Fachhochschule Brandenburg
University of Applied Sciences
Fachbereich Informatik und Medien

**AMS-Projekt**

**MASDAR CITY – Personal Rapid Transit**

**im BA-Studiengang Informatik**

**(WS 2013/14)**


vorgelegt von:

Rico Wildenhein

Matrikelnummer: XXXXXXXX

Benjamin Dietrich

Matrikelnummer: XXXXXXXX


vorgelegt am 17.01.2014

# Inhalt

## 1 Vorwort

Ein PRT (Personal Rapid Transit) – System ist eine Flotte kleiner Fahrzeuge, die jeweils eine oder wenige Personen ohne Zwischenhalt zu individuellen Zielen transportieren. Das derzeit größte geplante PRT-Netz mit 30.000 Fahrzeugen entsteht in Masdar City, eine Stadt in der Wüste der Vereinigten Arabischen Emirate. Auf eine Bahnlänge von ca. 33 Km sollen später 83 Stationen angefahren werden. Eine erste Testinstallation eines PRT-Netzes der Firma 2getthere (Niederlande) mit 10 Fahrzeugen, zwei Personen- und drei Frachtstationen ist seit August 2011 in Masdar City in Betrieb. Probleme im PRT-Netz entstehen bei Überlastung(globaler Stau) oder durch liegenbleibende Fahrzeuge. Da man aber einen direkten Einfluss auf die Fahrzeuge hat, kann man leicht den Verkehrsfluss optimieren. (Wikipedia, 2014)


**Abbildung 1: PRT - Kabine ULTra am London Heathrow Airport (Wikipedia, 2014)**

## 2 Aufgabenstellung

Ziel dieses Projektes ist es einen Roboter zu konstruieren, der einen Fahrauftrag abarbeitet. Dieser Fahrauftrag enthält eine oder mehrere Personen die abgeholt und zum Ziel gebracht werden sollen. Das Streckennetz ist ein einfaches Gitter, in dem es allerdings zu Störungen und damit unbefahrbaren Kreuzungen kommen kann. Der Roboter erhält vor dem Start die Informationen, welche Wege befahrbar und welche gesperrt sind. Ziel ist es innerhalb von 120 Sekunden den Fahrauftrag abzuarbeiten.

# 3 Lösungsweg

## 3.1 Zeitplanung

| | Oktober | November | Dezember | Januar |
|---|---|---|---|---|
| Roboter konstruieren | X | X | | |
| Breitensuche implementieren | | X | | |
| Steuerung implementieren | | X | | |
| Testen | | | X | X |

Tabelle 1: Zeitplanung

## 3.1 Entwurfsphase

Für die Konstruktion des Roboters stand eine Reihe von Bauteilen zur Verfügung. In Abbildung 2 dargestellt sind in der oberen Reihe: Aksenboard, Akkumulator, Servomotor, Elektromotoren. In der unteren Reihe: Infrarot-Sensor, Optokoppler, Infrarot-Led und ein Photosensor.

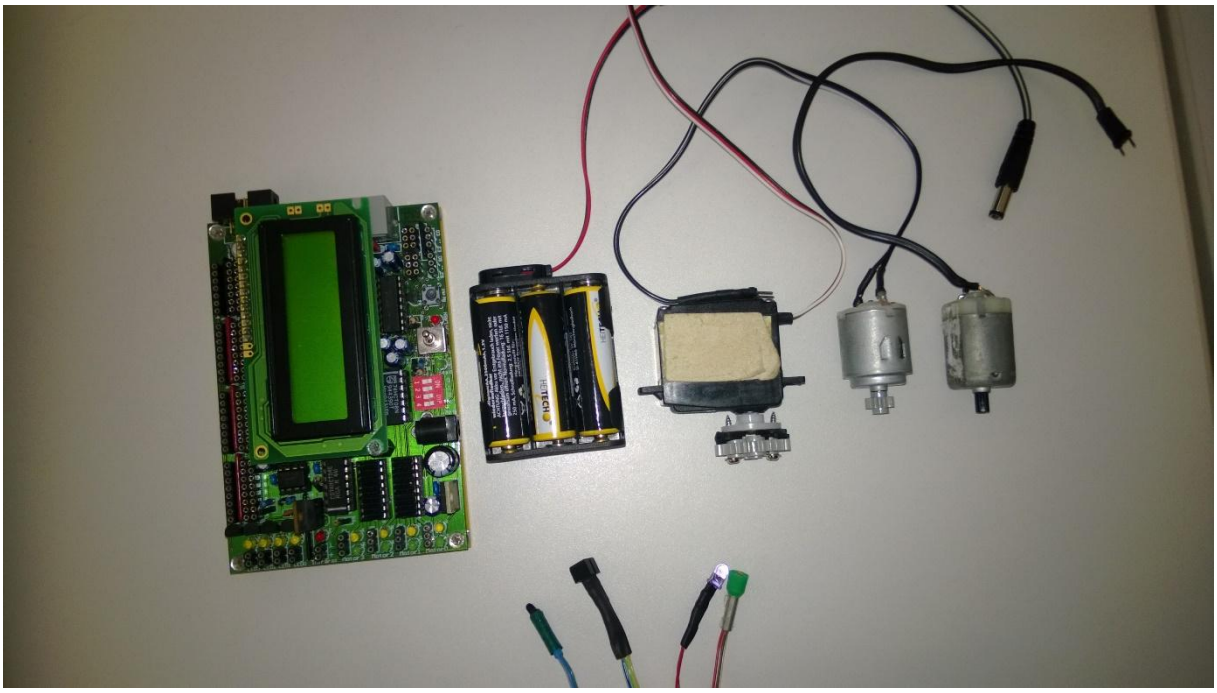

Abbildung 2: Bauteile

Zu Beginn stellte sich die Frage nach der geeigneten Art des Antriebs. Der Differentialantrieb war die erste Wahl. Er ist leicht umzusetzen und jedes Rad kann einzeln angesteuert werden. Die rechtwinkligen Kurven können durch das Anhalten

eines Rads abgefahren werden. Zusätzlich besteht hier durch das gegensätzliche Schalten der Motordrehrichtung die Möglichkeit einer Drehung um den Mittelpunkt der Achse.

Der nächste Ansatzpunkt der Konstruktionsplanung war die Übersetzung des Getriebes. Das Ziel einen möglichst schnellen Roboter zu erhalten war dabei maßgeblich. Die Auswahl der zu verwendenden Zahnräder beschränkte sich auf 8, 16, und 24 Zahn, da die größte Variante mit 40 Zähnen sehr groß und damit ungünstig für einen leichten und kleinen Rahmen erschien. Nach einem Test mit den zur Verfügung stehenden Motoren sollte das Übersetzungsverhältnis i = 81/1 betragen. Diese Übersetzung lieferte den Ansprüchen entsprechend eine hohe Drehzahl der Räder und eine ausreichende Kraftübertragung.

Die Auswahl der Räder unterlag hauptsächlich dem Kriterium der Robustheit, d.h. es sollte zu keiner Beeinflussung durch zu starke Verformung kommen, wie es z.B. bei Reifen mit luftgepolstertem Mantel der Fall sein kann. Das zweite Kriterium war eine möglichst geringe Auflagefläche, um den Energieverlust durch Reibung zu minimieren.

Die Konstruktion des Rahmens erwies sich schwieriger als vermutet. Die Maße des Rahmens sollten symmetrisch zur Radachse sein, damit der Roboter so wendig wie möglich bleibt. Das Anbringen der Motoren verlängerte den Roboter sehr stark, wodurch das Kriterium der Achsensymmetrie vernachlässigt werden musste. Es entstand eine hecklastige Rahmenkonstruktion. Nur der Servomotor und der Greifer für die Passagiere waren für die Positionierung vor der Achse vorgesehen.

Der nun fahrtüchtige Roboter musste an dieser Stelle noch mit der nötigen Sensorik ausgestattet werden. Eine Leiste aus 5 Optokopplern im Abstand einer halben Achsenlänge zum Achsenmittelpunkt an der Vorderseite des Rahmens angebracht, bildet die technische Grundlage des Spurfolgens. Der Abstand zur Tischoberfläche musste hier sehr gering gehalten werden, um eine Verfälschung der Messwerte durch äußere Lichtquellen zu minimieren.

An dieser Stelle erwies sich das Ziel eines besonders schnellen Roboters auch als besondere Schwierigkeit. Die Steuerung war durch die hohe Geschwindigkeit nicht zufriedenstellend umsetzbar. Die Verzögerung von der Registrierung einer Kreuzung bis zur Einleitung des Kurvenfahrens war zu groß. Die gewählte Übersetzung machte den Roboter unkontrollierbar. Die Verwendung einer geeigneteren Übersetzung

brachte ein weiteres Problem mit sich. Der Rahmen war nicht für den Einbau eines größeren Zahnrads im Getriebe vorgesehen und musste überarbeitet werden. Durch Anpassung der Höhe der Rahmenkonstruktion konnte die neue Übersetzung nach anfänglichen Schwierigkeiten dennoch erfolgreich eingearbeitet werden. Das neue Übersetzungsverhältnis beträgt nun i = 135/1. Die Steuerbarkeit des Roboters war nun deutlich erhöht. Das Spurfolgen und Kurvenfahren konnten erfolgreich umgesetzt werden.

