Entwicklung und Implementierung des „Wippe-Experiments“ mit Hilfe von LEGO® Mindstorms® NXT

Masterarbeit

im Fach Informatik

zur Erlangung des Grades eines

Master Of Education

vorgelegt von

Matthias Querbach

Mat.-Nr.: 207 200 362

Studiengang: Master Of Education (Mathematik, Informatik, Bildungswissenschaften)

wohnhaft in: Schulstrasse 39, 56332 Burgen

Betreuer:
Prof. Dr. Dieter Zöbel,
Institut für Informatik: AG Echtzeitsysteme
Alexander Hug,
Arbeitsgebiet Fachdidaktik der Informatik

Universität Koblenz-Landau
Campus Koblenz
Fachbereich 4: Informatik

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbstständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium (CD-ROM).

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Echtzeitsysteme finden sich in einer großen Anzahl von Anwendungen wieder. Ihren Ursprung haben Echtzeitsysteme in der Produktions- und Energietechnik, also in großtechnischen Anlagen. Inzwischen werden sie in vielen Anwendungsgebieten eingesetzt, auch im privaten und persönlichen Bereich. Eines der bedeutendsten Anwendungsgebiete ist die Fahrzeugtechnik. Beispiele aus diesem Gebiet, wie der Airbag oder das Anti-Blockier-System (ABS), verdeutlichen, dass solche Prozesse nicht nur funktional korrekt, sondern auch innerhalb zeitlicher Rahmenbedingungen ausgeführt werden müssen. Diese zwei Anforderungen definieren die Haupteigenschaften eines Echtzeitsystems. Anhand dieser Beispiele ist auch erkennbar, dass Echtzeitsysteme Teil der Erfahrungswelt des Menschen sind und somit auch in gewissen Grundzügen bekannt sein sollte, wie diese Systeme arbeiten.
Die Bachelorarbeit von Irina Schmidt [16] beschäftigt sich dahingehend, wie das Fachgebiet Echtzeitsysteme für den Informatikunterricht verschiedener Schulstufen allgemeinbildender Schulen aufbereitet werden kann. Es wird sowohl der Anwendungs- und Alltagsbezug von Echtzeitsystemen betrachtet als auch eine Strukturierung und didaktische Aufbereitung des Themas durchgeführt. Die Arbeit zeigt, dass die Umsetzung dieses Themengebiets bereits in der Sekundarstufe I in einem gewissen Umfang möglich ist. Besonders die Erarbeitung

des Grundmodells eines Echtzeitsystems sollte an einem anschaulichen Beispiel schülernah erfolgen.

Wie Videos auf Youtube[1] zeigen, ist es möglich, solche schülernahen Beispiele aus dem Anwendungsgebiet der Echtzeitsysteme mithilfe der Baukästen der LEGO® Mindstorms® NXT-Serie erfolgreich umzusetzen. Die Videos zeigen beispielsweise die Funktionsweise eines LEGO® Mindstorms® NXT-„Segways“[2] oder einen Versuchsaufbau, der an der Universität Koblenz-Landau, Fachbereich Informatik, unter dem Namen „Wippe“[3] bekannt ist. Für viele dieser Umsetzungen mit LEGO® Mindstorms® NXT existiert jedoch keine ausreichende Dokumentation. Andreas Stahlhofen hat im Rahmen seiner Diplomarbeit [21] das vorhandene Wippe-Experiment in der AG Echtzeitsysteme überarbeitet und erfolgreich einen Regelalgorithmus zur automatischen Steuerung der Wippe entworfen.

Irina Schmidts Arbeit hat gezeigt, dass es sehr gut möglich ist, Echtzeitsysteme anhand des Wippe-Versuchsaufbaus für den Schulunterricht aufzubereiten. Paradigmatische Beispiele, die in der Lehre Verwendung finden, zeugen nicht nur von der Reife des Fachgebiets [9], sondern bieten auch einen eindrucksvollen Einstieg in das Themengebiet und lassen sich immer wieder hervorholen, um tiefer in die Materie einzudringen. Der Versuchsaufbau Wippe ist ein solches paradigmatisches Beispiel, das jedoch bisher nur an der Universität oder anderen meist akademischen Institutionen zugänglich ist. Der breiten Masse bleibt die Nutzung eines solchen Aufbaus versagt.

An vielen Schulen haben jedoch die LEGO® Mindstorms® NXT-Bausätze Einzug gefunden. Sie werden in Robotik- und Informatik-AGs eingesetzt und für den Informatikunterricht verwendet [4, S. viii]. Es existiert auch eine spezielle Education-Version dieser Bausätze, die innerhalb dieser Ausarbeitung verwendet wird und speziell für den Einsatz in Schulen ausgestattet ist. Diese Bausätze können genutzt werden, um Schülerinnen und Schülern u.a. das Fachgebiet Echtzeitsysteme anwendungsbezogen näherzubringen, da die Vorarbeit von Andreas Stahlhofen es ermöglicht, den Regelalgorithmus gezielt zu exportieren und in einer LEGO® Mindstorms® NXT-Version des Wippe-Experiments zu impor-

[1] `http://www.youtube.de`, zuletzt aufgerufen am 13.09.2012.

[2] `http://www.youtube.com/watch?v=4ulBRQKCwd4`, zuletzt aufgerufen am 05.09.2012.

[3] `http://www.youtube.com/watch?v=04MLqlNZwHY`, zuletzt aufgerufen am 13.09.2012.

tieren.

Ziel dieser Ausarbeitung ist es, das Wippe-Experiment gemäß dem Aufbau innerhalb der AG Echtzeitsysteme unter Leitung von Professor Dr. Dieter Zöbel mithilfe eines LEGO® Mindstorms® NXT Education-Bausatzes funktionsfähig nachzubauen und das Vorgehen zu dokumentieren. Der dabei entstehende Programmcode soll didaktisch aufbereitet und eine Bauanleitung zur Verfügung gestellt werden. Dies soll gewährleisten, dass Schülerinnen und Schüler auch ohne direkten Zugang zu einer Hochschule oder ähnlichem Institut den Versuchsaufbau Wippe möglichst unkompliziert im Klassenraum erleben können.

1.2 Überblick

Zu Beginn der Arbeit wird in Kapitel 2 der momentane Aufbau des Wippe-Experiments an der Universität Koblenz-Landau, Fachbereich Informatik, beschrieben. Im weiteren Verlauf werden in Kapitel 3 zunächst einige wesentliche Grundlagen der Themen aufgegriffen, die für den Aufbau der Wippe relevant sind. Dies beinhaltet einen Überblick über das Themengebiet Echtzeitsysteme, eine Zusammenfassung der grundlegenden Gesetze der Mechanik sowie eine Einführung in die Regelungstechnik. In Kapitel 4 werden die für den Aufbau verwendeten Materialien vorgestellt und die Installation und Konfiguration der benötigten Komponenten und Programme erläutert. Anschließend erfolgt in Kapitel 5 eine Analyse des zugrundeliegenden Systems. Einen besonderen Platz nimmt die Beschreibung des Vorgehens zur Messung der Latenzzeiten des Systems ein sowie die Verifizierung, ob und, wenn ja, in welchem Umfang das Wippe-Experiment mit LEGO® Mindstorms® NXT umgesetzt werden kann. Im Anschluss daran wird in Kapitel 6 der Entwurf und die darauf aufbauende Umsetzung des Regelalgorithmus zur automatischen Steuerung der Wippe erläutert. Die Testphase aller Komponenten wird in Kapitel 7 beschrieben. Abschließend erfolgt in Kapitel 8 eine Zusammenfassung der wichtigsten Ergebnisse dieser Arbeit und ein Ausblick auf mögliche Verbesserungen und Erweiterungen der LEGO® Mindstorms® NXT-Wippe.

Kapitel 2

Der Versuchsaufbau Wippe

Der Versuchsaufbau Wippe stellt eine Erweiterung des bekannten „Ball on a Beam“-Experiments dar, bei dem eine Kugel auf einer beweglich gelagerten Schiene balanciert werden soll, ohne die Strecke an einem Ende der Schiene zu verlassen [8, S. 197 ff.]. Die Arbeitsgruppe Echtzeitsysteme der Universität Koblenz-Landau hat dieses Experiment in sofern erweitert, dass die Kugel nicht nur auf einer eindimensionalen Strecke, sondern auf einer zweidimensionalen Ebene balanciert werden soll. Dieser Aufbau ist unter dem Namen „Ball on a Plate“ bekannt und wird im Fachbereich Informatik der Universität Koblenz-Landau kurz als „Wippe“ bezeichnet. Im Folgenden wird der Aufbau der Wippe und die bisher realisierten Funktionalitäten beschrieben.

2.1 Aufbau

Der mechanische Aufbau der Wippe (siehe Abbildung 2.1) besteht aus einer quadratischen, schwarzen Platte (1) mit einer Seitenlänge von 50 cm, die in einem um die x- und y-Achse der Platte beweglichen Rahmen gelagert ist. Der Rahmen ist mithilfe von Schrittmotoren (2), die durch ein Steuergerät (5) geregelt werden, präzise auszulenken. Eine Kugel, die auf der Platte balanciert werden soll, wird mit einer Digitalkamera (3) erfasst, die oberhalb der Platte an einem Rahmen angebracht ist. Zu diesem Zweck wurde eine helle Kugel gewählt, so dass der Kontrast zwischen der dunklen Platte und der Kugel möglichst groß ist. Ein Desktop-Computer (4) führt die Software zum Auswerten der Bildda-

ten, Erfassen der Kugelposition und Bestimmung der Auslenkung der Platte aus. [21, S. 5 f.]

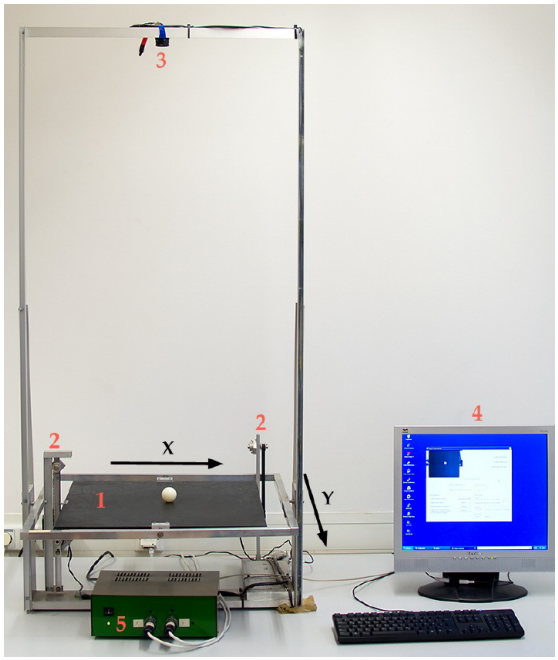


Abbildung 2.1: Der Original-Versuchsaufbau des Wippe-Experiments [21, S. 6] mit Platte (1), Schrittmotoren (2), Kamera (3), Desktop-Computer (4) und Steuergerät (5).

2.2 Funktionalität

Die Grundfunktion der Wippe ist die automatische Steuerung eines Versuchs mit dem Ziel einen Ball, der auf die Fläche geworfen wird, abzubremsen, in die Mitte der Platte zu lenken und dort zum Stillstand zu bringen.
Ferner kann die Wippe auch manuell gesteuert werden. Dies kann momentan sowohl mithilfe eines 2-Achsen-Joysticks als auch der sogenannten Wii™ Remote-Fernbedienung von Nintendo® erreicht werden.

Kapitel 3

Grundlagen

Um den Versuchsaufbau Wippe in vollem Umfang zu verstehen, werden einige Grundlagen aus den Fachgebieten der Echtzeitsysteme, der Mechanik und der Regelungstechnik benötigt. In diesem Kapitel werden diese Themen behandelt. Zunächst erfolgt ein Überblick über das Fachgebiet Echtzeitsysteme. Anschließend werden die benötigten Grundlagen aus dem Bereich der mechanischen Physik zusammengefasst. Abschließend erfolgt eine Beschreibung der Funktionsweise von PID-Reglern, die im Versuchsaufbau der LEGO® Mindstorms® NXT-Wippe verwendet werden.

3.1 Echtzeitsysteme

Die Einhaltung zeitlicher Rahmenbedingungen ist die kennzeichnende Eigenschaft vieler Anwendungsaufgaben und der damit verbundenen Echtzeitsysteme. In der Regel wird ein technischer Vorgang durch ein Rechensystem kontrolliert und gesteuert. Beispiele für solche Systeme finden sich in nahezu allen technischen Bereichen und übernehmen dort Automatisierungsaufgaben. Großtechnische Anwendungen wie Papiermaschinen oder automotive Anwendungen wie das Antiblockiersystem (ABS) oder der Airbag sind nur einige Beispiele hierfür [25, S.29 ff.]. In allen Fällen existiert ein externes System, das zeitliche Vorgaben vorgibt und ein internes System, das diese Vorgaben zu erfüllen hat. Diese Art von Systemen heißt **Echtzeitbetrieb** (oder Realzeitbetrieb). Sie sind nach DIN 44300 [6] definiert:

> *„Ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungergebnisse innerhalb einer vorgeschriebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“*

Die Definition lässt erkennen, dass der Zeitaspekt eine entscheidende Rolle innerhalb eines Echtzeitsystems einnimmt. Nachfolgend werden die Begriffe harte und weiche Echtzeit zur weiteren Differenzierung eingeführt.

3.1.1 Harte und weiche Echtzeit

Die Zeitspannen, die das externe System vorgibt, müssen von den anderen Komponenten des Echtzeitsystems eingehalten werden. Dies betrifft sowohl das Messsystem, als auch das Rechensystem und das Stellsystem (vgl. Abschnitt 3.1.2). Die Dringlichkeit, mit der diese Zeitbedingungen eingehalten werden müssen, können jedoch variieren und sind in der Regel von der Anwendung abhängig, die diese Zeitbedingungen stellt.
Nach [25, S. 4] spricht man von **weichen Zeitbedingungen**, wenn

- „es genügt, die Zeitbedingungen für den überwiegenden Teil der Fälle zu erfüllen“ oder
- „sich geringfügige Überschreitungen der Zeitbedingungen ergeben.“

Es existieren jedoch auch Systeme, bei denen eine Unterbrechung eines Produktionsprozesses, z.B. in einer großtechnischen Anlage, oder eine Gefährdung der Sicherheit von Leib und Leben nicht hinnehmbar ist [22, S. 205]. Muss eine Zeitbedingung strikt eingehalten werden, d.h. die Bedingung A, dass eine Aufgabe mit Startzeitpunkt r (*readytime*) und Ausführungszeit Δe (*execution time*) zu einem Endzeitpunkt d (*deadline*) beendet sein muss, muss erfüllt sein, so spricht man von **harten Zeitbedingungen**.

$$A \equiv r + \Delta e \leq d$$

Für harte Zeitbedingungen sind günstige Randbedingungen zu definieren, so dass die Zeitbedingung A unter allen Umständen erfüllt wird.

$$P(A|B) = 1$$

Im Allgemeinen steht nach [25, S. 5] die Bedingung B dafür, dass

- „weder technische Ausfälle auftreten“,
- „noch wichtigere Aufgaben zu erledigen sind.“

Beispiel 3.1.1. *Das Wippe-Experiment stellt die Forderung, dass die Kugel die Fläche unter keinen Umständen verlassen darf. Auch wenn dies keine Bedrohung für Leib und Leben darstellt, ist dies dennoch eine harte Zeitbedingung, denn fällt die Kugel von der Fläche, gilt der Versuch als gescheitert.*

Neben harten und weichen Echtzeitbedingungen wird oftmals noch eine weitere Kategorie von Echtzeitbedingungen definiert. Nach [17, S. 26] liegt eine **feste Echtzeitbedingung** dann vor,

- „wenn bei Nichteinhalten der Bedingung zwar kein Schaden droht, aber“
- „das Ergebnis der Operation wertlos wird.“

In [24, S. 322] wird eine Bewertung der Überschreitung der Zeitbedingungen durch eine Wertfunktion (*Time Utility Function*) übernommen. Je höher der Funktionswert in Abhängigkeit zur Ausführungszeit ist, desto höher ist der Nutzen der Aktion. Ein negativer Wert steht für einen verursachten Schaden. Eine wertlose Aktion hat den Wert 0.

> „Bei weicher Echtzeit sinkt der Wert [...] nach Überschreiten der Zeitschranke langsam ab, bis der Wert 0 erreicht wird. [...] Bei fester Echtzeit sinkt der Wert mit Erreichen der Zeitschranke auf 0 [...]. [...] Bei harter Echtzeit fällt der Wert bei Erreichen der Zeitschranke unter 0 und zeigt damit auftretenden Schaden an.“

Abbildung 3.1 zeigt exemplarisch Wertfunktionen für harte, feste und weiche Echtzeit.

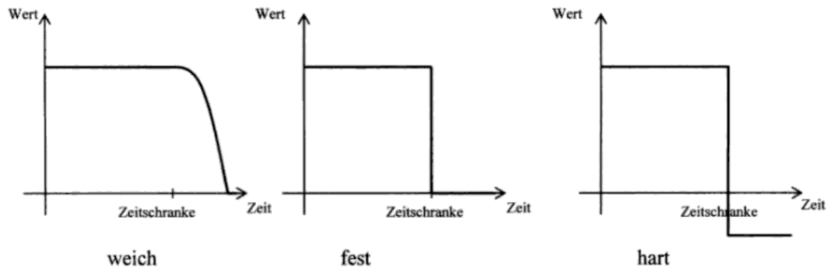


Abbildung 3.1: Wertfunktion zur Bewertung von Zeitbedingungen [24, S. 322]

Beispiel 3.1.2. *Der Versuchsaufbau des Wippe-Experiments mithilfe von LEGO® Mindstorms® NXT ist so aufgebaut, dass das Herunterfallen der Kugel durch einen Rahmen verhindert wird. Ziel ist selbstverständlich, die Kugel auszubalancieren, ohne den Rahmen zu berühren. Sollte dieser Fall dennoch eintreten, so ist der Versuch nicht gescheitert, da sich die Kugel weiterhin auf der Platte befindet. Die durchgeführten Mess- Rechen- und Stelloperationen sind jedoch wertlos. Dies stellt also eine feste Echtzeitbedingung dar. Sollte die Kugel unerwarteterweise trotz aller Berechnungen von der Platte rollen (z. B. durch ein Aufschaukeln der Kugel) so gilt der Versuch als gescheitert. Dies stellt die harte Zeitbedingung dar.*

3.1.2 Grundmodell eines Echtzeitsystems

Die Unterscheidung in ein externes und ein internes System und die Betrachtung der Schnittstellen zwischen diesen Systemen führt zum Grundmodell eines Echtzeitsystems, das in Abbildung 3.2 schematisch dargestellt ist. Das externe

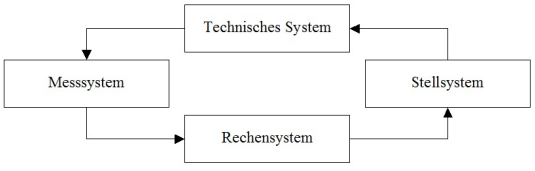


Abbildung 3.2: Grundmodell eines Echtseitsystems [25, S. 7]

oder technische System ist die zu kontrollierende Einheit. Mithilfe des Messsystems (Sensorik) werden die Daten des technischen Systems ermittelt und an das Rechensystem weitergeleitet. Im Rechensystem werden anschließend Ausgabe-

daten berechnet, die an das Stellsystem (Aktorik) weitergeleitet werden. Dieses wirkt anschließend unmittelbar auf das technische System ein. [25, S. 7]

Beispiel 3.1.3. *Beim Wippe-Experiment, aufgebaut aus dem LEGO® Mindstorms® NXT-Bausatz, ist das technische System die bewegliche Fläche und die sich darauf befindende Kugel. Das Messsystem besteht aus einer Kamera, die über dem Versuchsaufbau angebracht ist. Die Aufgabe des Rechensystems übernimmt der NXT-Baustein, der nach der Berechnung der Stellgrößen die Motorsteuerbefehle an die NXT-Servomotoren übermittelt. Die Servomotoren stellen das Stellsystem dar. In diesem Aufbau werden Teilaufgaben des Rechensystems bereits von der Kamera übernommen, die in der Lage ist, mithilfe eines Mikrocontrollers die Kugelposition zu bestimmen. Dies stellt jedoch keine Einschränkung dar, das Grundmodell eines Echtzeitsystems auf den Versuchsaufbau zu übertragen. Schließlich sind im Wesentlichen das technische System, das Mess-, Rechen- und Stellsystem auch physisch voneinander abgrenzbar.*

3.2 Physikalische Grundlagen

Das Wippe-Experiment ist ein in sich komplexes, geschlossenes System, das physikalischen Gesetzen unterliegt. Diese Gesetze bestimmen das erwartete Verhalten der Kugel auf der Platte und setzen somit die Zeitbedingungen, für die das Wippe-Experiment als Echtzeitanwendung genügen muss.
Die folgenden Annahmen werden getroffen, um die Komplexität der Problemstellung zu reduzieren und orientieren sich an den Annahmen, die auch im Original-Aufbau des Wippe-Experiments getroffen wurden (vgl. [21, S. 13]):

- Es wird angenommen, dass die Kugel die Platte zu jedem Zeitpunkt berührt. Ein Hüpfen der Kugel wird ausgeschlossen.
- Der Luftwiderstand der Kugel wird vernachlässigt. Dies ist legitim, da der Luftwiderstand betragsmäßig sehr gering ist und entgegen der Bewegungsrichtung der sich bewegenden Kugel wirkt und die Kugel somit zusätzlich abbremst.
- Die Rollreibungskraft zwischen Kugel und Platte wird vernachlässigt. Die Rollreibungskraft ist sehr gering [3, S.39] und wirkt ebenfalls entgegen der

Bewegungsrichtung.

