Abgabe für das KI-Projekt

Bearbeitende Studenten:

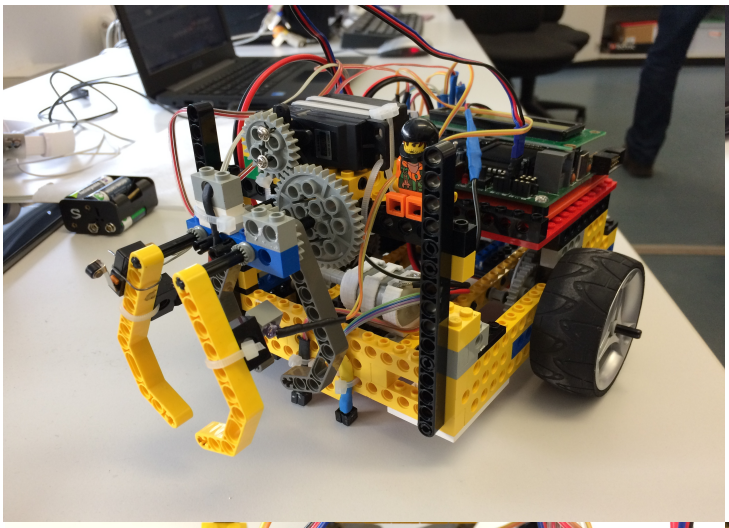
Benjamin Yüksel (20140017, yueksel@th-brandenburg.de)

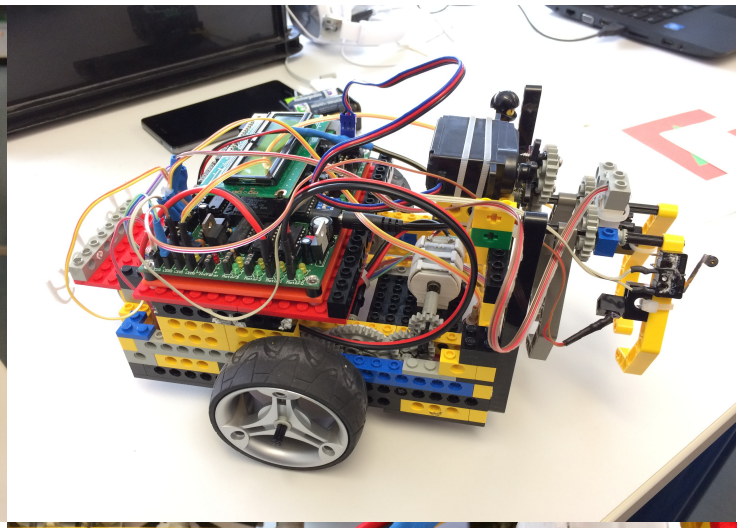
Robert Zeigner (20140042, zeigner@th-brandenburg.de)

