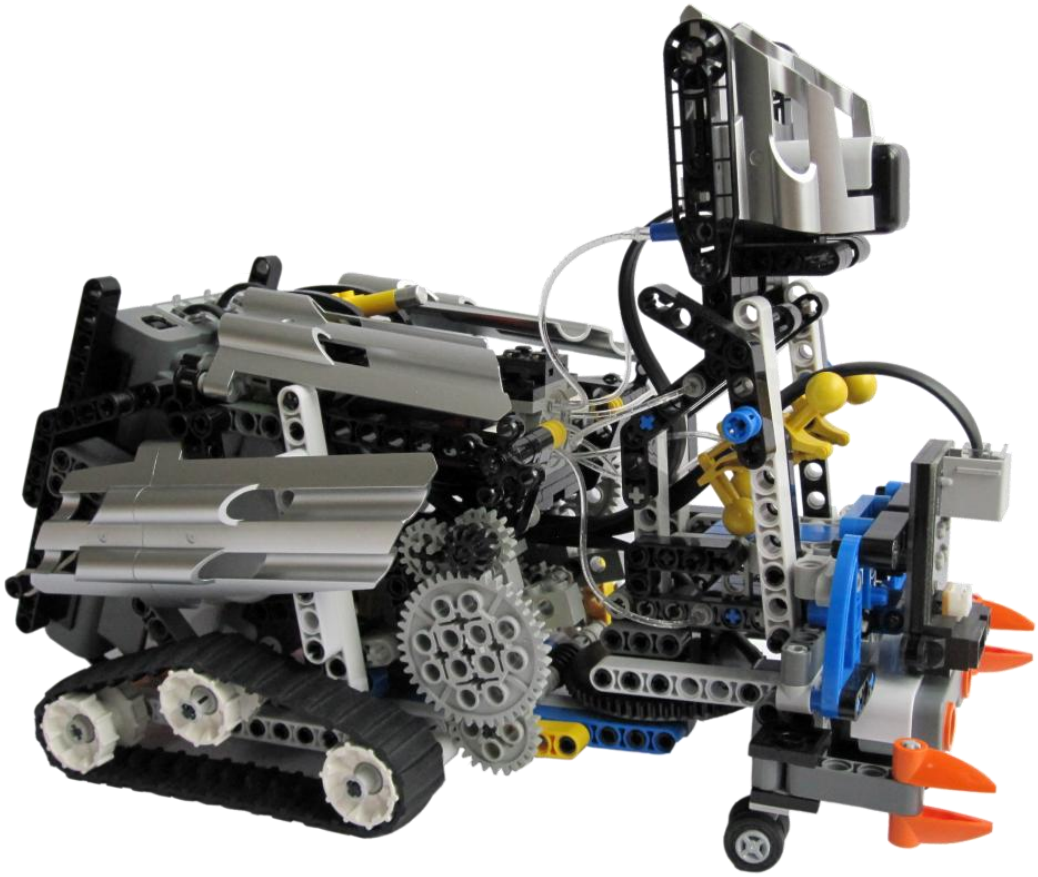
Kartografie mittels Robotik

Schuljahr 2009/2010

Kantonsschule Freudenberg



Maturitätsarbeit von Christian Tschudi

Betreuer: Herr Dr. Niklaus Emmenegger

Schulfach: Physik

Abgabedatum: 06.01.2010

Inhaltsverzeichnis

1 Vorwort 3
2 Einleitung 4
3 Hauptteil 5
3.1 Material und Methoden 5
3.1.1 Lego Mindstorms 5
3.1.2 Navigation 9
3.1.3 Erfassen von Objektpunkten 10
3.1.4 Übermittlung von Daten 13
3.1.5 Generierung der digitalen Karte 14
3.1.6 Programmierung 15
3.2 Resultate 16
3.2.1 Evolution des Roboters 16
3.2.2 Programm 39
3.2.3 Digitale Karte 47
3.3 Diskussion 52
4 Schlusswort 54
5 Abkürzungsverzeichnis und Glossar 55
6 Literatur- und Abbildungsverzeichnis 58
7 Anhang 62
7.1 Java 62
7.2 Programmcode 66
8 Anhang 2 156

1 Vorwort

Die Maturitätsarbeit ist die grosse Herausforderung für alle Gymnasiasten! Und der erste Schritt ist zugleich der Wichtigste: Die „richtige“ Themenwahl! Ich überlegte mir über einige Monate, mit welcher Thematik ich mich eingehend befassen wollte. In einem Punkt war ich mir allerdings schon von Anfang an sicher: Ich hatte im Sinn eine praktische Arbeit zu schreiben, denn ich wollte „Neues“ schaffen und nicht eine Literaturrecherche über Sachen, die andere schon vor mir geschrieben haben.

Lange Zeit dachte ich an ein Thema der Bewegungswissenschaft, beziehungsweise an ein Projekt über meine grosse Leidenschaft „Schwimmen“. Doch dies hätte eine grössere Abhängigkeit von externen Institutionen provoziert und ausser Interpretationen von Testresultaten hätte ich wohl nicht viel Nachhaltiges gelernt. Mein Ziel war es, wenn ich mich schon Stunden mit einer Arbeit befasse, etwas Neues und Fortwährendes zu lernen.

Ein weiteres Hobby, welches mich fasziniert, ist Programmierung. Sollte ich ein Programm entwerfen? Ja, aber wofür? Dann las ich einen Artikel über einen neuen japanischen Roboter, der verschiedenste Arbeiten erledigen konnte. Da ich sehr an Technik interessiert bin und ich mich über deren Fortschritt erfreue, war das die zündende Idee; einen Roboter zu bauen und ihn zu programmieren. Fehlte nur noch der Zweck des solchen. In der Geografie kam ich dann auf die Idee, dass das Kartografieren mit Robotern eine Menge mühsame Arbeit ersparen könnte. Somit war mein Ziel einen Roboter zu konstruieren und dann die Software zu entwickeln, die die Rohdaten in eine digitale Karte umwandelt. Dies ist zwar nicht vollkommen neu (US Army forscht beispielsweise schon lange an unbemannten Aufklärungsfahrzeugen), aber es ist dennoch eine grosse Herausforderung für mich, von der ich mir erhoffe, viel zu lernen.

2 Einleitung

In den letzten Jahrzehnten gab es enorme Fortschritte, was die Robotertechnik betrifft. Viele mühsame Arbeiten werden nun vollautomatisch von unscheinbaren Robotern erledigt. Hinter der genialen Technik steckt aber auch ein grosser Aufwand in der Entwicklung. Von der Idee des Roboters, über die Konstruktion des solchen, bis hin zum programmierten, vollfunktionsfähigen Roboter ist ein langer Weg zurückzulegen.

Das Ziel dieser Arbeit ist, einen Roboter zu bauen und zu programmieren, der ein unbekanntes Gebiet kartografiert. Der Roboter soll dabei dauernd in Kontakt mit dem Computer stehen und dessen Befehle ausführen. Dieser erstellt im Gegenzug die digitale Karte aus den Messpunkten und steuert den Roboter, auch ohne menschlichen Einfluss.

Aufgrund der verfügbaren Mittel und der Komplexität der Aufgabe, ist das Testgebiet beschränkt auf eine Fläche in der Grösse eines Zimmers und auf eine zweidimensionale Struktur.

Als Grundlage für den Roboter dient Legomindstorms. Dieser Roboterbausatz birgt viele Vorteile und wird deshalb auch von vielen technischen Hochschulen und Universitäten für Projektarbeiten verwendet. Für die Programmierung wird „LeJos“ und Java verwendet.

Der Aufbau der wissenschaftlichen Arbeit ist grob gegliedert in „Material und Methoden“, „Resultate“ und „Diskussion“. Im ersten Kapitel werden die theoretischen Grundlagen und Überlegungen geschildert. Im zweiten und zugleich längsten Kapitel werden die Resultate, chronologisch und thematisch geordnet, erläutert. Dabei wird der Fortschritt der Arbeit von den ersten Prototypen bis hin zum Endprodukt ersichtlich. Im letzen Teil „Diskussion“ wird dann das Erreichte interpretiert und mögliche Folgen abgeleitet.

3 Hauptteil

3.1 Material und Methoden

3.1.1 Lego Mindstorms[1]

3.1.1.1 Übersicht[2]

Lego Mindstorms ist eine Produktserie des Spielwarenherstellers Lego. Sie beinhaltet alles Mögliche, was man für den Roboterbau benötigt. Das Hauptprodukt ist der programmierbare Steuerblock NXT. Von ihm ausgehend können Motoren gesteuert und mit Sensoren gezielt Daten erfasst werden. Vorteilhaft ist vor allem, dass man es mit üblichen Legosteinen kombinieren kann und man so eine riesige Auswahl an Bauteilen hat. Der Kreativität sind also kaum Grenzen gesetzt.

3.1.1.2 NXT – das zentrale Element[3]

Der NXT kann via USB-Kabel oder Bluetooth programmiert werden. Er hat vier Sensorausgänge und drei Motorausgänge. Zudem verfügt er über einen integrierten Lautsprecher und einen LCD (liquid crystal display).

Abb. 1: NXT-Block, die zentrale Steuerung des Roboters

Technische Spezifikationen

- *Atmel-32-Bit-ARM-Prozessor, AT91SAM7S256; 256 kB Flash-Speicher, 64 KB RAM, 48 MHz*
- *Koprozessor: Atmel 8-Bit AVR, ATmega48; 4 KB Flash-Speicher, 512 Byte RAM, 8 MHz*
- *Bluetooth: CSR BlueCore 4 v2.0 +EDR; unterstützt das Serial Port Profile (SPP), 26 MHz*
- *USB-2.0-Anschluss, 12 Mbit/s*
- *ein Eingang kann als High-Speed-Port, entsprechend IEC 61158 Type 4/EN 50170, genutzt werden*
- *Punktmatrix LC-Anzeige; 100 × 64 Pixel, Abmessungen: 26 × 40,6 mm*
- *Soundausgabe mit 8-Bit-Auflösung und einer Samplingrate von 2 bis 16 kHz*
- *Firmware kann geändert werden*

[1] Offizielle Lego Website: http://mindstorms.lego.com/eng/Overview/default.aspx (Stand 30.11.2009).

[2] Legomindstorms: http://de.wikipedia.org/wiki/Lego_Mindstorms (Stand 30.11.2009).

[3] NXT: http://de.wikipedia.org/wiki/NXT (Stand 30.11.2009).

3.1.1.3 Motoren

Der in der NXT Grundausstattung enthaltene Servomotor (siehe Bild rechts) hat einen integrierten Drehsensor. Dieser hat eine Genauigkeit von einem Grad.

Abb. 2: NXT-Motor mit Drehsensor

Mit Hilfe eines Adapterkabels kann man auch ältere Legomotoren betreiben, diese haben aber den Nachteil, dass sie über keinen Rotationssensor verfügen und die Genauigkeit deshalb wesentlich geringer ist.

Desweiteren ist es auch möglich, mehrere Motoren über einen Anschluss (Port) zu steuern, sofern diese immer simultan drehen sollen. Dafür wird ein Motor-Multiplexer gebraucht und je nach Leistung auch eine externe Stromquelle.

3.1.1.4 Sensoren[4]

Es gibt unzählige Sensortypen welche für den NXT geeignet sind. Es wäre utopisch eine Liste als vollständig zu deklarieren, da es unzählige Anbieter gibt und auch Eigenproduktionen möglich sind.

3.1.1.4.1 „LEGO certified NXT Sensors“

Die in diesem Kapitel erwähnten Sensoren sind von der Firma Hi-Technic und haben das Zertifikat, die offiziellen Legosensoren zu sein. Somit haben sie den Vorteil, dass sie auch das Legogehäuse benützen dürfen. Die alten Lego Mindstorms Sensoren werden nicht aufgezählt, obwohl sie mit Hilfe eines Adapterkabels auch gebraucht werden können.

Abb. 3: NXT-Temperatursensor

Der Temperatursensor gibt die Temperatur von -20° bis + 120° Celsius an. Je nach Bedarf, kann er auch auf Fahrenheit umgestellt werden.

[4] Sensoren: http://shop.educatec.ch/legoeducationaldivision/legomindstormseducation/legosensoren/index.php (30.11.2009).

Abb. 4: Weitere NXT-Sensoren

Der Tastsensor ist der am häufigsten gebrauchte NXT-Sensor. Er kann für allerlei Anwendungen gebraucht werden. Hauptsächlich wird er als Navigationshilfe eingesetzt, um Hindernisse einfach erkennen zu können. Mit Hilfe eines Verteilers können auch mehrere Tastsensoren an einem Port angeschlossen werden.

Der Lichtsensor nimmt die Lichtintensität war und kann so zwischen hell und dunkel unterscheiden. Beschränkt kann er so auch Farben unterscheiden, weil sie eine andere Lichtintensität aufweisen.

Der Soundsensor kann die Laustärke der Umgebungsgeräusche in Dezibel angeben. Ausserdem kann er Veränderungen wahrnehmen. Auf diese Weise kann er auch bestimmte Töne/Muster erkennen und dementsprechend darauf reagieren.

Der Ultraschallsensor kann sowohl als Bewegungsmelder, als auch als Distanzmesssensor gebraucht werden. Er kann die Distanz bis zu Objekten, die weniger als 255 Zentimetern entfernt sind, angeben.

Der digitale Kompasssensor misst das magnetische Erdfeld und gibt dann die Ausrichtung an. Der Wert wird auf ein Grad gerundet und dann als Zahlwert zwischen 0 und 359 zurückgegeben. Mit Hilfe dieses Sensors ist eine viel präzisere Navigation möglich.

Der Beschleunigungssensor misst die Beschleunigung im Bereich -2g bis +2g, aufgeteilt in die Richtung der drei Koordinatenachsen. Zudem kann er auch die Neigung feststellen und somit weiss der Roboter immer, wo oben ist.

Abb. 5: NXT-Farbsensor

Der Farbsensor wurde speziell dafür entwickelt, dass der Roboter die Farben besser erkennen kann (als mit Hilfe des Lichtsensors). Dadurch kann der Roboter Gegenstände sortieren oder den Umständen gerecht reagieren.

Abb. 6: NXT-RFID Sensor

Der RFID Sensor erleichtert das Erkennen von erfassten Objekten, in dem er den auf den Gegenständen befestigten Transponder ausliest. Man kann dieses System auch dazu verwenden, dass der Roboter nur dann einen Code ausführt, wenn man den spezifischen Transpondern in die Nähe des Sensors hält.

Es gibt noch einige weitere offizielle Sensoren und es werden auch laufend neue entwickelt. Man findet sie im Internet unter: „http://www.hitechnic.com“

3.1.1.4.2 „Mindsensors“

Auch eine beliebte Adresse für Sensoren, die man mit dem NXT kombinieren kann ist: „http://www.mindsensors.com“. Dort findet man Bauteile, nützliche Tipps und Links, um Sensoren nach den eigenen Bedürfnissen kreieren zu können. Es gibt aber auch eine Produktpalette mit fertigen Sensoren. Eine kleine Auswahl wird hier kurz vorgestellt:

Abb. 7: Kamera

Die kleine Kamera kann dazu verwendet werden, bis zu acht Objekte oder Linien gleichzeitig zu erkennen und zu verfolgen (Objekte werden durch verschiedene Farben definiert). Sie hat eine Aktualisierungsrate von 30 fps („frames per second“) und der Roboter kann so extrem schnell reagieren. Die Kamera hat aber einen stolzen Preis und die Programmierung einer Bildanalyse ist ziemlich komplex!

Abb. 8: IR-Sensor

Es gibt verschiedene Arten von Infrarotsensoren. Sie messen die Distanz zu Objekten, die in ihrer Nähe liegen. Der rechts abgebildete Sensor misst den Abstand zu Gegenständen, die zwischen zehn und 80 Zentimetern entfernt sind, im Millimeterbereich. Er ist ziemlich klein und kann mit Hilfe eines offiziellen Lego Kabels mit dem NXT verbunden werden.

Abb. 9: NXT-LineLeader

Der „NXTLineLeader“ erfasst Linien effizient mit Hilfe von acht unabhängigen Sensoren und der Roboter kann dadurch Linien mit einem rasanten Tempo entlang fahren.

Wenn man den NXT mit Pneumatik kombiniert (es gibt pneumatik Teile von „Lego Technics“), so ist dieser Sensor sehr hilfreich, da er den momentanen Druck angibt.

Abb. 10: Drucksensor

3.1.1.4.3 Allgemein

Es gibt noch unzählige weitere Sensoren, die zum Beispiel folgende Attribute messen: pH-Wert, Luftfeuchtigkeit, Luftdruck, …
Und sollte der gesuchte Sensor auch im riesigen Internet nirgends gefunden werden, so findet man dort immerhin zahlreiche „Bastelanleitungen“, wie man eigene Sensoren kreiert.

3.1.2 Navigation

Die präzise Navigation ist die absolute Grundvoraussetzung, um nachher eine mehr oder weniger genaue digitale Karte zu erzielen. Denn wenn die Position des Roboters nicht stimmt, sind auch die Koordinatenpunkte der Objekte ungenau und das Resultat ist dementsprechend schlecht.

Kleinste Abweichungen in der Fahrtrichtung resultieren in einer ziemlich grossen Ungenauigkeit, denn am Ende eines Weges summieren sich die Fehler auf und die Position stimmt überhaupt nicht mehr.

Das Ziel ist also, dass die Position des Roboters (temporärer digitaler Koordinatenpunkt) exakt mit der Realität übereinstimmt.

Hierfür gibt es mehrere Möglichkeiten:

- Die Motoren haben einen Drehsensor eingebaut und sie registrieren auf ein Grad genau, wie viele Umdrehungen der Motor geleistet hat. Umgerechnet auf den Raddurchmesser/Achsendurchmesser kann so bestimmt werden, wie weit der Roboter gefahren ist. Damit ergibt sich die Möglichkeit die gefahrene Distanz in x-Richtung/y-Richtung aufzuzeichnen und so die Position zu bestimmen.
- Mit Hilfe eines Beschleunigungssensors lässt sich die Lage des Roboters auf ähnliche Art und Weise bestimmen, wie im vorherigen Beispiel mit den Motoren. Mit Hilfe des Integrals der Beschleunigung kommt man auf die Geschwindigkeit. Mit einem weiteren Integral über die Geschwindigkeit kann man schliesslich den exakt zurückgelegten Weg in X- und Y-Richtung berechnen.
- Eine zusätzliche Option wäre eine Art „Messstange“, welche der Roboter am Kordinaten-Ursprung (0/0) deponiert. Dadurch kann der Roboter seinen Abstand zum Punkt (0/0) abmessen. Problematisch daran ist, dass auf diese Weise ein Kreis entsteht, auf welchem sich der Roboter befinden könnte. Verbessern liesse sich diese Methode durch eine weitere „Messstange“→zwei Schnittpunkte oder durch drei Fixpunkte→ein Knotenpunkt (Triangulation)
- Eine Unterstützung dazu liefert der Kompasssensor, der die Himmelsrichtung extrem genau ermitteln kann. Somit kann man die Richtung der Fahrt überprüfen.

Keine der oben genannten Methoden gibt eine genügend verlässliche Sicherheit. Darum wird die Lösung für eine genaue Position des Roboters eine Kombination sein und eine daraus resultierende Fehlerrechnung liefert die Verlässlichkeit des Resultates.

Natürlich muss der Roboter auch Hindernissen ausweichen können. Die simpelste Handhabung wäre mit vier Tastsensoren gegeben. Auf jeder Seite zwei Tastsensoren,

wovon einer vorne und einer hinten befestigt ist. Sobald der Roboter einen Gegenstand touchiert stoppt er und beginnt ein Ausweichmanöver. Der Vorteil an diesem System ist sicherlich die kleine Fehlerquote, da die Tastsensoren sehr konstant und sensibel funktionieren. Im Gegenzug ist diese Methode nicht sehr elegant, weil der Roboter unweigerlich und immer wieder Objekte trifft. Vor allem weiche Objekte werden zu spät erkannt und könnten deshalb verschoben werden.

Es muss also noch geschicktere Methoden geben und wenn möglich eine, die nicht unnötigen mehr Aufwand bringt. Also soll die Funktion gegebene Sachen verwenden, um eine Kollision zu verhindern. Für die Erstellung der digitalen Karte sendet der Roboter die Rohdaten (Objektpunkte) an den Computer und dieser (beziehungsweise die entwickelte Software) erstellt dann die digitale Karte. Logischerweise bewegt sich der Roboter auch auf dieser Karte (Koordinatensystem). Wenn das Generieren der Karte in Echtzeit geschieht, kann die Position des Roboters mit derjenigen der Hindernisse verglichen werden. Und da der Computer den Befehl sendet, wohin der Roboter als nächstes fahren soll, könnte er zuerst überprüfen, ob die Position schon von Objektpunkten belegt ist. Mit einem geschickten Algorithmus kann die Software also erkennen, ob der Roboter sich auf Kollisionskurs befindet und ihm somit eine andere, geeignete Position schicken.

3.1.3 Erfassen von Objektpunkten

Die eigentliche Aufgabe des Roboters ist die Erfassung der Objekte. Deshalb bedarf dieser Vorgang einer sorgfältigen Planung und Umsetzung. Denn es sollte am Schluss eine möglichst gute Qualität der Karte vorhanden sein und gleichzeitig sollte der Vorgang auch nicht allzu lange dauern.

3.1.3.1 Die Wahl des richtigen Sensors

3.1.3.1.1 Tastsensor

Mit Hilfe von Tastsensoren kann man auf einfachste Art und Weise einen (1) Objektpunkt erhalten. Die Funktionsweise ist simpel: Der Tastsensor unterscheidet zwischen zwei Zuständen (O=keine Berührung, 1=gedrückt). Folglich generiert der Roboter mit Hilfe eines Tastsensors immer dann einen Objektpunkt, wenn er in ein Hindernis fährt. Diese Methode ist aber weder elegant noch effizient. Einerseits berührt der Roboter die Objekte, was eine geringe Verschiebung der Gegenstände verursachen könnte, andererseits würde die Position des Roboters ungenau, da die auftretenden Reaktionskräfte schwer abzuschätzen sind (wird der Roboter sofort gestoppt, oder verschiebt er das Objekt, bzw. sich selber nach der Berührung noch ein bisschen?). Des weiteren dauert der Vorgang auf diese Weise ziemlich lange, da der Roboter jedes Hindernis einzeln abtasten muss.

Abb. 11: NXT-Tastsensor

3.1.3.1.2 Ultraschallsensor [5,6]

Weitaus bessere Resultate wird man mit einem Ultraschallsensor erreichen.

Das Funktionsprinzip:
Ein spezieller Lautsprecher sendet zuerst eine Ultraschallwelle aus, welche sich kegelförmig ausbreitet. Wenn diese senkrecht auf einen Gegenstand auftrifft, so wird sie genau Richtung Ursprung reflektiert. Schliesslich nimmt ein Mikrofon am Ultraschallsensor das Echo wahr. Aus der für diesen Vorgang gebrauchten Zeit kann die Distanz bestimmt werden:

$$s = \frac{c \cdot t}{2} \quad | \, c = Schallgeschwindigkeit\ im\ spezifischen\ Medium$$

In der Luft beträgt die Schallgeschwindigkeit c unter Normbedingungen 343 $^{m}/_{s}$.

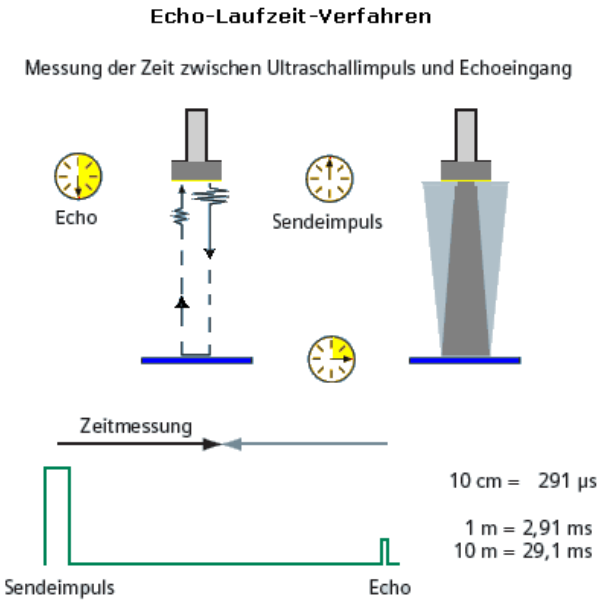


Abb. 12: Prinzip des Ultraschallsensors

Der Ultraschallsensor hat aber auch seine Tücken:

Ein erstes Problem verursacht die Richtungsunsicherheit. Der Sensor kann zwar die Distanz messen, aber da sich die Schallwellen kegelförmig ausbreiten, kann man nicht genau bestimmen von wo das Echo kommt. Dieses Problem kann aber mit Hilfe einer guten Bauweise des Sensors drastisch minimiert werden.

Umwelteinflüsse, wie Temperaturschwankungen (die oben erwähnte Schallgeschwindigkeit ist Temperatur abhängig) und Turbulenzen haben einen gewissen Einfluss auf die Ergebnisse. Doch wird diese Problematik nur einen marginalen Einfluss auf das Resultat der Arbeit haben, da die Versuchsbedingungen relativ konstante Umweltverhältnisse bieten und diese Abweichungen auf kleine Distanzen ohnehin praktisch keinen Einfluss haben.

Scheinechos können die Resultate viel eher beeinflussen. Sie entstehen unter anderem dann, wenn ein Schall zuerst an mehreren Objekten reflektiert wird, bevor das Echo gemessen wird (folglich wird eine viel zu grosse Distanz gemessen). Eine andere Ursache ist, wenn die Abstände der Messungen zu gering sind. Dann kann es vorkommen, dass ein Echo von einem weit entfernten Gegenstand während der nächsten Messung registriert wird (ergo; die gemessene Distanz ist viel zu klein). Dasselbe kann passieren wenn mehrere Ultraschallsensoren simultan eingesetzt werden.

---

[5] Ultraschall: http://de.wikipedia.org/wiki/Ultraschall (Stand 30.11.2009).

[6] Ultraschall: http://projektlabor.ee.tu-berlin.de/projekte/roboter/downloads/referate/ultraschall/referat.pdf (Stand 30.11.2009).

Das exakte Bestimmen, wann das Echo zurückkommt ist nicht ganz trivial. Der Schall ist verhältnismässig schnell und somit entscheiden Sekundenbruchteile (bei Normbedingungen: 1cm / 0.000058s) über grosse Distanzen. Die logische Konsequenz ist eine Streuung der Messresultate um den wahren Wert. Dieser Faktor lässt sich nicht direkt verringern, doch die Folge davon ist relativ klein, wenn der Ultraschallsensor gut geeicht ist.

Das grösste Problem für diese Arbeit, ist die variable Intensität der Echos. Diese hängt stark von der Form des Objektes ab. Ein Würfel reflektiert den Schall beispielsweise viel stärker als eine Kugel, bei welcher der Schall in alle Richtungen abgelenkt wird. Noch extremer ist der Unterschied zwischen einem festen Gegenstand (z.B. aus Holz) und einem Teddybären (flauschiges Material). Bei diesem wird der Schall in alle Richtungen gestreut, beziehungsweise der Schall wird vom Fell des Teddybärs absorbiert. Somit ist das Echo wesentlich schwächer und das Plüschtier ist sehr wahrscheinlich „unsichtbar" für den Sensor.

