Dokumentation zum Projekt SS 2013/2014
MASDAR City - Personal Rapid Transit

Sebastian Berndt
20110020
17.01.2014

Fachhochschule
Brandenburg
University of
Applied Sciences

Inhaltsverzeichnis

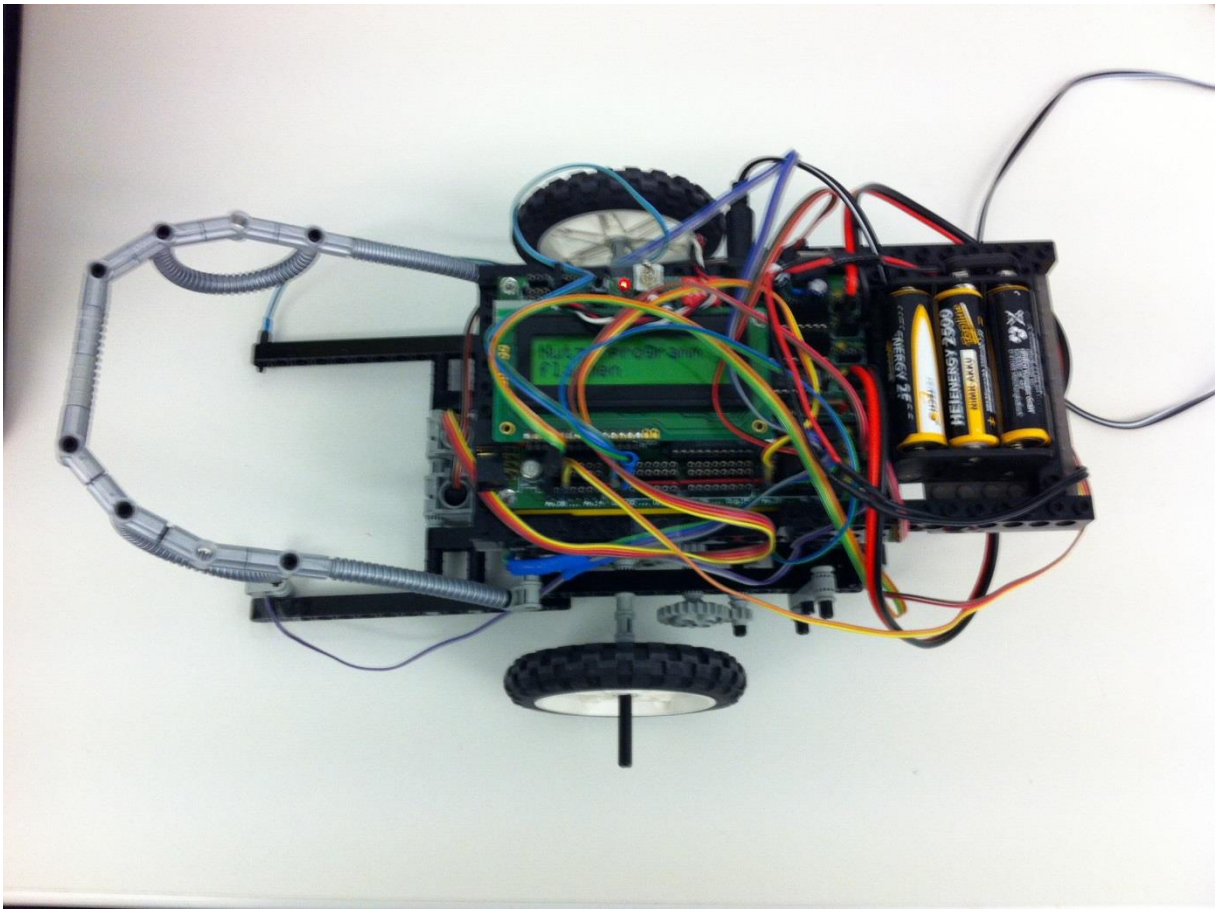
1. Aufgabe

Ihr Roboter erhielt den Auftrag, eine oder mehrere Personen abzuholen und zum Ziel zu bringen. Das Streckennetz ist ein einfaches Gitter, in dem es allerdings zu Störungen und damit unbefahrbaren Kreuzungen kommen kann. Die gute Nachricht ist, das Sie über globales Wissen verfügen und die aktuelle Karte der befahrbaren Wege dem Roboter kurz vor dem Start zur Verfügung gestellt wird - als Papierstreifen.
Ein PRT-System ist eine Flotte kleiner Fahrzeuge, die jeweils eine oder wenige Personen ohne Zwischenhalt zu individuellen Zielen transportieren. Das derzeit größte geplante PRT-Netz mit3000 Fahrzeugen entsteht in Masdar City, einer am Reißbrett entworfenen Stadt in der Wüste der Vereinigten Arabischen Emirate. In Masdar City sollen 50.000 Menschen CO2- und energie-neutral leben und arbeiten. Fahrzeuge mit Verbrennungsmotoren wird es dort nicht geben. 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. Der damit verbundene Verlust befahrbarer Strecken erfordert ein Neuplanen von Fahrtrouten. Und hier kommen Sie ins Spiel.

2. Der Roboter

Der finale Roboter kennzeichnet sich durch eine hohe Geschwindigkeit aus. Leider hat er ebenfalls die daraus resultierenden Probleme der Ungenauigkeit geerbt.

Da seine Fertigstellung erst wenige Stunden vor Wettkampfstart stattfand, konnte er jedoch leider nicht mit höherer Geschwindigkeit punkten, da der Akku nahezu verbraucht war. Ebenfalls war er nicht genau genug und hätte noch einer längeren Testphase bedurft, um zuverlässig alle Passagiere zu holen. Trotzdem waren alle wichtigen Features implementiert und er konnte einige Fahrten ohne Fehler vollbringen.



