ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Claudia Frischknecht & Thomas Other

LEGO Mindstorms NXT
– Next Generation –

Semesterarbeit SA-2006.18
Sommersemester 2006
07. Juli 2006

Betreuer: Christian Plessl
Co-Betreuer: Andreas Meier
Professor: Dr. Lothar Thiele

Zusammenfassung

In dieser Arbeit werden die einzelnen Komponenten der Vorabversion von „LEGO Mindstorms NXT“ untersucht, um Kenntnisse über die Genauigkeit und Zuverlässigkeit der Sensoren und Motoren und deren Zusammenspiel mit dem NXT-Baustein zu erhalten.
Auf den gewonnenen Erkenntnissen basierend wird mit einem selbst konstruierten Roboter ein Projekt durchgeführt, welches das Potential und die Limiten der Sensoren und Motoren ausreizt. Das Ziel dieses Projekts ist die Erforschung eines a priori unbekannten Raumes. Das Problem lässt sich aufteilen in die Bewegungsplanung, die Kollisionsverhinderung und die Konstruktion einer Karte aus den gewonnen Sensordaten. Anhand von Prototypen werden diese Aufgaben mittels „Next Byte Codes“, einer Programmiersprache mit Assemblersyntax, einzeln gelöst und schrittweise vereint.
Im Vergleich zur älterern RCX Version ist der leistungsstärkere NXT dank eines neuen Ultraschallsensors in der Lage, Objekte in Entfernungen von bis zu 150 cm mit einem maximalen Fehler von ± 3 cm zuverlässig zu erkennen. Eine Ausnahme bildet der Bereich zwischen 30 cm und 50 cm, in dem der Sensor mehrfach den falschen Wert 48 cm liefert. Neu sind auch die in den Motoren integrierten Rotationssensoren, die eine präzisere Steuerung ermöglichen.
Das für den Datenaustausch mit dem NXT-Baustein verwendete LMS2006-Protokoll wird anhand eines Programms zur Kommunikation mittels Bluetooth untersucht.

Inhaltsverzeichnis

1 Vorwort **6**
- 1.1 Einführung . . . 6
- 1.2 Persönliche Motivation . . . 6
- 1.3 Struktur . . . 7

2 LEGO Mindstorms (Grundlagen) **8**
- 2.1 Historischer Hintergrund . . . 8
- 2.2 NXT Generation . . . 9
  - 2.2.1 Vergleich zu RCX . . . 9
  - 2.2.2 Produktinhalt – Angaben des Herstellers . . . 10
  - 2.2.3 LEGO Software – LabView . . . 12
  - 2.2.4 NXT Byte Codes . . . 12

3 Versuche mit dem NXT-Kit **14**
- 3.1 Konstruktionen . . . 14
  - 3.1.1 TriBot . . . 14
  - 3.1.2 Humanoid Alpha Rex . . . 15
- 3.2 Technische Spezifikationen . . . 15
- 3.3 Statische Sensorenversuche . . . 20
  - 3.3.1 Tastsensor . . . 20
  - 3.3.2 Tonsensor . . . 20
  - 3.3.3 Lichtsensor . . . 20
  - 3.3.4 Ultraschallsensor . . . 23
- 3.4 Dynamische Sensorversuche . . . 28
- 3.5 Motorenversuche . . . 31
- 3.6 Oszilloskop . . . 33
  - 3.6.1 Analoge Sensoren . . . 34
  - 3.6.2 Digitaler Sensor . . . 35
  - 3.6.3 Motoren . . . 36
- 3.7 Kommunikation via Bluetooth . . . 36
  - 3.7.1 LMS2006 - LEGO Protokoll . . . 36
  - 3.7.2 Bluetoothkommunikation mit Perl . . . 37

4 Projekt **38**
4.1 Planung . . . 38
4.1.1 Idee und Ziel . . . 38
4.1.2 LabView vs. NXT Byte Codes . . . 39
4.2 Umsetzung . . . 39
4.2.1 Roboterkonstruktion . . . 40
4.2.2 Explorer #1 . . . 41
4.2.3 Explorer #2 . . . 43
4.2.4 Explorer #3 . . . 44
4.3 Resultat . . . 46
4.4 Fazit . . . 47
4.4.1 Erkentnisse . . . 47
4.4.2 Ausblick . . . 47

5 Schlussfolgerung **49**

A Aufgabenstellung **53**
A.1 Einführung . . . 53
A.2 Aufgabenstellung . . . 54
A.3 Teilaufgaben . . . 54
A.4 Organisatorisches . . . 55

B Tabellen mit den Messresultaten **56**
B.1 Lichtsensor . . . 56
B.2 Ultraschallsensor . . . 58
B.3 Motoren . . . 62

C NBC Sourcecode **63**

Abbildungsverzeichnis

2.1 RCX: Erster programmierbarer LEGO Baustein 9
2.2 Funktionsweise des Tastsensors 11
2.3 Funktionsweise des Lichtsensors 11
2.4 NXT-Baustein mit angeschlossenen Sensoren und Motoren . . 12
2.5 Verschiedene Robotermodelle 12

3.1 Schema eines Ausgangsports 15
3.2 Schema eines Eingangsports 16
3.3 Verhältnis von dB- und NXT-Werten 17
3.4 Helligkeitstest im Reflected Light-Modus 22
3.5 Helligkeitstest im Ambient Light-Modus (weisse LED) 22
3.6 Helligkeitstest im Ambient Light-Modus (Halogenlampe) . . . 23
3.7 Versuchsanordung Ultraschallsensor – 1.Versuch 24
3.8 Nullpunkt des Ultraschallsensors 25
3.9 US: Abweichungen gegenüber den exakten Werten 25
3.10 Versuchsanordnung Ultraschallsensor – 3. Versuch 27
3.11 Sichtfeld des Ultraschallsensors in der xy-Ebene 28
3.12 Dynamischer Ultraschalltest mit Power 40 29
3.13 Ausschnitt aus dem dynamischen Ultraschalltest 30
3.14 Dynamischer Ultraschalltest mit Power 60 30
3.15 Umdrehungsgeschwindigkeit der Motoren im Vergleich 32
3.16 Bild des T-Steckers zum Anschluss an ein Oszilloskop 33
3.17 LMS2006-Protokoll 37

4.1 Testumgebung des Explorers 40
4.2 Bild des Explorers 40
4.3 Bild des Sensorturms 42
4.4 Funktionsflussdiagramm von Explorer #1 43
4.5 Funktionsflussdiagramm von Explorer #2 44
4.6 Abhängigkeit der Prozeduren 45
4.7 Auswertung der Raumerkennung des Explorers #3 46
4.8 Raumerkennung des Explorers #3 mit Problemen 47

Tabellenverzeichnis

3.1 Analoge Sensorsignale 34
3.2 Digitale Sensorsignale 35
3.3 Motorsignal zur Regulierung der Geschwindigkeit 36

4.1 NXT Byte Codes vs. LabView 39

Kapitel 1

Vorwort

1.1 Einführung

Das Institut für Technische Informatik und Kommunikationsnetze (TIK) der ETH Zürich bietet Studenten schon seit mehreren Jahren im Rahmen des Blocks PPS (Projekte, Praktika und Seminare) im Elektrotechnikgrundstudium LEGO Mindstorms-Projekte an. Ab Herbst 2006 wird die nächste Generation, das LEGO-Produkt Mindstorms NXT, auf dem Markt erhältlich sein. LEGO stellte 100 Mindstorms NXT-Entwicklerkits (MINDSTORMS Developer Program) als Vorabversion zur Verfügung, für die sich jeder bewerben konnte. Das TIK Institut bekam schliesslich die Gelegenheit, eines dieser Mindstorms NXT Entwicklerkits zu erwerben. Dies führte zur Vergabe der Mindstorms NXT Semesterarbeit, bei der das neue Produkt erforscht und getestet, sowie ein eigenes Projekt erarbeitet werden soll. Erkenntnisse aus dieser Semesterarbeit sollen als Grundlage für weitere PPS-Projekte dienen. (Aufgabenstellung: siehe Appendix A)

1.2 Persönliche Motivation

Von den zahlreichen Angeboten an Semesterarbeiten des TIK-Institutes erweckte vor allem ein Projekt unsere Aufmerksamkeit: Mindstorms Next Generation (NXT). Diese Semesterarbeit versprach einen äusserst vielseitigen Arbeitsbereich, der von Testen der verschiedenen Sensoren, Motoren und Schnittstellen (vor allem Bluetooth), sowie Konstruktion und Programmierung eines eigenen Roboters, über die Umsetzung eines Projekts bis zum Verfassen eines Berichts über die Erkenntnisse und Ergebnisse reicht.
Unsere Hauptziele sind das Kennenlernen der verschiedenen Sensortypen, der Motoren sowie der Bluetooth-Kommunikation, deren Grenzen zu erforschen und aus diesen Ergebnissen ein Projekt mit einem selbst konstruierten Roboter zu definieren und zu realisieren. Ebenso ist ein Ziel herauszufinden, wie ein solcher Roboter programmiert wird und welche Möglichkeiten und

Schranken sich uns dabei aufzeigen. Zusätzlich hoffen wir neue Erfahrungen beim Verfassen eines Berichtes und in der Zusammenarbeit im Team zu erlangen.

1.3 Struktur

Das zweite Kapitel vermittelt einen Einblick in das LEGO-Produkt Mindstorms sowie in dessen historischen Hintergrund und die Weiterentwicklung der RCX- zur neuen NXT-Generation. Detailliertere Information zum Produktinhalt und zu den selbst durchgeführten Versuchen mit dem NXT-Kit befinden sich im dritten Kapitel. In Kapitel vier wird das von uns durchgeführte Projekt beschrieben. Anschliessend wird in Kapitel fünf eine Schlussforlgerung der Arbeit gegeben. Im Anhang befindet sich die Aufgabenstellung unserer Semesterarbeit, die Tabellen mit den Messresultaten aus den Sensor- und Motorenversuche und den für das Projekt geschriebenen NBC-Sourcecode.

Kapitel 2

LEGO Mindstorms (Grundlagen)

In diesem Kapitel werden die Hintergründe zu LEGO Mindstorms sowie der technische Weiterentwicklung von der RCX- zur NXT-Generation geschildert. Alle Bestandteile (Hardware wie auch Software) des NXT Entwicklerkits werden aufgelistet und ihre Funktionsweisen werden kurz beschrieben. Weiterführende Angaben zu den Elementen des Entwicklerkits befinden sich in Kapitel 3.2 – Technische Spezifikationen.

2.1 Historischer Hintergrund

Der erste Versuch, LEGO mit Computertechnologie zu kombinieren, resultierte 1986 in der LEGO Technic Computer Control-Serie, die in Zusammenarbeit mit dem Massachusetts Institute of Technology (MIT) entwickelt wurde. Allerdings konnte sich dieses Produkt auf dem Markt nicht durchsetzen.
LEGO gelang erst 1998, nach erneuter Zusammenarbeit mit dem MIT, die erfolgreiche Markteinführung von LEGO Mindstorms[a] in den USA. Seit Ende 1999 wird LEGO Mindstorms auch in Europa angeboten. Mit diesem Produkt kann ein Roboter gebaut und mittels Computer programmiert werden. Das Herzstück dieser Roboter ist das so genannte RCX (siehe Abbildung 2.1), ein batteriebetriebener „programmierbarer LEGO-Baustein“, der zur Programmierung über eine Infrarotschnittstelle an einen PC angeschlossen werden kann. Er verfügt über ein LCD und Anschlüsse für verschiedene Sensoren (Temperatur-, Licht-, Rotations- und Berührungssensoren) und 3 Elektromotoren. Von der original LEGO-Block-Programmiersprache bis Java existieren diverse Programmiersprachen für den RCX.
Im Herbst 2006 bringt LEGO die neue Serie Mindstorms NXT (Next Gene-

[a] Die erste Version wurde 'Robotics Invention System 1.0' genannt

Abbildung 2.1: Erster programmierbarer LEGO Baustein (RCX) mit angeschlossenen Sensoren und Motoren

ration) auf den Markt, die eine Weiterentwicklung von Mindstorms darstellt. Dabei werden verbesserte Sensoren präsentiert und es wird weitgehend auf die Noppenstruktur verzichtet. Im Vordergrund steht nun das stabilere und leichtere LEGO TECHNIC Konstruktionssystem.

2.2 NXT Generation

2.2.1 Vergleich zu RCX

Bei der neuen Mindstorms NXT Serie wurde der Tastsensor sowie der Lichtsensor verbessert. Neu entwickelt wurde der Ultraschallsensor, der Tonsensor und der NXT-Baustein, der das neue Herzstück des Roboters darstellt. Dieser NXT-Baustein kann nun erstmals auch von einem Apple-Computer aus programmiert werden. Die vielseitige Software baut auf dem LabView-Programm von National Instruments auf. Dank der neuen Bluetooth-Schnittstelle können Programme kabellos auf den NXT-Baustein übertragen werden und der Roboter kann durch Fernbedienen gesteuert werden (z.B. von einem Handy oder einem PDA aus). Ausserdem ist auch eine Verbindung via USB-Kabel möglich. Die 3 Servomotoren besitzen neu eingebaute Rotationssensoren zur präzisen Geschwindigkeitskontrolle. Für grössere Stabilität des Roboters sorgen 519 LEGO-TECHNIC Elemente, welche wie oben erwähnt grösstenteils keine Noppenstruktur besitzen.

2.2.2 Produktinhalt – Angaben des Herstellers

Die Angaben in diesem Kapitel stammen von LEGO und werden in den Kapiteln 3.3-3.6 verifiziert.

Der NXT-Baustein besitzt:

- Drei Output Ports zum Anschliessen der Motoren - Port A, B und C
- Vier Input Ports zum Anschliessen der Sensoren - Port 1, 2, 3 und 4 (Port 4 beinhaltet einen IEC 61158 Type 4/EN 50 170 kompatiblen Erweiterungsport für zukünftigen Gebrauch)
- USB 2.0 Schnittstelle (12 Mbit/s) für die Verbindung zum Computer (PC oder Mac)
- Bluetooth (Class II V2.0, BlueCore 4 Chip, 460.8 Kbit/s) Schnittstelle für die drahtlose Verbindung zum Computer sowie zur Fernsteuerung. Unterstützt das Serial Port Profile (SPP), besitzt 1 MB externen FLASH Speicher für den Bluetooth stack
- Lautsprecher für das Abspielen von Soundeffekten: 8 kHz Soundqualität, Soundkanal mit 8 Bit Auflösung und 2-16 kHz Sample-Rate
- Buttons:
  - Orange: On/Enter/Run
  - Hellgrau: Move left and right in the NXT menu
  - Dunkelgrau: Clear/Go back
- Bildschirm: 100 x 64 Pixel LCD Display
- Prozessoren:
  - CPU: 32-bit ARM7 Mikrocontroller – AT91SAM7S256 (256 kBytes FLASH, 64 kBytes RAM, 48MHz - 0.9 MIPS/MHz)
  - Co-Prozessor: 8-bit AVR Mikrocontroller – ATMEGA48 (4 kBytes FLASH, 512 Byte RAM, 4MHz - 1 MIPS/MHz)
- 6 AA Batterien zur Stromversorgung

Kabel: Sieben Stück 6-adriger Kabel unterschiedlicher Länge

Sensoren:

Der Tastsensor gibt dem Roboter einen Tastsinn, er detektiert 1, wenn er betätigt wird und 0, wenn er wieder freigegeben ist. Die Funktionsweise des Tastsensors ist in Abbildung 2.2 schematisch dargestellt.

Der Tonsensor kann sowohl unangepasste Dezibelwerte [dB] wie auch angepasste Dezibelwerte [dBA] ermitteln (Siehe Kapitel 3.2). Die Einheit Dezibel ist ein Mass für den Schalldruckpegel, wobei Werte zwischen 55 dB und 90 dB gemessen werden können. Diese Messwerte werden durch den NXT in Prozenten ausgedrückt [%]. Je niedriger der Prozentwert, desto leiser ist die Tonquelle.

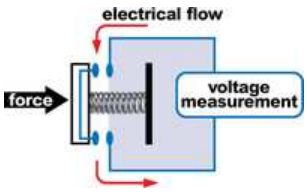


Abbildung 2.2: Funktionsweise des Tastsensors

Der Lichtsensor ermöglicht dem Roboter, zwischen Hell und Dunkel zu unterscheiden. Er kann die Lichtintensität in einem Raum (Helligkeit) oder einer farbigen Oberflächen messen. Einige Farben und deren Helligkeitswert werden in Abbildung 2.3 gezeigt.

Abbildung 2.3: Verschiedene Farben und deren Helligkeitswert aus der Sicht des Lichtsensors

Der Ultraschallsensor ermöglicht dem Roboter Abstände zu Objekten zu berechnen, indem er die Zeit misst, die benötigt wird, bis eine von ihm ausgesendete Schallwelle an ein Objekt stösst und deren Echo wieder empfangen wird. Er misst Abstände in der Einheit Zentimeter oder Zoll und ist in der Lage Distanzen bis zu 255 cm mit einer Präzision von $\pm$ 3 cm zu messen.

Motoren:

Die drei Motoren geben dem Roboter die Fähigkeit sich zu bewegen. Jeder Motor hat einen eingebauten Rotationssensor, welcher die Ansteuerungspräzision erhöht. Der Rotationssensor misst Umdrehungen in Winkelgrad (Genauigkeit von einem Grad). Dank diesem Sensor kann durch verändern der Powereinstellung jeder Motor mit unterschiedlicher Geschwindigkeit betrieben werden. Wenn man den Bewegungsblock in der LEGO Mindstorms NXT Software benützt, um mehrere Motoren gleichzeitig anzusteuern, werden die Motoren automatisch synchronisiert, damit sich der Roboter gradlinig bewegt.

Der „LEGO Mindstorms NXT“-Baukasten bietet zahlreiche Möglichkeiten, einen auf die individuelle Vorstellungen und Wünsche angepassten Roboter zu konstruieren (Siehe Abb. 2.5).

Abbildung 2.4: NXT-Baustein mit angeschlossenen Sensoren und Motoren

Abbildung 2.5: Verschiedene Robotermodelle (von links nach rechts): RoboArm T-56, TriBot, Humanoid Alpha Rex, Spike

2.2.3 LEGO Software – LabView

Die „LEGO Mindstorms NXT“-Software ermöglicht, den selbst konstruierten NXT-Roboter zu programmieren und die Programme über die USB- oder Bluetooth-Schnittstelle auf den NXT-Baustein zu übertragen. Diese Mac und PC kompatible „drag and drop“-Software, die auf dem LabView Programm aufbaut, besitzt zum jetzigen Zeitpunkt vier Roboterkonstruktions- sowie dazugehörige Programmieranleitungen. Das Software User Interface stellt ein Arbeitsfenster (work area) bereit, in dem einzelne Blöcke (z.B. Move, Display, Sound, Record/Play, Wait, Loop, Switch) aus der Programmierpalette grafisch einfach verknüpft werden können.

2.2.4 NXT Byte Codes

Next Byte Codes (NBC) ist eine Alternative zur LEGO-Software LabView. Bei NBC handelt es sich um eine Programmiersprache, die auf einer Assemblersyntax basiert und zur Programmierung des „Mindstorms NXT“-Bausteins verwendet werden kann. Diese Byte Codes werden von der NXT-Firmware interpretiert. In NBC können Structs, Arrays und Skalare definiert

werden. Die gegebenen Datentypen sind: Int, Word, Dword (jeweils Signed oder Unsigned). Des weiteren ermöglicht NBC den Aufruf von Systemcalls, die im wesentlichen dem Zugriff auf die Hardware und die angeschlossenen Sensoren/Motoren dienen.
NBC ist eine frei erhältliche Software, die unter der allgemeinen Lizenz von Mozilla (MPL) freigegeben ist. NBC wurde von John Hansen [7] entwickelt und ist für jedermann zugänglich.

Kapitel 3

Versuche mit dem NXT-Kit

Dieses Kapitel enthält eine Beschreibung der Roboter TriBot und Alpha Rex, sowie detaillierte Angaben zu den Sensoren, Motoren und den Eingangs- und Ausgangsports des Bausteins. Nach diesen technischen Spezifikationen werden die selbst durchgeführten statischen und dynamischen Sensortests sowie Motorentests mit den dazugehörigen Auswertungen und Schlussfolgerungen dokumentiert. Anschliessend werden Oszilloskop-Diagramme der Sensorsignale, die zwischen Sensoren und Baustein verschickt werden, ausgewertet. Am Ende des Kapitels wird über die Kommunikationsmöglichkeiten mit dem Baustein berichtet.

3.1 Konstruktionen

Mit Hilfe der Roboterkonstruktionsanleitungen, die in der LEGO Software integriert ist, kann Schritt für Schritt einer von vier Roboter-Modellen (TriBot, RoboArm T-56, Spike, Humanoid Alpha Rex) gebaut werden. Bei jedem dieser Roboter ist eine Konstruktions-Schwierigkeitsstufe in Anzahl Sternen angegeben. Drei Sterne (z.B. TriBot) kennzeichnen einen eher einfach konstruierbaren Roboter, fünf Sterne (z.B. Humanoid Alpha Rex) hingegen eine etwas komplexere Konstruktionsweise. LEGO empfiehlt aus diesem Grunde als erstes den TriBot zu konstruieren und zu erforschen. Bei beinahe allen Robotern werden sämtliche Motoren und Sensoren verwendet.

3.1.1 TriBot

Das Zusammensetzen des TriBots (Siehe Kap. 2.2.2 – Abb. 2.5) dauert etwa zwei Stunden. Dieser Roboter fährt auf zwei Reifen und auf einem kleineren Plastikrad. Er besitzt zwei Greifarme mit denen er Gegenstände (vorzugsweise kleine, leichte Bälle) aufheben und transportieren kann. Zwei Motoren benötigt der TriBot für die Fortbewegung, den Dritten für die Steuerung der Greifarme.

3.1.2 Humanoid Alpha Rex

Um den Humanoiden Alpha Rex (Siehe Kap. 2.2.2 – Abb. 2.5) zu konstruieren benötigt man etwa vier Stunden. Wie ein Mensch läuft er auf zwei Beinen, die von zwei Motoren angetrieben werden. Um sich fortzubewegen verlagert er jeweils seinen Schwerpunkt auf eine Seite und schiebt dann das Bein auf der anderen Seite nach vorne. Mit dem dritten Motor bewegt er gleichzeitig seinen Kopf und seine Arme, an denen jeweils Sensoren befestigt sind.

3.2 Technische Spezifikationen

Sämtliche Angaben in diesem Kapitel basieren auf Dokumentationen von LEGO [2].

Ausgangsport: Um von den Motoren auch Informationen zurück an den NXT-Baustein senden zu können, ohne einen weiteren Eingangsport zu benötigen, wurde ein sechsadriges Kabel zur digitalen und analogen Datenübertragung entwickelt. Die Abbildung 3.1 zeigt schematisch die Pinbelegung eines dieser Ausgangsports.

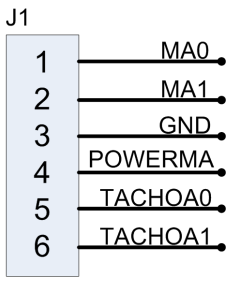


Abbildung 3.1: Schema eines Ausgangsports im NXT-Baustein

Der NXT-Baustein verändert durch Pulsweitenmodulation (PWM – siehe auch Kap. 3.5) die MA0- und MA1-Ausgangssignale, welche die Umdrehungsgeschwindigkeit des Motors regulieren. Jeder Motor kann über diese Leitungen einen maximalen Strom von 700 mA beziehen.
GND ist die Erdung des Ausgangsports.
Die Versorgungsspannung POWERMA beträgt 4,3 Volt und wird im Falle des Motors vom Rotationssensor benötigt. Der POWERMA Pin teilt sich mit den übrigen Motoren und Sensoren die auf allen Ports über den Pin 4 total verfügbaren 270 mA, was bedeutet, dass jeder Port durschnittlich mit 38 mA gespiesen wird.
Die TACHOA0 und TACHOA1 Signale werden für die Auswertung der

Anzahl Rotationsimpulse und der Rotationsrichtung benutzt. Ein Schmitt-Trigger ist zwischen dem Ausgangsport und dem ARM7 Prozessor angebracht.

Eingangsport: Damit sowohl analoge als auch digitale Sensoren genutzt werden können, wird auch bei den Eingangsports ein sechsadriges Kabel verwendet. Bei Port 4 sind die digitalen Pins (DIGIAI0 und DIGIIAI1) an einen RS485 Controller angeschlossen, der die Highspeed-Kommunikation steuert. Die Abbildung 3.2 zeigt schematisch die Pinbelegung eines Eingangsports.

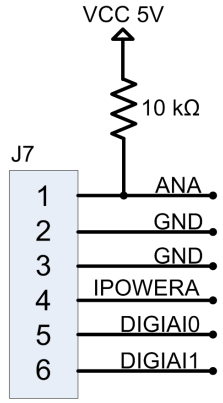


Abbildung 3.2: Schema eines Eingangsports im NXT-Baustein

Der analoge Eingang ANA ist an den AVR Prozessor angeschlossen, der über einen A/D-Konverter verfügt.
GND ist die Erdung des Eingangsports.
Die IPOWERA-Leitung hat die gleiche Funktionalität wie die POWERMA-Leitung des Ausgangsports.
Die DIGIAI0- und DIGIAI1-Leitungen dienen der digitalen Kommunikation, welche das I^2C-Protokoll [8] benutzt (9600 bit/s). Der NXT-Baustein kann bezüglich der I^2C-Kommunikation nur als Master agieren, somit sind alle Sensoren stets als Slave angeschlossen.

Der Tastsensor wird analog betrieben. Dieser passive Sensor hat eine sehr geringe Stromaufnahme.

