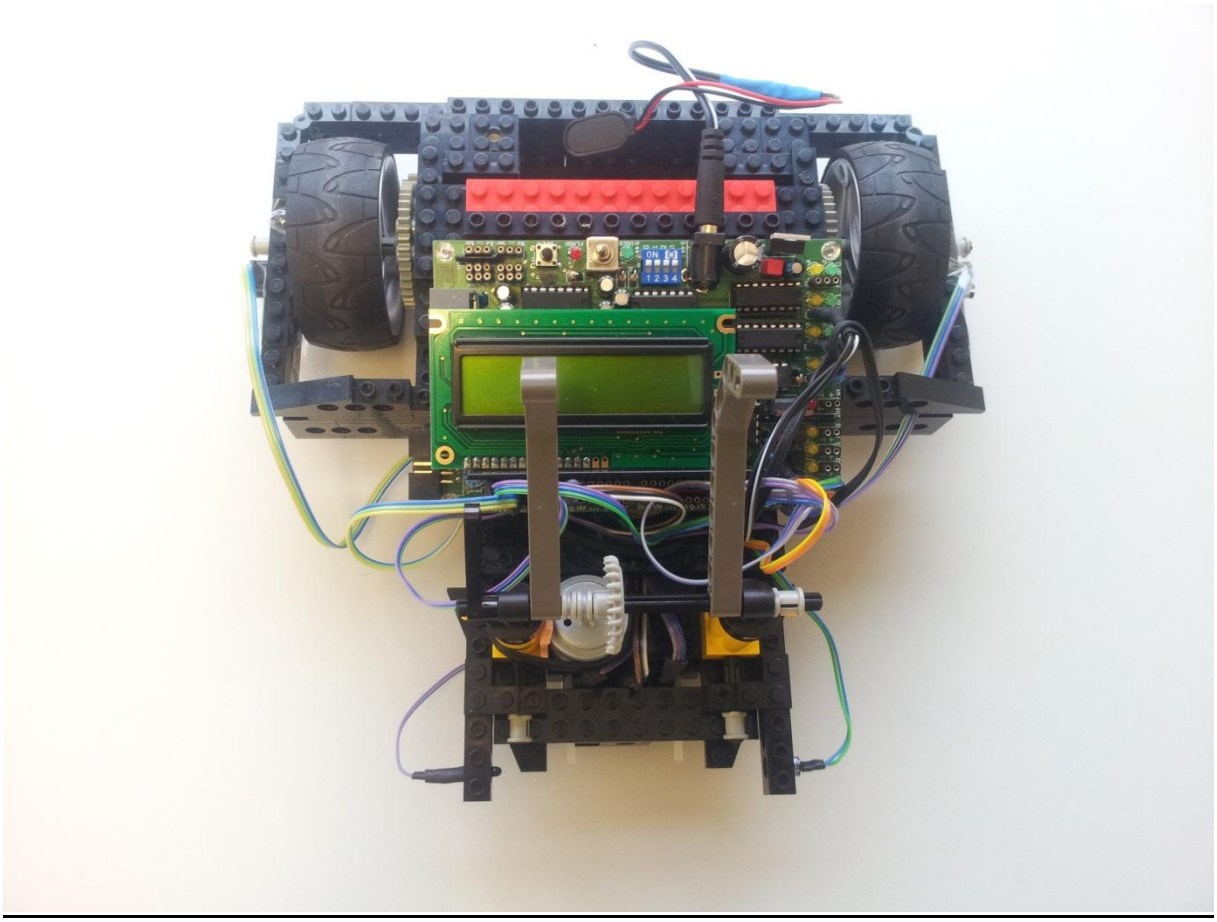
Dokumentation AMS-Projekt Wintersemester 12/13



Vorgelegt von Marcel Hinderlich und Jenny Ludwig

Inhalt

Aufgabe

Die Aufgabe des Projekts bestand darin einen Roboter mit Hilfe eines Mikroprozessorboards, einem Sensorsatz, verschiedenen Aktoren, Akkumulatoren und Lego so zu bauen, dass dieser in der Lage ist einen Fahrauftrag zu erhalten und eine oder mehrere „Personen“ abzuholen und zum Ziel zu bringen. Dabei befindet sich der Roboter auf einem Gitterstreckennetz, in dem sich alle befahrbaren und gesperrten Kreuzungen, sowie „Personen“ befinden, die der Roboter zur Planung seines Weges beachten muss.

Lösungsweg oder auch Trial-And-Error-Strategie

Unser Lösungsweg war simple gestrickt:

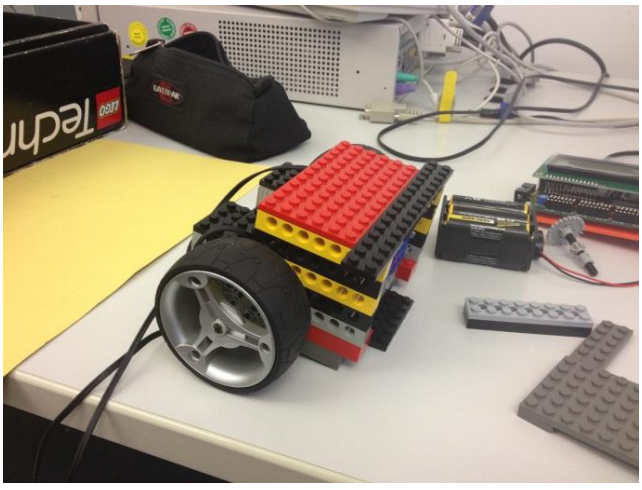
Wenn wir eine Idee hatten, wurde diese umgesetzt und das Ergebnis begutachtet, denn der Vorteil des Projekts bestand darin, dass man meist sofort seine Erfolge oder auch Misserfolge sehen konnte.

In jeder Woche setzten wir uns ein neues Ziel, so machten wir uns beispielsweise in der ersten Woche nur mit den Bauelementen des Projekts bekannt und in der zweiten Woche wurde dann am ersten fahrbaren Untersatz gearbeitet. Dabei entstanden viele kleine Zwischenschritte, welche das strukturierte Voranschreiten im Projekt erst ermöglichten.