Der Ultraschallsensor von Lego kann laut Datenblatt Objekte zwischen 0 und 255cm erkennen können. Aus der Praxis kann man aber sagen, dass er nur zwischen 10cm und circa 70cm gute Ergebnisse liefert (natürlich stark von den oben genannten Faktoren abhängig!)

Abb. 13: NXT-Ultraschallsensor

3.1.3.1.3 Infrarot Sensor[7]

Das Prinzip eines Infrarotsensors hat Ähnlichkeiten mit demjenigen des Ultraschallsensors; eine Lichtquelle erzeugt Infrarotstrahlung und ein Sensor registriert die Reflexion. Die Funktionsweise ist aber wesentlich komplexer und von Modell zu Modell ziemlich variabel. Komplexer unter anderem deshalb, weil Licht sich etwa eine Million Mal schneller ausbreitet als der Schall und so eine noch viel genauere Zeitmessung erfolgen muss.

Hier spielt die Oberflächenstruktur der Objekte eine wichtige Rolle. Vor allem die Helligkeit (Farbton) und die Temperatur können die Messungen beeinflussen.

Die Infrarotsensoren von mindsensors.com können laut Datenblatt die Distanz auf wenige Millimeter genau angeben können. Die Kalibration ist schon erfolgt, aber der Sensor kann bei Bedarf neu geeicht werden.

Abb. 14: IR-Sensor

Im Sensor integriert ist eine Batterie, um genügend Leistung zur Verfügung zu haben (die maximale Leistung liegt bei $P = 0.03A \cdot 4.7\,V = 0.141W$).

Die Verbindung mit dem NXT-Block kann über eine digitale I2C bus Schnittstelle hergestellt werden.

[7] Ultraschallsensor & Infrarotsensor: http://www.mikrocontroller.net/mc-project/Pages/Robotik/Sensoren/sensoren.html (Stand 30.11.2009).

Es gibt drei verschiedene Variationen:

- *Long Range Infrared distance sensor*; 20cm - 150cm
- *Medium Range Infrared distance sensor*; 10cm - 80cm
- *Short Range Infrared distance sensor*; 4cm - 30cm

3.1.3.2 Die Vorgehensweise

Das primäre Ziel beim Erstellen der Karte ist, so viele Fehlerquellen wie möglich zu vermeiden. Daher erfolgen die Distanzmessungen nur dann, wenn der Roboter in Ruhe ist. Auf diese Weise werden die Resultate qualitativ besser. Denn wenn sich der Roboter bewegt, muss der Distanzpunkt immer im Verhältnis zur gefahrenen Distanz ausgedrückt werden. Die Bildpunkte könnten auch verzogen sein, da der Schall einen weiteren Weg zurücklegt, beziehungsweise nicht direkt zum Sensor zurück kommt (Doppler-Effekt).

Andererseits sollen die Messungen auch in einer möglichst geringen Zeit erfolgen. Damit der Roboter an Ort und Stelle stehen kann und trotzdem mehrere verschiedene Messungen durchgeführt werden können, wird eine Art Drehplattform gebaut. Mit dessen Hilfe kann er alle Objekte die zwischen +50° und -50° (0°:= vorne) liegen aufspüren. Der Sensor wird zudem so tief wie möglich montiert, damit er auch kleine Gegenstände erfassen kann.

Damit die eigentlichen Messungen möglichst genau sind, müssen, abgesehen von der Position des Roboters, zwei Parameter korrekt sein: Erstens muss die Distanzangabe möglichst genau bekannt sein und zweitens muss die Richtung in welcher der Messpunkt liegt stimmen. Der zweite Faktor kann wesentlich beeinflusst werden. Eine sehr grosse Übersetzung vom Motor auf die Drehplattform, resultiert in einer viel genauer einstellbaren Richtung (der Motor muss circa 10s volle Leistung liefern, um die Plattform um 180° zu drehen!). Um auch hier das Risiko von systematischen Fehlern zu minimieren, wird der Kompasssensor auch auf die Drehplattform gebaut. Dieser misst, wenn der Roboter stillsteht die Richtung der Distanzpunkte (auf 1° genau!). Bevor der Roboter weiterfährt, wird der ganze Drehteller wieder auf 0° ausgerichtet und der Kompasssensor dient wieder zur Navigation.

3.1.4 Übermittlung von Daten

Für die Interaktion zwischen Computer und NXT wird Bluetooth verwendet. Diese Technologie erleichtert den Umgang wesentlich, weil kein störendes Kabel gebraucht wird. Dieses könnte den Roboter bei der Navigation bremsen und somit dessen Position verfälschen.

Vom Computer wird als erstes der Programmcode auf den Roboter übertragen, sofern eine neuere Version zur Verfügung steht. Dieser meldet darauf, ob dieser Vorgang erfolgreich war oder nicht. Danach wird das Programm vom Computer gestartet.

Im Betrieb schickt der Roboter dem Computer die Rohdaten (Koordinatenpunkte) zur weiteren Verarbeitung. Dieser berechnet darauf die digitale Karte und sendet dem Roboter Befehle, wie er fortfahren soll, so dass er keine Hindernisse trifft und das Gebiet trotzdem möglichst schnell und komplett scannt.

Allgemein ist das Ziel möglichst viel „Outsourcing“ zu betreiben (vom Roboter aus gesehen) um eine gute Performance zu erreichen. Der Micro-Prozessor im Roboter ist im Vergleich zu einem modernen CPU, wie dem „Intel- Core 2 Duo E6600 Prozessor“ wesentlich langsamer und deshalb ist es sinnvoll, die Daten via Bluetooth auf den Computer zu übertragen, dort zu rechnen und die daraus resultierenden Befehle an den Roboter zurückzuschicken.

3.1.5 Generierung der digitalen Karte

Das Prinzip ist einfach, die Umsetzung dafür umso schwieriger: Der Roboter sendet seine Position und die Messpunkte an den Computer. Dieser bildet mit diesen Fakten ein Koordinatensystem, welches sich stetig füllt. Das Zwischenresultat ist eine Menge von Koordinatenpunkten. Danach kann mit diesen eine digitale Karte (grafisches Koordinatensystem) erstellt werden. Bei einer ausreichenden Anzahl an Messungen sollte auf einer solchen Grafik das Gebiet schon ziemlich gut sichtbar sein (Gegenstände bestehen aus Punktewolken). Um die Angelegenheit zu verbessern muss eine Fehlerrechnung erfolgen. Eine Interpolation der Messpunkte führt beispielsweise zu einer geraden Kante (anstelle von Punkten, die in deren Nähe liegen). Die so erreichte Approximation wiederspiegelt die Realität einiges besser, als die Punktewolken. Allerdings dürfte dieser Vorgang ziemlich schwierig zu programmieren sein!

Es muss hier allerdings gesagt werden, dass die Qualität der Karte sehr stark von der Messgenauigkeit der Sensoren abhängig ist und auch eine gute Fehlerrechnung nicht alles „retten“ kann.

3.1.6 Programmierung

3.1.6.1 „LeJos“ [8]

LeJos wurde speziell für Lego Mindstorms entwickelt. Nach dem die Firmware des Roboters (Standard Lego Mindstorms) mit derjenigen von LeJos überschrieben wurde, kann man den Roboter mit Java[9] programmieren. Dabei ist die Firmware von LeJos vergleichbar mit einer Java-VM.

Auf der Entwickler Homepage[10] kann auch eine ganze Bibliothek heruntergeladen werden, welche alle wichtigen Befehle zur Steuerung des Roboters enthält. Dies beinhaltet auch die Eichung für die gängigsten Sensoren (welcher Wert entspricht welchem Widerstand).

[8] LeJos: http://de.wikipedia.org/wiki/LeJOS (Stand 30.11.2009).

[9] siehe Anhang: Java

[10] http://lejos.sourceforge.net/

3.2 Resultate

3.2.1 Evolution des Roboters

3.2.1.1 Prototyp I

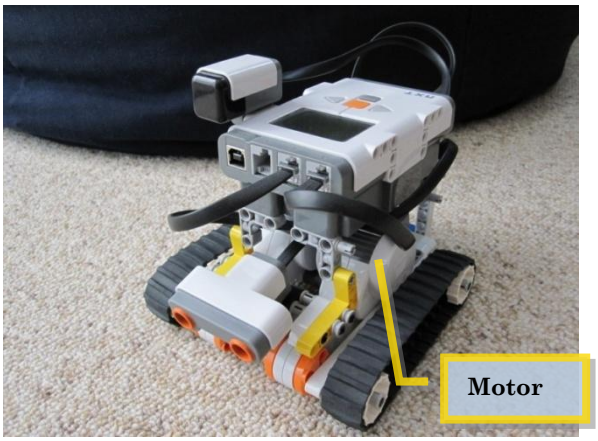


Abb. 15: Prototyp I

Als erster Prototyp diente eine möglichst einfache Konstruktion. Dabei ging es vor allem darum, das neue „Betriebssystem“ LeJos zu testen. Die Tests waren zufriedenstellend, da alle Funktionen einwandfrei funktionierten. Es gelang auch eine Bluetooth Verbindung aufzubauen, sodass nicht ein störendes USB-Kabel benötigt wurde.

Der Roboter konnte eine bestimmte Distanz mehr oder weniger genau zurücklegen. Ein Problem waren die Raupen, welche noch nicht die ideale Straffheit hatten.
Mit Hilfe des Kompasssensors war es möglich seine Ausrichtung zu ändern, beziehungsweise in eine bestimmte Himmelsrichtung zu navigieren.

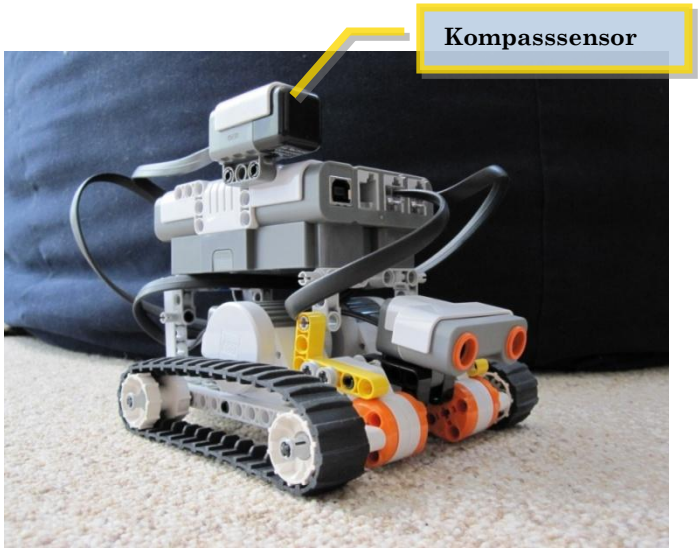


Abb. 16: Prototyp I

Abb. 17: Prototyp I

Mit dem Ultraschallsensor konnten einzelne Distanzmessungen durchgeführt werden, welche +/- 5cm genau waren. Diese doch ziemlich groben Messungen lagen aber auch daran, dass der Sensor nicht perfekt horizontal ausgerichtet war.

3.2.1.2 Prototyp II α

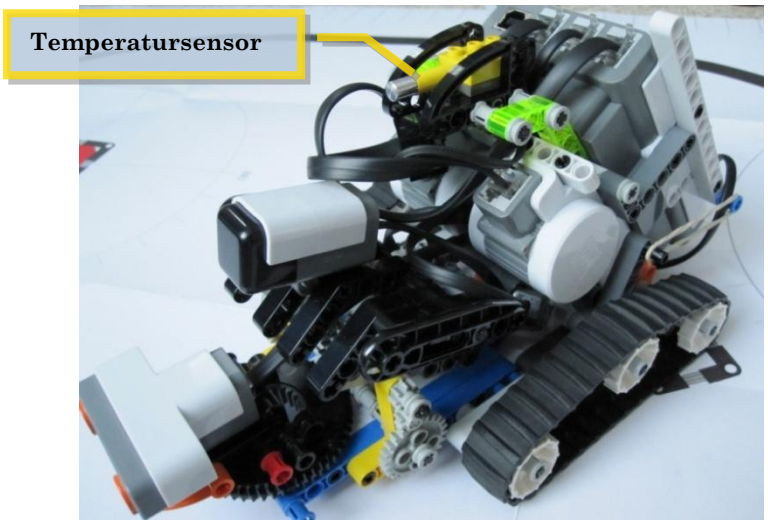


Abb. 18: Prototyp II a

Im zweiten Prototypen wurden zahlreiche Verbesserungen implementiert und der Aufbau wurde dadurch schon wesentlich kompakter.

Neu war der Temperatursensor, der erkennen sollte, wenn sich das Raupenfahrzeug einer Wärmequelle näherte.

Neu war auch die Raupenkonstruktion: Mit dem dritten Rad war die Raupe straffer und der Antrieb somit effizienter. Für genügend Elastizität sorgte das Gummiband, welches das dritte Rad mit Hilfe der Hebelkraft nach oben drückte. Durch die ausgleichende Wirkung der Raupe veränderte sich die horizontale Lage des Roboters weniger, sollte er über ein kleines Hindernis fahren.

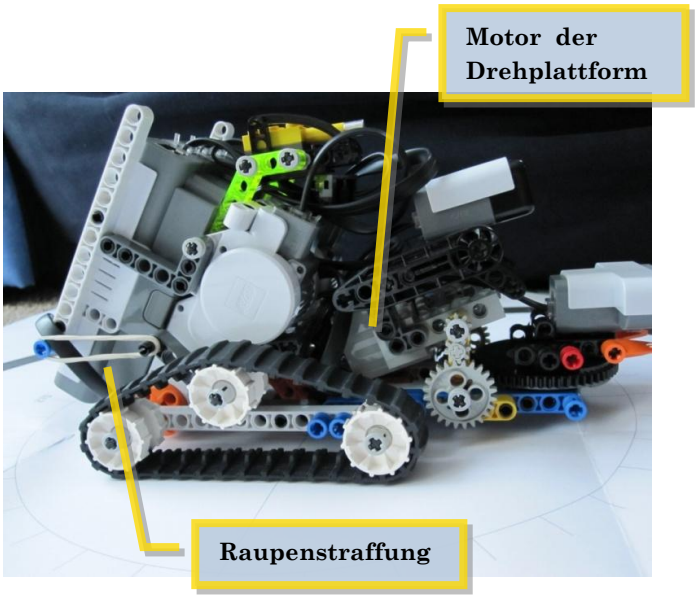


Abb. 18: Prototyp II a

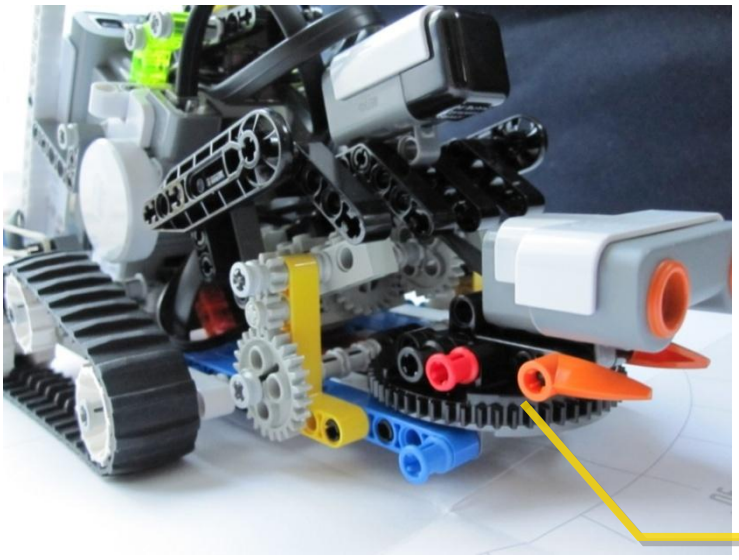


Abb. 19: Prototyp II a

Die grösste Neuerung war die Drehplattform an der Vorderseite. Der Motor trieb den schwarzen Drehteller mit einer grossen Übersetzung an. Somit brauchte er etwa vier Sekunden um den Ultraschallsensor um 90° zu drehen.

Erste Navigationstests sollten zeigen, ob der Antrieb einwandfrei funktionierte. Leider war dies nicht der Fall und die Fahrt ging nach weniger als einem halben Meter zu Ende. Die Raupenaufhängung musste daraufhin wesentlich stärker befestigt werden, da sie den Belastungen nicht stand hielt. Mit Hilfe eines neuen Unterbodens und ein paar zusätzlichen Verstrebungen war das Problem behoben.

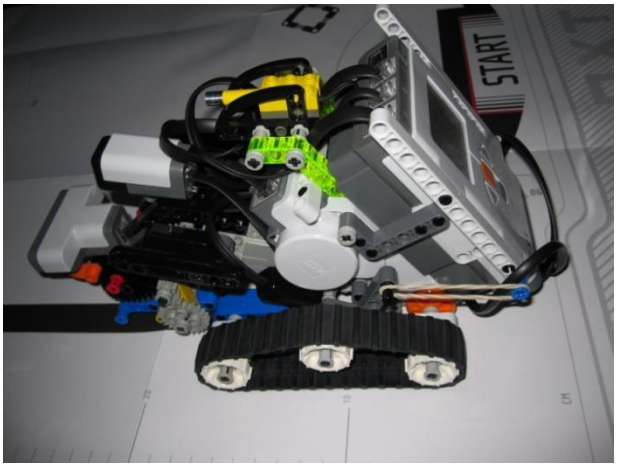


Abb. 20: Prototyp II a

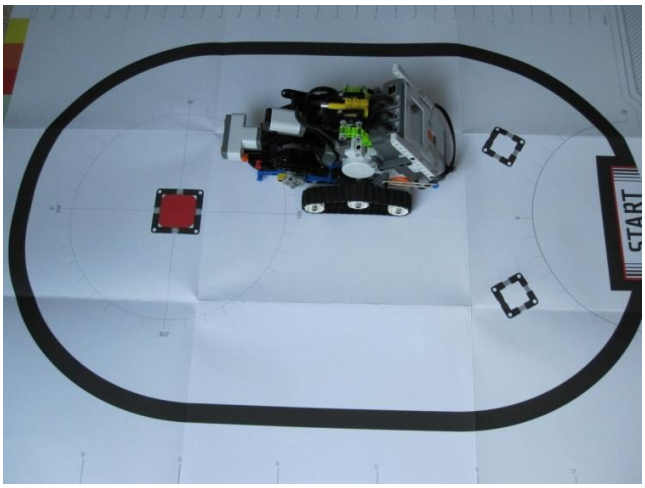


Abb. 21: Prototyp II a

Auf diesem Bild sind die Dimensionen des Roboters gut erkennbar. Seine Länge betrug 25 Zentimeter, seine Breite 14 Zentimeter und seine maximale Höhe 13 Zentimeter.

Die Unterlage diente mit ihren vielen Markierungen als ideales Testgebiet.

Die Rotationseigenschaften des Roboters waren noch nicht optimal. Auf dem Bild rechts ist schnell zu erkennen, dass der Schwerpunkt, aufgrund des NXT-Blockes und der beiden Antriebsmotoren, ziemlich weit hinten lag.

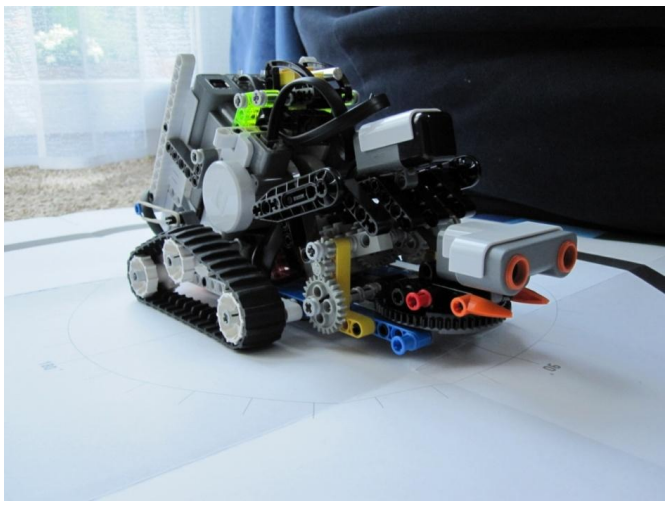


Abb. 22: Prototyp II a

3.2.1.3 Prototyp II β

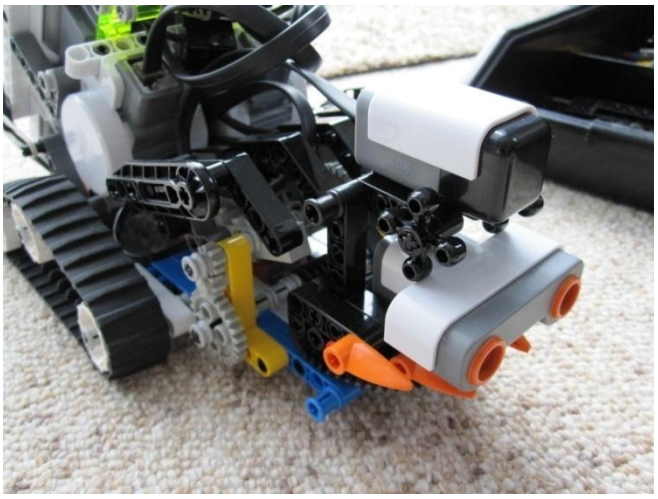


Abb. 23: Prototyp II β

Hauptunterschied zum Prototyp II α war, dass der Kompasssensor nun auch auf der Drehplattform montiert war. Es war gedacht, dass der Winkel der Drehplattform mit Zeitintervallen und einer bestimmten Geschwindigkeit des Elektromotors berechnet würde, dies deshalb weil dieser Motor, im Gegensatz zu den Antriebsmotoren keine integrierten Drehzahlmesser hatte. Doch bei ersten Tests stellte sich heraus, dass der Kompasssensor die Ausrichtung viel genauer bestimmte und deshalb nicht als Quelle für Fehlerrechnungen diente, sondern als Hauptmessinstrument.

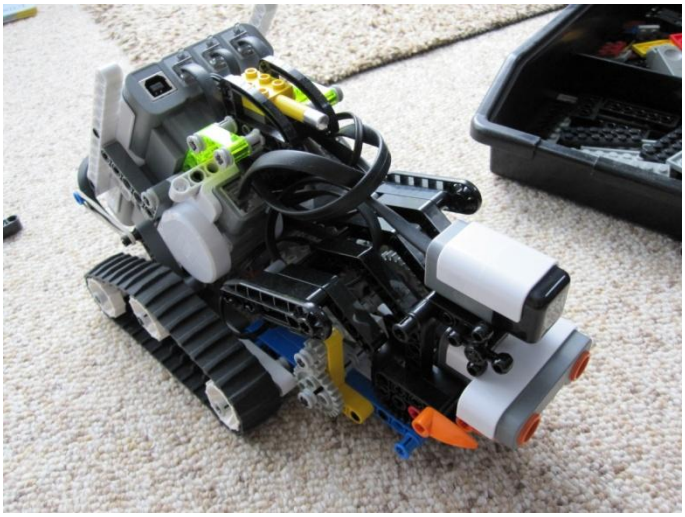


Abb. 24: Prototyp II β

Im Innern des Roboters wurden einige unnötige Teile entfernt um Gewicht zu sparen. Dabei wurde darauf geachtet, dass die Stabilität keinesfalls unter diesem Ausbau litt.

Zu guter Letzt wurden noch einige ästhetische Aspekte verbessert. Mit den zusätzlichen Teilen wurde dann, als netter Nebeneffekt, der Schwerpunkt noch weiter nach unten verlegt. Denn je weiter unten dieser ist, desto stabiler sind die Fahreigenschaften des Roboters (Kippmoment). Unter Umständen muss der Schwerpunkt des Roboters später nochmals verändert werden, um optimale Rotations-eigenschaften zu erreichen.

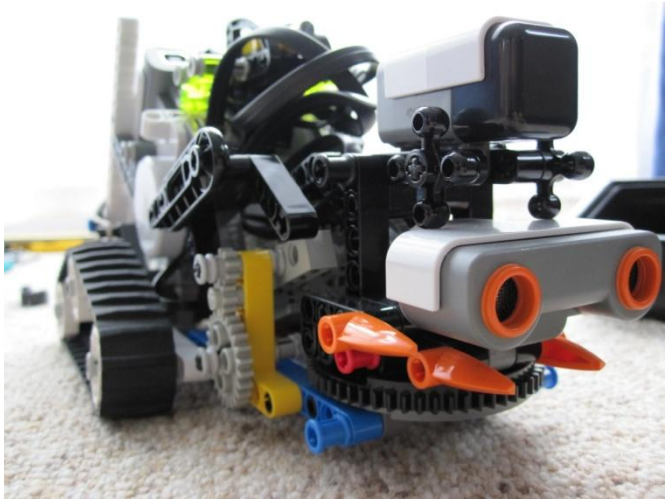


Abb. 25: Prototyp II β

3.2.1.4 Prototyp II γ

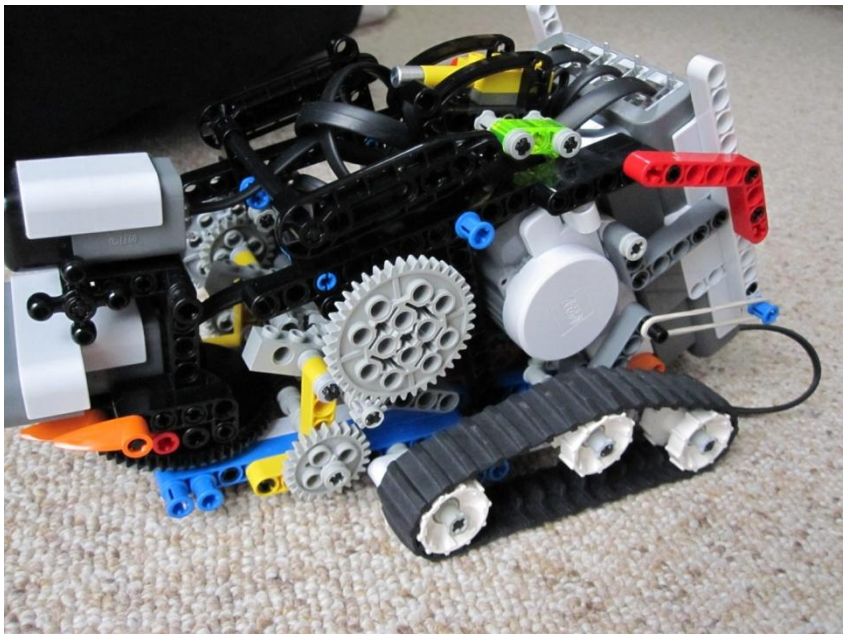


Abb. 26: Prototyp II γ

Bei Tests mit dem Prototyp II β stellte sich heraus, dass die Drehplattform Messungen mit einem Winkel grösser als 30° verunmöglichte, da sie am Roboter anstiess. Zudem behinderten auch die Kabel, die zum Ultraschallsensor und zum Kompasssensor führten eine regelmässige Rotation, weil sie eine starke Kraft ausübten. Dazu kam noch, dass das Getriebe und dessen Aufhängung nicht stabil genug waren. Die Zahnräder griffen nicht genügend gut ineinander. Das Umbauen des Getriebes stellte sich als schwieriger heraus als angenommen. Denn es musste einerseits eine gute Übersetzung liefern, andererseits sehr kompakt sein und es sollte zudem ästhetisch noch zum Roboter passen.