Strom (Taste ungedrückt): 0 mA
Strom (Taste gedrückt): 2.2 mA

Pinbelegung des Tastsensors:

- 1 - Trigger

- 2 - Unbenutzt
- 3 - Ground
- 4 - Unbenutzt
- 5 - Unbenutzt
- 6 - Unbenutzt

Der analoge Tonsensor misst Schalldruck entweder in dB oder in dBA. Das Ändern des Operationsmodus zwischen dem Messen von dB zu dBA resultiert in einer Zeitverzögerung von ungefähr 300 ms. Der Modus wird durch das Spannungsniveau an Pin 5 festgelegt (high: dB-Modus, low: dBA-Modus). Es können Werte im Bereich von 55 dB bis 90 dB gemessen werden. Die dB-Skala ist logarithmisch aufgebaut, d.h. eine Verdoppelung der Amplitude führt zu einer Erhöhung um 3 dB. Das menschliche Ohr empfindet eine Erhöhung um 6 dB als Verdoppelung der Lautstärke, wenn sich der Schalldruck vervierfacht hat. Der Zusatz A gibt an, dass die unterschiedlichen Tonfrequenzen ähnlich dem menschlichen Hörempfinden unterschiedlich bewertet werden, d.h. mittlere Frequenzen (Peak bei 2 kHz) werden stärker berücksichtigt.
Der Tonsensor kann Töne aus allen möglichen Richtungen messen. Er besteht aus folgenden hintereinander geschalteten Blöcken: einem eingebauten Mikrophon, einem Gleichrichter und einem Mittelwertbilder. Folglich ist es nicht die elektrische Darstellung des Tones selbst, die zum NXT-Baustein gesendet wird, sondern ein DC Niveau, das mit dem Schalldruck oder dem Volumen schwankt. Im Gegensatz zum Signal auf Pin 1 wird das Signal auf Pin 6 direkt nach dem Mikrophonblock an den NXT-Baustein weiter geleitet.

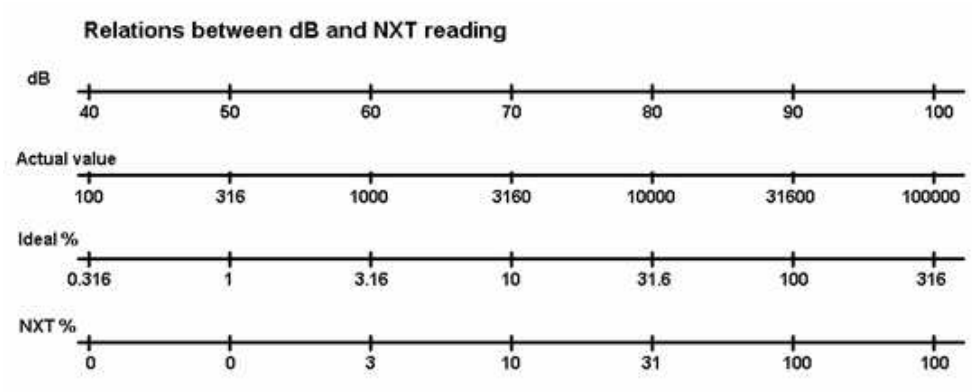


Abbildung 3.3: Verhältnis von dB- und NXT-Werten

Stromverbrauch: 2 mA
Frequenzbereich: 20 Hz bis 18 kHz

Signal-noise ratio: 40 dB

Pinbelegung des Tonsensors:

- 1 - Analoges Tonsignal
- 2 - Ground
- 3 - Ground
- 4 - Versorgungsspannung von 4.3 Volt
- 5 - Moduswahl dB/dBA
- 6 - Direkter Ausgang

Der Lichtsensor ist ebenfalls ein analoger Sensor, dessen Messwerte mit dem integrierten AVR-Mikrocontroller ausgewertet und in ein digitales Signal umgewandelt werden, welches dieser alle 3 ms an den ARM-Prozessor weiter leitet.
Wenn sich der Lichtsensor im Reflected Light Modus befindet, wird eine integrierte LED aktiviert, welche das Messen von Helligkeitswerten auch bei schwachem Umgebungslicht möglich macht. Das Umschalten vom Reflected Light-Modus in den Ambient Light-Modus nimmt 12 ms in Anspruch. Erst nach Ablauf dieser Zeit wird wieder ein gültiger Wert angezeigt.
Die Empfindlichkeit des Sensors ist bei frontalem Lichteinfall und bei Licht mit verhältnismässig langer Wellenlänge (rotes und infrarotes Licht) am grössten. Je grösser der Winkel des Lichteinfalls, desto schlechter wird das Licht vom Sensor detektiert. Wie schon erwähnt, kann dieser Sensor in zwei unterschiedlichen Modi betrieben werden, mit oder ohne eingebauter Lichtquelle. Die Stromaufnahme hängt dementsprechend vom gewählten Modus ab, der mit Pin 5 (high: Reflected Light-Modus, low: Ambient Light-Modus) bestimmt wird.

Stromverbrauch im Ambient Light-Modus: 2.5 mA
Stromverbrauch im Reflected Light-Modus: 15 mA

Pinbelegung des Lichtsensors:

- 1 - Analoges Lichtsignal
- 2 - Ground
- 3 - Ground
- 4 - Versorgungsspannung von 4.3 Volt
- 5 - Lichtquelle Ein/Aus
- 6 - Unbenutzt

Der Ultraschallsensor ist der einzige digitale Sensor, er besitzt einen integrierten Mikrocontroller, der alle Ultraschallmesswerte zum NXT-Baustein per I^2C-Kommunikation[a] sendet. Dieser Sensor agiert

[a] I^2C ist ein serielles Protokoll, das mit nur zwei Signalleitungen betrieben wird

als Slave, d.h. die gesamte Kommunikation zwischen dem Sensor und dem NXT-Baustein wird durch den Letztgenannten gesteuert.
Der Ultraschallsensor eruiert den Abstand zu Gegenständen, indem er 12 Signalstösse bei 40 kHz aussendet und dann die Zeit misst, bis er eine Reflektion des ausgesandten Signals empfängt. Die Zeit zwischen der Emission und dem Empfangen des reflektierten Stosses ist proportional zum Abstand zwischen Objekt und Sensor. Der messbare Distanzbereich liegt zwischen 0 cm und 255 cm und kann daher durch ein einziges Byte kodiert werden.
Der Ultraschallsender benötigt die von Pin 1 gelieferten 9 Volt, daher nimmt die Messgenauigkeit mit abnehmender Batteriespannung ab.

Stromverbrauch (messend): 4.0 mA
Stromverbrauch (leerlauf): 3.5 mA

Pinbelegung des Ultraschallsensors:

- 1 - Versorgungsspannung von 9 Volt
- 2 - Ground
- 3 - Ground
- 4 - Versorgungsspannung von 4.3 Volt
- 5 - I^2C-Kommunikation SCL (Serial Clock)
- 6 - I^2C-Kommunikation SDA (Serial Data)

In jedem Motor ist ein Rotationssensor eingebaut, welcher ein Signal an den NXT-Baustein sendet, das 180 Impulse bei einer vollen Umdrehung generiert. Das Rotationssignal beinhaltet zwei Werte: Einerseits die Drehrichtung des Motors und andererseits die Anzahl der Drehimpulse. Sowohl bei der steigenden wie auch bei der fallenden Flanke misst der NXT-Baustein den Drehimpuls und erhält dadurch 360 Zählimpulse pro Umdrehung.
Der Motor kann entweder in den Blockier- oder in den Leerlaufmodus geschaltet werden. Im Blockiermodus bezieht der Motor stetig Strom und kann somit trotz äusserer Krafteinwirkung in einer bestimmten Positionslage verharren.

Drehkraft bei maximaler Leistung: 56 Nmm
Drehkraft im Blockiermodus: 400 Nmm

Stromverbrauch (während der Rotation): 100 mA
Stromverbrauch (blockiert): 900 mA (Spitzenstrom)

3.3 Statische Sensorenversuche

LEGO beschreibt jeden Sensor und seine Fähigkeiten in einem Textdokument [2]. Um diese Angaben zu verifizieren und um weitere Eigenheiten der Sensoren zu ergründen, führen wir eigene Sensorversuche durch. Ziel dieser Tests ist die Gewinnung von essentiellen Informationen, welche später als Grundlage für die Definition und Realisation unseres Projektes dienen.

3.3.1 Tastsensor

Fragestellung:

- Wie gross ist die minimal aufzuwendende Kraft, damit der Sensor den Wert 1 („gedrückt“) übermittelt?

Versuchsanordnung:
Der Drucksensor wird vertikal befestigt.

Messverfahren:
Schrittweise werden kleine Gewichte auf den Sensor gelegt. Sobald der Sensor den Wert 1 für „gedrückt“ zurückgibt, werden die Gewichte mit Hilfe einer Waage gemessen.

Auswertung:
Bei einem Gewicht von 34 Gramm ist der Sensor gedrückt. Dies entspricht einer Kraft von 0.34 Newton.

3.3.2 Tonsensor

Mit dem Tonsensor wurde kein Versuch durchgefürht, da er nur Schalldruck messen kann und alle massgebenden Informationen bereits vom Hersteller geliefert werden (Siehe Kap. 3.2).

3.3.3 Lichtsensor

Fragestellung:

- Welcher Modus (Ambient Light oder Reflected Light) liefert bessere Resultate?
- Welchen Einfluss hat die Umgebungshelligkeit?
- Wie gut sind verschiedenfarbene Körper unterscheidbar?
- Welche Distanz zum Körper ist optimal?

Versuchsanordnung:
Der Lichtsensor wird in einer bestimmten Entfernung zur Unterlage (zwi-

schen 1 cm und 4 cm) montiert. Abwechslungsweise werden verschiedenfarbene (rote, grüne, blaue, gelbe, weisse und schwarze) Papier- und Plastikstreifen darunter positioniert. Der Versuch findet in einem verdunkelten Raum statt. Vor dem Messen der einzelnen Farbwerte werden Referenzwerte bezüglich der Unterlage (Spiegel oder schwarzer Plastik) bestimmt. Diese Prozedur wird in den verschiedenen Betriebsmodi (Reflected- und Ambient Light-Modus) durchgeführt.

Messverfahren:
Die Messresultate werden der Anzeige des NXT-Bausteins entnommen und tabellarisch erfasst. Alle Resultate sind Relativwerte und werden in Helligkeitsprozenten angegeben (100 totale Reflektion wie zum Beispiel bei einem Spiegel, 0 keine Reflektion). Da der Lichtsensor nur eine Graustufenerkennung besitzt, werden verschiedene Farbflächen (rot, grün, blau, gelb, schwarz und weiss) untersucht. Der Abstand zur Unterlage beträgt für die verschiedenen Testreihen jeweils 1 cm, 2 cm, 3 cm und 4 cm.

Auswertung:
Der Sensor benützt eine Photodiode um die Intensität des einfallenden Lichts zu messen. Im Reflected Light-Modus wird im Gegensatz zum Ambient Light-Modus der Untergrund zusätzlich mit Hilfe einer roten LED beleuchtet, die unmittelbar neben der Photodiode montiert ist. Da die LED Licht im roten Bereich des Lichtspektrums aussendet, werden vor allem Farben mit einem hohen Rotanteil stark reflektiert. Die minimale Distanz von 0 cm zum Körper ist nur im Reflected Light-Modus möglich, da im Ambient Light-Modus kein Licht mehr auf die Photodiode trifft.

Da weder bei den verwendeten Materialien (Plastik und Papier) noch bei unterschiedlicher Unterlage (Spiegel oder schwarzer Plastik) grosse Unterschiede bei den einzelnen Farbwerten erkennbar sind, werden an dieser Stelle nur Versuche mit unterschiedlich farbenen Papierstreifen auf schwarzem Plastik gezeigt. Die Messresultate aus dem Versuch im Reflected Light-Modus sind in Abbildung 3.4 in einem Diagramm dargestellt. In den Abbildungen 3.5 und 3.6 sind die Diagramme mit den Messwerten im Ambient Light-Modus jedoch mit unterschiedlichen Lichtquellen abgebildet.

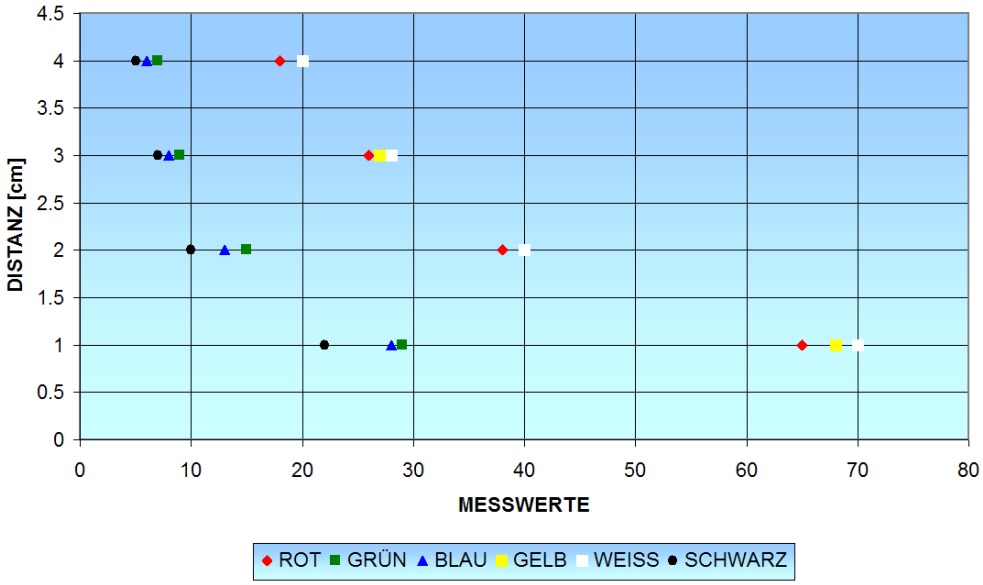


Abbildung 3.4: Helligkeitstest im Reflected Light-Modus

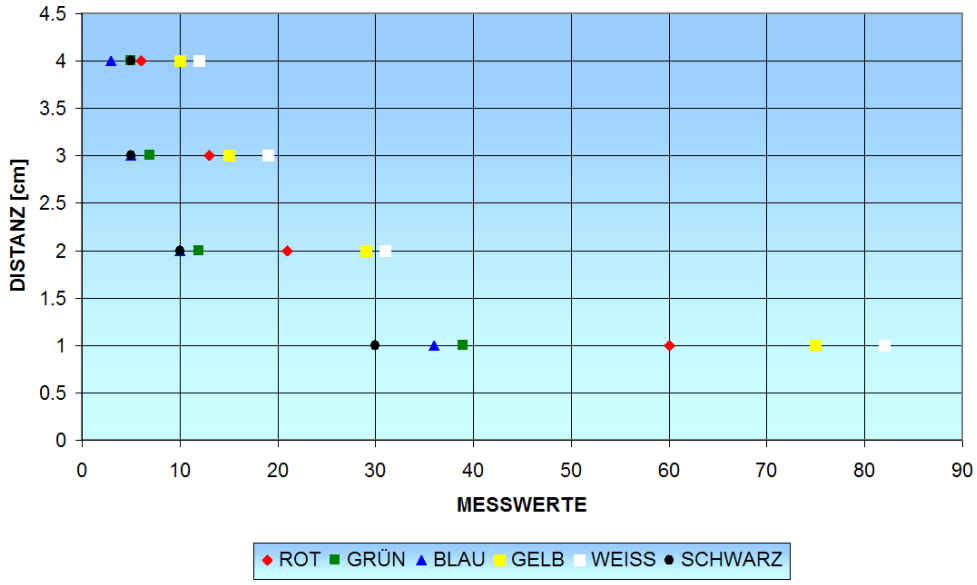


Abbildung 3.5: Helligkeitstest im Ambient Light-Modus - Beleuchtet durch eine weisse LED

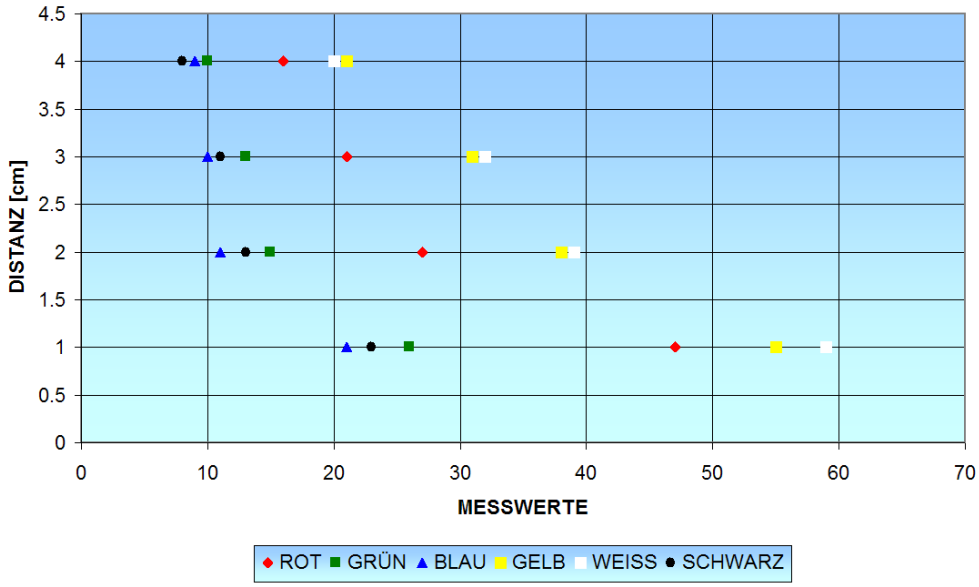


Abbildung 3.6: Helligkeitstest im Ambient Light-Modus - Beleuchtet durch eine Halogenlampe

Schlussfolgerung:
Im Ambient Light Modus werden die Farben besser unterschieden als im Reflected Light Modus. Je näher sich der Sensor bei einem Objekt befindet, desto besser können die Farben voneinander unterschieden werden. Einer Farbe einen konkreten Wertebereich zuzuordnen ist jedoch nicht möglich, da die Messwerte zu stark von der Beleuchtung und der Distanz zum Sensor abhängen.

3.3.4 Ultraschallsensor

1. Versuch
Um einen ersten Eindruck des Ultraschallsensors zu bekommen wird ein verhältnismässig einfacher Test durchgeführt, bei dem nur geringe Distanzen untersucht werden.

Fragestellung:
- Wo befindet sich der Distanz-Nullpunkt des Sensors?
- Wie werden die Messwerte gerundet?
- Wie gross ist die kleinste, noch messbare Distanz zwischen dem Ultraschallsensor und einem Quader?

Versuchsanordnung:
Auf einer Arbeitsfläche wird der Ultraschallsensor horizontal positioniert.

Die Höhe zwischen der Arbeitsfläche und dem Mittelpunkt des Sensors beträgt 4 cm. Die Entfernung zwischen dem Sensor und einem quaderförmigen Körper (14.5 cm x 9.5 cm x 6 cm, Material: Karton) wird von 0 cm bis 14 cm variiert, indem der Körper auf einer Zentimeter-Skala zum Sensor hin verschoben wird.

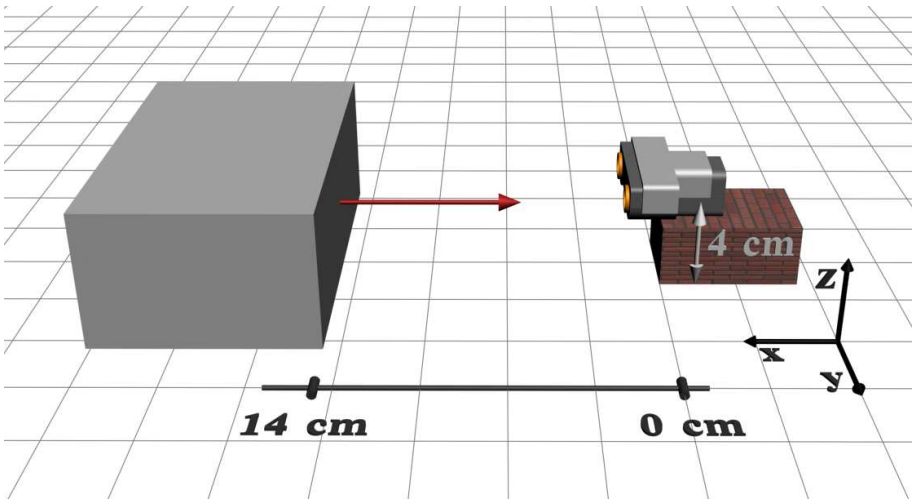


Abbildung 3.7: Versuchsanordung Ultraschallsensor – 1.Versuch

Messverfahren:
Die angezeigten Werte auf dem Bildschirm des NXT-Bausteins werden in einer Tabelle festgehalten. Da auf der Anzeige nur ganzzahlige Werte ausgegeben werden, verschiebt man den Körper jeweils um 1.2 cm in Richtung des Sensors, um das Rundungsverhalten analysieren zu können. Da in diesem Versuch vor allem kurze Distanzen von Interesse sind, finden die Messungen im Bereich von 0 cm bis 14 cm statt. Insgesamt wurde dieser Versuch sieben Mal durchgeführt.

Auswertung:
Die Daten aus den ersten vier Messreihen und die daraus folgende mittlere Abweichung bezüglich des exakten Wertes werden im Diagramm in Abbildung 3.9 dargestellt. Aus den Messwerten folgt, dass sich der relative Nullpunkt des Sensors schätzungsweise im Zentrum der Sensorhülle befindet (Siehe Abb. 3.8). Bei Abständen von weniger als 3 cm bezüglich des Nullpunktes liefert der Sensor keinen sinnvollen Wert mehr (auf der Anzeige wird „??????“ angezeigt). Somit beträgt der kleinste messbare Wert 3 cm. Die Genauigkeit des Sensors im Bereich von 0 cm bis 20 cm ist $\pm$ 3 cm, und entspricht somit dem vom Hersteller publizierten Wert. Die maximal gemessene Abweichung von 2.6 cm wurde bei einer Distanz von 2.5 cm ermittelt. Aus den Messdaten folgt eine Standardabweichung von 0.408 cm.

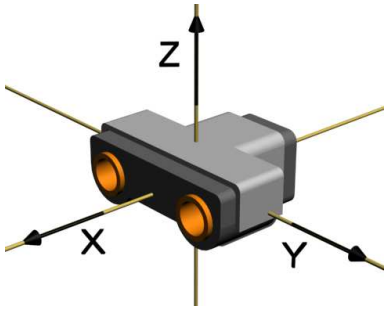


Abbildung 3.8: Nullpunkt des Ultraschallsensors

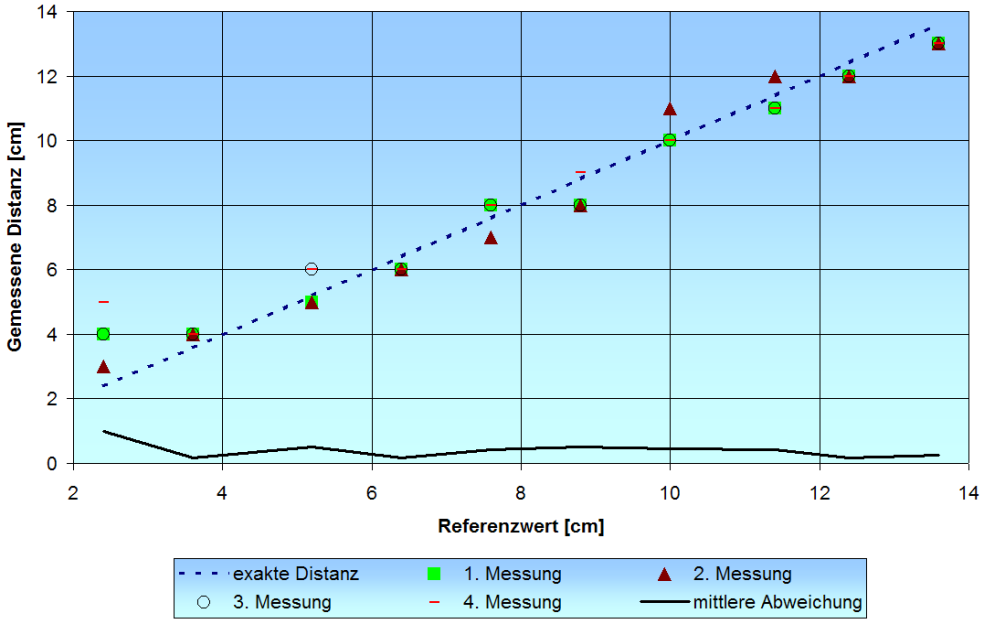


Abbildung 3.9: Abweichungen der Messresultate des Ultraschallsensors gegenüber den exakten Werten

Schlussfolgerung:
Distanzen von weniger als 3 cm sind nicht messbar. Eine Standardabweichung von weniger als 0.5 cm zeugt von einem mathematisch korrekten Rundungsverhalten und einem präzisen Sensor.

2. Versuch

Fragestellung:

- Wie ist die Distanzgenauigkeit des Sensors bezüglich Körper unterschiedlichen Materialien und Formen?

Versuchsanordnung:
Gleiche Versuchsanordnung wie beim 1. Versuch, jedoch ist der Körper mal rechteckig, rund, zylindrisch oder konisch und mal aus Plastik, Metall, Holz oder Stoff. Gemessen werden neu Abstände im Bereich von 5 cm bis 160 cm.

Messverfahren:
Jede Distanz wird bei jedem der verschiedenen Körper mehrmals gemessen. Der gemittelte Wert wird in einer Tabelle, nach Art der Körper gegliedert, eingetragen. Falls der Sensor keine Distanz eruieren kann, wird das Feld leer gelassen. Die Eigenschaften der Körper wie Form, Material, Abmessung und Art des Gegenstandes werden ebenfalls in der Tabelle festgehalten.

Auswertung:
Bei Abständen von weniger als 20 cm haben 95% aller Messresultate eine Genauigkeit von ± 2 cm. Der Bereich zwischen 25 cm und 50 cm scheint ein kritischer Bereich zu sein, bei dem auf der Anzeige des NXT-Bausteins entweder „??????“ oder ein sehr ungenauer Wert ausgegeben wird. Ab 50 cm Entfernung sind gewölbte, eher kleine Objekte für den Sensor nicht mehr sichtbar, jedoch gilt für die übrigen Resultate wieder eine Exaktheit von ± 2 cm.

Schlussfolgerung:
Die genausten Distanzergebnisse erzielt man durch grosse, rechteckige Körper mit glatter und harter Oberflächenstruktur. Gut geeignet sind Metall- oder Plastikoberflächen. Je kleiner der Gegenstand, desto „kurzsichtiger“ ist der Sensor. Der Reflektionswinkel übt grossen Einfluss auf das Messresultat aus, selbst geringe Abweichung des Winkels bezüglich der Einfallsebene der Ultraschallwelle bewirken ein Versagen der Distanzdetektion.

3. Versuch
Fragestellung:

- Bis zu welchem Winkel wird ein Quader (0.5 cm x 9.5 cm x 6 cm) oder ein Ball (Durchmesser: 14 cm) erkannt?
- Bei welcher Ausrichtung des Ultraschallsensors (vertikal oder horizontal) erzielt man die besseren Resultate?

