

RobotC mit NXT und/oder Tetrrix

Eine Einführung in die Programmierung in C des Lego NXT Roboters und dem Tetrrix System für Schule und Ausbildung mit Aufgaben

Die Programmierung für das Board „Arduino“ ist äquivalent mit der Software RobotC für NXT. Die Software muss aber separat erworben werden.

Beispiele und Programme auch unter www.engeln.info

Herstellung und Verlag:
BoD – Books on Demand, Norderstedt
ISBN 978-3-8482-5733-1

Inhalt

RobotC mit NXT und/oder Tetrrix.....	1
Eine Einführung in die Programmierung in C des Lego NXT Roboters und dem Tetrrix System für Schule und Ausbildung mit Aufgaben.....	1
Einführung	6
Ein bisschen Geschichte	7
Was ist Programmierung?	7
Was ist ein Programm?	8
Systemvoraussetzung:	9
Installation der Software	10
Download der Firmware.....	11
Machen wir eine Überprüfung – ein kleiner Test	12
Software testen	13
Bevor es nun wirklich los geht – wichtige Hinweise	14
Multitasking.....	14
Kommentare.....	16
Hallo Welt! – Text und Zeichenausgabe	17
Erste Töne.....	20
Sound Kontrolle.....	23
Musik, oder ... wie bringe ich dem NXT die Noten bei?.....	24
Töne und Klänge in der Physik	24
Motorenausgänge	26
Timer	27
Eine einfache Ampelanlage	28
Programmanweisungen	30
Variablen und Konstanten.....	31
Durchmesser, Radius und Umfang.....	32
Motoren Geschwindigkeit/ Leistungskontrolle.....	34
Sensoren – Umdrehungssensor	35
Brake und Coast/Float Modus.....	37
„Gespiegelte“ Motoren – Leistung umdrehen.....	38
Subroutinen.....	40

Sensoren – Ultraschallsensor	41
Sensoren – Lichtsensor.....	42
Lichtmessung mit dem NXT?	44
Sensoren – Tastsensor.....	45
Sensoren – Geräuschsensor	46
Sensoren – andere Sensoren.....	47
Winkelsensor	52
Magnetsensor.....	54
EOPD Sensor	56
Kontrollstrukturen – if-Anweisung.....	57
Kontrollstrukturen – while-Schleife.....	58
Robot C und Tetrrix.....	60
RobotC und Tetrrix starten	63
Servomotoren.....	65
Mehr als 2 DC Motoren und mehrere Servos	66
Aufgabenblatt.....	69
Mathematische Funktionen	70
abs	70
acos.....	70
asin	70
atan.....	71
atof	71
atoi.....	71
ceil	71
cos.....	72
cosDegrees	72
degreesToRadians	72
exp	72
floor	73
log	73
log10.....	74
PI.....	74

pow.....	74
radiansToDegrees.....	75
rand	75
sgn	76
sqrt.....	76
Batterie & Leistungs-Kontrolle	77
Schlüsselwörter in C (reservierte Wörter)	80
Operatoren.....	80
Bluetooth.....	81
Joystick Control	83
RobotC Virtual World	84
Versuche zur Wärmelehre	85
Geräuschpegelmessung im Straßenraum	86
Wir bauen ein Fernthermometer.....	87
Lichtmessung mit dem NXT.....	87
Ein automatischer Feuermelder.....	88
Beispiele	89
Links.....	92
Robotc für andere Systeme.....	93
Vex Robotics Roboterbausätze	94

Unterrichten mit dem NXT, Tetrix und der Programmiersprache RobotC. RobotC ist eine Sprache zur Programmierung des Lego Mindstorms NXT in C-ähnlicher Syntax. Besonders bei der Lösung komplexer Probleme bietet die Programmierung des Lego-Roboters in RobotC gegenüber der Benutzung der mitgelieferten Mindstorms-Software vor allem den Vorteil hoher Flexibilität.

Die Syntax ähnelt der von C bzw. C++. Ein Übergang zu anderen Systemen mit höheren Programmiersprachen wird hierdurch erleichtert.

Es gibt auch eine RobotC-Version für Roboter von Vex Robotics. RobotC ist eine komplette IDE-Festplattenschnittstelle mit einem sehr praktischen und leistungsstarken interaktiven Echtzeit-Debugger. Mit RobotC ist Multitasking möglich: Bis zu 10 Aufgaben können gleichzeitig vom Roboter ausgeführt werden. Andererseits installiert RobotC, wie in der Folge beschrieben, eine eigene Firmware auf dem Roboter, die leistungsfähiger als die Standard-Firmware ist. Im Vergleich zu anderen Programmiersoftware-Programmen sind RobotC-Programme wesentlich schneller.

RobotC unterstützt standardmäßig alle gängigen Sensoren von Lego. Eine Treiberbibliothek ist als Open-Source-Software verfügbar, die alle im Handel erhältlichen Sensoren und Zubehörteile umfasst. Somit kann der NXT-Roboter mit RobotC ohne Einschränkungen programmiert werden.

Die Carnegie Mellon University hat ein 3D-Simulationstool, das mit RobotC kompatibel ist, veröffentlicht. Das Handbuch ist eine aus der Praxis entstandene Konzeption zur Einführung in das LEGO®MINDSTORMSR NXT und Tetrix System für Schüler und Lehrer.

©2013 von Frank Engeln,
Mehring Str. 35, D-48499 Salzbergen Germany
Alle Rechte vorbehalten.
Autor: Frank Engeln

Einführung

Dieses Dokument richtet sich an alle jene, die noch nie programmiert haben, sich einen Überblick über eine Programmiersprache verschaffen wollen oder ihre Kenntnisse in bestimmten Bereichen auffrischen möchten.

Wenn bereits Programmiererfahrung mit einer Entwicklungsumgebung gesammelt wurde, bzw. mit einem Editor und Compiler/Interpreter keine Probleme auftreten, kann man diese Einführung bedenkenlos überspringen.

LEGO Education bietet erstmals als Alternative zu den graphischen Programmieroberflächen ROBO LAB und NXT-G mit der Software ROBOT-C, eine textbasierte Programmiersprache für Schulen an. ROBOT-C wurde entwickelt von der Carnegie Mellon Robotics Academy und kann zur Programmierung von Robotern mit dem NXT (auch RCX) verwendet werden.

ROBOT-C basiert auf der weltweit eingesetzten **Standardprogrammiersprache C**.

Eigenschaften:

- Kompatibel zu NXT und RCX und Tetrrix
- Bluetoothfähig
- Debug-Modus
- Unterstützt LEGO externe Sensoren
- Hilfsfunktionen für Schüler und Lehrer
- Text Editor
- Trigonometrische Funktionen
- Statistische Funktionen
- uvm.

Die Software ist ausschließlich in englischer Version erhältlich und wird als **Einzellizenz** sowie als **Schullizenz** angeboten. Die Schullizenz ist limitiert auf 12 Arbeitsplätze.

Erhältlich unter www.nxt-roboter.de



Ein bisschen Geschichte

Die Programmiersprache C

C ist eine imperative Programmiersprache (Imperative Programmierung, in der Informatik die geradlinige Abfolge von Befehlen), die der Informatiker Dennis Ritchie in den frühen 1970er Jahren an den Bell Laboratories für das Betriebssystem Unix entwickelte. Seitdem ist sie auf vielen Computersystemen verbreitet.

Die Anwendungsbereiche von C sind sehr verschieden. Es wird zur System- und Anwendungsprogrammierung eingesetzt. Die grundlegenden Programme aller Unix-Systeme und die Systemkerne vieler Betriebssysteme sind in C programmiert. Zahlreiche Sprachen, wie C++, Objective-C, C#, Java, PHP oder Pearl orientieren sich an der Syntax und anderen Eigenschaften von C.

C ist eine Programmiersprache und C++ die Erweiterung, also die verbesserte Variante von C. RobotC ist eine Mischung aus beiden Sprachen, es sind Elemente vorhanden, die es in C normalerweise nicht gibt. Es fehlen aber einige wichtige Funktionen der Sprache C++. Um einen Lego Mindstorms Roboter zu programmieren ist RobotC aber die richtige Wahl.

Was ist Programmierung?

Programmierung bezeichnet...

... die Tätigkeit, Computerprogramme (Software) zu erstellen. Im weiteren Sinne versteht man dabei alle Tätigkeiten, die mit dieser Programmerstellung verbunden sind, insbesondere auch den konzeptionellen Entwurf. Im engeren Sinne bezeichnet Programmierung lediglich das Umsetzen dieses konzeptionellen, abstrakten Entwurfes in konkreten Quelltext. (aus Wikipedia)

Im Klartext:

Mit „Programmierung“ wird die Entwicklung von Computerprogrammen bezeichnet.

Für das Fach:

Die Programmierung ist ein Teilgebiet der Informatik, das sich im weiteren Sinne mit Methoden und Denkweisen bei der Lösung von Problemen mit Hilfe von Computern und im engeren Sinne mit dem Vorgang der Programmerstellung befasst. Unter einem Programm versteht man dabei eine in einer speziellen Sprache verfasste Anleitung zum Lösen eines Problems durch einen Computer. Programme werden auch unter dem Begriff Software subsumiert. Konkreter ausgedrückt ist das Ziel der Programmierung bzw. Softwareentwicklung, zu gegebenen Problemen Programme zu entwickeln, die auf Computern ausführbar sind und die Probleme korrekt und vollständig lösen, und das möglichst effizient.

Unter Programmieren versteht man das Erstellen von Programmen. Zuständig für die Entwicklung von Programmen ist der Programmierer. Die Formulierung von Programmen erfolgt in der Regel unter Verwendung der uns bekannten Zeichen wie Buchstaben, Ziffern und Sonderzeichen.

Eine Programmbeschreibung wird auch als Programmcode, Quellcode oder Source-Code bezeichnet. Der Programmcode selbst kann im Allgemeinen noch nicht direkt von einem Computer ausgeführt werden, er muss zuvor noch mit Hilfe eines Compilers in eine maschinenverständliche Form - ein ausführbares Programm - transformiert werden. Die Programmausführung wird durch den Aufruf des ausführbaren Programms gestartet. (aus: Programmieren spielend gelernt mit dem Java-Hamster-Modell, Dr.-Ing. Dietrich Boles, Universität Oldenburg)

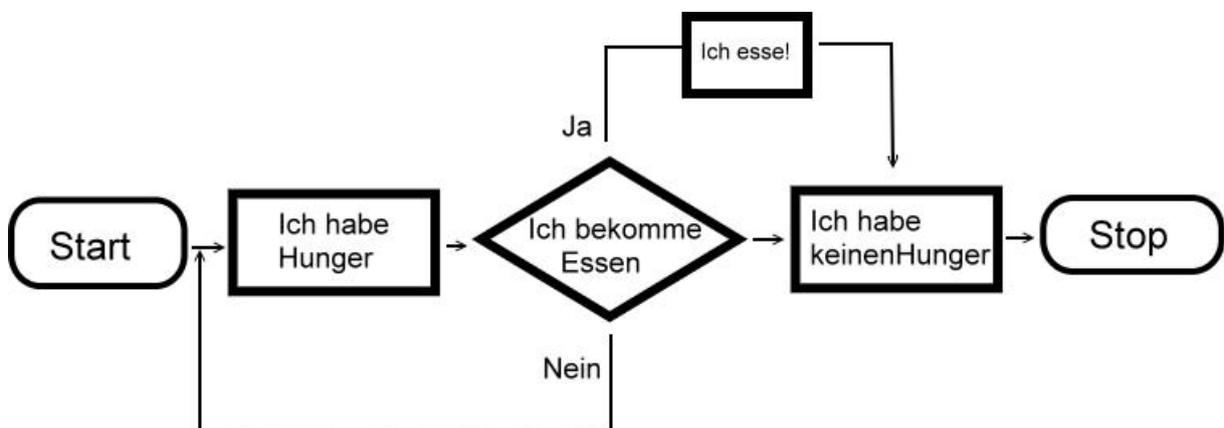
Was ist ein Programm?

Ein Programm ist nach DIN 44300 definiert als eine zur Lösung einer Aufgabe vollständige Anweisung zusammen mit allen erforderlichen Vereinbarungen. Diese Definition sagt nichts darüber aus

- wer diese Anweisung als Prozessor auszuführen hat, ob Durchschnittsmensch, Spezialist, Organisation oder Rechnersystem;
- mit welchen Mitteln Anweisungen und Vereinbarungen beschrieben sind.
- Als Programm wird ein in einer Programmiersprache formulierter Algorithmus bezeichnet. Die Formulierung von Programmen erfolgt in sehr konkreter und eingeschränkter Form:
- Programme werden im exakt definierten und eindeutigen Formalismus einer Programmiersprache verfasst.
- Daten, auf denen Programme operieren, unterliegen einer festgelegten Darstellungsform.

Ein Programmablaufplan ist ein Ablaufdiagramm für ein Computerprogramm, das auch als Flussdiagramm (engl. flowchart) oder Programmstrukturplan bezeichnet wird. Es ist eine graphische Darstellung zur Umsetzung eines Algorithmus in einem Programm und beschreibt die Folge von Operationen zur Lösung einer Aufgabe.

Hier ein Versuch eines Ablaufdiagramms zu meiner Person:



Wenn auch nicht 100% richtig – aber ich hoffe verständlich. Es geht um das Prinzip.
Die Symbole für Programmablaufpläne sind in der DIN 66001 genormt.
In diesem Dokument kann man die Symbole nachlesen.
<http://www.fh-jena.de/~kleine/history/software/DIN66001-1966.pdf>

Systemvoraussetzung:

- Windows® XP, Service Pack 2 oder höher
- Windows® Vista 32-bit und 64-bit
- Windows® 7 32-bit und 64-bit

Die aktuelle Version ist die **Version 3.5.4**

Neue und aktualisierte Debugging Debugger-Fenster
Unterstützung für Arduino
Unterstützung für das MATRIX building system
Verbesserte Unterstützung für virtuelle Roboter-Simulation

LEGO Start Page

ROBOTC for MINDSTORMS Start Page

RVW allows you to program virtual robots in a fun and exciting new way

OPERATION RESET LATEST WORLD

FOR A LIMITED TIME Upgrade to **ROBOTC 3.0** and get a one year RVW license for **FREE** »

Latest News: [RSS](#)

New Robot Virtual World: Operation Reset!

Posted on Friday, November 30th, 2012

The Robot Virtual World team is thrilled to announce their latest level pack: **Robots to the Rescue – Operation Reset**.

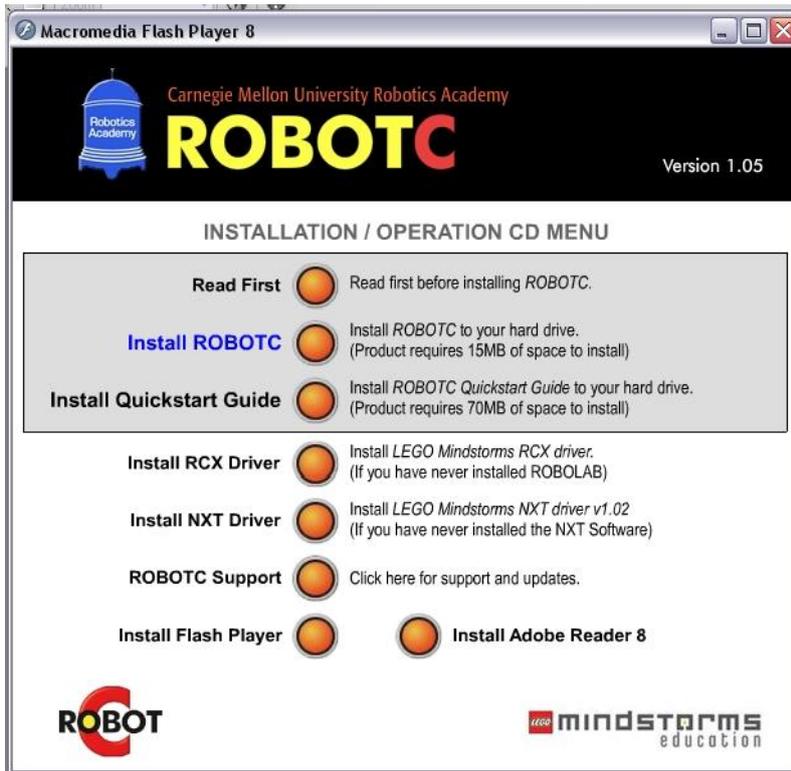
Robots to the Rescue – Operation Reset is the third version of our virtual world set in a crystal mining colony on Planet H99. An intergalactic storm has knocked out all of the systems in the colony, and it's up to you to program the colony's robots to **restore power to Communication Towers**, **collect the Unobtanium energy crystals**, and **refuel the rocket**. In game testing, some of the things middle and high school kids said were "It's fun!", "It's like a video game!", "I like that I get to see my code work immediately.", and "I like the story behind the missions."

Operation Reset represents several months of hard work on new features and huge

not as tpty.

Errors

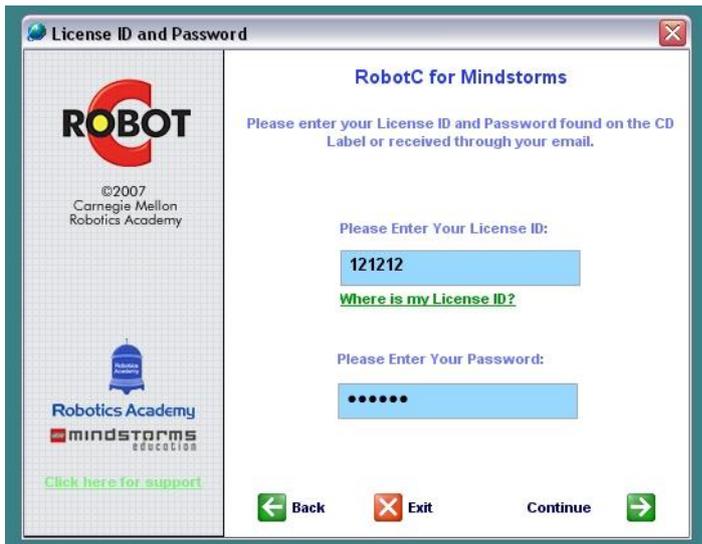
Installation der Software



Einfach die CD einlegen und den Anweisungen folgen. Anschließend das Programm aktivieren.



Nun die Lizenznummer eingeben und das Passwort. Fertig.



Den Aktivierungscode findet man bei bzw. auf der CD.

Wenn die Software auf dem Computer installiert ist, muss man die Firmware auf den NXT spielen.

Download der Firmware

Wenn die RobotC Firmware auf den NXT Baustein aufgespielt wurde, kann keine andere Software auf dem NXT laufen. Die Firmware kann aber immer durch andere Firmware ersetzt werden. Die Firmware nimmt eine Zwischenstellung (Verbindungsglied) zwischen Hardware(NXT) und der Software(RobotC) ein.

1. NXT mit dem Computer verbinden.
2. NXT einschalten.
3. RobotC für Mindstorms auf dem Computer starten (Start – Programme – RobotC – RobotC für Mindstorms).
4. Wenn die Software gestartet ist, im Menü auf „Robot“ klicken und dann „Download Firmware“. In dem Fenster NXT Brick Download sollte nun der Name des NXT stehen, sowie die Adresse. Auf den „F/W Download Button“ klicken. Es öffnet sich ein Fenster mit einer Auswahl der Firmware. Normalerweise steht dort nur eine Firmware zur Auswahl. Stehen mehrere zur Auswahl, immer die Firmware mit der höchsten Zahl wählen. Diese auswählen und auf „ok“ klicken. Der Download startet. Im Fenster wird „download completed“ angezeigt.
5. Fenster schließen.

