



FAKULTÄT FÜR **INFORMATIK**

Einsatz von Lego Mindstorms im Unterricht der HTL für Informationstechnologie

Diplomarbeit

zur Erlangung des akademischen Grades

MAGISTER der Sozial- u. Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Peter Schmidt

Matrikelnummer 0226040

an der

Fakultät für Informatik der Technischen Universität Wien

Betreuerin: Ass.Prof. Dipl.-Ing. Dr.techn. Monika Di Angelo

Wien, 16.04.2009

(Unterschrift Verfasser)

(Unterschrift Betreuerin)

Peter Schmidt
Blaschkegasse 6
2231 Strasshof

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Wien, den 16. April 2009

Peter Schmidt

Abstract

Using Lego Mindstorms combined with LeJOS¹ for educational purpose allows using and extending programming skills not only in intended subjects but also integrating interesting projects in other interdisciplinary subjects. The hardware of the Mindstorms NXT will as well be explained in this thesis as the software installation procedure. This allows fast success, also for beginners.

After the examination of current concepts for using Mindstorms for educational purpose, the biggest part of this thesis is attended to the elaboration of projects, which can be implemented in different subjects, since Mindstorms so far does not appeal too much in educational purpose, especially aside from computer science. Besides programming with LeJOS, part of the Java Virtual Machine implemented for Mindstorms, also design and construction of the robots need to be kept in mind.

¹ Lego Java Operating System

Kurzfassung

Der Einsatz von Lego Mindstorms in Kombination mit LeJOS² im Unterricht erlaubt es, die Programmierung nicht nur in den dafür vorgesehenen Fächern zum Einsatz zu bringen, sondern äußerst interessante Projekte auch fächerübergreifend in andere Unterrichtsgegenstände zu integrieren. Auf die Hardware des Mindstorms NXT wird in der vorliegenden Arbeit ebenso eingegangen wie auf die Installation der benötigten Software. Dadurch wird es auch Einsteigern ermöglicht, schnell Erfolge zu erzielen.

Nach Betrachtung der aktuellen Konzepte zum Einsatz von Mindstorms im Unterricht widmet sich der größte Teil der vorliegenden Arbeit der Ausarbeitung von Projekten, welche in diversen Unterrichtsgegenständen umgesetzt werden können, da bisher Mindstorms v.a. abseits der Informatik nur wenig zum Einsatz kommt. Neben der Programmierung mittels LeJOS, einem für Mindstorms implementierten Teil der Java Virtual Machine, ist ebenso auf die Konzeption und den Bau der Roboter zu achten.

² Lego Java Operating System

Inhaltsverzeichnis

1	Einleitung	6
1.1	Motivation & Ziel	6
1.2	Aufbau	6
1.3	Voraussetzungen zum Verständnis der Arbeit	7
2	Grundlagen	8
2.1	Hardware	8
2.1.1	Programmierbare NXT-Einheit	9
2.1.2	Servomotoren	10
2.1.3	Ultraschallsensor	10
2.1.4	Helligkeitssensor	11
2.1.5	Lautstärkesensor	11
2.1.6	Druckkontaktsensor	11
2.2	Software	11
2.2.1	Java installieren	13
2.2.2	Lego-Treiber installieren	13
2.2.3	LeJOS installieren	13
2.2.4	Eclipse installieren und konfigurieren	14
2.2.5	Firmware Upload	15
2.2.6	Schreiben und Upload eines Testprogrammes	15
2.2.7	Installation der Original Lego-Firmware	16
3	Bestehende Ansätze	17
3.1	Einsatz in informatiknahem Unterricht	17
3.2	Einsatz in anderen Unterrichtsgegenständen	19
4	Entwurf	20
4.1	Angewandte Mathematik	20
4.1.1	Längenmessungen	20
4.1.2	Domino-Strecke	25
4.2	Angewandte Physik	34
4.2.1	Geschwindigkeit und Beschleunigung	34

4.2.1.1	Vorführung durch Lehrkraft	35
4.2.1.2	Programmierung durch Schüler	35
4.2.2	Zwei Roboter für simulierten Überholvorgang	38
4.2.3	Fahrzeug mit integrierter Schaltung	42
4.2.4	Startrampe für Bälle oder Papierflugzeuge	48
4.2.5	Zentrifuge	51
4.2.6	Wellengeneratoren	52
4.2.7	Fallbeschleunigung	54
4.2.8	Messung des Schalldruckpegels	56
4.3	Angewandte Programmierung	59
4.3.1	Funktionstest für Sensoren und Motoren	59
4.3.2	Line Follower	61
4.3.3	Labyrinth	64
4.3.4	Sortieren & Suchen	71
4.3.4.1	Sortieren	71
4.3.4.2	Suchen	74
4.3.5	Music Machine	75
4.4	Netzwerktechnik	78
4.4.1	Toilettenmanager	78
4.4.2	Parkplatzmanagement	81
4.5	Fächerübergreifende Projekte	84
4.5.1	Projekte und Projektmanagement	84
4.5.1.1	Pflichtenheft & Meilensteine & Gruppenarbeit	85
4.5.1.2	Simulierte Abnahme durch Kunden	86
4.5.2	Betriebswirtschaft	86
4.5.3	Computer Aided Design	86
5	Zusammenfassung und Ausblick	88
	Programmcodeverzeichnis	90
	Abbildungsverzeichnis	91
	Literaturverzeichnis	92

1 Einleitung

1.1 Motivation & Ziel

Das Lernen in der Informatik gestaltet sich nach eigener Erfahrung sowohl in der Schule, als auch an der Universität, meist aus reinem Studieren von Büchern, Arbeitsunterlagen oder Online-Anleitungen. Wird eine praktische Aufgabe am Computer gelöst, so ist ein Erfolg meist nur durch eine Bildschirmausgabe ersichtlich, was v.a. dem Spieltrieb nicht sehr zuträglich ist. Weiters wird die Informatik sehr isoliert betrachtet. Häufig ist zwar von fächerübergreifendem Unterricht die Rede, im Lehrplan wie auch in der Praxis nimmt dieser jedoch nur einen geringen Stellenwert ein.

Mit Lego Mindstorms kann solch ein fächerübergreifender Unterricht leichter erreicht werden. So sind zwar für jede Art der Anwendung von LeJOS, einem Java-basierten Betriebssystem für den NXT, Java-Kenntnisse erforderlich, allerdings müssen auch, abhängig von der Aufgabenstellung und den beteiligten Fächern, etwa Physik-, Mathematik- oder Wirtschaftskenntnisse vorhanden sein.

Der wohl größte Vorteil von Lego Mindstorms ist, abgesehen vom Einsatz im Unterricht, dass es sowohl als Spielzeug als auch zur Simulation realer Gegebenheiten verwendet werden kann. So können damit etwa genauso wie mit anderen Robotern Wettkämpfe in verschiedenen Disziplinen (z.B. Fußball) ausgetragen werden. Ebenso können aber reale Situationen nachgestellt werden und damit Rückschlüsse auf Kosten und Fehlerquellen gezogen werden. Eine solche Möglichkeit wäre etwa der Nachbau einer Fließbandproduktion.

1.2 Aufbau

Diese Arbeit zeigt Möglichkeiten zur Integration der Programmierung von Lego Mindstorms in möglichst unterschiedliche Unterrichtsfächer der HTL für Informationstechnologie. Das bedeutet, dass sich der Einsatz hier v.a. nicht auf den Informatikunterricht beschränken soll, sondern speziell auch als Hilfsmittel in anderen Fächern genutzt werden soll. Hierbei muss erwähnt werden, dass die in der vorliegenden Arbeit erwähnten Beispiele keinesfalls die einzig möglichen sind und

dass diese natürlich jederzeit erweitert oder gekürzt werden können. Als Grundlage für die Auswahl der Beispiele diene der Lehrplan der höheren Lehranstalt für Informationstechnologie [Bund06].

1.3 Voraussetzungen zum Verständnis der Arbeit

Zum Verständnis dieser Arbeit sollte Grundwissen in den Bereichen Mathematik sowie Java vorhanden sein. Vor allem Kenntnisse der Geometrie sind wichtig, um die jeweiligen Schritte nachvollziehen zu können. Die zusätzlich benötigten Kenntnisse sind abhängig vom jeweils behandelten Stoffgebiet, allerdings keinesfalls allzu tiefgreifend.

Für die Installation von LeJOS werden Grundkenntnisse im Umgang mit Windows XP ebenso vorausgesetzt wie die Tatsache, dass ein entsprechendes System bereits vorliegt, etwa als Gastsystem einer virtuellen Maschine.

2 Grundlagen

In diesem Kapitel werden die verwendete Hard- und Software sowie deren Installation beschrieben. Das hier Beschriebene wird dann im weiteren Verlauf der Arbeit als Wissen vorausgesetzt.

2.1 Hardware

Grundvoraussetzung für die Umsetzung der Projekte ist ein Lego Mindstorms Bausatz. Hierbei gibt es zwei unterschiedliche Generationen: das 1998 vorgestellte Mindstorms RCX und das 2006 vorgestellte Mindstorms NXT. Für diese Arbeit wurde das Mindstorms-NXT-System (Produktnummer 8527) verwendet. Wichtigste Bestandteile dieses Systems neben Elementen wie Rädern oder Kabeln sind:

Art	Anzahl
Servomotor	3
Ultraschallsensor	1
Lichtsensoren	1
Geräuschsensor	1
Druckkontaktsensor	1
NXT-Baustein	1

Mindstorms stellt generell ein anschauliches Beispiel für ein sog. „embedded system“, also ein eingebettetes System dar, bei welchem ein zentraler Controller etwa mit Sensoren kommuniziert. Solche Eingebetteten Systeme sind aus dem täglichen Leben nicht mehr wegzudenken und finden sich etwa in DVD-Spielern, modernen Haushaltsgeräten oder PKW (u.a. zur Kontrolle des ABS) wieder.

Insgesamt enthält das System 577 Bauteile, zu welchen etwa Kabel zur Verbindung der Sensoren mit dem NXT-Baustein oder Reifen zählen. Batterien oder Akkus sind in dem System nicht enthalten. In der Schulversion des Mindstorms-NXT-Bausatzes (Produktnummer 9797) ist jedoch ein Lithium-Ionen-Akku enthalten. Auffällig an den enthaltenen Bausteinen ist, dass diese völlig auf die Legotypischen Noppen verzichten. Stattdessen werden Verbindungen nun durch Lochstangen, bekannt aus Lego Technik, hergestellt.

2.1.1 Programmierbare NXT-Einheit

Der NXT-Baustein ist das Herzstück des Mindstorms-Systems. Er besitzt einen 32-Bit-Mikroprozessor, ein Display, Anschlüsse für je drei Motoren und vier Sensoren, einen USB-Anschluss, Bluetooth sowie einen internen Lautsprecher. Die Kommunikation des NXT mit einem PC kann über USB oder Bluetooth erfolgen. Einzelne NXT können ebenfalls mittels Bluetooth kommunizieren oder es kann etwa ein NXT per Mobiltelefon gesteuert werden, wenn auf diesem ein entsprechendes Programm installiert ist. Wichtig ist dabei, dass solch eine Verbindung stets aus einem sog. „receiver“ und einem sog. „initiator“ besteht. Der „receiver“ wartet auf den Verbindungsaufbau, muss den „initiator“ aber nicht kennen. Der „initiator“ baut die Verbindung zum „receiver“ auf, muss diesen daher auch kennen. Sobald die Verbindung aufgebaut ist, können beide Teilnehmer Daten austauschen.

Programmieren lässt sich der NXT standardmäßig mittels des mitgelieferten, auf LabVIEW basierenden NXT-G. Die Firmware des NXT kann durch beliebige andere zur Verfügung stehende Firmware ersetzt werden, um somit eine andere Programmierumgebung nutzen zu können. Für alle hier folgenden Projekte wird das Java-Betriebssystem LeJOS¹ NXJ verwendet, auf dessen Installation und Konfiguration im Abschnitt Software eingegangen wird. Alle nun folgenden Angaben beziehen sich auf den Einsatz von LeJOS.

Das Display des NXT ist monochrom, d.h. es kann nur die Farbe Schwarz angezeigt werden. Die Auflösung beträgt 100 * 64 Pixel. Für das Erzeugen von Ausgaben am Display ist auch anzugeben, in welcher x-y-Position diese statzufinden hatt. Bei Grafikausgaben bedeutet dies eine Pixelangabe, bei Ausgabe von Zeichen eine Angabe von Stellen, wofür das Display 16 Spalten in der x-Koordinate und 8 Zeilen in der y-Koordinate bietet. Text kann auch ohne eine solche Angabe einer genauen Position mittels „System.out.println()“ ausgegeben werden. Dieser wird dann allerdings am Ende der jeweiligen Zeile automatisch umbrochen, ohne dabei irgendwelche sprachlichen Regeln zu beachten. Ist der Text zu lang für das gesamte Display, so wird er durch dieses gescrollt.

Durch Tastennavigation kann über das Display etwa Bluetooth aktiviert werden, es können alle am NXT vorhandenen Programme aufgelistet und per Tastendruck gestartet oder gelöscht werden, die Lautstärke des Lautsprechers kann geändert werden oder der aktuell freie Speicher und Ladestand des Akkus angezeigt werden.

Die vorhandenen Buttons dienen nicht nur der Navigation im Menü, diese können auch in Programmen verwendet werden, etwa durch Befehle wie „if (But-

¹ Lego Java Operating System

ton.ENTER.isPressed())“ oder „while (!Button.LEFT.isPressed())“. Der Lautsprecher kann entweder im Programm festgelegte Töne oder wav-Dateien abspielen.

2.1.2 Servomotoren

Mittels des integrierten Rotationssensors kann sowohl die Geschwindigkeit als auch die Distanz, welche zurückgelegt werden soll, recht genau bestimmt werden. Die Genauigkeit liegt hier bei einem Grad. Mittels LeJOS werden alle Angaben zur Geschwindigkeit in $\frac{\text{Umdrehungsgrad}}{\text{Sekunde}}$ angegeben. Die höchste Geschwindigkeit liegt bei $900 \frac{\text{Umdrehungsgrad}}{\text{Sekunde}} \hat{=} 2,5 \frac{\text{Umdrehungen}}{\text{Sekunde}}$. Diese kann natürlich mittels einer geeigneten Übersetzung (bzw. Untersetzung) noch gesteigert werden, wodurch allerdings die Ungenauigkeit mit dem Grad der Untersetzung steigt. Die kleinstmögliche Rotation beträgt ein Grad. Das bedeutet etwa für Fahrzeuge, dass die Genauigkeit umso größer ist, je kleiner der Umfang ihrer angetriebenen Räder ist. Einem Motor muss aber nicht eine fixe Gradzahl als Rotationsziel vorgegeben werden, also etwa 3600° für zehn Umdrehungen, sondern er kann auch schlicht gestartet und gestoppt werden, ohne konkretes Rotationsziel.

Sind mehrere Motoren in einem Roboter verbaut, welche in ihrer Bewegung aufeinander abgestimmt werden müssen, weil diese etwa die Hinterachse eines Fahrzeuges antreiben, so ist dies mittels der pilot-Klasse möglich. Dadurch muss etwa nicht selbst das Fahren einer Rechtskurve programmiert werden, es müssen den jeweiligen Methoden nur die passenden Parameter übergeben werden, um etwa eine Rotation oder eine Kurvenfahrt auszuführen.

In [Geba07] wurde festgestellt, dass die Motoren bei höherer Last mehr Strom verbrauchen und dass deren Leistung sinkt, je weniger Strom der Akku zur Verfügung stellt.

2.1.3 Ultraschallsensor

Der Ultraschallsensor sendet Ultraschall aus und misst die Zeit, bis die Reflexion wieder bei ihm anlangt. Daraus wird die Entfernung möglicher Hindernisse berechnet. Die größte Entfernung, in welcher dieser ein Hindernis erkennen kann, hängt auch von dessen Material ab, liegt aber maximal bei etwa 170 Centimeter [leJO08], die Genauigkeit liegt etwa bei drei Centimeter [Geba07]. Die Ultrasonic-Klasse stellt etwa die Methode „getDistance()“ zur Verfügung, mittels welcher der gemessene Abstand als Integer-Wert in Centimeter retourniert wird. Wird kein Hindernis erkannt, so erfolgt eine Rückgabe von „255“. Die Einheit der Rückgabewerte sind Centimeter im Intervall von 0 bis 255.

2.1.4 Helligkeitssensor

Der Helligkeitssensor kann sowohl die Helligkeit des Umgebungslichtes als auch, mithilfe des zusätzlichen Infrarotlichtes, die Helligkeit präziser zu bestimmender Orte eruieren, etwa die Helligkeit einer Linie. Die Lightsensor-Klasse bietet diverse Methoden, u.a. kann das Infrarotlicht mittels „setFloodlight()“ beliebig ein- und ausgeschalten werden oder mittels „readValue()“ der aktuelle Helligkeitswert ausgelesen werden. Mittels dieses Sensors können etwa sog. „Line Follower“ verwirklicht werden, welche z.B. einer schwarzen Linie auf weißem Grund folgen. Weiters ist eine Unterscheidung von Farben anhand ihrer Helligkeitswerte möglich.

2.1.5 Lautstärkesensor

Der Lautstärkesensor misst den Schalldruckpegel. Dabei kann zwischen den beiden Maßstäben dB und dB(A) gewählt werden. dB(A) entspricht dem menschlichen Hörempfinden, bei welchem die empfundene Lautstärke auch von der jeweiligen Frequenz abhängt. Die SoundSensor-Klasse stellt die beiden Methoden „setDBA()“ und „readValue()“ zur Verfügung. Mittels ersterer wird der Messmodus zwischen dB und dB(A) umgeschalten, mittels zweiterer der gemessene Wert als Integer-Zahl ausgelesen.

2.1.6 Druckkontaktsensor

Dieser Sensor ist der technisch einfachste aller mitgelieferten Sensoren. Es sind nur zwei Zustände möglich: „gedrückt“ oder „nicht gedrückt“. Er kann beispielsweise zum Start oder Abbruch von Programmen oder zur Kollisionserkennung verwendet werden. Zur Auslösung ist eine Kraft von 0,34 Newton nötig [Geba07]. Die Klasse TouchSensor stellt folglich auch nur die Methode „isPressed()“ zur Verfügung, mittels welcher anhand eines boolean-Wertes überprüft wird, ob der Sensor gedrückt ist.

2.2 Software

Alle in dieser Arbeit vorgestellten Projekte basieren nicht auf der vorinstallierten Software, sondern auf dem Java-basierten LeJOS. Um dieses verwenden zu können, muss erst die entsprechende Firmware auf den NXT überspielt werden und jener PC, mit welchem der NXT kommunizieren soll, entsprechend konfiguriert werden. Diese Schritte werden in den folgenden Unterkapiteln beschrieben.

Für den Einsatz im Unterricht erscheint es sinnvoll, solch eine Konfiguration nicht auf jedem einzelnen Schul-PC vorzunehmen, sondern auf virtuelle Maschinen zurückzugreifen. Diese bieten den Vorteil, dass etwa eine einzelne Konfiguration beliebig vielfältigt werden kann und somit die Funktionstüchtigkeit bei entsprechenden Voraussetzungen gewährleistet werden kann. Weiters können solche virtuellen Maschinen auf den meisten gebräuchlichen Gastsystemen betrieben werden, was etwa dann von Vorteil ist, wenn die Schüler ihre Roboter auch zu Hause programmieren und ausprobieren sollen. Sie brauchen sich dann nicht um die Konfiguration ihres Betriebssystems für die Kommunikation mit dem NXT zu kümmern.

Eine weitere Möglichkeit stellt die Erstellung von Live-Systemen dar. Solche Systeme können direkt von einem Medium wie CD, DVD oder USB-Stick gebootet werden, ohne Veränderungen am installierten Betriebssystem vorzunehmen. Der größte Nachteil besteht hier allerdings darin, dass das jeweilige Medium im BIOS² als Boot-Medium festgelegt sein muss. Ist dies nicht der Fall und hat der Schüler keinen Zugriff auf das BIOS, so ist kein Einsatz des Live-Systems möglich.

Für alle Projekte dieser Arbeit wurde zum Einsatz von LeJOS ein mittels VirtualBox³ erstelltes Gastsystem basierend auf Windows XP Professional verwendet. LeJOS ist neben der 32-Bit-Version für Windows auch für Linux und Mac OS X erhältlich. Da Windows XP wohl die einfachste Konfiguration bietet, fiel die Entscheidung hier auf eben dieses Betriebssystem. Weiters hat Windows generell einen Marktanteil von knapp 90%, während etwa jener von Linux noch unter einem Prozent liegt [Mark08]. Das bedeutet, dass wohl deutlich mehr Schüler mit einem Windows-Betriebssystem umgehen können als mit Linux.

Beim Einsatz von Windows XP ist eine Größe von vier Gigabyte für die virtuelle Festplatte ausreichend. Dies bietet genügend Platz für das Betriebssystem sowie alle nötigen Werkzeuge. Großer Vorteil ist hier, dass sich eine Datenmenge von vier Gigabyte etwa mittels DVD oder geeigneten USB-Sticks und MP3-Playern transportieren lässt.

Größter Nachteil v.a. gegenüber Linux wäre klarerweise, dass hier Lizenzgebühren für jede einzelne virtuelle Maschine anfallen würden. Linux wäre wohl genau dann aufgrund der Kosten der Vorzug zu geben, wenn die Vorteile von Windows bezüglich der einfacheren Konfiguration und der weiteren Verbreitung außer Acht gelassen werden. Als Entwicklungsumgebung kommt Eclipse zum Einsatz, dessen Konfiguration im folgenden auch beschrieben wird. Eclipse selbst wurde mittels Java programmiert, ist Open Source und bietet u.a. den Vorteil einer

² Basic Input Output System

³ <http://www.virtualbox.org>

Kompilierung schon während der Code-Eingabe.

Die folgenden Unterkapitel beschreiben nun, wie nach der Installation von Windows XP in einer virtuellen Maschine vorgegangen werden muss.

2.2.1 Java installieren

Es wird mindestens Version 1.5 des Java Runtime Environment (JRE) benötigt, welches unter <http://www.java.com/de/download/manual.jsp> heruntergeladen werden kann. Das deutlich größere Java Development Kit wird nicht benötigt, kann aber ebenso installiert werden. Java wird unter `C:\Programme\Java` installiert, ein Setzen von Path oder Classpath ist nicht nötig.

2.2.2 Lego-Treiber installieren

Bevor der NXT mittels USB mit dem PC verbunden wird, muss auf dem PC der entsprechende Treiber installiert werden. Im Laufe dieser Arbeit findet auch der Upload der erstellten Programme mittels USB statt. Wurde die Software der dem Mindstorms-Set beigelegten CD bereits in der virtuellen Maschine installiert, so ist auch dieser Treiber bereits installiert. Diese komplette Software ist zur Verwendung von Mindstorms in Verbindung mit LeJOS allerdings nicht notwendig, d.h. es wird ausschließlich der entsprechende Treiber benötigt. Die neueste Version davon findet sich unter <http://mindstorms.lego.com/Support/Updates/>. Dort ist der Punkt „MINDSTORMS NXT Driver vx.xx“ zu suchen und die PC-Version herunterzuladen. Nach dem Entpacken kann das Setup gestartet werden. Ist dieses vollendet, sollte die virtuelle Maschine neu gestartet werden. Nach dem Neustart kann der NXT mit dem PC verbunden werden. Im Gerätemanager von Windows sollte der NXT nun in der Kategorie „Lego Devices“ angezeigt werden. Ist dies nicht der Fall, so kann dies daran liegen, dass etwa bei VirtualBox der USB-Controller nicht aktiviert ist oder im Menüpunkt „Devices - USB Devices“ der laufenden Maschine der Anschluss des Mindstorms-Roboters nicht aktiviert ist.

2.2.3 LeJOS installieren

Hierfür ist von <http://lejos.sourceforge.net/nxj-downloads.php> Lejos herunterzuladen. Hier sollte die Archiv-Datei gewählt werden und diese in einem Pfad entpackt werden, welcher keine Leerzeichen enthält, um Probleme mit Java zu vermeiden, also z.B. nach `C:\Programme\`. Dort erscheint dann

ein Ordner namens `lejos_nxj`.

Anschließend werden die entsprechenden Umgebungsvariablen gesetzt. Dazu erfolgt ein Rechtsklick auf das Desktop- bzw. Explorericon „Arbeitsplatz“, um im Drop-Down-Menü „Eigenschaften“ auszuwählen. In der Registerkarte „Erweitert“ erfolgt anschließend ein Klick auf den Button „Umgebungsvariablen“, um nun im neuen Fenster im Bereich „Systemvariablen“ auf den Button „Neu“ zu klicken. Die neue Variable trägt den Namen `NXJ_HOME` und erhält als Wert den Installationspfad von LeJOS, also `C:\Programme\lejos_nxj`.

Anschließend wird die `LeJOS_HOME`-Variable der Path-Variable hinzugefügt. Dazu wird diese doppelt geklickt und der Wert `;%NXJ_HOME%\bin` am Ende hinzugefügt.

2.2.4 Eclipse installieren und konfigurieren

Unter <http://www.eclipse.org/downloads/> kann Eclipse heruntergeladen werden. Hier ist die Classic-Version zu wählen, da sich alle folgenden Angaben auf eben diese Version beziehen. Die zip-Datei sollte dann nach `C:\Programme\Eclipse` entpackt werden. Um Eclipse zu starten, muss die Datei „eclipse.exe“ im Hauptordner ausgeführt werden. Daher ist es sinnvoll, eine Verknüpfung im Startmenü bzw. am Desktop anzulegen.

Beim ersten Start wird nach einem Speicherort für den sog. Workspace-Ordner gefragt. Dieser kann beliebig gewählt werden, ist also unabhängig vom Speicherort von LeJOS oder Java. In diesem Ordner werden die Java-Projekte abgelegt, also auch alle java- sowie die kompilierten class-Dateien. Alle im Folgenden gemachten Angaben zu Menübezeichnungen aus Eclipse beziehen sich auf den Wortlaut aus dem derzeit aktuellen Eclipse 3.4.2 (Stand: 16. April 2009).

Für den Upload der erstellten Programme direkt aus Eclipse steht ein eigenes Plugin zur Verfügung, mit welchem es weiters möglich ist, die benötigte Firmware auf den NXT zu laden. Um dieses Plugin zu installieren erfolgt im Menü von Eclipse ein Klick auf „Help - Software Updates...“. Im folgenden Fenster erfolgt ein Klick auf die Registerkarte „Available Software“ und anschließend den Button „Add Site...“, um hier die Adresse <http://lejos.sourceforge.net/tools/eclipse/plugin/nxj/> anzugeben. Nach Zustimmung zu den Lizenzbedingungen und Download der Daten sollte Eclipse neu gestartet werden.

Nun fällt sofort auf, dass in der sog. Toolbar ein Button für den Firmwareupload hinzugefügt wurde und in der Menüleiste befindet sich nun ebenfalls ein zusätzliches Element mit der Bezeichnung „leJOS NXJ“. Nun muss dem Plugin noch der Installationsort von „LeJOS“ mitgeteilt werden. Klicken Sie dazu auf „Window - Preferences“ und im neuen Fenster auf „leJOS NXJ“. Klicken Sie den

„Browse...“-Button, um den NXJ_Home-Pfad festzulegen und wählen Sie den Installationsordner von LeJOS, also `C:\Programme\lejos_nxj`.

2.2.5 Firmware Upload

Anschließend wird die Original-Firmware durch LeJOS ersetzt. Dazu muss der NXT mit dem PC verbunden und erkannt sein, sowie in den Firmware-Upload-Modus versetzt werden. Dies geschieht durch Drücken eines versteckt an der Rückseite des NXT angebrachten Knopfes (siehe Abbildung 2.1), welcher etwa mittels einer aufgebogenen Büroklammer oder eines kleinen Schraubenziehers erreicht werden kann. Sobald der besagte Modus erreicht ist, ist ein leise pulsierender Ton zu hören.

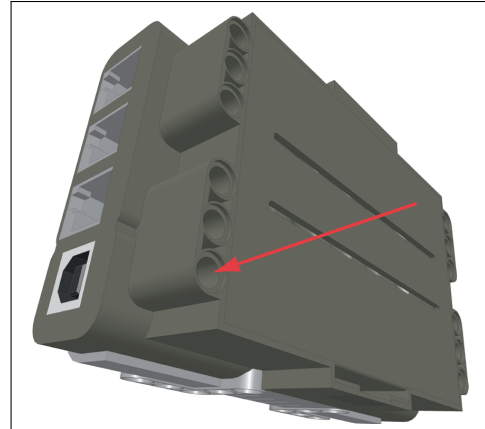


Abbildung 2.1: Button zum Erreichen des Firmware-Upload-Modus

Sobald dies der Fall ist, klicken Sie in Eclipse auf den Button „Upload the leJOS NXJ firmware to the NXT brick“. Die Firmware wird nun installiert, der NXT startet neu und es erscheint zum Abschluss das LeJOS-Logo und anschließend das Menü.

2.2.6 Schreiben und Upload eines Testprogrammes

Zuerst muss in Eclipse ein neues Java-Projekt erstellt werden. In diesem Projekt-Ordner werden anschließend die erstellten Java-Dateien gespeichert. Es können mehrere solcher Projekt-Ordner erstellt werden. Beim Erstellen einer neuen Java-Klasse kann dann ausgewählt werden, welchem Projekt diese zuzuordnen ist.

Um nun ein solches Projekt zu erstellen, klicken Sie im Menü der Reihe nach auf „File - New - Java Project“, geben dem Projekt anschließend einen Namen, etwa „lejos“, und klicken dann auf „Finish“. Im „Package Explorer“ erscheint daraufhin ein neuer Ordner, welcher den angegebenen Namen trägt. Das Projekt ist allerdings noch kein LeJOS-Projekt. Dies ist u.a. dann zu erkennen, wenn im „Package Explorer“ per Rechtsklick auf den Projektordner „Properties“ ausgewählt werden und im neuen Fenster links der Punkt „Java Build Path“ und rechts das Register „Libraries“. Hier steht nun etwa „JRE System Library [jre6]“.

Um das Java-Projekt in ein LeJOS-Projekt umzuwandeln genügt ein Rechtsklick auf den Projektordner (im Package Explorer) und die Auswahl von „leJOS

NXJ - Convert to leJOS NXJ project“. Kontrolliert man nun nochmals die Libraries, so wurde dieser Wert nun auf den Pfad von LeJOS geändert.

Mittels Klick auf „File - New - Class“ kann nun ein neues Programm erstellt werden. Im folgenden Fenster muss dazu noch ein passender Name gewählt und „public static void(String[] args)“ aktiviert werden. Wird der Code aus Programmcode 2.1 verwendet, so muss die Java-Klasse bei der Erstellung mit „HelloWorld“, benannt werden, da Name der Klasse und Dateiname stets ident sein müssen. Der Code für das Beispielprogramm dient nur der Überprüfung, ob der Firmwareupload erfolgreich war und ob Eclipse korrekt konfiguriert ist.

Programmcode 2.1: Beispielcode - Anzeige von „Hello World“

```
import lejos.nxt.*; /*Import des lejos.nxt-Pakets*/
public class HelloWorld { /*Beginn der Klasse und Festlegung des
    Namens(muss dem Dateinamen entsprechen!)*
    public static void main (String[] args) { /*Beginn der main-Methode
        */
        LCD.drawString("Hello World", 0, 0); /*"Hello World" in 1. Zeile
            und 1. Spalte ausgeben*/
        Button.waitForPress(); /*Auf Drücken eines Buttons warten*/
    } /*Ende der main-Methode*/
} /*Ende der Klasse*/
```

Um den Upload auf den NXT zu starten, muss zuerst die Klasse gespeichert werden und der NXT mit dem PC verbunden und eingeschalten werden. Anschließend erfolgt im „Package Explorer“ von Eclipse ein Rechtsklick auf die entsprechende Java-Datei und es wird „leJOS NXJ - Upload Program To The NXT Brick“ ausgewählt. Daraufhin erfolgt sofort der Upload des Programmes und es kann am NXT nun durch Auswahl von „Files - HelloWorld.nxj - Execute program“ ausgeführt werden. Vor jedem Upload sollte sichergestellt werden, dass in dem jeweiligen Programm keine Fehler enthalten sind. Eclipse markiert fehlerhafte bzw. unnötige Stellen schon während der Code-Eingabe farblich.

2.2.7 Installation der Original Lego-Firmware

Um wieder die Original Lego-Firmware auf den NXT aufzuspielen wird die original „Lego Mindstorms NXT“-Software benötigt. Verbinden Sie dazu den NXT mittels USB mit dem PC und versetzen Sie erstere wie in Abschnitt 2.2.5 beschrieben in den Firmware-Upload-Modus. Starten Sie die „Lego Mindstorms NXT“-Software und wählen Sie im Menü den Punkt „Werkzeuge - NXT-Firmware aktualisieren...“. Im folgenden Fenster können Sie nach neueren verfügbaren Firmware-Versionen suchen und mit einem Klick auf den Button „Herunterladen“ wieder die Original-Firmware auf Ihren NXT laden.