- Bedingt durch die Vernachlässigung der Reibungskräfte wird auch ein möglicher Schlupf der Kugel auf der Platte vernachlässigt. Der Schlupf eines rollenden Objektes beschreibt das Durchdrehen des Objektes sobald dessen Haftreibung durch sein Drehmoment überschritten wird [23, S. 113].

Basierend auf diesen Annahmen wird nachfolgend das Modell der schiefen Ebene hergeleitet. Die schiefe Ebene kann als einfaches Modell der Wippe angesehen werden, bei dem die Platte lediglich um eine Achse drehbar ist. Eine Übertragung auf das Gesamtmodell der Wippe erfolgt dann, wenn eine Neigung der Ebene um die zweite Achse ebenfalls mit einbezogen wird. Ohne Beschränkung der Allgemeinheit wird im Folgenden angenommen, dass die Wippe nur um die x-Achse drehbar ist.

3.2.1 Gleichförmige Bewegung

Das Trägheitsgesetz besagt, dass ein sich bewegender Körper, sofern keine von außen auf ihn wirkende Kraft einwirkt, sich immer mit gleichbleibender Geschwindigkeit weiterbewegt. Eine solche Bewegung mit konstanter Geschwindigkeit und gleichbleibender Richtung nennt man **gleichförmige Bewegung**. Die Geschwindigkeit v berechnet sich aus dem Quotienten der Ortsänderung Δs und der dazu benötigten Zeit Δt:

$$v = \frac{\Delta s}{\Delta t} \tag{3.1}$$

[3, S. 16]

3.2.2 Beschleunigte Bewegung

Wirkt eine Kraft auf einen Körper, so ändert sich dessen Geschwindigkeit, er wird beschleunigt. Man spricht von einer **beschleunigten Bewegung**. Die Beschleunigung a berechnet sich aus dem Quotienten der Geschwindigkeitsänderung Δv und der dazu benötigten Zeit Δt:

$$a = \frac{\Delta v}{\Delta t} \tag{3.2}$$

Ist die beschleunigende Kraft konstant, so spricht man von einer **gleichmäßig beschleunigten Bewegung**. [3, S. 26]

3.2.3 Beschleunigungsgesetz

Die Grundgleichung der Mechanik, auch bekannt als das **Newtonsche Beschleunigungsgesetz**, besagt, dass die beschleunigende Kraft eines Körpers das Produkt aus Masse des Körpers und dessen Beschleunigung ist:

$$F = m \cdot a \tag{3.3}$$

Die hierbei auftretende Geschwindigkeitsänderung erfolgt in Richtung der beschleunigenden Kraft, da Kraft wie Geschwindigkeit und Beschleunigung eine vektorielle Größe ist:

$$\vec{F} = m \cdot \vec{a} \tag{3.4}$$

F wird in Newton (N) gemessen. [3, S. 34]

3.2.4 Kraftkomponenten an der schiefen Ebene

Die beschleunigende Kraft, die in einer schiefen Ebene auf einen Körper wirkt, die *Hangabtriebskraft* F_H, ist abhängig vom Winkel α der schiefen Ebene. Die Hangabtriebskraft ist die Komponente der *Gewichtskraft* F_G in Richtung einer möglichen Bewegung. Die zur Ebene senkrechte Kraft ist die *Normalkraft* F_N. Sie ist verantwortlich für den Anpressdruck des Körpers. Abbildung 3.3 verdeutlicht dies. Die Hangabtriebskraft F_H lässt sich wie folgt berechnen:

$$F_H = F_G \cdot sin(\alpha) \tag{3.5}$$

Die Gewichtskraft F_G lässt sich wie folgt berechnen:

$$F_G = m \cdot g \text{ mit } g \approx 9,81 \ \frac{m}{s^2} \tag{3.6}$$

[3, S. 14]

Demnach ist die Beschleunigung eines Körpers auf einer schiefen Ebene bestimmt durch:

$$a = \frac{F_H}{m} \Leftrightarrow a = \frac{F_G \cdot sin(\alpha)}{m} \Leftrightarrow a = \frac{m \cdot g \cdot sin(\alpha)}{m} \Leftrightarrow a = g \cdot sin(\alpha) \tag{3.7}$$

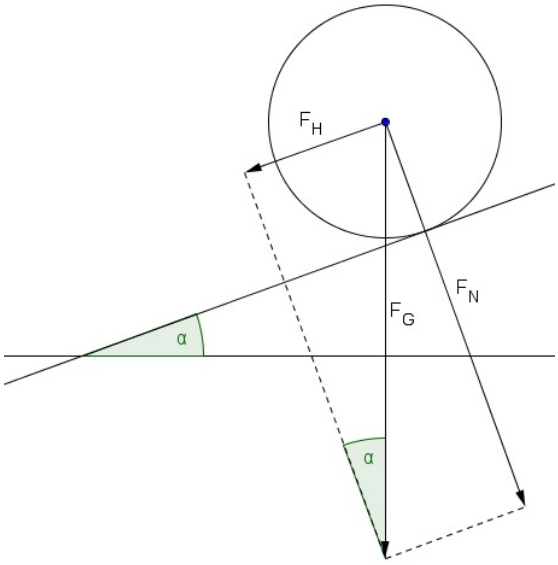


Abbildung 3.3: Die Gewichtskraft F_G eines Objekts auf einer schiefen Ebene kann in die Komponenten Hangabtriebskraft F_H und Normalkraft F_N gespalten werden. (Angefertigt mit GeoGebra)

Die Trägheitsgesetze einer massiven Kugel auf einer schiefen Ebene geben jedoch vor, dass dieser Wert mit dem Vorfaktor $\frac{5}{7}$ multipliziert werden muss [26].

$$a_{max} = g \cdot \frac{5}{7} \cdot sin(\alpha) \tag{3.8}$$

3.2.5 Das Weg-Zeit-Gesetz

Die Strecke, die ein gleichmäßig beschleunigter Körper innerhalb einer bestimmten Zeit zurücklegt, kann mithilfe der Differentialrechnung und Gleichung 3.2 hergeleitet werden, wobei die Geschwindigkeit v die Wegänderung pro Zeit ist:

$$\dot{s} = v \Leftrightarrow s = \int v\,dt \Leftrightarrow s = \int a \cdot t\,dt \Leftrightarrow s = \frac{1}{2} \cdot a \cdot t^2 \tag{3.9}$$

Diese Gleichung gilt für aus der Ruhelage gestartete Körper.

3.2.6 Zeitaspekt

Mithilfe der bisher gezeigten Formeln kann nun der Zeitaspekt des Wippe-Experiments betrachtet werden. Es gilt herauszufinden, in welchem Zeitraum

t_{min} sich die Kugel bei maximaler Auslenkung der Platte α_{Platte} auf deren Zentrum bewegt, falls dies nicht durch eine Änderung des Auslenkwinkels der Platte verhindert wird.
Zunächst muss mit Gleichung 3.8 die Beschleunigung einer massiven Kugel ermittelt werden.
Wird anschließend Gleichung 3.9 nach t umgestellt, erhält man hierfür [1]:

$$s = \frac{1}{2} \cdot a \cdot t^2 \Leftrightarrow t^2 = 2 \cdot \frac{s}{a} \Leftrightarrow t = \sqrt{2 \cdot \frac{s}{a}} \tag{3.10}$$

In diese Gleichung kann nun die soeben ermittelte maximale Beschleunigung eingesetzt werden und die minimale Zeitspanne berechnet werden. Die berechnete Zeitspanne ist jene, die die Kugel benötigt um eine vorgegebene Strecke, die um einen bestimmten Winkel geneigt ist, zurückzulegen.
Der mit Gleichung 3.10 hergeleitete Wert $t = t_{min}$ gibt im Anwendungsfall des Wippe-Experiments einen minimalen Wert für die benötigte Zeit zum Zurücklegen der Strecke vom Rand bis zum Zentrum der Platte an. Die Bestimmung des minimalen Werts wird folgendermaßen begründet:

- Der Luftwiderstand der Kugel, welcher zunächst vernachlässigt wurde, wirkt entgegen der Bewegungsrichtung der Kugel. Die Kugel erreicht somit eine niedrigere Geschwindigkeit und benötigt in der Praxis länger zum Erreichen des Zentrums der Platte.

- Die Rollreibungskräfte sind sehr gering, wirken aber ebenfalls entgegen der Bewegungsrichtung und verlangsamen somit ebenfalls die Kugel.

- Bereits während sich die Kugel auf das Zentrum der Platte zubewegt, wird der Neigungswinkel der Platte verändert, um die Bewegung abzufangen. Die Beschleunigung der Kugel kann also im Durchschnitt kleiner als die maximale Beschleunigung angenommen werden ($a_{avg} \leq a_{max}$).

Die Berechnung der minimal benötigten Zeit zum Erreichen des Plattenzentrums t_{min} kann somit als großzügig bezeichnet werden. Die Zeitspanne beschreibt den schlechtesten Fall, der eintreffen kann.

[1] Gleichung 3.10 liefert im Allgemeinen zwei Lösungen mit $t_1 \geq 0$ und $t_2 \leq 0$, wobei die Lösung $t_1 = t_2 = 0$ den Spezialfall darstellt, dass sich die Kugel bereits im Zentrum der Platte befindet. In diesem speziellen Anwendungsfall des Wippe-Experiments kann die Lösung $t_2 \leq 0$ vernachlässigt werden.

Beispiel 3.2.1. *Die Beschleunigung a einer Kugel, die auf einer schiefen Ebene mit Auslenkwinkel* $\alpha_{Platte} = 10°$ *liegt, beträgt* $a = 9,81\ \frac{m}{s^2} \cdot sin(10°) \approx 1,70\ \frac{m}{s^2}$. *Die angepasste maximale Beschleunigung a einer Kugel auf einer schiefen Ebene mit Auslenkwinkel* $\alpha_{Platte} = 10°$ *beträgt nun* $a = 9,81\ \frac{m}{s^2} \cdot \frac{5}{7} \cdot sin(10°) \approx 1,22\ \frac{m}{s^2}$. *Sei* $s = 0,06\ m$ *die Strecke vom Rand bis zum Mittelpunkt der Platte und* $a = a_{max} = 1,22\ \frac{m}{s^2}$, *so lässt sich die Zeit, die zum Zurücklegen der Strecke s benötigt wird, angeben durch* $t = \sqrt{2 \cdot \frac{0,06\ m}{1,22\ \frac{m}{s^2}}} \approx 0,314\ s$.
Dies stellt die minimale Zeitspanne dar, die die Kugel benötigt, um das Plattenzentrum zu erreichen.

3.3 PID-Regler

Die Grundfunktionalität der Wippe umfasst, wie in Abschnitt 2.2 beschrieben, die automatische Steuerung des Versuchs. Der Begriff **Steuerung** ist gemäß DIN 19 226 Teil 1 [7] wie folgt definiert:

> *„Das Steuern, die Steuerung ist ein Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungsweg oder ein geschlossener Wirkungsweg, bei dem die durch die Eingangsgrößen beeinflussten Ausgangsgrößen nicht fortlaufend und nicht wieder über dieselben Eingangsgrößen auf sich selbst wirken.“*

Die Definition beschreibt, dass es sich bei der Kontrolle der Wippe nicht um eine Steuerung handeln kann, da es sich um ein geschlossenes System mit geschlossenem Wirkungsweg handelt, bei dem sich Eingangs- und Ausgangsgröße kontinuierlich gegenseitig beeinflussen. Abgrenzend hierzu wird der Begriff **Regelung** ebanfalls nach DIN 19 226 Teil 1 [7] folgendermaßen definiert:

> *„Das Regeln, die Regelung ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (die zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für*

> *das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.“*

Unter dieser Betrachtung ist erkennbar, dass es sich bei dem Wippe-Versuch nicht um einen Steuerungs-, sondern um einen Regelungsvorgang handelt, der den Einsatz von Reglern notwendig macht.
Im Folgenden wird die Funktionsweise eines **PID-Reglers** erläutert. PID-Regler finden sich in einer Vielzahl automatischer Kontrollanwendungen wieder und regulieren zum Beispiel Temperatur, Druck, Geschwindigkeiten und andere messbare physikalische Größen [18]. Auch in der LEGO® Mindstorms® NXT-Wippe kommt zur Steuerung der Servomotoren ein PID-Regler zum Einsatz, um absolute Positionen anzusteuern und eine möglichst exakte Steuerung zu gewährleisten.

PID-Regler sind benannt nach den ***P**roportional-*, ***I**ntegral-* und ***D**erivative-*Gliedern, aus denen sie bestehen und entstammen der Regelungstechnik. Nachfolgend wird das P-, I- und D-Glied des Reglers beschrieben.

3.3.1 P-Glied

Das **Proportional**-Glied erzeugt zu einer dynamischen Eingangsgröße $e(t)$ eine Ausgangsgröße $y(t)$, die proportional mit Proportionalitätsfaktor K_P zu der Eingangsgröße ist. Die Eingangsgröße berechnet sich aus der momentanen Differenz des Ist- und des Sollwerts, der sogenannten *Regelabweichung*. Im Idealfall handelt es sich hierbei um einen reinen Verstärker. [14]
Ein reiner P-Regler ist besonders dann fehleranfällig, wenn sich die Eingangsgröße nicht sprunghaft, sondern stetig ändert. Zu Beginn der Änderung der Eingangsgröße erfolgt die Änderung der Ausgangsgröße nur sehr schwach. Die Ausgangsgröße wird solange verstärkt bis die momentan auftretende Änderungsrate der Eingangsgröße ausgeglichen werden kann. Die Eingangsgröße bleibt konstant und die bis dahin aufgetretene Regelabweichung kann nicht ausgeglichen werden. [18]
Infolgedessen hat der P-Regler folgende Eigenschaften:

1. Ändert sich die Eingangsgröße so reagiert der Regler unmittelbar.

2. Der Regler ist stabil.

3. Auftretende Regelabweichungen können nicht immer ausgeglichen werden.

Die Reglergleichung eines P-Reglers lautet:

$$y(t) = K_R \cdot e(t) \tag{3.11}$$

Ein PID-Regler (Gleichung 3.13) kann durch Nullsetzen des I- und D-Glieds zu einem P-Regler umfunktioniert werden.

Beispiel 3.3.1. *Im Anwendungsfall des LEGO® Mindstorms® NXT-Servomotors ließe sich ein P-Regler folgendermaßen beschreiben: Der Befehl veranlasst den Servomotor sich um eine bestimmte Gradzahl zu drehen. Je größer die Abweichung der momentanen Position zur gewünschten Position ist, umso schneller muss sich der Motor drehen.*

3.3.2 I-Glied

Das **Integral**-Glied liefert eine Ausgangsgröße durch Integration der Eingangsgröße, die Regelabweichung, multipliziert mit einem Proportionalitätsfaktor K_I [14]. Die Ausgangsgröße steigt linear, bedingt durch den meist sprunghaften Anstieg der Eingangsgröße.
Bedingt durch die Integration der Eingangsgröße ist die Ausgangsgröße bei Erreichen des Sollwertes ungleich 0. Für das Erreichen von 0 muss die Eingangsgröße mindestens einmal das Vorzeichen wechseln. Erreicht anschließend die Ausgangsgröße den Wert 0 so ist der Sollwert überschritten und die Eingangsgröße wird wieder integriert. Dies führt zu einer Schwingung des Istwerts des I-Reglers um den Sollwert, die sich diesem nur langsam nähert. [14]
Abgeleitet daraus ergeben sich für den I-Regler folgende Eigenschaften:

1. Ändert sich die Eingangsgröße, so reagiert der Regler nur langsam.

2. Der Regler neigt zu Schwingungen um den Sollwert.

3. Eine präzise Regelung stellt sich nur sehr langsam ein.

Die Reglergleichung eines PI-Reglers lautet:

$$y(t) = K_R \cdot e(t) + K_I \cdot \int_0^t e(T)dT \tag{3.12}$$

Ein PID-Regler (Gleichung 3.13) kann durch Nullsetzen des P- und D-Glieds zu einem I-Regler umfunktioniert werden. Dieser findet jedoch nur selten eine alleinige Anwendung und wird meist nur in Kombination mit einem P- und, wenn nötig, einem D-Glied verwendet.

Beispiel 3.3.2. *Im Anwendungsfall des LEGO® Mindstorms® NXT-Servomotors integriert das reine I-Glied die Regelabweichung, indem der Unterschied zwischen dem Soll- und dem Istwert innerhalb einer bestimmten Zeitspanne gemessen und aufsummiert wird. Je länger eine Regelabweichung vorliegt, umso stärker und schneller wird sich der Motor drehen. In Verbindung mit einem P-Glied unterstützt der Regelungsanteil des I-Glieds das P-Glied oder wirkt diesem entgegen, je nach Vorzeichen des Integrals.*

3.3.3 D-Glied

Das **Derivative**-Glied erzeugt eine Ausgangsgröße, die proportional zur zeitlichen Änderung der Eingangsgröße, der Regelabweichung, ist. Durch den meist sprunghaften Anstieg der Eingangsgröße entspricht die Ausgangsgröße häufig einer Impulsfunktion. [14]
Das D-Glied sorgt im Moment der Regelabweichung maßgeblich für die Berechnung der Ausgangsgröße, da beim Auftreten der Regelabweichung auch eine große Änderungsgeschwindigkeit dieser existiert. Wird die Änderungsgeschwindigkeit betragsmäßig kleiner verringert sich auch der D-Anteil der Ausgangsgröße. [14]
Der D-Regler hat folgende Eigenschaften:

1. Der D-Regler kann nicht als alleiniger Regler genutzt werden.
2. Nur eine Änderung der Eingangsgröße verursacht eine Reaktion des Reglers.
3. Konstante Regeldifferenzen werden nicht geregelt.
4. Das D-Glied bewirkt eine schnelle Regelung von Störungen.

Die Reglergleichung eines PID-Reglers lautet:

$$y(t) = K_R \cdot e(t) + K_I \cdot \int_0^t e(T)dT + K_D \cdot \frac{de(t)}{dt} \tag{3.13}$$

Ein PID-Regler (Gleichung 3.13) kann durch Nullsetzen des P- und I-Glieds zu einem D-Regler umfunktioniert werden. Dieser findet jedoch keine Anwendung, da konstante Eingangsgrößen nicht reguliert werden können, und wird nur in Kombination mit einem P- und, wenn nötig, einem I-Glied verwendet.

Beispiel 3.3.3. *Das D-Glied reagiert auf die Änderungsrate der Regelabweichung. Je schneller sich die Regelabweichung ändert, umso größer ist der Regelungsanteil des D-Glieds. Im Fall des LEGO® Mindstorms® NXT-Servomotors wirkt das D-Glied besonders nach dem Beschleunigen und vor dem Abbremsen des Motors, wenn der Motor seine Maximalgeschwindigkeit erreicht hat.*

Je nach Anwendungsfall werden dem P-, I- und D-Glied unterschiedliche Werte zwischen 0 und 100 zugeordnet. Häufig können diese Werte nur durch Testreihen ermittelt werden. Auch in der Programmiersprache NXC, die die PID-Motorsteuerbefehle für den LEGO® Mindstorms® NXT unterstützt, können diese Werte individuell kalibriert werden. Für die verwendeten Befehle zur absoluten Regelung der Position stehen jedoch nur Werte von 0 bis 7 zur Verfügung [20, S. 774].

