

Maturaarbeit

Kantonale Mittelschule Uri, 6460 Altdorf

Programmiersprachenvergleich für einen NXT LEGO Roboter

Daniel Walker

begleitet von Lukas Wariwoda

2007

1 Abstract

In dieser Arbeit werden die drei Programmiersprachen NBC, NXC und Robolab verglichen. Ich möchte aufzeigen, wo die Stärken und Schwächen der einzelnen Sprachen liegen, und ob es eine „perfekte“ Programmiersprache gibt.

Mit Hilfe eines Bewertungskataloges soll ein Vergleich der Programmiersprachen ermöglicht werden. Basierend auf diesen Bewertungskriterien werden zwei Aufgaben erstellt. Beide Aufgaben werden in jeder Programmiersprache programmiert. Entsprechend der Leistung in der praktischen Anwendung erfolgt die Bewertung der Programmiersprachen.

Um Eindrücke von Dritten zu erlangen, werden Laien vereinfachte Aufgaben gestellt.

2 Typografische Festlegungen

kursiv

im Glossar erläuterte Ausdrücke

kursiv-fett

Befehle

Inhaltsverzeichnis

1	Abstract	2
2	Typografische Festlegungen	2
3	Einleitung	4
4	Theorie	5
4.1	Geschichte	5
4.2	Programmiersprachen	6
4.3	Aktoren und Sensoren des NXTs	8
4.4	Funktionen der Programmiersprachen	11
4.5	Vom Entwurf bis zum Programm auf dem NXT	14
5	Durchführung	15
5.1	Vorbereitungsphase	15
5.2	Lernphase	16
5.3	Vergleichsphase	16
6	Auswertung	17
6.1	Actoren	17
6.2	Sensoren	18
6.3	Kontrollfunktionen	19
6.4	Aufgaben und Subroutinen	21
6.5	Kommunikation	22
7	Eindrücke anderer Personen	23
7.1	Oberägeri	23
7.2	Experimente mit Testpersonen	24
8	Zusammenfassung	25
9	Glossar	26
10	Literaturverzeichnis	29
11	Bildnachweis	31
12	Anhang	32
12.1	Aufgaben	32
12.2	Programme	33
12.3	Bewertungsbögen	41

3 Einleitung

Nahezu jedem Kind sind die kleinen, farbigen Bausteine der Firma LEGO bekannt. Dass LEGO allerdings nicht nur etwas für die Kleinen ist, sollte spätestens seit der Veröffentlichung der LEGO MINDSTORMS Produktreihe klar sein. Mittlerweile wird an vielen Universitäten und Forschungsanstalten mit den LEGO-Robotern experimentiert.

Auch ich habe Bekanntschaft mit dem kleinen zuerst gelben und später grauen programmierbaren Baustein gemacht, und wurde so in die Welt der Roboter eingeführt. Schon bald reichte mir der Funktionsumfang der LEGO-Programmiersprache *NXT-G* nicht mehr aus, und ich schaute mich nach anderen Sprachen um. Durch die grosse Auswahl an Programmiersprachen, die teilweise speziell für den LEGO MINDSTORMS NXT entwickelt wurden, entschloss ich mich ein Teil dieser Sprachen zu vergleichen. Dies war der Beginn meiner Maturaarbeit.

Wie bereits erwähnt ist die Anzahl an Programmiersprachen für den NXT sehr gross. Alle erhältlichen Programmiersprachen zu vergleichen, würden den Rahmen dieser Arbeit um ein Vielfaches sprengen. Aus diesem Grund habe ich mich darauf beschränkt, drei unterschiedliche Sprachen zu vergleichen.

Zu den verglichenen Sprachen gehören NBC als *Assembler-ähnliche* Sprache, NXC als *C-basierte* Programmiersprache und schliesslich Robolab als *grafische Sprache*.

Die Programmiersprachen werden mit Hilfe des neusten LEGO Roboters NXT der MINDSTORMS Reihe verglichen. Es gilt herauszufinden, wo die Stärken und Schwächen der einzelnen Sprachen liegen, und ob es eine "perfekte" Programmiersprache gibt.

4 Theorie

4.1 Geschichte

- *LEGO MINDSTORMS Robotics Invention System (RIS)*

LEGO wollte eine Schnittstelle zwischen Spielzeug und Computer schaffen, die auch ältere Kinder, Jugendliche und sogar Erwachsene fasziniert. Dabei sollte die Schnittstelle nicht zu verspielt sein, sich aber auch nicht zu sehr an reiner Informatik orientieren. Zusammen mit dem MIT (Massachusetts Institute Of Technology) wurde dieses Projekt von Seymour Papert, der schon in den 70er Jahren die Kinder-Programmiersprache Logo entwickelt hatte, begründet. Papert ging es stets darum, dass die Kinder nicht auswendig lernen, sondern ausprobieren und aus diesen Erfahrungen ihre Kenntnisse ziehen.

„MINDSTORMS“, ein erstes Ergebnis der Zusammenarbeit, ist kein Baukasten, sondern ein Schlüssel! Es sind keine fest formulierten Anwendungen vorhanden, das Programm als auch das Handbuch bieten vielmehr lockere Vorschläge, die dazu anregen dieses oder jenes auszuprobieren.¹

Im Paket sind unter anderem ein Lichtsensor, zwei Berührungssensoren und zwei Motoren sowie etliche LEGO TECHNIC Bauteile enthalten.

Zentraler Bestandteil des RIS ist ein Hitachi H8 *Mikrocomputer* in einem größeren LEGO-Baustein, der sogenannte Programmable Brick zu Deutsch programmierbarer Baustein oder *RCX*. Die Programme werden mittels *serieller Schnittstelle* und *Infrarot Tower* vom Computer auf den RCX übertragen.²



Abbildung 2: RCX

- *LEGO MINDSTORMS NXT*



Abbildung 1: NXT

Die neue Generation von LEGO MINDSTORMS baut auf den Erfolg des weltweit renommierten Robotics Invention System auf und ist in der Handhabung noch schneller und einfacher.

Mit den neuen technischen Funktionen und weiteren, noch leistungsfähigeren Sensoren werden auch erfahrene Roboterbauer angesprochen. Der RCX wurde durch den NXT ersetzt, ein 32-Bit ARM 7 *Mikroprozessor*, der nicht nur am PC, sondern erstmalig auch am *Mac* programmiert werden kann. Die beigelegte Software ist einfach in der Handhabung und basiert auf *LabVIEW* von National InstrumentsTM.

Sind die Programme programmiert, können Sie entweder kabellos mit *Bluetooth* oder mit dem beigelegten *USB 2.0* Kabel auf den NXT übertragen werden. Dank Bluetooth kann der Roboter von einem Handy oder *PDA* aus ferngesteuert werden.³

1 Sinngemäss aus: siehe Literaturverzeichnis (LV) [1].
2 Sinngemäss aus: siehe LV [2].
3 Sinngemäss aus: siehe LV [3].

4.2 Programmiersprachen

- *Definition*

Programmiersprachen sind künstlich geschaffene Sprachen in Form von Textkürzeln, mit denen Programme für Datenverarbeitungsanlagen (in unserem Fall der NXT) erstellt werden können. Programmiersprachen werden in fünf Generationen klassifiziert. Je höher die Generation, desto leistungsfähiger ist die Programmiersprache.¹

- *Generationen*

Im Folgenden werde ich auf die einzelnen Generationen näher eingehen. Dabei möchte ich mich auf die Generationen, in welche die verglichenen Programmiersprachen einzuordnen sind, beschränken.

- *2. Generation: Assembler*

Die Assemblersprache ist eine maschinennahe Sprache, bei der allerdings die Instruktionen durch bestimmte Kürzel ersetzt wurden. Da die Anzahl der Kürzel gering ist, lassen sich Assemblersprachen nur geringfügig leichter schreiben als Sprachen der 1. Generation. Im Gegensatz zu *Maschinencode* können Assemblersprachen nicht direkt ausgeführt werden, sondern müssen zuerst, mittels *Assembler* in Maschinencode umgewandelt werden.²

Die verglichene Programmiersprache NBC ist eine Assembler-ähnliche Programmiersprache.

Eine If (wenn) -Abfrage würde in einer Assemblersprache wie folgt aussehen:

```
brcmp EQ, Then, X, 24
Else:
wenn x ≠ 24, führe diesen Programmteil aus
jmp EndIf
Then:
wenn x = 24, führe diesen Programmteil aus
EndIf:
hier ist die Abfrage zu Ende
```

1 Sinngemäss aus: siehe LV [4].

2 Sinngemäss aus: siehe LV [5].

○ 3. Generation: *Höhere Programmiersprachen*

Da höhere Programmiersprachen nicht nur aus maschinennahem Code bestehen sondern auch aus Teilen natürlicher Sprache, oft Englisch, sind sie einfacher erlernbar. Damit höhere Programmiersprachen von einem Prozessor ausgeführt werden können, müssen sie zu erst mit einem Übersetzer (Compiler oder Interpreter) in Maschinencode übertragen werden.¹

In höheren Programmiersprachen würde die If-Abfrage wie folgt aussehen:

```
if(x=24)
{
  wenn x = 24, führe diesen Programmteil aus;
}
else
{
  wenn x ≠ 24, führe diesen Programmteil aus;
}
```

Höhere Programmiersprachen werden in vier Kategorien eingeteilt. Auch hier werde ich mich auf diejenigen beschränken, die in meiner Arbeit von Bedeutung sind.

Imperative Programmiersprachen:

Ein Programm besteht aus einer Folge von Anweisungen oder Befehlen. Die verglichene Programmiersprache NXC ist dieser Kategorie zuzuordnen.²

Funktionale Programmiersprachen:

Programme berechnen Funktionen, die Eingabedaten in Ausgabedaten abbilden. Robolab kann dieser Kategorie zugeordnet werden.³

1 Sinngemäss aus: siehe LV [6].
2 Sinngemäss aus: siehe LV [6].
3 Sinngemäss aus: siehe LV [6].

4.3 Aktoren und Sensoren des NXTs

- *Aktoren*

Mit Hilfe der Motoren ist es dem NXT möglich, Bewegungen auszuführen.