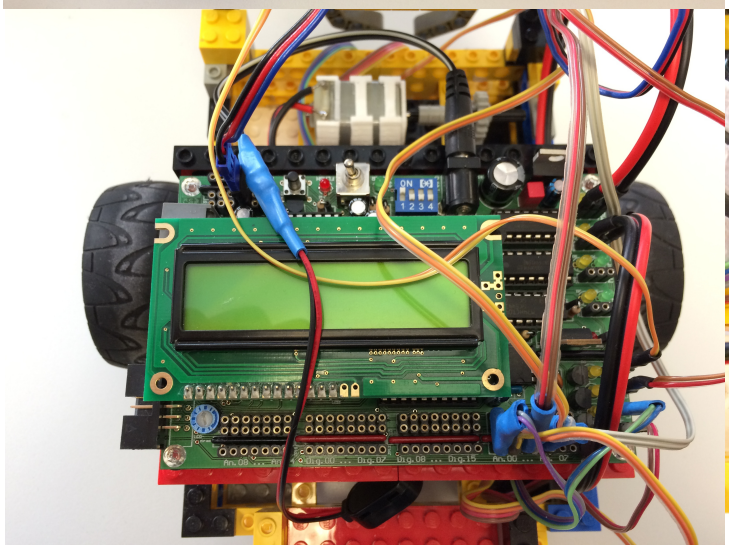
Vorwort

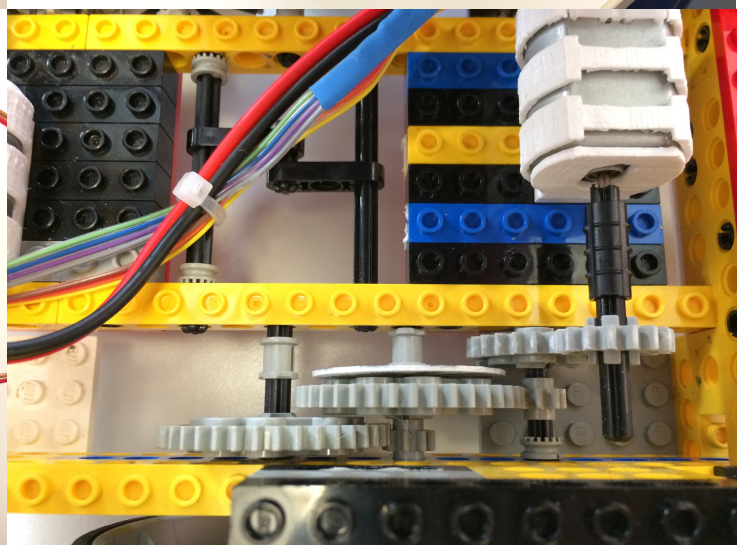
Hiermit bedanken wir uns recht herzlich bei Prof. Dr.-Ing. Jochen Heinsohn und Dipl.-Inform. Ingo Boersch für die tatkräftige Unterstützung, sowie Hilfestellungen und Anregungen im Laufe des Projektes.

Bilder









Der Roboter allgemein

Den Roboter, den wir während des Projektes konstruiert haben, tauften wir auf den Namen „Herby“. Dieser stammt aus dem gleichnamigen Film „Herby – Ein kleiner Flitzer“. Wir hielten ihn für passend, da sich unser Roboter vom Beginn an am schnellsten fortbewegte.
Besonders gut gelang uns der kräftige und schnelle Antrieb des Roboters, sowie der Greifer um die Fahrgäste zu transportieren.

Während der Konstruktion des Roboters sowie der Entwicklung der Software, traten jedoch gerade wegen des schnellen Antriebs immer wieder Probleme auf. Er fuhr in manchen Situationen einfach zu schnell, um dem Linienmuster sicher zu folgen.
Dieses Problem konnten wir jedoch beheben, indem wir die Untersetzung durch eine weitere Getriebestufe erhöht haben. Somit wurde der Roboter langsamer und das Folgen der Linien war nun fehlerfrei möglich.

Die Arbeit an unserem Roboter war für uns herausfordernd, lustig aber manchmal auch etwas nervenaufreibend. Es traten des öfteren unvorhergesehene Probleme und Fehler auf.
Teilweise war es auch schwer, die Fehlerquelle ausfindig zu machen. Insgesamt hat das Projekt aber sehr viel Spaß gemacht und es würde uns freuen, erneut ein solches Projekt durchführen zu können.

Die Hardware

Der Grundaufbau des Roboters besteht aus Lego-Bausteinen. Wir haben dabei versucht, den Rahmen etwas breiter, dafür aber eher flach aufzubauen. Das verringert die Gefahr, dass der Roboter beginnt, zu kippeln. Das ist besonders wichtig, da unser Roboter nur zwei Räder und einen einfachen Schleifpunkt im hinteren Teil besitzt.

