Fachhochschule Brandenburg
University of Applied Sciences
Fachbereich Informatik und Medien

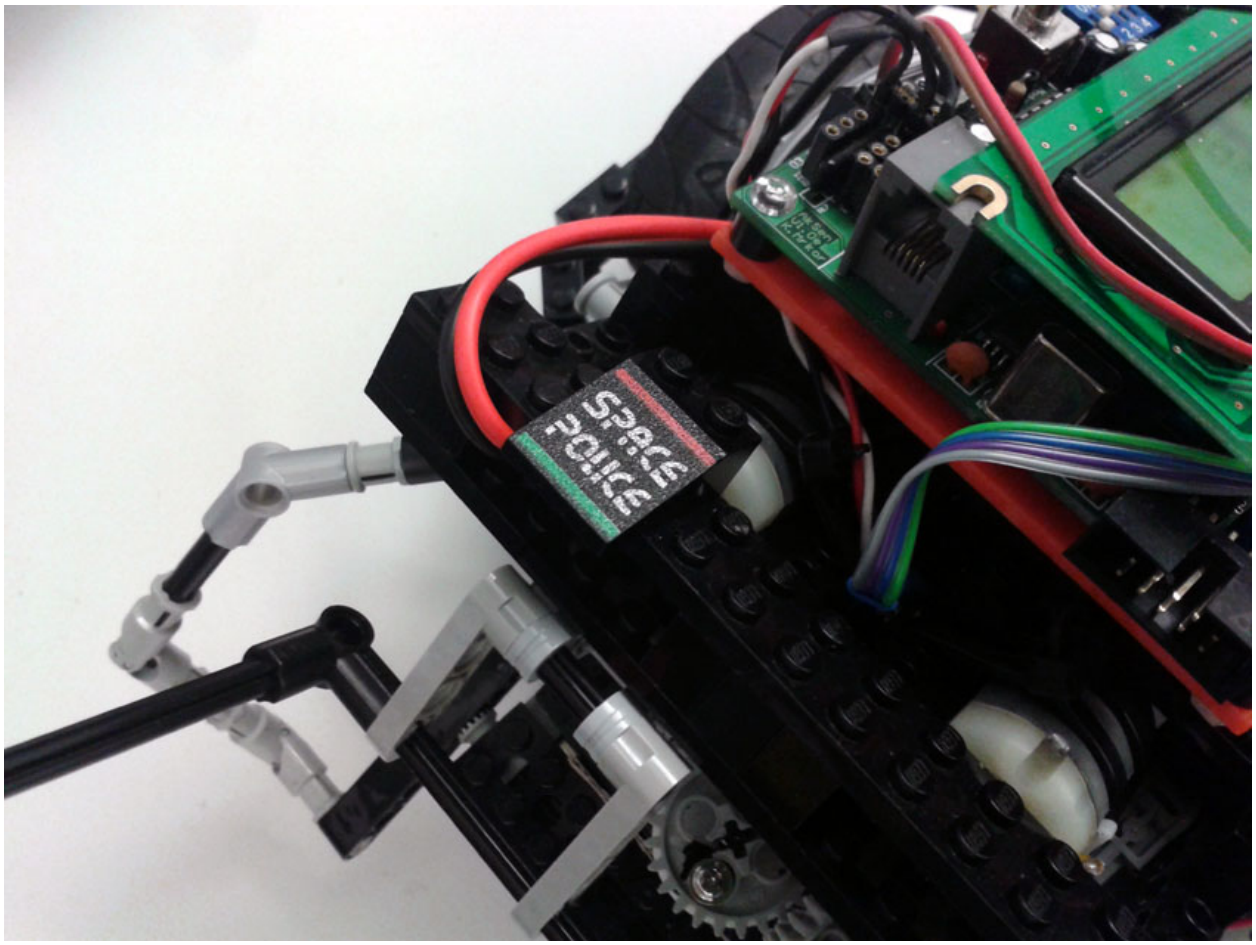
AMS-Projekt “Masdar City 2013/2014”
Prof. Dr. Heinsohn
Dipl. Inf. Boersch

Stephan Bönisch
20110096

Vera Bunde
20110039

AMS-Projekt: “Spacepolice”

Abgabedatum: 16. Januar 2014


SPACE
POLICE

1 Einleitung

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 mit 30:000 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 energieneutral 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. Hier setzt das AMS-Projekt “Masdar City” an.

1.1 Aufgabenstellung

Der Roboter erhält 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. Globales Wissen und die aktuelle Karte der befahrbaren Wege werden dem Roboter kurz vor dem Start zur Verfügung gestellt.

1.2 Vorbereitung

Das Streckennetz (Abb. 1, Aufgabenstellung “AMS-Projekt Masdar City”) besteht aus Strecken, Kreuzungen und den Haltestellen H1 bis H21. Einige Kreuzungen können gesperrt sein, wie bei F2 und F3 in Abb. ??. Die an den Haltestellen wartenden Fahrgäste sollen geholt und bei den Startpunkten A bzw. B südlich der Strecke AB abgesetzt werden. Da die Fahrgäste auf Privatsphäre bestehen, dürfen sie nur einzeln transportiert werden.

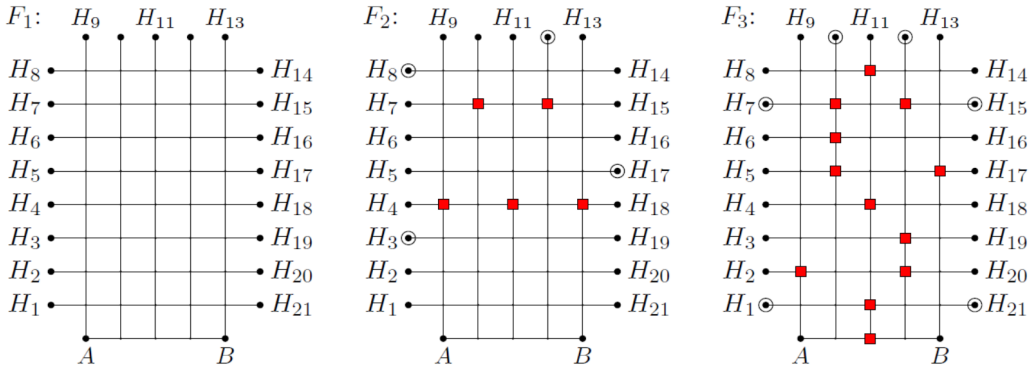


Abb. 1: Streckennetz und Fahraufträge aus der Aufgabenstellung

2 Aufbau des Roboters

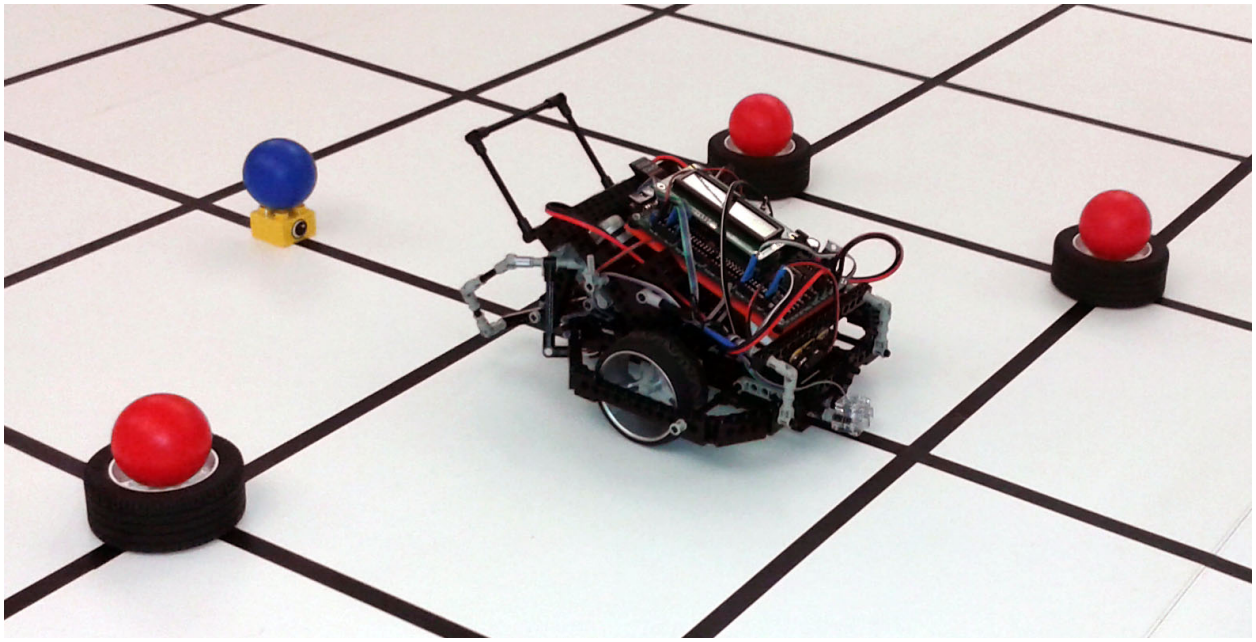


Abb. 2: Roboter im Versuchsaufbau

Die grundlegende Idee war es, einen kompakten und leichten Roboter zu bauen. Dieser sollte sowohl präzise als auch zuverlässig arbeiten. Die Geschwindigkeit sollte der Aufgabe entsprechend ausreichend sein.

Von Anfang an wurde bei der Konstruktion auf Leichtbau geachtet. Dabei kamen zwei grundlegende Konstruktionsprinzipien zum Einsatz: Ein Gitterrahmen als Basis für Radaufhängung und Anbauteile sowie ein flaches Getriebe, welches in einem 45° Winkel versetzt zum Grundrahmen an der Antriebsachse angebracht wurde.

Da es sich beim dritten Auflagepunkt des Roboters um einen abgerundeten Schleifpunkt vor der Antriebsachse handelt, wurde eine Gewichtsverteilung von 35:65 angestrebt. Im Verlauf der Arbeiten stellte sich dies als ausschlaggebender Faktor für die endgültige Geschwindigkeit des Roboters heraus. Bei einer Gewichtsverteilung von 60:40 in der Testphase wurde der Schleifpunkt in kürzester Zeit abgenutzt und die Geschwindigkeit des Roboters sank um etwa 50%.