- *Rotation*

Jeder *Servomotor* ist mit einem *Rotationssensor* ausgestattet, der eine präzise Steuerung des NXTs ermöglicht. Die Motorumdrehungen werden entweder in Grad, mit einer Genauigkeit von +/- einem Grad, oder in ganzen Umdrehungen gemessen. Eine Umdrehung entspricht 360 Grad.



Abbildung 3: Servomotor
(aussen)

- *Bremsen/Auslaufen*

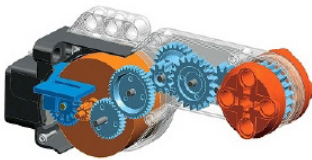


Abbildung 4: Servomotor
(innen)

Die Motoren des NXTs unterstützen zwei Arten zum Anhalten des Roboters. Bei der einen Art wird der Motor abgebremst. Dies hat zur Folge, dass der Roboter, sobald er den entsprechenden Befehl im Programm erreicht, stehen bleibt. Die andere Art dagegen schaltet nur die Stromversorgung des Motors aus, sodass die Motoren noch ein klein wenig weiterdrehen.

- *Synchronisation*

Ein Roboter, der sowohl auf der linken als auch auf der rechten Seite einen Motor besitzt, soll geradeaus fahren. Schon nach kurzer Zeit kommt der Roboter von seiner programmierten Fahrbahn ab. Dieses Verhalten ist auf die Motoren zurückzuführen. Es ist schlichtweg nicht möglich identische Motoren zu produzieren. Ist nun der eine Motor ein wenig schneller als der andere, kommt der Roboter bereits von seiner Fahrbahn ab.

Damit dieser Fehler behoben werden kann, wurde die Funktion Synchronisation eingeführt. Mit dieser Funktion kann man definieren, welche Motoren synchronisiert werden. Einer dieser synchronisierten Motoren wird zusätzlich als *Master* definiert, die übrigen Motoren werden als *Slaves* bezeichnet. Der als Master definierte Motor ist der Führer unter den Motoren, die Slaves dagegen haben eine reaktive Rolle. Im Programmcode muss lediglich die Geschwindigkeit des Masters definiert werden. Die Geschwindigkeit der Slaves wird aus folgender Formel errechnet:

$$\text{Slave Geschwindigkeit} = \text{Master Geschwindigkeit} * \text{Drehzahlverhältnis} : 100$$

Mit dem Drehzahlverhältnis wird das Verhältnis der Anzahl Umdrehungen zwischen Master und Slave angegeben. Haben wir ein Drehzahlverhältnis von 2:1, so führt der Slave die Hälfte der Umdrehungen des Masters durch.

Die ganze Fahrt über werden die Positionen der Motoren, die durch den integrierten *Rotationssensor* gemessen werden, miteinander verglichen. Wird nun ein Motor aus irgendeinem Grund gestoppt, so stoppen auch all die anderen synchronisierten Motoren. ¹

1 Sinngemäss aus: siehe LV [7].

- *Sensoren*

Dank der Sensoren kann der NXT auf verschiedenen Ereignisse reagieren.

- *Berührungssensor:*



Abbildung 5: Berührungssensor

Der Berührungssensor des Roboters funktioniert wie ein Schalter. Er kann auf Drücken und Loslassen reagieren. In unberührtem Zustand besitzt der Sensor den Wert 0. Wird der Sensor berührt, so ändert sich sein Wert auf 1.¹

Der NXT selbst kann nicht zwischen berührt und unberührt unterscheiden. Für ihn gibt es nur die Zustände “Strom fließt” und “Strom fließt nicht”. Dem Zustand “Strom fließt nicht” ist der Wert 0 zugeordnet, da kein Strom vorhanden ist, dem Zustand “Strom fließt” der Wert 1.

- *Lichtsensoren*

Mit Hilfe des Lichtsensors wird es dem Roboter ermöglicht, die Helligkeit des Lichtes festzustellen. Der Lichtsensor kann also keine Farben erkennen sondern nur zwischen deren Helligkeit unterscheiden. Die aufgezeichneten Werte werden standardmässig auf einer Skala zwischen 0 und 100 angegeben. Der Wert 0 steht dabei für kein Licht und der Wert 100 für sehr helles Licht.²



Abbildung 6: Lichtsensor

- *Geräuschsensor*



Abbildung 7: Geräuschsensor

Mit dem Geräuschsensor kann der Lautstärkepegel von Geräuschen ermittelt werden. Auch kann der Geräuschsensor bestimmte Klangmuster erkennen. Beispielsweise kann er auf einmaliges Klatschen komplett anders reagieren als auf zweimaliges. Die Erkennung kann sowohl in dB als auch in dBA erfolgen.³

In Dezibel (dB) wird der Schalldruck gemessen, sprich der tatsächliche Geräuschpegel. In Dezibel A (dBA) hingegen wird die A-Skala verwendet, die die Lautstärke wiedergibt, wie sie das menschliche Gehör wahrnimmt.⁴

Geräusche von bis zu 90 dB können mit dem Geräuschsensor aufgezeichnet werden.⁵

1 Sinngemäss aus: siehe LV [8]. Berührungssensor. S. 23.

2 Sinngemäss aus: siehe LV [8]. Lichtsensor. S. 27f.

3 Sinngemäss aus: siehe LV [9].

4 Sinngemäss aus: siehe LV [10].

5 Sinngemäss aus: siehe LV [8]. Geräuschsensor. S. 25.

- *Ultraschallsensor*

Der Ultraschallsensor wird zum Messen von Distanzen verwendet. Die Messungen können Wahlweise entweder in Zentimetern oder in Inch erfolgen. Abstände zwischen 0 und 2.5 Metern können mit einer Genauigkeit von +/- 3 cm bestimmt werden.

Um eine Distanz zu messen, sendet der Ultraschallsensor eine Welle aus und misst die Zeit, die von der Welle benötigt wird, bis sie beim Objekt angelangt und wieder zurückgekehrt ist. Aus dem Wert der Zeit wird anschliessend die Strecke berechnet. Je länger es dauert bis das Signal zum Sensor zurückkehrt, desto weiter ist das Objekt entfernt.¹



Abbildung 8: Ultraschallsensor

1 Sinngemäss aus: siehe LV [8]. Ultraschallsensor. S. 29

4.4 Funktionen der Programmiersprachen

- *Aufzeichnungsmöglichkeiten der Sensorwerte*

Die Programmiersprachen erlauben das Verwenden verschiedener Sensormodi. Die Modi legen fest, wie die Rohwerte des Sensors verarbeitet werden. Es eignen sich allerdings nicht alle Modi für alle Sensoren.

Im *Rawmodus* werden die Werte verarbeitet, wie sie der Sensor aufzeichnet. Die Werte können zwischen 0 und 1023 liegen.

Im Sensormodus *Bool* liegen die Werte zwischen 0 und 1. Ist der Rohwert höher als 562, wird der Wert 0 ausgegeben, ansonsten der Wert 1. Dieser Modus wird meistens beim Berührungssensor eingesetzt.

Die Modi *Celcius* und *Fahrenheit* sind nur beim Einsatz von Temperatursensoren sinnvoll. Die gemessene Temperatur kann wahlweise in Celsius oder Fahrenheit weiterverarbeitet werden.

Im *Percentmodus* werden die Rohwerte in einen Wert zwischen 0 und 100 umgerechnet. Standardmässig werden die Werte des Lichtsensors in diesem Modus aufgezeichnet.

Ein weiterer Modus wäre der *Rotationsmodus*, dieser wird ausschliesslich für den Rotationsensor verwendet.

Die Sensormoden *Edge* und *Pulse* zählen Übergänge. Unter Übergängen versteht man die Änderung von einem tiefen zu einem hohen Rohwert oder umgekehrt. Der Modus *Pulse* zählt nur Übergänge von einem tiefen zu einem hohen Wert. Der Modus *Edge* dagegen zählt beide Übergänge. Im Modus *Pulse* zählt zum Beispiel das Berühren und Loslassen des Berührungssensors als einen Übergang, im *Edge* Modus dagegen als zwei Übergänge.¹

- *Aufgaben und Subroutinen*

Wird ein Programmteil mehrmals in einem Programm verwendet, bieten die Programmiersprachen verschiedene Möglichkeiten an, um die mehrmalige Verwendung zu erleichtern.

- *Subroutinen*

Ein mehrfach verwendeter Programmteil kann beispielsweise einer Subroutine hinzugefügt werden. Subroutinen werden nur einmal auf dem NXT gespeichert. Dadurch wird weniger Speicherplatz in Anspruch genommen. Jeder Subroutine wird ein eindeutiger Name zugeordnet. Über diesen Namen kann der Programmteil (in der Subroutine) an beliebig vielen Stellen im Programm aufgerufen werden.

In einigen Programmiersprachen können Subroutinen Parameter verarbeiten, in anderen nicht. Parameter sind variable Werte, die bei jedem Aufruf der Subroutine individuell festgelegt werden können.

Subroutinen werden in der Regel für grössere Programmteile, welche mehrmals im selben Task verwendet werden, eingesetzt. Der NXT lässt insgesamt nicht mehr als 255 Subroutinen und Threads resp. Tasks zu.

¹ Sinngemäss aus: siehe LV [11]. Sensor mode and type. S. 35.

- *Inline Funktionen*

Inline Funktionen werden im Gegensatz zu Subroutinen nicht nur einmal gespeichert, sondern an jede Stelle kopiert, an der sie verwendet werden. Dies hat zur Folge, dass sie mehr Speicher belegen als Subroutinen.

Im Gegensatz zu Subroutinen können sie beliebig oft benutzt werden. Inline Funktionen werden für kleinere Programmteile, die an verschiedenen Stellen in verschiedenen Tasks verwendet werden, eingesetzt.

- *Makros*

Makros werden nur für kleine Programmteile verwendet und können Parameter verarbeiten.

- *Kontrollfunktionen*

Kontrollfunktionen erlauben den Ablauf eines Programms zu kontrollieren.