Versuchsanordnung:
Ein Objekt wird in verschiedenen Distanzen zum Ultraschallsensor (zwischen 5 cm und 80 cm) auf horizontalen Strahlen (-30°, -15°, -7.5°, 0°, 7.5°, 15°, 30°) verschoben. Die Testreihe wird zuerst mit horizontal liegendem Sensor und später in vertikaler Lage durchgeführt. In der ersten Testreihe ist das Objekt ein quaderförmiger Körper (14.5 cm x 9.5 cm x 6 cm, Material: Karton), in der Zweiten ein runder Lederfussball (Durchmesser: 14 cm).

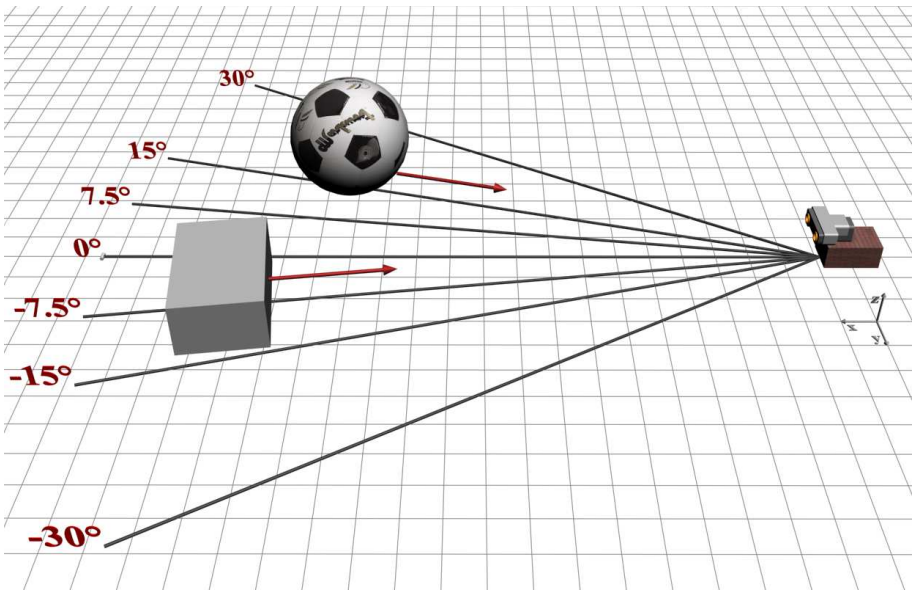


Abbildung 3.10: Versuchsanordnung Ultraschallsensor – 3. Versuch

Messverfahren:
Alle Angaben auf der Anzeige des NXT-Bausteins werden in Tabellen, sortiert nach Winkel, Abstand, Objektart und Lage des Sensors (horizontal oder vertikal), aufgelistet.

Auswertung:
Damit der Sensor den quaderförmigen Körper (14.5 cm x 9.5 cm x 6 cm) noch erkennen kann, darf der Winkel zwischen Mittelpunkt des Körpers und x-Achse des Sensors (siehe Abb. 3.10) nicht mehr als 30° betragen. Zwischen 30° und 15° wird das Objekt im Allgemeinen nur bei geringen Abständen ($\leq$ 20 cm) noch detektiert. Verwendbare Resultate erzielt man im Bereich von 0° bis $\pm$7.5° mit einer Distanz kleiner als 60 cm. Wenn sich der Sensor in horizontaler Lage befindet, detektiert er Körper auf der rechten Seite seines Blickfeldes besser als solche auf der linken Seite (Siehe Abb. 3.11). Dies ist darauf zurückzuführen, dass das rechte „Auge“ des Sensors der Ultraschallsender und das linke „Auge“ der Ultraschallempfänger ist.
Ähnlich ist dieser Zusammenhang auch in vertikaler Lage zu beobachten. Hier sieht der Sensor den Quader der Höhe 9.5 cm bei Winkeln bis zu 30° besser, wenn das Sender-Auge näher am Untergrund ist wie das Empfänger-Auge, da in diesem Fall ein grösserer Teil der ausgesendeten Welle am Untergrund reflektiert wird. In diesen vertikalen Positionen existiert wieder ein kritischer Bereich zwischen 20 cm und 50 cm. Im Gegensatz zur horizontalen Lage des Sensors werden in vertikaler Lage nur Distanzen bis maximal 80 cm angezeigt. Bei der zweiten Testreihe, in der ein Lederball mit einem Durchmesser von 14 cm als Testobjekt fungiert, werden nur noch selten Werte auf

der Anzeige des NXT-Bausteins angezeigt. Die besten Resultate erhält man bei Distanzen kleiner als 20 cm und Winkeln kleiner als 15°.

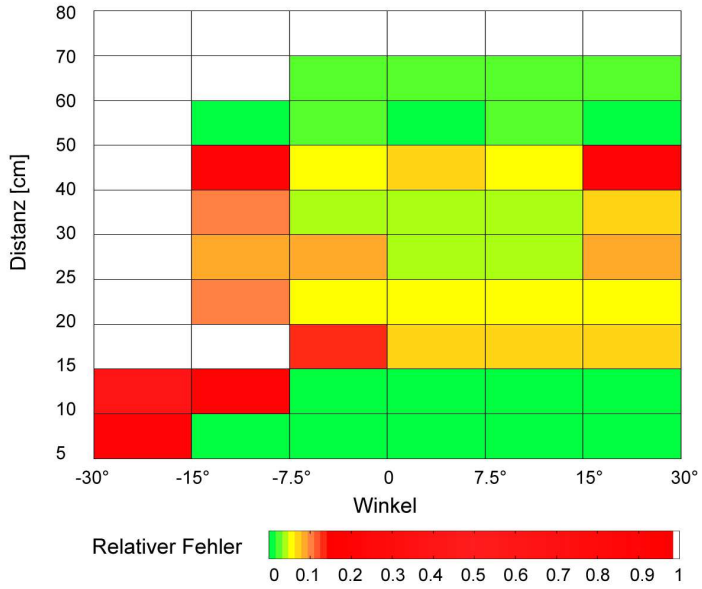


Abbildung 3.11: Sichtfeld des Ultraschallsensors in der xy-Ebene

Schlussfolgerung:
Der Sensor sollte stets horizontal positioniert werden, da im Vergleich zur vertikalen Lage der kritische Bereich kleiner ist und er eine höhere Genauigkeit auch bei grösseren Distanzen aufweist. Winkel von mehr als 15° bezüglich der Einfallsebene der Ultraschallwellen sowie kugelförmige Gegenstände sollten vermieden werden.

3.4 Dynamische Sensorversuche

LEGO selbst hat keine dynamischen Sensortests durchgeführt. Zentral ist jedoch die Frage, welche Messwerte der Ultraschallsensor in Bewegung liefert. Sind im Vergleich zu den statischen Resultaten Unterschiede zu erkennen?

Fragestellung:

- Wie viele kritische Bereiche existieren zwischen 10 cm und 180 cm und wo befinden sich diese?

Versuchsanordnung:
Zuerst wird ein Programm mit Hilfe der LEGO-Software geschrieben, das

den Roboter veranlasst gradlinig 10 ganze Umdrehungen beider Räder vorwärts zu fahren (entspricht einer Distanz von 170 cm) und gleichzeitig die Distanz zu einem Objekt zu messen. Dann wird das Programm auf den NXT-Baustein geladen. Bei diesem Versuch wird der TriBot Roboter verwendet, welcher 180 cm von einer Wand entfernt gestartet wird.

Messverfahren:
Sowohl die Messresultate des Abstandes wie auch die Winkelgrade der Motoren werden vom Programm in einer Datei gespeichert. Der Versuch wird mit 2 verschiedenen Geschwindigkeiten (Power1 = 40, Power2 = 60) jeweils 4 Mal durchgeführt. Alle Daten werden aus der Datei ausgelesen und tabelliert. Dieser Versuch wird in beide Fahrtrichtungen (auf die Wand zu, von der Wand weg) ausgeführt.

Auswertung:
Die Diagramme in Abbildung 3.12 und 3.14 zeigen die Distanzmessungen und deren mittlere Abweichungen. Anhand der Abweichung sind die kritischen Bereiche deutlich sichtbar. Sie befinden sich zwischen 15 cm und 35 cm, zwischen 45 cm und 55 cm, zwischen 80 cm und 105 cm und ab 155 cm bis 180 cm.
Das Intervall von 20 cm bis 50 cm ist der ausgeprägteste der kritischen Bereiche, hier liefert der Sensor immer wieder den falschen Wert 48 cm. Dieser Sachverhalt wird durch das Diagramm in Abbildung 3.13 veranschaulicht.

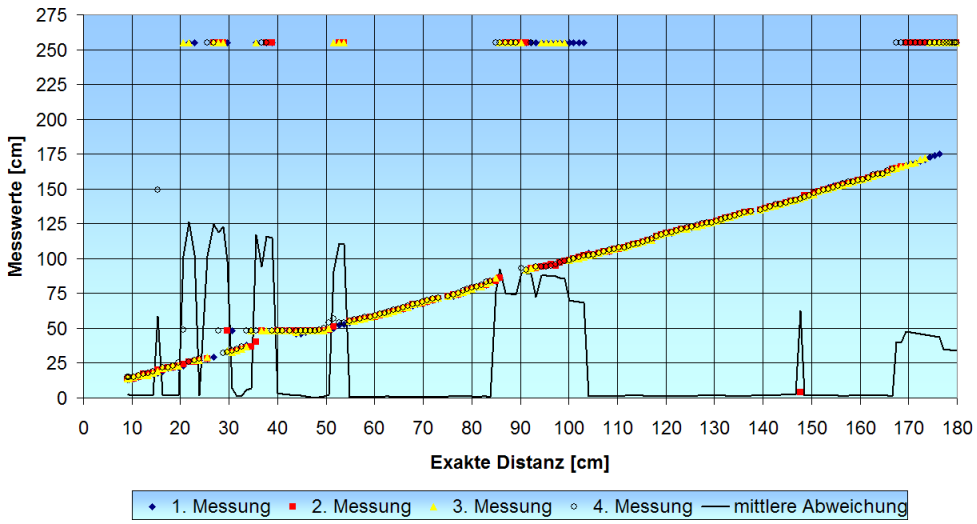


Abbildung 3.12: Dynamischer Ultraschalltest mit Power 40

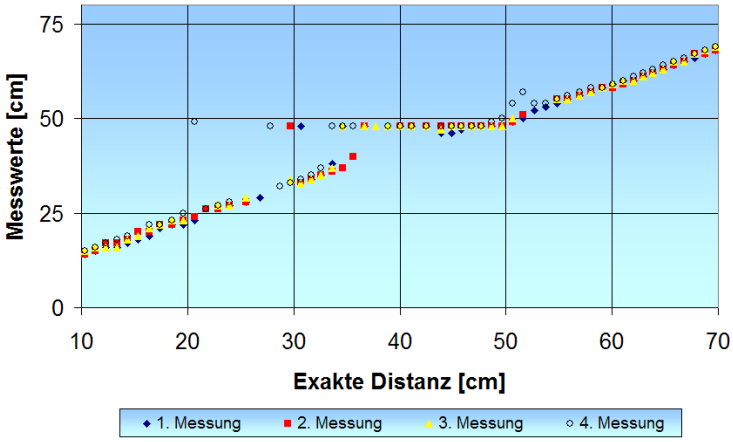


Abbildung 3.13: Detailsansicht des dynamischen Ultraschalltests im kritischen Bereich (Power 40)

Bei allen Messreihen zeigen sich Intervalle, in denen mehrere aufeinanderfolgende Messwerte einer Messreihe stark von den übrigen Messreihen im gleichen Intervall abweichen (üblicherweise wird hier der Wert 255 cm ausgegeben). Dies deutet auf ein zeitbeschränktes Aussetzen der Messfähigkeit hin, das sich in unregelmässigen Abständen manifestiert. Beim Vergleich der Testreihen zeigt sich, dass die Fahrtrichtung des Roboters keinen Einfluss auf die Messresultate ausübt.

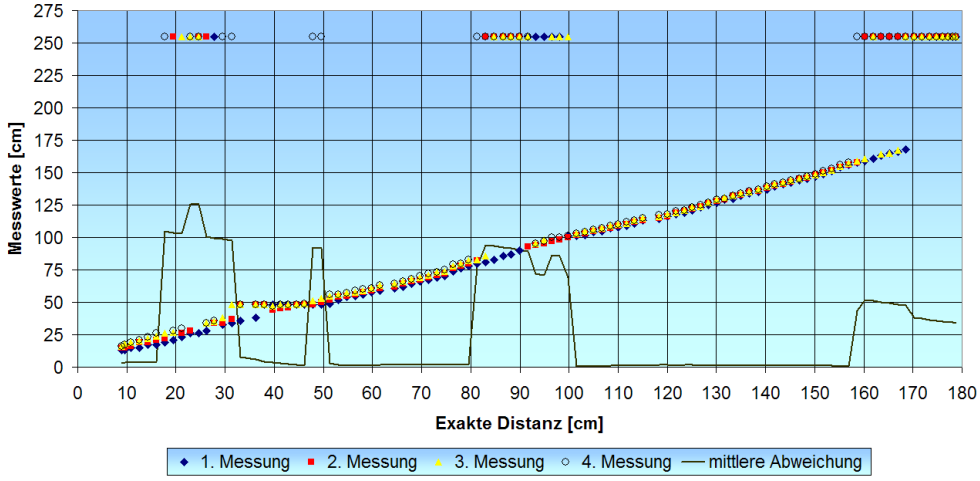


Abbildung 3.14: Dynamischer Ultraschalltest mit Power 60

Schlussfolgerung:
Dieser Versuch zeigt einerseits einen Schwachpunkt des Ultraschallsensors in einigen Distanzbereichen und andererseits eine weitere Unzulänglichkeit durch die zufälligen Aussetzer. Jedoch werden in diesem dynamischen System mehrheitlich exakte Distanzwerte gemessen.

3.5 Motorenversuche

1. Versuch
Fragestellung:

- Wie gross ist die kleinste Umdrehungseinheit?
- Wie wirken sich parallel laufende Motoren auf deren Geschwindigkeit aus?
- Besteht ein linearer Zusammenhang zwischen Power[b] und Umdrehungsgeschwindigkeit?
- Wie gross ist die maximale Rotatiosgeschwindigkeit (Umdrehungen pro Minute)?

Versuchsanordnung:
Ein Motor wird an den NXT-Baustein angeschlossen und via Software gesteuert. Die Versuche finden ohne externe Krafteinwirkung auf die Motorenachsen statt.

Messverfahren:
Zuerst wird das Verhalten eines einzelnen Motors untersucht, indem für verschiedene Powereinstellungen die benötigte Zeit pro Umdrehungen ermittelt wird. Danach werden zwei Motoren parallel angesteuert und der gleiche Versuch wird wiederholt.

Auswertung:
Die kleinste ausführbare Umdrehungseinheit entspricht einer Rotation von 1°. Bei einer maximalen Power von 100 benötigt ein Motor 0,44 Sekunden für eine Umdrehung. Die kleinste Powereinstellung, bei welcher sich der Motor noch dreht, beträgt 5. Wie in Abbildung 3.15 ersichtlich, nimmt die Umdrehungsgeschwindigkeit ab, sobald mehrere Motoren gleichzeitig angesteuert werden. Zusätzlich zeigt die Graphik den linearen Zusammenhang von Power und Umdrehungsgeschwindigkeit.

[b] Mit der Powereinstellung wird die Umdrehungsgeschwindigkeit eingestellt

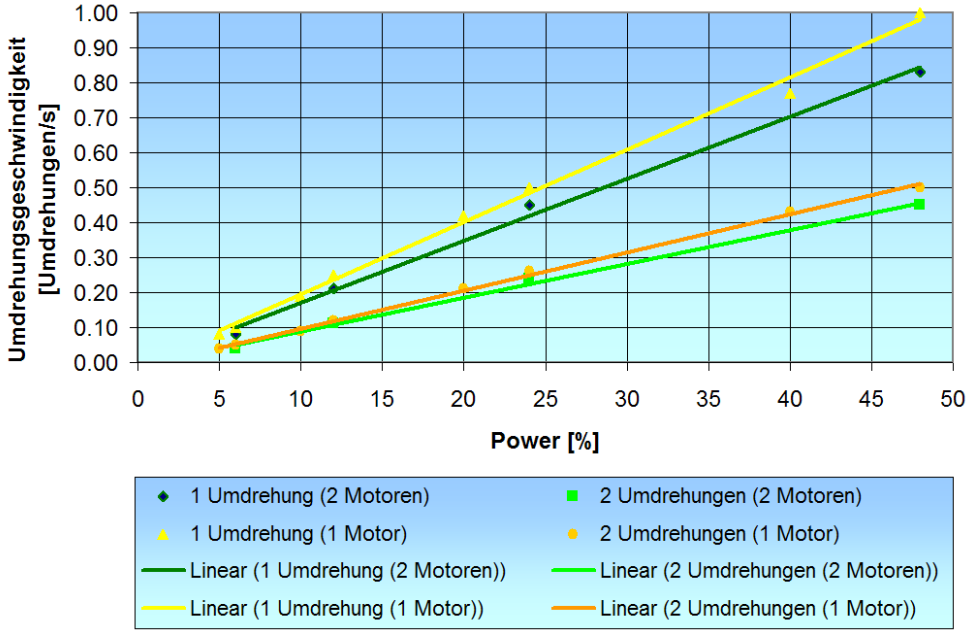


Abbildung 3.15: Umdrehungsgeschwindigkeit der Motoren im Vergleich

Schlussfolgerung:
Je mehr Motoren gleichzeitig mit gleicher Powereinstellung angesteuert werden, desto kleiner wird deren Umdrehungsgeschwindigkeit. Die kleinste Umdrehungseinheit von einem Grad entspricht dem vom Hersteller publizierten Wert.

2. Versuch
Fragestellung:

- Mit welcher Präzision können die Motoren eines Roboters angesteuert werden?
- Wie hoch ist die erzielte Motorengenauigkeit bei einer vorgegebenen Anzahl Umdrehungen?

Versuchsanordnung:
Der TriBot wird mit der LEGO-Software so programmiert, dass er einen vorgegebenen Pfad vorwärts und denselben rückwärts fährt. Für den zweiten Teil des Versuchs wird durch die Software eine bestimmte Anzahl Umdrehungen von den Motoren gefordert.

Messverfahren:
Im ersten Teil des Versuches wird die Startposition mit der Endposition verglichen und ausgewertet. Für den zweiten Teil wird ein Tool in der LEGO-Software genutzt, das die Anzahl der getätigten Umdrehungen anzeigt.

Auswertung:
Beim Abfahren eines vorgegebenen Pfades erreicht der TriBot seine Startposition mit einer Verschiebung von einigen Zentimetern, abhängig von der gefahrenen Distanz und Anzahl Kursänderungen. Dieses Verhalten bestätigt auch der zweite Versuch, bei dem die Anzahl Rotationen jeweils bis zu zwei Grad von den Geforderten abweicht.

Schlussfolgerung:
Trotz integriertem Rotationssensor ist keine hoch präzise Ansteuerung möglich.

3.6 Oszilloskop

Welche Messwerte werden von den Sensoren tatsächlich gemessen? Werden diese Messresultate durch den NXT-Baustein verändert bevor sie auf dem Display ausgegeben werden? Um diese Fragen zu beantworten wird ein T-Stecker angefertigt (Siehe Abb. 3.16), der zwischen dem jeweiligen Sensorausgang und dem NXT-Baustein-Eingang platziert wird. Von diesem T-Stecker aus leitet man die Sensorsignale direkt an ein Oszilloskop weiter.

Abbildung 3.16: Bild des T-Steckers zum Anschluss an ein Oszilloskop

3.6.1 Analoge Sensoren

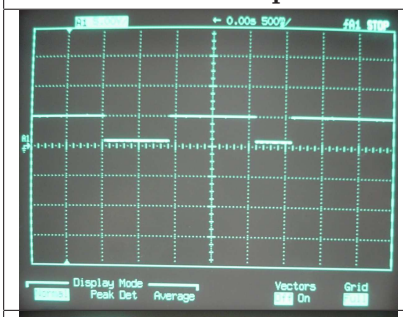
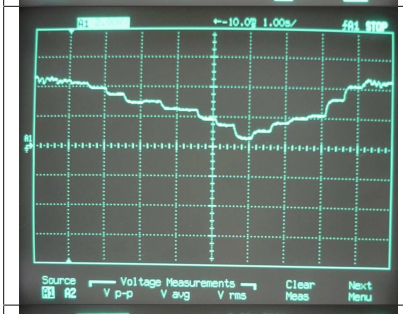
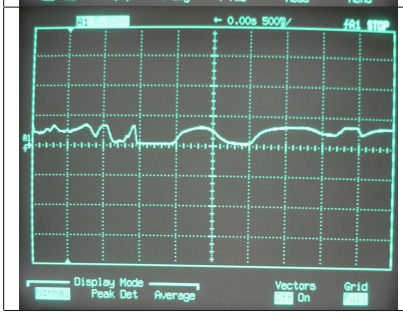
Oszilloskop	Beschreibung
	**Tastsensor** Der Tastsensor sendet dem Baustein entweder den Wert High oder Low, was in nebenstehendem Diagramm deutlich zu erkennen ist. Hier wurde der Tastsensor zwei Mal gedrückt.
	**Tonsensor** In diesem Diagramm sind die vom Tonsensor gemessenen Lautstärken im dB-Modus sichtbar. Die unterschiedlichen Lautstärken wurden durch ein Matlab Programm, das Sinusschwingungen generiert, erzeugt.
	**Lichtsensor** Der Lichtsensor funktioniert ähnlich wie der Tonsensor. Bei diesem Versuch wurden durch Variation der Helligkeit in der Umgebung des Sensors unterschiedliche Messwerte generiert.

Tabelle 3.1: Analoge Sensorsignale

3.6.2 Digitaler Sensor

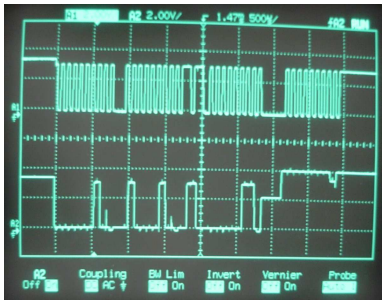
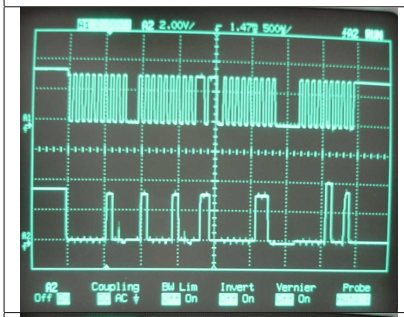
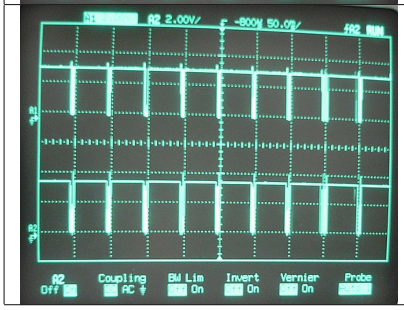
Oszilloskop	Beschreibung
	**Gemessene Distanz = 255 cm** In der oberen Hälfte ist das Signal dargestellt, das der Baustein an den Sensor schickt und in der Unteren, das in die entgegengesetzte Richtung gesendete Signal. Die Distanzwerte werden in den letzten Bits (rechts) kodiert.
	**Gemessene Distanz = 4 cm** Analoge Situation wie bei 255 cm, jedoch mit einer Distanz von 4 cm. (Vergleiche rechte Bits des unteren Signals in diesem Diagramm mit der gleichen Sequenz im oberen Diagramm)
	**Abfrage durch Polling** Alle 50 Millisekunden schickt der Baustein (oberes Signal) eine Anfrage an den Ultraschallsensor, der darauf hin ohne Zeitverzögerung den letzten Messwert (unteres Signal) an den Baustein sendet.

Tabelle 3.2: Digitale Sensorsignale

3.6.3 Motoren

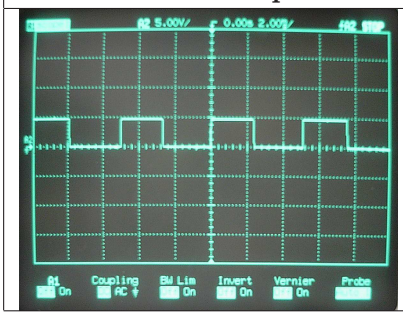
Oszilloskop	Beschreibung
	**PWM bei Power = 50** Das Signal, mit dem die Motoren angesteuert werden, ist mit Pulsweitenmodulation durch den NXT-Baustein generiert worden.

Tabelle 3.3: Motorsignal zur Regulierung der Geschwindigkeit

3.7 Kommunikation via Bluetooth

Mit dem NXT-Baustein kann entweder über die USB-Schnittstelle oder über die drahtlose Bluetooth-Schnittstelle kommuniziert werden. Bei Kommunikation via Bluetooth agiert der NXT-Baustein entweder als Master oder als Slave, wobei auf Seiten des Computers das Serial Port Profile zur Anwendung kommt. Wenn der NXT-Baustein als Master handelt, kann er mit bis zu drei weiteren Bluetooth-Geräten kommunizieren.

3.7.1 LMS2006 - LEGO Protokoll

Um dem LEGO-Baustein Befehle übermitteln zu können (sowohl via USB als auch Bluetooth), hat LEGO das LMS2006-Protokoll [3] definiert. Dabei handelt es sich um ein Protokoll mit fixer Instruktionslänge. Es unterscheidet dabei drei Grundarten von Kommandos:

- Direkte Kommandos [4] (0x00: Antwort erwartet, 0x80: keine Antwort erwartet)
- System-Kommandos [3] (0x01: Antwort erwartet, 0x81: keine Antwort erwartet)
- Bestätigungs-Kommando (0x02)

Beim Umschalten zwischen Sende- und Empfangsmodus ergibt sich in der Bluetooth-Hardware des LEGO-Bausteins eine Verzögerung von 30 ms. Daher werden zur Verbesserung der Kommunikationsleistung via Bluetooth jedem Paket 2 Bytes vorangestellt (siehe Abb. 3.17), die die Länge des zu sendenden Pakets im little-endian Format enthalten. Der LEGO Baustein antwortet vor den möglichen Nutzdaten zuerst mit einer Empfangsbestätigung, welche sich aus dem Bestätigungs-Kommando, einer Wiederholung des gesendeten Kommandos und einem Statusbyte zusammensetzt.