Aksenboard und Akkumulator waren bisher nur provisorisch untergebracht. Der Akkumulator wurde mit einer Käfigkonstruktion achsen-lastig positioniert und das Aksenboard in einem klappbaren Rahmen auf dem Roboter befestigt.

Der letzte Schritt bestand aus der Entwicklung einer Greifvorrichtung für die Passagiere und der nötigen Sensorik zur Erkennung eines sich in richtiger Position befindlichen Passagiers. Der Servomotor muss weniger Kraft aufbringen, wenn der Greifer nur in horizontaler Richtung bewegt werden muss. Die Verwendung eines Zangensystems war naheliegend. Die Erkennung eines Passagiers in Reichweite des Greifers übernimmt eine einfache bodennahe angebrachte Lichtschranke aus einander gegenüberliegend platziertem Infrarotsender und Infrarotempfänger.

Das Abholen der Passagiere ließ ein neues Problem aufkommen. Der Roboter musste ein Stück zurücksetzen, um seinen Weg zum Startpunkt antreten zu können. Die Realisierung der Rückwärtsfahrt über die vorn angebrachten Sensoren erwies sich als äußerst fehleranfällig. Die Verwendung einer weiteren Sensorleiste am hinteren Teil des Roboters war für eine korrekte Umsetzung zwingend erforderlich.

Nach einigen Testszenarien ergab sich eine weitere Schwierigkeit. Der Roboter war nicht in der Lage einen Passagier direkt nach einer Kurve aufzunehmen. Der Radabstand war zu groß, um die Kurve eng genug fahren zu können. Nach einer letzten Modifikation der Rahmenkonstruktion war der Roboter kompakt genug auch diese Aufgabe erfolgreich zu absolvieren.

## 4 Vorstellung

„Bug“

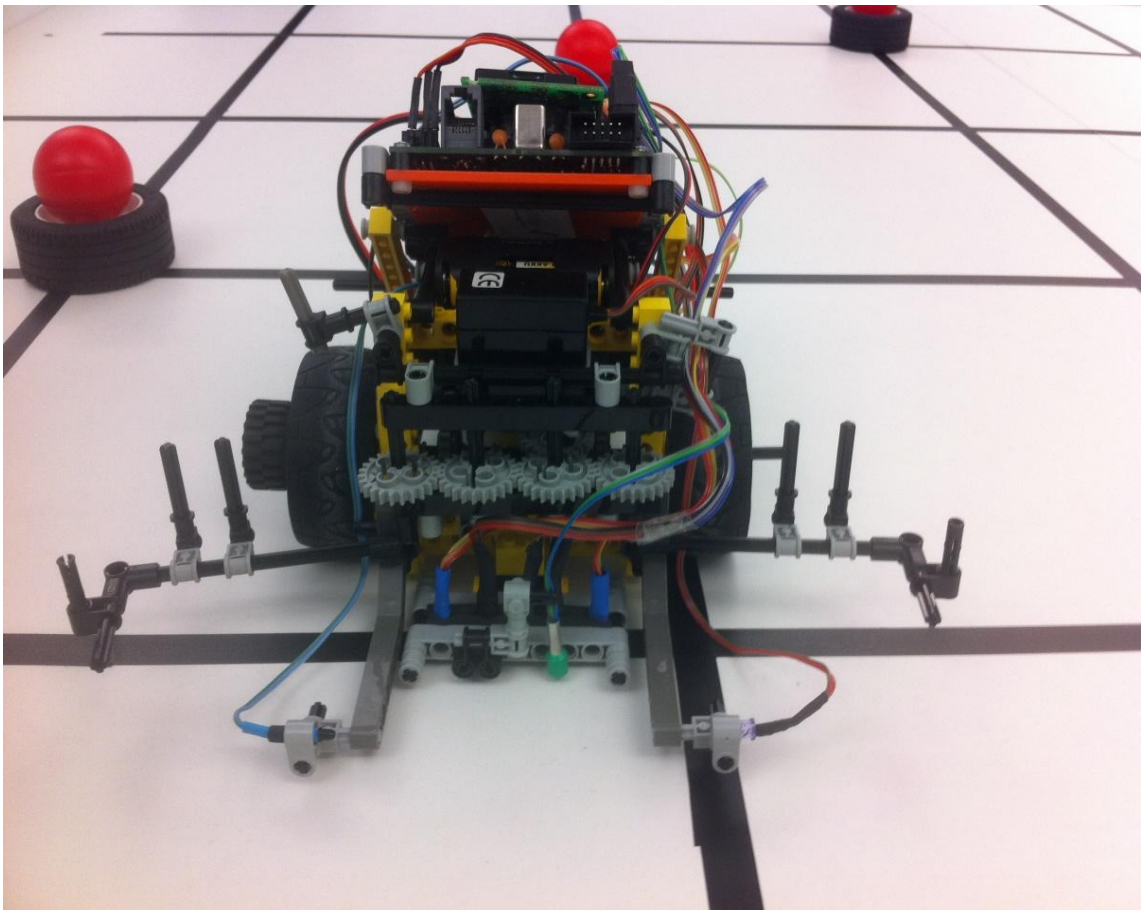

Abbildung 3: Roboter "Bug"

Kompakt, leicht und schnell sind die hervorstechenden Eigenschaften dieses Roboters. Der Rahmen besteht nur aus den für die Konstruktion notwendigen Teilen und der Schwerpunkt befindet sich zwischen Achse und hinterem Auflagepunkt, um das Gewicht gleichmäßig zu verteilen. Charakteristisch ist außerdem noch das durch einen Servomotor angetriebene insektenähnliche Greifwerkzeug (Abb. 3), mit welchem es selbst bei ungenauer Positionierung möglich ist einen Passagier aufzunehmen. Die nach der Aufnahme und Abgabe durchgeführte 180°-Drehung (geeignete Strecke vorausgesetzt) verkürzt die benötigte Zeit für das Abfahren eines Weges enorm. Der Klappmechanismus mit welchem das Aksenboard angebracht ist und der darunterliegende klappbare Käfig für den Akkumulator ermöglichen einen schnellen problemlosen Wechsel, falls es erforderlich sein sollte. Die Energieeffizienz des Roboters ist allerdings so hoch, dass problemlos 5-6 Fahraufträge mit je bis zu drei Passagieren gefahren werden können. Die Verteilung der einzelnen Bauteile ist

exakt aufeinander abgestimmt. Nähme man etwas Gewicht von der Achse, bestünde die Gefahr durchdrehender Reifen.

# 5 Hardware

## 5.1 Antrieb

Zwei Elektromotoren, die ihre Kraft über ein Differentialgetriebe an die Räder weitergeben, treiben den Roboter an. Durch die Verwendung zweier Motoren wird die gesamte Kraft eines Motors auf ein Rad übertragen (Abb. 4), was zwar den Stromverbrauch, dafür aber auch die Geschwindigkeit erhöht.