3 Bestehende Ansätze

In diesem Kapitel wird beleuchtet, ob und in welcher Form Lego Mindstorms derzeit im Unterricht eingesetzt wird. Dabei findet eine Unterscheidung nach informatiknahem Unterricht und „anderen“ Unterrichtsgegenständen statt, da die Aufnahme von Mindstorms in informatiknahen Unterricht deutlich einfacher erscheint, zumindest was den Zeitaufwand für Lehrkräfte und eventuell anfallende Kosten betrifft. Einen Informatiklehrer darf die Programmierung eines Mindstorms-Roboters nicht vor allzu große Probleme stellen, während etwa ein Physiklehrer vor komplett anderen Voraussetzungen steht.

3.1 Einsatz in informatiknahem Unterricht

Wie in [Ehma06] beschrieben, eignet sich Lego Mindstorms für den Informatikunterricht ab der Sekundarstufe I. Der Vorteil liegt hier für die Schüler v.a. in dem gewohnten Umfeld und im direkten Sichtbarwerden von Fehlern durch ein Fehlverhalten des Roboters. Das wohl am häufigsten eingesetzte Beispiel zur Einführung in die Programmierung bei gleichzeitigem Einsatz von Sensoren und Motoren stellt wohl der sog. Line Follower dar [Virt08a] [Virt08b]. Bei diesen Beispielen geht es darum, einen Roboter einer am Boden angebrachten Linie folgen zu lassen. Diverse Variationen sind hier etwa durch die Anzahl der verwendeten Motoren oder durch die Anzahl der verwendeten Sensoren möglich. Ein Line Follower wird auch in Abschnitt 4.3.2 als Beispiel angeführt.

Insbesondere beim Einsatz im informatiknahen Unterricht muss beachtet werden, dass sich die mitgelieferte Software, mittels welcher durch rein grafische Befehle programmiert wird, nur zum Erlernen grundlegender Programmierkonzepte eignet. Mittels LeJOS oder auch anderer alternativer Systeme lassen sich fortgeschrittene Konzepte wie Arrays, Methoden oder Klassen erlernen [Egge01].

LegoKara ermöglicht das Ausprobieren von Java-Programmen mittels eines Mindstorms-Roboters. Die in Java mittels JavaKara programmierten Programme können dadurch in ihrer Auswirkung nicht nur am Bildschirm, sondern auch in der realen Welt beobachtet werden. JavaKara, welches LegoKara zugrunde liegt, eignet sich insbesondere zur Darstellung von Schleifen und Verzweigungen [Swis08].

Die „Spelman College Computer Science Olympiade“ richtet sich an Studenten. In Form eines Wettbewerbs werden hier diverse Kategorien der Informatik behandelt. Im Bereich der Robotik kommt dabei Lego Mindstorms zum Einsatz. Bewertet werden sowohl die Konstruktion der Roboter als auch die Programmierung sowie die Präsentationsfähigkeiten der Studenten [Hard08].

An der United States Military Academy in West Point, New York, wird Lego Mindstorms sowohl zum Erlernen grundlegender Programmierkenntnisse eingesetzt, als auch für den weitergehenden Unterricht, etwa zur Programmierung autonomer Fahrzeuge oder eingebetteter Systeme. Hier wurde auch „Jago“ entwickelt, mit welchem das Verhalten eines Roboters mittels eines Computers simuliert werden kann. Beispiel einer zu lösenden Aufgabe ist etwa das Bewältigen eines Labyrinths durch einen Roboter [Flow02].

In der Schweiz veranstalten die Firmen Netcetera und AdNovum mit der sog. „Robot Team Challenge“ einen Wettbewerb im Bereich Robotik, an welchem Mittelschüler zwischen 16 und 20 Jahren teilnehmen können. Für die Teilnahme wird nur ein Mindstorms-Set benötigt. Thema des Bewerbes im Jahr 2008 war die „Roboterunterstützung im Mittelschulalltag“ [Info08]. Auch in Österreich findet mit der RobotChallenge [Öste08] ein ähnlicher Wettbewerb statt, allerdings sind hier alle Personen mit einem Alter von 12 bis 60 Jahren zur Teilnahme berechtigt. Kategorien sind etwa „Slalom“, „Slalom Enhanced“ oder „Puck Collect“. Bei „Slalom“ muss der Roboter einer Linie folgen, d.h. es handelt sich um einen klassischen Line Follower. Bei „Slalom Enhanced“ sind in den Kurs verschiedene Hindernisse eingebaut. Auf einem Spielfeld verteilte, farblich markierte Pucks müssen bei „Puck Collect“ vom jeweiligen Roboter eingesammelt werden.

Den Einsatz im Unterricht erschwert allerdings die Tatsache, dass das Verhalten des jeweiligen Roboters vorhersehbar sein muss, insbesondere wenn Schüler für ihre Leistungen bewertet werden sollen. Ein Mindstorms-Roboter kann sich allerdings in der Schule anders verhalten als zu Hause beim jeweiligen Schüler, was unter anderem an der Abhängigkeit vom Akkuladestand und den analogen Komponenten liegt [Kuma04].

Eine Studie der National Taiwan Normal University verglich die Leistungen der Schüler von vier High-School-Klassen miteinander, wobei in zwei Klassen Mindstorms-Roboter und in den restlichen zwei Klassen ein Mindstorms-Simulator zum Einsatz kam. Ziel war es, herauszufinden, welche Auswirkungen der „physische“ Effekt auf die Schüler hat. Dabei stellte sich heraus, dass sich die Leistung der beiden Gruppen nicht unterschied, die Mindstorms-Anwender konnten sich allerdings Programmläufe besser vorstellen. Außerdem wollten sie im Gegensatz zu den Simulator-Programmierern mehr über das Programmieren erfahren

und hatten mehr Spaß daran [Huan08].

3.2 Einsatz in anderen Unterrichtsgegenständen

Der Einsatz von Lego Mindstorms in Unterrichtsgegenständen, welche nicht direkt der Informatik zuzurechnen sind, ist bisher wesentlich unpopulärer. Gründe hierfür lassen sich allerdings leicht ausfindig machen: so ist es für eine Lehrkraft der Informatik selbstverständlich, programmieren zu können. Mindstorms stellt in diesem Bereich eine Erweiterung dar. In anderen Unterrichtsgegenständen wurde zumindest bisher von Lehrkräften allerdings nicht erwartet, dass diese programmieren können. Außerdem ist ein gewisser Zeitaufwand mit Konzeption, Bau und Programmierung eines Roboters verbunden. Dabei sollte allerdings bedacht werden, dass ein einmal konzipierter Roboter beliebig oft eingesetzt werden kann. Wird die Planung am PC vollzogen, so lässt sich eine Bauanleitung speichern und auch der Programmcode kann gespeichert und wiederverwendet werden.

Im Mathematikunterricht wird Lego Mindstorms bereits eingesetzt, etwa um den größten gemeinsamen Teiler und das kleinste gemeinsame Vielfache zu berechnen [Cont07]. Dafür sind weder Sensoren noch Motoren nötig, einzig zwei Zahlen müssen eingegeben werden. Dies kann etwa durch ein Auswählen der Ziffern 0 bis 9 für jede einzelne Dezimalstelle stattfinden.

Ein Projekt in New York City aus dem Jahre 2003, welches sich v.a. an sozial Schwächere richtete, widmete sich Themen der Mathematik und Physik. Dabei kam neben Mindstorms die klassische Programmieroberfläche RoboLab zum Einsatz. Behandelte Themen waren dabei u.a. Winkel, Drehmoment, Reibung oder der Zusammenhang zwischen Radumfang und Geschwindigkeit. Bei diesem Projekt zeigte sich auch, dass ein Interesse des Betreuers an den Arbeiten der Teilnehmer deren Motivation steigert. Weiters empfanden auch 14 bis 16-jährige das Arbeiten und Vorführen der Roboter als „cool“ [Eguc04].

In Griechenland wird Mindstorms u.a. an technischen und berufsbildenden Schulen eingesetzt. Eine Art des Einsatzes bezieht sich hier auf den Bau eines Fahrzeuges mit Gangschaltung. Schlagworte für ein solches Projekt sind etwa Geschwindigkeit und Drehmoment [Kali08].

4 Entwurf

Die folgenden Kapitel beschreiben, wie Lego Mindstorms und LeJOS im Unterricht unterschiedlicher Gegenstände sinnvoll eingesetzt werden können. Dabei handelt es sich teilweise um Roboter zum bloßen Vorführen von gewissen Sachverhalten durch die Lehrkraft, die meisten Roboter sollen allerdings von Schülern selbst gebaut und programmiert werden. Dabei ist stets zu beachten, dass es für Aufbau und Programmierung der jeweiligen Roboter nie eine einzelne korrekte Lösung gibt. Gerade wenn auch die jeweiligen Roboter, d.h. die „Hardware“, durch Schüler selbst konzipiert und gebaut werden, so muss auch die dafür programmierte Software darauf abgestimmt werden, was zu einer Vielzahl an Lösungen führt. Alle im Folgenden angeführten Codebeispiele sind nur zur Orientierung gedacht und um zu zeigen, welche Möglichkeiten die API¹ von LeJOS bietet.

4.1 Angewandte Mathematik

Da sich die Informatik aus der Mathematik entwickelt hat, sind diese beiden Wissenschaften untrennbar miteinander verbunden. Allerdings kann die Mathematik durch Lego Mindstorms deutlich anschaulicher gestaltet werden. Beispiele hierfür wären das Lösen eines Rubik's Cube (mit oder ohne Hilfe durch eine Datenbank), der Aufbau eines Dominostein-Kurses oder scheinbar simple Längenmessungen. Die beiden letztgenannten Beispiele werden hier aufgegriffen, da diese etwa sehr leicht auf korrektes Funktionieren überprüft werden können. Dies ist bei dem Lösen des Rubik's Cube zwar auch der Fall (alle sechs Flächen müssen einfarbig sein), allerdings sind sowohl Aufbau als auch Programmierung noch deutlich anspruchsvoller. Daher wird dieses an sich interessante Beispiel hier nicht beachtet.

4.1.1 Längenmessungen

Die Längenmessung ist die Grundlage für jeglichen Roboter, der Geschwindigkeiten oder Beschleunigungen messen oder selbständig, ohne Hilfe durch Sensoren

¹ Application Programming Interface - Programmierschnittstelle

(etwa Licht- oder Ultraschall), in einem bekannten Raum navigieren soll. Die Längenmessung soll hier auf dem selben Prinzip beruhen wie beim Tachometer, d.h. durch Berechnung des zurückgelegten Weges anhand der Umdrehungszahl. Für den zurückgelegten Weg gilt somit: $s = x * U$ (x : Anzahl der Umdrehungen; U : Umfang der Räder).

Mittels LeJOS ist es möglich, einen Motor genau ein Grad zu drehen. Kleinere Abstufungen sind nicht möglich. Das bedeutet für eine Längenmessung, dass hier die Verwendung von möglichst kleinen Rädern wichtig ist, da pro Umdrehungsgrad ein kürzerer Weg zurückgelegt wird und somit der mögliche Fehler geringer ist als bei größeren Rädern. Weiters kann die Geschwindigkeit in der Einheit $\frac{\text{Umdrehungsgrad}}{\text{Sekunde}}$ angegeben werden. D.h. „360“ entspricht einer Umdrehung pro Sekunde. Der höchstmögliche Wert liegt bei 900 ($\hat{=}$ 2,5 Umdrehungen) und ist durch die Servomotoren begrenzt.

Für die Längenmessung selbst gibt es zahlreiche Möglichkeiten was den Einsatz der zu verwendenden Sensoren betrifft und daraus resultierend auch zahlreiche Variationen sowohl für die Programmierung, als auch für das Starten und Beenden des Messvorganges. So kann dazu etwa der Helligkeitssensor verwendet werden, falls die Länge eines Objekts mit klarem Kontrast zum Hintergrund gemessen werden soll, z.B. ein am Boden mittels Klebestreifen abgeklebter Kurs. Genauso kann auch der Drucksensor eingesetzt werden, wenn sich am Ende der Messstrecke ein geeignetes Hindernis befindet, um diesen auszulösen.

Auch die abschließende Berechnung der Länge kann auf mehrere Arten geschehen. Eine Möglichkeit basiert auf der Anzahl der Umdrehungen der Räder. Dazu wird mittels einer Schleife der Roboter um den kleinstmöglichen Wert vorwärts bewegt. Abbruchbedingung muss hier die Meldung des verwendeten Sensors sein, also etwa das Auslösen des Druckkontaktsensors oder das Registrieren einer gewissen Helligkeitsstufe durch den Lichtsensor.

```
while (licht.readNormalizedValue() <= 400) {  
    Motor.A.rotate(1)  
}
```

Diese while-Schleife wird so lange ausgeführt, so lange die licht-Instanz des Helligkeitssensors einen Wert von höchstens 400 retourniert. Nur wenn in die Schleife eingetreten wird, wird der Motor A um ein Grad gedreht. Sobald der Helligkeitswert über 400 liegt, wird die Schleife verlassen (falls sie gestartet wurde).

Eine weitere Möglichkeit bietet eine schlichte if-Anweisung. Der Roboter muss sich dazu bewegen, sobald er nicht mehr den benötigten Wert misst, stoppt er. Dies kann wie folgt aussehen.

```
Motor.A.forward();
```

```
if (licht.readNormalizedValue > 400)
    Motor.A.stop();
```

Um die Länge der Strecke zu bestimmen, kann entweder ein eigener Zähler verwendet werden oder man verwendet die bereits in der API vorhandenen Methoden „resetTachoCount()“ und „getTachoCount()“, durch welche sich die Gradzahl der Umdrehungen zurücksetzen bzw. auslesen lässt.

Eine andere Möglichkeit als das Zählen der Umdrehungen wäre das Messen der benötigten Zeit. Die Geschwindigkeit sollte der Einfachheit halber über die komplette Strecke als konstant angenommen werden und ist durch die Angabe in $\frac{\text{Umdrehungsgrad}}{\text{Sekunde}}$ bekannt, muss für die Berechnung aber mittels des Raddurchmessers in eine Einheit wie $\frac{\text{Centimeter}}{\text{Sekunde}}$ umgerechnet werden. Da Geschwindigkeit und Zeit dann bekannt sind, gilt für die zurückgelegte Strecke: $s = v * t$.

Interessant an den unterschiedlichen Methoden für den Einsatz im Mathematikunterricht ist v.a. die Möglichkeit, die bei den jeweils verschiedenen Methoden auftretenden Messfehler zu berechnen und zu vergleichen. Ein mögliches Projekt wäre auch eine Minimierung der Messfehler, sei es durch bessere Roboter oder durch besseren Code. Dabei wäre es z.B. bei der Programmierung denkbar, Messwerte je nach Größe gezielt zu manipulieren, um den jeweils auftretenden Fehler zu minimieren. Dazu müssten jedoch Messungen vorangehen, durch welche der auftretende Fehler in bestimmten Intervallen möglichst exakt bestimmt wird.

Hier wird das Modell des Zählens der Umdrehungen bevorzugt. Ein Aktivitätsdiagramm für dieses Modell ist in Abbildung 4.1 dargestellt, der entsprechende Java-Code findet sich in Programmcode 4.1.

Das Zählen der Umdrehungen mittels „getTachoCount()“ ist auch deshalb eine „bequeme“ Art der Programmierung, da die Gradzahl der Umdrehungen direkt vom jeweiligen Servomotor ausgelesen werden kann. D.h., wie im Diagramm zu sehen, erfolgt eine Rücksetzung des Zählers („resetTachoCount()“) nachdem zum ersten Mal zwei gleiche Helligkeitswerte hintereinander gemessen wurden. Der Zähler läuft dann genau so lange, bis wieder ein anderer Helligkeitswert gemessen wird. Sobald dies eintritt, wird der Motor gestoppt, der Zähler (x) ausgelesen und anhand dessen die Länge der Strecke berechnet. Dazu notwendig ist die Angabe der Radgröße, z.B. des Durchmessers (d), welcher bei Lego-Rädern meist angegeben ist. Der hier konzipierte Roboter misst also die Länge einer Linie. Der Helligkeitssensor muss dazu noch vor Beginn der Linie mit der Messung beginnen können. Die zurückgelegte Strecke berechnet sich dann wie folgt:

$$s[\text{mm}] = \frac{x[^\circ]}{360^\circ} * d[\text{mm}] * \pi$$

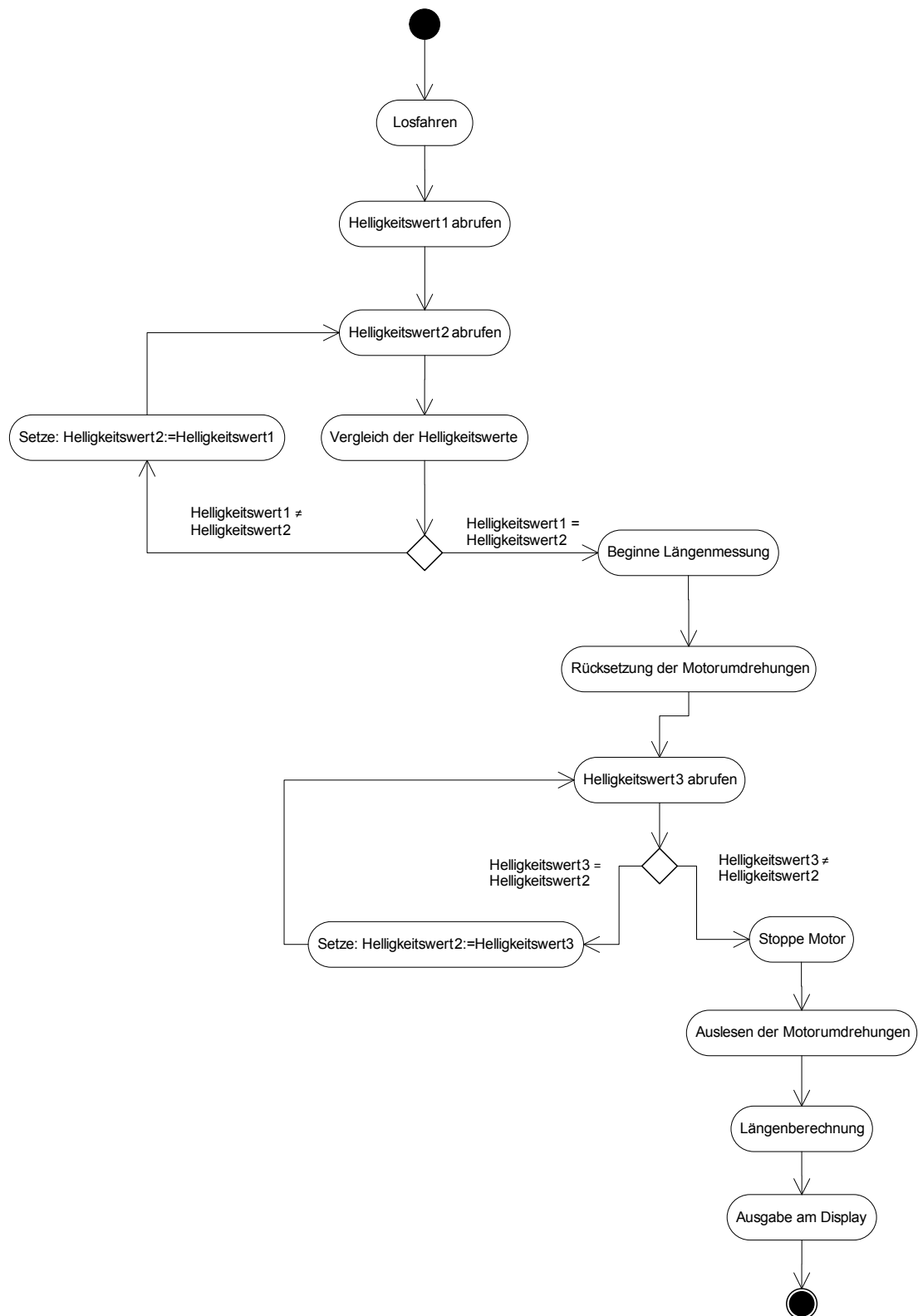


Abbildung 4.1: Längenmessung - Aktivitätsdiagramm

Folglich könnte die Codepassage zur Berechnung des zurückgelegten Weges wie in Programmcode 4.1 aussehen.

Programmcode 4.1: Längenmessung - Berechnung des zurückgelegten Weges

```
//stoppe Motor an Port A
Motor.A.stop();
//Festlegung des Raddurchmessers in mm
float durch = 56.0f;
//Umdrehungszahl von Motor A in Integer "Winkel" speichern
int winkel = Motor.A.getTachoCount();
final float PI = 3.141593f;
//Zurückgelegte Strecke berechnen
float strecke = ((float>winkel/360)*durch*PI;
//Umwandeln der Vorkommastellen der Gleitkommazahl in ein Integer
int vk = (int)strecke;
//Berechnung von drei Nachkommastellen mittels float
float nkh = (strecke - vk)*1000;
//Umwandeln der Nachkommastellen in int-Wert
int nk = (int)nkh;
//Ausgabestring generieren
String ausgabe = vk+"."+nk;
//Anzeige des Ausgabestrings solange Escape-Taste nicht gedrückt
wird
while (!Button.ESCAPE.isPressed()) {
    LCD.clear();
    //Ausgabe in Zeile 1 (zweite 0), nicht eingerückt (erste 0)
    LCD.drawString("Gemessene Strecke in mm:", 0,0);
    //Ausgabe in Zeile 2, nicht eingerückt
    LCD.drawString(ausgabe, 0,1);
    LCD.refresh();
}
```

Für die Berechnung des tatsächlich zurückgelegten Weges findet ein sogenanntes Casting statt, d.h. es wird dem vom Motor ausgelesenen Winkel der Datentyp „float“ zugewiesen. Dies ist nötig, da sonst bei Winkeln $< 360^\circ$ die entsprechende

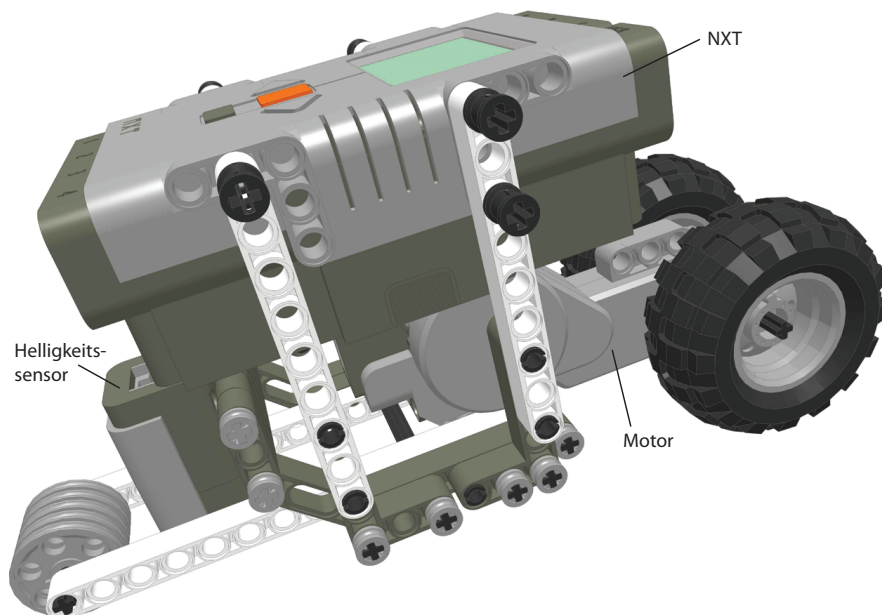


Abbildung 4.2: Längenmessung - Aufbau

Division den Wert „0“ ergeben würde, was auch „0“ als Endergebnis zur Folge hätte. Ein Winkel von beispielsweise 180° bedeutet aber klarerweise nicht, dass sich der Roboter nicht fortbewegt hat.

Gleitkommazahlen können allerdings nicht direkt an das Display zur Ausgabe übergeben werden, weshalb in diesem Beispiel eine Umwandlung in Vor- und Nachkommastellen erfolgt, welche dann, getrennt durch einen Punkt, als String-Datentyp ausgegeben werden.

Ein Modell zur Realisierung dieser Planungen könnte wie in Abbildung 4.2 aussehen. Dabei sorgt ein Motor für den Antrieb, vorne befinden sich ein Stützrad sowie der Helligkeitssensor und in der Mitte ist der NXT angebracht.

4.1.2 Domino-Strecke

Der jährlich stattfindende Domino Day ist mittlerweile wohl einem Großteil der fernsehenden Bevölkerung bekannt, gewollt oder ungewollt. Dabei geht es um das richtige Aufstellen möglichst vieler Domino-Steine, so dass sie sich beim Umfallen gegenseitig umstoßen und somit eine Kettenreaktion auslösen. Dabei soll der Kurs möglichst problemlos aufgebaut werden und an einem vorgegebenen Stein die Reaktion ausgelöst werden.

Mittels Mindstorms-Robotern sind mehrere Arten des Aufbaus solcher Domino-Strecken denkbar. Die größte Schwierigkeit stellt dabei, neben dem Absetzen der Steine, das korrekte Aufstellen von Kurven dar, je nachdem welche Art des Aufbaus gewählt wird.

Denkbar wäre zum einen ein Aufbau nach dem Prinzip des Line-Followers, d.h. der Roboter folgt mittels eines Helligkeitssensors einer bestimmten Linie und stellt auf oder neben dieser die Domino-Steine ab. Größtes Problem bei dieser Methode wäre, dass der Roboter den Streckenverlauf nicht kennt und somit der Aufbau der Kurven zu erheblichen Problemen führen kann. Leicht vorzustellen ist dies etwa an dem Beispiel, wenn eine aufgezeichnete Kurve nicht rund ist, sondern etwa einen spitzen Innenwinkel von 30° hat.

Eine weitere Möglichkeit wäre das fixe Programmieren eines zu bauenden Kurses. Möglich wäre dies etwa durch die Angabe der Radumdrehungszahlen, welche der Roboter bis zur nächsten Wendung auszuführen hat. Kurven könnten als Methoden definiert werden, etwa „ 15° -Kurve nach rechts“. Mittels solch einer Methode wäre eine 15° -Drehung nach rechts ebenso möglich wie etwa eine 120° -Drehung ($120 = 8 * 15$).

Vergleichbar mit der fixen Programmierung wäre auch die Möglichkeit der Fernsteuerung des Roboters via Bluetooth. Als Fernsteuerung könnte hier entweder ein Computer oder ein geeignetes Mobiltelefon eingesetzt werden. Sinnvoll wäre

auch hier die fixe Programmierung der Kurven und mittels Fernsteuerung könnte dann die jeweils zu bauende Kurve bzw. die Geradeausfahrt ausgewählt werden.

Die für den Einsatz im Mathematikunterricht sinnvollste Variante ist allerdings wohl jene, bei welcher der Roboter so lange geradeaus fährt, bis ein Hindernis in einem definierten Abstand auftaucht. Der Abstand zu Hindernissen wird mittels des Ultraschallsensors gemessen. Taucht solch ein Hindernis auf, so fährt/baut der Roboter eine Kurve. Es kann auch durch ein Drehen des Ultraschallsensors erreicht werden, dass der Roboter nicht in die falsche Richtung „abbiegt“.

Wichtig beim Aufbau eines Domino-Kurses ist v.a. die Beachtung des Abstandes der einzelnen Steine zueinander, d.h. der Abstand zwischen den Steinen (d) muss geringer sein als die Kantenlänge (h) der senkrecht stehenden Kante (siehe Abb. 4.3).

Denkbar ist hier auf einer Geraden etwa ein Abstand von $d = \frac{1}{2} * h$. Somit sind die nötigen Angaben für das Aufstellen der Steine auf einer Geraden: Umfang der angetriebenen Räder des Roboters sowie Kantenlänge und Dicke der Dominosteine.

Die Dicke ist deshalb erforderlich, da diese zum jeweiligen Abstand addiert werden muss. Leicht vorzustellen ist dies an einem Extrembeispiel etwa mit einem Abstand von 2cm und einer Dicke von 4cm. Würde der Roboter nur 2cm weit fahren, um den nächsten 4cm breiten Stein abzusetzen, würde dieser den vorherigen umstoßen. Fährt er jedoch $2+4 = 6$ cm und setzt erst dort den Block ab, so hat der 4cm breite Block genau 2cm Abstand zum Vorherigen.

Die nötigen Angaben können entweder alle als fixe Werte festgelegt oder zumindest teilweise vor dem Start durch Nutzereingaben erfragt werden. Es wäre auch durchaus möglich, den Abstand zu einem Hindernis, bei welchem der Roboter eine Kurve fährt, vom Benutzer etwa aus vorgegebenen Werten wählen zu lassen. Je größer der Abstand zum Hindernis, desto größer könnte auch der Kurvenradius gewählt werden. Wesentlich einfacher ist es allerdings, nur zwei Arten von Kurven zu integrieren, Linkskurve und Rechtskurve, da jeder unterschiedliche Kurvenradius extra implementiert werden müsste. Ein Aktivitätsdiagramm zur Umsetzung dieses Roboters könnte wie in Abbildung 4.4 dargestellt aussehen.

Daraus geht klar die Notwendigkeit zur Verwendung von Methoden hervor.

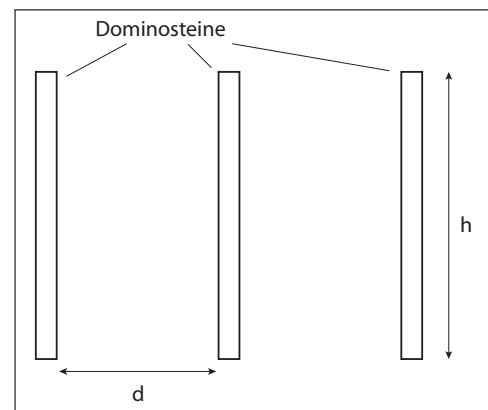


Abbildung 4.3: Domino-Strecke - Abstand der Steine - Profil

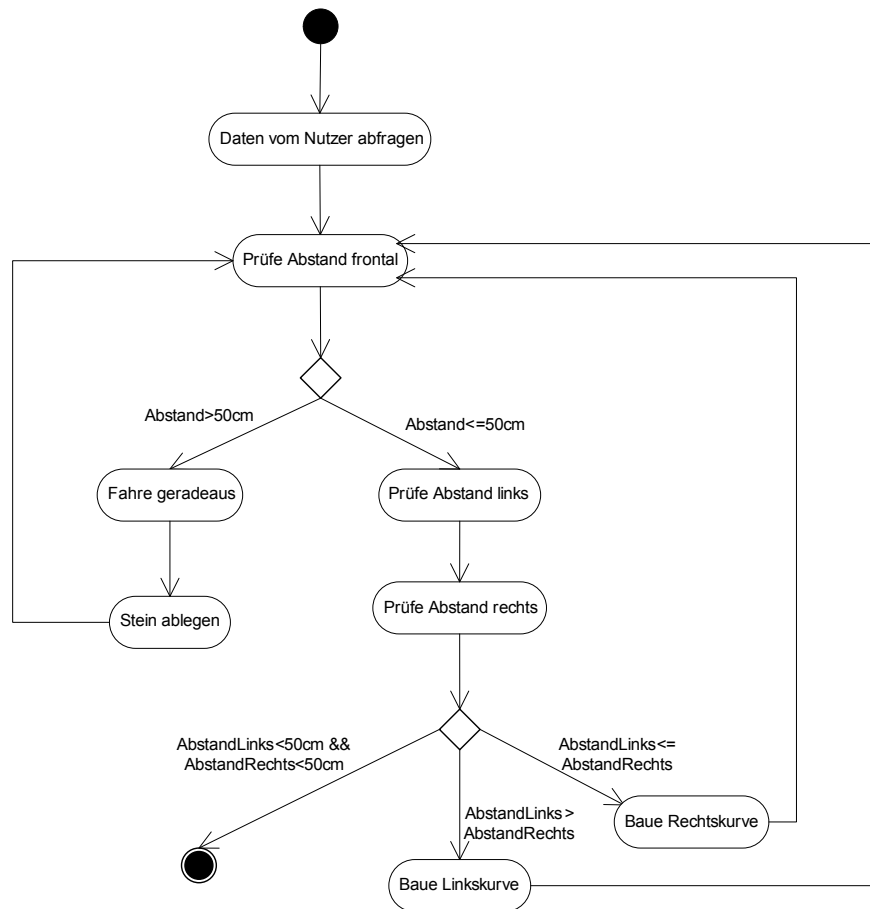


Abbildung 4.4: Domino-Strecke - Aktivitätsdiagramm

Wäre es im vorherigen Beispiel der Längenmessung noch möglich, den kompletten Code in der Main-Methode unterzubringen, ist dies hier nicht mehr auf sinnvolle Weise möglich. Daher sollten hier für das Fahren der Links- und Rechtskurven, für das Abstellen der Domino-Steine und auch für das Fahren auf der Geraden eigene Methoden deklariert werden.