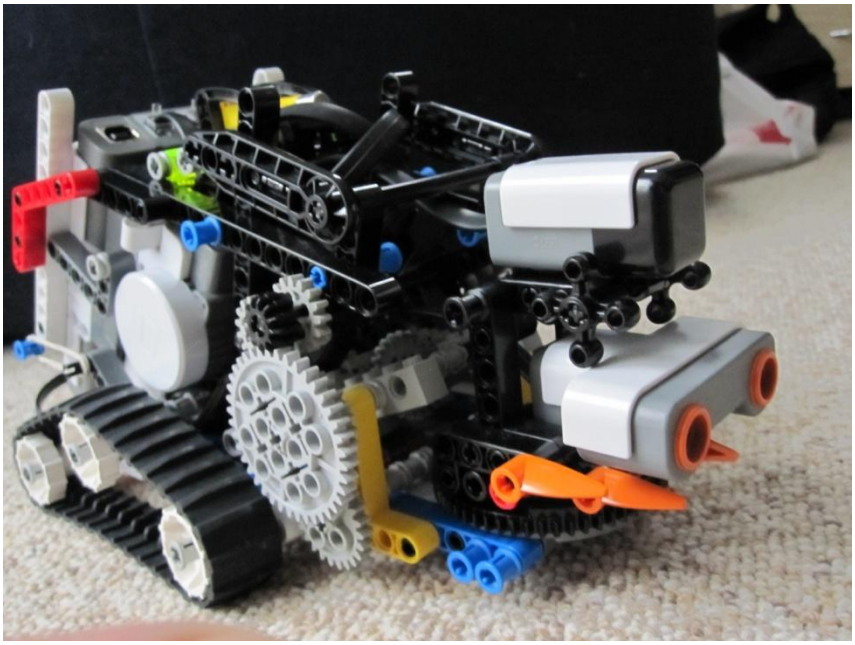


Abb. 27: Prototyp II γ

Damit die Zahnräder gut aufeinander lagen, musste die Stabilität des Roboters markant verbessert werden. Dafür wurde ein neuer Unterboden gebaut und einige Verstrebungen mehr eingebaut. Das Resultat liess sich sehen, da das Getriebe jetzt viel ruhiger lief und weniger anfällig auf Pannen war.

Damit die Kabel nicht ins Getriebe gerieten oder einen zu grossen Druck auf die Drehplattform ausübten, musste die Kabelführung geändert werden.

3.2.1.5 Erste Tests um digitale Karten zu erstellen

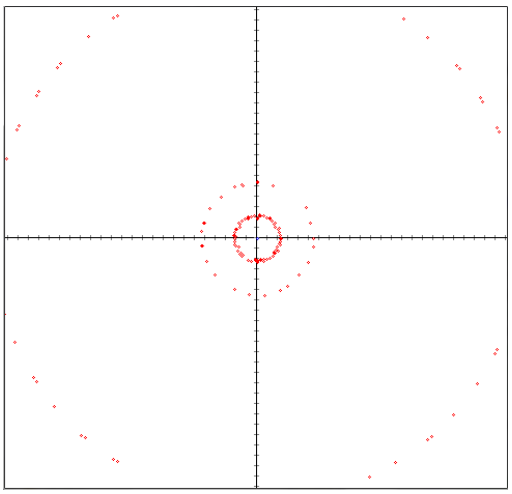


Abb. 28: Fehler in der Testkarte

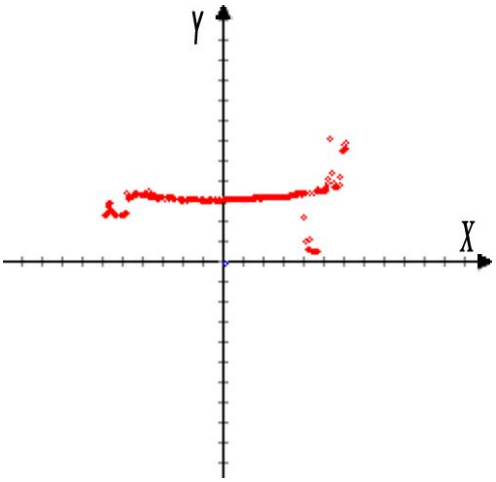


Abb. 29: X-Achse Winkel, Y-Achse Distanz

Bei den allerersten Messungen mit dem Prototypen II γ klappte die Livevorschau, es wurden immer mehr Punkte im Koordinatensystem dargestellt. Allerdings gab die Verteilung der Punkte einige Rätsel auf, denn sie wurden kreisförmig eingetragen (→Bild oben links). Um zu sehen, wie sich die Punkte bezüglich Distanz und Winkel veränderten, wurde nochmals die gleiche Messung durchgeführt. Dieses Mal wurde der Winkel auf der X-Achse dargestellt und die Distanz auf der Y-Achse. Der Winkel veränderte sich stetig, was darauf hindeutete, dass er stimmen musste. Die Distanz schien aber auch richtig zu sein, denn die Hand, welche während der Messung direkt vor den Sensor gehalten wurde verursachte eine deutliche Änderung. Die nächste Vermutung war, dass die Berechnung mit dem Kosinus fehlerhaft sein musste. Doch nach einigen Tests stellte sich heraus, dass der Navigator den Winkel standardmässig in Bogenmass angibt.

Abb. 30: Realität des Kartenausschnittes

Links ein Foto der echten Szenerie. Die davon generierte Karte hat noch einige Schwachpunkte, doch ist die Ähnlichkeit erkennbar.

Wie im Kapitel Material und Methoden prophezeit gab es mit dem Ultraschallsensor das Problem der Scheinechos.
In der Ecke (links oben) wurde das Echo via die zwei „Wände“ reflektiert und dies resultierte in einer zu grossen Entfernung der Punkte.

Die Messungenauigkeit war auch allgemein ziemlich hoch. Das konnte noch dadurch verbessert werden, dass die Messungen in Ruhe gemacht werden, beziehungsweise die Konstruktion des Roboters verfeinert wurde.

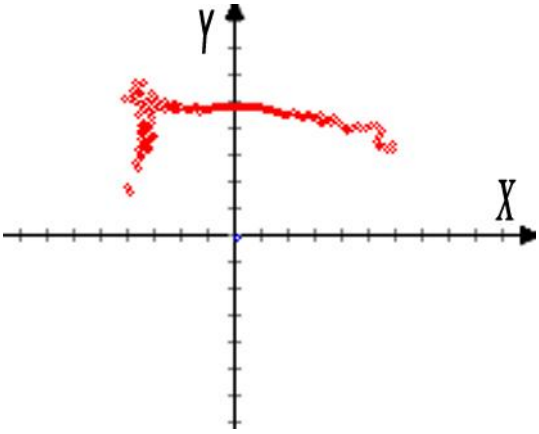


Abb. 31: Neue Testkarte

Desweiteren ist zu bemerken, dass der Kompasssensor zu nahe an den Elektromotoren befestigt war und somit, auch mit einer Kalibrierung vor der Messung, durch das Elektromagnetische Feld des Elektromotors gestört wurde.

Die Distanzmessung könnte vielleicht durch die Verwendung des Infrarotsensors verbessert werden.

3.2.1.6 Prototyp III

Der Prototyp III basiert auf der Grundlage des Prototyps II γ, doch die Ähnlichkeiten beschränken sich auf ein Minimum. Der neue Roboter besteht aus der circa doppelten Anzahl an Bausteinen und bringt zahlreiche Fortschritte. Die extrem kompakte Bauweise erforderte beim Bau Fingerspitzengefühl. Unter anderem wurde das Getriebe für die Drehplattform neu konzipiert. Letztere ist die markanteste Änderung. Sie ist nun wesentlich grösser und schwerer. Um das Gewicht zu tragen und zugleich eine perfekte horizontale Ausrichtung zu haben,

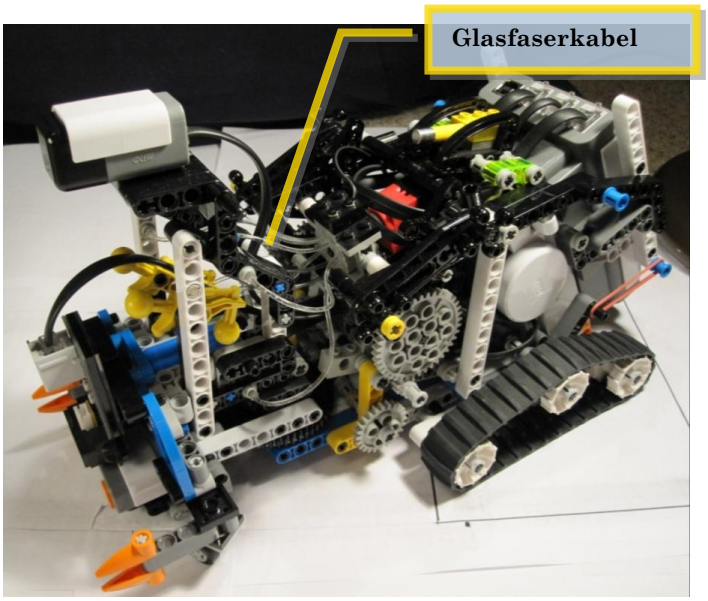


Abb. 32: Prototyp III

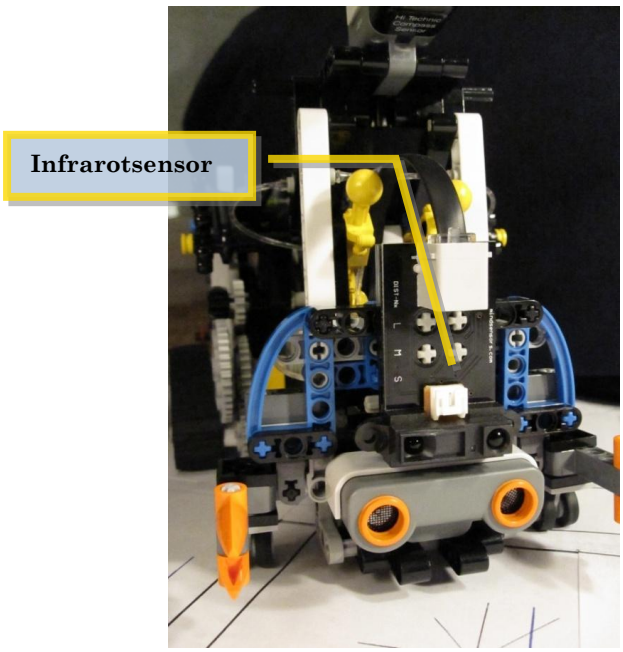


Abb. 33: Prototyp III

benötigt sie zwei Stützräder. Diese können sich in alle Richtungen bewegen und verursachen wenig Reibung. Somit wird die Navigierungsfähigkeit des Roboters nicht eingeschränkt. Der Grund für die Mehrbelastung der Drehplattform ist einerseits der neue Infrarotdistanzsensor und andererseits die Fixation des Kompasssensors. Dieser hat nun, aufgrund seiner neuen Position, genügend Abstand von den Elektromotoren und sollte deshalb verlässlichere Werte liefern. Die Hauptschwierigkeit war die Befestigung der Distanzsensoren, denn diese müssen absolut im Lot zum Boden fixiert sein. Anderenfalls würde dies einen grossen systematischen Fehler verursachen,

bedingt durch falsche Distanzangaben.

Bei ersten Tests wurde dann gleich festgestellt, dass der NXT-Block und die unten herausragenden Sensorkabel den Boden touchieren und so Reibung verursachen. Folglich verminderte sich die Präzision des Roboters im Navigieren[11]. Daher wurde der NXT nach oben verschoben und einige Kabelführungen verändert.

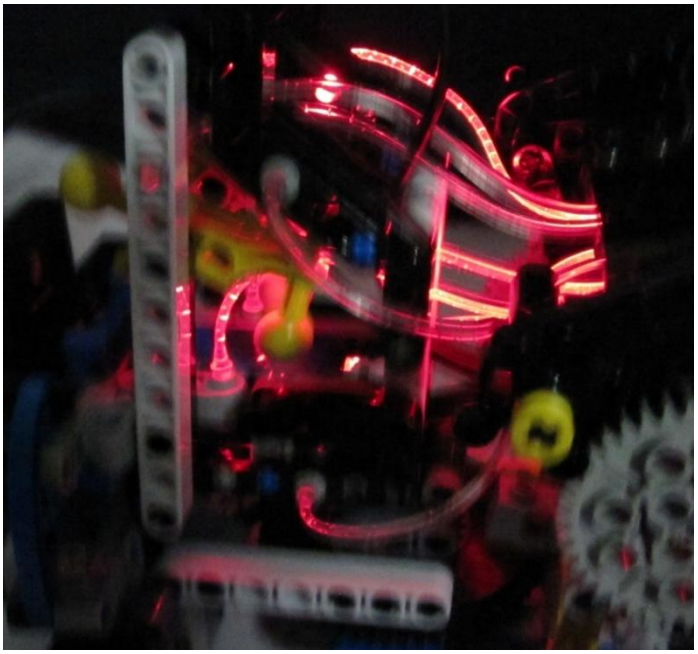


Abb. 34: Prototyp III

Auch ästhetisch ist der Prototyp III eine Verbesserung. Zahlreiche Accessoires verschönern das Design. Höhepunkt davon sind die Glasfaserkabel, die alternierend aufleuchten. Mit Hilfe eines Schalters kann das Licht ausgeschaltet werden. Wenn die Lichtquelle eingeschaltet ist, dann leuchten sie während des Messvorgangs auf. Für diesen Zusatz musste unter anderem ein weiterer Elektromotor (rot), eine Lichtquelle, ein Schalter und zahlreiche Kabel eingebaut werden. Das resultierende zusätzliche Gewicht wurde dazu verwendet, um den Schwerpunkt des Roboters möglichst in die Mitte der Raupenkonstruktion zu bringen. Damit ist eine präzisere Rotationsbewegung möglich. Die immer grösser werdende Kompaktheit verursacht aber auch Probleme. Wenn zum Beispiel ein Zahnrad im Getriebe defekt ist, oder wenn ein Elektromotor Störungen hätte, dann ist die Wartung des Roboters extrem zeitaufwändig.

3.2.1.7 Ausführliche Tests des Prototyps III

Beim Bau des Prototyps III wurde sehr viel Wert auf die Präzision der Konstruktion gelegt. So zum Beispiel, dass die Distanzmesssensoren und der Kompasssensor exakt parallel, bezüglich des Bodens sind und folglich genauere Werte liefern. Die grösste Verbesserung war aber wohl der neue Infrarotdistanzsensor. Nun galt es die Neuerungen zu testen, um systematische Fehler möglichst zu vermeiden. Dabei wurden die Tests in vier Kapitel unterteilt. Im ersten geht es darum, dass der Roboter eine Distanz so genau wie möglich geradeaus zurücklegen soll. Im nächsten Teil wird die Rotationsbewegung thematisiert. Im dritten Kapitel werden dann die beiden Aspekte kombiniert. Schliesslich im letzten Teil werden der Ultraschallsensor und der Infrarotsensor auf ihre Funktionstüchtigkeit überprüft und einander gegenübergestellt.

Für die Tests wurde eine Papierunterlage (115cm*60cm) verwendet. Darauf wurden zahlreiche Markierungen angebracht, um die Resultate präzise ablesen zu können.

[11] siehe Kapitel 3.2.1.7.1 Navigation: Eine bestimmte Strecke geradeaus fahren

3.2.1.7.1 Navigation: Eine bestimmte Strecke geradeaus fahren

3.2.1.7.1.1 Methode

Das Ziel dieser Testreihe ist es, den Roboter so zu optimieren, dass er eine bestimmte Distanz so exakt wie möglich zurücklegt. Um das ideale Handling zu finden, wurden mehrere Faktoren getestet.

```
Motor.A.setSpeed(10);
Motor.C.setSpeed(10);
Motor.A.smoothAcceleration(true);
Motor.C.smoothAcceleration(true);
navigator.goTo(900, 0);
Thread.sleep(10000);
navigator.setPosition(0, 0, 0);
Thread.sleep(5000);
Motor.A.setSpeed(20);
Motor.C.setSpeed(20);
navigator.goTo(900, 0);
Thread.sleep(10000);
navigator.setPosition(0, 0, 0);
Thread.sleep(5000);


Motor.A.forward();
Motor.B.forward();
Thread.sleep(5000);
speeda = Motor.A.getActualSpeed();
speedc = Motor.C.getActualSpeed();
LCD.clear();
LCD.drawInt(speeda, 1, 1);
LCD.drawInt(speedc, 1, 2);
Motor.A.stop();
Motor.B.stop();
```

Abb. 35: Verwendeter Programmcode

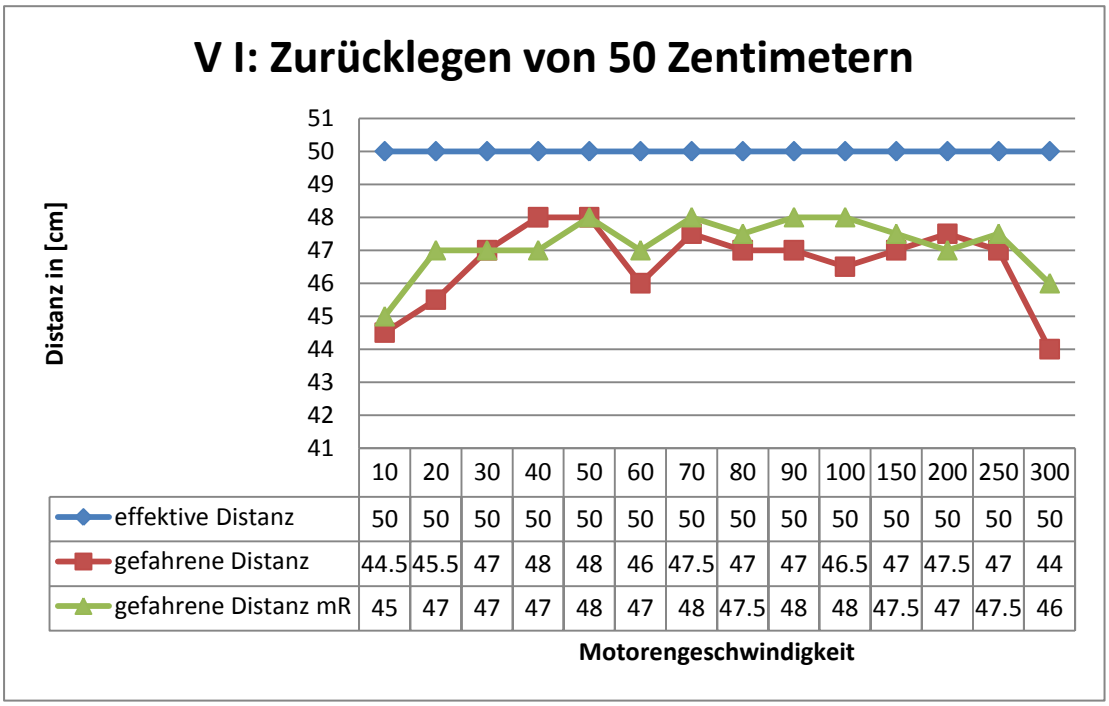
Um die Werte miteinander vergleichen zu können, wurde immer dasselbe simple Programm verwendet. Zuerst wurde die Motorgeschwindigkeit festgelegt. Danach kam je nach Test das Extra der ruhigen Beschleunigung dazu. Schliesslich wurde dem Roboter über die Navigator-Klasse der Befehl gegeben den Punkt (X/Y) zu erreichen. Sobald dieser Punkt erreicht war, legte der Roboter eine Pause ein und setzte seine Position wieder auf (0/0). In dieser Zeit wurde der Roboter auch materiell wieder auf den Ursprungspunkt gesetzt. Auf diese Weise konnten die Tests auf effizienteste Art und Weise durchgeführt werden.

Der untere Programmcode wurde nur testmässig ab und an verwendet, um festzustellen, welche maximale Geschwindigkeit der Roboter überhaupt zurücklegen kann.

Alle Tests wurden bei 14 verschiedenen Geschwindigkeiten durchgeführt, um zu sehen, ob eine Geschwindigkeit besonders gut geeignet ist.

3.2.1.7.1.2 Resultate und Folgerungen

In einem ersten Versuch wurde dem Roboter der Befehl erteilt, 50 Zentimeter in X-Richtung zurückzulegen. Einmal wurden die Raupen durch die Hebelapparatur zusätzlich gestrafft („gefahrene Distanz mR“) und einmal nicht („gefahrene Distanz“).



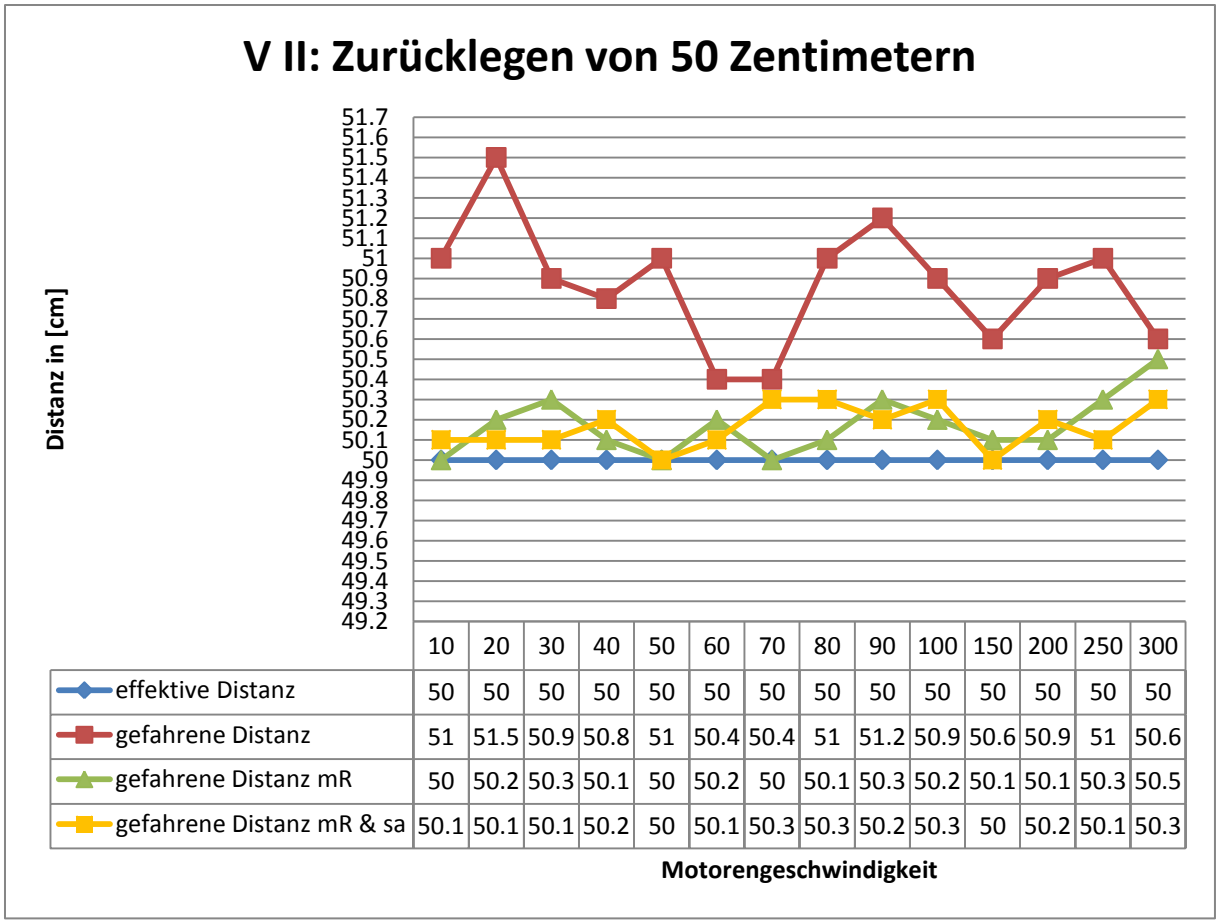
	10	20	30	40	50	60	70	80	90	100	150	200	250	300
effektive Distanz	50	50	50	50	50	50	50	50	50	50	50	50	50	50
gefahrene Distanz	44.5	45.5	47	48	48	46	47.5	47	47	46.5	47	47.5	47	44
gefahrene Distanz mR	45	47	47	47	48	47	48	47.5	48	48	47.5	47	47.5	46

Diagramm 1: VI: Zurücklegen von 50 Zentimetern

Bei der ersten Messung, ohne eine zusätzlichen Straffung der Raupen, betrug der Mittelwert der Distanzen 46.61 Zentimeter (→durchschnittliche Abweichung 3.39 Zentimeter). Bei der zweiten Messung war der Schnitt mit 47.18 Zentimeter (Abweichung 2.82 Zentimeter) schon etwas besser.

Bei diesen beiden Messungen wurde festgestellt, dass der NXT-Block und die unten herausragenden Sensorkabel Reibung verursachen und so die Navigationspräzision verringern. Daraufhin wurde der Roboter umgebaut. Der NXT-Block wurde nach oben verschoben und einige Kabelführungen geändert, sodass ausser den Raupen nichts mehr den Boden touchierte. Der so entstandene, verbesserte Roboter wurde jetzt hier im Testbericht V II genannt.

Die ersten Tests wurden nun nochmals wiederholt, um die Werte in Relation zu stellen und so die Verbesserung ersichtlich zu machen.