2.1 Antrieb

Beim Antrieb handelt es sich um zwei eigenständige Elektromotoren, welche parallel an einem Gitterrahmen angebracht sind. Die Rotation wird direkt am Motor um 90° in der Vertikalen gedreht. Damit ist gewährleistet, dass Getriebe und Motoren in einer sehr flachen Bauweise die Kraft direkt auf die Antriebsachse leiten können. Die beiden Räder werden unabhängig voneinander angetrieben und ermöglichen so eine drehzahlgesteuerte Lenkung des Roboters.

Um eine hohe Verwindungssteifigkeit und möglichst reibungsfreie Funktion des Getriebes zu erreichen, wurde die Getriebeaufhängung mit drei Querstreben stabilisiert, gekapselt und mit einer Abdeckung versehen. Mit einer Untersetzung von 81:1 liefert das Getriebe eine vergleichsweise langsame Bewegungsgeschwindigkeit. Die reduzierte Geschwindigkeit des Roboters bringt jedoch Vorteile in der Genauigkeit und Zuverlässigkeit, da die sehr geringe Verarbeitungsgeschwindigkeit des verwendeten Rechners und die geringe Abtastrate so relativiert werden.

2.2 Greifarm

Der Greifarm ist ein simpler Rahmen, welcher um etwa 45° aus der Horizontalen nach oben geschwenkt werden kann. Die Bewegung des Greifarms wird durch einen Servomotor gesteuert. Die Höhe, in welcher der Greifarm angebracht wurde, erlaubt es ihm, die Passagierbälle von ihrem Sockel zu ziehen und danach sicher in alle Richtungen zu manövrieren. Unterstützt wird die Stabilität des Transports durch die Rahmenkonstruktion des versetzten Getriebes.

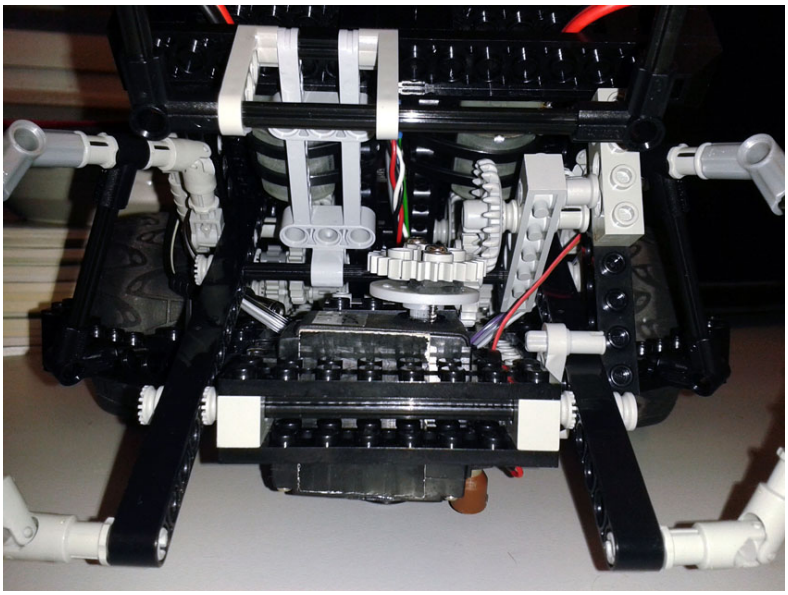


Abb. 3: Servomotor und Greifarm

2.3 Sensorik

Die drei Optokoppler, welche für Linienfolgen und Kreuzungen zählen verwendet werden, sind in einem kompakten Verbund direkt unter der Radachse angebracht. Die Entfernung der beiden vorderen Sensoren (Optokoppler [7] und [0], siehe Abb. 4) entspricht der 1,5-fachen Breite der Linien im Versuchsaufbau. Der dritte Sensor (Optokoppler [14], siehe Abb. 4) liegt genau mittig. Damit ist eine hohe Genauigkeit in der Geradeausfahrt sichergestellt.

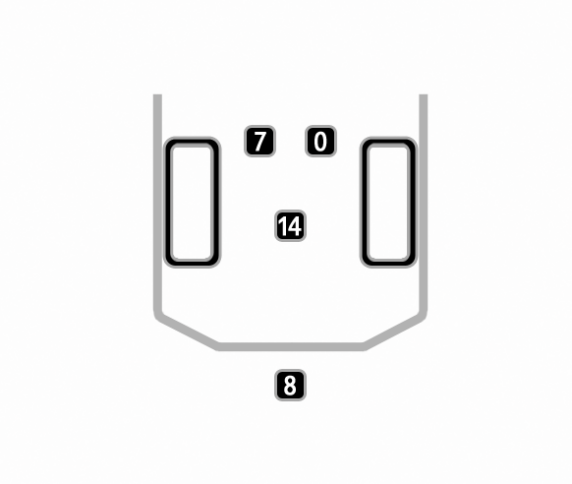


Abb. 4: Optokoppler unter dem Roboter

Ein vierter Sensor (Optokoppler [8], siehe Abb. 4) befindet sich an einem Ausleger hinter dem Roboter. Dieser Sensor ist für die Erkennung der Rotation des Roboters beim Abbiegen notwendig.

2.4 Rechner

Das “Gehirn” des Roboters ist ein Aksen-Board, welches über einen speziellen C-Compiler angesprochen werden kann. Über einen seriellen Anschluss wird das vorher kompilierte Programm auf das Aksen-Board übertragen.

Um die Sensoren und Motoren verwenden zu können, bietet das Aksen-Board digitale und analoge Ein- und Ausgänge sowie LED-Ausgänge und Motortreiber. Des Weiteren gibt es auf dem Board einen DIP-Schalter. Dieser wird im vorliegenden Anwendungsfall dafür genutzt, die Startposition des Roboters festzulegen. Ein LCD-Display kann zur Ausgabe von Informationen genutzt werden. Die Stromversorgung des Boards wird während der Fahrt durch einen Akkumulator gewährleistet.

3 Programme und Funktionen

Die Aufnahme von Sensordaten, deren Verarbeitung zusammen mit vorher definierten Karten sowie die daraus resultierende Steuerung der Aktoren bestimmt die Funktionsweise des Roboters.

3.1 Grundfunktionen

Die Grundfunktionen teilen sich in zwei Gruppen: Die Bewegung und das Greifen. Für die Bewegung werden die 5 Funktionen Geradeausfahren, Links Abbiegen, Rechts Abbiegen, Drehung um 180° sowie Anhalten implementiert. Darüber hinaus erkennt der Roboter über einen Lichtsensor, wann das Startzeichen gegeben wird. Basis für die Lokalisierung der Position des Roboters ist das Erkennen von Kreuzungen sowie das Zählen der selbigen.

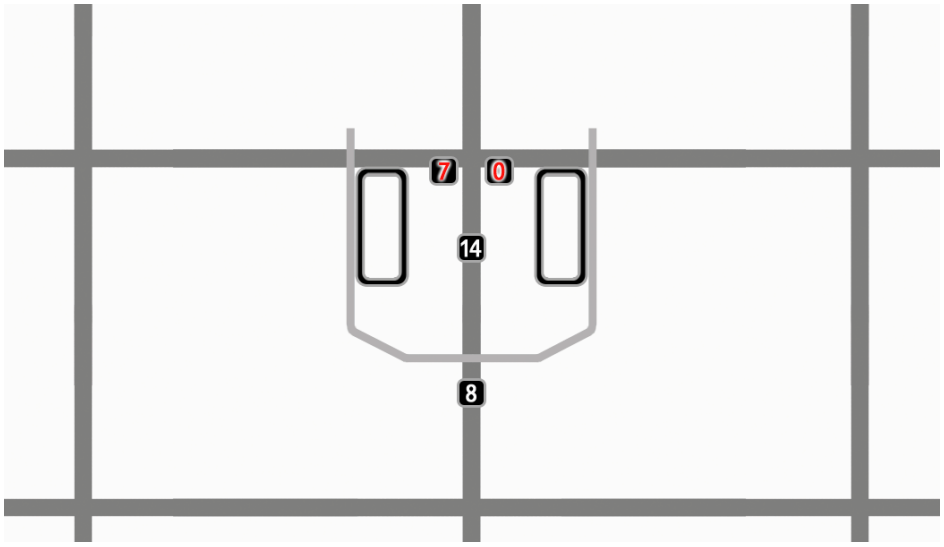


Abb. 5: Kreuzungen erkennen

Wenn die Sensoren [7], [0] und [14] mit Werten für dunkle Flächen ausgelesen werden, befindet sich der Roboter an einer Kreuzung. Wichtig ist dabei vor allem die Änderung der beiden vorderen Sensoren. Um ein Anhalten oder eine Kurskorrektur an Kreuzungen zu verhindern, wird als Kontrollwert Sensor [14] abgefragt. Beim Erreichen einer Kreuzung wird diese gezählt und Trigger für 180°-Drehung und Kurvenfahrt werden freigegeben, um diese Funktionen zu ermöglichen.

3.2 Geradeausfahren

Hierbei handelt es sich im Prinzip um eine einfache Linienfolge. Der Roboter befindet sich auf einer weißen Ebene mit einem Gitternetz aus schwarzen, homogenen Linien. Verwendet werden hierbei die drei Sensoren [7], [0] und [14] (siehe Abb. 6). Trigger für Kurvenfahrt und 180°-Drehung werden gesperrt.

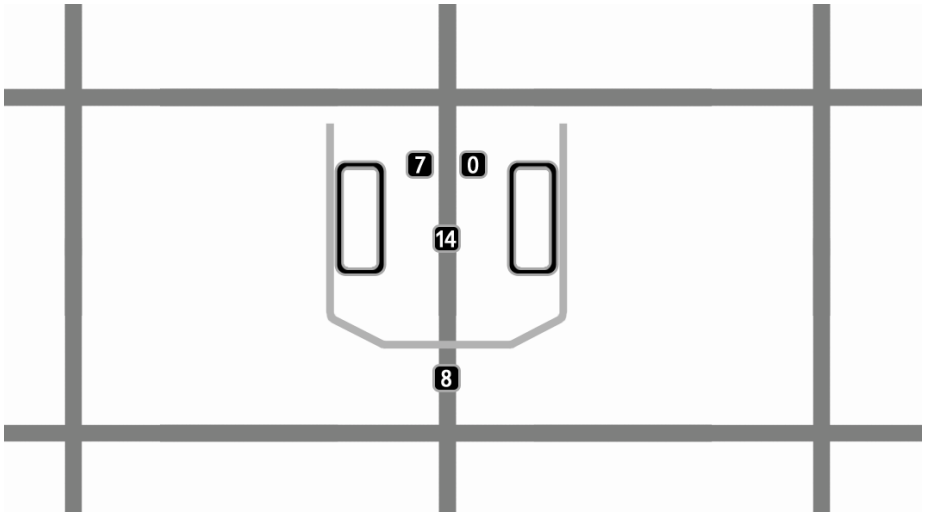


Abb. 6: Linienfolge

Bei der korrekten Vorwärtsbewegung sollten die Optokoppler [7] und [0] einen Wert für helle und der Optokoppler [14] einen Wert für dunkle Oberflächen aufnehmen. Bei einer Abweichung eines der vorderen Sensoren von den erwarteten Werten wird eine Korrektur eingeleitet. Da die Drehung des Roboters über die Drehzahl der Motoren gesteuert wird, wird der Motor, auf dessen Seite der Sensor einen schwarzen Wert zurückgibt, auf 10% gedrosselt, bis beide Sensoren wieder weiße Werte registrieren. Durch die Position der vorderen Sensoren ergibt sich eine ¼-Radumdrehung für die Korrekturen. Damit wird gewährleistet, dass bei Korrekturen nur minimal übersteuert wird.