Programmcode 4.2 zeigt eine Möglichkeit der Programmierung der Methode „goStraight(float durch, float h, float d, float ab)“, mittels welcher der Roboter durch die Übergabe der Variablen „Durchmesser der Räder“, „Höhe der Steine“, „Dicke der Steine“ sowie „Abstand der Radmittelpunkte zueinander“ die Distanz $\frac{1}{2} * h + d$ geradeaus fährt.

Programmcode 4.2: Domino-Strecke - Vorwärtsbewegung

```

public static void goStraight(float durch, float h, float d, float
    ab) {
    /*Berechnung der zu fahrenden Strecke, hier: d + h/2*/
    float strecke = d+(h/2);
    /*Erstellen einer Instanz der Pilot-Klasse, mittels welcher die
       Bewegungen zweier Motoren synchronisiert werden können. Nötig
       sind dazu die Angabe des Durchmessers (durch), des Abstandes der
       Radmittelpunkte (ab) sowie die Ports der beiden Motoren. Die
  
```

```

    Einheiten von Durchmesser und Abstand müssen nicht angegeben
    werden, müssen jedoch in den selben Einheiten vorliegen (etwa cm
    ) und als float angegeben sein*/
Pilot pilot = new Pilot(durch, ab, Motor.A, Motor.B);
/*Angabe der Geschwindigkeit für die in Pilot zusammengefassten
    Motoren in Grad/Sekunde*/
pilot.setSpeed(360);
/*Pilot-Motoren synchron die Strecke fortbewegen*/
pilot.travel(strecke);
/*Aufrufen der Methode zum Absetzen eines Steines (hier nicht
    angeführt)*/
dropStone();
}

```

Die hier angeführte Pilot-Klasse ist eine Möglichkeit der Synchronisation zweier Motoren. Laut Oliver Bückner [Bück07] ist allerdings auch dadurch keine vollständige Synchronisation zu erreichen, was an der Ungenauigkeit von einem Grad bei jedem einzelnen Motor liegt. Eine Synchronisation könnte auch selbst programmiert werden, mittels while-Schleifen und Vergleich von „getTachoCount()“ der beiden Motoren. Der langsamere Motor kann dabei beschleunigt oder der schnellere gebremst werden. Näher wird auf diese Methode hier nicht eingegangen.

Die im vorhergehenden Beispiel schon angeführte Methode „dropStone()“ dient dem Absetzen der Domino-Steine. Sie wird immer nach dem Fahren eines gewissen Teilstückes ausgeführt, sei es eine Gerade oder ein Teilstück einer Kurve, da das Absetzen immer auf die gleiche Weise funktionieren wird. Auf die Programmierung des Absetzens selbst wird hier nicht näher eingegangen, da es dazu äußerst viele Möglichkeiten gibt, wesentlich mehr als zur korrekten Navigation und wesentlich stärker abhängig von der jeweils gewählten Bauweise. Wichtig ist hier nur, dass der Stein korrekt aufgesetzt wird (d.h. etwa parallel zur Achse des Roboters) und dass er nach dem Aufsetzen nicht umfällt, weder durch unsauberes Aufstellen, noch durch eine Berührung mit dem Roboter.

Wesentlich schwieriger wird es bei der Gestaltung der Kurvenfahrten. Prinzipiell gibt es hier drei Möglichkeiten, diese auszuführen: sie kann komplett einhändig programmiert werden, d.h. jedem Motor werden die nötigen Anweisungen einzeln gegeben. Die Pilot-Klasse bietet allerdings auch zwei gut dafür geeignete Methoden, „steer(int turnRate, int angle)“ und „rotate(int angle)“. Der Unterschied dieser Methoden liegt darin, dass „rotate(int angle)“ den Roboter am Stand dreht, indem sich beide Motoren in entgegengesetzte Richtungen drehen, während „steer(int turnRate, int angle)“ den Roboter in einer Kreisbahn bewegt.

Programmcode 4.3 zeigt den Einsatz der Methode „rotate(int angle)“. Die Umsetzung kann hier wesentlich einfacher erfolgen als mittels „steer()“.

Programmcode 4.3: Domino-Strecke - Kurvenfahrt nach rechts

```

public static void goRight(float durch, float h, float d, float ab)
{
    /*Berechnung der tatschlich zu fahrenden Strecke, gleich lang wie
       in goStraight. berprfung, ob diese Strecke lang genug ist,
       findet in main-Methode statt*/
    float strecke = d+(h/2);
    /*Erstellen einer Instanz der Pilot-Klasse*/
    Pilot pilot = new Pilot(durch, ab, Motor.A, Motor.B);
    /*Angabe der Geschwindigkeit fr die in Pilot zusammengefassten
       Motoren in Grad/Sekunde*/
    pilot.setSpeed(360);
    /*Zhlervariable erstellen*/
    int c = 0;
    /*Schleife wird genau 3 Mal durchlaufen, d.h. es erfolgt eine
       Umdrehung um 3*30=90 Grad*/
    while(c<3) {
        /*Pilot-Motoren synchron die festgelegte Strecke fortbewegen*/
        pilot.travel(strecke);
        /*Roboter um 30 nach rechts drehen (negativer Wert bedeutet
           Drehung nach rechts, positiver Wert Drehung nach links). Die
           Motoren drehen sich dabei entgegengesetzt, um eine Drehung am
           Stand zu erreichen.*/
        pilot.rotate(-30);
        /*Aufrufen der Methode zum Absetzen eines Steines (hier nicht
           angefhrt)*/
        dropStone();
        /*Erhhung der Zhlervariable um 1*/
        c++;
    }
}

```

Wie in dem Code zu sehen ist, erfolgt vor dem jeweiligen Drehen noch eine genauso lange Fahrt geradeaus wie in der Methode „goStraight(float durch, float h, float d, float ab)“ um zu verhindern, dass durch das Drehen ein Stein umgestoen wird. Die Drehung selbst erfolgt dann mittels einer while-Schleife, welche drei Mal ausgefhrt wird, um drei Mal eine Drehung von je 30 auszufhren. Es kommen natrlich auch andere Gradzahlen fr die Drehung in Frage, fr 90-Kurven bieten sich aber klarerweise Teiler ohne Restwert wie eben 30 oder auch 15 oder 18 an.

Doch nun stellt sich eine grundstzliche Frage: wie berechnet man, ob sich der Roboter in der vorgegebenen Entfernung $\frac{1}{2} * h + d$ um die vorgegebene Gradzahl (hier 30) drehen kann, ohne den vorher aufgestellten Stein umzuwerfen? Dazu sind einige Annahmen zu treffen. Zum einen ist fr die Berechnung davon auszugehen, dass die aufzustellenden Domino-Steine die hinterste und uerste Stelle des Roboters darstellen und somit durch kein anderes Teil des Roboters eine Gefahr fr die bereits stehenden Steine besteht. Weiters gilt es anzunehmen, dass die Mitte der Dominosteine auch Mittelpunkt des Rotationskreises ist. Dies ist dann der Fall, wenn der Mittelpunkt der Antriebsachse und das Zentrum der zu plat-

zierenden Dominosteine übereinander liegen. Dies ist zwar nur näherungsweise zu erreichen, für die hier zu tätige Berechnung kann dies allerdings vernachlässigt werden. Abbildung 4.5 zeigt eine Skizze zur Veranschaulichung der Berechnung.

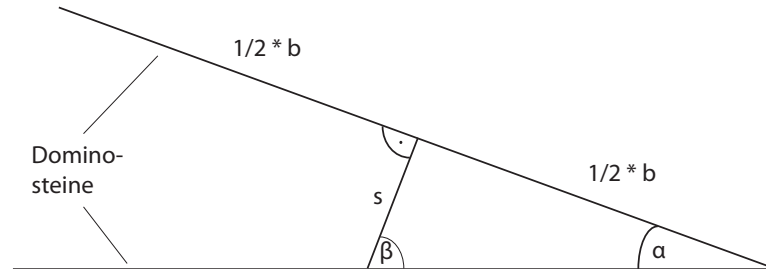


Abbildung 4.5: Domino-Strecke - Mindestabstand der Steine in Kurven

Der Winkel α stellt jenen Winkel dar, um welchen sich der Roboter jeweils stückweise dreht, im Codebeispiel also 30° . β stellt den Winkel zwischen dem zuletzt aufgestellten Stein und der zu fahrenden Geraden dar, d.h. bei einer Drehung um 30° beträgt dieser Winkel $90^\circ - 30^\circ = 60^\circ$. b stellt die Breite der Dominosteine und s die Strecke zwischen diesen dar. Die Skizze zeigt den Fall, dass sich die Steine an der Innenseite gerade noch berühren, wodurch ein rechtwinkliges Dreieck entsteht, von welchem mindestens zwei Angaben bekannt sein müssen, um auf die dritte schließen zu können. Für das rechtwinklige Dreieck gilt: $\tan \alpha = \frac{s}{\frac{1}{2} * b}$. Ist der Winkel bekannt, in welchem gedreht werden soll, so wie es hier der Fall ist, so gilt für s : $s = \tan \alpha * \frac{1}{2} * b$. Dies entspricht der mindestens zu fahrenden Strecke, um nicht durch die Drehung den vorherigen Stein umzu stoßen. Die in der Gerade zwischen zwei Steinen zu fahrende Strecke beträgt wie bereits erläutert $\frac{1}{2} * h + d$ (d = Dicke der Steine). Soll der Roboter auch in Kurven jeweils Abschnitte von $\frac{1}{2} * h + d$ fahren, so muss gelten: $\tan \alpha * \frac{1}{2} * b \leq \frac{1}{2} * h + d \Rightarrow \tan \alpha * b \leq h + 2 * d$. Natürlich könnten hier auch andere Grenzwerte gesetzt werden. Wenn etwa ein Drehen um den vorgegebenen Winkel wegen Kollision nicht möglich ist, so könnte auch geprüft werden, wie weit der Roboter noch geradeaus fahren kann, um einerseits ein kollisionsfreies Drehen zu erreichen, andererseits aber den Abstand nicht zu groß werden zu lassen, um die Kettenreaktion nicht zu unterbrechen.

Durch die zuletzt angegebene Gleichung wird klar, dass der „Mindestabstand“ mit dem Drehungswinkel ($0 \leq \alpha < 90$) und der Breite der Steine zunimmt. Bei der hier angewendeten Methode ist außerdem klar, dass sich die Untergrenze des Mindestabstandes umso weiter nach oben verschiebt, je höher und dicker die Steine sind.

Die wohl letzte entscheidende Frage ist nun, wann der Roboter eine Kurve fährt und in welche Richtung er diese fahren soll. Wie bereits im Aktivitätsdiagramm

dargestellt, soll der Roboter bis auf eine Distanz von 50cm an ein Hindernis heranfahren, dann den Abstand nach links und rechts prüfen und daraufhin entscheiden, in welche Richtung er fährt oder ob Aufgrund der zu geringen Distanz von Hindernissen beidseits der Bau gestoppt wird. Die hier gewählte kritische Distanz von 50cm kann natürlich verändert werden, wichtig ist v.a. dass der jeweils eingesetzte Roboter nicht während der Drehung aufgrund zu großer Länge an einem Hindernis ansteht. Der Abstand muss außerdem groß genug sein, um mit den gewählten Steinen eine 90°-Kurve bauen zu können, ohne dass diese von einem Fremdkörper behindert werden.

Es ist jedenfalls davon auszugehen, dass der dritte verfügbare Motor verwendet wird, um den Ultraschallsensor jeweils um 90° nach links und rechts drehen zu können. Die einfachste Möglichkeit wäre es nun, den Sensor jeweils die kompletten 90° zu drehen, mittels „getDistance()“ die jeweilige Entfernung abzufragen und in jene Richtung zu drehen, in welcher der Abstand größer ist, bei Gleichstand der Entfernung in eine festgelegte oder zufällig ausgewählte Richtung zu fahren und bei zu geringer Entfernung beidseits wäre die Fahrt abubrechen, da ja auch eine Weiterfahrt geradeaus nicht mehr möglich ist. Hierbei tritt allerdings das Problem auf, dass die im Winkel von 90° gemessene Distanz nicht sehr aussagekräftig ist. Denn da der Roboter sich nicht am Stand dreht, sondern er noch eine Kurve bauen soll und er sich dabei sowohl nach vorne, als auch in die jeweilige Kurvenrichtung bewegt, ist es sinnvoller zu überprüfen, ob die Distanz zu wichtigen Kurvenpunkten groß genug ist. Natürlich soll auch die Entfernung im 90°-Winkel überprüft werden, allerdings gibt dies keine finale Auskunft darüber, ob tatsächlich eine Weiterfahrt nach Vollendung des Kurvenbaus möglich ist, da während der Kurve weiterhin eine Vorwärtsbewegung stattfindet, und innerhalb dieser ein Hindernis auftauchen kann. Um solche Hindernisse frühzeitig zu erkennen, müsste der eingesetzte Ultraschallsensor etwa ausfahrbar gelagert oder weit genug vor dem Mittelpunkt der Antriebsachse (= Mittelpunkt der Rotation) angebracht sein, um somit auch die folgende Spur des Roboters auf Hindernisse überprüfen zu können.

Programmcode 4.4 führt eine Überprüfung des Abstandes nach rechts an. Die Methode „checkRight(double winkel, float h, float d)“ soll genau dann aufgerufen werden, wenn der Ultraschallsensor frontal einen kritischen Abstand misst, etwa 50 cm oder weniger wie im angeführten Aktivitätsdiagramm. Diese kritische Entfernung muss natürlich an die jeweils verwendeten Steine und die Distanz, in welcher diese platziert werden sollen, angepasst werden. Mittels „checkDistance-Right()“ wird für alle drei mittels „goRight“ aufzustellenden Steine überprüft, ob der Abstand zur jeweiligen Mitte der zu positionierenden Steine groß genug ist

oder ob ein Hindernis den Weg versperrt. Mit einigem Mehraufwand könnte etwa auch geprüft werden, ob die linke und rechte Ecke der Steine ebenfalls platziert werden kann, dies hier soll jedoch lediglich eine Veranschaulichung darstellen. „Positiv“ an einer Überprüfung aller Ecken der Steine ist jedenfalls ein nicht zu vernachlässigender mathematischer Aufwand. Wie bereits beschrieben wird außerdem die Entfernung im 90°-Winkel gemessen, um zu bestimmen, in welche Richtung bei möglicher Positionierung der Steine gefahren werden soll. Ist der Abstand für alle Steine groß genug, so wird der Abstand im 90°-Winkel retourniert, schon bei Unterschreitung des Minimalwertes durch nur einen gemessenen Wert wird -1 retourniert, um die rechte Seite als Alternative auszuschließen. Eine grafische Veranschaulichung dieser Methode findet sich in Abbildung 4.6.

Programmcode 4.4: Domino-Strecke - Abstandsprüfung nach rechts

```
public static int checkRight(double winkel, float h, float d) {
    /*Erstellen einer Instanz des Ultraschall-Sensors namens "sonic"*/
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S1);
    /*Festlegen der Geschwindigkeit von Motor C auf 180°/Sekunde*/
    Motor.C.setSpeed(180);
    /*Gegenwinkel = 90° - Winkel; der Winkel ist jener Winkel, um
       welchen sich der Roboter dreht und wird der Methode übergeben*/
    double gegenwinkel = 90 - winkel;
    /*Berechnung von Cosinus und Sinus für Winkel und Gegenwinkel
       mittels der durch die Math-Klasse zur Verfügung gestellten
       Methoden*/
    double sinw = Math.sin(winkel);
    double cosw = Math.cos(winkel);
    double sing = Math.sin(gegenwinkel);
    double cosg = Math.cos(gegenwinkel);
    /*Festlegung der Distanz zwischen den Mittelpunkten für die
       abzusetzenden Steine. Hier genauso lang wie auf der Geraden.
       Einheit: mm*/
    float dist = d+(h/2);
    /*Festlegen des Mindestabstandes für 90°-Winkel in cm. cm deshalb,
       da der Ultraschallsensor cm-Werte retourniert*/
    int mindest = 50;
    /*Das erste Minimum entspricht genau dist, da zur Vorsicht schon
       bei 0° die Entfernung gemessen wird. Da der Ultraschall-Sensor
       Integer-Werte retourniert, muss der float-Wert zum Vergleich in
       einen int-Wert gecastet werden*/
    int min1 = (int)dist;
    /*Zweites und drittes Minimum berechnen sich mittels Satz des
       Pythagoras*/
    double min2 = Math.sqrt((cosg*dist)*(cosg*dist)+(dist+sing*dist)*(
        dist+sing*dist));
    double min3 = Math.sqrt((cosg*dist+cosw*dist)*(cosg*dist+cosw*dist)
        +(sinw*dist+dist+sing*dist)*(sinw*dist+dist+sing*dist));
    /*Berechnung des Winkels zwischen Standpunkt und Messpunkt, um zu
       bestimmen, um welchen Winkel der Motor gedreht werden muss.
       Dabei gilt: tan(winkel)=höhe/breite. Hier wird zuerst die Höhe
       durch die Breite dividiert, um anschließend einen einzelnen Wert
       übergeben zu können*/
    double w1 = (cosg*dist)/(dist+sing*dist);
```

```

double w2 = (cosg*dist+cosw*dist)/(sinw*dist+dist+sing*dist);
/*Berechnung der Winkel mittels arctan aus den eben berechneten w1
   und w2*/
double winkel1 = Math.atan(w1);
double winkel2 = Math.atan(w2);
/*Umwandeln des Datentyps von float zu int, da int für das
   Rotieren des Motors benötigt wird*/
int win1 = (int) winkel1;
int win2 = (int) winkel2;
/*Messung des Abstandes bei 0° in Centimeter*/
int dist1 = sonic.getDistance();
/*Falls Abstand kleiner als das nötige Abstandsminimum, so wird
   dist1 auf -1 gesetzt, um später überprüfen zu können, ob einer
   der Abstände die Anforderungen nicht erfüllt. dist1 wird mit 10
   multipliziert, da er die Einheit cm hat, min1 aber die Einheit
   mm*/
if(dist1*10 < min1)
    dist1 = -1;
/*Motor um berechneten Winkel win1 rotieren. Abhängig von
   Einbaurichtung des Motors wäre es auch möglich, ein negatives
   Vorzeichen zu setzen.*/
Motor.C.rotate(win1);
int dist2 = sonic.getDistance();
if(dist2*10 < (int)min2)
    dist2 = -1;
/*Motor um Differenz aus Winkel2 und Winkel1 drehen, um genau
   Winkel2 zu erreichen.*/
Motor.C.rotate(win2-win1);
int dist3 = sonic.getDistance();
if(dist3*10 < (int)min3)
    dist3 = -1;
/*Motor um die Differenz aus 90° und den beiden Winkeln drehen, um
   eine Drehung von genau 90° zu erreichen*/
Motor.C.rotate(90-win2-win1);
int dist4 = sonic.getDistance();
if(dist4*10 < mindest)
    dist4 = -1;
/*Motor C wieder in Ausgangsstellung bringen*/
Motor.C.rotate(-90);
/*Falls bei einer oder mehreren Messungen ein Wert unter dem
   benötigten Wert festgestellt wurde*/
if(dist1 == -1 || dist2 == -1 || dist3 == -1 || dist4 == -1) {
    /*Rückgabe des Wertes -1*/
    return -1;
}
else
    /*Sonst Rückgabe des bei Drehung um 90° gemessenen Abstandes*/
    return dist4;
}

```

Die Berechnung der Seitenlängen erfolgt zwar mittels Pythagoras, allerdings nicht als $a^2 + b^2 = c^2$ sondern als $a * a + b * b = c * c$. Dies liegt daran, dass bei Verwendung der durch die Math-Klasse zur Verfügung gestellten Potenz-Methode häufig Fehler am NXT auftraten.

Abschließend ist zu bemerken, dass der Roboter nicht zwangsweise 90°-Kurven

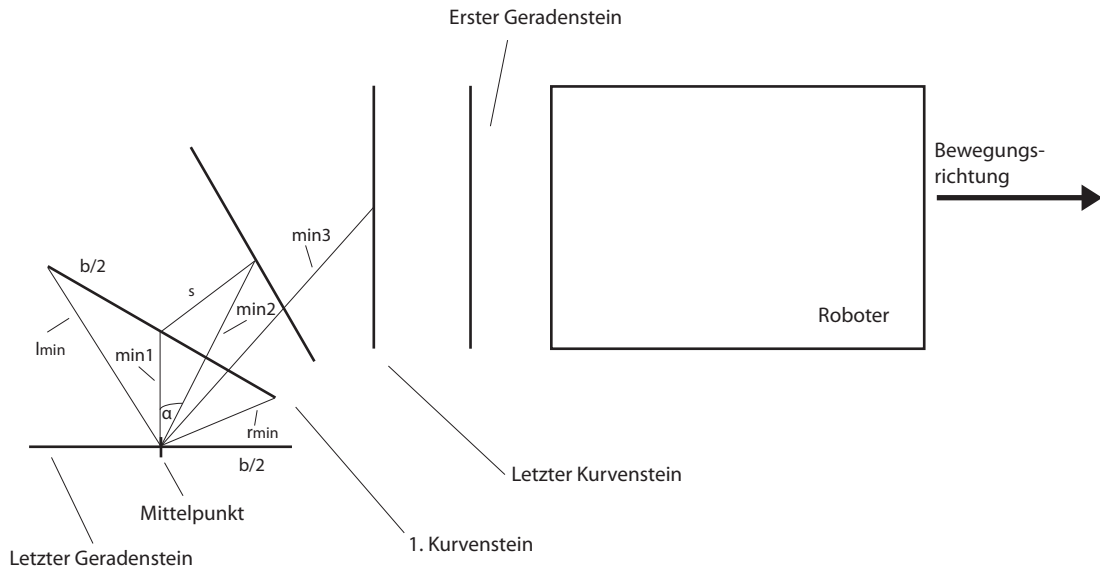


Abbildung 4.6: Domino-Strecke - Kurvenfahrt nach rechts

bauen muss. Es wäre natürlich auch möglich, etwa schon nach einer Drehung um 30° zu prüfen, ob nun schon eine Weiterfahrt gradeaus möglich ist. Ebenso wäre ein Absetzen des Steines auch vor der jeweiligen „Fahreinheit“ möglich. Jedoch gilt hier wie bei allen Programmieraufgaben: es gibt nahezu unendlich viele Arten der Umsetzung. Weiters wäre natürlich durch das Verbauen einer Radübersetzung auch das Erreichen einer höheren Umdrehungszahl als $2,5 \frac{\text{Umdrehungen}}{\text{Minute}}$ möglich, was hier jedoch kaum sinnvoll erscheint. Durchaus sinnvoll wäre eine Untersetzung, um bei Kurven, Abstand zwischen Domino-Steinen oder der Drehung des Ultraschallsensors eine Minimierung des motorbedingten Fehlers zu erreichen.

4.2 Angewandte Physik

Der Physikunterricht bietet zwar auch ohne Lego Mindstorms einiges an Praxisbezug, mittels Mindstorms können jedoch einzelne Sachverhalte sehr gut simuliert werden. Beispiele hierfür wären „Abschussrampen“ für Bälle oder Papierflugzeuge, eine genaue Messung der Fallbeschleunigung oder auch eine Messung des Schalldruckpegels. Auf diese und weitere Beispiele für den Physikunterricht wird im Folgenden genauer eingegangen.

4.2.1 Geschwindigkeit und Beschleunigung

Im ersten Jahrgang der Angewandten Physik sind laut Lehrplan Geschwindigkeit und Beschleunigung zu unterrichten. Dazu gibt es hier jeweils zwei verschiedene Arten, nach welchen der Stoff gelehrt werden kann.

4.2.1.1 Vorführung durch Lehrkraft

Bei dieser Art des Unterrichts führt die Lehrkraft mittels eines Roboters Beschleunigungsvorgänge und Fahrten mit bestimmten Geschwindigkeiten vor. Die Geschwindigkeit kann durch die Schüler etwa manuell durch Festlegen einer gewissen Strecke und das Stoppen der Zeit ermittelt werden. Ein sinnvolles Messen der Beschleunigung ist wohl nur dann möglich, wenn die Motoren mittels der Methode „flt()“ auslaufen anstatt mittels „stop()“ sofort angehalten werden.

4.2.1.2 Programmierung durch Schüler

Für Schüler wesentlich anspruchsvoller und wohl auch interessanter ist die selbstständige Programmierung eines Roboters, welcher vorgegebene Geschwindigkeiten fährt bzw. bestimmte Beschleunigungsvorgänge durchführt.

Der Aufbau eines solchen Roboters kann recht beliebig gewählt werden. Da hier nur Geschwindigkeit und Beschleunigung demonstriert werden sollen, ist keine Lenkung und somit nur ein einzelner Motor notwendig. Weiters ist neben dem zentralen NXT kein Einsatz von Sensoren nötig, wenngleich diese natürlich immer verwendet werden können, etwa um Kollisionen mit Wänden zu verhindern oder einen Abbruch des Programms durch Knopfdruck zu erreichen.

Die Programmierung eines Roboters, welcher eine von der Lehrkraft etwa als $v = \frac{\text{cm}}{\text{sec}}$ vorgegebene Geschwindigkeit fährt, ist recht leicht umzusetzen. Dabei kann auch leicht begrenzt werden, dass der Roboter z.B. nur zehn Meter weit oder für eine Dauer von 20 Sekunden fahren soll. Am einfachsten ist es hier, mit einer while-Schleife zu arbeiten. Diese wird bis zur jeweiligen Abbruchbedingung ausgeführt, also etwa dem Drücken des Escape-Knopfes oder dem Erreichen einer bestimmten Motorumdrehungszahl, welche mittels „getTachoCount()“ ausgelesen werden kann.

Da hier von der Lehrkraft aber sinnvoller Weise die Geschwindigkeit nicht in $\frac{\text{Umdrehungsgrad}}{\text{sec}}$ sondern eben in $\frac{\text{cm}}{\text{sec}}$ angegeben wird, muss noch eine Umrechnung durch die Schüler stattfinden. Dazu ist es nötig, Radius, Durchmesser oder Umfang der für den Roboter verwendeten Räder zu kennen. Bei Lego-Rädern sind Durchmesser und Breite immer in mm-Werten angegeben. Mittels $U = d * \pi$ berechnet sich der Umfang, klarerweise auch als mm-Wert. Nun muss noch berechnet werden, wie viele Umdrehungen benötigt werden, um die vorgegebene Geschwindigkeit zu erreichen. Dafür gilt: $\frac{\text{Geschwindigkeit}[\frac{\text{cm}}{\text{sec}}]}{\text{Umfang}[\frac{\text{cm}}{\text{Umdrehung}}]} = \text{Umdrehungen}[\frac{\text{Umdrehungen}}{\text{sec}}]$.

Mittels dieser Berechnung lässt sich jene Umdrehungszahl in Winkelgrad berechnen, welche für die Methode „setSpeed()“ des Motors benötigt wird. Diese Geschwindigkeit wird am Beginn des Programms festgelegt und braucht nicht mehr geändert zu werden. Sollen etwa genau 3 Meter gefahren werden, müssen

die dafür nötigen Umdrehungsgrad berechnet werden und diese z.B. entweder als Abbruchkriterium einer Schleife festgelegt werden, also etwa

```
while (Motor.A.getTachoCount() <= 6139)
```

oder es wird dem Motor direkt mittels „rotateTo()“ die nötige Umdrehungszahl mitgeteilt. Soll die Fahrt genau fünf Sekunden dauern, so kann dazu die Klasse „stopwatch“ mit ihren Methoden „elapsed()“ und „reset()“ verwendet werden. Dazu muss als erstes etwa mittels „Stopwatch stoppuhr = new Stopwatch()“ eine Instanz der Stopwatch-Klasse erstellt werden. Anschließend sollte die Stoppuhr mittels „stoppuhr.reset()“ zurückgesetzt werden, um sie anschließend verwenden zu können. Eine Möglichkeit ist auch hier, diese als Abbruchkriterium einer while-Schleife heranzuziehen, also etwa „while(stoppuhr.elapsed()<=5000)“. Zu beachten ist hier, dass die retournierten Werte die Einheit Millisekunden tragen.

Durch die so mögliche Zeitmessung kann auch jeweils durch den Roboter selbst eine Berechnung der Geschwindigkeit stattfinden, da sowohl benötigte Zeit als auch die Anzahl der Umdrehungen („getTachoCount()“) bekannt sind. Diese kann laufend oder zu einem bestimmten Ereignis (also etwa die Durchschnittsgeschwindigkeit bis zum Drücken eines Buttons) etwa mittels der Methode „LCD.drawInt()“ am Display ausgegeben werden.

Programmcode 4.5 zeigt eine Möglichkeit, einen Roboter für eine konstante Geschwindigkeit zu programmieren.

