# AMS Projekt WS 2014/2015

# Dokumentation

---

Brandenburg a.d. Havel, 23. Januar 2015

vorgelegt von

Vanessa Vogel (20120016),
Maximilian Regner (20120025)

Betreuender Professor: I. Boersch, J.Heinsohn

Inhaltsverzeichnis


1. Vorwort
2. Aufgabe
3. Lösungsweg
    a. Konstruktion
    b. Software
    c. Hardware
4. Quellcode
5. Anhang
    a. Literaturverzeichnis
    b. Abbildungsverzeichnis

## 1. Vorwort

Das Konzept des Personal Rapid Transit (PRT) ist ein Personentransportsystem, also eine Flotte von führerlosen, spurgeführten Fahrzeugen, die ohne Fahrplan, individuell auf Bestellung, Fahrgäste abholt und sie zum gewünschten Ziel bringt. Dieses System wird bereits an vielen Orten der Welt eingesetzt, beispielsweise in London . Das bisher größte System befindet sich in Masdar City, eine Stadt der Vereinigten Arabischen Emirate.[1]

Das Ziel des Projekts ist, das Konzept des PRT im KI-Labor der Fachhochschule Brandenburg umzusetzen , demnach werden kleine Roboter gebaut und programmiert, die Fahraufträge übernehmen und Fahrgäste autonom transportieren.

Abbildung 1: PRT Kabine aus London [2]

[1] http://de.wikipedia.org/wiki/Personal_Rapid_Transit#Aktuelle_Projekte ; Zugriff: 15.01.2015
[2] http://de.wikipedia.org/wiki/Personal_Rapid_Transit#mediaviewer/File:ULTra_001.jpg ; Zugriff: 15.01.2015

## 2. Aufgabe

Der Roboter erhält den Auftrag, eine oder mehrere Personen (Kugeln) abzuholen und zum Ziel zu bringen. Das Streckennetz ist ein einfaches Gitter, in dem es allerdings zu Störungen und damit zu unbefahrbaren Kreuzungen kommen kann. Die Karte der jeweiligen Strecke und die Position der Fahrgäste stehen zur Verfügung. Es soll mit den zur Verfügung stehenden Teilen ein Roboter gebaut werden, der diese Aufgabe meistert .

Die Karteninformationen werden in Form eines Fahrauftrages vor dem Wettbewerb zur Verfügung gestellt. Am letzten Projekttag findet der Wettbewerb, bei welchem der Sieger des Projektes gekürt wird, statt .

# 3. Lösungsweg

## a. Konstruktion

Der erste Schritt des Projekts war, einen Roboter aus Lego zu bauen. Dazu standen verschiedene Bauteile zur Verfügung: LEGO-Teile, ein Aksen-Board, Akku, Servomotor, Elektromotoren, Infrarotsensor, Optokoppler, Infrarot-LED und ein Photosensor.