Dank dieser Strategie haben wir es geschafft nie komplett die Übersicht zu verlieren, konnten die jeweiligen Aufgaben besser untereinander verteilen und letztendlich zu einem unserer Meinung nach recht gelungenen Abschluss des Projektes kommen.

Vorstellung von Hurby

Schon am Anfang des Projekts standen wir vor einer schwerwiegenden Entscheidung, denn schließlich musste unser kleiner Kampfroboter einen aussagekräftigen und angsteinflößenden Namen erhalten, um für den Wettbewerb bestens gerüstet zu sein. Nach hitzigen Diskussionen und einer schier endlosen Reihe von Namensvorschläge war es dann endlich soweit: Unser Roboter sollte von nun an „Hurby“ genannt werden.

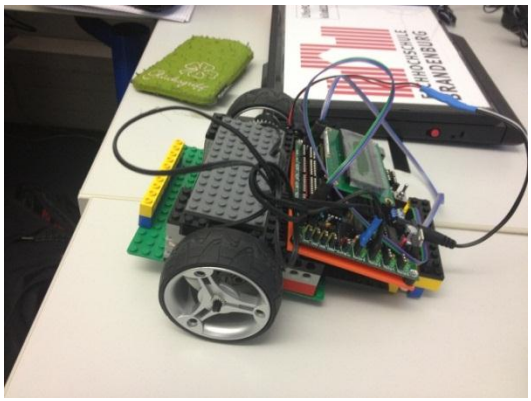


Nachdem diese Entscheidung getroffen wurde, folgte die Produktion. Gerüstet mit einem AKSEN-Board, verschiedenen Motoren, Sonsoren, Legobausteinen

und vielen anderen nützlichen Dingen, wie Schere, Klebeband, Kabelbinder, usw. machten wir uns ans Werk.

In der zweiten Projektwoche stand dann der erste Entwurf von Hurby vor uns. Dieser sollte bislang nur in der Lage sein das AKSEN-Board zu tragen und mithilfe von zwei Motoren einen Differentiellen Antrieb zu realisieren.

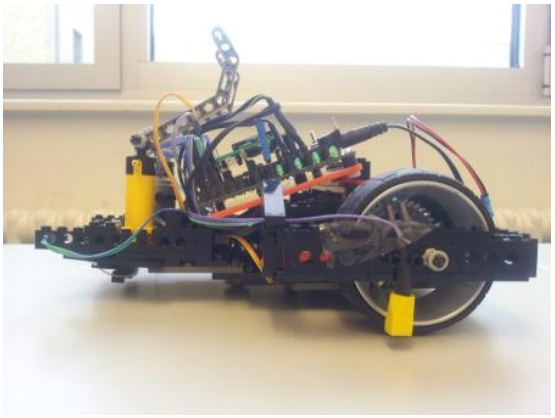
Voller Tatendrang setzten wir Hurby auf die Rennstrecke und wollten uns von seiner Fähigkeit „geradeaus zu fahren“ beeindrucken lassen, doch auf seiner Jungfernfahrt ereilte uns der erste kleine Rückschlag, denn Hurby fuhr nur wilde Kreise. Nachdem unsere Tränen getrocknet waren, erkannten wir unseren Fehler: Die Motoren drehten in unterschiedlichen Richtungen. Unsere Leidenschaft für schwierige Lösungswege verleitete uns dazu, den Fehler in der Software zu suchen: so änderten wir die Motorrichtungen im Quellcode, anstatt einfach einen Motor umzupolen. Endlich konnte Hurby, wenn auch mehr schlecht als recht, eine Linie fahren.



Der nächste Schritt stand an: Hurby sollte sich vom stupiden „Ich fahr wohin mich meine Motoren tragen“-Roboter zum intelligenteren „Ich folge einer schwarzen Linie“- Roboter getunt werden. Dies stellte sich schnell als besonders trickreich heraus, denn Hurbys Aufbau war weder dazu geeignet Sensoren zu tragen, noch dazu den abschließenden Wettbewerb zu gewinnen. Somit musste Hurby sterben, um aus seiner Asche wieder aufzuerstehen: Hurby 2.0 war geboren.

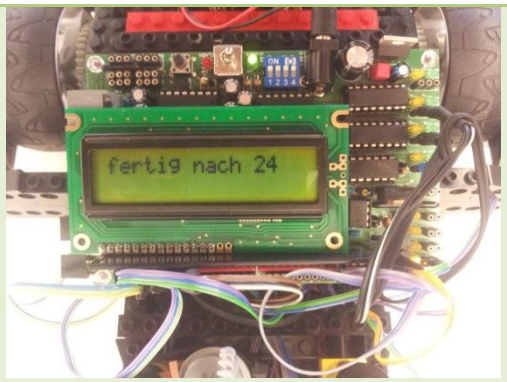
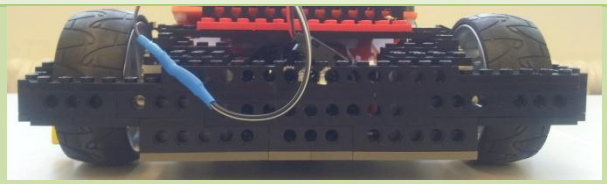
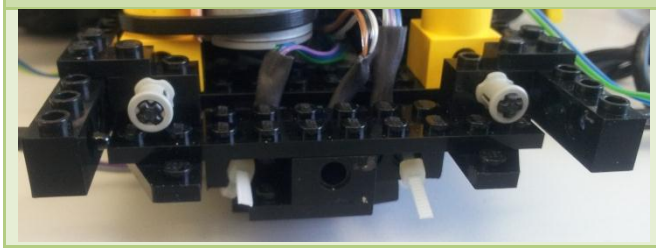
Einige Wochen später sollte sich jedoch herausstellen, dass leider auch dieser Entwurf von Hurby nur mit der Note „befriedigend“ abschließen würde. So wurde in einer Nacht-und Nebelaktion und unter vollem Einsatz unserer mentalen Fähigkeiten zunächst Hurbys Unterbau optimiert, dann bekamen seine Räder eine Umrandung zur Stabilisierung und letztlich der Greifer montiert.