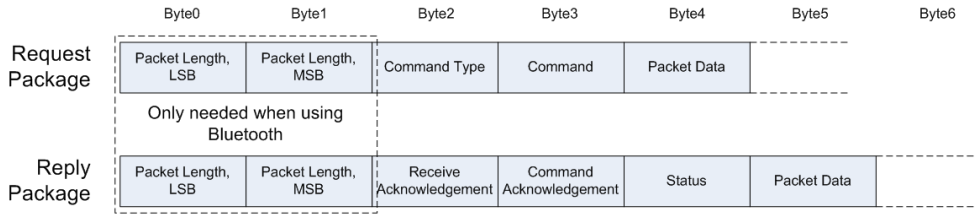


Abbildung 3.17: LMS2006-Protokoll

Das Softwarepaket von LEGO beinhaltet einen speziellen Bluetooth-Treiber namens Fantom, zu dem ein Software Development Kit zur Verfügung gestellt wird. Dieser Treiber implementiert Teile[c] des LMS2006-Protokolls, ermöglicht es aber auch direkte Kommandos auszuführen. Der Fantom-Treiber befindet sich dabei zwischen der LEGO-Software und dem Bluetooth Treiber.

3.7.2 Bluetoothkommunikation mit Perl

In den ersten Versionen der LEGO-Software war es noch nicht möglich Dateien direkt vom LEGO-Baustein auf den Computer zu laden, daher haben wir diese Funktion in Perl anhand des LMS2006-Protokolls implementiert. Da das Serial Port Profile (SPP) unterstützt wird, konnte für die Ansteuerung das Win32::SerialPort Packet in Perl verwendet werden.

[c]Zur Zeit sind nur System Kommandos implementiert

Kapitel 4

Projekt

Teil der Semesterarbeit ist die Definition und Umsetzung eines Projekts. Das Ziel des Projekts ist das Erforschen der Grenzen und Möglichkeiten der LEGO Mindstorms NXT Hard- und Software. Dabei sollen möglichst alle Sensoren und Motoren verwendet werden.

4.1 Planung

Anhand der im Vorfeld getätigten Sensorversuche soll ein Projekt definiert werden, das im Rahmen der zu erwartenden Genauigkeit der Sensoren und Motoren, die an das Projekt gestellten Anforderungen erfüllen kann.

4.1.1 Idee und Ziel

Unsere Projektdefinition lautet wie folgt: „Konstruktion und Programmierung eines Roboters, der einen a priori unbekannten, geschlossenen Raum erforscht und dabei eine Karte dieses Raums zeichnet, währenddem zusätzlich die Licht- und Lärmverhältnisse kontinuierlich aufgezeichnet werden. Die Wände des unbekannten Raumes sollten in etwa rechtwinklig angeordnet sein.“

In der Robotik existiert seit langem das Bedürfnis die Umgebung, in der sich ein Roboter befindet, zu erkunden, um sich darin autonom bewegen zu können, ohne mit unbeweglichen oder mobilen Objekten zu kollidieren. Diese Thematik wurde bereits von Buhmann [11] und Leonard [10] behandelt. Anhand dieser Berschreibungen muss der Roboter folgende Teilaufgaben lösen:

- Bewegungsplanung / Kollisionsverhinderung
- Konstruktion einer Karte aus den Sensordaten
- Lokalisation im Raum

Je länger ein Roboter unterwegs ist, desto grösser wird der Fehler zwischen tatsächlicher und vermuteter Position im Raum. Um diesen Ungenauigkeiten Rechenschaft zu tragen, werden die aktuellen Messwerte des Ultraschall- oder Lasersensors mit dem bereits erforschten Teil des Raumes verglichen, um eine Aufenthaltswahrscheinlichkeit pro Kartengitterpunkt zu definieren. Diese Problematik wird in den Publikationen mit dem Begriff 'Lokalisation im Raum' bezeichnet. Hierbei kommen komplexe Algorithmen zum Einsatz, die mit Hilfe der Wahrscheinlichkeitstheorie eine Erhöhung der Genauigkeit erzielen.
Da der Vergleich mit den bereits bekannten Daten viel Rechenleistung beansprucht, wird dieser Teil in unserem Projekt nicht berücksichtigt. Die von uns verwendete Lokalisation beschränkt sich also auf das Auslesen der Rotationssensoren der Motoren.

4.1.2 LabView vs. NXT Byte Codes

Bei der Wahl der zu verwendenden Programmiersprache fällt die Auswahl auf „Next Byte Codes“ (NBC) [7], da die Einschränkungen bei LabView zu gross sind. Zum Zeitpunkt der Projektdurchführung stehen keine weiteren Programmiersprachen zur Verfügung.

Eigenschaft	**NBC**	**LabView**
Arrays	Ja	Nein
Dateizugriff	Ja	Eingeschränkt [a]
Funktionen	Ja	Eingeschränkt [b]
Semaphoren	Ja	Nein
Definition von Variablen	Ja	Eingeschränkt [c]

Tabelle 4.1: NXT Byte Codes vs. LabView

[a] Nach jedem Schreibzugriff wird eine Newline eingefügt
[b] Keine Procedure Calls möglich
[c] Variablen nur vom Typ Logic, Number oder Text

4.2 Umsetzung

Bei der Umsetzung wird als Erstes ein Roboter konstruiert.
Als Zweites wird die Umsetzung der Teilaufgaben, die der Roboter Softwaretechnisch erfüllen muss, in Angriff genommen. Dabei haben wir den Teil der Kartenkonstruktion in Programm Explorer #1 implementiert. Die Bewegungsplanung und Kollisionsverhinderung wurde in Explorer #2 umgesetzt. Neben diesen Hauptaufgaben wurden auch sämtliche kleineren Aufgaben wie z.B. das Auslesen des Lichtsensors oder das Abspeichern der Karte in diesen Programmen getestet. Nachdem wir mit den Resultaten zufrieden

waren, wurden die gewonnen Erkenntnisse im Programm Explorer #3, das nun den vollen Funktionsumfang enthält, vereint.

Abbildung 4.1: Testumgebung des Explorers

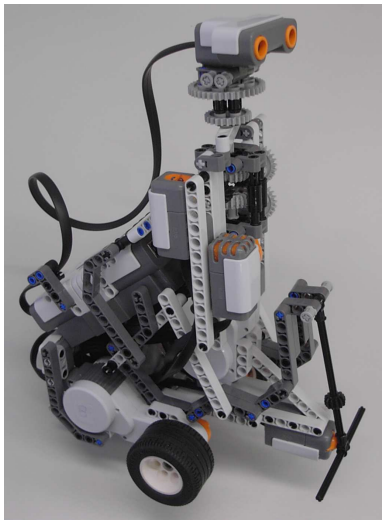


Abbildung 4.2: Bild des Explorers

4.2.1 Roboterkonstruktion

Der Roboter wurde so konstruiert, dass er folgenden Ansprüchen genügt:

- Fahrbare Dreirad-Konstruktion, die an Ort und Stelle Drehungen von ± 90 Grad vollführen kann, ohne dass sich der Mittelpunkt des Roboters um mehrere Zentimeter verschiebt, damit das Wenden in engen

Räumen möglich ist.
- Sensorturm, an dem der Licht- und Tonsensor vertikal nach oben gerichtet montiert werden kann, damit einerseits die Lichtqualität des Raumes wiedergegeben wird und andererseits Motorengeräusche durch die Ausrichtung des Mikrophons abgeschwächt werden.
- Drehbare Plattform, auf welcher der Ultraschall Sensor montiert werden kann, damit sich dieser unabhängig von der fahrenden Basis bewegen kann.
- Der Schwerpunkt des Roboters sollte möglichst nahe bei der Motorenachse liegen, um eine möglichst hohe Stabilität zu erreichen.
- Die Dimensionierung des Roboters sollte möglichst kompakt sein, damit sich dieser auch in engen Räumen fortbewegen kann.

Bei der Konstruktion des Roboters zeigt sich eine grosse Schwäche des LEGO NXT-Kits, zumindest bei der Vorabversion. Die Steckverbindungen, wie sie LEGO TECHNIC schon seit Jahren verwendet, tragen das Gewicht des NXT-Bausteins und der Motoren nicht genügend, so dass sich diese leicht verbiegen. So konnte auch nach langem Tüfteln keine optimale Lösung für das dritte Rad am Explorer gefunden werden, welches genügend Stabilität und trotzdem geringe Reibung aufweist. Die von LEGO vorgeschlagene Lösung (siehe TriBot) ist nur in den ersten Stunden geeignet, solange die einzelnen Teile keine Verschleisserscheinungen zeigen. Dieses mechanische Problem führt dazu, dass die am Explorer eingesetzte Lösung bei Kurvenfahrt zu einer erhöhten Reibung führt. Darum ist der Explorer in der momentanen Bauart nur auf einem Untergrund mit geringer Haftreibung (z.B. ebener Holztisch) einsetzbar. Falls es in Zukunft eine mechanische Lösung für dieses Problem geben wird, ist der Explorer auch im Stande auf anderen Oberflächenstrukturen wie z.B. Teppich oder Karton seine Aufgaben zu erfüllen.

4.2.2 Explorer #1

Dieses Programm veranlasst den Roboter in 5 cm-Schritten viermal geradeaus zu fahren, während er in jedem Schritt mit dem Ultraschallsensor bei einer 360° Rundumsicht in 5° Schritten die gemessen Distanzen von Polakoordinaten in Kartesische Koordinaten nach folgendem Muster übersetzt:

```
x = Aktuelle Position - X Koordinate
y = Aktuelle Position - Y Koordinate
for (n=0; n<4; n++)
{
  for (phi=-180; phi<180; phi+=5)
  {
    r = Ultraschall-Messung
```

```
    dx = r * cos(phi)
    dy = r * sin(phi)
    mx = x + dx
    my = y + dy
    StoreMapPoint(mx,my)
  }
  x+=5;
}
```

Abbildung 4.3: Bild des Sensorturms

Die Funktionsweise wird in Abbildung 4.4 dargelegt.
Um den Abstand vom ersten zum zweiten Messpunkt in der Karte masstabgetreu einzutragen, haben wir das Verhältnis von Rotationseinheiten der Antriebsmotoren (B und C) zum gemessenen Weg ermittelt. Dieses Verhältnis entspricht 21.6 Winkelgrad pro zurückgelegtem Zentimeter bei den im NXT-Kit enthaltenen Rädern.
Die präzise Ansteuerung der Motoren erwies sich während des Projekts als grösste Herausforderung. In der NBC-Dokumentation wird weder eine ausführliche Beschreibung noch ein nützliches Beispiel bezüglich korrekter Motorensteuerung präsentiert. Im Verlaufe des Projekts wurden mehrere Ansätze zur Lösung dieses Problems versucht, die in späteren Explorer-Versionen ebenfalls thematisiert werden. Nach dem Inbetrieb setzen der Motoren wird im ersten Ansatz in einer Schleife der Motorenwinkel ständig ausgelesen. Erst wenn die geforderte Anzahl Rotationseinheiten erreicht ist, werden die Motoren gestoppt. Dieses Verfahren wies je nach Motorengeschwindigkeit eine Ungenauigkeit von bis zu mehreren dutzend Rotationsgrad auf.
Ein weiteres Problem entsteht, wenn ein Motor in kurzen Zeitabständen immer wieder angesteuert wird. Das Betriebsystem des LEGO-Bausteins

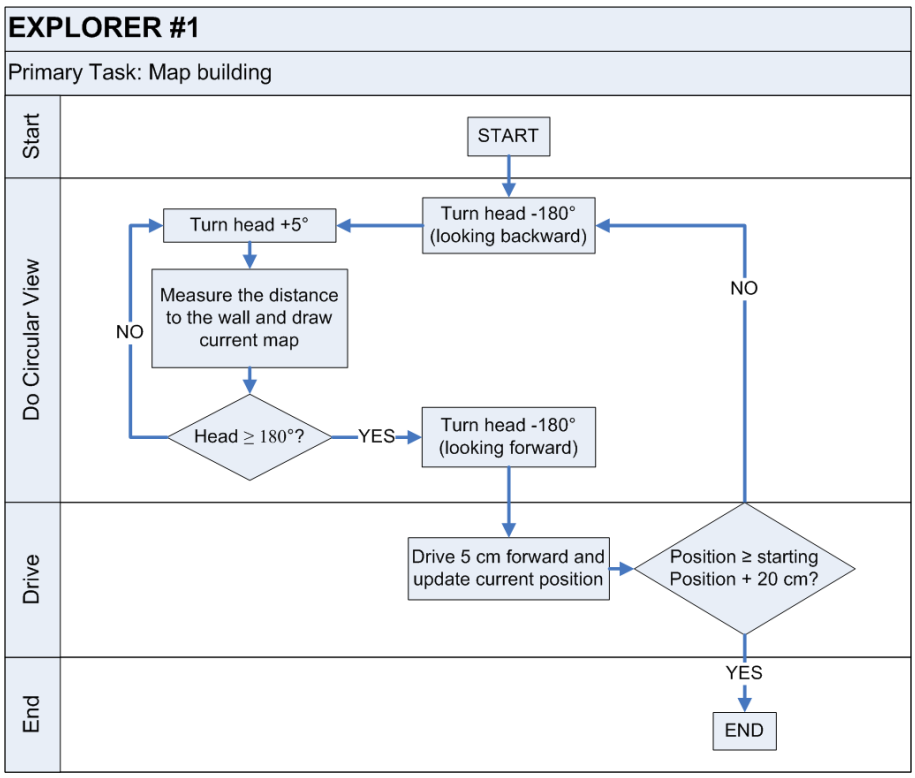


Abbildung 4.4: Funktionsflussdiagramm von Explorer #1

ist dabei nicht mehr im Stande, die genaue Anzahl Rotationseinheiten des angesteuerten Motors korrekt zu aktualisieren. Die Differenz zwischen dem ausgelesenen und dem tatsächlichen Rotationswert beträgt je nach Rotationsgeschwindigkeit mehrere Grad. Um dieses Problem zu beheben, muss nach dem Befehl zum Anhalten des Motors jedes Mal eine Pause von mindestens 50 ms eingelegt werden, in der sich der Rotationswert stabilisieren kann.

4.2.3 Explorer #2

Im Programm Explorer #2 detektiert der Roboter mit dem Ultraschallsensor als erstes die aus seiner Sicht nächste Wand, zu der er bis zu einem Abstand von 20 cm hinfährt und sich dann parallel zu ihr positioniert. Diesen Standort definiert er als Ausgangspunkt und fährt dann den Raum ab, bis er zu einem späteren Zeitpunkt diesen Ausgangsort wieder erreicht.
Damit der Roboter einen unbekannten Raum erforschen kann, benötigt er eine Strategie um auftretenden Hindernissen auszuweichen. Der von uns dafür angewendete Algorithmus veranlasst den Roboter stets der Wand auf seiner linken Seite entlang zu fahren. Sobald er linkerhand keine Wand mehr

erkennen kann, muss er sich um 90° nach links drehen. Falls ein Hindernis in Fahrtrichtung in einer Distanz kleiner als 20 cm erkannt wird, dreht er sich um 90° nach rechts. In Abbildung 4.5 wird diese Bewegungsplanung beschrieben. Alle 5 cm bleibt er kurz stehen, um die Distanzen zu den Hindernissen neu zu eruieren und in einem Array zu speichern.
Damit das Programm festellen kann ob die Raumerforschung komplett ist, prüft es vor jedem Eintrag in das Array, ob schon ein Eintrag vorhanden ist. Sobald dies der Fall ist speichert das Programm das Array als Bild im TGA-Format [9].
Aus jeder Bewegung des Roboters resultiert eine Abweichung zwischen aktueller und vermuteter Position. Durch wiederholtes, fehlerbehaftetes Wenden und geradeaus Fahren des Roboters kumulieren sich diese Fehler, im Extremfall so stark, dass der Roboter mit der Wand zu seiner linken kollidiert. Daher wurde bei Explorer #3 eine Kurskorrektur-Funktion hinzugefügt.

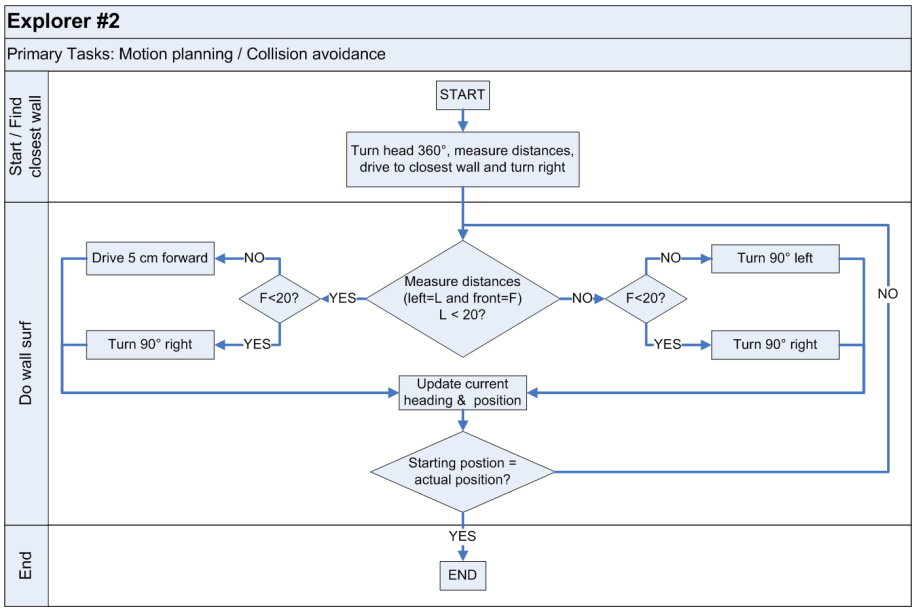


Abbildung 4.5: Funktionsflussdiagramm von Explorer #2

4.2.4 Explorer #3

In Explorer #3 werden die in Explorer #1 und #2 bereits gelösten Aufgaben übernommen. Das Programm wird durch die Messung von Umgebungsbedingungen und durch die Fähigkeit der Kurskorrektur erweitert.

Ziel der Raumerkundung ist einen möglichst ruhigen Ort mit guten Lichtverhältnissen zu finden. Dazu werden die Umgebungsverhältnisse an jedem durch den Roboter erforschten Punkt im Raum ausgelesen und mit folgen-

der Gewichtung ausgewertet:

$$\text{Umgebungswert} = \frac{100 + \text{Helligkeitswert} - \text{Schalldruck}}{2}$$
$$\text{Umgebungswert, Helligkeitswert, Schalldruck} \in [0, 100]$$

Die Kurskorrektur erfolgt mit Hilfe eines FIFO-Buffers, der den aktuellen und die letzten 3 vergangenen Distanzmessungen zur linken Wand enthält und diese miteinander vergleicht. Bei diesem Vergleich wird ermittelt, mit welcher Wahrscheinlichkeit sich der Roboter entweder der linken Wand nähert, oder sich von dieser entfernt. Sobald diese Wahrscheinlichkeit einen Schwellwert überschreitet wird der Kurs korrigiert, so dass der Roboter sich möglichst parallel zur Wand fortbewegt.

Die Abhängigkeiten der Prozeduren in Explorer #3 sind in Abbildung 4.6 dargestellt. Insgesamt besteht das Programm aus 17 Funktionen. Da es sich bei NBC um eine prozedurale Programmiersprache handelt, wurde grossen Wert auf die saubere Trennung von lokalen und globalen Variabeln gelegt. Im Anhang C befindet sich der Quelltext zu Explorer #3.

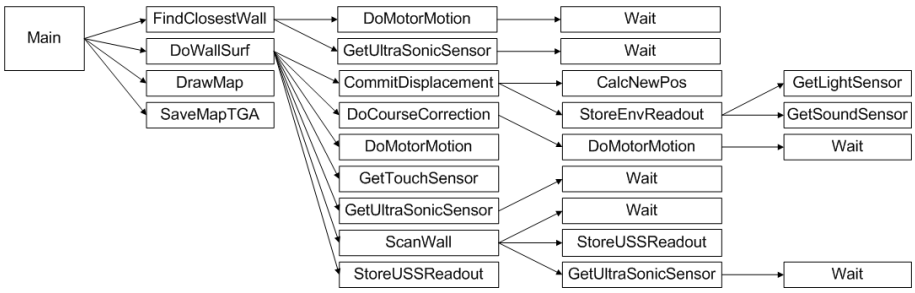


Abbildung 4.6: Abhängigkeit der Prozeduren

Da sich auch nach mehreren Wochen noch keine Lösung für eine präzise Motorensteuerung finden liess, haben wir zusammen mit dem Autor des NBC-Kompilierers [7] einen neuen Lösungsansatz erarbeitet. Dabei hat sich herausgestellt, dass es zum gegenwärtigen Zeitpunkt keine allgemein bewährte Methode gibt. Die nun verwendete Funktion macht von den im Betriebssystem des LEGO-Bausteins integrierten Möglichkeiten der Motorenregulierung gebrauch. Als grosser Nachteil erweist sich der hohe Energieverbrauch des Regulierungsprozesses. Nach einigen Minuten Betrieb mit regelmässiger Motorregulierung sinkt die Quellspannung so stark, dass sich der NXT-Baustein ausschaltet.

4.3 Resultat

Um die Genauigkeit der Software im Zusammenspiel mit der Hardware zu ermitteln, wird eine Testumgebung (siehe Abb. 4.1) aus geeigneten Materialien (in unserem Fall Kartonschachteln) aufgebaut. Die Testumgebung wird vermessen und anschliessend mit der Karte, die der Explorer #3 erzeugt hat, verglichen. Für das Erkunden dieses Raumes benötigte der Roboter etwa 8 Minuten.

Bei der vom Explorer #3 generierten Karte entspricht 1 Pixel einer Fläche von 3 cm x 3 cm. Berücksichtigt man die ± 3 cm Messungenauigkeit des Ultraschallsensors, entspricht die generierte Karte im Grossen und Ganzen den exakten Abmessungen der Testumgebung (siehe Abb. 4.7). Vereinzelte Fehler in der generierten Karte sind auf Aussetzer des Ultraschallsensors wie auch auf Phantom-Wände[a] zurückzuführen. Durch die Kurskorrektur entstehen z.T. Abweichungen von einigen Grad eines gradlinigen Pfades.
Das Fehlen der Lokalisationsfunktion führt dazu, dass die generierte Karte sehr oft ein auf- oder zuklappendes Verhalten zeigt (siehe Abb. 4.8). Diese Fehlerquelle kann durch einen hohen Batterieladestand minimiert werden. Je schwächer die Batterien desto stärker tritt dieser Effekt zu Tage.

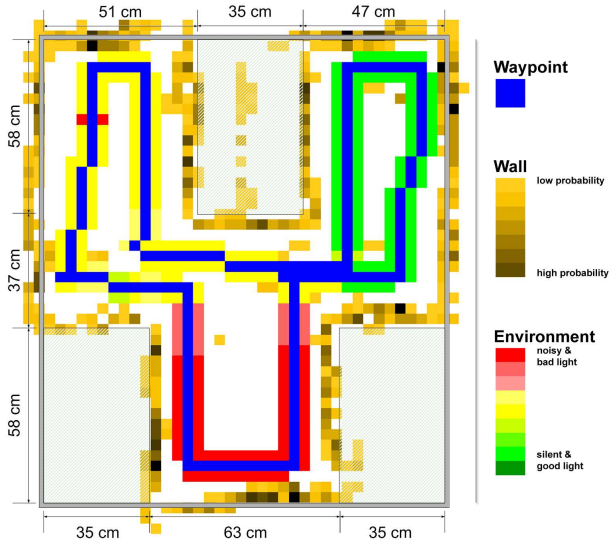


Abbildung 4.7: Gelungene Auswertung der Raumerkennung des Explorers #3, bedingt durch vollen Batterieladestatus

[a]Wenn die Ultraschallwelle an mehr als einer Wand reflektiert wird, entstehen Phantom-Wände.

Abbildung 4.8: Auswertung der Raumerkennung des Explorers #3 mit Problemen, bedingt durch schwache Batterien

4.4 Fazit

4.4.1 Erkentnisse

Trotz Schwierigkeiten und Hindernissen, gegeben durch die spärliche Dokumentation des „NXT Byte Codes“ und den Limiten der Hardware, konnte das hochgesteckte Ziel der Raumerforschung im kleinen Rahmen zufriedenstellend erreicht werden. Als grösste Hindernisse erwiesen sich hierbei die mangelnde Präzision der Motorensteuerung wie auch das Fehlen einer Lokalisationsfunktion. Auch das zeitweise Versagen des Ultraschallsensors führte manchmal zu ungewollten und schwer vorhersagbaren Effekten. Die Programmierung in NBC und das Übertragen der Programme auf den NXT-Baustein via Bluetooth erwiesen sich als solide. In seltenen Fällen musste die Hardware neu gestartet werden, danach war die fehlerfreie Kommunikation wieder gegeben.

4.4.2 Ausblick

Aus Zeitgründen konnten nicht alle während der Arbeit am Projekt entstanden Ideen realisiert werden. Folgende Punkte wären bei einer Weiterführung des Projekts von primärem Interesse:

- **Nicht rechtwinklige Raumecken** – In der aktuellen Programmversion kann der Roboter seine Fahrtrichtung bereits einer nicht gradlinigen Wand anpassen, um Kollosionen zu vermeiden. Interessant wäre es, die Funktionalität so zu erweitern, dass beliebige Räume, die der Einschränkung der nahezu Rechtwinklingen Ecken nicht mehr genügen, erfoscht werden können.
- **Lokalisation** – Wie bereits in Kapitel 4.1.1 dargelegt wird, würde diese Funktion die Präzision der Karte und damit die Grösse des zu erforscheden Raumes erhöhen.
- **Bluetooth** – Zur Zeit muss die gespeicherte Datei manuell vom NXT-Baustein heruntergeladen werden. Ziel wäre es, nach dem Program-

mende die gewonnene Karte automatisch via Bluetooth zu einem Computer zu übertragen.

- **Prallelisierung** – Das aktuelle Programm führt seine Funktionen sequenziell aus. Durch Paralellsierung einzelner Programmschritte könnte der Erforschungsablauf beschleunigt werden.