3.3 Abbiegen

Um möglichst wenig Zeit bei der Kurvenfahrt zu verlieren gilt es, Kurveneintritts- und Kurvenaustrittswinkel möglichst exakt auf die optimale Fahrtrichtung abzustimmen. Damit ist die beste Lösung eine Kreisverbindung der alten und der neuen Bewegungsrichtung auf dem der Bewegung zugrunde liegenden Gitter. Die einfachste Umsetzung hierfür ist es, die Sensoren, welche den Beginn einer Kurvenfahrt erkennen sollen, so weit vor die Radachse zu legen, dass die Drehung durch Anhalten des kurveninneren Rades genau auf der Winkelhalbierenden der sich kreuzenden Linien stattfindet.

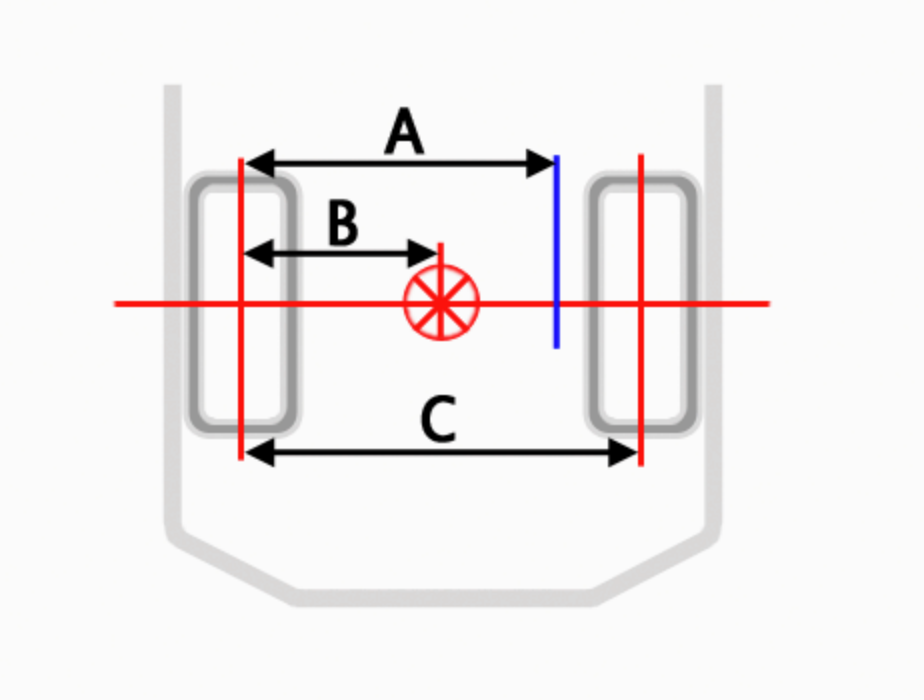


Abb. 7: Drehpunkt

Bauartbedingt ist der Drehpunkt für “Spacepolice” jedoch nicht der Auflagepunkt des kurveninneren Rades (Abstand B vom Mittelpunkt, siehe Abb. 7). Da die Sensoren hier weiter hinten liegen als für die simple Lösung, muss der Drehpunkt durch eine Rückwärtsdrehung des kurveninneren Rades auf der Radachse nach Innen verschoben werden (Schnittpunkt blaue und rote Linie in Abb. 7).

Beginn und Ende einer Kurvenfahrt müssen klar definiert sein. Die Kurvenfahrt beginnt immer an einer Kreuzung. Problematisch ist es, das Ende der Kurvenfahrt verlässlich zu bestimmen. Ein zeitbasierter Ansatz ist aufgrund der schwankenden Akkuleistung nicht realistisch. Abhilfe schafft hier ein vierter Sensor, welcher sich an einem Ausleger hinter dem Roboter befindet.

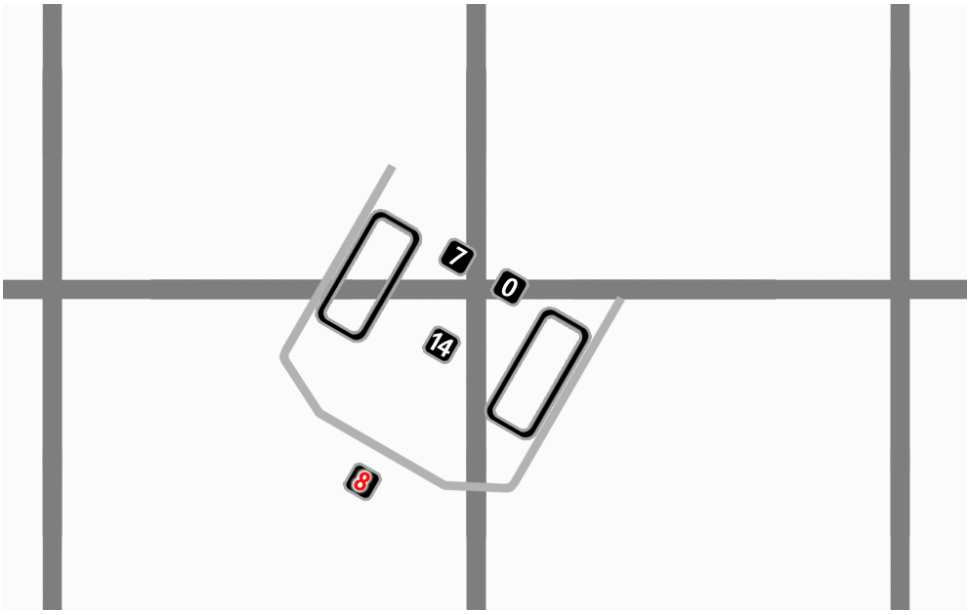


Abb. 8: Kurvenfahrt, Aktivierung des Hinteren Sensors

Nachdem die Kurvenfahrt eingeleitet wurde, dreht sich der Roboter auf dem Drehpunkt B (siehe Abb. 7) ohne Sensoren auszulesen für eine festgelegte Zeit von 400 ms. Nach dieser Zeit befindet sich der Sensor (Optokoppler [8] in Abb. 8) mit hoher Wahrscheinlichkeit über der weißen Fläche und der hintere Sensor wird ab diesem Zeitpunkt ausgelesen. Gibt der Sensor einen Wert für dunkle Flächen zurück (siehe Abb. 8), wird die Kurvenfahrt beendet. Je nach Ladezustand des Akkus ist der Roboter nun genau auf seiner neuen Fahrtlinie ausgerichtet.

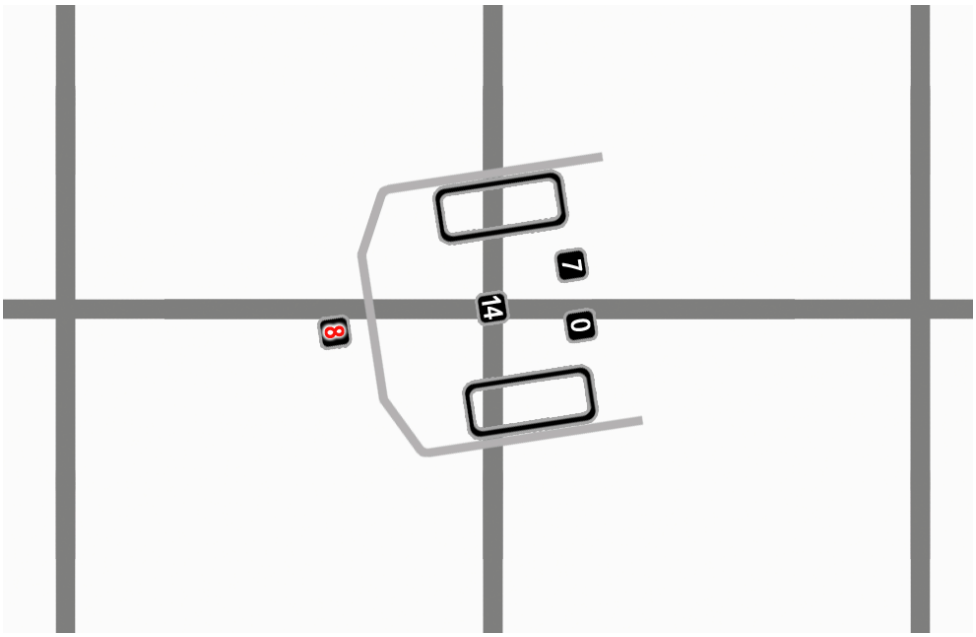


Abb. 9: Kurvenfahrt, erkennen des Kurvenendes

```
void Rechts()
{
  sleep(100);
  if(trigger_kurve == 0)
  {
    trigger_kurve = 1;
    motor_richtung(0,1);
    motor_richtung(2,0);
    motor_pwm(2,10);     //linker motor an
    motor_pwm(0,3);      //rechter motor zurück
    sleep(400);
    while(analog(8) < 25)    //so lange dreh-Sensor = weiß:
    {
        motor_pwm(2,10);     //linker motor an
        motor_pwm(0,3);      //rechter motor zurück
    }
  }
}
```

Im Codebeispiel für die Kurvenfahrt ist gut zu erkennen, wie der Drehpunkt durch die Änderung der Drehzahl beeinflusst wird. Da beide Motoren entgegengesetzt drehen, wäre der Drehpunkt bei voller Leistung genau in der Mitte zwischen den Rädern (Mittelpunkt, siehe Abb. 7). Durch eine 10:3 Verteilung der Leistung wird hier der effektive Drehpunkt auf 75-77% der Achsenlänge verschoben (Strecke A, siehe Abb. 7).