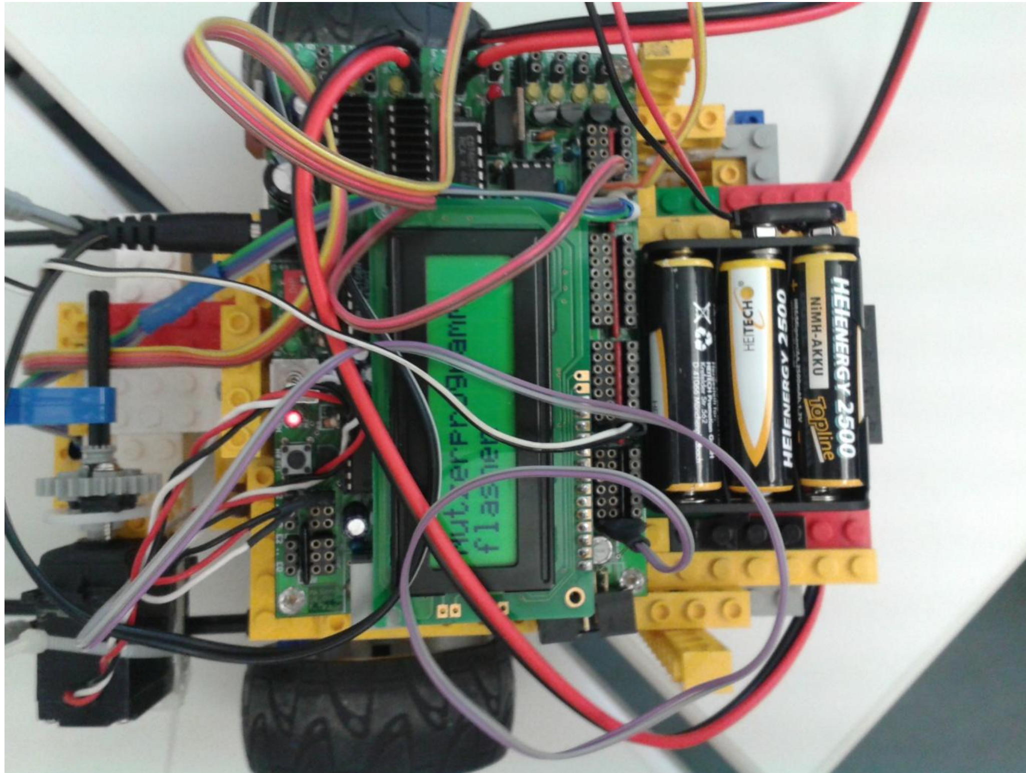


Abbildung 2: Bauteile

Einige der Bauteile sind in Abbildung 2 sichtbar. Als erstes haben wir begonnen, einen geeigneten Rahmen für unseren Roboter zu bauen, um ihn als Grundgerüst für die weiteren Schritte zu benutzen. In diesen Rahmen haben wir, im nächsten Schritt, den Antrieb gebaut. Dies erwies sich als einer der schwierigsten Schritte. Wir haben uns schnell für den differentiellen Antrieb entschieden, da er am leichtesten umzusetzen schien und jedes Rad einzeln angesteuert werden kann. Das Problem war die Konstruktion des Antriebes. An diesem Punkt hatten wir einige Schwierigkeiten, denn wir hatten Probleme diesen Antrieb mit LEGO Teilen zu bauen, auch wegen der Stabilität des Rahmens. Dieser musste zum späteren Zeitpunkt erneut zusammen gebaut werden, aber deutlich stabiler, als vorher. Als die Hürde des Antriebs überwunden war, war das Resultat eine Untersetzung von $i = 81/1$. Sie kann den Ansprüchen entsprechend eine hohe Drehzahl der Räder und eine ausreichende Kraftübertragung liefern.

Da nicht sehr viele Räder zur Verfügung standen, war die Auswahl der passenden Räder nicht schwer. Wir entschieden uns für die breiteren Räder, um die Robustheit des Roboters zu gewährleisten.

Im nächsten Schritt war die Auflagefläche  für das Aksenboard und der Akku dran. Dazu erhöhten wir den Rahmen mit Lego-Teilen, um das Getriebe und die Räder nicht zu beeinträchtigen und legten große Platten über den Rahmen, die dann als Auflagefläche dienten. Damit die beiden Teile nicht vom Rahmen rutschen  können, wurde eine Grenze herum errichtet. Es folgten die Sensoren zum Linienfolgen.

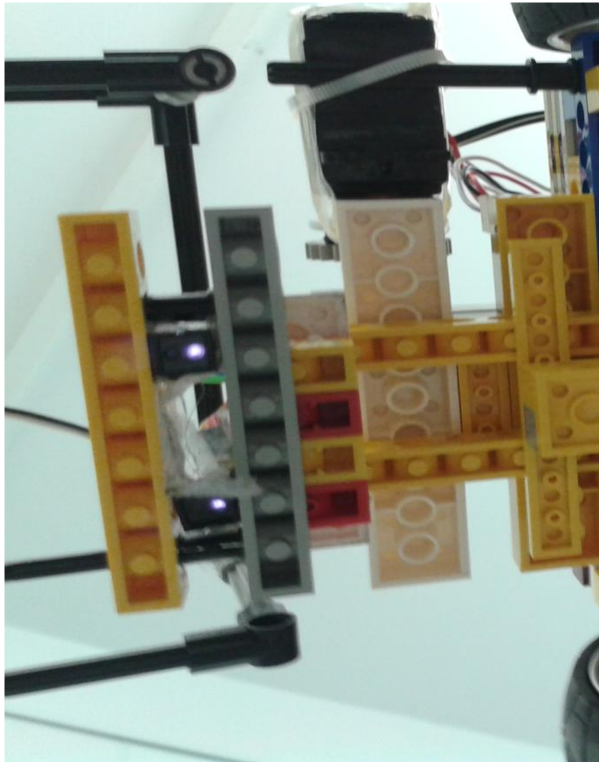

Abbildung 3: Sensoren

Wie in Abbildung 3 sichtbar, wurde ein Konstrukt aus Lego Bausteinen nach vornhin gebaut, um die Sensoren möglichst weit vorn zu platzieren, um ein sicheres Linienfolgen zu gewährleisten. Die Sensoren stehen etwas mehr als eine Linienbreite auseinander, dass  die Linien genau  und keine großen Schwankungen gefahren werden. Sie sind zwischen 2 Legosteinen befestigt.

An der hinteren Seite des Rahmens ist ein Führungselement angebracht, damit der Roboter nicht nach hinten kippen kann. Dieses behindert nicht beim Fahren und lässt sich einfach von den Rädern führen.

Im letzten Schritt wurde der Greifer angebaut. Er ist aus LEGO-Stangen zusammen gesetzt und bildet einen Kasten, in dem der Ball transportiert  wird.

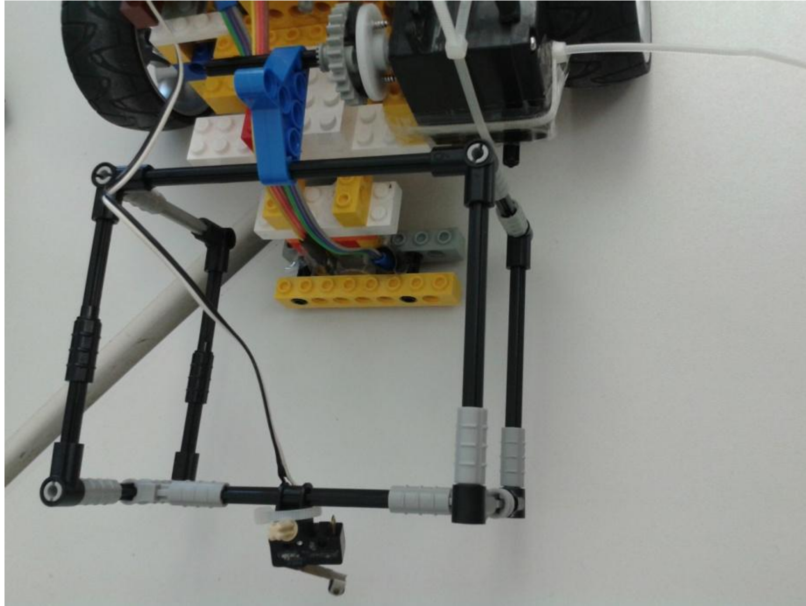

Abbildung 4: Greifer

Wie in Abbildung 3 ersichtlich, ist der Greifer direkt über die Sensoren zum Linienfolgen befestigt.

