in Anspruch. Deswegen begannen wir parallel mit dem Implementieren der Abwurffunktion und dem Einbau unseres „Abwurfarmes“. Den dazu gebrauchten Servo-Motor setzten wir einfach auf den linken Motor damit der

Arm schön mittig am Roboter platziert werden konnte. Weiterhin montierten wir an vorderster Stelle einen Tast-Sensor der in Verbindung mit dem „Abwurfarm“ von nun an für die Pizzalieferung verantwortlich war. Der Lichtsensor an der Unterseite war genauso schnell implementiert und montiert wie die Funktion der Dip-Schalter zur Feststellung des Startfelds. Sorgen machte uns in den letzten Wochen nur die Fahrweise des Roboters, der bei jedem Start anders fuhr: mal schnell und gerade, dann wieder ruckelig und langsam. Durch einen Wechsel auf größere Reifen wurde es besser, durch die Stabilisierung der Radachsen für uns auch ausreichend. Nachdem der Roboter nun funktionierte, er fand jetzt seinen Hin- und Rückweg und lieferte die Pizza mit ca. 60% Sicherheit beim Kunden ab, benötigten wir vor dem Wettbewerb nur einen Zusatztermin um ihn komplett fertig zu stellen. Bei diesem Treffen implementierten wir noch die geforderte Anhaltefunktion nach 2 Minuten und optimierten unseren „Abwurfarm“ durch 2 längere Legostangen die dafür sorgten dass der Ball nun besser und sicherer in den Reifen gelegt wurde.

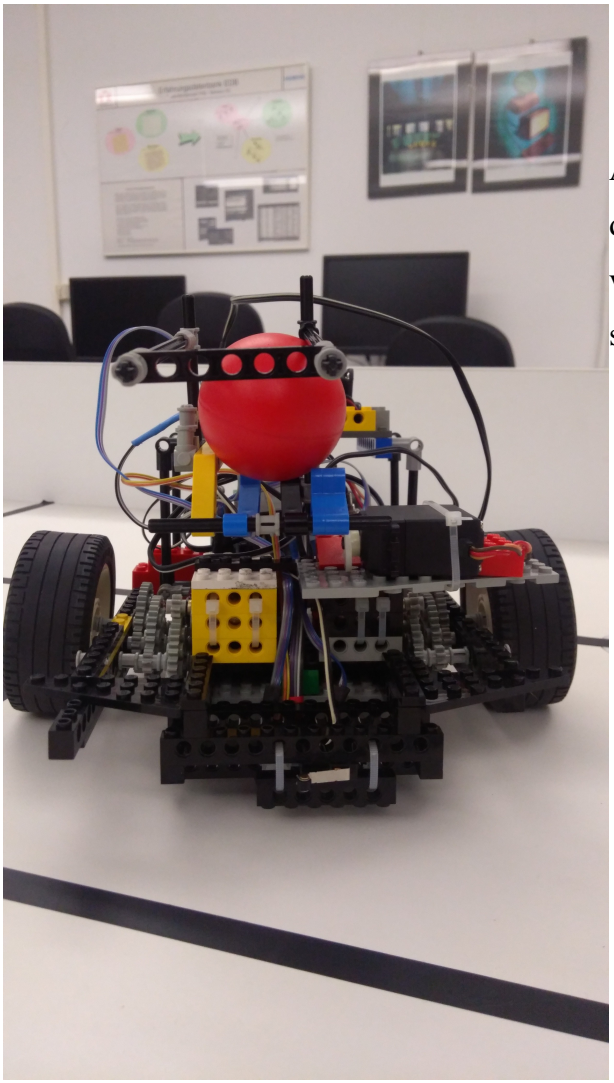

Abbildung 4: Hier zu sehen das leichte „Durchhängen“ des Roboters aufgrund seines Gewichtes. Außerdem vorne zu sehen der Taster der erkennt wann der Roboter sein Ziel erreicht hat.