Machen wir eine Überprüfung – ein kleiner Test

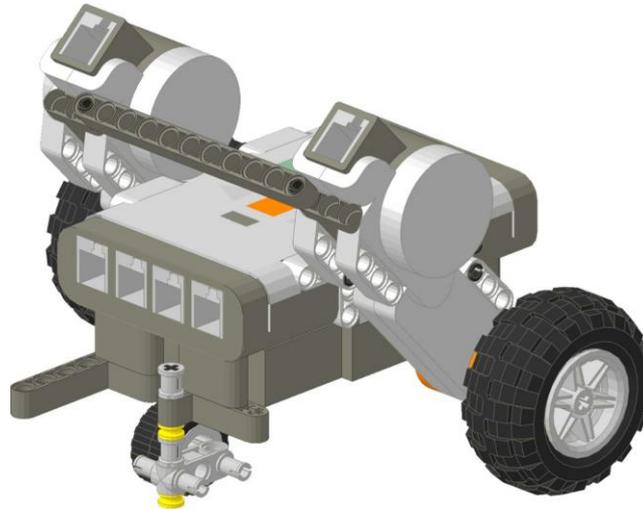
Beantworte folgende Fragen mit wahr oder falsch.

1. Jedes Mal, wenn ich ein Programm auf den NXT laden möchte, muss die Firmware aufgespielt werden.
2. Ohne Firmware kann kein Programm in RobotC ausgeführt werden.
3. Alle Firmware sind identisch. Ist eine Firmware aufgespielt, kann man in verschiedene Sprachen programmieren.
4. Software und Firmware sind dasselbe.
5. Die Firmware wird in RobotC unter "kompilieren und download" aufgespielt.

Software testen

Nachdem die Software und Firmware aufgespielt wurden benötigen wir einen einfachen Roboter zum testen. **Eine Bauanleitung findet man unter:**

http://legoengineering.com/library/cat_view/30-building-instructions/38-nxt-based-creations.html



Übertragen wir nun ein Demo-Programm auf den Roboter.

1. Klicke in der Software auf „File“ – „Open Sample Programm“
2. Wechsel in den Ordner „Training Samples“ und öffne das Beispiel „MotorC Forward“.

Auf der rechten Seite (im Editor) steht nun dieser Programmcode:

```
task main()
{
    motor[motorC] = 100;
    wait1Msec(3000);
}
```

Um die linke Seite (im Editor) kümmern wir uns später.

Der Roboter muss mit dem Computer verbunden sein. Das Programm soll nun auf den Roboter übertragen werden. Dazu klickt man im Menü „Robot“ auf „Download“ oder durch drücken der Funktionstaste F5. Es öffnet sich ein Fenster. Klicke auf „Start“ in dem Fenster „Program Debug“. Der Roboter sollte nun starten und sich im Kreis drehen. Wenn JA, dann funktioniert alles einwandfrei.

Auf dem NXT Display unter „My Files“ – „Software Files“ findet sich nun das erste Programm mit dem Namen „MotorC Forward“. Das Programm ist auf dem NXT gespeichert und kann jeder Zeit abgerufen werden.

Bevor es nun wirklich los geht – wichtige Hinweise

```
task main()
{
    motor[motorC] = 100;
    wait1Msec(3000);
}
```

Dieses Programm ist ein Standard-Programm, geschrieben in C. Die Textsprache wird in dem Compiler „RobotC“ in Maschinensprache umgewandelt und auf den Roboter übertragen, wo es ausgeführt werden kann.

Der Text auch als Code bezeichnet. **Dabei ist die Groß- und Kleinschreibung zu beachten!**

Wenn in dem Programm (RobotC) bekannte Worte als Anweisung geschrieben werden, so werden diese farblich dargestellt.

Eine einfache Anweisung ist:

```
motor[motorC] = 100;
```

Diese sagt dem Roboter: Starte den Motor C mit einer Leistung von 100%.

```
wait1Msec(3000);
```

bedeutet: Der Roboter soll 3000 Millisekunden warten, also 3 Sekunden den Motor C mit 100% Leistung laufen lassen.

Jede Anweisung muss mit einem Semikolon enden.

Zeilenumbrüche und Leerzeichen in RobotC werden vom Compiler nicht berücksichtigt. Sie dienen der Überschaubarkeit des Programmcodes.

task main { }

Der Einstiegspunkt von C-Programmen ist der **main**-Task, der in jedem Programm definiert werden muss. Es können jedoch zusätzlich mehrere Tasks simultan ausgeführt werden (**Multitasking**). Ein Task mit der Bezeichnung name wird wie folgt definiert:

Multitasking

```
task name()
{
    // Programmcode dieses Tasks
}
```

Beispiel mit Tetrrix Servo Motor:

```
task Rheben()      // frei gewählt
{
  for(int i = 0; i < 3; i++)
  {
    servo[RM] = 60;
    wait1Msec(1000);
    servo[RM] = 40;
    wait1Msec(1000);
  }
}

task fahren()     // frei gewählt
{
  motor[motorA] = 100;
  motor[motorB] = 100;
  wait1Msec(1000);
}

task main()
{

  StartTask (Rheben); // hier wird das 1. Programm gestartet
  StartTask (fahren); // hier wird das 2. Programm gestartet
  while(true)
  {
    wait1Msec(300);
    nxtDisplayCenteredBigTextLine(0, "TASK Rheben");
  }
  return;

}
```

Der Begriff **Multitasking**, auch **Mehrprozessbetrieb** genannt, bezeichnet die Fähigkeit eines Programms, mehrere Aufgaben (Tasks) nebenläufig auszuführen

Kommentare

In allen Programmiersprachen gibt es die Möglichkeit, im Quelltext Notizen zu machen. Für andere oder auch für sich selbst, denn nach ein paar Wochen werden Sie möglicherweise Ihren eigenen Quelltext nicht mehr ohne Weiteres verstehen.

Kommentare eignen sich zur Unterbringung von Informationen, welche nicht Teil des eigentlichen Programms sind.

- Aufgabe:**
1. Erstellt ein Programm und schreibt Kommentare in der Form wie es unten zu sehen ist.
 2. Versuche einmal ein Bild aus Kommentare zu zeichnen.
 3. Macht einen Urheber oder Programmierhinweis

In RobotC gibt es zwei Arten von Kommentaren. Die Erste kommt von der Programmiersprache C und beginnt mit `/*` und endet mit `*/`. Der Compiler ignoriert Kommentare

```
/* Das ist ein Kommentar */
```

Bsp.:

```
/*
    """"
    (o o)
---ooO---( )---Ooo---
*/
```

Die zweite Art ist `//` und dieser endet mit Beginn einer neuen Zeile.
`//` ein Zeilen Kommentar

Bsp.:

```
// Dieses Programm soll eine Hilfe sein.....
```

```
/******Frank Engeln*****/
```

Hallo Welt! – Text und Zeichenausgabe

Eine alte Tradition in der Informatik ist das „Hallo Welt!“ Programm (HWP). Die HWP ist ein einfaches Programm, deren primäres Ziel ist es, den Programmierer auf die Details der Schreiben, Speichern, Kompilieren und Ausführen eines Programms.

Der NXT hat ein schwarz-weißes Display mit der Auflösung 100*64. Das LC Display ermöglicht die Anzeige von Texten und beispielsweise erfassten Messwerten. In der unteren linken Ecke ist der Punkt (0, 0) und oben rechts (99, 63).



Den Standort der Ausgaben gibt man mit x und y Koordinaten an (x von links, y von unten).

Für die xy Koordinaten kann man auch folgendes benutzen
⇒ xPosition
⇒ yPosition

```
// Programm zum Ausgabe auf das Display
task main()
{
  nxtDisplayTextLine(4, "Hallo Welt"); // Hallo Welt auf dem Display
  wait1Msec(5000); //5 Sekunden Dauer
}
```

Es gibt 8 Textlinien. 0 ist oben und 7 ist die untere Linie im Display.

eraseDisplay(); löscht den Bildschirminhalt

// Programm zur Ausgabe von einer bestimmten Position auf dem Display

nxtDisplayStringAt(xPosition, yPosition, text, parm1, parm2)

Param1 etc. sind optionale Variablen, die mit dem Text ausgegeben werden können. Die Angabe von %d im Textparameter ist nötig. Statt parm1 (Parameter 1) kann auch x,y,z genommen werden.

Z.B.:

```
task main()
{
  int x = 3;
  int y = 24;
  nxtDisplayStringAt(5, 25, "Lottozahlen %d %d", x, y);
}
```

nxtDisplayTextLine(nLineNumber, s Zeichenkette, parm1, parm2) // ähnlich
nxtDisplayString()
nxtDisplayCenteredTextLine(); // ähnlich nxtDisplayTextLine, nur zentrierter Text

Aufgabe: 1. Lasse dich von dem NXT mit deinem Namen in der Mitte des Displays begrüßen. Speicher die Aufgabe unter "halloName.c" ab.

Auf der nächsten Seite gibt es eine Übersicht der Display Anweisungen.

n wird häufig zur Benennung von Variablen verwendet, deren Werte auf natürliche Zahlen beschränkt sind.

var1, var2 sind Platzhalter für Werte (Variablen)

String = Zeichenketten (**s** als Variable)

nZeilenNummer oder nLinienNummer

nViewStateNXT

Steuert, welche der vier verschiedene vordefinierte LCD-Bildschirme angezeigt werden sollen. Die verschiedenen Bildschirme können wahlweise aktiviert werden, über die linke und rechte Taste. Es gibt das "Standard" NXT-Display, ein Sensor-Display, ein Motor-Display, eine grafische Darstellung (z.B. Datenlogging) .

eraseDisplay()

Das NXT Display löschen.

nxtDisplayString(nZeilenNummer, sZeichenkette, var1, var2)

Formatiert eine Zeichenkette nach dem Format in **sZeichenkette** anhand von Parametern var1 und var2. Das Ergebnis wird auf der Textzeile in **nZeilenNummer** gezeigt. Der Rest der Zeile wird nicht verändert, so dass, wenn das Ergebnis 8 Zeichen ist, nur die ersten acht Zeichen der Zeile aktualisiert werden. Der Inhalt der übrigen Zeichen in der Zeile wird nicht verändert. Zu beachten ist, dass var1 und var2 optionale Parameter sind und der Wert Null ersetzt wird, wenn sie nicht angegeben werden.

nxtDisplayClearTextLine(nZeilenNummer)

Löschen (löscht, füllt mit Leerzeichen) der Textzeile in **nZeilenNummer**

nxtDisplayTextLine(nZeilenNummer, sZeichenkette, var1, var2)

Das Gleiche wie **nxtDisplayString**, außer dass die ganze Textzeile ersetzt wird. Die Linie wird am Ende mit Leerzeichen gefüllt.

nxtDisplayCenteredTextLine(nZeilenNummer, sZeichenkette, var1, var2)

Die gleiche Funktion wie **nxtDisplayTextLine** , nur dass der Text zentriert wird, statt linksbündig.

nxtDisplayStringAt(xPos, yPos, sZeichenkette, var1, var2)

Formatiert eine Zeichenkette und gibt es auf dem Display an den Punkten (**xPos, yPos**) aus.

nxtDisplayBigStringAt (xPos, yPos, sZeichenkette, var1, var2)

Die gleiche Funktion wie `nxtDisplayStringAt`, es wird aber eine 16 Pixel große Schrift genommen.

nxtScrollText(sZeichenkette, var1, var2)

Schiebt das LCD Bild eine Zeile nach oben. Dann wird eine Textfolge formatiert und an der untersten Zeile auf dem LCD geschrieben. (Scrollen kennt man von den PC Programmen).

Es gibt auch eine Reihe von weiteren Befehlen, mit denen man recht aufwändige Zeichnungen erstellen kann.

nxtClearPixel(xPos, yPos)

Löschen eines Punktes an den angegebenen Koordinaten

nxtSetPixel(xPos, yPos)

Setzen eines Punktes an den angegebenen Koordinaten

nxtDrawCircle(Links, Oben, Durchmesser)

Zeichnet einen Kreis an einen Punkt (**Links, Oben**) mit einem bestimmten Durchmesser.

nxtDrawLine(xPos, yPos, xPosTo, yPosTo)

Zeichnet eine Linie zwischen den Punkten (**xPos, yPos**) und (**xPosTp, yPosTo**).

nxtDrawRect(links, oben, rechts, unten)

nxtEraseRect(links, oben, rechts, unten)

nxtFillRect(links, oben, rechts, unten)

Arbeitet an einem Rechteck, definiert durch die Position (links, oben, rechts, unten). Entweder zeichnet (d.h. einen Umriss zeichnen), füllt oder löscht man eine Zeichnung - abhängig von der Funktion.

nxtDrawEllipse(links, oben, rechts, unten)

nxtEraseEllipse (links, oben, rechts, unten)

nxtFillEllipse(links, oben, rechts, unten)

Arbeitet auf einer Ellipse begrenzt durch das Rechteck (links, oben, rechts, unten). Entweder zeichnet (d.h. einen Umriss zeichnen), füllt oder löscht man eine Zeichnung - abhängig von der Funktion.

nxtDisplayIconFile(xPos, yPos, sFileName)

Zeigt ein Icon-Datei an der markierten Stelle (xPos, yPos) Hinweis: Dieser Befehl ist derzeit nicht umgesetzt worden. Es wird in einer zukünftigen Version RobotC eingearbeitet.

Aufgabe: 1. Entwerfe Grafiken – sei ein kleiner Kandinsky. Speicher die Aufgabe unter “kandinsky.c“ ab.

Zur Information: Kandinsky war ein Künstler des Expressionismus und vor allem der abstrakten Kunst. Er stammte aus einer wohlhabenden Teehändlerfamilie aus Moskau.



Erste Töne

Der **Morsecode** oder **Morsekode** ist ein Verfahren zur Übermittlung von Buchstaben und Zeichen. Dabei wird ein konstantes Signal ein- oder ausgeschaltet.

Der Code verwendet drei Symbole, die Punkt (·), Strich (–) und Pause () genannt werden, gesprochen als Dit, Dah und „Schweigen“. Genauer gilt Folgendes:

- Ein Dah ist üblicherweise dreimal so lang wie ein Dit.
- Die Pause zwischen zwei gesendeten Symbolen ist ein Dit lang.
- Zwischen Buchstaben in einem Wort wird eine Pause von Dah eingeschoben.
- Die Pause zwischen Wörtern beträgt sieben Dits.

--- -- ···· / --- -- ····
M O R S E (space) C O D E

Lateinische Buchstaben Buchstabe im Code

A · —	J · — — —	S · · ·
B — · · ·	K — · —	T —
C — · — ·	L · — · ·	U · · —
D — · ·	M — —	V · · · —
E ·	N — ·	W · — —
F · · — ·	O — — —	X — · · —
G — — ·	P · — — ·	Y — · — —
H · · · ·	Q — — · —	Z — — · ·
I · ·	R · — ·	

- Aufgabe:**
1. Erstelle ein neues Programm. Speichere dieses mit dem Namen „morsen.c“ ab.
 2. Was bedeutet: · — · · · — — · — — — / — · — · — — ?
-

Der NXT kann Töne erzeugen oder Sound Daten abspielen.

Um einen Ton erklingen zu lassen benötigt man die Anweisung:

PlayTone(Frequenz, Dauer);

Der Ton wird vom internen Lautsprecher des NXT gespielt. Die Einheiten sind dabei 1 = 1 Hertz. Die Dauer wird angegeben in 1 = 1/100tel einer Sekunde.

Bsp.:

```
task main()
{
  PlayTone(440, 200); // Spiele Ton(Frequenz (Kammerton a1), Dauer (2sec.))
}
```

3. Schreibe das folgende Programm ab, speichere es und übertrage es auf den NXT Baustein. Starte das Programm.

```
Task main()
{
PlayTone(440,200); PlayTone(440,75); wait10Msec(200); //Pause
PlayTone(440,200); PlayTone(440, 75); PlayTone(440, 75); PlayTone(440,200);
wait10Msec (200); PlayTone(440,200); PlayTone(440, 75); PlayTone(440,200);
PlayTone(440, 75);
wait10Msec (400);
PlayTone(440,200); PlayTone(440, 75); PlayTone(440, 200); PlayTone(440, 75);
wait10Msec (200);
PlayTone(440, 200); PlayTone(440, 200); PlayTone(440,200); wait10Msec (200);
PlayTone(440,200); PlayTone(440, 75); PlayTone(440,75); wait10Msec (200);
PlayTone(440, 75); wait10Msec (200);
}
```

4. Was ist die Bedeutung? Notiere Punkte und Striche und Übersetze. Was muss geändert werden, damit es noch mehr nach „Morsen“ klingt?

5. Programmiere deinen Namen im Morsecode. Schreibe dazu erst deinen Namen in Punkte, Strichen und Pausen auf. Schreibe nun das Programm und speichere es unter „morseNamen.c“ ab. Spiele es nun deinem Tischnachbarn vor und lasse ihn den Morsecode übersetzen.

Sound Kontrolle

RobotC bietet ein umfassendes Angebot von Funktionen für die Kontrolle der NXT-Lautsprecher.

Die RobotC Firmware bietet bis zu 10 Sound-Elemente für die Wiedergabe an.

RobotC spielt sowohl komprimierte als auch unkomprimierte Audiodateien ab. RobotC, die Entwicklungsumgebung, bietet einem die Möglichkeit Audiodateien zu komprimieren. Er befindet sich im Menü "Windows-> NXT Brick-> Datei-Management".

ClearSounds();

Löscht alle bestehenden und sich im Speicher befindenden Sound Befehle.

PlayImmediateTone(Frequenz, Dauer);

Sofort den Ton spielen mit der angegebenen Frequenz und Dauer.

PlaySound(sound);

Spiele einen vorgegebenen Sound ab (buzz, beep, click, ...).

PlaySoundFile(sFileName);

Spielt eine Sound-Datei aus dem NXT-Datei-System. Diese Datei muss auf dem NXT sein.

Wenn ein Programm auf den NXT geladen wird, wird eine Prüfung durchgeführt, um sicherzustellen, dass alle Sound-Dateien auf dem NXT sind; wenn nicht, dann wird RobotC versuchen, diese Dateien von dem PC auf den NXT zu laden.

PlayTone(Frequenz, Dauer);

Spielt einen konstanten Ton auf der angegebenen Frequenz und Dauer.

kMaxVolumeLevel

Eine Konstante (k). Die maximale Lautstärke, die gesetzt werden kann.

nVolume

Lautstärke. Bereich 0 bis 4 (lauteste).

bPlaySounds , bSoundActive, bSoundQueueAvailable ist für Fortgeschrittene und wird später behandelt.

Musik, oder ... wie bringe ich dem NXT die Noten bei?

- Aufgabe:**
1. Erstelle ein neues Programm. Speichere dieses mit dem Namen „meineMusik.c“ ab.
 2. Schreibe das Programm Bsp. a) ab, speicher es, kompiliere es (F5) und übertrage es auf den NXT (F6).
 3. Ändere die Töne ab, erfinde Melodien.
 4. Versuche dich als Komponist und erfinde Musikstücke. Speichere diese ab.

Eine Warnung noch vorweg, C ist case sensitive, es kommt also sehr darauf an, ob man etwas GROSS oder klein schreibt.