Die Position der vorderen Sensoren zusammen mit der geringen Bewegungsgeschwindigkeit des Roboters ergeben ein sehr konsistentes und zuverlässiges Kurvenverhalten. In einem Testlauf mit 5 verschiedenen Kurslayouts (Headerdatei 2013) wurden die Anforderung in mehr als 80% der dokumentierten Fahrten erreicht. Dabei waren zu keinem Zeitpunkt Fehlfunktionen der Mechanik oder der Software Ursache des Versagens.

Layout	Start	Erg.	Zeit Start/Ziel	Kommentar
FA3	A	o	1:52 min	-
FA3	A	o	1:53 min	-
FA5	A	o	-	Keine Zeitmessung
FA5	B	o	-	Keine Zeitmessung
FA9	A	x	1:39 min	Falsche Seite eingestellt
FA9	A	o	1:30 min	-
FA9	B	o	1:32 min	-
FA10	A	o	2:08 min	Akku nach 11 Fahrten
FA10	B	o	1:58 min	Frischer Akku
FA20	B	x	1:31 min	Ein Ball im Ziel weggeschleudert
FA20	A	o	1:28 min	-

Tabelle 1: Testlauf vom 11.01.2014

3.4 Drehen um 180°

Die Drehung um 180° erfolgt durch einfaches gegeneinander Wirken der Drehrichtung beider Motoren. Hierbei kommt es jedoch zu einem entscheidenden Problem: Wenn die Drehung auf der Kreuzung erfolgt, ist es nicht mehr möglich, direkt nach der 180°-Drehung auf dieser Kreuzung abzubiegen. Um dieses Problem zu lösen, fährt der Roboter eine festgelegte Zeit weiter geradeaus bevor die Drehung eingeleitet wird. Dieses Zeitfenster ist so gewählt, dass es sowohl bei vollem als auch bei niedrigem Ladezustand zum Erfolg führt. Obwohl diese Lösung nicht sehr elegant ist, soll sie für die hier vorliegenden Bedingungen ausreichende Sicherheit gewährleisten.

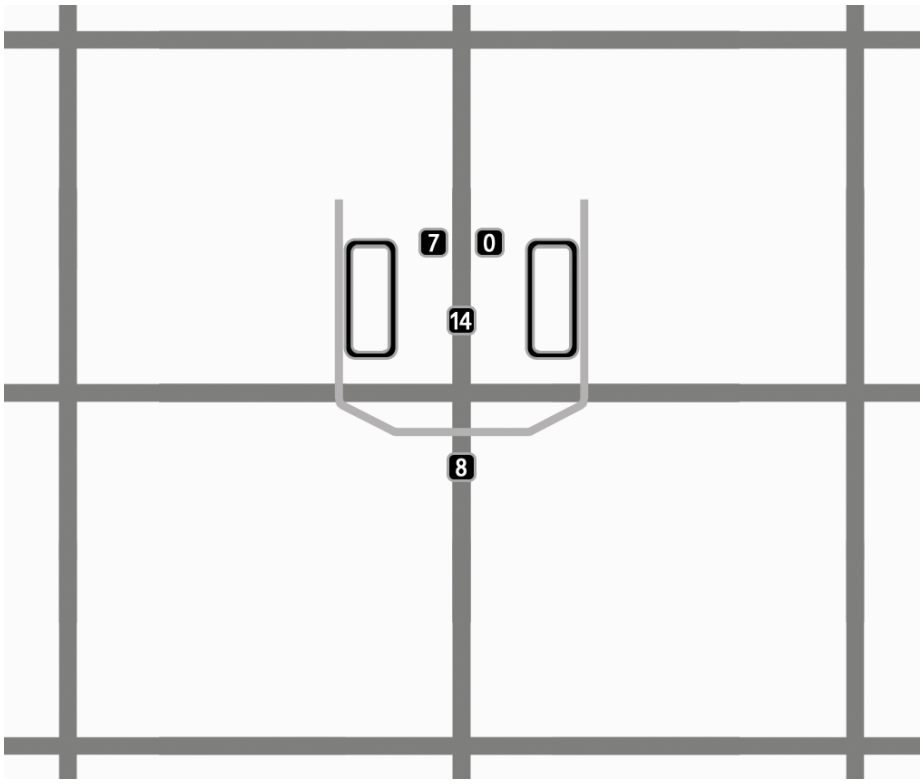


Abb. 10: Sleep-Funktion zur korrekten Positionierung vor der 180°-Drehung

Da Sensor [8] das Ende der Drehung und die korrekte Ausrichtung des Roboters feststellen soll, gibt es zwei mögliche Lösungsansätze: Zählen von beiden überstrichenen schwarzen Linien oder blindes Drehen und Abfragen der ersten schwarzen Linie nach einer bestimmten Zeit. Beide Linien zu zählen führte hier zu guten Ergebnissen. In vier von fünf Testläufen konnte der Roboter das Ende der Drehung erkennen.

Der Ansatz, die Zeit zu bestimmen, hat sich jedoch auch hier als der zuverlässigere herausgestellt. Nachdem die Drehung eingeleitet wurde, wird wie bereits bei der Kurvenfahrt für eine festgelegte Zeit gedreht. Danach erkennt Sensor [8] die Linie zur korrekten Ausrichtung.

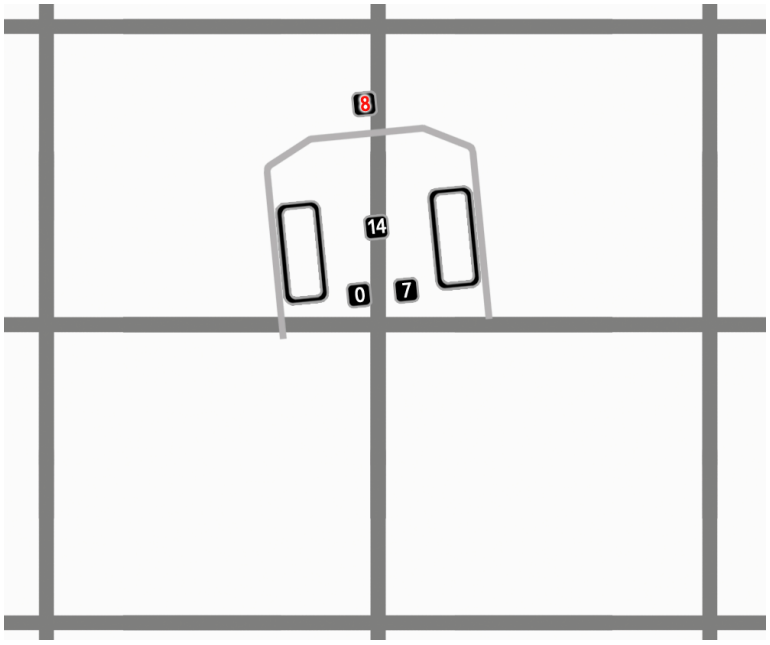


Abb. 11: Optokoppler [8] erkennt Ende der Drehung

Da sich die beiden vorderen Sensoren zur Kreuzungserkennung hinter der querverlaufenden Linie befinden, ist es nun wieder möglich, an der selben Kreuzung abzubiegen.

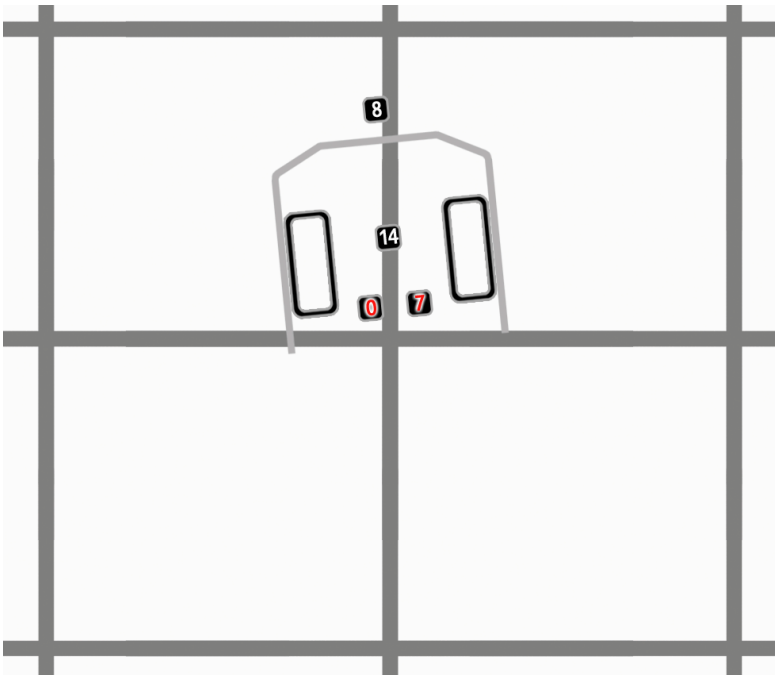


Abb. 12: Optokoppler [0] und [7] können Kreuzung erkennen

3.5 Greifen

Der Greifarm wird über einen Servomotor direkt angesteuert. Es sind die Funktionen “Greifer Hoch” und “Greifer Runter” implementiert. Diese Funktionen setzen den Winkel des Servomotors auf die korrekte Position. Der Greifer selbst besteht aus einem Rahmen. Dieser ist auf einer Höhe angebracht, in der er im herabgelassenen Zustand sowohl Bälle vom jeweiligen Podest ziehen, als auch sicher vor sich her bewegen kann. Die Podeste werden dabei nicht berührt. Durch die Gitterrahmenkonstruktion und die stabilisierenden Streben am Getriebe stehen Links und Rechts des Greifers zwei Streben etwa 2,5 cm ab. Diese können bei einer weit nach vorne verlagerten Passagierposition das jeweilige Podest einklemmen. Der ungeplante Transport eines Podestes stellt dabei kein Problem für die Pfadsicherheit des Roboters dar, die Geschwindigkeit leidet jedoch. Zur Kontrolle wird auf dem LCD-Display des Aksen-Boards die Position des Greifers ausgegeben.

```
void GreiferHoch()
{
  servo_arc(0,85);
  lcd_cls();
  lcd_puts("HOCH");
}

void GreiferRunter()
{
  servo_arc(0,43);
  lcd_cls();
  lcd_puts("RUNTER");
}
```

3.6 Erweiterte Funktionen

Erweiterte Funktionen sind zusammengesetzt aus Grundfunktionen. Darunter fallen alle Möglichkeiten, die unter normalen Bedingungen auf dem Feld zur Anwendung kommen können: Das Greifen eines Objektes mit anschließender 180°-Drehung sowie Abbiegen mit anschließendem Greifen und 180°-Drehung jeweils nach Links und Rechts. Auch das Abliefern eines Passagiers mit anschließender 180°-Drehung nach einer Links- oder Rechtskurve sowie aus der Geradeausfahrt sind modelliert.