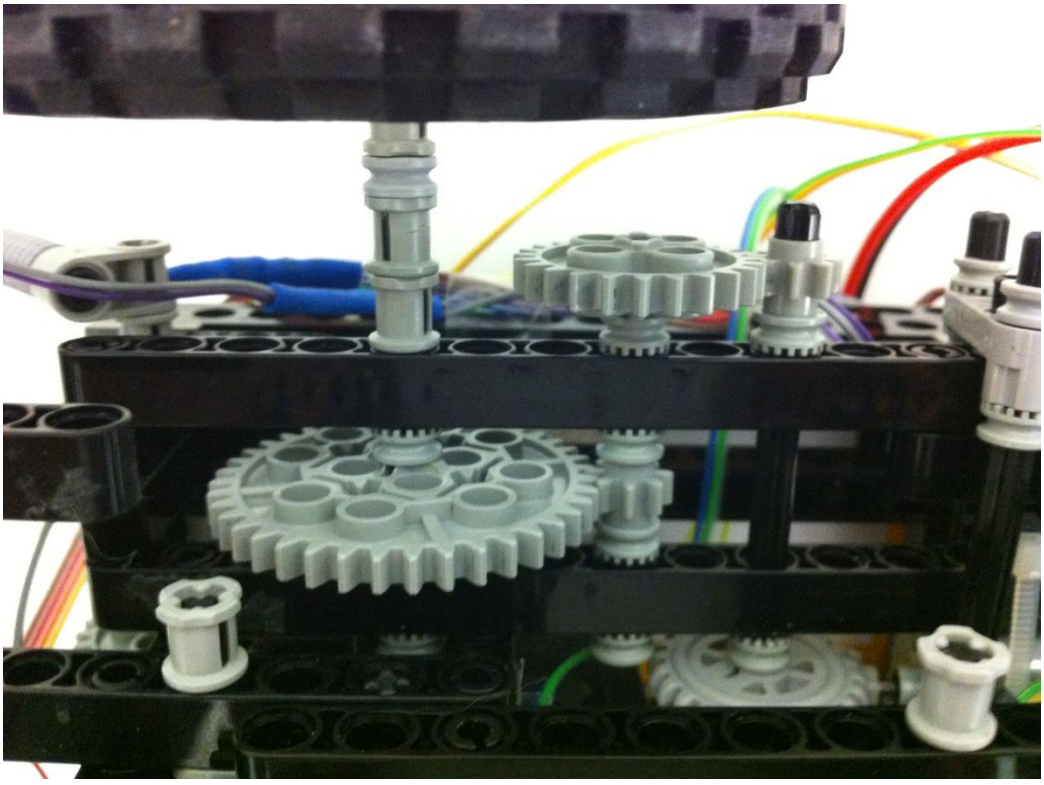
3. Hardware

3.1. Allgemein

Grundlegende Ziele beim Bau des Roboters waren ihn möglichst schnell zu gestalten, notfalls auch auf Kosten der Genauigkeit. Beim Bau des Roboters viel auf, dass vor allem der Rahmen sehr steif sein muss und keinen Spielraum bieten darf. Nur so konnte das Getriebe gleichbleibend leicht betrieben werden. Deshalb wurde auf die Verwendung von Lego-Technik Bauteilen geachtet, da diese etwas leichter bei höherer Passgenauigkeit sind. Das Aksenboard liegt in einem befestigten Modul oben auf, und ist daher leicht zu bedienen. Der Akkumulator wurde lediglich hinten in einem Fach untergebracht um ihn leicht wechseln zu können.

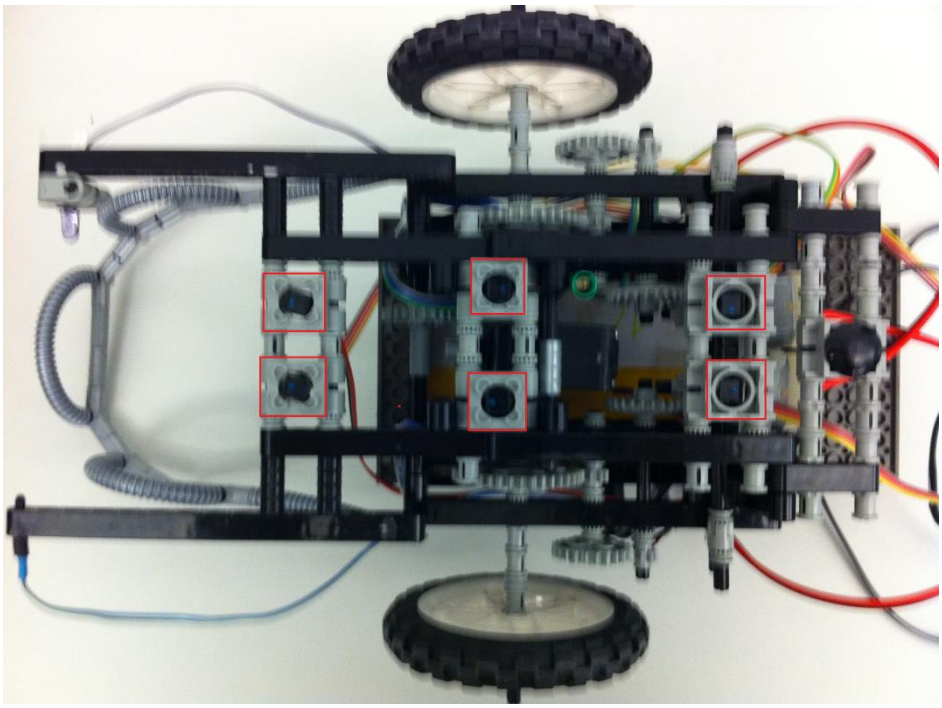
3.2. Antrieb

Das Getriebe war eine der Herausforderungen bei der Erstellung des Roboters und wurde fast während der gesamten Entwicklung weiter verändert. Als finale, jedoch nicht perfekte Lösung, wurden drei Untersetzungen mit einer Gesamtuntersetzung 1:45 realisiert. (1:3,1:3,1:5) Dadurch ist der Roboter insgesamt durchschnittlich schneller als die Konkurrenz, verliert jedoch, bei nachlassender Akkuleistung, auch am schnellsten an Geschwindigkeit. Als Räder wurden die größtmöglichen Legoräder mit gleichzeitig relativ weichem Gummi gewählt, um das Bremsen auf der glatten Wettkampffläche zu vereinfachen und ein aktives Bremsen in Form eines Richtungswechsels der Motoren zu vermeiden.