3.2 Fahraufträge

Um den kürzesten Weg für KB-Jack zu seinem Ziel zu finden haben wir alle Kreuzungen im Streckennetz mit Zahlen belegt die ihre Entfernung zum Zielfeld widerspiegeln. Angefangen wird bei der letzten Kreuzung vor dem Ziel. Diese wird mit „1“ belegt. Anschließend werden mithilfe der Funktion nachbarnerhoehen() alle Nachbarkreuzungen, die noch nicht belegt wurden, mit „2“ belegt, deren Nachbarkreuzungen mit „3“ usw. Dies erfolgt bis alle Kreuzungen die erreichbar sind belegt wurden. Auch sein aktuelle Startkreuzung erhält eine dementsprechende Zahl. Von diesem Feld aus werden nun die Nachbarkreuzungen abgesucht indem geprüft wird welcher Kreuzung eine Zahl zugewiesen wurde die um 1 kleiner ist als die der aktuellen Kreuzung. Ist eine solche Kreuzung gefunden wird aus ihrer Lage und KB-Jacks aktueller Ausrichtung der Befehl für die später abzuarbeitende Befehlskette abgeleitet. Dies wird fortgeführt bis die Kreuzung mit dem Wert 1 erreicht wurde. Auf diese Weise findet KB-Jack immer den kürzesten Weg zu seinem Ziel und läuft nicht Gefahr Umwege zu fahren. Allerdings wird bei unserer Lösung für jeden Lieferauftrag das Streckennetz neu belegt, was bei erhöhter Auftragslage zu erhöhtem Rechenaufwand führt. Würde man die Kreuzungsbelegung am Startfeld starten müsste man das Streckennetz nur einmal belegen, dafür wären dann die Zielkreuzungen nicht mehr eindeutig mit einer Zahl belegt wie in unserem Fall mit der „1“.

4 Probleme/Schwierigkeiten

Eine Schwierigkeit beim Bau des Roboters war der Umgang mit einige Bauteilen, die schnell kaputt gingen oder die schon defekt waren. So ließ uns ein Motor schon früh in der Konstruktionsphase im Stich und einer der 3 Optokoppler ärgerte uns bis zuletzt. Auch die Akkubeladung wirkte sich teilweise stark auf die Fahreigenschaften von KB-Jack aus. War der Akku voll drehte er sich manchmal so schnell dass er schwarze Linien übersah und deswegen falsch abbog. War der Akku fast leer ruckelte er nur noch über die Linien und stand stets schief vor dem Zielreifen.
Ein weiteres Problem mit KB-Jack war seine Gewichtsverteilung. Durch die Anordnung des Getriebes und die somit zu weit voneinander entfernten Reifen erfolgt eine starke Biegung in der Mitte des Roboters. Die Folge davon war unnötig erzeugte Reibung, die die Räder bremste und auch die Kraft der Motoren leiden ließ. Dieses Problem konnte durch größere und breitere Reifen zwar verringert aber nicht vollständig gelöst werden. Durch ein paar stabilisierende Steine konnte jedoch zu große Reibung vermieden werden was auch dazu führte dass KB-Jack seinen Fahrstil wieder deutlich geradliniger ausführen konnte(Abb. 5 , rote Steine) Auch der Servo-Motor der über dem linken Motor befestigt ist hat zu Problemen mit der Kraftverteilung der Motoren geführt, der rechte Motor hatte viel mehr Kraft als der linke aufgrund des Gewichtsunterschiedes. Durch eine Anpassung der Motorenstärken wurde dieses Problem ein wenig ausgeglichen.

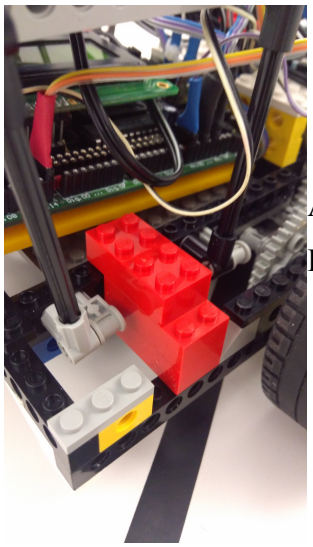

Abbildung 5 : Diese Stabilisation der Reifenachse reichte um zu große Reibung zu vermeiden und KB-Jack wieder schön gerade fahren zu lassen

Bis zuletzt nicht gelöst werden konnten die Launen von KB-Jack nach längerer „Ruhezeit“ (in der Regel 7 Tage, aber auch ca. 30 Minuten Programmierpause reichten aus) erstmal eine gewisse „Warmlaufphase“ zu brauchen und erst nach einigen Versuchen einwandfrei zu funktionieren. Zumj Verhängnis wurde uns dieses Verhalten auch im Wettbewerb bei dem KB-Jack zuverlässig jeden 1. Versuch sprichwörtlich gegen die Wand fuhr um dann anschließend im 2. Versuch zu zeigen dass er seine Aufgaben eigentlich mit Bravour lösen kann.