Töne und Klänge in der Physik

Töne und Klänge sind die Grundbausteine der Musik. Physikalisch gesehen ist ein Ton eine Sinusschwingung, charakterisiert durch die Parameter Frequenz (Tonhöhe = Schallschwingungen pro Sekunde) und Amplitude (Lautstärke). Das menschliche Ohr nimmt Schallschwingungen ab einer Frequenz von etwa 16 Hz (Einheit Hertz, 1 Hz = 1 Schwingung pro Sekunde) als sehr tiefen Ton wahr. Der höchste wahrnehmbare Ton liegt bei Kindern und Jugendlichen bei etwa 20000 Hz (20 kHz) und lässt mit höherem Alter nach. Als Bezugspunkt für die Tonfrequenzen in der Musik ist der Kammerton a^1 mit einer Frequenz von 440 Hz festgelegt.

Bsp. a):

```
task main()
{
  PlayTone(440, 400); // Spiele Ton(Frequenz (Kammerton  $a^1$ ), Dauer (4 sec.))
  wait10Msec (500);
}
```

Übersicht der Frequenzen:

Ton	Frequenz								
	1.	2.	3.	4.	5.	6.	7.	8.	9.
B	247	494							
A#	233	466							
A	220	440							
G#		415							
G		392							
F#		370							
F		349							
E		330							
D#		311							
D		294							
C#		277							
C		262							

Wenn aus dem Kammerton A (fett gedruckt) ein tiefes A werden soll, so wird die Frequenz halbiert: 220 Hz.

Soll ein hoher Ton erklingen, so verdoppelt man den Wert.

A 440 880 1760 3520 7040 14080

Der NXT wartet nicht, bis der Ton zu Ende gespielt ist. Wenn du also eine Tonfolge für ein Lied programmierst, füge besser wait10Msec (); - Befehle hinzu, die etwas länger sein sollten, als die Tondauer!

Aufgabe: Fülle die Tabelle bis zur 9. Stufe aus.



Für die nächsten Aufgaben brauchst du einen einfachen Roboter mit 2 Motoren an Ausgang A und C.



Bauanleitungen vom Lego Designer 2.3 : <http://ldd.lego.com/download>

Motorenausgänge

Für die Ansteuerung der Motoren gibt es natürlich eine Vielzahl von Funktionen. Der NXT verfügt über drei Motorausgänge mit den Bezeichnungen A, B und C. Für diese Ausgänge sind passende Konstanten definiert:

- motorA
- motorB
- motorC

Die Motorenleistung kann von -100 (100% Leistung rückwärts) bis 100 (100% Leistung vorwärts) eingestellt werden. Der Wert 0 stoppt den Motor.

Ein einfaches Programm könnte also folgendermaßen aussehen:

```
// Programm Motoren an

task main()
{
    motor[motorA]= 100; // vorwärts
    motor[motorC]= -100; // rückwärts
}
```

Hier werden die Motoren A und C eingeschaltet und zwar mit 100% der eigentlichen Leistung. Was fehlt ist noch die Dauer der Leistung.

Timer

Um Verzögerungen in ein Programm einzubauen benötigt man den Befehl „wait“. Darüber hinaus unterstützt der Timer die Arbeit wie Stoppuhren oder einfach zählen.

„Wait“ wird dazu führen, dass der Roboter wartet und zwar eine bestimmte Anzahl von Millisekunden, bevor die nächste Anweisung in einem Programm ausgeführt wird.

"Wait_time" ist ein Integer-Wert (1 = 1/1000tel von einer Sekunde).

Der maximale Wert für „wait1Msec“ ist 32768 Millisekunden, oder 32,768 Sekunden.

Um den Roboter sich für 2 Sekunden im Kreis drehen zu lassen schreibt man:

```
task main()
{
    motor[motorA]= 100; // vorwärts
    motor[motorC]= -100; // rückwärts
    wait1Msc(2000);
}
```

Um größere Werte zu erhalten, bzw. die Dauer des Wartens zu erhöhen nimmt man „wait10Msec“, also den 10-fachen Wert.

Der maximale Wert für „wait10Msec“ ist 32768 Zehntelsekunden, oder 327,68 Sekunden.

ClearTimer(theTimer);

Setzt den Wert des angegebenen Timers auf null.

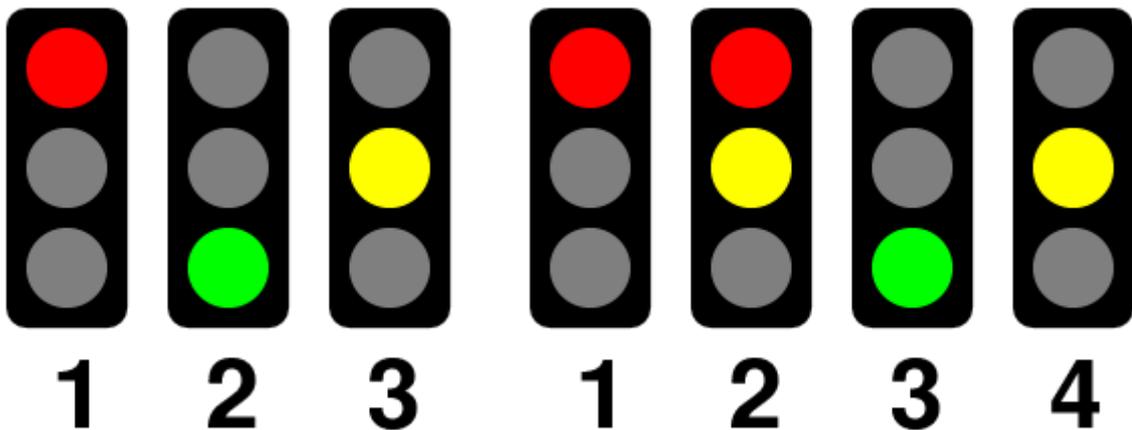
Eine einfache Ampelanlage

Für diese Aufgabe benötigst du 3 Lampen mit den Farbbausteinen rot, gelb, grün. Die Lampen werden an die Motorenausgänge des NXT angeschlossen.

Aufgabe: 1. Erstelle eine Ampelschaltung wie in dem Bild unten links. Speichere dieses mit dem Namen „Ampel.c“ oder „Ampel.c“ab.

Eine normale europäische Lichtzeichenanlage steuert den Verkehr dabei mit Hilfe der drei Signalfarben Grün, Gelb und Rot. Zur Regelung des Verkehrs werden diese Farben einzeln oder in Kombination angezeigt. Die Reihenfolge (auch Signalfolge oder Farbbildfolge genannt) solch einer Lichtzeichenanlage ist dabei immer:

- Grün: Der Verkehr ist freigegeben
- Gelb: Auf nächstes Signal warten
- Rot: Keine Einfahrerlaubnis



Aus: <http://de.wikipedia.org>

Varianten der Standard-Lichtzeichenanlage in verschiedenen Ländern

In einzelnen Ländern sind noch zusätzliche Farbkombinationen zugleich oder hintereinander möglich:

Land	Ampelphase	Bemerkung
Deutschland, Großbritannien, Österreich, Ungarn, Schweiz, Polen, Norwegen, Russland,	Rot-Gelb (gleichzeitig): Zwischen rot und grün (Siehe Bild rechts)	Achtung, gleich wird die Erlaubnis zur Fahrt gegeben
Schweden	Grün-Gelb:	Achtung, es wird gleich rot
Belgien, Dänemark, Frankreich, Griechenland, Irland, Italien, Luxemburg, Niederlanden, Australien, Taiwan, Brasilien und den Vereinigten Staaten.	Grün folgt direkt auf Rot	-----
Vereinigte Staaten, Südafrika, Taiwan	Rotes Blinklicht	Stopp! Anhalten, dann langsam weiterfahren, wenn Kreuzung frei.
Österreich, Litauen, der Türkei, Mexiko und Israel	Grünes Blinklicht am Ende der Grünphase	Achtung, es wird gleich gelb gezeigt
China	Grünes Blinklicht am Ende der Grünphase	Achtung, es wird gleich rot gezeigt

Aufgabe: 2. Erstelle die verschiedenen Ampelschaltungen der unterschiedlichen Länder. Speichere diese mit dem Namen z.B. „Ampel_Deutschland.c“ ab.

Programmanweisungen

Dieses Programm hat 2 Anweisungen.

```
task main()
{
    motor[motorA]= 80;
    wait1Msc(2000);
}
```

„motor[motorA]= 80;“ Diese Anweisung sagt dem Roboter den Motor, der an Ausgang A angeschlossen ist, mit einer Vorwärtsdrehung zu starten. Die Leistung des Motors ist 80%.

„wait1Msc(2000);“ Diese Anweisung sagt dem Roboter, dass er für 2 Sekunde warten soll.

Aufgaben:

1. Tippe das unten stehende Programm ab.
2. Schreibe neben jede Zeile, was das Programm macht
3. Beseitige alle Fehler im Programm
4. Kompiliere und übertrage das Programm
5. Speicher das Programm unter deinem Namen ab.
6. Drucke das Programm aus.

```
// Der Roboter soll.....
```

```
// Name und Datum
```

```
task main          // _____
{
    motor[motorA]= 80;
    motor[motorC]= 80 // _____
    waitMsc(3000);   // _____
    motor[motorC]= -60;
    motor[motorC]= -60; // _____
    wait1Msc(3000); // _____
    motor[motor] = 0;
    motor[motorC]= 0; // _____
}
```

- Aufgaben:** 1. Lasse den Roboter vorwärts fahren für 5 Sekunden mit einer Leistung von 75 %. Die Motoren sind an A und C angeschlossen. Ändere die Werte und probiere den Roboter fahren zu lassen.
2. Der Roboter soll sich um 90⁰ nach rechts drehen. Speichere die Ergebnisse unter „drehung90.c“ ab.

Variablen und Konstanten

Wenn wir ein langes Programm haben und sehen, der Roboter fährt zu schnell, so muss man im Programm evtl. überall die Werte für die Leistung der Motoren ändern. Um dieses zu vereinfachen bedient man sich der Konstanten oder Variablen.

```
const int Leistung = 80; // Konstante wird definiert : Leistung. Leistung erhält den Wert 80.
```

```
task main()  
{  
  motor[motorA] = Leistung; //Motor A läuft mit 80% Power  
  motor[motorB] = Leistung; //Motor B läuft mit 80% Power  
  Wait1Msec(1000);  
}
```

Eine Variable repräsentiert eine Speicherstelle, deren Inhalt während der gesamten Lebensdauer der Variable jederzeit verändert werden kann. Klingt gut? Ist es auch. Neben dem Bezeichner einer Variable, muss der Typ* mit angegeben werden. Über den Typ kann der Compiler ermitteln, wie viel Speicher eine Variable im Arbeitsspeicher benötigt.

Bei einer Variablen ist der Wert (z.B. bei einem Lichtsensor)während des Programmierens noch unbekannt. Der Wert ergibt sich vielleicht in der Erprobung des Roboters.

Variablen werden so erzeugt:

Datentypname Variablenname = Wert;

*= Der Typ int besitzt laut Standard eine "natürliche Größe". Allerdings muss short kleiner oder gleich groß wie int und int muss kleiner oder gleich groß wie long sein.

Es gibt Variablentypen die typisieren welche Datentypen aufgenommen werden können. Für Festkommazahlen gibt es int, unsigned int, short, long und unsigned long. Für Zustände gibt es den Typ bool. Für Zeichen gibt es char, für Zeichenkettenstring . Für zusammengesetzte Typen (Strukturen) struct.

Beispiel:

```
int a = 13;  
int b = a + 45;
```

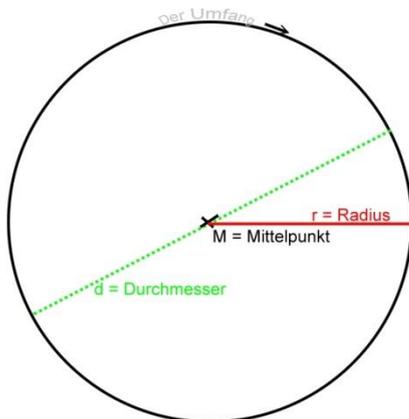
Es gibt für Variablennamen Einschränkungen:

- Das erste Zeichen muss ein Buchstabe oder ein Unterstrich sein
- Weiter dürfen Zahlen, Zeichen und Unterstriche folgen.

Durchmesser, Radius und Umfang

Radius, Durchmesser und Umdrehung. Diese drei Faktoren sind sehr wichtig in der Robotik. Soll ein Roboter besonders schnell fahren oder soll er eine Strecke mit Hindernissen (Unebenheiten auf dem Boden) meistern, ist es ein Fahrzeug für draußen (Off-Road)?

Als **Radius** (von lat. radius; „Strahl“) (deutsch: **Halbmesser**) bezeichnet man in der Geometrie den Abstand zwischen dem Mittelpunkt M eines Kreises und der Kreislinie.



Der Radius r entspricht dem halben Durchmesser d. Zum Kreisumfang U verhält sich der

Radius wie folgt: $r = \frac{U}{2\pi}$.

Bei dem NXT Rad ist das:



Der **Durchmesser** (griech. Diameter) ist die Entfernung zwischen den Schnittpunkten eines Kreises mit einer Geraden, die dessen Mittelpunkt schneidet.

Der Umfang ist die Strecke, die das Rad bei einer Umdrehung zurücklegt. Den Umfang errechnet man mit der Formel:

$$d \cdot \pi = U \quad // \quad \text{Der Durchmesser} \times \text{Pi} = \text{der Umfang}$$

Pi (π) ist eine mathematische Konstante, eine Kreiszahl mit dem unendlichen Wert:
3,14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230
78164 06286 20899 86280 34825 34211 70679 ...

Der Umfang des Rades bzw. Reifen bedeutet abgewickelt auf die Straße den zurückgelegten Weg.

Bsp.: Ein Legorad mit einem Durchmesser von 3,18 cm hat einen Umfang von 9,99 cm.

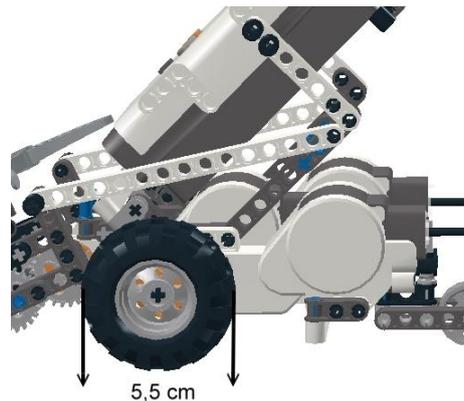
Nach einer Umdrehung hat das Rad also 9,99 cm zurückgelegt. Jetzt ein paar Aufgaben zum Üben. Die richtige Radgrößenwahl kann manchmal einen Wettbewerb entscheiden!

Aufgaben:

1. Du hast diesen Roboter:

Der Durchmesser des Rades beträgt 5,5 cm.

Wie weit ist der Roboter gefahren, wenn sich das Rad 8-mal gedreht hat?



cm

2. Du bereitest dich auf einen Wettkampf vor. Die Aufgabe lautet: Der Roboter soll eine Strecke von 10 m so schnell es geht fahren. Du hast 4 Räder mit folgenden Durchmessern zur Auswahl:

2,86 cm , 3,18 cm , 5,5 cm und 7,94 cm.

Wie viele Umdrehungen muss dein gewähltes Rad machen?

Erstelle eine Tabelle in der du die Umdrehungen vergleichen kannst.

Durchmesser	Formel	Umdrehungen
2,86 cm	_____ . π =	_____
3,18 cm		
5,5 cm		
7,94 cm		

Wenn der Roboter eine bestimmte Streckenlänge fahren soll, er das aber nicht exakt so macht, kann es verschiedene Gründe dafür geben. Manchmal hängt es vom Antrieb deines Roboters, dem Ladezustand der Batterien oder des Akku und von der Art der Oberfläche ab, auf welcher der Roboter läuft.

Hier hilft es den Durchschnitt zu errechnen.

Aufgabe:

1. Nimm deinen Roboter und lasse ihn 3 Mal drei Sekunden vorwärts fahren.

Markiere mit Kreide einen Punkt auf den Reifen und zähle die exakten Umdrehungen.



d= 5,5 cm

Was stellst du fest?

Berechne nun den Durchschnitt.

Ergebnis 1 + Ergebnis 2 + Ergebnis 3

3

Der Roboter legt in 3 Sekunden im Durchschnitt eine Strecke von _____ cm zurück.

Motoren Geschwindigkeit/ Leistungskontrolle

Wer eine Strecke genau fahren möchte, benötigt eine exakte Kontrolle der Motoren. In den Lego-NXT-Motoren ist ein Umdrehungsmesser, ein so genannter Encoder, integriert. Der Encoder zählt 360 Schritte für eine Umdrehung.

Hier zeigen sich die Vorzüge von z.B. RobotC. Alle notwendigen Funktionen sind in der Software vorhanden, werden aber von der grafischen Oberfläche (NXT-G) nicht genutzt. Für die Steuerung der Motoren gibt es sehr viele unterschiedliche Funktionen.

Für Leistung ist eine Zahl zwischen -100 und 100 anzugeben. Negative Werte bedeuten dabei eine Umkehr der Drehrichtung.

Für Versatz wird ebenfalls eine Zahl zwischen -100 und 100 verlangt. Dabei werden für 0 beide Motoren synchronisiert (geeignet z.B. für gerade Fahrt), bei 100 sowie -100 genau entgegengesetzt gedreht (jeweils andere Richtung, z.B. zum Drehen auf der Stelle). Vorsicht: Bei Werten knapp unterhalb von +50 bewegt sich nur einer der beiden Motoren, der andere bleibt nahezu regungslos.

Sensoren – Umdrehungssensor

In jedem Motor ist ein Rotationssensor eingebaut. Das Rotationssignal beinhaltet zwei Werte:

1. die Drehrichtung des Motors
2. die Anzahl der Drehimpulse.

Der NXT-Baustein erhält 360 Drehimpulse. Das entspricht 360° für eine Umdrehung.

Mit **nMotorEncoder[]** wird der Umdrehungsmesser im Motor angesprochen. Um den Motor bis zu einer bestimmten Umdrehungsanzahl laufen zu lassen, schreibt man (warten bis die Anzahl erreicht ist):

```
while (nMotorEncoder[motorA] < 1000) //siehe Seite 39
{
  Motor[motorX] = 100; // hier werden den Motoren (dem Motor) die Leistung zugewiesen
}
```

Für Fortgeschrittene:

Die Winkelstellung eines Motors am Anschluss Port kann jederzeit, also auch während der Drehung, mit **nMotorEncoder[motor]** abgefragt und mit **nMotorEncoder[motorA]=0;** auf 0 zurückgesetzt werden.

nMotorEncoder[] ist eine 16-bit Variable. Der Maximale Wert beträgt 32,767 welcher 95 Umdrehungen entspricht.

nMotorEncoder[motorA]=0; zurücksetzen Motor an Ausgang A

Man kann einem Motor eine bestimmte Anzahl von Umdrehungen zuordnen. Dieses geschieht mit der Anweisung **nMotorEncoderTarget[]**. Der Umdrehungsmesser zählt dann die Umdrehungen. Mit dieser Anweisung hat der Motor aber noch keine Leistung. Dazu braucht man die Anweisung der Leistungszuordnung für den Motor.

```
nMotorEncoderTarget[motorA] = 500; // zählt 500 Drehimpulse und stoppt
motor[motorA] = 50; // Bewegung mit 50% Leistung
```

Die Motoren drehen sich also solange, bis sie einen bestimmten Drehwinkel erreicht haben. Der nächste Befehl erlaubt es die Motoren zu synchronisieren, er bietet sich für normale gerade Fahrtstrecken an:

```
nSyncedMotors = synchType;
```

Bei

```
nSyncedMotors = synchNone; // werden die Motoren nicht synchronisiert  
und bei
```

```
nSyncedMotors = synchAC; // Motor 'C' wird mit Motor 'A' synchronisiert. Motor C ist  
dabei "Slave" und Motor A ist der "Master".
```

Um die Leistung beider Motoren synchron anzugeben verwendet man die Variable nSyncedTurnRatio. Auch hier gibt man die Werte an von -100% bis +100%.

```
nSyncedMotors = synchAC; // Motor C ist Slave von Motor A  
//  
// .....  
//  
nSyncedTurnRatio = +100;  
nMotorEncoder[motorA] = 0;  
nMotorEncoderTarget[motorA] = -1000;  
motor[motorA] = 100;  
  
while (nMotorEncoder[motorA] < 1000) // warte so lange, bis die Impulse erreicht  
sind  
{  
//  
// .....  
//  
nSyncedTurnRatio = -100;  
nMotorEncoderTarget[motorA] = 200;  
motor[motorA] = 50;
```

Die Motor-Befehle im ersten Teil setzen die Motoren nur in Gang, das Abschalten der Motoren erfolgt dann in Abhängigkeit von Sensor-Werten, oder der Zeit. Bei den folgenden Motor-Befehlen wird ein Drehwinkel vorgegeben, Dieses Verhalten ist z.B. dann nützlich, wenn man eine genau festgelegte Strecke zurücklegen möchte, oder der Roboter seine Richtung um einen bestimmten Winkel ändern soll.