Alle Schritte vereint, ergibt das unseren fertigen Roboter "Driver".

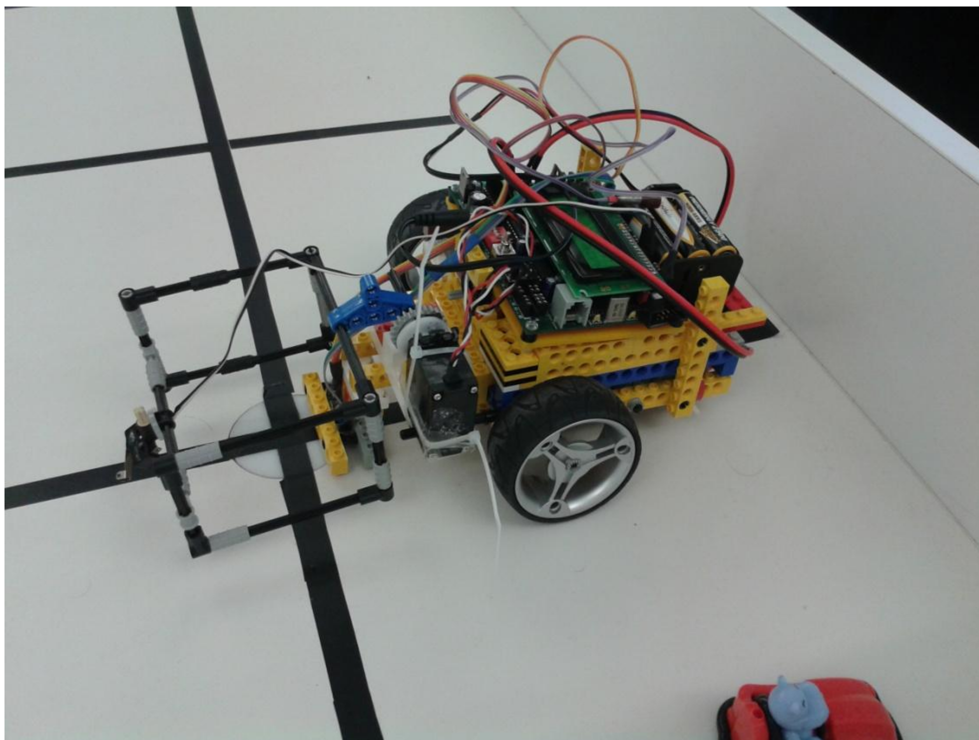

Abbildung 5: fertiger Roboter

Das Ergebnis des Projekts zeigt sich in Abbildung 5. Es ist ein einfach gehaltender, minimalistisch konzipierter Roboter, der nur die wichtigsten Teile besitzt, die er benötigt. Trotz seiner Einfachheit ist er sehr leistungsstark und schnell. Der Schwerpunkt befindet sich im hinteren Teil, auf dem Führungselement.

## b. Software

Die Software dient einerseits zur Steuerung der Hardware und Abfrage der Sensorwerte und anderseits zur Planung des Ablaufplanes.

Die Karte, die der Roboter benutzen soll, besteht aus 70 Feldern. Das Feld hat 7 Felder in der Länge und 10 Felder in der Breite. Die nördlichste Reihe entspricht hierbei den ersten 7 Eintragen im Array (Index von 0 bis 6). Die Kartenwerte sind jeweils von West nach Ost eingetragen. Die zweite Reihe enthält dementsprechend die Einträge 7 – 13. Dies wird bis zum 69. Eintrag nach unten fortgeführt. Dementsprechend muss der Roboter, um einen Schritt nach Norden zu machen, die aktuelle Position um 7 verringern rechnen, um einen Schritt nach Süden zu fahren, die aktuelle Position um 7 erhöhen, für einen Schritt nach Westen, die aktuelle Position um 1 verringern und für einen Schritt nach Osten die aktuelle Position um 1 erhöhen.

Der Ablaufplan wird mit Hilfe der Breitensuche gefunden, d. h. es wird jeweils von einem Startpunkt aus in nördliche, südliche, westliche und östliche Richtung nach freien Knoten gesucht, diese Werte werden in ein „erreichbareKnoten“-Array, welches alle erreichbare Knoten beinhaltet, eingetragen und nach Betrachtung des ersten Elements wird das nächste Element im Array betrachtet. Dies wird solange durchgeführt, bis das aktuelle Ziel erreicht ist. Das Ziel kann zudem durch ein Intervall definiert werden. Bei der Suche nach einem Fahrgast beschränkt sich das Intervall auf genau die eine Stelle, wo sich der Fahrgast befindet, bei der Zielfindung werden die erreichbaren Elemente der südlichste Reihe als Intervall festgelegt (64 bis 68).Desweiteren wird für jedes eingetragene Element der Vorgänger, also das Element durch welchen das neu eingetragene Element gefunden wurde, in ein separates Array eingetragen. Diese zwei Listen werden benötigt, um später den Weg vom Zielelement bis zum Startelement nachzuvollziehen. Jedes Element kann zudem nur ein Mal in das Array mit den zu betrachtenden Elementen eingetragen werden, damit Zyklen vermieden werden und da es ausreichend ist, von einem Knoten aus einen anderen zu erreichen.