- *Programm-Kontrollfunktionen*

Mit Hilfe von Programm-Kontrollfunktionen können Tasks und der Zugriff auf Selbige gesteuert werden. So kann festgelegt werden, ob sie nacheinander oder parallel ausgeführt werden sollen. Auch ist es möglich den Zugriff auf einen Task zu blockieren, solange er in Verwendung ist. Ein Task kann ausserdem sofort beendet werden, und das Programm kann zu einem bestimmten Task weitergeleitet werden.

- *Verzweigungen*

Ist eine Entscheidung innerhalb eines Programms erforderlich, so werden Verzweigungen zur Hilfe gezogen. Eine Verzweigung prüft, ob ein Wert dem einen oder dem anderen Kriterium entspricht. Normalerweise können Verzweigungen zwischen zwei Fällen unterscheiden. Es gibt allerdings spezielle Arten von Verzweigungen, bei denen mehrere Fälle ausgewertet werden können.

- *Sprünge/Wiederholungen*

Mit Sprüngen kann innerhalb eines Programms von der einen Stelle zu einer anderen gesprungen werden. Diese Funktion wird verwendet um in einem bestimmten Fall ein Teil des Programms zu überspringen. Durch die Verwendung von Sprüngen wird der Ablauf komplexer Programme nur noch schwer kontrollierbar. Deshalb werden sie heute nur noch selten verwendet.

Es gibt verschiedene Arten von Wiederholungen. Die einen Wiederholungen können so konfiguriert werden, dass sie einen bestimmten Teil des Programms eine eingestellte Anzahl Mal durchlaufen, andere Wiederholungen können den Programmteil so oft wiederholen, bis ein Argument ein bestimmtes Kriterium erfüllt.

- *Kommunikation zwischen Robotern*

Damit die Kommunikation zwischen zwei oder mehreren NXTs möglich ist, müssen sie zuerst über das Bluetooth-Menü des Bausteines miteinander verbunden werden.

Jener Baustein, der die Kommunikation startet, ist der Master. Er kann zeitgleich mit drei Bausteinen verbunden sein. Der Slave kann Nachrichten zu zehn verschiedenen *Briefkästen* senden. Diese Art von Kommunikation ist unter dem Namen Master-Slave Kommunikation bekannt.

- *Nummern austausch*

Eine andere Art der Kommunikation ist der Austausch von Nummern. Der Master sendet eine Nummer an einen Slave. Der Slave, der kontinuierlich seinen Briefkasten prüft, sendet eine Bestätigung an den entsprechenden Master zurück. Erst wenn der Master die Bestätigung erhalten hat, sendet er die nächste Nachricht. Oft kommt der Nummern austausch beim Initialisieren einer Bluetoothverbindung zum Einsatz. Mit diesem Austausch kann sichergestellt werden, dass die Kommunikation zwischen den Robotern einwandfrei funktioniert.

- *Befehls austausch*

Diese Art der Kommunikation erlaubt das Steuern eines zweiten Roboters. Der Master kann so spezielle Befehle, zum Beispiel zum Abspielen von Musik oder zum Steuern der Motoren, an seinen Slave senden. Der Slave benötigt dafür nicht einmal ein Programm, da die Nachrichten von der *Firmware* empfangen und verwaltet werden.

4.5 *Vom Entwurf bis zum Programm auf dem NXT*

Da ich mit dem NXT und den Programmiersprachen nun vertraut war, konnte ich mich mit der Entstehung eines Programms auseinandersetzen.

- *Planung/Entwurf*

Bevor der Programmierer mit dem Programmieren beginnt, überlegt er sich, welche Funktionen das endgültige Programm beinhalten soll und wie das Programm ablaufen soll. Sehr oft wird der Ablauf des Programms auf Blättern skizziert, da dies das spätere Programmieren enorm erleichtern wird.

Diese Planung fand in meinem Fall mit dem Erstellen der Aufgaben statt. Dies war mein Leitfaden zum Programmieren der Programme.

- *Programmierung*

Das Programm muss nun vom Papier in den Computer übertragen werden. Um das Programm auf dem Computer programmieren zu können, wird vom Programmierer eine Programmierumgebung benutzt. Die Programmierumgebung ist ebenfalls ein Programm, das viele nützliche Funktionen bereitstellt, die das Leben eines Programmierers erleichtern. Die Programmierumgebung ist das Textverarbeitungsprogramm des Programmierers. Es gibt spezifische Programmierumgebungen, welche nur eine Programmiersprache unterstützen und universelle, welche für mehrere Programmiersprachen benutzt werden können.

Ich benutzte zwei Programmierumgebungen: Robolab als eine spezifische Programmierumgebung für die Sprache Robolab und eine universelle namens Bricx Command Center für die Sprachen NBC und NXC.

- *Compilieren/Downloaden*

Ist das Programm einmal verfasst, muss es in Maschinencode übersetzt werden. Dieser Übersetzungsvorgang wird als compilieren bezeichnet. Bevor der Compiler mit dem Übersetzen beginnt, wird das Programm auf Fehler untersucht. Tritt ein solcher auf, weist eine Meldung den Programmierer auf den Fehler hin. Erst wenn das Programm fehlerfrei ist, wird der Übersetzungsvorgang gestartet.

In meinem Fall musste das compilierte Programm noch vom Computer auf den NXT gelangen. Dieser Übertragungsprozess wird als downloaden bezeichnet. Bevor das Programm gedownloadet wird, wird es nochmals compiliert.

5 Durchführung

5.1 Vorbereitungsphase

- *Roboterbau*

Anfänglich wollte ich den Vergleich der Programmiersprachen mittels Alpha Rex, einem humanoiden Roboter, durchführen. Ein humanoider Roboter ist ein Roboter, dessen Gestalt der eines Menschen ähnelt. Da Alpha Rex allerdings Fortbewegungsprobleme hatte, musste ich den Roboter kurzerhand zu einem TriBot, einem Fahrzeug-ähnlichen Roboter, umbauen. Mit dem TriBot konnten sämtliche Vergleiche durchgeführt werden.



Abbildung 9: Alpha Rex



Abbildung 10: TriBot

- *Systemkonfiguration*

Nachdem der Roboter gebaut war, galt es den Computer für den Programmiersprachenvergleich einzurichten. Dazu musste ich die aktuellste Version des Bricx Command Center (3.3), welches die grafische Oberfläche für NBC und NXC bietet, aus dem Internet beziehen und installieren. Ebenso mussten die Compiler (Version 1.0.1 b28) für NBC und NXC eingerichtet werden.

Zusätzlich musste ich Robolab installieren und auf die aktuellste Version (2.9.3) aktualisieren. Die Aktualisierung war erforderlich, da die Version 2.9.3 um viele neue Funktionen ergänzt wurde.

5.2 Lernphase

- *Softwareumgebung*

Bevor ich mit dem Erlernen der Programmiersprache beginnen konnte, galt es die Softwareumgebung näher kennen zu lernen. So nahm ich die einzelnen Funktionen genauer unter die Lupe und versuchte deren Nutzen in der Praxis nachzuvollziehen.

- *Programmiersprachen*

Mit der Programmierumgebung war ich nun vertraut, somit konnte mit dem Erlernen der Programmiersprachen begonnen werden.

Zum Erlernen der beiden textbasierten Programmiersprachen verwendete ich zuerst deren Tutorials. Die Tutorials führen den Programmierer Schritt für Schritt in die Programmiersprache ein. Zum Vertiefen der Kenntnisse benutzte ich die entsprechenden Manuals, in welchen alle verfügbaren Funktionen aufgeführt sind. Sowohl die Tutorials als auch die Manuals sind auf der Website der jeweiligen Programmiersprache zum Download verfügbar.

Für das Erlernen der Robolab Programmiersprache benötigte ich keine Anleitung, da die verwendeten Symbole nahezu selbsterklärend sind.

- *Erste Anwendungen*

Die Programmiersprachen waren somit erlernt, und es galt nun die Theorie in die Praxis umzusetzen. Zu Beginn schrieb ich ganz kleine Programme, um das soeben Erlernte anwenden zu können. Die Komplexität der Programme steigerte ich mit zunehmender Erfahrung.

Dass die neuste Version nicht immer die beste ist, musste ich schon bald feststellen. In einem Testprogramm lieferte der Ultraschallsensor andauernd falsche Werte. Als ich das Programm zu Testzwecken in Robolab programmierte, funktionierte der Ultraschallsensor einwandfrei.

Anfänglich ging ich von einem Fehler in meinem Programm aus, jedoch konnte ich den Fehler einfach nicht finden. Auf einmal kam mir die Idee, dass ich eine andere Version (1.0.1 b32) des Compilers verwenden könnte. Und siehe da, der Ultraschallsensor lieferte wieder die erwarteten Werte.

Dies ist der Nachteil von Software, die sich noch in der *Betaphase* befindet.

5.3 Vergleichsphase

- *Kriterien*

Damit ein effizienter Vergleich der Programmiersprachen möglich war, war es nötig, zuerst einen Kriterienkatalog zu erstellen. Anhand dieses Kriterienkatalogs verglich ich die einzelnen Programmiersprachen. Die Hauptkriterien, nach denen die Programmiersprachen verglichen wurden, sind Motoren, Sensoren, Kontrollfunktionen, Aufgaben und Subroutinen und Kommunikation.

- *Aufgaben*

Nachdem ich die Kriterien festgelegt hatte, konnte ich mit dem Erstellen der Aufgaben beginnen. Insgesamt erstellte ich zwei Aufgaben, welche zusammen alle Kriterien beinhalteten. Die Aufgaben bauten aufeinander auf, und waren von unterschiedlicher Komplexität. Im Anhang können die Aufgaben eingesehen werden.

6 Auswertung

6.1 Actoren

- *NBC*

NBC unterstützt zwei Arten zum Bewegen der Motoren: Auf der einen Seite können die Motoren vorwärts oder rückwärts bewegt werden, auf der anderen Seite gibt es die Möglichkeit die Motoren eine bestimmte Anzahl Grad rotieren zu lassen. Die Befehle **OnFwd** und **OnRev** erlauben das Einstellen der Anschlüsse und der Geschwindigkeit der Motoren. Beim Rotationsbefehl **RotateMotor** kann zusätzlich die Anzahl Grad eingestellt werden. Daneben gibt es noch die erweiterten Befehle, bei welchen spezielle Konstanten zum Zurücksetzen von verschiedenen Zählern konfiguriert werden können. Die Leistung der Motoren lässt sich stufenlos zwischen 0 und 100 einstellen.