Kapitel 5

Schlussfolgerung

Die in der Anfangsphase durchgeführten Sensor- und Motorentests geben einen vielseitigen Einblick in die Funktionsweise der NXT-Komponenten und zeigen die Stärken und Schwächen dieser Komponenten. Zum Beispiel besitzt der Ultraschallsensor in einigen Distanzbereichen mangelnde Zuverlässigkeit, während sich die gezielte Ansteuerung der Motoren innerhalb eines kleinen Fehlerbereichs gut bewältigen lässt. Anhand der durchgeführten Versuche kann der Einsatzbereich, wie auch das zu erwartende Resultat der einzelnen Bestandteile des NXT-Kits abgeschätzt werden.
Auf den Unterschieden zwischen digitalen und analogen Sensoren aufbauend, stellen die Beobachtungen der Signale mit Hilfe eines Oszilloskops und die technischen Angaben zu den Bauteilen ein Fundament für die Entwicklung neuartiger Sensoren dar.
Anschliessend an die Sensorversuche folgen Untersuchungen zur Kommunikation mittels Bluetooth. Durch Analyse des LMS2006-Protokolls wird ein Perlskript zur Kommunikation mit dem NXT-Baustein entwickelt, welches Dateizugriffe auf den NXT-Baustein ermöglicht. Dieses Skript stellt die Basis für die Umsetzung weiterer Teile des Protokolls, sowohl für die Windows-, als auch für die Unixarchitektur dar. Zu erforschen sind die Möglichkeiten, die sich durch die Kommunikation zwischen mehreren Bluetooth-Geräten ergeben.
Anhand der gewonnenen Erkentnisse war nun die Umsetzung eines eigenen Projekts das Ziel. Die Idee eines Roboters, der möglichst alle Sensoren[a] sinnvoll einsetzt, führte zu unserem Projekt, das sich der Erforschung eines unbekannten Raumes widmet. Eine wichtige Rolle spielt der Ultraschallsensor, der für die Ausmessung des Raums verantwortlich ist. Da zur Positionsbestimmung lediglich die aktuellen Rotationswerte der Räder berücksichtigt werden, ergibt sich über die Zeit ein immer grösserer Fehler. Um diesen Fehler genügend gering halten zu können, müssen die Motoren mit höchstmög-

[a] Vorallem der Ultraschallsensor und die Rotationssensoren in den Motoren sind von primärem Interesse

licher Präzision gesteuert werden, was sich als schwierig erweist.
Die während des Projekts demonstrierte Zuverlässigkeit und die im Resultat erzielte Genauigkeit zeugen von der Ausgereiftheit der Vorabversion des Lego-Produktes.
Der in „Next Byte Codes“ programmierte Quellcode kann als Referenz für die Steuerung von Sensoren wie auch Motoren genutzt werden. Fehler im Resultat gründen auf der Absenz einer Lokalisationsfunktion und auf der Abhängigkeit der Präzision von der Versorgungsspannung.
Vergleiche mit einer höheren Programmiersprache könnten zeigen, welche Aufgaben sich mit NBC besonders elegant lösen lassen und welche Nachteile sich durch NBC ergeben. Vorallem im Bereich der Motorensteurung dürfte dies von grossem Interesse sein, da hier die höchste Präzision mit gleichzeitig geringem Energieverbrauch gefordert ist. Eine weitere wichtige Frage ist die Möglichkeit von Alternativen zu dem von Lego konzipierten Betriebssystem.
Durch die Steigerung der Rechenleistung des Bausteins, wie auch dessen neue zusätzliche Fähigkeit Daten mit digitalen Sensoren auszutauschen, ist es möglich die Erkennung eines Hindernisses ohne Berührung festzustellen und diese in einer Karte des Raums zu erfassen. Die kabellose Kommunikation, auch ausserhalb der Sichtweite, und die mehrzeilige Anzeige erhöhen die Benutzerfreundlichkeit im Vergleich zum älteren RCX-Modell wesentlich.
Als grösste Nachteile erweisen sich, wie auch schon beim RCX-Modell die Abhängigkeit der Sensor- und Motorpräzision von der Versorgungsspannung und die unzureichende Belastbarkeit der neuen LEGO TECHNIC-Bauteile.
Da es sich bei dem von LEGO zur Verfügung gestellten NXT-Entwicklerkit um eine Vorabversion handelt, kann es durchaus sein, dass bis zur Markteinführung noch einige Änderungen vorgenommen werden. Interessant wäre zu wissen, ob der Ultraschallsensor zukünftig immer noch die selben Unzulänglichkeiten besitzt.

Literaturverzeichnis

[1] LEGO Dokumentation,
LEGO MINDSTORMS User Guide,
2006.

[2] LEGO Dokumentation,
LEGO MINDSTORMS documentation,
2006.

[3] LEGO Dokumentation,
LEGO LMS2006 protocol,
2006.

[4] LEGO Dokumentation,
LEGO LMS2006 Direct Commands,
2006.

[5] Offizielle LEGO Homepage,
http://www.lego.com/,

[6] Produkt Homepage von LEGO Mindstorms,
http://mindstorms.lego.com/,

[7] Next Byte Codes - alternative Programmiersprache
Autor: John Hansen 2006,
http://bricxcc.sourceforge.net/nbc/,

[8] Philips - Angaben zum I^2C Protokoll,
http://www.semiconductors.philips.com/i2c,

[9] Spezifikation des Targa Bildformats,
http://www.scs.fsu.edu/~burkardt/pdf/targa.pdf,

[10] Leonard et al., Dynamic map building for an autonomous mobile robot. *Int. J. Rob. Res. 11*, 4(Aug.) 1992, 286-298.

[11] Buhmann et al., The Mobile Robot Rhino. *AI Magazin*, Vol. 16 Number 1, 1995.

Anhang A

Aufgabenstellung

Betreuer:	Christian Plessl <plessl@tik.ee.ethz.ch>
Co-Betreuer:	Andreas Meier <andreas.meier@tik.ee.ethz.ch>
Studenten:	Claudia Frischknecht <fclaudia@ee.ethz.ch> Thomas Other <tother@ee.ethz.ch>
Dauer:	3. April 2006 – 7. Juli 2006

A.1 Einführung

Die Firma LEGO bietet seit einigen Jahren das Produkt „LEGO Mindstorms“ an. Mindstorms ist eine Plattform die Lego Technik Bausteine mit einem Mikrocontroller System kombiniert. Zusätzlich zum Mikrocontroller Baustein werden elektromechanische Teile wie Motoren und Tastsensoren geliefert. Diese Bausteine erlauben es, autonome Roboter mit den vielfältigsten Aufgaben zu konstruieren und programmieren.

Obwohl LEGO Mindstorms ursprünglich als technisches Spielzeug konzipiert wurden, fanden sie in an vielen Schulen und Universitäten Einzug in die Lehre, da sich Mindstorms gut eignen, um Studenten in die Thematik von eingebetteten Systemen einzuführen. Auch unser Institut bietet seit vielen Jahren ein PPS Seminar an, in dem eine Studentengruppe selbständig ein Mindstorms Projekt durchführt.

Im Rahmen dieser Projekte sind die Studenten oft an die Grenzen der Mindstorms Hardware vorgestossen. Es hat sich gezeigt, dass die mitgelieferten Sensoren und Aktoren den hohen Präzisionsanforderungen bei komplexen Aufgaben nicht immer gerecht werden können. Diese Nachteile wurden auch von anderen Anwendern und von LEGO selbst erkannt und LEGO hat vor kurzem nächste Generation von LEGO Mindstorms vorgestellt, welche diese Nachteile beheben sollen.

Die neue Mindstorms Generation heisst „Mindstorms Next Generation (NXT)“ uns soll im Herbst 2006 in den Verkauf kommen. Die Mind-

storms NXT sind den bisherigen Mindstorms in vielerlei Hinsicht überlegen. Die Zentraleinheit hat nun eine 32bit CPU mit wesentlich mehr Rechenleistung. Die Sensoren und Aktoren wurden gründlich überarbeitet und erlauben nun eine präzisere Erfassung von Sensordaten (Taster, Mikrofon, Licht, Ultraschall-Bewegungssensor) und eine präzisere Ansteuerung der Motoren. Eine integrierte Bluetooth-Schnittstelle ermöglicht die drahtlose Kommunikation mit dem Mindstorms-Roboter. LEGO stellt 100 Mindstorms NXT Entwicklerkits als Vorabversion zur Verfügung. Unser Institut hat eines dieser Entwicklerkits erhalten.

A.2 Aufgabenstellung

Wir möchten diese neuen Mindstorms NXT in Zukunft auch in unseren Studentenprojekten einsetzen und möchten in dieser Arbeit erste Erfahrungen mit der neuen Hardware erhalten. Ihre Ergebnisse werden in die Ausgestaltung der zukünftigen PPS Praktika einfliessen.

Diese Arbeit hat im wesentlichen zwei Zielsetzungen: Einerseits sollen die Eigenschaften der einzelnen Komponenten der Mindstorms NXT untersucht werden (Genauigkeit, Zuverlässigkeit, Anwendungsgebiete). Dadurch erhoffen wir uns Kenntnisse über das Potential und die Limitation der neuen Sensoren und Aktoren. Zweitens sollen Sie in Absprache mit Ihren Betreuern ein Projekt definieren und umsetzen, welches die Möglichkeiten von Mindstorms NXT mit einer konkreten Anwendung demonstriert.

A.3 Teilaufgaben

1. **Grundlagen Minstorms**
   Machen Sie sich mit den Grundlagen der Mindstorms NXT und der Software Entwicklungsumgebung vertraut. Arbeiten Sie dazu das mitgelieferte Tutorial durch. Verfolgen Sie auch die Entwicklungen und Diskussionen auf den Mindstorms Entwickler Foren.
2. **Sensor Tests**
   Untersuchen Sie die mitgelieferten Sensoren und Aktoren. Entwickeln Sie dazu eine Reihe von reproduzierbaren Tests, welche die Genauigkeit und Zuverlässigkeit der Sensoren testet. Protokollieren Sie ihre Testbedingungen und Messresultate.
3. **Bluetooth Kommunikation**
   Untersuchen Sie wie die Bluetooth Kommunikationsschnittstelle in einer Anwendung benutzt werden kann. Ist eine Interaktion von Mindstorms mit anderen Bluetooth Geräten, z.B. mit einem Mobiltelefon oder Computerperipheriegeräten möglich? Falls ja, welche Funktionen können implementiert werden.

4. **Definition Projekt**
   Definieren Sie in Absprache mit Ihren Betreuern ein Projekt, das die Leistungsfähigkeit der Minstorm NXT mit einer konkreten Anwendung demonstriert. Erstellen Sie einen Plan zur Umsetzung des Projekts.
5. **Umsetzung Projekt**
   Setzen Sie das Projekt gemäss Ihrem Plan um.
6. **Feedback an Lego**
   Wir möchten LEGO gerne ein Feedback zur Vorabversion des Mindstorm Entwicklerkit geben. Dokumentieren Sie Ihre wichtigsten Erfahrungen mit dem Entwicklerkit dazu in einem kurzen Bericht, den wir LEGO als Feedback senden können.
7. **Erkenntnisse für PPS**
   Dokumentieren Sie die wichtigsten Erkenntnisse Ihrer Arbeit, welche für ein zukünftiges PPS Seminar wichtig sind.

A.4 Organisatorisches

- **Zeitplan** Erstellen Sie am Anfang Ihrer Arbeit zusammen mit dem Betreuer einen realistischen Zeitplan. Halten Sie Ihren Arbeitsfortschritt laufend fest.
- **Meetings** Vereinbaren Sie mir Ihren Betreuern einen festen wöchentlichen Termin für ein Treffen. Präsentieren Sie an diesem Treffen jeweils den aktuellen Stand der Arbeit und diskutieren Sie offene Fragen.
- **Dokumentation** Dokumentieren Sie Ihre Arbeit sorgfältig. Legen Sie besonderen Wert auf die Beschreibung Ihrer Überlegungen und Entwurfsentscheide. Beginnen Sie frühzeitig mit der Dokumentation und diskutieren Sie die Struktur mit Ihren Betreuern. Ihr Bericht ist ein wesentlicher Bestandteil Ihrer Arbeit und hat auch bei der Bewertung entsprechendes Gewicht.

Anhang B

Tabellen mit den Messresultaten

B.1 Lichtsensor

ambient light mode (weisses LED) [%]	1 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	39	22	20	52	56	17	100
Farbige Plastikstreifen auf Spiegel:	35	17	18	49	55	12	100
Farbige Papierstreifen auf schwarzem Plastik:	60	39	36	75	82	30	22
Farbige Plastikstreifen auf schwarzem Plastik:	52	28	32	73	83	19	22
	2 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	21	10	9	27	30	7	70
Farbige Plastikstreifen auf Spiegel:	18	9	7	26	30	7	70
Farbige Papierstreifen auf schwarzem Plastik:	21	12	10	29	31	10	10
Farbige Plastikstreifen auf schwarzem Plastik:	17	11	9	28	29	9	10
	3 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	10	5	3	14	17	4	52
Farbige Plastikstreifen auf Spiegel:	9	4	4	15	15	4	52
Farbige Papierstreifen auf schwarzem Plastik:	13	7	5	15	19	5	7
Farbige Plastikstreifen auf schwarzem Plastik:	7	7	7	19	19	6	7
	4 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	9	7	4	13	17	6	43
Farbige Plastikstreifen auf Spiegel:	8	4	5	15	16	6	43
Farbige Papierstreifen auf schwarzem Plastik:	6	5	3	10	12	5	5
Farbige Plastikstreifen auf schwarzem Plastik:	6	5	4	10	12	5	5

ambient light mode (Halogenlampe) [%]	1 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	38	22	17	48	48	18	100
Farbige Plastikstreifen auf Spiegel:	32	21	17	46	48	16	100
Farbige Papierstreifen auf schwarzem Plastik:	47	26	21	55	59	23	22
Farbige Plastikstreifen auf schwarzem Plastik:	34	22	18	52	62	20	22
	2 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	28	14	12	34	36	11	72
Farbige Plastikstreifen auf Spiegel:	25	13	11	34	37	11	72
Farbige Papierstreifen auf schwarzem Plastik:	27	15	11	38	39	13	10
Farbige Plastikstreifen auf schwarzem Plastik:	25	14	12	36	41	12	10
	3 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	19	10	7	29	30	9	51
Farbige Plastikstreifen auf Spiegel:	19	10	7	29	30	8	51
Farbige Papierstreifen auf schwarzem Plastik:	21	13	10	31	32	11	7
Farbige Plastikstreifen auf schwarzem Plastik:	21	13	11	31	33	10	7
	4 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	16	9	9	23	22	7	42
Farbige Plastikstreifen auf Spiegel:	16	9	8	23	24	6	42
Farbige Papierstreifen auf schwarzem Plastik:	16	10	9	21	20	8	5
Farbige Plastikstreifen auf schwarzem Plastik:	15	9	8	20	21	8	5

reflected light mode [%]	0 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	35	7	7	45	60	5	100
Farbige Plastikstreifen auf Spiegel:	31	4	4	41	45	4	100
Farbige Papierstreifen auf schwarzem Plastik:	18	5	5	25	29	4	3
Farbige Plastikstreifen auf schwarzem Plastik:	23	4	4	29	22	3	3
	1 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	74	34	32	78	76	25	100
Farbige Plastikstreifen auf Spiegel:	70	24	24	75	77	25	100
Farbige Papierstreifen auf schwarzem Plastik:	65	29	28	68	70	22	22
Farbige Plastikstreifen auf schwarzem Plastik:	62	21	22	68	72	22	22
	2 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	43	16	15	46	46	11	70
Farbige Plastikstreifen auf Spiegel:	40	11	12	44	46	10	70
Farbige Papierstreifen auf schwarzem Plastik:	38	15	13	40	40	10	10
Farbige Plastikstreifen auf schwarzem Plastik:	36	10	11	40	43	11	10
	3 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	28	10	9	30	31	7	52
Farbige Plastikstreifen auf Spiegel:	26	7	7	30	31	7	52
Farbige Papierstreifen auf schwarzem Plastik:	26	9	8	27	28	7	7
Farbige Plastikstreifen auf schwarzem Plastik:	25	7	7	27	30	7	7
	4 cm zwischen Objekt und Sensor						
	rot	grün	blau	gelb	weiss	schwarz	untergrund
Farbige Papierstreifen auf Spiegel:	21	7	7	22	23	5	42
Farbige Plastikstreifen auf Spiegel:	19	6	6	22	23	5	42
Farbige Papierstreifen auf schwarzem Plastik:	18	7	6	20	20	5	5
Farbige Plastikstreifen auf schwarzem Plastik:	17	6	5	20	21	5	5

B.2 Ultraschallsensor

	Ultrasonic-Display-Anzeige [cm]									
exakte Distanz	1. Messung	2. Messung	3. Messung	4. Messung	5. Messung	6. Messung	7. Messung	Durchschnitt	Abweichung	max. Abweichung
2.4	4	3	4	5	3	4	4	3.86	-1.46	2.6
3.6	4	4	4	4	4	4	4	4	-0.4	0.4
5.2	5	5	6	6	5	5	5	5.29	-0.09	0.8
6.4	6	6	6	6	6	6	6	6	0.4	0.4
7.6	8	7	8	8	8	7	8	7.71	-0.11	1.4
8.8	8	8	8	9	8	9	8	8.29	0.51	0.8
10	10	11	10	10	10	10	10	10.14	-0.14	1
11.4	11	12	11	11	12	11	11	11.29	0.11	0.6
12.4	12	12	12	12	13	12	12	12.14	0.26	0.6
13.6	13	13	13	13	13	13	13	13	0.6	0.6
									0.408	

horizontal	Ultrasonic-Display-Anzeige										
exakte Distanz	1. Körper	2. Körper	3. Körper	4. Körper	5. Körper	6. Körper	7. Körper	8. Körper	9. Körper	10. Körper	11. Körper
5	6	5	5	6	5	5	5	6	6	5	5
10	10	10	10	10	10	10	10	15	11	10	9
15	17	15	15	17	15	15	15	20 / 48	16	15	15
20	23	21	20	23		21	23	32	22	20	20
25	29	26	25	29		26		27	28	25	25
30		32	30		34	31		48	48	31	31
40		48	48		48	41	48	48	48	48	48
50	53	49	49	53	50	48	50	52	51	50	49
60	61	60	61		61	60		63	61	60	60
70	71	69	70			70	72	74 / 84	71	69	69
80		78	79			80		91	81	78	79
90		88	89			90		104	91	89	90
100		99	100			101			103	98	99
110		110	109			112				108	111
120		119	119			118				118	120
130		130	130			129				128	129
140			139			140				138	141
150			149			149					144
160			160			159					157
# ??????:	11	3	0	13	12	0	12	7	6	2	0
standart Abw.:	1.875	1.25	0.84	2.17	1.86	0.74	1.86	7.08	3.23	1.35	1.26
Körper-Eigenschaften											
Material:	Plastik	Plastik	Plastik	Plastik	Metall	Metall	Metall	Stoff	Stoff	Holz	Holz
Form:	Zylinder	Rechteck	Rechteck	Konus	Zylinder	Rechteck	Zylinder	Rechteck	Rechteck	Rechteck	Rechteck
Masse [cm]:	D = 12.5	12.5 x 8.5	19 x 13.5	D1 = 12.4	D = 16	40 x 33.5	D = 20	43 x 12.5	43 x 12.5	23 x 15	35 x 35
Was?:	Messbech.	Nes-ROM	DVD-Hülle	D2 = 12.7	Pfanne	Backblech	Pfanne	T = 10	T = 0	Holzbrett	Holzplatte
								Abtrocknungstuch			
Legende											
Leeres Feld = ?????? (keine Anzeige auf dem Display)											
D = Durchmesser											
T = Tiefe (vor Holz Hindergrund)											
vertikal / horizontal = Lage der Sensoraugen (vertikal: Augen von 4 - 9cm ab Tischfläche)											

			(li) = Befestigung des Sensors auf der linken Seite				
			(re) = Befestigung des Sensors auf der rechten Seite				
			Leeres Feld = ?????? (keine Anzeige auf dem Display)				
Rechteck							
horizontal	**Ultrasonic-Display-Anzeige**						
exakte Distanz	**30°**	**15°**	**7.5°**	**0°**	**7.5°**	**15°**	**30°**
5	6	5	5	5	5	5	5
10	14	12	10	10	10	10	11
15			17	16	16	16	18
20		22	21	21	21	21	23
25		27	27	26	26	27	29
30		33	31	31	31	32	33
40		48	42	43	42	48	48
50		50	49	50	49	50	50
60			61	61	61	61	
	Linke Seite vom Sensor				Rechte Seite vom Sensor		
Abweichung:	2.5	2.42	1.11	0.89	0.89	1.67	2.75
vertikal (li)	**Ultrasonic-Display-Anzeige**						
exakte Distanz	**30°**	**15°**	**7.5°**	**0°**	**7.5°**	**15°**	**30°**
5	5	5	5	5	5	5	6
10	11	10	10	10	10	10	12
15	18	16	16	16	16	18	
20	22	20	20	20	21	21	23
25							
30							
40	48	48	48	48	48	48	48
50	50	49	49	49	49	50	51
60	61	61	59	59	61	61	63
70	72	71	70	70	70	71	
80		80	80	80	81		
	Linke Seite vom Sensor				Rechte Seite vom Sensor		
Abweichung:	2.13	1.33	1.22	1.22	1.44	1.75	3
vertikal (re)	**Ultrasonic-Display-Anzeige**						
exakte Distanz	**30°**	**15°**	**7.5°**	**0°**	**7.5°**	**15°**	**30°**
5	8	6	4	5	5	4	5
10		11	10	9	10	9	11
15		17	15	16	14	16	16
20			23	21	22		
25			28	28	28	27	
30							
40		48	48	48	48	48	
50	51	50	49	49	49	49	50
60	64	61	61	61	60	60	61
70		71	70	70	70	69	70
80		82	81	80	80	80	81
	Linke Seite vom Sensor				Rechte Seite vom Sensor		
Abweichung:	2.67	2	1.8	1.6	1.6	1.67	0.57

Lederfussball (Durchmesser = 14cm)							
horizontal	Ultrasonic-Display-Anzeige						
exakte Distanz	30°	15°	7.5°	0°	7.5°	15°	30°
5	5	5	6	6	5	7	7
10		12	11		15	17	12
15				19	18	18	
20							
25							
30							
40							
50							
60							
	Linke Seite vom Sensor				Rechte Seite vom Sensor		
vertikal (li)	Ultrasonic-Display-Anzeige						
exakte Distanz	30°	15°	7.5°	0°	7.5°	15°	30°
5	5	5	5	5	5	6	10
10		10	11	11	12	18	
15		15	15	16	20	19	
20						24	
25							
30							
40			48	48		48	
50			52				
60				62	61		
70							
80							
	Linke Seite vom Sensor				Rechte Seite vom Sensor		
vertikal (re)	Ultrasonic-Display-Anzeige						
exakte Distanz	30°	15°	7.5°	0°	7.5°	15°	30°
5	7	6	5	5	5	5	5
10			11	10	10	10	10
15			18	18	16		
20				22	23		
25							
30							
40		48			48	48	
50		52	53		52	53	
60		63		63	64	62	
70							
80							
	Linke Seite vom Sensor				Rechte Seite vom Sensor		

B.3 Motoren

1 Motor (C) ohne Rad				
Power	48.0	24.0	12.0	6.0
Zeit für 1 Umdrehung [sek.]	1.0	2.0	4.0	10.2
Kehrwert der Zeit	1.00	0.50	0.25	0.10
Power	48.0	24.0	12.0	6.0
Zeit für 2 Umdrehung [sek.]	2.0	3.8	8.2	20.7
Kehrwert der Zeit	0.50	0.26	0.12	0.05
Speed	48.0	24.0	12.0	6.0
Zeit für 3 Umdrehung [sek.]	3.0	5.8	12.3	31.4
Kehrwert der Zeit	0.33	0.17	0.08	0.03
Power	40.0	20.0	10.0	5.0
Zeit für 1 Umdrehung [sek.]	1.3	2.4	5.2	12.5
Kehrwert der Zeit	0.77	0.42	0.19	0.08
Power	40.0	20.0	10.0	5.0
Zeit für 2 Umdrehung [sek.]	2.3	4.8	11.0	27.4
Kehrwert der Zeit	0.43	0.21	0.09	0.04
Power	40.0	20.0	10.0	5.0
Zeit für 3 Umdrehung [sek.]	3.9	7.3	16.9	41.4
Kehrwert der Zeit	0.26	0.14	0.06	0.02

2 Motoren (B und C) ohne Rad				
Power	48.0	24.0	12.0	6.0
Zeit für 1 Umdrehung [sek.]	1.2	2.2	4.7	12.0
Kehrwert der Zeit	0.83	0.45	0.21	0.08
Power	48.0	24.0	12.0	6.0
Zeit für 2 Umdrehung [sek.]	2.2	4.2	9.1	23.2
Kehrwert der Zeit	0.45	0.24	0.11	0.04
Power	48.0	24.0	12.0	6.0
Zeit für 3 Umdrehung [sek.]	3.2	6.3	13.5	35.6
Kehrwert der Zeit	0.31	0.16	0.07	0.03