Zuerst werden alle Fahrgäste von einem bestimmten Startelement aus ermittelt. Dabei wird das soeben beschriebene Verfahren (Breitensuche) benutzt. Die gefundenen Fahrgäste werden in einem Fahrgäste-Array gespeichert. Dieses Array legt zugleich die Reihenfolge fest, in welches die Fahrgäste abgeholt werden.
Nachdem die Fahrgäste ermittelt wurden, wird zunächst der schnellste Weg vom Startelement zum Fahrgast, ebenfalls durch den soeben beschriebenen Ablauf, ermittelt. Wenn das Zielelement durch die Breitensuche gefunden wurde, wird die Rückverfolgung des Wegs, also welche Elemente in welcher Reihenfolge angefahren werden sollen, in einem Teilstrecke-Array gespeichert. Danach wird die ermittelte Teilstrecke in umgedrehter Reihenfolge, aufgrund der Rückwärtsverfolgung des Weges im vorherigen Schritt, in ein Gesamtstrecke-Array eingetragen, in dem dann später der vollständige Plan eingetragen wird.
Da ein anderer Weg verwendet wird, um den Fahrgast zum Ziel zu bringen, wird nun vom gefundenen Fahrgast aus, der Schnellste Weg zur südlichsten Reihe des Feldes gesucht, was das neue Zielintervall darstellt. Um die Teilstrecke zu erstellen, wird derselbe Ablauf wie eben verwendet, um die Teilstrecke zu finden. Die Teilstrecke wird dann in umgedrehter Reihenfolge am Ende des Gesamtstrecke-Array eingetragen.
Die letzten beiden Schritte, welche die Suche des Weges zu einem Fahrgast und das Suchen des

Weges von der Haltestelle zum Ziel beinhalten, werden nun solange wiederholt bis alle erreichbaren Fährgäste ihr Ziel erreicht haben.

Da bisher nur die Elemente ermittelt wurden, die der Roboter nacheinander anfahren soll, und diese noch nicht vom Roboter ausführbar sind, werden diese nun in Aktionen noch codiert.

Dafür wird in einem ersten Schritt für jeden Eintrag im Gesamtstrecke-Array der Wert, welcher angibt in welcher Richtung der nächste Knoten liegt , durch die Differenz aus dem nächstfolgendem Element und dem aktuellen Element ermittelt. Die Werte 7, -7, 1 und -1 können entstehen. Diese Werte werden noch umgewandelt.

Die -7 wird in eine 0 für Norden umgewandelt, die 1 bleibt 1 für Osten, die 7 in eine 2 für Süden und die -1 in eine 3 für Westen. Das beschreibt die Sollrichtung, in welche der Roboter fahren soll.

In einem nächsten Schritt wird für jeden Eintrag im entstandenen Gesamtstrecke-Array die aktuelle Ausrichtung mit berechnet. Die Sollrichtung wird dazu verringert um die aktuelle Ausrichtung, wodurch eine Bezeichnung für eine Aktion entsteht. Die aktuelle Ausrichtung kann Werte von 0 bis 3 annehmen. Da bei dieser Rechnung negative Werte entstehen können und nur Werte von 0 bis 3 verwendet werden sollen, werden die Werte dann noch einmal um 4 erhöht und Modulo 4 gerechnet. Das sind nun Bezeichnungen für Aktionen, die der Roboter später verwenden kann, um den Plan richtig abzufahren. Die Werte 0 bis 3 haben nun eine neue Bedeutung angenommen. 0 steht für geradeaus Weiterfahren, 1 für Rechtsabbiegen, 2 für Wenden und 3 für Linksabbiegen.

Am Anfang wird durch Schalter festgelegt, ob der Roboter von Position 64 oder 68 startet. Die Ausführung des Planes startet, sobald das Licht am Start eingeschaltet wurde.
Der Roboter verwendet Sensoren um Linien zu folgen. An einer Kreuzung, wenn beide Sensoren eine schwarze Fläche erfassen, werden dann jeweils die Befehle aus dem Gesamtstrecke-Array ausgeführt. Ein Befehl wird übersprungen, wenn der Roboter gegen die Wand fährt und somit ein Schalter betätigt wird. Stattdessen wird das Einsammeln des Fahrgastes und Wenden und das Fahren bis zur nächsten Kreuzung ausgeführt.
Zum Linksabbiegen wird der linke Motor ausgeschaltet und der Roboter dreht sich solange, bis der rechts vorne angebrachte Sensor eine schwarze Fläche sieht und fährt dann zur nächsten Kreuzung.
Beim Rechtsabbiegen wird der rechte Motor ausgeschalten und der Roboter dreht sich solange, bis der Sensor links vorne wieder eine schwarze Fläche sieht und fährt dann bis zur nächsten Kreuzung.
Beim Wenden laufen die Motoren in unterschiedlicher Geschwindigkeit solange bis die schwarze Linie vom Sensor erkannt wurde und der Roboter fährt dann bis zur nächsten Kreuzung vor.
Wenn das Ziel des Fahrgastes erreicht ist, wird der Greifer nach oben geklappt und der Fahrgast somit abgesetzt.

## c. Hardware

## Antrieb

Zwei Elektromotoren, die ihre Kraft über den Differentialantrieb an die Räder weitergeben, treiben den Roboter an(Siehe Abbildung 6).

