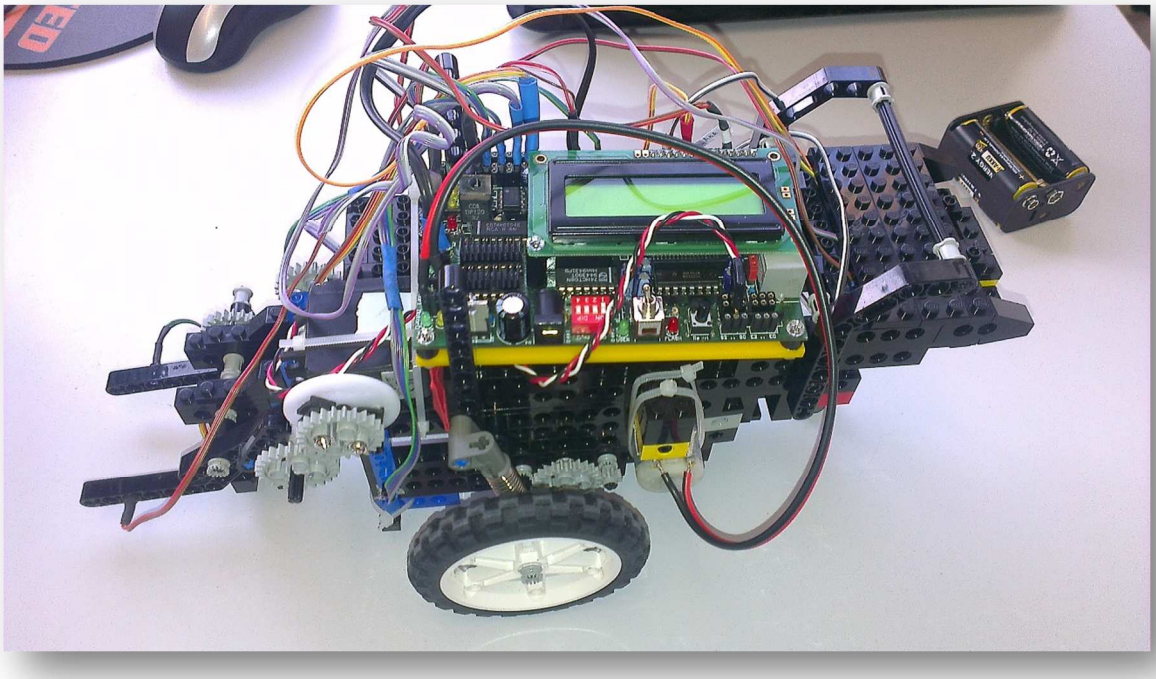
Dokumentation KI-Projekt Wintersemester 2014/2015

Ein Projekt
von
Alexandra Müller
und
Steven Stolze



Inhaltsverzeichnis

Projektaufgabe

Die Aufgabe bestand darin, einen Roboter zu konstruieren, der in der Lage ist einen vorgegebenen Fahrauftrag erfolgreich abzuarbeiten.

Um diese Aufgabe zu erfüllen standen verschiedene Sensoren, Aktoren sowie Akkumulatoren zur Verfügung. Diese wurden mit Hilfe von Legobausteinen um ein Aksen-Board platziert.

Als Karte nutzen wir ein Gitternetz mit Sperrungen in Form von Reifen mit roten Bällen.

Die Fahrgäste werden durch einen blauen Ball auf einem gelben Podest dargestellt.

Über den Roboter

Wir schreiben das Jahr 2014/2015. In diesem Jahr erwacht Robi, alias Phönix, alias Bumblebee zum Leben. Im Folgenden verbleiben wir bei der Benennung des Roboters mit dem Namen Robi, um Missverständnissen vorzubeugen.

In den ersten Wochen beschäftigten wir uns mit dem grundlegenden Aufbau des Roboters und den Basisfunktionalitäten des Aksen-Boards.

Schon in der zweiten Woche stellten wir fest, dass unser Roboter von der Statik her zu instabil aufgebaut war. Es erfolgte eine erste Terminierung unseres Schützlings. Als wir die Statik halbwegs verbessert hatten stellten wir fest, dass das Getriebe schlecht übersetzte und die Motoren nicht richtig positioniert waren. Somit musste unser Roboter sterben. Nach gefühlten tausendmaligen Zerstören und neu - Konstruieren war Robi endlich soweit und durfte auf die Welt losgelassen werden.

Der erste Meilenstein bestand darin, den Roboter geradeaus fahren zu lassen. Dies setzte Robi wunderbar um. Der nächste Schritt bestand darin, dass Robi auch die Fahrspur einhalten sollte. Dazu wurden zwei Optokoppler vorne angebracht. Leider reagierte Robi bei den ersten Fahrversuchen schlecht auf unsere Anweisungen. Nach kurzem Überlegen fanden wir auch den Fehler: die vorderen Optokoppler waren zu dicht an der Radachse positioniert und mussten noch weiter vorne angebracht werden.

Als Robi auch diesen Meilenstein erfolgreich absolviert hatte, war es nun an der Zeit, dass er auch eine einfache Aufgabe abarbeitete und an den Kreuzungen den jeweils übergebenen Befehl ausführte. Um diese Aufgabe zu bewältigen wurde vor den Rädern auf beide Seiten jeweils ein Optokoppler angebracht. Diese sind für die Erkennung der einzelnen Kreuzungen zuständig. Zusätzlich wurde ganz vorn ein weiterer Optokoppler angebracht, mit dem die Rotation an den Kreuzungspunkten koordiniert werden konnte.

Bearbeitet von Alexandra Müller & Steven Stolze

Nachdem Robi es zuerst nicht geschafft hatte eine stupide Aufgabe, wie eine simple Acht abzufahren, zu bewältigen, wären wir beinahe vom Glauben abgefallen. Auf der Suche nach Fehlern im Quellcode sind wir dann auf die glorreiche Idee gekommen, die Optokoppler zu testen. Diese erwiesen sich als defekt und mussten ausgetauscht werden. Nach diesem Tuning hatte es Robi endlich geschafft die Acht zu fahren ohne dass ihm schwindlig dabei wurde.

Es folgte die Fahrplanausführung anhand einer vorgegebenen Karte und des von uns entwickelten Algorithmus.

Nun schlossen sich mehrere schwarze Stunden an, die sich auf Wochen ausweiteten. Einerseits hatten einige Sensoren wieder Problem, andererseits gab es etliche semantische Fehler im Quellcode.

Dank unserer mentalen Stärke haben wir es letztlich geschafft den Code zum stabilen „laufen“ zu bewegen.

Nun erfolgte die Implementierung des Greifers samt dazugehöriger Logik. Aber auch hier gab es Probleme, wie defekte Infrarotempfänger oder die schwierige Befestigung des Servomotors.

In Vergessenheit geraten…

Nachdem wir das Gehäuse noch etwas weiter stabilisiert hatten, kümmerten wir uns um unsere versäumten Aufgaben.

Es gibt zwei mögliche Startpositionen (A und B). Von welcher Position der Roboter startet ist mit Hilfe der Dip-Schalter geregelt. Soll der Punkt A als Startpunkt verwendet werden, so muss der Dip-Schalter 0 auf 1 stehen. Für den Punkt B als initialer Punkt muss der Dip-Schalter 3 auf 1 stehen. Die restlichen Dip-Schalter befinden sich in der Position 0.
Gewünscht war eine Art Ampelsystem, um den Status der Fahrt-Planung darzustellen. Zu Beginn leuchtete eine rote Lampe gefolgt von einer gelben Lampe, die besagt, dass die Fahrt-Planung in Gange ist. Konnte ein Fahrplan erstellt werden leuchten eine blaue Lampe und eine rote Lampe, da keine grüne Lampe mehr zu finden war.
Sobald die Startlampe angeht, soll unser Robi in eine Art Stromsparmodus versetzt und die Lampen ausgeschaltet.

Spezielle Funktionen, wie dem Gegner die Fahrgäste abspenstig machen oder ~~die~~ eine von außen aktivierbare Fernsteuerung mussten wir nun leider ausbauen, da dies sich dies bei genauerem lesen der Wettbewerbs-Regeln als verboten erwies.

Lösungsweg

Wie oben schon angemerkt, konnten wir die Fernsteuerung unseres Robis nicht verwenden. Auch von der Vorgabe eines Lösungsweges zu jedem gegebenen Fahrplan mussten wir uns verabschieden, da Robi diese immer wieder vergaß.

Also spielten wir „Steinbeißer“ und entwickelten eine ganz neue Strategie.
Nachdem Robi dank viel Überredungskunst auf dem Gitternetz zurechtkam, begannen wir uns Gedanken zu machen, um einen mehr oder weniger optimalen Weg für Robi zu den jeweiligen Fahrgästen und zurück zu planen.
Zunächst erschien es uns sinnvoll die Karte mit Kosten zu belegen. Jeder „Schritt“ kostete also eine Einheit. Berücksichtigt wurden hier nur erreichbare Knoten (Kreuzungen).
Als weiterer Schritt wurde der „günstigste“ Weg zu einem jeweiligen Fahrgast gesucht und eine Sequenz von Anweisungen erstellt, die unser Robi verstanden hatte . Danach mussten wir die einzelnen Sequenzen nur noch aneinander reihen und „voila“ – es funktionierte.

Die Arbeitsaufteilung mit unserem Robi erwies sich immer wieder als nicht so einfach. Er ist ein sehr widerspenstiger Roboter und nur sehr selten war es sicher, sich ihm ohne Verstärkung zu nähern. Normalerweise lag die Aufteilung darin, dass einer ihn festhielt und der andere ihm den … Trichter ansetzte, um ihm die gemeinsam erarbeiteten Erkenntnisse einzuarbeiten.

Wie gesagt, erwies sich Robi als sehr widerspenstig und auch kreativ in seiner Art, sich gegen unsere Planungen und Vorgaben zur Wehr zu setzen.
An manchen Tagen ignorierte er uns völlig und an anderen Tagen bombardierte er uns mit Fremdsprachen.
Es hat uns viel unserer Überredungskunst gekostet ihn davon zu überzeugen, dass wir es besser wissen.

