Dokumentation

Entwicklertagebuch und technische Hintergründe

Zur Entwicklung von

V4P8S

Während des

AMS-Projektes im WS 12/13

Inhaltsverzeichnis

Die Aufgabe

Das Ziel des Projektes ist es einen autonomen Lego-Roboter zu entwickeln, der sich auf einem schachbrettartigen Spielfeld selbstständig orientiert und verschiedene Zielpunkte anfährt. Dabei werden auf dem Spielfeld bis zu 3 Zielobjekte platziert, die der Roboter selbstständig ansteuern, aufsammeln und in den Zielbereich befördern muss.
Das Spielfeld setzt sich zusammen aus 70 Straßenkreuzungen, einigen gesperrten Kreuzungen, 2 Start- bzw. Zielpunkte und bis zu insgesamt 6 Zielobjekten (max. 3 Zielobjekte pro Team).
Die einzige Eingabe die der Bot erhält ist der aktuelle Aufbau des Spielfeldes. Daraus berechnet er den optimalen Weg zu den einzelnen Zielobjekten, die er nacheinander in den Zielbereich befördern muss. Der Roboter enthält ein Navi, das für jede mögliche Karte den optimalen Weg berechnen kann. Für die Abarbeitung des Fahrauftrages hat jedes Team 2 Minuten Zeit. Nach Ablauf dieser Zeit muss der Bot seine Arbeit einstellen.

Das Entwickler-Team

Arne Maury – Aufbau des Roboters, Ingenieursarbeiten

Denis Weil – Hardwareabstraktion des Roboters, Programmierung

Christoph Gresch – Logik und Streckenfindung, Navigation

Entwickler-Tagebuch

Woche 1, 24. - 28. September

Zu Beginn des Projektes macht sich das Team mit der Aufgabenstellung und der Zielsetzung vertraut. Das Team diskutiert die Aufgabe und denkt über erste Umsetzungs- und Planungsansätze nach. Des Weiteren schaffen wir uns einen Überblick über den Bausatz, den vorhandenen Teilen und den Sensoren.
Wir studieren das Handbuch des AKSEN-Boards und testen die Funktionalität der einzelnen Komponenten (u.a. Optokoppler, Motoren, …), und wie diese angesteuert werden.
Nun werden erste Ideen zum physischen Aufbau diskutiert. Wie soll der Antrieb aussehen? Kleine Räder, große Räder oder gar Ketten? Die Idee des Kettenantriebs wird jedoch schnell verworfen, da aufgrund der zu hohen Reibungszahl unnötig Geschwindigkeit verloren gehen würde. Der klassische Radantrieb mit kleinen Rädern wird nun beschlossen, um den Bot wendiger und stabiler zu halten. Ein weiteres zentrales Problem ist das Ein- und Ausladen der Zielobjekte. Ideen sind unter anderem der klassische Greifarm, ein Gabelstapler-Modell oder auch eine Art Bagger. Im weiteren Verlauf werden einzelne mögliche Greifer-Methoden konstruiert, getestet und diskutiert. Die Entscheidung wird jedoch vertagt und der Bau vorsichtig begonnen.
Parallel beginnen erste Überlegungen zur Umsetzung des Navigationssystems. Wir diskutieren über Tiefen- und Breitensuche sowie Bestensuche und A*-Algorithmus.

Woche 2, 01. - 05. Oktober

Wir beginnen sofort mit dem weiteren Bau des Roboters. Das Grundgerüst des Bots wird fertiggestellt. Somit kann eine erste hart programmierte Testfahrt stattfinden, um die Motorik zu testen. Dazu wird auch ein zeitbedingtes Abbiegen einprogrammiert. Um nun auch eine stabile Fahrt zu gewährleisten, diskutieren wir den Einsatz der Optokoppler. Wie viele benötigen wir? Eins, zwei oder drei? Die Umsetzung mit einem Optokoppler wird als schwierig eingestuft, da damit die Erkennung von Kreuzungen nahezu unmöglich ist. Auch bei der Variante mit zwei Optokopplern sind für uns Probleme erkennbar, diesmal im Abbiegeprozess. Deswegen erscheint uns die Installation mit drei Optokopplern vorerst optimal.
Nachdem die Optokoppler montiert wurden, wenden wir uns nun dem Transport-Modul zu. Zwei Ideen werden getestet: der Greifarm und das Gabelstapler-Modul. Die anfänglichen Bedenken beim erschwerten Be- und Entladen mit dem Gabelstapler-Modell bewahrheiten sich und wir verwerfen diese Idee. Der Greifarm wird nun in Zusammenarbeit mit einem Servomotor montiert.

Woche 3, 08. - 12. Oktober

Nachdem sich auch die Variante mit drei Optokopplern als unzureichend erwiesen hat, werden nun vier Optokoppler verarbeitet, zwei zentral und jeweils einer links bzw. rechts. Der Servo-Motor wird demontiert, da er sich als fehleranfällig offenbarte und auch der Greifarm wird demontiert und die Idee wegen Unzuverlässigkeit verworfen.
Im weiteren Verlauf des Tages optimieren wir die einzelnen Hard- und Softwarekomponenten. Die Optokoppler werden durch Anpassen der Schwellwerte optimiert und der Abbiegeprozess wird durch verschiedene Einstellungen feinjustiert. Auch die Motorengeschwindigkeit wird angepasst, da trotz gleicher Werte die physische Umsetzung beider Motoren teils stark variiert!?
Am Ende des Tages widmen wir uns erneut dem Schließmechanismus. Der Servo-Motor wird ausgetauscht und der Greifarm durch eine einfache Schließkabine mit Schranke ersetzt. Dadurch erhoffen wir uns Platzeinsparungen am Bot und ein effizienteres Arbeiten des Greifers.

Woche 4, 22. - 26. Oktober

Ab diesem Zeitpunkt des Projektes spaltet sich die Gruppe in zwei Verantwortungsbereiche. Arne und Denis widmen sich weiterhin dem Bau und der physischen Umsetzung des Bots, ich beginne parallel mit der Entwicklung der Navigationskomponente.
Die erste Frage in der Entwicklung des Navis ist der Einsatz des Arrays. Wäre ein ein- oder ein zweidimensionales Array effizienter? Ich beginne mit der Entwicklung beider Varianten, um beide hinsichtlich Effizienz und Rechenzeit zu testen. Beide Varianten werden mit einem Fahrcounter auf jeder Kreuzung und der Labyrinthregel umgesetzt. Die erste Variante wird aufgrund von Algorithmus-Schwierigkeiten verworfen, die Idee mit einem zweidimensionalen Array wird intensiv fortgesetzt. Ein Routenplan konnte mit dieser Idee erfolgreich zurückgegeben werden. Im weiteren Verlauf der Entwicklung wird aber auch diese Implementierung verworfen, da sie bei bestimmten Karten weder flexibel, noch effizient arbeiten würde.
Indes haben Arne und Denis teils unverständliche Probleme mit dem Roboter. Er weicht von der Linie ab und findet nicht zurück, das Korrigieren ist teilweise zu stark oder schwach und der Bot dreht sich ab und zu auch in einer Endlosschleife. Bei der Fehlersuche werden verschiedene Ansätze gewählt und Komponenten getestet. Sind die Optokoppler in Ordnung? Arbeiten die Motoren zuverlässig? Funktionieren die Sensoren? Nach einigen Tests und Anpassung von Motorengeschwindigkeit und Schwellwerte der Optokoppler kann wieder eine fehlerfreie Fahrt nach hart programmiertem Fahrplan erzielt werden.

Woche 5, 29. Oktober - 02. November

Zu Beginn des Tages wird die Fahrt des Bots erneut getestet. Unverständlicherweise kann keine erfolgreiche Fahrt beobachtet werden, obwohl das zuletzt genutzte Programm noch immer auf dem Board vorhanden war. Es folgt eine erneute gemeinsame Fehleranalyse. Alle Komponenten werden getestet, verschiedenste Probleme werden entdeckt. Die einzig nachvollziehbare Diagnose für uns war ein Defekt am AKSEN-Board. Deshalb wird ein neues

Board aufgesetzt, die Verkabelung und die Anschlüsse neu verlegt. Nachdem das Programm erneut auf dem Board geladen wurde, konnte es fehlerfrei durchlaufen werden.
Verschiedene Fahrten werden getestet: ein Quadrat, eine kleine Acht, eine große Acht. Alle Fahrten konnten, auch in einer Endlosschleife, erfolgreich durchgeführt werden. Der Bewegungsablauf war flüssiger als bei früheren Fahrten. Die Steuerung des Bots erfolgte mithilfe eines hart programmierten char-Arrays mit den Befehlen l (links), r (rechts), g (gerade).
Auch beim Navi wird ein neuer Ansatz gewählt. Die Idee vom eindimensionalen Array wird wieder aufgegriffen. Aber diesmal ersetze ich Fahrcounter und Labyrinthregel durch eine Kostenmatrix, die mittels Breitensuche aufgebaut wird. Die Planung für die neue Idee wird noch theoretisch erarbeitet und noch nicht implementiert.

Woche 6, 05. - 09. November

Erneut wird beim ersten Testen ein fehlerhafter Programmablauf beim Roboter entdeckt. Wieder testen Arne und Denis die einzelnen Komponenten und führen einen Code-Review durch. Am Ende der Überarbeitung stellt sich heraus, dass die physische Umsetzung sehr stark von der Akkuleistung abhängig ist und dadurch auch identische Programmstrukturen bei späteren Durchläufen Fehler aufzeigen können. Nach dieser Beobachtung achten wir immer verstärkt auf die Akkuleistung. In den folgenden Testfahrten zeigt sich ein neuer gravierender Fehler auf: eine LED ist ausgeschaltet. Diese war für einen fehlerfreien Programmablauf jedoch notwendig. Nachdem Arne und Denis auch diesen Fehler beheben kann der Bot wieder fehlerfrei fahren.
Des Weiteren nimmt Arne leichte Umbauten an der Fahrgastkabine vor, um diese stabiler und effizienter zu gestalten.
Parallel beginne ich mit dem Aufbau des Grundgerüsts für das Navi und überlege welche Methoden und Abläufe implementiert werden müssen. Ich beginne mit der Implementierung des Programmrahmens.

Woche 7, 12. - 16. November

Und täglich grüßt das Murmeltier… Wieder einmal kann die obligatorische Erstprüfung des alten Programmablaufs nicht erfolgreich beendet werden. Es wird mittlerweile zum gewohnten Muster, Arne und Denis testen die Koppler, die Sensoren, die Motoren, den Akku. Nach erneuten Anpassungen an Variablen und Schwellwerten kann wieder eine fehlerfreie Fahrt ablaufen, jedoch ist diese sehr ruckelig.
Für das Navi werden inzwischen die ersten Methoden implementiert. Eine der umfangreichsten Methoden ist die Erstellung der Kostenmatrix. Diese Methode wird von mir als erstes implementiert. Dazu kommt eine kleinere Methode, die das nächstgelegene Fahrtziel berechnet. Detaillierte Beschreibungen sind im Anhang zu entnehmen.

Woche 8, 19. - 23. November

Nachdem wir zum gefühlten tausendsten Mal Probleme mit dem Programmablauf haben, entscheiden wir uns für einen kleinen Code-Rework, um etwaige Problematiken zu beheben oder optimieren. Arne und Denis konsultieren den Professor und erarbeiten mithilfe kleiner Anreize einen zuverlässigeren Bewegungsablauf. Von nun an ist die Fahrt zuverlässiger, ruckelfreier und gleitender. Die einzig verbleibende Dauer-Baustelle ist die Schranke des Schließ-Mechanismus.
Auch die Entwicklung des Navis schreitet voran. Neben der Kostenmatrix berechnet das Navi inzwischen schon einen optimalen Weg für Hin- und Rückfahrt. Nur bei der Kommunikation nach außen stoße ich noch auf kleinere Schwierigkeiten.

Woche 9, 26. - 30. November

Der Bau und die Entwicklung der physischen Komponenten neigen sich dem Ende zu. Nur noch kleinere Verfeinerungen werden vorgenommen und kleinere Codepassagen werden optimiert. Sensoren, Motoren und die Schranke werden nochmals getestet und auf Zuverlässigkeit überprüft.
Auch in der Entwicklung des Navi ist das Ziel im wahrsten Sinne des Wortes in Sicht. Nachdem bereits ein Fahrplan nach Wegpunkten vorhanden war, wird dieser nun in relative (Himmelsrichtungen) und absolute (Links-Rechts-Befehle) Fahrpläne konvertiert. Das Ergebnis resultiert in einem Array, dass an den physischen Programmteil übergeben werden kann. Zum Abschluss bespricht unser Team noch die Vorbereitungen, die für den Zusammenschluss der Codefragmente benötigt werden. Wir arbeiten uns in beide Codes ein, um zu verstehen, wie wir den Zusammenschluss am besten angehen können.

Woche 10, 03. - 07. Dezember

Nach wochenlanger Arbeit und ständigem Testen beider Code-Module, die inzwischen separat einwandfrei funktionieren, beginnt nun die Stunde der Wahrheit: der Zusammenschluss der beiden Code-Teile. Bevor wir nun aber beide Teile fusionieren lassen können, müssen noch Kleinigkeiten zwecks Kompatibilität angepasst werden, wie zum Beispiel Variablennamen oder Array-Werte. Die Funktionen der einzelnen Methoden und ihre Zusammenwirkung müssen klar verständlich und nachvollziehbar sein. Nachdem alle Vorbereitungen nun getroffen worden sind, wird das Navi in die physische Komponente mit eingegliedert. Wir starten einen ersten Testlauf mit der Standardkarte auf der rechten Fahrbahnseite. Leider misslingt der Versuch. Allerdings konnten wir dadurch noch einige kleinere Kompatibilitätsprobleme aufdecken, wie z.B. die Rückgabe des Bot nach der Ballaufnahme, sowie der Auswurf des Balles im Zielbereich. Nachdem diese Probleme behoben sind, starten wir einen erneuten Versuch. Erfolg! Bis auf kleineren Problemen an der Schranke absolviert der Bot den Kurs erfolgreich. Sowohl die physische als auch die logistische Komponente arbeiten zuverlässig. Nun wollen wir die Aufgabe auch auf der linken Seite des Kurses testen. Schon auf der ersten Kreuzung Richtung Fahrgast biegt der Bot falsch ab, ratlose Gesichter. Nachdem das Navi in einer weiteren Trockenübung auf die soeben gescheiterte Fahrt überprüft wurde, stellen wir fest, dass dieses nicht die Ursache ist.

Wir durchforsten die physischen Code-Fragmente und entdecken eine alte Funktion, die noch aus der separaten Testphase stammt und nun, nach dem Zusammenführen des Codes, zu Schwierigkeiten und Verwirrung führt. Wir entfernen die Funktion und der Roboter kann die Aufgabe souverän bewältigen.

Woche 11, 17. - 21. Dezember

Die Aufgabe ist eigentlich schon gelöst. Aber wir investieren nochmal eine Stunde, um verschiedenste Strecken zu testen. Dies s
Nutzen wir als Ausdauer- und Zuverlässigkeitstest. Leichte Probleme macht weiterhin nur die Schranke. Ansonsten löst der Roboter alle Karten und wir sind bereit für den Wettkampf.