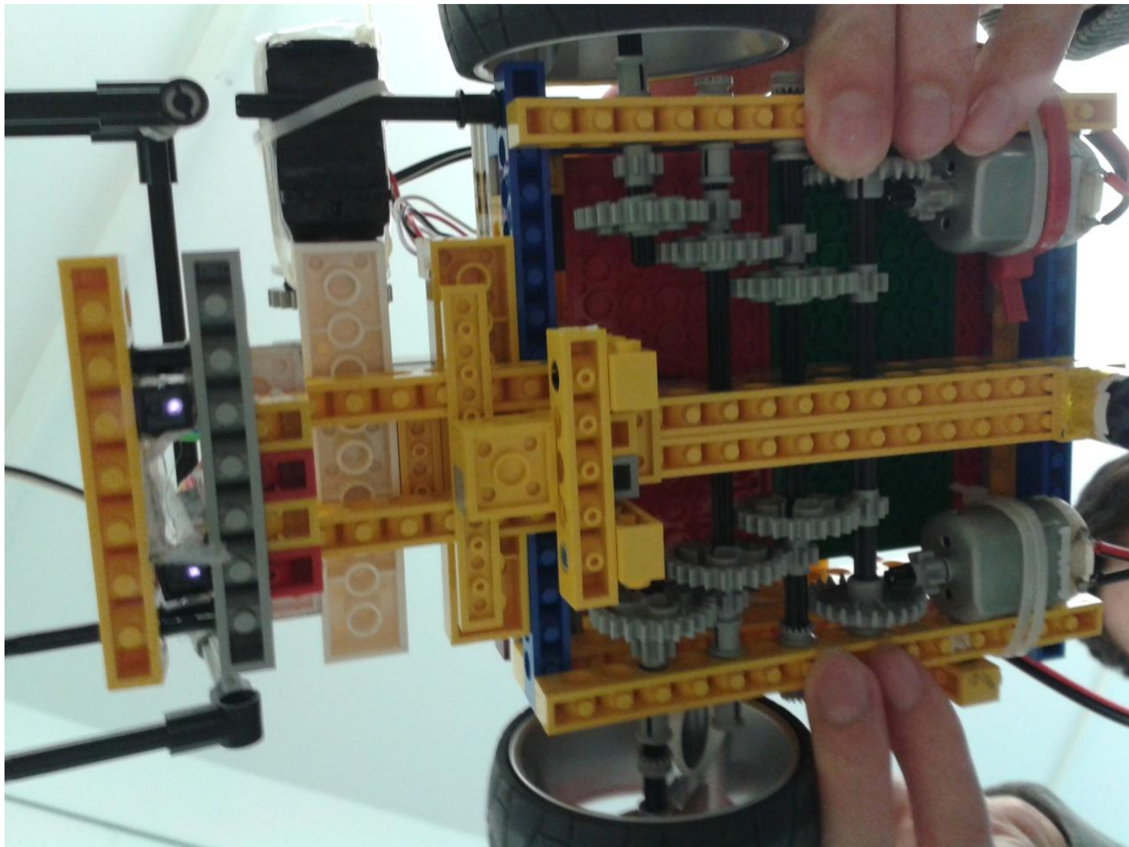

Abbildung 6: Getriebe und Motoren

Sie sind jeweils am Rahmen befestigt. Der Differentialantrieb ermöglicht ein gutes Lenkverhalten. Es können Kurven zuverlässig gefahren werden, indem ein Rad ausgestellt wird und sich das andere allein dreht.

## Sensoren :

Wie in Abbildung 3 ersichtlich, werden zum Linien fahren lediglich 2 Oktokoppler verwendet. Diese genügen um der Spur sicher und zuverlässig zu folgen.

Der Taster wurde am Greifer befestigt, mit ihn wird gestoppt, wenn der Roboter gegen eine Wand fährt. Auch zum Fahrgasttransport wird er benutzt . An der Stelle, wo sich ein Fahrgast befindet , fährt der Roboter zuerst gegen die Wand, dass er die richtige Position findet und anhält. Danach fährt er einige Zeit rückwärts, um sich von der Wand zu entfernen und den Greifer runter zu fahren . Wenn dann der Fahrgast am Ziel ist, wird der Greifer wieder nach oben gefahren und der Fahrgast abgesetzt.

Auf Abbildung 4 kann man gut erkennen, wie und wo der Taster am Greifer angebracht ist.

Der Photosensor, der das Startsignal erkennen muss, ist ebenfalls vorn am Greifer angbracht. Er reagiert, wenn das Licht der Lampe zum Start angeht.

## 4. Quellcode

Diese Variablen sind größtenteils die, die für das Programm verwendet werden.

```
// enthält die Werte, die durch Breitesuche gefunden wurden von einem Startwert  bis zu einem
// Zielwert
char erreichbareKnoten[70];
// Vorgaenger für jedes eingetragene Element
char vorgaenger[70];
// Liste der erreichbaren Fahrgaeste
char fahrgaeste[25];
// Gibt an wie viele Fahrgaeste in Liste vorahnden sind
int neusterUnbelegterFahrgast = 0;
// gibt an wie viele Elemente in erreichbareKnoten und vorgaenger stehen
int anzahlGefundeneElemente = 0;
// gibt an welcher wert in erreichbareKnoten gerade betrachtet wird
int aktuellePosition = 0;
// gibt das Startelement an
char startElement;

// enthält konkrete Abfolge von Elementen vom Ziel- zum Startelement
char teilstrecke[70];
//gibt die Anzahl der Elemente in der Teilstrecke an
int anzahlElementeTeilstrecke = 0;

// beinhaltet den gesamten Ablauf für alle erreichbaren Fahrgäste und ihre Ziele
char gesamtstrecke[500];
```

```
// Anzahl der Elemente Gesamtstrecke

int anzahlElementeGesamtstrecke = 0;



// gibt die aktuelle Ausrichtung des Roboters an

int ausrichtung;
```