Der Antrieb unseres Legoroboters besteht aus zwei Gleichstrommotoren, welche jeweils über ein Getriebe mit den Rädern verbunden sind.
Dabei haben wir einen Motor im vorderen und einen im hinteren Teil des Roboters platziert.
Dies war aus Platzgründen erforderlich, denn wie wir sehr schnell bemerkten, passten die beiden

Motoren nicht nebeneinander in den Grundaufbau unseres Roboters.
Um die Motoren auf dem Rahmen zu befestigen, haben wir uns eine spezielle Halterung angefertigt. In diese können die Motoren einfach hineingesteckt werden. Die Motorhalterungen können dann einfach auf die Legosteine aufgesteckt werden.
Um die Halterungen anzufertigen, haben wir die Motoren genau abgemessen und anschließend ein 3D Modell mit Hilfe von Cinema 4D erstellt.
Dieses haben wir dann mit der slicer Software „cura“ in eine gcode Datei umgewandelt und anschließend mit einen 3D Drucker ausgedruckt.

Im aktuellen Aufbau besteht jedes der beiden Getriebe aus 3 Getriebestufen. Die ersten beiden Stufen haben jeweils ein Untersetzungsverhältnis von 5:1, während die dritte Stufe ein Verhältnis von 1,5:1 bereitstellt. So ergibt sich insgesamt eine Untersetzung von 37,5:1. Diese ermöglicht sowohl eine hohe Geschwindigkeit, als auch ein ausreichendes Drehmoment.
Im ersten Aufbau verwendeten wir hingegen nur zwei Getriebestufen mit einen Untersetzungsverhältnis von insgesamt 25:1. Dieses war jedoch ungeeignet, da der Roboter wie bereits erwähnt, zu schnell wurde und nicht mehr in der Lage war, dem Linienmuster zuverlässig zu folgen. Hinzu kam, dass die Akkus durch den höheren Stromfluss schneller entladen wurden.

Um den Roboter sicher durch das Liniennetz zu steuern, haben wir zwei Optokoppler vorn am Rahmen angebracht. Diese sind mittig ausgerichtet und haben einen Abstand zueinander, der etwas größer als die Linienbreite ist.
Die Optokoppler selbst bestehen jeweils aus einer Infrarot-LED, sowie einer infrarotempfindlichen Photodiode. Die Infrarot-LED hat die Funktion, den Untergrund mit Infrarotlicht zu beleuchten. Das ausgestrahlte Licht wird je nach Untergrund unterschiedlich stark reflektiert.
Je nach Stärke der Reflektion ändert die Photodiode ihren elektrischen Widerstand. Dieser kann dann gemessen und ausgewertet werden.
Auf diese Art und Weise kann ermittelt werden, ob der Roboter sich über einer der schwarzen Linien oder über dem weißen Untergrund befindet.

Des Weiteren befindet sich unter dem Roboter ein Fotowiderstand. Dieser ändert ebenfalls, wie der Name es schon vermuten lässt, seinen elektrischen Widerstand in Abhängigkeit vom einfallenden Licht. Diesen Fotowiderstand nutzen wir, um das Startsignal der Startleuchten zu erkennen und die Fahrt zu beginnen.
Zuletzt verfügt der Roboter noch über einen Greifer, mit welchen er die Fahrgäste aufsammelt.
Der Greifer ist ebenfalls aus Lego-Bausteinen aufgebaut.