Woche 12, 07. - 11. Januar

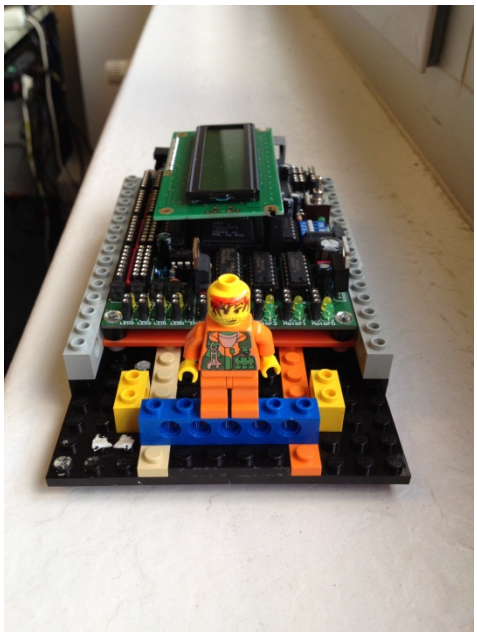
Sozusagen das Warm-up. Es endet aber noch in einem Haufen Arbeit… Zuerst lösen wir endlich unser Schranken-Problem (der Akku hatte oft nicht genug Saft für den Antrieb des Servo-Motors). Außerdem haben wir einen zwölfseitigen Streckenkatalog mit Protospielfeldern. Von denen wird jede einzelne beidseitig auf Zuverlässigkeit und Effizienz getestet. Die Strecken 1-6 sowie 9-12 sind ein Kinderspiel für unseren Bot. Aber die Strecken 7 und 8 tricksen unseren Bot aus, das Navi hat versagt. Das zwingt mich erneut sofort mit der Fehlersuche zu beginnen. Zuerst Strecke 7. Wo ist der Fehler? Auf dieser Karte ist das Spielfeld oben in der Mitte geöffnet und beide Parteien haben je einen Ball auf ihrer eigenen Seite, sowie den gemeinsamen Fahrgast oben in der Mitte. Der Bot schnappt sich zuerst den eigenen Fahrgast. Alles völlig in Ordnung… Doch dann holt er den gegnerischen, statt den gemeinsamen Fahrgast, obwohl dieser eindeutig näher dran ist. Die Frage ist: wieso? Da sich keiner so recht erklären kann woher der Fehler resultiert, bleibt mir nur eine Wahl – ein komplettes Debuggen der ganzen Rechnung. Nun deckt sich aber ein völlig neues Problem auf. Der Befehlskette des Roboters ist falsch und nicht nachvollziehbar. Rechts startend fährt er gerade und rechts… Gegen eine Wand!? Was ist passiert? Kurz bevor die Fehlersuche in Verzweiflung gipfelt fällt es mir der Fehler auf. Ausgetrickst von der eigenen Logik. Natürlich ist gerade und rechts richtig! Warum? Der Bot schaut zwar anfangs nach Norden. Allerdings habe ich ihn so programmiert, dass er sich bei Versperren des direkten Fahrtweges direkt zu Beginn nach Westen bzw Osten dreht, wie es hier der Fall war – also kein Fehler! Aber es bleibt das Problem des ignorierten Fahrgastes. Ein weiterer Debug-Durchlauf zeigt, dass sowohl die Kostenmatrix, als auch die Umsetzung der relativen und absoluten Fahrtwege funktionieren fehlerfrei. Auch die Folge der Knotenpunkte ist korrekt. Somit kann der Fehler nur noch in der Priorisierung der Fahrgäste. (Für technische Details siehe Kapitel ´Das Navi`) Der Fehler ist dank Debugging schnell gefunden. Kleiner Fehler, große Wirkung: Bei einer if-Bedingung fehlt eine Klammer und das führt zu einem fehlerhaften Ablauf, der auf den herkömmlichen Karten nie zu Tage getreten wäre. Auf Karte 8 gibt es ein anderes Problem, was zum kompletten Systemabsturz des Navis führt. Es gibt Fahrgäste, die nicht erreicht werden können, die aber bei der Berechnung berücksichtigt werden, da sie Nachbarn von erreichbaren Fahrgästen sind. Zwischen diesen besteht aber auf der realen Karte keinerlei Verbindung. Das heißt der Bot möchte zu einem Knoten, von dem er weiß, dass er da nicht

hinkommt und der Bot schaltet auf stur. Und sture Bots arbeiten nicht gerne, Systemabsturz. Aber auch dieser Fehler ist schnell behoben. Hierbei kommt uns zu Hilfe, dass Fahrgäste immer nur am Rand des Spielfeldes warten. Zwei Karten, viel Ärger. Und doch ist wieder eine Herausforderung gemeistert, alle Karten können problemlos abgefahren werden (auch innerhalb des zweiminütigen Zeitlimits). Die letzte Hürde vor dem Wettkampf ist genommen.

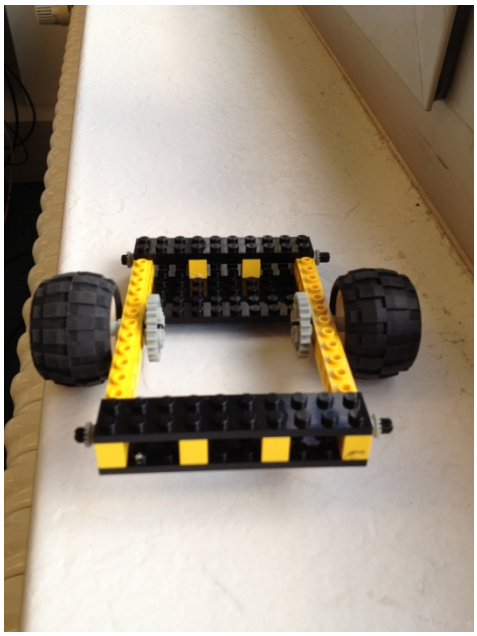
Der Aufbau

Der Roboter

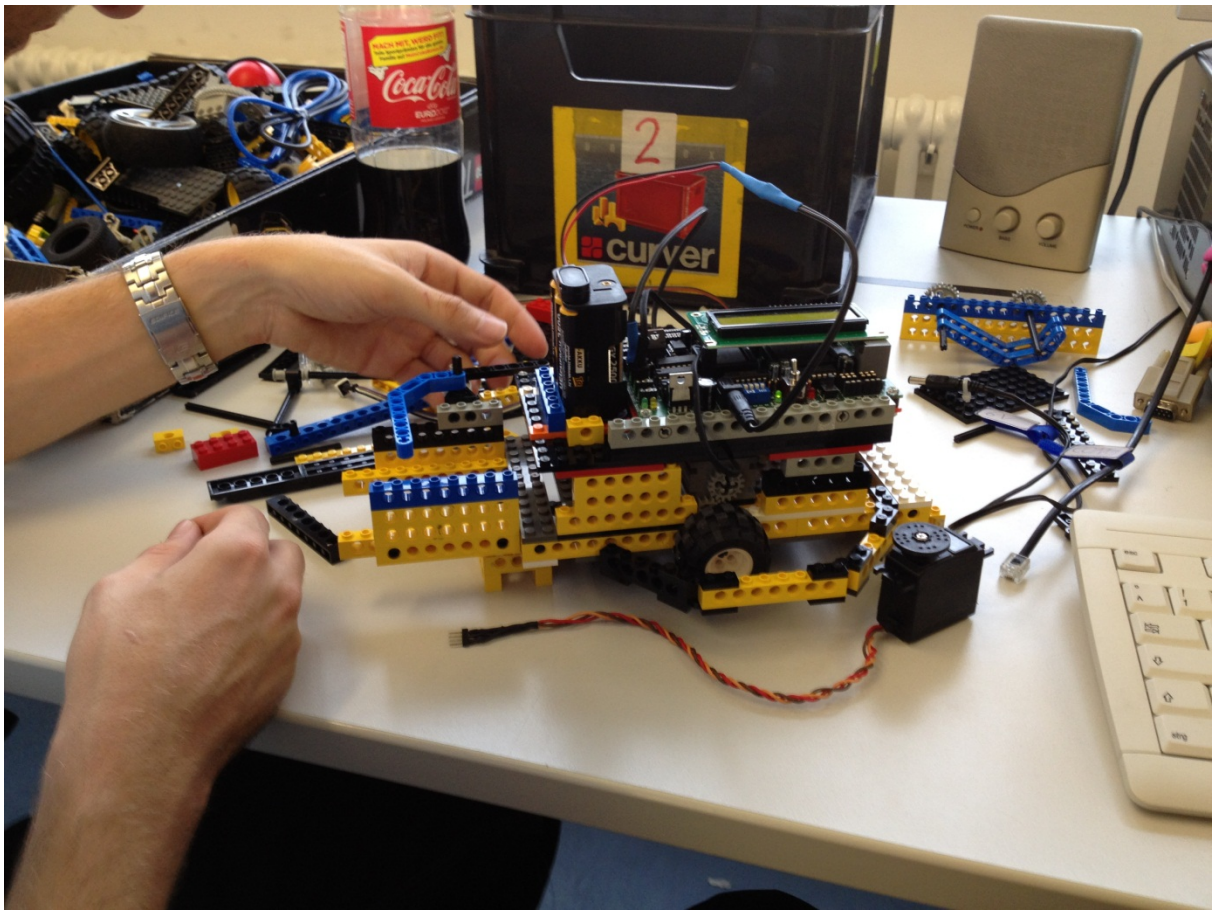
Die Idee war es den Roboter möglichst nicht größer als das Aksen-Board zu machen, um ihn so wendig wie möglich zu gestalten. Dazu wurde zunächst das Aksen-Board in einen Rahmen aus Lego eingepasst, um es fest mit dem Roboter zu verbinden.



Eine Lagermöglichkeit für den Akku war ebenfalls vorgesehen, um den Akku einerseits während der Fahrt sicher zu lagern, ihn aber andererseits schnell wechseln zu können. Nun ging es an die Konstruktion des Fahrwerkes. Dieses sollte prinzipiell nicht breiter oder länger als der Oberbau mit dem Aksen-Board sein. Diese Idee musste jedoch verworfen werden, da eine Montage der Räder im Inneren des Roboters eine Steuerung unmöglich gemacht hätte. Daher wurden die Räder nach außen gelegt, was den Unterbau leider erheblich verbreiterte und dazu führte, dass die Räder anfälliger für Hindernisse wurden, welche sie blockieren konnten.

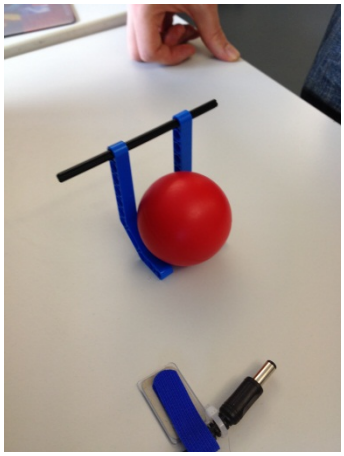


Dieses Problem wollten wir später angehen. Zunächst sollte der Roboter fahrtüchtig gemacht werden. Als nächstes wurde die Positionierung der Motoren in Angriff genommen. Anfangs wollten wir eine hohe Übersetzung in die Getriebe einbauen, um mehr Kraft zu erhalten, jedoch verloren wir dadurch viel Geschwindigkeit, weshalb diese Idee ebenfalls verworfen wurde. Das Plus an Kraft, welches wir dadurch erhalten hätten, wurde nicht erbracht, weil wir sehr geringe Verluste durch Reibung eingeplant hatten. Wir entschlossen uns, die Motoren mehr oder weniger direkt an die Achsen der Räder zu montieren, wodurch unser Roboter sich sehr schnell bewegte.

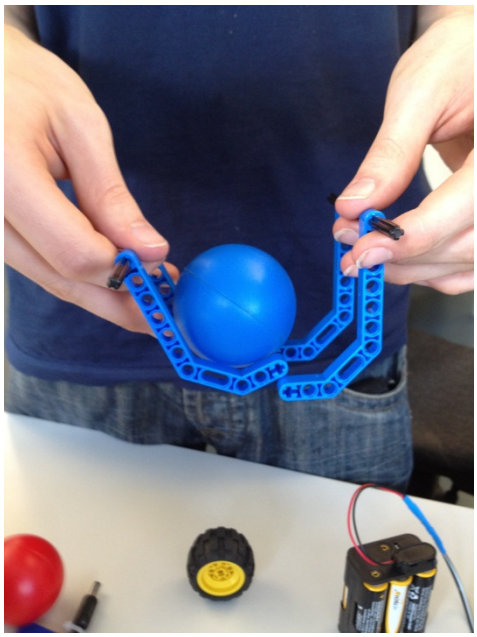


Auf diesem Bild kann man die Motoren sehr gut erkennen. Als Getriebe wählten wir eine 2/3 Übersetzung.
Dies brachte uns ausreichend Kraft, bei einem hohen Maß an Geschwindigkeit und Steuerbarkeit.

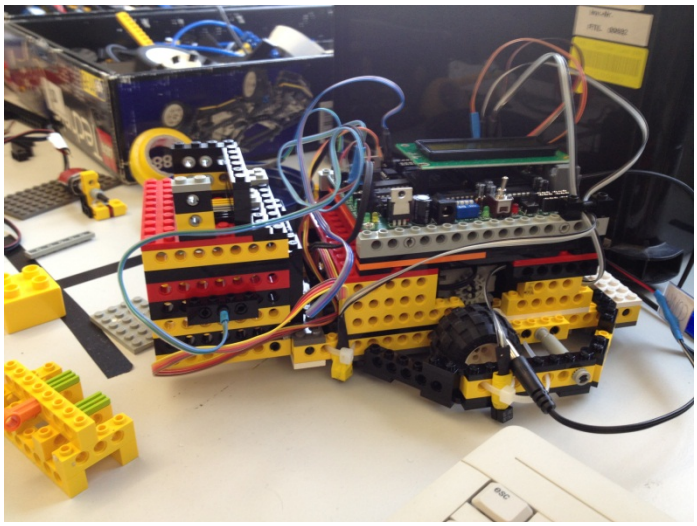
Nun ging es darum wie wir den Fahrgast einsammeln. Die erste Idee war es das ganze nach dem Prinzip des Gabelstaplers zu gestalten und die Fahrgäste mit einer Hubvorrichtung von unten anzuheben.