3.3. Bewegung

Die Anbringung der Sensoren war einer der entscheidendsten Faktoren für die spätere Steuerung und war insbesondere durch den Abstand vom Boden sowie der Abschirmung sehr beeinflussbar. Durch die hohe Geschwindigkeit bei voller Akkuleistung war eine hohe Präzision äußerst schwierig zu erreichen. Deshalb wurden insgesamt 6 Optokoppler verbaut: jeweils zwei vorne, zwei mittig auf der Achse und zwei hinten. Die beiden vorderen Optokoppler sind zum Linienfolgen und zur Erkennung einer abgeschlossenen Drehung. Es wurden verschiedene Abstände für die vorderen Optokoppler ausprobiert. Je weiter die zwei vorne waren umso genauer konnte der Roboter gegensteuern. Um eine adäquate Länge zu halten, ist der endgültige Abstand ca. der Drehradius der zwei Räder. Alle Optokoppler sind durch ein Block-Element umschlossen und so gegen äußere Einflüsse abgeschirmt.

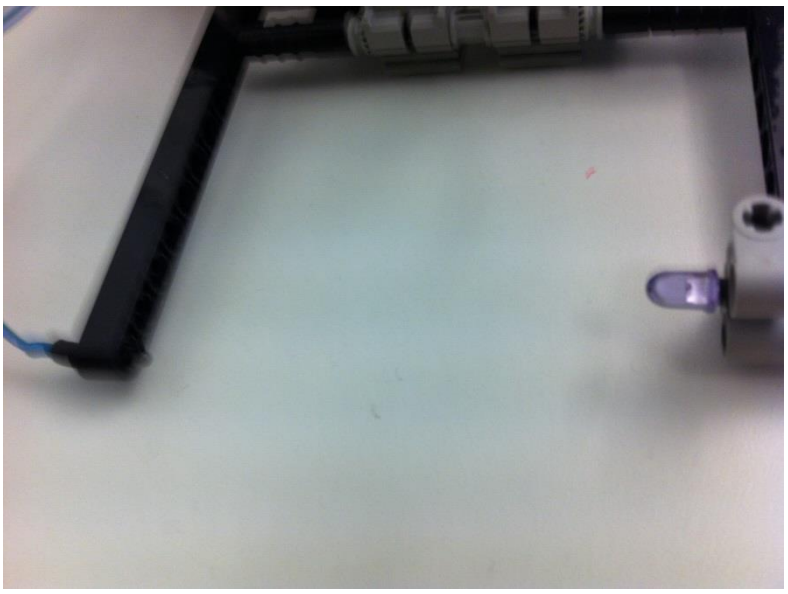


Die zwei mittleren Optos werden lediglich zur Erkennung der Kreuzung genutzt und sind etwas weiter auseinander als die Restlichen, um die Fehlerquote durch falsche Signale zu reduzieren.

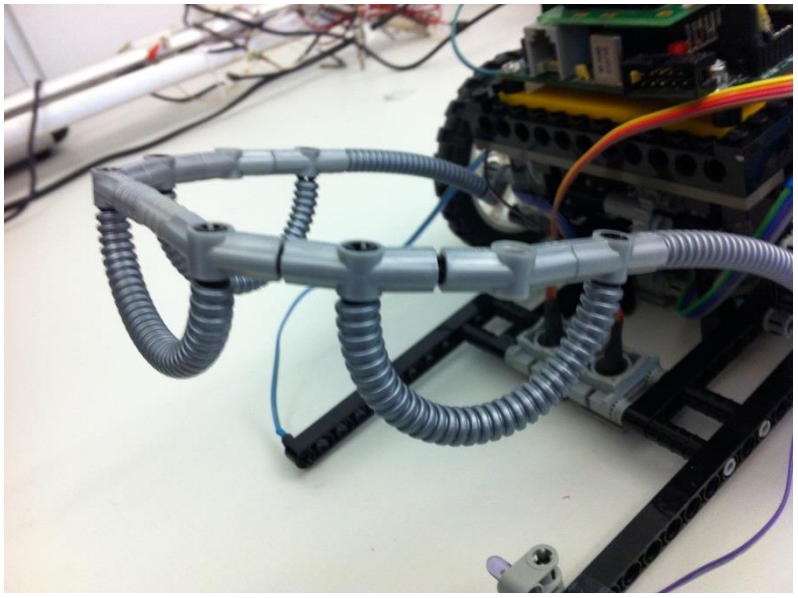
Die zwei hinteren Optokoppler sind zum Rückwärtsfahren nach dem Abholen und Abliefern des Passagiers.

Für die Drehung wurden lediglich die beiden Motoren in entgegengesetzte Richtungen betrieben und die Anzahl der übertretenen schwarzen Linien der beiden vorderen optischen Sensoren ausgewertet. Durch die Nutzung von lediglich zwei Rädern und einer Stütze war das Drehen auf der Stelle sehr einfach und erleichterte damit das generelle Steuern des Roboters. Der Versuch ohne Sleep(); befehle oder zeitgesteuerte Schleifen zu drehen scheiterte und erwies sich als sehr ungenau.