Aufgabe: 1. Schreibt das obere Programm ab, kompiliert es und überträgt es auf den NXT bzw. euren Roboter. (Speichern nicht vergessen) – Was passiert?

2. An welcher Stelle dreht der Roboter? Tragt es in den Quellcode ein.

Brake und Coast/Float Modus

Die Motorengeschwindigkeit wird durch die Pulsweitenmodulation (PWM) geregelt. Die **Pulsweitenmodulation** (PWM) (auch **Unterschwingungsverfahren**) ist eine Modulationsart, bei der eine technische Größe (z. B. elektrischer Strom) zwischen zwei Werten wechselt. Dabei wird das Tastverhältnis bei konstanter Frequenz moduliert. Ein anschauliches Beispiel für diese Modulationsart ist ein Schalter, mit dem man eine Heizung ständig ein- und ausschaltet. Je länger die Einschaltzeit gegenüber der Ausschaltzeit ist, umso höher die mittlere Heizleistung. Die Temperatur der Heizung kann nur vergleichsweise langsam dem Ein- und Ausschaltvorgang folgen und ergibt so das notwendige Tiefpassverhalten zur Demodulation. (aus wikipedia.de)

Während der "Off"-Zeit wird die Leistung nicht auf die Motoren übertragen. Während der Off-Zeit der Motoren kann die Leistung entweder "freilaufend" oder "kurzgeschlossen" sein. „Kurzschluss“ führt zu einer Bremsung/Blockung. RobotC hat eine kontrollierende Variable die unterscheidet, welcher Modus verwendet wird.

bFloatDuringInactiveMotorPWM = false; // bremsen, wenn die Motoren inaktiv sind

bFloatDuringInactiveMotorPWM = true; // freilaufen, wenn die Motoren inaktiv sind

Der "Brake-Modus" ist der Bessere für den NXT und gleichzeitig der Standard-Modus.

Aufgabe: 1. Schreibe ein kleines Programm, wobei der Roboter mit einer Leistung von 75% 2 Sekunden vorwärts, bremsen, 5 Sekunden warten und dann 2 Sekunden rückwärtsfahren soll. Nun soll der Roboter stoppen aber nicht bremsen. Mache eine Markierung, von wo du gestartet bist. Was stellst du fest? Beschreibe dein Ergebnis und erkläre den Unterschied von **bFloatDuringInactiveMotorPWM "false" und "true"**.

2. Schreibe ein Programm mit synchronisierten. Lasse deiner Fantasie freien Lauf. Beobachte den Roboter und verändere die Werte.

Die Leistung der Motoren hängt z.T. von den Batterien oder der Oberfläche ab. Mit nagelneuen Batterien kann ein Motor sich mit fast 1.000 Impulsen (Encoder) pro Sekunde bewegen. Aber mit einem teilweise entladenen Motor und einem Motor, der irgendeine Abnutzung (Alter...) hat, kann sich die Höchstgeschwindigkeit auf 750 Zählimpulsen verringern. Dieses hat eine Auswirkung auf Geschwindigkeitsregelung. Mit völlig belasteten Batterien kann sich eine durchschnittliche Leistung von 75% ergeben, für die Motoren eine regulierte Geschwindigkeit von 750 Zählimpulsen pro Sekunde. Aber mit neuen Batterien kann 100% Energie genutzt werden!

Damit die Geschwindigkeitsregelung richtig funktioniert, muss dort „Raum“ sein.
nMotorPIDSpeedCtrl[.].

nMotorPIDSpeedCtrl[motorA] = mtrNoReg; // sperrt die Motordrehzahlregelung
nMotorPIDSpeedCtrl[motorA] = mtrSpeedReg; // ermöglicht die konstante
Motordrehzahlregelung

Beispiel:

```
nMotorPIDSpeedCtrl[motorA] = mtrSpeedReg; //PID Motor A an
nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg; //PID Motor C an

motor[motorA] = 50; // Motor A hält die Drehbeschleunigung mit 50% Leistung ...
auch wenn Reibung/Batteriezustand variieren
motor[motorC] = 50; // Motor C hält die Drehbeschleunigung mit 50% Leistung ...
auch wenn Reibung/Batteriezustand variieren

wait1Msec(5000);
```

Wenn die Batterien teilweise entladen werden, sollte man nicht versuchen, eine regulierte Geschwindigkeit über 750 Zählimpulsen pro Sekunde zu erzielen. Dies heißt, dass, wenn man gleich bleibende Geschwindigkeiten über Batterieniveau erzielen möchte, man entweder die Geschwindigkeiten über dem 75% Niveau nicht spezifizieren oder das Höchstgeschwindigkeitniveau verringern muss. Letzteres kann mit der folgenden Variablen eingestellt werden:

nMaxRegulatedSpeed = 750;

„Gespiegelte“ Motoren – Leistung umdrehen

Wenn man einmal einen Legoroboter baut, wo die Motoren aus baulichen Gründen anders herum eingebaut werden müssen, also wenn die Motoren sich vorwärts drehen der Roboter aber rückwärtsfährt, kann man die Drehrichtung spiegeln mit:

bMotorReflected[motorA] = true;

Um einen Wert umzudrehen gibt es auch den Modus **bMotorFlippedMode[.]**. Es gibt zwei Einstellungen: 0; ist normal und 1; ist rückwärts.

Bsp.:

```
motor[motorA]= 100; // Motor A – volle Kraft vorwärts
motor[motorC]= 100; // Motor C – volle Kraft vorwärts

bMotorFlippedMode[motorA]= 1; // „umdrehen“
motor[motorA]= 100; // Motor A – volle Kraft rückwärts
motor[motorC]= 100; // Motor C – volle Kraft vorwärts
```

Hier eine Liste der Anweisungen in der Übersicht :

motor[]

Setzt die Geschwindigkeit (-100 bis +100) für einen Motor.

nMotorPIDSpeedCtrl[]

Ermöglicht oder sperrt die Geschwindigkeitsregelung auf einem Motor. Die Regelung stellt eine exakte Steuerung über die Motordrehzahl unter Verwendung des Rückgabewertes von dem Encoder zur Verfügung.

nSyncedMotors

Synchronisiert Paare der Motoren, um die Bewegung eines „Slave“-Motors mit dem Primärmotor zu synchronisieren. Der `slaved Motor folgt der Geschwindigkeit und der Bewegung des `primary Motors.

nSyncedTurnRatio

Bestimmt die Bewegung für den Slave-Motor für das Paar der synchronisierten Motoren.

nMotorEncoder[]

Lese/Schreibe die aktuelle Position des Umdrehungsmessers.

nMotorEncoderTarget[]

Setzt den Rahmen für die Bewegung eines Motors mit Hilfe des Umdrehungsmessers (Encoders). Bsp.: der Motor soll 8 Umdrehungen machen:
8MotorEncoderTarget[motorA].
Das Ziel (target) soll sein: 8 Motorumdrehungen.

nMotorRunState[]

Enthält den Zustand (der Leerlauf, Fahrt nach vorn oder zurück, Encoder-Position während die Motoren gestoppt werden), eines Motors. Diese Variable liefert eine einfache Methode der Prüfung, wann ein Motor einen gezielten Kodiererzählimpuls erreicht hat - der Bewegungszustand sich ändert z.B. „zum Leerlauf“ oder wenn eine Position erreicht wird und Motor gestoppt worden ist.

Subroutinen

Eines der wichtigsten Ziele der Programmierung ist es, wieder verwendbare Programmteile zu erstellen um ein Programm übersichtlicher zu gestalten.

Was nützt eine Subroutine? Sobald man mehr als nur ein paar Zeilen Code in den Skripts wiederholen muss, hat man einen guten Grund, diesen Code in eine Subroutine zu packen. Das erspart Tipparbeit.

Zwei Subroutinen: "fahren" und "stop."

Über dem "task main ()" wird folgendes eingegeben:

```
void fahren()
{
  motor[motorD] = 100;
  motor[motorE] = 100;
  wait1Msc(2000);
}

void stop()
{
  motor[motorD] = 0;
  motor[motorE] = 0;
}

task main()
{
  fahren();
  stop();
}
```

Mit Übergabe eines Wertes:

```
void fahren(int power)
{
  motor[motorD] = power;
  motor[motorE] = power;
  wait1Msc(2000);
}

task main()
{
  fahren(50);
  stop();
}
```



Sensoren – Ultraschallsensor

Delphine benötigen ihn zur Navigation, Kommunikation, Lokalisierung von Fischen, Fledermäuse zur Navigation und Lokalisierung von Beutetieren. Der Ultraschallsensor.

Mit **Ultraschall (US)** bezeichnet man Schall mit Frequenzen, die oberhalb des vom Menschen wahrgenommenen Bereiches liegen. Das umfasst Frequenzen zwischen etwa 20 kHz und 1...10 GHz. Der Vorteil eines Ultraschallsensors besteht darin, dass diese Sensoren ohne eine Abtastung des zu vermessenden Objektes materialunabhängig funktionieren. Der Ultraschallsensor ist der einzige digitale Sensor, er besitzt einen integrierten Mikrocontroller, der alle Ultraschallmesswerte zum NXT-Baustein per I2C-Kommunikation sendet.

Wie funktioniert das?

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 gelieferten 9 Volt, daher verringert sich die Messgenauigkeit mit abnehmender Batteriespannung.

Um ihn als solchen zu erkennen benutzt man folgende Funktion:

```
SetSensorType(S1, sonarSensor); // Setzt Port 1 als Ultraschallsensor
```

Um die Werte des Sensors abzurufen geht man folgendermaßen vor:

```
SensorValue(sensor_input) // der Wert, den der Sensor liest  
ClearSensorValue(S1); //setzt den Wert zurück
```

Aufgabe: Probiere was das folgende Programm macht. Schreibe die Erklärung auf die Linie. Tippe das Programm ab und übertrage es auf deinen Roboter. Achte darauf, dass der Sensor und die Motoren am richtigen Ausgang/Eingang angeschlossen sind.

```
task main()  
{  
    int x=0;    //Die Variable "x" ist als integer Wert deklariert und initialisiert als  
               gleich Null  
    while(true) //eine unendliche while-Schleife ist dem Wert wahr zugewiesen  
        x = SensorValue(sonarSensor); //Die Variable „x“ ist der Wert des Sonar Sensors  
        nxtDisplayTextLine(1,"Ultraschall liest:%d",x); // " Ultraschall liest:" .....
```

```
wait1Msec(1000);           //.....  
eraseDisplay();           //.....  
}  
}
```

Aufgabe: Schreibe nun ein Programm für eine Alarmanlage. Sobald ein Hindernis in 25 cm Nähe des Sensors kommt, soll ein lautes Signal ertönen.

Sensoren – Lichtsensor

Der Lichtsensor ist ein analoger Sensor.



Es gibt zwei verschiedene Arten (Modus) den Lichtsensor zu nutzen:

- im Reflected Light Modus wird eine integrierte LED aktiviert, welche das Messen von Helligkeitswerten auch bei schwachem Umgebungslicht möglich macht und
- den Ambient Light-Modus (ohne LED).

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.

Der Lichtsensor erkennt keine Farben, kann diese aber „erahnen“.



Er misst die Helligkeit im Bereich von 0 bis 100%.

Angesprochen wird der Sensor mit:

```
SetSensorType(S2, lightSensor); // Setzt Port 2 als Lichtsensor
```

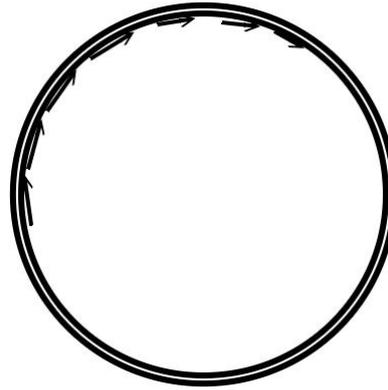
Um die Werte des Sensors abzurufen geht man folgendermaßen vor:

```
SensorValue(sensor_input) // der Wert, den der Sensor liest  
ClearSensorValue(S2); //setzt den Wert zurück
```

Wir benötigen auch die Testauflage mit der schwarzen Linie, die mit dem NXT Satz ausgeliefert wird und einen Roboter mit Lichtsensor. Dieser sollte möglichst nah über den Boden angebracht werden. Das Grundprinzip der „Linie folgen“ ist, dass der Roboter versucht, auf dem Rand der schwarzen Linie zu bleiben und sich von der hellen Umgebung abwendet.

```
task main()
{
  while(true)
  {
    if(SensorValue(lightSensor) < 45)
    {
      motor[motorA] = -75;
      motor[motorB] = 0;
    }

    {
      motor[motorA] = 0;
      motor[motorB] = -75;
    }
  }
}
```



- Aufgabe:**
1. Schreibe ein Programm wobei der Roboter einer schwarzen Linie folgt, aber gegen den Uhrzeigersinn fährt.
 2. Baue einen Roboter, der sich nur innerhalb eines mit dunklen Rand begrenzten Feldes bewegt.
 3. Schreibe ein Programm, welches verhindert dass dein Roboter über die Tischkante hinausfährt und damit vom Tisch fällt.

Soll die interne Lichtquelle nicht aktiviert sein, so muss man sie mittels

```
SetSensorType(S3, lightSensorInactive);
```

ausgeschaltet werden.

SensorType []

Diese Lese / Schreib-Variable dient zur Konfiguration der Art eines Sensors (Touch-, Licht-, Sonar, etc.)

SensorMode []

Diese Lese / Schreib-Variable wird verwendet, um den Modus eines Sensors zu konfigurieren.

SensorValue []

Diese Variable enthält die aktuelle normalisierten Werte des Sensors.

SensorRaw []

Diese Variable enthält die aktuelle UN-normalisierten Werte des Sensors. (Sensoren von Hitechnic oder mindsensors)

Lichtmessung mit dem NXT?

Aufgabe: Ermittle die Sensorwerte für die Farbe rot, blau und gelb. Trage die Ergebnisse in die Tabelle ein und vergleicht eure Werte (Sensor an Port 2). Die Werte sollen auf dem Display des NXT ausgegeben werden.

Farbe	rot	blau	gelb	weiß	schwarz
Wert					

Der Temperatur Sensor

Der NXT Temperatursensor ist ein digitaler Sensor und dient zur Messung von Temperaturen in Celsius und Fahrenheit (-20 °C to +120 °C/-4 °F to +248 °F).

Der Sensor verfügt über einen 10cm langen metallischen Messfühler zur Bestimmung von Temperaturen in Flüssigkeiten.

```
#pragma config(Sensor, S1, LEGOTMP, sensorI2CCustom)
```



Sensoren – Tastsensor



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

Der Tastsensor des NXT erkennt nur zwischen 2 Zuständen.

- 0 = nicht gedrückt
- 1 = gedrückt

Angesprochen wird der Sensor durch:

```
SetSensorType(S2, touchSensor); // Setzt Port 2 als Tastsensor
```

Um die Werte des Sensors abzurufen geht man folgendermaßen vor:

```
SensorValue(sensor_input) // der Wert, den der Sensor liest
```

```
ClearSensorValue(S2); //setzt den Wert zurück
```

Aufgabe: Schließe den Tastsensor an Port 3. Der Roboter soll **so lange** geradeausfahren, **bis** er gegen ein Hindernis stößt. Versuche das Programm zu erarbeiten, indem du den Text unten durcharbeitest.

- Schreibe ein neues Programm.
- Speichere es unter dem Namen „tastsensor. c“ ab.
- Definiere den Sensortyp an Port 3.
- Der Roboter soll nun vorwärtsfahren, **so lange bis** der Taster gedrückt wird.
- Der Befehl „**do-while**“ ist sehr leistungsfähig. Der Programmablauf wird an dieser Stelle unterbrochen, bis die Bedingung innerhalb der Klammern () zutrifft.
- Die Bedingung ist, das der Wert von Sensor_3 == 1 sein muss. Das bedeutet, dass der Sensor gedrückt wurde. Wird der Sensor nicht gedrückt ist der Wert = 0. Trifft der Roboter auf einen Gegenstand und der Tastsensor wird gedrückt, die Bedingung wird also wahr, wird das Programm fortgesetzt und der Roboter soll stoppen.

Wie sieht der Quelltext aus? Schreibe das Programm.

```
task main ( )  
{  
  Programm  
}
```



Sensoren – Geräuschsensor

Der Geräuschsensor, Tonsensor oder Soundsensor ist dem Lichtsensor in der Handhabung sehr ähnlich. Dieser analoge Sensor misst Schalldruck entweder in dB oder in dBA.

Es können Werte im Bereich von 55 dB bis 90 dB gemessen werden. 90 dB haben Autohupen oder sind LKW-Fahrgeräusche.

Dieser Sensor kann also nur hören wie laut etwas ist.

Der Zusatz A gibt an, dass die unterschiedlichen Tonfrequenzen ähnlich dem menschlichen Hörempfinden unterschiedlich bewertet werden, d.h. mittlere Frequenzen (Spitze bei 2 kHz) werden stärker berücksichtigt. Der allgemeine Frequenzbereich reicht von 20 Hz bis 18 kHz.

Schalldruckpegel sind extrem schwierig zu messen, also werden die Sensor-Messwerte auf dem MINDSTORMS NXT in Prozent angezeigt [%].

- 4-5% ist wie ein leises Wohnzimmer
- 5-10% Menschen die in einem Abstand miteinander sprechen
- 10-30% ist normales Gespräch - nah an dem Sensor - oder Musik, die mit einer „normalen“ Zimmerlautstärke gespielt wird
- 30-100% ist das Schreien von Menschen oder Musik, die mit hoher Lautstärke gespielt wird.

Um ihn als Geräuschsensor zu definieren benutzt man folgende Anweisung:

```
SetSensorType(S4, soundSensor); // Setzt Port 4 als Geräuschsensor
```

Um die Werte des Sensors abzurufen geht man folgendermaßen vor:

```
SensorValue(sensor_input) // der Wert, den der Sensor liest  
ClearSensorValue(S4); //setzt den Wert zurück
```

Der Roboter soll beim Klatschen stehen bleiben.

```
task main()  
{  
    wait1Msec(1000); //Das Programm wartet 1 Sekunde vor dem Start  
  
    while(SensorValue(soundSensor) <= 50) //eine while-Schleife, die auf einen Wert  
von 50 und kleiner wartet bis zur weiteren Programmausführung  
    {  
        motor[motorA] = 75; //Motor A mit 75% Leistung  
        motor[motorB] = 75; // Motor A mit 75% Leistung  
    }  
}
```

- Aufgaben:**
1. Nutze den NXT als Messgerät. Lege eine Tabelle mit verschiedenen Geräuschen und der Lautstärke in dB.
 2. Schreibe ein Programm wobei der Roboter 5 Sekunden lang vorwärts fahren soll, wenn ein Mal laut geklatscht wird.