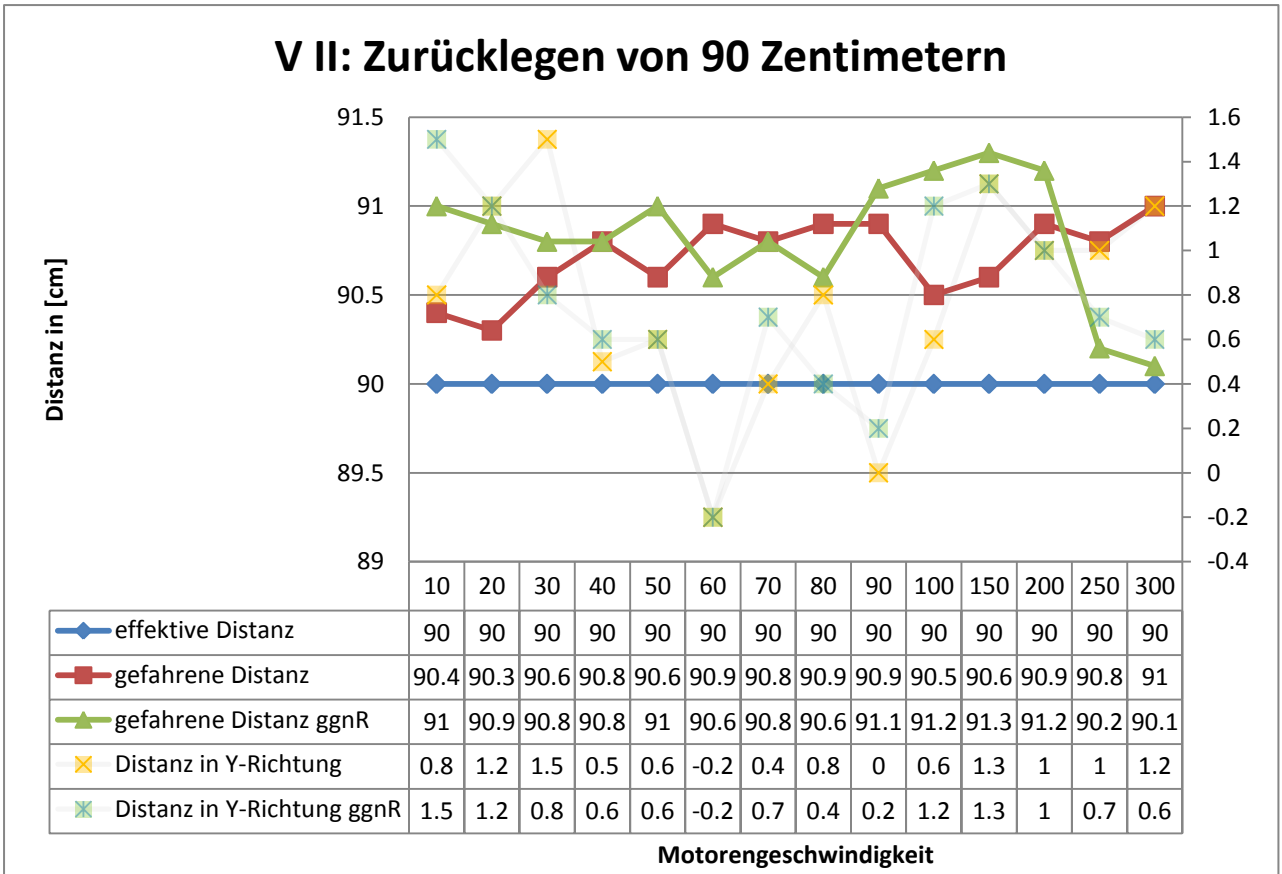
	10	20	30	40	50	60	70	80	90	100	150	200	250	300
effektive Distanz	50	50	50	50	50	50	50	50	50	50	50	50	50	50
gefahrene Distanz	51	51.5	50.9	50.8	51	50.4	50.4	51	51.2	50.9	50.6	50.9	51	50.6
gefahrene Distanz mR	50	50.2	50.3	50.1	50	50.2	50	50.1	50.3	50.2	50.1	50.1	50.3	50.5
gefahrene Distanz mR & sa	50.1	50.1	50.1	50.2	50	50.1	50.3	50.3	50.2	50.3	50	50.2	50.1	50.3

Diagramm 2: VII: Zurücklegen von 50 Zentimetern

Man sieht auf einen Blick, dass der Roboter durch die verringerte Reibung weiterfuhr, als bei der ersten Version. Der Mittelwert der „gefahrenen Distanz“ betrug 50.87 Zentimeter (Abweichung 0.87 Zentimeter), derjenige der „gefahrenen Distanz mR“ 50.17 (Abweichung 0.17 Zentimeter). Bei den Messungen „gefahrene Distanz mR & sa“ wurde zusätzlich die sanfte Beschleunigung integriert. Die Verbesserung war aber mit dem Mittelwert 50.16 (Abweichung 0.16 Zentimeter) marginal.

Daraufhin wurde die Distanz auf 90 Zentimeter vergrössert. Auf diese Weise war die Abweichung in der Fahrgeschwindigkeit deutlicher, da die Faktoren Anfahren und Anhalten immer konstant sind. Das Problem bei den Messungen war, dass der Roboter nicht exakt geradeaus fuhr, sondern auch Abweichungen in y-Richtung verursachte. Diese Abweichungen wurden auf der sekundären, vertikalen Achse eingetragen. Dabei steht eine positive Abweichung für eine Abweichung nach links und analog eine Negative für rechts.

Damit diese Abweichungen nicht nur als systematische Fehler, bedingt durch das Testgebiet, gewertet werden, wurde das Zurücklegen der 90 Zentimeter von beiden Seiten durchgeführt.



	10	20	30	40	50	60	70	80	90	100	150	200	250	300
effektive Distanz	90	90	90	90	90	90	90	90	90	90	90	90	90	90
gefahrene Distanz	90.4	90.3	90.6	90.8	90.6	90.9	90.8	90.9	90.9	90.5	90.6	90.9	90.8	91
gefahrene Distanz ggnR	91	90.9	90.8	90.8	91	90.6	90.8	90.6	91.1	91.2	91.3	91.2	90.2	90.1
Distanz in Y-Richtung	0.8	1.2	1.5	0.5	0.6	-0.2	0.4	0.8	0	0.6	1.3	1	1	1.2
Distanz in Y-Richtung ggnR	1.5	1.2	0.8	0.6	0.6	-0.2	0.7	0.4	0.2	1.2	1.3	1	0.7	0.6

Diagramm 3: Zurücklegen von 90 Zentimetern

Der Mittelwert der „gefahrenen Distanz“ betrug 90.71 Zentimeter (Abweichung 0.71 Zentimeter). In umgekehrte Richtung „gefahrene Distanz ggnR“ 90.83 Zentimeter (Abweichung 0.83 Zentimeter). Diese Ungenauigkeiten waren wohl einerseits durch den Schlupf entstanden (Anfahren/Bremsen) und andererseits durch auf Millimeter gerundete Angabe des Raddurchmessers in der Navigator-Klasse.

Die Tendenz, dass der Roboter vermehrt nach links vom Kurs abwich, könnte durch die Motoren verursacht worden sein, weil sie unterschiedlich stark agieren. Oder wahrscheinlicher durch eine grössere Reibung an der linken Raupenaufhängung.

3.2.1.7.1.3 Zusammenfassung

Mit Hilfe dieser Tests konnte die Genauigkeit erheblich verbessert werden. Der Roboter kann nun eine bestimmte Strecke mit einer Unsicherheit von circa 2% (der Gesamtstrecke) zurücklegen. Aus den Abweichungen in X- und Y-Richtung kann man schliessen, dass es Sinn macht, während des Fahrens auf Abweichungen bezüglich der Richtung zu prüfen.

3.2.1.7.2 Rotationsbewegungen

3.2.1.7.2.1 Methode

Nachdem die Resultate des letzten Kapitels ausgewertet waren und die Schlüsse daraus gezogen waren, kam nun die Rotationsbewegung dazu. Mit einem einfachen Skript wurden dem Roboter die Befehl gegeben eine 90° Drehung, beziehungsweise eine 180° Drehung zu machen und zwar in positiver und negativer Drehrichtung. Im zweiten Teil des Programms wurde manuell geprüft, ob die Navigatorklasse die Drehung, anhand der Werte, die der Kompasssensor liefert, korrekt ausführte. Bei den Messungen wurde, neben der effektiv gedrehten Grade, die Abweichung in X- und Y-Richtung gemessen.

```
navigator.getCompassPilot().resetCartesianZero();
rotationa = (int) cs.getDegrees();
navigator.setTurnSpeed(4);
Sound.playTone(800, 200, 80);
navigator.rotate(90);
rotationb = (int) cs.getDegrees();
   if(rotationb > rotationa)
   {rotationc = (360- rotationb) + rotationa; }
   else
   {rotationc = rotationa - rotationb;}
   if (rotationc > 180 )
   {rotationd = rotationc - 360;}
   else
   {rotationd = rotationc;}
LCD.clear();
LCD.drawInt(rotationd, 1, 1);
Sound.playTone(800, 200, 80);
Thread.sleep(15000);
```

Abb. 36: Verwendeter Programmcode

3.2.1.7.2.2 Resultate und Folgerungen

Im ersten Test sollte der Roboter um -90° Drehen, dass heisst von Position A nach Position B.

Die Koordinaten-Achsen geben Auskunft darüber in welche Richtung die Abweichungen, von der schlussendlichen Position B, entstanden. Der Roboter startete immer mit der Ausrichtung 0°[12].

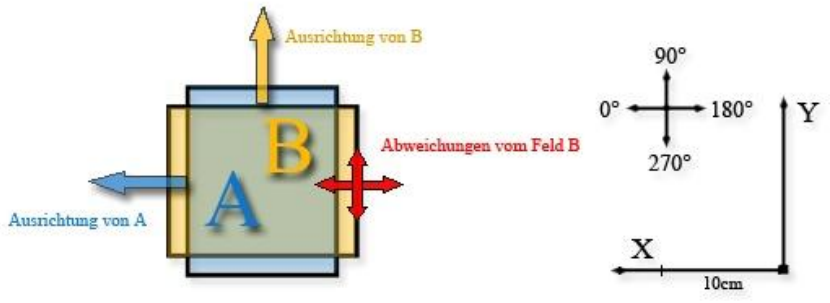
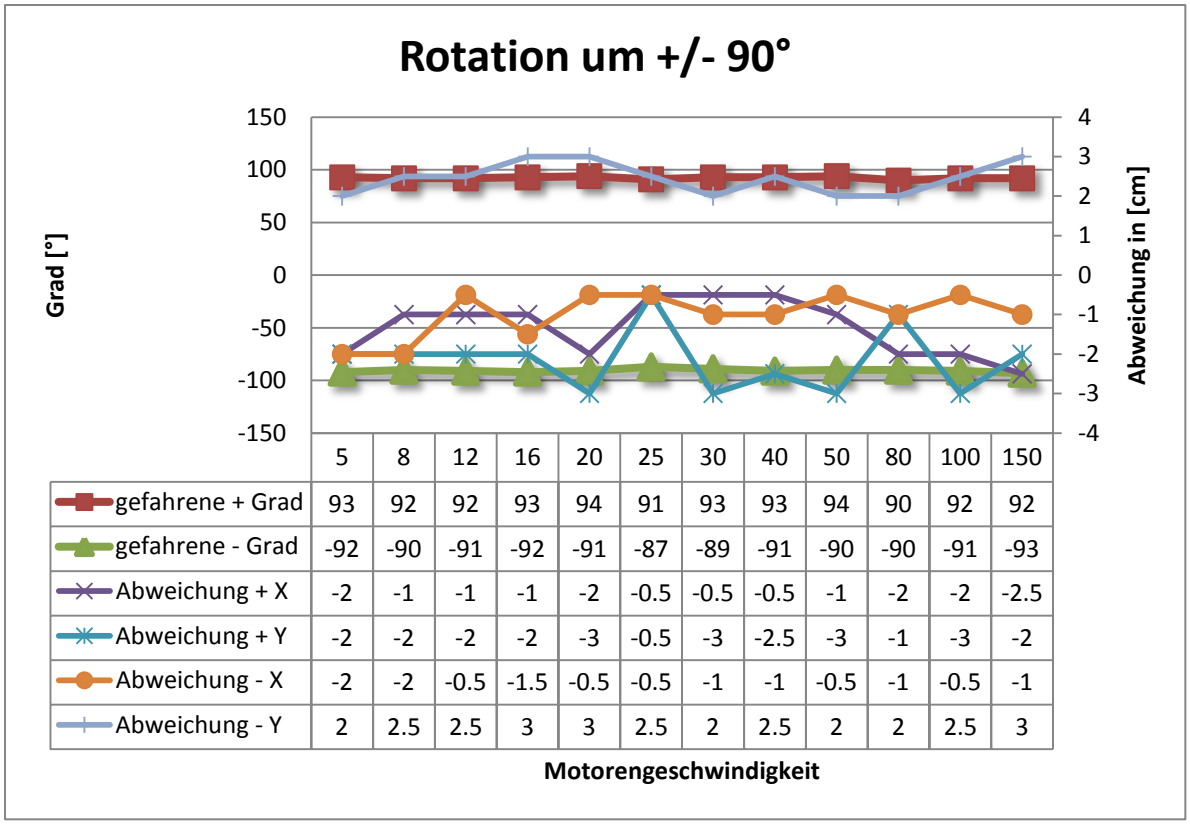


Abb. 37: Versuchsanordnung: Die Rechtecke symbolisieren die Raupen

Mit +90° ist eine Drehung von der Ausrichtung 0° zur Ausrichtung 270° gemeint.

[12] Position A auf der Abbildung 37

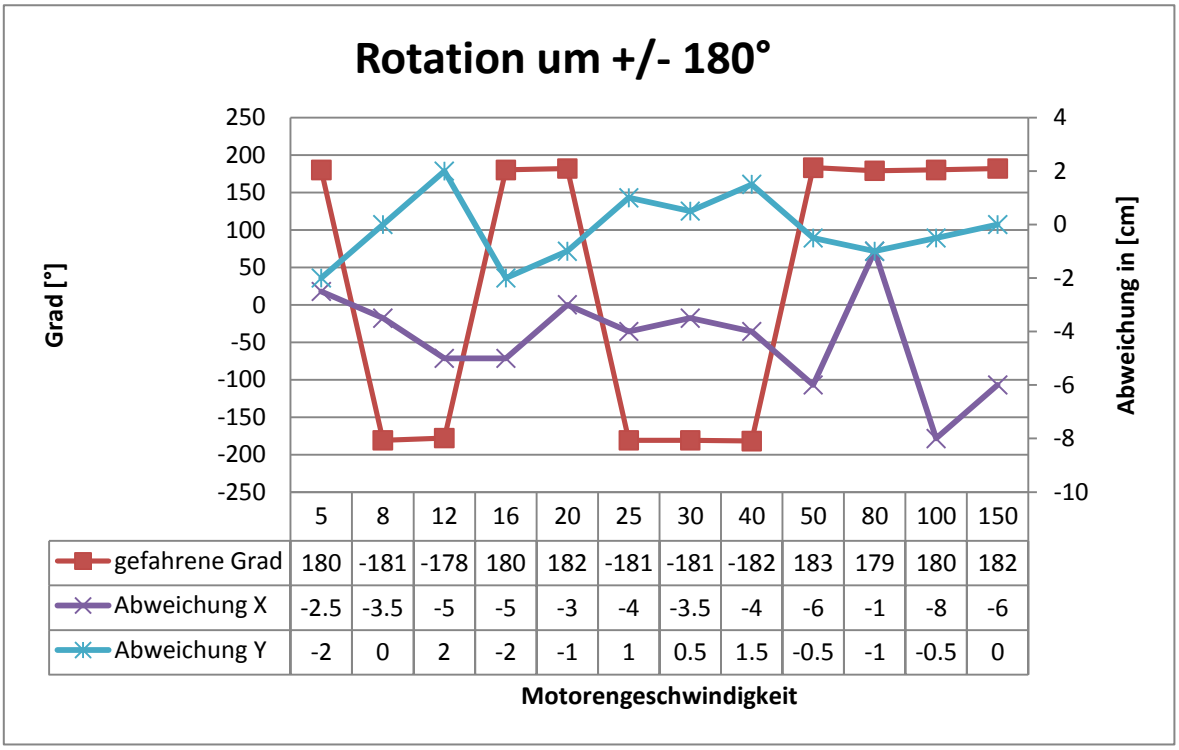


	5	8	12	16	20	25	30	40	50	80	100	150
gefahrene + Grad	93	92	92	93	94	91	93	93	94	90	92	92
gefahrene - Grad	-92	-90	-91	-92	-91	-87	-89	-91	-90	-90	-91	-93
Abweichung + X	-2	-1	-1	-1	-2	-0.5	-0.5	-0.5	-1	-2	-2	-2.5
Abweichung + Y	-2	-2	-2	-2	-3	-0.5	-3	-2.5	-3	-1	-3	-2
Abweichung - X	-2	-2	-0.5	-1.5	-0.5	-0.5	-1	-1	-0.5	-1	-0.5	-1
Abweichung - Y	2	2.5	2.5	3	3	2.5	2	2.5	2	2	2.5	3

Diagramm 4: Rotation um +/- 90°

Auf der horizontalen Achse wurde die Drehgeschwindigkeit variiert (von links nach rechts gesteigert). Auf der primären, vertikalen Achse wurden die Rotationsgrade dargestellt und auf der sekundären die Abweichung in Zentimetern.

Im zweiten Test wurden analog um +/- 180° Drehungen durchgeführt.



	5	8	12	16	20	25	30	40	50	80	100	150
gefahrene Grad	180	-181	-178	180	182	-181	-181	-182	183	179	180	182
Abweichung X	-2.5	-3.5	-5	-5	-3	-4	-3.5	-4	-6	-1	-8	-6
Abweichung Y	-2	0	2	-2	-1	1	0.5	1.5	-0.5	-1	-0.5	0

Diagramm 5: Rotation um +/- 180°

Zuerst stellte sich die Frage, wieso die gedrehten Grade nicht absolut genau stimmten. Diesmal lag das Problem nicht an ungenauen Werten des Kompasssensors, sondern am Schlupf der Drehplattform. Denn auf dieser ist der Kompasssensor befestigt und sie war nicht ganz starr. Das Zahnrad der Drehplattform hatte einen kleinen Spielraum in der Antriebsspindel und sie kann sich deshalb um circa 3° frei bewegen. Wenn sich nun der ganze Roboter drehte konnte sich die Drehplattform um eben diese Grade verschieben. Dadurch stimmte die Ausrichtung des Roboters womöglich um genau diese Abweichung nicht. Die Lösung hierfür war eine neue Antriebsspindel, die keinen Spielraum bietet.

Desweiteren fiel bei den Tests auf, dass die Abweichung in X-Richtung immer negativ war und dass die Y-Abweichung negativ oder positiv war. Negativ war sie immer bei positiver Drehrichtung und positiv bei negativer Drehrichtung. Grundsätzlich kommen die Abweichungen dadurch zustande, dass der Schwerpunkt des Roboters nicht genau in der Mitte der Raupenkonstruktion liegt. Einen weiteren Einfluss könnten aber auch die Motoren und die Raupen haben.

Da sich die Objekte immer um den Schwerpunkt drehen, kann sowohl aus der X-Abweichung, als auch aus der Y-Abweichung gefolgert werden, dass der Schwerpunkt zu weit hinten ist.

Abb. 38: Schwerpunkt/Drehpunkt

Wenn man aber den Schwerpunkt des ganzen Roboters mit Hilfe von Gleichgewichtstests eruiert, kommt man zum gegenteiligen Schluss. Daraus folgt, dass noch andere Faktoren die Drehung beeinflussen. Die Normalenkraft wirkt wohl nicht genau unterhalb des Schwerpunktes. Und die, eventuell ungleichmässig verteilte, Reibung der Stützräder und vor allem der Raupen darf nicht vernachlässigt werden.

Auf dem Bild links ist die Raupenkonstruktion des Roboters symbolisiert. Der Schwerpunkt (S) liegt weiter vorne als der Drehpunkt (P).

Um die Rotationseigenschaften positiv zu verändern, ist Ausprobieren und Testen wohl die einfachste Lösung.

3.2.1.7.3 Von A nach B navigieren

3.2.1.7.3.1 Methode

Der Roboter sollte bei diesen Tests von A nach B navigieren und dort einen bestimmte Ausrichtung einnehmen. Dabei ist B irgendein beliebiger Punkt.

Schlussendlich wird die Genauigkeit dieses Vorgangs ausschlaggebend sein, wie genau die Roboterposition während des Kartografierens ist.

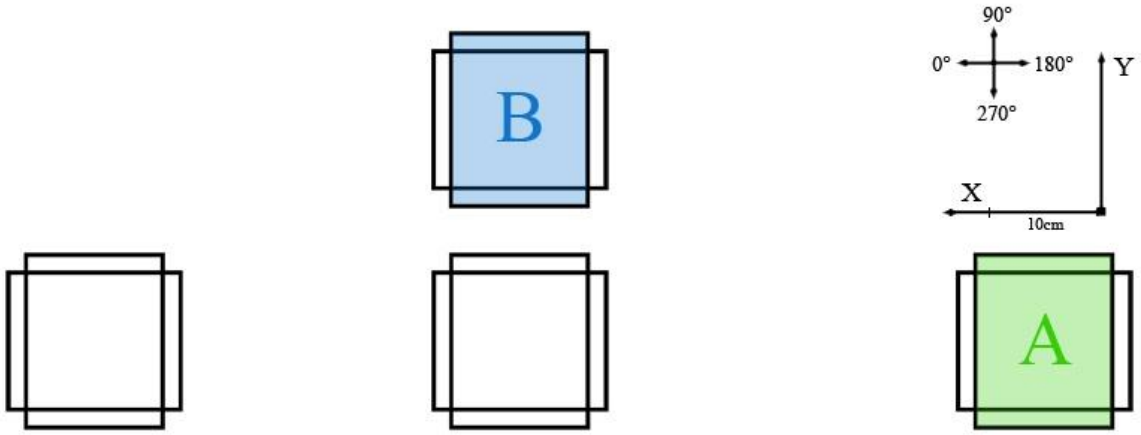


Abb. 39: Versuchsanordnung

Auf dem Bild oben sieht man die Koordinatenachsen und die Grade, damit man weiss in welche Richtung Abweichungen stattfanden.

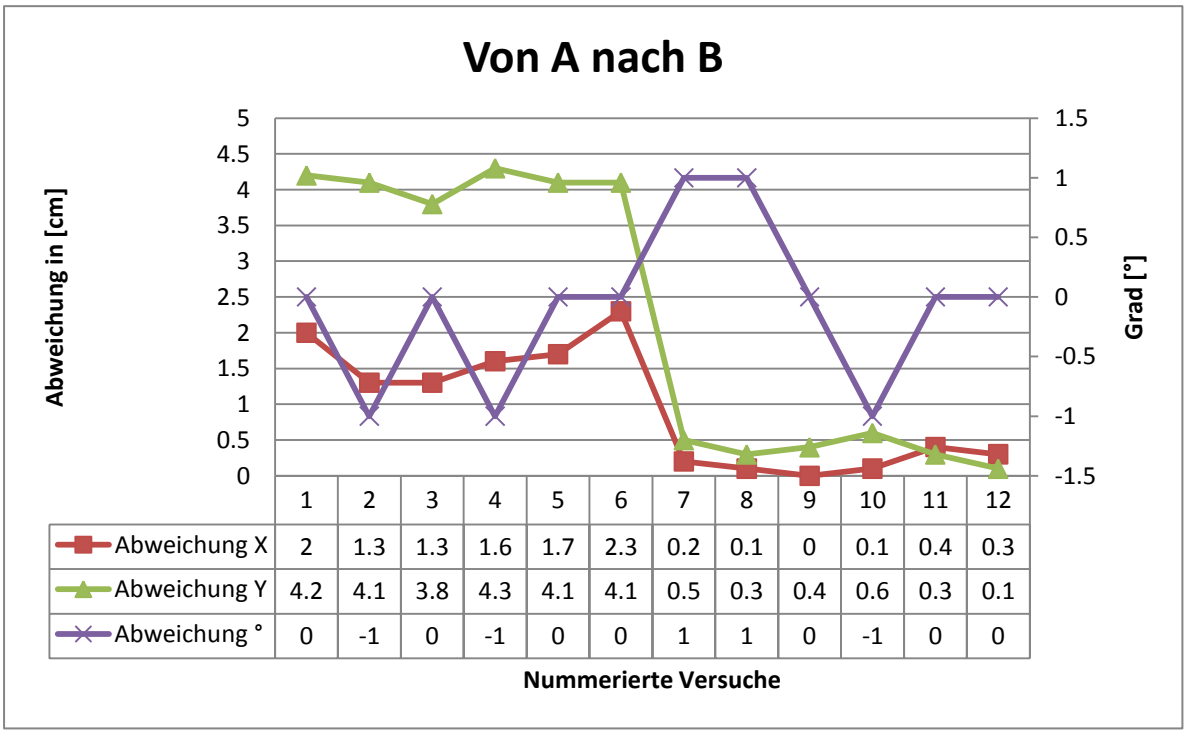
Für die Navigation wurde eine eigene Navigatormethode („DriveTo“) erstellt. Dies deshalb, weil die Standard LeJos Navigatorklasse die Ausrichtung nicht genau einnimmt und dann während des Fahrens auch nicht auf Abweichungen prüft (mit Hilfe des Kompasssensors). Kurze Tests mit de LeJos Klasse gaben Abweichungen von bis zu 15 Zentimetern! Für mehr Details bezüglich der Klasse „DriveTo“ siehe Kapitel 5.2.3 Programm.

3.2.1.7.3.2 Resultate und Folgerungen

Beim ersten Test startete der Roboter bei A (0/0) mit der Ausrichtung 0° und sollte zu B (50/20) navigieren und dann wieder die gleiche Ausrichtung einnehmen.

Die Motorengeschwindigkeit für das vorwärts und rückwärts Fahren, sowie die ideale Drehgeschwindigkeit wurden aus den Versuchen der vorherigen Kapitel übernommen (Drehgeschwindigkeit: 150, Fahren: 50).

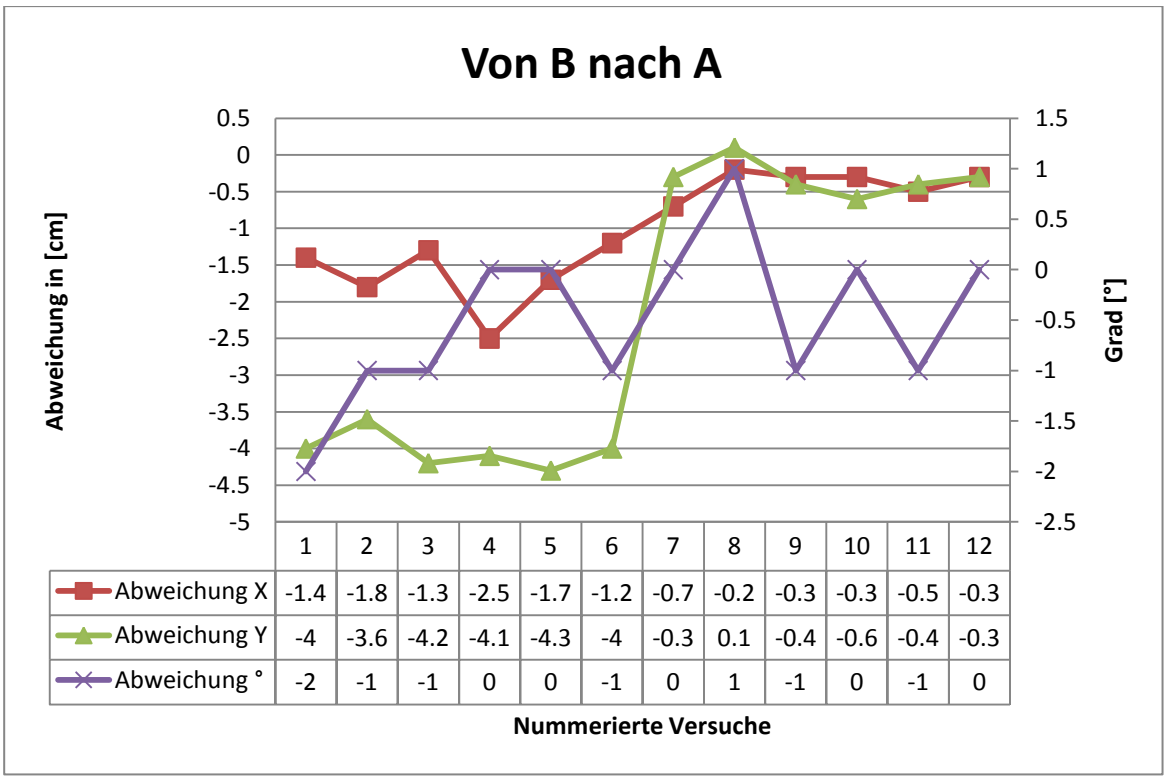
In den ersten sechs Versuchen wurde die elektronische Distanzangabe beim Standardwert belassen (bezüglich Rad- und Achsendurchmesser). Bei den zweiten sechs Tests wurden die Parameter zuerst manuell verändert, so dass die Distanzangabe wesentlich genauer wurde.