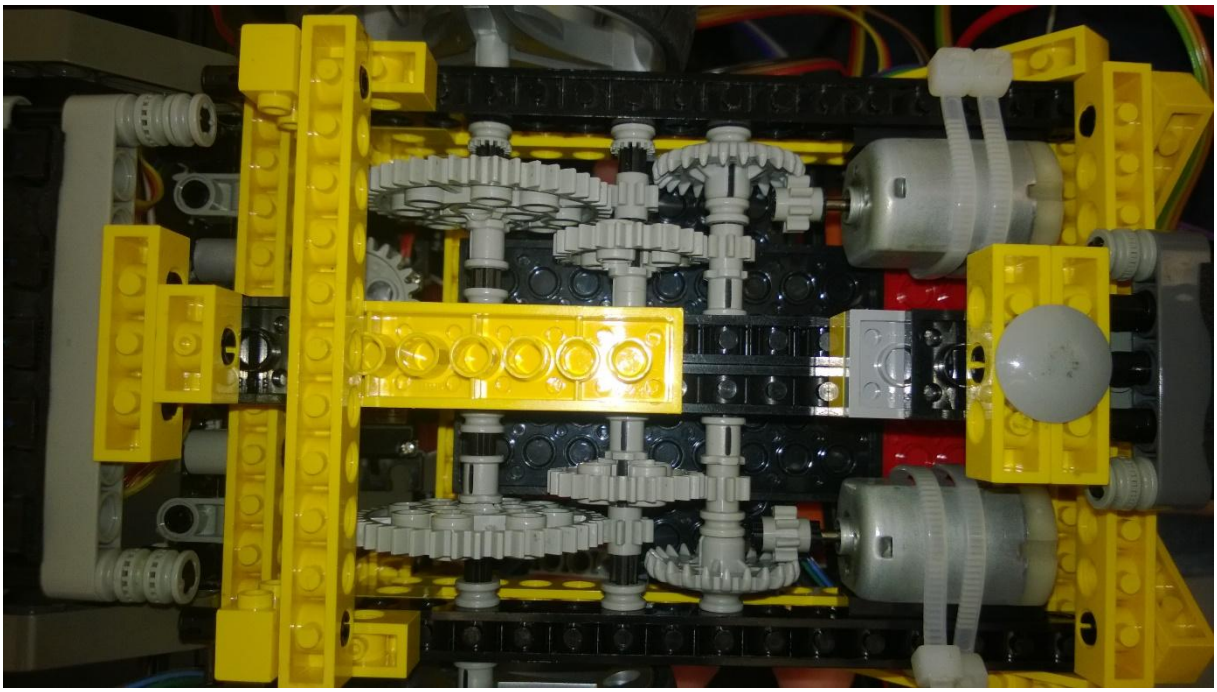

Abbildung 4: Getriebe und Motor

## 5.2 Wendigkeit und Steuerbarkeit

Der Differentialantrieb ermöglicht ein gutes Lenkverhalten. Einfache Kurven werden durch stoppen eines Rades gefahren. Dieses Verfahren hat sich als sicherer erwiesen, um die Spur nach der Kurve zuverlässig zu finden. Durch gegensinnige Motordrehrichtung kann der Roboter auf der Stelle drehen, was die Wendigkeit enorm erhöht. Dies findet allerdings nur bei einer Drehung von 180° Anwendung, da es durch die enorme Geschwindigkeit schwierig ist den Roboter dabei zu beherrschen.

### 5.3 Sensoren

Die Steuerung des Roboters erfolgt durch den Zugriff auf zwei Sensorleisten, die jeweils am vorderen und hinteren Ende angebracht sind. Um eine möglichst hohe Sicherheit bei der Spurverfolgung zu gewährleisten, sind in der vorderen Leiste fünf Optokoppler verbaut (Abb. 5). Die breite Spanne ermöglicht es dem Roboter die Spur auch bei einer größeren Abweichung zu halten. Wie in Abbildung 5 zu sehen ist, sind in der hinteren Sensorleiste aus Sparsamkeitsgründen nur zwei Optokoppler verbaut. Der Roboter muss nur kurze Distanzen rückwärts zurücklegen, weswegen hier auf eine großzügigere Ausstattung verzichtet wurde.

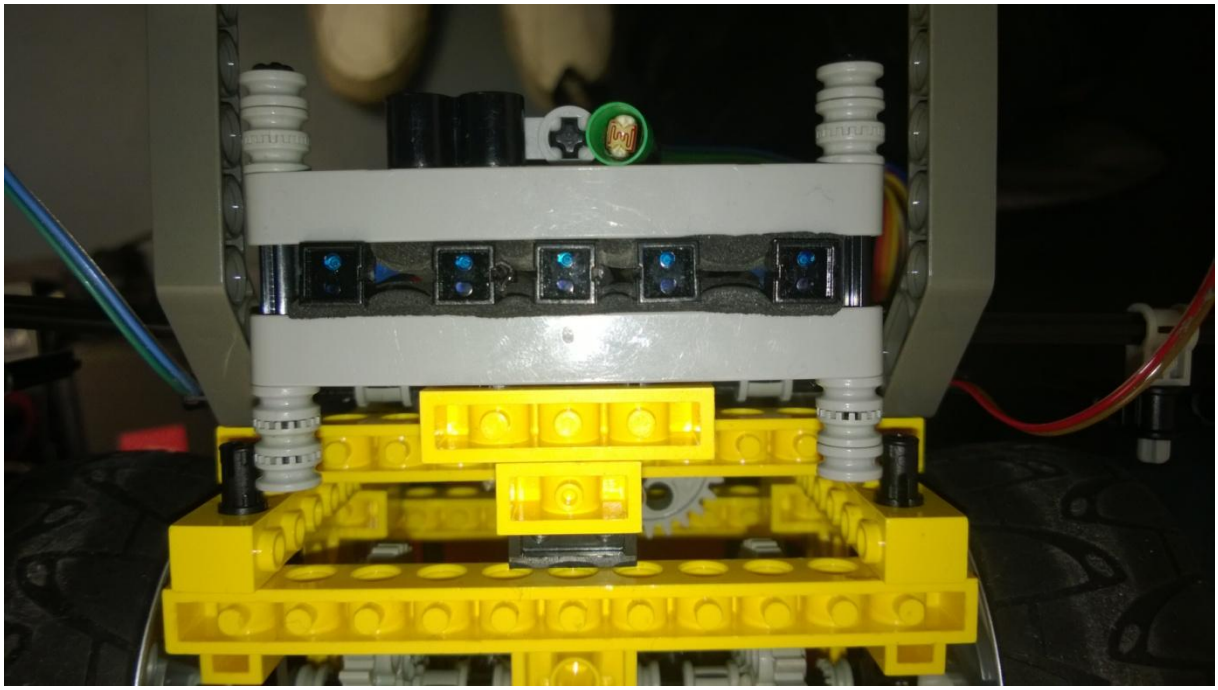

**Abbildung 5: Sensorleiste vorne**

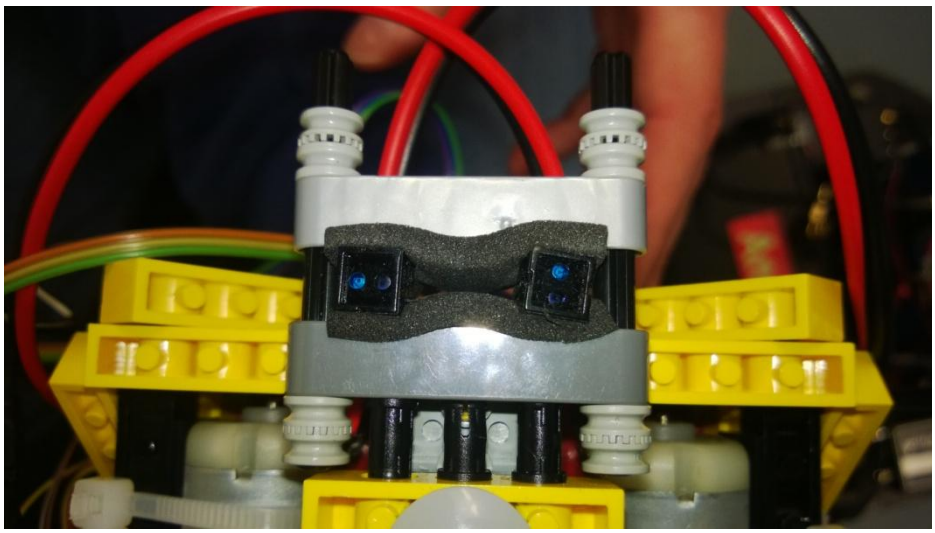

Abbildung 6: Sensorleiste hinten

Eine Infrarot-Led und ein Infrarotsensor bildet die Lichtschranke (Abb. 7), durch welche der Greifer ausgelöst wird.