Programmcode 4.5: Geschwindigkeit & Beschleunigung - Konstante Geschwindigkeit

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.Motor;

public class geschwindigkeit {

    public static void main (String[] aArg) throws Exception {

        /*Festlegung der zu fahrenden Geschwindigkeit in cm/s & des
           Raddurchmessers (cm). Anschließend Berechnung des Radumfanges
           und der Umdrehungsgrad pro Sekunde anhand der bisherigen Daten.
           Anschließend Umwandlung in Integer-Zahl, da nur diese dem
           Motor übergeben werden kann.*/
        float speedCMS = 20.0f;
        float durchmesser = 5.6f;
        final float PI = 3.141593f;
        float umfang = durchmesser * PI;
        float umdrehungsgradProSec = (speedCMS/umfang)*360;
        int motorUmdrehungen = (int) umdrehungsgradProSec;

        /*Festlegung der zu fahrenden Distanz in cm, anschließend
           Umrechnung dieser in entsprechende Gradzahl und wieder
           Umwandlung in Integer-Zahl*/
```

```

int distanz = 200;
float gradzahl = (distanz*360/umfang);
int grad = (int)gradzahl;

/*Wenn gewünschte Geschwindigkeit nicht erreicht werden kann, dann
  Fehlermeldung ausgeben*/
if(motorUmdrehungen > 900) {
    while (!Button.ESCAPE.isPressed()) {
        LCD.clear();
        LCD.drawString("Umdrehungszahl", 0,0);
        LCD.drawString("zu hoch", 0,1);
        LCD.refresh();
    }
}

/*Sonst die Geschwindigkeit auf den berechneten Wert festlegen und
  die berechnete Distanz zurücklegen. Anschließend am Display
  die Gradzahl der Umdrehungen ausgeben.*/
else {
    Motor.A.setSpeed(motorUmdrehungen);
    Motor.A.rotateTo(grad);
    Motor.A.stop();
    while (!Button.ESCAPE.isPressed()) {
        LCD.clear();
        LCD.drawString("Gradzahl der", 0,0);
        LCD.drawString("Umdrehungen:", 0,1);
        LCD.drawInt(grad, 0,2);
        LCD.refresh();
    }
}
}
}

```

Neben der Geschwindigkeit kann auch die Beschleunigung gemessen und programmiert werden. Zum bloßen Messen der (negativen) Beschleunigung eignet sich die Methode „flt()“, mittels derer die Kraftzufuhr zum Motor abgestellt wird, im Gegensatz zur Methode „stop()“ wird der Motor allerdings nicht umgehend gestoppt, sondern er läuft so lange weiter, bis er von selbst zum Stillstand kommt. Hier können Messbedingungen gut angepasst werden, etwa durch eine Veränderung der Steigung, auf welcher die (negative) Beschleunigung stattfindet. Hier gibt es natürlich wieder zahlreiche Möglichkeiten, wie dies umgesetzt werden soll, also v.a. wann der Roboter die Methode „flt()“ ausführen soll. Sinnvoll erscheint hier der Einsatz des Helligkeitssensors, damit der Roboter etwa beim Überqueren einer Linie den gewünschten Vorgang startet. So kann dabei etwa das Auslaufen des Motors und eine Stoppuhr gestartet werden. Die Stoppuhr soll genau so lange laufen, so lange sich der Motor bewegt. Dies ist etwa mittels einer while-Schleife möglich. Da klarerweise auch die Geschwindigkeit vor dem Start von „flt()“ bekannt sein muss, lässt sich nun die mittlere Beschleunigung mittels $\vec{a} = \frac{\Delta \vec{v}}{\Delta t}$ berechnen. Möglich wäre hier zum einen eine laufende Ausga-

be der aktuellen Beschleunigung und/oder eine Ausgabe der durchschnittlichen Beschleunigung nach Abschluss (hier etwa bei Stillstand).

Neben der Messung einer vorgegebenen, selbst nur durch Änderung äußerer Einflüsse manipulierbaren Beschleunigung, ist es auch möglich, eigene Beschleunigungsvorgänge selbst zu programmieren. D.h. die Lehrkraft kann neben der Vorgabe einer bestimmten Geschwindigkeit (oder dem Zurücklegen einer bestimmten Strecke in gegebener Zeit) auch eine bestimmte Beschleunigung fordern. Wie oben bereits erwähnt, entspricht die mittlere Beschleunigung $\vec{a} = \frac{\Delta \vec{v}}{\Delta t}$. Die zum Ansteuern des Motors nötige Größe ist hier $\Delta \vec{v} = \vec{a} * \Delta t$. Da $\vec{a}$ angegeben ist und die Zeit mittels der durch die „Stopwatch“-Klasse zur Verfügung stehenden Methode „elapsed()“ gemessen werden kann, lässt sich der jeweils nötige Geschwindigkeitsunterschied ($\Delta \vec{v}$) berechnen.

Ist $\vec{a}$ in $\frac{\text{cm}}{\text{s}^2}$ angegeben, so kann diese Beschleunigung mittels zweier while-Schleifen ausgeführt werden. Dabei kann mittels der äußeren Schleife überprüft werden, wann eine gewisse Zeit vergangen ist oder eine gewisse Distanz zurückgelegt wurde. Mittels der „elapsed()“-Methode der stopwatch-Klasse kann die innere Schleife jeweils genau eine Sekunde lang ausgeführt werden. Bei jedem erneuten Durchlauf der inneren Schleife wird die Geschwindigkeit um $\Delta \vec{v} = \vec{a} * \Delta t$ erhöht. Die Beschleunigung hier geschieht stufenweise, also durch eine Erhöhung der Geschwindigkeit im Abstand von z.B. einer Sekunde. Um diese Stufen möglichst klein zu halten, kann die Ausführungsdauer der inneren Schleife auch reduziert werden, wofür aber auch die Angabe von $\vec{a}$ in passender Einheit nötig ist.

4.2.2 Zwei Roboter für simulierten Überholvorgang

Wohl jeder kennt folgende Art der Rechenaufgabe aus dem Physikunterricht: Ein PKW fährt mit 80 km/h auf der rechten Spur. Diesen soll ein zweiter PKW mit einer Geschwindigkeit von 100 km/h überholen. Beide PKW sind jeweils fünf Meter lang und der überholende PKW soll 20 Meter vor und nach dem überholten PKW aus- bzw. einscheren. Wie viele Sekunden bzw. Meter dauert der Überholvorgang?

Dieses Projekt ist dem ersten Jahrgang zuzuordnen, da hier Geschwindigkeit und Beschleunigung behandelt werden sollen. Für dieses Projekt werden mindestens zwei Roboter benötigt: einer stellt das zu überholende Fahrzeug dar, der zweite das überholende. Es wäre auch der Einsatz eines dritten Roboters denkbar, welcher den Gegenverkehr simulieren könnte.

An den zu überholenden Roboter wird nur jener Anspruch gestellt, dass er sowohl mit konstanter Geschwindigkeit, als auch geradeaus fahren muss. Dazu ist also keinerlei Lenkung und kein Verbau eines Sensors nötig. Es bietet sich hier an,

einen einzelnen Motor zum Antrieb zu verwenden, um den Roboter nicht durch Motorungenauigkeiten „schief“ fahren zu lassen. Sollte Gegenverkehr ebenfalls simuliert werden, so könnte dies mittels des gleichen Robotertyps geschehen.

Der überholende Roboter muss neben der Fahrt mit konstanter Geschwindigkeit auch das Fahren von Kurven beherrschen, um am Beginn und Ende des Überholvorganges aus- und einscheren zu können. Um diese Lenkung realisieren zu können, sind mehrere Möglichkeiten denkbar. So wäre etwa wie beim zu überholenden Roboter der Einsatz nur eines Motors zum Antrieb möglich, ein zweiter Motor könnte dann den Roboter mittels Stützrad oder einer kompletten Achse in die gewünschte Richtung lenken. Hier fällt die Wahl auf Antrieb und Steuerung durch zwei Motoren (z.B. Antrieb des linken und rechten Hinterrads eines Dreirades), welche mittels einer pilot-Klasse gesteuert werden können. Um den korrekten Abstand zumindest für das Ausscheren erkennen zu können, ist der Ultraschallsensor nötig.

Die Aktivitätsdiagramme für „zu überholender Roboter“, „Gegenverkehr“ (Abb. 4.7a) und „überholender Roboter“ (Abb. 4.7b) sind im Folgenden dargestellt, wobei hier für „Gegenverkehr“ und „zu überholender Roboter“ das exakt gleiche Aktivitätsdiagramm gewählt wurde.

Die Programmierung von „zu überholender Roboter“ wie „Gegenverkehr“ dürfte keinerlei Schwierigkeiten bereiten. Es kann entweder direkt im Code eine fixe Geschwindigkeit einprogrammiert werden, oder diese wird, wie im Aktivitätsdiagramm dargestellt, vom Benutzer erfragt. Dabei ist es natürlich sinnvoll, den Benutzer nicht nach der Rotationsgeschwindigkeit in $\frac{\text{Umdrehungsgrad}}{\text{Sekunde}}$ zu fragen, sondern nach der zu fahrenden Geschwindigkeit in $\frac{\text{Centimeter}}{\text{Sekunde}}$. Dem Motor muss zwar die Gradzahl der Umdrehungen pro Sekunde mitgeteilt werden, da aber die Dimension der Räder bekannt ist, ist diese Berechnung keine Schwierigkeit. Es gilt: $\frac{\frac{\text{Umdrehungsgrad}}{\text{Sekunde}}}{360^\circ} * \text{Umfang[cm]} = \text{Geschwindigkeit}[\frac{\text{cm}}{\text{sec}}]$. Die so zu berechnende Winkelgeschwindigkeit wird mittels „setSpeed()“ für den verbauten Motor festgelegt. Eine Möglichkeit ist nun jene ebenfalls im Aktivitätsdiagramm dargestellte, den Motor so lange mit der festgelegten Umdrehungszahl rotieren zu lassen, bis die Escape-Taste am NXT gedrückt wird. Natürlich gäbe es auch hier noch zahlreiche andere mögliche Abbruchkriterien, wie etwa eine bestimmte zurückgelegte Distanz oder eine Kollision, etwa mit einer Wand oder dem überholenden Fahrzeug (erfragt mittels Druckkontaktsensor), diese werden hier allerdings außer Acht gelassen, auch deshalb, da sie leicht zu implementieren sind und keine Auswirkungen auf das Projekt haben.

Die Programmierung des überholenden Fahrzeuges gestaltet sich deutlich anspruchsvoller. Die Berechnung der Geschwindigkeit erfolgt hier auf gleiche Weise

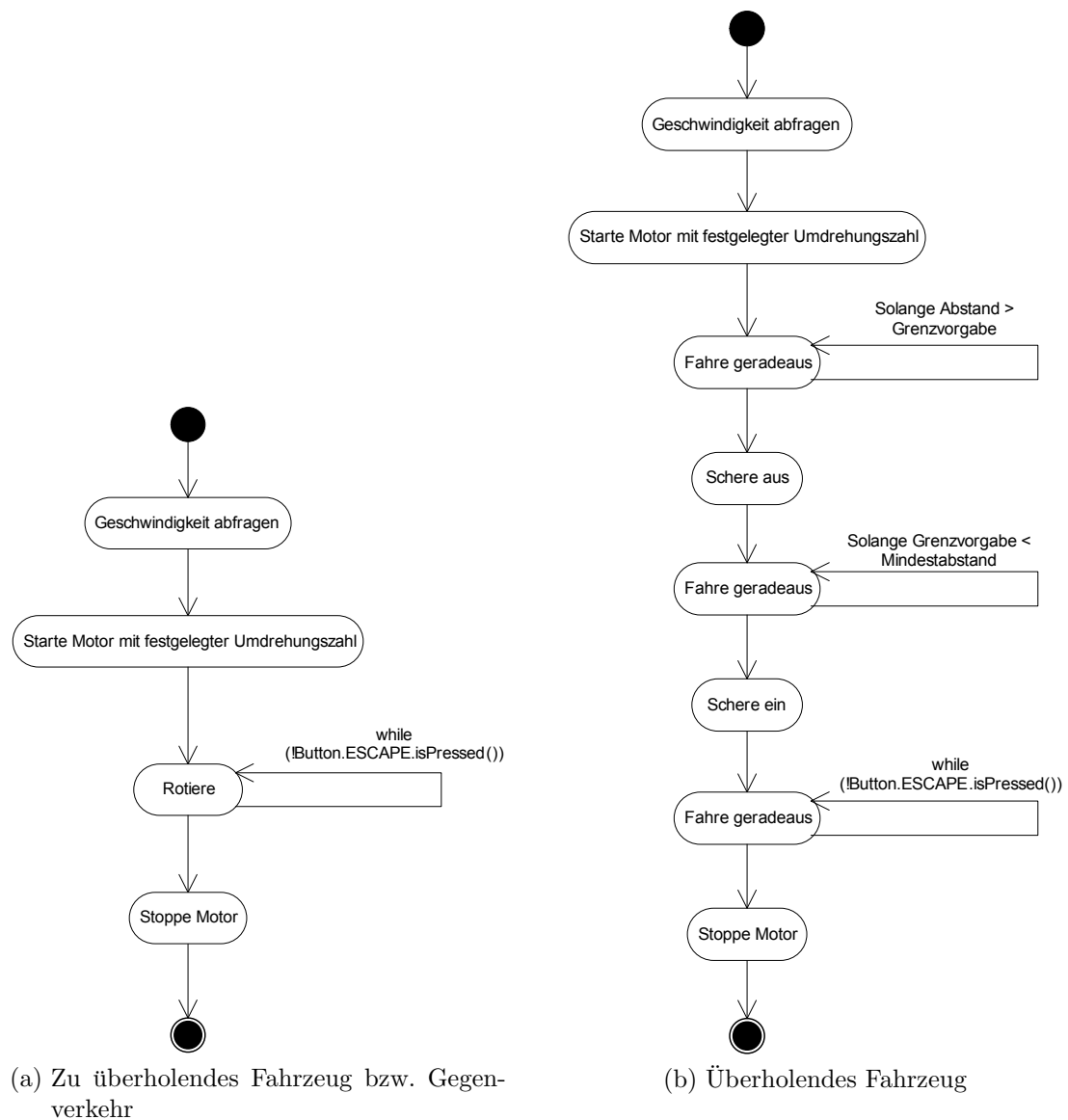


Abbildung 4.7: Überholvorgang - Aktivitätsdiagramme

wie bei den beiden anderen Fahrzeugen. Die beiden eingesetzten Motoren werden in einer pilot-Klasse zusammengefasst, um möglichst große Genauigkeit und möglichst einfache Steuerung zu erreichen.

Um herauszufinden, wann das Fahrzeug zum Überholen ausscheren soll, muss mittels des Ultraschallsensors laufend der Abstand überprüft werden. Der kritische Abstand, bei welchem der Roboter ausscheren soll, kann entweder fix einprogrammiert werden, also etwa 50cm, oder er wird in Abhängigkeit der gewählten Geschwindigkeit berechnet. Dabei reicht es zur Berechnung allerdings nicht aus, nur die Geschwindigkeit des überholenden Fahrzeugs zu kennen, es muss auch jene des zu überholenden Fahrzeugs bekannt sein. D.h. diese sollte bei einer solchen Programmierung auch entweder fix einprogrammiert oder eben auch vom

Benutzer erfragt werden. Ebenso muss die Länge der beiden Fahrzeuge bekannt sein, um zu wissen, nach welcher Fahrzeit das zu überholende Fahrzeug passiert wurde. Es wäre auch möglich, mittels eines drehbar gelagerten Ultraschallsensors diesen direkt bei Ausscheren nach rechts, also in Richtung des zu überholenden Fahrzeuges zu drehen. Damit wäre es dann möglich, das Passieren des vorderen Endes des Fahrzeuges zu erkennen (sobald der registrierte Abstand wieder zunimmt) und somit könnte das überholende Fahrzeug selbständig entscheiden, wann der Überholvorgang beendet ist, d.h. wann eingeschert werden kann.

Zu Beginn wird auch für das überholende Fahrzeug eine Geschwindigkeit abgefragt, mit welcher dieses fahren soll. Etwa mittels einer while-Schleife soll nun das Fahrzeug also solange geradeaus fahren, bis der mittels der Methode „getDistance()“ abgefragte Abstand zu dem vorausfahrenden Fahrzeug den kritischen Wert erreicht. Dieser kritische Wert bietet viel Spielraum, d.h. es kann ausprobiert werden, wie sich der nötige Abstand verändert, wenn etwa die Geschwindigkeiten beider Fahrzeuge um 10% erhöht werden etc. Die Fahrt geradeaus kann auf diverse Arten realisiert werden, z.B. mittels „forward()“ oder „rotate(int angle)“. Wird etwa forward() verwendet, müsste nach Verlassen der while-Schleife „stop()“ aufgerufen werden, um die Vorwärtsbewegung zu stoppen, bei der Bewegung mittels „rotate(int angle)“ würden die Motoren pro Schleifendurchlauf immer nur um den angegebenen Winkelbetrag gedreht werden, d.h. es wäre nach Verlassen der Schleife kein Stoppen der Motoren nötig.

Sobald der Frontalabstand den kritischen Wert erreicht hat, muss der Überholvorgang gestartet werden. Als erstes muss der überholende Roboter ausscheren, es folgt eine Geradeausfahrt im „Gegenverkehr“ und danach das Einscheren. Anders als in typischen Rechenaufgaben aus dem Physikunterricht ist hier nun allerdings auch die Breite des zu überholenden Fahrzeuges eine wichtige Größe, denn das überholende Fahrzeug muss beim Aus- und Einscheren die komplette Breite des zu überholenden Fahrzeuges überwinden, davon ausgehend, dass sich der rechte Rand beider Fahrzeuge auf einer Linie befindet. Für dieses Ausscheren kann die steer-Methode der pilot-Klasse verwendet werden. Von dieser stehen mehrere Varianten zur Verfügung, d.h. es ist abhängig von Art und Anzahl der übergebenen Variablen, welche steer-Methode ausgeführt wird. Die steer-Methode ist also „überladen“.

Zum Ausscheren ist die Verwendung von „steer(int turnRate, int angle)“ ratsam. „turnRate“ gibt das Verhältnis der Umdrehungszahlen des inneren Rades zum äußeren an. Ein positiver Wert bedeutet, dass sich das linke Rad innen befindet. „angle“ gibt den Winkel an, bis zu welchem die Kurve gefahren werden soll. Für das Aus- und Einscheren ist jeweils eine aus zwei steer-Kurven zusammen-

gesetzte S-Kurve ratsam. Bis zu welchem Winkel dabei rotiert werden soll kann entweder berechnet werden (dies ist durch die jeweiligen Angaben der pilot-Klasse möglich), oder durch Versuche ausprobiert werden.

Das bedeutet, sobald der kritische Abstand zum vorausfahrenden Fahrzeug gemessen wird, wird die S-Kurve gestartet. Dies geschieht durch Fahren einer Links- und anschließendes Fahren einer Rechtskurve. Der Code dazu kann wie folgt aussehen:

```
pilot.steer(25, 30); //30°-Linkskurve  
pilot.steer(-25, 30); //30°-Rechtskurve
```

Nach Abschluss dieses Ausscherens muss das überholende Fahrzeug nun entweder einen zweiten Ultraschallsensor besitzen oder den vorhandenen zum überholten Fahrzeug drehen und mittels „getDistance()“ abfragen, wann das Fahrzeug passiert wurde. Denn mindestens so lange muss das überholende Fahrzeug gesichert bleiben. Sobald der Sensor wieder einen größeren Wert misst kann entweder sofort mit dem Einscheren begonnen werden, oder es wird, abhängig von Geschwindigkeit und Lage des Ultraschallsensors, noch eine bestimmte Zeit t im „Gegenverkehr“ gefahren.

Das Einscheren geschieht auf selbe Weise wie das Ausscheren, nur in umgekehrter Reihenfolge, d.h. es folgt erst die Rechts- und dann die Linkskurve. Anschließend kann weiter geradeaus gefahren oder sofort gestoppt werden.

4.2.3 Fahrzeug mit integrierter Schaltung

Dieses Projekt ist dem Bereich „Kinematik und Dynamik“ des ersten Jahrgangs der Physik zuzuordnen. Hierbei soll ein Fahrzeug gebaut werden, welches Steigungen zumindest in einem gewissen Rahmen erkennt und jeweils in einen passenden Gang schaltet.

Erste Idee war hier ein Fahrzeug, welches abhängig von der tatsächlich gefahrenen Geschwindigkeit einen passenden Gang wählt. Dies ist allerdings aufgrund der Servomotoren und der vorgegebenen Umdrehungszahl nicht möglich. Daher bietet sich eine Anpassung der Übersetzung des Getriebes an die jeweilige Neigung der Strecke an. Neben der Möglichkeit des Einsatzes fertig zu kaufender Neigungssensoren bietet sich auch die Möglichkeit, diese durch die Schüler selbst konzipieren zu lassen.

Hier wird nur eine mögliche Verwirklichung eines solchen Neigungssensor angeführt. Diese bietet den Vorteil, dass sie mit den Sensoren eines einzelnen Mindstorms-Bausatzes auskommt. Dazu wird ein freidrehbarer Arm benötigt, welcher sich bei einer Steigung dem einen Sensor, bei einem Gefälle dem anderen Sensor

nähert. Es könnte hier auch alleine mit dem Ultraschallsensor gearbeitet werden, mit welchem dann auch nahezu beliebig viele Gänge verwirklicht werden könnten. Nachteil des Ultraschallsensors ist allerdings, dass dieser in kurzen Distanzen, welche hier vorliegen, eine relativ hohe Messungenauigkeit aufweist [Geba07]. Abbildung 4.8 zeigt einen solchen möglichen Aufbau eines Neigungssensors.

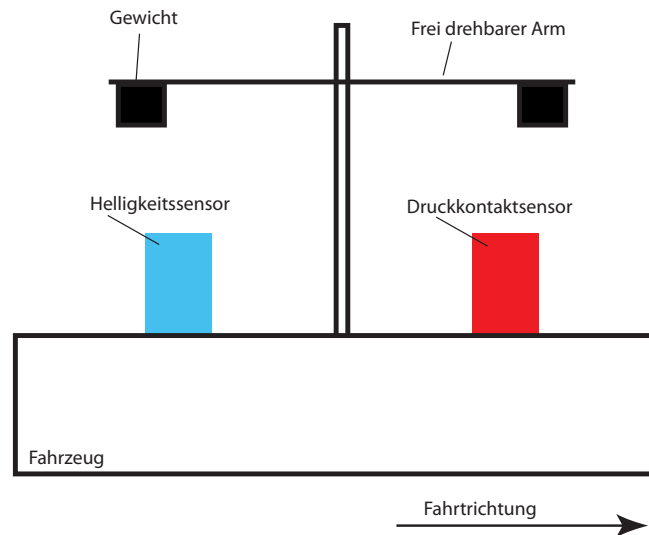


Abbildung 4.8: Neigungssensor - Aufbau

Direkt am Fahrzeug befindet sich dabei ein „Mast“, an dem ein frei drehbarer Arm montiert ist. Dies bedeutet, dass dieser immer waagrecht ausgerichtet bleibt, auch wenn das Fahrzeug selbst nach vorne oder hinten geneigt ist. Um dies auch sicherzustellen, sollten an den äußeren Enden des Armes Gewichte angebracht werden.

Fährt nun das Fahrzeug auf eine Steigung, so nähert sich das auf dieser Abbildung rechte Gewicht dem Drucksensor. Um diesen auch tatsächlich bei der ersten Berührung auszulösen, muss der Drucksensor vorher schon mit Gewichten besetzt werden, da die Berührung allein nicht die zur Auslösung nötige Kraft von 0,34 Newton [Geba07] aufbringen wird.

Fährt das Fahrzeug auf ein Gefälle, so nähert sich das auf dieser Abbildung linke Gewicht dem Helligkeitssensor. Mit diesem Helligkeitssensor wird dann der Helligkeitswert des Gewichts erkannt und genau dann, wenn dieser erkannt wird, kann davon ausgegangen werden, dass sich das Fahrzeug in einem Gefälle befindet.

Auf diese Weise kann ein simpler Neigungssensor integriert werden. D.h., wenn der Berührungssensor ausgelöst wird, so liegt eine Steigung vor und wenn vom Helligkeitssensor ein vordefinierter Farbwert erkannt wird, so liegt ein Gefälle vor. Liegt weder der eine noch der andere Zustand vor, so befindet sich der Roboter in der Waagrechten.

Der nächste Punkt betrifft die Umsetzung der Schaltung. Zu Beginn war hier eine Umsetzung wie in Bausatz 8880 (Supercar) von Lego Technic geplant, in welchem ein Fahrzeug mit manueller Schaltung realisiert wurde. Die Entscheidung wurde dann jedoch gegen eine solche Umsetzung getroffen, da sowohl diese Teile den meisten Schülern fehlen dürften, als auch die Bedienung des Schalthebels nur durch zwei Motoren möglich ist. Der Aufwand dafür erscheint deutlich zu hoch.

Daher fiel die Entscheidung auf eine Umsetzung, welche auf dem direkten Verschieben von Zahnrädern basiert, ähnlich wie in [Kali08]. Dabei sind Motor und Rad nicht direkt miteinander verbunden, sondern die Verbindung führt über Zahnräder. Durch einen Austausch von bestimmten Zahnrädern findet eine Änderung der Übersetzung statt. So sollte etwa die Verbindung für den für das Gefälle zu wählenden Gang über kleine Zahnräder führen, welche wenige Zähne haben. Die Räder selbst können also mehr Umdrehungen zurücklegen als der Motor, d.h. es liegt eine Untersetzung vor. Für die Steigung sollte die Verbindung über möglichst große Zahnräder führen, wodurch der Motor mehr Umdrehungen ausführt als die Räder. Es kommt zu einer Übersetzung und zu einer Freisetzung von mehr Kraft. In der Waagrechten sollte eine Übersetzung zwischen den beiden Extrema gewählt werden, möglicherweise auch schlicht eine 1:1-Übersetzung.

Wie kommt es aber nun zu einem Wechsel des Ganges? Dazu ist ein Motor nötig, welcher auf Befehl, eben wenn eine entsprechende Neigung erreicht ist, ein oder mehrere Zahnräder in Position schiebt. Das bedeutet konkret, dass diese quasi vertauscht werden. Abbildung 4.9 soll diese Funktionsweise verdeutlichen.

Hier ist zu erkennen, dass immer nur ein Zahnrad auf der Achse der Räder angetrieben wird. Durch die unterschiedliche Größe dieser kommt es zu unterschiedlichen Übersetzungen. Würden sich mehrere verschieden große Zahnräder gleichzeitig berühren und dabei drehen, käme es zu einer Blockade. Der in Abbildung 4.9 eingelegte Gang hat bei der hier gewählten Konstruktion ein Verhältnis von Antriebs- zu Abtriebsdrehzahl von 1:9. D.h. pro Umdrehung des Motors finden neun Radumdrehungen statt. Es könnten also etwa statt der durch den Servomotor limitierten 2,5 Umdrehungen pro Sekunde nun $9 * 2,5 = 22,5$ Umdrehungen pro Sekunde erreicht werden. Bei der hier dargestellten Konstruktion ist zu beachten, dass sich die Räder genau in die entgegengesetzte Richtung des Motors drehen.

Der mittlere Gang wird durch ein „Herausziehen“ des Schaltarmes um eine Position erreicht. Das Übersetzungsverhältnis entspricht hier genau 5:1, d.h. der Antriebsmotor führt in der gleichen Zeit fünf Umdrehungen aus, in welcher die Räder nur eine ausführen.

Wird der Schaltarm um eine weitere Position herausgezogen, so wird der letz-

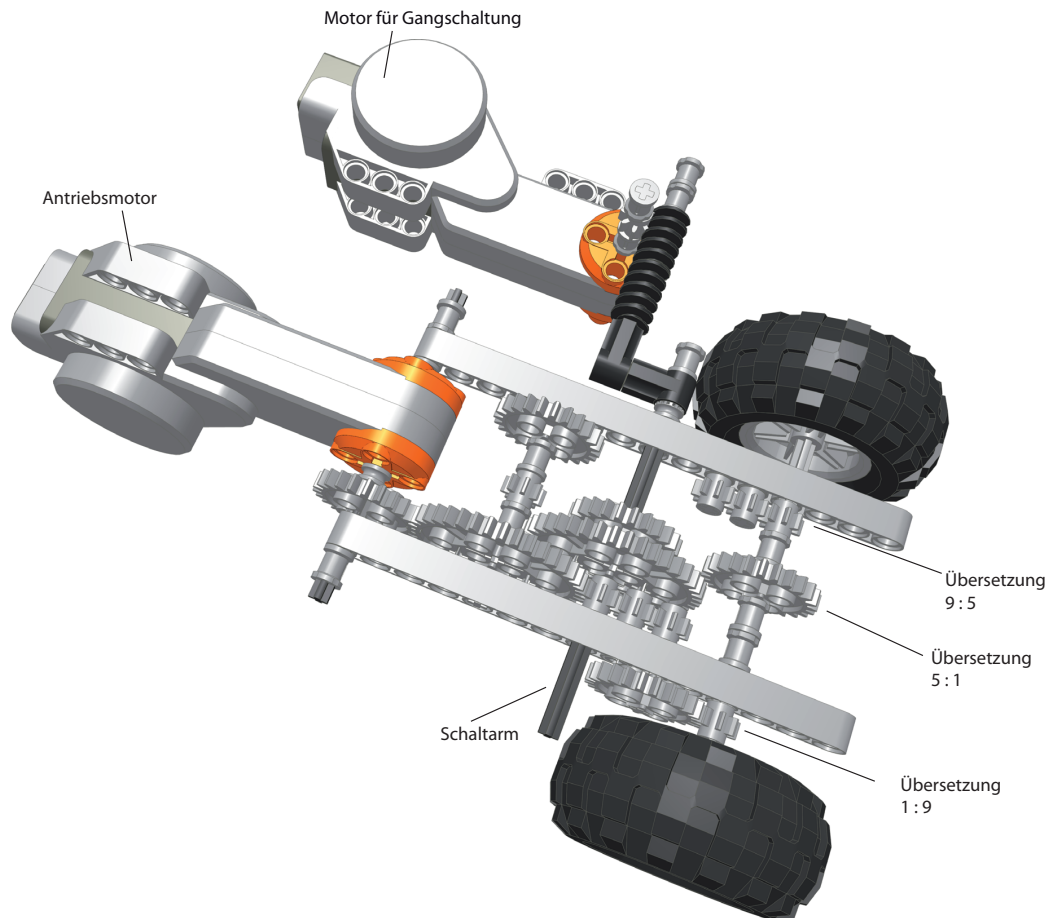


Abbildung 4.9: Getriebe - Aufbau

te verfügbare Gang erreicht, welcher bei der hier vorliegenden Umsetzung eine Übersetzung von 9:5 aufweist. Bei neun Motorumdrehungen kommt es also zu fünf Radumdrehungen.

Dem für das Wechseln der Gänge zuständigen Motor muss dabei mitgeteilt werden, wie weit und in welche Richtung er sich zu drehen hat, da er selbst nicht „weiß“, in welchem Gang das Fahrzeug aktuell fährt. Bei drei Gängen gibt es $(3 - 1) * 3 = 6$ verschiedene Gangwechsel, wobei zwischen $1 \rightarrow 3$ und $3 \rightarrow 1$ unterschieden wird, da die Bewegungsrichtung hier jeweils entgegengesetzt ist. Die einfachste Variante ist hier also, für all diese sechs Möglichkeiten eigene Methoden zu schreiben. Sinnvoller ist die Integration einer Methode, mittels welcher alle Gänge gewechselt werden können. Leicht möglich ist dies dann, wenn die Abstände zwischen den Zahnrädern auf den jeweiligen Achsen immer gleich groß sind.

Wichtig an der hier gewählten Umsetzung ist, dass der Schaltarm frei drehbar ist und dass nicht durch den antreibenden Motor auch der daran angehängte Arm mit der daran befestigten Spirale gedreht wird. Dies würde bei einer fixen Montage ebenfalls zu einer Blockade führen.

Die zentrale Einheit, der NXT, „kennt“ immer den aktuellen Gang und sobald der Bewegungssensor eine Änderung der Lage meldet, kann darauf durch einen entsprechenden Wechsel des Ganges reagiert werden. Für solch einen Wechsel sollte allerdings die Fahrt kurzzeitig unterbrochen werden. Eine Methode „change(3, 1)“ könnte etwa von dem dritten in den ersten Gang wechseln, die dazu für den zuständigen Motor nötige Umdrehungszahl kann entweder durch vorheriges Probieren oder durch Berechnungen herausgefunden werden. Eine Methode für den Gangwechsel könnte dabei wie in Programmcode 4.6 aussehen.

Programmcode 4.6: Getriebe - Gangwechsel

```
public static void change(int from, int to) throws
    InterruptedException {
    //Antriebsmotor stoppen
    Motor.A.stop();
    //200ms warten (Zeitpolster)
    Thread.sleep(200);
    //Distanz zwischen Gängen berechnen
    int distance = from - to;
    //Geschwindigkeit für Gangwechsel-Motor
    Motor.B.setSpeed(180);
    //distance*180° rotieren (muss ggf angepasst werden)
    Motor.B.rotateTo(distance*180);
    //Mittels der Math-Klasse den Absolutbetrag von distance erruieren
    int entfernung = Math.abs(distance);
    /*Motor B benötigt bei der hier angeführten Konfiguration für den
       Wechsel zum benachbarten Gang genau eine Sekunde. Daher
       Entfernung * 1000ms. 200ms werden auch hier als Puffer addiert.
       */
    Thread.sleep(entfernung*1000+200);
    //Nach Abschluss Fahrt fortsetzen
    Motor.A.forward();
}
```

Dabei wird mit der Variablen „distance“ berechnet, wie groß die Entfernung der beiden Gänge ist. Diese kann sowohl positiv (z.B. 3 → 1) als auch negativ sein (z.B. 2 → 3). Die Umdrehungsgeschwindigkeit des für das Schalten zuständigen Motors wird auf 180° pro Sekunde festgelegt, wobei hier auch jeder andere Wert gewählt werden kann. Mittels „rotateTo“ dreht dieser Motor dann um genau distance * 180°. 180° deshalb, da das hier am Schaltarm greifende Zahnrad genau eine halbe Umdrehung benötigt, um einen Gang zu wechseln.

Für die Umsetzung der Schaltung muss dem Fahrzeug zu Beginn bekannt sein, welcher Gang derzeit eingelegt ist, da sonst ja nicht klar sein kann, ob in einer bestimmten Lage ein Schaltvorgang durchgeführt werden muss oder ob dieser Gang schon eingelegt ist. Eine mögliche Umsetzung von Lageabfrage und der daraus resultierenden Wahl des passenden Ganges kann wie in Programmcode 4.7 aussehen.

Programmcode 4.7: Neigungssensor - Lageabfrage & Einleitung der Schaltung

```
//Instanzen von Druckkontakt- und Helligkeitssensor erstellen
static TouchSensor touch = new TouchSensor(SensorPort.S1);
static LightSensor light = new LightSensor(SensorPort.S2);
//Mittels gangAlt den aktuell eingelegten Gang übergeben
public static void gang(int gangAlt) throws InterruptedException {
    //gangNeu initialisieren
    int gangNeu = 0;
    while(!Button.ESCAPE.isPressed()) {
        //Wenn Druckkontakt ausgelöst, so ist Gang 3 zu wählen
        if (touch.isPressed() == true)
            gangNeu = 3;
        /*Wenn bestimmter Helligkeitsswert erkannt wird, ist Gang 1 zu
           wählen*/
        if (light.readNormalizedValue() < 300)
            gangNeu = 1;
        /*Weder Druckkontakt- noch Lichtsensor liefern definierten Wert, d
           .h. Arm schwingt in der Mitte und es ist Gang 2 zu wählen*/
        if (touch.isPressed() == false && light.readNormalizedValue() >=
            300)
            gangNeu = 2;
        /*Wenn aktuell eingelegter Gang und der nun bestimmte Gang
           verschieden sind, so ist der Schaltvorgang einzuleiten*/
        if (gangNeu != gangAlt)
            change(gangAlt, gang);
    }
}
```

Weiters könnte es sinnvoll sein, einen Gangwechsel erst dann vorzunehmen, wenn der Lagesensor für eine bestimmte Zeit die gleiche Lage meldet. D.h. es könnte etwa erst eine Sekunde nach Änderung der Lage auch tatsächlich der Gangwechsel vollzogen werden. Findet während dieser Zeit erneut eine Lageänderung statt, so muss dieser Zähler zurückgesetzt werden. Der Sinn dieser Maßnahme wird etwa dann deutlich, wenn man sich ein Fahrzeug auf einer Parabelkurve vorstellt. Im Gefälle wird der schnellste Gang gewählt. Im Minimum zeigt der Lagesensor eine Waagrechte an, wodurch der mittlere Gang gewählt wird. Allerdings folgt direkt darauf eine Steigung, d.h. es ist auch in Kürze der nächste Schaltvorgang nötig.

Sinnvoll kann es bei der hier beschriebenen Integration des Lagesensors auch sein, die Beleuchtung des Helligkeitssensors zumindest so lange zu deaktivieren, so lange der Drucksensor einen Kontakt meldet, um auf diese Weise Strom zu sparen. Möglich ist dies mittels folgender Anweisungen.