	1	2	3	4	5	6	7	8	9	10	11	12
Abweichung X	2	1.3	1.3	1.6	1.7	2.3	0.2	0.1	0	0.1	0.4	0.3
Abweichung Y	4.2	4.1	3.8	4.3	4.1	4.1	0.5	0.3	0.4	0.6	0.3	0.1
Abweichung °	0	-1	0	-1	0	0	1	1	0	-1	0	0

Diagramm 6: Von A nach B

Im zweiten Versuch wurde diesbezüglich analog verfahren. Doch navigierte der Roboter dieses Mal von B (Ausrichtung 0°) zu A (auch Ausrichtung 0°). Dabei legte der Roboter die Strecke rückwärts zurück, um Zeit zu sparen.



	1	2	3	4	5	6	7	8	9	10	11	12
Abweichung X	-1.4	-1.8	-1.3	-2.5	-1.7	-1.2	-0.7	-0.2	-0.3	-0.3	-0.5	-0.3
Abweichung Y	-4	-3.6	-4.2	-4.1	-4.3	-4	-0.3	0.1	-0.4	-0.6	-0.4	-0.3
Abweichung °	-2	-1	-1	0	0	-1	0	1	-1	0	-1	0

Diagramm 7: Von B nach A

Die Resultate dieser Tests zeigen, dass die Methode „DriveTo“ ausgezeichnet funktioniert. Allerdings zeigte sich in den ersten sechs Tests, dass auch der praktisch letzte Teil der LeJos Klasse, der noch gebraucht wurde, ziemlich ungenau ist. Mit einer manuellen Korrektur gewisser Parameter war das Problem dann gelöst. Erfreulich ist vor allem, dass die Werte sehr konstant sind und so zufällige Fehler keine grosse Rolle spielen.

Die Abweichung der Grade ist nun, nachdem der Schlupf an der Drehplattform behoben wurde, praktisch null.

3.2.1.7.3.3 Zusammenfassung

Mit Hilfe der eigenen Navigatormethode und materiellen Veränderungen am Roboter konnten die Navigationseigenschaften positiv verändert werden. Der Roboter hat nun eine genügend genaue Navigation, damit die Roboterposition während des Kartografierens ausreichend exakt sein wird.

3.2.1.7.4 Distanzmessungen

3.2.1.7.4.1 Methode

Auf dem Testgebiet sind auch Markierungen für die Distanzmessungen vorhanden. Mit Hilfe der Duden und dickem Papier wurden Testgegenstände simuliert und so konnten die beiden Distanzmesssensoren ausgiebig getestet werden. Es galt vor allem herauszufinden, ob der neue Infrarotsensor genauere Werte lieferte, als der Ultraschallsensor und unter welchen Bedingungen eventuell Probleme auftreten würden. Mit Hilfe der unten abgebildeten „for-Schleife“ wurden die Messungen unter einheitlichen Bedingungen durchgeführt.

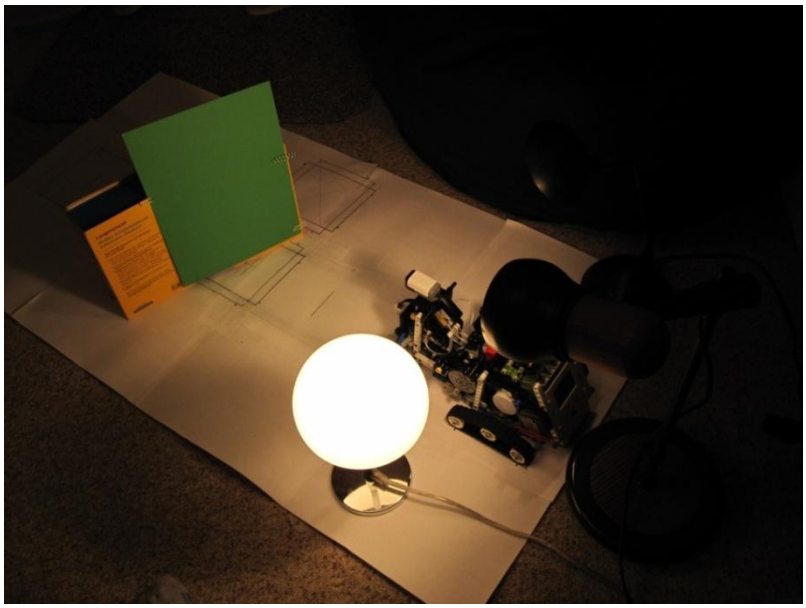


Abb. 40: Versuchsanordnung

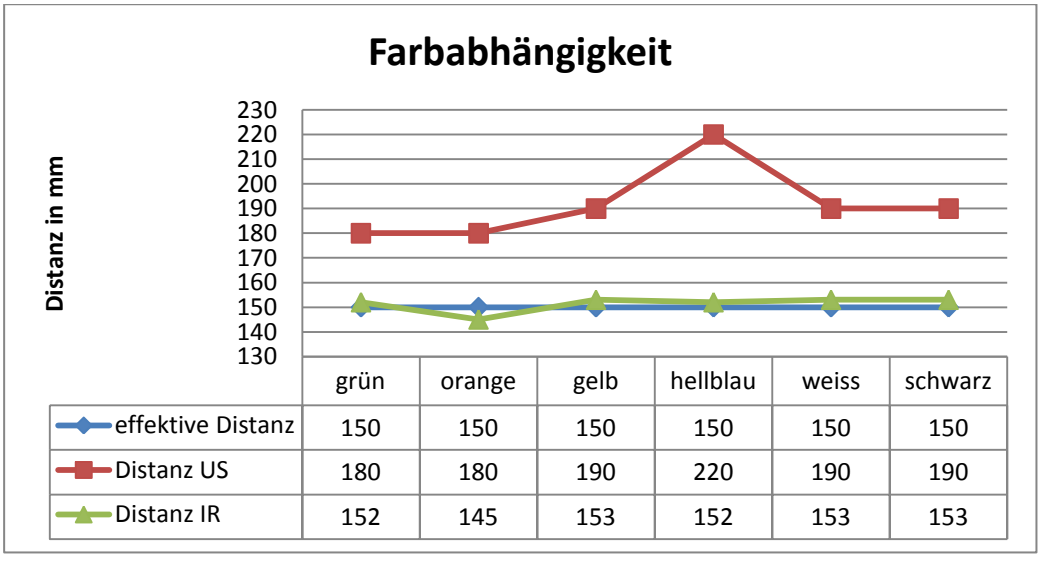
Wie man unschwer erkennen kann wurde jede Messung 100-mal durchgeführt und schliesslich der Mittelwert davon genommen. Wobei dazu zu sagen ist, dass zuerst auch die Standardabweichung berücksichtigt wurde. Doch diese war praktisch immer exakt null und somit war das Risiko auf zufällige Fehler vernachlässigbar klein. Folglich wären die hundert Messungen auch nicht nötig gewesen, da diese häufig alle exakt den gleichen Wert hervorbrachten.

```
for(int i = 0;i<=100;i++){
distus = us.getDistance();
distop = optical.getDistLSB();
distussum = distussum + distus;
distopsum = distopsum + distop;
LCD.drawInt(distussum, 2, 0, 0);
LCD.drawInt(distopsum, 2, 0, 2);
LCD.drawInt(i, 2, 0, 4);
Thread.sleep(1000);
}
Sound.playTone(400, 500, 300);
Thread.sleep(100);
Sound.playTone(900, 600, 200);
Thread.sleep(100);
distussumx = distussum / 100;
distopsumx = distopsum / 100;
LCD.clear();
LCD.drawInt(distussumx, 2, 0, 0);
LCD.drawInt(distopsumx, 2, 0, 2);
Thread.sleep(40000);
distussum = 0;
distopsum = 0;
```

Abb. 41: Verwendeter Programmcode

3.2.1.7.4.2 Resultate und Folgerungen

Zuerst wurden die Sensoren auf Farbabhängigkeit untersucht. Logischerweise sollte der Ultraschallsensor (US) nicht davon betroffen sein, doch könnte es sein, dass er bei gewissen Farben präzisere Resultate lieferte, als der Infrarotsensor (IR).



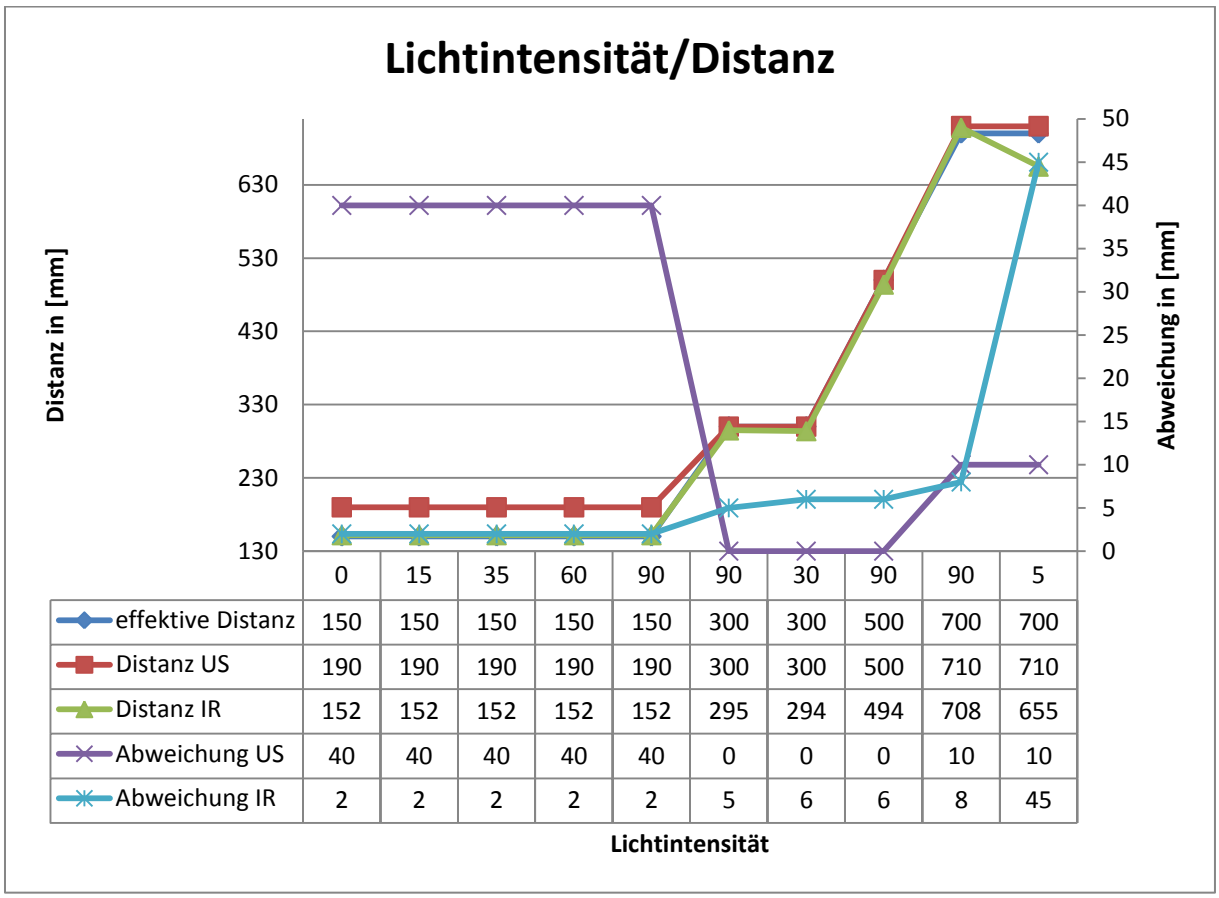
	grün	orange	gelb	hellblau	weiss	schwarz
effektive Distanz	150	150	150	150	150	150
Distanz US	180	180	190	220	190	190
Distanz IR	152	145	153	152	153	153

Diagramm 8: Farbabhängigkeit

Bei einer konstanten Distanz von 150mm wurden immer andere Papiere an den Duden befestigt. Das Resultat zeigte deutlich, dass praktisch keine Farbabhängigkeit vorhanden war und der Infrarotsensor wesentlich präzisere Resultate lieferte. Bei der Messung mit dem hellblauen Papier hatte ein Luftstoss die Messresultate des Ultraschallsensors womöglich verfälscht.

In der nächsten Testreihe wurde die Abhängigkeit der Sensoren von der Lichtintensität getestet. Mit den vier Lampen konnte eine gut abgestufte Helligkeit simuliert werden. Diese wurde dann auf einer Skala von 1-100 abgeschätzt. Der daraus folgende Fehler war nicht gravierend, da unsere Augen die Helligkeit relativ gut abschätzen kann. Die Distanz wurde auch verändert, um zu sehen, ob dies einen Einfluss auf die Ergebnisse hätte. Auch hier sollte der Ultraschallsensor nicht auf die veränderten Lichtintensitäten reagieren.

Abb. 42: Roboter in Aktion



	0	15	35	60	90	90	30	90	90	5
effektive Distanz	150	150	150	150	150	300	300	500	700	700
Distanz US	190	190	190	190	190	300	300	500	710	710
Distanz IR	152	152	152	152	152	295	294	494	708	655
Abweichung US	40	40	40	40	40	0	0	0	10	10
Abweichung IR	2	2	2	2	2	5	6	6	8	45

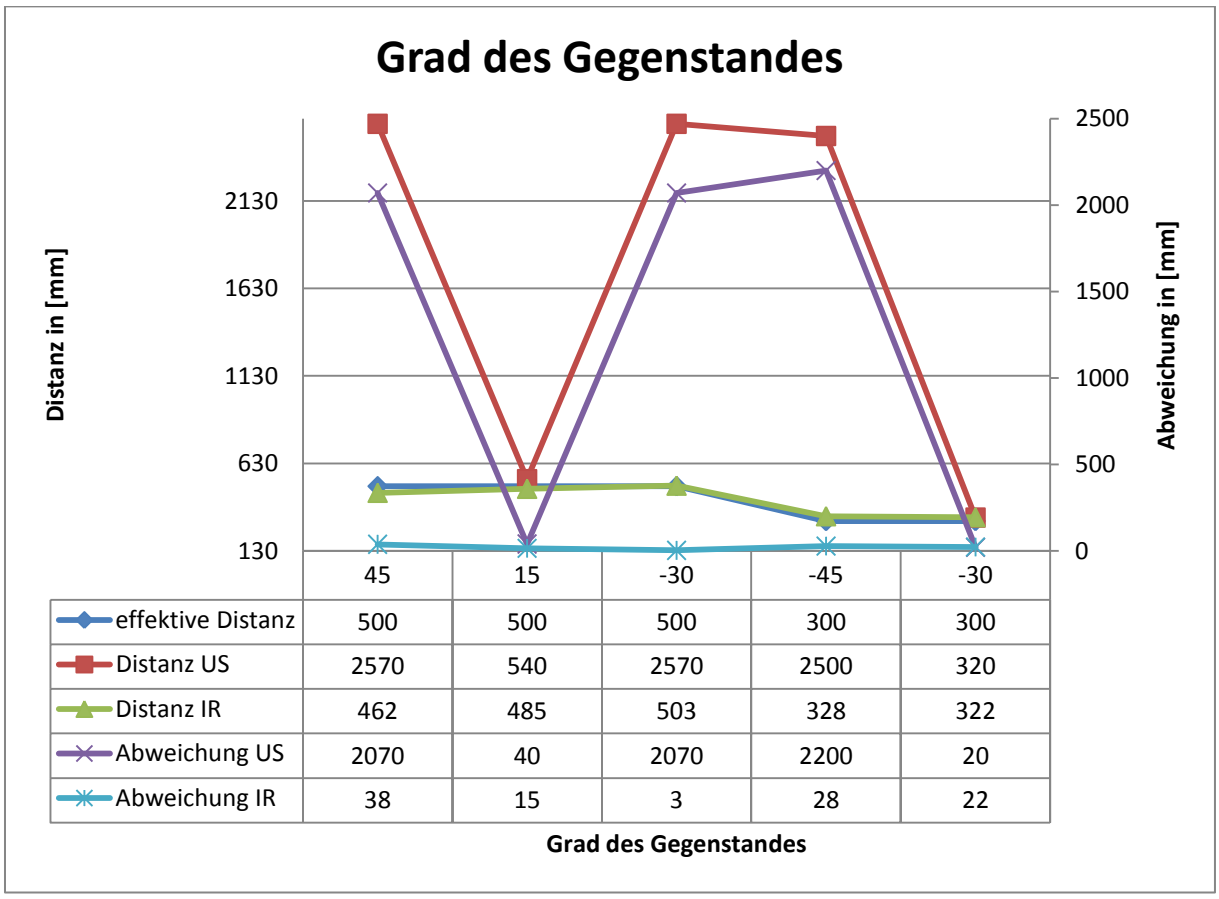
Diagramm 9: Lichtintensität/Distanz

Im Diagramm dargestellt ist sowohl die effektive Distanz, als auch der von den beiden Sensoren gemessene Abstand. Um die Abweichungen vom wahren Wert zu verdeutlichen, wurden diese auf der sekundären vertikalen Achse eingezeichnet.

Es ist klar ersichtlich, dass der Infrarotsensor im Nahbereich (15cm) keine Lichtabhängigkeit zeigte und wesentlich genauere Werte lieferte. Auf einer Distanz zwischen 30 und 50 Zentimetern gab der Ultraschallsensor extrem gute Werte an, doch auch der Infrarotsensor zeigte keine Indizien auf eine Lichtabhängigkeit und lieferte +/- 6mm genaue Werte. Die interessanteste Messung war die letzte. Sie zeigte, dass der Infrarotsensor bei grosser Distanz Mühe hatte, wenn die Lichtintensität klein war. Dies kann man damit erklären, dass er eine integrierte Infrarotlampe hat, die für diese Reichweite nicht genügend stark ist.

Dies muss bei Messungen im Dunklen berücksichtigt werden, damit dann der Ultraschallsensor stärker gewichtet wird.

Im dritten Teil wurde untersucht, ob die Messungen durch schräge Gegenstände beeinflusst werden. Bei den vorhergehenden Messungen war die Kante des Gegenstandes immer im Lot zur Messung. Dieser Idealfall wird aber beim Kartografieren praktisch nie der Fall sein.



	45	15	-30	-45	-30
effektive Distanz	500	500	500	300	300
Distanz US	2570	540	2570	2500	320
Distanz IR	462	485	503	328	322
Abweichung US	2070	40	2070	2200	20
Abweichung IR	38	15	3	28	22

Diagramm 10: Grad des Gegenstandes

Bei diesem Diagramm wurde das identische Schema gewählt, wie bei den Messungen bezüglich der Lichtintensität.

Diese Messungen könnten noch sehr zentral für das Resultat der Arbeit werden. Es ist klar ersichtlich, dass eine enorme Abhängigkeit vom Winkel des Gegenstandes besteht. Vor allem der Ultraschallsensor reagiert darauf, denn wie im Kapitel Material und Methoden[13] geschildert, ergaben sich Scheinechos und die Distanz konnte nicht abgeschätzt werden. Doch auch der Infrarotsensor liess sich täuschen und lieferte Werte die um bis zu 10% vom wahren Wert abwichen.

Die Fehlerrechnung diesbezüglich könnte sehr kompliziert bis praktisch unmöglich werden, da der Roboter bei der Messung nicht erkennen kann, in welchem Winkel der Gegenstand zum Roboter steht. Die wohl idealste Lösung wäre ein präziser Laserdistanzsensor.

3.2.1.7.4.3 Zusammenfassung

Aus den Tests kann geschlossen werden, dass der Infrarotsensor stärker gewichtet werden soll als der Ultraschallsensor, da er vor allem im Nahbereich deutlich genauere Werte liefert. Einzig bei schwacher Lichtintensität und einer grossen Distanz ist der Ultraschallsensor zu bevorzugen.

Der wichtigste Punkt ist aber die Abhängigkeit von den Winkeln und der Oberfläche der Gegenstände. Die Folge davon ist, dass mehrere Messungen von verschiedenen Standpunkten nötig sein werden und wenn möglich eine Fehlerrechnung gemacht werden sollte.

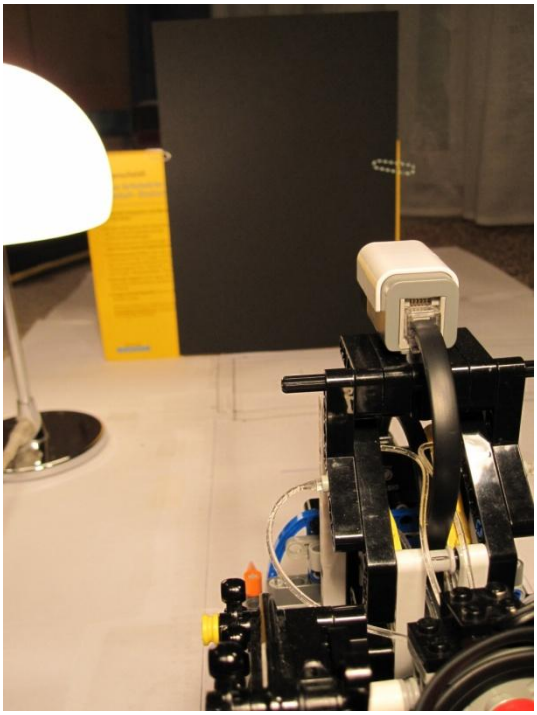


Abb. 43: Roboter in Aktion

[13] genauer Kapitel 3.1.3.1.2 Ultraschallsensor

3.2.1.8 Endroboter

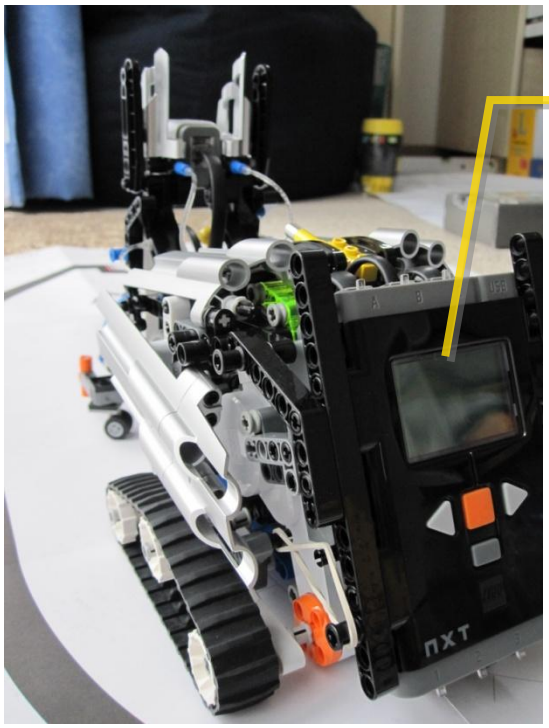


Abb. 44: Endroboter

Der Endroboter ist stark auf dem dritten Prototypen fundiert. Er enthält alle Verbesserungen die aus den zahlreichen Tests gefolgert wurden.

Der neue, schwarze NXT-Block ermöglicht es dem Roboter wieder Töne abzuspielen, da der Lautsprecher beim Alten kaputt gegangen war.

Neben den technischen Verbesserungen wurde auch das Design berücksichtigt. Mit einigen speziellen Legoteilen wirkt der Roboter nun kompakter. Beim Einbau dieser Teile wurden gleichzeitig auch noch Verspannungen im Grundgerüst behoben.