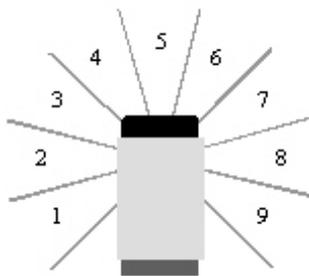
Sensoren – andere Sensoren

Die neuen Sensortypen findet man in der BricxCC Oberfläche unter dem Punkt DIRECT CONTROLLER.

Am Beispiel des **Infrarot-Sucher Sensor** von der Firma Hitechnic soll gezeigt werden, wie andere Sensoren angesprochen werden. Der HiTechnic-Infrarotdetektor ermöglicht das Registrieren und Orten von Infrarot-Lichtquellen.

Um ihn als Sensor zu definieren benutzt man folgende Anweisung:
SensorUS(IN_1)

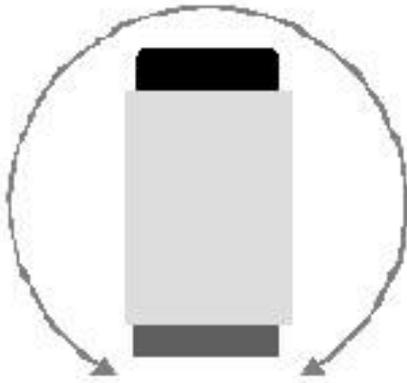
Um den IR „Seeker“ zu testen, stecke den Anschluss in Port 1 und wähle auf dem NXT Baustein „View“, Ultraconic cm, Port 1. Wenn du den Infrarotsensor bewegst, ändern sich die Werte auf dem Display von 1-9. Auf der Grafik siehst du, welcher Bereich welchen Wert



hat. 0 heißt, dass der Sensor kein Signal empfängt. Beim Roboterfußball sendet der spezielle Ball vom RoboCupJunior Infrarotstrahlen. Der Sensor kann diese empfangen. Vorsicht: Sonnenlicht beeinflusst die Werte.

```
task main()
{
  SetSensorLowspeed(IN_1);
  OnFwd(OUT_BC, 70);

  while(true)
  {
    while(SensorUS(IN_1)<=4);
    Off(OUT_C);
    OnFwd(OUT_B, 100);
    Wait(100);
  }
}
```



Der **Gyro-Sensor** ist von der Art her wie der Schall-Sensor. Er gibt einen Wert zwischen -360 und $+360^\circ$ aus. Dieser Wert wird alle $0,1$ Sekunden aktualisiert. Er gibt aber nicht die Neigung an sondern die Drehgeschwindigkeit. Dieser Wert gibt nur an, um wie viel $^\circ$ sich der Sensor in 1 Sekunde dreht. Wenn dieser Wert also bei 0° , in der nächsten Messung bei 50° und dann wieder bei 0° liegt, dann hat sich der NXT um $50^\circ/10$ (Wegen 10 Messungen pro Sekunde) um 5° gedreht.

Ein Gyrosensor wird so eingesetzt:

Bsp.:

```
#define GYRO_OFFSET 4
```

```
task main()
{
  SetSensorHTGyro(S1);
  until(false)
  {
    NumOut(0, LCD_LINE1, SensorHTGyro(S1,GYRO_OFFSET), true);
    Wait(100);
  }
}
```

Angaben zum Einsatz der Sensoren finden sich auf der Herstellerseite.

Problem: Der Sensor baut immer einen kleinen Messfehler ein. Dieser liegt bei ca $+5$. Da das aber nur ca. gesagt ist kann man ihn nicht komplett ausblenden. Das ist schade.

Der Kompass-Sensor

Der NXT Kompass-Sensor ist ein digitaler Kompass, der das magnetische Erdfeld misst und einen Wert ausgibt, der die gegenwärtige Orientierung darstellt. Die magnetische Orientierung wird zum nächsten 1° errechnet und als Zahl von 0 bis 359 zurück gegeben.

Es ist aber darauf zu achten, dass er möglichst eben eingebaut wird, sonst ergeben sich unbrauchbare Werte.

Bsp.:

```
task main()
{
  SetSensorLowSpeed(S2);
  until(false)
  {
    NumOut(0, LCD_LINE1, SensorHTCompass(S2), true);
    Wait(100);
  }
}
```

Der neue RGB Farb- und Lichtsensor

Der RGB Farbsensor kann drei Funktionen erfüllen: Als Farbsensor kann er zwischen sechs Farben unterscheiden, als Lichtsensor bestimmt er die Helligkeit von direktem Licht oder Umgebungslicht, und als Farblampe leuchtet er rot, grün oder blau.

Wie bei jedem anderen Sensor muss erst festgelegt werden, an welchem Eingang der Farbsensor angeschlossen ist. Dazu dient der Befehl:

```
SetSensorColorFull(IN_1); // Sensor ist an 1 angeschlossen
```

Der Wert des Sensor kann dann wie beim Tastsensor abgerufen werden:

```
int x = SENSOR(IN_1);
```

x kann dabei folgende Werte („Farben“) annehmen:

1 = Schwarz, 2 = Blau, 3= Grün, 4= Gelb, 5= Rot,

6= Weiß

Es können die drei LEDs des Farbsensors auch einzeln angesprochen werden und somit verschiedene Farben erzeugt werden. Die Befehle zum ansteuern der LEDs lauten:



Programm für den Farbsensor:

```
SetSensorColorRed(IN_1); //rot
SetSensorColorGreen(IN_1); //grün
SetSensorColorBlue(IN_1); //blau
SetSensorColorNone(IN_1); //alle aus!
```

Bsp.:

```
#include "NXCDefs.h"

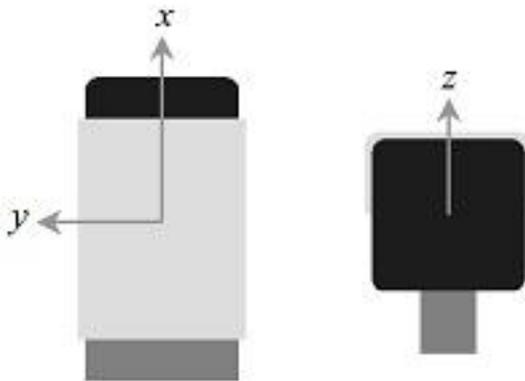
task main()
{
  // Sensor
  SetSensorColorFull(IN_1);
  while(true)
  {
    while(SENSOR_1 == 2) //blau
    {
      OnFwd(OUT_AC, 80);
    }
    while(SENSOR_1 == 5) //rot
    {
      OnRev(OUT_AC, 80);
    }
  }
}
```

(nach Daniel Braun)

Beschleunigungs- bzw. Tilt Sensor

Der HiTechnic Beschleunigungs- bzw. Tilt Sensor misst die Beschleunigung in drei Achsen. Es misst auch kippen entlang jeder Achse. Mit dem Sensor kann man die Beschleunigung des Roboters im Bereich -2g bis + 2g mit einer Skalierung von etwa 200 Zählungen pro g messen. Die Beschleunigung Messung für jede Achse wird etwa 100 mal pro Sekunde aktualisiert.

Die drei Achsen der Messung markiert sind x, y und z wie abgebildet.



Beispiele:

- Beim freien Fall im Schwerfeld der Erde beträgt die Beschleunigung $g = 9,81 \text{ m/s}^2$
- Auf der Achterbahn Silverstar im Europa-Park herrschen Vertikalbeschleunigungen von bis zu 4 g (40 m/s^2).

Der Beschleunigungssensor kann auch zur Neige in drei Achsen gemessen werden. Dies ist möglich, weil die Schwerkraft als Beschleunigung wahrgenommen wird.

Die aktuelle Version von NXC enthält eine API-Funktion zum Auslesen der HiTechnic Beschleunigungssensors.

```
bool ReadSensorHTAccel (const Byte-Port, int & x, int & y, int & z)
```

```
// HiTechnic Beschleunigungs-Sensor angeschlossen an Port 1
```

```
task main()
```

```
{
```

```
  int x,y,z;
```

```
  SetSensorLowspeed ( S1 );
```

```
  Wait ( 50 );
```

```
while ( true )
{
  ReadSensorHTAccel ( S1 , x, y, z);
  TextOut ( 0 , LCD_LINE1 , "x: " );
  NumOut ( 6 * 2 , LCD_LINE1 , x);
  TextOut ( 0 , LCD_LINE2 , "y: " );
  NumOut ( 6 * 2 , LCD_LINE2 , y);
  TextOut ( 0 , LCD_LINE3 , "z: " );
  NumOut ( 6 * 2 , LCD_LINE3 , z);

  Wait ( 100 );
}
}
```



Winkelsensor

Die HiTechnic Winkelsensor wird mit einer Standard LEGO Technik Kreuzachse verbunden und misst drei Rotation einer Achse besitzenden.

Diese sind

- Absolute Winkel: der Drehwinkel einer Achse von 0 Grad bis 359 Grad, Genauigkeit 1 Grad.
- Kumulierte Winkel: die kumulierte Anzahl der Grade die die Achse gemacht hat.
- Umdrehungen pro Minute (RPM): Die aktuelle Schätzung der Drehrate von 1 U / min bis 1.000 U / min.

Der Winkelsensor ist ideal für den Einbau in komplexe Modelle, wo die Winkel oder die Drehzahlüberwachung gewünscht ist z.B. eine Wetterstation (Windfahne) oder ein Messrad.

Beispielprogramm von Hitechnic:

```
//=====
// HiTechnic - Angle Sensor Sample Program
//
#define ANGLE S1
//=====
// ReadSensorHTAngle(port, Angle, AccAngle, RPM)
// Reads the HiTechnic Angle Sensor and returns the current:
// Angle          degrees (0-359)
// Accumulated Angle degrees (-2147483648 to 2147483647)
// RPM            rotations per minute (-1000 to 1000)
void ReadSensorHTAngle(int port, int &Angle, long &AccAngle, int &RPM)
{
```

```

int count;
byte cmndbuf[] = {0x02, 0x42}; // I2C device, register address
byte respbuf[]; // Response Buffer
bool fSuccess;
count=8; // 8 bytes to read
fSuccess = I2CBytes(port, cmndbuf, count, respbuf);
if (fSuccess) {
    Angle = respbuf[0]*2 + respbuf[1];
    AccAngle = respbuf[2]*0x1000000 + respbuf[3]*0x10000+
        respbuf[4]*0x100 + respbuf[5];
    RPM = respbuf[6]*0x100 + respbuf[7];
} else {
    // No data from sensor
    Angle = 0; AccAngle = 0; RPM = 0;
}
}

// ReserSensorHTAnalog(port, resetmode)
// resetmode:
// HTANGLE_MODE_CALIBRATE Calibrate the zero position of angle.
// Zero position is saved in EEPROM on sensor.
// HTANGLE_MODE_RESET Reset the rotation count of accumulated.
// angle to zero. Not saved in EEPROM.
#define HTANGLE_MODE_CALIBRATE 0x43
#define HTANGLE_MODE_RESET 0x52
void ResetSensorHTAngle(int port, int resetmode)
{
    int count;
    byte cmndbuf[] = {0x02, 0x41, 0}; // I2C device, register address
    byte respbuf[]; // buffer for inbound I2C response
    cmndbuf[2] = resetmode; // Set reset code
    count=0; // 0 bytes to read
    I2CBytes(port, cmndbuf, count, respbuf);
    if (resetmode == HTANGLE_MODE_CALIBRATE)
        Wait(50); // Time to allow burning EEPROM
}

//=====

task main()
{
    int angle;
    long acc_angle;
    int rpm;

    SetSensorLowspeed(ANGLE);
    Wait(100);

```

```
TextOut(0, LCD_LINE1, "HiTechnic");
TextOut(0, LCD_LINE2, " Angle Sensor");
TextOut(0, LCD_LINE8, "");

while(true) {
  Wait(100);

  ReadSensorHTAngle(ANGLE, angle, acc_angle, rpm);
  TextOut(0, LCD_LINE4, "Angle :   ");
  NumOut(7*6, LCD_LINE4, angle);

  TextOut(0, LCD_LINE5, "AccAng:   ");
  NumOut(7*6, LCD_LINE5, acc_angle);

  TextOut(0, LCD_LINE6, "RPM :   ");
  NumOut(7*6, LCD_LINE6, rpm);

  if (ButtonPressed(BTNLEFT,0)) {
    while(ButtonPressed(BTNLEFT,0));
    ResetSensorHTAngle(ANGLE, HTANGLE_MODE_RESET);
  }
  if (ButtonPressed(BTNRIGHT,0)) {
    while(ButtonPressed(BTNRIGHT,0));
    ResetSensorHTAngle(ANGLE, HTANGLE_MODE_CALIBRATE);
  }
}
```

<http://www.hitechnic.com>

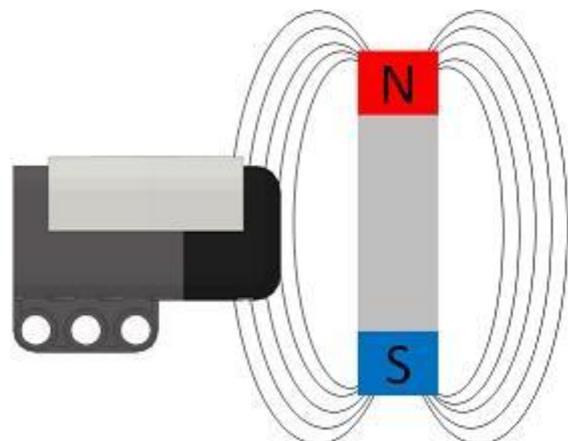
Magnetsensor

Der NXT Magnetsensor ermöglicht es mit dem Roboter Magnetfelder zu finden. Der Sensor erkennt Magnetfelder, die auf der ganzen Vorderseite des Sensors in einer vertikalen Ausrichtung sind.

Siehe Abbildung

Wenn die Ausrichtung des Magnetfeldes „Norden“ wie in der Abbildung oben ist, wird das Magnetfeld erkannt. Wird der Magnet seitlich gestellt, kann es nicht nachgewiesen werden.

Der Sensor kann bis zu etwa 300 mal pro



Sekunde gelesen werden.

Test: SchlieÙe den Sensor an Ausgang 1 an und drücke „View – Ambient light – Port 1“ auf dem NXT. Halte einen Magneten vor den Sensor und beobachte, was passiert.

Der Magnetsensor hat das gleiche Interface wie der Gyro-Sensor.

```
#define SensorHTMagnet(port,offset) SensorHTGyro(port,offset)
#define SetSensorHTMagnet(port) SetSensorHTGyro(port)

#define MAGNET IN_1

task main()
{
  int offset, magnetic_value;

  SetSensorHTMagnet(MAGNET);

  TextOut(0, LCD_LINE1, "HiTechnic");
  TextOut(0, LCD_LINE2, " Magnetic Sensor");

  TextOut(0, LCD_LINE4, "Calibrating, ");
  TextOut(0, LCD_LINE5, "keep magnets");
  TextOut(0, LCD_LINE6, "away...");
  Wait(1000);
  //Get inital offset assuming no magnetic field is present.
  offset = SensorHTMagnet(MAGNET, 0);
  TextOut(0, LCD_LINE4, " Value: ");
  TextOut(0, LCD_LINE5, " ");
  TextOut(0, LCD_LINE6, " ");

  while(true) {
    magnetic_value = SensorHTMagnet(MAGNET, offset);

    TextOut(6*7, LCD_LINE5, " ");
    NumOut(6*7,LCD_LINE5, magnetic_value);

    Wait(100);
  }
}
```

EOPD Sensor

HiTechnic-Abstandssensor, EOPD-Sensor (Electro Optical Proximity Detector)

Dieser optisch-elektronische Sensor kann Objekte bis zu einer Entfernung von 20cm sowie kleine Distanzänderungen präzise erkennen. Damit schließt er die Lücke zwischen dem Ultraschall- und dem Berührungssensor. Durch sein pulsiertes Lichtsignal können externe Lichteinflüsse ausgeblendet werden.



In zwei Modi kann er auch auf helle und dunkle Objekte justiert werden.

Der Sensor kann mehr als 300 mal pro Sekunde eingelesen werden und reagiert schnell auf Änderungen.

Jeder EOPD läuft auf einer leicht anderen Samplingrate sodass es auch untereinander kaum Interferenzen gibt.

Der EOPD Sensor ist ein analoger Sensor wie die LEGO Lichtsensor

```
# Define EOPD IN_3 //Sensor in Port 3
```

```
task main()
{
  int eopd;
  ...
```

Auszug für die Werteausgabe auf dem NXT:

```
...
while(true) {
  // EOPD Sensor lesen
  eopd = 1023-SensorRaw(EOPD);

  TextOut(0, LCD_LINE4, "eopd: ");
  NumOut(6*5, LCD_LINE4, eopd);

  iSteer = EOPD_TARGET - eopd;
  ...
```

Kontrollstrukturen – if-Anweisung

Kontrollstrukturen dienen dazu, ein Problem übersichtlich und strukturiert zu formulieren. Die wichtigsten Kontrollstrukturen sind die Auswahl (Entscheidungen) und die Schleifen. Dazu kommen noch einige andere Kontrollanweisungen wie z.B. Sprünge. Die **if**-Anweisung führt eine Programmierzeile oder einen Anweisungsblock nur dann aus, wenn eine bestimmte Bedingung erfüllt ist (true = wahr). Die Bedingung könnte etwa sein, ein Motor laufen soll, wenn der Tastsensor gedrückt wurde. Wenn also diese Bedingung – `sensorValue(touch) == 1` – erfüllt ist, dann soll etwas geschehen, nämlich der Motor soll laufen. Was geschieht, ist die Ausführung einer oder mehrerer Anweisungen (Zeilen). Die **if**-Anweisung hat in ihrer einfachsten Form folgende Syntax:

```
if (Bedingung)
    Anweisung;
```

In dieser Form wird genau eine Anweisung ausgeführt, wenn die Bedingung, die zwischen den Klammern steht, erfüllt ist.

```
if(sensorValue(touch) == 1)
{
    motor[motorA]= 80;
}
```

Wichtig: Die Anweisung endet mit dem Semikolon (;). Auch wenn nur das Semikolon vorhanden wäre, wäre das eine gültige **if**-Anweisung. Ist die Bedingung nicht erfüllt, geht die Programmausführung normal weiter.

Sollen mehrere Anweisungen bei erfüllter Bedingung ausgeführt werden, müssen diese zu einem Anweisungsblock zusammengefasst werden.

```
if(sensorValue(touch) == 1)
{
    motor[motorA]= 80;
    motor[motorB]= 70;
    motor[motorC]= 60;
}
```

Die **if**-Anweisung kann noch um einen **else**-Block erweitert werden, d.h. eine Ersatzanweisung wird hinzugefügt. Ist die **if**-Bedingung erfüllt, wird die Anweisung ausgeführt, wenn nicht, wird die Ersatzanweisung nach dem **else** ausgeführt.

```
if(sensorValue(touch) == 1)
{
    motor[motorA]= 80; //mehrere Anweisungen möglich
}
else
{
    motor[motorA]= 0; //mehrere Anweisungen möglich
}
```

Kontrollstrukturen – while-Schleife

Eine weitere Schleife ist die **while**-Schleife. Sie wird so lange durchlaufen, solange die angegebene Bedingung erfüllt ist:

```
while (Bedingung)
    Anweisung;
```

Sollen mehrere Anweisungen ausgeführt werden, wie üblich mit Anweisungsblock:

```
while (Bedingung)
{
    Anweisung1;
    Anweisung2; //mehrere Anweisungen möglich
}
```

Der Anweisungsblock wird nur dann ausgeführt, wenn die Bedingung erfüllt ist (wahr). Das gilt auch für die erste Ausführung! Das heißt, ist die Bedingung bereits zu Beginn nicht erfüllt, werden die Anweisungen im Schleifenrumpf überhaupt nie ausgeführt.