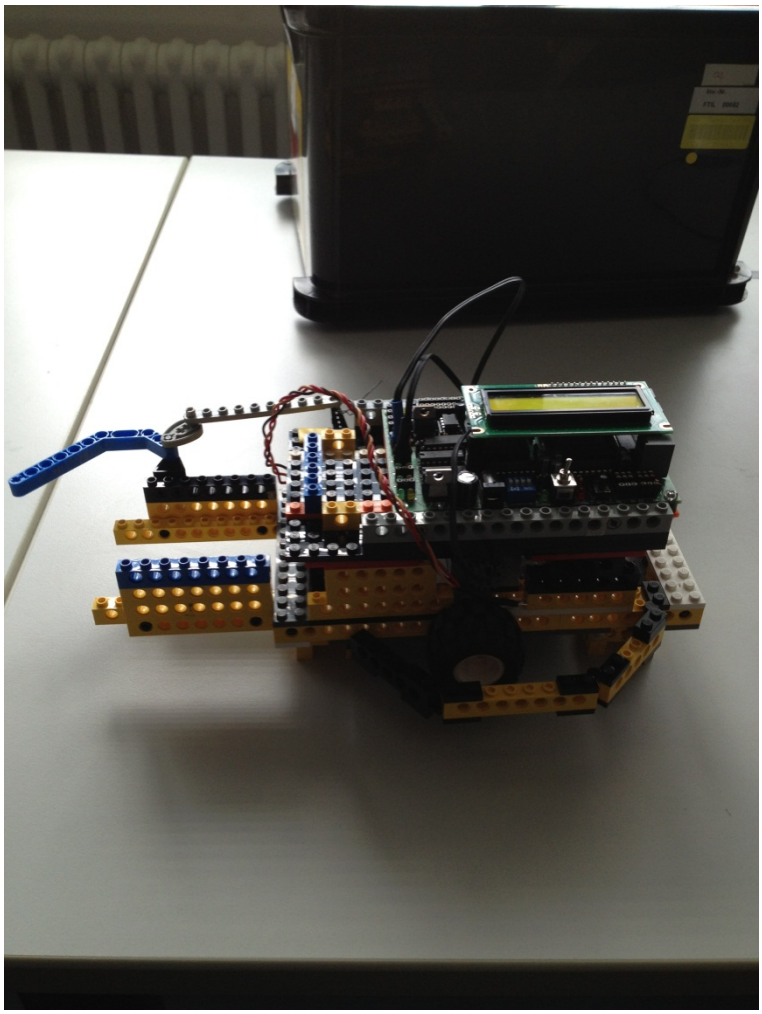
Dies scheiterte daran, dass wir erstens nicht bei jedem Anfahren perfekt gerade auf den Fahrgast zusteuern konnten und unsere Hubvorrichtung den Fahrgast mit seinem Wartesitz einfach weggrissen hätte, zweitens wir nicht genug Hubhöhe erreicht haben, um den Fahrgast anheben zu können, und drittens der Fahrgast während der Fahrt heruntergefallen wäre. Also musste eine neue Idee gefunden werden. Wir behielten den Grundgedanken des Greifens bei und versuchten eine Art Greifer zu konstruieren, der den Fahrgast von oben greift und im Ziel fallen lässt.



Aufgrund des komplexen Mechanismus dahinter, um die Greifer synchron zu steuern, beschlossen wir auch diese Idee zu verwerfen.
Da uns zunächst die Ideen ausgegangen waren, entschieden wir uns, uns dem Schutz der Räder zu widmen. Wie bereits anfangs erwähnt lagen diese außerhalb des Roboters und es bestand die Gefahr, dass sie an Hinternissen hängen bleiben und blockieren. Wir konstruierten eine Art Rahmen um die Räder, welcher gleichzeitig als Halterung für ein paar Optokoppler dienen sollte. Leider wurde durch diesen Rahmen der gesamte Roboter fast doppelt so breit wie anfangs geplant. Da die Breite des Roboter beim eigentlichen Wettkampf weniger von Belangen ist, erschien uns diese Idee als nicht mehr essentiell wichtig.

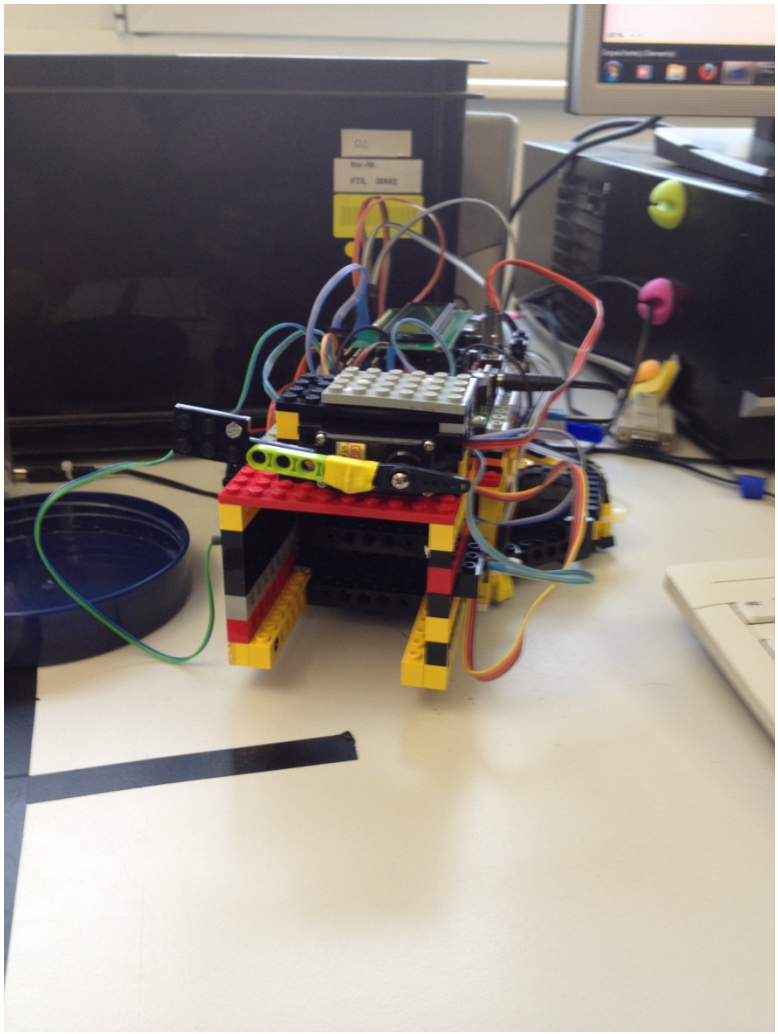


Gut zu erkennen ist der Schutzrahmen um die Räder mit den am Rahmen montierten Optokopplern. Welche Funktion diese haben wird später erläutert.
Inzwischen hatten wir eine neue Idee für den Mechanismus zum Einsammeln und Transportieren des Fahrgastes. Wir entwarfen einen „Käfig“, mit dem wir den Fahrgast umschließen wollten und ihn dann von vorne zu verschließen. Anfangs war es nur eine flache Wand, die rechts und links vom Fahrgastes sitzen sollte und den Fahrgast samt Sitz ins Ziel bringen sollte.



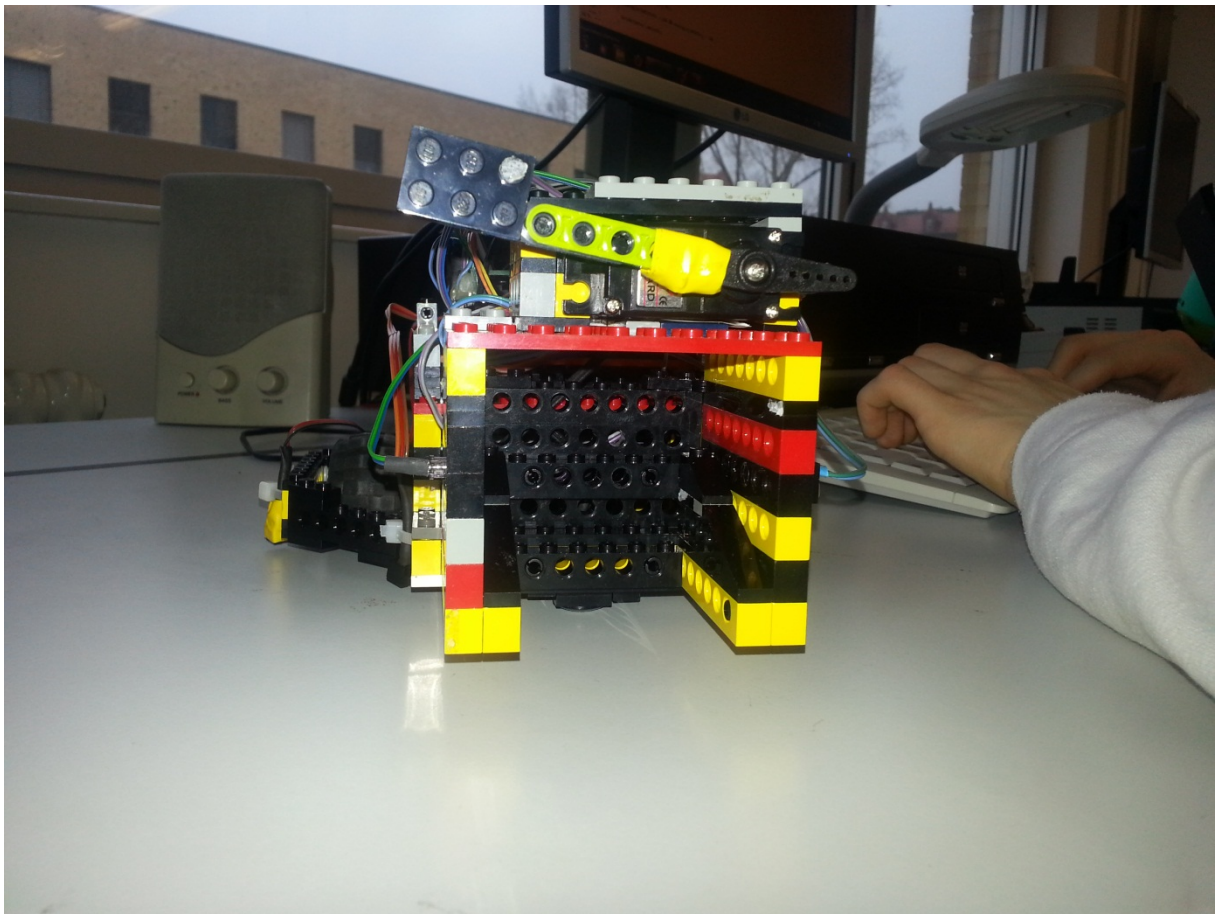
Der Grundgedanke dieser Kontruktion war gut, aber nicht ausgereift. Der Mechanimus mit dem der Transportkäfig verschlossen werden sollte war nicht zuverlässig genug für den Einsatz in der Praxis. Da die Verbindung zwischen Servomotor und Legoteil über einen Draht realisiert war und sehr instabil war, verhackte sich das Ganze häufig und funktionierte einfach nicht. Also musste eine andere Möglichkeit gefunden werden, um den Käfig zu schließen.
Eine Schranke wurde entworfen, welche über einen Motor, der seitlich vom Fahrgastraum montiert war, angetrieben wurde. Doch auch hier gab es diverse Probleme. Das Hauptproblem war allerdings, dass ein Schließen und Öffnen sich nur über ein zeitlich begrenztes Ein- und Ausschalten des Motors umsetzen ließ. Durch diese Sleeps ergaben sich Probleme für den restlichen Programmablauf. Des Weiteren stellte sich heraus, dass der Motor beim Rückwärtsdrehen sehr schwach war und die Schranke mit einem gebrauchten Akku nicht öffnen konnte. Daher wurde auch dies wieder verworfen. Das Prinzip der Schranke war sehr gut, nur hatten wir noch keine optimale Möglichkeit der Umsetzung gefunden. Dann hatten wir die Idee die Schranke von oben schließen zu lassen und diese über einen Servomotor ansteuern zu lassen. Dieser ist weniger störanfällig und besitzt zuverlässige Bewegungsamplituden. Aus unserem ersten Versuch mit dem Servomotor hatten wir gelernt, dass unsere Schranke direkt mit dem Servo verbunden werden musste.

Wir erweiterten unseren Transportkäfig zu einer „Höhle“. Durch diese Erweiterung hatten wir oberhalb des Fahrgastraumes Platz, den Servo zu montieren. Das Dach über dem Fahrgastraum machte unsere Lichtschranke zudem nahezu unempfindlich gegenüber Fremdeinstrahlung. Auch wenn diese aufgrund unseres Algorithmusses eigentlich nicht notwenig war.



Um unabhängig von äußeren Einflüssen, wie unterschiedlicher Beleuchtungsstärke und sich dadurch wechselnden Schwellwerten, zu sein, übertrugen wir einen Algorithmus aus einem anderen Projekt in unseres. Dieser funktioniert wie folgt: Die IR-LED wird zeitlich begrenzt eingeschaltet, der vom IR-Empfänger gemessene Wert wird in eine variabel (a) gespeichert. Nun wird die IR-LED zeitlich begrenzt abgeschaltet, am IR-Empfänger wird ein anderer Wert gemessen. Dieser wird in eine neue Variabel (b) gespeichert. Nun wird a von b subtrahiert und es ergibt sich c. Sollte nun die Lichtschranke unterbrochen sein ergibt sich für c ein Wert x. Sollte die Lichtschranke frei sein, ergibt sich immer ein Wert größer x. Somit ist es auch völlig unerheblich, ob der Empfänger nur Licht von der IR-LED oder auch noch von anderen Lichtquellen empfängt. Durch unsere Kontruktion ergibt sich ein Wert von nahezu Null, sollte die Lichtschrank unterbrochen sein, und ein Wert von weit über 100, sollte die Lichtschranke

frei sein. Somit sind die Schwellenwerte sehr weit auseinander und wir können immer sicher sein, dass unser Fahrgast erkannt wird.
Nachdem nun die Hürde des Fahrgastaufnehmen und -transportieren genommen war, zeigt sich ein neues Problem. Wie bekommen wir den Fahrgast wieder heraus? Anfangs versuchten wir das ganze durch Rückwärtsfahren zu lösen und hofften, dass unser Roboter schnell genug ist und die Fliehkraft ihr übriges tut. Jedoch klappte dies nicht zuverlässig genug.
Also musste erneut eine Änderung am Roboter vorgenommen werden. Eine Lösung wurde schnell gefunden. Die Auflagen in der Innenseite des Fahrgastraums wurden mit Schrägen versehen, sodass der Ball nach Öffnen der Schranke mithilfe der Schwerkraft von ganz alleine den Roboter verlässt. Dies erwies sich als höchst effektiv.