Kapitel 4

Aufbau und Installation

Dieses Kapitel befasst sich mit dem Aufbau der LEGO® Mindstorms® NXT-Wippe sowie der Installation und Einrichtung aller benötigter Softwarekomponenten. Dies umfasst die Anwendung NXTCamView zur Kalibrierung der Kamera, die Entwicklungsumgebung Bricx zur Übertragung der notwendigen Daten an den NXT-Baustein und die damit verbundene Neuinstallation der NXT-Firmware.

4.1 Materialien

Für den Aufbau des Wippe-Experiments mithilfe von LEGO® Mindstorms® NXT werden folgende Materialen verwendet:

- LEGO® Mindstorms® Education Basisset NXT 9797 (Abbildung 4.1a)
- LEGO® Mindstorms® Ergänzungsset NXT 9695 (Abbildung 4.1b)
- Vision Subsystem v4 for NXT (NXTCam-v4) (Abbildung 4.1c)
- Mini-USB-Kabel (Das Mini-USB-Kabel ist nicht im Lieferumfang der NXTCam enthalten.) (Abbildung 4.1d)
- Plexiglasplatte ($12\ cm \times 12\ cm$)
- Ball (möglichst leicht, rot oder weiß)
- Sprühlack (schwarz, matt)

- Fotokarton (DIN A3, schwarz)

4.2 Entwicklung des Versuchsaufbaus

Für die Realisierung der LEGO® Mindstorms® NXT-Wippe wurden mehrere Versuchsaufbauten erstellt. Der endgültige Aufbau befindet sich in der vierten Version.

4.2.1 MK1

Der erste Aufbau der Wippe lieferte bereits die meisten grundlegenden Funktionalitäten, die gefordert wurden.
Der Servomotor zur Steuerung der y-Achse befand sich, wie im Original-Versuchsaufbau im Labor der AG Echtzeitsysteme (vgl. Abbildung 2.1), innerhalb der Aufhängung der x-Achse (siehe Abbildung 4.2). Durch diesen Aufbau konnten jedoch die gewünschten Auslenkwinkel der Platte nicht direkt angesteuert werden.
Die Größe der Wippe orientierte sich zunächst an dem in Kapitel 1 genannten Youtube-Video[1], der weitere Aufbau und damit verbunden auch die Mechanik am Original-Aufbau.
Infolgedessen konnte die Achse der Servomotoren nicht exakt unterhalb der Aufhängung der Platte angebracht werden. Die Auslenkwinkel mussten mithilfe eines mathematischen Modells umgerechnet werden und konnten nur näherungsweise bestimmt werden. Aus diesem Grund und da die Breite der Platte mit 12 *cm* weniger als einem Viertel der Breite des Original-Aufbaus entsprach und dementsprechend die Zeitbedingung, die eingehalten werden mussten, deutlich kürzer waren, wurde ein zweiter Aufbau installiert.

4.2.2 MK2

Der Rahmen des zweiten Aufbaus umfasste eine Platte mit 25 *cm* Länge, was der Hälfte der Abmessungen der Original-Wippe entspricht. Mit zunehmender Ausdehnung stieg jedoch auch die Instabilität des Gesamtaufbaus, da der Bau

[1] `http://www.youtube.com/watch?v=04MLqlNZwHY`, zuletzt aufgerufen am 13.09.2012.

(a) LEGO® Mindstorms® Education Basisset NXT 9797 (http://education.lego.com/en-us/lego-education-product-database/mindstorms/9797-lego-mindstorms-education-base-set/, letzter Zugriff am 22.10.2012)

(b) LEGO® Mindstorms® Ergänzungsset NXT 9695 (http://education.lego.com/en-us/lego-education-product-database/mindstorms/9695-lego-mindstorms-education-resource-set/, letzter Zugriff am 22.10.2012)

(c) Vision Subsystem v4 for NXT (http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=78, letzter Zugriff am 22.10.2012)

(d) Mini-USB-Kabel (http://www.amazon.de/Micro-USB-Kabel-USB-A-Stecker-Micro-B/dp/B000X26AYU/ref=sr_1_6?ie=UTF8&qid=1350899742&sr=8-6, letzter Zugriff am 22.10.2012)

Abbildung 4.1: Die verwendeten Materialen im Überblick

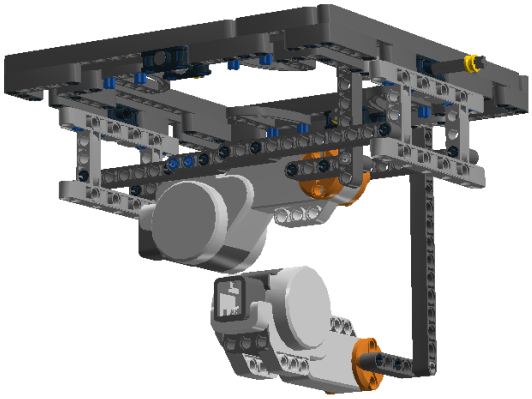


Abbildung 4.2: Die Aufhängung der y-Achse in der ersten Version der Wippe machten ein exaktes Ansteuern des berechneten Auslenkwinkels nahezu unmöglich. (Angefertigt mit LEGO Digital Designer)

auf die im vorherigen Abschnitt genannten Baukästen beschränkt ist.
Weiterhin konnte die Kamera, da diese nun doppelt so weit von der Platte entfernt sein musste als zuvor, die Kugel nicht zu jedem Zeitpunkt ausreichend präzise verfolgen. Dies stellt eine grobe Verletzung der Bedingung B dar, die u.a. gewährleisten soll, dass keine technischen Ausfälle auftreten (vgl. Abschnitt 3.1.1).

4.2.3 MK3

Die dritte Version der Wippe wurde in den ursprünglichen Maßen von 12 cm gebaut. Ferner führte die Aufhängung unterhalb der x-Achse zu einem niedrigen Schwerpunkt des Aufbaus, so dass Schwingungen ähnlich einem mechanischen Pendel nicht auszuschließen waren. Um dem Servomotor dieser Mehrbelastung nicht weiter auszusetzen und die Einhaltung der Zeitbedingungen nicht zu gefährden, wurde der Servomotor zur Steuerung der y-Achse nicht weiter an der Aufhängung der x-Achse befestigt, sondern unterhalb der Wippe. Die Steuerung der Auslenkung der x-Achse erfolgte nun unmittelbar am äußeren Rahmen der Wippe und die Steuerung der y-Achse indirekt über eine Lenkstange[2].

[2]Die indirekte Steuerung mithilfe der Lenkstange führt zu einem Fehler der Auslenkung der y-Achse, falls gleichzeitig die x- und y-Achse ausgelenkt werden. Dieser Fehler ist jedoch, bedingt durch die nur sehr kleine maximale Auslenkung der Platte, verschwindend gering und

Während einiger Testreihen der Wippe in der dritten Version zeigte sich, dass die Steuerung der Servomotoren, die sich auf einen Grad genau steuern lassen [13], nicht ausreicht, um eine präzise Steuerung der Wippe zu gewährleisten.

4.2.4 MK4

Aus diesem Grund wurde eine vierte Version der Wippe entwickelt, die durch ein Zahnradgetriebe eine fünffache Untersetzung der Motoren erreicht (siehe Abbildung 4.3). Die Untersetzung eines Zahnradgetriebes berechnet sich durch [15, S. 2 f.]:

$$i = \frac{Z_{Antrieb}}{Z_{Abtrieb}} = \frac{40}{8} = 5 \tag{4.1}$$

Somit ist eine präzise Auslenkung der Platte in Abständen von $\frac{1}{5}^\circ = 0,2^\circ$ möglich.

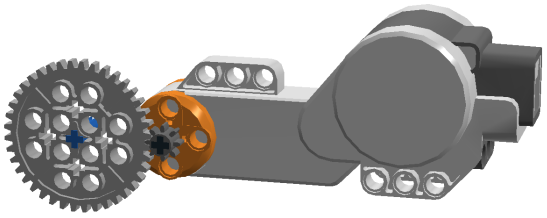


Abbildung 4.3: Mithilfe eines Zahnradgetriebes wurde eine fünffache Untersetzung der Motoren erreicht. Dies ermöglicht eine Auslenkung der Platte in Abständen von $0,2^\circ$. (Angefertigt mit LEGO Digital Designer)

4.3 Aufbau

Der derzeitige Aufbau der LEGO® Mindstorms® NXT-Wippe (siehe Abbildung 4.4) wurde vollständig aus dem in Abschnitt 4.1 aufgelisteten Material aufgebaut. Alle benötigten Bausteine sowie eine ausführliche Bauanleitung der Wippe sind in Anhang A zu finden.

Die NXTCam kann während des Zusammenbaus an der Halterung am oberen Ende des Wippe-Aufbaus (2) angebracht werden. Die Verbindungsstecker sollten hierfür vor der Anbringung an der Kamera befestigt werden. Im Anschluss

kann vernachlässigt werden.

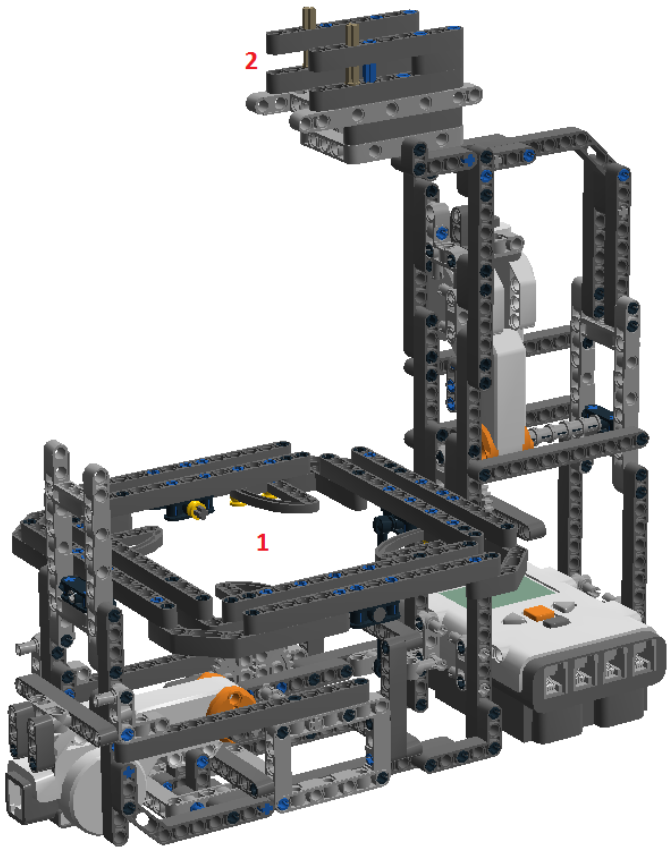


Abbildung 4.4: Die LEGO® Mindstorms® NXT-Wippe in der endgültigen Version. Es fehlt die Plexiglasplatte, die in die Aufhängung (1) gelegt werden muss und die NXTCam, die an der dafür vorgesehenen Halterung (2) angebracht werden muss. (Angefertigt mit LEGO Digital Designer)

sollten die Verbindungsstecker möglichst wenig ausgetauscht werden um Verschleiß an der Kamera vorzubeugen [12, S. 3].
Ist die Wippe vollständig aufgebaut können die Anschlusskabel der NXT-Servomotoren und der Kamera angebracht werden. Die Kamera muss an Sensor Port 1 angeschlossen werden, der Servomotor zur Steuerung der x-Achse (dieser befindet sich im Aufbau der Kameraaufhängung, seitlich der Wippe) an Motor Port A und der Servomotor zur Steuerung der y-Achse (dieser befindet sich unterhalb der beweglichen Aufhängung) an Motor Port B.
Bei der Plexiglasplatte handelt es sich um eine 12 $cm \times 12$ cm große und ca. 1 mm starke Platte. Die Plexiglasplatte kann in die um die x- und y-Achse schwenkbare Aufhängung (1) gelegt werden und muss nicht weiter befestigt werden.
Um eine besonders zuverlässige Objektverfolgung der Kamera zu gewährleisten, ist es notwendig, für einen starken Kontrast zwischen Kugel, Platte und Hintergrund zu sorgen. Im vorliegenden Versuchsaufbau wurde zum einen die verwendete Plexiglasplatte mit schwarzen, matten Sprühlack grundiert, zum anderen wurden die LEGO®-Bausteine, aus denen der bewegliche Rahmen der Wippe besteht, in dem die Plexiglasplatte gelagert ist, ebenfalls mit dem selben Sprühlack grundiert. Versuche zeigten, dass, bedingt durch auftretende Lichtreflexionen an den glänzenden Kanten der LEGO®-Bausteine, eine dauerhaft korrekte Objektverfolgung nur möglich ist, wenn diese Reflexionen auf einen Bruchteil reduziert werden. Dies ist durch die Verwendung des schwarzen, matten Sprühlacks zu erreichen. Ferner sollte aus dem gleichen Grund der Hintergrund des Kamerabildes, also die Auflagefläche der Wippe, mit einem schwarzen Karton abgedunkelt werden. Besonders wichtig ist, dass sich die Farbe des verwendeten Balls von der der Plexiglasplatte sowie von der Farbe des Rahmens und des Untergrundes unterscheidet. Die optimale Farbe des verwendeten Balls kann, bedingt durch die verwendete Beleuchtung, variieren. Mit einem roten oder weißen Ball lassen sich in den meisten Fällen gute Ergebnisse erzielen.
Falls die vorhandenen Lichtverhältnisse nicht genügen, den Versuchsaufbau ausreichend zu beleuchten, kann eine Lampe oberhalb des Versuchsaufbaus angebracht werden. Abhängig von den Lichtverhältnissen kann es auch notwendig sein, die Kugel durch eine andersfarbige auszutauschen. Bei Einsatz unter weißem Tageslicht hat sich eine rote Kugel bewährt, bei der Verwendung in

künstlichem Licht mit Gelbstich eine weiße Kugel.

4.4 Installation und Konfiguration der Kamera

Die NXTCam-v4 ist eine speziell für das LEGO® Mindstorms® NXT-System entwickelte Kamera. Die Kamera kann mithilfe eines Mini-USB-Kabels an einem Rechner oder mit einem NXT-Anschlusskabel an einem der drei NXT-Sensor Ports am NXT-Baustein angeschlossen werden. Anders als gewöhnliche Webcams verfügt die NXTCam über einen Mikroprozessor und ein Protokoll Interface, auf das mit dem NXT-Sensor Port zugegriffen werden kann. Über dieses Interface wird nicht das aufgenommene Bild gesendet, sondern aufbereitete Informationen zu diesem Bild. Diese Informationen beinhalten im Wesentlichen die Koordinaten der sogenannten Bounding Box von Objekten, die im Tracking Mode verfolgt werden. Die Bounding Box eines Objektes ist in der Bildverarbeitung durch das kleinste Rechteck definiert, das das gesamte Objekt umfasst und wird durch die Minima und Maxima der x- und y-Werte beschrieben [2, S. 229]. Des weiteren wird die Anzahl aller verfolgten Objekte und deren Farbzuordnung übertragen. Mithilfe der Software NXTCamView kann die Kamera am Rechner konfiguriert und das Bild betrachtet werden. [12, S. 1]

4.4.1 Installation der Gerätetreiber

Damit die NXTCam ohne Probleme konfiguriert werden kann, müssen zunächst die USB-Treiber installiert werden. Die Gerätetreiber der NXTCam können von Mindsensors.com[3] heruntergeladen werden. Auf derselben Internetseite findet man ausführliche Installationsanleitungen für alle gängigen Betriebssysteme.

4.4.2 Installation und Inbetriebnahme von NXTCamView

NXTCamView ist eine Windows Applikation mit deren Hilfe die NXTCam konfiguriert und kontrolliert werden kann.
Die Installationsdatei von NXTCamView kann von Sourceforge.net[4] herunter-

[3] `http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=83`, letzter Zugriff am 10.07.2012.

[4] `http://sourceforge.net/projects/nxtcamview/`, letzter Zugriff am 10.07.2012.

geladen werden. Nach der Installation kann das Programm gestartet werden. Es wird umgehend versucht, die Kamera unter den verfügbaren Ports zu erkennen. Zu Beginn kann meist keine Verbindung zur Kamera hergestellt werden, es erscheint eine Fehlermeldung wie in Abbildung 4.5 dargestellt. Durch Drücken des

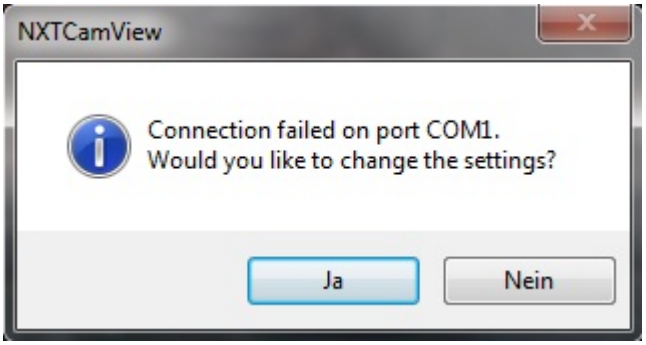
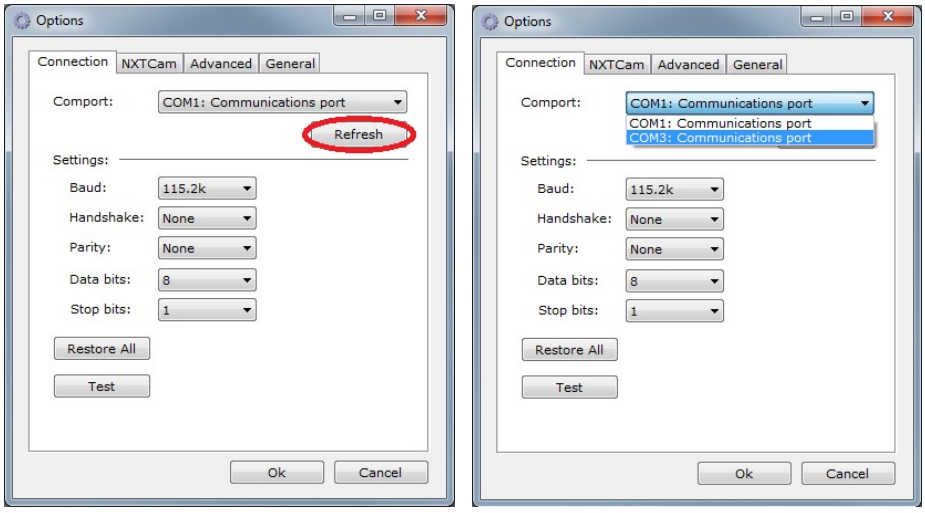


Abbildung 4.5: Fehlermeldung von NXTCamView, falls der korrekte Comport nicht erkannt wird.

Ja-Buttons gelangt man in das Optionsmenü von NXTCamView. Durch Betätigen des Refresh-Buttons, markiert in Abbildung 4.6a, wird der ComPort wie in Abbildung 4.6b der Auswahlliste hinzugefügt.