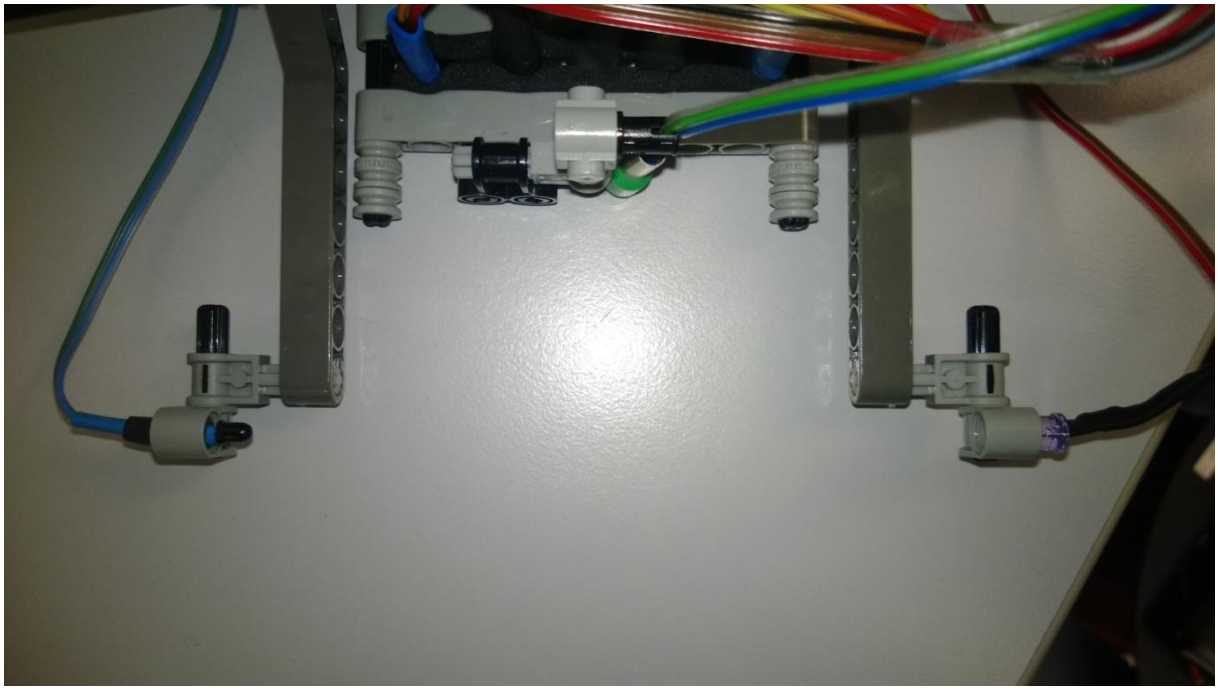

Abbildung 7: Lichtschranke

Das Startsignal wird durch eine im Tisch eingelassene Lampe gegeben. Der in Abbildung 8 dargestellte Photosensor (grünes Bauteil) am vorderen Ende des Roboters erkennt, wenn die Lampe aufleuchtet.

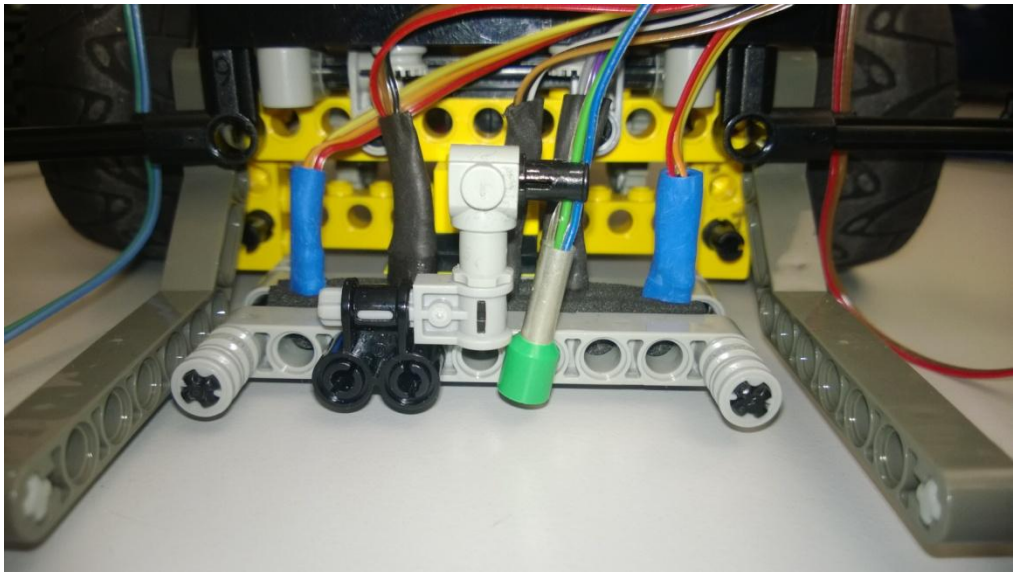

**Abbildung 8: Photosensor**

## 5.4 Besondere Elemente

Ein wichtiges Element zur Stabilisierung des Fahrverhaltens, ist in Abbildung 9 dargestellt. Zwei zusätzliche Räder an der rechten Außenseite stabilisieren durch ihr Eigengewicht die Achse und wirken so korrigierend einem vorher deutlich ausgeprägten Rechtsdrall entgegen.

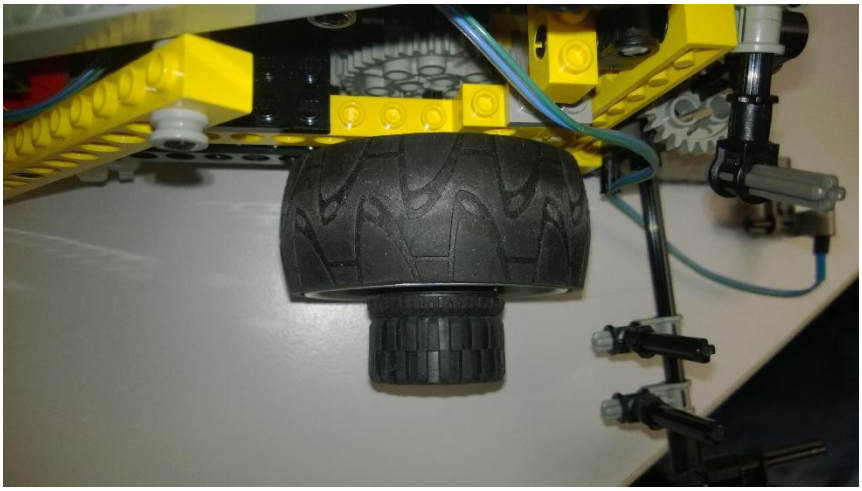

**Abbildung 9: Ausgleichsgewicht für die Achse auf der rechten Seite**

# 6 Software

## 6.1 Implementierung des Suchverfahrens

Die Implementierung des Suchalgorithmus unterteilt sich in zwei Bereiche:

- erstellen der Kostenmatrix
- Implementierung der Breitensuche

Erstellen der Kostenmatrix

Der Fahrauftrag FA1, der in Abbildung 10 dargestellt wird, wird in Form eines Feldes (_fa[]) mit Symbolen eingelesen. „F“ steht für Passagier, „X“ für nicht befahrbare Kreuzung und „.“ für befahrbare Kreuzung.

unsigned char _fa[] =
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx...x.xxx..x.xF..x..Fx..x..x"

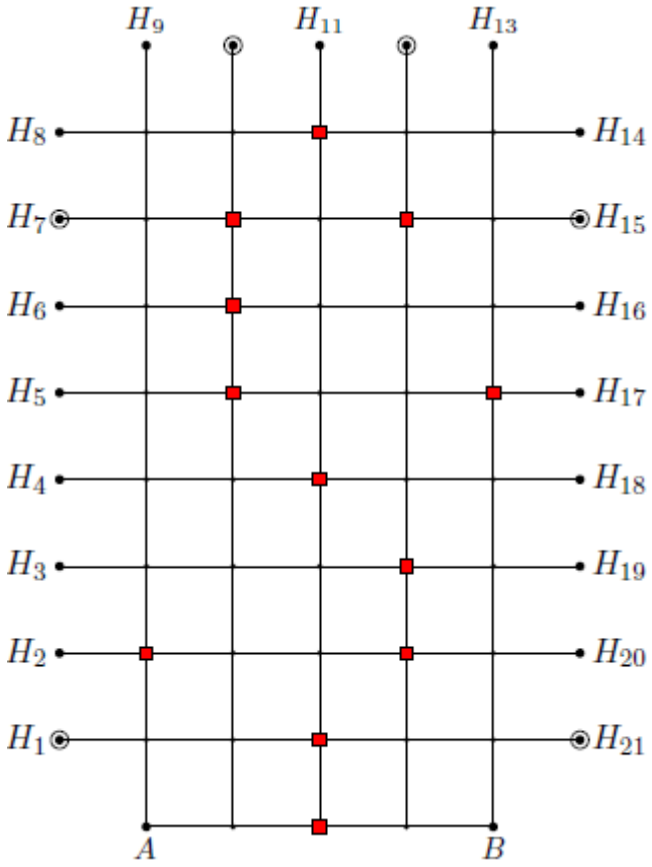


Abbildung 10: Fahrauftrag FA1

Dieses Feld wird dann in eine zweidimensionale Matrix abgebildet (Abbildung 11). Dabei wird für jedes „F“ und jeden „.“ eine -1, für jedes „X“ eine -2 und für den Startpunkt, der vorher ausgewählt wird, eine 0 eingetragen.