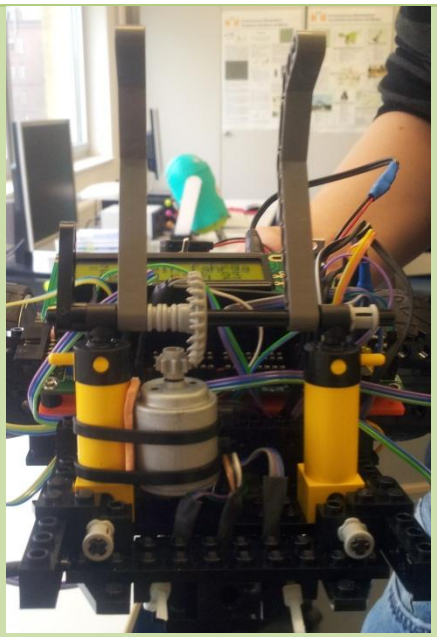
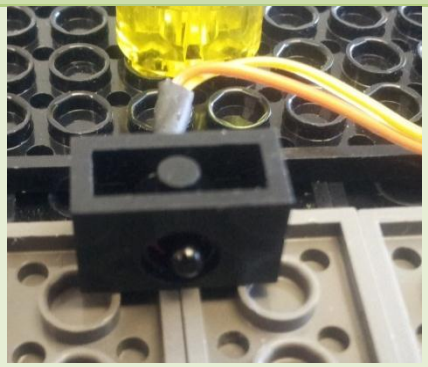
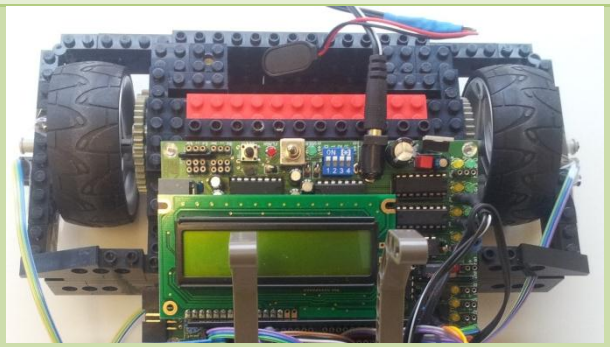
Besonders frustrierend dabei war, dass wir unseren Kampfroboter in einem edlen Design daherkommen lassen wollten, somit war es eine Pflicht, dass jedes Element aufeinander abgestimmt werden musste. In stundenlanger Kleinstarbeit wurden alle Legokästen durchforstet und nach einigen Flüchen und Beschimpfungen war es schließlich soweit:



Wir hatten es geschafft! Wir hatten den Roboter gebaut, der voll unseren Vorstellungen entsprach.

Hardware

	Das Kernstück von Hurby ist das AKSEN-Mikroprozessorboard, welches die Schnittstelle zwischen der Hard- und Software darstellt.
	Hurby wird durch zwei Lego-Motoren angetrieben, welche jeweils ein Rad mit der Umsetzung 1-zu-5 drehen.
	Unterhalb von Hurbys Greifer befinden sich drei Optokoppler, die die Linienfolgerung und Kreuzungserkennung umsetzen. Diese nehmen wahr, ob sie sich auf einer schwarzen Linie oder einer weißen Fläche befinden.
	Um sich bei Kreuzungen genau auf der Radachse zu drehen, befindet sich an den beiden Rädern jeweils noch ein Optokoppler.
	Mittels eines Infrarotsenders und Empfängers haben wir das Konzept einer „Lichtschranke“ realisieren wollen. Somit erkennt Hurby wann ein Fahrgast direkt vor ihm ist und kann den Greifer schließen.

	Der Greifer wird durch einen Motor angetrieben, welcher das Öffnen und Schließen realisiert. Hurby kann somit Fahrgäste aufnehmen und zum Ziel bringen.
	Mittig unter Hurby befindet sich ein weiterer Infarotempfänger, der lediglich dazu dient, den Start des Roboters zu regeln. Hurby erkennt wann das Licht unter ihm ausgeschaltet wird und weiß somit ab wann er losfahren kann.
	Hurby wurde durch einen Rahmen aus Legosteinen zusätzlich stabilisiert, damit die Radachsen nicht durchgebogen werden.
	Als letzte relevante Hardwarekomponente ist noch der Akku zu erwähnen, ohne den Hurby nicht selbstständig fahren könnte.

Software

Der Funktionsumfang des AKSEN-Boards bot einige Möglichkeiten die uns gestellte Aufgabe zu lösen. Wir haben uns für folgende entschieden:

Die Anzahl der Codezeilen wuchs exponentiell und so wurden schnell 500 Zeilen erreicht. Um nicht den Faden zu verlieren musste eine Struktur her. Wir unterteilten den Quelltext in drei Dateien.

Wir benutzen die aktionen.h für die Definition von Präprozessordirektiven und Deklaration aller verwendeten Funktionen.

Die ringbuffer.h stellt einen Ringpuffer mit Funktionen zur dynamischen Speicherverwaltung nach dem FIFO – Prinzip bereit.

In der hello.c werden sämtliche zuvor deklarierte Methoden initialisiert, des Weiteren enthält sie die Hauptschleife zur Programmdurchführung.