Diese Methode setzt die Arrays „erreichbareKnoten“ und „vorgaenger“, sowie die Integer-Variablen, welche angeben, wie viele Elemente in den Arrays vorhanden sind und die aktuelle Position, auf Standardwerte zurück, um sicherzugehen, dass keine falschen Werte für das Programm verwendet werden.

```
void neuenSuchlaufVorbereiten(){

	int i;

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

		erreichbareKnoten[i] = -1;

		vorgaenger[i] = -1;

		anzahlGefundeneElemente = 0;

		aktuellePosition = 0;

	}

}
```

Diese Methode überprüft, ob ein gefundener Knoten bereits im Array der erreichbaren Knoten vorhanden ist und fügt ihn bei Nichtvorhandensein in das Array ein.

```
// Eintragen eines neuerreichbaren Knotens

void trageNeuenKnotenEin(char wert){

	int i;

	int elementBereitsVorhanden = 0;

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

		if(erreichbareKnoten[i] == wert){

			elementBereitsVorhanden = 1;

		}
```

```
		}
		if(!elementBereitsVorhanden){
			anzahlGefundeneElemente++;
			erreichbareKnoten[i] = wert;
			vorgaenger[i] = erreichbareKnoten[aktuellePosition];
		}
}
```

Diese Methode findet die Nachbarn eines Knotens, die in nördlicher, südlicher, westlicher und östlicher Richtung liegen. Ist ein Nachbarknoten nicht befahrbar, hat also den Wert ‚x‘, dann wird er nicht mit in die Liste aufgenommen. Wenn der gerade betrachtete Knoten den Wert ‚F‘ für Fahrgast enthält, werden für ihn die Nachbarn nicht bestimmt, da von dort aus keine neuen Knoten erreichbar sind und um den direkten Übergang von einem Fahrgast zu einem anderen Fahrgast zu verbieten. Handelt es sich jedoch bei dem Startknoten um die Position eines Fahrgastes, ist dies zulässig, da von dort aus der Rückweg geplant werden muss.

```
// finden der erreichbaren Nachbarknoten
void findeNachbarn(){
	char aktElement;
		if(_fa[erreichbareKnoten[aktuellePosition]] != 'F' ||
		_fa[erreichbareKnoten[aktuellePosition]] != startElement){
			// noerdliches Element
			aktElement = erreichbareKnoten[aktuellePosition] – 7;
			if(aktElement > 0 &&
			_fa[aktElement != 'x'){
				trageNeuenKnotenEin(aktElement);
			}
			aktElement = erreichbareKnoten[aktuellePosition] + 7:
			// suedliches Element
			if(aktElement < 70 &&
			_fa[aktElement] != 'x'){
				trageNeuenKnotenEin(erreichbareKnoten[aktElement);
			}
```

```
// westliches Element

                        aktElement = erreichbareKnoten[aktuellePosition] – 1;

                        if(aktElement > 0 &&
                        _fa[aktElement] != 'x'){

                                tragenNeuenKnotenEin(erreichbareKnoten[aktElement);

                        }

                        // oestliches Element

                        aktElement = erreichbareKnoten[aktuellePosition] +1;

                        if(aktElement < 70 &&
                        _fa[aktElement] != 'x'){

                                trageNeuenKnotenEin(erreichbareKnoten[aktElement] + 1);

                        }

        }

}
```

Durch diesen Abschnitt werden alle erreichbaren Fahrgäste von einer Startposition aus gefunden. Dabei wird mit Hilfe der Breitensuche und unter mehrmaliger Verwendung der Methode „findeNachbarn“ das Array „fahrgaeste“ mit den erreichbaren Fahrgästen gefüllt.

```
void findeErreichbareFahrgaeste(){

        erreichbareKnoten[aktuellePosition] = startElement;

        anzahlGefundeneElemente++;

        while(aktuellePosition <= anzahlGefundeneElemente){

                if(_fa[erreichbareKnoten[aktuellePosition]] != 'F'){

                        // Nachbarn finden

                        findeNachbarn();

                }else{

                        // Neuen Fahrgast eintragen

                        fahrgaeste[neusterUnbelegterFahrgast] =

                        erreichbareKnoten[aktuellePosition];

                        neusterUnbelegterFahrgast++;
```

```
			}

			aktuellePosition++;

		}

}
```

Durch diese Methode werden alle Elemente durch Breitensuche von einer Startposition zu einer Zielposition, welche durch „zielMin“ und „zielMax“ definiert wurde, gefunden und im Array „erreichbareKnoten“ gespeichert und zudem die Vorgänger im Array „vorgaenger“ gespeichert.

```
void findeStreckeZumGesuchtenZiel(char zielMin, char zielMax){

		erreichbareKnoten[aktuellePosition] = startElement;

		anzahlGefundeneElemente++;

		// solange nach Nachbarn suchen bis Ziel erreicht wurde

		while(!(erreichbareKnoten[aktuellePosition] >= zielMin &&
		erreichbareKnoten[aktuellePosition] <= zielMax)){

			if(_fa[erreichbareKnoten[aktuellePosition]] != 'F' ||
			 _fa[erreichbareKnoten[aktuellePosition]] != startElement){

				findeNachbarn();

			}

			aktuellePosition++;

		}

}
```

Diese Methode setzt das „teilstrecke“-Array auf Standardwerte zurück.

```
void teilsteckeVorbereiten(){
		int i;

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

			teilstrecke[i] = -1;

		}

		anzahlElementeTeilstrecke = 0;

}
```