5 Fazit

Dieses Projekt hat uns sehr viel Spaß gemacht und unser Roboter „Kabelbinder-Jack“ ist ein Ergebnis guter und konstanter Teamarbeit. Wir haben alle während des Semesters geforderten Ziele umgesetzt und rechtzeitig zum Wettbewerb einen konkurrenzfähigen Roboter an den Start gebracht. Auch wenn es trotzdem noch einiges zu verbessern gibt, sind wir mit unserer Arbeit zufrieden, da der Roboter jede Route aus der Datei der Fahraufträge erfolgreich bewältigen kann.

6 Vorschläge

Die Ladesituation der Akkus ist nach wie vor nicht optimal. Trotz der extra gekennzeichneten Körbchen hatten wir immer wieder andere Akkus in unserem Körbchen, einmal sogar 3. Vielleicht sollte man selbst die Akkupacks noch extra beschriften.

7 Quellcode

```
#include <stub.h>

#include <string.h>

#define FA11

#include "fa.h"



char befehle[60]; //Array für Fahrbefehle

int zaehler = 1;  //Befehlszähler

int startfeld;    //Variable für Startfeld

int nplan[70];    //Array für Feldbelegung mit Zahlen

int felderzahl = 70; //Variable zur Überprüfung der belegten Felder in nplan

int ffeld = 0;    //aktuelles Zielfeld

int fanzahl = 0;  //Anzahl Zielfelder

int ffelder[7] = { 0, 0, 0, 0, 0, 0, 0 }; //Array für Zielfelder

int aktpos = 0;   //Feldposition von KB-Jack bei der Routenberechnung

char ausrichtung = 'V'; //Ausrichtung von KB-Jack, V entspricht Nordrichtung

int fahren = 1;   // Variable um While-Schleife zu verlassen


void routeplanen(int start, int strecke[]){

   /*Mit "routeplanen" wird die Befehlskette für KB-Jack erstellt.

   Von seinem Startpunkt aus wird immer die nächste Kreuzung gesucht deren Wert

   um 1 kleiner ist als der seiner aktuellen. Dementsprechend wird ein Befehl in das

   Array "befehle" geschrieben sowie wenn nötig seine Ausrichtung verändert.  */

         int i = 0, laenge = 0, j = 0;
```

```
        aktpos = start;
        /*If-Abfrage falls das Zielfeld nicht erreichbar ist(Kreuzungen um das Startfeld herum
sind dann mit "0" belegt da sie nicht erreicht werden können) oder das Startfeld blockiert ist
(in diesem Fall wären die Kreuzungen mit "100" belegt). In diesen Fällen wird die Befehlserstellung
übersprungen*/
        if (((strecke[aktpos - 7] == 0) || (strecke[aktpos - 7] == 100))
    && ((strecke[aktpos - 1] == 0) || (strecke[aktpos - 1] == 100))
    && ((strecke[aktpos + 1] == 0) || (strecke[aktpos + 1] == 100))){
                befehle[0] = 0;
        }
else {
                while (strecke[aktpos] != 1){
                        if (strecke[aktpos - 7] < strecke[aktpos]){
                                aktpos = aktpos - 7;
                                switch (ausrichtung){
                                  case 'V':befehle[i] = 'G'; break;
                                  case 'R':befehle[i] = 'L'; ausrichtung = 'V'; break;
                                  case 'L':befehle[i] = 'R'; ausrichtung = 'V'; break;
                                }
                        }
        else
        if (strecke[aktpos + 1] < strecke[aktpos]){
           aktpos = aktpos + 1;
           switch (ausrichtung){
             case 'V':befehle[i] = 'R'; ausrichtung = 'R'; break;
             case 'R':befehle[i] = 'G'; break;
```

```
      case 'H':befehle[i] = 'L'; ausrichtung = 'R'; break;
    }
  }
                else
                if (strecke[aktpos - 1] < strecke[aktpos]){
                        aktpos = aktpos - 1;
                        switch (ausrichtung){
                          case 'V':befehle[i] = 'L'; ausrichtung = 'L'; break;
                          case 'H':befehle[i] = 'R'; ausrichtung = 'L'; break;
                          case 'L':befehle[i] = 'G'; break;
                        }
                }
                else
                if (strecke[aktpos + 7] < strecke[aktpos]){
                        aktpos = aktpos + 7;
                        switch (ausrichtung){
                          case 'H':befehle[i] = 'G'; break;
                          case 'R':befehle[i] = 'R'; ausrichtung = 'H'; break;