Auf dem AKSEN-Board wurden neben der Hauptfunktion

```
void AksenMain(void);
```

folgende Funktionen implementiert:

```
int fahr_geradeaus(int speed);
int korrigiere(int speed, int richtung);
int korrigiere_links(int speed);
int korrigiere_rechts(int speed);
int turnAround(int speed);
int stop();
int stueck_zurueck(int speed);
int kreuzung_links(int speed);
int kreuzung_rechts(int speed);
void check_fahrplan(char p_fahrplan);
void aufnehmen();
char fahrplan_auslesen(int start);
void breitensuche(int start);
```

Der Roboter fährt standartmäßig geradeaus. Hierbei müssen beide Motoren mit der gleiche Geschwindigkeit in verschiedene Richtungen drehen, also einer gegen und einer mit dem Uhrzeigersinn.

```
int fahr_geradeaus(int speed){
   motor_richtung(2, VORWAERTS);
   motor_pwm(2, speed);
   motor_richtung(3,RUECKWAERTS);
   motor_pwm(3,speed);
   return 0;
}
```

Hurby korrigiert seine Position je nachdem ob jeweils der linke oder rechte Optokoppler zu hohe Werte annehmen (auf die schwarze Fahrlinie zeigen) und gleichzeitig der mittlere geringe Werte zeigt (weiß)

```
  if((analog(8)>150)&&(analog(14)<150)){
     korrigiere_rechts(10);
  }else if((analog(14)<150)&&(analog(7)>150)){
     korrigiere_links(10);
  }else fahr_geradeaus(10);
```

Das Korrigieren erfolgt durch Anpassen der Motorgeschwindigkeiten der jeweiligen Seite.
Beim Linkskorrigieren fährt der rechte Motor weiter wie gehabt und der linke stoppt.
Das Rechtskorrigieren erfolgt simultan, lediglich der rechte stoppt und der linke fährt weiter

```
int korrigiere_links(int speed){
   motor_richtung(2, VORWAERTS);
   motor_pwm(2, 0);
   motor_richtung(3,RUECKWAERTS);
   motor_pwm(3,speed);
   return 0;
}
```

```
int korrigiere_rechts(int speed){
   motor_richtung(3, RUECKWAERTS);
   motor_pwm(3, 0);
   motor_richtung(2,VORWAERTS);
   motor_pwm(2,speed);
   return 0;
```

Das Abbiegen bzw. Geradeausfahren erfolgt anhand einer Anweisungskette, die Charakterweise abgearbeitet wird. Sofern einer der mittleren Optokoppler auf schwarz kommt befindet sich Hurby auf einer Kreuzung.

```
if(((analog(10)>150)||(analog(12)>150))&&aktivierung<akt_time()){
      sleep(80);
      check_fahrplan(abarbeiten[index]);
      ...

void check_fahrplan(char p_fahrplan){
   switch(p_fahrplan){
       case 'l':  kreuzung_links(10);
          break;
       case 'r':  kreuzung_rechts(10);
          break;
```

```
        case 'g'   //fahr_geradeaus(10);
            break;
        default:
      }
}
```

Hurby dreht um den Mittelpunkt der Radachse und schafft dadurch das Drehen auf der Stelle. Technisch gesehen drehen sich beide Motoren gegen bzw. mit dem Uhrzeigersinn.

```
int kreuzung_links(int speed){
   motor_richtung(3, RUECKWAERTS);
   motor_pwm(3, speed);
   motor_richtung(2,RUECKWAERTS);
   motor_pwm(2,speed);
   sleep(500);
   while(analog(14)<150);
   return 0;
}
```

```
int kreuzung_rechts(int speed){
   motor_richtung(3, VORWAERTS);
   motor_pwm(3, speed);
   motor_richtung(2,VORWAERTS);
   motor_pwm(2,speed);
   sleep(500);
   while(analog(14)<150);
   return 0;
}
```

Hierbei war besonders ärgerlich, wenn Kreuzungen mehrmals erkannt wurden, obwohl nur eine überfahren wurde, weil die Abarbeitung zu schnell von statten ging. Des Rätsels Lösung fand sich in der temporären Deaktivierung dieser Funktion mittels Counter. Die Anweisung greift demnach nur sofern die aktuelle Zeit größer ist als die Aktiverungszeit, also eine bestimmte Zeitspanne (bei uns 400 Zeiteinheiten) verging.

```
#define DEAKTIV 400

if(((analog(10)>150)||(analog(12)>150))&&aktivierung<akt_time()){
      sleep(80);
      check_fahrplan(abarbeiten[index]);
      aktivierung = akt_time() + DEAKTIV;
      ...
```

Fahrgast aufnehmen

Anhand des Infrarotsender und Infarotempfängers wird eine Art Lichtschranke aufgebaut, die sofern unterbrochen den Aufgreifmechanismus in Gang setzen. Der Sender wird aktiviert und strahlt den gesamten Vorgang über Licht aus.

```
...
led(3,1);

if(analog(0)>200 && klappe==0){
      aufnehmen();
      klappe=1;
       stueck_zurueck(SPEED);
```

```
      sleep(200);
      turnAround(SPEED);
      sleep(200);
      while(analog(7)<150);
      sleep(200);
      while(analog(7)<150);
   }
```

Sofern die Lichtschranke unterbrochen wird und das Flag für die Klappe nicht gesetzt (die Klappe also geöffnet)ist, wird der Greifmotor angesteuert und nach unten gefahren, das Flag gesetzt, Huby fährt ein Stück zurück, um von der Wand wegzukommen und dreht sich anschließend 180° um die eigene Achse.

Die Breitensuche erfolgt einmalig am Anfang. Die Matrix wird ausgehend vom Startpunkt (linke Position 64, rechte Position 68) aus, angefangen bei der Null, mit den Kosten gefüllt. Lediglich das Auslesen erfolgt ggf. öfter, sofern mehrere Passagiere vorhanden sind. Einmalig wiederum am Anfang zur Initialisieren direkt nach der Breitensuche

```
breitensuche(LEFTSTART);

do{
/*auslesen des Fahrplans*/
checkout = fahrplan_auslesen(LEFTSTART);

if(checkout==0){
            lcd_cls();
            lcd_puts("0");
            sleep(2000);
}
else if(checkout==-1){
   //Fahrgast vorhanden, kein Weg gefunden
    lcd_cls();
     lcd_puts("-1");
      sleep(5000);
      //naechsten fahrgast suchen
      map[fahrgast_pos]='x';
      //stack leeren

}else if(checkout==-2){
   //kein Fahrgast vorhanden
   lcd_cls();
     lcd_puts("-2");
      sleep(20000);
}
}while(checkout != 0);
```