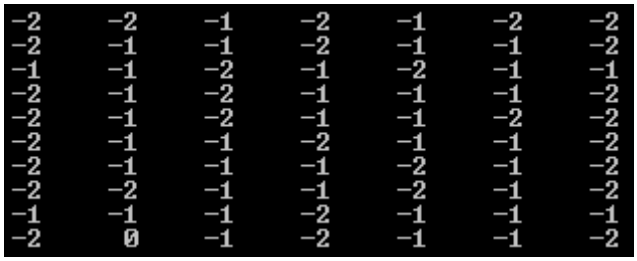


**Abbildung 11: zweidimensionale Matrix**

Das Füllen der Matrix mit den Kosten erfolgt über zwei Schleifen, die bei Position [9][0] starten und horizontal und vertikal durch die Matrix iterieren. Es wird geprüft ob an der Stelle [X][Y] eine positive Zahl (Kosten) steht. Ist das der Fall, dann wird in allen vier Himmelsrichtungen (Norden, Osten, Süden, Westen) geprüft ob dort eine -1 (befahrbare Kreuzung) steht. Diese wird dann mit den Kosten aus der Stelle [X][Y] + 1 ersetzt. Sind in einer der vier Himmelsrichtungen schon Kosten vorhanden die kleiner sind als die an der Stelle [X][Y], dann werden die Kosten an der Stelle [X][Y] durch die Kosten der einen Himmelsrichtung + 1 ersetzt. Die Abbildung 12 zeigt die vollständige Kostenmatrix

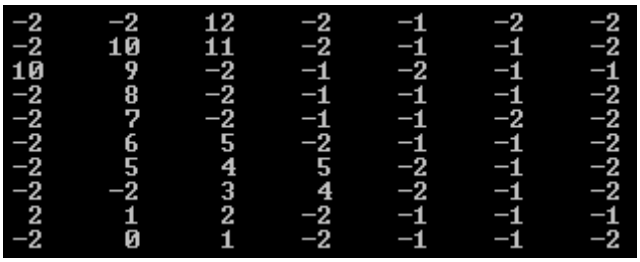


**Abbildung 12: Kostenmatrix**

Implementierung der Breitensuche

Die Breitensuche besteht aus der Wegfindung und der Wegkodierung.

Die Suche startet immer vom Zielpunkt aus und endet erst, wenn der Startpunkt gefunden wurde. Um den kürzesten Weg zu finden, werden die Kosten von dem aktuellen Feld mit den benachbarten Feldern verglichen. Hat ein Feld niedrige Kosten, dann wird diese Position in einem Feld (einzelWeg[]) gespeichert. Dabei wird immer die Position vom Feld _fa[] des Fahrauftrages genommen. Die Position berechnet sich aus der aktuellen Position + 1 für das rechte Feld, -1 für das linke Feld, +7 für das untere Feld und -7 für das obere Feld. Zum besseren Verständnis wird in Abbildung 13 ein Beispiel gezeigt.

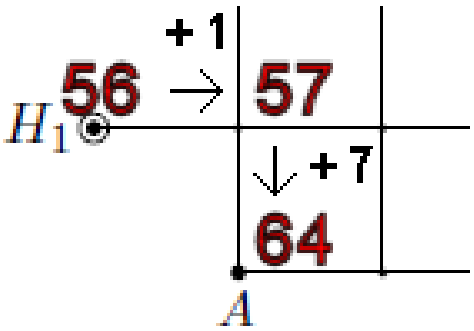


Abbildung 13: Positionsberechnung

Im Anschluss der Suche steht im Feld einzelWeg[] der optimale Weg vom Zielpunkt zum Startpunkt. Für die Steuerung des Roboters müssen die Werte im Feld einzelWeg[] noch kodiert werden. Das Feld wird von hinten nach vorne durchlaufen. Der aktuelle Inhalt wird dann mit dem Inhalt des Vorgängers subtrahiert. Das Ergebnis wird überprüft und mithilfe der aktuellen Ausrichtung(Himmelsrichtung) kodiert. Das Zeichen wird dann ins Feld einzelWegZeichen[] geschrieben. Die Kodierung des Weges zum Zielpunkt endet mit einem Z. In der Abbildung 14 werden alle möglichen Kombinationen dargestellt.

| Start zum Ziel | | | | |
|---|---|---|---|---|
| | Norden | Osten | Süden | Westen |
| 1 (1. Ziel) | L | ----- | R | V |
| 1 (2.-3. Ziel) | R | ----- | L | H |
| (-)1 (1. Ziel) | R | V | L | ----- |
| (-)1 (2.-3. Ziel) | L | H | R | ----- |
| 7 (1. Ziel) | V | L | ----- | R |
| 7 (2.-3. Ziel) | H | R | ----- | L |
| (-)7 (1. Ziel) | ----- | R | V | L |
| (-)7 (2.-3. Ziel) | ----- | R | H | L |
| | | | | |
| Legende: | L | Links | | |
| | R | Rechts | | |
| | V | Vor | | |
| | H | Drehung | | |

Abbildung 14: Wegkodierung zum Ziel

Für die Wegkodierung vom Ziel zum Start wird das Feld einzelWegZeichen[] von vorne nach hinten durchlaufen. Ist es das 1. Ziel, dann wird R und L vertauscht. Ist es das 2. oder 3. Ziel und das erste Zeichen, dann bleibt die Kodierung so wie sie ist. Bei den nachfolgenden Zeichen wird R und L vertauscht. Bei allen Zielen wird beim vorletzten Zeichen nur das V für ein H ersetzt, damit der Roboter am Startpunkt sich dreht. Am Ende wird für den Startpunkt ein S geschrieben.

Der kodierte Fahrplan, für alle Ziele, wird in dem Feld gesamtWeg[] hinterlegt. Für den Fahrauftrag FA1 ergibt sich dann folgende Kodierung:

VRLVVLRVVVRLZLLVVVLRVVRLVSHRLVVLRVVLZLVVLRVVRLVSHLZLVS

Hinweis:

- beim Erreichen eines Zieles fährt der Roboter rückwärts zur nächsten Kreuzung
- beim Startpunkt dreht sich der Roboter um 180°

## 6.2 Implementierung der Steuerung

Die Steuerung des Roboters unterteilt sich in 2 Arten:

- fahren nach Linie
- fahren nach Zeichen

Fahren nach Linie

Der Roboter führt diesen Zyklus ständig aus, wenn er nicht gerade durch eine Kreuzung, Ziel oder Startpunkt unterbrochen wird. Er versucht ständig auf der Linie zu bleiben. Dafür verwendet er die Werte von den Optokopplern 2 und 4. Steigt der Wert von Optokoppler 2, dann kommt der Roboter links von der Linie ab. Steigt der Wert von Optokoppler 4, dann kommt der Roboter rechts von der Linie ab. Beim Gegenlenken gibt es 2 Stufen.

Stufe 1: Liegt der Wert unter 10, dann wird die Geschwindigkeit der Motoren auf 9, 10 bzw. 10, 9 gesetzt.

Stufe 2: Liegt der Wert zwischen 10 und 80, dann wird die Geschwindigkeit der Motoren auf 10, 5 bzw. 5, 10 gesetzt.

Stufe 3: Ist der Wert größer 80, dann wird die Geschwindigkeit der Motoren auf 10, 0 bzw. 0, 10 gesetzt.

Fahren nach Zeichen

Durch die Optokoppler 1 und 5 erkennt der Roboter ob er eine Kreuzung erreicht hat. Ist dies der Fall, dann wird ein Zeichen aus dem Feld gesamtWeg[] ausgelesen. Anhand des Zeichens wird die passende Funktion aufgerufen.

- L = links(char Ziel)
- R = rechts(char Ziel)
- V = geradeaus()
- Z
- S

In der Funktion links(char Ziel) wird das linke Rad gestoppt und das rechte Rad fährt mit eine Geschwindigkeit von 10 weiter. Der Roboter dreht sich solange nach links, bis der Wert von Optokoppler 4 größer 50 ist. Ist es die letzte Kreuzung vor dem Ziel,

dann muss der Wert von Optokoppler 8 größer 50 sein. Die Funktion rechts(char Ziel) ist identisch mit der Funktion links(char Ziel), nur das man nach rechts fährt. Nach der Funktion wird der Standardzyklus „Linie folgen“, die auch durch die Funktion geradeaus() aufgerufen wird, ausgeführt.

Es gibt zwei Spezialfälle, Z und S. Ist der Inhalt von gesamtWeg[Index +1] ein Z, so wird der Wert der Lichtschranke abgefragt. Wenn der Wert 1 ist, dann greift er das Objekt und fährt rückwärts bis zur nächsten Kreuzung. Das S bedeutet, dass er nach der Kreuzung anhält, den Passagier freigibt und bis zur nächsten Kreuzung zurückfährt.

Der Roboter hält automatisch, wenn alle Zeichen im Feld gesamtWeg[] ausgelesen wurden.