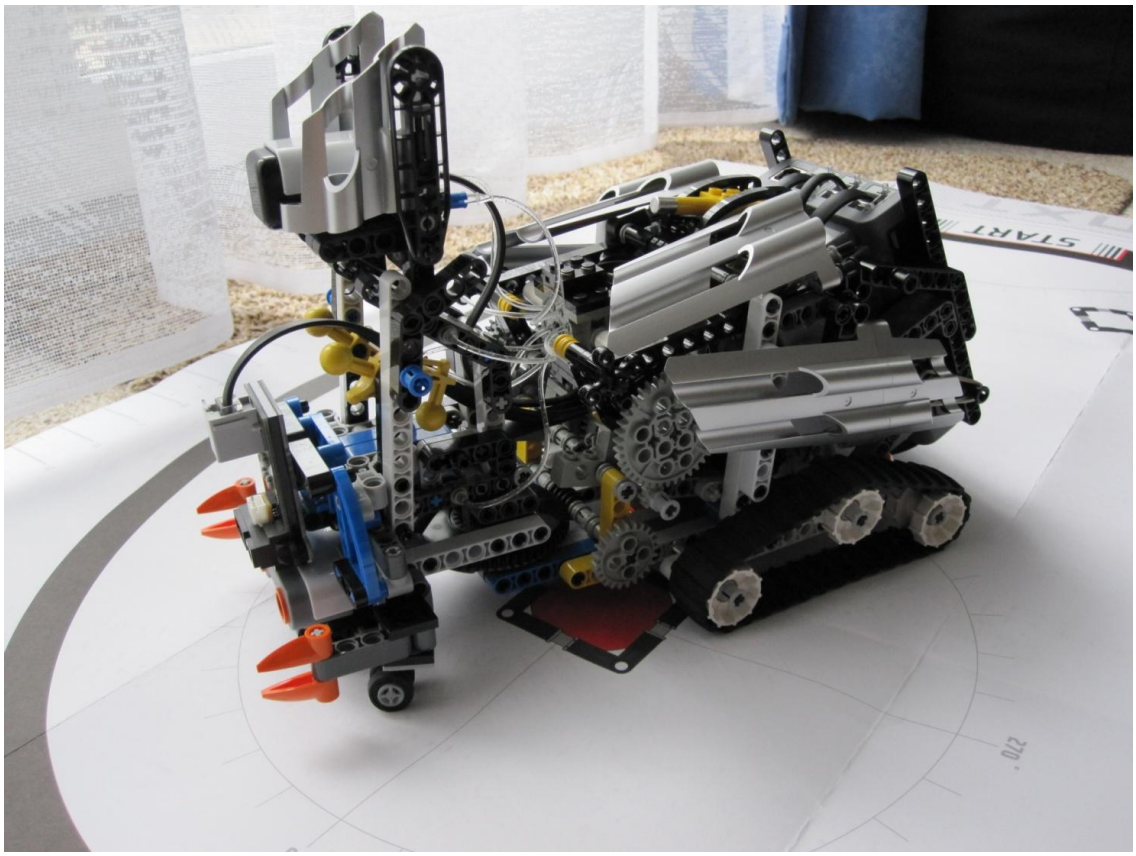


Abb. 45: Endroboter

3.2.2 Programm

3.2.2.1 Funktionsprinzip – Überblick

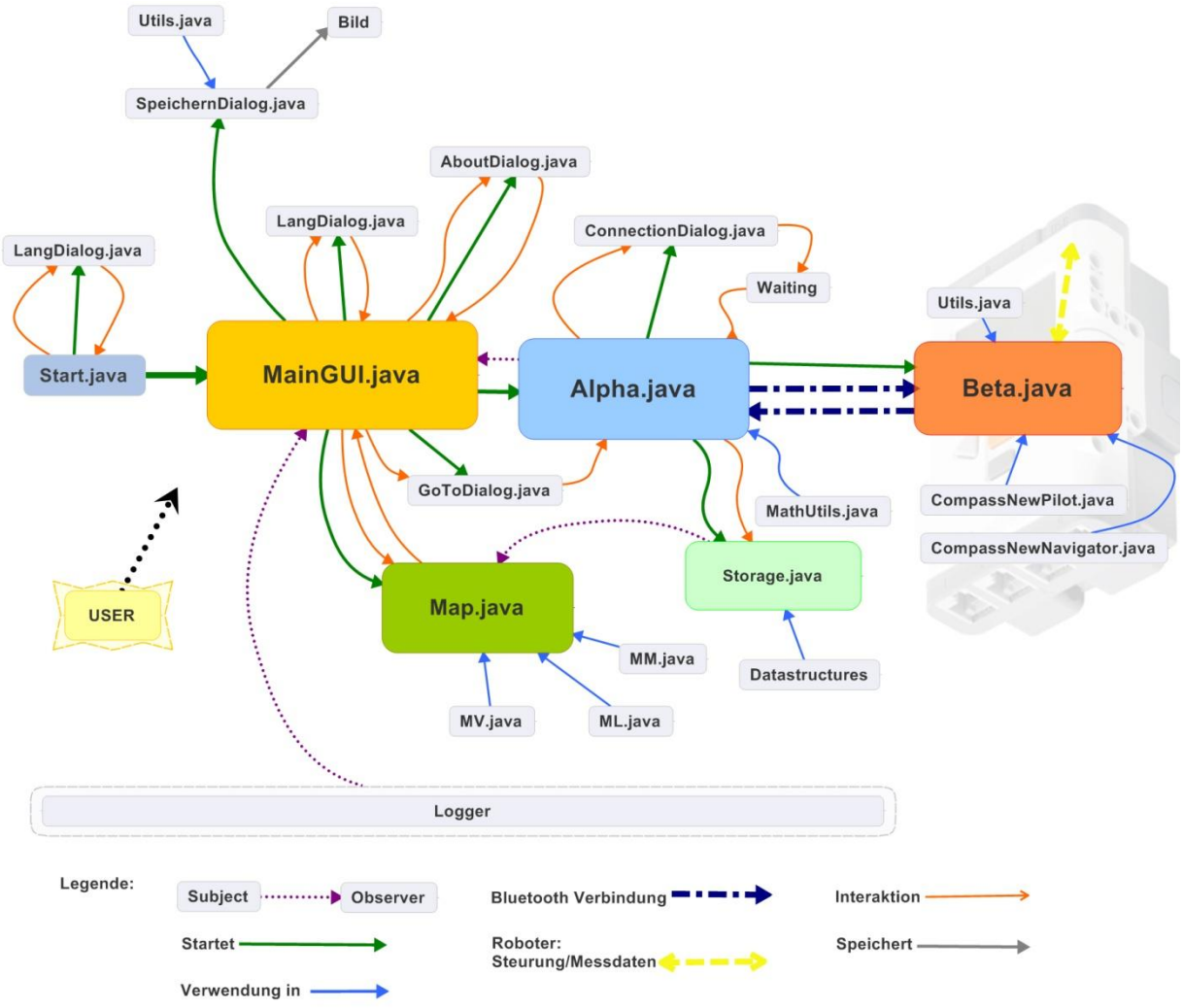


Abb. 46: Programmschema

Das Schema soll einen kurzen Überblick über die Funktionsweise des Programmes geben. Die einzelnen Funktionen werden im übernächsten Kapitel kurz erläutert. Dabei muss erwähnt sein, dass das Schema nicht vollständig ist, um die Übersichtlichkeit zu gewähren.

3.2.2.2 GUI – Graphical User Interface

Mit Hilfe von Screenshots soll in diesem Kapitel die Funktionen des Programmes erläutert werden.

Abb. 47: Splashscreen

Verbindung zum NXT via Bluetooth herstellen. Wenn das Feld Adresse leer gelassen wird, so versucht der Computer der Computer die Verbindung automatisch aufzubauen. Tests auf verschiedenen Systemen ergaben eine Erfolgsquote von 100 %.

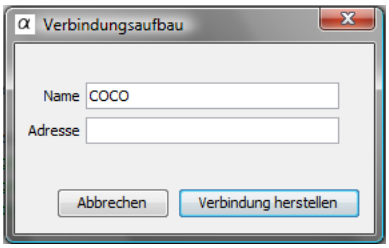


Abb. 48: Verbindungsaufbau

Die Übersicht des Programmes:

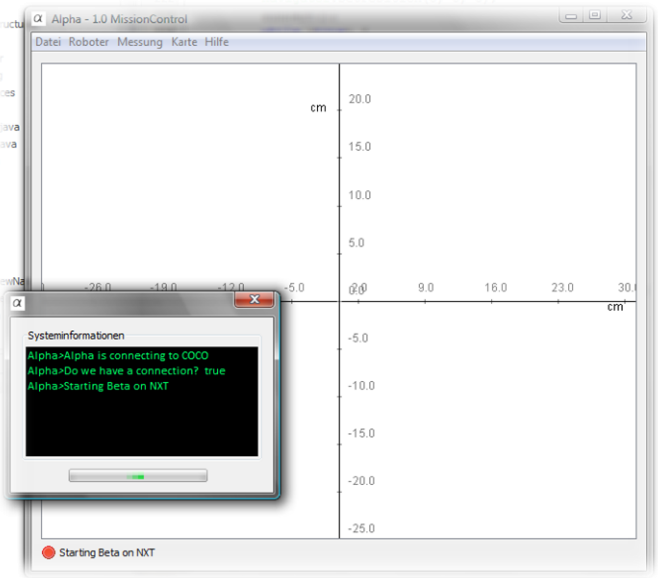


Abb. 50: Programmübersicht

Unten eingeblendet ist der WaitingDialog, der die aktuellen Startfortschritte anzeigt. Unterhalb der der Map befindet sich eine Statuszeile, die die Verbindung zum Roboter anzeigt.

Abb. 49: Verbindung steht

Mit der Funktion „Neustart der Verbindung“, kann die Verbindung mit dem Roboter wiederhergestellt werden.

Die Funktion Kalibrierung ist für eine Eichung des Kompasssensors nötig. Dadurch können permanent vorhandene Magnetfelder, die zum Beispiel von den Elektromotoren erzeugt werden, vernachlässigt werden.

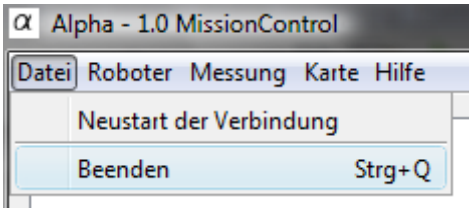


Abb. 51: Menü: Datei

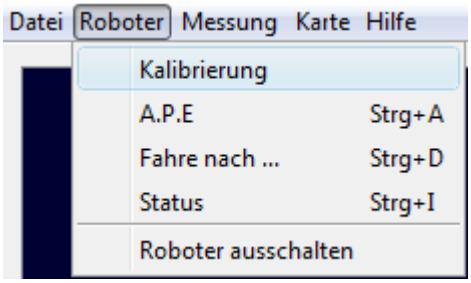


Abb. 52: Menü: Roboter

A.P.E ist eine Testfunktion, die von keiner relevanten Bedeutung für das Projekt ist.

Mit der Methode „Fahre nach…“ navigiert der Roboter zu der gewünschten Position {x, y, Ausrichtung}.

Status gibt eine Übersicht über die Messwerte des Roboters an. Dadurch ist bei einem Problem schnell ersichtlich, welcher Sensor die Ursache ist. Auf diese Weise kann man auch schnell prüfen, ob die Batterien noch genügend geladen sind. Im dümmsten Fall würde nämlich der Kartografiervorgang abgebrochen, weil die Batterien zu schwach sind.

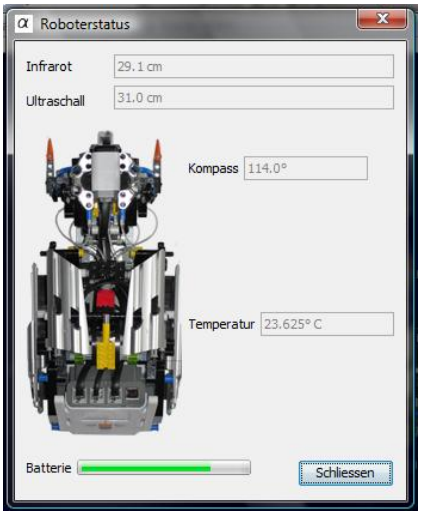


Abb. 53: Status des Roboters

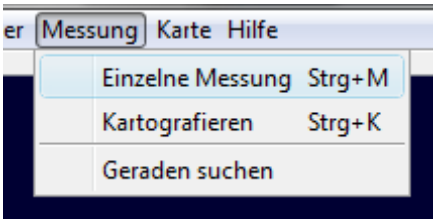


Abb. 54: Menü: Messung

Eine Einzelmessung führt der Roboter an der Stelle aus, wo er sich gerade befindet.

Dadurch entstehen etwa 700 neue Messpunkte.

Bei der Funktion „Kartografieren“ läuft alles vollautomatisch ab. Der Roboter führt immer eine „einzelne Messung“ durch und dann sucht er sich (bzw. der Computer) eine, mehr oder weniger zufällig gewählte, neue Position. Es gibt aber eine Kollisionswarnung-Methode, die verhindert, dass er in einen Gegenstand fährt.

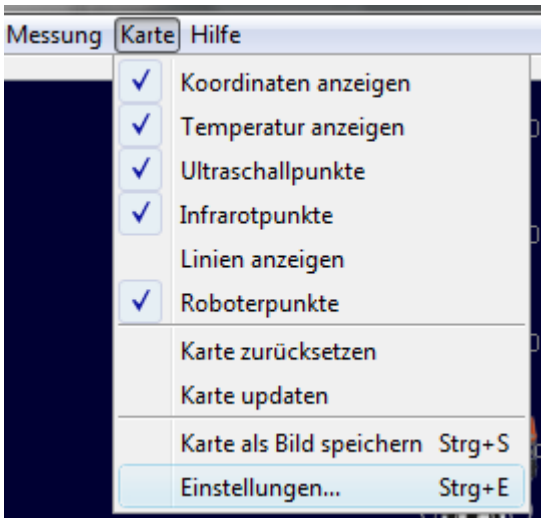


Abb. 55: Menü: Karte

Mit „Geraden suchen“ versucht das Programm neue Interpolationslinien zu finden.

Mit den einzelnen Hacken kann man verändern, was man auf der Map eingezeichnet haben möchte. Dabei sind mit Linien, die berechneten Strecken gemeint, die als Interpolationslinien dienen.

Die Roboterpunkte zeigen, wo sich der Roboter bei den Messungen aufgehalten hatte und wo sich der Roboter zum aktuellen Zeitpunkt befindet.

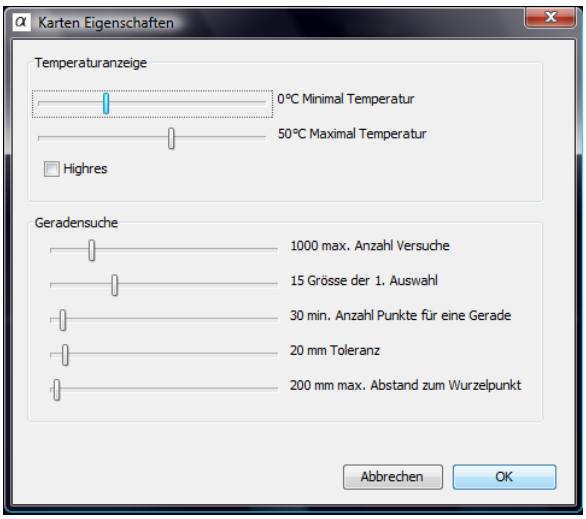


Abb. 56: Einstellungen

Mit der Funktion „Karte zurücksetzen“ hat man wieder den gleichen Zoomfaktor und Ausschnitt wie beim Start des Programms. Die Messpunkte werden dabei aber nicht gelöscht.

Die Karte kann als .jpg oder als .png gespeichert werden. Dabei kann man zusätzlich auch noch die gewünschte Auflösung wählen.

Unter Eigenschaften können verschiedene Parameter definiert werden. Die ersten beiden beeinflussen die Temperaturskala.

Mit der Option „Highres“ kann die Qualität der Temperaturkarte verbessert werden. Die Rechenintensität ist dadurch aber stark erhöht. Die unteren Parameter beeinflussen die Interpolationslinien. Mit einer guten Einstellung dieser Faktoren, dass das Ergebnis deutlich verbessert werden.

Unter Hilfe, kann man zwischen deutscher und englischer Sprache wählen.

Über das Programm gibt Auskunft über die Version.

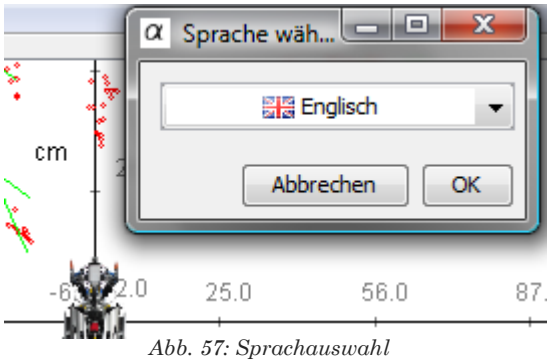


Abb. 57: Sprachauswahl

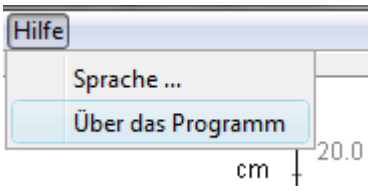


Abb. 58: Menü: Hilfe

3.2.2.3 Programmteile

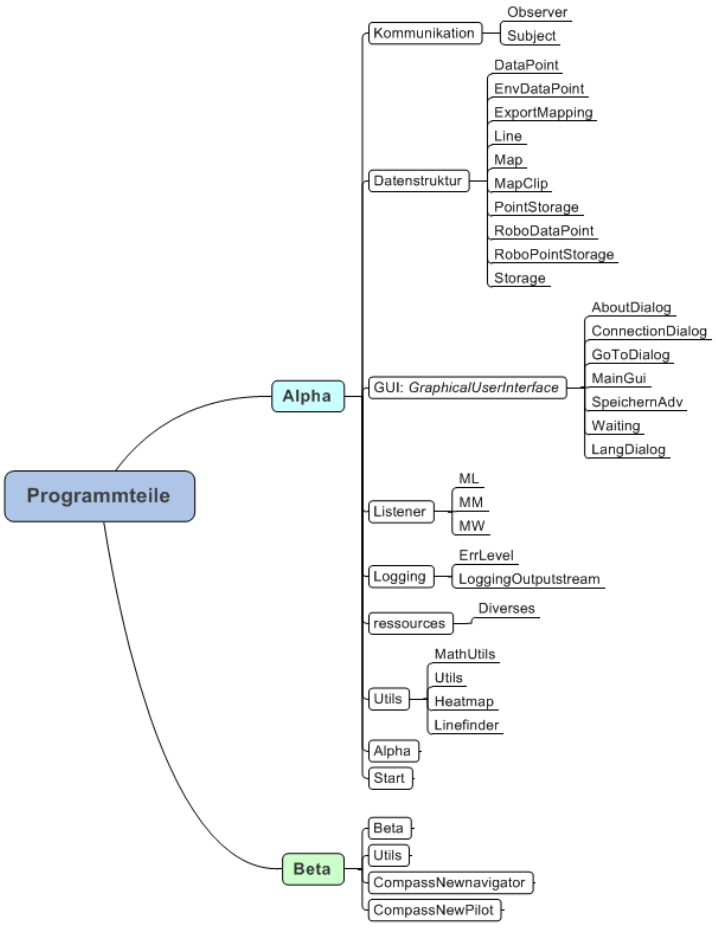


Abb. 59: Wichtigste Programmteile

In diesem Abschnitt wird das Wesentlichste zu den einzelnen Programmteilen erläutert. Für eine genauere Beschreibung ist der Quellcode im Anhang zu konsultieren. In diesem sind auch viele Kommentare enthalten, die die einzelnen Methoden erläutern.

Grob kann das Programm in zwei Teile gegliedert werden:

Alles unter Alpha ist auf dem Computer gespeichert und wird auch auf diesem verwendet.

Die Betabestandteile werden auf den Roboter übertragen und dort ausgeführt.

3.2.2.3.1 Alpha

3.2.2.3.1.1 Alpha

Die Alpha Klasse ist das zentralste Element. Hier fliessen alle Daten zusammen und werden weitergeleitet. Sie sorgt dafür, dass alle Verbindungen korrekt sind und gibt anderenfalls Fehlermeldungen aus. Mit Hilfe dieser Klasse können dem Roboter Befehle gesendet werden und im Gegenzug auch wieder Daten erhalten werden.

Alpha→Beta (Roboter)	Beta (Roboter)→Alpha
0) Stoppt den Roboter 10) Einzelmessung 20) Kompass kalibrieren 70) DriveTo {x/y/angle} 998) Neustarten 999) Ausschalten -1) A.P.E	40) Datenpunkte verarbeiten (Messung fertig) 50) Roboterpunkt lesen {x/y/exact/temp/angle} 60) Datenpunkt lesen {x/y/exact/type} 80) Roboterpunkte verarbeiten 100) Es kommen keine Daten mehr 666) Fehler {code}

Abb. 60: Interaktion Computer - Roboter

3.2.2.3.1.2 Start

Die Klasse Start enthält als einzige Klasse eine Main-Methode und wird beim Starten des Programms als erstes (durch Aufruf der Main-Methode) erstellt. Diese Klasse prüft zuerst die Java-Version. Falls noch keine Sprache festgelegt ist, wird LangDialog.java aufgerufen. Danach zeigt sie den Splashscreen (mit Versionsangabe) an. Nach zwei Sekunden wird eine Instanz von MainGui.java erstellt.

3.2.2.3.1.3 Kommunikation

3.2.2.3.1.3.1 Observer

Eine Klasse kann das Interface Observer implementieren. In diesem Fall kann sie sich bei einer Klasse, die Subject implementiert, „anmelden“, wodurch sie direkt über Änderungen im Subject informiert wird.

3.2.2.3.1.3.2 Subject

Eine Klasse die Subject implementiert, verwaltet eine Liste von Observern, welche dann bei bestimmten Änderungen im Subject benachrichtigt werden.

3.2.2.3.1.4 Datenstruktur

3.2.2.3.1.4.1 DataPoint

Ein Datenpunkt ist eine Datenstruktur, welche die Koordinaten x, y und die Genauigkeit der Messung enthält.

3.2.2.3.1.4.2 EnvDataPoint

Ein Umweltdatenpunkt erweitert „DataPoint“ und enthält zusätzlich die Distanz zum Roboter während der Messung, sowie den verwendeten Sensortyp.

3.2.2.3.1.4.3 ExportMapping

Diese Klasse rendert ein Kartenbild nach festgelegten Attributen.

3.2.2.3.1.4.4 Line
Line ist eine Datenstruktur zur Darstellung einer Strecke.

3.2.2.3.1.4.5 Map
Die Map stellt die gesammelten Daten in einem Koordinatensystem dar. Sie ist zoom und verschiebbar. Zudem erweitert sie Observer und „beobachtet“ das Storage.

3.2.2.3.1.4.6 MapClip
Der Mapclip dient zur Verwaltung des Kartenausschnitts und bietet Methoden an, um verschiedene Umrechnungen Pixel ←→ Koordinaten zu machen.

3.2.2.3.1.4.7 PointStorage
Dient zur Speicherung von beliebig vielen EnvDataPoints.

3.2.2.3.1.4.8 RoboDataPoint
Ein Roboterpunkt erweitert „DataPoint“ und enthält zusätzlich Ausrichtung des Roboters und die Temperatur.

3.2.2.3.1.4.9 RoboPointStorage
Diese Klasse dient der Speicherung der Roboterpunkte.

3.2.2.3.1.4.10 Storage
Das Storage ist die zentrale Datenspeicherung des Programms, hier werden schlussendlich alle gemessenen/berechneten Objekte gespeichert und der Map zur Verfügung gestellt. Das Storage bietet auch eine Methode zur Kollisionswarnung an.

3.2.2.3.1.5 GUI: GraphicalUserInterface

3.2.2.3.1.5.1 MainGui
Das MainGui ist das zentrale User Interface. Mit diesem Fenster, in welches auch die Map eingebettet ist, steuert der Benutzer das ganze Programm und bekommt Informationen zum aktuellen Status des Roboters.

3.2.2.3.1.5.2 AboutDialog
Dieser Dialog gibt einen Hinweis auf die Version und Urheberschaft.

3.2.2.3.1.5.3 ConnectionDialog
Dieser Dialog erlaubt es dem Benutzer eine Verbindung zu einem bestimmten Roboter zu forcieren.

3.2.2.3.1.5.4 GoToDialog
Durch diesen Dialog kann der Roboter angewiesen werden, an einen bestimmten Punkt zu fahren.

3.2.2.3.1.5.5 InformationDialog
Der InformationDialog ermöglicht es dem Benutzer Statusmeldungen des Roboters einzusehen.

3.2.2.3.1.5.6 MapDialog
Einstellungen für die Map

3.2.2.3.1.5.7 SpeichernAdv
In diesem Dialog kann der aktuelle Kartenausschnitt als .jpg oder .png gespeichert werden.

3.2.2.3.1.5.8 WaitingDialog
Der Waiting Dialog wird durch den ConnectionDialog aufgerufen und dient zur Unterhaltung und Information des Benutzer während Alpha versucht eine Verbindung zum Roboter herzustellen.

3.2.2.3.1.5.9 LangDialog
Diese Klasse ermöglicht es dem User, zwischen deutscher und englischer Sprache wählen zu können.

3.2.2.3.1.6 Listener

3.2.2.3.1.6.1 ML
Der MouseListener reagiert auf Mausklicks in der Map. Dadurch wird ein eventuelles Dragging erkannt.

3.2.2.3.1.6.2 MM
Der MouseMotionListener wird gebraucht, um bei einem Dragging den neuen Ausschnitt zu berechnen.

3.2.2.3.1.6.3 MW
Der MouseWheelListener registriert Mausradbewegungen und zoomt entsprechend die Map.

3.2.2.3.1.7 Logging

3.2.2.3.1.7.1 ErrLevel
Eine Hilfsklasse für den LoggingOutputstream, welche die Ströme klassifiziert.

3.2.2.3.1.7.2 LoggingOutputstream
Dient zur Umleitung aller Standard-Outputströme in ein Logfile. Zudem informiert diese Klasse Alpha via Subject/Observer falls ein Outputstream verwendet wurde und gibt den letzten Output aus.

3.2.2.3.1.8 ressources
In diesem Abschnitt sind alle Mediendateien gespeichert, die vom Programm benötigt werden.

3.2.2.3.1.9 Utils

3.2.2.3.1.9.1 MathUtils
Enthält verschieden mathematische Methoden, die für Berechnungen gebraucht werden.

3.2.2.3.1.9.2 Utils
Wird vom SpeicherAdv gebraucht, um die korrekten Bildtypen anzugeben.

3.2.2.3.1.9.3 Heatmap
Diese Klasse ermöglicht es die Temperatur auf der Karte darzustellen.

3.2.2.3.1.9.4 Linefinder
Linefinder sucht Interpolationslinien, die die Realität besser darstellen, als die Punktewolken.

3.2.2.3.2 Beta
Dieser Teil des Programms wird auf den Roboter geladen und dort ausgeführt. Dafür ist eine Bluetoothverbindung notwendig.

3.2.2.3.2.1 Beta
Die Hauptklasse des Betateils enthält alle wichtigen Funktionen für den Roboter. Einerseits wird damit die „In- & Outputstreams“ via die Bluetoothverbindung auf Seiten des Roboters komplettiert. Andererseits sind darin die essentiellen Methoden „Measure“ und „DriveTo“ vorhanden.

Mit Hilfe der „Measure“ Methode kann der Roboter eine vollständige Messung durchführen. Damit gemeint ist eine Messung von -50° bis +50°, die durch das Drehen der Drehplattform ermöglicht wird. Dabei werden bei jedem Ausführen circa 300 Messpunkte, des Infrarotsensors und des Ultraschallsensors, aufgenommen und an Alpha gesendet. Zusätzlich dazu wird auch noch ein Roboterpunkt gesendet, welcher neben den Koordinaten auch die lokale Temperatur enthält.

Die Methode „DriveTo“ ist für die Ausführung der Navigation verantwortlich. Alpha sendet dieser Methode eine neue X Koordinate, Y-Koordinate und die gewünschte Ausrichtung. Danach versucht der Roboter so schnell und so genau wie möglich dieses Ziel zu erreichen. Die Klasse ist ein Ersatz für die eigentlich vorhandene Navigatorklasse, welche von LeJos zur Verfügung gestellt wird. Wesentliche Vorteile sind, dass die eigene Methode viel genauer um einen bestimmten Winkel drehen kann, dass sie während des Fahrens auf Abweichungen prüft und dass sie je nach Möglichkeit den Roboter rückwärtsfahren lässt und auf diese Weise Zeit gespart werden kann.

3.2.2.3.2.2 Utils
Darin gespeichert sind einige ausgelagerte Funktionen von Beta. So zum Beispiel die Definitionen, wie Beta Alpha einen Messpunkt schicken soll oder Funktionen, die die Methode „DriveTo“ verwendet. Dadurch ist eine grössere Übersichtlichkeit in Beta gegeben.

3.2.2.3.2.3 CompassNewnavigator & CompassNewPilot
Dies ist eine Kopie der CompassNavigatorklasse und der CompassPilotklasse von LeJos. Sie wird nach dem Einbau der Methode DriveTo praktisch nicht mehr gebraucht. Sie würde aber je nach Bedarf noch einige nützliche Funktionen enthalten.

3.2.3 Digitale Karte

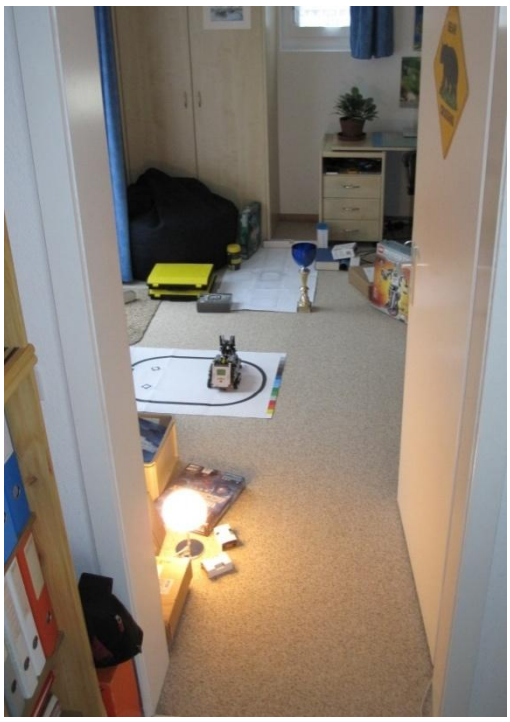


Abb. 61: Realität

Dieses Kapitel soll zeigen, wie gut ein Gebiet kartografiert wird und welche Probleme noch auftraten.