Die Tachos erlauben das Aufzeichnen der Umdrehungen des Motors in Grad. Will man die Anzahl vollständiger Umdrehungen messen, so muss man den Wert in Grad durch 360 dividieren. Eine Funktion zum Zählen von vollständigen Umdrehungen würde ich bevorzugen.

Mit NBC können verschiedene Informationen des Motors ausgegeben werden. Dazu zählen zum Beispiel die aktuelle Leistung, der aktuelle Tachostand, der aktuelle Modus und viele Weitere. Damit die Werte einer Variable zugeordnet werden können, muss der Befehl **getout** verwendet werden.

NBC unterstützt sowohl das Kontrollieren der Leistung der Motoren als auch deren Synchronisation.

Sowohl das Abbremsen als auch das Auslaufen der Motoren wird von NBC unterstützt.

- *NXC*

Im Gegensatz zu NBC lassen sich in NXC die Werte der Statusinformationen direkt einer Variable zuordnen. Ansonsten unterscheiden sich die Programmiersprachen in diesen Punkten kaum.

- *Robolab*

Um den Motor vorwärts resp. rückwärts zu drehen, gibt es vordefinierte Blöcke für die verschiedenen Motoren und Richtungen, als auch einen Block, bei welchem die zu steuernden Motoren manuell festgelegt werden können. Durch die vorgefertigten Blöcke wird ein schnelleres Arbeiten ermöglicht. Zum Rotieren der Motoren stehen drei Blöcke zur Verfügung. Ein Block erlaubt das präzise Rotieren des Motors auf die eingestellte Gradzahl genau. Bei den beiden anderen Blöcken rotiert der Motor, bis er die angegebene Gradzahl überschritten hat. Einer der beiden Blöcke setzt den Rotationssensor zurück, der andere dagegen nicht.

Die Leistung der Motoren kann durch fünf verschiedenen Stufen eingestellt werden. Ein präzises Einstellen der Leistung ist daher nicht möglich.

Im Gegensatz zu NBC und NXC steht in Robolab nur ein Tacho zur Verfügung der die Umdrehungen in Grad misst. So kann auch nur dieser Wert von den Motoren ausgelesen werden. Das Auslesen von weiteren Statusinformationen ist nicht möglich.

Wie NBC und NXC erlaubt Robolab das kontrollieren der Geschwindigkeit der Motoren als auch deren Synchronisation.

Auch in Robolab wird sowohl das Auslaufen als auch das Abbremsen der Motoren unterstützt.

6.2 Sensoren

- *NBC*

NBC unterstützt alle Standardsensoren des NXTs. Unter Standardsensoren verstehe ich den Ultraschallsensor, den Geräuschsensor, den Lichtsensor und den Berührungssensor. Ebenfalls werden alle Sensorfunktionen von NBC unterstützt.

Bevor die Sensoren in NBC verwendet werden können, müssen sie initialisiert werden. Dazu wird dem Programm mitgeteilt, an welchem Anschluss sich welcher Sensor befindet. Es gibt zwei Möglichkeiten zum Initialisieren: normal und erweitert. Bei der normalen Initialisierung wird mit einem sensorspezifischen Befehl z.B. **SetSensorTouch** der Anschluss des Sensors dem Programm mitgeteilt. Bei der erweiterten Initialisierung wird mit dem für alle Sensoren gültigen Befehl **SetSensorType** dem Programm der Typ des Sensors und dessen *Port* mitgeteilt. Mit dem zweiten für alle Sensoren gültigen Befehl **SetSensorMode** wird dem Programm der Aufzeichnungsmodus des Sensors übermittelt. Die erweiterte Initialisierung erlaubt dem Programmierer ein individuelles Konfigurieren der Sensoren, da bei der normalen Initialisierung die Modi nicht definiert werden können. Folgende Aufzeichnungsmoden stehen in NBC zur Verfügung: Rawmodus, Boolmodus, Celcius- /Fahrenheitmodus, Percentmodus und Edge-/Pulsemodus. Um einen aufgezeichneten Wert eines Sensors im Programm zu verwenden, wird der Befehl **ReadSensor** benötigt. Für den Ultraschallsensor gibt es den speziellen Befehl **ReadSensorUS**.

Das Einstellen von benutzerdefinierten Sensoreigenschaften, z.B. eine benutzerdefinierte Prozentskala oder eine benutzerdefinierte Nullpunktabweichung, ist nicht möglich.

Statusinformationen beispielsweise der Sensormodus können nur in Zahlen ausgegeben werden. Erschwerend kommt hinzu, dass auf dem NXT die Werte dezimal ausgegeben werden, in der Anleitung der Programmiersprache werden die Werte dagegen *hexadezimal* angegeben.

- *NXC*

In NXC wird kein spezieller Befehl zum Auslesen von aufgezeichneten Sensorwerten benötigt, die Angabe des Sensoranschlusses genügt. Zusätzlich unterstützt NXC das Festlegen von benutzerdefinierten Sensoreigenschaften.

- *Robolab*

Die Sensoren müssen in Robolab vor deren Verwendung nicht initialisiert werden. Ein weiterer Unterschied zu NBC und NXC besteht darin, dass in Robolab keine Aufzeichnungsmoden festgelegt werden können. Wie in NBC können auch in Robolab keine benutzerdefinierten Sensoreigenschaften festgelegt werden. Des Weiteren ist es in Robolab nicht möglich Statusinformationen der Sensoren auszulesen.

6.3 Kontrollfunktionen

- *NBC*

NBC bietet eine Fülle an Programm-Kontrollfunktionen. Die Befehle *precedes* und *follows* erlauben festzulegen, wie und in welcher Reihenfolge die Threads ausgeführt werden. Mit dem Befehl *precedes* werden die Threads parallel ausgeführt, mit dem Befehl *follows* dagegen nacheinander.

Der Befehl *acquire* greift auf einen *Mutex* zu. Der Befehl *release* gibt den *Mutex* wieder frei. Somit wird sichergestellt, dass nur immer ein Thread Zugriff auf den Code zwischen den Befehlen *acquire* und *release* hat. Um ein Thread zu beenden, können die Befehle *exit* und *exitto* verwendet werden. *exit* beendet den aktuellen Thread und plant die Ausführung von abhängigen Threads. *exitto* dagegen beendet den aktuellen Thread und das Programm wechselt zu dem angegebenen Thread.

Vergleiche können mit den Befehlen *cmp* und *tst* realisiert werden. Der Befehl *cmp* vergleicht zwei Werte, während *tst* einen Wert dem Wert 0 gegenüberstellt. Beide Befehle geben das Ergebnis in einer Variable aus.

Ähnlich den Befehlen *cmp* und *tst* sind die Befehle *brcmp* und *brtst*. Sie werden für Verzweigungen benötigt. Im Gegensatz zu *cmp* und *tst* können diese Befehle je nach Ergebnis entscheiden, ob die eine oder die andere Aktion ausgeführt werden soll. Die Befehle *brcmp* und *brtst* sind meiner Ansicht nach sehr unübersichtlich, da mit *Labels* gearbeitet werden muss.

Um von einer Stelle im Programm zu einer anderen Stelle zu gelangen, kann der Befehl *jmp* verwendet werden.

NBC stellt keine spezifische Befehle zur Verfügung, um bestimmte Teile eines Programms zu wiederholen. Dies ist, wie ich finde, ein grosser Nachteil dieser Programmiersprache, da diese Funktion von Programmierern sehr häufig verwendet wird.

- *NXC*

Die Programm-Kontrollfunktionen von NXC unterscheiden sich nur minim von den NBC-Programm-Kontrollfunktionen. Der NBC-Befehl *exit* ist in NXC unter dem Namen *stop* verfügbar. Mit dem Befehl *stop* kann man allerdings die Ausführung von abhängigen Tasks nicht planen.

In NXC gibt es keine spezielle Befehle, die es dem Programmierer erlauben, Werte miteinander zu vergleichen und das Ergebnis in einer Variable abzulegen.

```
brcmp GT, Then, x, 24
Else:
    //hier geschriebener Code wird ausgeführt
    //wenn x kleiner oder gleich 24 ist
    jmp EndIf
Then:
    //hier geschriebener Code wird ausgeführt
    //wenn x grösser als 24 ist
EndIf:
    //hier ist die Abfrage zu Ende
```

Abbildung 11: Verzweigung in NBC

```
if (x > 24)
{
    //hier geschriebener Code wird ausgeführt
    //wenn x grösser als 24 ist
}
else
{
    //hier geschriebener Code wird ausgeführt
    //wenn x kleiner oder gleich 24 ist
}
```

Abbildung 12: Verzweigung in NXC

Dagegen bietet NXC gleich zwei Möglichkeiten um Verzweigungen zu programmieren. Der erste Befehl ist der **if** Befehl. Dieser entspricht dem **brcmp** Befehl von NBC. Der **if** Befehl ist allerdings viel übersichtlicher als der **brcmp** Befehl, da er nicht mit Labels arbeitet. Beim **if** Befehl können die auszuführenden Aktionen direkt unter den Befehl programmiert werden. Dies erhöht die Übersichtlichkeit beträchtlich. Neben dem Befehl **if** bietet NXC auch den Befehl **switch**. Im Gegensatz zum **if** Befehl kann der **switch** Befehl nicht nur zwischen zwei Aktionen entscheiden, sondern kann eine Vielzahl von Aktionen je nach Wert ausführen.

Sprünge werden in NXC mit dem Befehl **goto** realisiert.

NXC bietet viele verschiedene Möglichkeiten um einzelne Teile eines Programms zu wiederholen. Der Befehl **repeat** gibt an, wie oft ein Programmteil durchlaufen werden soll. Mit dem Befehl **while** wird ein Teil des Programms solange ausgeführt, wie das Argument wahr ist.

Nehmen wir als Argument $x=10$: Nun wird der Programmteil solange ausgeführt, bis die Variable x nicht mehr den Wert 10 beinhaltet.