Mit dieser Methode werden die Aktionen für eine Teilstrecke ermittelt. Eine Teilstrecke stellt den Weg von einem Startelement zu einem Zielelement dar. Es werden die durch die Breitensuche gefüllten Arrays „erreichbareKnoten“ und „vorgaenger“ genutzt, um den Weg von einem Zielknoten zu dem Startknoten zu ermitteln. Es wird jeweils nach dem Vorgänger eines Elements in dem „erreichbareKnoten“ - Array gesucht, um wieder einen Vorgänger für den nächsten Durchgang zu finden und sich so bis zum Startknoten vorzubewegen. Die so erstellte Strecke wird im Array „teilstrecke“ gespeichert.

```
void erstelleTeilStrecke(){
        char gesucht;
        int j = 0;
        teilsteckeVorbereiten();
        teilstrecke[anzahlElementeTeilstrecke] = erreichbareKnoten[aktuellePosition];
        anzahlElementeTeilstrecke++;
        gesucht = vorgaenger[aktuellePosition];
        // sucht solange Vorgaenger bis Anfangswert gefunden
        while(teilstrecke[anzahlElementeTeilstrecke - 1] != startElement){
                j = 0;
                // Stelle des Wertes des Vorgaengers in den erreichbaren Werten finden
                while(erreichbareKnoten[j] != gesucht){
                        j++;
                }
                teilstrecke[anzahlElementeTeilstrecke] = erreichbareKnoten[j];
                anzahlElementeTeilstrecke++;
                gesucht = vorgaenger[j];
        }

}
```

Diese Methode fügt jeweils die Teilstrecke am Ende der Gesamtstrecke ein. Die Teilstrecke wird hierbei rückwärts eingefügt, da das „teilstrecke“ – Array von Ziel zum Start hin gefüllt wurde.

```
void fuegeTeilstreckeInGesamtstreckeEin(){

        int i = 0;

        // auf gleiche Elemente am Anfang von neuer Teilstrecke und Ende von Gesamtstrecke
        //pruefen



        if(gesamtstrecke[anzahlElementeGesamtstrecke - 1] ==
        teilstrecke[anzahlElementeTeilstrecke - 1]){

                anzahlElementeGesamtstrecke = anzahlElementeGesamtstrecke - 1;

        }

        // Teilstrecke zu Gesamtstrecke hinzufuegen

        while(anzahlElementeTeilstrecke - i > 0){

                gesamtstrecke[anzahlElementeGesamtstrecke] =
                teilstrecke[anzahlElementeTeilstrecke - i - 1];

                i++;

                anzahlElementeGesamtstrecke++;

        }

        startElement = gesamtstrecke[anzahlElementeGesamtstrecke - 1];

}
```

Diese Methode wandelt die Gesamtstrecke in Befehle für den Roboter um. Dazu wird der aktuelle Wert im Array „gesamtstrecke“ vom Folgewert im Array „gesamtstrecke“ abgezogen, sodass die Werte -7, 7, -1, 1 möglich sind. Dann wird die -7 in eine 0, die 1 in eine 1, die 7 in eine 3 und die -1 in eine 3 umgewandelt. Diese Werte geben an, in welche Richtung der nächste Knoten liegt. Mögliche Werte sind Nord (0), Ost (1), Süd (2) und West(3). Da hierbei die aktuelle Ausrichtung nicht betrachtet wird, wird die aktuelle Ausrichtung von der gewünschten Richtung abgezogen, um dies zu berücksichtigen. Die Ausrichtung wird dabei jedes Mal aktualisiert. Da bei dieser Rechnung auch negative Werte herauskommen können, wird das Ergebnis jeweils mit 4 addiert und dann nochmal Modulo 4 gerechnet. So kann der Roboter die definierten Befehle ausführen.

```
void wandleGesamtstreckeInBefehleUm(){

	int i;

	// Richtung der Knoten unter Beachtung der Ausrichtung bestimmen
	// neue Ausrichtung festlegen
	for(i = 0; i < anzahlElementeGesamtstrecke - 1; i++){

		gesamtstrecke[i] = gesamtstrecke[i + 1] - gesamtstrecke[i];

		switch(gesamtstrecke[i]){

		case -7:

			gesamtstrecke[i] = (0 - ausrichtung + 4)%4;

			ausrichtung = 0;

			break;

		case 7:

			gesamtstrecke[i] = (2 - ausrichtung + 4)%4;

			ausrichtung = 2;

			break;

		case -1:

			gesamtstrecke[i] = (3 - ausrichtung + 4)%4;

			ausrichtung = 3;

			break;

		case 1:

			gesamtstrecke[i] = (1 - ausrichtung + 4)%4;

			ausrichtung = 1;

			break;

		}
	}

	// nach Umwandlung eine Stelle weniger benoetigt

	anzahlElementeGesamtstrecke = anzahlElementeGesamtstrecke - 1;

	gesamtstrecke[anzahlElementeGesamtstrecke] = 0;

}
```

Diese Methode erstellt einen kompletten Plan zu allen Fahrgästen und zurück.

```
void erstellePlan(){
	int i = 0;
	neuenSuchlaufVorbereiten();
	findeErreichbareFahrgaeste();
	// ueberpruefen, ob Fahrgaeste vorhanden
	if(neusterUnbelegterFahrgast != 0){
		while(i < neusterUnbelegterFahrgast){
			// Weg zum Fahrgast finden
			neuenSuchlaufVorbereiten();
			findeStreckeZumGesuchtenZiel(fahrgaeste[i],fahrgaeste[i]);
			erstelleTeilStrecke();
			fuegeTeilstreckeInGesamtstreckeEin();
			// Weg zum Ziel des Fahrgastes finden
			neuenSuchlaufVorbereiten();
			findeStreckeZumGesuchtenZiel(64,68);
			erstelleTeilStrecke();
			fuegeTeilstreckeInGesamtstreckeEin();
			i++;
		}
		wandleGesamtstreckeInBefehleUm();
	}
}
```