**Literaturverzeichnis**

(12. Januar 2014). Abgerufen am 12. Januar 2014 von Wikipedia: http://commons.wikimedia.org/wiki/File:ULTra_001.jpg

(12. Januar 2014). Abgerufen am 12. Januar 2014 von Wikipedia: http://de.wikipedia.org/wiki/Personal_Rapid_Transit

**Abbildungsverzeichnis**

**Tabellenverzeichnis**

## Quellcode

### hellowold.c

```
/*****************************
*  Autoren:                 *
*  Dietrich,     Benjamin   *
*  Wildenhein,  Rico        *
*****************************/

#include "Steuerung.h"
#include <stub.h>

// ******************************
// Einlesen eines Fahrauftrages:
// Aendern des #define auf den gewuenschten Fahrauftrag, bspw. '#define FA4' oder '#define FA12'
// -> der Fahrauftrag befindet sich in der Variablen _fa vom Typ 'unsigned char[71]'
// -> die Nummer des Fahrauftrags, bspw. 4 oder 12, befindet sich in der Variablen _fa_nr vom Typ 'unsigned char'
// und sollte zur Kontrolle angezeigt werden - Ausgabe bspw. mit 'lcd_ubyte(_fa_nr);'
#define FA4
#include "fa.h"
#include "functions.h"
// ******************************

// 10 Zeilen, 7 Spalten
char const maxSpalten = 6;
char const maxZeilen = 9;
char Matrix[10][7];
char ziele[8];
char i,j,counterI, counterJ;
char count = 0;
char zurueck_ = 0;
char zurueckinne = 0;
char anzFahrgaeste = 0;
char countStarts = 0;
char countPath = 1;
char counterKreuzung = 0;

char wegWert = 0;

enum Boolean sucheWeiter = TRUE;
enum Boolean keinWegGefunden = TRUE;


char startRechtsLinks = 0;

char xAxis;
char yAxis;

char einzelWeg[25];
char einzelWegZeichen[25];
char einzelWegZurueck[25];
char gesamtWeg[150];

char anzahl = 0;
char counter = 0;

char vor = 1;

char ueberKreuzung = 1;

char lichtschrankeAn = 0;
char gefunden = 0;

char startsignal = 0;

char schrankeOffen = 0;
```

```
//Hauptprogrammroutine
void AksenMain(void)
{

lcd_cls();

// Startpunkt setzen
// Pin1 = 0 = auf 'ON'
// Pin1 = 1 = auf '1'
// Links = 1, Rechts = 5
if (dip_pin(0) == 0)
{
startRechtsLinks = 1;
lcd_puts("Links ");
}
else
{
startRechtsLinks = 5;
lcd_puts("Rechts ");
}

// Array fllen
for(i = 0; i < 10; i++)
{
for(j = 0; j < 7; j++)
{
if ((i == 9 && j == startRechtsLinks))
{
Matrix[i][j] = 0;
}
else
{
if (_fa[count] == '.' || _fa[count] == 'F')
Matrix[i][j] = -1;
else if (_fa[count] == 'x')
Matrix[i][j] = -2;

if (_fa[count] == 'F')
{
ziele[anzFahrgaeste] = count;
anzFahrgaeste ++;
}


}

count ++;
}
}

// Array mit Zahlen fllen, bei [9][0] starten
// i = oben/unten
// j = rechts/links

zurueck_ = 0;

for(i = 9; i >= 0; i--)
{
if (zurueck_ == 1)
{
counterI = 9-i;
}
else
counterI = i;
```

```
zurueckinne = 0;

for(j = 0; j < 7; j++)
{
if ( zurueckinne == 1)
{
counterJ = 6 - j;
}
else
counterJ = j;


if (Matrix[counterI][counterJ] >= 0)
{

// 1. nach rechts schauen
if (counterJ < maxSpalten)
{
if (Matrix[counterI][counterJ + 1] == -1)
Matrix[counterI][counterJ + 1] = Matrix[counterI][counterJ] + 1;
else if (Matrix[counterI][counterJ + 1] >= 0)
{
if ( Matrix[counterI][counterJ] > (Matrix[counterI][counterJ + 1] + 1))
Matrix[counterI][counterJ] = Matrix[counterI][counterJ + 1] + 1;
}
}
// 2. nach links schauen
if (counterJ > 0)
{
if (Matrix[counterI][counterJ - 1] == -1)
Matrix[counterI][counterJ - 1] = Matrix[counterI][counterJ] + 1;
else if (Matrix[counterI][counterJ - 1] >= 0 )
{
if ( Matrix[counterI][counterJ] > (Matrix[counterI][counterJ - 1] + 1))
Matrix[counterI][counterJ] = Matrix[counterI][counterJ - 1] + 1;
}
}
// 3. nach unten schauen
if (counterI < maxZeilen)
{
if (Matrix[counterI + 1][counterJ] == -1)
Matrix[counterI + 1][counterJ] = Matrix[counterI][counterJ] + 1;
else if (Matrix[counterI + 1][counterJ] >= 0)
{
if ( Matrix[counterI][counterJ] > (Matrix[counterI + 1][counterJ] + 1))
Matrix[counterI][counterJ] = Matrix[counterI + 1][counterJ] + 1;
}

}
// 4. nach Oben schauen
if (counterI > 0)
{
if (Matrix[counterI - 1][counterJ] == -1)
Matrix[counterI - 1][counterJ] = Matrix[counterI][counterJ] + 1;
else if (Matrix[counterI - 1][counterJ] >= 0)
{
if ( Matrix[counterI][counterJ] > (Matrix[counterI - 1][counterJ] + 1))
Matrix[counterI][counterJ] = Matrix[counterI - 1][counterJ] + 1;
}
}


}

if (j == 6 && zurueckinne != 1)
{
zurueckinne = 1;
j = 0;
}
```

```
}

if (i == 0 && zurueck_ != 1)
{
zurueck_ = 1;
i = 9;
}



}

// Informierte Breitensuche
// vom Ziel zum Startpunkt !!!

// probiere alle Ziele aus
for (i = 0; i<anzFahrgaeste; i++)
{
// Wenn count 0 dann starte ich beim Ziel
count = 0;
sucheWeiter = TRUE;
// fge Ziel zum Weg hinzu
addCharToArray(einzelWeg, ziele[i], myStrlen(einzelWeg));

// Schaue jetzt solange bis man eine 0 findet, sonst l□□en
while(sucheWeiter)
{
keinWegGefunden = TRUE;
// berechne x-y-Koordinaten
calcXYAxis(einzelWeg[count], &xAxis, &yAxis);

// oben
if (yAxis > 0)
{
if ((Matrix[yAxis - 1][xAxis] != -2) && (_fa[XYAxisToOne(xAxis, yAxis-1)] != 'F') && Matrix[yAxis - 1][xAxis] <
Matrix[yAxis][xAxis])
{
keinWegGefunden = FALSE;
wegWert = einzelWeg[count]-7;
}
}
// unten
if (yAxis < maxZeilen)
{
if ((Matrix[yAxis + 1][xAxis] != -2) && (_fa[XYAxisToOne(xAxis, yAxis+1)] != 'F') && Matrix[yAxis + 1][xAxis] <
Matrix[yAxis][xAxis])
{
keinWegGefunden = FALSE;
wegWert = einzelWeg[count]+7;
}
}
// links
if (xAxis > 0)
{
if ((Matrix[yAxis][xAxis - 1] != -2) && (_fa[XYAxisToOne(xAxis-1, yAxis)] != 'F') && Matrix[yAxis][xAxis - 1] <
Matrix[yAxis][xAxis])
{
keinWegGefunden = FALSE;
wegWert = einzelWeg[count]-1;
}
}
// rechts
if (xAxis < maxSpalten)
{
if ((Matrix[yAxis][xAxis + 1] != -2) && (_fa[XYAxisToOne(xAxis+1, yAxis)] != 'F') && Matrix[yAxis][xAxis + 1] <
Matrix[yAxis][xAxis])
{
```

```
keinWegGefunden = FALSE;
wegWert = einzelWeg[count]+1;
}
}

if (Matrix[yAxis][xAxis] == 0)
{
sucheWeiter = FALSE;
keinWegGefunden = TRUE;
//Weg zum Ziel
getPathToTarget(einzelWegZeichen,myStrlen(einzelWeg),einzelWeg, countStarts);
// Weg zurck
getPathToStart(einzelWegZurueck, myStrlen(einzelWegZeichen), einzelWegZeichen, countPath);
//ganzer Weg
myStrcat_s(gesamtWeg,25,einzelWegZeichen);
myStrcat_s(gesamtWeg,25,einzelWegZurueck);
countStarts ++;
countPath = 0;
}

// Feld l□□en
if (keinWegGefunden == TRUE)
{
clearArray(einzelWeg, myStrlen(einzelWeg));
clearArray(einzelWegZeichen, myStrlen(einzelWegZeichen));
clearArray(einzelWegZurueck, myStrlen(einzelWegZurueck));
sucheWeiter = FALSE;
}
// Element hinzufgen
else
{
addCharToArray(einzelWeg, wegWert, myStrlen(einzelWeg));
count ++;
}

}


}


// Suche fertig !!!!

// 70 = auf, 140 = zu
servo_arc(0,70);
motor_richtung(3,0);
motor_richtung(2,0);

anzahl = myStrlen(gesamtWeg);
//anzahl = 14;
//lcd_cls();
lcd_puts("Start!");

// schon vorher einschalten
led(3,1);


while(1)
{
if(digital_in(0) == 0)
{
startsignal = 1;
}

if(startsignal == 1)
{

if ((lichtschrankeAn == 1) && (digital_in(7)))
```

```
{

motor_pwm(2,0);
motor_pwm(3,0);
servo_arc(0,140);
sleep(300);
vor = 0;
lichtschrankeAn = 0;

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

}


if(counter > anzahl)
{

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

gefunden = 1;

//fertig!!!
break;
}

else if (gefunden != 1)
fahren(vor);
else
{
motor_pwm(2,0);
motor_pwm(3,0);
}

if (kreuzung() == 0)
ueberKreuzung = 1;

if (kreuzung() == 1 && ueberKreuzung == 1)
{

if (schrankeOffen == 1)
{
servo_arc(0, 70);
schrankeOffen = 0;
}

// Ziel gefunden
if(gesamtWeg[counter +1] == 'Z')
{
lichtschrankeAn = 1;
}

ueberKreuzung = 0;
// nur zum testen, b geh□□nicht in die fahrliste!!!
// vor muss auf 1 gesetzt werden
if (gesamtWeg[counter] == 'b')
vor = 0;
else
vor = 1;

// bin am Startpunkt
if(gesamtWeg[counter +1] == 'S')
{


if(gesamtWeg[counter] == 'V')
{
```

```
motor_pwm(3,6);
motor_pwm(2,5);
sleep(500);

}
else
{
// erst anhalten
stop();
steuern(gesamtWeg[counter],1);
}
motor_pwm(3,0);
motor_pwm(2,0);
sleep(200);
servo_arc(0, 105);
schrankeOffen = 1;
sleep(200);

motor_pwm(2,9);
motor_pwm(3,10);

counter ++;
vor = 0;

}
else
{
if(gesamtWeg[counter +1] == 'Z')
if(gesamtWeg[counter] != 'V')
// erst anhalten
stop();


//steuern(gesamtWeg[counter],(gesamtWeg[counter +1] == 'Z'));

if((counter > 1) && gesamtWeg[counter -1] == 'S')
{
// Stopp frs Rckw鴎sfahren
stop2();
}


steuern(gesamtWeg[counter],0);

}

if(gesamtWeg[counter +1] == 'Z')
counter ++;

counter ++;

}
}
}

}
```