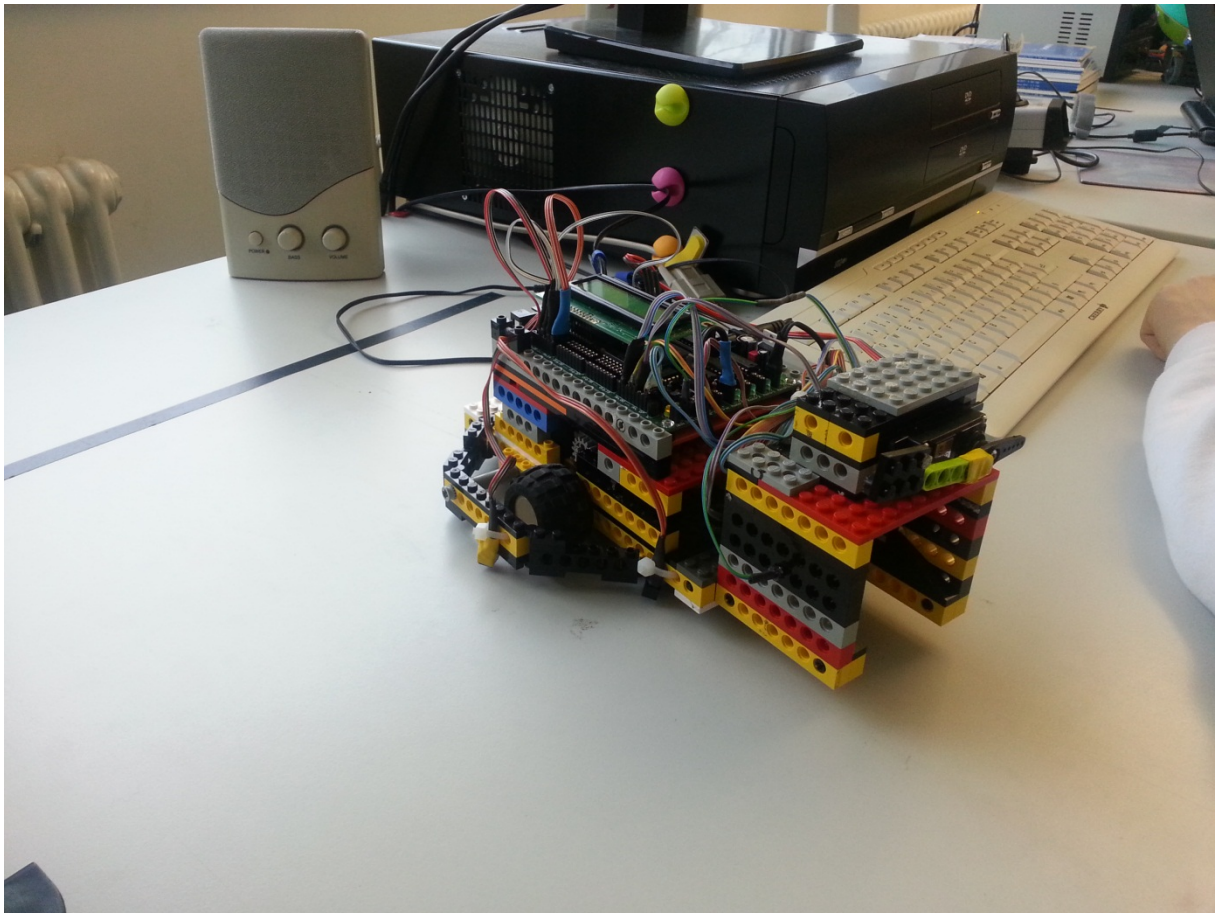


Während der Konstruktionsphase stellten wir fest, dass sich die Lagerung des Akkus oben auf dem Roboter als unsicher herausstellte. Zum einen wurde der Schwerpunkt nach sehr weit oben verlagert, zum anderen bestand die Gefahr, dass der Akku herunter fällt. Also beschlossen wir ihn in das Innere des hinteren Teils des Roboters zu legen, wodurch wir den Schwerpunkt senkten und das Übergewicht an der Vorderseite durch den Fahrgastraum und den Servomotor ausgleichen konnten.

Positionierung der Optokoppler

Um ein sauberes Linienfolgen gewährleisten zu können, verbauten wir 2 Optokoppler mittig zur Längsachse des Roboters. Somit erreichen wir ein Linienfolgen mit sehr geringer Abweichung. Anfangs hatten wir dadurch Probleme, dass unser Roboter sehr „unzufrieden“ war und etwas zu viel korrigierte. Doch durch Austauschen eines Motors und Anpassung der Drehgeschwindigkeit beim Gradeausfahren ist das Korrigieren kaum noch zu erkennen. Ein weiterer Vorteil dieser Konstruktion ist, dass wir bei Drehungen nahezu perfekt mittig auf der Linie zum Stehen kommen.

Zur Erkennung von Kreuzungen haben wir 2 Koppler-Paare verbaut. Eines vorne, direkt hinter dem Fahrgastraum, ein zweites direkt an den Achsen. Warum 2 Paare? Um zu verhindern, dass der Roboter beim Drehen auf einer Kreuzung dieselbe Kreuzung doppelt erkennt. Die Kreuzungserkennung läuft bei uns folgendermaßen: Wir fahren geradeaus. Die vorderen Kreuzungskoppler erkennen eine Kreuzung. Wenn danach die hinteren Kreuzungskoppler ebenfalls eine Kreuzung erkennen, wird die zu der Kreuzung passende Aktion durchgeführt, also abbiegen oder geradeaus fahren. Ohne dass die vorderen Koppler etwas erkannt haben, werden die hinteren nicht abgefragt. Wenn wir uns auf einer Kreuzung befinden und dort mehrere 90° Drehungen machen müssen, werden selbstverständlich nur die hinteren Koppler benutzt. Dadurch haben wir eine sichere Kreuzungserkennung mit nahezu keiner Fehlererkennung und daraus resultierenden Fehlern beim Abarbeiten des Fahrplans.



Hardware und Hardwareabstraktion

Methoden der Implementierung

void korrigieren(short m1, short m2)
In dieser Methode wird die Leistung der jeweiligen Motoren angepasst, um eine geregelte Steuerung des Roboters zu gewährleisten. Die beiden short-Variablen stehen hierbei für die jeweiligen Motoren.

void korrigierRichtung(short m1, short m2, short octL, short octR)
Diese Methode und ihr Nachfolger werden stets da aufgerufen, wo der Bot eine Zeit lang gerade fahren muss. Die short-Variablen m1 und m2 bezeichnen wiederum die beiden Motoren und die beiden short-Variablen octL und octR steuern die beiden Optokoppler in der Mitte des Roboters. Mithilfe dieser Methode kann bestimmt werden in welche Richtung bei Abweichungen korrigiert werden muss. Dabei werden 3 Fälle unterschieden:

- interpretiert octR weiß und octL schwarz korrigiert der Bot nach links
- interpretiert octR schwarz und octL weiß korrigiert der Bot nach rechts
- interpretieren sowohl octR, als auch octL weiß, korrigiert der Bot in die zuletzt korrigierte Richtung

void rueckwaerts(short m1, short m2)
Diese einfache Methode ermöglicht es dem Roboter rückwärts zu fahren. Erneut bezeichnen die Variablen m1 und m2 die jeweiligen Motoren. Um das Rückwärtsfahren zu ermöglichen wird lediglich die Richtung der beiden Motoren auf 0 gesetzt (1 wäre dementsprechend vorwärts).

void abbiegen(short m1, short m2, short m1P, short m2P, short cou)
Den Rahmen dieser Funktion bildet eine for-Schleife, die die Anzahl der Drehungen umsetzt. Hier wird der Motor, der sich auf der Seite des Richtungswechsels befindet, rückwärts gedreht. In dieser Schleife befinden sich weiterhin zwei while-Schleifen. Die erste while-Schleife wird so lange ausgeführt, wie beide äußeren Optokoppler schwarz zeigen. Sobald beide äußeren Optokoppler weiß zeigen betritt Programm die nächste while-Schleife. Diese wird wiederum beendet, wenn beide Optokoppler wieder schwarz zeigen. Das Zusammenwirken beider while-Schleifen gewährleistet, dass jede Drehung nur um 90 Grad ausgeführt wird. Neben den bekannten Motoren-Variablen m1 und m2 geben die beiden short m1P und m2P die Geschwindigkeit an, mit denen sich die jeweiligen Motoren bewegen. Die Variable cou gibt die Anzahl der benötigten Drehungen an.

void schliessen(short m1, short m2)
Diese Methode beginnt mit einer Differenzrechnung, damit die nötigen Sensoren unabhängig von den gegebenen Lichtverhältnissen zuverlässig arbeiten können. Sobald nun die Lichtschranke in der Passagierkabine unterbrochen wird, schließt sich die Schranke durch

unseren Servomotor. Nachdem die Schranke geschlossen wurde fährt der Bot so lange rückwärts, bis einer der beiden äußeren Optokoppler schwarz erkennen. Nun richtet sich der Roboter so aus, dass er um 180 Grad versetzt wieder auf der Kreuzung des Methoden-Aufrufs steht. Dies vereinfacht die Umsetzung der Navigation erheblich.

void oeffnen(short m1, short m2)
Beim Aufruf dieser Methode erhält der Roboter den Befehl zum Auswurf seines Passagieres. Ähnlich wie bei der schliessen-Methode richtet sich der Roboter erneut aus. Diesmal allerdings immer in Startrichtung. Dies kann abhängig von der Karte ein beliebiges Vielfaches von 90 Grad sein.

void main()
Hier beginnt der Ablauf des Programms. Die eigentliche Fahrt beginnt nachdem die Berechnung beendet ist. Der Roboter wartet auf das Aufleuchten der Signallampe, damit er mit seinem berechneten Auftrag beginnen kann. Hier befindet sich auch die Funktion, mit der der Bot sein Verhalten an Kreuzungen bestimmt. Dabei helfen ihm die beiden vorderen Optokoppler. Für den Ablauf des Bots wird einfach das berechnete guide-Array durchlaufen. Dafür wird, wenn die Kontrolle der Optokoppler keine Fehler aufwirft, eine while-Schleife solange ausgeführt, bis die hinteren Optokoppler die schwarze Linie erkennen. Daraufhin gelangt das Programm in eine switch-case-Auswahl, die je nach momentaner Position den kommenden Befehl aus dem guide-Array ausführt. Nach jeder Drehung wird die Variable hc auf den Wert der Blickrichtung des Roboters gesetzt: Dabei entspricht 0 Nord, 1 Ost, 2 Süden und 3 Westen. Nach jedem Befehl wird der Fahrcounter, der bestimmt an welcher Stelle des guide-Arrays sich der Bot zur Zeit befindet, um eins erhöht und die Kreuzungsvariable auf 0 gesetzt, damit die Schleife verlassen werden kann. Sind alle Fahrbefehle aus guide abgearbeitet wartet der Roboter in einer while-Schleife auf seine Ausschaltung.

Anmerkung zu fehlgeschlagenen Implementierungen

Das Öffnen und Schließen, sowie das Drehen nach der Ballaufnahme bzw. –abgabe, die nun in den Methoden Öffnen und Schließen fusioniert sind, wurden vorher in einem parallelen Prozess in Abhängigkeit der für den Ball zuständigen Sensoren realisiert. Da sich der parallele Prozess jedoch als sehr fehleranfällig und unzuverlässig erwies, wurde er entfernt und die Methoden angepasst.

Die Navigations-Komponente

Anmerkung zur Verwendung der globalen Variablen + Arrays

Für den fehlerfreien und effizienteren Ablauf der Navigations-Komponente sind mehrere Arrays notwendig. Da alle Arrays den wiederholten Zugriff in verschieden Methoden ermöglichen müssen, werden alle Arrays global verwaltet. Folgende Arrays wurden von uns deklariert:

- arena (später umbenannt in _fa): Die Karte (x – gesperrte Kreuzung, . – freie Kreuzung, F – Fahrgast)
- rating: Die Kostenmatrix
- schedule/wayback: Die Knotenpunkte für Hin- bzw. Rückweg
- guideSchedule/guideBack: relativer Streckenplan für Hin- bzw. Rückweg (links-rechts-Befehle, l links, r rechts, g geradeaus)
- guide: Gesamtroute, stückweises Zusammensetzen von guideSchedule und guideBack
- alignment: Abfolge der Ausrichtungen des Bots währende der Fahrt (n Nord, s Süd, w West, e Ost)

Des Weiteren werden einige Variablen global gesetzt bzw. gespeichert. Der Startknoten (startNode), sowie die Ausrichtung zu Beginn (startAlignment). Der Startknoten wird später über die Schalter ausgelesen (64 oder 68) und die Ausrichtung zu Beginn ist immer Norden. Auch die Anzahl der Fahrgäste (anzF) wird mit 0 vorinitialisiert und später berechnet. Zu guter Letzt ist noch ein globaler Index (guideIndex) von Nöten, da während des Zusammensetzens des guide-Arrays ein ständig fortlaufender Index, der nicht zurückgesetzt wird, benötigt wird.

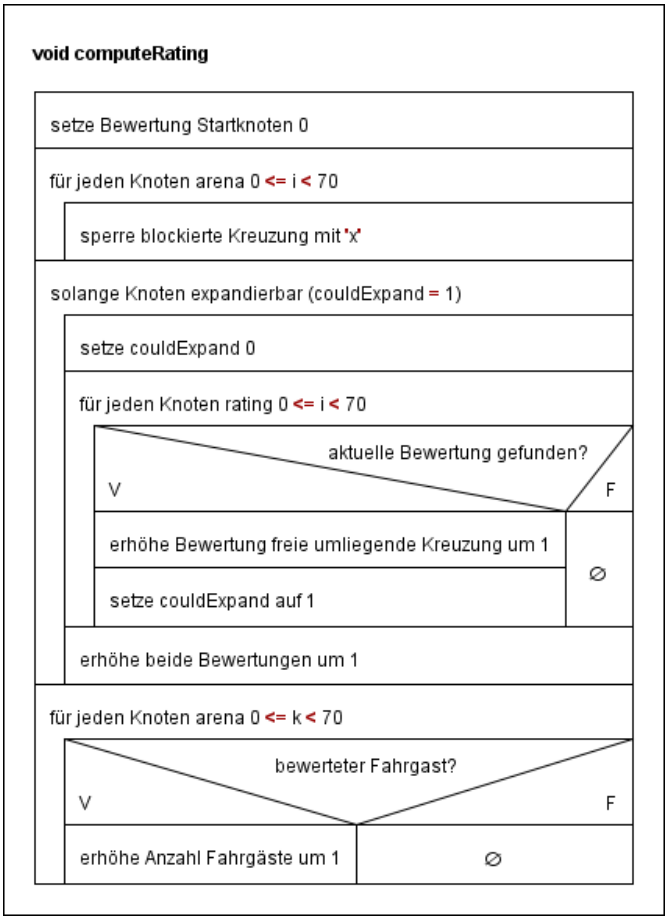
Methoden der Implementierung

void computeRating()

Neben den Index-Variablen i, j und k sind in dieser Methode vor allem couldExpand und die Rating-chars von zentraler Bedeutung. Zu Beginn der Berechnung wird der Startknoten mit 0 bewertet. Bevor die Breitensuche abgearbeitet wird, werden in einer ersten for-Schleife einige Sonderfälle abgefangen. Diese Sonderfälle verhindern, dass blockierte Fahrgäste, die durch direkte Nachbarschaft zu erreichbaren Objekten, scheinbar angefahren werden können. Bevor diese Sonderfälle abgefangen wurden, führte dies zu einem Programmabsturz, da der Roboter Knoten erreichen wollte, von denen er wusste, dass er dort nicht hinkommt. Da die Fahrgäste immer nur von einer Seite angefahren werden können, werden diese Sonderfälle durch einfache Modulo-Rechnung unterschieden. Unerreichbare Fahrgäste, sowie gesperrte Kreuzungen – der vierte Sonderfall – werden aus der Kostenmatrix gestrichen und auf x gesetzt.