                          case 'L':befehle[i] = 'L'; ausrichtung = 'H'; break;
                        }
                }
                i++;
                laenge++;
        }
        /*Ist das Zielfeld erreicht, gibt es einen Extra-Befehl für KB-Jack damit er weiß wie
        er abbiegen muss*/
```

```
                if (aktpos - 7 == ffeld){
                        switch (ausrichtung){
                          case 'V':befehle[i] = 'W'; break;
                          case 'R':befehle[i] = 'A'; ausrichtung = 'V'; break;
                          case 'L':befehle[i] = 'D'; ausrichtung = 'V'; break;
                        }
                }
                else
        if (aktpos + 1 == ffeld){
           switch (ausrichtung){
             case 'V':befehle[i] = 'D'; ausrichtung = 'R'; break;
             case 'R':befehle[i] = 'W'; break;
             case 'H':befehle[i] = 'A'; ausrichtung = 'R'; break;
           }
        }
        else
        if (aktpos - 1 == ffeld){
                        switch (ausrichtung){
                          case 'V':befehle[i] = 'A'; ausrichtung = 'L'; break;
                          case 'H':befehle[i] = 'D'; ausrichtung = 'L'; break;
                          case 'L':befehle[i] = 'W'; break;
                        }
        }
                laenge++;
                /*Nach dem Extra-Befehl wird nun der Rückweg erstellt indem der Hinweg in
umgekehrter
                Reihenfolge angehängt wird und "R"-Befehle (Rechtsabbiegen) in "L"-Befehle
```

```
umgewandelt werden und umgekehrt*/
                for (j = 0; j < laenge; j++){
                        if (befehle[i - j] == 'G')
                                befehle[i + j] = 'G';
                        else
                        if (befehle[i - j] == 'R')
                                befehle[i + j] = 'L';
                        else
                        if (befehle[i - j] == 'L')
                                befehle[i + j] = 'R';
                }
                /*Zum Schluss wird ein "S"-Befehl eingefügt der KB-Jacks Aktion auf dem Startfeld
erhält*/
                befehle[i + j - 1] = 'S';
    }
}


void nachbarnerhoehen(int feld, int wert){
    /*Hier werden die Nachbarkreuzungen einer mitgegebenen Kreuzung mit dem um 1
    erhöhten Wert dieser Kreuzung belegt falls sie noch nicht belegt wurden ("0") und die
    Variable "felderzahl" wird um 1 erniedrigt damit nicht mehr als 70 Kreuzungen belegt werden */
        if ((nplan[feld + 7] == 0) && (-1<feld + 7) && (feld + 7<70)){
                nplan[feld + 7] = wert + 1;
                felderzahl = felderzahl - 1;
        }
        if ((nplan[feld - 7] == 0) && (-1<feld - 7) && (feld - 7<70)){
                nplan[feld - 7] = wert + 1;
```

```
            felderzahl = felderzahl - 1;
        }
        if ((nplan[feld + 1] == 0) && (-1<feld + 1) && (feld + 1<70)){
            nplan[feld + 1] = wert + 1;
            felderzahl = felderzahl - 1;
        }
        if ((nplan[feld - 1] == 0) && (-1<feld - 1) && (feld - 1<70)){
            nplan[feld - 1] = wert + 1;
            felderzahl = felderzahl - 1;
        }
}

void planerstellen(int zielfeld){
   /*Vom Zielfeld ausgehend werden hier die freien Kreuzungen in "nplan" mit Werten belegt.
   Die Kreuzung vor dem Ziel erhält die 1, deren Nachbarn die 2 usw. Kreuzungen die nicht erreicht werden können
   behalten ihre "0"*/
        int j, i;
        nachbarnerhoehen(zielfeld, 0);
        for (j = 1; j<30; j++){
            if (felderzahl > 0){
                for (i = 0; i<70; i++){
                    if (nplan[i] == j)
                        nachbarnerhoehen(i, j);
                }
            }
        }
```

```
}


void planuebernehmen( unsigned char plen[], int np[]){

    /*Hier wird der geforderte Fahrauftrag eingelesen und in ein eigenes Int-Array übertragen

    (x->100,F->99,.->0). Wird ein "F"-Feld übertragen wird dieses außerdem im Array "ffelder"
gespeichert sowie