Wettkampf

Oder der Preis ist heiß…

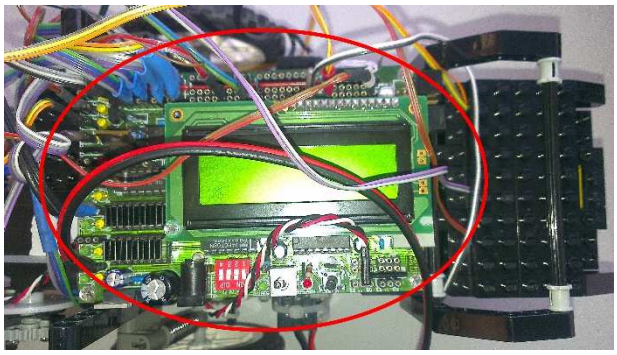
Trotz ausgeklügelter Technik schaffte Robi es nicht Platz eins zu sichern.

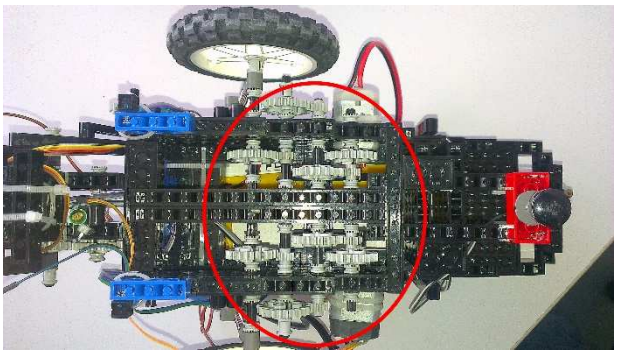
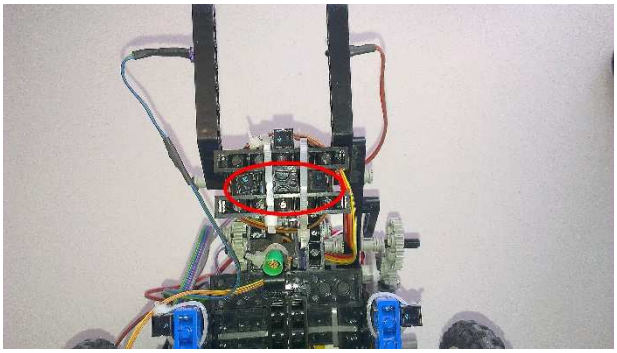
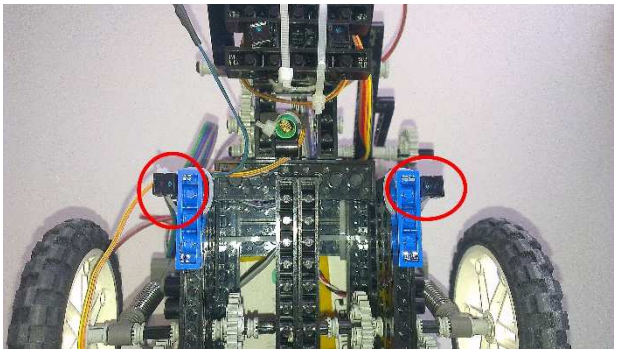
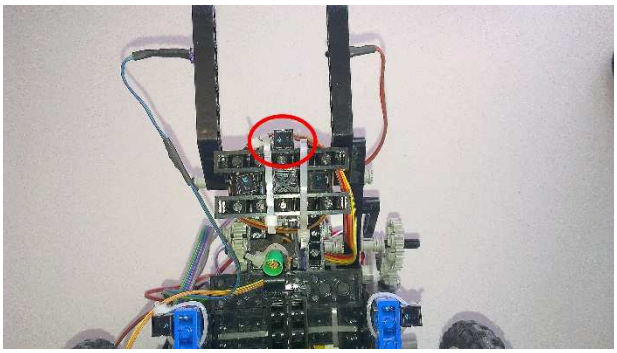
Dennoch hat sein starker Wille ausgereicht, das Finale zu erreichen.

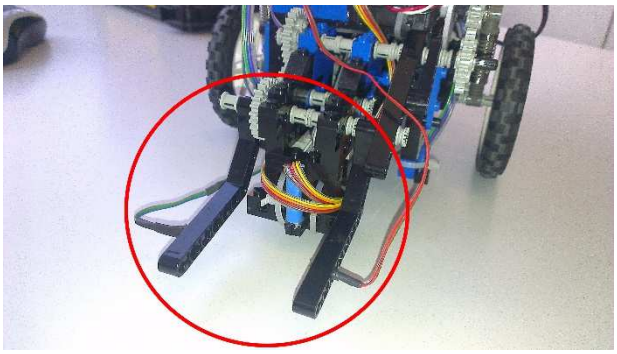
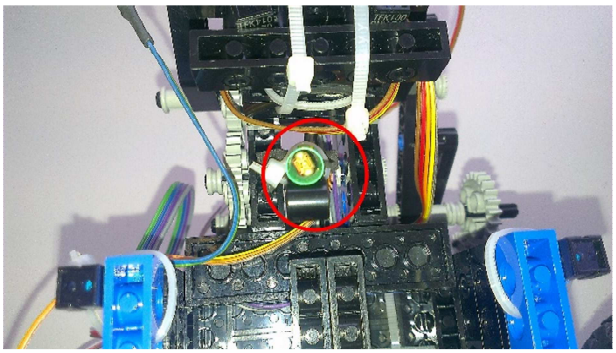
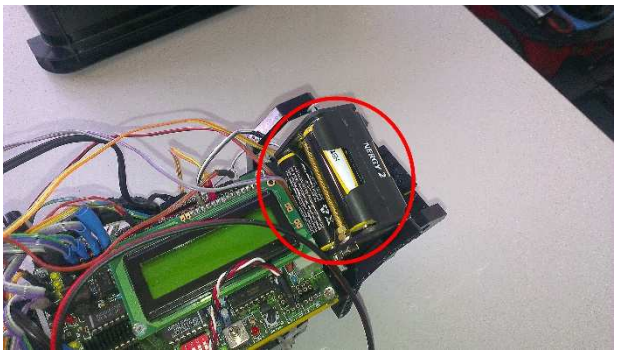
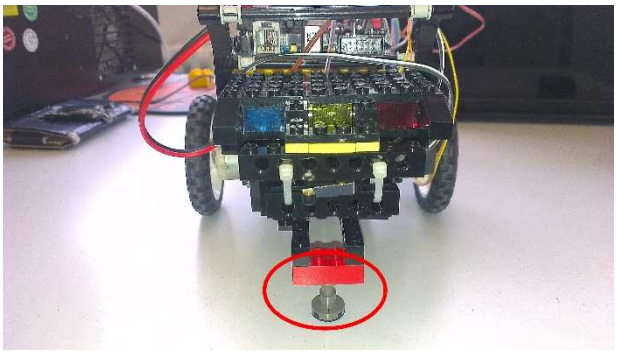
Es war ein schöner und spannender Wettkampf.

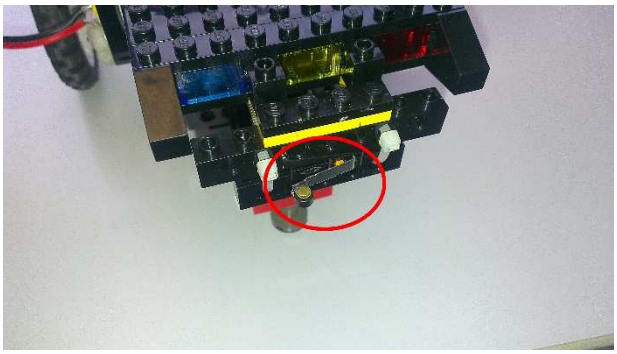
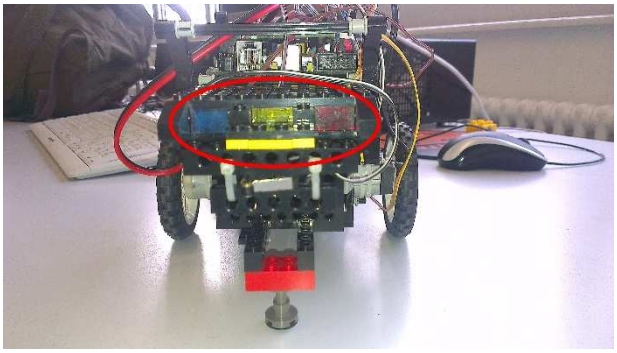
Sollten es zukünftige Wettkämpfe erlauben, Waffen oder andere Hilfsmittel einzusetzen, dann würde Robi über ein Comeback nachdenken.

Hardware / Robi unter der Lupe

	Das Aksen-Board, welches das Gehirn unseres tapferen kleinen Robis darstellt, dient als Interface zwischen Hardware und Software. Ohne dies wäre Robi nicht „überlebensfähig.

	Robi besitzt einen differentiellen Antrieb mit einer Untersetzung von 1:81.
	Diese vorderen Optokoppler dienen der Spureinhaltung.
	Die seitlichen Optokoppler sind für die Erkennung der Kreuzungen zuständig.
	Der vordere Optokoppler sorgt für den Stopp der Rotation an Kreuzungspunkten.

	Der Greifer besitzt einen Infrarotsender und –empfänger, welche eine „Lichtschranke“ für den Kontakt mit dem Fahrgast realisieren sollen.
	Vorne befindet sich ein Photosensor für die Überprüfung des Startsignals.
	Der Akkumulator dient als „Lebensenergie“ für Robi.
	Damit Robi sein Gleichgewicht während der Fahrt halten kann wurde ihm hinten eine „Stütze“ angebracht.

	Damit Robi beim Abladen eines Fahrgastes die erste Kreuzung wieder mit berücksichtigt, muss er nach einer 180° Drehung zurückfahren bis der im Bild sichtbare Taster die untere Begrenzung berührt. Dann darf er seinen nächsten Fahrauftrag abarbeiten.
	Die Lampen, kein Relikt aus „Knight Rider“, sondern stellen Robis internen Status vor dem Start visuell dar.

Software / Codeschnipsel – Erklärung

Für eine optimale Strukturierung wurde der Code in mehrere Funktionen und Dateien aufgeteilt.