Ein weiterer Befehl ist der **do-while** Befehl. Er ist nahezu identisch mit dem **while** Befehl, nur dass beim **do-while** Befehl der zu wiederholende Teil mindestens einmal ausgeführt wird. Der **for** Befehl führt zuerst den ersten Teil des Teilprogramms aus und prüft danach, ob die Bedingung immer noch wahr ist. Wenn dies der Fall ist, wird auch der restliche Teil des Teilprogramms ausgeführt. Die letzte Möglichkeit um Programmteile zu wiederholen ist der **until** Befehl. Der Programmteil wird solange ausgeführt, bis das Argument wahr wird.

- **Robolab**

In Robolab wird der Zugriff auf einzelne Tasks mit Hilfe von Prioritäten gesteuert. Zuerst werden jene Tasks mit der höchsten Priorität ausgeführt, dann jene mit der zweithöchsten und so weiter. Robolab bietet Befehle um Tasks zu starten, zu beenden und zu löschen.

Wie in NXC ist auch in Robolab kein spezifischer Befehl vorhanden um Vergleiche durchzuführen und das Ergebnis in einer Variable zu speichern. In Sachen Verzweigungen bietet Robolab einen eingeschränkteren Funktionsumfang im Vergleich zu den anderen Programmiersprachen. Um Verzweigungen zu erstellen, sind in Robolab vordefinierte Blöcke verfügbar. Es gibt zwei verschiedene Typen solcher Verzweigungen: Ein Block kann den Wert auf grösser oder kleiner/gleich prüfen, der andere Block prüft die Werte auf gleich oder ungleich einem eingestellten Wert.

Sprünge innerhalb des Programms werden in Robolab mit verschiedenfarbigen Blöcken realisiert.

Um einen Teil eines Programms zu wiederholen, bietet Robolab eine Fülle an vordefinierten Blöcken.

6.4 Aufgaben und Subroutinen

- *NBC*

Jeder Thread wird in NBC mit dem Befehl **endt** geschlossen. Damit die Programme übersichtlicher gestaltet werden können, bietet NBC sogenannte Labels. Labels sind nichts weiter als Überschriften, die dem Unterteilen des Programms dienen.

Um eine Subroutine im Programm aufzurufen, muss der Befehl **call** verwendet werden. Eine in NBC definierte Subroutine kann keine Parameter verarbeiten. Am Ende jeder Subroutine muss der Befehl **return** eingefügt werden, damit die Subroutine zum Hauptprogramm zurückkehrt.

Neben Subroutinen können in NBC auch Makros definiert werden. Sie können im Gegensatz zu Subroutinen Parameter verarbeiten.

- *NXC*

Um einen Task, eine Subroutine, eine Inline Funktion oder ein Makro in NXC abzuschliessen, genügt das Schliessen der geschweiften Klammer. Wie in NBC gibt es auch in NXC Labels zum Gliedern der Programme.

In NXC können Subroutinen Parameter verarbeiten. Zum Aufrufen einer Subroutine wie auch einer Inline Funktion oder eines Makros genügt deren Name. Neben Subroutinen und Makros werden in NXC auch Inline Funktionen unterstützt.

- *Robolab*

Tasks werden in Robolab mit einer grünen Ampel begonnen und mit einer roten Ampel beendet. Zur Gliederung der Programme kann das Textwerkzeug verwendet werden.

Subroutinen werden mit einem speziellen Block begonnen und ebenfalls mit einer Ampel beendet. Auch in Robolab können Subroutinen keine Parameter verarbeiten. Eigene Funktionen wie Inline Funktionen oder Makros, stellt Robolab keine zur Verfügung.

6.5 Kommunikation

- *NBC*

NBC nutzt die Bluetooth-Funktion des NXTs, um die Kommunikation zwischen Robotern zu ermöglichen. Der Status der Bluetoothverbindung wird mit dem Befehl ***BluetoothStatus*** abgerufen. In NBC ist es nicht möglich, den Status der Bluetoothverbindung manuell festzulegen.

Bei einer Master-Slave Kommunikation dürfen nicht mehr als 58 Zeichen je Befehl übertragen werden.

Der direkte Befehlsaustausch ist leider nur für einige wenige Befehle verfügbar. Die Funktion der direkten Kommunikation bietet riesige Vorteile in der Fernsteuerung eines zweiten Roboters. Doch in diesem eingeschränkten Umfang, wie die Funktion aktuell vorliegt, ist sie nahezu nutzlos.

- *NXC*

Im Gegensatz zu NBC erlaubt NXC das manuelle Festlegen des Status der Bluetoothverbindung. Ansonsten gibt es keine Unterschiede zwischen den beiden Programmiersprachen.

- *Robolab*

Leider war es nicht möglich, die Kommunikation in Robolab zu testen, da Robolab die Bluetoothschnittstelle noch nicht unterstützt. Robolab würde zwar die Kommunikation über Infrarot unterstützen, da der NXT allerdings über keine Infrarotschnittstelle verfügt, kann diese Kommunikationsart nicht verwendet werden.

Von den Entwicklern von Robolab wäre ein *Patch* verfügbar, welcher Robolab um Bluetoothfunktionen erweitert. Dieser lässt sich jedoch nur anwenden, wenn der Computer auf dem der Patch angewandt wird, via Bluetooth mit dem NXT verbunden ist.

7 Eindrücke anderer Personen

7.1 Oberägeri

Während meiner Maturaarbeitszeit hatte ich die Gelegenheit, den LEGO Roboter in einer Schulklasse in Oberägeri vorzustellen. Die Schüler waren zwischen acht und zwölf Jahren alt. Meine Aufgabe bestand darin, den Schülern das Programmieren mittels Robolab etwas näher zu bringen. Diese Gelegenheit nutzte ich natürlich für meine Maturaarbeit, um die Eindrücke der Schüler einzufangen.

Da die meisten Schüler Neulinge auf dem Gebiet der Programmierung waren, begann ich damit, den Schülern Schritt für Schritt die grundlegendsten Funktionen vorzuführen. Die Begeisterung der Schüler war schon gross, als sie den Roboter sahen. So konnten sie es denn kaum abwarten, den Roboter in Aktion zu sehen.

Zuerst zeigte ich den Schülern, wie sie den Roboter dazu bringen können, eine bestimmte Zeit vorwärts zu fahren. Nachdem meine Demonstration zu Ende war, konnten die Schüler das soeben gesehene Programm selbst nachprogrammieren. Die Schüler meisterten diese Hürde recht gut. Am meisten Schwierigkeiten hatten sie mit dem Verbinden der einzelnen Blöcke. Auf der einen Seite führte die automatische Verbindung für Verwirrung, auf der anderen Seite war es den Schülern nicht immer klar, wie die Blöcke zu verbinden sind. Die Bedeutung der Blöcke konnten die Schüler problemlos deuten. Auch konnten die Schüler die Funktionen von unbekanntem Blöcken ohne weiteres identifizieren.

Die Schüler konnten nun das selbst programmierte Programm auf den RCX übertragen. Als sie sahen, dass alles einwandfrei funktioniert, war ihre Begeisterung noch um einiges grösser. Sofort wollten sie ans Werk machen und selbstständig die Zeit der Fahrt verlängern, oder die Leistung des Motors erhöhen.

Als ich ihnen das Rückwärtsfahren erläutert hatte, stellte ich ihnen eine Aufgabe, welche die soeben erlernten Funktionen beinhaltet. Die meisten Schüler konnten diese Aufgabe lösen, als hätten sie schon eine Woche Programmierkurs absolviert. Wenn Probleme auftraten, lag es häufig an der Firmware des RCXs. War den Schülern etwas unklar, erkundigten sie sich sofort bei mir.

Schon bald machten sich die Schüler daran, selbstständig ihre Programme zu verändern. War ihnen die Funktion eines Blockes nicht klar, fragten sie sofort nach. Das Interesse als auch die Begeisterung waren riesig. Für die Schüler war es ein tolles Erfolgserlebnis, einer Maschine beizubringen, was sie tun soll, und dann zu sehen, dass sie wirklich auch macht, was sie wollen. Dieses Erfolgserlebnis gab den Schülern neue Motivation.

Ich war erstaunt, wie schnell die Schüler sich mit der Programmiersprache vertraut fühlten und wie gross ihre Begeisterung und ihre Motivation war. Und lief einmal nicht alles nach Plan, versuchten die Schüler selbst das Problem in den Griff bekommen.

7.2 *Experimente mit Testpersonen*

Um weitere Eindrücke zu sammeln, führte ich verschiedene Experimente mit weiteren Personen durch. Wie in Oberägeri wurden auch diesen Personen zuerst die Grundlagen der einzelnen Programmiersprachen erläutert. Allerdings machten diese Personen nicht nur mit einer, sondern mit allen drei Programmiersprachen Bekanntschaft. Die Personen erhielten verschiedene Aufgaben, die sie in jeder Sprache programmieren mussten. Am Schluss wurden die Personen nach ihren Eindrücken zu den verschiedenen Programmiersprachen gefragt.

In NBC sehen die Personen ein grosses Potenzial von Flüchtigkeitsfehlern. Dies ist nicht zuletzt darauf zurückzuführen, dass die Programmiersprache keine einheitliche Syntax hat. So stehen nach dem Befehl *wait* keine Klammern, nach dem Befehl *OnFwd* dagegen schon. Auch die Fehlersuche wird von den Testpersonen als mühsam empfunden. Die Probleme mit den Flüchtigkeitsfehlern und der Fehlersuche treten auch in NXC auf. Der Strichpunkt am Ende jeder Zeile bereitet den Personen insofern Probleme, da er häufig vergessen wird. Dagegen ist in NXC die Syntax einheitlich. Daher werden Parameter immer in Klammern geschrieben.

In Robolab sei es nicht möglich Fehler beim Verwenden der Blöcke zu machen, so die Testpersonen. Es traten lediglich Fehler beim Einstellen der Parameter auf. Auch empfanden sie Robolab als ansprechender, da der Funktionsumfang auf den ersten Blick ersichtlich ist.

8 Zusammenfassung

Da jetzt alle Tests durchgeführt und ausgewertet sind, möchte ich die Fragen, die ich mir zu Beginn der Arbeit gestellt hatte, beantworten.