Zum lokalisieren des Fahrgastes wurde eine Lichtschranke bestehend aus einer Infrarot LED und einem entsprechenden Sensor genutzt. Diese Kombination funktioniert auch bei größerem Abstand beider Sensoren äußerst gut.



Der Greifer des Roboters ist einfach und sehr flexibel um einen möglichst großen Spielraum für Fehler zu lassen, sodass selbst bei starkem Anstoßen der Wand, der Fahrgast immer noch abgeholt wird. Etwas negativ viel auf, dass der Greifer ab und zu den Stellplatz der Passagiere mitnimmt.



4. Software

4.1. Robotersteuerung

Der Roboter verfügt lediglich über 5 modulare Bewegungen:

forward()
turn_left()
turn_right()
grab()
deliver()

void forward()
Die Funktion forward(); setzt eine Bewegung von einer Kreuzung zur Nächsten um. Dabei wird erst der Motor entsprechend eingestellt um dann zeitgesteuert von der Kreuzung gefahren und gestoppt sobald beide mittleren Optokoppler wieder Schwarz erkennen. Die beiden vorderen Optokoppler setzen die Motorgeschwindigkeit auf der gleichen Seite herunter, sobald sie schwarz erkennen. Wenn beide vorderen Sensoren Schwarz oder Weiß erkennen, fahren sie mit voller Geschwindigkeit weiter.

void turn_left()/void turn_right()
Für das Drehen werden die zwei vorderen Optokoppler verwendet. Durch die ungewisse Anfangsposition bei Drehbeginn, wird erst einmal durch eine Pause sichergestellt, dass beide Optokoppler immer auf weiß sind. Danach wird auf Das nächste Schwarz-Signal einer der beiden vorderen Optokoppler gewartet.

void grab()
Zum Greifen eines Fahrgastes folgt der Roboter solange der Linie bis die eingebaute Lichtschranke ein Objekt erkennt. Als nächstes wird der Greifer geschlossen und der Roboter fährt er solange Rückwärts bis die beiden mittleren Optokoppler wieder schwarz sind, also die vorherige Kreuzung erkannt haben.

void deliver()

Das Abliefern des Fahrgastes wurde ähnlich wie die Funktion zum greifen umgesetzt. Der Roboter fährt eine Zehntelsekunde vorwärts öffnet den Greifer und fährt wieder zurück zur letzten Kreuzung durch Benutzung der hinteren zwei Optokoppler.

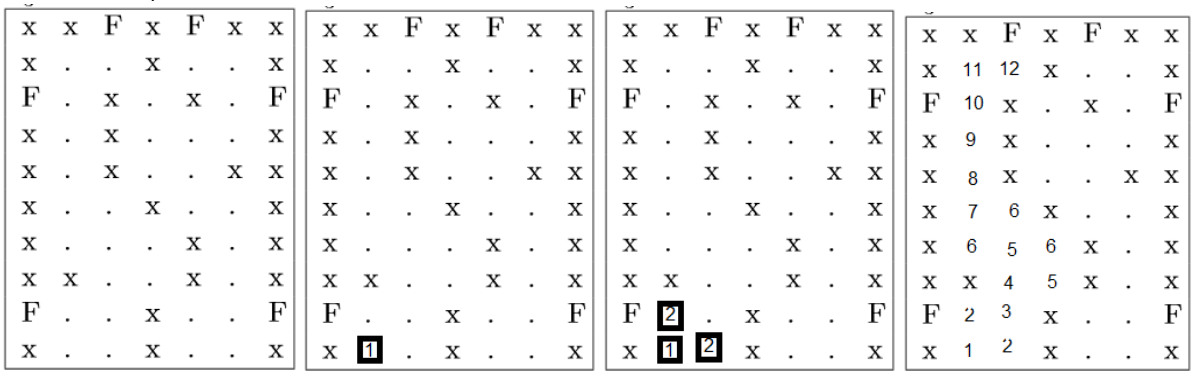
4.2. Hauptprogramm

Das Programm besteht hauptsächlich aus folgenden Funktionen die nacheinander aufgerufen werden an denen im Folgenden die Wegfindung gezeigt werden soll:

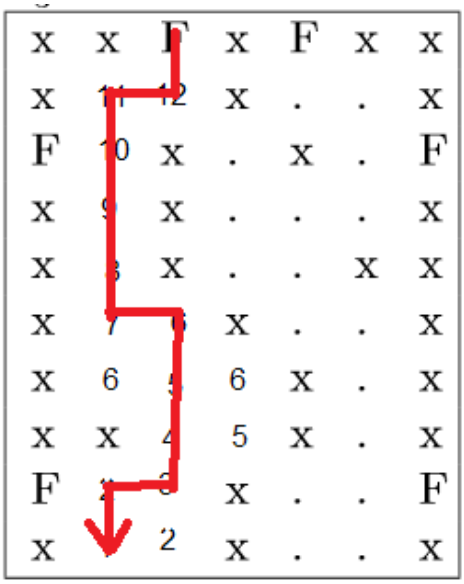
initialize()
flood()
findWays()
calculate()
drive()

void initialize()
Die initialize() Funktion stellt die LEDs an und setzt den Greifer zurück. Außerdem wird der Startpunkt über einen DIP-Schalter ausgewählt und im Kartenarray mit dem Wert 1 initialisiert. Es wurde 1 gewählt, da die Darstellung der 0 schwierig war und zu Problemen führte.

void flood()
Die Funktion flood() flutet die Karte mit einer Kostenmatrix vom Startpunkt aus. Das heißt es wird geprüft ob in der Karte ein immer steigender Wert ist. Von diesem aus werden dessen unbeschriebene Nachbarn mit einen um 1 höheren Wert erweitert, sofern diese nicht blockiert sind durch „F“ oder „x“. Es wird also erst die 1 erweitert, dann die 2, dann die 3 usw.