Folgende Dateien entstanden im Verlaufe des Projektes:

phoenix.c

actions.c

actions.h

agenda.c

agenda.h

- In der agenda.c sind die Pufferfunktionen untergebracht.
- Die actions.c stellte die rudimentären Funktionalitäten von Robi dar.
- Zusätzlich sind in der actions.h Präprozessordirektiven und Defines untergebracht.
- Die phoenix.c stellt die Main-Datei dar, welche die ganze Logik bereitstellte.

Folgende Funktionen wurden von uns implementiert:

- void AksenMain();
- void navigate();
- void crossroads();

- unsigned char fillPathArrays();
- void initAllPaths();
- void sortGuestPositions();
- unsigned char breitenSuche(unsigned char startPunkt);
- char checkStartPoint();
- void checkStart();
- void drive();
- void stop();
- void back();
- void back2();
- void steerLeft();
- void steerRight();
- void spin_around();
- void biegeLinksAb();
- void biegeRechtsAb();
- void closeGripper();
- void openGripper();
- void pushKnoten (unsigned char aktKnoten, const unsigned char MAX);
- unsigned char getKnoten(const unsigned char MAX);
- void pushGuestPosition(unsigned char position);
- unsigned char getGuestPosition();
- void initKarte(const unsigned char MAX);

Die Standardaktion von Robi ist drive(). Um dies zu ermöglichen müssen beide Motoren mit der gleichen Geschwindigkeit drehen:

```
void drive()
{
        motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_LINKS, SPEED);
        motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_RECHTS, SPEED);
}
```

Für die Spureinhaltung müssen die beiden vorderen Optokoppler überprüft werden. Ein Wert größer 150 entsprach Schwarz:

```
…
else if(OPTOKOPPLER_VORNE_LINKS > 150)
        steerLeft();
else if(OPTOKOPPLER_VORNE_RECHTS > 150)
        steerRight();
```

Damit die Spur korrigiert wird, muss sich ein Rad langsamer drehen als das andere Rad:

```
void steerLeft()
{
        motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_LINKS, MINSPEED);
        motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_RECHTS, SPEED);
}

void steerRight()
{
        motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_LINKS, SPEED);
        motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
        motor_pwm(MOTOR_RECHTS, MINSPEED);
}
```

Die Abarbeitung der Befehle an den Kreuzungen erfolgt dann, wenn einer der seitlichen Optokoppler auf schwarz steht, also einen Wert größer 150 hat. Damit eine Kreuzung nicht mehrfach erkannt wird und dort mehrere Befehle verarbeitet werden, musste zusätzlich ein Delay implementiert werden, welcher an jeder Kreuzung mit überprüft wurde:

```
if((OPTOKOPPLER_MITTE_LINKS > 150 || OPTOKOPPLER_MITTE_RECHTS > 150)
&& (delay <= akt_time()))
```

Sobald ein Befehl aus dem Pfadarray ausgelesen ist, wird die entsprechende Aktion ausgeführt:

```
if(mIndex < MAX2)
{

        switch(guestPath[pathCounter][mIndex])
        {
                case 'r':       lcd_puts("goRight");
                                biegeRechtsAb();
                                break;
                case 'l':       lcd_puts("goLeft");
                                biegeLinksAb();
                                break;
                case 'g':       lcd_puts("goStraight");
                                drive();
                                break;
                case 's':       lcd_puts("dance");
                                spin_around();
                                drive();
                                while(OPTOKOPPLER_MITTE_LINKS > 18 ||
OPTOKOPPLER_MITTE_RECHTS > 18);
                                break;
        }
}
```

Falls sich Robi dem Startpunkt von links bzw. rechts genähert hat, so soll er sich vor dem Abladen des Fahrgastes in die entgegengesetzte Richtung drehen:

```
if(guestPath[pathCounter][mIndex+1] == 'c')
{
        char difference;
        difference = guestPath[pathCounter][mIndex-1] - guestPath[pathCounter][mIndex];
        switch(difference)
        {
                case -1:        lcd_puts("goRight");
                                biegeRechtsAb();
                                break;
                case 1:         lcd_puts("goLeft");
                                biegeLinksAb();
                                break;
        }
…
}
```

Nach dem Robi sein Fahrgast abgesetzt hat, soll er sich um 180° drehen und solange zurückfahren, bis der Taster die hintere Begrenzung berührt. Somit wird sichergestellt, dass die Startkreuzung bei der Planabarbeitung mitberücksichtigt wird:

```
…
spin_around();
while(TASTER == 1)
        back2();
stop();
…
```

Mit Hilfe der „Laserschranke“ kann der Fahrgast zuverlässig aufgenommen werden, dabei wird vorher mit der Variable isGripperOpen der Zustand des Greifers überprüft:

```
if((LASERSCHRANKE > 200) && (isGripperOpen == 1))
{
	stop();
	closeGripper();
	isGripperOpen = 0;
	lcd_cls();
	lcd_setxy(0,0);
	lcd_puts("Gast aufgeladen");
	sleep(100);
	back();
}
```

Als Suchverfahren wird die Breitensuche verwendet, welche den aktuellen Startpunkt übergeben bekommt (entweder 64 oder 68) und anhand dessen die Kosten für diese Position und den Nachbarpositionen vergeben werden. Die Breitensuche erfolgt einmalig am Anfang vor dem Start. Das Auslesen erfolgt öfters, abhängig von der Anzahl der Fahrgäste. Damit die Kosten auch gerecht verteilt werden, sind innerhalb der While-Schleife zwei do-While-Schleifen implementiert:

```
unsigned char breitenSuche(unsigned char startPunkt)
{
	unsigned char kosten = 0;
	unsigned char status = 1;
	char counter1 = 0;
	char counter2 = 0;
	unsigned char aktKnoten = startPunkt;
	initKarte(MAX);
	pushKnoten(aktKnoten, MAX);
	_fa[aktKnoten] = kosten;

	while(aktKnoten != 99)
	{
```

```
kosten++;
counter1 = 0;

do{
	aktKnoten = getKnoten(MAX);

	if(_fa[aktKnoten-1] == '.' || _fa[aktKnoten-1] == 'F')
	{
		if(_fa[aktKnoten-1] == 'F')
			pushGuestPosition(aktKnoten-1); //Position merken

		pushKnoten(aktKnoten-1, MAX);
		_fa[aktKnoten-1] = kosten;
		counter1++;
	}
	if(_fa[aktKnoten-7] =='.' || _fa[aktKnoten-7] == 'F')
	{
		if(_fa[aktKnoten-7] == 'F')
			pushGuestPosition(aktKnoten-7); //Position merken

		pushKnoten(aktKnoten-7, MAX);
		_fa[aktKnoten-7] = kosten;
		counter1++;
	}
	if(_fa[aktKnoten+1] == '.' || _fa[aktKnoten+1] == 'F')
	{
		if(_fa[aktKnoten+1] == 'F')
			pushGuestPosition(aktKnoten+1); //Position merken

		pushKnoten(aktKnoten+1, MAX);
		_fa[aktKnoten+1] = kosten;
```

```
                counter1++;
        }
        if(_fa[aktKnoten+7] == '.' || _fa[aktKnoten+7] == 'F')
        {
                if(_fa[aktKnoten+7] == 'F')
                        pushGuestPosition(aktKnoten+7); //Position merken

                pushKnoten(aktKnoten+7, MAX);
                _fa[aktKnoten+7] = kosten;
                counter1++;
        }
        counter2--;
}while(counter2 > 0);

kosten++;
counter2 = 0;

do{
        counter1--;
        aktKnoten = getKnoten(MAX);

        if(_fa[aktKnoten-1] == '.' || _fa[aktKnoten-1] == 'F')
        {
                if(_fa[aktKnoten-1] == 'F')
                        pushGuestPosition(aktKnoten-1); //Position merken

                pushKnoten(aktKnoten-1, MAX);
                _fa[aktKnoten-1] = kosten;
                counter2++;
        }
        if(_fa[aktKnoten-7] =='.' || _fa[aktKnoten-7] == 'F')
```

```
                        {
                                if(_fa[aktKnoten-7] == 'F')
                                        pushGuestPosition(aktKnoten-7); //Position merken

                                pushKnoten(aktKnoten-7, MAX);
                                _fa[aktKnoten-7] = kosten;
                                counter2++;
                        }
                        if(_fa[aktKnoten+1] == '.' || _fa[aktKnoten+1] == 'F')
                        {
                                if(_fa[aktKnoten+1] == 'F')
                                        pushGuestPosition(aktKnoten+1); //Position merken

                                pushKnoten(aktKnoten+1, MAX);
                                _fa[aktKnoten+1] = kosten;
                                counter2++;
                        }
                        if(_fa[aktKnoten+7] == '.' || _fa[aktKnoten+7] == 'F')
                        {
                                if(_fa[aktKnoten+7] == 'F')
                                        pushGuestPosition(aktKnoten+7); //Position merken

                                pushKnoten(aktKnoten+7, MAX);
                                _fa[aktKnoten+7] = kosten;
                                counter2++;
                        }
                }while(counter1 > 0);
        }
        return 0;
}
```

Die Positionen der Gäste wurden nach Höhe ihrer Kosten in der Funktion sortGuestPositions() sortiert:

```
void sortGuestPositions()
{
	unsigned char i, j, temp;
	unsigned char guestArrayLength = 3;

	for(i = 0; i < 3; i++)
	{
		guestArray[i] = getGuestPosition();
		if(guestArray[i] != 99)
			guestCounter++;
	}

	for(i = 0; i < guestArrayLength -1; i++)
	{
		for(j = 0; j < guestArrayLength - i - 1; j++)
		{
			if(_FA[guestArray[j]] > _FA[guestArray[j+1]])
			{
				temp = guestArray[j];
				guestArray[j] = guestArray[j+1];
				guestArray[j+1] = temp;
			}
		}
	}
}
```

Mit initAllPaths() wurden alle Gastpfade am Anfang mit 99 initialisiert. Dies sollte die Pfade davor schützen keine „Müllwerte“ vorzuhalten:

```
void initAllPaths()
{
	unsigned char i,j;

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < MAX2; j++)
		{
			guestPath[i][j] = 99;
		}
	}
}
```

Die komplexeste aller Funktionen ist die fillPathArrays(). Diese füllt und konvertiert die Gastpfade, damit am Ende maximal drei vollständige und korrekte Gastpfade konstruiert werden. Zuerst erfolgt die Befüllung des Pfades mit den Knoten, welche die geringsten Kostenwerte vom Fahrgast zum Start aufweisen. Danach wird der nächste Richtungsbefehl unter der Berücksichtigung der aktuellen Blickrichtung gespeichert. Um noch nicht abgearbeitete Pfadpositionen während dieser Speicherung vor dem Überschreiben zu bewahren, wird ein temporärer Pfad verwendet. Nachdem der Hinweg erfolgreich abgespeichert wurde, erfolgt die Speicherung des Rückweges. Aus Platzgründen wird hier auf den Code verzichtet. Daher verweisen wir auf den Anhang:

```
unsigned char fillPathArrays(unsigned char startPoint, unsigned char index)
{
	…

}
```

Fazit

Es war ein langer und steiniger Weg.