```
if(touch.isPressed == true) {
    floodlight.setFloodlight(false);
}
if(touch.isPressed == false) {
    floodlight.setFloodlight(true);
}
```

Außerdem sollten die Gewichte des Lagesensors eine für den Helligkeitssensor klar erkennbare Helligkeit aufweisen.

4.2.4 Startrampe für Bälle oder Papierflugzeuge

Dieses Projekt ist den Bereichen „Kinematik und Dynamik“ sowie „Aeromechanik und Hydromechanik“ des 1. Jahrgangs zuzuordnen. Dabei soll anhand einer NXT-gesteuerten Startrampe für Bälle und/oder Papierflugzeuge gezeigt werden, welche Abhängigkeit zwischen Abwurfwinkel und -geschwindigkeit und der resultierenden Wurf- bzw. Flugweite besteht. Priorität sollte hier auf einer Vorführung im Unterricht durch die Lehrkraft liegen, ein Nachbau und Programmieren durch die Lernenden kann aber etwa durch einen Wettbewerb gefördert werden, z.B. bei welcher Gruppe Ball und Flugzeug in Summe am weitesten fliegen.

Für diese Startrampe wird neben der zentralen Einheit NXT mindestens ein Motor zur Beschleunigung der jeweiligen Gegenstände benötigt. Ein zweiter Motor kann verwendet werden, um eine automatische Einstellung des Abwurfwinkels vornehmen zu können. Sensoren werden hier keine benötigt. Abbildung 4.10 zeigt skizzenhaft, wie solch ein Aufbau aussehen könnte.

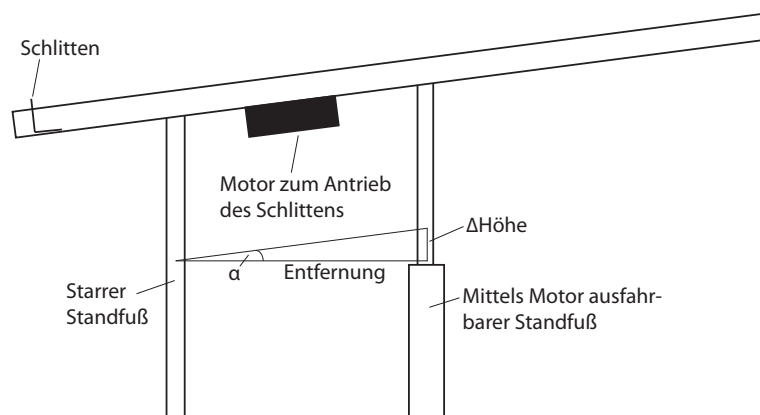


Abbildung 4.10: Startrampe - Aufbau

Der Schlitten, mittels welchem die Objekte beschleunigt werden, kann z.B. mittels einer Kette mit Zahnrädern und somit dem Motor verbunden werden. Wichtig ist hier ja, dass der Motor am selben Ort bleibt, das anzutreibende Objekt hingegen (der Schlitten) den Ort wechselt. Dies ist anders als etwa bei Fahrzeugen, da dort Motor und anzutreibendes Objekt (Räder) relativ zueinander immer am gleichen Ort bleiben.

Bei diesem Projekt ist es sinnvoll, dem beschleunigenden Motor die genaue Anzahl an Umdrehungen mitzuteilen, welche er zu tätigen hat. D.h. er soll diese fixe Anzahl an Umdrehungen ausführen, sinnvollerweise natürlich genau so viele,

bis sich der Schlitten am vorderen Ende der Rampe befindet. Danach soll der Schlitten wieder in die Ausgangsposition gebracht werden. Die benötigte Anzahl an Umdrehungen bzw. Umdrehungsgrad muss dabei aus der Länge der Rampe sowie der gewählten Übersetzung berechnet werden. Alternativ wäre auch die Angabe der Zeit möglich, welche der Motor rotieren soll. Die erste Alternative kann mittels „rotateTo(int limitAngle)“, die zweite mittels „Motor.A.forward()“ und „Thread.sleep (ZeitInMillisekunden)“ verwirklicht werden. Diese Bewegung des Schlittens, also das Vor- und Zurückfahren, sollte als eigene Methode deklariert werden, um sie durch ein gewisses auslösendes Ereignis, etwa einen Knopfdruck, beliebig oft ausführen zu können. Wichtigster Faktor dieser Methode ist die Geschwindigkeit des Abwurfs, welche mittels „Motor.A.setSpeed (int speed)“ festgelegt wird.

Der zweite wichtige Punkt in diesem Projekt ist die Höhenverstellung. Wie bereits erwähnt, kann diese entweder händisch oder durch Einsatz eines weiteren Motors erfolgen. Hier wird klarerweise auf zweite Möglichkeit eingegangen. Dazu sollten am Display des NXT bestimmte Winkel zur Auswahl vorgegeben werden. Wichtig bei der gewählten Art der Höhenverstellung: befindet sich der ausfahrbare Standfuß vor dem starren Standfuß, wie in der angeführten Skizze, so muss ein Ausfahren stattfinden, um einen größeren Anstieg zu erreichen. Befindet sich der ausfahrbare Standfuß dagegen hinter dem starren Standfuß, so muss dazu ein Einfahren stattfinden. Um berechnen zu können, wie weit der ausfahrbare Standfuß für einen gewünschten Winkel ausgefahren werden muss, muss der Abstand der beiden Standfüße bekannt sein. Denn unter der Annahme, dass bei einem Winkel von $\alpha = 0^\circ$ beide Standfüße gleich hoch sind, gilt:

$$\tan \alpha = \frac{\Delta \text{Höhe}}{\text{Entfernung}}.$$

Will man also etwa einen Winkel von 20° erreichen, so bedeutet dies, dass der Höhenunterschied $\text{Entfernung} * \tan \alpha$ betragen muss. Da der Abstand zwischen den Standfüßen klarerweise bekannt ist, lässt sich die nötige Höhendifferenz berechnen. Bei einem Abstand von 25cm würde diese etwa 9,1cm betragen, um einen Winkel von 20° zu erreichen. Und genau dieser Höhenunterschied muss mittels LeJOS berechnet werden, um den zweiten Standfuß auf die korrekte Länge auszufahren. Die Math-Klasse stellt dazu die geeigneten Methoden zur Verfügung, d.h. es kann zuerst mittels „Math.tan(double a)“ der entsprechende Tangens-Wert bestimmt werden und anschließend mit der Konstanten E (Entfernung) multipliziert werden. Das Ergebnis hat die selbe Einheit wie die Entfernung, also etwa cm oder mm.

Doch der Motor, mit welchem die Stütze aus- bzw. eingefahren wird, wird nicht durch Übergabe von Längeneinheiten wie cm oder mm gesteuert, sondern durch

Übergabe von Gradzahlen. Das bedeutet, dass bekannt sein muss, um welchen Betrag sich der Fuß hebt/senkt, wenn der Motor um eine gewisse Gradzahl rotiert. Dies kann entweder durch Beachtung aller Übersetzungen berechnet werden, oder es wird gemessen, wie weit sich der Fuß bei einer bestimmten Umdrehungszahl des Motors hebt. Diese Umdrehungszahl sollte möglichst groß sein, um den Anteil des möglichen Messfehlers möglichst klein zu halten. Angenommen, es wären für eine Erhöhung des zweiten Standfußes um 40cm genau 160 Motorumdrehungen nötig, so würde gelten: $\frac{40\text{cm}}{160 \text{ Umdrehungen}} = \frac{1\text{cm}}{4 \text{ Umdrehungen}} \hat{=} \frac{1\text{cm}}{1440^\circ} = \frac{0,00694\text{mm}}{1^\circ}$

Auf diese Weise lässt sich für die kleinste Einheit, das einzelne Umdrehungsgrad, berechnen, um welchen Betrag es den Fuß hebt bzw senkt. Um die für den jeweils gewünschten Winkel α nötigen Umdrehungen zu berechnen, muss nun nur der berechnete Höhenunterschied ΔHoehe durch diese kleinste Einheit dividiert werden, und man erhält somit die nötigen Umdrehungsgrad für den gewünschten Höhenunterschied: $\frac{\Delta\text{Hoehe}[\text{mm}]}{\frac{0,00694\text{mm}}{1^\circ}}$

Beim ersten Wurf ist ein Einstellen des Winkels wohl kein Problem. Aus der Waagrechten wird die Rampe in die entsprechende Position gebracht. Doch wie wird vorgegangen, wenn der Wurf ein zweites Mal ausgeführt wird, und zwar mit einem anderen Winkel? Die einfachste Art wäre es, nach jedem Wurf die Rampe wieder in die Ausgangsposition zu bringen, wodurch allerdings viel Zeit verloren gehen würde. Eine wesentlich bessere Art ist es, etwa mit der Methode „getTachoCount()“ oder eigenen Zählern zu arbeiten. Mit solchen kann die aktuelle Lage der Rampe etwa als Integer-Wert an die Methode übergeben werden, und diese kann nun entscheiden, ob die Rampe aus- oder eingefahren werden muss oder ob sie gar nicht bewegt werden muss. Nach der Bewegung der Rampe wird der Lageparameter auf den aktuellen Wert aktualisiert.

Vor dem endgültigen Beenden muss die Rampe auf jeden Fall in eine bestimmte (waagrechte) Lage gebracht werden, da sonst bei einem erneuten Programmstart keine korrekten Werte erzielt werden können, da das Programm ja nicht „sehen“ kann, in welcher Lage die Rampe hinterlassen wurde.

Wird die Rampe nicht für Bälle, sondern für Papierflugzeuge verwendet, so muss anstatt eines Schlittens eine geeignete Haltevorrichtung verwendet werden. Diese sollte das Flugzeug genau so lange festhalten, bis dieses sich am Ende der Rampe befindet. Denkbar wäre dabei etwa ein Schnappmechanismus, welcher sich öffnet, sobald die Haltevorrichtung am vorderen Ende der Rampe anstößt. Erst dadurch würde das Flugzeug „freigegeben“ und es würde verhindert, dass es schon vor Ende der Rampe mit zu geringer Geschwindigkeit oder mit falschem Winkel wegfleht.

4.2.5 Zentrifuge

Folgendes Projekt ist dem 1. Jahrgang zuzuordnen, da in diesem der Bereich „Kinematik und Dynamik“ behandelt werden soll. Das Projekt kann bei Bedarf auch so erweitert werden, dass es das Thema „Auftrieb“ des Bereiches „Aeromechanik und Hydromechanik“ behandelt. Der Schwerpunkt liegt hier klar auf der Demonstration im Unterricht, ein Nachbau und Programmieren durch Schüler ist eher nicht zu erwarten, auch da diesen wohl die nötigen Instrumente wie etwa Kraftmesser fehlen. Abbildung 4.11 zeigt ein Aktivitätsdiagramm einer solchen Zentrifuge.

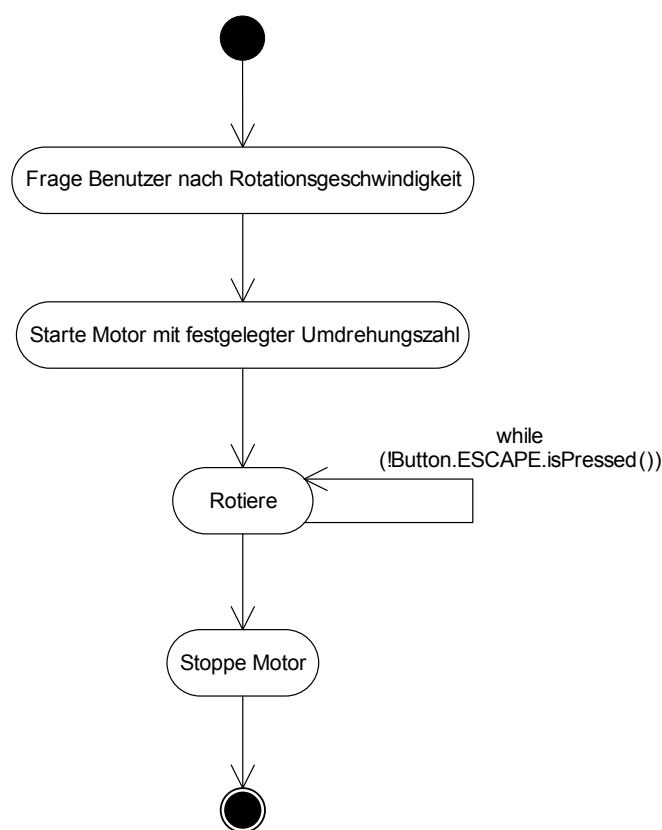


Abbildung 4.11: Zentrifuge - Aktivitätsdiagramm

Für die Umsetzung erforderlich ist neben den nötigen Bausteinen nur der NXT und mindestens ein Motor. Es könnten auch mehrere Motoren verwendet werden, um mehr Kraft aufwenden zu können. Der Motor dient nun nicht wie meist zur Bewegung eines Roboters, sondern zur Herstellung einer senkrecht rotierenden Achse. Solch eine Achse kann entweder durch eine Übersetzung der Rotation des Motors erreicht werden oder dadurch, dass der Motor schon so eingebaut wird, dass dessen Achse senkrecht steht. Diese Achse sollte möglichst stabil sein, um den später auftretenden Zentrifugalkräften standzuhalten. An dieser Achse können nun Kraftmesser mit unterschiedlichen Gewichten angebracht werden, um

die auftretenden Fliehkräfte bei unterschiedlichen Rotationsgeschwindigkeiten zu messen. Sehr hilfreich wäre auch eine Möglichkeit zur einfachen Erweiterung des Umdrehungsradius, da sich dadurch die auftretenden Fliehkräfte bei gleichbleibender Winkelgeschwindigkeit ändern. Dies ist durch die Lego-Technic-typischen Loch-Steine zu realisieren, z.B. wenn solch ein Stein so verbaut wird, dass in die jeweiligen Löcher der Kraftmesser eingehängt werden kann. Einen solchen Aufbau stellt Abbildung 4.12 schematisch dar.

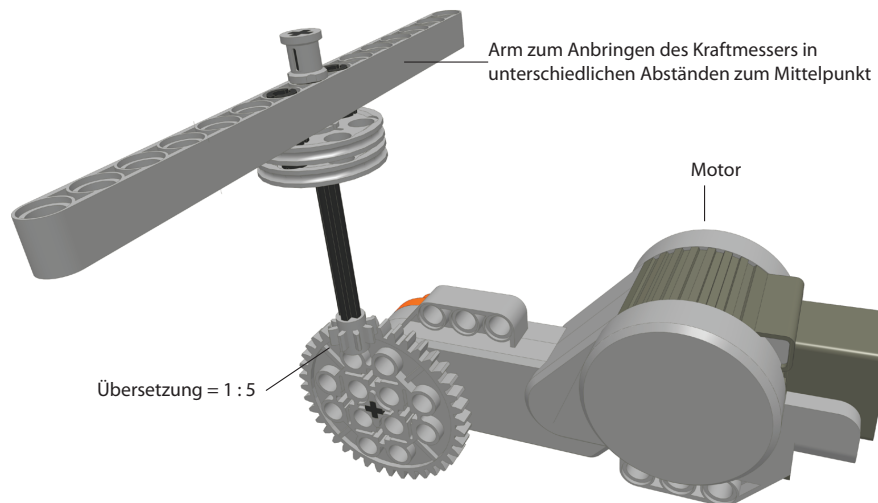


Abbildung 4.12: Zentrifuge - Aufbau

Die Erweiterung des Projekts auf den Bereich „Auftrieb“ hätte das selbe Grundgerüst als Basis, nur könnte an der rotierenden Achse etwa mittels mehrere Gelenke z.B. ein Flügel eines Modellflugzeuges angebracht werden. Nun kann der Auftrieb in Abhängigkeit der Geschwindigkeit, welche sich durch die Umdrehungszahl des Motors und durch eine Erweiterung des Durchmesser erreichen lässt, verfolgt werden.

4.2.6 Wellengeneratoren

Diese Projekte sind dem 2. Jahrgang Physik zuzuordnen, da hier die Behandlung der Ausbreitung von Wellen, stehenden Wellen und Interferenz gefordert wird. Auch hier gilt das Hauptaugenmerk der Vorführung im Unterricht, nicht dem Bau und der Programmierung durch Schüler. Ein solcher „Einsatz moderner medialer Hilfen“ [Bund06] wird aber auch vom Lehrplan gefordert.

Beide der hier geplanten Wellengeneratoren sind der Mechanik zuzuordnen und produzieren Transversalwellen². Einer der Generatoren soll dabei schlicht Wellen in einem Band produzieren, das an einem Ende an einem fixen Punkt, am an-

² Die Ausbreitung findet senkrecht zur Auslenkung statt

deren Ende an einem mittels Mindstorms-Motoren vertikal beweglichen Punkt befestigt ist. Dadurch lassen sich Wellen, stehende Wellen sowie Interferenzen in dem jeweiligen Medium, also etwa dem Band, erzeugen.

Der zweite geplante Wellengenerator kann Wellen im Medium Wasser erzeugen. Er lässt dazu entweder in einem bestimmten zeitlichen Abstand kleine Gegenstände ins Wasser fallen oder er regt das Wasser durch direkten Kontakt zu Schwingungen an, etwa durch das rhythmische Bewegen einer kleinen Platte direkt an der Wasseroberfläche. Um Interferenzen beobachten zu können, müssen die erzeugten Wellen entweder am Rand des verwendeten Beckens reflektiert werden oder es muss ein zweiter Wellengenerator vorhanden sein.

Um beim ersten Typ simple Wellen zu erzeugen, ist nur die Angabe der Auslenkungsamplitude sowie der Frequenz (f) bzw. der Periodendauer ($T = \frac{1}{f}$) erforderlich. Abbildung 4.13 zeigt skizzenhaft einen entsprechenden Aufbau. Sollen

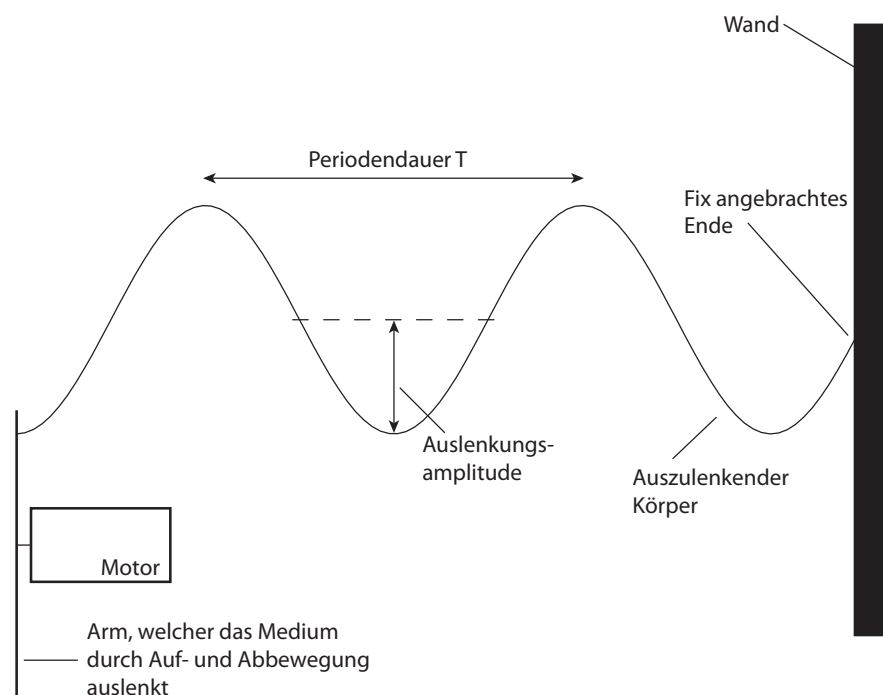


Abbildung 4.13: Wellengenerator - Aufbau Typ 1

stehende Wellen erzeugt werden, so ist auch die Angabe der Länge des schwingenden Mediums, also etwa des Bandes, nötig. Auf einer vorgegebenen Länge kann nicht mit jeder beliebigen Frequenz eine stehende Welle erzeugt werden, d.h. es könnte die angegebene Frequenz als Richtwert angenommen werden und die nächst mögliche Frequenz für eine stehende Welle gesucht werden. Um die Wellen zu erzeugen, kann ein Motor mit geeigneter Übersetzung die vertikalen Auslenkungen ausführen. Dieser muss im Wechsel vor und zurück bewegt werden, um die Auslenkung nach oben und unten durchzuführen. Dazu wird etwa

mittels verschachtelten while-Schleifen jeweils die Auslenkung nach oben und unten durchgeführt. Wichtig ist dabei, dass die erste Auslenkung, wenn sich der Arm in neutralem Zustand befindet, nur um die Hälfte der gewöhnlichen Auslenkung stattfindet. Ein Ablauf könnte also so aussehen: „0,5 nach oben, 1 nach unten, 1 nach oben, 1 nach unten ...“

Beim zweiten Roboter, dessen Aufbau in Abbildung 4.14 skizzenhaft dargestellt ist, lässt ein mittels Motor gesteuerter Arm in vorgegebenen Abständen Steine in eine Flüssigkeit fallen. Dadurch können, vorausgesetzt es ist ein geeignetes Becken

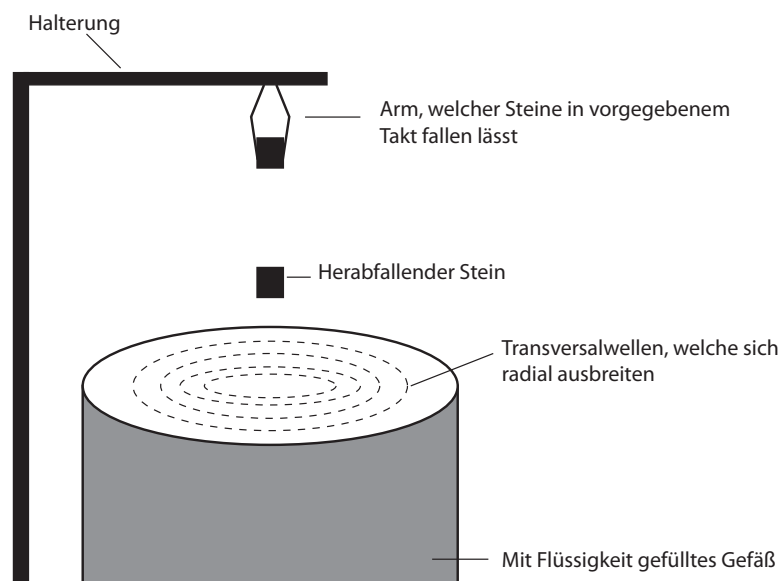


Abbildung 4.14: Wellengenerator - Aufbau Typ 2

vorhanden, Interferenzen beobachtet werden. Ebenso kann ein mittels Motor gesteuerter Arm direkt die Wasseroberfläche berühren und diese zu Schwingungen anregen, vergleichbar mit einer auf die Wasseroberfläche aufgelegten und vertikal bewegten Hand. Die Steuerung basiert hier auf dem selben Prinzip wie beim ersten Roboter, d.h. es muss eine vertikale Auslenkung stattfinden. Bei neutralem Startzustand findet die erste Auslenkung auch hier nur um die Hälfte der dann folgenden Auslenkungen statt.

4.2.7 Fallbeschleunigung

Dieses Projekt ist dem 1. Jahrgang zuzuordnen, in welchem u.a. Beschleunigung und Gravitation zu behandeln sind. Anhand der Messung der Fallzeit von Objekten soll die Erdbeschleunigung gemessen werden. Dieses Projekt eignet sich sowohl zur bloßen Vorführung durch die Lehrkraft, als auch zum Bauen und Programmieren durch Schüler. Diesen kann dazu eine Aufgabe wie etwa „Bauen und

programmieren Sie einen Roboter, welcher anhand mehrerer Objekte die Erdbeschleunigung ermittelt“ gestellt werden.

Der hier benötigte Roboter muss auf einer bekannten Strecke die Fallzeit messen, um daraus die (durchschnittliche) Beschleunigung auf dieser Strecke nach der Formel $\vec{a} = \frac{\Delta \vec{v}}{\Delta t}$ berechnen zu können. Diese sollte der Konstanten $g = 9,81 \frac{\text{m}}{\text{s}^2}$ möglichst nahe kommen.

Die Strecke ist dadurch bekannt, dass schlicht die Höhendifferenz von Anfangs- bis Endpunkt gemessen wird. Diese muss bekannt sein, um die Geschwindigkeit berechnen zu können. Δt lässt sich mittels der „elapsed()“-Methode der Klasse „Stopwatch“ messen. Nun stellt sich allerdings die Frage: wann wird die Stoppuhr gestartet bzw. beendet und wie lässt man überhaupt Objekte fallen?

Um Objekte fallen zu lassen eignen sich am besten motorgesteuerte Arme oder Brücken. Eine zweiteilige Brücke kann auseinanderfahren und das Objekt in der Mitte hinabfallen lassen, ein Arm kann sich auf Befehl öffnen. Abbildung 4.15 zeigt skizzenhaft die Verwirklichung eines solchen Aufbaus mittels Greifarm.

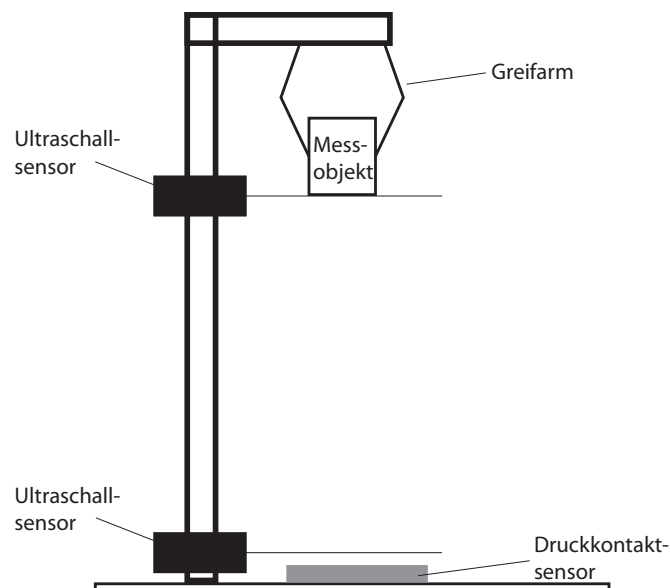


Abbildung 4.15: Fallbeschleunigung - Aufbau

Wie bereits angesprochen muss zur Messung der Geschwindigkeit die benötigte Zeit gemessen werden. Dazu können mehrere Möglichkeiten angewandt werden, welche teilweise schon in der Skizze angedeutet sind.

Zum einen kann die Messung der Zeit beginnen, sobald der Greifarm sich öffnet bzw. einen gewissen Öffnungsgrad erreicht. Dazu muss allerdings bekannt sein, wie weit sich dieser öffnen muss, damit das jeweilige Objekt auch tatsächlich fällt. Dies wäre sinnvoll etwa mittels „rotateTo()“ möglich, wodurch der Motor, welcher den Greifarm antreibt, eine fixe Anzahl an Umdrehungen ausführt. Wesentlich

genauer ist ein Start der Messung mittels eines Ultraschallsensors. So kann hier die Messung etwa begonnen werden, sobald die gemessene Entfernung einen bestimmten Grenzwert unterschreitet. Die Messung kann bei Postierung unter dem Objekt direkt nach der Freigabe starten, ohne die Umdrehungszahl des Greifarm-Motors zu kennen. Ein Starten der Zeit bedeutet im Code ein zurücksetzen der Stoppuhr mittels der „reset()“-Methode.

Um die Zeit zu stoppen, also mittels „elapsed()“ auszulesen, stehen ebenfalls mehrere Möglichkeiten zur Verfügung. Zum einen kann diese gestoppt werden, sobald das Objekt den Druckkontaktsensor auslöst, entweder durch direkten oder indirekten Kontakt, etwa durch eine darüber liegende Platte, um die Kollisionsfläche zu vergrößern und die für die Auslösung benötigte Kraft zu verkleinern, welche 0,34 Newton beträgt [Geba07]. Ein Ultraschallsensor kann auch wieder zum Stoppen der Zeit verwendet werden, wenn das Objekt an diesem „vorbeifliegt“.

Eine etwas außergewöhnliche Möglichkeit bietet die Verwendung des Geräuschsensors, mittels welchem die Zeit gestoppt werden kann, sobald ein bestimmter Schalldruckpegel erreicht wird, also das Aufschlagen des Objekts registriert wird. Dabei muss sich der Sensor möglichst nahe am Ort des Aufschlages befinden, um keine Verfälschung des Resultats aufgrund der Schallgeschwindigkeit zu erhalten. Diese könnte natürlich auch in die Berechnung mit einbezogen werden, wenn der Aufschlagort genau bekannt ist, da ja nur dann die Entfernung bekannt ist. Die Lautstärke des Aufschlags des Objekts muss weiters laut genug sein, um von dem Sensor erkannt zu werden, d.h. es dürfen keine allzu hohen Nebengeräusche vorhanden sein.

Auf eine dieser Arten wird nun also die Zeit gemessen, welche das Objekt für den Fall aus einer bekannten Höhe benötigt. Daraus berechnet sich die Geschwindigkeit mittels $v = \frac{s}{t}$. Die Beschleunigung berechnet sich nun wie bereits angegeben als Division des Geschwindigkeitsunterschiedes durch den Zeitunterschied, also $\vec{a} = \frac{\Delta \vec{v}}{\Delta t}$. Dabei ist wichtig, dass die Messung der Zeit bei $v \approx 0$ beginnt, da sonst die berechnete Geschwindigkeit nicht Δv entspricht.

4.2.8 Messung des Schalldruckpegels

Dieses Projekt ist dem 2. Jahrgang zuzuordnen, in welchem laut Lehrplan u.a. Akustik, Schwingungen und Wellen behandelt werden sollen. Dafür drängt sich der Lautstärkesensor geradezu auf. Ziel des hier vorgesehenen Projektes ist es, einen Roboter zu bauen und v.a. zu programmieren, welcher ausgehend von einem gemessenen Referenzschallpegel Orte markiert, an welchen eine für den Menschen nur noch halb so hohe Lautstärke vorliegt, d.h. der gemessene Wert um 10 dB(A)

niedriger ist.

Mittels „SoundSensor sound = new SoundSensor(SensorPort.S1, dba)“ wird eine Instanz des Lautstärkesensors erstellt, der durch das Hinzufügen eines Filters (A) das Hörempfinden des menschlichen Ohres nachahmt, welches verschieden auf unterschiedliche Frequenzbereiche reagiert. Mittels der Methode „readValue()“ kann der aktuelle Wert als Integer ausgelesen werden, allerdings nicht als dB(A)-Wert, sondern als Prozentangabe. D.h. die größte messbare Lautstärke wird mit 100% angegeben, absolute Stille (welche real nicht zu erreichen ist) entspräche 0%. Hier liegt auch das Problem der Genauigkeit dieses Sensors: Messungen lauter Quellen liefern stets ein Ergebnis von 100% [Geba07]. Erst mit deutlich zunehmendem Abstand sind die Messergebnisse zu gebrauchen.

Ein bestimmter Schalldruckpegel verringert sich bei einer Verdopplung des Abstandes um ca. 6 dB(A). Dies ist für das menschliche Gehör allerdings kein prägnanter Wert, da durch das logarithmische Gehör v.a. Veränderungen um 10 dB(A) und Vielfache davon deutlich sind. Eine Verringerung um 10 dB(A) entspricht einer Halbierung der empfundenen Lautstärke. Eine solche Halbierung findet bei einer Vervierfachung des ursprünglichen Abstandes statt.

Auf diese Art soll auch dieser Roboter funktionieren. Er soll in der Nähe einer Schallquelle aufgestellt werden, sich von dieser gleichmäßig entfernen und durch Messung des Schalldruckpegels jene Punkte markieren, an denen die „gefühlte Lautheit“ z.B. der Hälfte oder einem Viertel des Aufstellungsortes entspricht. Dabei ist zu beachten, dass andere Schallquellen oder Schallreflexionen die Messung möglichst wenig stören sollen. Für den Einsatz in der Schule bieten sich als Geräuschquellen etwa Radio, Fernseher, KFZ von Lehrern und Schülern oder andere NXT-Geräte an, welche mittels der Sound-Klasse zahlreiche Funktionen zur Erzeugung von Klängen bieten. Abbildung 4.16 zeigt in Form eines Aktivitätsdiagramms, wie eine solche Markierung von Punkten erfolgen kann.

Das Aktivitätsdiagramm zeigt, dass direkt nach Start des Roboters ein Referenzschallpegel gemessen wird. Dies geschieht mittels der Methode „readValue()“, welche als Ergebnis einen Integer-Wert liefert. Dies ist, wie bereits erwähnt, ein Prozentwert, kein dB(A)-Wert. Das bedeutet auch, dass nun nicht ein Punkt gesucht wird, an dem der nächste mittels „readValue()“ ausgelesene Wert um zehn Punkte niedriger ist, sondern ein Punkt, an welchem der gemessene Wert genau halb so groß ist wie der Referenzwert. Nun soll der Roboter also so lange geradeaus fahren, bis er einen Wert von „0,5 * Referenz“ misst. An jenem Ort, an welchem er solch einen Wert misst, soll er etwa einen Lego-Stein ablegen, um diesen zu markieren und somit ein Nachvollziehen der Messergebnisse zu ermöglichen. Nach dem Ablegen soll der Pegel an genau diesem Ort als neue Referenz festgelegt wer-

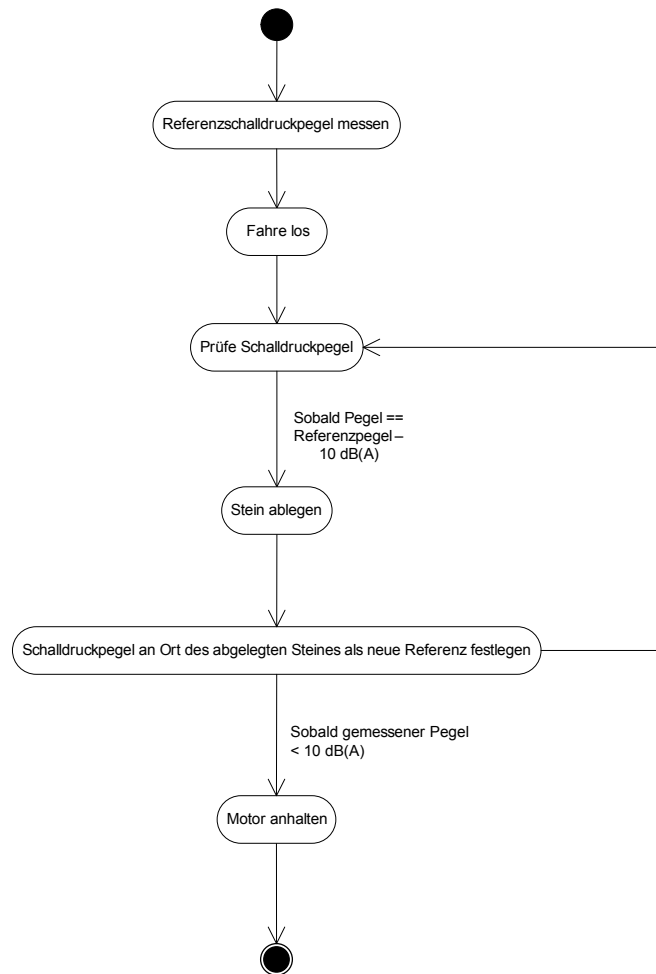


Abbildung 4.16: Schalldruckpegel - Aktivitätsdiagramm

den und nun wieder jener Ort gesucht, für welchen gilt: „gemessener Wert == 0,5 * Referenz“. Dies kann am einfachsten mittels einer while- oder do-while-Schleife realisiert werden. Als Abbruchkriterien kommen mehrere Variablen in Frage. Etwa ein Überschreiten einer gewissen Distanz zum Ausgangspunkt (gemessen mittels „getTachoCount“), ein Unterschreiten einer vorgegebenen Lautstärke, ein Unterschreiten eines gewissen Anteils des Referenzschalldruckpegels oder auch ein Ansteigen des Pegels, sollte sich der Roboter einer anderen Schallquelle nähern.

Für diesen Roboter wäre es völlig ausreichend, ihn mit einem Motor sowie dem notwendigen Geräuschsensor auszustatten. Ein für eine Lenkung notwendiger zweiter Motor wäre etwa dann notwendig, wenn der Roboter selbständig Kurven mit gleichem Schalldruckpegel markieren sollte. Andere Sensoren könnten natürlich auch verbaut werden, etwa der Druckkontaktsensor zum Start des Programms oder der Ultraschallsensor zur Erkennung von Hindernissen. Auf die Funktionalität der Messung hätte dies allerdings keinen Einfluss.

Zu Beginn der Messung wäre auch eine Überprüfung der Fahrtrichtung des

Roboters möglich. Sollte der Roboter also während der Fahrt eine Erhöhung des Schalldruckpegels erkennen, so könnte er etwa mittels der Methode „changeDirection()“ schlicht die Umdrehungsrichtung des Motors ändern, zum Ausgangspunkt zurückkehren (möglich etwa mittels „getTachoCount“, um genau jene Umdrehungszahl zu eruieren, welche in die falsche Richtung gefahren wurde) und die Fahrt in die entgegengesetzte Richtung fortsetzen.

4.3 Angewandte Programmierung

Der Unterricht der angewandten Programmierung bietet klarerweise die meisten Möglichkeiten zum Einsatz von Lego Mindstorms in Verbindung mit LeJOS. Im Folgenden werden einige für den Einsatz im Unterricht besonders nützlich erscheinende Beispiele angeführt.

4.3.1 Funktionstest für Sensoren und Motoren

Dieses Projekt ist dem 1. Jahrgang der Angewandten Programmierung zuzuordnen und sollte von allen Schülern in Einzel- oder Gruppenarbeit durchgeführt werden, zum einen, um das Programmieren mittels LeJOS zu erlernen, zum anderen, um die Funktionstüchtigkeit ihres NXT zu überprüfen. Für dieses Projekt werden neben dem NXT genau jene Sensoren und Motoren benötigt, welche auf ihre Funktionstüchtigkeit überprüft werden sollen, also im Idealfall drei Motoren und alle vier Sensoren. Bausteine werden für eine simple Ausführung nicht benötigt.

Bezüglich der Programmierung am einfachsten umzusetzen ist ein alleinstehender Test der jeweiligen Sensoren bzw. der Motoren, d.h. es wird für diese jeweils ein eigenes Programm geschrieben. Sinnvoll kann es dabei auch sein, nicht nur die Peripheriegeräte, sondern auch den NXT zu überprüfen, indem etwa die Peripheriegeräte an verschiedenen Ports angeschlossen werden. Dabei ist natürlich zu beachten, dass das jeweilige Programm daran anzupassen ist, da dem NXT mitgeteilt werden muss, welcher Port angesprochen werden soll.

Ein Test für die Motoren kann etwa aus einer schlichten while-Schleife bestehen, mittels welcher der bzw. die angeschlossenen Motoren so lange zum Rotieren gebracht werden, bis ein Abbruchkriterium erfüllt ist. Dies könnte etwa schlicht das Drücken der Escape-Taste am NXT sein. Folgende while-Schleife lässt den Motor solange um einen Winkel von 1° Weiterdrehen, bis die Escape-Taste gedrückt wird.