Beispiel:

```
while(time1[T1]<5000) //während der Timer T1 kleiner (weniger) als 5 Sekunden zählt...
{
    motor[motorA]= 80; // ... Motor A läuft mit 80% Leistung
}
```

Beispiel für eine Endlosschleife. Der Wert wird ständig abgefragt.

```
while(true)
{
    eraseDisplay();
    x = SensorRaw[S2];
    nxtDisplayTextLine(0, "%4d", x);
    wait1Msec(100);
}
```

Zählen in der while Schleife

Eine Anweisung soll mehrfach ausgeführt werden:

```
int i = 0;
while (i<4)
{
    fronthoch();
    wait1Msec(1200);
    frontherab();
    wait1Msec(1200);
    i++;
}
```

Do-While-Schleife

Im Gegensatz zur while -Schleife findet bei der do-while-Schleife die Überprüfung der Wiederholungsbedingung am Schleifenende statt. So kann garantiert werden, dass die Schleife mindestens einmal durchlaufen wird. Sie hat die folgende Syntax:

```
do
    Anweisung
while (Ausdruck);
```

Beispiel:

```
int main()
{
    int i = 1;

    do
    {
        printf ("%d\n", i);
        i++;
    }
    while (i <= 10);

    return 0;
}
```

for-Schleife

Die for-Schleife ist eine **Zählschleife**. Man benötigt zunächst eine Variable, die mitzählt, wie oft die Schleife bereits durchlaufen wurde: Die **Zählervariable**. Sie wird zu Beginn, wenn *aufwärts* gezählt werden soll, häufig mit 0 (Null) initialisiert. Hier ist aber auch 1 oder jeder andere Wert möglich. In der Informatik wird oft bei 0 (und nicht 1) zu zählen begonnen, daher ist 0 gebräuchlich.

Ein Beispiel wäre:

```
for (i = 1; i <= 5; i++)
```

Als Zählervariable wurde *i* verwendet, was in der Programmierung üblich ist.

Das Beispiel zählt von 1 bis 5 und gibt die Werte untereinander aus. Die Variable *i* ist der Schleifenzähler und muss vor der Ausführung bereits deklariert worden sein:

```
int i;
```

Robot C und Tetrrix

Um den NXT mit Servomotoren oder DC Motoren anzusteuern, wird kein ganzer Baukasten benötigt. Das Herzstück bilden beide Controller:

DC Motor Controller



und der Servo Controller.



Die Controller (<http://www.educatech.ch>), Metallteile und Servozubehör können einzeln gekauft werden. Z.B. bei <http://www.lynxmotion.com>

Ein 12V 3000mah NIMH Akku kann bei z.B. Conrad erworben werden.

Das TETRIX Education Basis-Set enthält alles was man braucht, um TETRIX-Roboter aus Metall zu bauen, die sich über den intelligenten LEGO MINDSTORMS NXT Baustein steuern lassen. Dazu gehört auch eine von der Carnegie Mellon Robotics Academy entwickelte CD-ROM, die in die Nutzung des TETRIX-Konstruktionssystems einführt und Neueinsteiger mit den wichtigsten Hardware- sowie ausgewählten Programmierungsgrundlagen vertraut macht. Außerdem sind im Set enthalten:

- TETRIX-Konstruktionssystem
- TETRIX Hard Point Connectors
- HiTechnic DC-Motor-Controller
- HiTechnic Servo-Controller
- 12V wiederaufladbarer NiMh-Akkupack

Pitsco, eine renommiertes US-amerikanischer Anbieter für Bedarf für den wissenschaftlichen Schulunterricht, bietet nun das TETRIX-System für jedermann an; es ist zwar ein großes Stück teurer als ein NXT-Kit, dürfte aber dennoch für jeden interessant sein, der schon mal über Roboter jenseits der Möglichkeiten des NXT-Set nachgedacht hat.



Diese Erweiterung zu dem TETRIX™ Basic Kasten ermöglicht das Bauen von größeren und komplexeren Robotern, sowie anspruchsvollere Engineering-Projekte. Dieses Set enthält 40 Bauelemente aus Metall und wird komplett mit einer Sortierbox geliefert.



TETRIX™ von Pitsco ist eine hervorragende Ergänzung für jedes LEGO® MINDSTORMS® NXT Robotik-Klassenzimmer. Dieses neue Metallbausystem wurde durch den Einsatz des innovativen Hard Point Connectors speziell für die Verbindung mit dem LEGO Technic-Bausystem entwickelt. TETRIX, in Kombination mit neuen benutzerdefinierten Steuergeräten, erlaubt, die Leistung der MINDSTORMS-Technologie zu integrieren und leistungsstarke Gleichstrom- und Servomotoren sowie Metallgetriebe zu steuern. Man kann jetzt noch vielseitigere und robustere Roboter bauen, die für differenziertere Aufgaben vorgesehen sind, während grundlegende Verkabelungen, Multi-Motorsteuerungen und vieles mehr beherrscht werden. *(aus der Pitsco Werbung)*

RobotC und Tetrrix starten



Zu Beginn wählt man, nach dem starten der Software, in dem Programm aus, ob man einen Roboter oder die virtuelle Welt programmieren möchte.

Dann sollte man alle Motoren, Servos und Sensoren definieren. Das ist wie die Definition der "Objekte" in einem Programm vor dem Erstellen der Methoden. RobotC hat zwei Menüs um diese Konfiguration zu erreichen (siehe Bilder).

Durchführung:

RobotC wird gestartet und File-New ausgewählt, um ein neues Programm zu starten.

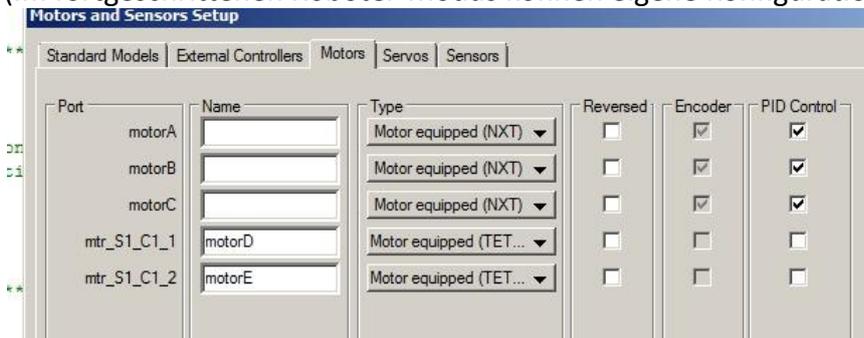
Unter "Robot-> Platform Type" wird aus der Menüleiste "NXT + Tetrrix" ausgewählt.



Anschließend klickt man auf "Robot-> Motoren und Sensoren Setup"

Unter "TETRIX Controller" wird die zweite Option: "Standard Konfiguration Ein Motor Regler, einen Servo-Controller auf Sensor-Anschluss S1" gewählt.

(Im fortgeschrittenen Roboter-Modus können eigene Konfiguration erstellt werden)



Auf "Übernehmen" klicken.

Unter der "Motoren"-Registerkarte werden die Motoren wie oben dargestellt benannt. Per Konvention Motoren "A, B und C" sind die NXT-Motoren, Motoren "D und E" sind die Tetrrix DC Motoren (12V). "Reversed" für Motoren anklicken, um die Motoren in beide Richtung

drehen zu lassen. Wird ein Encoder verwendet, so sollte man auch hier ein Häkchen setzen. Die Motoren können frei benannt werden. Das macht den Code später leichter zu verstehen.

"Übernehmen".

Bei den Servos geht man genauso vor. Unter Sensoren werden dieses dementsprechend ausgewählt.

Die ersten 10 Zeilen des Programms enthalten jetzt einen automatisch generierten Code, der die Hubs konfiguriert, Sensoren, Motoren und Servos (die "pragma config"). Diese definieren die Roboter Objekte im Programm.

Um Tetrrix nutzen zu können müssen diese „pragma Statements“ eingebunden sein.

Basis:

```
#pragma platform(Tetrrix)
```

Motoren:

```
#pragma config(Motor, mtr_S1_C1_1, motorD, tmotorNormal, openLoop)
#pragma config(Motor, mtr_S1_C1_2, oderIrgendEinName, tmotorNormal, openLoop)
```

Motor D und E werden angesprochen an 1 und 2.

Für die Servomotoren gilt:

```
#pragma config(Servo, srvo_S1_C2_1, irgendeinName, tServoNormal)
```

Für die Sensoren:

```
#pragma config(Sensor, S2, touchSensor, sensorTouch)
#pragma config(Sensor, S3, sonarSensorL, sensorSONAR)
```

Ein einfaches Programm um einen Tetrrix Roboter fahren zu lassen:

```
task main()
{
    motor[motorA] = 30; // Konstante
    motor[motorD] = 30;
    wait1Msec(1000);
}
```



Servomotoren

Hitec HS-475HB

Modulation:	Analog
Drehmoment:	4.8V: 61.0 oz-in (4.39 kg-cm) 6.0V: 76.0 oz-in (5.47 kg-cm)
Geschwindigkeit:	4.8V: 0.23 sec/60° 6.0V: 0.18 sec/60°
Gewicht:	1.41 oz (40.0 g)
Abmessungen:	Länge: 1.52 in (38.6 mm) Breite: 0.77 in (19.6 mm) Höhe: 1.41 in (35.8 mm)
Motortyp:	3-polig
Getriebe:	Plastik
Rotation/Support:	Single Bearing
Drehbereich:	200°
Pulse Cycle:	20 ms
Pulse Width:	900-2100 µs

servoValue [servo #]

Diese Nur-Lese-Funktion wird verwendet, um die aktuelle Position der Servos auf einem Sensor-Anschluss zu bestimmen. Werte von 0 bis 255 sind möglich. Der Wert in dieser Variablen, der zurückgegeben wird, ist die letzte Position, der der Firmware sagt, sich zu bewegen. Um die Position eines Servo festzulegen, werden die "servoTarget" oder "servo"-Funktionen verwendet.

Servo [Servo1] = 160; // Ändert die Position des Servo1 bis 160.
servoTarget [Servo1] = 160; // Ändert die Position des Servo1 bis 160.

Bsp.:

```
task main()
{
  servo[servo1] = 0;
  wait1Msc(1000);
  servo[servo1] = 155;
  wait1Msc(1000);
}
```

servoChangeRate [Servo #] = changeRate;

Gibt die Rate an, mit der ein individueller Servo Wert geändert wird. Ein Wert von Null veranlasst einen Servo sich mit maximaler Geschwindigkeit zu bewegen. Die Änderungsrate ist eine nützliche Variable für eine "Glättung" der Bewegung der Servos und verhindert ruckartige Bewegungen, die von Software durch die rasche Berechnung verursacht wird. Die Servo Werte werden alle 20 Millisekunden abgerufen.

```
while (true)
{
  servo[servo1] = 126;
  int delta = 1; // langsame Bewegung
  servoChangeRate[servo1] = delta;
  wait1Msec(2000);
  servo[servo1] = 150;
  wait1Msec(1000);
}
```

Mehr als 2 DC Motoren und mehrere Servos

Um mehr als 2 Motoren anzusteuern können mehrere DC Controller verbunden werden. Ein DC Controller ist immer die Voraussetzung! Es können an jedem Port 1 DC Controller und ein Servo Controller angeschlossen werden.



S1_C1

S1_C2

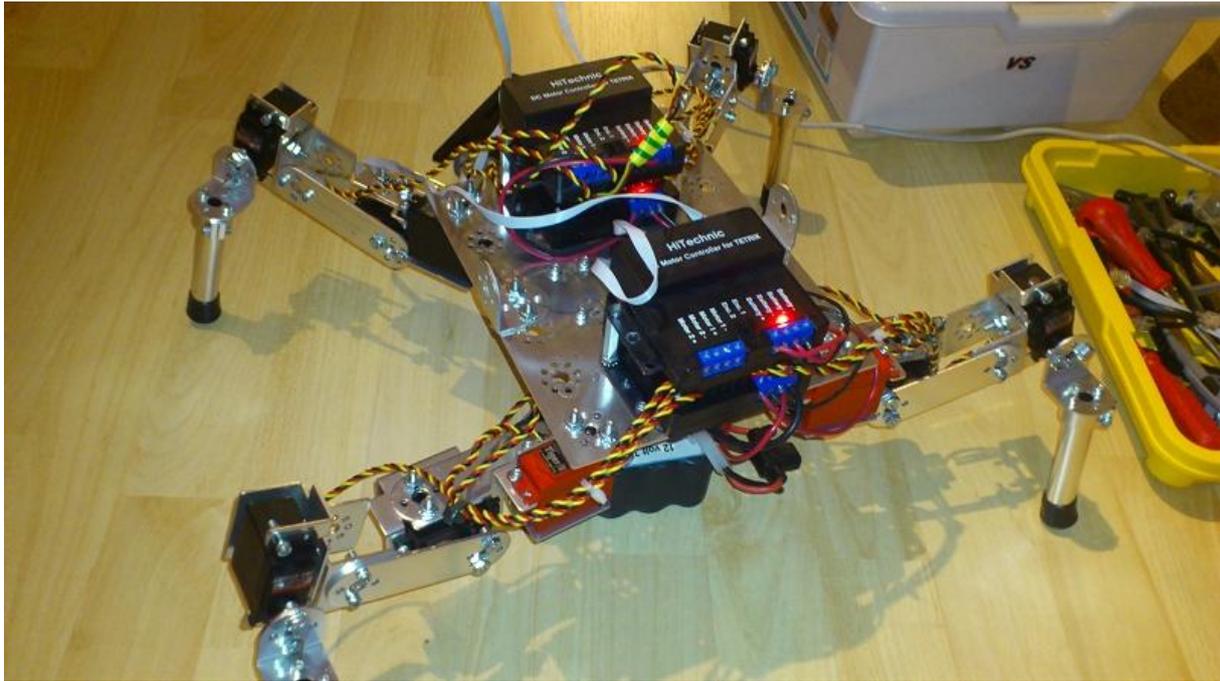
Bis zu vier Controller können angeschlossen werden.

Die Verkettung funktioniert so, dass der erste Controller in der Kette alle Signale des NXT durch den Controller sendet.

Der Controller wird alle die für ihn nicht bestimmten Nachrichten passieren lassen.

Die Nachrichten werden dann sequentiell an ihren endgültigen Bestimmungsort weiter geleitet.

Beispiel ein Quadropod. Diesen habe ich mit 12 Servos, 2 DC Controller, 2 Servo Controller und 1 NXT Baustein gebaut. (Foto nächste Seite) – Video unter www.engeln.info



```
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Hubs, S2, HTMotor, HTServo, none, none)
#pragma config(Sensor, S1, , sensorI2CMuxController)
#pragma config(Sensor, S2, , sensorI2CMuxController)
#pragma config(Servo, srvo_S1_C2_1, RA, tServoStandard)
#pragma config(Servo, srvo_S1_C2_2, RM, tServoStandard)
#pragma config(Servo, srvo_S1_C2_3, RI, tServoStandard)
#pragma config(Servo, srvo_S1_C2_4, LI, tServoStandard)
#pragma config(Servo, srvo_S1_C2_5, LM, tServoStandard)
#pragma config(Servo, srvo_S1_C2_6, LA, tServoStandard)
#pragma config(Servo, srvo_S2_C2_1, HRA, tServoStandard)
#pragma config(Servo, srvo_S2_C2_2, HRM, tServoStandard)
#pragma config(Servo, srvo_S2_C2_3, HRI, tServoStandard)
#pragma config(Servo, srvo_S2_C2_4, HLI, tServoStandard)
#pragma config(Servo, srvo_S2_C2_5, HLM, tServoStandard)
#pragma config(Servo, srvo_S2_C2_6, HLA, tServoStandard)
#pragma config(SrvoPosition, Position03, 130, 57, 75, 85, 100, 100, 125, 45)
```

RobotC 3.5.4 hat hier einen Fehler, da der automatisierte Code der Servopositionen nur 8 Servos erkennt. Darum sollte man die Anfangsstellung der Servos initialisieren.

```
void initializeRobot()
{
servo[RA] = 130; // Startpostion
servo[RM] = 57; // Startpostion
servo[RI] = 75; // Startpostion
servo[LI] = 85; // Startpostion
servo[LM] = 100; // Startpostion
servo[LA] = 100; // Startpostion
```

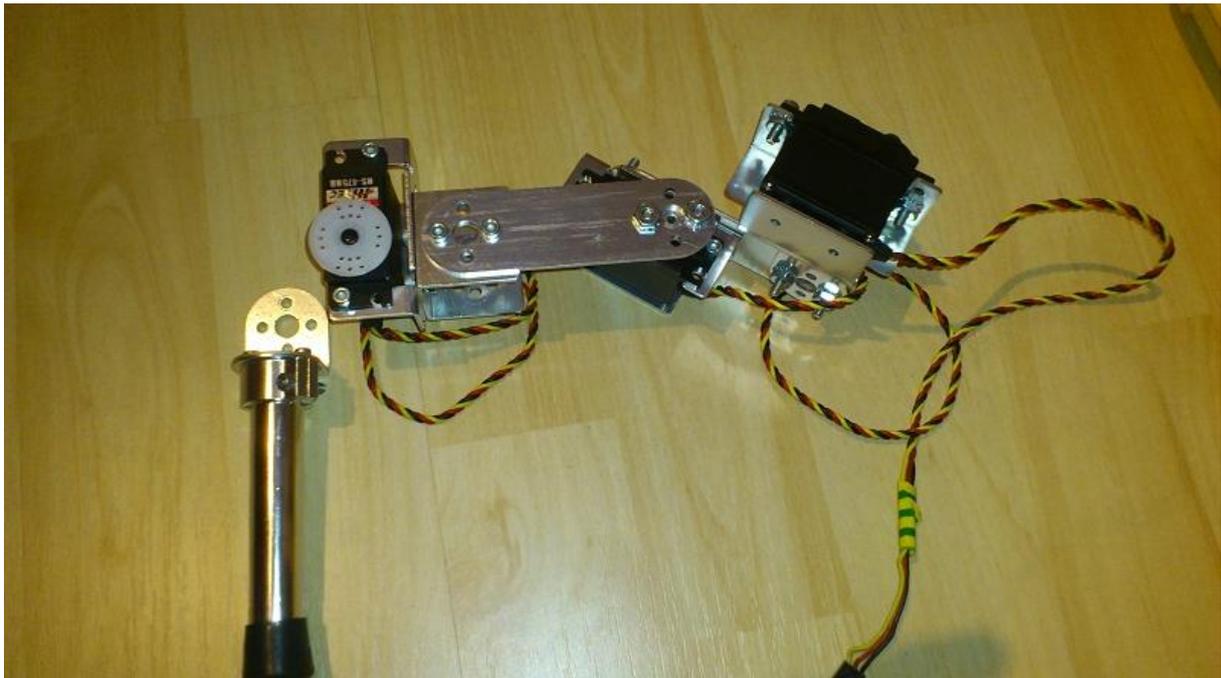
```
servo[HRA] = 125; // Startpostion
servo[HRM] = 45; // Startpostion
servo[HRI] = 115; // Startpostion
servo[HLI] = 120; // Startpostion
servo[HLM] = 210; // Startpostion
servo[HLA] = 120; // Startpostion
wait1Msec(1000); // warten
return;
}

task main()
{

initializeRobot();
wait1Msec(1000);

}
```

Ein einzelnes Bein sieht so aus:



Um den Roboter laufen zu lassen, werden die Beine diagonal in Bewegung gesetzt.