void findWays()
Als nächstes wird durch findWays() die gesamte, nun geflutete Karte, durchlaufen und alle gefundenen „F“ werden auf Erreichbarkeit überprüft, indem man sich die 4 Nachbarn anschaut. Ist dieser Fahrgast erreichbar, wird ausgehend von diesem Fahrgast der Weg zurück zum Startpunkt errechnet indem man immer den nächsten, kleineren Wert in den Nachbarn sucht. Alle besuchten Knoten der Karte werden durch Richtungsangaben gespeichert, genauso wie die Länge des jeweiligen Fahrauftrags. Richtungsangaben werden durch Buchstaben ‚n‘ – Norden, e‘ – Osten, ‚s‘- Süden, ‚w‘ – Westen. Dabei ist aufgrund der Aufgabe die momentane Fahrtenanzahl auf 3 begrenzt.



void calculate()
Nachdem der Rückweg von den Fahrgästen bekannt ist, muss noch der Hinweg errechnet werden. Dies geschieht in der Funktion calculate(), die das Rückfahrtenarray einmal durchläuft und jede gespeicherte Richtung durch ihre Gegenrichtung ersetzt. Danach wird ein „F“ sowie den Rückweg gefolgt von einem „a“ zusammenkopiert und dann der betreffenden Rückweg im Fahrtenarray überschreibt. Die Zeichen „F“ und „a“ geben den Befehl zum Aufnehmen (F) sowie das Abliefern (a) des Fahrgastes an.

void drive()
Die Funktion drive(); setzt die Richtungsangaben der gespeicherten Fahraufträge in Roboterbewegungen um.

Unterfunktionen

void go(short i)

Die go() Funktion stellt verschiedene Motoreinstellungen bereit, abhängig des übergebenen Wertes:

0 - vorwärts

1 - links

2 - rechts

3 - rückwärts

4 - stopp

5. Verbesserungen

5.1. Hardwareverbesserungen

Durch das Einbeziehen der hinteren Optokoppler könnte man sicherlich die Sicherheit beim Kurvendrehen und Linienfolgen verbessern. Ebenfalls ist die Erkennung des Drehens mit nur einem Sensor vergleichsweise fehleranfällig.

Außerdem hat sich während des Projektabschlusses gezeigt, dass robustes Fahren durchaus schwer zu erreichen und daher sehr gute Platzierungen, trotz geringerer Geschwindigkeit, erzielen kann. Deshalb ist eine geringere Übersetzung aufgrund der mangelnden Akkuleistung zu bevorzugen.

Ebenfalls erwies sich die ebenfalls hinten angebrachte Stütze als Quelle einer erhöhten Ungenauigkeit beim Rückwärtsfahren, sodass dies auf ein Minimum reduziert wurde.

5.2. Softwareverbesserung

Das Programm ließe sich deutlich hinsichtlich Übersichtlichkeit, Wartbarkeit, Robustheit und Erweiterbarkeit verbessern, indem man z.B. weitere Code-Teile in Funktionen auslagert. Außerdem sollte man weniger globale Elemente verwenden und Funktionen eher mit Übergabewerten/Rückgabewerten ausstatten. Ebenfalls hätte man die Karten über eine Headerdatei einbinden können müssen.