Anhang C

NBC Sourcecode

```
//   *************************
//   *   LEGO Mindstorms NXT  *
//   *   -------------------  *
//   *   Project:Explorer #3  *
//   *************************
//
// Description: This project's goal is to explore an 'a priori' unknown room. This can be achieved
//              by solving following problems:
//                      - Map building
//                      - Motion planning
//                      - Collision avoidance
//              During the programming process the motor control imposed the biggest problems.
//              An accurate control can only be achieved by using the firmware's built in regulation,
//              that has the big drawback of using a lot of battery power.
//
// Exploration Method: The robot first searches for the nearest wall (FindClosestWall). Then it follows
//              that wall, keeping it always on the left side (DoWallSurf). The robot takes a 'look'
//              around (ScanWall) before doing any movements. When the room is considered as explored the
//              results are saved to the (TGA graphics formatted) file map.tga stored in the NXT's internal
//              flash memory. There are several other threads called from the ones named above.
//
// Notes:       - 1615 rotations on motor B equeals 360 turn to the left
//              - 1575 rotations on motor C equeals 360 turn to the right
//
// -------------------------------------------------------------------------------------------------------
// File:        explorer#3.nbc
// Compiler:    nbc-0.1.3.b2            http://bricxcc.sourceforge.net/nbc/
// Authors:     Claudia Frischknecht    [fclaudia@ee.ethz.ch]
//              Thomas Other            [tother@ee.ethz.ch]
// -------------------------------------------------------------------------------------------------------

// preprocessor instructions not yet implemented by this early nbc compiler version
//#include "NXTDefs.h"



dseg segment
// ..........................................................................................
//
//  ..............
//  .  Type Defs  .
//  ..............
//
// ..........................................................................................

    TCommLSWrite struct
        Result      sbyte
        Port        byte
        Buffer      byte[]
        ReturnLen   byte
    TCommLSWrite ends

    TCommLSRead struct
        Result      sbyte
        Port        byte
        Buffer      byte[]
        BufferLen   byte
    TCommLSRead ends
```

```
TSetScreenMode struct
    Result sbyte
    ScreenMode dword
TSetScreenMode ends

TLocation struct
    X sword
    Y sword
TLocation ends

TDrawText struct
    Result sbyte
    Location TLocation
    Text byte[]
    Options dword
TDrawText ends

TDrawPoint  struct
     Result sbyte
     Location   TLocation
     Options    dword
TDrawPoint  ends

TDrawLine   struct
     Result sbyte
     StartLoc   TLocation
     EndLoc TLocation
     Options    dword
TDrawLine   ends

TDrawCircle struct
    Result sbyte
    Center TLocation
    Size   byte
    Options    dword
TDrawCircle ends

TSize   struct
    Width  sword
    Height sword
TSize   ends

TDrawRect   struct
    Result sbyte
    Location   TLocation
    Size   TSize
    Options    dword
TDrawRect   ends

TFileOpen   struct
    Result word
    FileHandle byte
    Filename   byte[]
    Length dword
TFileOpen   ends

TFileReadWrite struct
    Result    word
    FileHandle    byte
    Buffer    byte[]
    Length    dword
TFileReadWrite ends

TFileClose  struct
    Result word
    FileHandle byte
TFileClose  ends

TFileDelete struct
    Result word
    Filename   byte[]
TFileDelete ends

SoundPF_def struct
result sbyte
filename byte[]
loop byte
vol byte
SoundPF_def ends

SoundGS_def struct
result byte
flags byte
SoundGS_def ends
```

```
// ..............................................................................................
//
//  ..................
//  .  Global Vars  .
//  ..................
//
// ..............................................................................................

    // declaration of structs
    // ----------------------
    lswArgs             TCommLSWrite            // lowSpeed Protocol
    lsrArgs             TCommLSRead

    ScrMode             TSetScreenMode          // Text Control
    TextCtl             TDrawText

    dpArgs              TDrawPoint              // Grapics Control
    dlArgs              TDrawLine
    dcArgs              TDrawCircle
    drArgs              TDrawRect

    FOW_A               TFileOpen               // File I/O Control
    FW_A                TFileReadWrite
    FC_A                TFileClose
    FD_A                TFileDelete

    PF_A                SoundPF_def             // Sound Control
    SGS                 SoundGS_def



    // constants
    // ---------
    cos sdword[]    0.0000,    1.7452,    3.4899,    5.2336,    6.9756,    8.7156,   10.4528,   12.1869,
    13.9173,   15.6434,   17.3648,   19.0809,   20.7912,   22.4951,   24.1922,   25.8819,   27.5637,
    29.2372,   30.9017,   32.5568,   34.2020,   35.8368,   37.4607,   39.0731,   40.6737,   42.2618,
    43.8371,   45.3990,   46.9472,   48.4810,   50.0000,   51.5038,   52.9919,   54.4639,   55.9193,
    57.3576,   58.7785,   60.1815,   61.5661,   62.9320,   64.2788,   65.6059,   66.9131,   68.1998,
    69.4658,   70.7107,   71.9340,   73.1354,   74.3145,   75.4710,   76.6044,   77.7146,   78.8011,
    79.8636,   80.9017,   81.9152,   82.9038,   83.8671,   84.8048,   85.7167,   86.6025,   87.4620,
    88.2948,   89.1007,   89.8794,   90.6308,   91.3545,   92.0505,   92.7184,   93.3580,   93.9693,
    94.5519,   95.1057,   95.6305,   96.1262,   96.5926,   97.0296,   97.4370,   97.8148,   98.1627,
    98.4808,   98.7688,   99.0268,   99.2546,   99.4522,   99.6195,   99.7564,   99.8630,   99.9391,
    99.9848,  100.0000,   99.9848,   99.9391,   99.8630,   99.7564,   99.6195,   99.4522,   99.2546,
    99.0268,   98.7688,   98.4808,   98.1627,   97.8148,   97.4370,   97.0296,   96.5926,   96.1262,
    95.6305,   95.1057,   94.5519,   93.9693,   93.3580,   92.7184,   92.0505,   91.3545,   90.6308,
    89.8794,   89.1007,   88.2948,   87.4620,   86.6025,   85.7167,   84.8048,   83.8671,   82.9038,
    81.9152,   80.9017,   79.8636,   78.8011,   77.7146,   76.6044,   75.4710,   74.3145,   73.1354,
    71.9340,   70.7107,   69.4658,   68.1998,   66.9131,   65.6059,   64.2788,   62.9320,   61.5661,
    60.1815,   58.7785,   57.3576,   55.9193,   54.4639,   52.9919,   51.5038,   50.0000,   48.4810,
    46.9472,   45.3990,   43.8371,   42.2618,   40.6737,   39.0731,   37.4607,   35.8368,   34.2020,
    32.5568,   30.9017,   29.2372,   27.5637,   25.8819,   24.1922,   22.4951,   20.7912,   19.0809,
    17.3648,   15.6434,   13.9173,   12.1869,   10.4528,    8.7156,    6.9756,    5.2336,    3.4899,
    1.7452,    0.0000,   -1.7452,   -3.4899,   -5.2336,   -6.9756,   -8.7156,  -10.4528,  -12.1869,
    -13.9173,  -15.6434,  -17.3648,  -19.0809,  -20.7912,  -22.4951,  -24.1922,  -25.8819,  -27.5637,
    -29.2372,  -30.9017,  -32.5568,  -34.2020,  -35.8368,  -37.4607,  -39.0731,  -40.6737,  -42.2618,
    -43.8371,  -45.3990,  -46.9472,  -48.4810,  -50.0000,  -51.5038,  -52.9919,  -54.4639,  -55.9193,
    -57.3576,  -58.7785,  -60.1815,  -61.5661,  -62.9320,  -64.2788,  -65.6059,  -66.9131,  -68.1998,
    -69.4658,  -70.7107,  -71.9340,  -73.1354,  -74.3145,  -75.4710,  -76.6044,  -77.7146,  -78.8011,
    -79.8636,  -80.9017,  -81.9152,  -82.9038,  -83.8671,  -84.8048,  -85.7167,  -86.6025,  -87.4620,
    -88.2948,  -89.1007,  -89.8794,  -90.6308,  -91.3545,  -92.0505,  -92.7184,  -93.3580,  -93.9693,
    -94.5519,  -95.1057,  -95.6305,  -96.1262,  -96.5926,  -97.0296,  -97.4370,  -97.8148,  -98.1627,
    -98.4808,  -98.7688,  -99.0268,  -99.2546,  -99.4522,  -99.6195,  -99.7564,  -99.8630,  -99.9391,
    -99.9848, -100.0000,  -99.9848,  -99.9391,  -99.8630,  -99.7564,  -99.6195,  -99.4522,  -99.2546,
    -99.0268,  -98.7688,  -98.4808,  -98.1627,  -97.8148,  -97.4370,  -97.0296,  -96.5926,  -96.1262,
    -95.6305,  -95.1057,  -94.5519,  -93.9693,  -93.3580,  -92.7184,  -92.0505,  -91.3545,  -90.6308,
    -89.8794,  -89.1007,  -88.2948,  -87.4620,  -86.6025,  -85.7167,  -84.8048,  -83.8671,  -82.9038,
    -81.9152,  -80.9017,  -79.8636,  -78.8011,  -77.7146,  -76.6044,  -75.4710,  -74.3145,  -73.1354,
    -71.9340,  -70.7107,  -69.4658,  -68.1998,  -66.9131,  -65.6059,  -64.2788,  -62.9320,  -61.5661,
    -60.1815,  -58.7785,  -57.3576,  -55.9193,  -54.4639,  -52.9919,  -51.5038,  -50.0000,  -48.4810,
    -46.9472,  -45.3990,  -43.8371,  -42.2618,  -40.6737,  -39.0731,  -37.4607,  -35.8368,  -34.2020,
    -32.5568,  -30.9017,  -29.2372,  -27.5637,  -25.8819,  -24.1922,  -22.4951,  -20.7912,  -19.0809,
    -17.3648,  -15.6434,  -13.9173,  -12.1869,  -10.4528,   -8.7156,   -6.9756,   -5.2336,   -3.4899,
    -1.7452,    0.0000
    sin sdword[]  100.0000,   99.9848,   99.9391,   99.8630,   99.7564,   99.6195,   99.4522,   99.2546,
    99.0268,   98.7688,   98.4808,   98.1627,   97.8148,   97.4370,   97.0296,   96.5926,   96.1262,
    95.6305,   95.1057,   94.5519,   93.9693,   93.3580,   92.7184,   92.0505,   91.3545,   90.6308,
    89.8794,   89.1007,   88.2948,   87.4620,   86.6025,   85.7167,   84.8048,   83.8671,   82.9038,
    81.9152,   80.9017,   79.8636,   78.8011,   77.7146,   76.6044,   75.4710,   74.3145,   73.1354,
```

```
    71.9340,   70.7107,   69.4658,   68.1998,   66.9131,   65.6059,   64.2788,   62.9320,   61.5661,
    60.1815,   58.7785,   57.3576,   55.9193,   54.4639,   52.9919,   51.5038,   50.0000,   48.4810,
    46.9472,   45.3990,   43.8371,   42.2618,   40.6737,   39.0731,   37.4607,   35.8368,   34.2020,
    32.5568,   30.9017,   29.2372,   27.5637,   25.8819,   24.1922,   22.4951,   20.7912,   19.0809,
    17.3648,   15.6434,   13.9173,   12.1869,   10.4528,    8.7156,    6.9756,    5.2336,    3.4899,
    1.7452,    0.0000,   -1.7452,   -3.4899,   -5.2336,   -6.9756,   -8.7156,  -10.4528,  -12.1869,
    -13.9173,  -15.6434,  -17.3648,  -19.0809,  -20.7912,  -22.4951,  -24.1922,  -25.8819,  -27.5637,
    -29.2372,  -30.9017,  -32.5568,  -34.2020,  -35.8368,  -37.4607,  -39.0731,  -40.6737,  -42.2618,
    -43.8371,  -45.3990,  -46.9472,  -48.4810,  -50.0000,  -51.5038,  -52.9919,  -54.4639,  -55.9193,
    -57.3576,  -58.7785,  -60.1815,  -61.5661,  -62.9320,  -64.2788,  -65.6059,  -66.9131,  -68.1998,
    -69.4658,  -70.7107,  -71.9340,  -73.1354,  -74.3145,  -75.4710,  -76.6044,  -77.7146,  -78.8011,
    -79.8636,  -80.9017,  -81.9152,  -82.9038,  -83.8671,  -84.8048,  -85.7167,  -86.6025,  -87.4620,
    -88.2948,  -89.1007,  -89.8794,  -90.6308,  -91.3545,  -92.0505,  -92.7184,  -93.3580,  -93.9693,
    -94.5519,  -95.1057,  -95.6305,  -96.1262,  -96.5926,  -97.0296,  -97.4370,  -97.8148,  -98.1627,
    -98.4808,  -98.7688,  -99.0268,  -99.2546,  -99.4522,  -99.6195,  -99.7564,  -99.8630,  -99.9391,
    -99.9848, -100.0000,  -99.9848,  -99.9391,  -99.8630,  -99.7564,  -99.6195,  -99.4522,  -99.2546,
    -99.0268,  -98.7688,  -98.4808,  -98.1627,  -97.8148,  -97.4370,  -97.0296,  -96.5926,  -96.1262,
    -95.6305,  -95.1057,  -94.5519,  -93.9693,  -93.3580,  -92.7184,  -92.0505,  -91.3545,  -90.6308,
    -89.8794,  -89.1007,  -88.2948,  -87.4620,  -86.6025,  -85.7167,  -84.8048,  -83.8671,  -82.9038,
    -81.9152,  -80.9017,  -79.8636,  -78.8011,  -77.7146,  -76.6044,  -75.4710,  -74.3145,  -73.1354,
    -71.9340,  -70.7107,  -69.4658,  -68.1998,  -66.9131,  -65.6059,  -64.2788,  -62.9320,  -61.5661,
    -60.1815,  -58.7785,  -57.3576,  -55.9193,  -54.4639,  -52.9919,  -51.5038,  -50.0000,  -48.4810,
    -46.9472,  -45.3990,  -43.8371,  -42.2618,  -40.6737,  -39.0731,  -37.4607,  -35.8368,  -34.2020,
    -32.5568,  -30.9017,  -29.2372,  -27.5637,  -25.8819,  -24.1922,  -22.4951,  -20.7912,  -19.0809,
    -17.3648,  -15.6434,  -13.9173,  -12.1869,  -10.4528,   -8.7156,   -6.9756,   -5.2336,   -3.4899,
    -1.7452,    0.0000,    1.7452,    3.4899,    5.2336,    6.9756,    8.7156,   10.4528,   12.1869,
    13.9173,   15.6434,   17.3648,   19.0809,   20.7912,   22.4951,   24.1922,   25.8819,   27.5637,
    29.2372,   30.9017,   32.5568,   34.2020,   35.8368,   37.4607,   39.0731,   40.6737,   42.2618,
    43.8371,   45.3990,   46.9472,   48.4810,   50.0000,   51.5038,   52.9919,   54.4639,   55.9193,
    57.3576,   58.7785,   60.1815,   61.5661,   62.9320,   64.2788,   65.6059,   66.9131,   68.1998,
    69.4658,   70.7107,   71.9340,   73.1354,   74.3145,   75.4710,   76.6044,   77.7146,   78.8011,
    79.8636,   80.9017,   81.9152,   82.9038,   83.8671,   84.8048,   85.7167,   86.6025,   87.4620,
    88.2948,   89.1007,   89.8794,   90.6308,   91.3545,   92.0505,   92.7184,   93.3580,   93.9693,
    94.5519,   95.1057,   95.6305,   96.1262,   96.5926,   97.0296,   97.4370,   97.8148,   98.1627,
    98.4808,   98.7688,   99.0268,   99.2546,   99.4522,   99.6195,   99.7564,   99.8630,   99.9391,
    99.9848,  100.0000


    // explorer specific variables
    // --------------------------
    globalStartingPosX          sword           // global positioning
    globalStartingPosY          sword
    globalStartingPosHeading    sword
    globalPosX                  sword
    globalPosY                  sword
    globalPosHeading            sword
    globalUSSHeading            sword

    globalExploringStarted      byte
    globalExitCond              sword           // exit condition for this programm

    globalMap                   byte[]          // environment mapping
    globalMapSize               dword
    globalMapWidth              sword
    globalMapScale              sword

    globalMotorAngleA           sdword          // motor control
    globalMotorAngleB           sdword
    globalMotorAngleC           sdword


    // global return adresses
    // ----------------------
    retDrawInfo                 sdword
    retDrawMap                  sdword


    // test/debugging purpose
    // ----------------------
    FCW_closestAngle            sword
    FCW_closestDist             byte

    DWS_USLeft                  byte
    DWS_USFront                 byte

    globalText                  byte[]

dseg ends
// ...........................................................................................................


// ***********************************************************************************************
// Thread: Wait
```

```
// ------------------------
// Desrciption: Sleeps for 100ms
//
// Arguments:       none
//
// Return Values:   none
// ***************************************************************************************************
dseg segment

    // define variables used only by this thread
    W_now sdword
    W_then sdword

    // define return address
    retWait sdword
dseg ends
// ...................................................................................................
thread Wait
    gettick W_now
    add W_then, W_now, 50               // 50ms wait

W_waiting:
    gettick W_now
    brcmp   LT, W_waiting, W_now, W_then

  subret retWait
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ***************************************************************************************************
// Thread: DoMotorMotion
// --------------------
// Desrciption: Rotates the desired motors stored in array DMM_motors until the rotation angle of DMM_angle
//              (in degrees) is reached.
//
// Arguments:       DMM_motors      -       array of motors to run
//                  DMM_angle       -       angle to turn the motors (!!! ALWAYS POSITIVE !!!)
//                  DMM_power       -       power settins to run the motors (0 to 100)
//
// Return Values:   none
// ***************************************************************************************************
dseg segment
    // define input arguments for this thread
    DMM_motors          byte[]                  // where DMM_motor is an array of motors (eg. OUT_B, OUT_C)
    DMM_angle           sdword                  // is a POSITIVE angle. rotation until angle is reached
    DMM_power           sbyte

    // define variables used only by this thread
    DMM_motor           byte                    // variable to store the motor from which the angle gets read
    DMM_motorsCount     byte                    // how many motors are there to controll
    DMM_startAngle      sdword
    DMM_endAngle        sdword
    DMM_updateFlags     byte                    // define motor specific variables
    DMM_outputMode      byte    OUT_MODE_MOTORON+OUT_MODE_BRAKE
    DMM_regMode1        byte    OUT_REGMODE_IDLE
    DMM_regMode2        byte    OUT_REGMODE_IDLE
    DMM_runState        byte    OUT_RUNSTATE_RUNNING
    DMM_currentRunState byte
    DMM_currentAngle    sdword                  // variable to store the current Angle readout
    DMM_oldCurrentAngle sdword                  //

    // define return address
    retDoMotorMotion    sdword                  // used to store return address
dseg ends
// ...................................................................................................
thread DoMotorMotion

    // 1. Get number of motors and first motor
    // ---------------------------------------
    index DMM_motor, DMM_motors, 0
    arrsize DMM_motorsCount, DMM_motors


    // 2. Are we moving 1 or many motors?
    // ----------------------------------
    brcmp EQ, DMM_SingleMotor, DMM_motorsCount, 1
    brcmp GT, DMM_MultipleMotors, DMM_motorsCount, 1
DMM_SingleMotor:
    set DMM_regMode1, OUT_REGMODE_IDLE
    set DMM_regMode2, OUT_REGMODE_SPEED
    jmp DMM_InitMotors
DMM_MultipleMotors:
```

```
    set DMM_regMode1, OUT_REGMODE_IDLE
    set DMM_regMode2, OUT_REGMODE_SPEED

    jmp DMM_InitMotors
DMM_InitMotors:
    set DMM_outputMode, OUT_MODE_MOTORON+OUT_MODE_BRAKE
    set DMM_updateFlags, UF_UPDATE_RESET_BLOCK_COUNT+UF_UPDATE_RESET_COUNT+
                         UF_UPDATE_TACHO_LIMIT+UF_UPDATE_SPEED+UF_UPDATE_MODE
    setout DMM_motors, OutputMode, DMM_outputMode, RegMode, DMM_regMode1, TachoLimit, DMM_angle,
                         RunState, DMM_runState, Power, DMM_power, UpdateFlags, DMM_updateFlags


    // 3. Waits till angle is reached
    // ------------------------------
DMM_Running:
    getout DMM_currentRunState, DMM_motor, RunState
    brcmp EQ, DMM_Running, DMM_currentRunState, OUT_RUNSTATE_RUNNING

    // 4. Regulates for speed = 0
    // --------------------------
    set DMM_power, 0
    set DMM_outputMode, OUT_MODE_MOTORON+OUT_MODE_BRAKE+OUT_MODE_REGULATED
    set DMM_updateFlags, UF_UPDATE_SPEED+UF_UPDATE_MODE
    setout DMM_motors, OutputMode, DMM_outputMode, RegMode, DMM_regMode2, RunState, DMM_runState,
                         Power, DMM_power, UpdateFlags, DMM_updateFlags


    // 5. Verifies that motor doesn't rotate for 50ms, else loops
    // ----------------------------------------------------------
DMM_Stabilize:
    getout DMM_currentAngle, DMM_motor, RotationCount
    mov DMM_oldCurrentAngle, DMM_currentAngle
    subcall Wait, retWait
    getout DMM_currentAngle, DMM_motor, RotationCount
    brcmp NEQ, DMM_Stabilize, DMM_oldCurrentAngle, DMM_currentAngle

    subret retDoMotorMotion

endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ***********************************************************************************************************
// Thread: GetUltraSonicSensor
// ---------------------------
// Desrciption: Reads value from ultrasonic sensor and returns it in GUSS_retVal.
//
// Arguments:       GUSS_port       -       stores the port the US sensor is attached to
//
// Return Values:   GUSS_retVal     -       US distance readout
// ***********************************************************************************************************
dseg segment
    // define input arguments for this thread
    GUSS_port           byte

    // define return values
    GUSS_retVal         byte

    // define variables used only by this thread
    GUSS_bufLSWrite1    byte[] 0x2, 0x42
    GUSS_readBuf        byte[]
    GUSS_lsBytesRead    byte

    // define return address
    retGetUltraSonicSensor sdword
dseg ends
// ...........................................................................................................
thread GetUltraSonicSensor
    setin IN_TYPE_LOWSPEED_9V, GUSS_port, IN_MODE_RAW

GUSS_ReadAgain:

    mov lswArgs.Port, GUSS_port
    set lswArgs.ReturnLen, 1
    mov lswArgs.Buffer, GUSS_bufLSWrite1
    syscall CommLSWrite, lswArgs

    mov lsrArgs.Port, GUSS_port
    set lsrArgs.BufferLen, 1
    syscall CommLSRead, lsrArgs

    mov GUSS_readBuf, lsrArgs.Buffer
    arrsize GUSS_lsBytesRead, GUSS_readBuf
```

```
    // have to loop and generate another readout if lsByteRead = 0
    brtst NEQ, GUSS_NoReadAgain, GUSS_lsBytesRead
    subcall Wait, retWait
    jmp GUSS_ReadAgain
GUSS_NoReadAgain:

    index GUSS_retVal, GUSS_readBuf, NA

    subret retGetUltraSonicSensor
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: GetLightSensor
// ---------------------------
// Desrciption: Reades value from light sensor and returns it.
//
// Arguments:       GLS_port       -       stores the port the light sensor is attached to
//
// Return Values:   GLS_retVal     -       light value readout
// ***************************************************************************************************
dseg segment
    // define input arguments for this thread
    GLS_port            byte

    // define return values
    GLS_retVal          word

    // define variables used only by this thread
    GLS_sensorType      byte
    GLS_sensorMode      byte
    GLS_isInvalid       byte

    // define return address
    retGetLightSensor   sdword
dseg ends
// .......................................................................................................
thread GetLightSensor

    // set sensor type and mode

    set GLS_sensorType, IN_TYPE_LIGHT_INACTIVE

    set GLS_sensorMode, IN_MODE_RAW
    setin GLS_sensorType, GLS_port, Type
    setin GLS_sensorMode, GLS_port, InputMode

    /* when changing the type and/or mode of a sensor the NXT blocks also set the InvalidData field to true */
    set GLS_isInvalid, TRUE
    setin GLS_isInvalid, GLS_port, InvalidData

    // make sure the sensor value is valid
GLS_StillInvalid:
        getin   GLS_isInvalid, GLS_port, InvalidData
        brtst   NEQ, GLS_StillInvalid, GLS_isInvalid
    /* if isInvalid not false (equal to zero) then loop until it is */


    getin GLS_retVal, GLS_port, RawValue
    sub GLS_retVal, 1000, GLS_retVal
    div GLS_retVal, GLS_retVal, 10

    brcmp GT, GLS_SetToMax, GLS_retVal, 100
    subret retGetLightSensor

GLS_SetToMax:
    set GLS_retVal, 100
    subret retGetLightSensor
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: GetSoundSensor
// ---------------------------
// Desrciption: Reades value from sound sensor and returns it in GSS_retVal.
//
// Arguments:       GSS_port       -       stores the port the sound sensor is attached to
//
// Return Values:   GSS_retVal     -       sound value readout
// ***************************************************************************************************
dseg segment
```

```
    // define input arguments for this thread
    GSS_port            byte

    // define return values
    GSS_retVal          word

    // define variables used only by this thread
    GSS_sensorType      byte
    GSS_sensorMode      byte
    GSS_isInvalid       byte

    // define return address
    retGetSoundSensor   sdword
dseg ends
// ------------------------------------------------------------------------------------------
thread GetSoundSensor

    // set sensor type and mode
    set GSS_sensorType, IN_TYPE_SOUND_DB
    set GSS_sensorMode, IN_MODE_RAW


    setin GSS_sensorType, GSS_port, Type
    setin GSS_sensorMode, GSS_port, InputMode

    /* when changing the type and/or mode of a sensor the NXT blocks also set the InvalidData field to true */
    set GSS_isInvalid, TRUE
    setin GSS_isInvalid, GSS_port, InvalidData

    // make sure the sensor value is valid
GSS_StillInvalid:
        getin   GSS_isInvalid, GSS_port, InvalidData
        brtst   NEQ, GSS_StillInvalid, GSS_isInvalid
    /* if isInvalid not false (equal to zero) then loop until it is */

    getin GSS_retVal, GSS_port, NormalizedValue
    div GSS_retVal, GSS_retVal, 10

    brcmp GT, GSS_SetToMax, GSS_retVal, 100
    subret retGetSoundSensor

GSS_SetToMax:
    set GSS_retVal, 100
    subret retGetSoundSensor
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ******************************************************************************************
// Thread: GetTouchSensor
// ---------------------------
// Desrciption: Reads value from touch sensor and returns it in GTS_retVal.
//
// Arguments:       GTS_port       -       stores the port the touch sensor is attached to
//
// Return Values:   GTS_retVal     -       touch sensor readout
// ******************************************************************************************
dseg segment
    // define input arguments for this thread
    GTS_port            byte

    // define return values
    GTS_retVal          word

    // define variables used only by this thread
    GTS_sensorType      byte
    GTS_sensorMode      byte
    GTS_isInvalid       byte

    // define return address
    retGetTouchSensor   sdword
dseg ends
// ------------------------------------------------------------------------------------------
thread GetTouchSensor
    set GTS_port, IN_3
    set GTS_sensorType, IN_TYPE_SWITCH
    set GTS_sensorMode, IN_MODE_BOOLEAN
    setin GTS_sensorType, GTS_port, InputMode

    set GTS_isInvalid, TRUE
    setin GTS_isInvalid, GTS_port, InvalidData
GTS_StillInvalid:
```

```
        getin GTS_isInvalid, GTS_port, InvalidData
        brtst NEQ, GTS_StillInvalid, GTS_isInvalid
    set GTS_retVal, 0
    getin GTS_retVal, GTS_port, ScaledValue
    cmp LT, GTS_retVal, GTS_retVal, 500

    subret retGetTouchSensor
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: CalcNewPos
// ------------------------
// Desrciption: Calculate the current position depenging on motor rotations and current heading.
//
// Arguments:       CNP_dMovement       - current displacement
//
// Return Values:   none
// ***************************************************************************************************
dseg segment

    // define input arguments for this thread
    CNP_dMovement       sword       // in milimeters

    // define variables used only by this thread
    CNP_angleItem       uword
    CNP_tmp             udword