Das Zimmer wurde mit Gegenständen so hergerichtet, dass es als ideales Testgebiet diente. Der Roboter steht auf beiden Fotos genau auf dem Koordinatenursprung.

Bei den Gegenständen ist zu beachten, dass nicht alle glatte Wände haben (zum Beispiel die gelben Kisten) und dass auch die unterschiedlichsten Materialien verwendet wurden.

Die Lichtverhältnisse waren ziemlich optimal, denn es herrschte Tageslicht. Dieses war aber nicht ganz so hell wie es auf den Fotos scheint, da der Blitz der Kamera eingeschaltet war.

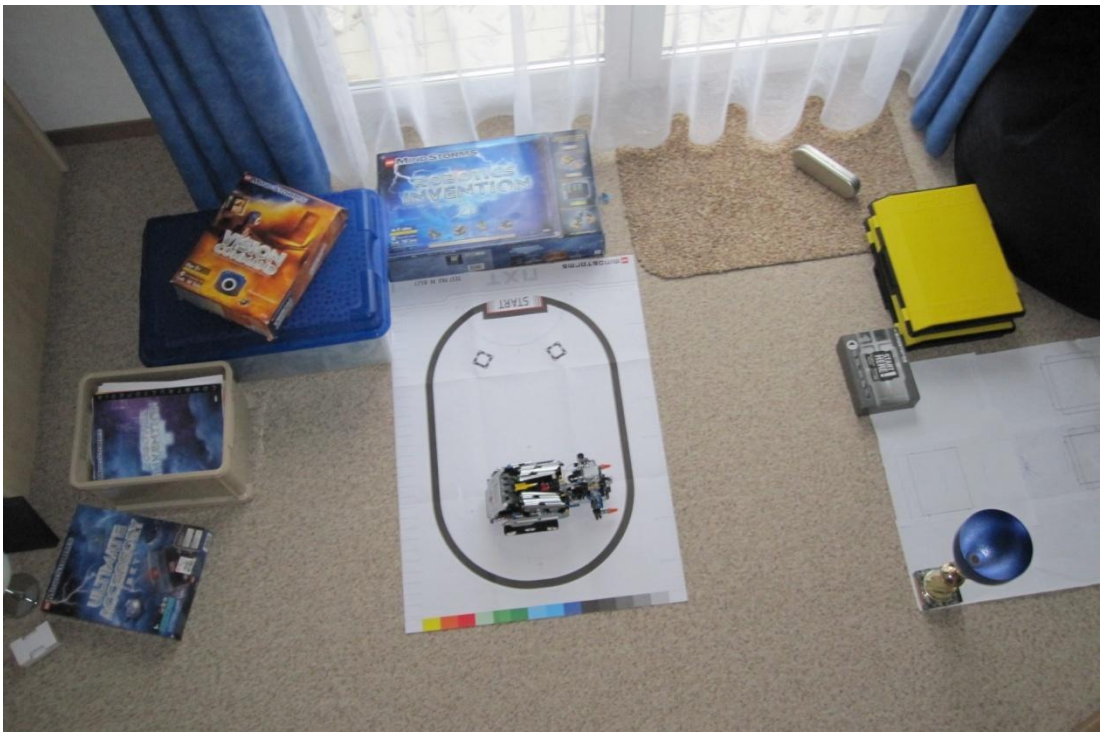


Abb. 62: Realität

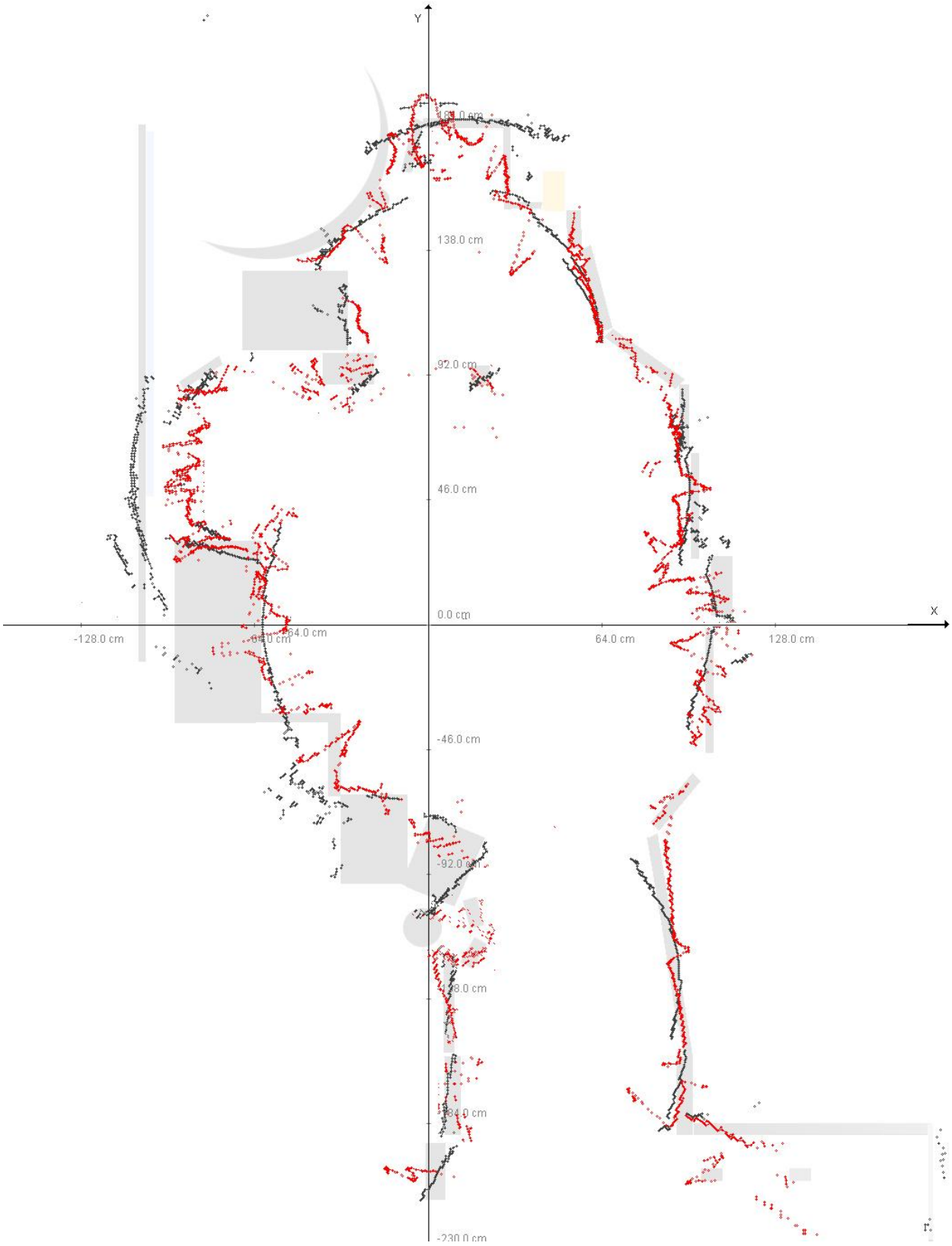


Abb. 63: Generierte, digitale Karte

Auf der Abbildung 63 dargestellt ist das Koordinatensystem mit der Distanzangabe in Zentimetern. Die Infrarotpunkte wurden mit roter Farbe eingezeichnet und die Ultraschallpunkte mit schwarzer Farbe. Der wahre Ort der Gegenstände wurde nachträglich, manuell mit grauer Farbe hinterlegt.

Grundsätzlich ist das Ergebnis des Vorgangs sehr gut. Die Punkte sind zwar nicht immer genau an der Kante der Gegenstände und doch sieht man die Relation zur Wirklichkeit deutlich. Der Vorgang des Kartografierens dauerte etwa eine Viertelstunde.

Die Roboterposition war nun mit der voll funktionsfähigen „DriveTo“ Methode sehr genau. Einzig in der Nähe des Türrahmens war die Ausrichtung des Roboters ein bisschen beeinträchtigt, da dort offensichtlich ein Magnetfeld herrschte. Die Kollisionswarnung an den Roboter funktionierte einwandfrei, denn dieser touchierte nie einen Gegenstand.

Die Messpunkte waren auch ziemlich genau. Allerdings hatte der Roboter je nach Form mehr oder weniger Mühe um einen Gegenstand zu erfassen. Das grösste Problem sind Ecken. Dort ist die Gefahr von Scheinechos am grössten und somit sind die Messpunkte dort am ungenausten. Desweiteren ist das Problem des Winkels (Messrichtung - Kante des Gegenstandes), wie nach den „ausführlichen Tests“[14] prophezeit wurde, eingetroffen. Je steiler der „Messstrahl“ auf einen Gegenstand auftrifft, desto genauer ist das Resultat der Messung.

Einige interessante Details kann man beobachten, wenn man die Karte etwas genauer betrachtet. Zuoberst auf der Karte lag eine Wasserwage aus Metall. Obwohl der Roboter relativ nahe an dieser stand, stimmten die Infrarotpunkte überhaupt nicht. Daraus kann geschlossen werden, dass der Infrarotsensor Mühe hat, die Distanz zu metallischen Gegenständen zu erfassen. Im zweiten Quadranten mass der Ultraschallsensor die Distanz bis zum Fensterrahmen und der Infrarotsensor die Distanz bis zum Vorhang. Dies hat damit zu tun, dass der Ultraschallsensor weiter unten befestigt ist und gerade auf dieser Höhe, zwischen dem Vorhang und dem Boden, ein kleiner Spalt vorhanden ist.

Das Resultat könnte einfach verbessert werden, indem der Roboter das Gebiet eine längere Zeit kartografierte, um zufällige Fehler auszugleichen.

Der Roboter mass während dieses Vorgangs auch die Temperatur. Wegen der langen Dauer waren die Verhältnisse nicht sehr konstant und der Computer hatte aufgrund der vielen Roboterpunkte grosse Mühe, die Temperatur darzustellen. Deshalb wurde der Roboter ein zweites Mal losgeschickt. Dem Roboter wurden manuell die Befehle gegeben, wo er genau durchfahren sollte. Danach wurden die beiden Karten zusammengesetzt.

[14] Kapitel 3.2.1.7.4 Distanzmessungen

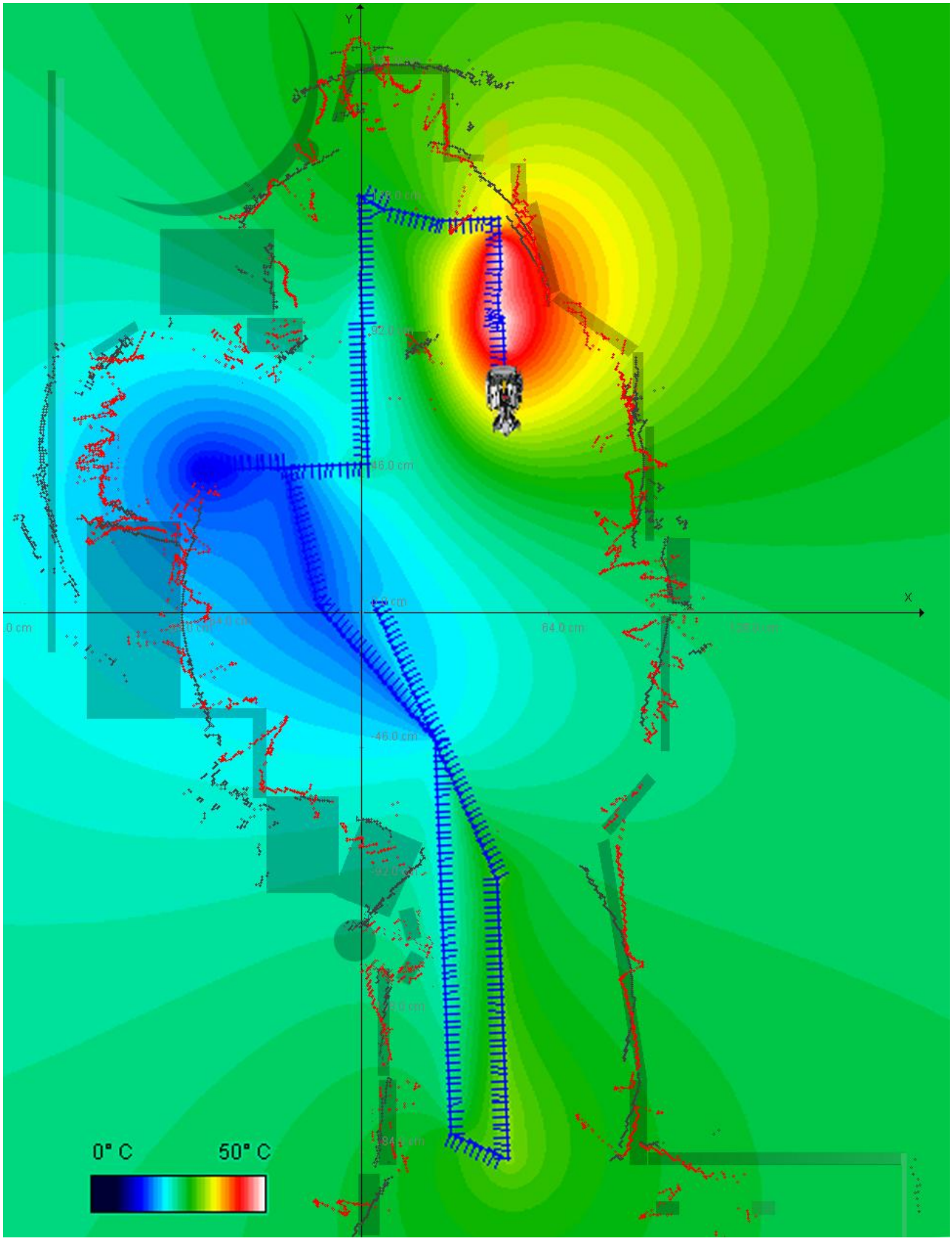


Abb. 64: Digitale Karte mit Temperaturangabe

Auf der linken Seite war das Fenster offen (draussen herrschte circa 0° C) und oben rechts (oranges Feld) lag ein eingeschalteter Föhn. Beim Türrahmen war die Luft deutlich wärmer, da dort nebenan viel elektronische Technik steht (Computer, Netzwerk, …).

Auf der Karte eingezeichnet sind nun auch die Roboterpunkte. Bei jedem Messpunkt mass er einmal die Temperatur. Die Karte ist nur teilweise relevant, da der Roboter die Temperatur nicht flächendeckend gemessen hat. Doch die Karte zeigt die Temperaturunterschiede deutlich und es ist ersichtlich, dass die Interpolation der Temperaturpunkte gut funktioniert.

Die Temperaturerfassung wird durch die Verzögerung vom Temperatursensor beeinträchtigt. Wenn der Roboter zu weite Strecken fährt, ohne dass er Pausen dazwischen einlegt, kann es sein, dass die Messwerte verfälscht werden. Denn der Temperatursensor braucht eine gewisse Zeit, bis er sich an die neue Temperatur angepasst hat. Auf der oben abgebildeten Karte ist dieser Effekt der Verzögerung ersichtlich.

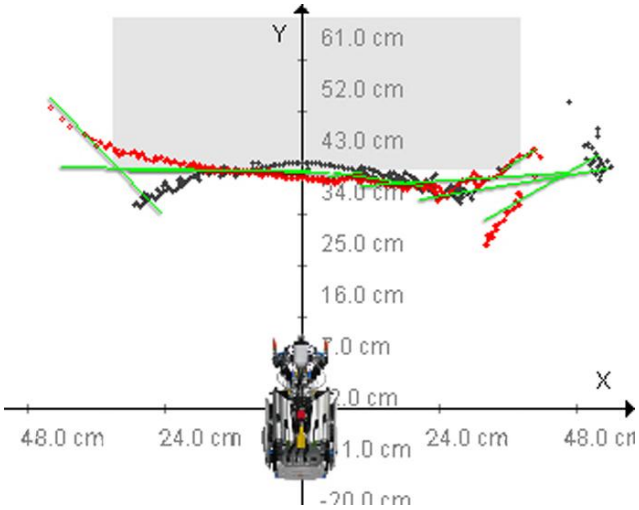


Abb. 65: Interpolationslinien

Bei Bedarf können auch noch Interpolationslinien generiert werden. Durch sie werden gerade Formen genauer angezeigt, da sie Mittelwerte von Messpunkten darstellen. Allerdings werden nur gerade Linien generiert und runde Formen können nicht direkt erkannt werden. Dennoch ist das Resultat meist zufriedenstellend, weil bei nicht linearen Formen häufig keine Linien oder viele kleine erzeugt werden. Mit einstellbaren Parametern, bezüglich der Linienfindung, können die Resultate optimiert werden.

3.3 Diskussion

Die in der Einleitung definierten Ziele wurden erreicht. Es wurde ein Roboter gebaut, der in Interaktion mit dem Computer steht und zusammen mit diesem ein Gebiet, auch ohne menschlichen Einfluss, kartografiert. Das Resultat der so erstellten digitalen Karten ist akzeptabel. In diesem Kapitel soll nun behandelt werden, wie die Qualität der Karte verbessert werden könnte.

Die Software spielt beim Kartografieren eine entscheidende Rolle. Durch geschickte Algorithmen kann einerseits die, für den Vorgang des Kartografierens gebrauchte Zeit reduziert werden und andererseits die Qualität der Karte verbessert werden.

Durch eine gute Erkennung von gewissen Formen wären weniger Messpunkte für die mindestens gleich gute Interpolation erforderlich. Dadurch müsste der Roboter weniger Messungen machen, womit erheblich Zeit gespart werden könnte. Wenn der Roboter Messungen während des Fahrens machen würde, könnte die Geschwindigkeit drastisch erhöht werden. Damit die Qualität darunter nicht leidet (zum Beispiel wegen des Dopplereffektes), werden allerdings sehr komplexe mathematische Funktionen benötigt.

Auch die Qualität der Punkte könnte durch das Programm verbessert werden. Mit Hilfe von verschiedenen Fehlerrechnungen könnte die Verlässlichkeit der Punkte besser bestimmt werden. Somit hätte man eine genauere Vorstellung, wo sich die wahren Gegenstände befinden. Wenn das Programm zum Beispiel abschätzen könnte mit welchem Aufprallwinkel die Messungen gemacht wurden, wäre es möglich die Exaktheit der Punkte besser zu bestimmen und somit auch anders zu gewichten. Für die Temperaturkarte könnte auch ein besserer Algorithmus programmiert werden. Einerseits könnte man die Verzögerung in der Temperaturanpassung ausgleichen und andererseits einen weniger rechenintensiver Vorgang haben. Denn schon bei circa 300 Roboterpunkten hat der Computer etwa 30 Sekunden um das Bild zu berechnen und dies, obwohl der Prozessor extrem schnell ist.

Die Hardware spielt aber auch eine sehr zentrale Rolle beim Kartografieren. Gerade bei diesem Vorgang ist eine extreme Präzision das alles Entscheidende. Deshalb stellt sie grosse Anforderungen an die Technik.

Die erste Grundvoraussetzung ist, dass die virtuelle Roboterposition der reellen entspricht. Dies bedingt einen präzisen Antrieb. Die Motoren müssen die Drehzahl exakt unter Kontrolle haben. Diese Vorgabe ist mit den NXT-Motoren erfüllt. Die genaue Navigation wurde aber durch Reibung gestört. Wenn die unabhängigen Antriebe links und rechts nicht den gleichen Widerstand erfahren, so weicht der Roboter vom Kurs ab. Dieses Problem konnte aber durch eine Fehlerrechnung mit Hilfe des Kompasssensors praktisch behoben werden, so dass die Roboterposition auch noch nach einer gewissen Zeit (Fehler summieren sich auf) erstaunlich genau war. Der Raupenantrieb hat den Nachteil, dass die Rotation um die eigene Achse nicht ganz trivial ist. Der Roboter dreht aufgrund der Reibung nicht genau in einem Punkt. Allerdings haben in Tests die Vorteile gegenüber Rädern überwogen, denn mit Raupen ist eine grössere Bodenhaftung gegeben. Die Raupen können kaum durchdrehen und somit ist die virtuell gefahrene Distanz mit der Realität praktisch übereinstimmend.

Die nächste Bedingung für eine präzise Messung ist die Angabe des Kompasssensors. Wenn die Ausrichtung nicht stimmt, so ist der gemessene Punkt unter Umständen am völlig falschen Ort auf der Karte. Der Hi-Technic Kompasssensor ist sehr sensibel. Er soll die Himmelsrichtung, laut Datenblatt, auf ein zehntel Grad genau bestimmen können. In praktischen Versuchen war ersichtlich, dass diese Behauptung wirklich stimmt. Dass er so sensibel ist hat aber nicht nur Vorteile. Er reagiert auf kleinste elektromagnetische Felder. Wenn er zum Beispiel beim Kartografieren am Handföhn vorbeifuhr, war seine Ausrichtung plötzlich beeinträchtigt. Bei Fahrten durchs ganze Haus musste festgestellt werden, dass Abweichungen von bis zu 10 Grad auftraten, sobald sich der Roboter in der Nähe von elektrischen Geräten befand. Abhilfe für dieses Problem schafft die Kalibrierungsfunktion. Damit kann der Roboter störende Magnetfelder erkennen und bei der Gradbestimmung berücksichtigen. Tests ergaben sehr positive Ergebnisse. Auch an den Orten wo er grosse Abweichungen hatte, konnte er die Ausrichtung nach der Kalibrierung einwandfrei bestimmen. Allerdings müsste der Roboter diese Kalibrierung nach jeder kleinen zurückgelegten Strecke wiederholen, da er dann nicht mehr von der genau gleichen elektromagnetischen Kraft gestört wird. Dies würde den Zeitaufwand enorm erhöhen, da ein Kalibrierungsvorgang etwa 30 Sekunden dauert.

Die dritte Voraussetzung für die Erfassung eines genauen Messpunktes ist die Präzision der Distanzmesssensoren. Der Ultraschallsensor von Lego und der Infrarotsensor von Mindsensors erkennen zwar die Distanz auf einen Gegenstand, der exakt im Lot zur Messrichtung steht, ziemlich genau. Doch dieser ideale Fall tritt leider selten ein. Dadurch stimmen die Distanzen und somit schliesslich der Koordinatenpunkt nicht genau mit der Realität überein. Vor allem die Ecken an den Gegenständen sind ein Problem. Dort ist die Gefahr von Scheinechos riesig, wodurch die Messpunkte zum Teil völlig falsch sind. Die zwei verwendeten Sensoren sind keine Präzisionsinstrumente und somit für eine professionelle Kartografierung unbrauchbar. Die Qualität der Karte könnte mit Hilfe eines Laserdistanzsensors enorm verbessert werden. Geeignete Sensoren kosten aber etwa 5000 Schweizerfranken und mehr. Für diese Arbeit wäre eine solche Anschaffung deshalb nicht verhältnismässig gewesen, da es hauptsächlich ums Prinzip ging. Und dieses funktioniert, auch mit den verwendeten Sensoren, sehr gut.

Auch die Temperaturmessungen könnten hardwaretechnisch verbessert werden, wenn ein anderer Sensor verwendet würde. Dieser müsste sich schneller an die neue Temperatur anpassen.

Abschliessend kann man sagen, dass die Qualität der Karte endlos verbessert werden könnte. Das erreichte Resultat ist, den Umständen entsprechend, sehr berfriedigend.

4 Schlusswort

Die gesetzten Ziele habe ich erreicht. Ich habe ein funktionsfähiges System, welches ein Gebiet kartografieren kann. Doch dies ist nur das oberflächliche Resultat der Arbeit. Viel wichtiger für mich ist der riesige Lernerfolg. Durch diese Arbeit konnte ich viele mathematische Formeln anwenden und ich lernte auch mit sehr komplexen Problemen umzugehen. Auch wenn ich bei der Programmierung gewisse Hilfe benötigte, so lernte ich innerhalb von einem halben Jahr enorm viel und habe dadurch ein nun viel grösseres Verständnis, was „Programmierung“ wirklich ist. Ich tüftelte am Roboter und verbesserte ihn kontinuierlich. Es machte mir viel Spass auf diese Weise auch praktisch zu arbeiten und nicht nur theoretische Überlegungen anzustellen. Ich rechnete von Anfang an mit einem grossen Zeitaufwand für diese Arbeit. Doch schlussendlich war der Aufwand mit mehr als 300 Stunden noch viel grösser als ich erwartet hatte. Ich möchte mich zum Schluss noch herzlichst bei meinem Bruder Daniel Tschudi bedanken, der mir viele Stunden lang, geduldig Java näher brachte und mir mit mathematischen Formeln half. Und mein Dank gilt auch Herrn Dr. Niklaus Emmenegger, der sich viel Zeit nahm, um mich in meiner Arbeit zu unterstützen.

5 Abkürzungsverzeichnis und Glossar

Algorithmus[15]: Unter einem Algorithmus (auch Lösungsverfahren) versteht man eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten.

Applet[16]: Ist ein Computerprogramm, das nicht als eigenständige Applikation betrieben wird. Meist ist damit ein Java-Applet gemeint.

Applikation[17]: Ein Anwendungsprogramm (kurz „Anwendung“, engl. „application software“) ist ein Computerprogramm, das Benutzer anwenden, um eine nützliche Funktion zu erreichen, zum Beispiel Bildbearbeitung, Textverarbeitung, Tabellenkalkulation oder auch Spiele. Aus dem englischen Begriff „application“ hat sich in der Alltagssprache auch die Bezeichnung „Applikation“ für Anwendungsprogramm eingebürgert.

Approximation[18]: Approximation (v. lat.: proximus, -a, -um = der, die, das Nächste) bezeichnet im mathematischen Sinn eine Näherung.

Bluetooth[19]: Bluetooth verbindet als drahtlose Schnittstelle technische Geräte wie Handys, MP3-Player oder Computer miteinander und dient dem Datenaustausch.

Byte-Code[20]: In der Informatik ist Bytecode eine Sammlung von Befehlen für eine virtuelle Maschine. Bei Kompilierung eines Quelltextes mancher Programmiersprachen oder Umgebungen – wie beispielsweise Java – wird nicht direkt Maschinencode, sondern ein Zwischencode, der Bytecode, erstellt. Dieser Code ist in der Regel maschinenunabhängig und im Vergleich zum Quelltext und zu Maschinencode oft relativ kompakt.

CPU[21]: Der Hauptprozessor (englisch central processing unit, CPU) im allgemeinen Sprachgebrauch oft auch nur als Prozessor bezeichnet, ist die zentrale Verarbeitungseinheit (ZVE) eines Computers, die in der Lage ist, ein Programm auszuführen.