4 Suche und Wegfindung

Die eigentliche Fahrtplanung des Roboters wird in drei Schritten realisiert. Im ersten Schritt werden die vorher unbekannten Karten mit den Informationen über Fahrgastpositionen und gesperrte Kreuzungen aus einer Header-Datei eingelesen. Anschließend wird anhand dieser Daten der kürzeste Weg von der per DIP-Schalter gewählten Startposition bis zu allen erreichbaren Fahrgästen gesucht. Im letzten Schritt werden diese Weginformationen in Bewegungsanweisungen für den Roboter übersetzt.

4.1 Breitensuche

Die Breitensuche besteht aus zwei Unterfunktionen: Dem Erstellen einer Kostenmatrix und dem Suchen des kürzesten Weges anhand der Kosten.

Die Kostenmatrix wird im Ausgangszustand mit hohen Startwerten gefüllt. Anschließend wird die aktuelle Startposition abgerufen und mit dem Kostenwert “0” versehen. Die Kostenmatrix wird nun so lange expandiert, bis ein Eintrag niedriger als der Startwert gefunden wird, welcher keinen Passagier enthält. Von diesem Punkt aus wird nun in jede Richtung ((1,0), (0,1), (-1,0), (0,-1)) auf gesperrte Kreuzungen geprüft. Ist die Kreuzung nicht gesperrt, wird der Kostenwert der Ausgangskreuzung um +1 erhöht und mit dem Kostenwert der gefundenen Kreuzung verglichen. Ist der erhöhte Kostenwert der Ausgangskreuzung geringer als der Kostenwert der gefundenen Kreuzung, so erhält die gefundene Kreuzung den neuen Kostenwert. Dieses Fluten der Kostenmatrix ist dann beendet, wenn in einem Durchgang keine Änderungen mehr an der Kostenmatrix vorgenommen werden.

Nach dem Fluten der Kostenmatrix wird die eigentliche Anzahl der zu erreichenden Passagiere bestimmt. Diese Anzahl zusammen mit den Indizes der jeweiligen Position wird gespeichert. Die vollständig geflutete Kostenmatrix wird von der Position aller erreichbaren Passagiere aus expandiert. Die Passagiere sind dabei die Ausgangsknoten und die Nachbarfelder ergeben jeweils die Kindknoten. Es wird jeweils der Knoten expandiert, welcher die geringsten Kosten hat. Gibt es mehrere Knoten mit den selben Kosten, wird der zuerst gefundene Knoten expandiert. Ist der Knoten mit den Kosten “0” (Startpunkt) gefunden, ist die Suche beendet. Die Indizes der expandierten Knoten ergibt die Reihenfolge der abzufahrenden Kreuzungen von den Passagieren zum Startpunkt. Diese Rückwege werden jeweils in einem eigenen Array gespeichert.

4.2 Fahrplan

Der Fahrplan wird direkt aus den Rückwegen der Suche erstellt. Dabei werden die jeweiligen gespiegelten Arrays als Hinwege mit ihren Rückwegen in einzelnen Arrays gespeichert. Anschließend werden die Hin- und Rückwege in das finale Fahrauftrags-Array in der richtigen Reihenfolge kopiert. Dabei werden nur Teilstrecken berücksichtigt, wenn diese mindestens einen Schritt enthalten.

Dieser Fahrplan bildet die Grundlage der Roboterbewegung. Im Array befinden sich nun die Indizes der Kreuzungen (siehe Abb. 13) in der Reihenfolge, in der sie vom Roboter überfahren werden müssen. Um die Richtungssteuerung für den Roboter nutzen zu können fehlen noch einige Informationen. Es werden sowohl Informationen über die Positionen der Fahrgäste, der Startpunkte und der jeweiligen Richtungsänderungen benötigt.

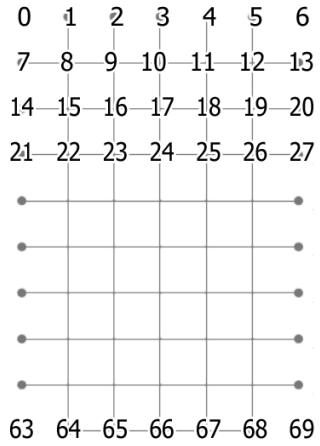


Abb. 13: Indizes der Kreuzungen im Array, Auszug

Die Analyse der Indizes zur Laufzeit ergibt nutzbare Richtungsvektoren zur Auftragsplanung. Dabei bewegt sich der Roboter in einer Ebene (x,y). Die Kreuzungen sind in Leserichtung indiziert (siehe Abb. 13). Ist beispielsweise die Differenz von *aktueller Index* - *folgender Index* = 7, so ist der Vektor, welcher an der aktuell anstehenden Kreuzung die Bewegung vorgibt (0,1). Für die drei anderen Differenzen ergibt sich dementsprechend -7 (0,-1), 1 (-1,0), -1 (1,0) (siehe Abb. 14).

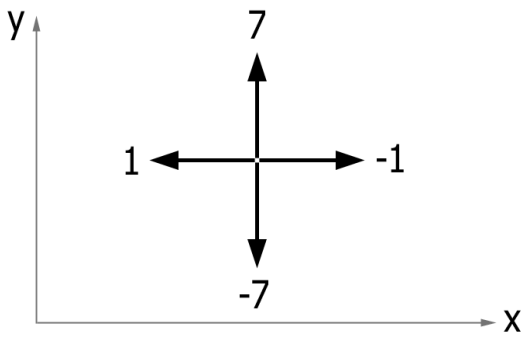


Abb. 14: Vektoren beschriftet mit ihren Index-Differenzen

Wird ein Fahrplan wie (22, 15, 8, 9, 2) übersetzt, werden die Vektoren aus der Index-Differenz zweier benachbarter Indizes mit der letzten bekannten Richtung (ebenfalls ein Vektor) des Roboters verglichen. Dieser Vergleich führt zu relativen Bewegungsanweisungen aus Robotersicht, welche in einem Array gespeichert werden.
Die relativen Bewegungsanweisungen sind 0 (keine Richtungsänderung), 1 (Kurve Rechts), -1 (Kurve Links), 2 (Kurve Rechts und Passagier aufnehmen), -2 (Kurve Links und Passagier aufnehmen), 3 (keine Richtungsänderung und Passagier aufnehmen), 4 (Kurve Rechts und Passagier abliefern), -4 (Kurve Links und Passagier abliefern) sowie 5 (keine Richtungsänderung und Passagier abliefern).

Das Anweisungsarray *CLF[]* wird abschließend bereinigt. Eine führende Null in der Startanweisung wird hinzugefügt, um die imaginäre “nullte” Kreuzung zu berücksichtigen. Konvertierungsfehler in Form von überflüssigen Nullen nach Aufnahme und Abgabe von Passagieren werden entfernt. Die Länge des resultierenden Arrays wird als *weglaengeF* hinterlegt. Das Anweisungsarray wird in der Hauptmethode des Aksen-Boards durchlaufen und in Verbindung mit dem Kreuzungszähler werden die dort enthaltenen Anweisungen an der jeweiligen Kreuzung ausgeführt (Fallunterscheidung).

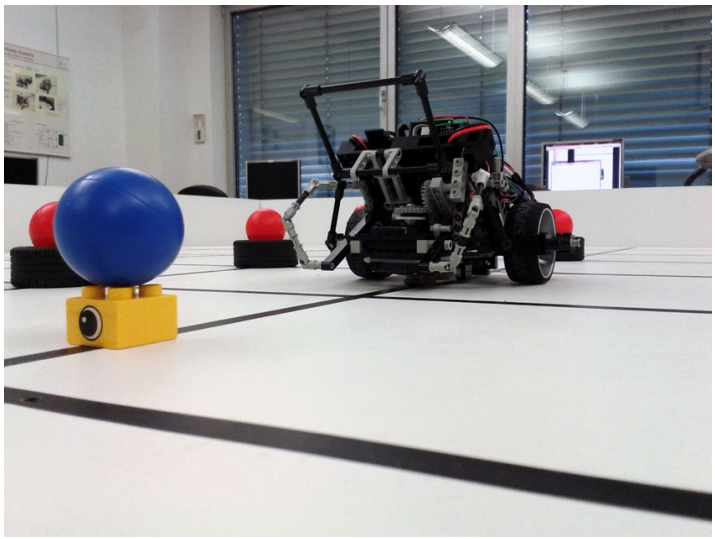


Abb. 15: “Spacepolice” in freier Wildbahn

5 Problemfelder

Während des Projektes gab es immer wieder Hürden, welche überwunden werden mussten. Generell waren diese Probleme in zwei Kategorien einzuteilen: Konstruktion und Programmierung. Beide konnten nicht unabhängig voneinander betrachtet werden. Die Unzulänglichkeiten in der Konstruktion bedingten Änderungen im Programm und rechenintensive oder komplexe und damit unsichere Funktionen bedingten eine Änderung der Konstruktion, um einfachere Ansätze verfolgen zu können.

5.1 Konstruktion

Das erste und schwerste Problem welches auftrat, war der hohe Leistungshunger der beiden Elektromotoren. Diese verlangten nach einem Anlaufstrom von mehr als 3,5 A. Da die Stromquelle diese Leistung nicht erbrachte und auch eine stärkere Untersetzung um den Faktor 3 nicht ausreichte, wurden die Motoren durch schwächere ausgetauscht. Damit reduzierte sich die Geschwindigkeit des Roboters erheblich.

Ein weiteres Problem war die Gewichtsverteilung. Die Position des Akkus war hierbei entscheidend. Anfänglich war die Position über der Radachse geplant. Um Gewicht zu sparen, wurde die Aufhängung für das Aksen-Board überarbeitet. Dies führte dazu, dass der ursprüngliche Platz für den Akku belegt wurde. Die neue Position des Akkus war nun über dem Schleifpunkt. Nach wenigen Testläufen zeigten sich bereits extreme Abnutzungserscheinungen am Schleifpunkt. Daraufhin wurde ein Kompromiss zwischen Gewichtsverteilung und endgültigem Gewicht gefunden, welcher die Geschwindigkeit des Roboters sowie die Lebenserwartung des Schleifpunktes erhöhte. Ein schnelles Tauschen des Akkus ist im finalen Design immer noch möglich, wobei ein fester und sicherer Halt gewährleistet ist. Das etwas zu kurze Kabel und die damit ungünstige Kabelführung für den Kurvensensor direkt vor der Akkuladeklappe verhindern jedoch ein wirklich zufriedenstellendes Ergebnis.