    CNP_dx              sword       // in milimeters
    CNP_dy              sword

    // define return address
    retCalcNewPos       sdword
dseg ends
// ...........................................................................................................
thread CalcNewPos

    // 1. add or subtract 360 degrees of
    //    the current heading if needed
    // ---------------------------------
    brcmp GTEQ,CNP_DoHeadingSub, globalPosHeading, 360
    brcmp LT,CNP_DoHeadingAdd, globalPosHeading, 0
    jmp CNP_EndChange
CNP_DoHeadingSub:
    sub globalPosHeading, globalPosHeading, 360
    jmp CNP_EndChange
CNP_DoHeadingAdd:
    add globalPosHeading, globalPosHeading, 360
    jmp CNP_EndChange
CNP_EndChange:
    brcmp GTEQ,CNP_DoHeadingSub, globalPosHeading, 360
    brcmp LT,CNP_DoHeadingAdd, globalPosHeading, 0


    // 2. Get cos and sin values for globalPosHeading
    // ----------------------------------------------
    mov CNP_angleItem, globalPosHeading
    index CNP_dx, cos, CNP_angleItem
    index CNP_dy, sin, CNP_angleItem
    mul CNP_dx, CNP_dx, CNP_dMovement                   // multiply with displacement
    mul CNP_dy, CNP_dy, CNP_dMovement
    div CNP_dx, CNP_dx, 100
    div CNP_dy, CNP_dy, 100


    // 3. Calc new position
    // ------------------------
    add globalPosX, globalPosX, CNP_dx
    add globalPosY, globalPosY, CNP_dy

    subret retCalcNewPos
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: StoreUSSReadout
// ------------------------
// Desrciption: Stores the current US sensor readout to the map with respect to the current US sensor
//              direction. Increase value of map point if we already saw an obstacle here.
//
// Arguments:       SUR_ussReadout
//
```

```
// Return Values:   none
// ***********************************************************************************************************
dseg segment

    // define input arguments for this thread
    SUR_ussReadout  byte

    // define variables used only by this thread
    SUR_middlePoint udword

    SUR_posX            sword
    SUR_posY            sword
    SUR_angle           sword
    SUR_angleItem       uword
    SUR_cos             sword
    SUR_sin             sword
    SUR_mapItem         udword
    SUR_mapValue        byte
    SUR_startValue      byte 14         // start us recognition count at value 14 and increase each
    SUR_stopValue       byte 22         // time we see an object at the same place

    // define return address
    retStoreUSSReadout  sdword
dseg ends
// .......................................................................................................
thread StoreUSSReadout

    brcmp GT, SUR_SkipReadout, SUR_ussReadout, 47           // exit thread if the readout is above specified
                                                            // value


    // 1. Calc coordinates relative to current position
    //    and middlepoint of Map array
    // ------------------------------------------------
    div SUR_middlePoint, globalMapWidth, 2
    div SUR_posX, globalPosX, globalMapScale                // respect map scale
    div SUR_posY, globalPosY, globalMapScale
    add SUR_posX, SUR_posX, SUR_middlePoint                 // Place position relative to Middlepoint of Map
    add SUR_posY, SUR_posY, SUR_middlePoint


    // 2. Determine relative Angle to US sensor
    //    and set in between 0 and 359 degrees
    // ----------------------------------------
    add SUR_angle, globalPosHeading, globalUSSHeading
    brcmp LT, SUR_DoHeadingAdd, SUR_angle, 0
    brcmp GTEQ, SUR_DoHeadingSub, SUR_angle, 360
    jmp SUR_EndChangeHeading
SUR_DoHeadingAdd:
    add SUR_angle, SUR_angle, 360
    jmp SUR_EndChangeHeading
SUR_DoHeadingSub:
    sub SUR_angle, SUR_angle, 360
    jmp SUR_EndChangeHeading
SUR_EndChangeHeading:
    brcmp LT, SUR_DoHeadingAdd, SUR_angle, 0
    brcmp GTEQ, SUR_DoHeadingSub, SUR_angle, 360


    // 3. Get cos and sin values for SUR_angle
    //    and multiply with us readout
    // -----------------------------------------------
    mov SUR_angleItem, SUR_angle
    index SUR_cos, cos, SUR_angleItem
    index SUR_sin, sin, SUR_angleItem
    mul SUR_cos,SUR_cos, SUR_ussReadout
    mul SUR_sin,SUR_sin, SUR_ussReadout
    mul SUR_cos,SUR_cos, 10                             // transform ultrasonic cm to milimeter
    mul SUR_sin,SUR_sin, 10
    div SUR_cos,SUR_cos, 100                            // 0 <= |cos(phi)| < 100
    div SUR_sin,SUR_sin, 100                            // 0 <= |sin(phi)| < 100

    // 4. Calc coordinates of current us readout
    // -----------------------------------------
    div SUR_cos,SUR_cos, globalMapScale
    div SUR_sin,SUR_sin, globalMapScale
    add SUR_posX, SUR_posX, SUR_cos
    add SUR_posY, SUR_posY, SUR_sin

    // 5. Calculate 1-dimensional array adress
    //    and increase value of array item by 1
    // ---------------------------------------
    mul SUR_mapItem, SUR_posY, globalMapWidth
    add SUR_mapItem, SUR_mapItem, SUR_posX
```

```
    index SUR_mapValue, globalMap, SUR_mapItem
    brcmp EQ, SUR_SkipReadout, SUR_mapValue,  1
    brcmp GT, SUR_DontAdOffset, SUR_mapValue, SUR_startValue
    brcmp GTEQ, SUR_DontIncrease, SUR_mapValue, SUR_stopValue
    mov SUR_mapValue, SUR_startValue
SUR_DontAdOffset:
    add SUR_mapValue, SUR_mapValue, 1
SUR_DontIncrease:
    brcmp GT, SUR_SkipReadout, SUR_posX, globalMapWidth             // don't write if we're out of bounds
    brcmp GT, SUR_SkipReadout, SUR_posY, globalMapWidth
    brcmp LT, SUR_SkipReadout, SUR_posX, 0
    brcmp LT, SUR_SkipReadout, SUR_posY, 0
    replace globalMap, globalMap, SUR_mapItem, SUR_mapValue        // value 2 stands for scanned wall

SUR_SkipReadout:

    subret retStoreUSSReadout
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: StoreEnvReadout
// ------------------------
// Desrciption: Read the light and sound sensor values and calculates an overall quality. This value get
//              stored to the map along with the current position.
//
// Arguments:       none
//
// Return Values:   none
// ***************************************************************************************************
dseg segment

    // define input arguments for this thread
    SER_ussReadout  byte

    // define variables used only by this thread
    SER_envValue    byte
    SER_middlePoint udword
    SER_posX        sword
    SER_posY        sword
    SER_angle       sword
    SER_angleItem   uword
    SER_rectAngle   sword
    SER_cos         sword
    SER_sin         sword
    SER_posX1       sword
    SER_posY1       sword
    SER_posX2       sword
    SER_posY2       sword
    SER_mapItem     udword
    SER_mapValue    byte
    SER_mapItem1    udword
    SER_mapItem2    udword


    // define return address
    retStoreEnvReadout sdword
dseg ends
// ...........................................................................................................
thread StoreEnvReadout

    // 1. Determine environmental properties
    //    and transform to number
    // -------------------------------------
    subcall GetLightSensor, retGetLightSensor           // get sensor readouts
    subcall GetSoundSensor, retGetSoundSensor
    set SER_envValue, 100                               // calculate environment specific value
    add SER_envValue, SER_envValue, GLS_retVal
    sub SER_envValue, SER_envValue, GSS_retVal
    div SER_envValue, SER_envValue, 20                  // to get values between 1 and 14
    add SER_envValue, SER_envValue, 3                   // color offset starting at position 3


    // 2. Calc coordinates based on current position
    //    and middlepoint of Map array
    // ---------------------------------------------
    div SER_middlePoint, globalMapWidth, 2
    div SER_posX, globalPosX, globalMapScale            // respect map scale
    div SER_posY, globalPosY, globalMapScale
    add SER_posX, SER_posX, SER_middlePoint             // Place position relative to Middlepoint of Map
    add SER_posY, SER_posY, SER_middlePoint
```

```
    // 3. Determine angle towards righthandside of
    //    current heading
    // -------------------------------------------
    add SER_angle, globalPosHeading, 90
    add SER_angle, SER_angle, 45                        // if we rotate the angle by 45 degrees we wont get
                                                        // rounded down when e.g. SER_angle = 85
    brcmp LT, SER_DoHeadingAdd, SER_angle, 0            // transform angle to   0 <= |angle| < 359
    brcmp GTEQ, SER_DoHeadingSub, SER_angle, 360
    jmp SER_EndChangeHeading
SER_DoHeadingAdd:
    add SER_angle, SER_angle, 360
    jmp SER_EndChangeHeading
SER_DoHeadingSub:
    sub SER_angle, SER_angle, 360
    jmp SER_EndChangeHeading
SER_EndChangeHeading:

    // 4. Get cos and sin values for SER_angle
    // ---------------------------------------------
    div SER_rectAngle, SER_angle, 90                    // rectify angle (0|90|180|270)
    mul SER_rectAngle, SER_rectAngle, 90

    mov SER_angleItem, SER_rectAngle
    index SER_cos, cos, SER_angleItem
    index SER_sin, sin, SER_angleItem
    div SER_cos,SER_cos, 100                            // 0 <= |cos(phi)| < 100
    div SER_sin,SER_sin, 100                            // 0 <= |sin(phi)| < 100

    // 5. Store the envValue  to the lefthand
    //    and righthand side of current position
    // -----------------------------------------
    add SER_posX1, SER_posX, SER_cos                            // right hand side
    add SER_posY1, SER_posY, SER_sin
    mul SER_mapItem1, SER_posY1, globalMapWidth                 // Calculate 1-dimensional adress
    add SER_mapItem1, SER_mapItem1, SER_posX1
    index SER_mapValue, globalMap, SER_mapItem1
    brcmp EQ, SER_SkipStoreRight, SER_mapValue, 1               // don't write if we would overwrite a
                                                                // position entry
    brcmp GT, SER_SkipStoreRight, SER_posX1, globalMapWidth     // don't write if we're out of bounds
    brcmp GT, SER_SkipStoreRight, SER_posY1, globalMapWidth
    brcmp LT, SER_SkipStoreRight, SER_posX1, 0
    brcmp LT, SER_SkipStoreRight, SER_posY1, 0
    replace globalMap, globalMap, SER_mapItem1, SER_envValue    // Alter Value of Array item
SER_SkipStoreRight:
    sub SER_posX2, SER_posX, SER_cos                            // left hand side
    sub SER_posY2, SER_posY, SER_sin
    mul SER_mapItem2, SER_posY2, globalMapWidth
    add SER_mapItem2, SER_mapItem2, SER_posX2
    index SER_mapValue, globalMap, SER_mapItem1
    brcmp EQ, SER_SkipStoreLeft, SER_mapValue, 1                // don't write if we would overwrite
                                                                // a position entry
    brcmp GT, SER_SkipStoreLeft, SER_posX2, globalMapWidth      // don't write if we're out of bounds
    brcmp GT, SER_SkipStoreLeft, SER_posY2, globalMapWidth
    brcmp LT, SER_SkipStoreLeft, SER_posX2, 0
    brcmp LT, SER_SkipStoreLeft, SER_posY2, 0
    replace globalMap, globalMap, SER_mapItem2, SER_envValue
SER_SkipStoreLeft:

    // 7. Write current position
    // -------------------------
    mul SER_mapItem, SER_posY, globalMapWidth                   // Calculate 1-dimensional adress
    add SER_mapItem, SER_mapItem, SER_posX
    brcmp GT, SER_SkipStorePos, SER_posX, globalMapWidth        // don't write if we're out of bounds
    brcmp GT, SER_SkipStorePos, SER_posY, globalMapWidth
    brcmp LT, SER_SkipStorePos, SER_posX, 0
    brcmp LT, SER_SkipStorePos, SER_posY, 0
    replace globalMap, globalMap, SER_mapItem, 1
SER_SkipStorePos:

    subret retStoreEnvReadout

endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***********************************************************************************************************
// Thread: DrawMap
// ------------------------
// Desrciption: Draws the globalPosMap on the Display
//
// Arguments:       none
//
// Return Values:   none
```

```
// ********************************************************************************************************
dseg segment

    // define input arguments for this thread
    DM_map       byte[]

    // define variables used only by this thread
    DM_x        byte
    DM_y        byte
    DM_i        udword
    DM_val      byte
    DM_tmp      byte

    // define return address
//    retDrawMap sdword         // defined above for debugging purpose
dseg ends
// ....................................................................................................
thread DrawMap

    // display the map on the screen
    // ----------------------------
    // ----------------------------
    //draw bounding frame
    mov DM_x,globalMapWidth
    mov DM_y,globalMapWidth

    //draw bounding frame
    set drArgs.Location.X, 1
    set drArgs.Location.Y, 1
    mov drArgs.Size.Width, DM_x
    mov drArgs.Size.Height, DM_y
    set drArgs.Options, 1 ; clear the screen
    syscall DrawRect, drArgs ; call the API

    //draw middlepoint
    set dcArgs.Center.X, 45
    set dcArgs.Center.Y, 45
    set dcArgs.Size, 1
    syscall DrawCircle, dcArgs ; call the API


    //then fill the map
    set DM_i, 0
DM_FillMap:

    // get point value
    index DM_val, globalMap, DM_i
    brtst EQ, DM_SkipMapPoint, DM_val

    //calc 2dim coordinates
    // y coord
    div DM_y, DM_i, globalMapWidth

    // x coord
    mul DM_tmp, DM_y, globalMapWidth
    sub DM_x, DM_i, DM_tmp

    // draw the point
    mov dpArgs.Location.X, DM_x
    mov dpArgs.Location.Y, DM_y
    set dpArgs.Options, 0 ; don't erase screen before drawing
    syscall DrawPoint, dpArgs ; call the API

DM_SkipMapPoint:

    add DM_i,DM_i,1
    brcmp LT, DM_FillMap, DM_i, globalMapSize

    subret retDrawMap

endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ********************************************************************************************************
// Thread: SaveMapTGA
// ----------------------
// Desrciption: Saves the maps in TGA format picture file. The TGA specifications are available from
//              http://www.martinreddy.net/gfx/2d/Targa.ps
//
// Arguments:       none
//
// Return Values:   none
```

```
// ***********************************************************************************************************
dseg segment
    // define variables used only by this thread
    SMT_fh              byte
    SMT_fileSize     dword
    SMT_result       word

    SMT_fileName byte[] 'map.tga'
    SMT_fileBuffer byte[]


    TGA_def struct                    // on the basis of the targa specifications
                                      // [http://www.martinreddy.net/gfx/2d/Targa.ps]

        // TGA File Header
        IDLength        byte[]      0
        Colormap        byte[]      1
        ImageType       byte[]      1
        ColorMapSpec    byte[]      0,0,23,0,24             // index start, # map entries, entry size
        ImageXOrigin    byte[]      0,0
        ImageYOrigin    byte[]      0,0
        ImageWidth      byte[]      90,0
        ImageHeight     byte[]      90,0
        PixelDepth      byte[]      8                       //bits per pixel
        ImageDescriptor byte[]      0                       //image starts bottom left

        // Image/Colormap Data      blue, green, red
        ImageId         byte[]      0
        ColormapData0   byte[]      255,255,255             // white
        ColormapData1   byte[]      255,0,0                 // blue
        ColormapData2   byte[]      0,0,0                   // unused
        ColormapData3   byte[]      0,0,255                 // red
        ColormapData4   byte[]      100,100,255             // ...
        ColormapData5   byte[]      150,150,255             // ...
        ColormapData6   byte[]      100,255,255             // ...
        ColormapData7   byte[]      0,255,255               // yellow
        ColormapData8   byte[]      0,255,200               // ...
        ColormapData9   byte[]      0,255,100               // ...
        ColormapData10  byte[]      0,255,0                 // ...
        ColormapData11  byte[]      0,150,0                 // ...
        ColormapData12  byte[]      0,80,0                  // green
        ColormapData13  byte[]      0,0,0                   // unused
        ColormapData14  byte[]      0,0,0                   // unused
        ColormapData15  byte[]      25,204,255              // brown light
        ColormapData16  byte[]      0,194,250               // ...
        ColormapData17  byte[]      0,171,220               // ...
        ColormapData18  byte[]      0,148,190               // ...
        ColormapData19  byte[]      0,124,160               // ...
        ColormapData20  byte[]      0,101,130               // ...
        ColormapData21  byte[]      0,78,100                // ...
        ColormapData22  byte[]      0,54,70                 // brown dark

        ImageData       byte[]                              //store the array containing the image here
    TGA_def ends


    SMT_TGA TGA_def                                         // MAX. TOTAL OF 9087 ITEMS PER ARRAY

    SMT_zeroString      byte[]      0

    // define return address
    retSaveMapTGA  sdword                                  // used to store return address
dseg ends
// ...........................................................................................................
thread SaveMapTGA

    // 1. Set Image Properties
    // -----------------------
    arrbuild SMT_TGA.ImageWidth, globalMapWidth, 0
    arrbuild SMT_TGA.ImageHeight, globalMapWidth, 0


    // 2. build TGA file format and determine file size
    // ------------------------------------------------
    arrbuild SMT_fileBuffer, SMT_TGA.IDLength, SMT_TGA.Colormap, SMT_TGA.ImageType, SMT_TGA.ColorMapSpec,
    SMT_TGA.ImageXOrigin, SMT_TGA.ImageYOrigin, SMT_TGA.ImageWidth, SMT_TGA.ImageHeight, SMT_TGA.PixelDepth,
    SMT_TGA.ImageDescriptor, SMT_TGA.ColormapData0, SMT_TGA.ColormapData1, SMT_TGA.ColormapData2,
    SMT_TGA.ColormapData3, SMT_TGA.ColormapData4, SMT_TGA.ColormapData5, SMT_TGA.ColormapData6,
    SMT_TGA.ColormapData7, SMT_TGA.ColormapData8, SMT_TGA.ColormapData9, SMT_TGA.ColormapData10,
    SMT_TGA.ColormapData11, SMT_TGA.ColormapData12,SMT_TGA.ColormapData13, SMT_TGA.ColormapData14,
    SMT_TGA.ColormapData15, SMT_TGA.ColormapData16, SMT_TGA.ColormapData17, SMT_TGA.ColormapData18,
    SMT_TGA.ColormapData19, SMT_TGA.ColormapData20, SMT_TGA.ColormapData21, SMT_TGA.ColormapData22,
    globalMap, SMT_zeroString
    arrsize SMT_fileSize, SMT_fileBuffer
```

```
    sub SMT_fileSize, SMT_fileSize, 1                  // subtract the the zero string terminator


    // 3. output TGA to file
    // ---------------------
    // 3a. First delete the file
    // -------------------------
    mov FD_A.Filename, SMT_fileName ; filename to be deleted
    syscall FileDelete, FD_A ; call the API
    mov SMT_result, FD_A.Result
    // 3b. Open file for writing
    // -------------------------
    mov FOW_A.Length, SMT_fileSize
    mov FOW_A.Filename, SMT_fileName
    syscall FileOpenWrite, FOW_A
    mov SMT_fh, FOW_A.FileHandle
    or SMT_result, SMT_result, FOW_A.Result
    // 3c. Write to the file
    // ---------------------
    mov FW_A.FileHandle, SMT_fh ; file handle obtained via FileOpenWrite
    mov FW_A.Buffer, SMT_fileBuffer ; copy data into write Buffer
    mov FW_A.Length, SMT_fileSize
    syscall FileWrite, FW_A ; call the API
    or SMT_result, SMT_result, FW_A.Result
    // 3d. Close the file
    // ------------------
    mov FC_A.FileHandle, SMT_fh ; file handle obtained via FileOpen*
    syscall FileClose, FC_A ; call the API
    or SMT_result, SMT_result, FC_A.Result

    subret retSaveMapTGA
endt
//:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ***********************************************************************************************************
// Thread: FindClosestWall
// ------------------------
// Desrciption: Turn head 360 degrees, detect closest wall, go to a good starting position next to this wall
//
// Arguments:       none
//
// Return Values:   none
// ***********************************************************************************************************
dseg segment

    // define variables used only by this thread
    FCW_i               byte
    FCW_tmp             sword

//    FCW_closestDist     byte                 // these are defined in the global section, so every thread
//    FCW_closestAngle    sword                // can access it

    // define return address
    retFindClosestWall  sdword                 // used to store return address
dseg ends
// ..........................................................................................................
thread FindClosestWall

    set FCW_closestDist, 255


    // 1. Turn head 180 to the left
    // ----------------------------
    set DMM_angle, 940                                  // 180*5.2222
    arrbuild DMM_motors, OUT_A
    set DMM_power, 80
    subcall DoMotorMotion, retDoMotorMotion

    // 2. Get distance to Object behind us
    // ----------------------------------
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    subcall GetUltraSonicSensor, retGetUltraSonicSensor         // have to read twice to be sure we don't get
                                                                // an obsolete value
    mov FCW_closestDist, GUSS_retVal
    set FCW_closestAngle, -180


    // 3. Loop 3 times while roating the head 90 to the right
    //    measure the distance to find any visible objects
    // -----------------------------------------------------
    // for i=1 to 3
    // {
```

```
    set FCW_i, 1
FCW_turn360:
    // 3a. Rotate head 90 to the right
    // -------------------------------
    set DMM_angle, 470                                      // 90*5.2222
    arrbuild DMM_motors, OUT_A
    set DMM_power, -80
    subcall DoMotorMotion, retDoMotorMotion
    // 3b. Get Distance
    // ---------------
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    subcall GetUltraSonicSensor, retGetUltraSonicSensor     // have to read twice to be sure we don't
                                                            // get an obsolete value
    // 3c. If we got a closer dinstance than stored in FCW_closestDist
    //    we'll overwrite with the current distance and store the current
    //    angle to FCW_closestAngle
    // ------------------------------------------------------------------
    brcmp GT, FCW_biggerValue, GUSS_retVal, FCW_closestDist
    mov FCW_closestDist, GUSS_retVal
    set FCW_closestAngle, -180
    mul FCW_tmp, FCW_i, 90
    add FCW_closestAngle, FCW_closestAngle, FCW_tmp
FCW_biggerValue:
    // 3d. Loop if necessary
    // --------------------
    add FCW_i,FCW_i,1
    brcmp LTEQ, FCW_turn360, FCW_i,3
    // }


    // 4. Turn head to forward looking position
    // ----------------------------------------
    getout DMM_angle, OUT_A, RotationCount
    set DMM_power, -80
    brcmp GT, DWS_LookForward, DMM_angle, 0
    neg DMM_angle, DMM_angle
    neg DMM_power, DMM_power
DWS_LookForward:
    subcall DoMotorMotion, retDoMotorMotion


    // 5. Determine wheter to turn left or right so we face the
    //    closest wall found (stored in FCW_closestDist and FCW_closestAngle) and do so
    // ------------------------------------------------------------------------------
    div FCW_tmp, FCW_closestAngle,90
    mul FCW_tmp, FCW_tmp, 400                               // 400 motor rotation is equal to a 90
    brtst LT, FCW_turnLeft, FCW_tmp                         // turn of the driving base
    brtst EQ, FCW_goForth, FCW_tmp
    // 5a. Okay we're turning right
    // ----------------------------
    mov DMM_angle, FCW_tmp
    arrbuild DMM_motors, OUT_C
    set DMM_power, 80
    subcall DoMotorMotion, retDoMotorMotion
    jmp FCW_goForth
    // 5b. Have to turn left
    // ---------------------
FCW_turnLeft:
    neg DMM_angle, FCW_tmp
    arrbuild DMM_motors, OUT_B
    set DMM_power, 80
    subcall DoMotorMotion, retDoMotorMotion


    // 6. Drive towards wall until distance of less than 48cm is reached
    // -----------------------------------------------------------------
FCW_goForth:
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    subcall GetUltraSonicSensor, retGetUltraSonicSensor     // have to read twice to be sure we don't
                                                            // get an obsolete value
    set DMM_angle, 54                                       // 2.5cm
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
    subcall DoMotorMotion, retDoMotorMotion
    brcmp GT, FCW_goForth, GUSS_retVal, 47


    // 6. Set gap between wall an ourselves to 18cm
    // --------------------------------------------
    sub FCW_tmp, GUSS_retVal, 18
    mul FCW_tmp, FCW_tmp, 24
    mov DMM_angle, FCW_tmp
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
```

```
    subcall DoMotorMotion, retDoMotorMotion


    // 7. Turn right so the wall will be to our left
    // ---------------------------------------------
    set DMM_angle, 400
    arrbuild DMM_motors, OUT_C
    set DMM_power, 80
    subcall DoMotorMotion, retDoMotorMotion


    subret retFindClosestWall
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


// ***************************************************************************************************
// Thread: DoCourseCorrection
// --------------------------
// Desrciption: Corrects the course if needed. Stores distances to the left wall in a FIFO buffer containing
//              4 items.
//
// Arguments:       none
//
// Return Values:   none
// ***************************************************************************************************
dseg segment

    // define input arguments for this thread
    DCC_currentDistance byte

    // define variables used only by this thread
    DCC_distance1           byte    0
    DCC_distance2           byte    0
    DCC_distance3           byte    0
    DCC_distance4           byte    0

    DCC_test                byte

    DCC_closingIn           byte
    DCC_closingOut          byte

    DCC_tachoB              sdword
    DCC_tachoC              sdword
    DCC_tachoDiv            sdword

    DCC_oldAngle            sdword
    DCC_newAngle            sdword

    DCC_heading             sdword

    // define return address
    retDoCourseCorrection  sdword               // used to store return address
dseg ends
// ...........................................................................................................
thread DoCourseCorrection

    brcmp GT, DCC_SkipCurrent, DCC_currentDistance, 25

    // 1. Update FIFO buffer