Es können auch Standart Servos aus dem Modellbau verwendet werden. Mit einem 4,5 mm Bohrer kann man die Löcher in die Servoscheibe bohren, um diese mit Tetrrix zu verbinden.

Aufgabenblatt

1. Schreibe ein Programm das deinen Roboter 2 Sekunden vorwärts fahren lässt, dann 2 Sekunden stehen bleibt und wieder zurückfährt.
2. Der Roboter soll vorwärtsfahren und dabei „beschleunigen“ (immer schneller werden). Nach Erreichen der Spitzengeschwindigkeit soll der Roboter allmählich wieder langsamer werden.
3. Finde heraus, mit welchen Parametern der Roboter um 90° , 45° oder 360° gedreht werden kann.
4. Lasse deinen Roboter ein geschlossenes Sechseck durchfahren. Benutze hierzu geeignete Schleifen.
5. Lasse deinen Roboter eine immer größer werdende Spirale fahren.
6. Der Roboter soll 2 Sekunden exakt geradeaus fahren und dann zufallsgesteuert entweder um 90° nach links oder rechts fahren.
7. Der Roboter soll geradeaus fahren, bis er auf einen Gegenstand trifft und dann stoppen.
8. Der Roboter soll selbstständig durch einen Raum navigieren und dabei Gegenständen ausweichen.
9. Der Roboter soll durch einen Raum fahren, ohne irgendwo gegen zu fahren. Bei einem lauten Geräusch soll er stoppen und sich um 180 Grad drehen und die Prozedur soll von vorne beginnen (Schleife).
10. Der Roboter soll einer schwarzen Linie folgen.
11. Schreibe ein Programm, dass dein Roboter nicht vom Tisch fallen kann.
12. Der Roboter soll einen Gegenstand kreisförmig Umfahren.

Mathematische Funktionen

ROBOTC verfügt über eine leistungsstarke Sammlung von nützlichen mathematischen Funktionen für NXT, TETRIX, VEX CORTEX und Arduino MEGA-basierten Plattformen. Die RCX, VEX PIC und Arduino 328P-basierten Plattformen verfügen allerdings nicht über genügend Speicher, um diese erweiterten mathematischen Funktionen zu speichern oder unterstützten keine Fließkommazahlen.

abs

float abs(const float input)

(float) Gibt den absoluten Wert einer Zahl zurück.

Parameter	Erklärung	Datentyp
<i>input</i>	Die Zahl um einen absoluten Wert zu nehmen (kann sein: int, long, short, float).	float

```
float D = -7.81;           // eine Variable 'D' erstellen und diese dem Wert -7.81 zuweisen
float zaehlen = abs(D); // eine Variable erstellen und auf 'zaehlen' und auf den absoluten
                        // Wert von 'D' (-7.81) setzen
```

acos

float acos(const float Cosine)

(float) Liefert den Arcus-Cosinus einer Zahl im Bogenmaß.

Parameter	Erklärung	Datentyp
<i>Cosine</i>	Gibt den Winkel im Bogenmaß zurück, dessen Kosinus der angegebene float-Ausdruck ist. Dies wird auch als Arkuskosinus bezeichnet.	float

```
float param = 0.5;           // erstelle eine Fließkommavariablen 'param' zu 0.5
float result = acos(param) * 180.0 / Pi; // erstelle eine Fließkommavariablen 'result' und
                                        // setze Sie auf den Bogen-Cosinus 'param' in Grad (60)
```

asin

float asin(const float Sine)

(float)

Parameter	Erklärung	Datentyp
<i>Sine</i>	Gibt den Winkel im Bogenmaß zurück, dessen Sinus der angegebene float-Ausdruck ist. Dies wird auch als Arkussinus bezeichnet.	float

atan

float atan(const float Tangent)
(float)

Parameter	Erklärung	Datentyp
<i>Tangent</i>	Gibt den Winkel im Bogenmaß zurück, dessen Tangente der angegebene float-Ausdruck ist. Dies wird auch als Arkustangente bezeichnet.	float

atof

float atof (string str)
(float) Gibt eine *float* "Darstellung" eines String (Zeichenkette) aus

Parameter	Erklärung	Datentyp
<i>str</i>	Zeichenkette zu float umwandeln	string

```
task main()
{
  string strPI = "3.14"; // string 'strPI' wird dem Wert "3.14" zugewiesen
  float test = atof(strPI); // Den String Wert von 'strPI' zu float und 'test' zur Zahl (3.14)
                          // setzen
  while(true); // Endlos-Schleife für den ROBOTC Debugger, so dass wir das Ergebnis
               // sehen können
}
```

atoi

long atoi (string str)
(long) Gibt eine *long* "Darstellung" eines String aus

Parameter	Erklärung	Datentyp
<i>str</i>	Zeichenkette zu long umwandeln	string

```
task main()
{
  string strPI = "3.14";
  long test = atoi(strPI); //Den String Wert von 'strPI' zu long und 'test' zur Zahl (3.14)
                          // setzen
  while(true);
}
```

ceil

float ceil(const float input)

(float) Gibt den kleinsten Integer-Wert, der größer oder gleich dem Eingang ist.

Parameter	Erklärung	Datentyp
<i>input</i>	Standardmäßig verfügt <code>ceil()</code> nicht über einen Parameter zur Bestimmung der Anzahl der Nachkommastellen auf die gerundet werden soll. Dieser lässt sich aber in einer eigenen Funktion leicht nachbilden.	float

```
float E = 2.72;
float ceiling = ceil(E);
```

cos

float cos(const float fRadians)

(float) Rückgabewert: Berechneter Cosinus

Parameter	Erklärung	Datentyp
<i>fRadians</i>	Die Funktion errechnet den Cosinus eines Winkels. Die Rückgabe liegt im Bereich zwischen -1 und 1.	float

```
float result = cos(PI); // float variable 'result' erstellen
```

cosDegrees

float cosDegrees(const float fDegrees)

Parameter	Erklärung	Datentyp
<i>fDegrees</i>	Die Gradzahl des Cosinus nehmen.	float

```
float result = cosDegrees(180); // eine Fließkomma-Variable 'result' anlegen und diese gleich dem Cosinus von 180 Grad (-1) setzen.
```

degreesToRadians

float degreesToRadians(const float fDegrees)

Parameter	Erklärung	Datentyp
<i>fDegrees</i>	Hilfsfunktion: Winkelumrechnung von Grad in Bogenmaß	float

```
float radiansPerDegree = DegreesToRadians(1.0);
```

exp

float exp(const float input)

(float) Berechnet e hoch *arg*.

Parameter	Erklärung	Datentyp
<i>input</i>	Eine Fließkomahochzahl um die Konstante e zu erhöhen. 'e' ist die Basis des natürlichen Logarithmus (ca. 2,718282).	float

```
float result = exp(4); // (e^4 = 54.598)
```

floor

float floor(const float input)

Parameter	Erklärung	Datentyp
<i>input</i>	Rundet eine Fließkommazahl auf die nächste Ganzzahl ab.	float

```
float E = 2.72;
float floorValue = floor(E); // 'E' (2)
```

log

float log(const float input)

Parameter	Erklärung	Datentyp
<i>input</i>	log berechnet den natürlichen Logarithmus, also zur Basis e , der Eulerschen Zahl, die etwa 2,7182818284590452 entspricht.	float

```
float E = 2.72;
float ln = log(E); // Variable 'ln' auf den natürlichen Logarithmus von 'E' (1.00) setzen
```

log10

float log10(const float input)

Parameter	Erklärung	Datentyp
<i>input</i>	Berechnet den dekadischen Logarithmus von <i>arg</i> , d.h. den Logarithmus zur Basis 10.	float

```
float E = 2.72;
float logBTen = log10(E);
```

PI

```
const float PI = 3.14159265358979323846264338327950288419716939937510
```

(float) Die Konstante π .

Nach der Initialisierung der const PI darf der Wert nicht mehr geändert werden.

```
float y = 0.0;           // 'y' auf 0.0 setzen
float LCD_width = 100.0; // Weite des NXT LCD in Pixel

nxtDrawLine(0, 31, 99, 31); // Eine Linie quer (schräg) auf dem NXT zeichnen

for(int x = 0; x < 100; x++) // Schleife
{
    y = sin(x * (2.0 * PI) / LCD_width) * 25;
    nxtSetPixel(x, y + 31);
    wait1Msec(100);
}
```

pow

float pow(const float base, const float exponent)

Mit der Funktion pow kann eine Variable potenziert werden. Negative Potenzen sind ebenfalls möglich.

Parameter	Erklärung	Datentyp
<i>base</i>	zu potenzierende Zahl	float
<i>exponent</i>	die Zahl wird exponent-Mal mit sich selbst multipliziert	float

```
float kilobyte = pow(10, 3);
```

radiansToDegrees

short radiansToDegrees(const float fRadians)

Parameter	Erklärung	Datentyp
<i>fRadians</i>	Die Zahl des Radianten in Grad umrechnen.	float

float degrees = radiansToDegrees(PI);

rand

rand liefert eine ganzzahlige Zufallszahl im Bereich von 0 und $2^{15} - 1$. *rand()* erzeugt eine 15 Bit lange Zufallszahl.

void srand(const long nSeedValue)

srand wird zur Initialisierung des Zufallszahlengenerators benutzt.
(Plattformabhängig).

Eine Zufallszahl kann mit dem Modulo-Operator in einen beliebigen Bereich gekürzt werden.

Pseudozufallszahlen sind eine Reihe von zufällig wirkenden Zahlen, die jedoch mit einer mathematischen Formel errechnet werden. Um die Zahlenreihe zu initialisieren verwendet man die Funktion *srand()*. Das Setzen des Zufallsgenerators auf einen bestimmten Seed-Wert führt dazu, dass die Zufalls sich in gleicher Reihenfolge wiederholen. Dies ermöglicht das Testen von Programmen, die mit Zufallszahlen verarbeiten, unter den reproduzierbaren Bedingungen.

```
task main()
{
  int min = -100;
  int max = 100;

  srand(nSysTime); // Zufallszahl aus der Systemzeit holen

  while(true)
  {
    motor[rightMotor] = (rand() % (max-min)) + min;
    motor[leftMotor] = (rand() % (max-min)) + min;
    wait1Msec(500);
  }
}
```

sgn

short sgn(const float input)

(short) Gibt einen Wert kleiner als 0 zurück, wenn der Input negativ ist, und ein Wert größer als 0, wenn der Input positive ist.

Parameter	Erklärung	Datentyp
<i>input</i>	Einen „Signalimpuls“ testen	short

```
int res1 = sgn(-9.81); // gibt -1 an 'res1' zurück
int res2 = sgn(3.14); // gibt 1 an 'res2' zurück
int res3 = sgn(0);    // gibt 0 an 'res3' zurück
```

sqrt

float sqrt(const float input)

Parameter	Erklärung	Datentyp
<i>input</i>	Mit der Funktion sqrt kann die Quadratwurzel aus einer Zahl gezogen werden.	float

```
float result = sqrt(1764); // Quadratwurzel von 1764 = 42
```

Batterie & Leistungs-Kontrolle

Die NXT Firmware verfolgt die Minuten der Inaktivität auf dem NXT. Wenn der NXT eine Zeit lang ungenutzt steht oder liegt, dann wird der NXT automatisch ausgeschaltet. Inaktivität ist definiert als eine „Benutzung ohne Tastendruck“.

Der Timeout-Wert kann mit der NXT-GUI gesetzt werden. Er kann auch über die RobotC Software bearbeitet werden. Wenn RobotC eine Verbindung mit dem NXT hat, wird der Timeout-Wert auf den Wert aus der Software angepasst.

Zu beachten ist, dass der NXT ein unvorhersehbares Verhalten, bei niedrigen Spannungen, beim Versuch ein Programm in den Flash-Speicher zu schreiben, haben kann. Also, wenn RobotC eine Verbindung zwischen PC und NXT aufbaut, überprüft es die Batteriespannung. Wenn die Spannung ist zu niedrig für einen zuverlässigen Betrieb sein sollte, wird die RobotC Verbindung abgebrochen.

alive()

Aufruf dieser Funktion setzt den "inaktiven Power- down"-Zähler zurück. Wenn ein Programm ausgeführt wird und es in regelmäßigen Abständen die "alive ()"-Funktion aufruft, wird verhindert, dass der NXT automatisch ausgeht.

powerOff()

Diese Funktion stellt den NXT aus.

bNoPowerDownOnACAdaptor

Eine boolsche Abfrage. Wenn der NXT mit einem AC Adapter angeschlossen ist, wird verhindert, dass sich der NXT automatisch nach einer gewissen Zeit ausstellt.

bNxtRechargable

Überprüft, ob der NXT mit Batterien oder mit einem Akku betrieben wird.

nAvgBatteryLevel

Der Durchschnitt einiger neuer Messungen der Batteriespannung. Ein Wert von 7500 zeigt eine Spannung von 7,5 Volt.

nImmediateBatteryLevel

Der jüngste Stichprobenwert der Batteriespannung.

nPowerDownDelayMinutes

Die Anzahl der Minuten der Inaktivität, das automatische Ausschalten des NXT.

nShutdownVoltage

Der geringe Spannungswert bei dem die automatische Abschaltung des NXT erfolgt. Ein Wert von 6100 zeigt eine Abschaltung bei 6.1V.


```
    PlaySound(soundLowBuzz);
    break;
}
nxtDisplayTextLine(0, "Battery Levels");

//
// Zeigen des Batterie Zustands
//
nxtDisplayTextLine(2, "Avg %3.1fV;Now %3.1f", nAvgBatteryLevel / (float) 1000,
nImmediateBatteryLevel / (float) 1000); //in einer Zeile schreiben!

//
// Anzeige der Spannung, bei dem der NXT "ausschaltet"
//
nxtDisplayTextLine(4, "Shutdown at%4.1fV", nShutdownVoltage / (float) 1000);

//
// Zeigen der aktuellen Zeit, wann der NXT "ausschaltet" (Restzeit)
//
nxtDisplayTextLine(5, "Sleep Tmr: %d min", nPowerDownDelayMinutes);

//
// Anzeige der Batterie-Typ: Akku oder nur Batterie
//
if (bNxtRechargable)
    nxtDisplayTextLine(7, "Rechargable");
else
    nxtDisplayTextLine(7, "Non-Rechargable");

//
// sofort Ausschalten des NXT wie folgt:
//
wait1Msec(10);
++nCounter;
if (nCounter > 15000)
    powerOff();
}
return;
}
```

Schlüsselwörter in C (reservierte Wörter)

break	case	char	const	continue
default	do	double	else	enum
extern	float	for	goto	if
int	long	register	return	short
signed	sizeof	static	struct	switch
typedef	union	unsigned	void	while

Operatoren

Operator	Bedeutung
<	kleiner als
>	größer als
<=	kleiner oder gleich
>=	größer oder gleich
!=	ungleich
==	gleich

Bluetooth

Bluetooth ist eine in den 1990er Jahren ursprünglich von Ericsson entwickelter Industriestandard für die Funkvernetzung von Geräten über kurze Distanz. Bluetooth bildet dabei die Schnittstelle, über die sowohl mobile Kleingeräte wie Mobiltelefone und PDAs als auch Computer und Peripheriegeräte (wie in unserem Fall der NXT) miteinander kommunizieren können. Hauptzweck von Bluetooth ist das Ersetzen von Kabelverbindungen zwischen Geräten. (www.wikipedia.de)

Bluetooth ermöglicht dem NXT-Stein mit allen Geräten die Bluetooth-fähig sind, kabellos zu kommunizieren. Ob Bluetooth Class I oder Class II ist ziemlich egal - auch wenn der NXT nur Class II hergibt. Ansonsten Bluetooth Stack auf dem PC installieren, Bluetooth-Stick in den USB Port stecken.

Die Kommunikation zwischen mehreren NXT Prozessoren über Bluetooth erfolgt auf der HighLevel Ebene auf einem sehr einfachen Prinzip. Es können grundsätzlich beliebig viele NXT Prozessoren miteinander kommunizieren. Dabei ist jedoch zu beachten, dass ein Baustein den Master übernimmt. Die restlichen NXT übernehmen hierbei die Slaves. (aus: *Martin Stypinski: Bluetooth with NXT*)

Testanweisung, ob Bluetooth aktiviert ist:

```
void checkBTLinkConnected()
{
    if (nBTCurrentStreamIndex >= 0)
        return;

    PlaySound(soundLowBuzz);
    PlaySound(soundLowBuzz);
    eraseDisplay();
    nxtDisplayCenteredTextLine(3, "BT nicht verb.");
    nxtDisplayCenteredTextLine(4, "Verbunden");
    wait1Msec(3000);
    StopAllTasks();
}
```

Einfaches Sender-Programm:

```
.....

if(SensorValue(HTEOPD) > 45)
{
    wait1Msec(200);
}
```

```
int my_number = 23;
sendMessage(my_number);
wait1Msec(10000);
}
.....
```

Empfänger:

```
...
while(true)
{
  checkBTLinkConnected();

  my_message = message; //message zur Variablen

  if(my_message == 23) // !=0 wenn die 23 empfangen wird
  {
    StopTask(DriveForward); // wird das Programm „fahren“ gestoppt
    StartTask(Arm); // und ein anderes Programm gestartet
    wait1Msec(4000);
    nxtDisplayTextLine(3,"1: %d",my_message); // Empfang wird angezeigt
    ClearMessage(); //Clear
    StopTask(Arm);
  }
  wait1Msec(500); //jede halbe Sekunde prüfen, ob ein Signal kommt
...

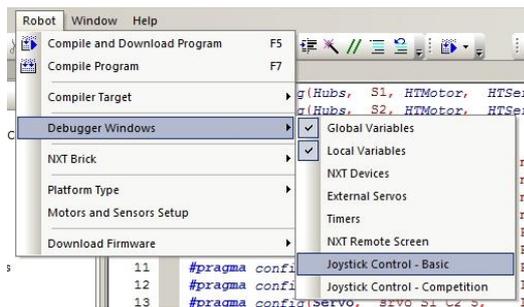
```

Joystick Control



Logitech Gamepad F310

Nach Installation der Logitech Software und Anschluss des Joysticks, wird RobotC gestartet. Nachdem ein Standardprogramm aufgerufen wurde, wird dieses kompiliert und gedownloadet (F5). Programm starten.



Zum Robot Menü - *Debug Windows* gehen und *Joystick Control – Basic* auswählen.

Der Joystick kann jetzt zur Steuerung des Roboters genutzt werden. Bewegt man den linken Daumen „Stick“ nach vorne oder zurück, so wird Motor D bewegt.

Die Motoren drehen vorwärts, wenn der Daumen-Stick gedrückt und nach vorn bewegt wird.

Ähnlich funktioniert es auch mit dem Funk-Gamepad F710. Zuerst die mitgelieferte Software installieren und dann s.o..