    die Variable "fanzahl" um 1 erhöht. Die Variable "felderzahl" wird um 1 erniedrigt wenn eine
Kreuzung mit

    einer anderen Zahl als 0 belegt wird.*/

        int i;

        for (i = 0; i < 70; i++) {

                if (plen[i] == 'x'){

                        np[i] = 100;

                        felderzahl = felderzahl - 1;

                }

                else

        if (plen[i] == 'F'){

                            np[i] = 99;

                            ffeld = i;

                            if (ffelder[fanzahl] == 0){

                                    ffelder[fanzahl] = ffeld;

           }

           fanzahl = fanzahl + 1;

                            felderzahl = felderzahl - 1;

                    }

        else

           np[i] = 0;
```

```
        }
}

void linienfolgen(){
    /*Solange der mittlere der 3 Optokoppler (analog(0)) schwarz sieht fährt Kb-Jack
    geradeaus. Sieht dieser Optokoppler kein schwarz wird überprüft ob der linke
    Optokoppler (analog(2)) weiß sieht. Dann korrigiert sich KB-Jack nach links sonst nach rechts
    bis der mittlere Optokoppler wieder schwarz sieht  */
        if (analog(0) > 50){
                motor_richtung(3, 0);
                motor_pwm(3, 8);
                motor_richtung(2, 0);
                motor_pwm(2, 10);
        }
        else
                if (analog(2) < 50){
                   motor_richtung(3, 0);
                   motor_pwm(3, 0);
                   motor_richtung(2, 0);
                   motor_pwm(2, 10);
                }
                else {
                        motor_richtung(3, 0);
                        motor_pwm(3, 7);
                        motor_richtung(2, 0);
                        motor_pwm(2, 0);
```

```
                }
}


void linienfolgen2(){
    /*verlangsamtes linienfolgen für die Anfahrt auf das Zielfefd*/
        if (analog(0) > 50){
                motor_richtung(3, 0);
                motor_pwm(3, 7);
                motor_richtung(2, 0);
                motor_pwm(2, 7);
        }
        else
                if (analog(2) < 50){
                   motor_richtung(3, 0);
                   motor_pwm(3, 0);
                   motor_richtung(2, 0);
                   motor_pwm(2, 8);
                }
                else{
         motor_richtung(3, 0);
                        motor_pwm(3, 8);
                        motor_richtung(2, 0);
                        motor_pwm(2, 0);
                }
}
```

```
void linksAbbiegen(){
    /*Zum Links abbiegen fährt KB-Jack über die Kreuzung und biegt dann links ab
    bis der mittlere Oktokoppler wieder schwarz erkennt*/
        motor_richtung(3, 0);
        motor_pwm(3, 8);
        motor_richtung(2, 0);
        motor_pwm(2, 8);
        sleep(300);

        motor_richtung(3, 0);
        motor_pwm(3, 10);
        motor_richtung(2, 1);
        motor_pwm(2, 7);
        sleep(500);
        while (analog(0) < 50);
}

void rechtsAbbiegen(){
    /*Zum Rechts abbiegen fährt KB-Jack über die Kreuzung und biegt dann rechts ab
    bis der mittlere Oktokoppler wieder schwarz erkennt*/
        motor_richtung(3, 0);
        motor_pwm(3, 10);
        motor_richtung(2, 0);
        motor_pwm(2, 10);
        sleep(300);
```

```
        motor_richtung(2, 0);
        motor_pwm(2, 10);
        motor_richtung(3, 1);
        motor_pwm(3, 7);
        sleep(500);
        while (analog(0) < 50);
}

void Pizza(){
   /*Der Greifarm wirft den Ball ab und wartet kurz damit der Ball auch wirklich abgeworfen wurde*/
        servo_arc(2, 45);
        sleep(1000);
}

void Pizza2(){
   /*Der Greifarm geht zurück in seine Ausgansstellung*/
        servo_arc(2, 5);
}

void abladen(){
   /*"abladen" enthält die Komponenten um den Ball abzuwerfen
   und den Greifarm wieder in Stellung zu bringen*/
        Pizza();
        Pizza2();
}
```

```
void linksAbbiegen2(){
   /* Geändertes linksabbiegen für die Drehung nach einer Pizzalieferung.
   Hier wird sofort links abgebogen ohne vorheriges kurzes Geradeausfahren*/