und dann falls nötig innerhalb, nachdem der erste Auftrag abgearbeitet wurde.

```
if(abarbeiten[index]==0){
      ...
checkout = fahrplan_auslesen(LEFTSTART);
        if(checkout==-2){
      ...


void breitensuche(int start)
{
   char i = 0;
   short kosten = '0';
   short pos = start;
   char count = 0;
   char bount = 0;
   map[pos]=kosten;
   put(pos);
   while (pos!=0){
      kosten++;
      count=0;
      do{
         pos=get();
         if (map[pos-1]=='.'){
            map[pos-1]=kosten;
            put(pos-1);
            count++;
         }
         if (map[pos-7]=='.'){
            map[pos-7]=kosten;
            put(pos-7);
            count++;
         }
         if (map[pos+1]=='.'){
            map[pos+1]=kosten;
            put(pos+1);
            count++;
         }
         if (map[pos+7]=='.'){
            map[pos+7]=kosten;
            put(pos+7);
            count++;
         }
         bount--;
      }while (bount>0);
      bount=0;
      kosten++;
      do{
         pos=get();
```

```
        count--;
        if (map[pos-1]=='.'){
           map[pos-1]=kosten;
           put(pos-1);
           bount++;
        }
        if (map[pos-7]=='.'){
           map[pos-7]=kosten;
           put(pos-7);
           bount++;
        }
        if (map[pos+1]=='.'){
           map[pos+1]=kosten;
           put(pos+1);
           bount++;
        }
        if (map[pos+7]=='.'){
           map[pos+7]=kosten;
           put(pos+7);
           bount++;
        }
     }while (count>0);
  }
```

Vom Startpunkt ausgehend wird links, rechts, über und unterhalb der currentPosition geprüft ob in der Karte ein Punkt (freie Kreuzung) vorhanden ist, nur dann kann dieser auch mit Kosten gefüllt werden. Mittels zweier while schleifen wird das fortlaufende Füllen gewährleistet. Dieser Vorgang wird wiederholt, solange die Position nicht 0 ist.

```
char fahrplan_auslesen(int start){

...
for (c=0;c<71;c++){
     if (map[c]=='F'){
        current_pos=c;
        fahrgast_pos=c;
        put(c);
        break;
     }
     //wenn
     if(c==70 && map[c]!='F')
        return -2;
}
```

Beim Auslesen des Fahrplans werden die Positionen der Karte von der Null ausgehend durchgegangen und nach Fahrgästen abgesucht (‚F') , ggf. die Position gespeichert oder bei Nichtfinden abgebrochen.

Ausgehend von der Fahrgastposition wird die Kostenmatrix zurückverfolgt bis hin zum Startpunkt und die jeweiligen Positionsübergänge landen umgewandelt und umgedreht im char Array der abzuarbeitenden Folge. Diese wird gespiegelt und wiederum an die Abarbeiten Anweisung angehangen und somit ist der vollständige Fahrauftrag fertig. Sofern alles abgearbeitet wurde, setzt man die in der Karte die Position des alten Fahrgastes auf x um diesen als abgehakt zu kennzeichnen.

Fazit

Durch das Projekt konnten wir sehr viel Praxiserfahrung sammeln und lernten, dass Misserfolg und Erfolg sehr nahe beieinander liegen.

Das Besondere an diesem Projekt ist, dass man mit den verschiedensten Komponenten arbeiten muss und so sein Auge für die Probleme in der Entwicklung schulen konnte. Wir bedanken uns an dieser Stelle auch bei Herrn Boersch, der uns in den verzweifelten Stunden stets zur Seite stand und uns immer wieder zum Weitermachen ermutigte.

Wir würden uns immer wieder für dieses Projekt entscheiden! Man hat nicht nur Spaß, sondern lernt auch sehr viel.

Danke.

Anhang

/*aktionen.h*/

```
#define VORWAERTS 1
#define RUECKWAERTS 0
#define SPEED 10
#define DEAKTIV 400
#define LEFTSTART 64
#define RIGHTSTART 68

/*
* DOKU
* analog7 - linker sensor
* analog8 - rechter sensor
* analog10/12 linker/rechter sensor auf Drehachse
* analog5 - Sensor, der Licht unterm Roboter erkennen soll
* motor2 - linker motor
* motor3 - rechter motor
* taster1 - digital7
*/


/*Roboter faehrt geradeaus in gegebener geschwindigkeit*/
int fahr_geradeaus(int speed);
/* einheitliche korrigiere methode (noch nicht implementiert)*/
int korrigiere(int speed, int richtung);
/* korrigiert nach links, wenn roboter zu weit rechts*/
int korrigiere_links(int speed);
/* korrigiert nach rechts, wenn roboter zu weit links*/
int korrigiere_rechts(int speed);
/*dreht den Roboter um 180 Grad*/
int turnAround(int speed);
/* hält den Roboter an*/
int stop();
/*Roboter ist mit Wand kollidiert und fährt zurück*/
int stueck_zurueck(int speed);
/*Roboter nimmt die linke Richtung der Kreuzung*/
int kreuzung_links(int speed);
/*Roboter nimmt die rechte Richtung der Kreuzung*/
int kreuzung_rechts(int speed);
/*checkt welche anweisung vorliegt, befolgt diese und setzt den pointer eine stelle
weiter*/
void check_fahrplan(char p_fahrplan);
/*passagier aufgabeln*/
void aufnehmen();
/*returns -1: Fahrgast vorhanden, kein Weg gefunden
          -2: kein Fahrgast vorhanden
          1: Auftrag abgearbeitet
*/
char fahrplan_auslesen(int start);
/*initiales füllen der map angafangen beim start*/
void breitensuche(int start);
```

/*ringbuffer.h*/

```
#include <stdio.h>
#include <stdio.h>
#include <ctype.h>
```

```
#define NMAX 30

char iput = 0;  /* Position fuer naechstes abzulegende Zeichen */
char iget = 0;  /* Position fuer naechstes zu lesendes Zeichen */
char n = 0;     /* Anzahl der Zeichen im Buffer */

short buffer[NMAX];

/* addring schreitet im Ring um eins weiter und sorgt dafuer dass
   (NMAX - 1) + 1 nicht NMAX sondern wieder 0 ist */

int addring(int i)
{
  return (i+1) == NMAX ? 0 : i+1;
}


short get(void)
{
  short pos;

  if (n > 0) {
    pos = iget;
    iget = addring(iget);
    n--;
    return buffer[pos];
    }
  else {
    //lcd_puts("Buffer ist leer\n");
    return 0;
  }
}

void put(short z)
{
  if (n < NMAX) {
    buffer[iput] = z;
    iput = addring(iput);
    n++;
    }
  //else
    //lcd_puts("Stack ist voll\n");
}

void clearBuffer(){
     iput=0;
     iget=0;
     n=0;
}
```