Der Servomotor und dessen Position unter dem Greifarm machten eine 1:1 Übersetzung der Drehbewegung erforderlich. Der Greifarm selbst durfte nicht behindert werden, wobei der übermäßig groß dimensionierte Servomotor einen festen Platz in der Gitterrahmenkonstruktion einnehmen musste. Die Umlenkung der Rotation erfolgt dabei mit nur einem Aufhängungspunkt freischwebend und wird nur durch die Querverstrebung am Getriebegehäuse ohne direkte Verbindung gestützt. Durch die so erreichte minimale Bewegungsfreiheit der Umlenkung ist diese Konstruktion unerwartet robust und der gesamte Greifmechanismus operiert über den Erwartungen.

5.2 Programmierung

Während der ersten Testläufe rückten verschiedene Sonderfälle von Bewegungskombinationen in den Fokus. So war es mit den anfänglich implementierten Funktionen nicht möglich, direkt nach einer Kurvenfahrt einen Fahrgast zu greifen. Ebenso mussten Anpassungen erfolgen, um nach 180°-Drehungen die gerade überfahrene Kreuzung erneut zu zählen, um so ein direktes Abbiegen zu ermöglichen.

Die Feinabstimmung der vielen zeitbasierten Funktionen erforderte viele Testläufe, da diese auch bei verschiedenen Ladezuständen des Akkus funktionieren mussten. Ein weiterer Faktor in diesen Feinabstimmungen waren Änderungen an der Untersetzung des Getriebes sowie der Gewichtsverteilung. Beide Faktoren veränderten die Geschwindigkeit des Roboters und mussten in den Zeitfenstern beachtet werden.

Die Implementierung des Suchverfahrens verzögerte sich erheblich. Die ersten Tests konnten erst am 11.01.2013 erfolgen. Durch den Zeitverzug von mehr als 3 Wochen über die geplante Zeit hinaus fielen gerade in diesem Bereich viele geplante Funktionen dem Rotstift zum Opfer. So war es nicht mehr möglich, die Drehungen in die Kostenrechnung einzubeziehen. Im Extremfall ergibt sich dadurch ein geschätzter Zeitverlust von etwa 15 Sekunden in einem typischen Fahrauftrag mit 3 Passagieren.

Die Umwandlung der Suchergebnisse in Kommandos zur Steuerung des Roboters erwies sich als tückisch. Die Bewegungsfunktionen waren zum Zeitpunkt der Implementierung der Umwandlungsmethode bereits finalisiert und stabil. Da die Aufnahmebewegungen automatisch die 180°-Drehung ausführen, mussten diese Drehungen in einem zusätzlichen Schritt gefiltert werden. Das Ergebnis ist eine unübersichtliche Umwandlungsmethode, die jedoch zuverlässige Ergebnisse liefert.

6. Fazit

Das AMS-Projekt Masdar City 2013/2014 stellte die Teilnehmer vor zahlreiche Herausforderungen. Mit guten Ratschlägen und viel Erfahrung stand das Betreuerteam bestehend aus Herrn Boersch und Herrn Heinsohn den Teilnehmern zur Seite. Die Tore des KI-Labors waren beinahe durchgehend geöffnet und ermöglichten so eine flexible und intensive Arbeit am Projekt. Insgesamt stellte dieses Projekt ein absolutes Highlight im Wintersemesters 2013/2014 dar und der hohe Arbeitsaufwand war angesichts der sichtbaren Ergebnisses gerechtfertigt.

Im abschließenden Wettkampf konnte sich “Spacepolice” mit hoher Genauigkeit und Zuverlässigkeit gegen die Konkurrenz durchsetzen. Erreicht wurde die maximale Punktzahl von 124 Punkten in 4 Läufen. Leider verhinderte die geringe Geschwindigkeit des Roboters in Kombination mit dem ungünstig ausgelosten Startpunkt einen Sieg im Finale.