Wir mussten während dieser Zeit sehr viel Leid und Schmerz ertragen. Aber wir konnten auch Erfahrung mitnehmen.

Das Projekt lehrte uns Teamarbeit und Problemlösung.

Wir möchten uns herzlich bei Hr. Börsch und Prof. Heinsohn bedanken, welche uns mit Rat und Tat zur Seite standen.

Anhang - Kompletter Sourcecode

Der Sourcecode wurde als zip-Datei angefügt.

```
1  #ifndef AGENDA_H
2  #define AGENDA_H
3
4  /*************** FUNKTIONSDEKLARATIONEN ****************/
5  void pushKnoten(unsigned char aktKnoten, const unsigned char MAX);
6  unsigned char getKnoten(const unsigned char MAX);
7  void pushGuestPosition(unsigned char position);
8  unsigned char getGuestPosition();
9  void initKarte(const unsigned char MAX);
10 /*************** FUNKTIONSDEKLARATIONEN ****************/
11
12 #endif
13
```

```c
//written by alex & steven

#include <stdio.h>
#include "agenda.h"


/*****************************************************************************/
/******************** Ab hier erfolgt die Implementierung ********************/
/*****************************************************************************/
//GLOBALE VARIABLEN
int n = 0;
char index1 = 0;
char index2 = 0;
char k = 0;

//Kostenkarte
unsigned char nachbarnKarte[30];

//Fahrgastpositionen
unsigned char guestPositions[3] = {99,99,99};

void initKarte(const unsigned char MAX)
{
    int j;
    for(j = 0; j < MAX; j++)
        {
            nachbarnKarte[j] = 99;
        }
}

/**************AGENDA FUER KOSTENKARTE**************/
void pushKnoten(unsigned char aktKnoten, const unsigned char MAX)
{
    if(n < MAX)
    {
        nachbarnKarte[index1] = aktKnoten;
        index1 = ((index1 + 1) == MAX ? 0 : index1 + 1);
        n++;
    }
}

unsigned char getKnoten(const unsigned char MAX)
{
    char temp;
    if(n > 0)
    {
        temp = index2;
        index2 = ((index2 + 1) == MAX ? 0 : index2 + 1);
        n--;
        return nachbarnKarte[temp];
    }
    else
        return 99;
}
/**************AGENDA FUER KOSTENKARTE**************/