Gibt es die perfekte Programmiersprache? Diese Frage muss mit einem klaren Nein beantwortet werden. Jede Programmiersprache hat ihre Vor- und Nachteile. Auch verfolgen die Programmiersprachen unterschiedliche Ziele. Die eine Sprache soll zum Beispiel möglichst viele Funktionen des NXTs ausreizen, die andere dagegen soll möglichst einfach erlernbar sein.

Der Testsieger unter den von mir durchgeführten Tests ist NXC. Die Sprache hat 278 von 306 möglichen Punkten erreicht. NXC bietet einen beachtlichen Funktionsumfang und reizt somit nahezu sämtliche Funktionen des NXTs aus. Mittels speziellen Befehlen können zusätzlich erweiterte Konfigurationen vorgenommen werden.

Auf dem zweiten Platz landete NBC mit 253 Punkten. Auch NBC bietet einen grossen Funktionsumfang. Allerdings stehen für wichtige Funktionen, wie zum Beispiel das Wiederholen von Programmteilen, in NBC keine Befehle zur Verfügung. Für den Programmierer hat dies einen erheblichen Mehraufwand zur Folge.

Ein weiterer Nachteil von NBC ist die uneinheitliche Syntax. Sie sorgt vor allem bei Neueinsteigern für Verwirrung.

Auf dem letzten Platz befindet sich Robolab mit 184 Punkten. Robolab hat hauptsächlich in der Kommunikation zwischen zwei Robotern ein grosses Defizit. Auch ist der Funktionsumfang bei weitem nicht so gross wie jener von NXC. Durch die vorkonfigurierten Blöcke werden zusätzlich die Konfigurationsmöglichkeiten eingeschränkt.

Robolab bietet, wie ich finde, einen einfachen Einstieg in die Welt der Programmierung. Da die Blöcke meist selbsterklärend sind, ist es nicht erforderlich, sich lange in die Programmiersprache einzuarbeiten. Dies wiederum steigert die Freude am Programmieren.

Hat man die nötige Erfahrung gesammelt, kann man sich mit Programmiersprachen wie NBC oder NXC beschäftigen. Diese bieten zwar einen grösseren Funktionsumfang, dafür ist eine längere Einarbeitungszeit erforderlich.

9 Glossar

A

Assembler: Wandelt in Assemblersprachen programmierte Programme in Maschinencode um
Assembler-ähnlich: Programmiersprache, die der maschineneigenen Programmiersprache sehr ähnlich ist

B

Betaphase: Entwicklungsstadium einer Software; die Software ist noch nicht fertig und wird oft nur zu Testzwecken veröffentlicht.

Betriebssystem: Programm zur Verwaltung der Ressourcen eines Computers

Bluetooth: Eine Technologie, die es erlaubt, kompatible Geräte kabellos miteinander zu verbinden¹

Boolmodus: Name eines Modus, der nur mit den Werten 0 und 1 arbeitet

Briefkasten: Speicherort für Nachrichten

C

C: Programmiersprache der dritten Generation

Celsiusmodus: Modus, bei dem die Werte in Celsius ausgegeben werden

E

Edgemodus: engl. für Kantenmodus; Modus, in dem Übergänge gezählt werden

EDV: Elektronische Datenverarbeitung; Sammelbegriff für Datenverarbeitung und Datenverarbeitungssysteme, die elektronisch arbeiten²

F

Fahrenheitmodus: Modus, der die Werte in Fahrenheit aufzeichnet

Firmware: Betriebssystem eines Geräts

G

Grafische Programmiersprache: Programmiersprache mit der Programme durch das Verknüpfen verschiedener Symbole erstellt werden können

H

Hexadezimal: Zahlensystem mit der Grundzahl 16, welches hauptsächlich in der EDV Verwendung findet

I

Infrarot Tower: engl. für Infrarot Turm; Sendestation, die Signale mittels Infrarot (Lichtwellen) zu einem Empfänger überträgt

1 Sinngemäss aus: siehe LV [12].

2 Sinngemäss aus: siehe LV [13].

L

Labels: engl. für Etiketten, Überschriften

LabVIEW: Grafische Programmiersprache der Firma National Instruments

M

Mac: Mac ist die Kurzform von Macintosh, einem Personalcomputer der Firma Apple; Mac gehört zu den grössten Konkurrenten von Microsoft, dem weltweit grössten Softwarehersteller

Maschinencode: Programmiersprachen der 1. Generation; können von der Maschine direkt verarbeitet werden

Master: engl. für Führer; Befehlsgeber

Mikrocomputer: Computersystem, bei dem nahezu alle Komponenten auf einem integrierten Baustein (Chip) untergebracht sind¹

Mikroprozessor: miniaturisierter Prozessor, der auf einem oder einigen wenigen integrierten Bausteinen verbaut ist; nur arbeitsfähig mit Mikrocomputern²

Mutex: mutual exclusion; engl. für gegenseitiger Ausschluss; Programmieretechnik die den gleichzeitigen Zugriff mehrere Programme auf eine Ressource verhindert³

N

NXT-G: Zum NXT mitgelieferte Programmiersprache, welche von LEGO basierend auf LabVIEW entwickelt wurde

P

Patch: Aktualisierung, die Verbesserungen an einem Programm vornimmt

PDA: Personal Digital Assistant; Handflächen grosser Taschencomputer, welcher hauptsächlich der persönlichen Organisation dient⁴

Percentmodus: engl. für Prozentmodus; Modus, in dem die Werte in einer Skala zwischen 0 und 100 ausgegeben werden

Port: engl. für Anschluss

Pulsemodus: engl. für Pulsmodus; in diesem Modus werden Übergänge gezählt

R

Rawmodus: engl. für Rohmodus; Name für einen Modus, bei dem Rohwerte aufgezeichnet werden

RCX: gelber, programmierbarer Baustein der Firma LEGO

Rotationsmodus: Für den Rotationssensor verwendeter Modus

Rotationssensor: Sensor, der die Umdrehungen des Motors in Grad misst

1 Sinngemäss aus: siehe LV [14].

2 Sinngemäss aus: siehe LV [15].

3 Sinngemäss aus: siehe LV [16].

4 Sinngemäss aus: siehe LV [17].

S

Serielle Schnittstelle: Schnittstelle zwischen zwei Geräten; Datenaustausch erfolgt in Serie d.h. es wird Datenpaket um Datenpaket gesendet

Servomotor: Motor dessen Position und Geschwindigkeit durch ein System kontrolliert wird¹

Slave: engl. für Sklave; führt Befehle des Master aus

Software: engl. für Programm

T

Tasks: engl. für Aufgabe; Ein Programm besteht aus mindestens einem Task

Threads: andere Bezeichnung für Tasks

U

USB: Universal Serial Bus; standardisiertes System zum Anschliessen von Geräten an den Computer²

Z

ZIP-Datei: Ein Archiv, welches mehrere Dateien beinhaltet

1 Sinngemäss aus: siehe LV [18].

2 Sinngemäss aus: siehe LV [19].

10 Literaturverzeichnis

- [5] akademie.de asp GmbH: Assemblersprachen. Online im Internet: URL: <http://www.akademie.de/direkt?pid=2516&tid=12297> [Stand 25.08.2007].
- [16] Babylon Ltd.: Mutex. Babylon Ltd. English-German. 29.05.2000.
- [3] Bachmann, Urs: die neue Generation von LEGO MINDSTORMS. Online im Internet: URL: http://www.educatec.ch/thenews/Press_info/LEGO_MINDSTORMS_PR [Stand 30.09.2007].
- [11] Bendetelli, Daniele: Programming LEGO NXT Robots using NXC. 09.04.2007.
- [15] Bibliographisches Institut & F. A. Brockhaus AG: Mikroprozessor. Online im Internet: URL: <http://lexikon.meyers.de/meyers/Mikroprozessor> [Stand 25.09.2007].
- [7] Carnegie Mellon Robotics Academy: RobotC Reference. Synching Motors.
- [1] Feibel, Thomas: LEGO MINDSTORMS Robotics Invention System Version 1.5. Online im Internet: URL: <http://www.feibel.de/index.php?id=7&catId=24&prodId=2875&cHash=f41745de4f> [Stand 30.09.2007].
- [9] Ginthum, Stefan: Der NXT Geräuschsensor. Online im Internet: URL: <http://www.nxt-inder-schule.de/lego-mindstorms-education-nxt-system/nxt-hardware/sensoren> [Stand 27.08.2007].
- [19] Heinemann, Matthias: Universal Serial Bus. Online im Internet: URL: http://www.it-administrator.de/lexikon/universal_serial_bus.html [Stand 10.09.2007].
- [14] Janssen, Wilhelm: Microcomputer. Online im Internet: URL: <http://www.at-mix.de/micro-computer.htm> [Stand 05.09.2007].
- [8] LEGO MINDSTORMS Education: NXT User Guide.
- [6] Lings, Thomas: Programmiersprachen. Online im Internet: URL: <http://www.oszhdh.be.schule.de/gymnasium/faecher/informatik/algo-progr/prog-sprachen.htm> [Stand 25.08.2007].
- [4] Lipinski, Klaus: Programmiersprache. Online im Internet: URL: http://www.itwissen.info/definition/lexikon//_plpl_plprogramming%20languagepl_plprogrammiersprache.html [Stand 25.08.2007].
- [17] Lüge, Timo: Was ist ein PDA?. Online im Internet: <http://www.usus.org/timo/ddp/service/pda.shtml> [30.09.2007].

[12] Nokia Schweiz AG: Bluetooth. Online im Internet: URL: <http://www.nokia.ch/A4334402> [Stand 30.08.2007].

[13] Odobasic, Sanel: Definition: EDV. Online im Internet: URL: http://www.mta-labor.info/front_content.php?idcat=16 [Stand 27.08.2007].

[10] Ohropax GmbH: Frequenz und Dezibel. Online im Internet: URL: <http://www.ohropax.de/49-0-funktion.html> [Stand 27.08.2007].

[2] Schreiner, Axel: Robotic Invention System. Online im Internet: URL: <http://www.vorlesungen.uos.de/informatik/robot00/html/skript-1.html> [Stand 30.09.2007].