(a) Refresh-Button (b) Optionsmenü mit erweiterten Comports

Abbildung 4.6: Das Optionsmenü von NXTCamView

Abbildung 4.7 zeigt die Benutzeroberfläche von NXTCamView. Mithilfe des

Connect-Buttons kann sich, sobald der ComPort eingerichtet wurde, NXTCamView mit der Kamera verbinden. Der Disconnect-Button unterbricht diese Verbindung wieder.
Durch Auswahl des Capture-Buttons wird eine Momentaufnahme der Kamera geliefert. Dieses Bild wird anschließend im Hauptfenster angezeigt. Markiert man einen Bereich in diesem Bild durch Linksklick, wird in der Colors-Anzeige die damit getätigte Farbauswahl angezeigt. Farbwerte innerhalb des aufgenommenen Bildes, die dieser Auswahl entsprechen, werden automatisch farblich im Bild hervorgehoben. Mithilfe von Strg+Linksklick können weitere Farbwerte hinzugefügt, mit Strg+Shift+Linksklick Farbwerte entfernt werden. Durch Drücken der Ziffern-Buttons im oberen Bereich der Colors-Anzeige können bis zu acht unterschiedliche Farbauswahlen getätigt werden. Die ausgewählten Farbwerte einer Auswahl müssen jedoch paarweise disjunkt zu den Farbwerten der anderen Auswahlen sein. Wird in der Colors-Anzeige der Upload-Button betätigt, werden die gespeicherten Informationen an die Kamera gesendet und dort gespeichert.
Dieser Vorgang muss einmalig vor der Benutzung der LEGO® Mindstorms® NXT-Wippe durchgeführt werden. Bei gleich bleibenden Lichtverhältnissen kann die Objektverfolgung anschließend beliebig oft benutzt werden. Die gespeicherte Auswahl bleibt auch nach dem Abschalten des NXT-Bausteins erhalten. Nachdem die Farbauswahl auf der Kamera gespeichert wurde, ist diese bereit zur Verfolgung von Objekten. Wird der Tracking-Button betätigt, kann die Objektverfolgung bereits getestet werden. Die Tracking-Anzeige im Hauptfenster öffnet sich. Durch Betätigung des Start-Buttons wird die auf der Kamera gespeicherte Farbauswahl verfolgt und in der Tracking-Anzeige angezeigt. Der Stop-Button beendet die Objektverfolgung.

4.5 Einrichten der Entwicklungsumgebung Bricx

Das Bricx Command Center (BricxCC) ist eine integrierte Entwicklungsumgebung (IDE, vom engl. integrated development environment) für Windows-Betriebssysteme zum Programmieren von LEGO®-Roboter-Bausteinen. Es un-

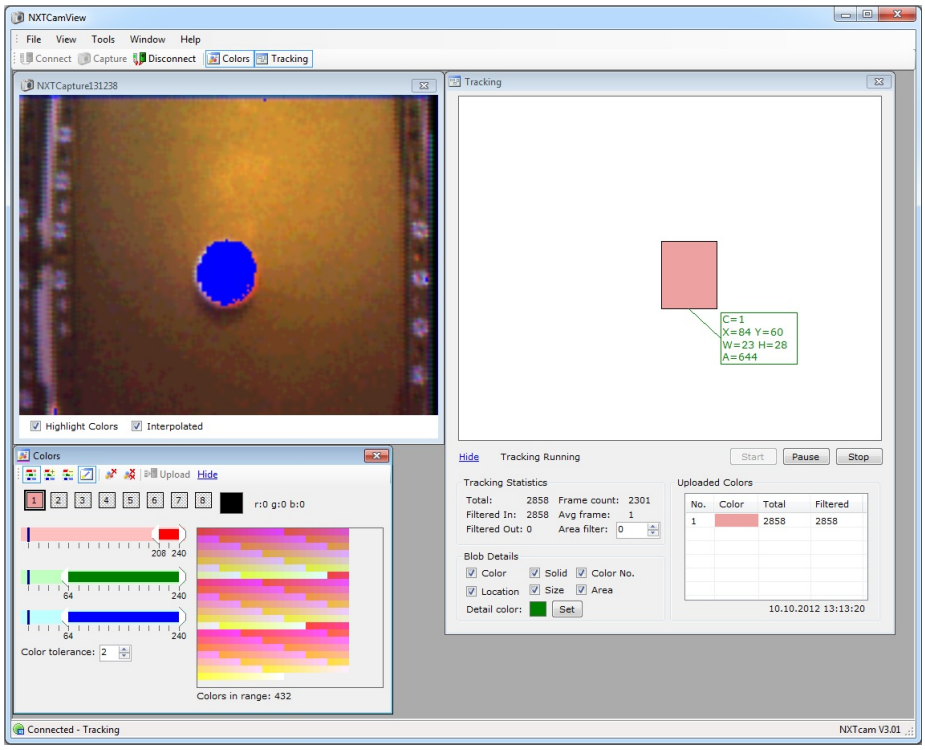


Abbildung 4.7: Die Benutzeroberfläche von NXTCamView

terstützt sowohl die älteren Roboter-Bausteine wie die RCX-Serie, den Scout, Cybermaster und Spybot als auch die neue Generation, die NXT-Serie. [19]

4.5.1 Installation

Zur Installation von Bricx muss zunächst die ausführbare Setupdatei von Sourceforge.net[5] heruntergeladen werden. Eine ausführliche Installationsanleitung findet man ebenfalls bei Sourceforge.net[6].
Wird das Bricx Command Center nun gestartet, sucht das Programm den NXT-Baustein, der mit dem mitgelieferten USB-Kabel am Rechner angeschlossen sein sollte. Der NXT-Baustein muss hierzu angeschaltet sein. Die Suchparameter können hierzu wie in Abbildung 4.8 beibehalten werden. Ist der NXT-Baustein

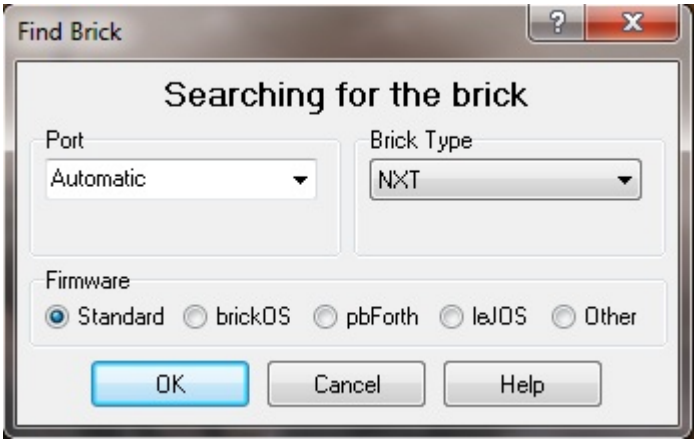


Abbildung 4.8: Auswahl des NXT-Bausteins

nicht am Rechner angeschlossen bzw. ausgeschaltet, kann das Bricx Command Center nicht in vollem Umfang genutzt werden. Es ist dennoch möglich, Programmcode zu schreiben und diesen kompilieren zu lassen. Möchte man nachträglich den NXT-Baustein mit dem Bricx Command Center verbinden, so kann man dies über die Schlatfläche *Tools* → *Find Brick* oder die Tastenkombination Strg+Shift+F3 tun.

[5] `http://sourceforge.net/projects/bricxcc/files/bricxcc/`, letzter Zugriff am 10.07.2012

[6] `http://bricxcc.sourceforge.net/`, letzter Zugriff am 10.07.2012.

4.5.2 Import benötigter Bibliotheken

Möchte man in seinen Programmen die NXTCam benutzen, so wird hierfür eine Bibliothek für NXC benötigt. Die Datei nxtcamlib-default.nxc kann von Mindsensors.com[7] heruntergeladen werden. Der Ordner kann anschließend in ein beliebiges Verzeichnis, z.B. C:\downloads, entpackt werden. Die Datei muss anschließend in den Ordner kopiert werden, in dem später auch die NXC-Programme abgespeichert werden. Der Befehl

```
1 #include "nxtcamlib-default.nxc"
```

Listing 4.1: Import der NXTCam Bibliothek

im Header des Programms inkludiert diese Datei anschließend in das gewünschte Programm, so dass dort die Bibliothek zur Verfügung steht.

4.5.3 Firmware-Update

Die Servomotoren der LEGO® Mindstorms® NXT-Wippe werden mithilfe von PID-Motorreglern gesteuert. Die vorinstallierte NXT-Firmware (v 1.31) unterstützt diese Befehle jedoch nicht. Um die PID-Motorsteuerung einzubinden wird die erweiterte NXT-Firmware (v 1.32) benötigt. Die erweiterte Firmware kann ebenfalls von Sourceforge.net[8] heruntergeladen werden. Der Ordner kann anschließend in ein beliebiges Verzeichnis, z.B. C:\downloads, entpackt werden. Im Bricx Command Center (der NXT-Baustein muss angeschlossen sein) kann man nun unter *Tools* → *Download Firmware* die Datei lms_arm_nbcnxc_132.rfw aus dem Ordner C:\downloads\lms_arm_nbcnxc als neue Firmware auswählen und durch Drücken des Öffnen-Buttons auf den NXT-Baustein laden. Hierbei wird die alte Firmware sowie alle bisher auf dem NXT-Baustein gespeicherten Dateien gelöscht. Anschließend können die Befehle des PID-Motorreglers ausgeführt werden.

[7] `http://www.mindsensors.com/index.php?module=documents&JAS_DocumentManager_op=downloadFile&JAS_File_id=1090`, letzter Zugriff am 10.07.2012.

[8] `bricxcc.sourceforge.net/lms_arm_nbcnxc.zip`, letzter Zugriff am 10.07.2012.

4.5.4 Datenübertragung

Nach dem Update der Firmware kann das zur automatischen Steuerung der Wippe verwendete Programm WIPPE.NXC auf den NXT-Baustein übertragen werden. Mithilfe des Öffnen-Buttons oder unter *File* → *Open...* kann der Programmcode im Editorfenster zur weiteren Bearbeitung geöffnet werden. Wie in Abschnitt 6.2 beschrieben wird, kann es ggf. nötig sein, das Offset zum Ausgleich schiefer Auflageflächen anzupassen. Anschließend kann das Programm über den Download-Button, die F6-Taste oder unter *Compile* → *Download* auf den NXT-Baustein geladen werden. Das Programm kann nun über den NXT-Baustein unter *My Files* → *Software Files* → *Wippe* ausgewählt und mittels *Run* oder unter *Compile* → *Run* bzw. durch Betätigen der F7-Taste gestartet werden. Es besteht auch die Möglichkeit, das Programm nach Beenden des Downloads automatisch unter *Compile* → *Download and Run* oder durch betätigen der Tastenkombination Strg+F5 zu starten.

Kapitel 5

Analyse des Wippe-Systems

Wie in Abschnitt 3.1 beschrieben unterliegt das Wippe-Experiment als Echtzeitsystem zeitlichen Bedingungen, die eingehalten werden müssen. Dies gilt natürlich auch für die Umsetzung des Versuchsaufbaus mit den Bausätzen der LEGO® Mindstorms® NXT-Serie. Im Folgenden werden die Latenzzeiten des Softwaresystems und des externen Systems ermittelt und analysiert, um die Ausführungszeiten der Arbeitsschritte abzuschätzen. Anschließend erfolgt die Verifizierung, ob die Latenzzeiten, denen das System unterliegt, den zeitlichen Bedingungen, die in Abschnitt 3.2 aufgestellt wurden, genügen.

5.1 Zeitbedingungen

Das technische System der Wippe stellt Zeitbedingungen, die das Messsystem, das Rechensystem und das Stellsystem einzuhalten haben. Dies betrifft die Startzeit, die Ausführungszeit und die Endzeit.
Durch Einsetzen maximaler Auslenkwinkel der Platte α_{Platte} in Gleichung 3.8 erhält man zunächst die maximale Beschleunigung einer massiven Kugel a_{max} und anschließend mit Gleichung 3.10 die minimale Zeit t_{min}, die diese benötigt, um bei voller Auslenkung das Plattenzentrum zu erreichen, wenn sie aus der Ruhelage vom Plattenrand aus startet (siehe Tabelle 5.1). Wird die Startzeit eines Zyklus der Arbeitsschritte mit $r = 0$ festgelegt, so ist die Endzeit durch den maximalen Auslenkwinkel der Platte und die Zeit, die die Kugel zum Erreichen des Plattenzentrums benötigt, mit $d = t_{min}(\alpha_{Platte})$ bestimmt.

α_{Platte}	a_{max}	t_{min}
15°	$1,814\ \frac{m}{s^2}$	$0,257\ s$
10°	$1,217\ \frac{m}{s^2}$	$0,314\ s$
9°	$1,096\ \frac{m}{s^2}$	$0,331\ s$
8°	$0,975\ \frac{m}{s^2}$	$0,351\ s$
7°	$0,854\ \frac{m}{s^2}$	$0,375\ s$
6°	$0,732\ \frac{m}{s^2}$	$0,404\ s$
5°	$0,611\ \frac{m}{s^2}$	$0,443\ s$

Tabelle 5.1: Die maximale Beschleunigung a_{max} und minimale Zeitspanne t_{min} zum Erreichen des Plattenzentrums bei gegebener Auslenkung α_{Platte}.

Die Ausführungszeit Δe setzt sich aus der Abarbeitungszeit eines vollständigen Zyklus zusammen. Ein vollständiger Zyklus besteht aus drei Arbeitsschritten, denen einzelne Ausführungszeiten zugeordnet werden können. Anders als im Arbeitszyklus der Original-Wippe (vgl. [21, S. 20 f.]) können den einzelnen Teilschritten sofort die Ausführungszeiten zugeteilt werden, da zum einen die Auswertung des Kamerabildes bereits vom Mikroprozessor der Kamera übernommen wird und somit keine Bildverarbeitung betrieben werden muss und zum anderen die Motorsteuerung unmittelbar durch den NXT-Baustein durchgeführt wird und keine Steuereinheit zwischengeschaltet ist. Die Arbeitsschritte der LEGO® Mindstorms® NXT-Wippe teilen sich wie folgt auf:

- Aufnahme des Kamerabildes und Bestimmung der Bounding Box der Kugel
  ($\Delta e_K \mathrel{\hat{=}}$ Alter des Kamerabildes / der Kugelposition)
- Berechnung des Einstellwinkels der Motoren
  ($\Delta e_R \mathrel{\hat{=}}$ Rechenzeit des Regelalgorithmus zur Berechnung des Einstellwinkels der Motoren)
- Auslenkzeit der Platte durch die Motoren
  ($\Delta e_M \mathrel{\hat{=}}$ Zeit, die die Motoren zum Ansteuern des Einstellwinkels benötigen)

Die Startzeiten der einzelnen Teilschritte sind für die Berechnung des Einstellwinkels und der Auslenkzeit der Motoren vom vorhergehenden Schritt abhängig. Die Berechnung des Einstellwinkels kann erst durchgeführt werden, wenn die Bounding Box der Kugel von der Kamera ermittelt wurde und die Auslenkung der Motoren kann erst beginnen, wenn der benötigte Einstellwinkel durch den Regelalgorithmus berechnet wurde. Lediglich die Aufnahme der Kamera kann periodisch ausgeführt werden. Somit ergibt sich für den Startzeitpunkt der Kameraaufnahme r_K, den Startzeitpunkt der Berechnung des Regelalgorithmus r_R und den Startzeitpunkt der Motorsteuerung r_M:

$$r_K = 0 \tag{5.1}$$

$$r_R = r_K + \Delta e_K \tag{5.2}$$

$$r_M = r_R + \Delta e_R \tag{5.3}$$

Die Rechtzeitigkeit des Systems wird dadurch gewährleistet, dass die Servomotoren den errechneten Einstellwinkel vor dem Endzeitpunkt angesteuert haben, also $d_M \leq d$. Die Endzeitpunkte der Dauer der Kameraaufnahme d_K und der Berechnung des Einstellwinkels d_R, mit $d_K < d_R < d_M$ sind demnach zu vernachlässigen. Folglich lautet die Zeitbedingung, die es einzuhalten gilt:

$$r_M + \Delta e_M \leq d_M \tag{5.4}$$

$$\Leftrightarrow \quad r_R + \Delta e_R + \Delta e_M \leq d_M \tag{5.5}$$

$$\Leftrightarrow \quad r_K + \Delta e_K + \Delta e_R + \Delta e_M \leq d_M \tag{5.6}$$

$$\text{mit } r_K = r = 0 \quad \Leftrightarrow \quad \Delta e_K + \Delta e_R + \Delta e_M \leq d_M \tag{5.7}$$

Die allgemeine Zeitbedingung ist demnach:

$$A \equiv r_K + \Delta e_K + \Delta e_R + \Delta e_M \leq d_M = d \tag{5.8}$$

Die allgemeine Zeitbedingung ist dann gültig, wenn sich der Regelalgorithmus bereits in der k-ten Durchführung, mit $k \geq 2$, befindet. Ist der Regelalgorithmus noch in der ersten Durchführung, kann die Geschwindigkeit der Kugel, z.B. nach einem Einwerfen der Kugel oder, weil die Platte bereits ausgelenkt ist, noch nicht bestimmt werden. Hierfür werden zunächst zwei Positionen der Kugel benötigt, um die Geschwindigkeit zu berechnen. Dies erfordert sowohl

zwei Kameraaufnahmen als auch zwei Durchläufe des Regelalgorithmus.
Die angepasste allgemeine Zeitbedingung lautet demnach:

$$A \equiv r_K + 2 \cdot \Delta e_K + 2 \cdot \Delta e_R + \Delta e_M \leq d_M = d \tag{5.9}$$

5.2 Latenzzeiten des Softwaresystems

Das Softwaresystem bildet mit dem Regelalgorithmus das Kernstück der LEGO® Mindstorms® NXT-Wippe. Im Regelalgorithmus werden aus den Koordinaten der Bounding Box des verfolgten Objektes dessen Position und Geschwindigkeit ermittelt und daraus der Einstellwinkel berechnet, der benötigt wird, um das Objekt abzubremsen (siehe Abschnitt 6.1). Die Latenzzeiten des Softwaresystems umfassen nur den Regelalgorithmus, da der Prozess der Bildverarbeitung in vollem Umfang von der NXTCam übernommen wird. Der Steuerbefehl der Motoren benötigt ebenfalls keine weitere Zeit, da die Servomotoren unmittelbar an der Recheneinheit, dem NXT-Baustein, angeschlossen sind und unmittelbar angesteuert werden.
Die Latenzzeit des Regelalgorithmus wurden mittels des Programms ALGORITHM_LATENCY.NXC (vgl. Anhang B.1) ermittelt. Das Programm enthält eine Endlosschleife mit einem Schleifenzähler n. Pro Schleifendurchlauf werden zunächst vier ganzzahlige Werte x_1, x_2, y_1, $y_2 \in \mathbb{N}$ mit $16 \leq x_1 < x_2 \leq 160$ und $0 \leq y_1 < y_2 \leq 144$ durch einen Zufallsgenerator erzeugt. Dies simuliert das Verfolgen eines Objektes durch die NXTCam. Anschließend wird der Regelalgorithmus ohne weitere Einschränkungen ausgeführt und die Einstellwinkel berechnet, wobei vor und nach der Ausführung die Laufzeit des NXT-Bausteins abgenommen wird, um die Dauer eines Schleifendurchlaufs zu bestimmen. Diese Zeiten werden aufsummiert $sum(t_R)$ und das Maximum $max(t_R)$ daraus gebildet. Nach einem Testzeitraum t_{Test} von einer Minute stoppt ein Kontrolltask die Endlosschleife. Der Schleifenzähler wird ausgelesen und es kann der Quotient $\frac{sum(t_R)}{n}$ gebildet werden. Der berechnete Wert ergibt die durchschnittliche Latenzzeit des Regelalgorithmus. Sie beträgt $0,002$ s. Das Maximum der Latenzzeiten beträgt ebenfalls $0,002$ s.
Die Ausführungszeit des Rechensystems ist demnach:

$$\Delta e_R = 0,002 \ s$$

5.3 Latenzzeiten des externen Systems

Genauso wie die Latenzzeiten des Softwaresystems spielen auch die Latenzzeiten des externen Systems eine entscheidende Rolle. Im Versuchsaufbau Wippe stellt die Kamera als Schnittstelle zwischen technischen System und Rechensystem und die Servomotoren, die die Platte auslenken, als Schnittstelle zwischen Rechensystem und technischen System das externe System dar. Die Latenzzeiten von Mess- und Stellsystem werden getrennt von einander ermittelt.