Als Antrieb für diesen haben wir zuerst einen normalen Gleichstrommotor mit einer entsprechenden Untersetzung verwendet. Dieser konnte den Greifer zwar ohne Probleme schließen, jedoch war das Haltemoment dieses Motors im ausgeschalteten Zustand zu gering.
Dadurch öffnete sich der Greifer durch das Gewicht der Fahrgäste von selbst. Den Motor dauerhaft eingeschaltet zu lassen wäre zwar theoretisch möglich, erzeugt jedoch unnötig Wärme im Motor sowie der Ansteuerung und wirkt sich zusätzlich stark auf die Akkulaufzeit aus.
Also entschieden wir uns dazu, einen Servomotor, wie er im Modellbau üblich ist, zu verwenden.
Diese bestehen in der Regel ebenfalls aus einem Gleichstrommotor und einem Untersetzungsgetriebe. Es gibt bei diesen Motoren jedoch eine Besonderheit. Sie verfügen über einen eingebauten Mikrocontroller, welcher die aktuelle Position des Servomotors überwacht und ggf. anpasst. Dies funktioniert mit Hilfe eines Potentiometers, welches sich am Ausgang des Servomotors befindet.
Die Ansteuerung des Gleichstrommotors erfolgt meistens mit einem Vierquadrantensteller und einer Pulsweitenmodulation, welche von dem internen Mikrocontroller bereitgestellt wird.
Von außen erfolgt die Ansteuerung der meisten Servomotoren mit einer Pulsweitenmodulation. Die übliche Pulsbreite erstreckt sich dabei von ca. 500 Mikrosekunden bis 2500 Mikrosekunden.
Dabei steuert der Servomotor bei ca. 500 Mikrosekunden den Linksanschlag (0°) und bei ca. 2500 Mikrosekunden den Rechtsanschlag (180° oder mehr) an.
Um zu erkennen, wann sich der Greifer direkt über einen Fahrgast befindet, verwenden wir zwei Infrarot-LED´s, eine weitere infrarotempfindliche Photodiode und zusätzlich einen Taster, um eine Kollision mit der Wand im Fehlerfall zu vermeiden.
Im Normalfall reflektiert ein Fahrgast das ausgesendete Licht der Infrarot-LED´s, so dass dieses mit Hilfe der Photodiode gemessen werden kann. Ist dies der Fall, so wird der Greifer geschlossen. Sollte dies einmal nicht zuverlässig funktionieren, so berührt der Taster die Wand. Auch in diesem Fall wird der Greifer geschlossen und der Fahrgast gilt als eingesammelt.

Gesteuert wird der Roboter durch ein AKSEN-Board. Dabei handelt es sich um ein Mikrocontroller-Board, vergleichbar mit Arduino und Ähnliche. Bei dem darauf verbauten Mikrocontroller handelt es sich um einen SAF-C515-LN von Infineon. Genaue Details zum Aufbau (Schaltplan, verwendete Komponenten, technische Daten) kennen wir leider nicht.
Es verfügt über einige digitale und analoge Pins sowie Motortreiber und einen „Servoport“.
Hier sind alle von uns verwendeten Sensoren und Motoren zur Auswertung und Ansteuerung angeschlossen.

Die Software

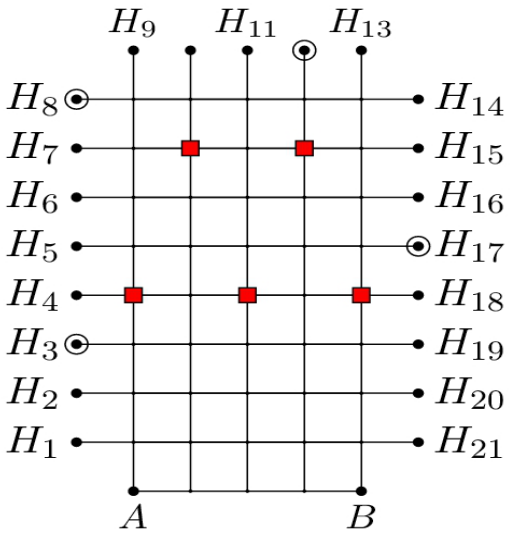
Allgemein:

Die Software wurde mit der Programmiersprache C entwickelt. Dabei haben wir unser Programm in viele kleine Funktionen zerlegt. Jede davon erfüllt einen Teil der Gesamtaufgabe. Einige Teilaufgaben sind z.B.: das Auslesen der Sensoren, dem Folgen einer Linie, das Greifen der Fahrgäste, das Planen der Fahrwege und einiges mehr.