Diese Methode nutzt der Roboter um rechts abzubiegen, indem der rechte Motor ausgeschaltet wird und er sich solange dreht, bis der linke Sensor wieder eine schwarze Line sieht.

```
void rechtsAbbiegen(){
	motor_pwm(2,7);
	motor_pwm(3,0);
	sleep(2500);
	while(analog(0) < 100);
	if(analog(0) >= 100){
		motor_pwm(3,8);
		motor_pwm(2,8);
	}
}
```

Diese Methode nutzt der Roboter zum Linksabbiegen, indem der linke Motor ausgeschaltet wird und sich solange dreht, bis der rechte Sensor wieder die schwarze Linie sieht.

```
void linksAbbiegen(){
	motor_pwm(2,0);
	motor_pwm(3,5);
	sleep(2500);
	while(analog(2) < 100);
	if(analog(2) >= 100){
	  motor_pwm(3,8);
	  motor_pwm(2,8);
	}
}
```

```
void geradeaus(){
		motor_pwm(2,10);
		motor_pwm(3,10);
		while(analog(0) > 100 && analog(2) > 100){
			sleep(50);
		}
}
```

Nutzt der Roboter zum Wenden.

```
void wenden(){
    motor_richtung(2,1);
    motor_richtung(3,1);
    motor_pwm(2,7);
    motor_pwm(3,4);
    sleep(2900);
        while(analog(2) < 100);
          motor_pwm(3,7);
          motor_pwm(2,7);
          motor_richtung(2,0);
          motor_richtung(3,1);
}


//Hauptprogrammroutine
void AksenMain(void)
{
    int i = 0;
    mod_ir_an();
    lcd_puts("Warte");


 while(dip() == 0){
```

```
     motor_pwm(2,0);

     motor_pwm(3,0);

    }

// Je nach Stellung der Schalter wird Startposition 64 oder 68 verwendet.

    if(dip_pin(0) == 1){

     startElement = 64; // A

    }else if(dip_pin(3) == 1){

     startElement = 68; // B

    }

    mod_ir_aus();

    lcd_cls();

    led(0,1);

    lcd_puts("Berechne");

// erstellt den Plan

    erstellePlan();

// klappt Greifer nach unten

    servo_arc(2,30);


    lcd_cls();

    led(0,0);

    lcd_puts("Bereit");

    while(analog(8) > 21);

// klappt den Greifer nach oben

        servo_arc(2,60);

//der Roboter faehrt los

        motor_richtung(2,0);

        motor_pwm(2,7);

        motor_richtung(3,1);

        motor_pwm(3,7);
```

```
        //while(lcd_ubyte(analog(1)) > 100)
          while(1){
            lcd_setxy(0,0);
            lcd_puts("Rechts:" );
            lcd_ubyte(analog(2));
            lcd_setxy(1,0);
            lcd_puts(" Links:");
            lcd_ubyte(analog(0));
            lcd_ubyte(digital_in(0));
// Wenn Schalter am Greifer betaetigt wird, klappt er nach unten
              if(digital_in(0) == 0){
                      motor_pwm(2,0);
                      motor_pwm(3,0);
                      sleep(100);
                      motor_richtung(2,1);
                      motor_richtung(3,0);
                      motor_pwm(2,4);
                      motor_pwm(3,4);
                      sleep(300);
                      servo_arc(2,30);
                      wenden();
                      i++;
                  }
                  if(i == anzahlElementeGesamtstrecke + 1){
                     motor_pwm(2,0);
                      motor_pwm(3,0);
                      servo_arc(2,60);
                      while(1);
    }
```

```
// Linienfolgen
        if(analog(2) >= 100 && analog(0) < 100){
          motor_pwm(3,6);
          motor_pwm(2,10);
        }else if (analog(2) < 100 && analog(0) < 100){
            motor_pwm(2,10);
            motor_pwm(3,10);
        } else if(analog(0) >= 100 && analog(2) < 100 ){
          motor_pwm(2,6);
          motor_pwm(3,10);
// Ausführung des Plans
        }else if (analog(0) >= 100 && analog(2) >= 100){
                switch(gesamtstrecke[i]){
                        case 0:
                                geradeaus();
                                break;
                        case 1:
                                rechtsAbbiegen();
                                break;
                        case 2:
                                motor_pwm(2,0);
                                motor_pwm(3,0);
                                 sleep(100);
                                 servo_arc(2,60);
                                  wenden();
                                  break;
                        case 3:
                                linksAbbiegen();
                                break;
```

```
                                default:
                                        motor_pwm(2,0);
                                        motor_pwm(3,0);
                                        break;
                                }
                                i++;
                }
        }
}
```

# 5. Anhang

## a. Literaturverzeichnis

[1] "Personal Rapid Transit" ; http://de.wikipedia.org/wiki/Personal_Rapid_Transit#Aktuelle_Projekte ; Zugriff: 15.01.2015

[2 ]Magnus Manske;"PRT Kabine am London Heathrow Airport"; http://de.wikipedia.org/wiki/Personal_Rapid_Transit#mediaviewer/File:ULTra_001.jpg ; Zugriff: 15.01.2015

## b. Abbildungsverzeichnis


| | | |
|---|---|---|
| Abbildung 1 | "PRT Kabine aus London" | Seite 3 |
| Abbildung 2 | "Bauteile" | Seite 5 |
| Abbildung 3 | "Sensoren" | Seite 6 |
| Abbildung 4 | "Greifer" | Seite 7 |
| Abbildung 5 | "fertiger Roboter" | Seite 7 |
| Abbildung 6 | "Getriebe und Motoren" | Seite 10 |