6. Anhang

Kompilierbarer Quellcode genutzt zur finale Präsentation des Projektes.

```
#include <stdbool.h> //for bools
#include <stdlib.h>
//Standard-Include-Files
#include <stdio.h>
#include <regc515c.h>
#include <string.h> //for string compare
//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>


unsigned short threshold = 50; // Schwellwert für weiß

short velocity = 10; //Geschwindigkeit bei Fahrt ohne Probleme
short brake = 8; //Wert um den gebremst wird beim Linienfolgen
char map[] =
"xFxxxFxx.x.x.xF..x..Fx..x..xx..x..xx.x.x.xF..x..Fx..x..xx..x..xx.x.x.x";//Karte
char way[3][70];//Array das später die Fahraufträge hält
char length[3];//Länge der einzelnen Fahraufträge
char next;
int neighbours[4];//Nachbarn des überprüften Punktes

//Die Funktion ist solange in der Schleife bis das Lichtsignal erscheint
void start()
{
	while(digital_in(1)==1)
	{
	}
}

//stellt verschiedene Motoreinstellungen abhängig des übergebenen short Parameters
//0 - vorwärts
//1 - links
//2 - rechts
//3 - rückwärts
//4 - stop
void go(unsigned short mode)
{
	switch(mode){
	case 0://forward
		motor_richtung(1, 1);
		motor_richtung(3, 0);

		motor_pwm(1,velocity);
		motor_pwm(3,velocity);
		break;

	case 1://left
		motor_richtung(1, 0);
		motor_richtung(3, 0);

		motor_pwm(1,velocity);
		motor_pwm(3,velocity);
		break;

	case 2://right
		motor_richtung(1, 1);
```

```
                motor_richtung(3, 1);

                motor_pwm(1,velocity);
                motor_pwm(3,velocity);
                break;

        case 3://backward
                motor_richtung(1, 0);
                motor_richtung(3, 1);

                motor_pwm(1,velocity);
                motor_pwm(3,velocity);
                break;

        case 4://stopp
                motor_richtung(1, 0);
                motor_richtung(3, 1);

                motor_pwm(1,0);
                motor_pwm(3,0);
                break;

        }
}


//stellt die Motoren auf Rechtsdrehen ein, wartet 300 Millisekunden und wartet dann
bis der rechte Sensor nichtmehr weiß anzeigt
void turn_right()
{
        go(2);//rechts

        clear_time();
        //wartet 300 Millisekunden oder solange der rechte Optokoppler noch schwarz ist
        while(analog(0) > threshold ||  akt_time() < 300)
        {
        }

        //wartet dann bis der rechte Sensor nichtmehr weiß ist
        while(analog(0) < threshold)
        {
        }
}

//stellt die Motoren auf Linksdrehen ein, wartet 300 Millisekunden und wartet dann bis
der linke Sensor nichtmehr weiß anzeigt
void turn_left()
{
        go(1);//links

        clear_time();
        //wartet 300 Millisekunden oder solange der linke Optokoppler noch schwarz ist
        while(analog(2) > threshold ||  akt_time() < 300)
        {
        }

        //wartet dann bis der linke Sensor nichtmehr weiß ist
        while(analog(2) < threshold)
        {
        }
}

//stellt den Motor auf vorwärtsfahren ein,
```

```
//fährt solange vorwärts bis beide mittleren Optokoppler nichtmehr schwarz ist
//oder 0,5 sec vergangen sind
void forward()
{
	//motor_richtung(unsigned char motor,unsigned char richtung)
	//motor_pwm(unsigned char motor,unsigned char power)
	//lcd_cls();
	go(0); //vorwärts

	front_left_opto = analog(2);
	front_right_opto = analog(0);

	middle_left_opto = analog(6);
	middle_right_opto = analog(4);
	//middle_opto = analog(4);

	clear_time();

	//fahr von der Kreuzung runter (beide mitte nichtmehr schwarz) oder 0,5 sec
vorbei
	while((middle_left_opto > threshold && middle_right_opto > threshold ||
akt_time() < 500))
	{
		front_left_opto = analog(2);
		front_right_opto = analog(0);
		middle_left_opto = analog(6);
		middle_right_opto = analog(4);

		//vorne links und vorne rechts wei�: vollgas
		if (front_left_opto < threshold && front_right_opto < threshold)
		{
			motor_pwm(1,velocity);
			motor_pwm(3,velocity);
		}

		//vorne links auf schwarz: bremse links
		if (front_left_opto > threshold && front_right_opto < threshold)
		{
			motor_pwm(1,velocity-brake);
			motor_pwm(3,velocity);
		}

		//vorne rechts auf schwarz: bremse rechts
		if (front_left_opto < threshold && front_right_opto > threshold)
		{
			motor_pwm(3,velocity-brake);
			motor_pwm(1,velocity);
		}
	}


	//Solange links oder rechts auf wei� sind
	while((middle_left_opto < threshold || middle_right_opto < threshold ))
	{
		front_left_opto = analog(2);
		front_right_opto = analog(0);
		middle_left_opto = analog(6);
		middle_right_opto = analog(4);

		//vorne links und vorne rechts wei�: vollgas
		if (front_left_opto < threshold && front_right_opto < threshold)
		{
			motor_pwm(1,velocity);
```

```
                        motor_pwm(3,velocity);
                }

                //vorne links auf schwarz: bremse links
                if (front_left_opto > threshold && front_right_opto < threshold)
                {
                        motor_pwm(1,velocity-brake);
                        motor_pwm(3,velocity);
                }

                //vorne rechts auf schwarz: bremse rechts
                if (front_left_opto < threshold && front_right_opto > threshold)
                {
                        motor_pwm(3,velocity-brake);
                        motor_pwm(1,velocity);
                }
        }
}

//füllt das Kartenarray mit der Kostenmatrix solange ein jeweils höherer Kostenwert
gefunden wird
void flood()
{
        int j = 1;
        int i = 0;
        //solange ein Kostenwert gefunden wird
        while(exist==1)
        {
                exist = 0;
                //flute das Labyrinth
                for(i = 0; i < 70; i++)
                {
                        if((map[i]) == j)
                        {
                                exist = 1;
                                //north
                                if((i-7>0)&&(map[i-7]== '.'))
                                {
                                        map[i-7] = j+1;
                                }

                                //west
                                if((map[i-1]== '.'))
                                {
                                        map[i-1] = j+1;
                                }

                                //south
                                if((i+7<70)&&(map[i+7]== '.'))
                                {
                                        map[i+7] = j+1;
                                }

                                //east
                                if((map[i+1]== '.'))
                                {
                                        map[i+1] = j+1;
                                }
                        }
                }
                j++;
        }
}
```

```
//Prüft ob der übergebene Punkt einen Nachbarn mit Kosten hat.
//Das bedeutet dieser Punkt ist erreichtbar
bool isReachable(int place)
{
	bool reachable = false;
	if((place-7>0) && map[place-7] != '.' && map[place-7] != 'F' && map[place-7] !=
'x') {reachable = true;}
	else if((place-1>0) && map[place-1] != '.' && map[place-1] != 'F' && map[place-