7 Anhang

Quellcode in Version 1.01 vom 11.01.2014

```
//Autoren: Vera Bunde und Stephan Bönisch
#include <stub.h>
#include <stdio.h>
#include <stdlib.h>
// ******************************
// Einlesen eines Fahrauftrages:
// Ändern des #define auf den gewünschten 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 FA20
#include "fa.h"
// ******************************
/*
 - - - SENSOREN - - - - - -

analog(0)                    Optokoppler vor Achse Rechts
analog(7)                    Optokoppler vor Achse Links
analog(14)                   Optokoppler hinter Achse Mitte
analog(11)                   Optokoppler im Ausleger als Kurventester
Optokopplerwerte             10-25 = Hell, 100-256 = Dunkel

analog(8)                    Lichttest bei Kunstlicht (Licht an <100, Licht aus >140, Regelfall ohne
                             Licht ~130-190)
- - - MOTOREN - - - - - - -

motor_pwm(3,X)       Motor Links
motor_pwm(0,X)       Motor Rechts
motor_richtung(X,0)  Motor X vor
motor_richtung(X,1)  Motor X zurück
servo_arc(0,45)      Servomotor 0, Winkel 45°
servo_arc(0,90)      Servomotor 0, Winkel 90°
*/
```

```
// - - - Globale Variablen - - -
int i = 0;
int j = 0;
int n = 0;
int m = 0;
int k = 0;
int l = 0;
int changeFlag = 0;              //Änderung an der Kostenmatrix
int endpunkt = -1;
int heading = 9;                 //bisher gefahrene Richtung
unsigned long endezeit;          //Timer für while (zum Begradigen der Laufrichtung zwischen Kreuzungen)
int timerGet = 260;
int timerDrop = 350;
int x = 0;                       //Kreuzungszähler
int sPos = 0;                    //Startposition
int sPosPoint = 0;
int drehung = 0;                 //Drehungszustand
int trigger_kreuzung = 0;        //wird gerade über eine Kreuzung gefahren?
int trigger_kurve = 0;           //wird gerade Abgebogen?
int trigger_start = 0;           //wird auf ein Startsignal gewartet?
int ready = 0;                   //Abschluss der Vorbereitung
int go = 0;                      //aktive Fahrt

// - - - Input und Suche - - -
int kosten[77];
int fahrgaeste[4];
unsigned char _faCopy[78];
int hweg1[30];
int hweglaenge1 = 0;
int hweg2[30];
int hweglaenge2 = 0;
int hweg3[30];
int hweglaenge3 = 0;
int rweg1[30];
int rweglaenge1 = 0;
int rweg2[30];
int rweglaenge2 = 0;
int rweg3[30];
int rweglaenge3 = 0;
int CL[180];
int CLH[180];
int weglaenge = 0;
int weglaengeF = 1;
int CLF[180];
```

```
// - - - Commandlist - - -

/*Commands werden direkt beim Überfahren einer Kreuzung erteilt, sie gelten für die nächstfolgende
Kreuzung:

"0" folgende Kreuzung wird geradeaus überfahren
"1" folgende Kreuzung wird rechts abgebogen
"-1" folgende Kreuzung wird links abgebogen

"2" an folgender Kreuzung wird Rechts abgebogen und ein Ball abgeholt
"-2" an folgender Kreuzung wird Links abgebogen und ein Ball abgeholt
"3" folgende Kreuzung wird kurz überfahren und es wird sich vor der übernächsten Kreuzung um 180°
gedreht, dabei wird ein Ball aufgenommen

"4" an folgender Kreuzung wird Rechts abgebogen und der Ball abgegeben
"-4" an folgender Kreuzung wird Links abgebogen und der Ball abgegeben
"5" folgende Kreuzung wird kurz überfahren und es wird sich vor der übernächsten Kreuzung um 180°
gedreht, dabei wird der Ball abgegeben
*/

// - - - Basisfunktionen - - - -

void GreiferHoch()
{
        servo_arc(0,85);
        lcd_cls();
        lcd_puts("HOCH");
}

void GreiferRunter()
{
        servo_arc(0,43);
        lcd_cls();
        lcd_puts("RUNTER");
}
```

```
void Kreuzungen()
{
	if((analog(0) > 50) && (analog(7) > 50))   //Frontsensoren = schwarz -> Kreuzung
	{
		if (trigger_kreuzung == 0)   //Wenn gerade keine Kreuzung überfahren wird...
		{
			trigger_kreuzung = 1;		//Kreuzung wird gerade überfahren
			trigger_kurve = 0;		//Kurvenfahrt wird freigegeben
			x = x + 1;
			lcd_cls();
			lcd_puts("X = ");
			lcd_uint(x);
		}
	}
	else if((analog(0) < 40) && (analog(7) < 40))
	//Frontsensoren weiß -> Kreuzung wurde erfolgreich überfahren
	{
		trigger_kreuzung = 0;
	}
}

void Vor()  //Geradeaus ohne Anhalten mit Korrekturen
{
	motor_richtung(0,0);
	motor_richtung(2,0);
	if((analog(0) > 50) && (analog(7) < 50))   //Rechter Sensor =  Schwarz
	{
		motor_pwm(0,1);			//Motor aus recht
	}
	else
	{
		motor_pwm(0,9);			//Motor an rechts
	}

	if ((analog(7) > 50) && (analog(0) < 40))  //Linker Sensor = schwarz
	{
		motor_pwm(2,1);			//Motor aus = links
	}
	else
	{
		motor_pwm(2,10);		//Motor an links
	}
}
```

```
void Rechts()
{
	endezeit = akt_time() + 40; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)		//Wenn Kurvenfahrt freigegeben...
	{
		trigger_kurve = 1;
		//Es wird gerade abgebogen, keine weitere Kurvenfahrt bis nächste Kreuzung
		motor_richtung(0,1);
		motor_richtung(2,0);
		motor_pwm(2,10);		//linker Motor an
		motor_pwm(0,3);		//rechter Motor zurück
		sleep(400);
		while(analog(11) < 25)	//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,10);		//linker Motor an
			motor_pwm(0,3);		//rechter Motor zurück
		}
	}
}

void Links()
{
	endezeit = akt_time() + 40; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0) //Wenn Kurvenfahrt freigegeben...
	{
		trigger_kurve = 1;
		//Es wird gerade abgebogen, weitere Kurvenfahrten bis
		zur nächsten Kreuzung gesperrt
		motor_richtung(0,0);
		motor_richtung(2,1);
		motor_pwm(2,3);			//linker Motor an
		motor_pwm(0,9);		//rechter Motor zurück
		sleep(400);
		while(analog(11) < 25)		//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,3);			//linker Motor an
			motor_pwm(0,9);		//rechter Motor zurück
		}
	}
}
```

```
void Dreh180drop()  //Drehung 180° wobei der Ball losgelassen wird
{
	endezeit = akt_time() + 710 //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)
	{
		trigger_kurve = 1;
		sleep(650);
		motor_pwm(2,0);			//Motoren aus
		motor_pwm(0,0);			//Motoren aus
		GreiferHoch();
		sleep(300);
		motor_richtung(0,1);
		motor_richtung(2,0);
		motor_pwm(2,10);		//Motor an links
		motor_pwm(0,9);			//Motor an rechts
		sleep(1200);
		while(analog(11) < 25)		//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,10);
			motor_pwm(0,9);
		}
	}
}

void Dreh180get()  //Drehung 180° wobei der Ball geholt wird
{
	endezeit = akt_time() + 650; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)
	{
		trigger_kurve = 1;
		sleep(550);
		motor_pwm(2,0);				//Motoren aus
		motor_pwm(0,0);				//Motoren aus
		GreiferRunter();
		sleep(300);
		motor_richtung(0,0);
		motor_richtung(2,1);
		motor_pwm(2,10);			//Motor an links
		motor_pwm(0,9);				//Motor an rechts
		sleep(1200);
```

```
                while(analog(11) < 25)                  //so lange Dreh-Sensor = weiß:
                {
                        motor_pwm(2,10);
                        motor_pwm(0,9);
                }
        }
}

void RechtsGet() //Rechts abbiegen und dabei Ball aufheben
{
        endezeit = akt_time() + 40; //von jetzt an genau n ms…
        while (endezeit > akt_time()) //...wird vorwärts gefahren
        {
                Vor();
        }
        if(trigger_kurve == 0)
        {
                trigger_kurve = 1;
                motor_richtung(0,1);
                motor_richtung(2,0);
                motor_pwm(2,10);                        //linker Motor an
                motor_pwm(0,3);                         //rechter Motor zurück
                sleep(400);
                while(analog(11) < 25)                  //so lange Dreh-Sensor = weiß:
                {
                        motor_pwm(2,10);                //linker Motor an
                        motor_pwm(0,3);                 //rechter Motor zurück
                }
        }
        endezeit = akt_time() + timerGet;       //von jetzt an genau n ms...
        while (endezeit > akt_time())           //...wird vorwärts gefahren
        {
                Vor();
        }
        GreiferRunter();
        sleep(300);
        motor_richtung(0,0);
        motor_richtung(2,1);
        motor_pwm(2,10);                        //Motor an links
        motor_pwm(0,10);                        //Motor an rechts
        sleep(1200);
        while(analog(11) < 25)                  //so lange Dreh-Sensor = weiß:
        {
                motor_pwm(2,10);
                motor_pwm(0,10);
        }
}
```

```
void LinksGet() //Links abbiegen und dabei Ball aufheben
{
	endezeit = akt_time() + 40; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)
	{
		trigger_kurve = 1;
		motor_richtung(0,0);
		motor_richtung(2,1);
		motor_pwm(2,3);			//linker Motor an
		motor_pwm(0,10);		//rechter Motor zurück
		sleep(400);
		while(analog(11) < 25)		//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,3);			//linker Motor an
			motor_pwm(0,10);		//rechter Motor zurück
		}
	}
	endezeit = akt_time() + timerGet;
	while (endezeit > akt_time())
	{
		Vor();
	}
	GreiferRunter();
	sleep(300);
	motor_richtung(0,0);
	motor_richtung(2,1);
	motor_pwm(2,10);			//Motor an links
	motor_pwm(0,10);			//Motor an rechts
	sleep(1200);
	while(analog(11) < 25)			//so lange Dreh-Sensor = weiß:
	{
		motor_pwm(2,10);
		motor_pwm(0,10);
	}
}
```

```
void RechtsDrop() //Rechts abbiegen und dabei Ball fallen lassen
{
	endezeit = akt_time() + 40; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)
	{
		trigger_kurve = 1;
		motor_richtung(0,1);
		motor_richtung(2,0);
		motor_pwm(2,10);			//linker Motor an
		motor_pwm(0,3);				//rechter Motor zurück
		sleep(400);
		while(analog(11) < 25)			//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,10);		//linker Motor an
			motor_pwm(0,3);			//rechter Motor zurück
		}
	}
	endezeit = akt_time() + timerDrop;	//von jetzt an genau n ms...
	while (endezeit > akt_time())		//...wird vorwärts gefahren
	{
		Vor();
	}
	GreiferHoch();
	sleep(300);
	motor_richtung(0,1);
	motor_richtung(2,0);
	motor_pwm(2,10);			//Motor an links
	motor_pwm(0,10);			//Motor an rechts
	sleep(1200);
	while(analog(11) < 25)			//so lange Dreh-Sensor = weiß:
	{
		motor_pwm(2,10);
		motor_pwm(0,10);
	}
}
```

```
void LinksDrop() //Links abbiegen und dabei Ball fallen lassen
{
	endezeit = akt_time() + 40; //von jetzt an genau n ms…
	while (endezeit > akt_time()) //...wird vorwärts gefahren
	{
		Vor();
	}
	if(trigger_kurve == 0)
	{
		trigger_kurve = 1;
		motor_richtung(0,0);
		motor_richtung(2,1);
		motor_pwm(2,3);			//linker Motor an
		motor_pwm(0,10);		//rechter Motor zurück
		sleep(400);
		while(analog(11) < 25)		//so lange Dreh-Sensor = weiß:
		{
			motor_pwm(2,3);		//linker Motor an
			motor_pwm(0,10);	//rechter Motor zurück
		}
	}
	endezeit = akt_time() + timerDrop;
	while (endezeit > akt_time())
	{
		Vor();
	}
	GreiferHoch();
	sleep(300);
	motor_richtung(0,1);
	motor_richtung(2,0);
	motor_pwm(2,10);		//Motor an links
	motor_pwm(0,10);		//Motor an rechts
	sleep(1200);
	while(analog(11) < 25)		//so lange Dreh-Sensor = weiß:
	{
		motor_pwm(2,10);
		motor_pwm(0,10);
	}
}

void Stop()
{
	motor_pwm(2,0);		//Motor aus links
	motor_pwm(0,0);		//Motor aus rechts
	sleep(200);
}
```

```
// - - - Vorbereitung/Suche - - -

void Fahrauftrag()
{
	lcd_cls();
	lcd_puts("FA Nr: ");
	lcd_ubyte(_fa_nr);
	for(i = 0; i < 70; i++)
	{
		_faCopy[i] = _fa[i];
	}
	for(i = 70; i < 77; i++)
	{
		_faCopy[i] = 'x';
	}
}

void Startposition()
{
	if(dip_pin(0)==1)
	{
		sPos = 64;
		lcd_cls();
		lcd_puts("Start A");
	}
	else if (dip_pin(0)==0)
	{
		sPos = 68;
		lcd_cls();
		lcd_puts("Start B");
	}
}

void CheckLight()
{
	go = 0;
	if(trigger_start == 0)
	{
		while(!go)
		{
			if(analog(8) < 85)
			{
				trigger_start = 1;
				go = 1;
			}
		}
	}
}
```

```
int checkX(int k)
{
	if((_faCopy[k] == 'x')){ return 1;}
	else{ return 0;}
}

int checkF(int k)
{
	if((_faCopy[k] == 'F')){ return 1;}
	else{ return 0;}
}

int checkStern(int i, int k)
{
	if(i < 7)
	{
		if(kosten[i + 7] == (k - 1)){ return (i + 7);}
	}
	else
	{
		if(kosten[i + 1] == (k - 1)){ return (i + 1);}
		if(kosten[i - 1] == (k - 1)){ return (i - 1);}
		if(kosten[i + 7] == (k - 1)){ return (i + 7);}
		if(kosten[i - 7] == (k - 1)){ return (i - 7);}
	}
	return 0;
}

void Suche()
{
//Zurücksetzen der Kostenmatrix:
	for(i = 0; i < 70; i++)
	{
		kosten[i] = 99;
	}
//Setzen der Startposition ohne Kosten
	kosten[sPos] = 0;
//Fluten der Kostenmatrix
	changeFlag = 1;
	while(changeFlag)
	{
		changeFlag = 0;
		for(i = 69; i >= 0; i--)
		{
			if(checkX(i) == 0 && checkF(i) == 0 && kosten[i] < 99)
			{
				if(!checkX((i + 1)) && (kosten[(i + 1)] > kosten[i]) &&
				((kosten[i] + 1) < kosten[(i + 1)]))
```

```
                        {
                                kosten[(i + 1)] = kosten[i] + 1;