5.3.1 Kameralatenz

Die Latenzzeiten der Kamera Δt_K wurde mithilfe des Hilfsprogramms CAMERA_LATENCY.NXC (vgl. Anhang B.2) ermittelt. Die NXTCam übernimmt bereits das PreProcessing des aufgenommenen Bildes und liefert die Bounding Box der verfolgten Objekte (siehe Abschnitt 4.4). Somit ist es möglich die Zeitspanne dieses Prozesses zu ermitteln, indem die Laufzeit des NXT-Bausteins vor und nach dem Befehl abgenommen und anschließend die Differenz daraus gebildet wird. Hieraus kann sowohl der Durchschnittliche Wert der Kameralatenz gebildet als auch ein Maximum bestimmt werden. In einer Messreihe wurde jeweils in einem Testzeitraum t_{Test} von einer Minute die Dauer aller Aufnahmen $sum(t_K)$ aufsummiert und die Anzahl der Schleifendurchläufe n gezählt. Der Quotient $\frac{sum(t_K)}{n}$ bildet die durchschnittlich benötigte Zeit $avg(t_K)$ eines Schleifendurchlaufs. Ferner wurde die maximal benötigte Zeit $max(t_K)$ ermittelt. Die Kameralatenz kann nach Auswertung der Daten (siehe Tabelle 5.2) mit $t_{K_{max}} = 0,062\ s$ abgeschätzt werden.
Die Ausführungszeit der Kameraaufnahme beträgt demnach:

$$\Delta e_K = 0,062\ s$$

5.3.2 Motorlatenz

Zur Ermittlung der Latenzzeiten der Servomotoren Δt_M wurde das Programm MOTOR_LATENCY.NXC (vgl. Anhang B.3) verwendet. Innerhalb des Testzeitraums t_{Test} von einer Minute wurde der NXT-Servomotor mit den Einstellungen, die dem Regelalgorithmus zu Grunde liegen, getestet. Die Servomotoren der x- und y-Achse wurden jeweils mit den möglichen maximalen Einstellwinkel

n	$avg(t_K)$	$max(t_K)$
1	$27,098\ ms$	$61\ ms$
2	$26,358\ ms$	$61\ ms$
3	$27,483\ ms$	$49\ ms$
4	$26,827\ ms$	$49\ ms$
5	$25,111\ ms$	$49\ ms$
6	$27,940\ ms$	$61\ ms$
7	$28,371\ ms$	$62\ ms$
8	$27,381\ ms$	$61\ ms$
9	$29,034\ ms$	$61\ ms$
10	$35,837\ ms$	$61\ ms$
$\emptyset$	$27,569\ ms$	$58,125\ ms$

Tabelle 5.2: Die durchschnittliche Kameralatenz kann bei guten Lichtverhältnissen mit $0,030\ s$ abgeschätzt werden. Die maximale Latenz beträgt bis zu $0,062\ s$.

entsprechend der Auslenkwinkel der Platte eingestellt. Bevor der Motorsteuerbefehl an den Servomotor gesendet wird, wird die Laufzeit des NXT-Bausteins abgenommen. Anschließend „hängt“ das Programm in einer Schleife bis der gewünschte Einstellwinkel des Motors erreicht ist. Anschließend wird nochmals die Laufzeit des NXT-Bausteins abgenommen und die Differenz gebildet. Aus den für die Auslenkung gemessenen Zeiten wurde sowohl der Durchschnitt gebildet, als auch das Maximum bestimmt (siehe Tabelle 5.3).
Die Ausführungszeit Δe_M ist abhängig vom maximalen Auslenkwinkel der Platte α_{Platte} und bestimmt durch das Maximum der Motorlatenzzeiten der x- und y-Achse:

$$\Delta e_M = max\left((avg(t_{M_x}), max(t_{M_x}), avg(t_{M_y}), max(t_{M_y}))\right)(\alpha_{Platte})$$

5.4 Verifizierung

Die Bestimmung der Ausführungszeiten der Kameraaufnahme, der Berechnung des Regelalgorithmus und der Auslenkzeit der Servomotoren, Δe_K, Δe_R und

α_{Platte}	$avg(t_{M_x})$	$max(t_{M_x})$	$avg(t_{M_y})$	$max(t_{M_y})$
15°	$0,369\ s$	$0,434\ s$	$0,375\ s$	$0,446\ s$
10°	$0,301\ s$	$0,353\ s$	$0,305\ s$	$0,359\ s$
9°	$0,287\ s$	$0,335\ s$	$0,288\ s$	$0,340\ s$
8°	$0,265\ s$	$0,305\ s$	$0,270\ s$	$0,314\ s$
7°	$0,249\ s$	$0,287\ s$	$0,255\ s$	$0,298\ s$
6°	$0,232\ s$	$0,256\ s$	$0,239\ s$	$0,272\ s$
5°	$0,215\ s$	$0,230\ s$	$0,221\ s$	$0,251\ s$

Tabelle 5.3: Die durchschnittliche und maximale Motorlatenz der x- und y-Achse bei gegebenem Auslenkwinkel der Platte α_{Platte}.

Δe_M, ermöglichen es nun zu verifizieren, ob das Wippe-Experiment mit LEGO® Mindstorms® NXT umsetzbar ist und wenn ja, mit welchem maximalen Auslenkwinkel der Platte dies möglich ist.
Um zu gewährleisten, dass die Kugel auf der Wippe rechtzeitig abgebremst werden kann, muss die angepasste allgemeine Zeitbedingung aus Gleichung 5.9 erfüllt sein. Mit den Voraussetzungen

$$\begin{aligned} r &= r_K = 0, \\ \Delta e &= 2 \cdot \Delta e_K + 2 \cdot \Delta e_B + \Delta e_M(\alpha_{Platte}), \\ \text{und} \quad d &= t_{min}(\alpha_{Platte}) \end{aligned}$$

erhält man Ausführungszeiten Δe für gegebene maximale Auslenkwinkel α_{Platte} (siehe Tabelle 5.4) und es kann überprüft werden ob die Bedingung

$$A \equiv r + \Delta e \leq d \tag{5.10}$$

erfüllt ist oder nicht.
Die Auswertung zeigt, dass die Bedingung A nur für Winkel $\alpha \leq 6°$ erfüllt ist. Für größere Winkel ist die LEGO® Mindstorms® NXT-Wippe nicht in der Lage den berechneten Einstellwinkel anzusteuern, bevor die aus der Ruhelage gestartete Kugel bei maximaler Auslenkung der Platte das Plattenzentrum erreicht. Dies würde bedeuten, dass die Kugel beim Überrollen des Plattenzentrums eine noch zu große Geschwindigkeit besitzt und gegen den Rand am entgegengesetz-

α	Δe	d	$A \equiv r + \Delta e \leq d$
15°	$0,574\ s$	$0,257\ s$	nicht erfüllt
10°	$0,487\ s$	$0,314\ s$	nicht erfüllt
9°	$0,468\ s$	$0,331\ s$	nicht erfüllt
8°	$0,442\ s$	$0,351\ s$	nicht erfüllt
7°	$0,426\ s$	$0,375\ s$	nicht erfüllt
6°	$0,400\ s$	$0,404\ s$	erfüllt
5°	$0,379\ s$	$0,443\ s$	erfüllt

Tabelle 5.4: Die LEGO® Mindstorms® NXT-Wippe kann nur für Winkel $\alpha \leq 6°$ die Bewegung der Kugel abbremsen.

ten Ende der Platte rollt, was die (feste) Zeitbedingung, beschrieben in Beispiel 3.1.3, verletzten würde.

Kapitel 6

Umsetzung des Regelalgorithmus

Der Regelalgorithmus ist das Kernstück der Softwarekomponente des Wippe-Experiments. Mithilfe des Regelalgorithmus wird die Position der Kugel auf der Platte erfasst, die Geschwindigkeit der Kugel ermittelt und anschließend der Einstellwinkel berechnet, der benötigt wird, um die Platte auf der Kugel auszubalancieren.
Zunächst wird der Entwurf des Regelalgorithmus beschrieben. Dies geschieht in Pseudocode. Im Anschluss daran erfolgt eine Beschreibung der Implementierung in einer für LEGO® Mindstorms® NXT entwickelten Programmiersprache.

6.1 Entwurf

Zu Beginn des Regelalgorithmus wird die Kugelposition anhand der Daten berechnet, die die NXTCam zur Verfügung stellt. Hierzu wird der Mittelpunkt der Kugel bestimmt. Die NXTCam liefert die Koordinaten der oberen linken Ecke sowie der unteren rechten Ecke der Bounding Box eines Objekts, wie in Abbildung 6.1 dargestellt. Die Orientierung der x-Achse verläuft dabei nach rechts, die Orientierung der y-Achse nach unten. Die Kamera hat eine maximale Auflösung von 176 $px \times 144$ px. Sie ist so montiert und zentriert, dass die Platte der Wippe die gesamte Höhe des Bildes einnimmt. Da die Breite des Kamerabildes

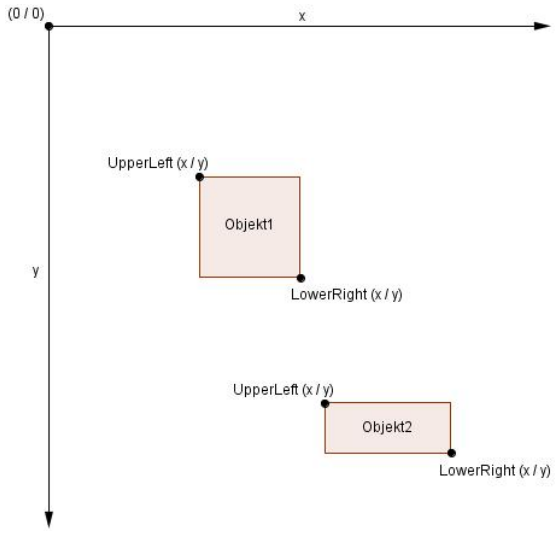


Abbildung 6.1: Die NXTCam liefert die Koordinaten der oberen linken und unteren rechten Ecke von erfassten Objekten. (Angefertigt mit GeoGebra)

größer ist als die Höhe entsteht an der linken und rechten Seite ein Rand mit einer Breite von 16 px. (Dies ist in Abbildung 6.2 erkennbar.) Das Zentrum der Platte befindet sich somit an der Koordinate $Z = (88/72)$. Um dieser Asymmetrie entgegenzuwirken, wird der x-Koordinate des ermittelten Zentrums diese 16 px abgezogen. Ferner soll die Kugelposition in Meter angegeben werden. Da die Bildhöhe exakt der Länge der Platte von 12 cm entspricht, ergibt sich hieraus der Faktor zur Berechnung der Kugelposition in Meter.

$$144\ px \mathrel{\hat{=}} 12\ cm \ \Leftrightarrow\ 1\ px \mathrel{\hat{=}} \frac{1}{12}\ cm \ \Leftrightarrow\ 1\ px \mathrel{\hat{=}} \frac{1}{1200} m \tag{6.1}$$

Die Berechnung der Koordinanten des Kugelzentrums sind somit für die x- und y-Koordinate verschieden.

```
1 xMitte = ((xLinks + xRechts) / 2 - 16) / 1200;
2 yMitte = ((yOben + yUnten) / 2) / 1200;
```

Listing 6.1: Berechnung der Kugelposition

Sollte die Kamera (z.B. aufgrund von schlechten Lichtverhältnissen, oder daweil die Kugel versehentlich von der Platte rollt) kein Objekt erkennen, wird der Regelalgorithmus auf die Einstellungen des ersten Zyklus zurückgesetzt und ein Warnton abgespielt.

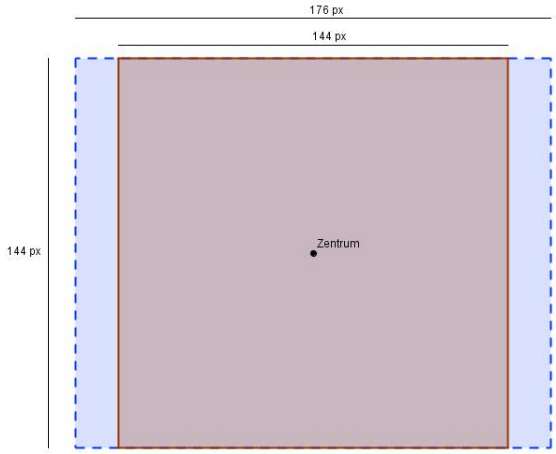


Abbildung 6.2: Die Platte (rot) ist exakt auf die Höhe des Kamerabildes (blau) zentriert. An der linken und rechten Seite entsteht jeweils ein Rand mit einer Breite von 16 *px*. (Angefertigt mit GeoGebra)

```
1 falls(Anzahl der erfassten Objekte = 0)
2   istErsterDurchlauf = wahr
3   xOld = 0
4   vxOld = 0
5   alphaX = 0
6   spiele Warnton
```

Listing 6.2: Fehlerbehandlung (Kein Objekt gefunden)

Da nun die Kugelposition in Meter in einem Koordinatensystem mit Koordinatenursprung in der oberen linken Ecke der Platte bestimmt ist, wird für die weitere Betrachtung des Regelalgorithmus ohne Beschränkung der Allgemeinheit lediglich die x-Achse betrachtet.

Im Anschluss zur Positionsbestimmung kann der Regelalgorithmus die aktuelle Geschwindigkeit und Beschleunigung der Kugel berechnen. Zur Berechnung der Geschwindigkeit werden zwei Kugelpositionen benötigt. Im ersten Zyklus kann somit lediglich die „alte“ Kugelposition gespeichert werden.

```
1 sonst
2   falls(istErsterDurchlauf){
```

```
3     istErsterDurchlauf = falsch
4     xOld = x
```

Listing 6.3: Erster Durchlauf

Erst im Zweiten Zyklus kann die zurückgelegt Strecke dsx und die aktuelle Geschwindigkeit vx berechnet werden. Außerdem wird erst hier die Distanz der Kugel zum Mittelpunkt der Platte dszx berechnet. Die zur Bestimmung der Geschwindigkeit verwendete Zeitspanne $\Delta t = 0,032\ s$ entspricht der Summe der durchschnittlichen Kameralatenz und der Latenz eines Zyklus des Regelalgorithmus (vgl. Abschnitt 5.2 und Abschnitt 5.3.1).

```
1 sonst
2   dsx = x - xOld
3   vx = dsx / 0.032
4   dszx = 0.06 - x
```

Listing 6.4: Berechnung benötigter Größen

Im Original-Aufbau der Wippe wird an dieser Stelle überprüft, ob sich der Regelalgorithmus noch im zweiten Zyklus befindet. Falls dies der Fall ist, wird die Beschleunigung der Kugel mit 0 initialisiert. Ab dem dritten Zyklus wird dann die Beschleunigung der Kugel durch den Quotienten der Differenz zweier Geschwindigkeiten und der dafür benötigten Zeit ermittelt, um zu prüfen, ob die Kugel auf der Platte balanciert werden kann oder nicht. Da die LEGO® Mindstorms® NXT-Wippe eine Bande um die Platte besitzt und die Echtzeitbedingungen angepasst wurden, wie in Beispiel 3.1.2 beschrieben, wird an dieser Stelle darauf verzichtet.

Der Regelalgorithmus ist so konzipiert, dass zunächst die Geschwindigkeit der Kugel auf 0 $\frac{m}{s}$ abgebremst wird, indem die Platte auf den Winkel eingestellt wird, bei dem die entgegenwirkende Beschleunigung die Kugel vor Erreichen des Plattenrandes abbremst. Die Kugel rollt anschließend in entgegengesetzter Richtung zurück und wird wieder abgebremst. Die zunächst in Abschnitt 3.2 vernachlässigten Kräfte wirken dabei stets auf die Kugel entgegen ihrer Bewegungsrichtung, so dass diese mit jedem Durchlauf langsamer wird.

In der Praxis passt sich der Einstellwinkel der Platte jedoch bereits während des

Abbremsens der Kugel der Momentangeschwindigkeit an, so dass die Kugel am Rand der Platte, also außerhalb des Plattenzentrums, zum Stillstand kommen kann. Ist dies der Fall, wird die Kugel gezielt zum Zentrum der Platte gelenkt. Dies geschieht, sobald die Geschwindigkeit eine Grenzgeschwindigkeit $v_x \leq 1 \frac{cm}{s}$ unterschreitet. Der verwendete Einstellwinkel der Platte, um die Kugel ins Plattenzentrum zu rollen, berechnet sich durch das Verhältnis des Abstandes der Kugel zum Plattenzentrum $\frac{dszx}{0{,}06}$ und beträgt maximal 3°. Aufgrund der Untersetzung des Getriebes muss der Faktor $i = 5$ multipliziert werden, so dass der mögliche Einstellwinkel der Motoren bis zu $3° \cdot i = 15°$ betragen kann.

```
1 falls(abs(vx) < 0.01)
2   alphaX = -(dszx / 0.06) * 5 * 3
```

Listing 6.5: Unterschreitung der Grenzgeschwindigkeit

Um die Kugel zunächst auf eine Geschwindigleit von 0 $\frac{m}{s}$ abzubremsen, muss der Einstellwinkel berechnet werden, bei dem die Kugel noch vor Erreichen des Plattenrandes zum Stillstand kommt. In Abschnitt 5.4 wurde gezeigt, dass bei einem maximalen Einstellwinkel von $\alpha_{max} = 6°$ maximal $0,404\ s$ verstreichen dürfen bis dieser Fall eintritt. Die maximale Geschwindigkeit, die die Kugel beim Passieren des Mittelpunkts der Platte hat, ist $v_{max} = 9,81 \frac{m}{s^2} \cdot \frac{5}{7} \cdot sin(6°) \cdot 0,404\ s = 0,296 \frac{m}{s}$. Sollen Geschwindigkeiten kleiner als v_{max} abgebremst werden, so kann der gesuchte Einstellwinkel mithilfe der Formeln 3.2 und 3.7 sowie dem in Abschnitt 3.2.6 verwendeten Trägheitsgesetz einer massiven Kugel berechnet werden.

$$v = a \cdot t = g \cdot \frac{5}{7} \cdot sin(\alpha) \cdot t \Leftrightarrow sin(\alpha) = \frac{v}{g \cdot \frac{5}{7} \cdot t} \Leftrightarrow \alpha = asin(\frac{v}{g \cdot \frac{5}{7} \cdot t}) \quad (6.2)$$

Die Variablen $g = 9,81 \frac{m}{s^2}$ und $t = 0,404\ s$ sind gegeben. Zusätzlich muss der Einstellwinkel mit dem Faktor $i = 5$ aufgrund der Untersetzung des Getriebes sowie dem Faktor $\frac{180}{\pi}$ wegen der Berechnung des *arcsin* im Radiantenmodus multipliziert werden.

Sollte der errechnete Winkel größer als der maximale Einstellwinkel $a_{max} = 6°$ sein (z.B. weil die Kugel mit einer größeren Geschwindigkeit eingeworfen wurde) wird der Einstellwinkel auf α_{max} bzw. $\alpha_{min} = -\alpha_{max}$ gesetzt.

```
1 sonst
```

```
2   alphaX = 5 * (180 / 3.14) * asin(vx / ((5.0 / 7.0) *
        9.81 * 0.404))
3   falls(alphaX > alphaMax)
4     alphaX = alphaMax
5   falls(alphaX < alphaMin)
6     alphaX = alphaMin
```

Listing 6.6: Berechnung des Einstellwinkels

Am Ende des Regelalgorithmus werden die Motorkontrollbefehle aufgerufen. Die Motorkontrollbefehle steuern die Servomotoren so, dass der errechnete Einstellwinkel exakt angesteuert und gehalten wird. Ferner wird überprüft, ob sich die Kugel innerhalb des Zentrums der Platte befindet ($dszx < 0,5\ cm$) und die Geschwindigkeit nahezu 0 ist ($v_x < 0,5\ \frac{cm}{s}$). Ist dies der Fall soll die Platte in Ruhelage gehalten werden. Dies soll ein Aufschaukeln der Kugel, bedingt durch kleine Korrekturen des Einstellwinkels und somit das sich Entfernen aus dem Plattenzentrum, verhindern.

```
1 falls(abs(dszx) >= 0.005 oder vx >= 0.005)
2   Stelle den Motorwinkel auf alphaX
3 sonst
4   Stelle den Motorwinkel auf 0
```

Listing 6.7: Motorsteuerung

6.2 Implementierung

Die Implementierung der LEGO® Mindstorms® NXT-Wippe erfolgte in der Programmiersprache NXC. NXC steht für **N**ot e**X**actly **C** (dt. nicht exakt C) und ist eine einfache Programmiersprache für den LEGO® Mindstorms® NXT-Bausatz, die auf der Programmiersprache C basiert [20, S. 1 f.].
Es stehen weitere Programmiersprachen zur Programmierung des LEGO® Mindstorms® NXT zur Verfügung. Die Roberta-Initiative[1] bietet eine umfangreiche Übersicht aller für den NXT zur Verfügung stehender Programmierspra-

[1] `http://www.roberta-home.de/de`, letzter Zugriff am 11.09.2012.

chen an. Nachfolgend erfolgt eine Kurzübersicht der wichtigsten Merkmale der vier markantesten Programmiersprachen [10, S. 9]:

NXT-G
- Grafische Benutzeroberfläche
- Arbeiten mit Variablen sehr umständlich
- Lizenz muss erworben werden[2]

leJOS
- Java-Derivat
- mit JVM-Firmware nutzbar
- nutzbar mit Entwicklungsumgebungen wie Eclipse, NetBeans, Javaeditor, die jedoch keine Unterstützung der LEGO® Mindstorms® NXT-Tools bieten
- kostenlos

nxtOsek
- ANSI C/C++
- mit nxtOsek-Firmware nutzbar
- nutzbar mit Entwicklungsumgebungen wie Eclipse
- fortgeschrittene Programmierkenntnisse benötigt
- kostenlos

NXC
- C-ähnlich
- Update der Firmware nur für PID-Motorsteuerung benötigt
- kostenlose Entwicklungsumgebung Bricx Command Center mit umfangreicher Unterstützung der LEGO® Mindstorms® NXT-Tools
- kostenlos

Weitere Programmiersprachen, wie etwa **LabVIEW**, **RoboLab 2.9**, **Robot C** oder **GNAT GPL** müssen entweder über eine Einzelplatz- oder Schullizenz erworben werden oder wenden sich an fortgeschrittene Programmierer und bieten somit für Schülerinnen und Schüler weniger Zugang [10, S. 9 ff.].
Die Hauptkriterien für den Einsatz von NXC sind

[2] Die NXT-G Lizenz ist in den LEGO® Mindstorms® NXT Education-Bausätzen im Gegensatz zu den Standard-Bausätzen nicht enthalten.