1] != 'x') {reachable = true;}
	else if((place+7<70) && map[place+7] != '.' && map[place+7] != 'F' &&
map[place+7] != 'x') {reachable = true;}
	else if((place+1<70) && map[place+1] != '.' && map[place+1] != 'F' &&
map[place+1] != 'x') {reachable = true;}

	return reachable;
}


//Findet den nächstkleineren Wert und gibt diesen als Richtungswert zurück.
//n - -7
//e - +1
//s - +7
//w - -1
int findNext(int place)
{
	int i = 0;
	for (i = 0;i < 4;i++)
	{
		neighbours[i] = 70;
	}

	if((place-7>0) && map[place-7] != '.'&& map[place-7] != 'F'&& map[place-7] !=
'x') {neighbours[0] = map[place-7];}
	if(/*div((place-1),7).quot == div((place),7).quot && */map[place-1] != '.'&&
map[place-1] != 'F'&& map[place-1] != 'x') {neighbours[1] = map[place-1];}
	if((place+7<=69) && map[place+7] != '.'&& map[place+7] != 'F'&& map[place+7] !=
'x') {neighbours[2] = map[place+7];}
	if(/*div((place+1),7).quot == div((place),7).quot && */map[place+1] != '.'&&
map[place+1] != 'F'&& map[place+1] != 'x') {neighbours[3] = map[place+1];}

	if (neighbours[0]<=neighbours[1] && neighbours[0]<=neighbours[2] &&
neighbours[0]<=neighbours[3]){return -7;}
	else if (neighbours[1]<=neighbours[0] && neighbours[1]<=neighbours[2] &&
neighbours[1]<=neighbours[3]){return -1;}
	else if (neighbours[2]<=neighbours[1] && neighbours[2]<=neighbours[0] &&
neighbours[2]<=neighbours[3]){return +7;}
	else if (neighbours[3]<=neighbours[1] && neighbours[3]<=neighbours[2] &&
neighbours[3]<=neighbours[0]){return +1;}

	return 0;
}

//findet die Rückwege aller erreichbaren 'F' in der momentanen Map.
void findWays()
{
	int i = 0;
	int j = 0;
	int t = 0;
	int t2 = 0;
	int direction = 0; // Richtung des Roboters
	char next;
	bool a = false;
```

```
	//erzeuge alle Rückwege
	for (i = 0; i < 70; i++)
	{
		if(map[i] == 'F')
		{
			map[i] = 'x';
			j = i;
			t2 = 0;

			a = isReachable(i);//wenn der nächste Fahrgast vom Startpunkt aus
erreichbar ist
			if( a == true)
			{
				while(map[j] != 1)
				{
					direction = findNext(j);
					j+=direction;

					switch(direction)
					{
					case -7: next = 'n';
						break;
					case -1: next = 'w';
						break;
					case 7: next = 's';
						break;
					case 1: next = 'e';
						break;
					default: next = '\0';
						break;
					}

					way[t][t2] = next;
					length[t]= length[t]+1;
					t2++;
				}
				t++;
			}
		}
	}
}

//Erstellung der gesamten Wege aus den im Wegearray gespeicherten Rückwege
void calculate()
{
	char changer[100];
	char len;
	int i = 0;
	int j = 0;

	//alle Fahrtwege einmal
	for (i = 0; i < 3; i++)
	{
		len = length[i]; //Länge des derzeitigen Weges
		//alle Zeichen einmal
		for (j = 0; j < len; j++)
		{
			//Hinweg wird errechnet
			switch(way[i][j])
			{
			case 'w': changer[len-j-1] = 'e';
				break;
```

```
                        case 'n': changer[len-j-1] = 's';
                                break;
                        case 's': changer[len-j-1] = 'n';
                                break;
                        case 'e': changer[len-j-1] = 'w';
                                break;
                        default :
                                break;
                        }
                }

                changer[len] = 'F'; // Markierung zum holen hinzufügen

                //Rückweg wird hinzugefügt
                for (j = 1; j < len; j++)
                {
                        switch(way[i][j])
                        {
                        case 'w': changer[len+j] = 'w';
                                break;
                        case 'n': changer[len+j] = 'n';
                                break;
                        case 's': changer[len+j] = 's';
                                break;
                        case 'e': changer[len+j] = 'e';
                                break;
                        default :
                                break;
                        }
                }

                changer[len*2] = 'a';//Markierung zum ablegen hinzufügen

                //der erstellte Weg wird zurück in das Fahrtenarray kopiert,
                //wo bisher nur die Rückwege standen
                for (j = 0; j < len * 2 + 1; j++)
                {
                        switch(changer[j])
                        {
                        case 'w': way[i][j] = 'w';
                                break;
                        case 'n': way[i][j] = 'n';
                                break;
                        case 's': way[i][j] = 's';
                                break;
                        case 'e': way[i][j] = 'e';
                                break;
                        case 'a': way[i][j] = 'a';
                                break;
                        case 'F': way[i][j] = 'F';
                                break;


                        default :
                                break;
                        }
                }

        }
}


void grab()
```

```
{
	go(0);//vorwärts
	while(digital_in(0) == 0 )
	{
		front_left_opto = analog(2);
		front_right_opto = analog(0);

		//vorne links und vorne rechts weiß: vollgas
		if (front_left_opto < threshold && front_right_opto < threshold)
		{
			motor_pwm(1,velocity);
			motor_pwm(3,velocity);
		}

		//vorne links auf schwarz: links bremsen
		if (front_left_opto > threshold && front_right_opto < threshold)
		{
			motor_pwm(1,velocity-brake);
			motor_pwm(3,velocity);
		}

		//vorne rechts auf schwarz: rechts bremsen
		if (front_left_opto < threshold && front_right_opto > threshold)
		{
			motor_pwm(3,velocity-brake);
			motor_pwm(1,velocity);
		}
	servo_arc(0, 70); // Greifer absenken
	go(4);//stopp
	sleep(500);
	go(3);//rückwärts

	middle_left_opto = analog(6);
	middle_right_opto = analog(4);

	clear_time();

	//zurück zur letzten Kreuzung
	while(middle_left_opto < threshold || middle_right_opto < threshold ||
akt_time() < 500)
	{
		back_left_opto = analog(8);
		back_right_opto = analog(10);
		middle_left_opto = analog(6);
		middle_right_opto = analog(4);

		//vorne links und vorne rechts weiß: vollgas
		if (back_left_opto < threshold && back_right_opto < threshold)
		{
			motor_pwm(1,velocity);
			motor_pwm(3,velocity);
		}

		//vorne links auf schwarz: links bremsen
		if (back_left_opto > threshold && back_right_opto < threshold)
		{
			motor_pwm(1,velocity-brake);
			motor_pwm(3,velocity);
		}

		//vorne rechts auf schwarz: rechts bremsen
		if (back_left_opto < threshold && back_right_opto > threshold)
		{
```

```
                        motor_pwm(3,velocity-brake);
                        motor_pwm(1,velocity);
                }
        }

}

//liefert einen Passagier ab
void deliver()
{
        go(0);//vorwärts
        clear_time();

        while(akt_time() < 100)
        {
                front_left_opto = analog(2);
                front_right_opto = analog(0);

                //vorne links und vorne rechts weiß: vollgas
                if (front_left_opto < threshold && front_right_opto < threshold)
                {
                        motor_pwm(1,velocity);