Unser erstes Ziel war es, dass der Roboter in der Lage ist, einer Linie zu folgen. Das funktioniert eigentlich auch ganz einfach. Zuerst lesen wir die Optokoppler an der Vorderseite des Roboters aus. Wird dabei ein Schwellwert unterschritten, so kann davon ausgegangen werden, dass sich der Sensor über der Linie befindet. Ist dies der Fall, so wird der Motor auf der entgegengesetzten Seite verlangsamt. Dadurch wird der Roboter immer auf der Linie gehalten.
Als nächstes versuchten wir einer Linie solange zu folgen, bis wir auf eine Kreuzung gestoßen sind. Das war ebenfalls sehr einfach umzusetzen. Eine Kreuzung ist genau dann erreicht, wenn der ausgelesene Wert beider Sensoren unter den Schwellwert sinkt, sich also beide Sensoren auf einer schwarzen Linie befinden.
Danach implementierten wir weitere Fahrbefehle, wie z.B. das Abbiegen nach links oder rechts und das Umdrehen auf der Stelle.
Damit das Ganze nach einer Weile nicht zu unübersichtlich wird, haben wir alle Fahrfunktionen in einen eigenen Header ausgelagert. Dieses Prinzip haben wir bis zum Ende beibehalten und alle zusammengehörigen Funktionen in eine eigene Datei verschoben.

Pfadplanung:

Der Ausgangspunkt für die Pfadplanung unseres Roboters ist eine Karte. Diese Karte hat einen Startpunkt, Kreuzungen, Sperren und Haltestellen, an denen die Fahrgäste warten.
Diese sollen von den Haltestellen (Kreise) abgeholt und zum Startpunkt (A oder B) zurückgebracht werden.
Sperren (rot markiert) dürfen nicht befahren werden.



Um die Karte im Speicher des Roboters abzulegen, haben wir diese als Array vom Datentyp char aufgefasst.

Zuerst wurde dafür eine Kreuzung mit einem '.', eine Sperre mit einem 'X' und einen Fahrgast mit einem 'F' gekennzeichnet (wie auch in der Aufgabenstellung).

X	X	X	X	F	X	X
F	.	.	.	.	.	X
X	.	X	.	X	.	X
X	.	.	.	.	.	X
X	.	.	.	.	.	F
X	X	.	X	.	X	X
F	.	.	.	.	.	X
X	.	.	.	.	.	X
X	.	.	.	.	.	X
X	.	.	.	.	.	X

Nun brauchten wir noch einen Algorithmus, um den kürzesten Pfad zu den Fahrgästen zu berechnen, ohne eine Sperre zu befahren.

Ansatz 1, iterative Tiefensuche:

Die iterative Tiefensuche ist wie die normale Tiefensuche ein Suchverfahren für Graphen. Sie findet jedoch nicht irgendeinen, sondern immer den kürzesten Weg, wenn dieser existiert.

Um dies zu erreichen, wird eine normale Tiefensuche mit einer begrenzten Suchtiefe verwendet.
Diese wird in einer Schleife mit einer größer werdenden Suchtiefe aufgerufen. Wird die Suche mit einem positiven Ergebnis (Ziel gefunden) beendet, so wurde der kürzeste Weg gefunden.
Wurde die maximale Suchtiefe erreicht ohne dass das Ziel gefunden wurde, so ist die maximale Suchtiefe zu klein, oder es existiert kein Weg.
Eine endlose Rekursion durch Schleifen, wie sie bei der normalen Tiefensuche vorkommen kann, wird durch eine maximale Suchtiefe verhindert.

Schritte: 1.) Beginne mit der maximalen Suchtiefe von 1
2.) Führe eine Tiefensuche mit begrenzter Suchtiefe durch
3.) Erhöhe die maximale Suchtiefe um 1
4.) Wurde das Ziel gefunden, dann beende die Schleife, sonst gehe zu 2

Formal sieht dieses Verfahren so aus:

```
iterative_tiefensuche(startknoten, zielKnoten)
{
	tiefe = 1
	maxTiefe = ...
	while(tiefensuche(startKnote, zielKnoten, tiefe) == false)
	{
		if(tiefe < maxTiefe)
		{
			tiefe = tiefe + 1
		}
		else
		{
			kein Weg
		}
	}
}
```

Dieses Suchverfahren funktioniert zwar, ist jedoch für diese Aufgabe ungeeignet. Es benötigt sehr viel Rechenzeit und baut, wie die normale Tiefensuche, auf einer Rekursion auf. Diese kann auf einem Mikrocontroller sehr schnell zu einem Stecküberlauf (wenig Speicher) und damit zum Absturz oder ungewollten Verhalten führen.