/*fa.h*/

```
// Autor: I. Boersch

// Definition der Fahrauftraege zum AMS-Contest MASDAR-City WS12/13
// Um Speicherplatz zu sparen, Ablage als Praeprozessor-Anweisung
// File generiert mit generate_configs_with_solution.py

// Fahrauftrag F1 - From task description - this should be easy!
```

```
#ifdef FA1
unsigned char _fa [] =
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx...x.xxx..x.xF..x..Fx..x..x";
unsigned char _fa_nr = 1;
#define _FA_OK
#endif

// Fahrauftrag F2 - From task description - one crossing modified
#ifdef FA2
unsigned char _fa [] =
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx..x..xxx..x.xF..x..Fx..x..x";
unsigned char _fa_nr = 2;
#define _FA_OK
#endif

// Fahrauftrag F3 - City Slalom
#ifdef FA3
unsigned char _fa [] =
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx..x..xxx.x..xF..x..Fx..x..x";
unsigned char _fa_nr = 3;
#define _FA_OK
#endif

// Fahrauftrag F4 - Race on the Highway
#ifdef FA4
unsigned char _fa [] =
"xFxxxFxx.x.x.xF..x..Fx..x..xx..x..xx.x.x.xF..x..Fx..x..xx..x..xx.x.x.x";
unsigned char _fa_nr = 4;
#define _FA_OK
#endif

// Fahrauftrag F5 - Big Slalom
#ifdef FA5
unsigned char _fa [] =
"xFxxxFxx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.x";
unsigned char _fa_nr = 5;
#define _FA_OK
#endif

// Fahrauftrag F6 - Symmetry
#ifdef FA6
unsigned char _fa [] =
"xxFxFxxF..x..Fx.x.x.xx..x..xF..x..Fx..x..xxx.x.xxx..x..xxx.x.xxx..x..x";
unsigned char _fa_nr = 6;
#define _FA_OK
#endif

// Fahrauftrag F7 - Connected Game
#ifdef FA7
unsigned char _fa [] =
"xxxFxxxx.....xx..x..xx..x..xF..x..Fx..x..xx..x..xxx.x.xxx..x..xx..x..x";
unsigned char _fa_nr = 7;
#define _FA_OK
#endif

// Fahrauftrag F8 - Blocked Passengers
#ifdef FA8
unsigned char _fa [] =
"xxxFxxxF..x..Fx..x..xFx.x.xFF..x..Fx...x.xx..x..xx.x...xx..x..xx..x..x";
unsigned char _fa_nr = 8;
#define _FA_OK
#endif
```

```
// Fahrauftrag F9 - Outdoor - Free Land
#ifdef FA9
unsigned char _fa [] =
"xFxxxFxF..x..Fx.x...xx.x...xx.x...xx.xxx.xx...x.xx...x.xF...x.Fx..x..x";
unsigned char _fa_nr = 9;
#define _FA_OK
#endif

// Fahrauftrag F10 - Long Distance - Endurance
#ifdef FA10
unsigned char _fa [] =
"xFxxxFxF..x..Fx.x...xx.x.xxxx.x...xx.xxx.xx...x.xxxx.x.xx...x.xx..x..x";
unsigned char _fa_nr = 10;
#define _FA_OK
#endif

// Fahrauftrag F11 - VIP
#ifdef FA11
unsigned char _fa [] =
"xxxFxxxx.....xx..x..xx.....xxx...xxx.....xx.....xx.....xx.....xx..x..x";
unsigned char _fa_nr = 11;
#define _FA_OK
#endif

// Fahrauftrag F12 - Lazy Passengers
#ifdef FA12
unsigned char _fa [] =
"xxxxxxxx..x..xx..x..xx..x..xx..x..xx..x..xF..x..FF..x..FF..x..Fx..x..x";
unsigned char _fa_nr = 12;
#define _FA_OK
#endif

#ifndef _FA_OK
#error Fehler: Kein Fahrauftrag, bspw. mit <<#define FA2>> definiert!
#endif
```