RobotC Virtual World

Simulationsumgebung für virtuelle NXT-Roboter

Dieser Teil von RobotC baut virtuelle Welten auf, so dass man die Sprache und Syntax von RobotC lernen kann ohne einen NXT anschließen zu müssen. Jeder kann so lernen, wie die LEGO-Roboter und VEX Verwendung durch dieses neue Tool programmiert werden.

Ein weiteres Ziel des Projektes ist es, anderen (z.B. Schülern) zu ermöglichen, virtuelle Welten für Benutzer im Programm zu bauen ... Wie cool ist das? Im Idealfall werden die Kursteilnehmer in der Lage sein, virtuelle Welten mit einer Kombination von Software zu entwickeln. Diese besondere Welt ist mit Hilfe Unity möglich.

Virtual World ist eine Simulation für Mindstorms NXT und Tetrrix. Programmiert wird in RobotC. Durch drücken von F5 wird das Programm kompiliert und der Simulator startet. Es stehen verschiedene Welten als Download zur Verfügung.

<http://www.robotc.net/download/rvw/>

RobotC's neue Robot Virtual World Simulationssoftware ermöglicht es zu programmieren und die Roboter in einer Simulationsumgebung zu testen, bevor man den Code auf den realen Roboter überträgt.

Eine Spiel ähnliche Umgebung, auf der die Programmierung Spaß macht.



Versuche zur Wärmelehre

Warmwasser

Beschreibung: Um Wasser zu erwärmen werden oft Tauchsieder genutzt. Der Versuch soll erklären, wie die im Tauchsieder gewonnene Wärmeenergie verschiedene große Wassermengen erwärmt.

Geräte: Becherglas 600 ml, NXT, Temperatursensor, Tauchsieder

Versuchsdurchführung: Erwärme verschiedene große Wassermengen (200, 300 ... 500 ml) in jeweils gleichen Zeiten (2 Minuten), misst die Temperaturerhöhung und überträgt sie in eine Tabelle.

Aufgaben:

1. Fertigt eine Tabelle an

Wassermenge in ml	Wassertemperatur vor dem Versuch in °C	Wassertemperatur nach 2 Minuten Heizdauer in °C	Temperatur-Unterschied zwischen Anfangs- und Endtemperatur in °C
200			
300			

2. Wie kann man die Beobachtungen in einem allgemeinen Ergebnissatz zusammenfassen?
3. Führe eine eigene Messreihe durch. Erhitze verschiedene Wassermengen auf ca. 50 °C. Notiere die Zeit die nötig ist, diese Temperatur zu erreichen.

Wassermenge in ml	Zeit in sec.

Geräuschpegelmessung im Straßenraum

Lärmpegel werden in dB(A) gemessen. Dies entspricht etwa der veralteten Einheit Phon. Bei 0 dB(A) liegt die Hörschwelle des Menschen. Ein Motorrad erzeugt im Leerlauf in einem Meter Abstand ca. 80 dB(A), und die Schmerzgrenze liegt bei etwa 120 dB(A). Die Messungen werden durch die Schülerinnen/Schüler gruppenweise als außerschulische Arbeit oder im Rahmen von Exkursionen und in verschiedenen Straßentypen durchgeführt. Die Auswahl ist nach lokalen Gegebenheiten zu kürzen oder zu erweitern.

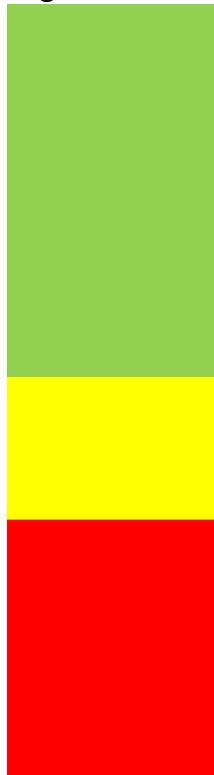
Hinweis:

Um die Sicherheit im Straßenverkehr zu gewährleisten, ist es angebracht, die Schülerinnen und Schüler auf die Regeln der Verkehrssicherheit hinzuweisen.

Aufgaben:

1. Führt an verschiedenen Orten Messungen des Straßenlärms durch. Die Messungen sollen alle 3 Minuten dauern. Das Intervall der Messung kann 5 Sekunden betragen. Zeichnet die Messpunkte in eine Karte. Speichert die Daten auf dem NXT und wertet diese in der Schule aus. Vergleicht die Werte.
 2. Nutze den NXT als Messgerät. Lege eine Tabelle mit verschiedenen Geräuschen und der Lautstärke in dB.
- Für Fortgeschrittene: Baut eine Lärmampel. Von 0-60 dB grün, 61-90 dB gelb und 91 – 160 dB rot.

Pegelbereiche für Lärm in der Umwelt dB(A) Beispiele Wahrnehmungsbereich



- 0 Definierte Hörschwelle
- 10 Blätterrauschen im Wald bei Windstille, Schneefall, normales Atmen
- 20 Tropfender Wasserhahn, leichter Wind, Blätterrauschen
- 30 Flüstern (1 m Abstand), Ticken eines Weckers
- 40 Brummen eines Kühlschranks (1 m Abstand); sehr leise Radiomusik
- 50 Leise Radiomusik (1 m Abstand)
- 60 Umgangssprache; PKW in 15 m Abstand
- 70 Rasenmäher (7 m Abstand), Schreibmaschine (1 m Abstand)
- 80 Pkw mit 50 km/h 1 m Abstand, max. Sprechlautstärke
- 90 Lkw-Motor 5 m Abstand; Pkw mit 100 km/h (1 m Abstand)
- 100 Kreissäge; Lärm in einem Kraftwerk, Posaunenorchester
- 110 Propellerflugzeug (7 m Abstand), Bohrmaschine; laute Diskothek
- 120 Verkehrsflugzeug 7 m Abstand, Beginn der Schmerzgrenze
- 130 Düsenflugzeug (7 m Abstand), Walkman Maximalbelastung
- 160 Gewehrschuss in Mündungsnähe

(Quelle: SEIDEL, 1998; GRIEFAHN; 1988; modifiziert und erweitert von: Ministerium für Umwelt und Forsten Rheinland-Pfalz, Ministerium für Umwelt und Naturschutz des Landes Nordrhein-Westfalen 2004)

Wir bauen ein Fernthermometer

Es gibt in der Technik Räume, deren Temperatur von der Ferne her überwacht werden muss. Entweder ist es in ihnen so warm, dass man nicht in unmittelbarer Nähe mit einem Thermometer messen kann, oder die Anwesenheit von Menschen ist aus anderen Gründen unmöglich. In solchen Fällen braucht man ein Fernthermometer.

Versuchsdurchführung:

Umwickelt das Thermometer des NXT mit Kupferdraht, so dass ihr so eine Art „Fühler“ habt. Stelle nun eine Kerze im bestimmten Abstand in die Nähe des Drahtes.

Aufgaben:

Wie ist die Wärmeentwicklung im Abstand zum Draht? Lege eine Tabelle an und übertrage die Werte. Fasse Deine Beobachtungen zusammen.

Lichtmessung mit dem NXT

Aufgabe: Ermittelt die Sensorwerte für die Farbe rot, blau und gelb. Trage die Ergebnisse in die Tabelle ein und vergleicht eure Werte.

Farbe	rot	blau	gelb	weiß	schwarz
Wert					

Ein automatischer Feuermelder

Früher wohnten Wächter auf hohen Türmen in den Städten, die ein entstehendes Feuer durch ein Hornsignal zu melden hatten.

Feuermelder in unserer Zeit sehen anders aus.

Öffentliche Feuermelder sind der Öffentlichkeit zugänglich und werden von Hand ausgelöst. Beim Betätigen des Feuermelders wird sofort Alarm gegeben; zugleich läuft eine Kontaktscheibe ab und gibt in der Meldezentrale den Standort an, von dem die Meldung kommt. Dennoch kommt es vor, dass die Feuerwehr zu spät an die Brandstelle gelangt.

Deshalb hat man automatische Feuermelder gebaut, die bei jeder gefährlichen Temperaturerhöhung in einem Raum, auf einem Schiff, in einem Bergwerk etc. selbständig Alarm geben, ja sogar von sich aus die Brandbekämpfung einleiten.



Aufgaben:

1. Konstruiere einen Feuermelder mit dem NXT.
2. Schreibe ein Programm, wobei bei einer Temperaturerhöhung ein Signalton und eine Lampe leuchten sollen (Alarm).

Tipp: Als Feuerstelle kannst du ein Teelicht nehmen.

Für Fortgeschrittene:

3. Entwerfe eine Löschvorrichtung und versuche die Kerze zu löschen (Ventilator, oder Hydraulikpumpe – Wasser)

Beispiele

Der Roboter soll nach Umdrehungen fahren

```
task main()
{
  nMotorEncoder[motorA] = 0;
  nMotorEncoder[motorB] = 0;

  while(nMotorEncoder[motorA] < 1800 || nMotorEncoder[motorB] < 1800)
  {
    motor[motorA] = 100;
    motor[motorB] = 100;
  }
}
```

Der Roboter soll beim Klatschen stehen bleiben.

```
task main()
{
  wait1Msec(1000);

  while(SensorValue(soundSensor) <= 50)
  {
    motor[motorA] = 75;
    motor[motorB] = 75;
  }
}
```

Auf Weiß warten

```
task main()
{
  wait1Msec(100);
  while(SensorValue(lightSensor) < 45)
  {
    motor[motorA] = 100;
    motor[motorB] = 100;
  }

  motor[motorA] = 0;
  motor[motorB] = 0;
}
```

```
//Vorwärts fahren

#pragma config(Motor, mtr_S1_C1_1, motorD, tmotorNormal, PIDControl, encoder)
#pragma config(Motor, mtr_S1_C1_2, motorE, tmotorNormal, openLoop)
#pragma config(Hubs, S1, HTMotor, HTServo, none, none)
#pragma config(Servo, srvo_S1_C2_1, lenkung, tServoNormal)
#pragma platform(Tetrrix)

void fahren();

task main()
{
  while(SensorValue(touchSensor) == 0)
  {
    //leer
  }
  if (SensorValue(touchSensor) == 1)
  {
    fahren();
  }
}

void fahren()
{
  int delta = 2;
  nMotorEncoder[motorA] = 0;
  int gS = 30;
  int richtung = 129;

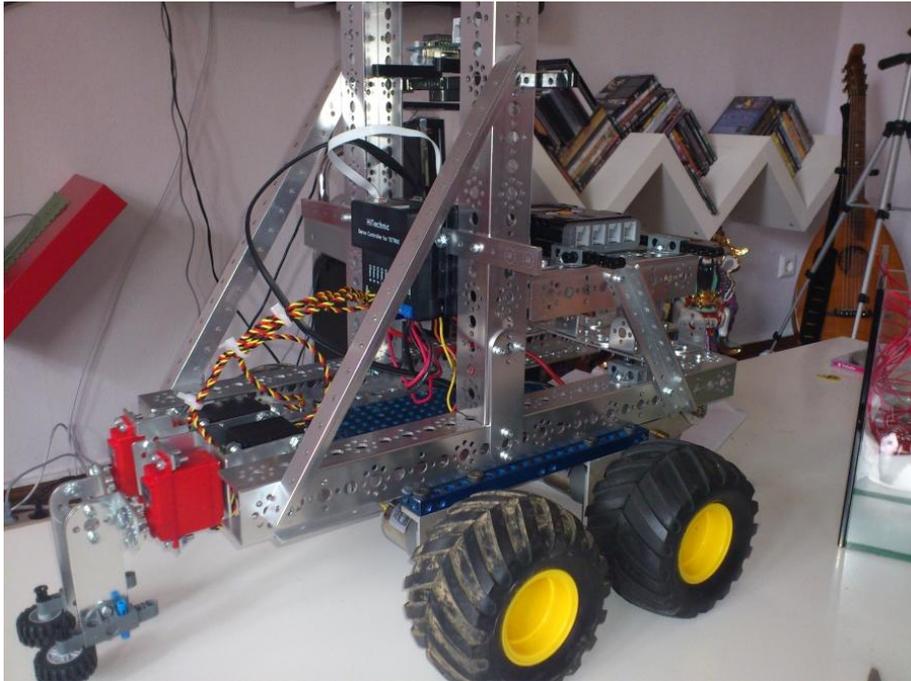
  while (nMotorEncoder[motorA] < 360) //1 Umdrehung = 75,5 cm
  {
    servoChangeRate[lenkung] = delta;
    servo[servo1] = richtung;
    motor[motorA] = gS;
    motor[motorD] = 30;
    motor[motorE] = -30;

  }
}
```

Meine Roboter:

Der Vorteil bei Tetrax ist ganz klar die Stabilität und die Möglichkeiten auch andere Dinge zu befestigen/kombinieren. Einfach mal in den Modellbauläden und ausprobieren. Ich habe einige Servos dort günstig erstanden, sowie die Räder.

Beispiele findet man unter RobotC.net und Filme auf youtube.com oder myvideo.de



Programme und Bilder unter: www.engeln.info

Links

<http://www.robotc.net/>

<http://www.education.rec.ri.cmu.edu/content/lego/curriculum/index.htm>

<http://www.nxt-roboter.de>

<http://www.nxt-forum.de>

<http://www.nxt-wissen.de>

<http://www.engeln.info>

<http://www.hitechnic.com>

<http://www.robotc.net>

<http://www.educatech.ch>

<http://dexterindustries.com/products.html>

<http://www.proggen.org/>

ROBOT-C basiert auf der weltweit eingesetzten Standardprogrammiersprache C.

Robot C – Steuert verschiedene Systeme (unter anderem RCX und NXT)

<http://www.robotc.net/>

RobotC-Download-Links :

http://www.robotc.net/content/lego_down/lego_down.html

RobotC-Anleitung:

http://www.robotc.net/support/nxt/MindstormsWebHelp/index.htm#page=nxt_functions/

RobotC-Forum (englischsprachig):

<http://www.robotc.net/forums/>

Robotc für andere Systeme

ARDUINO Duemilanove, Uno, Mega 1280 and Mega 2560



(Abb. kann vom Original abweichen)

Der Arduino Due ist das erste Arduino-Board, das einen 32-bit ARM-Mikrocontroller verwendet. Hierdurch lassen sich viel mehr Projekte verwirklichen, die die anderen Boards nicht oder nur mit speziellen Shields realisieren konnten.

Über die zwei Analog-Output-Pins ist zum Beispiel Musikwiedergabe ein Kinderspiel! Mit seinen 54 Digital-I/O-Pins, 12 Analog-In- und 2 -Out-Pins und der einfach anhand von Beispielen zu erlernenden Programmiersprache lassen sich viele Aufgaben der Mess-, Steuer- und Regeltechnik im Handumdrehen und Kinderleicht verwirklichen.

Das raffinierte Arduino-Konzept sieht es vor, das Grundboard durch Zusatzplatinen, die sogenannten Shields, mit unzähligen weiteren Funktionen auszustatten.

Der 328P-basierende Arduino verfügt über 6 analoge Sensor- und 13 digitale Sensor-ports.

Vex Robotics Roboterbausätze

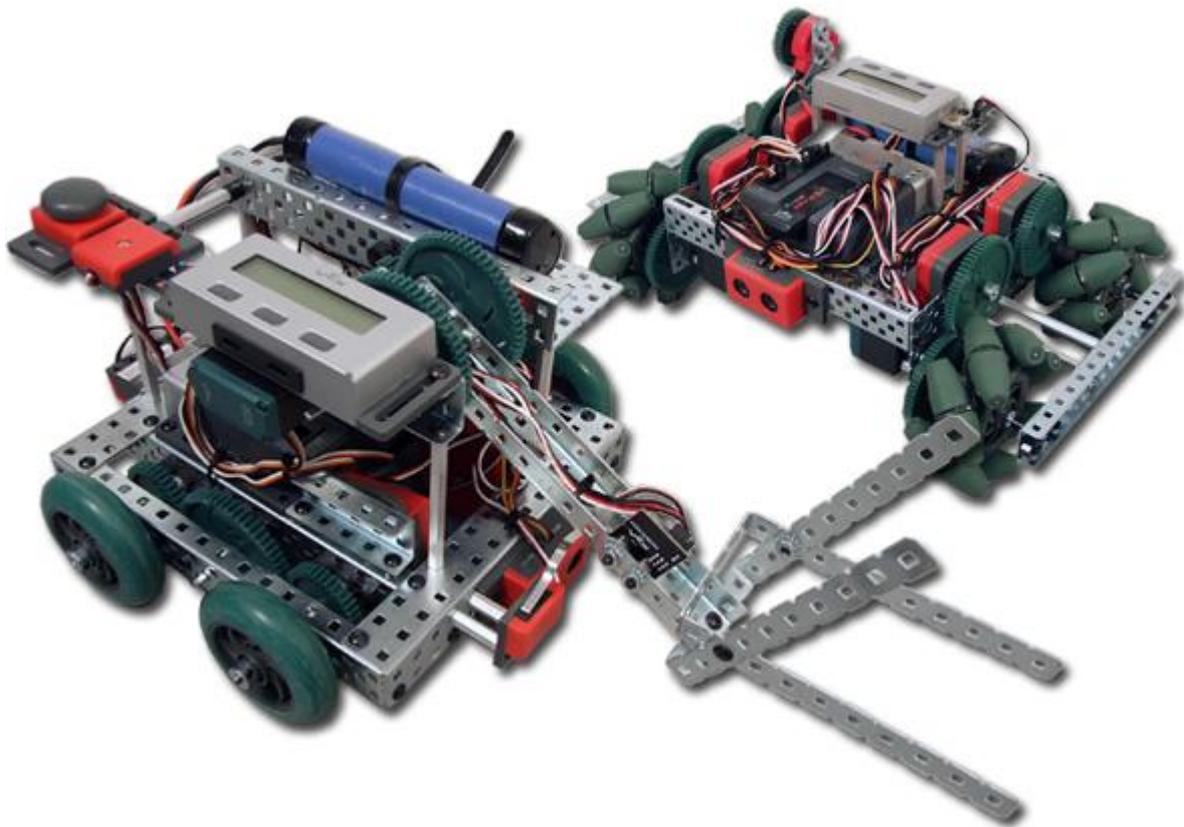


Bild: Robotc.net

Der Cortex Mikrocontroller für Programmierer bildet das Gehirn des Roboters und besitzt eine native Unterstützung drahtloser Technologien. Dadurch wird eine bidirektionale Kommunikation ermöglicht (die VexNet-Technologie ist bei diesem Mikrocontroller nativ integriert). So können Programme drahtlos heruntergeladen werden, und auch das Debuggen erfolgt im Direktzugriff.

- 8 Ports für Servos oder 3-Kanal-Standardmotoren.
- 2 Ports für 2-Kanal-Motoren (vom Typ H-Bridge).
- 1 I2C-Port „Smart Sensor“
- 2 serielle UART-Schnittstellen für den Anschluss des LCD-Bildschirms von Vex Robotics
- 8 12-Bit-Analogschnittstellen
- 12 hochschnelle digitale Ein-/Ausgänge (50 Hz)
- Ports Rx1 und Rx2: Unterstützen außerdem zwei 75 MHz Crystal UHF-Sender/Empfänger
- 1 DAC Lautsprecher
- 1 Mikrocontroller STMicroelectronics ARM Cortex-M3 (90 MIPS, 64 KB RAM, 384 KB Flash-Memory für Programme)

Achtung : Derzeit kann der Vex Cortex Mikrocontroller lediglich mit einer RobotC Betaversion gesteuert werden, die Inhabern einer gültigen RobotC-Lizenz kostenlos zur Verfügung steht. Alle anderen RobotC-Versionen könnten den Mikrocontroller dauerhaft beschädigen.

<http://www.generationrobots.de>