```

```
57
58  /**************AGENDA FUER Fahrgastpositionen**************/
59  //Gibt solange eine Position zurück bis k > 0 ist, ansonsten 0
60  unsigned char getGuestPosition()
61  {
62      if(k > 0)
63      {
64          k--;
65          return guestPositions[k];
66      }
67      return 99;
68  }
69
70  void pushGuestPosition(unsigned char position)
71  {
72      if(k < 3)
73      {
74          guestPositions[k] = position;
75          k++;
76      }
77  }
78  /**************AGENDA FUER Fahrgastpositionen**************/
79
80
81
```

```
1  #ifndef ACTIONS_H
2  #define ACTIONS_H
3
4  /*************** INCLUDES ***************/
5  #include <stdio.h>
6  #include <regc515c.h>
7  #include "stub.h"
8  /*************** INCLUDES ***************/
9
10 /*************** DIREKTIVEN **************/
11 #define PHOTOSENSOR (analog(14))
12 #define SCHWARZ 100
13 #define LICHT 50
14 #define MOTOR_LINKS 0                          // Motor linkes Rad, Port 0 rot vorne
15 #define MOTOR_RECHTS 2                         // Motor rechtes Rad, Port 2
16 #define MOTOR_RICHTUNG_VORWAERTS 1
17 #define MOTOR_RICHTUNG_RUECKWAERTS 0
18 #define SPEED 10                                   // Standardgeschwindigkeit
19 #define MEDSPEED 2
20 #define MINSPEED 3                             // Korrekturgeschwindigkeit
21 #define STOP 0                                 // Geschwindigkeit auf 0 setzen
22 #define Startpoint_A 64                        // Startpunkt A ist der 64.te Punkt auf dem Gitternetz
23 #define Startpoint_B 68                        // Startpunkt B ist der 68.te Punkt auf dem Gitternetz
24 #define OPTOKOPPLER_MITTE_LINKS (analog(4))
25 #define OPTOKOPPLER_MITTE_RECHTS (analog(6))
26 #define OPTOKOPPLER_VORNE_LINKS (analog(7))
27 #define OPTOKOPPLER_VORNE_RECHTS (analog(2))
28 #define OPTOKOPPLER_VORNE (analog(0))
29 #define TASTER (digital_in(0))
30 #define LASERSCHRANKE (analog(8))
31 #define GREIFER_OFFEN 0
32 #define GREIFER_GESCHLOSSEN 1
33 #define SERVO 1
34 #define WINKEL60 60
35 #define WINKEL0 0
36 #define LAMPE_ROT 1
37 #define LAMPE_GELB 0
38 #define LAMPE_BLAU 2
39
40 #define FA6                                  //Fahrplan definieren, siehe fa.h
41
42
43 /*************** DIREKTIVEN **************/
44
45
46
47 /****** GITTERNETZ ******/           // Wobei die Ecken 0, 6, 63, 69 nicht beruecksichtigt werden
48 /*  0  1  2  3  4  5  6 */
49 /*  7  8  9 10 11 12 13 */
50 /* 14 15 16 17 18 19 20 */
51 /* 21 22 23 24 25 26 27 */
52 /* 28 29 30 31 32 33 34 */
```

```
53 /* 35 36 37 38 39 40 41 */
54 /* 42 43 44 45 46 47 48 */
55 /* 49 50 51 52 53 54 55 */
56 /* 56 57 58 59 60 61 62 */
57 /* 63  A 65 66 67  B 69 */
58 /****** GITTERNETZ ******/
59
60 /*************** FUNKTIONSDEKLARATIONEN ***************/
61 // Startpunkt pruefen - Punkt 64 oder Punkt 68?
62 char checkStartPoint();
63
64 //Start pruefen - Licht aus?
65 void checkStart();
66
67 // Beschleunigen Vorwaerts, Rueckwaerts, Stoppen
68 void drive();
69 void stop();
70 void back();
71 void back2();
72
73 // Spurassistent - Links und Rechts steuern
74 void steerLeft();
75 void steerRight();
76
77 // Richtungswechsel inkl. 180 Grad Drehung
78 void spin_around();
79 void biegeLinksAb();
80 void biegeRechtsAb();
81
82 // Kundenservice
83 void closeGripper();
84 void openGripper();
85 /*************** FUNKTIONSDEKLARATIONEN ***************/
86
87 #endif
88
```

```c
/*************** INCLUDES ***************/
#include "actions.h"
/*************** INCLUDES ***************/

/*****************************************************************************/
/******************** Ab hier erfolgt die Implementierung ********************/
/*****************************************************************************/
char checkStartPoint()
{
    //LOKALE VARIABLE
    char startPoint = 0;

    while(startPoint == 0)
    {
        if(dip_pin(0) == 1)
        {
            startPoint = 64;
            lcd_cls();
            lcd_setxy(0,0);
            lcd_ubyte(startPoint);
        }
        else if(dip_pin(3) == 1)
        {
            startPoint = 68;
            lcd_cls();
            lcd_setxy(0,0);
            lcd_ubyte(startPoint);
        }
        else
        {
            lcd_cls();
            lcd_setxy(0,0);
            lcd_puts("Kein Startpunkt");
            lcd_setxy(1,0);
            lcd_puts("vergeben");
            sleep(500);
        }
    }
    return startPoint;
}

void checkStart()
{
    //Wenn Photosensor bemerkt, dass Startlampe aus ist, dann beende Endlosschleife while(1);
    while(PHOTOSENSOR > LICHT)
    {
        led(LAMPE_ROT,1);
        led(LAMPE_BLAU,0);
        sleep(100);
        led(LAMPE_ROT,0);
        led(LAMPE_BLAU,1);
        sleep(100);
    }
    led(LAMPE_BLAU,0);
}
```

```
56
57  void drive()
58  {
59      //Linker und rechter Motor muessen sich entgegengesetzt drehen -> Polung
60      motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
61      motor_pwm(MOTOR_LINKS, SPEED);
62      motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
63      motor_pwm(MOTOR_RECHTS, SPEED);
64  }
65
66  void stop()
67  {
68      //linkes und rechtes Rad stoppen
69      motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
70      motor_pwm(MOTOR_LINKS, STOP);
71      motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
72      motor_pwm(MOTOR_RECHTS, STOP);
73  }
74
75  void back()
76  {
77      //Wie drive nur umgekehrt
78      motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_RUECKWAERTS);
79      motor_pwm(MOTOR_LINKS, SPEED);
80      motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_RUECKWAERTS);
81      motor_pwm(MOTOR_RECHTS, SPEED);
82      while(OPTOKOPPLER_MITTE_LINKS < 150 && OPTOKOPPLER_MITTE_RECHTS < 150);
83  }
84  void back2()
85  {
86      //Wie drive nur umgekehrt
87      motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_RUECKWAERTS);
88      motor_pwm(MOTOR_LINKS, SPEED);
89      motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_RUECKWAERTS);
90      motor_pwm(MOTOR_RECHTS, SPEED);
91  }
92
93  void steerLeft()
94  {
95      //linkes Rad abbremsen, rechtes Rad drehen
96      motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
97      motor_pwm(MOTOR_LINKS, MINSPEED);
98      motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
99      motor_pwm(MOTOR_RECHTS, SPEED);
100 }
101
102 void steerRight()
103 {
104     //linkes Rad drehen, rechtes Rad abbremsen
105     motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
106     motor_pwm(MOTOR_LINKS, SPEED);
107     motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
108     motor_pwm(MOTOR_RECHTS, MINSPEED);
109 }
110
111 void biegeRechtsAb()
```

```
112 {
113     //Raeder drehen sich entgegengesetzt mit gleicher Staerke
114     motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_VORWAERTS);
115     motor_pwm(MOTOR_LINKS, SPEED);
116     motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_RUECKWAERTS);
117     motor_pwm(MOTOR_RECHTS, MEDSPEED);
118     //while(OPTOKOPPLER_VORNE < SCHWARZ);
119     //Mehr Drehgeschwindigkeit - aber Fehler abfangen
120     //motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
121     //motor_pwm(MOTOR_RECHTS, STOP);
122     sleep(350);         //Damit hintere Opto von schwarz runterkommt
123     while(OPTOKOPPLER_VORNE < SCHWARZ);         //solange weiss
124 }
125
126 void biegeLinksAb()
127 {
128     //Raeder drehen sich entgegengesetzt mit gleicher Staerke
129     motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_RUECKWAERTS);
130     motor_pwm(MOTOR_LINKS, MEDSPEED);
131     motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
132     motor_pwm(MOTOR_RECHTS, SPEED);
133     //Damit hintere Opto von schwarz runterkommt
134     sleep(350);
135     while(OPTOKOPPLER_VORNE < SCHWARZ);
136 }
137
138 void spin_around()
139 {
140     //linkes Rad und rechtes Rad in gleiche Richtung drehen
141     motor_richtung(MOTOR_LINKS, MOTOR_RICHTUNG_RUECKWAERTS);
142     motor_pwm(MOTOR_LINKS, SPEED);
143     motor_richtung(MOTOR_RECHTS, MOTOR_RICHTUNG_VORWAERTS);
144     motor_pwm(MOTOR_RECHTS, SPEED);
145     sleep(200);
146     while((OPTOKOPPLER_VORNE < SCHWARZ));
147     sleep(200);
148     while((OPTOKOPPLER_VORNE < SCHWARZ));
149 }
150
151 void closeGripper()
152 {
153     char i;
154     for(i = WINKEL0; i <= WINKEL60; i++)
155         servo_arc(SERVO,i);
156 }
157
158 void openGripper()
159 {
160     char i;
161     for(i = WINKEL60; i >= WINKEL0; i--)
162         servo_arc(SERVO,i);
163 }
164
165
```

```
1  /*********************************/
2  /* project: phoenix v2_0         */
3  /* authors: alexandra, steven    */
4  /* date:    02.10.2014           */
5  /*********************************/
6
7  /*************** INCLUDES ***************/
8  //Standard Inlcude Dateien
9  #include <regc515c.h>
10
11 //Aksen-Bibliothek
12 #include <stub.h>
13
14 //Eigene Inludes
15 #include "actions.h"
16 #include "agenda.h"
17 #include "fa.h"
18 /*************** INCLUDES ***************/
19
20 /*************** FUNKTIONSDEKLARATIONEN ***************/
21 void navigate();                                        // Kontrolle der Fahrt - ↪
   > Spur, Kreuzung erkennen
22 void crossroads();                                      // Kreuzungbefehle      ↪
   abarbeiten, Gast abladen
23
24 //Pfaderstellung
25 unsigned char fillPathArrays(unsigned char startPoint, unsigned char i);
26
27 //Pfade mit 99 initialisieren
28 void initAllPaths();
29
30 //Die max. 3 Fahrgastpositionen hinsichtlich ihrer Kostenwerte sortieren
31 void sortGuestPositions();
32
33 //Suchverfahren, um die gegebene Karte in eine Kostenkarte umzuwandeln und die   ↪
   Fahrgastpositionen zu speichern
34 unsigned char breitenSuche(unsigned char startPunkt);
35 /*************** FUNKTIONSDEKLARATIONEN ***************/
36
37 /*GLOBALE VARIABLEN*/
38 //Variable, welche entscheidet wann die Kreuzung wieder erkannt wird
39 unsigned long delay = 0;
40 //Offset, der dazu dient, dass eine Kreuzung nicht mehrfach erkannt wird
41 const long offset = 400;
42 //Index zum Abarbeiten der einzelnen Befehle eines Endpfades in der Funktion     ↪
   crossroads()
43 unsigned char mIndex = 0;
44
45 //Arraygrenze fuer die Nachbarpositionen
46 const unsigned char MAX = 30;
47
48 //Arraygrenze fuer die Pfade
49 const unsigned char MAX2 = 33;
50
51 //Sortiertes Fahrgastarray
52 unsigned char guestArray[3] = {99,99,99};
```

```
53
54  //Fahrgastzaehler
55  unsigned char guestCounter = 0;
56
57  //Pfadindex der jeweils 3 Pfade
58  unsigned char pathCounter = 0;
59
60  //Fahrgastpfade
61  unsigned char guestPath[3][33];
62
63  //Greiferstatus
64  char isGripperOpen = 1;     //1 == offen; 0 == geschlossen
65
66  //Missionsstatus - Pfad abgearbeitet?
67  char isMissionComplete = 0; //Pfad abgearbeitet 1==ja 0==nein
68
69  /*MAINFUNKTION*/
70  void AksenMain(void)
71  {
72
73      /*LOKALE VARIABLEN*/
74      //Startpunkt des Roboters
75      char startPoint = 0;
76      //guestPathIndex
77      unsigned char index;
78
79      led(LAMPE_ROT,1);
80      sleep(1000);
81
82  /*****************************************************************************
    ******************/
83  /************************************ STARTPUNKT PRUEFEN
     ************************************/
84  /*****************************************************************************
    ******************/
85
86      startPoint = checkStartPoint();
87
88  /*****************************************************************************
    ******************/
89  /************************************ STARTPUNKT PRUEFEN