[18] The LEGO Group: Was ist ein Servomotor?. Online im Internet: <http://www.lego.com/deu/service/faqs.asp?section=ConsumerService-FAQ-Products&catid=ECA3C995-0DE6-4FCB-9004-55FD1D266F44&faqid=18973#18973> [30.09.2007].

11 Bildnachweis

Abbildung 1: Weisheit, Tracey Lynn: RCX. Online im Internet: URL: <http://www.acm.org/crossroads/xrds10-4/gfx/toys1.jpg> [Stand 20.09.2007].

Abbildung 2: Moore Educational: NXT. Online im Internet: URL: <http://www.mooreed.com.au/CustomServiceItems/images/9841big.jpg> [Stand 20.09.2007].

Abbildung 3: LPE Technische Medien GmbH: Servomotor (ausen). Online im Internet: URL: <http://www.nxt-in-der-schule.de/bilder/motorneu.jpg> [Stand 25.09.2007].

Abbildung 4: LPE Technische Medien GmbH: Servomotor (innen). Online im Internet: URL: http://www.nxt-in-der-schule.de/bilder/motor_technisch.jpg [Stand 25.09.2007].

Abbildung 5: LPE Technische Medien GmbH: Berührungssensor. Online im Internet: URL: <http://81.169.177.144:9090/produkte/40303.230.215.jpg> [Stand 23.09.2007].

Abbildung 6: LPE Technische Medien GmbH: Lichtsensor. Online im Internet: URL: <http://81.169.177.144:9090/produkte/40303.230.216.jpg> [Stand 23.09.2007].

Abbildung 7: LPE Technische Medien GmbH: Geräuschsensor. Online im Internet: URL: <http://81.169.177.144:9090/produkte/40303.230.217.jpg> [Stand 27.09.2007].

Abbildung 8: LPE Technische Medien GmbH: Ultraschallsensor. Online im Internet: URL: <http://81.169.177.144:9090/produkte/40303.230.218.jpg> [Stand 24.09.2007].

Abbildung 9: Brandl, Michael: Alpha Rex. Online im Internet: URL: http://lego.brandls.info/nxt_alpharex.jpg [Stand 30.07.2007].

Abbildung 10: Brandl, Michael: TriBot. Online im Internet: URL: http://lego.brandls.info/nxt_tribot.jpg [Stand 30.07.2007].

12 Anhang

12.1 Aufgaben

- *Aufgabe 1: Motoren, Aufgaben und Subroutinen, Kontrollfunktionen*

Der Roboter fährt so gerade wie möglich rückwärts, bis die Motoren B und C 5 Umdrehungen erreicht haben.

Nun soll sich der Roboter um 90° drehen. Diese Drehung soll als eigene Funktion vorliegen. Anschliessend fährt der Roboter mit 1/3 Leistung 3 Sekunden vorwärts. Dies soll mittels Subroutine erfolgen. Insgesamt soll dieser Teilcode 3 Mal ausgeführt werden. Die ganze Fahrt über zeichnet der Lichtsensor seine Messwerte auf.

Zum Schluss bremst der Roboter die Motoren ab.

- *Aufgabe 2: Sensoren, Kommunikation*

Zu Beginn gibt der Roboter den Modus des Lichtsensor auf dem Display aus. Der Roboter wartet, bis ein Objekt näher als 50 cm vor ihm ist. Der Ultraschallsensor wird dabei mit der erweiterten Konfiguration eingebunden. Ist ein Objekt näher als 50 cm vor dem Roboter so beginnt er gerade aus zu fahren. Der Roboter fährt nun so lange vorwärts bis er ein Geräusch das lauter als 200 Einheiten (Rawmodus) ist wahrnimmt. Nach der Wahrnehmung stoppt der Roboter, fährt eine halbe Sekunde bei voller Leistung Rückwärts und dreht sich um 180°. Nun kommt der zweite Roboter ins Spiel.

Roboter 1 stellt eine Bluetoothverbindung zu Roboter 2 her. Wird der Berührungssensor gedrückt sendet Roboter 1 an Roboter 2 eine Nummer und wartet auf die Bestätigung von Roboter 2. Roboter 1 erteilt nun Roboter 2 den Befehl 2 Sekunden vorwärts und 2 Sekunden rückwärts zu fahren und anschliessend die Motoren zu bremsen. Roboter 2 soll nun Roboter 1 mitteilen, dass er die Aufgabe erledigt hat. Roboter 1 gibt diese Meldung auf dem Display aus.

12.2 Programme

- *Aufgabe 1, NBC*

```
#include "NXTDefs.h"

#define rotate(angle) \
  RotateMotor(OUT_B, 100, angle)\
  RotateMotor(OUT_C, 100, -angle)

subroutine forward
  OnFwd(OUT_BC, 33)
  wait 3000
  return
ends

dseg segment
  tacho slong
  count byte
dseg ends

thread main
  forward:
    OnRevSyncEm(OUT_BC, 75, 0, RESET_NONE)
    getout tacho, OUT_B, TachoCount
    div tacho, tacho, 360
    brcmp LT, forward, tacho, 5
    Off(OUT_BC)
  repeat:
    rotate(180)
    call forward
    add count, count, 1
    brcmp LT, repeat, count, 3
    Off(OUT_ABC)
  exit
endthread
```

- *Aufgabe 1, NXC*

```
#include "NXCDefs.h"

#define rotation(angle) \
  RotateMotor(OUT_B, 100, angle); \
  RotateMotor(OUT_C, 100, -angle);

int Tacho;

sub drive_fwd()
{
  OnFwd(OUT_BC, 33);
  Wait (3000);
}

task main()
{
  forward:
  OnRevSyncEm(OUT_BC, 75, 0, RESET_NONE);
  Tacho = MotorTachoCount(OUT_B);
  Tacho /= 360;
  NumOut(10, 38, Tacho);
  until (Tacho == 5)
  {
    goto forward;
  }
  Off(OUT_ABC);
  repeat(3)
  {
    rotation(180);
    drive_fwd();
  }
  Off(OUT_ABC);
}
```


- *Aufgabe 2 (Master), NBC*

```
#include "NXTDefs.h"

#define LINE 1
#define INBOX 2
#define OUTBOX 3

values segment
  Light byte
  IN_Touch byte
  IN_Sound byte
  IN_Sonic byte
  BT_Stat dword
  vRandom byte
  Result byte
  Answer byte
  lRandom byte
ends

#define Receive(Value, Label)\
  Label: \
  brcmp NEQ, Label, Answer, Value \
  ReceiveRemoteNumber(INBOX, FALSE, Answer, Result)

thread main
  SetSensorTouch(IN_1)
  ResetSensor(IN_1)
  SetSensorType(IN_2, IN_TYPE_SOUND_DB)
  SetSensorMode(IN_2, IN_MODE_RAW)
  ResetSensor(IN_2)
  SetSensorLight(IN_3)
  ResetSensor(IN_3)
  SetSensorType(IN_4, IN_TYPE_LOWSPEED_9V)
  SetSensorMode(IN_4, IN_MODE_RAW)
  ResetSensor(IN_4)
  getin Light, IN_3, InputMode
  NumOut(10, 38, Light)
  wait 1500
  CheckUltrasonic:
    ReadSensorUS(IN_4, IN_Sonic)
    brcmp GT, CheckUltrasonic, IN_Sonic, 51
  CheckSound:
    OnFwd(OUT_BC, 50)
    ReadSensor(IN_2, IN_Sound)
    brcmp LT, CheckSound, IN_Sound, 255
    Off(OUT_ABC)
    OnRev(OUT_BC, 100)
    wait 500
    RotateMotor(OUT_B, 100, 360)
    RotateMotor(OUT_C, 100, -360)
  CheckTouch:
    ReadSensor(IN_1, IN_Touch)
    wait 1
    brtst EQ, CheckTouch, IN_Touch
    BluetoothStatus(LINE, BT_Stat)
    brcmp EQ, BTok, BT_Stat, NO_ERR
    TextOut(5, LCD_LINE3, "Fehler!")
    wait 1000
    stop TRUE
  BTok:
    Random(vRandom, 1000)
    SendRemoteNumber(LINE, OUTBOX, vRandom, Result)
    Receive(57, a)
    SendRemoteNumber(LINE, OUTBOX, 1, Result)
    TextOut(5, LCD_LINE3, "1. Auftrag sent")
    Random(vRandom, 1000)
    Receive(7, b)
    set Answer, 0
    SendRemoteNumber(LINE, OUTBOX, 2, Result)
    TextOut(5, LCD_LINE3, "2. Auftrag sent")
    Receive(7, c)
    set Answer, 0
    SendRemoteNumber(LINE, OUTBOX, 3, Result)
    TextOut(5, LCD_LINE3, "3. Auftrag sent")
```

- *Aufgabe 2 (Slave), NBC*

```
#include "NXTDefs.h"

#define LINE 1
#define INBOX 3
#define OUTBOX 2

values segment
  BT_Stat dword
  Answer byte
  Result byte
  Count byte 0
ends

#define Receive(Value, Label)\
Label: \
  brcmp NEQ, Label, Answer, Value \
  ReceiveRemoteNumber(INBOX, TRUE, Answer, Result)

subroutine Check
A1:
  OnFwdSync(OUT_AC, 100, 0)
  wait 2000
  jmp End_Check
A2:
  OnRevSync(OUT_AC, 100, 0)
  wait 2000
  jmp End_Check
A3:
  Off(OUT_ABC)
  jmp End_Check
brcmp EQ, A1, Answer, 1
brcmp EQ, A2, Answer, 2
brcmp EQ, A3, Answer, 3
End_Check:
return
ends

thread main
  BluetoothStatus(LINE, BT_Stat)
  brcmp EQ, BTok, BT_Stat, NO_ERR
  TextOut(5, LCD_LINE3, "Fehler!")
  wait 1000
  stop TRUE
  BTok:
  Receive(0,a)
  TextOut(5, LCD_LINE3, "Number received")
  wait 1000
  ClearScreen()
  SendResponseNumber(OUTBOX, 57, Result)
  TextOut(5, LCD_LINE3, "Answer sent")
  set Answer, 0
  Repeat:
  Receive(0,b)
  call Check
  set Answer, 0
  SendResponseNumber(OUTBOX, 7, Result)
  add Count, Count, 1
  brcmp LT, Repeat, Count, 4
  SendResponseNumber(OUTBOX, 5, Result)
  wait 1000
endt
```