```
while (!Button.ESCAPE.isPressed()) {  
    Motor.A.rotate(1)
```

```
}
```

Die Bezeichnung des Motors ist dabei natürlich an den jeweils verwendeten Port anzupassen, wobei auch alle drei Motoren gleichzeitig getestet werden können, da am NXT ja auch drei Ports dafür zur Verfügung stehen. Dazu werden schlicht noch zwei Code-Zeilen in der while-Schleife eingefügt, welche jeweils Motor B und C ansprechen.

Gerade bei dem Test der Motoren können gut die verschiedenen Schleifenarten wie „while“ oder „do-while“ ausprobiert werden, da dabei festgestellt werden kann, ab wann und wie lange sich ein Motor dreht. Um diese Beobachtung zu unterstützen, kann hier auch etwa eine Ausgabe der vom Motor getätigten Umdrehungen am Display stattfinden, welche mittels „getTachoCount()“ ausgelesen werden kann. Weiters ist hier natürlich der Test nicht auf eine bloße Vorwärtsumdrehung beschränkt, genauso können natürlich die Methoden „stop()“ zum abrupten Stopp oder „flt()“ zum Auslaufen des Motors oder auch die pilot-Klasse, mittels welcher die Bewegungen zweier Motoren aufeinander abgestimmt werden können, behandelt werden.

Der Test der Sensoren muss im Gegensatz zu jenem der Motoren einzeln für jeden Sensortyp programmiert werden, da diese ja eine jeweils spezifische Funktion erfüllen, welche mit jeweils spezifischem Code abzurufen ist. Sinnvoll erscheint dabei jeweils eine Ausgabe des aktuell gemessenen Zustandes am Display des NXT.

So kann also etwa in der ersten Zeile des Displays „0“ oder „1“ ausgegeben werden, je nachdem, ob der Druckkontaktsensor gedrückt ist oder nicht.

```
int status;  
if(touch.isPressed()) {  
    status = 1;  
}  
if(!touch.isPressed()) {  
    status = 0;  
}  
LCD.drawInt(status, 0,0);
```

Dies ist als Minimalbeispiel anzusehen, da hier eben nur „0“ und „1“ angezeigt werden. In einer feineren Ausführung kann etwa vor dieser Angabe die Portnummer angeführt werden und anstatt der Ausgabe von Zahlen kann auch ein Text ausgegeben werden.

Die Messwerte der restlichen drei Sensoren können direkt am Display ausgegeben werden, ohne zuerst wie in obigem Beispiel den jeweiligen Zuständen Integer-Werte zuordnen zu müssen, welche dann am Display ausgegeben werden können. Dies liegt daran, dass diese jeweils schon Integer-Werte liefern, welche

als Ausgabewerte genutzt werden können, wohingegen der Druckkontaktsensor einen Boolean-Wert liefert, welcher nicht direkt ausgegeben werden kann.

Hier wird nun als Beispiel der Helligkeitssensor angeführt. Um in der zweiten Zeile des Displays laufend die aktuell gemessene Helligkeit anzuzeigen, kann folgender Code verwendet werden:

```
LCD.drawInt(light.readNormalizedValue(), 0,1);
```

Bevor dieses Auslesen von Messwerten stattfinden kann, muss eine neue Instanz des jeweiligen Sensors erstellt werden und diesem ein eindeutiger Port sowie ein eindeutiger Name zugewiesen werden. Dies kann etwa für den Helligkeitssensor folgendermaßen geschehen:

```
LightSensor light = new LightSensor(SensorPort.S1);
```

Nach dem gleichen Schema funktioniert der Test der anderen beiden Sensoren. Sie werden, zumindest falls der Test zeitgleich stattfinden soll, an jeweils verschiedene Ports angeschlossen, die Benennung der Sensoren fällt klarerweise anders aus und die aufzurufende Methode ist jeweils eine andere. Aber die beiden hier angeführten Schritte sind notwendig.

Eine Erweiterung dieses Tests kann durch die Integration eines grafischen Menüs am NXT erreicht werden. Ein sehr gutes Beispiel für eine mögliche Umsetzung findet sich in View.java im Ordner `lejos_nxj\projects\samples\View\` der LeJOS-Installation, weshalb hierauf nicht näher eingegangen wird.

4.3.2 Line Follower

Dieses Projekt ist ebenfalls dem 1. Jahrgang der Angewandten Programmierung zuzuordnen, da für diesen die Behandlung von Anweisungsfolgen und Wiederholungen gefordert wird. Jeder Schüler bzw. in Kleingruppen zusammengeschlossene Schüler sollen dieses Projekt umsetzen, um das Programmieren, wie etwa while-Schleifen, zu erlernen.

Ein Line-Follower bezeichnet einen Roboter, welcher, wie der Name vermuten lässt, einer vorgegebenen Linie folgt. Im Lieferumfang des Mindstorms ist bereits eine Matte inkludiert, auf welcher sich eine gut dafür geeignete breite Linie befindet. Es können aber auch eigene Linien verwendet werden. Dazu kann z.B. mit einem dicken Filzstift auf Papier gezeichnet oder ein farbiges Klebeband auf den Boden geklebt werden. Da für solch einen Line Follower der Helligkeitssensor verwendet wird, muss in jedem Fall darauf geachtet werden, dass der Kontrast zwischen Linie und Untergrund stark genug ist, um eine Unterscheidung für den Sensor zu ermöglichen.

Für den Aufbau des Roboters ist hier anzunehmen, dass sich der Helligkeitssensor am vorderen Ende befindet und dass er von zwei Motoren angetrieben wird, wodurch ein Drehen am Stand möglich ist. Wird der Roboter mit einem einzelnen Helligkeitssensor ausgestattet, so sind grundsätzlich zwei Arten der Fortbewegung möglich: es kann zum einen so lange geradeaus gefahren werden, bis der Sensor nicht mehr über der Linie ist, um dann in die entsprechende Richtung zu drehen, oder es wird generell nie geradeaus gefahren sondern anstatt dessen immer abwechselnd nur das rechte und linke Rad angetrieben, bis der Sensor die Linie verlässt. Hier wird auf die zweite Möglichkeit eingegangen.

Um zu wissen, welcher Linie denn überhaupt gefolgt werden soll, kann ebenfalls auf zwei Arten vorgegangen werden. Zum einen kann ein bestimmter Helligkeitswert direkt in den Java-Code geschrieben werden, welcher einer Linie entspricht. Dadurch ist es nun egal, ob der Roboter auf der Linie oder daneben gestartet wird. Er dreht sich nun so lange, bis er diese gefunden hat. Natürlich könnten auch bestimmte Methoden zur Suche von Linien implementiert werden.

Die zweite Möglichkeit besteht darin, dass für die Linie kein fixer Helligkeitswert vorgegeben wird, sondern dieser gemessen wird. Dazu ist es allerdings nötig, dass sich der Helligkeitssensor beim Start über dieser Linie befindet. Hier wird auf diese Art der Lösung eingegangen und diese ist in Abbildung 4.17 anhand eines Aktivitätsdiagramms dargestellt.

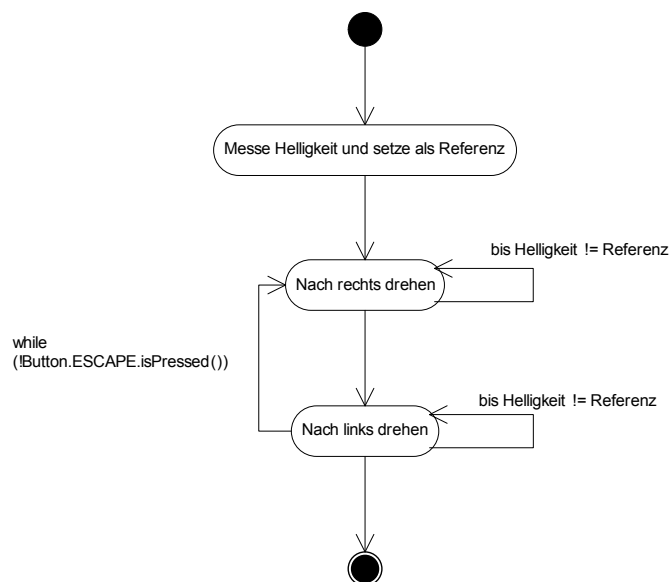


Abbildung 4.17: Line Follower - Aktivitätsdiagramm

Zu Beginn muss mittels „`LightSensor licht = new LightSensor(SensorPort.S1)`“ eine Instanz des Lichtsensors erstellt werden, wobei „S1“ für den verwendeten Port steht, hier also Port 1. Der Sensor muss beim Ausführen des Programmes

dann auch an den entsprechenden Port angeschlossen werden, um mit diesem kommunizieren zu können.

Mittels „`licht.readValue()`“ kann nun der jeweilige Helligkeitswert ausgelesen werden. Ausgegeben wird ein Integer-Wert, welcher laut API dem Prozentsatz der Differenz zwischen Maximum und Minimum entspricht. Um gute Ergebnisse zu erzielen, sollte mittels „`setFloodlight(boolean floodlight)`“ die Beleuchtung des Sensors aktiviert werden. Der so ermittelte Referenzwert kann dann etwa als „`final int`“ festgelegt werden.

Nun soll mit der Fortbewegung begonnen werden. Dazu kann entweder komplett auf die Programmierung eigener Methoden verzichtet werden, indem Schleifen verschachtelt werden. So könnte die äußere Schleife als Abbruchkriterium das Drücken eines Knopfes besitzen, die Mittlere wie die Innere das Messen eines nicht dem Referenzwert entsprechenden Helligkeitswertes. In den beiden inneren Schleifen kann damit jeweils einmal der linke und einmal der rechte Motor angetrieben werden, während der andere jeweils mittels „`flt()`“ ausläuft oder komplett still steht, um die Linie im Zick-Zack-Muster abzufahren. Für die Drehungen nach links und rechts können auch jeweils eigene Methoden geschrieben werden, wodurch dann eine einzelne Schleife ausreichen würde. Also etwa: „Solange (Escape nicht gedrückt): Fahre nach rechts, fahre nach links“.

Denkbar wäre auch, den Helligkeitswert nicht als fixen Integer-Wert anzunehmen, sondern als Intervall, also etwa $[0,9 \cdot \text{Referenz}; 1,1 \cdot \text{Referenz}]$. Sinnvoll ist dies besonders dann, wenn die Linie, welcher der Roboter folgen soll, nicht an jeder Stelle die gleichen Helligkeitswerte aufweist. Beim Einsatz von Klebestreifen ist dies u.a. aufgrund von Blasen, Faltenbildung oder unterschiedlicher Dehnung sinnvoll.

Bei dieser Umsetzung ist es völlig egal, ob sich der Roboter auf einer Geraden oder in einer Kurve befindet, er pendelt immer zwischen den beiden äußeren Enden der Linie. Auf einer Gerade ist er dadurch in der Fortbewegung eher langsam, in Kurven dagegen ist er auffallend schnell, da hier längere Passagen in die gleiche Richtung zu drehen sind.

Eine Verfeinerung des Line Followers kann durch den Einsatz von zwei Helligkeitssensoren erreicht werden. Dazu können sich entweder größere Gruppen bilden oder diejenigen, die den Sensor gerade nicht benötigen, diesen an andere ausleihen. Der zweite Helligkeitssensor bietet die Möglichkeit zu erkennen, auf welcher Seite die Linie verlassen wurde. Bei dem vorherigen Zick-Zack-Kurs war dies zwar auch möglich, allerdings nur wegen eben dieses Zick-Zack-Fahrens. Würde ein Roboter, welcher geradeaus fährt und mit nur einem Sensor ausgestattet ist, von der Linie abkommen, so wäre nicht festzustellen, auf welcher Seite sich nun die Linie

befindet, ohne den Roboter zu drehen.

Mittels des zweiten Sensors ist dies möglich. Die beiden Sensoren müssen dazu nebeneinander angeordnet werden. Wenn sich nun etwa nur der linke Sensor nicht mehr über der Linie befindet, so ist klar, dass eine Rechtskurve folgen muss. Die restlichen Abläufe sind ähnlich wie beim vorherigen Modell. Auch hier kann ein fixer Helligkeitswert festgelegt werden oder dieser kann durch mindestens einen der Sensoren abgerufen werden. Grundlegender Unterschied hier ist, dass der Roboter so lange geradeaus fährt, bis einer der Sensoren nicht mehr den Referenzwert misst. Sobald dies der Fall ist, wird der Motor auf der anderen Seite z.B. mittels „flt()“ angehalten, während der Motor auf jener Seite, auf welcher der Referenzwert nicht mehr gemessen wurde, weiterläuft. Dadurch kommt eine Drehung zu Stande. Wie lange diese Drehung stattfinden soll, muss für den jeweiligen Fall entschieden werden. Ob etwa schon bei einem erneuten Messen des Referenzwertes die Drehung gestoppt wird oder ob die Drehung noch darüber hinaus gehen soll. Nach Abschluss der Drehung kann die Fahrt wieder geradeaus fortgesetzt werden.

Für solch eine Umsetzung eignen sich wieder zum einen verschachtelte Schleifen oder eigene Methoden. Hier kommt es auch auf persönliche Präferenzen an, übersichtlicher und v.a. bezüglich der objektorientierten Programmierung sinnvoller ist sicherlich die Umsetzung mittels Methoden.

4.3.3 Labyrinth

Dieses Projekt kann sowohl im ersten als auch zweiten Jahrgang der Angewandten Programmierung ausgeführt werden, da für diese etwa die Behandlung von Wiederholungen, Verzweigungen, einfachen Datentypen aber auch Operationen auf Datenstrukturen (2. Jahrgang) gefordert wird. Der Einsatz von Lego Mindstorms dient hier in erster Linie der Veranschaulichung.

Für dieses Projekt wird bei dem hier dargestellten Aufbau ein lenkbarer Roboter mit Helligkeitssensoren benötigt. Der Roboter selbst ist dabei ein Line-Follower. Dieser folgt so lange einem auf dem Boden aufgezeichneten Labyrinth, bis er das farblich markierte Ziel erreicht. Ein solches Labyrinth könnte etwa wie in Abbildung 4.18 aussehen.

Dabei wird von einem der farblich markierten Felder gestartet, das andere farbige Feld stellt das zu erreichende Ziel dar. Um Abzweigungen möglichst einfach erkennen zu können, sollten mehrere Helligkeitssensoren nebeneinander eingesetzt werden, optimaler Weise drei Stück. Der mittlere Sensor dient der Fahrt geradeaus, mittels der beiden äußeren Sensoren, welche rechts und links der zu verfolgenden Linie platziert sind, kann erkannt werden, wo Abzweigungen vor-

tes sind nur geringe LeJOS-spezifische Kenntnisse erforderlich. So müssen vom Helligkeitssensor Werte abgefragt werden und mit einem kritischen Wert verglichen werden. Das Drehen des Roboters ist am sinnvollsten mittels der `pilot`-Klasse zu erreichen.

Der Prozess der Fahrt durch das Labyrinth kann grundsätzlich auf zwei unterschiedliche Arten umgesetzt werden. Zum einen kann allein mit Schleifen gearbeitet werden, allerdings drängt sich hier der Einsatz der Rekursion auf, da zum einen stets die gleichen, aufeinander aufbauenden Entscheidungen getroffen werden müssen, und zum anderen ist die Rekursion ein zentraler Bestandteil der Programmierung.

Die Umsetzung einer solchen rekursiven Fahrt durch das Labyrinth, bei welcher der schnellste Weg nicht ermittelt wird, ist in Form einer Methode in Programmcode 4.8 angeführt.

Programmcode 4.8: Labyrinth - Rekursives Suchen eines Weges

```
public static void rekursion() {
    /*Die Variablen "links", "vorne" und "rechts" für die jeweiligen
       Sensoren anlegen und ihnen die entsprechenden Sensorwerte
       zuordnen*/
    int links = licht1.readNormalizedValue();
    int vorne = licht2.readNormalizedValue();
    int rechts = licht3.readNormalizedValue();
    /*Falls Zielwert erreicht wurde, wird dieser solange am Display
       angezeigt, bis der Escape-Button gedrückt wird*/
    if (vorne == zielwert) {
        while (!Button.ESCAPE.isPressed()) {
            LCD.clear();
            LCD.drawString("Ziel erreicht", 0,0);
        }
    }
    /*Wenn Zielwert noch nicht erreicht*/
    else {
        /*Falls nur Fahrt nach vorne möglich: geradeaus fahren und
           Methode erneut starten*/
        if (vorne == linie && rechts != linie && links != linie) {
            vorwaerts(5); //5° geradeaus fahren (wie LineFollower)
            rekursion();
        }
        /*Falls Roboter in Sackgasse: Umdrehen, geradeausfahren und
           Methode erneut starten*/
        if (vorne != linie && rechts != linie && links != linie) {
            pilot.rotate(180); //Roboter dreht um
            vorwaerts(5);
            rekursion();
        }
        /*Falls Fahrt in alle drei Richtungen möglich: nach rechts
           abbiegen, geradeaus fahren und Methode erneut starten*/
        if (vorne == linie && rechts == linie && links == linie) {
            pilot.rotate(-90); //Roboter dreht sich 90° nach rechts
            vorwaerts(30); //Roboter fährt 30° geradeaus, um seitliche
                           Sensoren aus Kreuzungsbereich zu entfernen
        }
    }
}
```

```

    rekursion();
}
/*Falls Fahrt geradeaus & rechts möglich: nach rechts abbiegen,
   geradeaus fahren und Methode erneut starten*/
if (vorne == linie && rechts == linie && links != linie) {
    pilot.rotate(-90);
    vorwaerts(30);
    rekursion();
}
/*Falls Fahrt geradeaus & links möglich: geradeaus fahren und
   Methode erneut starten*/
if (vorne == linie && rechts != linie && links == linie) {
    vorwaerts(30);
    rekursion();
}
/*Falls Fahrt rechts & links möglich: nach rechts abbiegen,
   geradeaus fahren und Methode erneut starten*/
if (vorne != linie && rechts == linie && links == linie) {
    pilot.rotate(-90);
    vorwaerts(30);
    rekursion();
}
/*Falls Fahrt nur nach rechts möglich: nach rechts abbiegen,
   geradeaus fahren und Methode erneut starten*/
if (vorne != linie && rechts == linie && links != linie) {
    pilot.rotate(-90);
    vorwaerts(30);
    rekursion();
}
/*Falls Fahrt nur nach links möglich: nach links abbiegen,
   geradeaus fahren und Methode erneut starten*/
if (vorne != linie && rechts != linie && links == linie) {
    pilot.rotate(90); //Roboter dreht sich 90° nach links
    vorwaerts(30);
    rekursion();
}
}
}

```

In diesem Beispiel wird als erstes bestimmt, ob der mittlere Sensor schon den Zielbereich erreicht hat. Ist dies der Fall, wird die Rekursion nicht weiter ausgeführt. Ist dies nicht der Fall, werden für alle möglichen Konstellationen Abläufe vorgegeben.

Sinnvoll wäre hier wie schon beim klassischen Line-Follower (siehe Abschnitt 4.3.2) angesprochen, bei der Überprüfung der Helligkeitswerte von Linie und Zielbereich nicht nur den exakt vorgegebenen Wert heranzuziehen, sondern etwa ein Intervall von $[0,9 * \text{kritischer Wert}; 1,1 * \text{kritischer Wert}]$. Im vorherigen Beispiel wurde dies aus Gründen der Übersichtlichkeit nicht umgesetzt. Bei der Überprüfung auf den Zielwert müsste der Code also wie folgt aussehen:

```

if (vorne >= 0.9*zielwert && vorne <= 1.1*zielwert) {
    LCD.drawString("Ziel erreicht", 0,0);
}

```

Befindet sich der Roboter auf einer Geraden ohne Abzweigung, fährt er immer weiter geradeaus. Diese Fahrt geradeaus wird als eigene Methode „vorwaerts()“ umgesetzt, welche nach dem gleichen System wie der Line-Follower arbeitet. D.h. es werden abwechselnd linkes und rechtes Rad angetrieben. Hierbei ist allerdings zu beachten, dass im Bereich einer Abzweigung die Linie scheinbar breiter ist, der Roboter also weiter drehen könnte. Durch die beiden zusätzlichen Sensoren kann dies allerdings verhindert werden.

In einer Sackgasse dreht sich der Roboter um 180° . Das Rechtsabbiegen oder Geradeausfahren wird dem Linksabbiegen immer vorgezogen. Nach einer Drehung sollte sich der Roboter ein größeres Stück als gewöhnlich fortbewegen. Dies ist deshalb nötig, da sich sonst nach einer Rechtsdrehung der rechte Sensor, welcher sich vor der Drehung neben der Strecke befand, dann über der Linie befindet, welche schon abgefahren wurde und es würde wieder eine Drehung eingeleitet werden. Der Roboter würde sich auf der Stelle drehen. Der hier gewählte Wert ist wieder abhängig von der Größe der verwendeten Räder und v.a. der Dicke der Linie des Labyrinthes.

Tabelle 4.1 zeigt noch einmal alle möglichen Konstellationen von Messwerten und deren Auswirkungen, abgesehen von der Messung des Zielwertes und unter der Voraussetzung, dass das Abbiegen nach rechts immer vorgezogen wird.

Messwert links	Messwert mitte	Messwert rechts	Resultat
schwarz	schwarz	schwarz	⇒ rechts abbiegen
schwarz	schwarz	weiß	⇒ geradeaus
schwarz	weiß	schwarz	⇒ rechts abbiegen
weiß	schwarz	schwarz	⇒ rechts abbiegen
weiß	weiß	schwarz	⇒ rechts abbiegen
weiß	schwarz	weiß	⇒ geradeaus
schwarz	weiß	weiß	⇒ links abbiegen
weiß	weiß	weiß	⇒ umdrehen

Tabelle 4.1: Labyrinth - Mögliche Konstellationen von Messwerten

Auch der kürzeste Weg durch das Labyrinth lässt sich mittels des Roboters ermitteln. Dazu kann entweder direkt in ein Array geschrieben werden oder es wird ein Stapelspeicher (Stack) implementiert, welcher ebenfalls auf einem solchen Array basiert. Hier wird kurz auf die Speicherung in einem Array eingegangen.

Sinnvoll erscheint die Definition eines zweidimensionalen Arrays mit 2 „Zeilen“. In der ersten Zeile werden die an den jeweils abgefahrenen Kreuzungen vorhandenen Abzweigungen gespeichert, in der zweiten Zeile wird die tatsächlich gewählte Richtung eingetragen. Die Reihenfolge des Abfahrens lässt sich schlicht durch die Adressierung der Felder eruieren. Um die Daten an der korrekten Stelle zu

speichern ist außerdem ein Zeiger nötig.

Um möglichst einfach mit dem Array umgehen zu können, sollten Integer-Werte verwendet werden, etwa 123 für die Möglichkeit, in alle drei Richtungen weiterzufahren, 23, wenn nur geradeaus oder nach links gefahren werden kann oder 0, wenn eine Sackgasse befahren wurde und gewendet werden muss. Kurven, an welchen nicht mehrere Alternativen zur Auswahl stehen, werden auch nicht im Array gespeichert.

Während des Abfahrens des Labyrinthes wird damit begonnen, das Array zu füllen. Es gibt für Abzweigungen nur folgende Werte: 123 (rechts & geradeaus & links), 12 (rechts & geradeaus), 13 (rechts & links), 23 (geradeaus & links). Beim zu fahrenden Weg wird immer die Reihenfolge rechts - geradeaus - links eingehalten. D.h. es wird erst dann der Weg geradeaus gewählt, wenn der Weg nach rechts in eine Sackgasse führt.

Führte der Weg in eine Sackgasse, so ist der Zeiger im Array wieder soweit nach vorne zu verschieben, bis sich eine Abzweigung findet, bei welcher noch nicht die letzte Lösung erreicht wurde. Dabei ist zu beachten, dass die eingetragene Fahrtrichtung nicht immer mit der tatsächlichen Fahrtrichtung übereinstimmen muss. Dies ist etwa dann der Fall, wenn der Roboter in einer Sackgasse umdreht und am Rückweg erneut abbiegt. Dann wird etwa in das Array „links“ eingetragen, obwohl der Roboter tatsächlich nach rechts fährt. Aus der ursprünglichen Richtung würde dies aber einem Abbiegen nach links entsprechen.

Wird das Labyrinth aus Abbildung 4.18 abgefahren, ergeben sich daraus die in Abbildung 4.19 angegebenen Zustände des Arrays. Die Länge des Arrays wurde dabei angepasst, d.h. es wurde mit einem Array mit einer „Spalte“ begonnen. Tatsächlich sollte hier mit fixen Werten gearbeitet werden, also etwa mit 30 Spalten. Abschließend kann ein neues Array erstellt werden, in welchem die finale Lösung dargestellt wird, wodurch auch ein Durchfahren des Labyrinths vom Ziel zum Start ermöglicht wird, indem die Werte umgekehrt werden (rechts wird zu links, links wird zu rechts, geradeaus bleibt unverändert).

Der Zeiger im Array zeigt immer auf die aktuell befahrene Kreuzung. Im Feld „Möglich“ werden die für die jeweilige Kreuzung möglichen Bewegungsrichtungen angegeben, also etwa „12“ für rechts und geradeaus. Im Feld „Gewählt“ werden die tatsächlichen Bewegungsrichtungen eingetragen. Im Feld „Gewählt“ wird stets mit dem niedrigsten Wert begonnen, welchen das jeweilige Feld „Möglich“ zulässt, also „2“ wenn „23“ möglich ist. Dieser Wert wird dann maximal soweit erhöht, wie es ebenfalls wieder der „Möglich“-Wert zulässt, also bis „2“ bei „12“.

Dieses Array wird während der Fahrt angelegt. Hierfür wird am besten eine eigene Methode geschrieben, auf welche in jedem Durchlauf der Rekursion zuge-

Zeiger	↓																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			</
--------	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Abbildung 4.19: Labyrinth - Beispielhafte Suche des kürzesten Weges
griffen wird. Es können jeweils passende Werte übergeben werden, da ja bekannt ist, in welche Richtung gefahren werden kann.

4.3.4 Sortieren & Suchen

Das Sortieren und Suchen von Daten stellt eine, wenn nicht sogar die zentrale Aufgabe in der Informatik dar. Es stehen diverse Algorithmen zur Verfügung, welche sich teilweise erheblich im Zeitaufwand unterscheiden. Im Lehrplan ist das Sortieren und Suchen für den dritten Jahrgang vorgesehen, weshalb auch dieses Projekt diesem Jahrgang zuzuordnen ist. Dieses Projekt sollte als Gruppenarbeit durchgeführt werden, da insbesondere das Sortieren mit großem Aufwand verbunden ist. Das Suchen könnte auch von jedem Schüler einzeln programmiert werden, allerdings wird dazu die passende Anlage benötigt.

Hier muss noch festgelegt werden, was denn überhaupt sortiert oder gesucht werden soll. Die Entscheidung fiel hier auf die wohl einfachste Variante: den Helligkeitswert von Objekten. Dazu werden Legosteine verschiedener Farben nach ihrem Helligkeitswert sortiert. Der einzig zwingend nötige Sensor ist hier also der Helligkeitssensor. Davon ist für Suche und Sortieren nur ein Exemplar nötig.

4.3.4.1 Sortieren

Der Aufbau einer Anlage zum Sortieren ist deutlich aufwändiger als bei den meisten anderen Projekten, insbesondere deshalb, da hier versucht werden sollte, eine Anlage für möglichst viele Sortieralgorithmen nutzen zu können. Abbildung 4.20 zeigt schemenhaft einen möglichen Aufbau einer solchen Sortiermaschine.

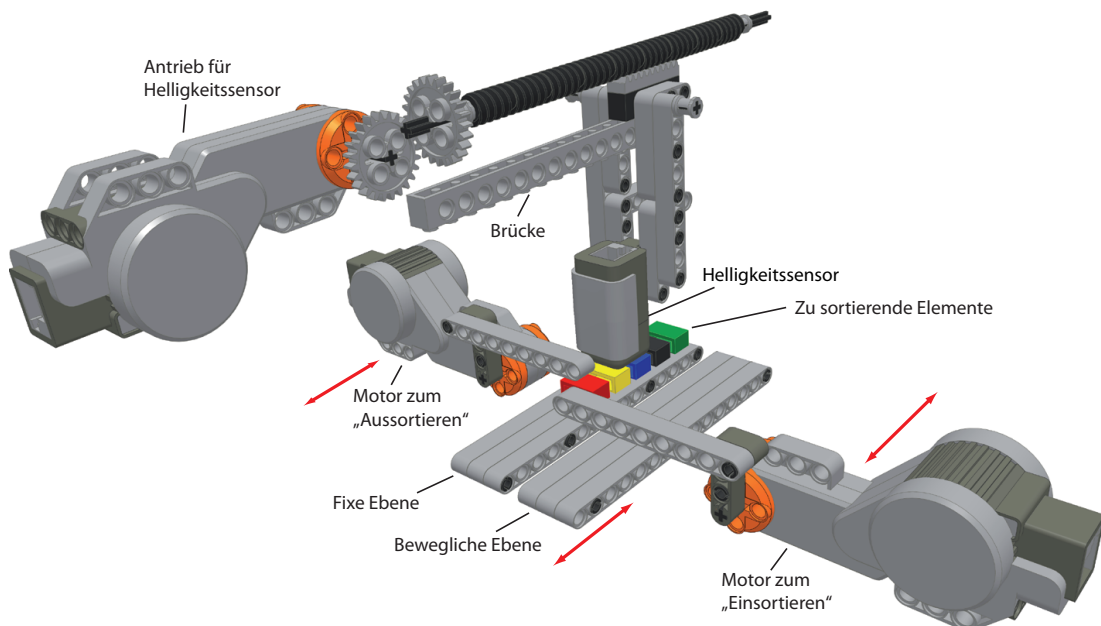


Abbildung 4.20: Sortieren - Aufbau einer Sortiermaschine

Auf der fixen Ebene werden die zu sortierenden Elemente platziert, deren Größe ebenso bekannt sein muss wie die Abstände zwischen diesen, um später den

Tausch der Positionen reibungslos abwickeln zu können. Über den zu sortierenden Elementen befindet sich der Helligkeitssensor, welcher von einer Brücke herabhängt. Mittels eines eigenen Motors kann der Sensor entlang der Brücke über die Elemente auf der fixen Ebene hinweg bewegt werden.

Zur Erklärung der weiteren Funktionsweise wird nun der Algorithmus „Selectionsort“ herangezogen. Dabei werden jeweils das erste Element und jenes Element mit dem kleinsten Wert getauscht, jeweils aus dem noch nicht sortierten Teilbereich. Mittels des Helligkeitssensors findet sich jenes Element mit dem niedrigsten Helligkeitswert, welches mit dem ersten Element des unsortierten Bereiches vertauscht werden soll. Dazu schiebt der Arm am „Motor zum Aussortieren“ diese beiden Elemente auf die bewegliche Ebene. Der Bewegungsablauf dieses Armes ist vergleichbar mit jenem einer Kurbelwelle, d.h. ein Ende wird dezentral vom Motor gedreht, das andere Ende (vergleichbar mit dem Kolben) verschiebt die zu sortierenden Elemente. Diese Konstruktion zum Aussortieren muss entlang der fixen Ebene verschoben werden können, wofür ein weiterer Motor benötigt wird.

Auf der beweglichen Ebene befinden sich nun jene beiden Elemente, welche ihren Platz tauschen sollen. Diese bewegliche Ebene sowie der Motor zum Einsortieren benötigen jeweils einen eigenen Antrieb, um sich ebenfalls parallel zur fixen Ebene und voneinander unabhängig bewegen zu können. Weiters darf zwischen den beiden Ebenen nur eine minimale Lücke auftreten, um etwa das Risiko des Verkantens beim Verschieben der Elemente zu minimieren.

Die Ebene bringt nun die beiden Elemente nacheinander an jene Stelle der Liste, an welcher sie einzuordnen sind. Der für das Einordnen zuständige Motor schiebt diese dann zurück auf die fixe Ebene und es beginnt ein neuer Durchgang.

Diese Konstruktion funktioniert mit den häufig beschriebenen Suchalgorithmen „Selectionsort“, „Insertionsort“, „Quicksort“, „Bubblesort“ oder „Mergesort“. Der Aufwand dafür kann aber stark zunehmen. So ist etwa beim Insertionsort-Algorithmus nicht immer nur ein Paar zu tauschen sondern es muss auch die restliche Liste „nach hinten“ verschoben werden, wenn ein Element neu eingeordnet wird. Dies bedeutet v.a. einen erhöhten Aufwand bei der Programmierung.

Vorteile der Verwendung von Lego Mindstorms in diesem Bereich sind etwa, dass Suchvorgänge direkt beobachtet und verglichen werden können. Weiters kann sofort festgestellt werden, ob ein Verfahren stabil³ ist oder nicht. Ginge es nur darum festzustellen, wie lange ein Suchvorgang dauert, so ließe sich dies auch auf einem PC etwa mittels der Methode „nanoTime()“ ermitteln. Werden Such-

³ Ein Sortierverfahren ist genau dann stabil, wenn die Reihenfolge gleicher Werte vor und nach dem Sortieren gleich ist

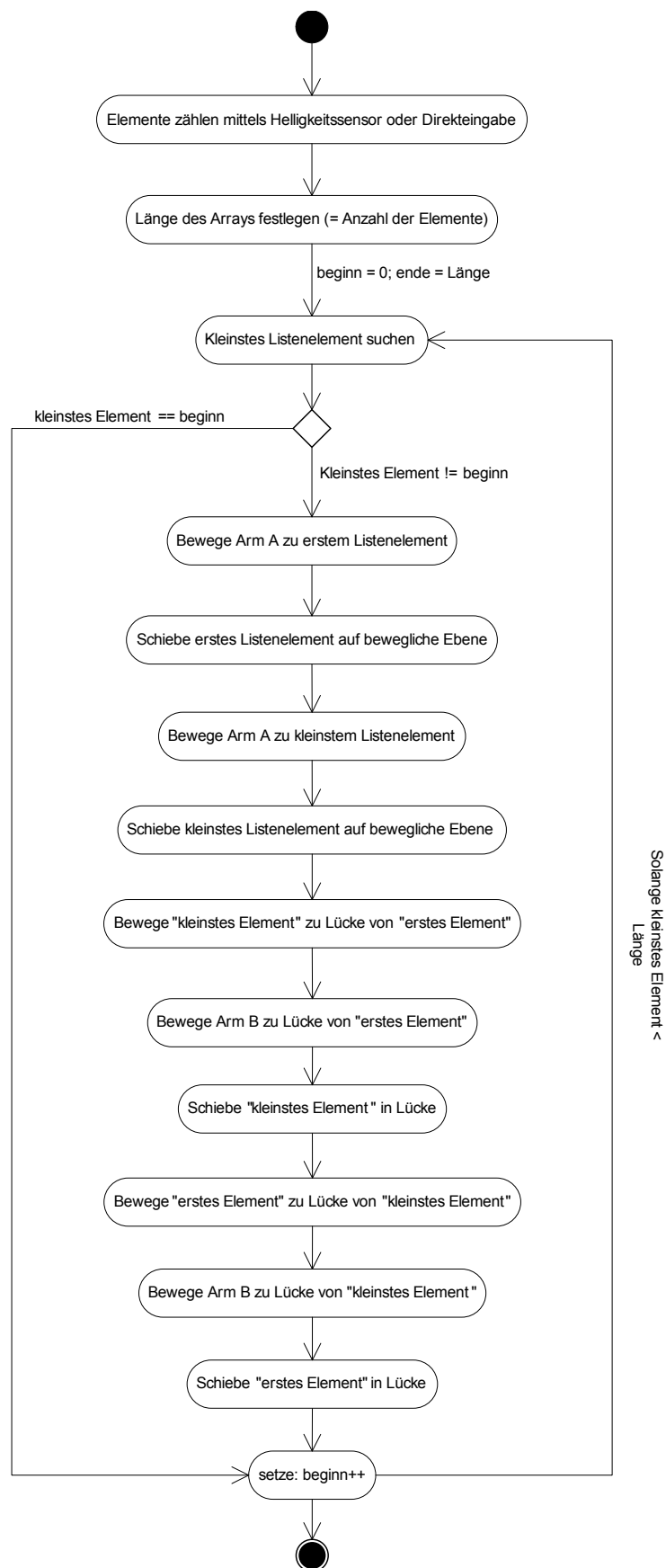


Abbildung 4.21: Sortieren - Aktivitätsdiagramm Selectionsort

vorgänge aufgrund der Ausführungsdauer verglichen, so sollten jedenfalls die Ausgangsbedingungen für jeden Algorithmus gleich sein, d.h. es muss die selbe Maschine zum Einsatz kommen. Auch LeJOS erlaubt mittels der stopwatch-Klasse das Messen der vergangen Zeit, welche dann am Display ausgegeben werden kann.

Abbildung 4.21 zeigt ein mögliches Aktivitätsdiagramm zur Sortierung mittels Selectionsort durch die in Abbildung 4.20 dargestellte Maschine. Genauere Angaben zur Programmierung werden hierzu nicht gemacht, da hier die Positionierung der Arme sowie der beweglichen Ebene im Vordergrund steht, was wieder mittels der entsprechenden Methoden der Motor-Klasse möglich ist, etwa „rotateTo()“. Als einziger Sensor kommt der Helligkeitssensor zum Einsatz, dessen Messwert etwa mittels „readNormalizedValue()“ ermittelt werden kann.

Da insbesondere der Bau einer solchen „Sortiermaschine“ mit einem gesteigerten Aufwand verbunden ist, sollte vorher geprüft werden, ob sich dieser Aufwand lohnt. Für die Integration nur eines einzelnen Sortieralgorithmus wäre von einer Umsetzung wegen des großen Zeitaufwandes abzuraten.

4.3.4.2 Suchen

Für die Suche ist keine Anpassung des Aufbaus nötig, da bei jedem Suchalgorithmus nur der Helligkeitswert des jeweiligen Steines erkannt werden muss, d.h. es werden keine Veränderungen an der Liste vorgenommen. Hier kann gut der Unterschied zwischen einer linearen [Möss01] und einer binären Suche [Möss01] demonstriert werden. Voraussetzung für die binäre Suche ist eine sortierte Liste, wohingegen die lineare Suche auf jeder beliebigen Liste ausgeführt werden kann. Es soll hier v.a. der Unterschied bezüglich der Laufzeiten demonstriert werden. Diese betragen $\mathcal{O}(n)$ (Landau-Notation, siehe [Drmo08]) für die lineare Suche und $\mathcal{O}(\log n)$ für die binäre Suche.

Der beweglich gelagerte Helligkeitssensor muss mit Hilfe eines Motors alle Elemente der Liste erreichen können. Bei der binären Suche muss die Länge der Liste bekannt sein, da ja stets die Länge des zu durchsuchenden Bereiches halbiert wird. Für die lineare Suche muss diese Länge nicht zwingend bekannt sein.

Wichtig ist hier ein genormter Abstand zwischen den einzelnen Elementen der Liste, um den Sensor möglichst genau zu diesen Elementen steuern zu können. Dafür eignet sich bei Verwendung eines einzelnen Motors die Methode „rotateTo()“.

Für den Algorithmus der linearen Suche bietet sich ein iteratives Verfahren an, d.h. es wird mit einer while-Schleife gearbeitet. Ebenso kann mit einer for-Schleife gearbeitet werden, allerdings muss dann die Länge der Liste bekannt sein. Der

Helligkeitssensor wird genau so lange in der Liste voranbewegt, bis das gesuchte Element gefunden ist. Für die binäre Suche bietet sich hingegen ein rekursives Verfahren, ähnlich der Fahrt durch das Labyrinth, an. Bei der binären Suche in der sortierten Liste wird bei jedem Durchlauf die Länge der verbleibenden Liste halbiert und das mittlere Element auf Übereinstimmung geprüft. Weist dieses Element den gesuchten Wert auf, so ist die Suche abgeschlossen. Ist der Wert größer als der gesuchte, so wird die Suche im Bereich links dieser Mitte fortgesetzt, ist er kleiner als der gesuchte, so wird die Suche im Bereich rechts der Mitte fortgesetzt. Deshalb ist es nötig, dass hier eine sortierte Liste vorliegt.

Code-Beispiele werden hier nicht angegeben, da sich solche in nahezu jeder Java-bezogenen Literatur finden und diese „nur“ an den jeweiligen Aufbau des Roboters anzupassen sind.

4.3.5 Music Machine

Dieses Projekt ist dem 1. Jahrgang der Angewandten Programmierung zuzuordnen, da hier Anweisungsfolgen, Wiederholungen oder Integer-Zahlen zu behandeln sind. Jeder Schüler bzw. in Kleingruppen zusammengeschlossene Schüler sollen dieses Projekt umsetzen, um das Programmieren von Schleifen und, bei einer speziellen Art der Music Machine, den Einsatz von Arrays und einer Suche auf diesen, zu erlernen.

Der Helligkeitssensor stellt die Basis der Music Machine dar. Gewissen gemessenen Helligkeitswerten werden bestimmte Töne zugeordnet, welche vom Lautsprecher des NXT ausgegeben werden sollen. Für den Ablauf selbst gibt es auch hier diverse Möglichkeiten. So können etwa händisch Gegenstände mit verschiedener Helligkeit über den Sensor bewegt werden, um die Töne zu erzeugen. Genauso kann ein fahrender Roboter erstellt werden, welcher die Helligkeit des Untergrundes misst und den jeweils zugeordneten Ton ausgibt. Eine gute Möglichkeit stellt ein stets am Stand rotierender Roboter dar. Unter diesem kann eine tortenförmige Scheibe platziert werden, auf welcher jedes „Tortenstück“ einen individuellen Helligkeitswert aufweist (siehe Abb. 4.22). Durch die Rotation werden die Töne nacheinander abgespielt. Dabei ergibt sich auch, wie in der Abbildung zu sehen, die Möglichkeit, die Abspieldauer des jeweiligen Tones von der Größe des jeweiligen Feldes abhängig zu machen.

Für das Rotieren des Roboters werden zwei Motoren benötigt, für welche allerdings nicht unbedingt wie sonst üblich die `pilot`-Klasse benötigt wird. Dies ist deshalb der Fall, da hier nur eine Rotation am Stand stattfinden soll. Dies kann ebenso durch die Methoden der `motor`-Klasse geschehen, indem sich die beiden Motoren schlicht mit gleicher Geschwindigkeit in entgegengesetzte Richtung dre-

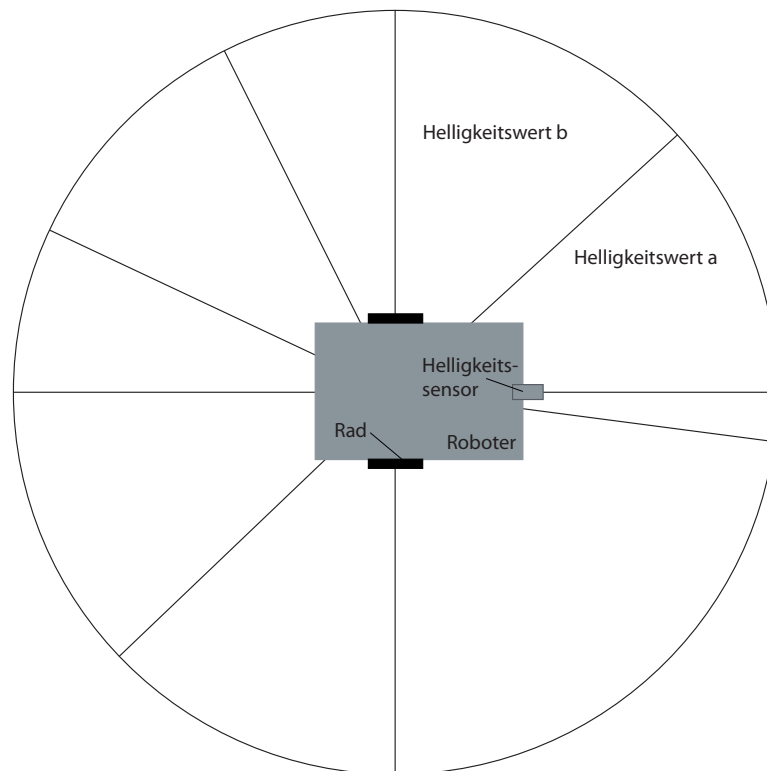


Abbildung 4.22: Music Machine - Aufbau

hen. Die Rotation sollte solange stattfinden, bis ein Abbruchkriterium erfüllt ist, also etwa bis der Escape-Knopf gedrückt wird.

Das Festlegen der Umdrehungsgeschwindigkeiten der Motoren geschieht mittels folgendem Code:

```
Motor.A.setSpeed(45);
Motor.B.setSpeed(45);
```

Hierbei wird den Motoren A und B jeweils eine Umdrehungsgeschwindigkeit von 45 Grad pro Sekunde zugewiesen. Um die Motoren nun mit dieser Geschwindigkeit in entgegengesetzte Richtung zu starten, ist folgender Code notwendig:

```
Motor.A.forward();
Motor.B.backward();
```

Die beiden Motoren drehen somit in entgegengesetzte Richtung. Dies kann mittels while-Schleife bis zum Drücken der Escape-Taste ausgeführt werden. Sollen nun die Motoren gestoppt werden, ist für beide Motoren die stop-Methode auszuführen:

```
Motor.A.stop();
Motor.B.stop();
```

Zum Abspielen der Töne muss nun die Helligkeit ermittelt werden und verschiedenen Helligkeiten müssen verschiedene Töne zugeordnet werden. Ein Wechsel

des Tones wird entweder exakt dann ausgeführt, wenn sich der gemessene Helligkeitswert ändert oder es wird etwa alle zehn Millisekunden der Wert ermittelt und mit dem vorhergehenden verglichen. Ersteres geschieht mit einer zusätzlichen Schleife, zweiteres mittels Schleife und einer sleep-Anordnung.

Um einen entsprechenden Ton abzuspielen, bietet sich die Methode „playTone()“ der sound-Klasse an. Ihr muss nur Frequenz und Dauer des zu spielenden Tones übergeben werden. Die Dauer kann etwa jeweils mit zehn Millisekunden angegeben werden. Die Frequenz ist jeweils abhängig vom ermittelten Helligkeitswert. Die Abfrage der Helligkeit erfolgt etwa mittels:

```
int hell = light.readNormalizedValue();
```

Die nach der Methode „readNormalizedValue()“ ermittelten Integer-Werte liegen alle zwischen „0“ und „1023“. Mit diesem Wert kann nun per if-Anweisungen eine Unterteilung stattfinden, bei welcher Helligkeit welcher Ton gespielt werden soll. Sinnvoll erscheint hier jedenfalls eine Aufteilung in Intervalle, d.h. es wird nicht genau einem Helligkeitswert ein Ton zugeordnet, sondern einem gewissen Intervall von Helligkeitswerten wird jeweils ein Ton zugeordnet. Ein Ausschnitt kann wie folgt aussehen:

```
if (hell >= 0 && hell < 100) {  
    sound.playTone(200, 10)  
}  
if (hell >= 100 && hell < 200) {  
    sound.playTone(400, 10)  
}
```

Die der Methode „playTone()“ übergebenen Werte geben die Frequenz (200 bzw. 400 Hz) und die Dauer (jeweils 10 Millisekunden) des zu spielenden Tones an.

Da dieses Projekt für den Informatikunterricht gedacht ist und da LeJOS seit Version 0.6 auch switch-Anweisungen unterstützt, kann dieses Projekt ebenso auf diese Weise umgesetzt werden.

Abgesehen von der bereits beschriebenen, eher einfach umzusetzenden Art der Programmierung allein mittels Schleifen, kann hier auch ein Array verwendet werden. Sinnvoll wäre hier ein zweidimensionales Array, wobei in der ersten „Zeile“ die jeweiligen Helligkeitswerte nach Größe sortiert eingetragen werden. In der zweiten Zeile werden die bei dem jeweiligen Helligkeitswert abzuspielenden Frequenzen eingetragen.

Hier muss beachtet werden, dass dieses Array laufend durchsucht werden muss. Wenn vom Helligkeitssensor ein neuer Wert retourniert wird, so muss festgestellt werden, welche Frequenz diesem zugeordnet ist, damit der entsprechende Ton ausgegeben werden kann. Das Array sollte eben deshalb nach den Helligkeitswerten sortiert werden, da somit der Algorithmus der binären Suche angewandt werden

kann. Ein solches Array, dargestellt als Tabelle, könnte etwa wie in Tabelle 4.2 aussehen.

Helligkeitswert	100	200	300	...
Frequenz[Hz]	1000	2000	3000	...

Tabelle 4.2: Music Machine - Zuordnung von Frequenzen zu Helligkeitswerten

Hier ist zu beachten, dass nicht jedem einzelnen Helligkeitswert eine individuelle Frequenz zugeordnet ist. Da der Helligkeitssensor 1024 Werte messen kann, wäre dies kaum sinnvoll umzusetzen, auch da LeJOS Arrays auf eine Größe von 511 Objekten beschränkt. Das bedeutet aber auch, dass bei dieser Art der Darstellung immer Intervalle geprüft werden müssen. Hier könnte etwa angenommen werden, dass der in Tabelle 4.2 angegebene Helligkeitswert nur die Obergrenze darstellt. Dadurch muss bei der Suche auch immer das Intervall „vor“ diesem Wert geprüft werden. Vorteil an dieser Art der Umsetzung ist neben dem Lerneffekt für die Schüler v.a. der verringerte Aufwand bei Änderungswünschen. Sollen Töne, d.h. die entsprechenden Frequenzen geändert werden, so ist dazu nur der entsprechende Wert im Array zu ändern. Die Suche selbst bleibt stets unverändert.

4.4 Netzwerktechnik

Im Unterrichtsfach Netzwerktechnik bieten sich auch ohne den Einsatz von Lego Mindstorms schon gute Gelegenheiten zum Einsatz von praxisnahen und anschaulichen Technologien. Nachdem im zweiten Jahrgang Grundlagen vermittelt wurden, kann im dritten Jahrgang mit einer Vertiefung, wie etwa der Behandlung von Client-Server-Systemen fortgefahren werden. Genau für diesen Schritt nach der Behandlung der Theorie eignet sich der Einsatz von Lego Mindstorms, um hier an praktischen, „anfassbaren“ Beispielen Netzwerke zu verwalten. Bei der Behandlung von theoretischen Modellen, wie etwa dem ISO/OSI-Schichtenmodell, ist der Einsatz von Mindstorms hingegen kaum denkbar.

4.4.1 Toilettenmanager

Dieses Projekt ist dem 3. Jahrgang zuzuordnen, in welchem die Behandlung von Client-Server-Systemen vorgesehen ist.

Schon öfters war er Thema von Rundfunksendungen oder Zeitungs- bzw. Zeitschriftenartikeln [Focu08]: der Zustand von Schultoiletten. Sie befinden sich häufig in einem desolaten Zustand, u.a. deshalb, weil sie absichtlich verstopft oder als „Raucherkabine“ zweckentfremdet werden. Teilweise werden sie auch von außen

abgeschlossen, um eine bestimmte Kabine quasi zu reservieren. Bei diesem Projekt sollen zumindest Teile dieser Umstände durch ein Client-Server-System verhindert werden. Die Umsetzung dieses Projekts zielt klarerweise auf die Schüler, eine reine Vorführung wäre sinnlos. Eine solche Umsetzung könnte etwa in Form einer großen Gruppenarbeit an Projekttagen stattfinden. Es werden auf jeden Fall mehrere Mindstorms-Bausätze benötigt: eine zentrale Einheit und jeweils eine Einheit pro Toilette.

Der Toilettenmanager ist nur für Kabinen gedacht, d.h. nicht für den Einsatz an offenen Pissoir-Plätzen, da in diesen auch der größte Schaden entstehen kann. Am Toiletteneingang soll sich eine zentrale Einheit (Server) in Form eines NXT befinden, welche anzeigt, wie viele Toiletten momentan frei oder besetzt sind. Weiters sollen die Nummern jener Kabinen angezeigt werden, welche schon „zu lange“ besetzt sind. Dadurch ist durch eine berechtigte Person eine Kontrolle der jeweiligen Kabinen möglich.

Dies bedeutet, dass in den einzelnen Kabinen kontrolliert werden muss, ob und wie lange diese schon als besetzt markiert ist. Dies ist am einfachsten dadurch festzustellen, indem gemessen wird, ob und wie lange die jeweilige Tür abgeschlossen ist. Einfache Möglichkeiten dazu bietet der Ultraschallsensor. Mittels diesem kann etwa die Drehung des Schlosses erkannt werden. Werden Schlösser mit farblicher Anzeige des Drehzustandes verwendet, so könnte zur Kontrolle der Kabine auch der Helligkeitssensor verwendet werden.

In Abbildung 4.23 ist die Funktionsweise bei Verwendung des Ultraschallsensors skizziert. Dazu wird am Türstock der Ultraschallsensor montiert, etwas über oder unter dem Zentrum des an der Tür montierten Schlosses. Wichtig ist hier, dass

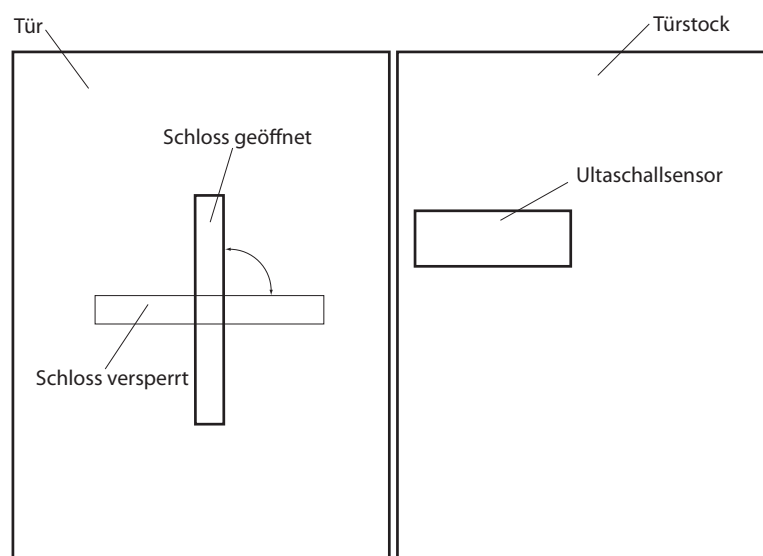


Abbildung 4.23: Toilettenmanager - Abschluss-Kontrolle

der Ultraschallsensor bei geöffnetem und versperrtem Schloss jeweils verschiedene Werte liefert. Dies sollte dadurch geschehen, dass durch die Drehung des normalerweise verwendeten Drehschlusses ein Hindernis vor den Sensor tritt, womit die gemessene Distanz geringer wird. Dadurch lässt sich erkennen, dass die Kabine nun besetzt ist. Ist diese Methode so in der jeweiligen Lokalität nicht praktikabel, so kann auf den Einsatz der Helligkeitssensoren zurückgegriffen werden. Die wohl einfachste Möglichkeit stellt außerdem der Einsatz des Druckkontaktsensors dar, welcher direkt durch das Schließen der Tür ausgelöst werden kann. Am aufwändigsten aber wohl auch schwierigsten zu sabotieren wäre eine Lösung mittels der Mindstorms Motoren. Diese könnten etwa in den Türrahmen integriert werden und somit die aktuelle Ausrichtung der Tür auslesen. Die hauptsächlich dazu benötigten Methoden sind „getTachoCount()“ für das Auslesen des Motorstandes als Gradzahl, „isPressed()“ für das Auslesen des Zustandes des Druckkontaktsensors als booleon-Wert und „getDistance()“ für das Abfragen der Distanz vom Ultraschallsensor.

All diese Methoden wurden bisher schon behandelt. Es findet ein bestimmtes auslösendes Ereignis statt, also etwa das Messen einer Distanz unter einem festgelegten Grenzwert oder das Drücken des Druckkontaktsensors, womit dann die jeweilige Toilette sowohl als besetzt markiert wird, als auch eine Stoppuhr zu laufen beginnen soll. Erreicht die Stoppuhr einen festgelegten Grenzwert, so wird mittels Bluetooth ein Alarm an die zentrale Einheit gesendet. Auch der Besetzt-Status ist der zentralen Einheit zu melden, damit diese Toilettenbesuchern direkt zeigen kann, welche Toiletten noch frei sind. Neben dieser Kontrolle der einzelnen Toiletten wäre auch denkbar, den Lautstärkepegel an einer zentralen Stelle mittels des dafür vorgesehenen Sensors zu überprüfen. Sinnvoll wäre dies etwa um zu überprüfen, ob über längere Zeit ein erhöhter Lautstärkepegel herrscht, was auf gewalttätige Auseinandersetzungen hindeuten kann.

Kernpunkt dieses Projekts ist die Kommunikation der Clients mit dem Server mittels Bluetooth. Dabei ist darauf zu achten, dass die einzelnen Clients jeweils in Reichweite des Servers liegen. Diese Reichweite ist stark abhängig von den jeweiligen äußeren Bedingungen, liegt aber bei einer Verbindung NXT↔PC bei etwa 25 Metern [Geba07]. Damit Bluetooth verwendet werden kann, muss dieses auf allen NXT im LeJOS-Menü aktiviert werden. Weiters muss die Sichtbarkeit (Visibility) aktiviert werden.

Wichtig ist nun, dass eine Verbindung generell, ob Bluetooth oder USB, aus einem sog. „receiver“ und einem sog. „initiator“ besteht. Der „receiver“ wartet auf den Verbindungsaufbau des „initiators“. Damit diese Verbindung zustande kommen kann, muss sich der „receiver“ im Wartezustand befinden. Sobald

die Verbindung hergestellt ist, können sowohl „initiator“ als auch „receiver“ die Verbindung zum Übermitteln von Daten verwenden. Wichtig ist, dass vor dem Verbindungsaufbau die jeweiligen „receiver“ manuell in die Liste der bekannten Bluetooth-Geräte (devices) des „initiators“ hinzugefügt werden, was über den entsprechenden Menüeintrag geschieht [leJO08].

Für das Beispiel des Klomanagers bedeutet dies nun, dass es am sinnvollsten und v.a. effektivsten ist, den Server als „initiator“ und die Clients als „receiver“ festzulegen. Dadurch müssen nur einmal alle Clients in die „Devices“-Liste des Servers hinzugefügt werden anstatt bei jedem einzelnen Client den Server hinzuzufügen. Die Aufforderung zum Verbindungsaufbau geht vom Server aus, da er der „initiator“ ist und nur er die Kommunikationspartner „kennt“. Eine Möglichkeit besteht hier darin, mittels einer Schleife nacheinander jeden Client nach dem aktuellen Zustand zu fragen.

4.4.2 Parkplatzmanagement

Dieses Projekt ist ebenfalls dem 3. Jahrgang zuzuordnen, für welchen die Behandlung von Client-Server-Systemen gefordert wird. Dabei soll die Verfügbarkeit von Parkplätzen ermittelt werden. Der Zustand „frei“ oder „belegt“ wird dabei an eine zentrale Einheit gemeldet. Ein vorbeifahrender Roboter kann dann abfragen, wo sich ein freier Parkplatz befindet und in diesen mit zur Verfügung stehenden Sensoren einparken.

Konkret gibt es auch hier wieder viele Möglichkeiten der Umsetzung, die Entscheidung fiel hier auf eine Unterscheidung der Parkplätze durch Farben. Das bedeutet, dass auf der Fahrspur des Roboters vor jedem Parkplatz eine jeweils eindeutige Farbmarkierung angebracht ist. Bevor der einzelne Roboter zu einem Parkplatz fährt, baut dieser eine Verbindung zum Server auf und lässt sich die Farbe des nächsten freien Parkplatzes schicken. Anschließend muss der Parkplatz klarerweise als besetzt markiert werden, was durch eigene NXT-Einheiten und Sensoren (etwa Ultraschallsensoren zur Prüfung auf einen gewissen Mindestabstand) an jedem einzelnen Parkplatz geschehen kann.

Sinnvoller, schneller umzusetzen und sowohl im Unterricht als auch im realen Leben deutlich günstiger ist allerdings ein Verzicht auf diese zusätzlichen NXT-Einheiten an jedem einzelnen Parkplatz. Die „Parkplatz besetzt“-Meldung kann dabei vom Server selbst getätigt werden, sobald einem Fahrzeug ein Parkplatz zugewiesen wurde. Bei Bedarf kann hier zusätzlich noch eine Rückmeldung stattfinden, ob das Fahrzeug tatsächlich am gewünschten Ort steht, etwa indem der komplette Parkplatz in der jeweiligen Farbe markiert ist. Dann kann das einparkende Fahrzeug eine „Besetzt“-Nachricht mit der jeweiligen Farbe des Parkplatzes an

den Server schicken, sobald der Parkplatz tatsächlich erreicht ist. Irrtümer durch fehlerhaftes Einparken wären somit ausgeschlossen. Ein entsprechender Aufbau könnte dann wie in Abbildung 4.24 aussehen.

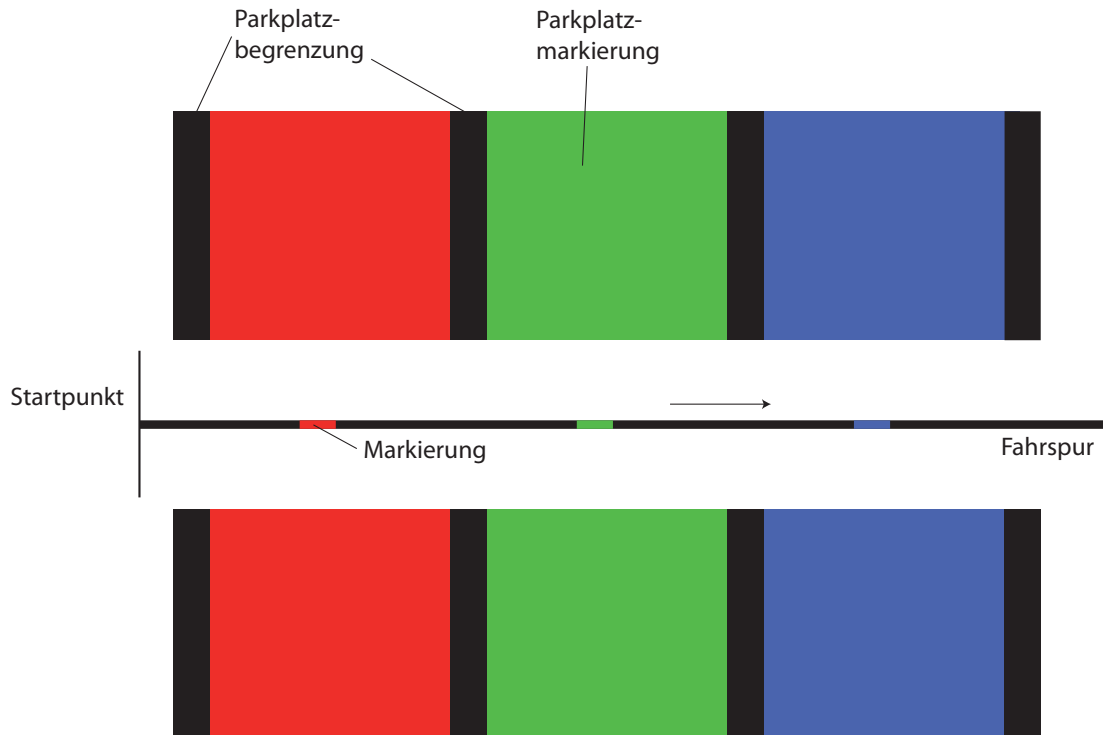


Abbildung 4.24: Parkplatzmanager - Aufbau

Die Meldung „Parkplatz frei“ wird vom jeweiligen Fahrzeug an den Server geschickt, da nur dieses Fahrzeug selbst „weiß“, wann es wieder losfährt. Dazu muss das jeweilige Fahrzeug entweder den vom Server zugewiesenen Farbwert speichern und diesen dann mit der Markierung „frei“ zurückschicken, oder der jeweilige Wert wird nicht gespeichert und beim Verlassen schlicht der Wert der Markierung (vorausgesetzt der Parkplatz ist markiert) gelesen und an den Server geschickt.

Das Einparken selbst erledigt jede Einheit autonom, d.h. der Server leistet dabei keine Hilfe. Dazu ist neben zwei Motoren, welche jeweils ein Rad antreiben und somit eine Steuerung ermöglichen, auch ein Ultraschallsensor sinnvoll, mittels welchem die Distanz zum Ende des Parkplatzes gemessen werden kann. Dieser Sensor kann mittels eines Motors auch drehbar gelagert werden, um ihn sowohl für die Vorwärts- als auch für die Rückwärtsfahrt nutzen zu können. Weiters ist an den Fahrzeugen der angesprochene Helligkeitssensor nötig, um die Farbe des jeweiligen Parkplatzes erkennen zu können. Am Server wird kein Sensor benötigt.

Die jeweiligen Roboter basieren auf dem Prinzip des Line-Followers. Sie folgen so lange einer Linie, bis eine Markierung erreicht ist. Diese Markierung sollte auf

der Linie liegen, damit kein zweiter Helligkeitssensor benötigt wird. Das bedeutet aber auch, dass der Line-Follower nicht nur etwa einer schwarzen Linie folgen darf, sondern auch etwa einer grünen oder roten Linie folgen muss, da er sonst bei der jeweiligen Markierung vom Kurs abkommen würde.

Das Einparken ist dabei nicht auf eine Seite der Linie beschränkt. D.h. es können sich auf beiden Seiten der Linie Parkplätze befinden, welche dann mit einer entsprechenden Anweisung des Servers vom jeweiligen Fahrzeug angesteuert werden können. Eine solche Anweisung muss aus einer Angabe von Farbe und Seite des zu wählenden Parkplatzes bestehen.

Ein wichtiger Punkt ist hier die Speicherung der Parkplatzbelegung durch den Server. Dafür eignet sich wohl am besten ein zweidimensionales Array, vergleichbar mit einer Tabelle. Erst durch solch eine Speicherung kann die Belegung eines Parkplatzes geändert werden oder der nächstgelegene Parkplatz gesucht werden. Die erste Dimension des Arrays entspricht dabei der Anzahl der Parkplätze bzw. bei Einparken beidseits der Linie der Hälfte der Parkplätze. Die zweite Dimension entspricht der Anzahl der zu speichernden Informationen. Tabelle 4.3 liefert dazu ein Beispiel. In diesem Fall entspricht die zweite Dimension der Zahl vier.

Distanz zum Start	Markierungsfarbe	Belegung links	Belegung rechts
1	rot	0	0
2	blau	1	1
3	gelb	1	0
4	grün	0	0
5	grau	1	0

Tabelle 4.3: Parkplatzmanager - Beispiel eines zweidimensionalen Arrays

Wie dieses Beispiel zeigt, ist es sinnvoll, die Parkplätze aufsteigend nach ihrer Distanz zum Startpunkt in das Array einzureihen. So kann jedem Fahrzeug durch schlichte lineare Suche (siehe Abschnitt 4.3.4.2) der nächstliegende Parkplatz zugewiesen werden. An nächster Stelle folgt die Farbe der Markierung des jeweiligen Parkplatzes sowie der zugehörigen Fahrbahnmarkierung, damit der Server einem Fahrzeug die Farbe des zu wählenden Parkplatzes mitteilen kann. Die letzten beiden Spalten stehen für die Markierung der Belegung des linken und rechten Parkplatzes zur Verfügung. Wird nur auf einer Seite der Linie geparkt, so wird hier eine Spalte weniger benötigt. Die Belegung eines Parkplatzes kann etwa durch Eintragen des Wertes „1“ am entsprechenden Ort markiert werden.

Die angegebene Tabelle stellt hier allerdings nur ein Beispiel dar. Der Mindstorms-Sensor liest Helligkeits-Werte in Form von Integer-Zahlen aus. Dadurch lässt sich auch leicht ein Integer-Array herstellen, also etwa „int[x][y]“, in welchem

ohne Umwandlungen nach Zahlen gesucht werden kann, was in einem String-Array eben nicht ohne weiteres möglich wäre.

Fordert ein Roboter einen Parkplatz an, so muss vom Server der erste (nächstgelegene) Parkplatz mit der Belegung „0“ (d.h. „frei“) gesucht werden (wohl mittels doppelter while-Schleife). Sobald dieser gefunden ist, werden entsprechende Helligkeit und evtl. die entsprechende Seite an das Fahrzeug gesendet und im Array wird der Wert auf „belegt“ geändert. Verlässt das Fahrzeug den Parkplatz wieder, so liest es mittels Infrarotsensor die Farbe des Parkplatzes aus und sendet diese an den Server. Beim Parken beidseits der Linie sollten die Fahrzeuge jedenfalls speichern, auf welcher Seite der Linie sie parken. Dies für das Senden der „Parkplatz frei“-Nachricht jeweils neu zu ermitteln wäre nur mittels neuer Hilfsmittel wie etwa einem Ultraschallsensor möglich.

Bei einem Aufbau der Testumgebung wie in Abbildung 4.24 folgt der Roboter ab dem Startpunkt so lange der Linie, bis er jene Markierung auf der Linie erreicht, welche genau die Farbe hat, welche ihm vom Server zugewiesen wurde. Die Markierung stellt dar, dass sich auf gleicher Höhe die mit eben dieser Farbe markierten Parkplätze befinden. Zusätzlich sind auch die Parkplätze selbst in der jeweiligen Farbe markiert, um auch eine Kontrolle durchführen zu können, ob der jeweilige Roboter tatsächlich auf dem zugewiesenen Platz hält.

Wie beim vorherigen Beispiel des Toilettenmanagers (Abschnitt 4.4.1) wird bei der Programmierung wieder ein „initiator“ und ein „receiver“ benötigt. Hier sollte allerdings genau umgekehrt vorgegangen werden, d.h. der Server fungiert als „receiver“ und die Clients als „initiators“. Das bedeutet, dass der Server in die „Devices“-Liste aller Clients aufgenommen werden muss. Dies ist deshalb sinnvoll und realistisch, da ein Parkplatzverwaltungssystem nicht alle möglichen Clients kennen kann, welche jemals das System verwenden werden. Somit kann sich jeder beliebige Teilnehmer, welcher den Server kennt, einen Parkplatz zuweisen lassen.

4.5 Fächerübergreifende Projekte

4.5.1 Projekte und Projektmanagement

Für den Unterrichtsgegenstand „Projekte und Projektmanagement“ werden Themen wie Teamarbeit, Gruppendynamik oder Planung und Kontrolle von Leistungen gefordert [Bund06]. Neben der theoretischen Behandlung der jeweiligen Gebiete sollen diese auch anhand von praktischen, fächerübergreifenden Projekten behandelt werden. Genau an diesem Punkt setzen die folgenden Projekte an.

4.5.1.1 Pflichtenheft & Meilensteine & Gruppenarbeit

Für den dritten Jahrgang werden u.a. Methoden zur Erstellung des Pflichtenheftes gefordert [Bund06]. Aber auch Meilensteine, welche vom Lehrplan nicht ausdrücklich verlangt werden, lassen sich mittels Lego Mindstorms gut in den Unterricht einbauen.

Ein Pflichtenheft könnte rein theoretisch zu jeder Programmieraufgabe erstellt werden. Es beschreibt die Anforderungen an das Produkt, welche vom Auftraggeber ausgehen (also etwa der Lehrkraft), in den Worten des Produzenten (also etwa des Schülers). Tatsächlich sinnvoll erscheint die Erstellung solcher Pflichtenhefte im Unterricht allerdings nur bei aufwändigeren Projekten, bei welchen Missverständnisse auftreten könnten. Wie auch im Lehrplan gefordert muss dies sogar fächerübergreifend stattfinden, da ein Pflichtenheft ja nicht ohne ein Projekt erstellt werden kann. Es bieten sich dafür nahezu alle in dieser Arbeit angeführten Programmierbeispiele an, welche von Schülern zu programmieren sind. Die Lehrkraft kann dabei bestimmte Einschränkungen treffen, um Aufwand und Schwierigkeitsgrad zu steigern. Beim Sortieren und Suchen könnte etwa eine maximale Größe, ein bestimmter Algorithmus oder eine maximale Sortier- bzw. Suchdauer festgelegt werden. Ein weiteres Beispiel wäre der Line Follower, bei welchem die Lehrkraft festlegen kann, welche Sensoren und in welcher Anzahl diese verwendet werden dürfen, ob der Roboter auch einer unterbrochenen (strichlierten) Linie folgen soll und dergleichen.

Die ebenfalls vom Lehrplan geforderte Gewichtung der Phasen wird durch das Setzen von Meilensteinen erreicht. Die Schüler sollen so sehen, wie viel Zeit etwa das Bauen des nötigen Modells und die Programmierung der nötigen Software benötigt und wie groß die Gefahr ist, bei den gesetzten Zielen in Verzug zu geraten. Je nach Größe des Projekts muss natürlich die Anzahl der Meilensteine angepasst werden. Eine Setzung mehrerer Meilensteine für den Bau eines Roboters zur Längenmessung scheint wenig sinnvoll. Sie machen v.a. dann Sinn, wenn mehrere Personen bzw. Gruppen parallel an Teilbereichen einer Aufgabenstellung arbeiten und zu einem bestimmten Zeitpunkt ein gewisser Status erreicht werden soll.

Die Erstellung von Pflichtenheft und Meilensteinen im Unterricht sind nur dann wirklich sinnvoll, wenn in Gruppen gearbeitet wird. Eine Einzelperson kann etwa Meilensteine nach hinten verschieben, ohne dabei mit anderen Terminen, abgesehen von einem Abgabetermin, zu kollidieren. Auch Aufgaben aus dem Pflichtenheft könnten theoretisch vernachlässigt werden, die einzelne Person müsste sich bezüglich der zu erwartenden schlechteren Bewertung durch die Lehrkraft nur gegenüber sich selbst verantworten.

Bei einer Gruppenarbeit dagegen entsteht eine komplett andere Dynamik, es ergeben sich schnell Konflikte, welche gelöst werden müssen, um zum Erfolg zu kommen.

Als im Lehrplan nicht vorgesehene Erweiterung könnten etwa auch Gantt-Diagramme erstellt werden. Diese sind ebenfalls bei besonders aufwändigen Projekten mit mehreren Teilnehmern sinnvoll, da hier klar Zuständigkeiten und der Zeitpunkt des geplanten Abschlusses (der Teilbereiche) ersichtlich sind und die Planung erlernt werden kann.

4.5.1.2 Simulierte Abnahme durch Kunden

Die simulierte Abnahme durch den Kunden ist eine Fortführung des vorherigen Projekts. Diese ist ebenfalls dem 3. Jahrgang zuzuordnen, da hier auch die Behandlung von Gesprächs- und Verhandlungsführung gefordert wird.

Dies ist deshalb eine Fortführung, da sich eine Abnahme am besten mit einem tatsächlich vorhandenen Produkt simulieren lässt. Eine Gruppe soll hier jene Aufgabe erfüllen, welche durch das Pflichtenheft vorgegeben ist und welche durch die Lehrkraft abgenommen wird. Dabei kann überprüft werden, inwieweit die einzelnen Punkte des Pflichtenheftes tatsächlich durch das von den Schülern abgegebene Produkt erfüllt werden. Werden bestimmte Punkte nicht erfüllt, so kann die Lehrkraft etwa eine Nachbesserung fordern oder auch direkt eine schlechtere Bewertung vergeben.

4.5.2 Betriebswirtschaft

Ab dem zweiten Jahrgang kann die Betriebswirtschaft mit jedem beliebigen anderen Unterrichtsfach gekoppelt werden. Sinn ist die Planung in einer arbeitsteiligen Gesellschaft, d.h. die Schüler müssen die Arbeit, sowohl beim Bau des Roboters als auch bei dessen Programmierung, untereinander abstimmen. Dazu ist klarerweise Gruppenarbeit nötig.

Eine weitere Möglichkeit, betriebswirtschaftliche Aspekte in Projekte mit einfließen zu lassen, ist eine Belegung aller Bauteile mit bestimmten „Kosten“. Die Aufgabe der Schüler besteht dann darin, einen Roboter mit minimalen Kosten zu fertigen. Weiters kann natürlich auch die Arbeitszeit der Schüler mit Kosten belegt werden.

4.5.3 Computer Aided Design

Ebenfalls begleitend zu jedem Projekt kann die rechnerunterstützte Konstruktion oder Computer Aided Design (CAD) eingesetzt werden. Dabei wird die zu bauen-

de Hardware zuerst am Computer virtuell konstruiert. Sinnvoll ist dies zum einen zum Erlernen des CAD, einem wichtigen Bestandteil vieler Konstruktionsbereiche der Industrie. Zum anderen kann dadurch auch Zeit gespart werden, wenn dann bekannt ist, welche Teile für ein funktionierendes Modell benötigt werden und wie diese zusammengebaut werden müssen.

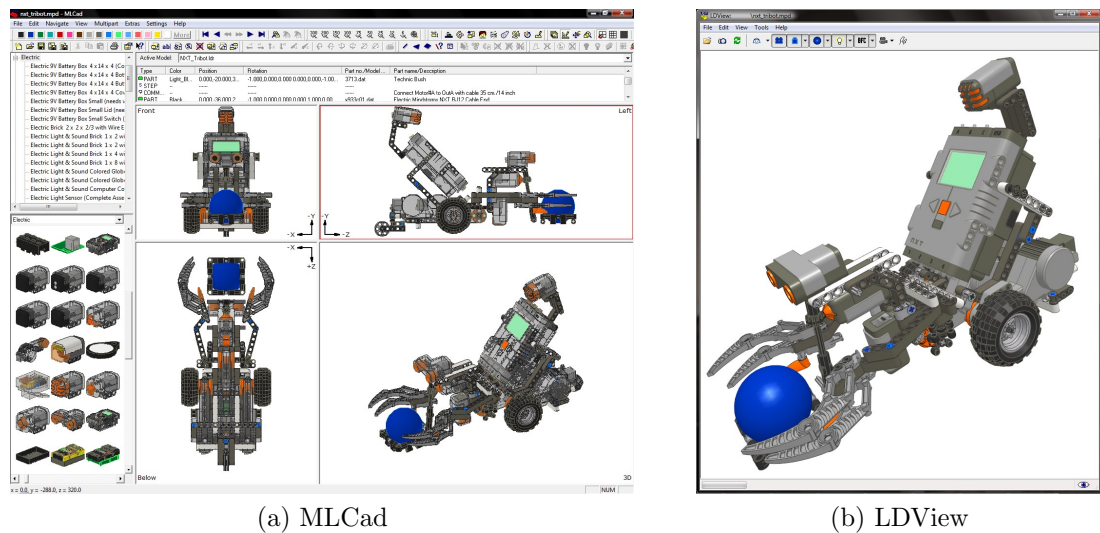


Abbildung 4.25: Computer Aided Design

Es stehen eigene Programme und Bibliotheken für CAD mit Legoteilen zur Verfügung. So wurde etwa für die Erstellung aller entsprechenden Modelle dieser Arbeit, etwa Abbildung 4.9 zum Aufbau des Getriebes, die Software MLCad⁴ (siehe Abb. 4.25a) verwendet. Für das bloße Darstellen von Modellen eignen sich Betrachtungsprogramme wie LDView⁵ (siehe Abb. 4.25b), mittels welchem auch die Grundlagen der für diese Arbeit benötigten Abbildungen erstellt wurden.

⁴ <http://www.lm-software.com/mlcad/>

⁵ <http://ldview.sourceforge.net>

5 Zusammenfassung und Ausblick

Ziel dieser Arbeit war das Aufzeigen der heute bestehenden Möglichkeiten, Lego Mindstorms in Kombination mit der Java-Firmware LeJOS im Unterricht einzusetzen. Dargestellt wurde hier sowohl die komplette Umstellung eines Roboters von der Original-Firmware auf LeJOS als auch der Einsatz eben dieser.

Bei der Beschreibung von Unterrichtsprojekten wurde primär nach Anforderungen des Lehrplanes vorgegangen, um so eine möglichst hohe Wahrscheinlichkeit der Umsetzung zu erreichen. Bei Bedarf ließen sich zu nahezu jedem Unterrichtsgegenstand Projekte für die Integration mittels LeJOS eruieren.

Eine Umsetzung im Unterricht erfordert neben den fachspezifischen Kenntnissen auch Kenntnisse der Programmiersprache Java, sowohl bei den Lernenden als auch bei den Lehrenden. An dieser Stelle liegt eine der Schwächen: das Durchschnittsalter der Lehrkräfte in Österreich ist hoch [Pres08], bis zum Jahr 2019 werden etwa 50% dieser den Ruhestand antreten [Stan08]. Hier stellt sich die Frage, wie groß die tatsächliche Bereitschaft jener Lehrkräfte ist, welche mit der Programmierung bisher nicht in Berührung kamen, eben diese nun zu erlernen. Für nachrückende junge Lehrkräfte sollte der Einsatz solcher, den Unterricht erweiternde, Hilfsmittel hingegen selbstverständlich sein.

Eine weitere Voraussetzung für den Unterricht mit Mindstorms-Robotern sind natürlich die Roboter selbst sowie Computer, im Idealfall Notebooks, um Programme erstellen sowie auf den Roboter transferieren zu können. Hier muss sowohl entschieden werden, welche Quote an Robotern pro Schüler (z.B. 0,5 für zwei Schüler pro Roboter) erreicht werden soll sowie von wem diese anzuschaffen sind, also von der jeweiligen Schule oder den Schülern bzw. deren Eltern. In jedem Fall entstehen nicht zu verachtende Kosten bei einem derzeitigen¹ Preis des Lego Mindstorms NXT (8527) von knapp 260 €.

Positiv fällt hier jedoch ins Gewicht, dass etwa Notebooks oder Subnotebooks in der Anschaffung stetig günstiger werden und somit die Möglichkeiten erweitern. Weiters bieten Java und Eclipse eine Umgebung, mittels welcher ohne Umwege mit dem NXT kommuniziert werden kann. Vergleicht man weiters den aktuellen Preis des Mindstorms NXT mit jenem von für den Unterricht geeigneten Robotern

¹ Stand: 16. April 2009

zu Beginn der 1990er Jahre, welcher bei bis zu 5000 \$ lag [Ande06], so lässt sich auch bezüglich des Preises ein positives Fazit ziehen.

Abschließend fällt positiv auf, dass durch die Einführung des Bausatzes WeDo [LEGO08] durch Lego mit Beginn des Jahres 2009 zukünftig Schüler schon deutlich früher mit den Bereichen Robotik und Informatik in Berührung kommen können. Dieser Bausatz ist auf Schüler im Alter zwischen sieben und elf Jahren ausgerichtet und wurde speziell für den Einsatz im Unterricht konzipiert. Dies könnte auch den Weg für einen deutlich breiteren und professionelleren Einsatz von Lego Mindstorms im Unterricht ebnen.

Programmcodeverzeichnis

2.1	Beispielcode - Anzeige von „Hello World“	16
4.1	Längenmessung - Berechnung des zurückgelegten Weges	23
4.2	Domino-Strecke - Vorwärtsbewegung	27
4.3	Domino-Strecke - Kurvenfahrt nach rechts	29
4.4	Domino-Strecke - Abstandsprüfung nach rechts	32
4.5	Geschwindigkeit & Beschleunigung - Konstante Geschwindigkeit .	36
4.6	Getriebe - Gangwechsel	46
4.7	Neigungssensor - Lageabfrage & Einleitung der Schaltung	47
4.8	Labyrinth - Rekursives Suchen eines Weges	66

Abbildungsverzeichnis

2.1	Button zum Erreichen des Firmware-Upload-Modus	15
4.1	Längenmessung - Aktivitätsdiagramm	23
4.2	Längenmessung - Aufbau	24
4.3	Domino-Strecke - Abstand der Steine - Profil	26
4.4	Domino-Strecke - Aktivitätsdiagramm	27
4.5	Domino-Strecke - Mindestabstand der Steine in Kurven	30
4.6	Domino-Strecke - Kurvenfahrt nach rechts	34
4.7	Überholvorgang - Aktivitätsdiagramme	40
4.8	Neigungssensor - Aufbau	43
4.9	Getriebe - Aufbau	45
4.10	Startrampe - Aufbau	48
4.11	Zentrifuge - Aktivitätsdiagramm	51
4.12	Zentrifuge - Aufbau	52
4.13	Wellengenerator - Aufbau Typ 1	53
4.14	Wellengenerator - Aufbau Typ 2	54
4.15	Fallbeschleunigung - Aufbau	55
4.16	Schalldruckpegel - Aktivitätsdiagramm	58
4.17	Line Follower - Aktivitätsdiagramm	62
4.18	Labyrinth - Mögliches Aussehen [Tech08]	65
4.19	Labyrinth - Beispielhafte Suche des kürzesten Weges	70
4.20	Sortieren - Aufbau einer Sortiermaschine	71
4.21	Sortieren - Aktivitätsdiagramm Selectionsort	73
4.22	Music Machine - Aufbau	76
4.23	Toilettenmanager - Abschluss-Kontrolle	79
4.24	Parkplatzmanager - Aufbau	82
4.25	Computer Aided Design	87

Literaturverzeichnis

- [Ande06] SCOTT D. ANDERSON und FRANK KLASSNER: *LEGO MindStorms: not just for K-12 anymore*. Robotics & Automation Magazine, IEEE, 10(2):12–18, Juni 2006.
- [Bode06] PHILIPP BODEWIG und CHRISTOPH WEILER: *Labyrinth-Roboter mit LEGO Mindstorms*.
<http://pille.iwr.uni-heidelberg.de/~legolab2/downloads/dokumentation.pdf>, August 2006.
- [Bund06] BUNDESMINISTERIUM FÜR UNTERRICHT, KUNST UND KULTUR: *Lehrplan der höheren Lehranstalt für Informationstechnologie*.
http://www.htl.at/fileadmin/content/Lehrplan/HTL_Informationstechnologie.pdf, August 2006.
- [Bück07] OLIVER BÜCKER: *Robotik, Skript zum Kurs „Programmierung eingebetteter Systeme“*.
www.fz-juelich.de/jsc/math/matse/materials/applications/robotik/script/Robotik.pdf, 2007.
- [Cont07] CHRISTOPHER CONTINANZA und FRANK KLASSNER: *Mindstorms without robotics: an alternative to simulations in systems courses*. In: J. D. DOUGHERTY, SUSAN M. HALLER, SUSAN H. RODGER und INGRID RUSSELL (Herausgeber): *SIGCSE*, Seiten 175–179. ACM, 2007.
- [Drmo08] MICHAEL DRMOTA, BERNHARD GITTENBERGER, GÜNTHER KARIGL und ALOIS PANHOLZER: *Mathematik für Informatik*. Heldermann Verlag, Lemgo, 2. Auflage, 2008.
- [Egge01] THOMAS W. EGGERS, BARRY S. FAGIN und LAURENCE D. MERKLE: *Teaching computer science with robotics using Ada/Mindstorms 2.0*. In: *SIGAda '01: Proceedings of the 2001 annual ACM SIGAda international conference on Ada*, Seiten 73–78, 2001.

- [Eguc04] AMY EGUCHI und ELIZABETH SKLAR: *RoboCupJunior - Four Years Later*. In: DANIELE NARDI, MARTIN RIEDMILLER, CLAUDE SAMMUT und JOSÉ SANTOS-VICTOR (Herausgeber): *RoboCup*, Band 3276 der Reihe *Lecture Notes in Computer Science*, Seiten 172–183. Springer, 2004.
- [Ehma06] MATTHIAS EHMANN: *Algorithmen und Programmierung: Einsatz von LEGO Mindstorms und Squeak im Informatikunterricht*. http://did.mat.uni-bayreuth.de/aktuelles/db/225/Algorithmen_und_Programmierung.pdf, Februar 2006.
- [Flow02] THOMAS R. FLOWERS und KARL A. GOSSETT: *Teaching problem solving, computing, and information technology with robots*. In: *Journal of Computing Sciences in Colleges*, Band 17, Seiten 45–55. Consortium for Computing Sciences in Colleges, USA, 2002.
- [Focu08] FOCUS SCHULE: *Schultoiletten - Es stinkt zum Himmel*. http://www.focus.de/schule/heft/tid-10180/schultoiletten-es-stinkt-zum-himmel_aid_299129.html, Mai 2008.
- [Geba07] NICO GEBAUER, STEFAN KIRST und FLORIAN TANKE: *Softwarepraktikum SS 2007 - Programmierung von Steuerungsalgorithmen für mobile Roboter (Lego-NXT)*. http://ivs.cs.uni-magdeburg.de/EuK/lehre/sopras/presentationen/2007_3/G3_Abschlussbericht_Folien.pdf, Juli 2007.
- [Hard08] CHARLES HARDNETT und IRETTA B. C. KEARSE: *Computer science olympiad: exploring computer science through competition*. In: J. D. DOUGHERTY, SUE FITZGERALD, MARK GUZDIAL und SUSAN H. RODGER (Herausgeber): *SIGCSE*, Seiten 92–96. ACM, 2008.
- [Huan08] SHIH-LUNG HUANG, I-CHIH TSENG und CHENG-CHIH WU: *Visualization of Program Behaviors: Physical Robots Versus Robot Simulators*. In: ROLAND T. MITTERMEIR und MACIEJ M. SYSLO (Herausgeber): *ISSEP*, Band 5090 der Reihe *Lecture Notes in Computer Science*, Seiten 53–62. Springer, 2008.

- [Info08] INFORMATICA08: *Robot Team Challenge*.
http://www.adnovum.ch/informatica08/pdf/robotteam_projektstart.pdf, April 2008.
- [Kali08] ALEXANDER KALINOGLU und MARIA MOUNDRIDOU: *Using LEGO Mindstorms as an Instructional Aid in Technical and Vocational Secondary Education: Experiences from an Empirical Case Study*. In: PIERRE DILLENBOURG und MARCUS SPECHT (Herausgeber): *EC-TEL*, Band 5192 der Reihe *Lecture Notes in Computer Science*, Seiten 312–321. Springer, 2008.
- [Kuma04] AMRUTH N. KUMAR: *Three years of using robots in an artificial intelligence course: lessons learned*. *ACM Journal of Educational Resources in Computing*, 4(3):1–15, 2004.
- [LEGO08] LEGO EDUCATION: *LEGO® Education WeDo™ Fact Sheet*.
http://www.lego.com/education/download/WeDoFactSheet_v1.pdf, Juni 2008.
- [leJO08] THE LEJOS NXJ TUTORIALS: *Communications*.
<http://lejos.sourceforge.net/nxt/nxj/tutorial/Communications/Communications.htm>, Dezember 2008.
- [Mark08] MARKET SHARE: *Operating System Market Share*. <http://marketshare.hitslink.com/report.aspx?qprid=8>, Dezember 2008.
- [Möss01] HANSPETER MÖSSENBÖCK: *Sprechen Sie Java? Eine Einführung in das systematische Programmieren*. Dpunkt.Verlag, Heidelberg, 1. Auflage, 2001.
- [Öste08] ÖSTERREICHISCHE GESELLSCHAFT FÜR INNOVATIVE COMPUTERWISSENSCHAFTEN: *RobotChallenge*.
<http://www.robotchallenge.at/>, Dezember 2008.
- [Pres08] DIE PRESSE: *Lehrer fühlen sich überlastet*.
<http://diepresse.com/home/bildung/schule/422513/>, Oktober 2008.
- [Stan08] DER STANDARD: *Beamtenapparat ist überaltert*.
<http://derstandard.at/?url=?id=1229702040467>, Dezember 2008.

- [Swis08] SWISSEdUC: *Arbeitsblätter für den Unterrichtseinsatz von JavaKara*. <http://www.swisseduc.ch/informatik/karatojava/javakara/material/>, Dezember 2008.
- [Tech08] TECHNISCHE UNIVERSITÄT KAISERSLAUTERN: *Tag der Informatik - Robotik-Wettbewerb*. <http://tag-der.informatik.uni-kl.de/wettbewerb/>, Dezember 2008.
- [Virt08a] VIRTUELLER CAMPUS PROJEKT, PHBERN: *Lego-Robotik mit Java*. <http://www.lego-robotik.ch/legoEnglish/home/belp/AuftragI.pdf>, Dezember 2008.
- [Virt08b] VIRTUELLER CAMPUS PROJEKT, PHBERN: *Lego-Robotik mit Java*. <http://www.lego-robotik.ch/legoEnglish/home/belp/AuftragII.pdf>, Dezember 2008.