## Functions.h

```
/*****************************
 *  Autoren:                 *
 *  Dietrich,    Benjamin    *
 *  Wildenhein,  Rico        *
 *****************************/



#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <stdlib.h>
#include <stdio.h>


//#define Schwarz-,Wei□Werte

enum Boolean
{
   FALSE = 0,
   TRUE,
};

enum Direction
{
   NORTH = 0,
   EAST,
   SOUTH,
   WEST,
};


// die Funktion berechnet die X und Y Koordinate
// und bergibt diese den Pointer
void calcXYAxis(char aValue, char *xAxis, char *yAxis);




// die Funktion berechnet aus der X und Y Koordinate
// eine Zahl zwischen 0 und 69 und gibt diese zurck
int XYAxisToOne(char xAxis, char yAxis);

// die Funktion fuegt ein Char zu einem Char-Array hinzu
// und gibt dieses zurck
void addCharToArray(char *path, char aValue, char aStrLen);

// die Funktion setzt alle Felder eines Arrays auf NULL
void clearArray(char *path, char aStrLen);

// die Funktion gibt die Wegrichtung zurck
// aToNewPosition:  0 = North
//                  1 = EAST
//                  2 = SOUTH
//                  3 = WEST
char getDirection(enum Direction aDirection, int aToNewPosition);

// die Funktion erstellt den Weg zum Ziel
// und fgt es dem Array hinzu
void getPathToTarget (char *strDestination, char numberOfElements, const char *strSource, const char type);

// die Funktion erstellt den Weg zum Startpunkt
// und fgt es dem Array hinzu
void getPathToStart (char *strDestination, char numberOfElements, const char *strSource, const char firstPath);
```

```
// die Funktion gibt die Anzahl der Zeichen zurck
char myStrlen (char *str);

// die Funktion kopiert 2 Strings, mit einer Anzahl von Elementen
void myStrncpy_s (char *strDestination,
                  char numberOfElements,
                  const char *strSource );

// die Funktion verbindet 2 Strings
void myStrcat_s(char *strDestination,
                char numberOfElements,
                const char *strSource);

#endif
```

## Functions.c

```
/*****************************
*  Autoren:                 *
*  Dietrich,     Benjamin   *
*  Wildenhein,  Rico        *
*****************************/

#include "functions.h"


void calcXYAxis(char aValue, char *xAxis, char *yAxis)
{
*xAxis = (char) (aValue % 7);
*yAxis = (char) (aValue / 7);
}

int XYAxisToOne(char xAxis, char yAxis)
{
return (yAxis * 7) + xAxis;
}

void addCharToArray(char *path, char aValue, char aStrLen)
{
if(aStrLen <= 25)
{
path[aStrLen] = aValue;
}
}

void clearArray(char *path, char aStrLen)
{
char i;

for(i=0; i<aStrLen; i++)
{
path[i] = '\0';
}
}

char getDirection(enum Direction aDirection, int aToNewPosition)
{
if (aDirection == NORTH)
{
if (aToNewPosition == NORTH)
return 'V';
if (aToNewPosition == EAST)
return 'R';
//if (aToNewPosition == SOUTH)
//  return ' ';
if (aToNewPosition == WEST)
return 'L';
}
```

```
if (aDirection == EAST)
{
if (aToNewPosition == NORTH)
return 'L';
if (aToNewPosition == EAST)
return 'V';
if (aToNewPosition == SOUTH)
return 'R';
//if (aToNewPosition == WEST)
//  return ' ';
}

if (aDirection == SOUTH)
{
//if (aToNewPosition == NORTH)
//  return ' ';
if (aToNewPosition == EAST)
return 'L';
if (aToNewPosition == SOUTH)
return 'V';
if (aToNewPosition == WEST)
return 'R';
}

if (aDirection == WEST)
{
if (aToNewPosition == NORTH)
return 'R';
//if (aToNewPosition == EAST)
//  return ' ';
if (aToNewPosition == SOUTH)
return 'L';
if (aToNewPosition == WEST)
return 'V';
}

return ' ';
}

char myStrlen (char *str)
{
char len = 0;
while (*str != '\0')
{
str++;
len++;
}
return len;
}

void myStrncpy_s (char *strDestination,
char numberOfElements,
const char *strSource )
{
while (numberOfElements > 0 && *strSource != '\0')
{
*strDestination = *strSource;
strDestination ++;
strSource ++;
numberOfElements --;
}

if (*strSource == '\0')
{
*strDestination = '\0';
}
}
```

```
void myStrcat_s(char *strDestination,
char numberOfElements,
const char *strSource)
{
// Ende finden
while (*strDestination != '\0')
strDestination ++;

// kopieren
while (numberOfElements > 0 && *strSource != '\0')
{
*strDestination = *strSource;
strDestination ++;
strSource ++;
numberOfElements --;
}

if (*strSource == '\0')
{
*strDestination = '\0';
}
}

void getPathToTarget (char *strDestination, char numberOfElements, const char *strSource, const char type)
{
char i;
enum Direction path = NORTH;

for(i = numberOfElements-1; i > 0; i--)
{
if ((strSource[i] - strSource[i-1]) == 1)
{
if (( i == numberOfElements -1 )  && type > 0)
{
// es gibt kein EAST, weil er nicht rckw鴆s f韃t
if (path == NORTH)
*strDestination = 'R';
else if (path == SOUTH)
*strDestination = 'L';
else if (path == WEST)
*strDestination = 'H';
}
else
{
// es gibt kein EAST, weil er nicht rckw鴆s f韃t
if (path == NORTH)
*strDestination = 'L';
else if (path == SOUTH)
*strDestination = 'R';
else if (path == WEST)
*strDestination = 'V';
}

strDestination ++;
path = WEST;
continue;
}

if ((strSource[i] - strSource[i-1]) == -1)
{
if (( i == numberOfElements -1 )  && type > 0)
{
// es gibt kein WEST, weil er nicht rckw鴆s f韃t
if (path == NORTH)
*strDestination = 'L';
else if (path == SOUTH)
*strDestination = 'R';
else if (path == EAST)
```

```
*strDestination = 'H';
}
else
{
// es gibt kein WEST, weil er nicht rckw鴆s f黶t
if (path == NORTH)
*strDestination = 'R';
else if (path == SOUTH)
*strDestination = 'L';
else if (path == EAST)
*strDestination = 'V';
}

strDestination ++;
path = EAST;
continue;
}

if ((strSource[i] - strSource[i-1]) == 7)
{
if (( i == numberOfElements -1 )  && type > 0)
{
// es gibt kein SOUTH, weil er nicht rckw鴆s f黶t
if (path == NORTH)
*strDestination = 'H';
else if (path == WEST)
*strDestination = 'L';
else if (path == EAST)
*strDestination = 'R';
}
else
{
// es gibt kein SOUTH, weil er nicht rckw鴆s f黶t
if (path == NORTH)
*strDestination = 'V';
else if (path == WEST)
*strDestination = 'R';
else if (path == EAST)
*strDestination = 'L';
}

strDestination ++;
path = NORTH;
continue;
}

if ((strSource[i] - strSource[i-1]) == -7)
{
if (( i == numberOfElements -1 )  && type > 0)
{
// es gibt kein NORTH, weil er nicht rckw鴆s f黶t
if (path == SOUTH)
*strDestination = 'H';
else if (path == WEST)
*strDestination = 'R';
else if (path == EAST)
*strDestination = 'L';
}
else
{
// es gibt kein NORTH, weil er nicht rckw鴆s f黶t
if (path == SOUTH)
*strDestination = 'V';
else if (path == WEST)
*strDestination = 'L';
else if (path == EAST)
*strDestination = 'R';
}
```

```
strDestination ++;
path = SOUTH;
continue;
}

}

*strDestination = 'Z';
}

void getPathToStart (char *strDestination, char numberOfElements, const char *strSource, const char firstPath)
{
char i;

for(i = numberOfElements-1; i >= 0; i--)
{
if(strSource[i] == 'Z')
continue;

if (i == numberOfElements - 2)
{
if (strSource[i] == 'L')
*strDestination = 'L';
else if (strSource[i] == 'R')
*strDestination = 'R';
else if (strSource[i] == 'V')
*strDestination = 'H';
}

else if (firstPath == 1)
{
if (strSource[i] == 'R')
*strDestination = 'L';
else if (strSource[i] == 'L')
*strDestination = 'R';
else
*strDestination = 'V';
}
else
{

if (i == 0)
{
if (strSource[i] == 'R')
*strDestination = 'R';
else if (strSource[i] == 'L')
*strDestination = 'L';
else
*strDestination = 'V';
}
else
{
if (strSource[i] == 'R')
*strDestination = 'L';
else if (strSource[i] == 'L')
*strDestination = 'R';
else
*strDestination = 'V';

}
}


strDestination ++;
}

*strDestination = 'S';
}
```