- eine einfache und leicht zu bedienende Programmierumgebung, die sämtliche Funktionen des NXT-Bausteins unterstützt,
- die hervorragende Dokumentation der Programmiersprache,
- die Unterstützung der Kamera- und PID-Motorkontrollbefehle sowie
- eine einfache Portierung des Programmcodes der Original-Wippe.

Nach der Wahl der Programmiersprache kann der Regelalgorithmus implementiert werden. Die Steuerung der Wippe umfasst jedoch mehr als nur den reinen Regelalgorithmus. Zunächst erfolgt die Kalibrierung der NXT-Servomotoren und die Initialisierung der NXTCam. Das Vorgehen wird exemplarisch anhand der x-Achse beschrieben und kann analog auf die y-Achse übertragen werden.

6.2.1 Kalibrierung der Servomotoren

Die Kalibierung der Servomotoren ist nach dem Start des Wippe-Programms notwendig, da es möglich ist, dass die Platte in einer Position ungleich der Nullstellung zur Ruhe kommt, nachdem ein vorheriger Durchgang des Wippe-Experimentes beendet wurde. Während der Initialisierung der PID-Motorregler wird jedoch der aktuelle Winkel der Motoren als Nullstellung herangezogen. Die Ansteuerung der berechneten Einstellwinkel ist somit nicht möglich und macht eine Kalibrierung erforderlich.
Nach der Initialisierung der benötigten Variablen dreht sich der Motor zunächst in positiver Richtung mit langsamer Geschwindigkeit. Dabei wird ständig der Winkel des Motors erfasst und gespeichert, um den sich der Motor bisher gedreht hat. Kann sich der Motor nicht weiter drehen, da der Ausleger durch die in der Wippe eingebaute Sperre behindert wird, entspricht der neu erfasste Winkel dem zuvor gespeicherten. Die Variable $isMotorOverload$ wird auf true gesetzt, der Motor dreht sich nicht weiter und der aktuelle Winkel wird in der Variablen $xMax$ gespeichert. Anschließend erfolgt derselbe Ablauf in negativer Richtung und der Winkel wird in der Variablen $xMin$ gespeichert. Abschließend wird der Motor um $\left(\frac{xMax-xMin}{2}\right)^\circ$ gedreht. Zusätzlich wird an dieser Stelle ein Offset mit eingerechnet (vgl. Listing B.4, Zeile 41 und Listing 6.8, Zeile 35), das einer etwaigen Schieflage der Auflagefläche der Wippe entgegenwirkt. Das Offset entspricht dem Winkel in Grad, um den die Auflagefläche der Wippe entlang der x-

und y-Achse geneigt ist, und wird zu dem berechneten Drehwinkel addiert. Bedingt durch die Anbringung der Motoren führt eine Erhöhung des x-Offsets zu einer Vergrößerung des Auslenkwinkels entlang der x-Achse und eine Erhöhung des y-Offsets zu einer Verringerung des Auslenkwinkels enlang der y-Achse.
Entspricht der aktuelle Winkel dem gewünschten wird der PID-Regler, der zur absoluten Positionsregulierung verwendet wird, mit dem aktuell eingestellten Winkel als Nullstelle initialisiert. Hierfür werden die entsprechenden Werte des P-, I- und des D-Glieds benötigt. Außerdem werden die maximale Geschwindigkeit und Beschleunigung der Motoren eingestellt und die Regulierungszeit des PID-Reglers festgelegt. Eines der größten Probleme zur präzisen Steuerung der LEGO® Mindstorms® NXT-Wippe stellte das Ermitteln der PID-Werte dar. Anders als bei handelsüblichen PID-Reglern stehen für die PosReg-Befehle in NXC nicht Werte von 0 bis 100 zur Verfügung sondern lediglich Werte zwischen 0 und 7. Der NXC Programmers Guide [20] bietet jedoch keine „Übersetzung“ der Werte, so dass es nicht möglich ist, übliche Verfahren zur Bestimmung der PID-Werte, wie zum Beispiel die Einstellregeln nach Ziegler/Nichols [11, S. 151 ff.] oder Chien/Hrones/Reswick [11, S. 153 f.], zu verwenden. Insgesamt existieren für die PosReg-Befehle $8^3 = 512$ Einstellungsmöglichkeiten, von denen einige kategorisch ausgeschlossen werden konnten. Nach einer großen Anzahl von Testreihen wurden als für den Versuch optimale Einstellung die Werte 5, 4, 5 gewählt (vgl. Listing B.4, Zeile 32).

```
1  task initPosReg(){
2
3    bool isMotorOverload;
4    int power;
5    int newRotationCount, oldRotationCount;
6    int xMax, xMin, position;
7
8    isMotorOverload = false;
9    newRotationCount = 0, oldRotationCount = 0;
10   while(!isMotorOverload){
11     OnFwd(xAxis, 15);
12     Wait(100);
```

```
13      newRotationCount = MotorRotationCount(xAxis);
14      if(oldRotationCount == newRotationCount){
15        isMotorOverload = true;
16      }
17      oldRotationCount = newRotationCount;
18    }
19    xMax = newRotationCount;
20
21    isMotorOverload = false;
22    newRotationCount = 0, oldRotationCount = 0;
23    while(!isMotorOverload){
24      OnRev(xAxis, 15);
25      Wait(100);
26      newRotationCount = MotorRotationCount(xAxis);
27      if(oldRotationCount == newRotationCount){
28        isMotorOverload = true;
29      }
30      oldRotationCount = newRotationCount;
31    }
32    xMin = newRotationCount;
33    position = (xMax–xMin)/2;
34
35    RotateMotorPID(xAxis, 15, position + xOffset, 30, 50,
          90);
36    Wait(100);
37    PosRegEnable(xAxis, PID_P, PID_I, PID_D);
38    PosRegSetMax(xAxis, maxSpeed, maxAcceleration);
39
40    SetMotorRegulationTime(10);
41
42  }
```

Listing 6.8: Kalibrierung der Motoren am Beispiel der x-Achse

Sind die x- und y-Achse kalibriert kann die Kamera initialisiert werden.

6.2.2 Initialisierung der Kamera

Mithilfe der in Abschnitt 4.4 beschriebenen Bibliotheken genügt ein einzelner Befehl, um die Kamera zu initialisieren. Schlägt die Initialisierung fehl (da zum Beispiel das Verbindungskabel nicht angeschlossen ist), wird ein Warnton wiedergegeben und das gesamte Programm gestoppt. Sinnvollerweise sollte aus diesem Grund die Initialisierung der Kamera vor der Kalibrierung der Motoren stattfinden. Versuche haben jedoch gezeigt, dass sich die Kamera während des Programmablaufs nach einer Ruhephase von einigen Sekunden abschaltet und somit erneut initialisiert werden muss. Die relativ lange Kalibrierungsphase der Servomotoren macht es notwendig, dass die Kamera erst kurz bevor der Regelalgorithmus gestartet wird, initialisiert wird.

```
1  task initCam(){
2    int init=0;
3    init = NXTCam_Init(camPort, CAMADDR);
4    if (init != 1) {
5      PlayTone(330, 100); Wait(200);
6      PlayTone(330, 100); Wait(200);
7      PlayTone(330, 100); Wait(200);
8      StopAllTasks();
9    }
10 }
```

Listing 6.9: Initialisierung der Kamera

Wurde die Kamera erfolgreich initialisiert, kann der Regelalgorithmus gestartet werden.

6.2.3 Regelalgorithmus

Zu Beginn des Regelalgorithmus wird mithilfe des Befehls NXTCam_GetBlobs die Anzahl aller erfassten Objekte, deren Farbcode (Werte zwischen 1 und 8, gemäß den Einstellungen in NXTCamView) und die Koordinaten der Bounding Box in entsprechende Arrays gespeichert. Sollte kein Objekt erfasst werden, wird

die Variable *isFirstRun* auf true gesetzt (vgl. Abschnitt 6.1) und ein Warnton abgespielt.

```
1  NXTCam_GetBlobs(camPort, nblobs, color, left, top, right,
       bottom);
2  x = ((((right[0] + left[0]) / 2) - 16) / 1200);
3  y = ((bottom[0] + top[0]) / 2) / 1200;
4  if(nblobs == 0){
5    isFirstRun = true;
6    xOld = yOld = 0;
7    vxOld = vyOld = 0;
8    alphaX = alphaY = 0;
9    PlayTone(110, 10);
10 }
```

Listing 6.10: Erfassung der Bounding Box

Wurde ein Objekt erfasst, wird die Berechnung des Einstellwinkels innerhalb des Regelalgorithmus, wie in Abschnitt 6.1 beschrieben, durchgeführt.

```
1  else{
2    if(isFirstRun){
3      isFirstRun = false;
4      xOld = x;
5    }
6    else{
7      dszx = 0.06 - x;
8      dsx = x - xOld;
9      vx = dsx / 0.032;
10
11     if((abs(vx) < 0.01)){
12       alphaX = - (dszx / 0.06) * alpha_v0;
13     }
14     else{
15       alphaX = 5 *(180 / 3.14) * asin(vx / ((5.0 / 7.0) *
             9.81 * 0.404));
```

```
16          if(alphaX > alpha_max_pos){
17            alphaX = alpha_max_pos;
18          }
19          if(alphaX < alpha_max_neg){
20            alphaX = alpha_max_neg;
21          }
22        }
23        xOld = x;
24        vxOld = vx;
25        ...
26      }
27    }
```

Listing 6.11: Regelalgorithmus

Die Steuerung der Servomotoren erfolgt anschließend durch einen einzigen Befehl. Die PosReg-Befehle (*Position Regulation*-Befehle) ermöglichen es, die NXT-Servomotoren um einen festen Winkel zu drehen oder gezielt einen bestimmten Winkel, gemessen von der Nullstellung des Motors aus, anzusteuern. Der gewünschte Einstellwinkel kann bei Stillstand des Motors, aber auch während der Drehung, geändert werden. [20, S. 305 ff.]

```
1  if((abs(dszx) >= 0.005) || (vx >= 0.005)){
2    PosRegSetAngle(xAxis, alphaX);
3  }
4  else{
5    PosRegSetAngle(xAxis, 0);
6  }
```

Listing 6.12: Motorsteuerung

Wurde der Befehl zur Motorsteuerung ausgeführt, startet der nächste Zyklus des Regelalgorithmus. Parallel dazu wird der PosReg-Befehl weitergeführt. Innerhalb der Zeitspanne $\Delta e_K + \Delta e_B = 0,032\ s$ wird der Motor mit einer Motorregulierungszeit von $0,010\ s$ (vgl. Listing 6.8, Zeile 40) nachgestellt. Dies bedeutet, dass alle $0,01\ s$ die Position des Motors überprüft und nachgestellt

wird. Innerhalb eines Zyklus des Regelalgorithmus wird der Motor somit drei Mal nachgesetellt, bevor der nächste berechnete Einstellwinkel angesteuert wird. Dies gewährleistet eine sehr präzise Steuerung der LEGO® Mindstorms® NXT-Servomotoren.

6.2.4 Zusammenführung

Der in NXC implementierte Regelalgorithmus sowie die Kalibrierung der Motoren und Initialisierung der Kamera wurden im Programm WIPPE.NXC (vgl. Listing B.4) zusammengeführt. Anschließend erfolgte eine im nachfolgenden Kapitel dokumentierte Testphase.

Kapitel 7

Dokumentation der Testphase

Die Testphase der LEGO® Mindstorms® NXT-Wippe kann in zwei Unterphasen eingeteilt. In einem ersten Schritt wurden durch Testreihen die benötigten Einstellungen der mechanischen Komponenten identifiziert. Im zweiten Schritt erfolgte der Test des Gesamtsystems inklusive der Softwarekomponente in Bezug auf die Echtzeitfähigkeit des Systems.

7.1 Testphase A

(während der Entwicklung)

Zu Beginn der Arbeit an der LEGO® Mindstorms® NXT-Wippe wurde ein Rohentwurf des Regelalgorithmus entwickelt. Dieser stellte eine vereinfachte Version des Regelalgorithmus dar und diente dazu, die Grundfunktionen des Aufbaus zu testen. Die Grundfunktionen umfassen das Erfassen der Kugel mithilfe der NXTCam, das Feststellen der Bewegungsrichtung und -geschwindigkeit der Kugel sowie die Auslenkung der Platte um einen festen Winkel.
Während des Aufbaus der LEGO® Mindstorms® NXT-Wippe wurde der Rohentwurf verwendet, um die optimale Positionierung der Kamera zu ermitteln und die Drehrichtung und -geschwindigkeit der Servomotoren einzustellen. In diesem Schritt wurden die in Abschnitt 4.2 zusammengefassten Änderungen am Versuchsaufbau vorgenommen.

Nach Abschluss der Entwicklung des Versuchsaufbaus konnte der Regelalgorithmus implementiert und partiell getestet werden. Diese Tests gestalteten sich schwierig, da aussagekräftige Ergebnisse aufgrund fehlender Informationen nicht ableitbar waren. Zur Ermittlung der Geschwindigkeit der Kugel werden die Latenzzeiten des internen und externen Systems benötigt. Die präzise Steuerung der Servomotoren ist nur mithilfe der korrekten PID-Werte möglich. Zur Bestimmung dieser Werte wird jedoch ein funktionsfähiger Regelalgorithmus vorausgesetzt.
Aus diesem Grund wurde während der Implementationsphase zunächst eine Testreihe zur Ermittlung der benötigten PID-Werte durchgeführt. Das Augenmerk lag dabei auf einer möglichst schnellen Beschleunigung der Servomotoren, um die Motorlatenz zu minimieren und einem weichen Abbremsen, um ein Übersteuern des Einstellwinkels und damit einhergehende Schaukelbewegungen zu vermeiden.
Abschließend konnten, wie in Abschnitt 5.2 und 5.3 beschrieben, die Latenzzeiten des internen und externen Systems in mehreren Testreihen ermittelt werden. Die Ermittlung der Latenzzeiten des Softwaresystems erfolgte dabei unter Verwendung geschätzter Zeitperioden, die in den Regelalgorithmus eingesetzt wurden. Diese Werte lagen jedoch in der Größenordnung der durch diese Versuche ermittelten Werte und verfälschen somit die Testergebnisse nicht.

7.2 Testphase B

(nach Fertigstellung)

Nach Abschluss der Datenerfassung und Auswertung der Latenzzeiten wurden alle ermittelten Werte in den in Kapitel 6 beschriebenen Regelalgorithmus eingesetzt und das Programm unter optimalen Bedingungen gestartet. Die Ausführung des Programms zeigt nicht nur, dass der Regelalgorithmus korrekt, die von ihm geforderte Aufgabe erfüllt, sondern auch, dass die in Abschnitt 5.4 verifizierten Zeitbedingungen aus Abschnitt 5.1 in einem Großteil der Fälle eingehalten werden.

Kapitel 8

Zusammenfassung und Ausblick

Im Zuge dieser Masterarbeit wurde das Wippe-Experiment, welches innerhalb der Arbeitsgruppe Echtzeitsysteme an der Universität Koblenz-Landau, Fachbereich Informatik, umgesetzt wurde, auf den LEGO® Mindstorms® NXT übertragen.

Zunächst wird das bisher entwickelte System in seinem derzeitigen Zustand vorgestellt und dessen Funktionsweise erläutert. Abgeleitet daraus ergeben sich die Anforderungen, die an die Umsetzung des Experiments mit LEGO® Mindstorms® NXT, gestellt werden.

In einem ersten großen Schritt wird der Aufbau des Wippe-Experiments aus dem LEGO® Mindstorms® Basis- und Ergänzungsset und der NXTCam dokumentiert. Dies umfasst die Entwicklung mehrerer Versionen der Wippe, um auftretenden Problemen, sowohl bedingt durch den Aufbau als auch durch äußere Gegebenheiten, entgegenzuwirken. Der somit geschaffene Versuchsaufbau legt den Grundstein für das weitere Vorgehen.

Die Analyse des Wippe-Systems erfolgt unter den Rahmenbedingungen, die der Versuchsaufbau stellt. Bedingt durch die geringe Größe der Platte unterliegt die Wippe sehr strikten Zeitbedingungen. Die Latenzzeiten des Softwaresystems und der Kamera können als feste Werte betrachtet werden. Die Latenzzeiten der Servomotoren sind jedoch vom maximalen Auslenkwinkel der Platte abhängig.

Bei einem Auslenkwinkel von bis zu 6° können die Zeitbedingungen eingehalten und die Kugel abgefangen und gestoppt werden.
Der Regelalgorithmus, der für die automatische Steuerung der Wippe benötigt wird, kann in großen Teilen aus dem Quellcode der Original-Wippe exportiert werden. Dies liegt zum Einen an der Wahl der Programmiersprache, in der der NXT-Baustein programmiert wird. Die Programmiersprache NXC orientiert sich sehr stark an der Sprache C, in welcher die Original-Wippe programmiert wurde. Unterstützt wurde dies zum Anderen durch den modularen Aufbau des Quellcodes der Original-Wippe, was eine einfache Extraktion ermöglichte. Die tatsächliche Implementierung des Regelalgorithmus macht es jedoch erforderlich, eigene Ideen zur Problemlösung zu entwickeln. Besonders die Kalibrierung und präzise Steuerung der Motoren erfordern den Einsatz unterschiedlicher Steuerbefehle und dadurch bedingt den mehrfachen Umbau der LEGO® Mindstorms® NXT-Wippe.

Die an diese Arbeit gerichteten Ziele konnten allesamt erreicht werden. Das Wippe-Experiment konnte mithilfe eines LEGO® Mindstorms® NXT Education-Bausatzes nachgebaut werden. Die Funktionalität konnte durch Tests gezeigt werden. Vor allem aber - unter Verwendung eines leicht vereinfachten physikalischen Modells - konnte die zeitliche Korrektheit des Versuchsaufbaus nachgewiesen werden. Eine Bauanleitung der LEGO® Mindstorms® NXT-Wippe sowie die ausführliche Dokumentation des Programmcodes ermöglichen es ferner das Wippe-Experiment schnell und einfach an Schulen, die über die LEGO® Mindstorms® NXT-Bausätze sowie die NXTCam verfügen, nachzubauen und es für den Unterrichtseinsatz im Themengebiet Echtzeitsysteme zu nutzen.