        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
        sleep(300);

        motor_richtung(3, 0);
        motor_pwm(3, 10);
        motor_richtung(2, 1);
        motor_pwm(2, 10);
        sleep(300);
        while (analog(0) < 50);
}

void umdrehen(){
   /*180 Grad Drehung nach Pizzalieferung um wieder richtig
   ausgerichtet zu sein für den Rückweg. Dazu wird erst ein Stück zurückgesetzt
   um den Zielreifen bei der Drehung nicht mitzunehmen, danach wird 2 mal links abgebogen */
        motor_richtung(3, 1);
        motor_pwm(3, 7);
        motor_richtung(2, 1);
        motor_pwm(2, 7);
        sleep(700);
```

```
        linksAbbiegen2();

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);

        linksAbbiegen2();

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
}

void umdrehenziel(){
    /*180 Grad Drehung am Startfeld, sowie eine kurze Pause um neu beladen zu werden*/
        linksAbbiegen2();

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);

        linksAbbiegen2();
```

```
        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
        sleep(500);
}

void umdrehenrechts(){
   /*90 Grad Drehung nach rechts nach der Pizzalieferung*/
        motor_richtung(3, 1);
        motor_pwm(3, 7);
        motor_richtung(2, 1);
        motor_pwm(2, 7);
        sleep(700);

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
        sleep(300);

        motor_richtung(2, 0);
        motor_pwm(2, 9);
        motor_richtung(3, 1);
        motor_pwm(3, 7);
```

```
        sleep(500);
        while (analog(0) < 50);

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);

}

void umdrehenlinks(){
   /*90 Grad Drehung nach links nach der Pizzalieferung*/
        motor_richtung(3, 1);
        motor_pwm(3, 7);
        motor_richtung(2, 1);
        motor_pwm(2, 7);
        sleep(800);

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
        sleep(300);

        motor_richtung(2, 1);
        motor_pwm(2, 7);
```

```
        motor_richtung(3, 0);
        motor_pwm(3, 10);
        sleep(500);
        while (analog(0) < 50);

        motor_richtung(3, 0);
        motor_pwm(3, 0);
        motor_richtung(2, 0);
        motor_pwm(2, 0);
}

void kreuzungsaktion(){
    /*Wurde eine Kreuzung erkannt schaut KB-Jack hier nach was er tun soll,
    anschließend wird die Variable "zaehler" um 1 erhöht*/
    /*Kreuzung geradeaus überfahren*/
        if (befehle[zaehler] == 'G'){
                lcd_puts("G");
                motor_richtung(3, 0);
                motor_pwm(3, 10);
                motor_richtung(2, 0);
                motor_pwm(2, 10);
                sleep(500);
        }
        else /* an der Kreuzung rechts abbiegen */
                if (befehle[zaehler] == 'R'){
                    lcd_puts("R");
```

```
        rechtsAbbiegen();
    }
    else /* an der Kreuzung links abbiegen */
            if (befehle[zaehler] == 'L'){
        lcd_puts("L");
        linksAbbiegen();
            }
            else /* KB-Jacks Ziel liegt vor ihm, d.h. er wird langsamer und fährt geradeaus
bis der Tastsensor (analog(8)) aktiviert wird und liefert die Pizza.
Danach dreht er sich um 180 Grad */
                    if (befehle[zaehler] == 'W'){
            lcd_puts("W");
            motor_richtung(3, 0);
            motor_pwm(3, 0);
            motor_richtung(2, 0);
            motor_pwm(2, 0);
            while (analog(8) != 0){
                    linienfolgen2();
            }
            motor_richtung(3, 0);
            motor_pwm(3, 0);
            motor_richtung(2, 0);
            motor_pwm(2, 0);
abladen();
            umdrehen();
                    }
```

```
                else /*KB-Jacks Ziel liegt links von ihm, d.h. er biegt links ab, wird
langsamer und fährt geradeaus
        bis der Tastsensor (analog(8)) aktiviert wird und liefert die Pizza.
        Danach dreht er sich um 90 Grad nach links */
        if (befehle[zaehler] == 'A'){
                        lcd_puts("A");
    linksAbbiegen();