     ************************************/
90  /*****************************************************************************
    ******************/
91
92
93  /*****************************************************************************
    ******************/
94  /***************************** INITIALISIERUNG DES SUCHVERFAHRENS
     *********************************/
95  /
```

```
    ********************************************************************************
    ******************/
 96     led(LAMPE_ROT,0);
 97     led(LAMPE_GELB,1);
 98     breitenSuche(startPoint);
 99
100 /                                                                              
    ********************************************************************************
    ******************/
101 /****************************** INITIALISIERUNG DES SUCHVERFAHRENS               
     *********************************/
102 /                                                                              
    ********************************************************************************
    ******************/
103
104 /                                                                              
    ********************************************************************************
    ******************/
105 /********************************** FAHRGASTPOSITIONEN SORTIEREN                 
     ***********************************/
106 /                                                                              
    ********************************************************************************
    ******************/
107
108     sortGuestPositions();
109
110 /                                                                              
    ********************************************************************************
    ******************/
111 /********************************** FAHRGASTPOSITIONEN SORTIEREN                 
     ***********************************/
112 /                                                                              
    ********************************************************************************
    ******************/
113
114 /                                                                              
    ********************************************************************************
    ******************/
115 /****************************** ALLE ARRAYS MIT 0 INITIALISIEREN                 
     *********************************/
116 /                                                                              
    ********************************************************************************
    ******************/
117
118     initAllPaths();
119
120 /                                                                              
    ********************************************************************************
    ******************/
121 /****************************** ALLE ARRAYS MIT 0 INITIALISIEREN                 
     *********************************/
122 /                                                                              
    ********************************************************************************
    ******************/
123
124 /                                                                              
```

```
    ****************************************************************************
    *****************/
125 /***************************** PFADARRAYS FUELLEN UND KONVERTIEREN
     *******************************/
126 /
    ****************************************************************************
    *****************/
127
128     //Wenn mindestens ein Fahrgast vorhanden ist, fuelle Pfad/e - ansonsten
        startet der Roboter erst gar nicht - siehe Hauptschleife else-Zweig
129     if(guestCounter != 0)
130     {
131         //LOKALE VARIABLE
132         char status = 5;        //fillPathStatus -beliebiger Wert ausser 0,1,2
133
134         for(index = 0; index < guestCounter; index++)
135         {
136             status = fillPathArrays(startPoint, index);
137
138             //Pfad existiert von Gast bis zum Startpunkt
139             if(status == 0)
140             {
141                 lcd_cls();
142                 lcd_setxy(0,0);
143                 lcd_puts("Pfad existiert.");
144                 sleep(200);
145             }
146             else if(status == 2)
147             {
148                 lcd_cls();
149                 lcd_setxy(0,0);
150                 lcd_puts("Pfad konnte");
151                 lcd_setxy(1,0);
152                 lcd_puts("nicht erstellt werden.");
153                 sleep(200);
154                 while(1);
155             }
156             else
157             {
158                 lcd_cls();
159                 lcd_setxy(0,0);
160                 lcd_puts("undefiniertes");
161                 lcd_setxy(1,0);
162                 lcd_puts("Verhalten.");
163                 sleep(200);
164                 while(1);
165             }
166         }//END OF FOR
167     }
168
169 /
    ****************************************************************************
    *****************/
170 /***************************** PFADARRAYS FUELLEN UND KONVERTIEREN
     *******************************/
171 /
```

```
    ********************************************************************************
    *****************/
172
173 /                                                                               ↵
    ********************************************************************************
    *****************/
174 /************************************** START PRUEFEN                           ↵
     ***************************************/
175 /                                                                               ↵
    ********************************************************************************
    *****************/
176     sleep(1000);
177     led(LAMPE_GELB,0);
178     checkStart();
179     clear_time();
180
181 /                                                                               ↵
    ********************************************************************************
    *****************/
182 /************************************** START PRUEFEN                           ↵
     ***************************************/
183 /                                                                               ↵
    ********************************************************************************
    *****************/
184
185 /                                                                               ↵
    ********************************************************************************
    *****************/
186 /************************************** HAUPTSCHLEIFE                           ↵
     ***************************************/
187 /                                                                               ↵
    ********************************************************************************
    *****************/
188     while(akt_time() <= 120000)
189     {
190         //Wenn es (noch) einen Fahrgast gibt, dann arbeite den Pfad ab solange  ↵
            aktuelle Zeit die 2 Minuten nicht erreicht hat!!
191         if(guestCounter != 0)
192         {
193             //Wenn ein Pfad abgearbeitet ist, dann zaehle Counter runter -      ↵
                Aenderung der isMissionComplete Variable in Crossroads()
194             if(isMissionComplete == 1)
195             {
196                 pathCounter++;
197                 mIndex = 0;                     //Arrayindex für nächsten       ↵
                    masterPath auf 0 zuruecksetzen
198                 isMissionComplete = 0;          //Fuer neue Pfadmission         ↵
                    zuruecksetzen
199
200                 lcd_cls();
201                 lcd_setxy(0,0);
202                 lcd_puts("Suche neue Ziel:");
203                 lcd_setxy(1,0);
204                 lcd_ubyte(guestArray[pathCounter]);
205             }
206
```

```
207             if(pathCounter < guestCounter)
208                 navigate();
209             else
210             {
211                 stop();
212                 lcd_cls();
213                 lcd_setxy(0,0);
214                 lcd_puts("Alles schoen");
215                 lcd_setxy(1,0);
216                 lcd_puts("FEIERABEND");
217                 sleep(12000);
218             }
219 
220         }
221         else
222         {
223             lcd_cls();
224             lcd_setxy(0,0);
225             lcd_puts("Kein Fahrgast");
226             lcd_setxy(1,0);
227             lcd_puts("vorhanden.");
228             sleep(120000);
229         }
230     }
231     stop();
232     lcd_cls();
233     lcd_setxy(0,0);
234     lcd_puts("Zeit");
235     lcd_setxy(1,0);
236     lcd_puts("abgelaufen.");
237     sleep(500);
238 
239 /*****************************************************************************
    ******************/
240 /************************************* HAUPTSCHLEIFE
     *************************************/
241 /*****************************************************************************
    ******************/
242 }
243 
244 
245 
246 //HILFSFUNKTIONEN
247 
248 /*****************************************************************************/
249 /* Diese Funktion dient der reinen Streckennavigierung. Sollte der Roboter an */
250 /* eine Kreuzung kommen, so wird die Ausfuehrung an die Funktion crossroads()
     */
251 /* uebergeben. Sollte der Greifer mit seinen Optokopplern eine
     */
252 /* Fahrgastposition wahrnehmen, so schliesst sich der Greifer egal welche
     */
253 /* Aktionen noch ausgefuehrt werden, denn dies hat hoechste Prioritaet.
```

```
 */
/*****************************************************************************/

void navigate()
{
    /***Konkuerrierende Aktionen***/
    //Hoechste Prioritaet
    if((LASERSCHRANKE > 200) && (isGripperOpen == 1))
    {
        //TODO
        stop();
        closeGripper();
        isGripperOpen = 0;      //Greifer schliessen
        lcd_cls();
        lcd_setxy(0,0);
        lcd_puts("Gast aufgeladen");
        back();                 //Bis zur letzten befahrenen Kreuzung
        rueckwaertsfahren
    }

    //wenn einer der beiden mittleren Optokoppler auf schwarz - Kreuzung (Zweit-
    hoechste Prioritaet)
    if((OPTOKOPPLER_MITTE_LINKS > 150 || OPTOKOPPLER_MITTE_RECHTS > 150) &&
    (delay <= akt_time()))
        crossroads();
    //Linker mittlerer Optokoppler schwarz UND rechter mittlerer Optokoppler
    weiß
    else if(OPTOKOPPLER_VORNE_LINKS > 150)
        steerLeft();
    //Rechter mittlerer Optokoppler schwarz UND linker mittlerer Optokoppler
    weiß
    else if(OPTOKOPPLER_VORNE_RECHTS > 150)
        steerRight();
    //beide Optokoppler weiß
    else
        drive();
}

/*****************************************************************************/
/* Diese Funktion ueberprueft an den Kreuzungspunkten die jeweils
 */
/* abzuarbeitende Anweisung. Diese Anweisung wird mit Hilfe der in actions.c
 */
/* implementierten Funktionen an die Motoren (mit dazugehoerigen
 */
/* Differentialgetriebe) uebertragen. Des Weiteren wird geprueft, ob wir beim
 */
/* einzulesenden Zeichen im Array masterPath[] schon an eine 99 angelangt
 sind,*/
/* welche ja hinter dem letzten Characterzeichen des Rueckweges folgt. Ist das
 */
/* der Fall, so wird der Greifer geoeffnet, der Fahrgast abgeladen, eine 180
 */
/* Grad Drehung gemacht und bis zur unteren Begrenzung zurückgefahren.