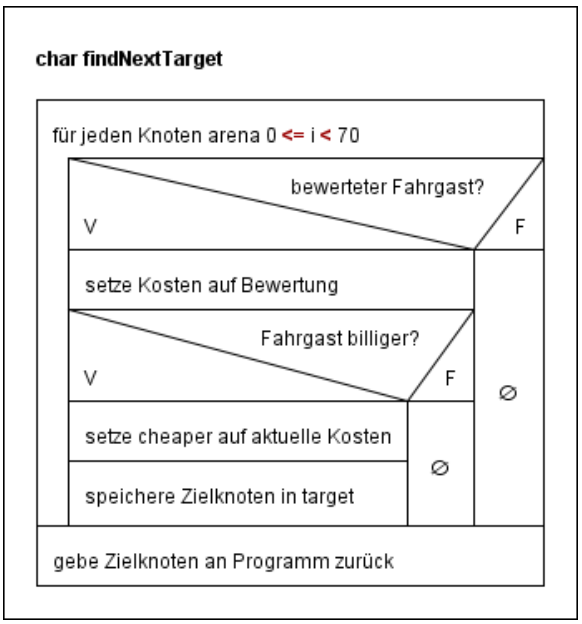
Nun beginnt die eigentliche Arbeit. Die Verarbeitung der Breitensuche erfolgt innerhalb von zwei Schleifen. Den äußeren Rahmen bildet eine while-Schleife. Hierbei ist die Variable

couldExpand von entscheidender Bedeutung. Diese hat keine andere Aufgabe, als festzustellen, ob die Breitensuche Knoten expandieren kann. Global ist couldExpand zu Beginn auf 1 gesetzt, damit die Schleife betreten werden kann. Zu Beginn der Schleife wird die Variable auf 0 gesetzt, um ein erneutes Betreten nur zu gewährleisten, wenn im momentanen Ablauf ein Knoten expandiert wird. Die for-Schleife im inneren Rahmen der Berechnung enthält die eigentliche Breitensuche. Für die primäre Bedingung wird die Kostenmatrix so lange durchlaufen, bis die aktuelle Bewertung – zu Beginn die 0 (der Startknoten) – gefunden wird. Dabei wird jeder Knoten befragt, ob er der momentanen Bewertung entspricht. Wird beim Abgleich ein Treffer erzielt, geht die Schleife eine Stufe tiefer, um die umliegenden Knoten zu bewerten. Dabei wird erneut in 4 Sonderfälle unterschieden, damit die Berechnung das Feld nicht verlässt. Ist ein Knoten mit der aktuellen Bewertung gefunden, und sind eine oder mehrere umliegende Knoten den Anforderungen entsprechend, werden diese um 1 erhöht (im Vergleich zur Bewertung des Vorgängers). Nur wenn einer der Knoten expandieren konnte, wird couldExpand wieder auf 1 gesetzt, damit die Breitensuche im nächsten Knoten weiterarbeiten kann. Ist eine Bewertung im kompletten Feld abgearbeitet werden beide Bewertungs-Variablen (oldRating und newRating) erhöht und die Suche beginnt mit den neuen Werten von vorne.
Nachdem die Kostenmatrix aufgestellt ist, wird in einer letzten Schleife der Methode die Anzahl der abzuarbeitenden Fahrgäste ermittelt. Dabei wird das Spielfeld einfach nach bewerteten Fahrgästen gescannt und die Anzahl bei jedem Treffer um eins erhöht.



char findNextTarget()

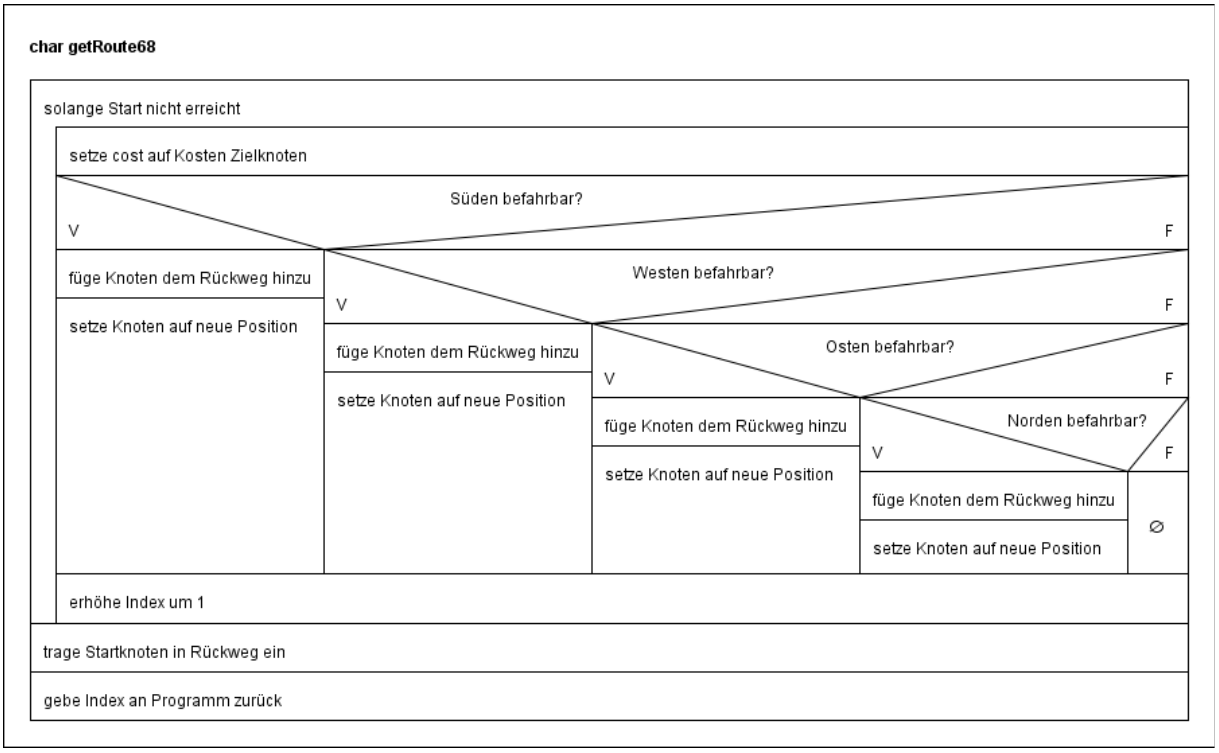
Diese Methode macht nichts weiter, als den am nächsten gelegenen Fahrgast zu berechnen. Dies gewährleistet es uns die ersten Fahrgäste schnellstmöglich einzusammeln. Die Methode beinhaltet neben der Index-Variable i die Variablen target, die den Zielknoten angibt, cost, die die Kosten des aktuellen Fahrgastes beinhaltet, und cheaper, die die Kosten des billigsten Knoten speichert. Cheaper ist mit dem Wert 80 vorinitialisiert, damit der Ersteintritt in die Bedingung später gewährleistet ist und ein Fahrgast nicht teurer als 70 Punkte sein kann. Das Spielfeld wird komplett durchlaufen und für jeden bewerteten Fahrgast werden die dazugehörigen Kosten gespeichert. In der if-Bedingung wird nun überprüft, ob der aktuell berechnete Fahrgast günstiger ist als der bisher billigste. Ist der aktuelle Fahrgast schneller zu erreichen, so wird sein Kostenpunkt in cheaper gespeichert und der Zielknoten auf den aktuellen Wert des Index gesetzt. Somit steht in target immer, der am schnellsten zu erreichende Zielknoten. Ist die Schleife komplett durchlaufen, wird target an das Programm zurückgegeben.



char getRoute68(char node) bzw. char getRoute64(char node)

Diese beiden Methoden generieren die Wegpunkte vom Ziel zum Start zurück. Sie unterscheiden sich lediglich in ihren Prioritäten. Das Ziel des Roboters ist es so schnell wie möglich zurück zur Homebase, die sich im Süden befindet, zu gelangen. Wichtig ist aber auch, in welche Richtung er ausweicht, wenn der südliche Weg versperrt ist. Da sich die Fahrgäste immer nur am Rand des Spielfeldes befinden, muss sich der Roboter auf der linken Seite immer nach Osten orientieren, falls der Süden nicht befahrbar ist, und auf der rechten Seite nach Westen. Die Berechnung durchläuft eine while-Schleife, die so lange arbeitet, bis die Berechnung am Startknoten angelangt ist, also solange die Bewertung des aktuellen Knoten nicht 0 ist. Zu Beginn der Schleife werden die Kosten von node – das ist der in der vorherigen Funktion berechnete Zielknoten – ausgelesen. Dann wird in der festgelegten Reihenfolge überprüft, welcher Knoten in einem Schritt erreicht werden kann (68: Süd-West-

Ost-Nord, 64: Süd-Ost-West-Nord). Dies geschieht in einer if-else-Unterscheidung. Ist der nächste Knoten gefunden, so wird der Knoten in das wayBack-Array eingetragen und node je nach abgefahrener Richtung um 1 bzw. 7 erhöht bzw. verringert. Danach wird der Index um eins erhöht und die Suche nach dem nächsten Knoten erfolgt gleichermaßen. Ist die Schleife komplett durchlaufen, wird am Ende des Arrays noch der Startknoten eingetragen, da dieser in der Schleife noch nicht berücksichtigt wurde. Zudem wird am Ende der Methode auch der Index an das Programm zurückgegeben. Das ist notwendig, damit die mit 71 Feldern (die Größe des Spielfeldes) vorimplementierten Arrays beim Auslesen später auch nur bis zum letzten Eintrag ausgelesen werden.



(getRoute64 kongruent, nur dass Osten vor Westen überprüft wird)

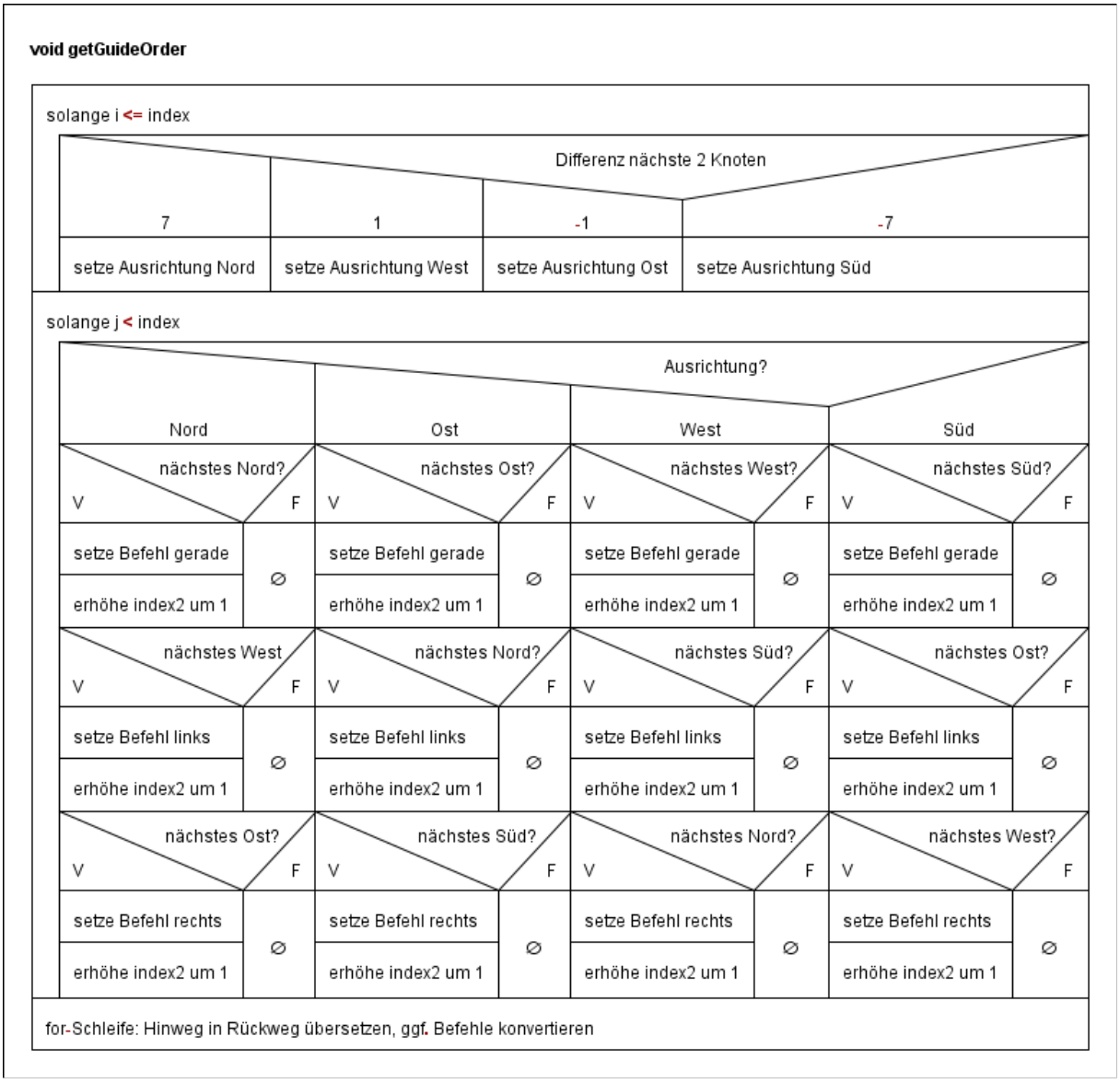
void fetchPassenger(char index)

Diese Methode besteht aus einer einzigen for-Schleife, deren Aufgabe es ist, aus dem Rückweg den Hinweg zu generieren. Dafür wird wayBack einfach rückwärts ausgelesen und in schedule eingetragen. Hierbei dient der aus der letzten Methode mitgelieferte Index als Absicherung, damit auch erst ab dem letzten Eintrag, das wayBack-Array ausgelesen wird, und nicht ab dem letzten Punkt.

void getGuideOrder(char index)

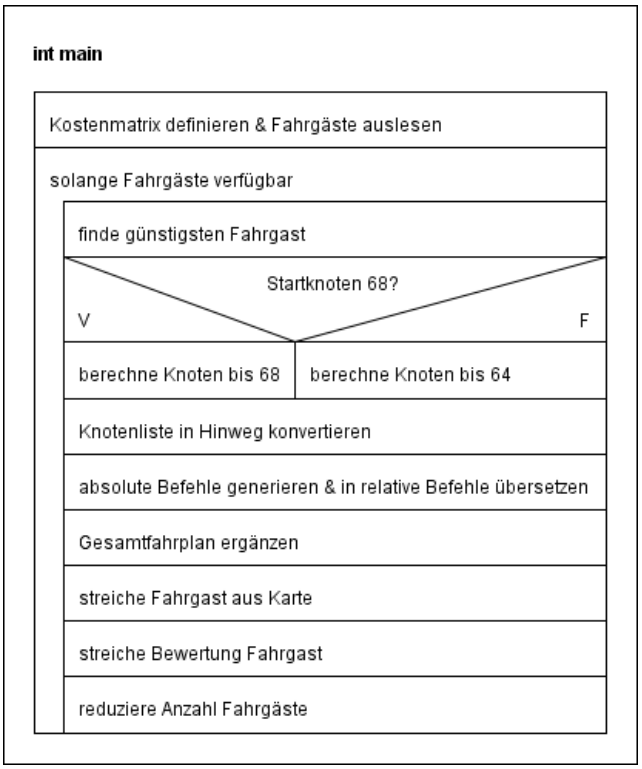
Zu Beginn dieser Methode wird die Ausrichtung des Roboters während der Fahrt bestimmt. Dies hilft uns später dabei die absoluten Befehle (Nord-Süd) in relative Befehle (links-rechts) zu übersetzen. Dafür wird das Spielfeld in einer for-Schleife erneut durchlaufen. Mithilfe einer inneren switch-case-Auswahl werden nun immer zwei aufeinander folgende Knotenpunkte aus schedule voneinander subtrahiert, um anhand ihrer Differenz zu

ermitteln, in welche Richtung der Roboter nach jedem Knoten schaut. Nachdem die Ausrichtungen nun ermittelt wurden, deckt der Roboter nun einen besonderen Sonderfall ab. Wird die Fahrt nach Norden bereits zu Beginn versperrt, so dreht er sich gleich zu Beginn nach Westen bzw. Osten. Der Roboter erkennt dies, wenn bereits die erste Differenz eine Position ermittelt, die nicht der Startausrichtung entspricht. Diese Funktion ist aber erst nach Zusammenschluss der einzelnen Roboter-Komponenten möglich.
In der nächsten for-Schleife wird nun das soeben generierte alignment-Array durchlaufen. In Abhängigkeit der Richtungswechsel werden nun die absoluten Befehle in relative übersetzt. Zum Beispiel: wechselt der Roboter von Nord nach West, so biegt er links ab. Wechselt der Roboter von Ost nach Süd, so biegt er rechts ab. Nach diesem Schema wird nun das guideSchedule-Array aufgebaut und die Laufvariable index2 nach jedem Eintrag hochgezählt. Zu guter Letzt wird in einer erneuten for-Schleife der Hinweg in den Rückweg mit relativen Befehlen übersetzt. Dabei das Array wiederum rückwärts ausgelesen. Dabei muss allerdings beachtet werden, dass jetzt auch die relativen Befehle negiert werden müssen, d.h. aus links wird rechts und aus rechts wird links.