    motor_richtung(3, 0);
                        motor_pwm(3, 0);
                        motor_richtung(2, 0);
    motor_pwm(2, 0);
                        while (analog(8) != 0){
                                linienfolgen2();
                        }
                        motor_richtung(3, 0);
                        motor_pwm(3, 0);
                        motor_richtung(2, 0);
                        motor_pwm(2, 0);
                        abladen();
                        umdrehenlinks();
                }
                        else /*KB-Jacks Ziel liegt rechts von ihm, d.h. er biegt rechts ab, wird
langsamer und fährt geradeaus
          bis der Tastsensor (analog(8)) aktiviert wird und liefert die Pizza.
          Danach dreht er sich um 90 Grad nach rechts */
                        if (befehle[zaehler] == 'D'){
         lcd_puts("D");
```

```
                                        rechtsAbbiegen();
                                        motor_richtung(3, 0);
                                        motor_pwm(3, 0);
                                        motor_richtung(2, 0);
                                        motor_pwm(2, 0);
                                        while (analog(8) != 0){
                                                linienfolgen2();
                }
                                        motor_richtung(3, 0);
                                        motor_pwm(3, 0);
                                        motor_richtung(2, 0);
                                        motor_pwm(2, 0);
                                        abladen();
        umdrehenrechts();
                                }
                                else /*KB-Jack ist wieder am Startpunkt angekommen, fährt ein
Stück geradeaus,
                                    dreht sich dann um 180 Grad und macht eine kurze Pause um die
neue Pizza zu erhalten*/
                                   if (befehle[zaehler] == 'S'){
          linienfolgen();
          sleep(400);
                                        motor_richtung(3, 0);
                                        motor_pwm(3, 0);
                                        motor_richtung(2, 0);
                                        motor_pwm(2, 0);
                                        umdrehenziel();
```

```
                                        sleep(600);
                        }
                                        else { /*KB-Jack erhält keinen Befehl und bleibt stehen*/
                                                lcd_puts("Fehler");
                                motor_richtung(3, 0);
                                                motor_pwm(3, 0);
                                                motor_richtung(2, 0);
                                                motor_pwm(2, 0);
                                                sleep(5000);
                                        }
                                        zaehler++;
}

void kreuzung(){
   /*Erkennen die äußeren beiden Optokoppler beide schwarz, befindet sich
   KB-Jack auf einer Kreuzung und "kreuzungsaktion()" wird ausgelöst. Andernfalls
   fährt er weiter geradeaus.*/

        if ((analog(2) > 50) && (analog(4) > 50)){
                kreuzungsaktion();
        }
        else{
                 linienfolgen();
        }
}
```

```
void starteinstellen(){
   /*Über die Dip-Schalter wird das Startfeld festgestellt
     und die Variable "startfeld" belegt*/

        if (dip_pin(1)) // Startfeld Links
                startfeld = 64;
        else
      if (dip_pin(2)) //Startfeld Rechts
                startfeld = 68;
        servo_arc(2, 5);
}

void fahrendlich(){
   /*fahrendlich() enthält den Code zur Abarbeitung der Auftäge von KB-Jack*/
        int i;
        int j;
        int k;
        for (i = 0; i < 7; i++){          //for-Schleife für max. 7 Aufträge
                ffeld = ffelder[i];          //das 1. Zielfeld wird ausgewählt und
                planerstellen(ffeld);        //auf dieser Grundlage wird "nplan" belegt