/*hello.c*/

```
#include <stub.h>
#include "aktionen.h"
#include "ringbuffer.h"
#define FA1
#include "fa.h"


char max;
char abarbeiten[30];
char fahrgast_pos=0;
// Hauptprogrammroutine

void AksenMain(void)
{
unsigned int START = 0;
unsigned char index = 0;
unsigned char klappe=0; // 0 ist auf 1 ist zu
unsigned long int aktivierung = 0;
char i = 0;
char checkout = 5;
char hilf;
```

```
led(3,1);

//Startposition auslesen anhand der Dipschalter
if(dip_pin(0)==0 && dip_pin(3)==0){
	lcd_cls();
	lcd_puts("kein Startposition vergeben");
	while(1);
}else if(dip_pin(0)==1){
	START = 68;
}else if (dip_pin(3)==1){
	START = 64;
}

/* initiales fuellen  */
breitensuche(START);

do{
/*auslesen des Fahrplans*/
checkout = fahrplan_auslesen(START);

if(checkout==0){
				lcd_cls();
		lcd_puts("0 - naechster fahrgast");
				lcd_setxy(1,1);
				lcd_ubyte(_fa_nr);
}
else if(checkout==-1){
	//Fahrgast vorhanden, kein Weg gefunden
	 lcd_cls();
	lcd_puts("-1 - kein weg");
		sleep(2000);
		//naechsten fahrgast suchen
		_fa[fahrgast_pos]='x';

		//abarbeiten null setzen
		for(i=0;i<30;i++){
			abarbeiten[i]=0; //oder '0'
		}

		//ringbuffer leeren
		do{
			hilf = get();
		}while(hilf!=0);

}else if(checkout==-2){
	//kein Fahrgast vorhanden
	lcd_cls();
    lcd_puts("-2 - kein fahrgast");
	while(1);
}
}while(checkout != 0);

//solange das licht an ist bleibe in schleife
while(analog(5)>200);


while(1) {
	lcd_cls();
	lcd_putchar(abarbeiten[index]);

	//passagieraufladen
	if(analog(0)>200 && klappe==0){
```

```
                aufnehmen();
                klappe=1;
                 stueck_zurueck(SPEED);
                 //warte 150ms
                 sleep(200);
                turnAround(SPEED);
                sleep(200);
                while(analog(7)<150);
                //ein zweites while, dass er nicht die 1. ausfahrt nimmt(rechts fährt)
anstelle um 180grad zu drehen
                // erst seit dem Anbau der Lichtschranke und des Greifers nötig
                sleep(200);
                while(analog(7)<150);
        }

        //aktivierung initialisiert variable mit einer bestimmten Wartezeit
        if(((analog(10)>150)||(analog(12)>150))&&aktivierung<akt_time()){
                sleep(80);
                check_fahrplan(abarbeiten[index]);
                aktivierung = akt_time() + DEAKTIV;
                index++;
                if(abarbeiten[index]==0){
                        //fahrgast abladen
                        klappe_auf();
                        klappe=0;
                        lcd_cls();
                        //umdrehen und gerade ausrichten
                        turnAround(SPEED);
                        sleep(1500);
                        while(analog(7)<150);
                        stop();
                        //stueck zurueck fuer ausgangsposition
                        stueck_zurueck(SPEED);
                        //warte 200ms
                        sleep(200);
                        stop();

                        lcd_cls();
                        lcd_puts("abgeladen, umgedreht");

                        //Stack leeren
                        //evtl nicht noetig
                        do{
                                hilf = get();
                        }while(hilf!=0);

                        //abarbeiten null setzen
                        for(i=0;i<30;i++){
                                abarbeiten[i]=0; //oder '0'
                        }

                        index=0;
                        do{

                                checkout = fahrplan_auslesen(START);

                                if(checkout==-2){
                                        lcd_cls();
                                        lcd_puts("-2 - kein fahrgast");
                                        lcd_setxy(1,1);
                                        lcd_puts("fertig nach ");
                                        lcd_ubyte(index);
```

```
                                        while(1);

                                }else if (checkout == -1){
                                        lcd_cls();
                                        lcd_puts("-1 - kein weg");

                                        //abarbeiten null setzen
                                        for(i=0;i<30;i++){
                                                abarbeiten[i]=0; //oder '0'
                                        }

                                        //ringbuffer leeren
                                        do{
                                                hilf = get();
                                        }while(hilf!=0);

                                }else{
                                        lcd_cls();
                                        lcd_puts("0 - naechster fahrgast");

                                }
                        }while(checkout!=0);


                }
        }

            //8 -->schwarz  7-->weiß
        if((analog(8)>150)&&(analog(14)<150)){
                korrigiere_rechts(10);

        //8-->weiß 7-->schwarz
        }else if((analog(14)<150)&&(analog(7)>150)){
                korrigiere_links(10);
            }

        //ansonsten
        else fahr_geradeaus(10);
}
}


/*
* ausgelagerte Hilfsmethoden
*
*/
int stueck_zurueck(int speed){
        motor_richtung(2, RUECKWAERTS);
     motor_pwm(2, speed);
     motor_richtung(3,VORWAERTS);
     motor_pwm(3,speed);
     return 0;
}

 int fahr_geradeaus(int speed){
     motor_richtung(2, VORWAERTS);
     motor_pwm(2, speed);
     motor_richtung(3,RUECKWAERTS);
     motor_pwm(3,speed);
     return 0;
}
```

```
int korrigiere_links(int speed){
    motor_richtung(2, VORWAERTS);
    motor_pwm(2, 0);
    motor_richtung(3,RUECKWAERTS);
    motor_pwm(3,speed);
    return 0;
}

int korrigiere_rechts(int speed){
    motor_richtung(3, RUECKWAERTS);
    motor_pwm(3, 0);
    motor_richtung(2,VORWAERTS);
    motor_pwm(2,speed);
    return 0;
}

int turnAround(int speed){
       motor_richtung(3, RUECKWAERTS);
    motor_pwm(3, speed);
    motor_richtung(2,RUECKWAERTS);
    motor_pwm(2, speed);
       return 0;
}

int stop(){
       motor_richtung(3, RUECKWAERTS);
    motor_pwm(3, 0);
    motor_richtung(2,RUECKWAERTS);
    motor_pwm(2,0);
       return 0;
}

int kreuzung_links(int speed){
       motor_richtung(3, RUECKWAERTS);
    motor_pwm(3, speed);
    motor_richtung(2,RUECKWAERTS);
    motor_pwm(2,speed);
       sleep(500);
       while(analog(14)<150);
       return 0;
}

int kreuzung_rechts(int speed){
       motor_richtung(3, VORWAERTS);
    motor_pwm(3, speed);
    motor_richtung(2,VORWAERTS);
    motor_pwm(2,speed);
       sleep(500);
       while(analog(14)<150);
       return 0;
}

void check_fahrplan(char p_fahrplan){

       switch(p_fahrplan){
          case 'l':
               // fahr an der naechsten kreuzung links
                                kreuzung_links(10);
               break;