void computeDrive()
Diese Methode wird benutzt, um die einzelnen Fahraufträge in einem großen Array zusammenzufassen – dem guide-Array. Dieses bietet Platz für 200 Steuerbefehle, da es laut Statuten nie mehr als drei Fahrgäste pro Bot geben wird. Damit kann dieses Limit nicht überschritten werden.
Für das Zusammensetzen des Arrays werden lediglich nacheinander das guideSchedule- und das guideBack-Array bis zum letzten eingetragenen Befehl ausgelesen und in guide eingebettet. Der Eintrag erfolgt dabei über den global verwalteten Index guideIndex, der nie zurückgesetzt wird, um ein fortlaufendes Eintragen der Befehle zu ermöglichen.
Beim Verbinden der beiden Teilsegmente der Roboterprogrammierung wird das Zusammensetzen des Arrays noch leicht modifiziert.

int main(int argc, const char *argv[])
In der main-Methode werden nur noch die Methoden-Aufrufe und leichte Modifizierungen vorgenommen. Die main benötigt zwei zusätzliche Variablen: char node, in der der Zielknoten für den aktuellen Auftrag gespeichert wird, und char index, in dem die Anzahl der Knotenpunkte des aktuellen Auftrags gespeichert wird.
Als erstes geht es in die Methode der Kostenmatrix. Die Berechnung hier wird nur einmal vorgenommen.
Für den weiteren Fortlauf ist eine while-Schleife von Nöten. Diese wird so lange durchlaufen, bis alle berechneten Fahrgäste abgearbeitet wurden.
Nun wird in node das Ziel der aktuellen Fahrt gespeichert. Eine kleine if-Bedingung entscheidet nun abhängig vom Startknoten, in welche getRoute-Methode der Roboter einsteigen muss. Diese liefert neben den Knotenpunkten auch die Anzahl dieser zurück und speichert sie für das spätere Auslesen in der Variable Index.
Jetzt werden in der oben genannten Reihenfolge die Fahraufträge weiterverarbeitet, bis der fertige Plan mit den relativen Befehlen entstanden und im guide-Array zusammengefügt ist.
Ist der aktuelle Fahrgast abgearbeitet wird er aus dem Plan gestrichen. Das bedeutet seine Position sowohl in arena als auch in rating mit x überschrieben wird. So wird gewährleistet, dass bereits abgearbeitete Fahrgäste spätere Berechnungen nicht in die Irre führen.
Nachdem nun noch die Anzahl der berechneten Fahrgäste um eins vermindert wird, beginnt der Durchlauf der while-Schleife erneut.



Änderungen nach dem Zusammenfügen der Codesegmente

Für den betriebsbereiten Roboter und das Zusammensetzen der Codesegmente wurden am Navi einige Änderungen notwendig.

Das arena-Array, das im separaten Navi noch hartkodiert initialisiert wurde, wird im fertigen Programm aus einer Headerdatei entnommen. Für ein einheitliches Nutzen der Headerdatei musste arena in _fa umbenannt werden.

Die noch nicht losgelöst implementierbare Funktion zum Korrigieren des Roboters, wenn die nördliche Fahrt beim Start blockiert wird, konnte nun auch technisch umgesetzt werden.

Eine signifikante Änderung musste auch in der physischen Komponente vorgenommen werden. Damit Hin- und Rückweg einfach konvertierbar sind, muss der Roboter sich nach jedem Ein- und Auswerfen des Fahrgastes um 180 Grad versetzt zurück auf die Position des Funktionsaufrufes begeben. Das heißt, schaut der Bot nach Westen, wenn der Befehl zum Einladen kommt, wird er danach nach Osten blickend an die Kreuzung zurückgegeben. Diese Neuausrichtung führt kurze Zeit zum Übersteuern des Roboters, da er während seiner Ausrichtung evtl. seine gewünschte Fahrtrichtung überspringt, bevor er sich wieder nach ihr ausrichtet. Der Ablauf dieser Funktion ist allerdings zu 100% fehlerfrei und ermöglicht einen vereinfachten Ablauf.

Eine weitere wichtige Änderung ergibt sich beim Zusammensetzen des guide-Arrays. Hier müssen nun noch die Befehle für das Be- und Entladen des Fahrgastes eingesetzt werden. Dafür wird einfach der Befehls-Buchstabe für das Beladen nach Eintragung des Hinweges und der Befehl für das Entladen nach Eintragung des Rückweges eingetragen. Die Befehle beinhalteten vor dem Zusammenschluss der Codesegmente noch eine eigene Abbiege-Funktion. Diese musste entfernt werden, da das letzte Abbiegen schon im Navi berechnet wird.