```

```
     */
295  /* Nun wird noch die Variable isMissionComplete auf True gesetzt.
     */
296  /* Wir sind nun wieder beim Startpunkt und koennen die naechste Mission
     */
297  /* starten/ naechsten Fahrgast abholen.
     */
298  /
    ******************************************************************************/
299  void crossroads()
300  {
301      if(mIndex < MAX2)
302      {
303          //Pfad abarbeiten - nur an Kreuzungen
304          lcd_cls();
305          lcd_setxy(0,0);
306          switch(guestPath[pathCounter][mIndex])
307          {
308          case 'r':   lcd_puts("goRight");
309                      biegeRechtsAb();
310                      break;
311          case 'l':   lcd_puts("goLeft");
312                      biegeLinksAb();
313                      break;
314          case 'g':   lcd_puts("goStraight");
315                      drive();
316                      break;
317          case 's':   lcd_puts("dance");
318                      spin_around();
319                      drive();
320                      while(OPTOKOPPLER_MITTE_LINKS > 18 ||
                        OPTOKOPPLER_MITTE_RECHTS > 18);
321                      break;
322          }
323      }
324      //ist der uebernaechste Kreuzungspunkt die Startposition?
325      if(guestPath[pathCounter][mIndex+1] == 'c')
326      {
327          //je nach letzter Ausrichtung des Roboters entsprechend drehen zum
          Abladen des Fahrgastes
328          char difference;
329          difference = guestPath[pathCounter][mIndex-1] - guestPath[pathCounter]
          [mIndex];
330
331          switch(difference)
332          {
333              case -1:    lcd_puts("goRight");
334                          biegeRechtsAb();
335                          break;
336              case 1:     lcd_puts("goLeft");
337                          biegeLinksAb();
338                          break;
339          }
340          openGripper();
341          isGripperOpen = 1;  //Gast abladen
342          lcd_cls();
```

```
343         lcd_setxy(0,0);
344         lcd_puts("Gast abgeladen");
345         spin_around();  //180 Grad Drehung + 1x zurueck bis Optokoppler auf Kreuzungsmitte stehen
346         while(TASTER == 1)
347             back2();
348         stop();
349         isMissionComplete = 1;  //Pfad abgearbeitet
350     }
351     mIndex++;
352     //Damit befahrene Kreuzung nicht mehrfach erkannt wird -> oben im IF-STATEMENT abfragen
353     delay = akt_time() + offset;
354 }
355
356 /*****************************************************************************/
357 /* Diese Funktion erstellt ein Positionsarray vom Fahrgast zum Startpunkt.Falls*/
358 /* dies moeglich ist, so konvertiert die Funktion die Positionsdifferenzen */
359 /* in Anweisungen der Form r,l,g um. Zuletzt wird mit Hilfe des Hinwegs der */
360 /* Rueckweg erstellt */
361 /* Statewerte: 0 - Norden; 1 - Osten; 2 - Sueden; 3 - Westen */
362 /*****************************************************************************/
363 unsigned char fillPathArrays(unsigned char startPoint, unsigned char index)
364 {
365     //LOKALE VARIABLEN
366     unsigned char j, aktPos;
367     char istState = 0;
368     signed char sollState = 0;
369     char action = 0;
370     char tmpPath[33];
371
372     for(j = 0; j < MAX2; j++)
373         guestPath[index][j] = 99;
374
375     /*******************PFADPOSITIONEN HINZUFUEGEN*******************/
376     j = 0;
377     //Fahrgastposition in aktuellePosition ablegen
378     aktPos = guestArray[index];
379
380     while(aktPos != startPoint)
381     {
382         //Fahrgastposition zuerst ins PathArray ablegen und dann ggf. die Nachbarpositionen - Startposition nicht mitnehmen
383         guestPath[index][j] = aktPos;
384
385         //Verbotener Bereich ausschließen -7,-6,-5,-4,-3,-2,-1,70,71,72,73,74,75,76 !!!
386         if((_fa[aktPos-1] < _fa[aktPos]) || (_fa[aktPos-7] < _fa[aktPos]) || (_fa[aktPos+1] < _fa[aktPos]) || (_fa[aktPos+7] < _fa[aktPos]))
```

```
387         {
388             if((aktPos-1 >= 0) && (aktPos-7 >= 0) && (aktPos+1 <= 69) && (aktPos
                +7 <= 69))
389             {
390                 if(_fa[aktPos-1] < _fa[aktPos-7])
391                 {
392                     if(_fa[aktPos-1] < _fa[aktPos+1] )
393                     {
394                         if(_fa[aktPos-1] < _fa[aktPos+7])       //pos-1 < pos+7
                        < pos+1 < pos-7
395                             aktPos = aktPos-1;
396                         else                    //pos+7 < pos-1 < pos+1 < pos-7
397                             aktPos = aktPos+7;
398                     }
399                     else if(_fa[aktPos+1] < _fa[aktPos+7])      //pos+1 < pos+7
                    < pos-1 < pos-7
400                         aktPos = aktPos+1;
401                     else
402                         aktPos = aktPos+7;                      //pos+7 < pos+1
                    < pos-1 < pos-7
403                 }
404                 else if(_fa[aktPos-7] < _fa[aktPos+1])
405                 {
406                     if(_fa[aktPos-7] < _fa[aktPos+7])
407                         aktPos = aktPos-7;                      //pos-7 < pos+7
                    < pos+1 < pos-1
408                     else
409                         aktPos = aktPos+7;                      //pos+7 < pos-7
                    < pos+1 < pos-1
410                 }
411                 else if(_fa[aktPos+1] < _fa[aktPos+7])
412                     aktPos = aktPos+1;                          //pos+1 < pos+7
                    < pos-7 < pos-1
413                 else
414                     aktPos = aktPos+7;                          //pos+7 < pos+1
                    < pos-7 < pos-1
415             }//END OF IF - ALLE NACHBARN INNERHALB DER GRENZEN
416             else if(aktPos-1 < 0)
417             {
418                     if(_fa[aktPos+1] < _fa[aktPos+7])
419                         aktPos = aktPos+1;
420                     else
421                         aktPos = aktPos+7;
422             }//END OF ELSE IF - linker Nachbar und somit auch oberer Nachbar
                außerhalb der Grenzen => nur noch 2 Nachbarn pruefen
423             else if(aktPos-7 < 0)
424             {
425                     if(_fa[aktPos-1] < _fa[aktPos+1])
426                     {
427                         if(_fa[aktPos-1] < _fa[aktPos+7])
428                             aktPos = aktPos-1;
429                         else
430                             aktPos = aktPos+7;
431                     }
432                     else if(_fa[aktPos+1] < _fa[aktPos+7])
433                         aktPos = aktPos+1;
```

```
434                     else
435                         aktPos = aktPos+7;
436             }//END OF ELSE IF  oberer Nachbar außerhalb der Grenzen => 3 Nachbarn pruefen
437             else if(aktPos+1 > 69)
438             {
439                     if(_fa[aktPos-1] < _fa[aktPos-7])
440                         aktPos = aktPos-1;
441                     else
442                         aktPos = aktPos-7;
443             }//END OF ELSE IF - rechter Nachbar und somit auch unterer Nachbar außerhalb der Grenzen => nur noch 2 Nachbarn pruefen
444             else if(aktPos+7 > 69)
445             {
446                     if(_fa[aktPos-1] < _fa[aktPos+1])
447                     {
448                         if(_fa[aktPos-1] < _fa[aktPos-7])
449                             aktPos = aktPos-1;
450                         else
451                             aktPos = aktPos-7;
452                     }
453                     else if(_fa[aktPos+1] < _fa[aktPos-7])
454                         aktPos = aktPos+1;
455                     else
456                         aktPos = aktPos-7;
457             }//END OF ELSE IF - unterer Nachbar außerhalb der Grenzen => 3 Nachbarn pruefen
458
459
460         }//END OF (_fa[aktPos-1] < _fa[aktPos]) || (_fa[aktPos-7] < _fa[aktPos]) || (_fa[aktPos+1] < _fa[aktPos]) || (_fa[aktPos+7] < _fa[aktPos])
461         else
462             //Keine Nachbarposition hat keinen kleineren Kostenwert als der Kostenwert des Fahrgastes(oder aktPos) - Kein Weg -> Aus While-Schleife herausspringen
463             return 2;
464         j++; //Knotenindex erhoehen fuer Nachfolgeknoten
465
466     }// End of While
467     /*******************PFADPOSITIONEN HINZUFUEGEN*******************/
468
469     //Startposition zum Array hinzufuegen
470     guestPath[index][j] = startPoint;
471
472     /*******************HINWEG*******************/
473     aktPos = 0;
474
475     for(j = 0; j < MAX2; j++)
476         tmpPath[j] = 99;
477
478
479     for(j = MAX2; j > 0; j--)
480     {
481
482         sollState = guestPath[index][j-1] - guestPath[index][j];
483         if(sollState == -1)
```

```
484             sollState = 3; //nach Westen
485         else if(sollState == 1)
486             sollState = 1; //nach Osten
487         else if(sollState == -7)
488             sollState = 0; //nach Norden
489         else if(sollState == 7)
490             sollState = 2; //nach Sueden
491         else
492             continue;
493
494         action = ((sollState-istState) % 4);
495
496
497         if(action == 0 || action == -4)
498         {
499             istState = action + istState;       //Nord
500             tmpPath[aktPos] = 'g';
501             aktPos++;
502         }
503         else if(action == 1 || action == -3)
504         {
505             istState = action + istState;       //Ost
506             tmpPath[aktPos] = 'r';
507             aktPos++;
508         }
509         else if(action == 3 || action == -1)
510         {
511             istState = action + istState;       //West
512             tmpPath[aktPos] = 'l';
513             aktPos++;
514         }
515     }//END OF FOR
516     /********************HINWEG********************/
517
518     for(j = 0; j < MAX2; j++)
519         guestPath[index][j] = 99;
520
521     for(j = 0, aktPos = 0; j < MAX2, aktPos < MAX2; j++, aktPos++)
522     {
523         if(tmpPath[aktPos] != 99)
524             guestPath[index][j] = tmpPath[aktPos];
525     }
526
527     /********************DREHUNG UND RUECKWEG********************/
528     j = 0;
529
530     while((guestPath[index][j] == 'r' || guestPath[index][j] == 'l' || guestPath ↵
        [index][j] == 'g') && j < MAX2)
531         j++;    //damit hinter dem letzen l/r/g der aktuelle Zeiger steht =>     ↵
            fuer Rueckweg
532
533     switch(guestPath[index][j-1])
534     {
535     case 'r':   guestPath[index][j] = 'r';
536                 break;
537     case 'l':   guestPath[index][j] = 'l';
```

```
538                 break;
539     default:    guestPath[index][j] = 's';
540                 break;
541     }
542
543     j++;
544     aktPos = j;
545
546     while(j >= 3)
547     {
548         if(guestPath[index][j-3] == 'g')
549             guestPath[index][aktPos] = 'g';
550         else if(guestPath[index][j-3] == 'r')
551             guestPath[index][aktPos] = 'l';
552         else if(guestPath[index][j-3] == 'l')
553             guestPath[index][aktPos] = 'r';
554
555         aktPos++;
556         j--;
557     }
558     /********************DREHUNG UND RUECKWEG********************/
559     return 0;
560 }
561
562 void initAllPaths()
563 {
564     unsigned char i,j;
565     //Pfad Arrays standardmaessig mit 99 initialisieren
566     for(i = 0; i < 3; i++)
567     {
568         for(j = 0; j < MAX2; j++)
569             guestPath[i][j] = 99;
570     }
571 }
572