Ansatz 2, Breitensuche:

Die Breitensuche ist ein Suchverfahren für Graphen, um einen Weg durch diesen zu finden.
Dabei wird immer der kürzeste Weg gefunden, wenn dieser existiert.
Im Gegensatz zur Tiefensuche, werden zuerst die direkt erreichbaren Knoten besucht.

Die Breitensuche benötigt einen Graphen, einen Startknoten, einen Zielknoten und eine Warteschlange.
Schritte: 1.) Füge den Startknoten zur Warteschlange hinzu
2.) Lese den nächsten Knoten von der Warteschlange
3.) Suche benachbarte Knoten (unbesuchte), füge sie zur Warteschlange hinzu und markiere sie als besucht
4.) Solange der Zielknoten nicht erreicht wurde und die Warteschlange nicht leer ist, springe zu Punkt 2

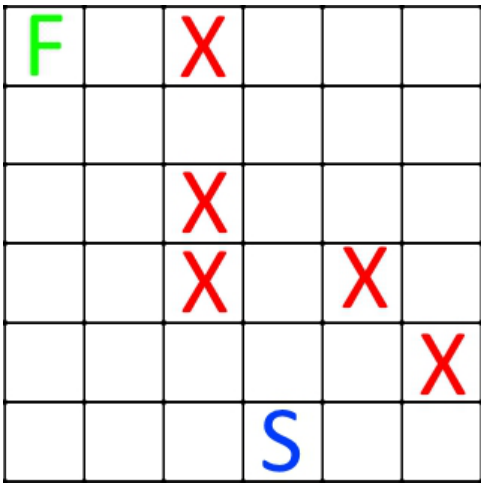
Formal sieht dieses Verfahren so aus:

```
breitensuche(startKnoten, zielKnoten)
{
	warteSchlange.hinzufügen(startKnoten)
	while(!warteSchlange.istLeer())
	{
		aktuellerKnoten = warteSchlange.nächsterKnoten()
		if(aktuellerKnoten.findeNachbarKnoten().enthält(zielKnoten)
		{
			return true;
		}
		warteSchlange.hinzufügen(aktuellerKnoten.findeNachbarKnoten())
		markiereKnoten(aktuellerKnoten.findeNachbarKnoten())
	}
}
```

Dieses Suchverfahren ist für die Aufgabenstellung gut geeignet. Es findet sehr schnell und direkt den kürzesten Weg zu einem Fahrgast, ohne eine Sperre zu betreten.
Das Suchverfahren muss allerdings mehrmals durchgeführt werden, um alle Fahrgäste zu finden.
Die Implementierung einer Warteschlange auf einem Mikrocontroller kann in einigen Fällen etwas schwierig sein.

Unsere finale Lösung, Fluten der Karte:

Unsere finale Lösung für das Problem der Pfadplanung basiert auf dem Verfahren der Breitensuche. Es wurde allerdings etwas abgeändert, um alle Fahrgäste mit einen Durchgang zu finden bzw. den Pfad zu ihnen zu markieren.
Dieser Algorithmus benötigt ebenfalls einen Graphen bzw. in diesem Fall unsere Karte, sowie ein Warteschlange.
In der Karte stellt das F einen Fahrgast dar, die X Hindernisse und das S das Startfeld.



Als Erstes weisen wir jedem befahrbaren Feld den wert ∞ zu.
Dann gehen wir zum Startfeld S und weisen diesem Feld den Wert 0 zu. Dieser Wert entspricht immer der Entfernung zum Startfeld.
Als nächstes suchen wird die Nachbarn von S, erhöhen unseren Abstand zum Startfeld um 1 und markieren alle gefunden Nachbarfelder mit diesen Wert. Alle gefunden Nachbarfelder fügen wir außerdem zur Warteschlange hinzu.
Nun holen wir uns das nächste Feld aus der Warteschlange, erhöhen unseren Abstand zum Startfeld wieder um 1, markieren alle Nachbarknoten mit diesen Wert und fügen sie zur Warteschlange hinzu. Dies wird solange wiederholt, bis die gesamt Karte mit den Entfernungswerten vom Startpunkt geflutet ist.