Eine Weiterentwicklung des Versuchsaufbaus der LEGO® Mindstorms® NXT-Wippe kann unter folgenden Gesichtspunkten vorgenommen werden:
Es besteht die Möglichkeit die Aufhängung der platte zu verbessern. Im bisherigen Aufbau ist die Platte in einer um die x- und y-Achse beweglichen Aufhängung gelagert. Bedingt durch die Größe der Platte und der Servomotoren ist der Motor zur Steuerung der y-Achse jedoch, anderes als im

Original-Aufbau der Wippe, statisch am Gestell der Wippe befestigt, so dass bei gleichzeitiger Auslenkung der x- und y-Achse der Auslenkwinkel der Platte vom berechneten Winkel abweicht. Eine Möglichkeit zur Vermeidung dieses Problem besteht darin, die Platte zentral auf einem Kugellager zu lagern und beide Servomotoren statisch anzubringen, insofern die Verbindungselemente zwischen Motoren und Platte exakt auf den Mittelachsen der Platte liegen.
Einen weiteren Verbesserungsaspekt bietet die Verwendung einer handelsüblichen Webcam, die die bisher verwendete NXTCam ablöst. Dies hat zwei Gründe:
Zum Einen bieten Webcams meist eine deutlich höhere Auflösung als die von der NXTCam zur Verfügung gestellte. Dies ermöglicht eine exaktere Bestimmung der Kugelposition beziehungsweise ermöglicht die Verwendung einer größeren Platte, um die maximale Auslenkung dieser zu erhöhen.
Zum Anderen stellt die NXTCam neben den NXT-Baukästen einen hohen Kostenfaktor dar. Bei einem ohnehin meist knappen Budget vieler Schulen liegt die Anschaffung eines Klassensatz Webcams in etwa der Höhe des Anschaffungspreis von zwei NXTCams. Da die NXT-Firmware die benötigten Protokolle zur Datenübertragung unterstützt, stellt dies kein Problem dar.
Für die Verwendung einer Webcam wird ein Rechner zur Bildverarbeitung benötigt [1]. Die Verwendung eines eingebetteten Kamerasystems macht die Programmierung eines Mikrocontrollers notwendig, ermöglicht jedoch die Loslösung vom Rechner [5].
Es besteht außerdem die Möglichkeit, die Funktionalität der LEGO® Mindstorms® NXT-Wippe zu erweitern. Es ist vorstellbar, dass der Regelalgorithmus dahingehend erweitert wird, so dass die Kugel auf eine beliebige Position auf der Platte gesteuert werden kann. Auch denkbar ist es, die Kugel entlang bestimmter Bahnen, z.B. entlang einer Kreisbahn, zu steuern.
Eine weitere Ergänzung der Funktionalität ist der Einbau einer manuellen Steuerung der Wippe, wie sie auch im Original-Aufbau verwendet wird. Eine intuitive Steuerung kann beispielsweise durch Verwendung der Nintendo® Wii™-Eingabegeräte erfolgen. Die sogenannte Wii™ Remote-Fernbedienung verfügt über eine Bluetooth-Schnittstelle, die jedoch nicht kompatibel mit der Bluetooth-Schnittstelle des NXT-Bausteins ist. Dies macht den Einsatz

eines Rechners als Übersetzer notwendig. Eine Alternative bietet das Wii™ NunChuck, das mit einer I^2C Verbindung mit der Wii™ Remote kommuniziert. Mithilfe einiger Umbaumaßnahmen kann diese Verbindung zur Kommunikation mit dem NXT-Baustein genutzt werden. Für beide Umsetzungen existieren Tutorials, wie zum Beispiel eine Umsetzung mit der Wii™ Remote der Universität Klagenfurt[1] und eine Umsetzung mit Wii™ NunChuck von Sebastian Deppisch[2].

Besonders der letztgenannte Aspekt, die Umsetzung einer manuellen Steuerung, könnte in Schulen sinnvoll genutzt werden, um die Parallelität zwischen menschlicher Auffassungsgabe und Steuerung und einem Echtzeitsystem zu verdeutlichen. Die Augen nehmen den Platz der Sensorik ein, das Gehirn stellt das Rechensystem dar und die Hände bedienen die Aktorik des Systems. Eine solche manuelle Steuerung kann verdeutlichen, welche Leistungen ein funktionierendes Echtzeitsystem bewältigt und bietet einen abwechslungsreichen methodischen Einstieg in das Themengebiet Echtzeitsysteme.

[1] `http://unikluni.uni-klu.ac.at/?p=154`, letzter Zugriff am 24.09.2012.

[2] `http://sebastian-deppisch.de/index.php?page=nxt-3d-nunchuck`, letzter Zugriff am 24.09.2012.

Anhang A

Bauanleitung

Auf den folgenden Seiten findet sich eine Liste aller benötigter Bauteile, die für das Erstellen der LEGO® Mindstorms® NXT-Wippe benötigt werden und eine ausführliche bebilderte Bauanleitung. Die Anleitung wurde mithilfe des Tools LEGO Digital Designer (LDD) erstellt. Das Programm kann unter Lego.com[1] heruntergeladen werden. Es ist möglich, dass einige Bauschritte nicht vollkommen korrekt ausgeführt werden können oder rückgängig gemacht werden müssen, da der LDD ein „flexibles“ Zusammensetzen beziehungsweise ein leichtes Dehnen der Bausteine nicht betrachtet.

[1] `http://ldd.lego.com/`, letzter Zugriff am 16.09.2012.

2 x		4297008	Tacho Motor - Klares Orange,Dunkel Steingrau,Helles Steingrau	1 x			NXT - Schwarz,Klares Orange,Grün Sandfarbe,Mittel Steingrau,Dunkel Steingrau,Helles Steingrau,Cool Silber,	7 x		4210751	TECHNIC 3M BEAM - Dunkel Steingrau
3 x		4210686	TECHNIC 5M BEAM - Dunkel Steingrau	10 x		4210752	TECHNIC 7M BEAM - Dunkel Steingrau	2 x		4210667	TECHNIC ANG. BEAM 4X2 90 DEG - Dunkel Steingrau
10 x			TECHNIC 9M BEAM - Dunkel Steingrau	3 x		4535773	T-BEAM 3X3 W/HOLE Ø4.8 - Dunkel Steingrau	6 x		4210755	TECHNIC 11M BEAM - Dunkel Steingrau
8 x		4261932	TECHNIC 13M BEAM - Dunkel Steingrau	12 x		4210687	TECHNIC 15M BEAM - Dunkel Steingrau	8 x		4210753	TECHNIC ANG. BEAM 3X5 90 DEG. - Dunkel Steingrau
7 x		4210668	DOUBLE ANGULAR BEAM 3X7 45° - Dunkel Steingrau	4 x		4239896	HALFBEAM CURVE 3X5 - Dunkel Steingrau	10 x		4239601	1/2 BUSH - Klares Gelb
2 x		273626	BALL W. CROSS AXLE - Schwarz	116 x		278026	CONNECTOR PEG W. FRICTION - Schwarz	1 x		4211807	CONNECTOR PEG - Mittel Steingrau