                routeplanen(startfeld, nplan); //KB-Jacks Befehle werden erstellt
                if (befehle[0] == 0){        //Wenn die Befehlskette leer ist, kann das Zielfeld nicht
erreicht werden
          lcd_puts("kein ziel");
        }
                else
                        while (befehle[zaehler] != 0){    //While-Schleife solange Befehle vorhanden sind
```

```
            if (akt_time() > 120000){  //Abbruchbedingung falls die Zeit 2 Minuten erreicht,
                                        fahren = 0;
                                        break;
            }
                                kreuzung();             //Befehlsausführung
                        }
                    if (akt_time() > 120000) //Abbruchbedingung für die for-Schleife falls 2 Minuten erreicht
sind
                        break;
                        /*Alle Variablen und Arrays werden wieder mit ihren Anfangswerten belegt
                        und anschließend wird für den neuen Auftrag bereits planuebernehmen()
durchgeführt*/
                    for (j = 0; j < 70; j++){
                            nplan[j] = 0;
                    }
                    for (k = 0; k < 60; k++){
                            befehle[k] = 0;
        }
        ausrichtung = 'V';
        zaehler = 1;
        ffelder[0] = 0;
        ffelder[1] = 0;
        ffelder[2] = 0;
        ffelder[3] = 0;
        ffelder[4] = 0;
        ffelder[5] = 0;
        ffelder[6] = 0;
```

```
        aktpos = 0;
        felderzahl = 70;
        fanzahl = 0;
        planuebernehmen(_fa, nplan);
        lcd_cls();
    }
}

//Hauptprogrammroutine

//
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
----------

void AksenMain(void){

        starteinstellen();  //Startfeld wird festgestellt
        befehle[0] = 0;
        planuebernehmen(_fa, nplan);  //nplan wird belegt
        while (1){
                led(0, 1);
                /*Erkennt der Lichtsensor(analog(14)) das Startlicht wird die Zeit zurückgesetzt*/
                if (analog(14) < 100){
                        lcd_puts(" licht an");
                        clear_time();
                        /*2 If-Abfragen für den Sonderfall dass die Kreuzung vor dem Startfeld versperrt
ist.
```

```
                Ist dies der Fall erfolgt je nach Startfeld zuerst eine Drehung nach links bzw.
rechts.*/

                if ((startfeld == 64) && (nplan[57] == 100)){

                        motor_richtung(2, 0);

                        motor_pwm(2, 9);

                        motor_richtung(3, 1);

                        motor_pwm(3, 9);

                        sleep(300);

                        while (analog(0) < 50);

                }

                if ((startfeld == 68) && (nplan[61] == 100)){

                        motor_richtung(3, 0);

                        motor_pwm(3, 10);

                        motor_richtung(2, 1);

                        motor_pwm(2, 7);

                        sleep(500);

                        while (analog(0) < 50);

                }

                /*While-Schleife die die Ausführung der Fahraufträge enthält. Wird nur
ausgeführt

                  wenn die Zeit unter 2 Minuten ist und die Variable "fahren" mit "1" belegt ist */

                while ((akt_time() <120000)&&fahren ==1){

        fahrendlich();

        /*wird fahrendlich() verlassen sind alle Aufträge abgearbeitet und die Variable "fahren"
wird auf "0"

        gesetzt damit die Abbruchbedingung erfüllt ist und nicht nochmal alle Aufträge ausgeführt
werden*/

        fahren = 0;
```

```
                break;

                        }

                        /*KB-Jack bleibt stehen, entweder weil die Zeit abgelaufen ist oder weil er alle
Aufträge erfüllt hat */

                        motor_richtung(3, 0);

                        motor_pwm(3, 0);

                        motor_richtung(2, 0);

                        motor_pwm(2, 0);

                }

        }

}
```