                                changeFlag = 1;
                        }
                        if(!checkX((i + 7)) && (kosten[(i + 7)] > kosten[i]) &&
                        ((kosten[i] + 1) < kosten[(i + 7)]))
                        {
                                kosten[(i + 7)] = kosten[i] + 1;
                                changeFlag = 1;
                        }
                        if(!checkX((i - 1)) && (kosten[(i - 1)] > kosten[i]) && ((kosten[i]
                        + 1) < kosten[(i - 1)]))
                        {
                                kosten[(i - 1)] = kosten[i] + 1;
                                changeFlag = 1;
                        }
                        if(!checkX((i - 7)) && (kosten[(i - 7)] > kosten[i]) && ((kosten[i]
                        + 1) < kosten[(i - 7)]))
                        {
                                kosten[(i - 7)] = kosten[i] + 1;
                                changeFlag = 1;
                        }
                }
        }
}
//Einfügen der Startposition als "-1" in Kostenmatrix (zusätzliche Zeile)
for(i = 70; i < 78; i++)
{
        if(i == (sPos + 7))
        {
                kosten[i] = -1;
        }
        else
        {
                kosten[i] = 99;
        }
}
//Finde erreichbare Fahrgäste (maximal 3)
fahrgaeste[0] = -1;
fahrgaeste[1] = -1;
fahrgaeste[2] = -1;
fahrgaeste[3] = -1;
j = 0;
```

```
        for(i = 0; i < 70; i++)
        {
                if((_faCopy[i] == 'F') && (kosten[i] < 99))
                {
                        j++;
                        fahrgaeste[j] = i;
                }
        }
        //Finde kürzesten Weg von Fahrgästen zum Start und speichere die
        Kreuzungsnummern in Arrays für Hinweg und Rückweg
        for(n = 1; n <= j; n++)
        {
                l = 0;
                i = fahrgaeste[n];
                k = kosten[i];
                if(n == 1){rweg1[l] = i; rweglaenge1 = 1;}
                if(n == 2){rweg2[l] = i; rweglaenge2 = 1;}
                if(n == 3){rweg3[l] = i; rweglaenge3 = 1;}
                while(!(kosten[i] < 0))
                {
                        l++;
                        i = checkStern(i,k);
                        if(n == 1){rweg1[l] = i; rweglaenge1++;}
                        if(n == 2){rweg2[l] = i; rweglaenge2++;}
                        if(n == 3){rweg3[l] = i; rweglaenge3++;}
                        k--;
                }
                if(n == 1){for(m = rweglaenge1-2; m > 0; m--){hweg1[hweglaenge1] =
                rweg1[m]; hweglaenge1++;} rweglaenge1--;}
                if(n == 2){for(m = rweglaenge2-2; m > 0; m--){hweg2[hweglaenge2] =
                rweg2[m]; hweglaenge2++;} rweglaenge2--;}
                if(n == 3){for(m = rweglaenge3-2; m > 0; m--){hweg3[hweglaenge3] =
                rweg3[m]; hweglaenge3++;} rweglaenge3--;}
        }
}
```

```
void MapToCommand()
{
//Startposition neu festlegen als Punkt in der Wand (wie Passagiere)
        sPosPoint = sPos + 7;
//Wege kombinieren
        weglaenge = 0;
        for(i = 0; i < hweglaenge1; i++){CLH[weglaenge] = hweg1[i];   weglaenge++;}
        for(i = 0; i < rweglaenge1; i++){CLH[weglaenge] = rweg1[i];   weglaenge++;}
        if(rweglaenge2 > 0){CLH[weglaenge] = sPosPoint; weglaenge++;}
        for(i = 0; i < hweglaenge2; i++){CLH[weglaenge] = hweg2[i];   weglaenge++;}
        for(i = 0; i < rweglaenge2; i++){CLH[weglaenge] = rweg2[i];   weglaenge++;}
        if(rweglaenge3 > 0){CLH[weglaenge] = sPosPoint; weglaenge++;}
        for(i = 0; i < hweglaenge3; i++){CLH[weglaenge] = hweg3[i];   weglaenge++;}
        for(i = 0; i < rweglaenge3; i++){CLH[weglaenge] = rweg3[i];   weglaenge++;}
        CLH[weglaenge] = sPosPoint;
        weglaenge++;
//Richtungen mappen und Fahrtplanung in CL schreiben
        CL[0] = 0;
        heading = 7;
        endpunkt = 0;
        for(i = 0; i < weglaenge; i++)
        {
                if(endpunkt == 1)
                {
                        endpunkt = 0;
                        continue;
                }
                else
                {
//-------------------------------------------------------------------------------
        if((!(checkF(CLH[i + 1]))) && (CLH[i + 1] != sPosPoint))
        //normale Fahrt die nicht als nächste Kreuzung einen Passagier oder den
        Startpunkt hat
        {
                if((CLH[i] - CLH[i+1]) == 1)
                {
                        if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine
                        änderung an heading*/ CL[i+1] = 0;}
                        if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach
                        links*/ CL[i+1] = -1; heading = 1;}
                        if(heading - (CLH[i] - CLH[i+1]) == -8){ /*änderung heading
                        nach rechts*/ CL[i+1] = 1; heading = 1;}
                }
                if((CLH[i] - CLH[i+1]) == -1)
                {
                        if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine
                        Änderung an heading*/ CL[i+1] = 0;}
```

```
                        if(heading - (CLH[i] - CLH[i+1]) == -6){ /*Änderung heading nach
                        links*/ CL[i+1] = -1; heading = -1;}
                        if(heading - (CLH[i] - CLH[i+1]) == 8){ /*Änderung heading nach
                        rechts*/ CL[i+1] = 1; heading = -1;}
                }
                if((CLH[i] - CLH[i+1]) == 7)
                {
                        if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine
                        Änderung an heading*/ CL[i+1] = 0;}
                        if((heading - (CLH[i] - CLH[i+1])) == -6){ /*Änderung
                        heading nach links*/ CL[i+1] = 1; heading = 7;}
                        if((heading - (CLH[i] - CLH[i+1])) == -8){ /*Änderung heading nach
                        rechts*/ CL[i+1] = -1; heading = 7;}
                }
                if((CLH[i] - CLH[i+1]) == -7)
                {
                        if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine
                        Änderung an heading*/ CL[i+1] = 0;}
                        if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach
                        rechts*/ CL[i+1] = 1; heading = -7;}
                        if(heading - (CLH[i] - CLH[i+1]) == 8){ /*änderung heading nach
                        links*/ CL[i+1] = -1; heading = -7;}
                }
        }
//-------------------------------------------------------------------------
else if(((checkF(CLH[i + 1])) && (CLH[i + 1] != sPosPoint))
//Passagier abholen
{
        if((CLH[i] - CLH[i+1]) == 1)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine
                Änderung an heading*/ CL[i+1] = 3; heading = -1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach links*/
                CL[i+1] = -2; heading = -1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -8){ /*änderung heading nach rechts*/
                CL[i+1] = 2; heading = -1; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == -1)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine änderung an
                heading*/ CL[i+1] = 3; heading = 1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -6){ /*änderung heading nach links*/
                CL[i+1] = -2; heading = 1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 8){ /*änderung heading nach rechts*/
                CL[i+1] = 2; heading = 1; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == 7)
        {
```

```
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine änderung an
                heading*/ CL[i+1] = 3; heading = -7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -6){ /*änderung heading nach rechts*/
                CL[i+1] = 2; heading = -7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -8){ /*änderung heading nach links*/
                CL[i+1] = -2; heading = -7; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == -7)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine änderung an
                heading*/ CL[i+1] = 3; heading = 7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach rechts*/
                CL[i+1] = 2; heading = 7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 8){ /*änderung heading nach links*/
                CL[i+1] = -2; heading = 7; endpunkt = 1;}
        }
}
//-------------------------------------------------------------------------------------
else if((!(checkF(CLH[i + 1]))) && (CLH[i + 1] == sPosPoint)) //Passagier abliefern
{
        if((CLH[i] - CLH[i+1]) == 1)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine Änderung an
                heading*/ CL[i+1] = 5; heading = -1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach links*/
                CL[i+1] = -4; heading = -1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -8){ /*änderung heading nach rechts*/
                CL[i+1] = 4; heading = -1; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == -1)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine Änderung an
                heading*/ CL[i+1] = 5; heading = 1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -6){ /*änderung heading nach links*/
                CL[i+1] = -4; heading = 1; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 8){ /*änderung heading nach rechts*/
                CL[i+1] = 4; heading = 1; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == 7)
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine Änderung an
                heading*/ CL[i+1] = 5; heading = -7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -6){ /*änderung heading nach rechts*/
                CL[i+1] = 4; heading = -7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == -8){ /*änderung heading nach links*/
                CL[i+1] = -4; heading = -7; endpunkt = 1;}
        }
        if((CLH[i] - CLH[i+1]) == -7)
```

```
        {
                if(heading == (CLH[i] - CLH[i+1])){ /*geradeaus + keine Änderung an
                heading*/ CL[i+1] = 5; heading = 7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 6){ /*änderung heading nach rechts*/
                CL[i+1] = 4; heading = 7; endpunkt = 1;}
                if(heading - (CLH[i] - CLH[i+1]) == 8){ /*änderung heading nach links*/
                CL[i+1] = -4; heading = 7; endpunkt = 1;}
        }
    }
  }
}
//Bereinigen von CLF[]
        weglaengeF = 1;
        CLF[0] = 0;
        for(i = 1; i < weglaenge; i++)
        {
                if((CL[i - 1] != 2) && (CL[i - 1] != -2) && (CL[i - 1] != 3) && (CL[i - 1] != 4) &&
                (CL[i - 1] != -4) && (CL[i - 1] != 5))
                {
                        CLF[weglaengeF] = CL[i];
                        weglaengeF++;
                }
        }
        ready = 1;
}

void init()
{
        int x = 0;              //Sicherheit: Kreuzungszähler wird auf Null gesetzt
        GreiferHoch();          //Sicherheit: Greifer wird in Ausgangsposition gebracht
        sleep(500);
        Fahrauftrag();          //Fahrauftrag wird eingelesen
        sleep(2000);
        Startposition();        //Startposition wird eingelesen
        sleep(2000);
        lcd_cls();
        lcd_puts("Berechne Weg...");
        Suche();                        //Kostenmatrix + Suche
        MapToCommand();         //Suchergebnis wird in Commandlist übersetzt
        if(ready == 1)
        {
                lcd_cls();
                lcd_puts("Bereit!");
                CheckLight(); //Warten auf Startsignal
        }
}

void MappingFahrt()
```

```
{
	if(x < weglaengeF)
	{
	//Kreuzungen werden immer gezählt als Grundlage für die Positionsbestimmung
		Kreuzungen();
		Vor();
	//Auslesen der Richtungsanweisungen aus der Commandlist CLF:
		if(CLF[x] == -1 && trigger_kurve == 0) { Links(); }

		else if(CLF[x] == 1 && trigger_kurve == 0) { Rechts(); }
		else if(CLF[x] == -2 && trigger_kurve == 0) { LinksGet(); }
		else if(CLF[x] == 2 && trigger_kurve == 0) { RechtsGet(); }
		else if(CLF[x] == 3 && trigger_kurve == 0) { Dreh180get(); }

		else if(CLF[x] == -4 && trigger_kurve == 0) { LinksDrop(); }
		else if(CLF[x] == 4 && trigger_kurve == 0) { RechtsDrop(); }
		else if(CLF[x] == 5 && trigger_kurve == 0) { Dreh180drop(); }
	}
	else
	{
	//Wenn das Ende der Commandlist erreicht ist -> Anhalten
		Stop();
		sleep(2000);
	}
}

void AksenMain(void)
{

// - - - Vorbereitung - - -
	init();
// - - - Hauptschleife - - -
	if(go)
	{
		while(1)
		{
			MappingFahrt();
		}
	}
}
```