    // ---------------------
    mov DCC_distance4, DCC_distance3
    mov DCC_distance3, DCC_distance2
    mov DCC_distance2, DCC_distance1
    mov DCC_distance1, DCC_currentDistance


    // 2. Correct trajectory if we're too nearor too far (while ignoring the FIFO buffer)
    // -----------------------------------------------
    set DCC_closingIn, 3                                    // results in a 10 course change
    set DCC_closingOut, 3
    brcmp LT, DCC_CorrectClosingIn, DCC_currentDistance, 11
    brcmp GT, DCC_CorrectClosingOut, DCC_currentDistance, 21


    // 3. Check the FIFO buffer
    // ------------------------
    // 3a. Check for closing in
    // ------------------------
    set DCC_closingIn, 0
    cmp LT, DCC_test, DCC_distance1, DCC_distance2
    add DCC_closingIn, DCC_closingIn, DCC_test
    cmp LT, DCC_test, DCC_distance2, DCC_distance3
    add DCC_closingIn, DCC_closingIn, DCC_test
```

```
    cmp LT, DCC_test, DCC_distance3, DCC_distance4
    add DCC_closingIn, DCC_closingIn, DCC_test
    // 3b. Check for closing out
    // ------------------------
    set DCC_closingOut, 0
    cmp GT, DCC_test, DCC_distance1, DCC_distance2
    add DCC_closingOut, DCC_closingOut, DCC_test
    cmp GT, DCC_test, DCC_distance2, DCC_distance3
    add DCC_closingOut, DCC_closingOut, DCC_test
    cmp GT, DCC_test, DCC_distance3, DCC_distance4
    add DCC_closingOut, DCC_closingOut, DCC_test


    // 4. compare closing in/out values to threshehold
    //    and correct trajectory if necessary
    // ---------------------------------------------
    brcmp GT, DCC_CorrectClosingIn, DCC_closingIn, 1
    brcmp GT, DCC_CorrectClosingOut, DCC_closingOut, 1
    subret retDoCourseCorrection

    // 4a. React on closing in on a wall
    // --------------------------------
DCC_CorrectClosingIn:
    getout DCC_oldAngle, OUT_C, RotationCount
    arrbuild DMM_motors, OUT_C                             // turning right either 5 or 10
    sub DCC_closingIn, DCC_closingIn, 1
    mul DMM_angle, DCC_closingIn, 22                       // do a course correction on the motor
    set DMM_power, 70
    subcall DoMotorMotion, retDoMotorMotion

    getout DCC_tachoC, OUT_C, TachoCount                   // alter heading if motors run unsyncronised
    mul DCC_tachoDiv, DCC_tachoC, 100
    div DCC_tachoDiv, DCC_tachoDiv, 437
    add globalPosHeading, globalPosHeading, DCC_tachoDiv

    subret retDoCourseCorrection

    // 4b. React on closing out on a wall
    // ---------------------------------
DCC_CorrectClosingOut:
    getout DCC_oldAngle, OUT_B, RotationCount
    arrbuild DMM_motors, OUT_B                              // turning left either 5 or 10
    sub DCC_closingOut, DCC_closingOut, 1
    mul DMM_angle, DCC_closingOut, 22                       // do a course correction on the motor
    set DMM_power, 70
    subcall DoMotorMotion, retDoMotorMotion

    getout DCC_tachoB, OUT_B, TachoCount                    // alter heading if motors run unsyncronised
    mul DCC_tachoDiv, DCC_tachoB, 100
    div DCC_tachoDiv, DCC_tachoDiv, 448
    sub globalPosHeading, globalPosHeading, DCC_tachoDiv

    subret retDoCourseCorrection

DCC_SkipCurrent:
    subret retDoCourseCorrection
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// *********************************************************************************************************
// Thread: ScanWall
// -------------------------
// Desrciption: Scans for walls by rotating the US sensor (attached to port A) while polling the distances.
//
// Arguments:       none
//
// Return Values:   none
// *********************************************************************************************************
dseg segment

    // define input arguments for this thread


    // define variables used only by this thread
    SW_area             uword
    SW_motor            byte
    SW_power            byte

    SW_updateFlags      byte                                // define motor specific variables
    SW_outputMode       byte    OUT_MODE_MOTORON+OUT_MODE_BRAKE
    SW_regMode          byte    OUT_REGMODE_IDLE
    SW_runState         byte    OUT_RUNSTATE_RUNNING
```

```
    SW_currentAngle     sdword
    SW_stopAngle        sdword

    // define return address
    retScanWall  sdword              // used to store return address
dseg ends
// ------------------------------------------------------------------------------------------
thread ScanWall

    // 3. Scan the wall to the left
    // ----------------------------
    set SW_area, 980                                        // 180 degrees turn
    mov SW_motor, OUT_A
    set SW_power, -70


    set SW_updateFlags, UF_UPDATE_RESET_COUNT                  // THIS RESETS THE TACHO COUNT
    setout SW_motor, UpdateFlags, SW_updateFlags
    subcall Wait, retWait

    set SW_regMode, OUT_REGMODE_IDLE
    set SW_outputMode, OUT_MODE_MOTORON+OUT_MODE_BRAKE
    set SW_updateFlags, UF_UPDATE_SPEED+UF_UPDATE_MODE+UF_UPDATE_TACHO_LIMIT
    set SW_runState, OUT_RUNSTATE_RUNNING
    setout SW_motor, OutputMode, SW_outputMode, RegMode, SW_regMode, RunState, SW_runState, UpdateFlags,
    SW_updateFlags, Power, SW_power, TachoLimit, SW_area



SW_TurnOn:
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    getout SW_currentAngle, SW_motor, RotationCount
    mul SW_currentAngle, SW_currentAngle, 100
    div globalUSSHeading, SW_currentAngle, 544
    neg globalUSSHeading, globalUSSHeading
    mov SUR_ussReadout, GUSS_retVal
    subcall StoreUSSReadout, retStoreUSSReadout


    getout SW_currentAngle, SW_motor, RotationCount
    brcmp GT, SW_TurnOn, SW_currentAngle, -490                 // until 90 degrees to the right

    // 4. Power down the motor(s) so movement stalls
    // ---------------------------------------------
    set SW_power, 0                                            // now we've reached the desired angle
    set SW_updateFlags, UF_UPDATE_SPEED+UF_UPDATE_MODE         // stall the motor by setting power to 0
    setout SW_motor, Power, SW_power, UpdateFlags, SW_updateFlags

    subret retScanWall
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ******************************************************************************************************
// Thread: DoCommitDisplacement
// ----------------------------
// Desrciption: Commits a displacement to the positioning system
//
// Arguments:       DCM_dMovement
//
// Return Values:   none
// ******************************************************************************************************
dseg segment

    // define input arguments for this thread
    DCP_dMovement               uword

    // define variables used only by this thread
    DCP_steps                   uword
    DCP_i                       uword
    DCP_residuum                uword

    // define return address
    retDoCommitDisplacement     sdword              // used to store return address
dseg ends
// ------------------------------------------------------------------------------------------
thread DoCommitDisplacement

    div DCP_steps, DCP_dMovement, globalMapScale

    set DCP_i, 0
DCP_Commit:
```

```
    mov CNP_dMovement, globalMapScale               // calculate one pixel displacement depending on Map Scale
    subcall CalcNewPos, retCalcNewPos
    subcall StoreEnvReadout, retStoreEnvReadout

    add DCP_i, DCP_i, 1
    brcmp LT, DCP_Commit, DCP_i, DCP_steps

    mul DCP_residuum, DCP_steps, globalMapScale     // consider remainings of total displacement
    sub DCP_residuum, DCP_dMovement, DCP_residuum
    mov CNP_dMovement, DCP_residuum
    subcall CalcNewPos, retCalcNewPos
    subcall StoreEnvReadout, retStoreEnvReadout

    subret retDoCommitDisplacement
endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ***************************************************************
// Thread: DrawInfo
// ------------------
// displays text stored in some variables

dseg segment

    // define variables used only by this thread
    DI_text byte[]

    DI_textRotA byte[] 'A'
    DI_textRotB byte[] 'B'
    DI_textRotC byte[] 'C'

    DI_textTacho byte[] 'tacho:'
    DI_textRotation byte[] 'rotat:'
    DI_textUSS byte[] ' USS:'
    DI_textFCWD byte[] 'FCWD:'
    DI_textFCWA byte[] 'FCWA:'

    DI_textDWSF byte[] 'DWSF:'
    DI_textDWSL byte[] 'DWSL:'

    DI_textPosX byte[] 'PX:'
    DI_textPosY byte[] 'PY:'
    DI_textHeading byte[] 'HEAD:'

    DI_rotCountA   sdword
    DI_rotCountB   sdword
    DI_rotCountC   sdword

    // define return address
//    retDrawInfo sdword

dseg ends
// ..............................................................
thread DrawInfo
    syscall SetScreenMode, ScrMode



    // rot Info start

    getout DI_rotCountA, OUT_A, RotationCount
    getout DI_rotCountB, OUT_B, RotationCount
    getout DI_rotCountC, OUT_C, RotationCount

    set TextCtl.Options, 1
    set TextCtl.Location.X, 10
    set TextCtl.Location.Y, 30
    mov TextCtl.Text, DI_textRotation
    syscall DrawText, TextCtl

    numtostr DI_text, DI_rotCountA
    set TextCtl.Options, 0
    set TextCtl.Location.X, 10
    set TextCtl.Location.Y, 20
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    numtostr DI_text, DI_rotCountB
    set TextCtl.Location.X, 10
    set TextCtl.Location.Y, 10
    mov TextCtl.Text, DI_text
```

```
syscall DrawText, TextCtl

numtostr DI_text, DI_rotCountC
set TextCtl.Location.X, 10
set TextCtl.Location.Y, 0
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl


set TextCtl.Location.X, 60
set TextCtl.Location.Y, 30
mov TextCtl.Text, DI_textTacho
syscall DrawText, TextCtl

set TextCtl.Location.X, 0
set TextCtl.Location.Y, 20
mov TextCtl.Text, DI_textRotA
syscall DrawText, TextCtl

set TextCtl.Location.X, 0
set TextCtl.Location.Y, 10
mov TextCtl.Text, DI_textRotB
syscall DrawText, TextCtl

set TextCtl.Location.X, 0
set TextCtl.Location.Y, 0
mov TextCtl.Text, DI_textRotC
syscall DrawText, TextCtl

getout DI_rotCountA, OUT_A, TachoCount
getout DI_rotCountB, OUT_B, TachoCount
getout DI_rotCountC, OUT_C, TachoCount


numtostr DI_text,DI_rotCountA
set TextCtl.Location.X, 60
set TextCtl.Location.Y, 20
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl

numtostr DI_text,DI_rotCountB
set TextCtl.Location.X, 60
set TextCtl.Location.Y, 10
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl

numtostr DI_text,DI_rotCountC
set TextCtl.Location.X, 60
set TextCtl.Location.Y, 0
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl

// rot Info end


// US info start

set TextCtl.Location.X, 0
set TextCtl.Location.Y, 40
mov TextCtl.Text, DI_textUSS
syscall DrawText, TextCtl

numtostr DI_text,GUSS_retVal               //USS readout
set TextCtl.Location.X, 30
set TextCtl.Location.Y, 40
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl

// US info end



// FindClosestWall info start

set TextCtl.Location.X, 0
set TextCtl.Location.Y, 50
mov TextCtl.Text, DI_textFCWD
syscall DrawText, TextCtl

numtostr DI_text,FCW_closestDist
set TextCtl.Location.X, 30
set TextCtl.Location.Y, 50
mov TextCtl.Text, DI_text
syscall DrawText, TextCtl
```

```
    set TextCtl.Location.X, 0
    set TextCtl.Location.Y, 60
    mov TextCtl.Text, DI_textFCWA
    syscall DrawText, TextCtl

    numtostr DI_text,FCW_closestAngle
    set TextCtl.Location.X, 30
    set TextCtl.Location.Y, 60
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    // FindClosestWall info end


    // DoWallSurf info start

    set TextCtl.Location.X, 50
    set TextCtl.Location.Y, 50
    mov TextCtl.Text, DI_textDWSF
    syscall DrawText, TextCtl

    numtostr DI_text,DWS_USFront           //WallSurfer Front
    set TextCtl.Location.X, 80
    set TextCtl.Location.Y, 50
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    set TextCtl.Location.X, 50
    set TextCtl.Location.Y, 60
    mov TextCtl.Text, DI_textDWSL
    syscall DrawText, TextCtl

    numtostr DI_text,DWS_USLeft           //WallSurfer Left
    set TextCtl.Location.X, 80
    set TextCtl.Location.Y, 60
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    set TextCtl.Location.X, 0
    set TextCtl.Location.Y, 35
    mov TextCtl.Text, DI_textHeading
    syscall DrawText, TextCtl

    // DoWallSurf info end


    // globalPosition info start

    numtostr DI_text,globalPosHeading           //Heading
    set TextCtl.Location.X, 30
    set TextCtl.Location.Y, 35
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    set TextCtl.Location.X, 50
    set TextCtl.Location.Y, 40
    mov TextCtl.Text, DI_textPosX
    syscall DrawText, TextCtl

    numtostr DI_text,globalPosX           //PosX
    set TextCtl.Location.X, 70
    set TextCtl.Location.Y, 40
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl


    set TextCtl.Location.X, 50
    set TextCtl.Location.Y, 35
    mov TextCtl.Text, DI_textPosY
    syscall DrawText, TextCtl

    numtostr DI_text,globalPosY           //PosY
    set TextCtl.Location.X, 70
    set TextCtl.Location.Y, 35
    mov TextCtl.Text, DI_text
    syscall DrawText, TextCtl

    // globalPosition info end

    subcall Wait, retWait
    subret retDrawInfo
endt
```

```
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::



// ************************************************************************************************************
// Thread: DoWallSurf
// -----------------------
// Desrciption: Explores an unknown space. The currnt algorigthm will always put ourselves next to a wall
//             on our lefthand side. We do as well take care of obstacle detection, motion planning,
//             position displacement/correction and the exit condition for the algorithm.
//
// Arguments:       none
//
// Return Values:   none
// ************************************************************************************************************
dseg segment

    // define variables used only by this thread
    DWS_i               byte
    DWS_tmp             sword

    DWS_divPos          sword
    DWS_divAngle        sword

    DWS_tachoB          sdword
    DWS_tachoC          sdword
    DWS_tachoDiv        sdword

//    DWS_USLeft          byte      // these are defined in the global section, so every thread can access it
//    DWS_USAhead         byte

    DWS_isWallLeft      byte
    DWS_isWallFront     byte
    DWS_logicSum        byte

    DWS_done            byte
    DWS_doneCord        sword
    DWS_doneHead        sword
    DWS_doneCond        byte

    // define return address
    retDoWallSurf       sdword      // used to store return address
dseg ends
// ..........................................................................................................
thread DoWallSurf

    subcall StoreEnvReadout, retStoreEnvReadout

DWS_GoForth:

    // 0. Debugging Functions
    // -------------------
//    subcall DrawInfo, retDrawInfo
    subcall SaveMapTGA, retSaveMapTGA
    subcall DrawMap, retDrawMap

    // 1. Check if we hit something
    // ----------------------------
    subcall GetTouchSensor, retGetTouchSensor
    brtst EQ, DWS_DontBackup, GTS_retVal
    add globalPosHeading, globalPosHeading, 180

    set DMM_angle, 173                                  // 7cm
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, -50
    subcall DoMotorMotion, retDoMotorMotion

    set DCP_dMovement, 72   ; milimeters
    subcall DoCommitDisplacement, retDoCommitDisplacement

    sub globalPosHeading, globalPosHeading, 180
DWS_DontBackup:


    // 2. Turn head 90 to the left
    // ----------------------------
    set DMM_angle, 470                                  // 90*5.22222
    arrbuild DMM_motors, OUT_A
    set DMM_power, 80
    subcall DoMotorMotion, retDoMotorMotion


    // 3. Take a look and store to DWS_USLeft
```

```
    // --------------------------------------
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    subcall GetUltraSonicSensor, retGetUltraSonicSensor        // have to read twice to be sure we don't
                                                               // get an obsolete value
    mov DWS_USLeft, GUSS_retVal
    set globalUSSHeading, -90
    mov SUR_ussReadout, GUSS_retVal
    subcall StoreUSSReadout, retStoreUSSReadout


    // 4. Scan surrounding walls
    // -------------------------
    subcall ScanWall, retScanWall


    // 5. Turn head to forward looking position
    // ----------------------------------------
    getout DMM_angle, OUT_A, RotationCount
    set DMM_power, -80
    brtst EQ, DWS_SkipLookForward, DMM_angle
    brcmp GT, DWS_LookForward, DMM_angle, 0
    neg DMM_angle, DMM_angle
    neg DMM_power, DMM_power
DWS_LookForward:
    subcall DoMotorMotion, retDoMotorMotion
DWS_SkipLookForward:

    // 6. Take a look and store to DWS_USAhead
    // ---------------------------------------
    subcall GetUltraSonicSensor, retGetUltraSonicSensor
    subcall GetUltraSonicSensor, retGetUltraSonicSensor        // have to read twice to be sure we don't
                                                               // get an obsolete value
    mov DWS_USFront, GUSS_retVal
    set globalUSSHeading, 0
    mov SUR_ussReadout, GUSS_retVal
    subcall StoreUSSReadout, retStoreUSSReadout


    // 7. Correct Heading if necessary
    // -------------------------------
    mov DCC_currentDistance, DWS_USLeft
    subcall DoCourseCorrection, retDoCourseCorrection


    // 8. Determine if we have to go on straight
    //    or turn to the left or to the right
    // ----------------------------------------
    cmp LTEQ, DWS_isWallLeft, DWS_USLeft, 25
    cmp LTEQ, DWS_isWallFront, DWS_USFront, 19
    mul DWS_logicSum, DWS_isWallLeft,2
    add DWS_logicSum, DWS_logicSum, DWS_isWallFront
    brcmp EQ, DWS_TurnLeft, DWS_logicSum, 0                    // neither front nor left wall detected
    brcmp EQ, DWS_TurnRight, DWS_logicSum, 1                   // should'nt happen during WallSurfing
                                                               // -> ERROR in ROOMDESIGN, turn right anyways
    brcmp EQ, DWS_TurnRight, DWS_logicSum, 3                   // front and left wall detected
    jmp DWS_NoTurn

DWS_TurnLeft:
    // 8a. Turn Left Procedure consist of several steps
    // ------------------------------------------------
    // go straight
    set DMM_angle, 151                                         // 6.3cm
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
    subcall DoMotorMotion, retDoMotorMotion

    set DCP_dMovement, 63   ; milimeters                       // 151/24 = 63mm
    subcall DoCommitDisplacement, retDoCommitDisplacement

    // turn left
    set DCP_dMovement, 61                                      // part of the left turn
    subcall DoCommitDisplacement, retDoCommitDisplacement


    arrbuild DMM_motors, OUT_B                                 // turning left 90
    set DMM_angle, 405
    set DMM_power, 70
    subcall DoMotorMotion, retDoMotorMotion
    getout DWS_tachoB, OUT_B, TachoCount                       // alter heading
    mul DWS_tachoDiv, DWS_tachoB, 100
    div DWS_tachoDiv, DWS_tachoDiv, 448
    sub globalPosHeading, globalPosHeading, DWS_tachoDiv

    set DCP_dMovement, 50                                      // part of the left turn
```

```
    subcall DoCommitDisplacement, retDoCommitDisplacement

    // go straight
    set DMM_angle, 151                                      // 6.3cm
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
    subcall DoMotorMotion, retDoMotorMotion
    getout DWS_tachoB, OUT_B, TachoCount                    // alter heading if motors run unsyncronised
    getout DWS_tachoC, OUT_C, TachoCount
    sub DWS_tachoDiv, DWS_tachoC, DWS_tachoB
    mul DWS_tachoDiv, DWS_tachoDiv, 100
    div DWS_tachoDiv, DWS_tachoDiv, 443
    add globalPosHeading, globalPosHeading, DWS_tachoDiv


    set DCP_dMovement, 63                                   // 151/24= 63mm
    subcall DoCommitDisplacement, retDoCommitDisplacement

    // go straight
    set DMM_angle, 108                                      // 4.5cm
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
    subcall DoMotorMotion, retDoMotorMotion
    getout DWS_tachoB, OUT_B, TachoCount                    // alter heading if motors run unsyncronised
    getout DWS_tachoC, OUT_C, TachoCount
    sub DWS_tachoDiv, DWS_tachoC, DWS_tachoB
    mul DWS_tachoDiv, DWS_tachoDiv, 100
    div DWS_tachoDiv, DWS_tachoDiv, 443
    add globalPosHeading, globalPosHeading, DWS_tachoDiv


    set DCP_dMovement, 45                                   // 108/24= 45mm
    subcall DoCommitDisplacement, retDoCommitDisplacement

    jmp DWS_CheckDone

DWS_TurnRight:
    // 8b. Turn Right Procedure consist of several steps
    // -------------------------------------------------
    set DCP_dMovement, 61                                   // part of the right turn
    subcall DoCommitDisplacement, retDoCommitDisplacement


    arrbuild DMM_motors, OUT_C
    set DMM_angle, 395                                      // turning right 90
    set DMM_power, 70
    subcall DoMotorMotion, retDoMotorMotion
    getout DWS_tachoC, OUT_C, TachoCount                    // alter heading
    mul DWS_tachoDiv, DWS_tachoC, 100
    div DWS_tachoDiv, DWS_tachoDiv, 437
    add globalPosHeading, globalPosHeading, DWS_tachoDiv

    set DCP_dMovement, 50                                   // part of the right turn
    subcall DoCommitDisplacement, retDoCommitDisplacement

    jmp DWS_CheckDone


    // 8c. No Turn - Determine how far we go
    // -------------------------------------
DWS_NoTurn:
    brcmp GT, DWS_Do72mm, DWS_USFront, 30
    brcmp GTEQ, DWS_Do45mm, DWS_USFront, 20
    brcmp LT, DWS_SkipMotorMotion, DWS_USFront, 19

DWS_Do45mm:
    // Going 45mm
    // --------------
    set DWS_divAngle, 108
    set DCP_dMovement, 45                                   // 108/24= 45mm
    subcall DoCommitDisplacement, retDoCommitDisplacement
    jmp DWS_PrepareMotors
DWS_Do72mm:
    // Going 72mm
    // ---------------
    set DWS_divAngle, 173
    set DCP_dMovement, 72                                   // 173/24= 72mm
    subcall DoCommitDisplacement, retDoCommitDisplacement
    // Setup Motors
    // ----------------
DWS_PrepareMotors:
    mov DMM_angle, DWS_divAngle
    arrbuild DMM_motors, OUT_B, OUT_C
    set DMM_power, 50
```

```
    subcall DoMotorMotion, retDoMotorMotion
    getout DWS_tachoB, OUT_B, TachoCount                        // alter heading if motors run unsyncronised
    getout DWS_tachoC, OUT_C, TachoCount
    sub DWS_tachoDiv, DWS_tachoC, DWS_tachoB
    mul DWS_tachoDiv, DWS_tachoDiv, 100
    div DWS_tachoDiv, DWS_tachoDiv, 443
    add globalPosHeading, globalPosHeading, DWS_tachoDiv

DWS_SkipMotorMotion:


    // 9. Determine if we're finished
    // ------------------------------
DWS_CheckDone:
    sub DWS_doneCord, globalStartingPosX, globalExitCond
    cmp GT,DWS_doneCond, globalPosX, DWS_doneCord
    mov DWS_done, DWS_doneCond

    add DWS_doneCord, globalStartingPosX, globalExitCond    // current position must be within starting area
    cmp LT,DWS_doneCond, globalPosX, DWS_doneCord
    and DWS_done, DWS_done, DWS_doneCond

    sub DWS_doneCord, globalStartingPosY, globalExitCond
    cmp GT,DWS_doneCond, globalPosY, DWS_doneCord
    and DWS_done, DWS_done, DWS_doneCond

    add DWS_doneCord, globalStartingPosY, globalExitCond
    cmp LT,DWS_doneCond, globalPosY, DWS_doneCord
    and DWS_done, DWS_done, DWS_doneCond

    add DWS_doneHead, globalStartingPosHeading, 45          // current heading must be less than +- 45
                                                            // degrees of starting Heading
    cmp LT, DWS_doneCond, globalPosHeading, DWS_doneHead
    and DWS_done, DWS_done, DWS_doneCond

    sub DWS_doneHead, globalStartingPosHeading, 45
    cmp GT, DWS_doneCond, globalPosHeading, DWS_doneHead
    and DWS_done, DWS_done, DWS_doneCond

    brtst NEQ, DWS_PossibleExit, DWS_done
    brtst EQ, DWS_ExploringBegan, globalExploringStarted    // once we're outside the starting area we'll
                                                            // set globalExploringStarted to ture
    jmp DWS_GoForth

DWS_PossibleExit:
    brtst EQ, DWS_GoForth, globalExploringStarted           // if we havn't started yet do not exit
    subret retDoWallSurf

DWS_ExploringBegan:
    set globalExploringStarted, 1
    jmp DWS_GoForth

endt
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::




// ********************************************************************************************************
// Thread: Main
// ------------
dseg segment
    soundFile byte[] 'Mario Riff.rso'
dseg ends

thread main

    // Hardware Configuration
    // ----------------------
    set GUSS_port, IN_4                 // ultrasonic sensor is attached to port 4
    set GSS_port, IN_2                  // sound sensor is attached to port 2
    set GLS_port, IN_1                  // light sensor is attached to port 1


    // Software Configuration
    // ----------------------
    set globalMapWidth, 90              // 90 items in width and height add up to 8100 map pixels
    set globalMapScale, 30              // the map scale is set to 30 milimeters per pixel
    set globalStartingPosX, -450        // relative to mapcenter
    set globalStartingPosY, -450
    set globalStartingPosHeading, 0
```

```
    set globalExitCond, 100             // if we're within 100 mm of starting point stop the exploring process
    set globalExploringStarted, 0


    // Global Variable Initialization
    // ------------------------------
    mul globalMapSize, globalMapWidth, globalMapWidth
    arrinit globalMap,0,globalMapSize
    mov globalPosX, globalStartingPosX
    mov globalPosY, globalStartingPosY
    mov globalPosHeading, globalStartingPosHeading
    set globalUSSHeading, 0


    // 1. Find Closest Wall
    // --------------------
    subcall FindClosestWall, retFindClosestWall


    // 2. Explore Space
    // ----------------
    subcall DoWallSurf, retDoWallSurf

    // 3. Show & Save Results
    // ----------------------
    subcall DrawMap, retDrawMap
    subcall SaveMapTGA, retSaveMapTGA


    // Once we're done play the Mario Riff twice
    // -----------------------------------------
    set PF_A.loop, 0    ; no looping
    set PF_A.vol, 4     ; percent/25
    mov PF_A.filename, soundFile
    syscall SoundPlayFile, PF_A
stillplaying:
    syscall SoundGetState, SGS
    brtst NEQ, stillplaying, SGS.flags

    subcall Wait, retWait
    subcall Wait, retWait
    subcall Wait, retWait

    set PF_A.loop, 0    ; no looping
    set PF_A.vol, 4     ; percent/25
    mov PF_A.filename, soundFile
    syscall SoundPlayFile, PF_A
stillplaying2:
    syscall SoundGetState, SGS
    brtst NEQ, stillplaying2, SGS.flags

Forever:

    jmp Forever

    exit
endt
```