Ist das guide-Array fertig zusammengesetzt, wird ein letzter Befehl an die Kette angefügt, der Stop-Befehl. Hat der Programmablauf den Stop-Befehl erreicht wurden alle Fahrgäste erfolgreich befördert und der Roboter wartet auf seine Abschaltung.

```
//Autor: K.-U. Mrkor

//Standard-Include-Files
#include <stdio.h>
#include <regc515c.h>

//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>
#define FA8
#include "fa.h"
char octoMitteL = 1, octoMitteR = 3, bed = 0;
char fahrcounter = 0;
char buh = 0;
unsigned char a = 0, b = 0, c = 255 ;
char motorL = 3  , motorR = 0, bed1 = 0, ball = 0;
char kreuzung = 0, octoAussenR = 10, octoAussenL =  11, octVL = 13, octVR = 14;
char hc = 0;

//char _fa[71] =                                                                ↵
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx...x.xxx..x.xF..x..Fx..x..x";
//char _fa[71] =                                                                ↵
"xFxxxFxx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.xF..x..Fxx.x.xxx..x..xx.x.x.x";
//char _fa[71] =                                                                ↵
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx..x..xxx..x.xF..x..Fx..x..x";

/*Array fuer Bewertungen*/
char rating[71] =                                                               ↵
".....................................................................";

/*Arrays fuer die Fahrplaene*/
char schedule[71] =                                                             ↵
".....................................................................";

/*Arrays fuer die Rueckwege*/
char wayBack[71] =                                                              ↵
".....................................................................";

/*Bewegungsablauf für den Roboter*/
char guideBack[] =                                                              ↵
".....................................................................";
char guideSchedule[] =                                                          ↵
"g....................................................................";
char guide[200];

/*Startpunkt und Ausrichtung*/
char startNode ;
char startAlignment = 'n';
char alignment[] =                                                              ↵
".....................................................................";

/*Anzahl der vorhandenen Fahrgäste*/
char anzF = 0;

char guideIndex = 0;
```

```
//Eigentliches Korrigieren
void Korrigieren(short m1 , short m2)
{
  //Zum Korriegieren in eine bestimmt Richtung wird das entgegengesetzte Rad langsamer
  gedreht
  motor_pwm(m1, 2);
  motor_pwm(m2, 10);
}

//KorrigierRichtung bestimmen
void KorrigierRichtung(short m1 , short m2,short octL, short octR)
{
  //Links Korriegieren wenn octo Rechts weiß anzeigt
  if(analog(octL) < 100 && analog(octR) > 100)
  {
    Korrigieren(m2,m1);
  }
  //Rechts Korriegieren wenn octo Links weiß anzeigt
  else if(analog(octR) < 100 && analog(octL) > 100)
  {
    Korrigieren(m1,m2);
  }
  //Wenn beide weiß dann letzte Korrektur wiederholen
  else if(analog(octR) < 100 && analog(octL) < 100)
  {
  }
}

//Rueckwaerts fahren
void Rueckwaerts(short m1, short m2)
{

  motor_richtung(m1, 0);
  motor_richtung(m2, 0);
  motor_pwm(m1, 10);
  motor_pwm(m2, 7);
}

//Gerade Fahren
void Gerade(short m1, short m2)
{
  motor_pwm(m1, 10);
  motor_pwm(m2, 10);
}

//Methode zum abbiegen
void Abbiegen(short m1, short m2, short m1P, short m2P, short cou)
{
  int x;
  //Schleife fuer Anzahl der Abbiegungen
  for(x = 1 ; x <= cou; x++)
  {
    motor_richtung(m1, 0);
    motor_richtung(m2, 1);
    motor_pwm(m1, m1P);
```

```
    motor_pwm(m2, m2P);

    //Solange die beiden Mittleren Octokoppler schwarz anzeige drehe dich
    while(bed != 1)
    {


      if(analog(octoMitteR) < 100 && analog(octoMitteL) < 100)
      {
        bed = 1;
      }
    }

    //Solange beide Mittleren Octokoppler noch weiß sind drehe dich weiter
    while(bed != 0)
    {

      if(analog(octoMitteR) > 100 && analog(octoMitteL) > 100)
      {
        bed = 0;
      }
    }
  }


}

//Methode zum Ballholen
void schliessen(short m1, short m2)
{
  motor_richtung(m2, 1); //MotorLinks nach Rechtsdrehen
  motor_richtung(m1, 1); //MotorRechts nach Rechtsdrehen

  //DifferenzMessung anhand der aeußeren Lichteinfluesse
  while(c  > 100)
  {
    led(2,0);
    sleep(10);
    a = analog(7);
    led(2,1);
    sleep(10);
    b = analog(7);
    c  = a - b ;
    KorrigierRichtung(m1,m2,octoMitteL,octoMitteR);
    if(analog(octoMitteR) > 100 && analog(octoMitteL) > 100)
    {
      Gerade(m1,m2);
    }
  }

  //Wenn c <= 100 sprich der Ball unterbricht den Lichtstrahl, Schranke schließen
  while(c <= 100)
  {
    if(buh == 0)
    {
      servo_arc(2, 80);
```

```
      buh = 1;
    }

    //Nach erhalt des Balls Ruckwaertsfahren bis einer der beiden außern octokoppler
    schwarz ist
    if(bed1 == 0)
      Rueckwaerts(m1, m2);

    if(analog(octoAussenR) > 100 || analog(octoAussenL) > 100 || bed1 == 1)
    {
      bed1 =1;
      //Wenn Fahrzeug nach osten schaut Drehung nach Rechts ansonsten nach Links
      if(hc == 1 )
      {
        Abbiegen(m2,m1,6,4,2);
        hc = 3;
      }
      else if(hc == 0)
      {
        Abbiegen(m2,m1,6,4,2);
        hc = 2;
      }
      else
      {
        Abbiegen(motorL,motorR,4,4,2);
        hc = 1;
      }
      ball = 0;
      c = 255;
      buh = 0;
      break;
    }

  }
}

//Methode zum Ballabladen
void oeffnen(short m1 , short m2)
{
  char x = 1;

  //Ballhalter oeffnen
  if(bed1 == 0 )
  {

    ball = 0;
    if(buh == 0)
    {
      servo_arc(2, 30);
      buh = 1;
    }
    Rueckwaerts(m1, m2);
  }
  //Wenn beim Ruckwaerts fahren ein Koppler schwarz erkennt Drehen je nach
  Himmelsrichtung
  if(analog(octoAussenR) > 100 || analog(octoAussenL) > 100 || bed1 == 1 )
```

```
  {
    buh = 0;
    bed1 = 1;


    //Wenn osten dann um 90° nach Links drehen
    if(hc == 1)
    {
      Abbiegen(m2,m1,4,4,3);
    }
    //Wenn Sueden dann um 180° nach Rechts drehen
    else if(hc == 2)
    {
      Abbiegen(m2,m1,6,4,2);
    }
    //Wenn Westen dann um 270° nach  Rechts drehen
    else if(hc == 3)
    {
      Abbiegen(m1,m2,4,4,3);
    }

    hc = 0;


  }


}

/*berechnet fuer alle Knotenpunkte die vom Start aus aufzubringenden Kosten*/
void computeRating(){
  char i;
  char j;
  char k;
  char couldExpand = 1;
  char oldRating = 0;
  char newRating = 1;

  rating[startNode] = 0;

  /*sperrt unbefahrbare Kreuzungen in der Kostenmatrix*/
  for(i = 0; i < (sizeof(_fa) - 1); i++){
    if(_fa[i] == 'F' && (i % 7 == 0) && _fa[i+1] == 'x')
      _fa[i] = 'x';
    if(_fa[i] == 'F' && (i % 7 == 6) && _fa[i-1] == 'x')
      _fa[i] = 'x';
    if(_fa[i] == 'F' && (i / 7 == 0) && _fa[i+7] == 'x')
      _fa[i] = 'x';
    if(_fa[i] == 'x')
      rating[i] = 'x';
  }


  /*solange Knoten expandiert werden konnten*/
  while(couldExpand == 1){
    couldExpand = 0;
```

```
    /*durchlaueft Array und expandiert alle Knoten nach aufsteigenden Kosten mittels
    Breitensuche*/
    for(j = 0; j < (sizeof(rating) - 1); j++){

      if(rating[j] == oldRating){
        if((j > 0) && (j % 7 != 0) && (rating[j-1] == '.')){
          rating[j-1] = newRating;
          couldExpand = 1;
        }
        if((j > 6) && (rating[j-7] == '.')){
          rating[j-7] = newRating;
          couldExpand = 1;
        }
        if((j < (sizeof(rating) - 2)) && (j % 7 != 6) && rating[j+1] == '.'){
          rating[j+1] = newRating;
          couldExpand = 1;
        }
        if((j < (sizeof(rating) - 8)) && (rating[j+7] == '.')){
          rating[j+7] = newRating;
          couldExpand = 1;
        }
      }
    }

    oldRating = newRating;
    newRating += 1;
  }

  for(k = 0; k < (sizeof(_fa) - 1); k++){


    if(_fa[k] == 'F' && rating[k] != '.')
      anzF++;
  }

}

/*findet das naechstgelegene Ziel*/
/*gibt den billigsten Zielknoten zurueck*/
char findNextTarget(){
  char i;
  char target = 0;
  char cost;
  char cheaper = 80;

  /*berechnet Kosten fuer das einzulesende Ziel*/
  for(i = 0; i < (sizeof(_fa) - 1); i++){

    if(_fa[i] == 'F' && rating[i] != '.'){
      cost = rating[i];


      /*wenn aktueller Zielknoten billiger ist wird er prioritisiert*/
      if(cost <= cheaper){
        cheaper = cost;
```

```
        target = i;
      }
    }
  }

  return target;
}

/*berechnet aus _fa und rating abzuarbeitenden Fahrauftrag vom Ziel zur Homebase*/
char getRoute68(char node){
  char cost;
  char index = 0;

  /*sucht sich den Weg vom Ziel zurueck zum Start in der Reihenfolge S-W-O-N*/
  while(rating[node] != 0){
    cost = rating[node];

    /*Sueden befahrbar?*/
    if(rating[node+7] == (cost - 1)){
      wayBack[index] = node;
      node += 7;
    }
    /*Westen befahrbar?*/
    else if(rating[node-1] == (cost - 1)){
      wayBack[index] = node;
      node -= 1;
    }
    /*Osten befahrbar?*/
    else if(rating[node+1] == (cost - 1)){
      wayBack[index] = node;
      node += 1;
    }
    /*Norden befahrbar?*/
    else if(rating[node-7] == (cost - 1)){
      wayBack[index] = node;
      node -= 7;
    }

    index++;
  }

  wayBack[index] = node;

  return index;
}

char getRoute64(char node){
  char cost;
  char index = 0;

  /*sucht sich den Weg vom Ziel zurueck zum Start in der Reihenfolge S-W-O-N*/
  while(rating[node] != 0){
    cost = rating[node];

    /*Sueden befahrbar?*/
    if(rating[node+7] == (cost - 1)){
```

```
      wayBack[index] = node;
      node += 7;
    }
    /*Osten befahrbar?*/
    else if(rating[node+1] == (cost - 1)){
      wayBack[index] = node;
      node += 1;
    }
    /*Westen befahrbar?*/
    else if(rating[node-1] == (cost - 1)){
      wayBack[index] = node;
      node -= 1;
    }
    /*Norden befahrbar?*/
    else if(rating[node-7] == (cost - 1)){
      wayBack[index] = node;
      node -= 7;
    }

    index++;
  }

  wayBack[index] = node;

  return index;
}

/*negiert den Ziel-Homebase-Weg, um zum Fahrgast zu gelangen*/
void fetchPassenger(char index){
  char i;

  /*initialisiert den Fahrplan mit dem umgekehrten Rueckweg*/
  for(i = 0; i <= index; i++){



    schedule[i] = wayBack[index - i];
  }
}

/*konvergiert den absoluten Fahrplan (NWES) in relativen Fahrplan (LRG)*/
void getGuideOrder(char index){
  char i;
  char j;
  char k;
  char index2 = 0;

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


    switch(schedule[i] - schedule[i+1]){
    case 7:
      alignment[i] = 'n';
      break;

    case 1:
```

```
      alignment[i] = 'w';
      break;

    case -1:
      alignment[i] = 'e';
      break;

    case -7:
      alignment[i] = 's';
      break;
    }
  }

  //korrigiert die Startposition, falls die Fahrt in Startrichtung nicht moeglich ist
  if(alignment[0] != startAlignment){
    if(alignment[0] == 'w')
      guideSchedule[0] = 'l';
    else if (alignment[0] == 'e')
      guideSchedule[0] = 'r';
  }

  for(j = 0; j <= (index - 1); j++){


    switch(alignment[j]){
    case 'n':
      if(alignment[j+1] == 'n'){
        guideSchedule[j+1] = 'g';
        index2++;
      }
      if(alignment[j+1] == 'w'){
        guideSchedule[j+1] = 'l';
        index2++;
      }
      if(alignment[j+1] == 'e'){
        guideSchedule[j+1] = 'r';
        index2++;
      }
      break;

    case 'e':
      if(alignment[j+1] == 'e'){
        guideSchedule[j+1] = 'g';
        index2++;
      }
      if(alignment[j+1] == 'n'){
        guideSchedule[j+1] = 'l';
        index2++;
      }
      if(alignment[j+1] == 's'){
        guideSchedule[j+1] = 'r';
        index2++;
      }
      break;

    case 'w':
```

```
      if(alignment[j+1] == 'w'){
        guideSchedule[j+1] = 'g';
        index2++;
      }
      if(alignment[j+1] == 's'){
        guideSchedule[j+1] = 'l';
        index2++;
      }
      if(alignment[j+1] == 'n'){
        guideSchedule[j+1] = 'r';
        index2++;
      }
      break;

    case 's':
      if(alignment[j+1] == 's'){
        guideSchedule[j+1] = 'g';
        index2++;
      }
      if(alignment[j+1] == 'o'){
        guideSchedule[j+1] = 'l';
        index2++;
      }
      if(alignment[j+1] == 'w'){
        guideSchedule[j+1] = 'r';
        index2++;
      }
      break;
    }
  }

  for(k = 0; k <= index2; k++){


    if(guideSchedule[index2 - k] == 'l')
      guideBack[k] = 'r';
    else if(guideSchedule[index2 - k] == 'r')
      guideBack[k] = 'l';
    else
      guideBack[k] = guideSchedule[index2 - k];
  }

}

void computeDrive(){
  char i;
  char j;

  for(i = 0; i < sizeof(guideSchedule); i++){


    if(guideSchedule[i] == '.')
      break;
    guide[guideIndex] = guideSchedule[i];
    guideIndex++;
  }
```

```
  guide[guideIndex] = 'b';
  guideIndex ++;

  for(j = 0; j < sizeof(guideBack); j++){
    if(guideBack[j] == '.')
      break;
    guide[guideIndex] = guideBack[j];
    guideIndex++;

  }

  guide[guideIndex - 1] = 'o';

  if(anzF == 1)
    guide[guideIndex] = 's';

}



//Hauptprogrammroutine
void AksenMain(void)
{


  char node;
  char index;

  //Wenn Servo nach fahrt noch zu, Funktion zum oeffnen
  if(dip_pin(0) == 0)
  {
    servo_arc(2, 30);
  }

  //Funktion fuer den Startpkt auf der Strecke
  if(dip_pin(3) == 0)
  {
    startNode = 64;
  }
  else
    startNode = 68;


  //Hier beginnt die Suche
  computeRating();



  while(anzF > 0){
    node = findNextTarget();

    if(startNode == 68)
      index = getRoute68(node);
    else
      index = getRoute64(node);
```

```
    fetchPassenger(index);
    getGuideOrder(index);
    computeDrive();
    _fa[node] = 'x';
    rating[node] = 'x';
    anzF--;

    if(anzF == 1)
      guide[guideIndex] = 's';
  }

  led(0,1);

  // Hier beginnt die Fahrt
  while(1)
  {
    lcd_cls();
    lcd_puts("Warte auf Start");
    lcd_setxy(1,0);
    lcd_ubyte(_fa_nr);
    sleep(100);

    //Warten auf Startsignal
    if(analog(4) < 150)
    {
      lcd_cls();
      lcd_puts("Start");
      while(fahrcounter >= 0)
      {


        motor_richtung(motorR, 1); //MotorLinks nach Rechtsdrehen
        motor_richtung(motorL, 1); //MotorRechts nach Rechtsdrehen


        KorrigierRichtung(motorL,motorR,octoMitteL,octoMitteR);

        //Stoppen wenn ende des Fahrplans erreicht
        while(guide[fahrcounter] == 's')
        {
          motor_pwm(motorR, 0);
          motor_pwm(motorL, 0);
        }
        if(analog(octoMitteR) > 100 && analog(octoMitteL) > 100 )
        {
          Gerade(motorL,motorR);
        }


        //Hier wird bestimmt was er macht wenn er auf die Kreuzung kommt
        if((analog(octVL) > 100 && analog(octVR) > 100) || ball == 0 || guide
        [fahrcounter] =='b ' )
        {
          kreuzung = 1;
          bed1 = 0;
```

```
          while(kreuzung == 1)
          {
            KorrigierRichtung(motorL,motorR,octoMitteL,octoMitteR);

            if(analog(octoAussenR) > 100 || analog(octoAussenL) > 100 || ball == 0 ||
            guide[fahrcounter] =='b ' )
            {
              ball = 1;
              switch(guide[fahrcounter])
              {
              case 'l':
                Abbiegen(motorL,motorR,4,4,1);
                if(hc == 0)
                  hc = 3;
                else
                  hc -= 1;
                break;
              case 'r':
                Abbiegen(motorR,motorL,6,3,1);
                if(hc == 3)
                  hc = 0;
                else
                  hc += 1;
                break;
              case 'g':
                Gerade(motorL,motorR);
                break;
              case 'o':
                oeffnen(motorL,motorR);
                break;
              case 'b':
                schliessen(motorL,motorR);
                break;

              }

              //Fahrcounter erhöhen und Abbiegen rücksetzen
              if((analog(octoMitteR) > 100 || analog(octoMitteL) > 100))
              {
                fahrcounter += 1;
                kreuzung = 0;
              }
            }
          }
        }


      }
    }
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

/*Array mit der Karte*/
/*char arena[71] = "xxFxFxx
                    x..x..x
                    F.x.x.F
                    x.x...x
                    x.x..xx
                    x..x..x
                    x...x.x
                    xx..x.x
                    F..x..F
                    x..x..x";*/

//char arena[71] =
"xxFxFxxx..x..xF.x.x.Fx.x...xx.x..xxx..x..xx...x.xxx..x.xF..x..Fx..x..x";
//char arena[71] =
"xxxFxxxx.....xx..x..xx..x..xF..x..Fx..x..xx..x..xxx.x.xxx..x..xx..x..x";
char arena[71] =
"xxxFxxxF..x..Fx..x..xFx.x.xFF..x..Fx...x.xx..x..xx.x...xx..x..xx..x..x";

/*Array fuer die Bewertungsmatrix*/
char rating[71] =
".....................................................................";

/*Arrays fuer die Fahrplaene*/
char schedule[71] =
".....................................................................";

/*Arrays fuer die Rueckwege*/
char wayBack[71] =
".....................................................................";

/*Bewegungsablauf für den Roboter die Plaene fuer Hin- und Rueckweg
  werden separat berechnet und spaeter in guide zusammengefuehrt,
  guideSchedule ist an der Stelle 0 mit g vorimplementiert, da sich der
  Roboter am Start so ausrichtet, dass er als erstes immer geradeaus fahren muss*/
char guideBack[] =
".....................................................................";
char guideSchedule[] =
"g....................................................................";
char guide[100];

/*Startpunkt und Ausrichtung des Roboters*/
char startNode = 68;
char startAlignment = 'n';
char alignment[] =
".....................................................................";

/*Anzahl der vorhandenen Fahrgäste
  wird erst spaeter berechnet und erfordert
  aber eine globale Nutzung*/
char anzF = 0;
```

```
/*Index fuer das Array guide
  global notwendig, da ein
  fortlaufen ohne ruecksetzen
  erforderlich ist*/
char guideIndex = 0;

/*berechnet fuer alle Knotenpunkte die vom Start aus aufzubringenden Kosten*/
void computeRating(){
    char i;
    char j;
    char k;
    char couldExpand = 1;
    char oldRating = 0;
    char newRating = 1;

    /*der Startpunkt wird mit 0 bewertet*/
    rating[startNode] = 0;

    /*sperrt unbefahrbare Kreuzungen in der Kostenmatrix
      und auch unerreichbare Fahrgaeste, die durch direkte
      Nachbarschaft zu befahrbahren Fahrgaesten zu Verwirrung
      fuehren*/
    for(i = 0; i < (sizeof(arena) - 1); i++){
        if(arena[i] == 'F' && (i % 7 == 0) && arena[i+1] == 'x')
            arena[i] = 'x';
        if(arena[i] == 'F' && (i % 7 == 6) && arena[i-1] == 'x')
            arena[i] = 'x';
        if(arena[i] == 'F' && (i / 7 == 0) && arena[i+7] == 'x')
            arena[i] = 'x';
        if(arena[i] == 'x')
            rating[i] = 'x';
        }

    /*solange Knoten expandiert werden konnten*/
    while(couldExpand == 1){