                        motor_pwm(3,velocity);
                }

                //vorne links auf schwarz: links bremsen
                if (front_left_opto > threshold && front_right_opto < threshold)
                {
                        motor_pwm(1,velocity-brake);
                        motor_pwm(3,velocity);
                }

                //vorne rechts auf schwarz: rechts bremsen
                if (front_left_opto < threshold && front_right_opto > threshold)
                {
                        motor_pwm(3,velocity-brake);
                        motor_pwm(1,velocity);
                }
        }
        servo_arc(0, 30); //Greifer anheben
        sleep(500);
        go(3);//rückwärts

        middle_left_opto = analog(6);
        middle_right_opto = analog(4);

        clear_time();

        //bis zur vorherigen Kreuzung
        while(middle_left_opto < threshold || middle_right_opto < threshold ||
akt_time() < 500)
        {
                back_left_opto = analog(8);
                back_right_opto = analog(10);
                middle_left_opto = analog(6);
                middle_right_opto = analog(4);

                //hinten links und hinten rechts wei:
                if (back_left_opto < threshold && back_right_opto < threshold)
                {
                        motor_pwm(1,velocity);
                        motor_pwm(3,velocity);
                }
```

```
		//hinten links auf schwarz: links bremsen
		if (back_left_opto > threshold && back_right_opto < threshold)
		{
			motor_pwm(1,velocity-brake);
			motor_pwm(3,velocity);
		}

		//hinten rechts auf schwarz: rechts bremsen
		if (back_left_opto < threshold && back_right_opto > threshold)
		{
			motor_pwm(3,velocity-brake);
			motor_pwm(1,velocity);
		}
	}
}

//Es wird geprüft ob der nächste Knoten der Karte ein Fahrgast ist
// wenn ja greifen, wenn nicht bis zur nächsten Kreuzung fahren
void checkPassanger()
{
	if (way[i][j+1] == 'F')
	{
		grab();
	}else
	{
		forward();
	}
}


//übersetzt alle Richtungsanweisungen in Bewegungsbefehle
void drive()
{
	int direction = 0;
	int i = 0;
	int j = 0;

	//fängt mit dem letzten gefunden Weg an
	for(i = 2; i >= 0;i--)
	{
		//geht durch jedes Element des gesamten Weges 2*Länge wegen Hin und
Rückweg
		for(j = 0; j < length[i]*2+1; j++)
		{
			//verzweigt jeweils nach der nächsten zu fahrenden Richtung
			//und der Richtung die der Roboter momentan hat
			switch(way[i][j]){
				//South
				case's':
					switch(direction){
					case 0:
						turn_right();
						turn_right();
						checkPassanger();
						direction = 2;
						break;

					case 1:
						turn_left();
						checkPassanger();
						direction = 2;
						break;
```

```
        case 2:
                checkPassanger();
                direction = 2;
                break;

        case 3:
                checkPassanger();
                direction = 2;
                break;
        default:
                break;
        }
        break;

//North
case'n':
        switch(direction){
        case 0:
                checkPassanger();
                direction = 0;
                break;

        case 1:
                turn_right();
                checkPassanger();
                direction = 0;
                break;

        case 2:
                turn_left();
                turn_left();
                checkPassanger();
                direction = 0;
                break;

        case 3:
                turn_left();
                checkPassanger();
                direction = 0;
                break;

        default:
                break;
        }
        break;

//West
case'w':
        switch(direction){
        case 0:
                turn_left();
                checkPassanger();
                direction = 1;
                break;

        case 1:
                checkPassanger();
                direction = 1;
                break;

        case 2:
                turn_right();
```

```
                checkPassanger();
                direction = 1;
                break;

        case 3:
                turn_right();
                turn_right();
                checkPassanger();
                direction = 1;
                break;

        default:
                break;
        }
        break;

//East
case'e':
        switch(direction){
        case 0:
                turn_right();
                checkPassanger();
                direction = 3;
                break;

        case 1:
                turn_right();
                turn_right();
                checkPassanger();
                direction = 3;
                break;

        case 2:
                turn_left();
                checkPassanger();
                direction = 3;
                break;

        case 3:
                checkPassanger();
                direction = 3;
                break;

        default:
                break;
        }
        break;

case'a':
        switch(direction){
        case 0:
                turn_right();
                turn_right();
                deliver();
                turn_right();
                turn_right();
                direction = 0;
                break;

        case 1:
                turn_right();
                deliver();
                turn_left();
```

```
                                        direction = 0;
                                        break;

                                case 2:
                                        deliver();
                                        turn_right();
                                        turn_right();
                                        direction = 0;
                                        break;

                                case 3:
                                        turn_left();
                                        deliver();
                                        direction = 0;
                                        break;

                                default:
                                        break;
                                }
                                break;

                        default:
                                break;
                        }
                }
        }
}

//Stellt den Startpunkt im Kartenarray abhängig von der Dipschalterposition ein
void setStart()
{
        switch(dip_pin(2))
        {
        case 0: map[64] = 1;
                lcd_cls();
                lcd_puts("a");
                break;

        case 1: map[68] = 1;
                lcd_cls();
                lcd_puts("b");
                break;

        default:
                break;
        }
}

//stellt den Greifer hoch und die LEDs an
void initialize()
{
        setStart();
        servo_arc(0, 30);
        led(3,1);
        led(2,1);
        led(1,1);
        led(0,1);
}


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

```
        initialize();
        start();
        flood();
        findWays();
        calculate();
        drive();
        go(4);

        while(1)
        {

        }
}
```