          case 'r': // fahr an der naechsten kreuzung rechts
                                kreuzung_rechts(10);
               break;
```

```
          case 'g':  // fahr an der naechsten kreuzung geradeaus
                                  //fahr_geradeaus(10);
                break;
        }
}

void aufnehmen(){
       motor_richtung(1, 0);
       motor_pwm(1, 5);
       sleep(600);
       motor_pwm(1, 0);
}

void klappe_auf(){
       motor_richtung(1, 1);
    motor_pwm(1, 5);
       sleep(300);
       motor_pwm(1, 0);
}


char fahrplan_auslesen(int start){
    char c = 0;
    char current_pos = 0;
    //char fahrgast_pos = 0;
       char i = 0;
    char auftrag[16];
    //char abarbeiten[32];
       char position = 'n';

//**nächste Haltestelle ausfindig machen, ggf in Ringbuffer schreiben, sonst fail
    for (c=0;c<70;c++){
        if (_fa[c]=='F'){
            current_pos=c;
            fahrgast_pos=c;
            put(c);
            break;
        }
        //wenn
        if(c==69 && _fa[c]!='F')
            return -2;
    }

//**NachbarnSuche ausgehend von der CurrentPosition  -->
//*sofern die current_pos nicht null ist und somit kein Fahrgast vorhanden
//*verfolge den Pfade und gehe zurück bis zur 0
//*ACHTUNG! wenn Fahrgast vorhanden aber kein Weg zum Anfang gefunden -->PROBLEM
    if (current_pos!=0){
        //backtracking
        do{

            if (_fa[current_pos-1]< 'F' && _fa[current_pos]>_fa[current_pos-1]){
                current_pos=current_pos-1;
                put(current_pos);
                c++;
            }
            else if (_fa[current_pos+7]< 'F' && _fa[current_pos]>_fa[current_pos+7]){
                current_pos=current_pos+7;
                put(current_pos);
                c++;
            }
            else if (_fa[current_pos+1]< 'F' && _fa[current_pos]>_fa[current_pos+1]){
```

```
                current_pos=current_pos+1;
                put(current_pos);
                c++;
            }
            else if (_fa[current_pos-7]< 'F' && _fa[current_pos]>_fa[current_pos-7]){
                current_pos=current_pos-7;
                put(current_pos);
                c++;
            }
            //falls kein weg zum start führt wie z.B in _fa F3 (linke und rechte Seite
separat)
            else if ((_fa[current_pos-7]=='.' || _fa[current_pos-7]=='F'||
_fa[current_pos-7]=='x')&&(_fa[current_pos-1]=='.' || _fa[current_pos-1]=='F'||
_fa[current_pos-1]=='x')&&(_fa[current_pos+7]=='.' || _fa[current_pos+7]=='F'||
_fa[current_pos+7]=='x')&&(_fa[current_pos+1]=='.' || _fa[current_pos+1]=='F'||
_fa[current_pos+1]=='x')){
               _fa[fahrgast_pos]='x';
               return -1;
               }

        }
        while (current_pos!=start);
    }
    max = n;
//**umdrehen des auftrag Arrays
    for (i=max;i>0;i--)
        auftrag[n-1] = get();

//**umwandeln in Zahlen Anweisungskette
    for (i=0;i<max-1;i++)
        abarbeiten[i]=(auftrag[i]-auftrag[i+1])*-1;

//kette in characterAnweisung umwandeln
    for (i=0;i<max-1;i++)
    {
        switch (position)
        {
        case 'n':
            if (abarbeiten[i]==-1)
            {
                abarbeiten[i]='l';
                position='w';
            }
            if (abarbeiten[i]==-7)
            {
                abarbeiten[i]='g';
                position='n';
            }
            if (abarbeiten[i]==+1)
            {
                abarbeiten[i]='r';
                position='o';
            }

            break;
        case 's':
            if (abarbeiten[i]==-1)
            {
                abarbeiten[i]='r';
                position='w';
            }
            if (abarbeiten[i]==+7)
```

```
            {
                abarbeiten[i]='g';
                position='s';
            }
            if (abarbeiten[i]==+1)
            {
                abarbeiten[i]='l';
                position='o';
            }

            break;
        case 'w':
            if (abarbeiten[i]==-1)
            {
                abarbeiten[i]='g';
                position='w';
            }
            if (abarbeiten[i]==-7)
            {
                abarbeiten[i]='r';
                position='n';
            }
            if (abarbeiten[i]==+7)
            {
                abarbeiten[i]='l';
                position='s';
            }

            break;
        case 'o':
            if (abarbeiten[i]==+1)
            {
                abarbeiten[i]='g';
                position='o';
            }
            if (abarbeiten[i]==+7)
            {
                abarbeiten[i]='r';
                position='s';
            }
            if (abarbeiten[i]==-7)
            {
                abarbeiten[i]='l';
                position='n';
            }

        }
    }

//**fahrgasteinladen rückweg mitberechnen
//*anhängen von f an Zeichenkette fuer Fahrgast
//*nutzen des Ringbuffers um umgekehrte Zeichenketten ebenfalls nochmal anzuhaengen
       for (i=max-2;i>=0;i--)
        put(abarbeiten[i]);
    for (i=max-1;n>0;i++){
              abarbeiten[i]=get();
              if(abarbeiten[i]=='l') abarbeiten[i]='r';
              else if (abarbeiten[i] == 'r') abarbeiten[i]='l';
       }

//Fahrgast als x Setzen auf der Karte
_fa[fahrgast_pos]='x';
```

```
return 0;
}


void breitensuche(int start)
{
    char i = 0;
    short kosten = '0';
    short pos = start;
    char count = 0;
    char bount = 0;
       _fa[pos]=kosten;
    put(pos);
    while (pos!=0)
    {
        kosten++;
        count=0;

        do
        {
            pos=get();
            if (_fa[pos-1]=='.')
            {
                _fa[pos-1]=kosten;
                put(pos-1);
                count++;
            }
            if (_fa[pos-7]=='.')
            {
                _fa[pos-7]=kosten;
                put(pos-7);
                count++;
            }
            if (_fa[pos+1]=='.')
            {
                _fa[pos+1]=kosten;
                put(pos+1);
                count++;
            }
            if (_fa[pos+7]=='.')
            {
                _fa[pos+7]=kosten;
                put(pos+7);
                count++;
            }
            bount--;
        }
        while (bount>0);
        bount=0;
        kosten++;
        do
        {
            pos=get();
            count--;
            if (_fa[pos-1]=='.')
            {
                _fa[pos-1]=kosten;
                put(pos-1);
                bount++;
            }
            if (_fa[pos-7]=='.')
            {
```

```
                _fa[pos-7]=kosten;
                put(pos-7);
                bount++;
            }
            if (_fa[pos+1]=='.')
            {
                _fa[pos+1]=kosten;
                put(pos+1);
                bount++;
            }
            if (_fa[pos+7]=='.')
            {
                _fa[pos+7]=kosten;
                put(pos+7);
                bount++;
            }
        }
        while (count>0);
    }
}
```