573 /*****************************************************************************/
574 /* Diese Funktion holt die einzelnen Fahrgastposition aus dem mit Hilfe von  */
    */
575 /* pushGuestPosition() gefuellten Array und sortiert diese nach Hoehe ihrer  */
    */
576 /* Kosten in ein neues Array (guestArray[])                                  */
    */
577 /*****************************************************************************/
578 void sortGuestPositions()
579 {
580     //LOKALE VARIABLEN
581     unsigned char i, j, temp;
582     unsigned char guestArrayLength = 3;
583
584     //POSITIONEN AUS guestPosition[] holen und in guestArray kopieren
585     for(i = 0; i < 3; i++)
586     {
587         //Wenn kein Fahrgast mehr da ist gibt diese Funktion 99 zurück Bsp. =>
        array[] = {62,30,99}
```

```
588         guestArray[i] = getGuestPosition();
589
590         //Hat das guestArray eine Position != 99, so zaehle den Zaehler fuer die
             Fahrgaseste hoch
591         if(guestArray[i] != 99)
592             guestCounter++;
593     }
594
595     //Bubblesort
596     for(i = 0; i < guestArrayLength -1; i++)
597     {
598         for(j = 0; j < guestArrayLength - i - 1; j++)
599         {
600             if(_fa[guestArray[j]] > _fa[guestArray[j+1]])
601             {
602                 temp = guestArray[j];
603                 guestArray[j] = guestArray[j+1];
604                 guestArray[j+1] = temp;
605             }
606         }
607     }
608 }
609
610 /
    ****************************************************************************/
611 /* Die BFS weist alle erreichbaren Punkte und Fahrgaeste in seine Kosten zu.
     */
612 /* Ausgangspunkt: Startpunkt 64 oder 68. Des Weiteren speichert diese Funktion
     */
613 /* mit der in agenda.c enthalten Funktion pushGuestKnoten() die Positionen der
     */
614 /* Fahrgaeste in ein Array ab.
     */
615 /
    ****************************************************************************/
616 unsigned char breitenSuche(unsigned char startPunkt)
617 {
618     //LOKALE VARIABLEN
619     unsigned char kosten = 0;
620     unsigned char status = 1;
621     char counter1 = 0;
622     char counter2 = 0;
623     unsigned char aktKnoten = startPunkt;
624     initKarte(MAX);
625     pushKnoten(aktKnoten, MAX);
626     _fa[aktKnoten] = kosten;
627
628
629     while(aktKnoten != 99)
630     {
631         kosten++;
632         counter1 = 0;
633
634         //Zweifache Do-While-Schleife damit die Kosten richtig verteilt werden
            auf einer Knotenebene
635
```

```
636         do{
637             aktKnoten = getKnoten(MAX);
638 
639 
640             if(_fa[aktKnoten-1] == '.' || _fa[aktKnoten-1] == 'F')
641             {
642                 if(_fa[aktKnoten-1] == 'F')
643                     pushGuestPosition(aktKnoten-1); //Position merken
644 
645                 pushKnoten(aktKnoten-1, MAX);
646                 _fa[aktKnoten-1] = kosten;
647                 counter1++;
648             }
649             if(_fa[aktKnoten-7] =='.' || _fa[aktKnoten-7] == 'F')
650             {
651                 if(_fa[aktKnoten-7] == 'F')
652                     pushGuestPosition(aktKnoten-7); //Position merken
653 
654                 pushKnoten(aktKnoten-7, MAX);
655                 _fa[aktKnoten-7] = kosten;
656                 counter1++;
657             }
658             if(_fa[aktKnoten+1] == '.' || _fa[aktKnoten+1] == 'F')
659             {
660                 if(_fa[aktKnoten+1] == 'F')
661                     pushGuestPosition(aktKnoten+1); //Position merken
662 
663                 pushKnoten(aktKnoten+1, MAX);
664                 _fa[aktKnoten+1] = kosten;
665                 counter1++;
666             }
667             if(_fa[aktKnoten+7] == '.' || _fa[aktKnoten+7] == 'F')
668             {
669                 if(_fa[aktKnoten+7] == 'F')
670                     pushGuestPosition(aktKnoten+7); //Position merken
671 
672                 pushKnoten(aktKnoten+7, MAX);
673                 _fa[aktKnoten+7] = kosten;
674                 counter1++;
675             }
676             counter2--;
677         }while(counter2 > 0);
678 
679         kosten++;
680         counter2 = 0;
681 
682         do{
683             counter1--;
684             aktKnoten = getKnoten(MAX);
685 
686             if(_fa[aktKnoten-1] == '.' || _fa[aktKnoten-1] == 'F')
687             {
688                 if(_fa[aktKnoten-1] == 'F')
689                     pushGuestPosition(aktKnoten-1); //Position merken
690 
691                 pushKnoten(aktKnoten-1, MAX);
```

```
692                 _fa[aktKnoten-1] = kosten;
693                 counter2++;
694             }
695             if(_fa[aktKnoten-7] =='.' || _fa[aktKnoten-7] == 'F')
696             {
697                 if(_fa[aktKnoten-7] == 'F')
698                     pushGuestPosition(aktKnoten-7); //Position merken
699 
700                 pushKnoten(aktKnoten-7, MAX);
701                 _fa[aktKnoten-7] = kosten;
702                 counter2++;
703             }
704             if(_fa[aktKnoten+1] == '.' || _fa[aktKnoten+1] == 'F')
705             {
706                 if(_fa[aktKnoten+1] == 'F')
707                     pushGuestPosition(aktKnoten+1); //Position merken
708 
709                 pushKnoten(aktKnoten+1, MAX);
710                 _fa[aktKnoten+1] = kosten;
711                 counter2++;
712             }
713             if(_fa[aktKnoten+7] == '.' || _fa[aktKnoten+7] == 'F')
714             {
715                 if(_fa[aktKnoten+7] == 'F')
716                     pushGuestPosition(aktKnoten+7); //Position merken
717 
718                 pushKnoten(aktKnoten+7, MAX);
719                 _fa[aktKnoten+7] = kosten;
720                 counter2++;
721             }
722         }while(counter1 > 0);
723     }
724     return 0;
725 }
726 
```

```
 1 // Autor: I. Boersch
 2
 3 // Definition der Fahrauftraege zum AMS-Contest MASDAR-City WS13/14
 4 // Um Speicherplatz zu sparen, Ablage als Praeprozessor-Anweisung
 5 // File generiert mit generate_configs_with_solution.py
 6
 7 // Fahrauftrag F1 - From task description - this should be easy!
 8 #ifdef FA1
 9 unsigned char _fa [] =                                                  ↵
   "xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx...x.xxx..x.xF..x..Fx..x..x";
10 unsigned char _fa_nr = 1;
11 #define _FA_OK
12 #endif
13
14 // Fahrauftrag F2 - From task description - one crossing modified
15 #ifdef FA2
16 unsigned char _fa [] =                                                  ↵
   "xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx..x..xxx..x.xF..x..Fx..x..x";
17 unsigned char _fa_nr = 2;
18 #define _FA_OK
19 #endif
20
21 // Fahrauftrag F3 - City Slalom
22 #ifdef FA3
23 unsigned char _fa [] =                                                  ↵
   "xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx..x..xxx.x..xF..x..Fx..x..x";
24 unsigned char _fa_nr = 3;
25 #define _FA_OK
26 #endif
27
28 // Fahrauftrag F4 - Race on the Highway
29 #ifdef FA4
30 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";
31 unsigned char _fa_nr = 4;
32 #define _FA_OK
33 #endif
34
35 // Fahrauftrag F5 - Big Slalom
36 #ifdef FA5
37 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";
38 unsigned char _fa_nr = 5;
39 #define _FA_OK
40 #endif
41
42 // Fahrauftrag F6 - Symmetry
43 #ifdef FA6
44 unsigned char _fa [] =                                                  ↵
   "xxFxFxxF..x..Fx.x.x.xx..x..xF..x..Fx..x..xxx.x.xxx..x..xxx.x.xxx..x..x";
45 unsigned char _fa_nr = 6;
46 #define _FA_OK
47 #endif
48
49 // Fahrauftrag F7 - Connected Game
50 #ifdef FA7
```

```
51 unsigned char _fa [] =
   "xxxFxxxx.....xx..x..xx..x..xF..x..Fx..x..xx..x..xxx.x.xxx..x..xx..x..x";
52 unsigned char _fa_nr = 7;
53 #define _FA_OK
54 #endif
55
56 // Fahrauftrag F8 - Blocked Passengers
57 #ifdef FA8
58 unsigned char _fa [] =
    "xxxFxxxF..x..Fx..x..xFx.x.xFF..x..Fx...x.xx..x..xx.x...xx..x..xx..x..x";
59 unsigned char _fa_nr = 8;
60 #define _FA_OK
61 #endif
62
63 // Fahrauftrag F9 - Outdoor - Free Land
64 #ifdef FA9
65 unsigned char _fa [] =
    "xFxxxFxF..x..Fx.x...xx.x...xx.x...xx.xxx.xx...x.xx...x.xF...x.Fx..x..x";
66 unsigned char _fa_nr = 9;
67 #define _FA_OK
68 #endif
69
70 // Fahrauftrag F10 - Long Distance - Endurance
71 #ifdef FA10
72 unsigned char _fa [] =
    "xFxxxFxF..x..Fx.x...xx.x.xxxx.x...xx.xxx.xx...x.xxxx.x.xx...x.xx..x..x";
73 unsigned char _fa_nr = 10;
74 #define _FA_OK
75 #endif
76
77 // Fahrauftrag F11 - VIP
78 #ifdef FA11
79 unsigned char _fa [] =
    "xxxFxxxx.....xx..x..xx.....xxx...xxx.....xx.....xx.....xx.....xx..x..x";
80 unsigned char _fa_nr = 11;
81 #define _FA_OK
82 #endif
83
84 // Fahrauftrag F12 - Lazy Passengers
85 #ifdef FA12
86 unsigned char _fa [] =
    "xxxxxxxx..x..xx..x..xx..x..xx..x..xx..x..xF..x..FF..x..FF..x..Fx..x..x";
87 unsigned char _fa_nr = 12;
88 #define _FA_OK
89 #endif
90
91 // Fahrauftrag F13 - DennisWeil Route.1
92 #ifdef FA13
93 unsigned char _fa [] =
    "xxFxFxxx..x..xx...x.xx.xx..xF..x..Fx..x.xxx..x..xxx..x.xF..x..Fx.x...x";
94 unsigned char _fa_nr = 13;
95 #define _FA_OK
96 #endif
97
98 // Fahrauftrag F14 - DennisWeil.Route.2
99 #ifdef FA14
```

```
100 unsigned char _fa [] =
     "xFxxxFxx.x...xx..x.xxxx.x..xF..x..Fxx.x.xxx..x..xx.x..xxF..x..Fx...x.x";
101 unsigned char _fa_nr = 14;
102 #define _FA_OK
103 #endif
104
105 // Fahrauftrag F15 - DennisWeil.Route.3
106 #ifdef FA15
107 unsigned char _fa [] =
     "xxFxFxxx..x..xx..x..xF...x.Fx...x.xx.xx..xx...x.xF..x..Fxx.x.xxx..x..x";
108 unsigned char _fa_nr = 15;
109 #define _FA_OK
110 #endif
111
112 // Fahrauftrag F16 - DennisWeil.Route.4
113 #ifdef FA16
114 unsigned char _fa [] =
     "xxFxFxxF..x..Fxx.x.xxx..x..xx.x...xx..x.xxx..x..xx...x.xF..x..Fx..x..x";
115 unsigned char _fa_nr = 16;
116 #define _FA_OK
117 #endif
118
119 // Fahrauftrag F17 - DennisWeil.Route.5
120 #ifdef FA17
121 unsigned char _fa [] =
     "xFxxxFxx.x...xx..x.xxx..x..xx..x..xF..x..Fx..x.xxxx.x..xF...x.Fx..x..x";
122 unsigned char _fa_nr = 17;
123 #define _FA_OK
124 #endif
125
126 // Fahrauftrag F18 - DennisWeil.Route.6
127 #ifdef FA18
128 unsigned char _fa [] =
     "xxxxxxxx..x..xx..x..xF..x..Fxx.x.xxF..x..Fx.xxx.xF..x..Fxx.x.xxx..x..x";
129 unsigned char _fa_nr = 18;
130 #define _FA_OK
131 #endif
132
133 // Fahrauftrag F19 - DennisWeil.Route.7
134 #ifdef FA19
135 unsigned char _fa [] =
     "xxxxxxxF..xx.Fx..x..xx.x..xxx.x...xF..x..Fx..x..xxx.x.xxF..x..Fx..x..x";
136 unsigned char _fa_nr = 19;
137 #define _FA_OK
138 #endif
139
140 // Fahrauftrag F20 - DennisWeil.Route.8
141 #ifdef FA20
142 unsigned char _fa [] =
     "xxFxFxxx..x..xx..x..xxx.x.xxx..x..xF..x..Fxx.x.xxxx.x.xxF..x..Fx.x.x.x";
143 unsigned char _fa_nr = 20;
144 #define _FA_OK
145 #endif
146
147 // Fahrauftrag F21 - DennisWeil.Route.9
148 #ifdef FA21
```

```
149 unsigned char _fa [] =
     "xxFxFxxx..x..xx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.xF..x..Fx.x.x.xx..x..x";
150 unsigned char _fa_nr = 21;
151 #define _FA_OK
152 #endif
153
154 // Fahrauftrag F22 - DennisWeil.Route.10
155 #ifdef FA22
156 unsigned char _fa [] =
     "xxFxFxxx..x..xx.xxx.xx.x...xF.xxx.Fx...x.xx..xx.xF..x..Fxx.x.xxx..x..x";
157 unsigned char _fa_nr = 22;
158 #define _FA_OK
159 #endif
160
161 // Fahrauftrag F23 - Finale 3 VIPs
162 #ifdef FA23
163 unsigned char _fa [] =
     "xxFFFxxx.....xx.xxx.xx.....xx.....xx.....xx.....xx.....xxx...xxx..x..x";
164 unsigned char _fa_nr = 23;
165 #define _FA_OK
166 #endif
167
168 #ifndef _FA_OK
169 #error Fehler: Kein Fahrauftrag, bspw. mit <<#define FA2>> definiert!
170 #endif
```