## Steuerung.h

```
/*****************************
 *  Autoren:                 *
 *  Dietrich,    Benjamin    *
 *  Wildenhein,  Rico        *
 *****************************/

#ifndef STEUERUNG_H
#define STEUERUNG_H

//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>

void steuern(unsigned char richtung, char ziel);
void geradeaus();
void links(char ziel);
void rechts(char ziel);
void zurueck();
void drehung();
void fahren(char vor);
void stop();
void stop2();
unsigned int kreuzung();
void ausrichten(unsigned char richtung);


#endif
```

## Steuerung.c

```
/*****************************
*  Autoren:                 *
*  Dietrich,     Benjamin   *
*  Wildenhein,  Rico        *
*****************************/

#include "Steuerung.h"

void steuern(unsigned char richtung, char ziel)
{
if (richtung == 'V')
{
motor_richtung(3,0);
motor_richtung(2,0);
geradeaus();
}
if (richtung == 'L')
{
motor_richtung(3,0);
motor_richtung(2,0);
links(ziel);
geradeaus();
}
if (richtung == 'R')
{

motor_richtung(3,0);
motor_richtung(2,0);
rechts(ziel);
geradeaus();
}
if (richtung == 'B')
{
motor_richtung(2,1);
motor_richtung(3,1);
zurueck();
}

if (richtung == 'H')
{
drehung();
}
}

void fahren(char vor)
{
if (vor == 1)
{
motor_richtung(3,0);
motor_richtung(2,0);
geradeaus();
}
else
{
motor_richtung(3,1);
motor_richtung(2,1);
zurueck();
}
}

void geradeaus(){

// Ausrichtung
// OK

if((analog(2) <= 10 && analog(6) <= 10) ||
```

```
(analog(2) > 30 && analog(6) > 30  ))
{
motor_pwm(2,9);
motor_pwm(3,10);
}
// von links nach rechts
// leicht
if(analog(2) > 10 && analog(2) < 30)
{
motor_pwm(2,10);
motor_pwm(3,5);
}
// stark
if(analog(2) > 30 && analog(2) < 80)
{
motor_pwm(2,10);
motor_pwm(3,5);
}
// sehr stark
if(analog(2) >= 80)
{
motor_pwm(2,10);
motor_pwm(3,0);
}
// von rechts nach links
// leicht
if(analog(6) > 10 && analog(6) < 30)
{
motor_pwm(3,10);
motor_pwm(2,6);
}
// mittel
if(analog(6) > 30 && analog(6) < 50)
{
motor_pwm(3,10);
motor_pwm(2,6);
}
// stark
if(analog(6) > 50 && analog(6) < 80)
{
motor_pwm(3,10);
motor_pwm(2,5);
}
// sehr stark
if(analog(6) >= 80)
{
motor_pwm(3,10);
motor_pwm(2,0);
}

}

void links(char ziel)
{
motor_richtung(2,0);
motor_richtung(3,0);
motor_pwm(2,0);
motor_pwm(3,10);


// test 4
motor_richtung(2,1);
motor_pwm(2,10);
sleep(50);
motor_richtung(2,0);
motor_pwm(2,0);


sleep(500);
```

```
// Warten
if (ziel == 1){
while (analog (8) < 50);
}
else
while (analog (4) < 50);  // 100

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

void rechts(char ziel)
{
motor_richtung(2,0);
motor_richtung(3,0);
motor_pwm(2,10);
motor_pwm(3,0);


motor_richtung(3,1);
motor_pwm(3,10);
sleep(50);
motor_richtung(3,0);
motor_pwm(3,0);


// von der Linie fahren ohne Schleife
sleep(500);

// Warten
if (ziel == 1){
while (analog (10) < 50);
//stop();
}
else
while (analog (4) < 50);  // statt 100

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


}

unsigned int kreuzung()
{
// OK
if(analog(0) > 80 && analog(7) > 80)
return 1;
else
return 0;
}

void drehung()
{
// 1 = zurck
// 0 = vor
// OK
motor_richtung(2,1);
motor_richtung(3,0);

motor_pwm(2,8);
motor_pwm(3,8);

sleep(500);

while(analog(4) < 80);

motor_richtung(2,0);
```

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

void zurueck()
{
// OK
if((analog(8) <= 10 && analog(10) <= 10) ||
(analog(8) > 30 && analog(10) > 30  ))
{
motor_pwm(2,9);
motor_pwm(3,10);
}
// von links nach rechts
// leicht
else if(analog(8) > 10 && analog(8) < 30)
{
motor_pwm(2,8);
motor_pwm(3,6);
}
// stark
else if(analog(8) > 30 && analog(8) < 80)
{
motor_pwm(2,8);
motor_pwm(3,4);
}
// sehr stark
else if(analog(8) >= 80)
{
motor_pwm(2,10);
motor_pwm(3,0);
}
// von rechts nach links
// leicht
else if(analog(10) > 10 && analog(10) < 30)
{
motor_pwm(3,8);
motor_pwm(2,6);
}
// start
else if(analog(10) > 30 && analog(10) < 80)
{
motor_pwm(3,8);
motor_pwm(2,4);
}
// sehr stark
else if(analog(10) >= 80)
{
motor_pwm(3,10);
motor_pwm(2,0);
}
}

void stop()
{
// Test Stop
motor_richtung(3,1);
motor_richtung(2,1);
motor_pwm(2,10);
motor_pwm(3,10);

sleep(80);

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

}
```

```
void stop2()
{
// Test Stop
motor_richtung(3,0);
motor_richtung(2,0);
motor_pwm(2,10);
motor_pwm(3,10);

sleep(80);

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

}

void ausrichten(unsigned char richtung)
{
if (richtung == 'R')
{

if ((analog(8) > 100))
{
while (analog(8) < 50)
{
// zurck fahren
motor_richtung(3,1);
motor_richtung(2,1);
motor_pwm(2,10);
motor_pwm(3,10);
}

} else
{
motor_richtung(2,1);
motor_pwm(2,10);
motor_pwm(3,0);

lcd_ubyte(analog(8));

while ((analog(8) < 50))
{
lcd_puts("R");
}
}

motor_pwm(2,0);
}

if (richtung == 'L')
{
motor_richtung(3,1);
motor_pwm(2,0);
motor_pwm(3,10);

while ((analog(10) > 100))
{
lcd_puts("L");
}

motor_pwm(3,0);
}
}
```