F	7	X	5	6	7
7	6	5	4	5	6
6	5	X	3	4	5
5	4	X	2	X	6
4	3	2	1	2	X
3	2	1	0	1	2

Wenn wir damit fertig sind, gehen wir zu unseren Fahrgast F. Von dort aus sehen wir uns die benachbarten Felder an und gehen immer zu dem Feld mit dem geringsten Wert.

Also: F->7->6->5.....->0 (Startfeld)
Wenn wir uns dabei für jeden Schritt die Richtung merken in welche wir gehen, so haben wir den Pfad von den Fahrgästen zum Startpunkt. Diesen brauchen wir nur noch umdrehen und rückwärts abfahren.

Bei der Implementierung dieses Algorithmus haben wird die Karte ebenfalls als Array dargestellt.
Hier allerdings mit dem Datentyp unsigned char. Außerdem stellen wir einen Fahrgast nun durch eine 254 und eine Sperre durch eine 255 dar.
Alle „leeren“ Felder werden am Anfang mit 253 initialisiert was ∞ darstellen soll.

Um beim Fluten der Karte einen Überlauf zu verhindern, sollte ein ausreichend großer Datentyp gewählt werden.
Für unsere Aufgabe mit einer 7x11 Karte reicht der Datentyp unsigned char aus.

Dieser Algorithmus ist für die Aufgabenstellung sehr gut geeignet. Er findet die kürzesten Wege zu allen erreichbaren Zielen und arbeitet effizient, da er bereits erlangtes Wissen wiederverwendet.

Nach dem wir unsere Implementierung noch etwas auf Geschwindigkeit optimiert hatten, erreichten wir Rechenzeiten von gerade einmal 2,6 Millisekunden zum Finden aller Wege.
Dabei wurde ein Arduino Uno mit 16MHz verwendet.

Verbesserungsvorschläge

Der erste Verbesserungsvorschlag wäre, die einzelnen Komponenten auf korrekte Funktionsweise zu prüfen. Bei uns ist es leider vorgekommen, dass einige nicht richtig funktioniert haben. Genauer war es eine lose Lötstelle an den Kontakten eines Optokopplers. Das haben wir jedoch erst später bemerkt und mussten den Roboter größtenteils zerlegen um diesen Optokoppler zu ersetzen. Des weiteren sind viele Getriebe in den Servomotoren bereits beschädigt und einige neue scheinen nicht kompatibel zu sein. Es würde die nächsten Studierenden sicherlich freuen, wenn an dieser Stelle für Nachschub gesorgt wird.

Das größte und zum Teil wirklich ärgerliche Problem waren die Akkus. Die meisten der Akkus verfügen scheinbar nur noch über eine geringe Kapazität und mussten oft schon nach zweimaligem Fahren erneut aufgeladen werden.
Des weiteren sind diese manchmal nicht mehr in der Lage, die etwas höheren Anlaufströme der Motoren zu liefern. Dies hat dann zur Folge, dass das aksen-Board wegen des Spannungseinbruchs abstürzt und neu startet. Manchmal passiert das auch relativ unauffällig während der Fahrt. Dann beginnt jedoch auch der Planungsalgorithmus erneut von vorn und der Roboter verhält sich entsprechend falsch.
Uns wurde zwar des öfteren gesagt, dass dies daran liegen würde, dass unsere Motoren zu viel Strom benötigen, jedoch lag der Strom beider Motoren zusammen im blockierten Zustand bei gerade einmal ca. 2A.
Diesen Strom können NiMh-Akkus aber problemlos und ohne großen Spannungseinbruch liefern, wenn diese in Ordnung sind.
Ein scheinbar relativ neues Akkupack hat dies auch auf dem Roboter gezeigt.

Abgesehen von diesen kleinen Problemen waren wir mit dem Projekt und der tollen Unterstützung von Prof. Dr.-Ing. Jochen Heinsohn und Dipl.-Inform. Ingo Boersch sehr zufrieden und hatten jeden Donnerstag viel Spaß.

---------- Vielen Dank für das tolle Projekt! ----------