Programmiersprachenvergleich für einen NXT Lego Roboter Programme

Aufgabe 2 (Master), NXC

```
#include "NXCDefs.h"

#define LINE 1
#define INBOX 2
#define OUTBOX 3

int Light, IN_Touch, IN_Sound, IN_Sonic, vRandom, Answer = 0, Answer_end;

sub BTCheck(int conn)
{
  if (!BluetoothStatus(conn)==NO_ERR)
  {
    TextOut(5, LCD_LINE2, "Fehler!");
    Wait(1000);
    Stop(TRUE);
  }
}

sub Receive(int Value)
{
  until(Answer == Value)
  {
    ReceiveRemoteNumber(INBOX, TRUE, Answer);
  }
}

task main()
{
  SetSensorTouch(S1);
  ResetSensor(S1);
  SetSensorType(S2, IN_TYPE_SOUND_DB);
  SetSensorMode(S2, IN_MODE_RAW);
  ResetSensor(S2);
  SetSensorLight(S3);
  ResetSensor(S3);
  SetSensorType(S4, IN_TYPE_LOWSPEED_9V);
  SetSensorMode(S4, IN_MODE_RAW);
  ResetSensor(S4);
  Light = SensorMode(S3);
  NumOut(10, 38, Light);
  Wait(1500);
  until(SensorUS(S4) < 51)
  {
    Wait(1);
  }
  until(SENSOR_2 > 255)
  {
    OnFwd(OUT_BC, 50);
  }
  Off(OUT_ABC);
  OnRev(OUT_BC, 100);
  Wait(500);
  RotateMotor(OUT_B, 50, 180);
  RotateMotor(OUT_C, 50, -180);
  until(SENSOR_1 == 1)
  {
    Wait(1);
  }
  BTCheck(LINE);
  vRandom = Random(1000);
  SendRemoteNumber(LINE, OUTBOX, vRandom);
  Receive(57);
  SendRemoteNumber(LINE, OUTBOX, 1);
  TextOut(5, LCD_LINE2, "1. Auftrag gesendet");
  Receive(7);
  Answer = 0;
  SendRemoteNumber(LINE, OUTBOX, 2);
  TextOut(5, LCD_LINE2, "2. Auftrag gesendet");
  Receive(7);
  Answer = 0;
  SendRemoteNumber(LINE, OUTBOX, 3);
  TextOut(5, LCD_LINE2, "3. Auftrag gesendet");
  Receive(5);
}
```

- *Aufgabe 2 (Slave), NXC*

```
#include "NXCDefs.h"

#define LINE 1
#define INBOX 3
#define OUTBOX 2

int Answer;

sub BTCheck(int conn)
{
  if (!BluetoothStatus(conn)==NO_ERR)
  {
    TextOut(5, LCD_LINE2, "Fehler!");
    Wait(1000);
    Stop(TRUE);
  }
}

sub Receive(int Value)
{
  until(!Answer == Value)
  {
    ReceiveRemoteNumber(INBOX, true, Answer);
  }
}

sub Check()
{
  switch (Answer)
  {
    case 1: OnFwdSync(OUT_AC, 100, 0);
           Wait(2000);
           break;
    case 2: OnRevSync(OUT_AC, 100, 0);
           Wait(2000);
           break;
    case 3: Off(OUT_ABC);
           break;
  }
}

task main()
{
  BTCheck(0);
  Receive(0);
  TextOut(5, LCD_LINE3, "Number received");
  Wait(1000);
  ClearScreen();
  SendResponseNumber(OUTBOX, 57);
  TextOut(5, LCD_LINE3, "Answer sent");
  Wait(1000);
  Answer = 0;
  repeat(3)
  {
    Receive(0);
    Check();
    Answer = 0;
    SendResponseNumber(OUTBOX, 7);
  }
  SendResponseNumber(OUTBOX, 5);
  Wait(1000);
}
```


12.3 Bewertungsbögen

Bewertung NBC

Kriterium	Bewertung	Kommentar
Motoren:		
Anwendung	9	
Geschwindigkeitsstufen	9	zwischen 0-100, stufenlos
Vorwärts/Rückwärts	9	4 Moden
Rotation	9	4 Moden
Tachos	7	nur in Grad, nicht in Umdrehungen; 3
Bremsen	9	
Auslaufen (Coast)	9	
Synchronisieren	9	IDLE, SPEED, SYNC
Statusinformationen	8	können nicht direkt einer Variable zugeordnet werden
Sensoren:		
Anwendung	7	Sensoren müssen initialisiert werden; ReadSensor
Unterstützung der Sensortypen	9	alle Standardensoren
Unterstützung der Sensorfunktionen	9	
Konfigurationsmöglichkeiten	9	normal, erweitert
Aufzeichnungsmoden (Raw, Bool, ...)	9	
Benutzerdefinierte Einstellungen (SetCustom...)	0	nicht verfügbar
Statusinformationen	7	Ausgabe nur in Zahlen; in Manual in Hex, auf NXT in dez
Datalogging	0	Werte müssten manuell aufgezeichnet werden
Aufgaben und Subroutinen:		
Anwendung	8	Subroutinen müssen mit call angesprochen werden
Definition	8	Threads müssen mit exit und endt geschlossen werden
Gliederung	9	Labels
Subroutinen	6	keine Parameter, längere Definition; return
Eigene Funktionen	8	Macros; wenig Auswahl
Kontrollfunktionen:		
Anwendung	5	uncomfortable Anwendung
Programm-Kontrollfunktionen	9	release, acquire, exitted, exit, precedes, follows
Vergleiche	9	cmp, tst
Verzweigungen	7	brmp, brst; nicht übersichtlich
Sprünge	9	jmp
Wiederholungen	0	keine Möglichkeit
Kommunikation:		
Anwendung	9	
Schnittstellen	9	Bluetooth
Master - Slave	9	max 58 bytes lang
Nummernaustausch	9	
Direkter Befehlsaustausch	5	nur für wenige Funktionen
Statusinformationen	6	Status abrufen

Bewertung NXC

Kriterium	Bewertung	Kommentar
<u>Motoren:</u>		
Anwendung	9	
Geschwindigkeitsstufen	9	zwischen 0-1600, stufenlos
Vorwärts/Rückwärts	9	4 Moden
Rotation	9	4 Moden
Tachos	7	nur in Grad, nicht in Umdrehungen; 3 verschiedene
Bremsen	9	
Auslaufen (Coast)	9	
Synchronisieren	9	IDLE, SPEED, SYNC
Statusinformationen	9	
<u>Sensoren:</u>		
Anwendung	8	Sensoren müssen initialisiert werden
Unterstützung der Sensortypen	9	alle Standardensoren
Unterstützung der Sensorfunktionen	9	
Konfigurationsmöglichkeiten	9	normal, erweitert
Aufzeichnungsmoden (Raw, Bool, ...)	9	
Benutzerdefinierte Einstellungen (SetCustom...)	9	
Statusinformationen	7	Ausgabe nur in Zahlen; in Manual in Hex, auf NXT in dez
Datalogging	0	Werte müssten manuell aufgezeichnet werden
<u>Aufgaben und Subroutinen:</u>		
Anwendung	9	Name der Sub, des Macros, oder der Inline Function
Definition	9	Tasks, Subs, Inlines können mit Klammern geschlossen werden
Gliederung	9	Labels
Subroutinen	9	Parameter; kurze Definition
Eigene Funktionen	9	Macros, Inline Functions
<u>Kontrollfunktionen:</u>		
Anwendung	9	einfache Anwendung
Programm-Kontrollfunktionen	9	stop, exitto, acquire, release, precedes, follows
Vergleiche	0	keine Möglichkeit das Ergebnis in einer Variable zu speichern
Verzweigungen	9	if-else-then, switch
Sprünge	9	goto
Wiederholungen	8	while, do-while, until, for, repeat
<u>Kommunikations:</u>		
Anwendung	9	
Schnittstellen	9	Bluetooth
Master - Slave	9	max 58 bytes lang
Nummernaustausch	9	
Direkter Befehlsaustausch	5	nur für wenige Funktionen
Statusinformationen	9	abrufen und festlegen

Bewertung Robolab

Kriterium	Bewertung	Kommentar
Motoren:		
Anwendung	9	
Geschwindigkeitsstufen	5	Stufen; 1-5
Vorwärts/Rückwärts	9	Vordefinierte Blöcke für versch. Motoren, als auch Standardblock; keine erweiterten Einstellungen
Rotation	9	3 Moden, mit oder ohne reset, genau die Gradzahl
Tachos	5	Angabe der Anzahl Grad die der Motor drehen soll
Bremsen	9	
Auslaufen (Coast)	9	
Synchronisieren	9	Speed und Synchronisierung
Statusinformationen	4	Drehung
Sensoren:		
Anwendung	9	
Unterstützung der Sensortypen	9	
Unterstützung der Sensorfunktionen	9	
Konfigurationsmöglichkeiten	7	normal
Aufzeichnungsmoden (Raw, Bool, ...)	0	nicht verfügbar
Benutzerdefinierte Einstellungen (SetCustom...)	0	keine
Statusinformationen	0	nicht verfügbar
Datalogging	9	
Aufgaben und Subroutinen:		
Anwendung	9	
Definition	9	
Gliederung	7	Beschriftungen
Subroutinen	7	keine Parameter
Eigene Funktionen	0	nicht verfügbar
Kontrollfunktionen:		
Anwendung	9	
Programm-Kontrollfunktionen	7	Prioritäten, Start Task, Stop Task, Delete Task
Vergleiche	0	keine Möglichkeit
Verzweigungen	7	if-else-then, nur 2 Parameter
Sprünge	9	
Wiederholungen	9	vordefinierte Schleifen
Kommunikations:		
Anwendung	0	Nur Infrarot unterstützt
Schnittstellen	0	
Master - Slave	0	
Nummernaustausch	0	
Direkter Befehlsaustausch	0	
Statusinformationen	0	