Firmware[22]: Unter Firmware (vom engl. „firm“ = fest) versteht man Software, die in elektronische Geräte eingebettet ist. Sie ist zumeist in einem Flash-Speicher, einem EPROM, EEPROM oder ROM gespeichert und durch den Anwender nicht, oder nur mit speziellen Mitteln bzw.

---

[15] Algorithmus: http://de.wikipedia.org/wiki/Algorithmus (Stand 30.11.2009).
[16] Applet: http://de.wikipedia.org/wiki/Applet (Stand 05.12.2009).
[17] Applikation: http://de.wikipedia.org/wiki/Anwendungsprogramm (Stand 05.12.2009).
[18] Approximation: http://de.wikipedia.org/wiki/Approximation (Stand 30.11.2009).
[19] Bluetooth: http://www.pixelplaner.com/de/service/glossar/index.html (Stand 30.11.2009).
[20] Byte-Code: http://de.wikipedia.org/wiki/Byte-Code (Stand 05.12.2009).
[21] CPU: http://de.wikipedia.org/wiki/CPU (Stand02.01.2010).
[22] Firmware: http://de.wikipedia.org/wiki/Firmware (Stand 30.11.2009).

Funktionen austauschbar. Der Begriff leitet sich davon ab, dass Firmware gewissermaßen eine Zwischenstellung zwischen Hardware (also den physikalischen Anteilen eines Gerätes) und der Software (den ggf. austauschbaren Programmen eines Gerätes) einnimmt.

Interpolation[23]: In der numerischen Mathematik bezeichnet der Begriff Interpolation eine Klasse von Problemen und Verfahren. Zu gegebenen diskreten Daten (z. B. Messwerten) soll eine kontinuierliche Funktion (die sogenannte Interpolante oder Interpolierende) gefunden werden, die diese Daten abbildet. Man sagt dann, die Funktion interpoliert die Daten.

Java[24]: Java ist eine objektorientierte Programmiersprache und als solche ein eingetragenes Warenzeichen der Firma Sun Microsystems.

Java-Applet[25]: Ein Java-Applet ist ein Computerprogramm, das in der Programmiersprache Java verfasst wurde und normalerweise in einem Webbrowser ausgeführt wird. Sie wurden eingeführt, um Programme in Webseiten ablaufen lassen zu können, die im Webbrowser (auf der Client-Seite) arbeiten und direkt mit dem Benutzer interagieren können, ohne Daten über die Leitung zum Server versenden zu müssen.

Lego[26]: Die Lego A/S (auch The LEGO Group, offizielle Schreibweise LEGO) ist ein dänisches Unternehmen, das durch die Legosteine, welche mittlerweile als Spielzeugklassiker gelten, bekannt wurde. Dabei handelt es sich um ein Baukastensystem, bei dem bunte Kunststoff-Klötzchen, Zahnräder, Figuren und andere Kleinteile zusammengesteckt werden, um damit Modelle von fast allen erdenklichen Dingen zu bauen. Das Wort Lego stammt vom dänischen "Leg Godt" = "Spiel Gut" ab.

LeJos[27]: leJOS ist ein Java-Betriebssystem für den LEGO Mindstorms RCX (leJOS RCX) und den NXT (leJOS NXJ). Diese Software erlaubt es, die Steuerung von LEGO-Konstruktionen in Java zu programmieren. Dazu wurde ein Teil der Java Virtual Machine auf den RCX/NXT portiert.

Maschinencode[28]: Maschinencode bezeichnet ein System von Instruktionen, die der jeweilige Prozessor eines datenverarbeitenden Systems direkt und ohne Kompilierung ausführen kann. Im Gegensatz zur Assemblersprache oder Hochsprachen handelt es sich um einen für den Menschen kaum verständlichen Binärcode, der nur von Experten für den jeweiligen Code gelesen werden kann und meist mit

---

[23] Interpolation: http://de.wikipedia.org/wiki/Interpolation_%28Mathematik%29 (Stand 30.11.2009).

[24] Java: http://de.wikipedia.org/wiki/Java_%28Programmiersprache%29 (Stand 30.11.2009).

[25] Java-Applet: http://de.wikipedia.org/wiki/Java-Applet (Stand 05.12.2009).

[26] Lego: http://de.wikipedia.org/wiki/Lego (Stand 30.11.2009).

[27] LeJos: http://de.wikipedia.org/wiki/LeJOS (Stand 30.11.2009).

[28] Maschinen-Code: http://de.wikipedia.org/wiki/Maschinencode (Stand 05.12.2009).

speziellen Programmen, so genannten Maschinensprachemonitoren, bearbeitet wird. Aber auch dabei wird der Code selbst in einfachsten Debuggern zwecks besserer Lesbarkeit fast ausschließlich in hexadezimaler Darstellung präsentiert.

Outsourcing[29]: Outsourcing bzw. Auslagerung bezeichnet in der Ökonomie die Abgabe von Unternehmensaufgaben und -strukturen an Drittunternehmen. Es ist eine spezielle Form des Fremdbezugs von bisher intern erbrachter Leistung, wobei Verträge die Dauer und den Gegenstand der Leistung fixieren. Das grenzt Outsourcing von sonstigen Partnerschaften ab.

Roboter[30]: Roboter sind stationäre oder mobile Maschinen, die nach einem bestimmten Programm festgelegte Aufgaben erfüllen.

Thread[31]: Ein Thread (auch: Aktivitätsträger oder leichtgewichtiger Prozess) bezeichnet in der Informatik einen Ausführungsstrang oder eine Ausführungsreihenfolge in der Abarbeitung eines Programms. Ein Thread ist Teil eines Prozesses.

Transponder[32]: Ein Transponder ist ein Funk-Kommunikationsgerät, das eingehende Signale aufnimmt und automatisch beantwortet bzw. weiterleitet. Der Begriff Transponder ist zusammengesetzt aus den Begriffen Transmitter und Responder.

Ultraschall[33]: Mit Ultraschall (oft als 'US' abgekürzt) bezeichnet man Schall mit Frequenzen, die oberhalb des vom Menschen wahrgenommenen Bereiches liegen. Das umfasst Frequenzen zwischen 20 kHz (obere Hörschwelle) und 1 GHz. Schall mit noch höherer Frequenz wird als Hyperschall bezeichnet, bei Frequenzen unterhalb des für Menschen hörbaren Frequenzbereichs spricht man dagegen von Infraschall.

---

[29] Outsourcing: http://de.wikipedia.org/wiki/Outsourcing (Stand 30.11.2009).

[30] Roboter: http://de.wikipedia.org/wiki/Roboter (Stand 30.11.2009).

[31] Thread: http://de.wikipedia.org/wiki/Thread_%28Informatik%29 (Stand 05.12.2009).

[32] Transponder: http://de.wikipedia.org/wiki/Transponder (Stand 05.12.2009).

[33] Ultraschall: http://de.wikipedia.org/wiki/Ultraschall (Stand 30.11.2009).

6 Literatur- und Abbildungsverzeichnis

6.1 Material und Methoden

6.1.1 Lego Mindstorms

[1] Legomindstorms: http://de.wikipedia.org/wiki/Lego_Mindstorms (Stand 30.11.2009).

[2] NXT: http://de.wikipedia.org/wiki/NXT (Stand 30.11.2009).

[3] Offizielle Lego Website: http://mindstorms.lego.com/eng/Overview/default.aspx (Stand 30.11.2009).

[4] Sensoren: http://shop.educatec.ch/legoeducationaldivision/legomindstormseducation/legosensoren/index.php (30.11.2009).

Abbildungen:

[Abb. 1] NXT: http://www.nxt-in-der-schule.de/bilder/nxt.jpg (Stand 30.11.2009).

[Abb. 2] Motoren: http://upload.wikimedia.org/wikipedia/de/3/38/Lego_mindstorms_nxt_motor.jpg (Stand 02.12.2009).

[Abb. 3] Temperatursensoren: http://shop.educatec.ch/images/9749temperaturesensor01.jpg (Stand 30.11.2009).

[Abb. 4] 6 Sensoren:

a. Tastsensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/downloadFA5D76AD21287C61FC20BEFD6B7E0549.jpg

b. Lichtsensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/download5E5F0131D6E1642499204ADF3429031C.jpg

c. Schall/soundsensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/download8D5C71F12BEC3A7CE24950619D079451.jpg

d. Ultraschallsensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/downloadAC25F70199CFD13268DDEC501807A543.jpg

e. Kompasssensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/download3F8559A18725C36437386ED9875B1D7F.jpg

f. Beschleunigungssensor: http://cache.lego.com/upload/contentTemplating/Mindstorms2Products/otherfiles/download674E428CB0D379B91A893115A3CDEB5F.jpg

g. Stand (02.12.2009).

[Abb. 5] Farbsensor: http://shop.educatec.ch/images/colorsensorw1.jpg (Stand 02.12.2009).

[Abb. 6] RFID-Sensor: http://shop.educatec.ch/images/lego20sensororangekarton11.jpg (Stand 02.12.2009).

[Abb. 7] Kamera von mindsensors.com: http://www.mindsensors.com/images/pagemaster/NXTCamv2w250.png (Stand 02.12.2009).

[Abb. 8] Infrarotsensor von mindsensors.com: http://www.mindsensors.com/images/products/DIST-Nx-med-s.png (Stand 02.12.2009).

[Abb. 9] LineLeader von mindsensors.com: http://www.mindsensors.com/images/pagemaster/LineLeaderw400.jpg (Stand 02.12.2009).

[Abb. 10] Abb. 10, Pneumatiksensor von mindsensors.com: http://www.mindsensors.com/images/pagemaster/pps35w400.jpg (Stand 02.12.2009).

6.1.2 Erfassen von Objektpunkten

[5] Ultraschall: http://de.wikipedia.org/wiki/Ultraschall (Stand 30.11.2009).

[6] Ultraschall: http://projektlabor.ee.tu-berlin.de/projekte/roboter/downloads/referate/ultraschall/referat.pdf (Stand 30.11.2009).

[7] Ultraschallsensor & Infrarotsensor: http://www.mikrocontroller.net/mc-project/Pages/Robotik/Sensoren/sensoren.html (Stand 30.11.2009).

Abbildungen:

[Abb. 11] Tastsensor: http://content.heutink.nl/catalog250/090089.JPG (Stand 30.11.2009).
[Abb. 12] Ultraschallprinzip: http://upload.wikimedia.org/wikipedia/de/5/5d/Dibujo_Prinzip_Ultraschall.PNG (Stand 30.11.2009).
[Abb. 13] Ultraschallsensor: http://wiki.mindstormsforum.de/lib/exe/fetch.php?cache=&media=ultraschall.jpg (Stand 30.11.2009).
[Abb. 14] Infrarotsensor: http://www.mindsensors.com/images/products/DIST-Nx-med-s.png (Stand 30.11.2009).

6.1.3 Programmierung

[8] LeJos: http://de.wikipedia.org/wiki/LeJOS (Stand 30.11.2009).

6.2 Resultate

6.2.1 Evolution des Roboters

Abbildungen:

[Abb. 15] Prototyp I, Eigenproduktion.
[Abb. 16] Prototyp I, Eigenproduktion
[Abb. 17] Prototyp I, Eigenproduktion
[Abb. 18] Prototyp II α, Eigenproduktion.
[Abb. 19] Prototyp II α, Eigenproduktion.
[Abb. 20] Prototyp II α, Eigenproduktion.
[Abb. 21] Prototyp II α, Eigenproduktion.
[Abb. 22] Prototyp II α, Eigenproduktion.
[Abb. 23] Prototyp II β, Eigenproduktion.
[Abb. 24] Prototyp II β, Eigenproduktion.
[Abb. 25] Prototyp II β, Eigenproduktion.
[Abb. 26] Prototyp II γ, Eigenproduktion.
[Abb. 27] Prototyp II γ, Eigenproduktion.
[Abb. 28] Fehler in der Testkarte, Eigenproduktion.
[Abb. 29] X-Achse Winkel, Y-Achse Distanz, Eigenproduktion.
[Abb. 30] Realität des Kartenausschnittes, Eigenproduktion.
[Abb. 31] Neue Testkarte, Eigenproduktion.
[Abb. 32] Prototyp III, Eigenproduktion.
[Abb. 33] Prototyp III, Eigenproduktion.
[Abb. 34] Prototyp III, Eigenproduktion.
[Abb. 35] Verwendeter Programmcode, Eigenproduktion.
[Abb. 36] Verwendeter Programmcode, Eigenproduktion.
[Abb. 37] Versuchsanordnungen, Eigenproduktion.
[Abb. 38] Versuchsanordnungen, Eigenproduktion.
[Abb. 39] Versuchsanordnungen, Eigenproduktion.
[Abb. 40] Versuchsanordnungen, Eigenproduktion.
[Abb. 41] Verwendeter Programmcode, Eigenproduktion.
[Abb. 42] Roboter in Aktion, Eigenproduktion.
[Abb. 43] Roboter in Aktion, Eigenproduktion.
[Abb. 44] Endroboter, Eigenproduktion.
[Abb. 45] Endroboter, Eigenproduktion.

Diagramme:

[Dia. 1] VI: Zurücklegen von 50 Zentimetern, Eigenproduktion.
[Dia. 2] VII: Zurücklegen von 50 Zentimetern, Eigenproduktion.
[Dia. 3] Zurücklegen von 90 Zentimetern, Eigenproduktion.
[Dia. 4] Rotation um +/- 90°, Eigenproduktion.
[Dia. 5] Rotation um +/- 180°, Eigenproduktion.
[Dia. 6] Von A nach B, Eigenproduktion.
[Dia. 7] Von B nach A, Eigenproduktion.
[Dia. 8] Farbabhängigkeit, Eigenproduktion.
[Dia. 9] Lichtintensität/Distanz, Eigenproduktion.
[Dia. 10] Grad des Gegenstandes, Eigenproduktion.

6.2.2 Programm

Abbildungen:

[Abb. 46] Programmschema, Eigenproduktion.
[Abb. 47] Splashscreen, Eigenproduktion.
[Abb. 48] Verbindungsaufbau, Eigenproduktion.
[Abb. 49] Verbindung steht, Eigenproduktion.
[Abb. 50] Programmübersicht, Eigenproduktion.
[Abb. 51] Menü: Datei, Eigenproduktion.
[Abb. 52] Menü Roboter, Eigenproduktion.
[Abb. 53] Status des Roboters, Eigenproduktion.
[Abb. 54] Menü Messungen, Eigenproduktion.
[Abb. 55] Menü Karte, Eigenproduktion.
[Abb. 56] Einstellungen, Eigenproduktion.
[Abb. 57] Sprachauswahl, Eigenproduktion.
[Abb. 58] Menü: Hilfe, Eigenproduktion.
[Abb. 59] Programmteile, Eigenproduktion.
[Abb. 60] Interaktion Computer – Roboter, Eigenproduktion.

6.2.3 Digitale Karte

Abbildungen:

[Abb. 61] Realität, Eigenproduktion.
[Abb. 62] Realität, Eigenproduktion.
[Abb. 63] Generierte, digitale Karte, Eigenproduktion.
[Abb. 64] Digitale Karte mit Temperaturangabe, Eigenproduktion.
[Abb. 65] Interpolationslinien, Eigenproduktion.

6.3 Glossar

[15] Algorithmus: http://de.wikipedia.org/wiki/Algorithmus (Stand 30.11.2009).
[16] Applet: http://de.wikipedia.org/wiki/Applet (Stand 05.12.2009).
[17] Applikation: http://de.wikipedia.org/wiki/Anwendungsprogramm (Stand 05.12.2009).
[18] Approximation: http://de.wikipedia.org/wiki/Approximation (Stand 30.11.2009).
[19] Bluetooth: http://www.pixelplaner.com/de/service/glossar/index.html (Stand 30.11.2009).
[20] Byte-Code: http://de.wikipedia.org/wiki/Byte-Code (Stand 05.12.2009).
[21] CPU: http://de.wikipedia.org/wiki/CPU (Stand02.01.2010).
[22] Firmware: http://de.wikipedia.org/wiki/Firmware (Stand 30.11.2009).
[23] Interpolation: http://de.wikipedia.org/wiki/Interpolation_%28Mathematik%29 (Stand 30.11.2009).
[24] Java: http://de.wikipedia.org/wiki/Java_%28Programmiersprache%29 (Stand 30.11.2009).
[25] Java-Applet: http://de.wikipedia.org/wiki/Java-Applet (Stand 05.12.2009).
[26] Lego: http://de.wikipedia.org/wiki/Lego (Stand 30.11.2009).
[27] LeJos: http://de.wikipedia.org/wiki/LeJOS (Stand 30.11.2009).
[28] Maschinen-Code: http://de.wikipedia.org/wiki/Maschinencode (Stand 05.12.2009).

[29] Outsourcing: http://de.wikipedia.org/wiki/Outsourcing (Stand 30.11.2009).
[30] Roboter: http://de.wikipedia.org/wiki/Roboter (Stand 30.11.2009).
[31] Thread: http://de.wikipedia.org/wiki/Thread_%28Informatik%29 (Stand 05.12.2009).
[32] Transponder: http://de.wikipedia.org/wiki/Transponder (Stand 05.12.2009).
[33] Ultraschall: http://de.wikipedia.org/wiki/Ultraschall (Stand 30.11.2009).

6.3.1 Anhang

[34] Java: Dr. T. Kröckertskothen, „Java 2, Grundlagen und Einführung“, RRZN, Hannover, 2006, S. 14-19.

7 Anhang

7.1 Java[34]

7.1.1 Überblick

Die Programmiersprache Java wurde von Grund auf neu entworfen. Sie sollte möglichst einfach sein, bewährte Merkmale vorhandener Sprachen beinhalten, dabei aber auf fehleranfällige und unnötige komplexe Bestandteile verzichten. Anwendungsbereiche der neuen Sprache sollten insbesondere moderne vernetzte Systeme sein.

Ziel:

Java ist eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, performante, nebenläufige und dynamische Programmiersprache

(„...simple, obejct-oriented, distributed, interpreted, robust, secure, architecturee neutral, portable, high-performance, multithreaded and dynamic language“).

Einfach:

Um dieses Ziel zu erreichen, sollte die Sprache möglichst wenige grundlegende Sprachkonstrukte haben. Es wurden bewährte Konstrukte anderer objektorientierter Sprachen wie C++ übernommen, auf komplexe und fehleranfällige Sprachkonstrukte wie zum Beispiel Mehrfachvererbung wurde verzichtet. Die Syntax von Java ist weitgehend an die Sprachen C und C++ angelehnt. Trotz der Einfachheit in der Struktur ist Java eine vollständige und sehr leistungsfähige Programmiersprache.

Es muss aber auch erwähnt werden, dass sich die einfache Struktur von Java erst bei voller Kenntnis der objektorientierten Konzepte erschliesst und Java sich dem unerfahrenen Programmieranfänger durchaus als sehr komplex und überhaupt nicht einfach darstellen kann.

[34] Java: Dr. T. Kröckertskothen, „Java 2, Grundlagen und Einführung“, RRZN, Hannover, 2006, S. 14-19.

Objektorientiert:

Java ist vollständig objektorientiert. Es sind keinerlei Kompromisse in Richtung prozeduraler Sprachelemente zugelassen. Ausser den wenigen Grunddatentypen für Zahlen, Zeichen und Wahrheitswerte sind alle anderen Daten in Java Objekte. Es gibt keine Unterprogramme oder Prozeduren in Java, sondern nur noch Methoden, die zwingend in Klassen definiert sind.

Objektorientierte Programmierung (OOP) ist ein modernes Programmierkonzept und für komplexe Anwendungen heutzutage unerlässlich. Das klare objektorientierte Konzept von Java ist sicher auch einer der Gründe dafür, dass Java zunehmend in der Informatik-Ausbildung eingesetzt wird.

Verteilt:

Java wurde von Anfang an mit dem Ziel entworfen, verteilte Anwendungen zu unterstützen, das heisst Anwendungen, die im Datennetz verteilte Ressourcen nutzen. Zu Java gehören standardmässig Programmierschnittstellen für die Datenkommunikation im Internet (Socket-Kommunikation über TCP/IP). Java-Programme können www-Seiten lesen, Dateien im Internet öffnen und bearbeiten (wenn entsprechende Sicherheitsregeln beachtet werden), und es lassen sich vollständige Client-Server-Applikationen in Java realisieren.

Interpretiert:

Bei Java erzeugt der Compiler normalerweise einen sogenannten Byte-Code (siehe dazu „5.1.6.1.2 Java-Byte-Code und die Virtuelle Maschine“). Um den Java-Byte-Code auszuführen, ist ein Java-Interpreter notwendig, der den Byte-Code interpretiert.

Vorteil des Java-Byte-Codes ist, dass ein einmal erzeugter Java-Byte-Code ohne weitere Änderungen auf jedem beliebigen Rechnersystem mit einem Java-Interpreter ausgeführt werden kann. Der Java Byte-Code ist plattformunabhängig. Nachteil davon ist aber, dass dadurch im Byte-Code systemspezifische Vorteile nicht genutzt werden können.

Robust

Getestete und für fehlerfrei befundene Java-Programme sollen robust und fehlerfrei im Anwendungsbereich eingesetzt werden können. Der aus anderen Programmiersprachen bekannte Effekt, dass ein Programm viele Male korrekt abläuft und dann plötzlich abstürzt, weil zum Beispiel Daten aus falsch belegten Speicherbereichen gelesen oder Grenzen für Datenbereiche überschritten werden, soll in Java vermieden werden. Dies wird erreicht durch strikte Regeln und viele Restriktionen. Die Anforderungen an Robustheit und Sicherheit begrenzen aber nicht nur Fehlerquellen, sondern auch Freiheiten bei der Programmierung.

Sicher

Das Sicherheitskonzept wurde mit jeder neuen Version des JDKs verbessert, womit Java-Anwendungen als ziemlich sicher eingestuft werden können.

Grundsätzlich wird unterschieden zwischen Java-Programmen die auf dem lokalen Rechner ausgeführt werden (local code) und Programmen die übers Internet geladen werden (remote code). Letztere beinhalten vor allem Applets die als nicht vertrauenswürdig eingestuft werden (not trusted) und haben somit sehr eingeschränkte Zugriffsmöglichkeiten.

Architekturneutral

Einmal erstellte und übersetzte Programme sind auf jeder Rechner-Architektur ausführbar, auf der ein Java-Laufzeitsystem vorhanden ist. Ein beispielsweise auf Windows 7 erstelltes Java-Programm kann ohne weitere Änderungen auf Mac OS X verwendet werden. Dieser grosse Vorteil wird mittlerweile nicht nur mehr für die Applets im Internet verwendet, sondern auch für grosse Anwender-Projekte, die so ohne weiteren Anpassungsaufwand auf allen Plattformen verwendet werden können.

Portabel

Die Sprachdefinition von Java erfüllt den Anspruch, keinerlei systemabhängige Elemente zu enthalten. So sind unter anderem die Grössen aller Grunddatentypen in Java systemunabhängig in der Sprache festgeschrieben. Java-Programme sind daher sowohl als Quelltext als auch als Byte-Code auf jede Rechnerplattform übertragbar (portierbar). Protabilität ist ein weitergehender Anspruch als die Architekturneutralität der übersetzten Programme.

Performant

Unter Performanz ist hier nicht die Geschwindigkeit gemeint, denn diese wird im Vergleich zu ähnlichen Programmen in zum Beispiel C langsamer sein. Dort wird nämlich der Quelltext in systemoptimierten Binärcode umgewandelt und auch die Sicherheitsüberprüfung in Java kostet Rechnerressourcen. Allerdings muss beachtet werden, dass die meisten Programme eine grafische Oberfläche haben und so der Benutzer die Gesamtausführungszeit massgeblich beeinflusst. Ausserdem sind mit den neusten Java-Compilern und Laufzeitsystemen kaum noch mit Performanzeinbussen im Vergleich zu C++ und ähnlichen Programmiersprachen zu rechnen.

Hier wird unter Performanz die Gesamtzeit der Programmentwicklung von der Erstellung bis zur fertigen Laufzeitversion verstanden. Durch die Einfachheit, Objektorientierung und Robustheit kann bei Java die Entwicklungszeit erheblich verkürzt werden.

Nebenläufig

Unter Nebenläufigkeit versteht man die Fähigkeit, dass mehrere Programmteile parallel ablaufen. Die Programmteile laufen dann in sogenannten Threads ab.

Multithreading-Fähigkeit ist zum Beispiel eine wesentliche Voraussetzung für Animationen. Denn während die Animation auf dem Bildschirm blinkt, sollte das System nicht für andere Tätigkeiten blockiert sein.

Dynamisch

Damit gemeint ist die dynamische Anpassung von Programmen während der Laufzeit. Java kann zum Beispiel Klassen erst dann laden, wenn sie auch tatsächlich vom Programm benötigt werden (dynamisches Binden von Klassen)

7.1.2 Java-Byte-Code und die Virtuelle Maschine

Die im vorherigen Abschnitt genannten Eigenschaften architekturneutral, portabel und verteilt lassen sich auch zusammenfassen als:

Write once – run everywhere.

Einmal geschriebene Java-Programme sind überall ausführbar. Realisiert wird dieser Anspruch durch den Java-Byte-Code und die virtuelle Java-Maschine (Java-VM oder kurz auch nur VM).

Java-Byte-Code wird vom Java-Compiler erzeugt und in Dateien mit der Endung *.class* gespeichert. Für die Programmierung bedeutet dies, dass das Java-Programm zunächst als Quelltext im Editor erstellt wird und in einer Quelltextdatei (mit der Endung *.java*) gespeichert wird. Die Quelltext-Datei wird dann dem Java-Compiler übergeben, und dieser erzeugt dann die Datei mit dem Byte-Code. Java-Byte-Code-Dateien können Applet oder Applikationen enthalten. Jede Byte-Code-Datei enthält eine übersetzte Klasse.

Java-Byte-Code ist architekturneutral, das heisst, der Java-Byte-Code enthält noch keinerlei maschinenspezifische Anteile. Java-Byte-Code-Dateien können daher auch nicht direkt vom Rechnersystem ausgeführt werden, sondern benötigen zur Ausführung die virtuelle Java-Maschine (Java-VM).

Die Java-VM sorgt für die Ausführung von Java-Programmen auf dem jeweiligen Rechnersystem (PC mit Windows, Linux, Workstation mit Solaris, etc.). Die Java-VM kann als Schnittstelle zwischen dem architekturneutralen Java-Byte-Code und der spezifischen Hard- und Software des ausführenden Systems angesehen werden.

Die Java-VM muss auf dem Rechnersystem vorhanden sein, sie wird in der Regel als Software installiert. Eine Java-VM ist zum Beispiel Bestandteil der Entwicklungsumgebung SDK.

Die Java-VM führt folgende Schritte durch:

- Der Byte-Code wird geladen (Byte-Code-Loader) und der Byte-Code-Loader sucht und lädt gegebenenfalls weitere im Programm benötigte Klassen aus Bibliotheken (Klassenbibliotheken, APIs)
- Der Byte-Code wird auf Vertösse gegen die Java-Sprachregeln untersucht (Byte-Code-Verifizierung).
- Abschliessend wird der Byte-Code interpretiert und als maschinenspezifischer Code auf dem System ausgeführt.