        couldExpand = 0;

        /*durchlaueft Array und expandiert alle Knoten nach
          aufsteigenden Kosten mittels Breitensuche,
          Berechnung greift sobald ein Knoten die aktuellen Kosten enthält
          und die jeweilige Himmelsrichtung frei und unbewertet ist*/
        for(j = 0; j < (sizeof(rating) - 1); j++){
            if(rating[j] == oldRating){
                if((j > 0) && (j % 7 != 0) && (rating[j-1] == '.')){
                    rating[j-1] = newRating;
                    couldExpand = 1;
                }
                if((j > 6) && (rating[j-7] == '.')){
                    rating[j-7] = newRating;
                    couldExpand = 1;
                }
                if((j < (sizeof(rating) - 2)) && (j % 7 != 6) && rating[j+1] == '.'){
                    rating[j+1] = newRating;
                    couldExpand = 1;
                }
```

```
                if((j < (sizeof(rating) - 8)) && (rating[j+7] == '.')){
                    rating[j+7] = newRating;
                    couldExpand = 1;
                }
            }
        }

        /*oldRating und newRating werden nach
          jedem Durchlauf hoch gezaehlt*/
        oldRating = newRating;
        newRating += 1;
    }

    /*zaehlt bewertete Fahrgaeste und gibt die Anzahl
      erreichbarer Fahrgaeste an die globale Variable anzF*/
    for(k = 0; k < (sizeof(arena) - 1); k++){
        if(arena[k] == 'F' && rating[k] != '.')
            anzF++;
    }

}

/*findet das naechstgelegene Ziel und
  gibt den billigsten Zielknoten zurueck*/
char findNextTarget(){
    char i;
    char target;
    char cost;
    char cheaper = 80;

    /*berechnet Kosten fuer das einzulesende Ziel
      mit Bewertung*/
    for(i = 0; i < (sizeof(arena) - 1); i++){
        if(arena[i] == 'F' && rating[i] != '.'){
            cost = rating[i];


        /*wenn aktueller Zielknoten billiger ist wird er prioritisiert*/
            if(cost <= cheaper){
                cheaper = cost;
                target = i;
            }
        }
    }

    /*target gibt den Fahrgast mit den geringsten Kosten zurueck*/
    return target;
}

/*berechnet aus arena und rating abzuarbeitenden Fahrauftrag vom Ziel zur Homebase
  wenn der Startpunkt der Knoten 68 war*/
char getRoute68(char node){
    char cost;
    char index = 0;

    /*sucht sich den Weg vom Ziel zurueck zum Start in der Reihenfolge S-W-O-N*/
```

```
    while(rating[node] != 0){
        cost = rating[node];

        /*Sueden befahrbar?*/
        if(rating[node+7] == (cost - 1)){
            wayBack[index] = node;
            node += 7;
        }
        /*Westen befahrbar?*/
        else if(rating[node-1] == (cost - 1)){
            wayBack[index] = node;
            node -= 1;
        }
        /*Osten befahrbar?*/
        else if(rating[node+1] == (cost - 1)){
            wayBack[index] = node;
            node += 1;
        }
        /*Norden befahrbar?*/
        else if(rating[node-7] == (cost - 1)){
            wayBack[index] = node;
            node -= 7;
        }

        /*Index wird nach allen eingetragenen Knoten erhoeht*/
        index++;
    }

    /*traegt nach Durchlauf des Plans den Startknoten
      zum Array noch hinzu*/
    wayBack[index] = node;

    /*der Index wird zurueck gegeben, damit er an die folgenden
      Funktionen übergeben wird, um alle Arrays später nur bis
      zum letzten eingetragenen Punkt auszulesen*/
    return index;
}

/*berechnet aus arena und rating abzuarbeitenden Fahrauftrag vom Ziel zur Homebase
  wenn der Startpunkt der Knoten 64 war*/
char getRoute64(char node){
    char cost;
    char index = 0;

    /*sucht sich den Weg vom Ziel zurueck zum Start in der Reihenfolge S-O-W-N*/
    while(rating[node] != 0){
        cost = rating[node];

        /*Sueden befahrbar?*/
        if(rating[node+7] == (cost - 1)){
            wayBack[index] = node;
            node += 7;
        }
        /*Osten befahrbar?*/
        else if(rating[node+1] == (cost - 1)){
            wayBack[index] = node;
```

```
            node += 1;
        }
        /*Westen befahrbar?*/
        else if(rating[node-1] == (cost - 1)){
            wayBack[index] = node;
            node -= 1;
        }
        /*Norden befahrbar?*/
        else if(rating[node-7] == (cost - 1)){
            wayBack[index] = node;
            node -= 7;
        }

        /*Index wird nach allen eingetragenen Knoten erhoeht*/
        index++;
    }

    /*traegt nach Durchlauf des Plans den Startknoten
      zum Array noch hinzu*/
    wayBack[index] = node;

    /*der Index wird zurueck gegeben, damit er an die folgenden
      Funktionen übergeben wird, um alle Arrays später nur bis
      zum letzten eingetragenen Punkt auszulesen*/
    return index;
}

/*negiert den Ziel-Homebase-Weg, um den Weg
  zum Fahrgast zu ermitteln*/
void fetchPassenger(char index){
    char i;

    /*initialisiert den Fahrplan mit dem umgekehrten Rueckweg*/
    for(i = 0; i <= index; i++){
        schedule[i] = wayBack[index - i];
    }
}

/*konvergiert den absoluten Fahrplan (NWOS) in relativen Fahrplan (LRG)*/
void getGuideOrder(char index){
    char i;
    char j;
    char k;
    char index2 = 0;

    /*berechnet aus dem Abstand von zwei aufeinanderfolgenden
      Knoten die aktuelle Ausrichtung des Roboters und schreibt
      sie in das Array hinein*/
    for(i = 0; i <= index; i++){
        switch(schedule[i] - schedule[i+1]){
            case 7:
                alignment[i] = 'n';
                break;

            case 1:
                alignment[i] = 'w';
```

```
                break;

            case -1:
                alignment[i] = 'e';
                break;

            case -7:
                alignment[i] = 's';
                break;
        }
    }

    /*korrigiert die Startposition, falls die Fahrt in Startrichtung nicht moeglich ist*/
    /*(in diesem Code nicht vollständig implementiert, da es erst nach dem
    Zusammenfuehren
      der Codes umsetzbar waere)*/
    /*if(alignment[0] != startAlignment){
        if(alignment[0] == 'w')
            linksDrehen;
        else if (alignment[0] == 'e')
            rechtsDrehen;
    }*/

    /*berechnet aus zwei aufeinanderfolgenden Ausrichtungen
      den auszufuehrenden Befehl des Roboters, konvertiert also
      absoluten in relativen Fahrplan*/
    for(j = 0; j <= (index - 1); j++){
        switch(alignment[j]){
            case 'n':
                if(alignment[j+1] == 'n'){
                    guideSchedule[j+1] = 'g';
                    index2++;
                }
                if(alignment[j+1] == 'w'){
                    guideSchedule[j+1] = 'l';
                    index2++;
                }
                if(alignment[j+1] == 'e'){
                    guideSchedule[j+1] = 'r';
                    index2++;
                }
                break;

            case 'e':
                if(alignment[j+1] == 'e'){
                    guideSchedule[j+1] = 'g';
                    index2++;
                }
                if(alignment[j+1] == 'n'){
                    guideSchedule[j+1] = 'l';
                    index2++;
                }
                if(alignment[j+1] == 's'){
                    guideSchedule[j+1] = 'r';
                    index2++;
                }
```

```
                break;

            case 'w':
                if(alignment[j+1] == 'w'){
                    guideSchedule[j+1] = 'g';
                    index2++;
                }
                if(alignment[j+1] == 's'){
                    guideSchedule[j+1] = 'l';
                    index2++;
                }
                if(alignment[j+1] == 'n'){
                    guideSchedule[j+1] = 'r';
                    index2++;
                }
                break;

            case 's':
                if(alignment[j+1] == 's'){
                    guideSchedule[j+1] = 'g';
                    index2++;
                }
                if(alignment[j+1] == 'o'){
                    guideSchedule[j+1] = 'l';
                    index2++;
                }
                if(alignment[j+1] == 'w'){
                    guideSchedule[j+1] = 'r';
                    index2++;
                }
                break;
        }
    }

    /*die Hinfahrt mit den relativen Befehlen wird in den
      Rueckweg mit den relativen Befehlen konvertiert,
      dafuer wird der Hinweg rueckwaerts ausgelesen und
      alle Abbiegungen negiert*/
    for(k = 0; k <= index2; k++){
        if(guideSchedule[index2 - k] == 'l')
            guideBack[k] = 'r';
        else if(guideSchedule[index2 - k] == 'r')
            guideBack[k] = 'l';
        else
            guideBack[k] = guideSchedule[index2 - k];
    }

}

/*diese Methode fuehrt die einzelnen Fahrtwege in ein einzelnes großes Array
  zusammen, es laueft ein globaler Index, damit das Array fortlaufend implementiert
  wird*/
/*das separate Zusammensetzen ist notwendig, da nach dem Zusammenfuehren beider
  Codes zwischen Hin- und Rueckfahrt weitere Befehle in das Array integriert werden*/
void computeDrive(){
    char i;
```

```
    char j;

    /*fuegt den Hinweg fuer den aktuellen Fahrgast in den großen Fahrplan ein*/
    for(i = 0; i < sizeof(guideSchedule); i++){
        if(guideSchedule[i] == '.')
            break;
        guide[guideIndex] = guideSchedule[i];
        guideIndex++;
    }

    /*fuegt den Rueckweg fuer den aktuellen Fahrgast in den großen Fahrplan ein*/
    for(j = 0; j < sizeof(guideBack); j++){
        if(guideBack[j] == '.')
            break;
        guide[guideIndex] = guideBack[j];
        guideIndex++;
    }

}

int main(int argc, const char *argv[]){
    char node;
    char index;

    /*einmaliges Berechnen der Kostenmatrix*/
    computeRating();

    /*laueft solange wie Fahrgaeste auf den Transport warten*/
    while(anzF > 0){
        node = findNextTarget();

        /*Entscheidungspunkt: die Berechnung des Weges
          haengt hier vom Startpunkt ab*/
        if(startNode == 68)
            index = getRoute68(node);
        else
            index = getRoute64(node);

        fetchPassenger(index);
        getGuideOrder(index);
        computeDrive();

        /*der eben abgefahrene Fahrgast wird aus der Karte entfernt
          und die Kosten an dieser Stelle werden gesperrt, fuer
          folgende Berechnung entsteht somit hier eine Sperre*/
        arena[node] = 'x';
        rating[node] = 'x';

        /*die Anzahl der abzuarbeitenden Fahrgaeste wird um eins vermindert*/
        anzF--;
    }

    return 0;
}
```

```c
//Autor: K.-U. Mrkor

//Standard-Include-Files
#include <stdio.h>
#include <regc515c.h>

//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>
char octoMitteL = 1, octoMitteR = 3, bed = 0; // 0 = Schranke auf , 1 = Schranke zu
int fahrcounter = 0;

/*
 void Ballhalter(void)
{

    unsigned char a = 0, b = 0, c = 0 ;

    while(1)
    {
            //Ballhalter
            if(schranke == 0)
            {
                led(2,0);
                sleep(10);
                a = analog(7);
                led(2,1);
                sleep(10);
                b = analog(7);
                c  = a - b ;
            }

            if(c <= 100 && schranke == 0 && prokom != 1 )
            {
                lcd_setxy(1,0);
                lcd_ubyte(c);
                servo_arc(2, 80);
                schranke = 1;


            }


            if(prokom == 1  )
            {
                lcd_setxy(0,0);
                lcd_puts("Aufmachen");
                lcd_setxy(1,0);
                lcd_ubyte(prokom);
                servo_arc(2, 30);
                prokom = 3;
                c = 101;
            }

            process_defer();
```

```
    }

 } */

 void Korrigieren(short m1 , short m2)
 {
     lcd_setxy(0,0);
     lcd_puts("Korrigieren");
     motor_pwm(m1, 2);
     motor_pwm(m2, 10);
 }
 //KorrigierRichtung bestimmen
 void KorrigierRichtung(short m1 , short m2,short octL, short octR)
 {
     if(analog(octL) < 100 && analog(octR) > 100)
     {
            Korrigieren(m2,m1);
     }
     else if(analog(octR) < 100 && analog(octL) > 100)
     {
            Korrigieren(m1,m2);
     }
     else if(analog(octR) < 100 && analog(octL) < 100)
     {
     }
 }

void Rueckwaerts(short m1, short m2)
{
    motor_richtung(m1, 0);
    motor_richtung(m2, 0);
    motor_pwm(m1, 10);
    motor_pwm(m2, 7);
}


//Methode zum abbiegen
void Abbiegen(short m1, short m2, short m1P, short m2P, short cou)
{
    int x;
    for(x = 1 ; x <= cou; x++)
        {
            lcd_setxy(0,0);
            lcd_puts("Abbiegen");
            motor_richtung(m1, 0);
            motor_richtung(m2, 1);
            motor_pwm(m1, m1P);
            motor_pwm(m2, m2P);

        while(bed != 1)
        {


            if(analog(octoMitteR) < 100 && analog(octoMitteL) < 100)
            {
                bed = 1;
```

```
            }
        }


        while(bed != 0)
        {

            if(analog(octoMitteR) > 100 && analog(octoMitteL) > 100)
            {
                bed = 0;
            }
        }
    }


}


//Hauptprogrammroutine
void AksenMain(void)
{
    char motorL = 3  , motorR = 0, bed1 = 0, ball = 1; // motoren
    char kreuzung = 0, octoAussenR = 10, octoAussenL =  11, octVL = 13, octVR = 14;
    unsigned char a = 0, b = 0, c = 255 ;
    //char fahrplan[] =
    {'l','g','r','g','r','g','l','g','l','g','l','g','l','g','r','g','r','s'};   //Kurzer
     Fahrplan zum testen
    char fahrplan[] = {'b','l','o','r','l','g','g','l','r','g','r','l','b','l',};

    if(dip_pin(0) == 1)
    {
        //process_start(Ballhalter, 300);
    }
    led(0,1);
    if(dip_pin(0) == 0)
    {
            servo_arc(2, 30);
    }
    lcd_setxy(0,0);
    lcd_ubyte(analog(4));

    while(1)
    {

        if(analog(4) < 150)
        {
            while(1)
            {


                motor_richtung(motorR, 1); //MotorLinks nach Rechtsdrehen
                motor_richtung(motorL, 1); //MotorRechts nach Rechtsdrehen

                KorrigierRichtung(motorL,motorR,octoMitteL,octoMitteR);
                if(analog(octoMitteR) > 100 && analog(octoMitteL) > 100)
                {
```

```
                lcd_setxy(0,0);
                lcd_puts("Gerade              ");
                lcd_setxy(1,0);
                motor_pwm(motorL, 10);
                motor_pwm(motorR, 10);
            }

            if((analog(octVL) > 100 && analog(octVR) > 100) || ball == 0)
            {
                kreuzung = 1;
                bed1 = 0;
                ball = 1;
                while(kreuzung == 1)
                {
                    KorrigierRichtung(motorL,motorR,octoMitteL,octoMitteR);

                    if(analog(octoAussenR) > 100 || analog(octoAussenL) > 100 )
                    {
                        switch(fahrplan[fahrcounter])
                        {
                            case 'l':
                                Abbiegen(motorL,motorR,4,4,1);

                                break;
                            case 'r':
                                Abbiegen(motorR,motorL,5,2,1);
                                break;
                            case 'g':
                                break;
                            case 's':

                                lcd_setxy(0,0);
                                lcd_puts("   Stop       ");
                                lcd_setxy(1,0);
                                lcd_puts("   Stop       ");
                                motor_pwm(motorL, 0);
                                motor_pwm(motorR, 0);
                                //fahrcounter = 1;
                                break;
                            case 'o':
                                if(bed1 == 0 )
                                {
                                    lcd_setxy(0,0);
                                    lcd_puts("Aufmachen");
                                    servo_arc(2, 30);
                                    Rueckwaerts(motorL, motorR);
                                }
                                if(analog(octoAussenR) > 100 || analog(octoAussenL) >
                        100 || bed1 == 1 )
                                {
                                        bed1 = 1;
                                        Abbiegen(motorR,motorL,6,3,3);
                                        break;
                                }
```

```
                    break;
                case 'b':
                    c  = 255;
                    Abbiegen(motorR,motorL,6,3,1);
                    motor_richtung(motorR, 1); //MotorLinks nach
Rechtsdrehen
                    motor_richtung(motorL, 1); //MotorRechts nach
Rechtsdrehen
                    //Ballhalter

                    while(c  > 100)
                    {
                        led(2,0);
                        sleep(10);
                        a = analog(7);
                        led(2,1);
                        sleep(10);
                        b = analog(7);
                        c  = a - b ;
                        KorrigierRichtung
(motorL,motorR,octoMitteL,octoMitteR);
                        if(analog(octoMitteR) > 100 && analog(octoMitteL)
 > 100)
                        {
                            lcd_setxy(0,0);
                            lcd_puts("Gerade               ");
                            motor_pwm(motorL, 10);
                            motor_pwm(motorR, 10);
                        }
                    }

                    while(c <= 100)
                    {
                        lcd_setxy(1,0);
                        lcd_ubyte(c);
                        servo_arc(2, 80);

                        if(bed1 == 0)
                        Rueckwaerts(motorL, motorR);

                        if(analog(octoAussenR) > 100 || analog
(octoAussenL) > 100 || bed1 == 1)
                        {
                            bed1 =1;
                            Abbiegen(motorR,motorL,6,3,2);
                            ball = 0;
                            break;
                        }

                    }

                    break;
                default:
                    break;

            }
```

```
                                //Fahrcounter erhöhen und Abbiegen rücksetzen
                              if((analog(octoMitteR) > 100 || analog(octoMitteL) > 100) &&
                        fahrplan[fahrcounter] != 's' )
                              {
                                          lcd_setxy(1,0);
                                          lcd_ubyte(fahrcounter);
                                          fahrcounter += 1;
                                          kreuzung = 0;

                              }
                          }
                      }
                  }


            }
            }
    }
}
```