2 x  4211815 CROSS AXLE 3M - Mittel Steingrau

13 x  4211622 BUSH FOR CROSS AXLE - Mittel Steingrau

48 x  4189110 CONN.BUSH W.FRIC./CROSSALE - Klares Blau

30 x  4514553 CONNECTOR PEG W. FRICTION 3M - Klares Blau

2 x  4566927 CROSSAXLE 3M WITH KNOB - Gelb Sandfarbe

4 x  4560177 CROSS AXLE 4M WITH END STOP - Dunkel Steingrau

1 x  4211805 CROSS AXLE 7M - Mittel Steingrau

9 x  4121667 DOUBLE CROSS BLOCK - Schwarz

3 x  4210857 CROSS BLOCK 3M - Dunkel Steingrau

1 x  4535768 CROSS AXLE 9M - Mittel Steingrau

1 x  370826 CROSS AXLE 12M - Schwarz

1 x  4141300 LT STEERING GEAR - Schwarz

9 x  4225033 BEAM 3 M. W/4 SNAPS - Mittel Steingrau

5 x  4296059 Angular beam 90degr. w.4 snaps - Mittel Steingrau

4 x  4539880 BEAM FRAME 5X7 Ø 4.85 - Mittel Steingrau

3 x  4540797 BEAM H. FRAME 5X11 Ø4.85 - Mittel Steingrau

2 x  4514559 GEAR WHEEL T=8, M=1 - Dunkel Steingrau

2 x  4211433 GEAR WHEEL 40T - Mittel Steingrau

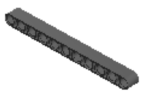
Step 1 of 121

2 x

1 x

Step 2 of 121

1 x



Step 3 of 121

2 x

1 x

1 x

Step 4 of 121

Step 5 of 121

1 x

2 x

1 x

Step 6 of 121

Step 7 of 121

1 x

1 x

1 x

Step 8 of 121

Step 9 of 121

1 x

1 x

1 x

Step 10 of 121

Step 11 of 121

1 x

1 x

Step 12 of 121

1 x

3 x

Step 13 of 121

2 x

1 x

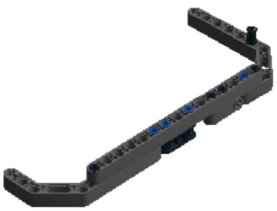
1 x

Step 14 of 121

1 x

2 x

1 x



Step 15 of 121

1 x

2 x

Step 16 of 121

1 x

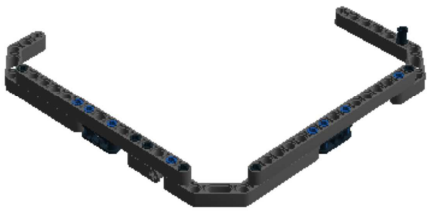
1 x

Step 17 of 121

1 x

2 x

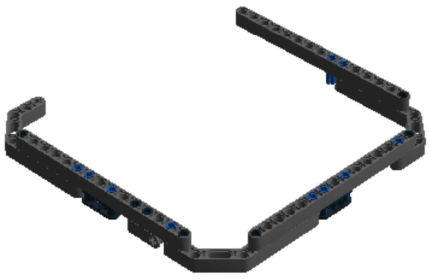
1 x



Step 18 of 121

1 x

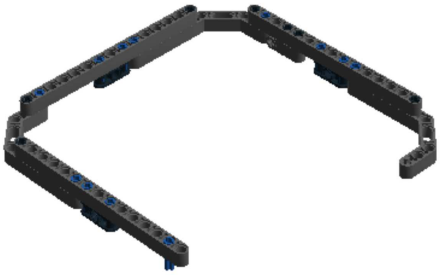
2 x



Step 19 of 121

2 x

1 x

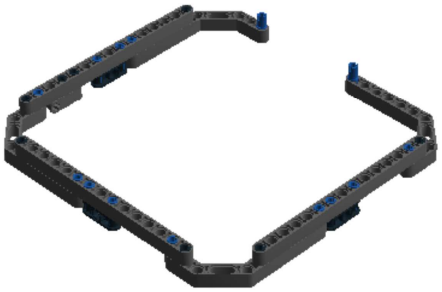


Step 20 of 121

1 x

1 x

2 x

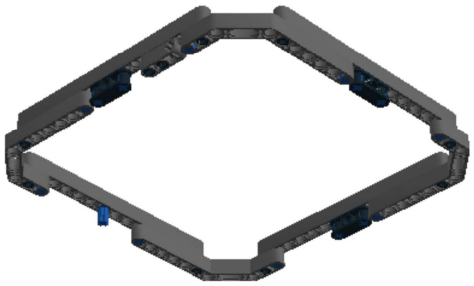


Step 21 of 121

1 x

2 x

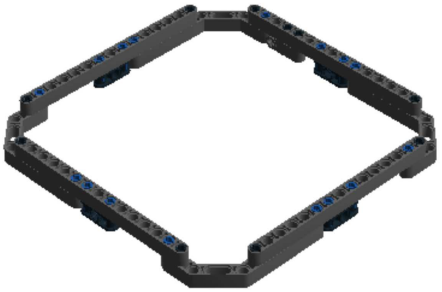
1 x



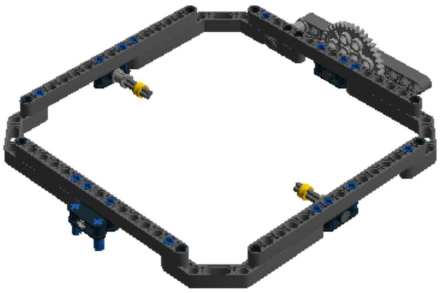
Step 22 of 121

1 x

1 x



Step 23 of 121

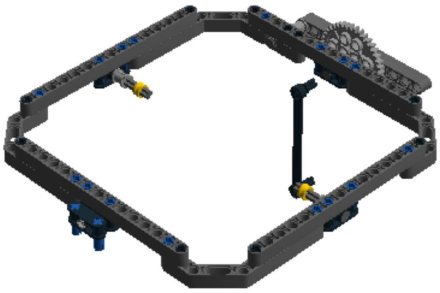


Step 24 of 121

2 x

1 x

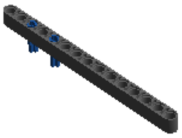
Step 25 of 121



Step 26 of 121

1 x

2 x



Step 27 of 121

1 x

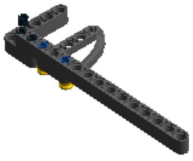
2 x

1 x

Step 28 of 121

1 x

2 x



Step 29 of 121

1 x

1 x

1 x

Step 30 of 121

3 x

1 x

Step 31 of 121

1 x

2 x

1 x

Step 32 of 121

1 x

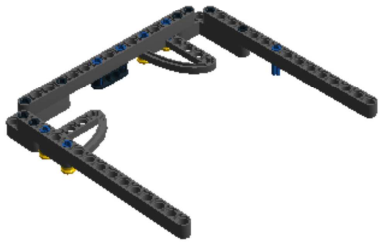
2 x

Step 33 of 121

1 x

1 x

1 x



Step 34 of 121

3 x

1 x

Step 35 of 121

1 x

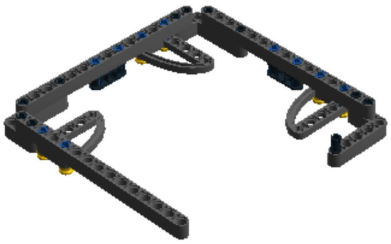
2 x

1 x

Step 36 of 121

1 x

2 x

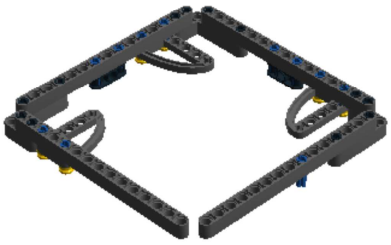


Step 37 of 121

1 x

1 x

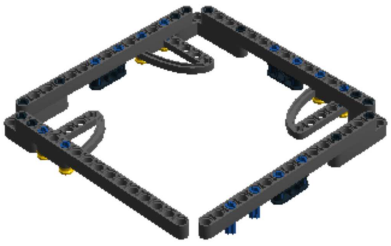
1 x



Step 38 of 121

3 x

1 x

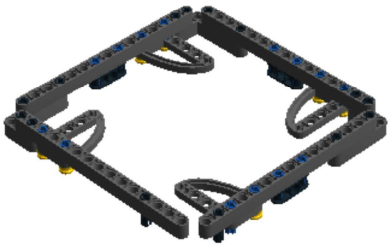


Step 39 of 121

1 x 

1 x 

2 x 

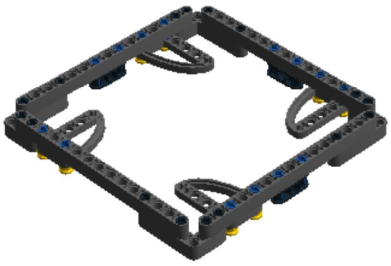


Step 40 of 121

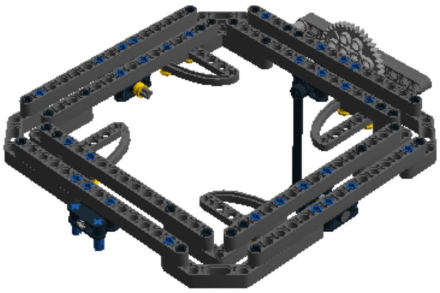
1 x 

1 x 

2 x 



Step 41 of 121



Step 42 of 121

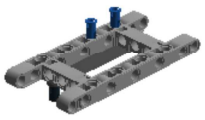
1 x

Step 43 of 121

1 x

2 x

1 x

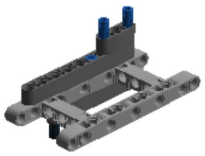


Step 44 of 121

1 x

1 x

2 x



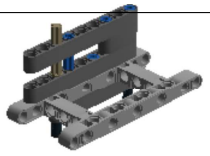
Step 45 of 121

1 x

1 x

1 x

1 x

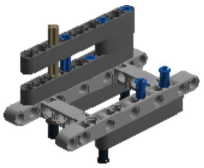


Step 46 of 121

1 x

1 x

2 x

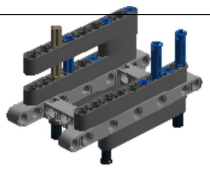


Step 47 of 121

1 x

1 x

2 x



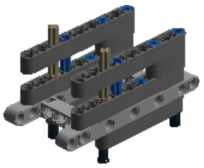
Step 48 of 121

1 x

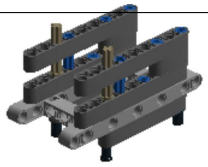
1 x

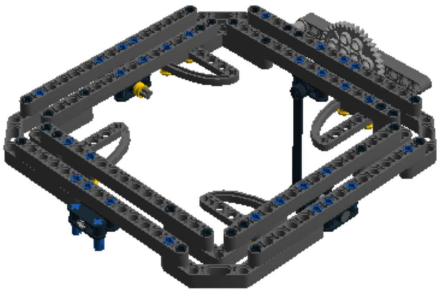
1 x

1 x



Step 49 of 121

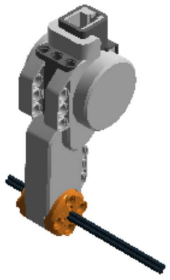




Step 50 of 121

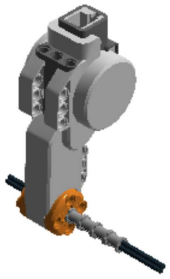
1 x

1 x



Step 51 of 121

4 x

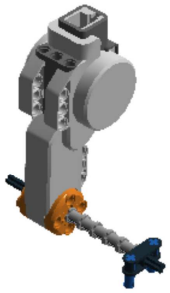


Step 52 of 121

1 x

2 x

1 x

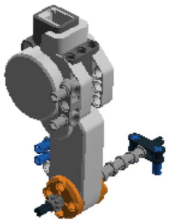


Step 53 of 121

2 x

1 x

1 x

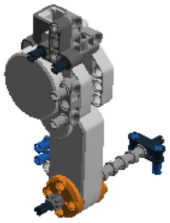


Step 54 of 121

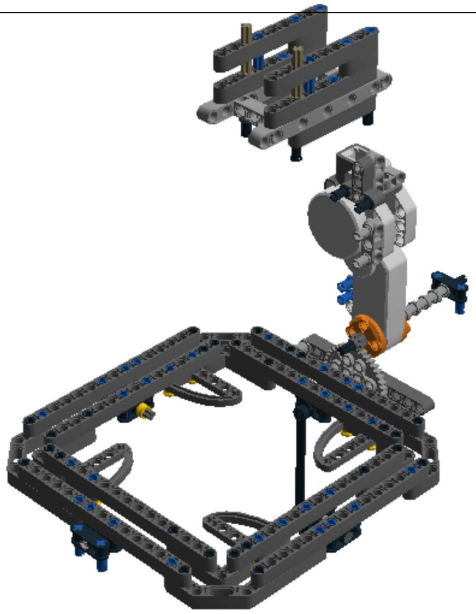
1 x

2 x

1 x



Step 55 of 121



Step 56 of 121

1 x

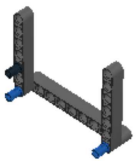
1 x

Step 57 of 121

1 x

1 x

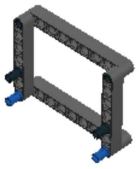
2 x



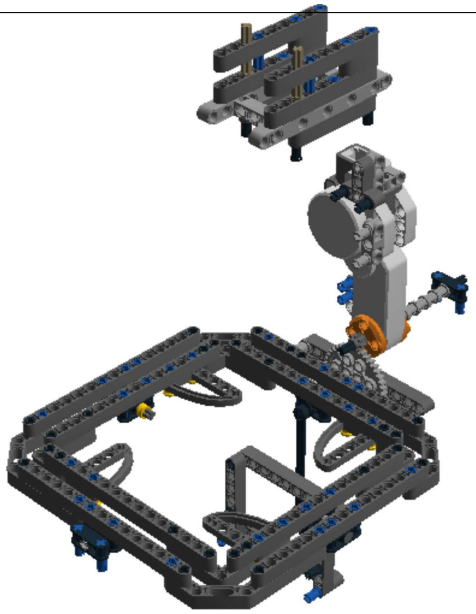
Step 58 of 121

1 x

3 x



Step 59 of 121



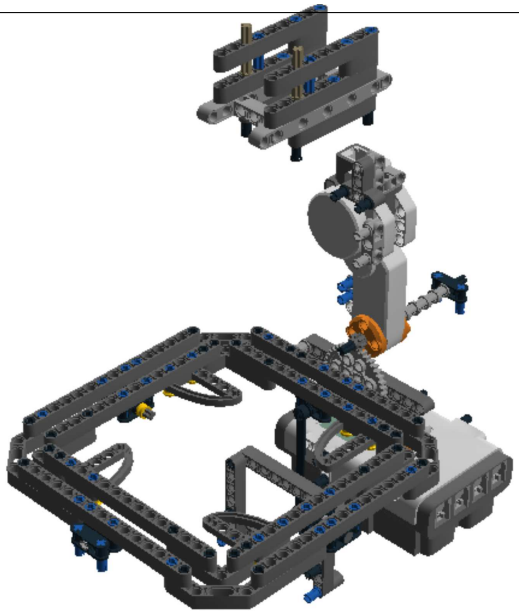
Step 60 of 121

1 x

Step 61 of 121

Step 62 of 121

Step 63 of 121



Step 64 of 121

1 x

1 x

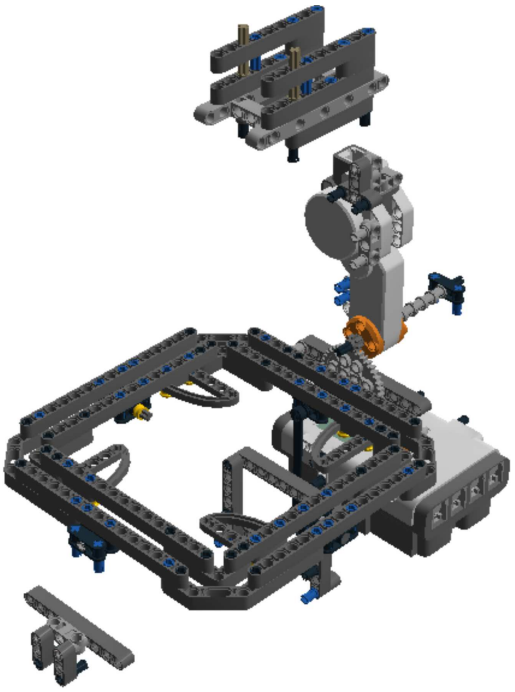
Step 65 of 121

1 x

1 x

2 x

Step 66 of 121

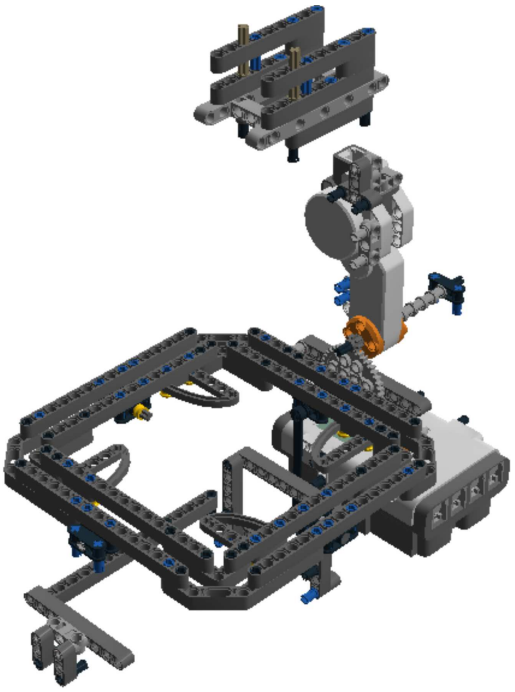


Step 67 of 121

1 x

2 x

Step 68 of 121

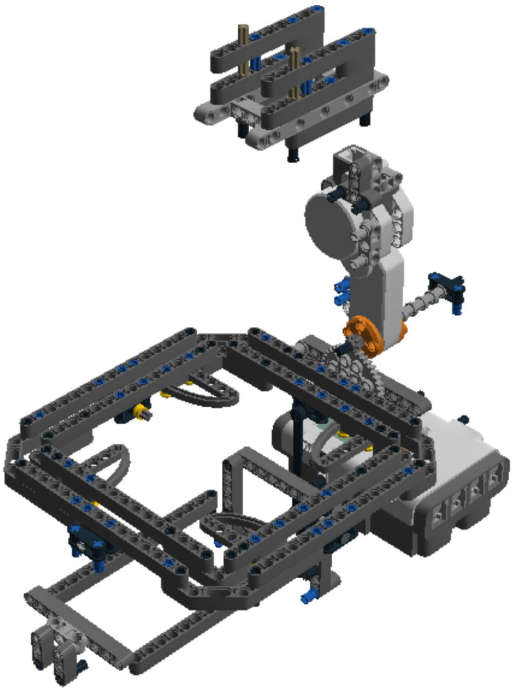


Step 69 of 121

1 x

2 x

Step 70 of 121

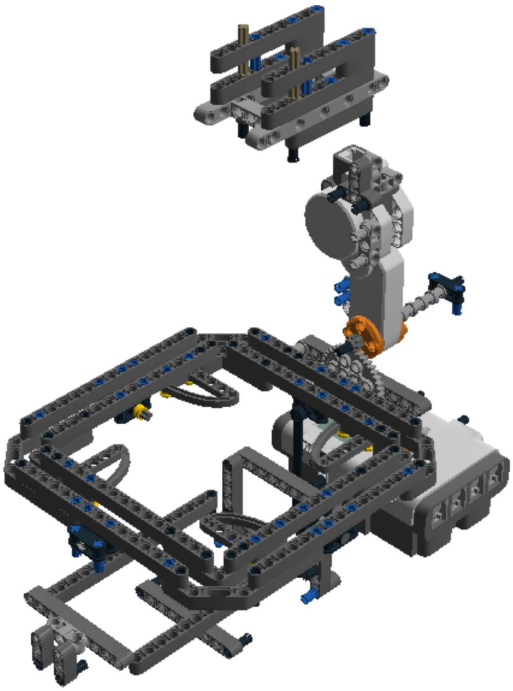


Step 71 of 121

1 x

2 x

Step 72 of 121



Step 73 of 121

1 x

1 x

Step 74 of 121

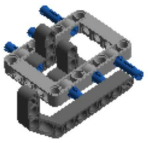
2 x

2 x

Step 75 of 121

1 x

3 x



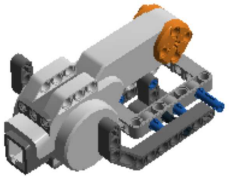
Step 76 of 121

1 x

1 x

1 x

1 x

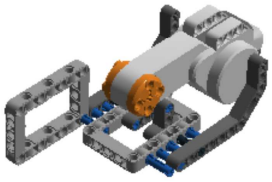


Step 77 of 121

1 x

2 x

1 x

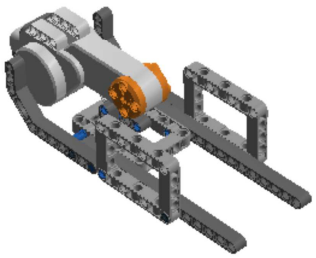


Step 78 of 121

2 x

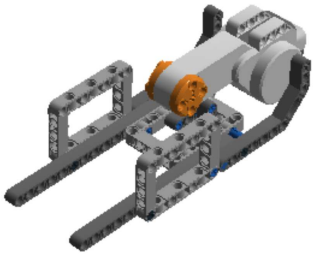
1 x

1 x



Step 79 of 121

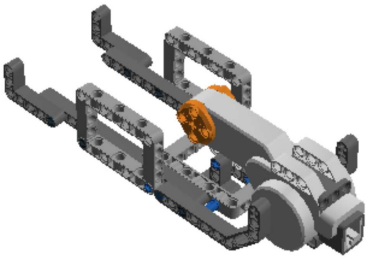
1 x



Step 80 of 121

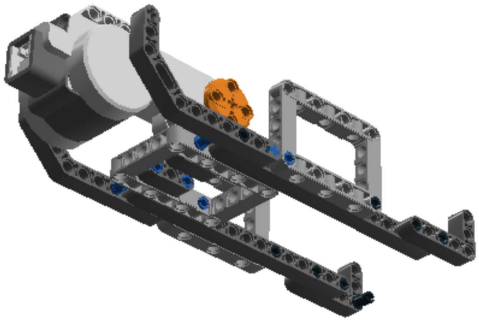
2 x

2 x



Step 81 of 121

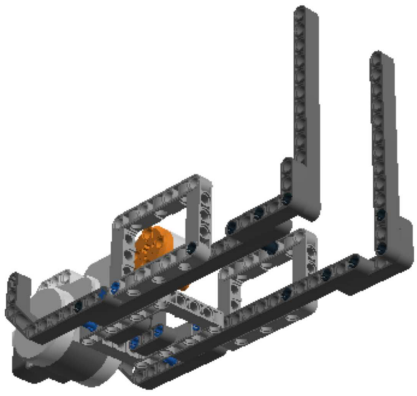
4 x



Step 82 of 121

2 x

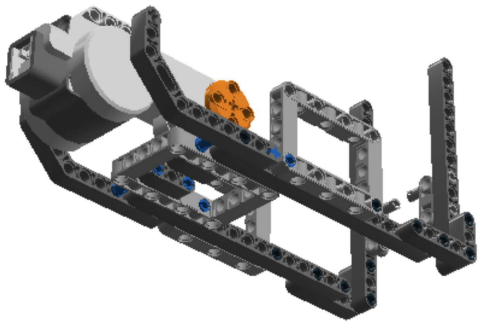
2 x



Step 83 of 121

1 x

2 x

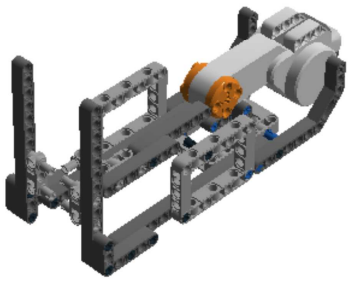


Step 84 of 121

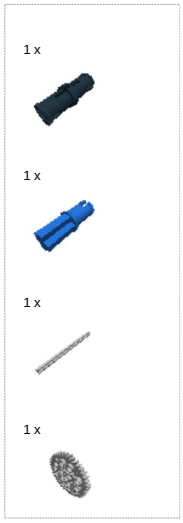
2 x

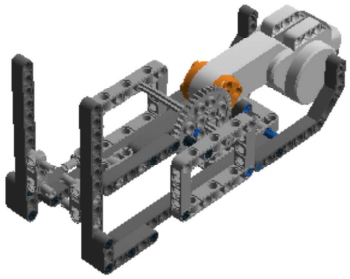
1 x

1 x

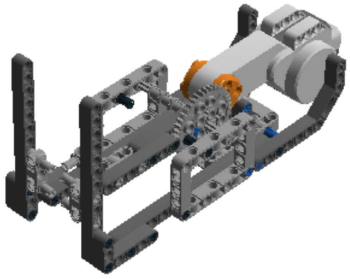


Step 85 of 121





Step 86 of 121

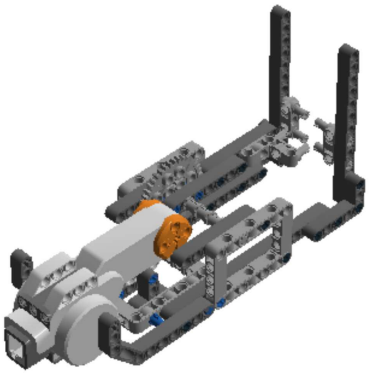


Step 87 of 121

2 x

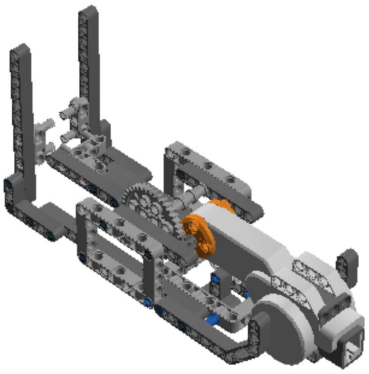
1 x

1 x

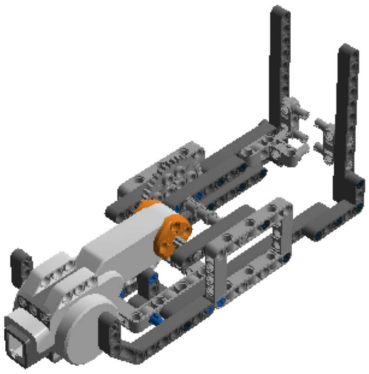


Step 88 of 121

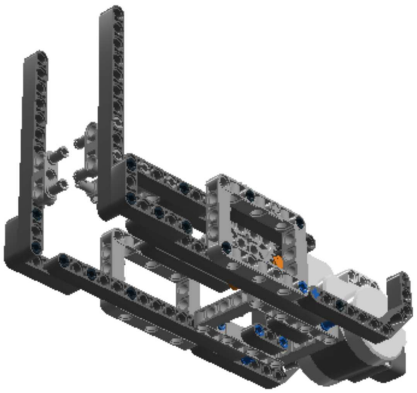
1 x



Step 89 of 121



Step 90 of 121

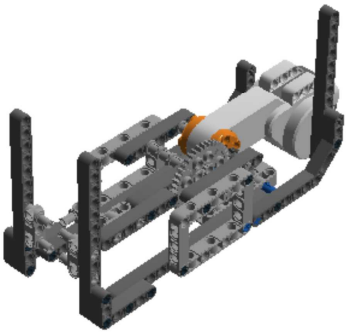


Step 91 of 121

1 x

1 x

1 x

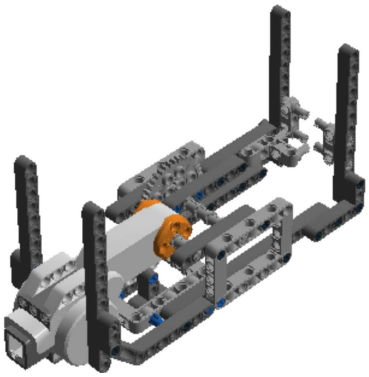


Step 92 of 121

1 x

1 x

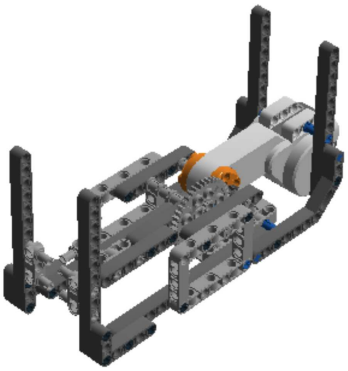
1 x



Step 93 of 121

1 x

1 x

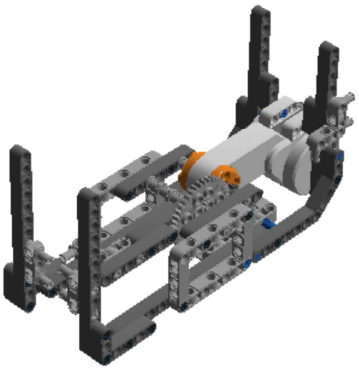


Step 94 of 121

2 x

1 x

1 x

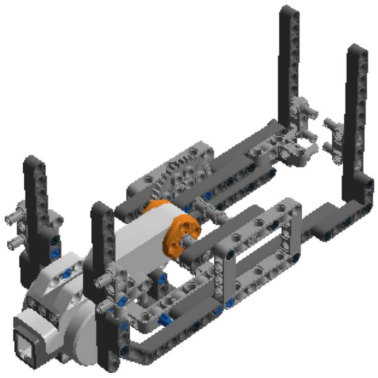


Step 95 of 121

1 x

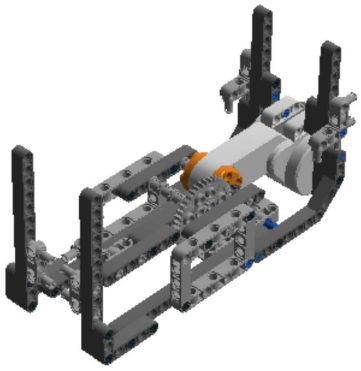
2 x

1 x



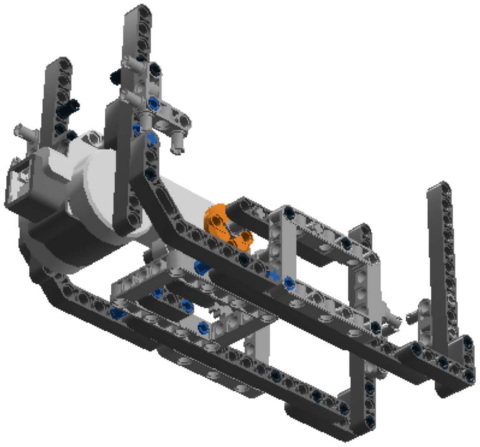
Step 96 of 121

1 x



Step 97 of 121

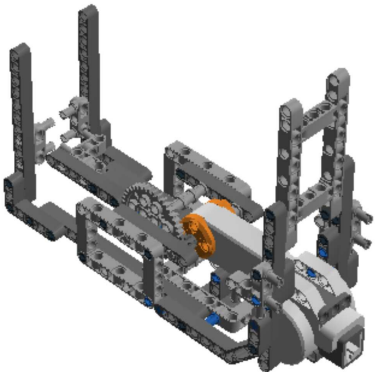
4 x



Step 98 of 121

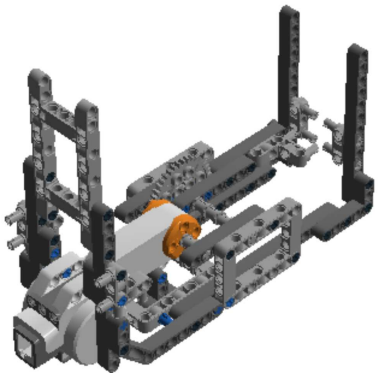
1 x

1 x



Step 99 of 121

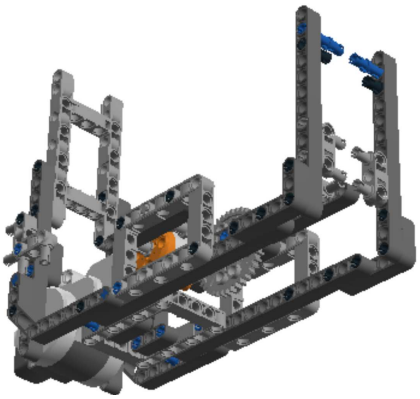
1 x



Step 100 of 121

2 x

2 x

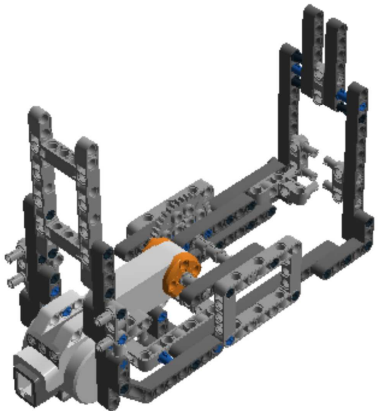


Step 101 of 121

2 x

1 x

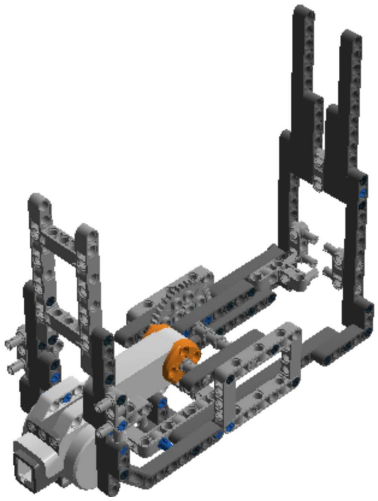
1 x



Step 102 of 121

2 x

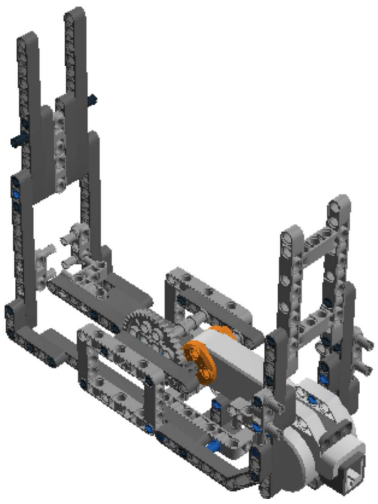
1 x



Step 103 of 121

3 x

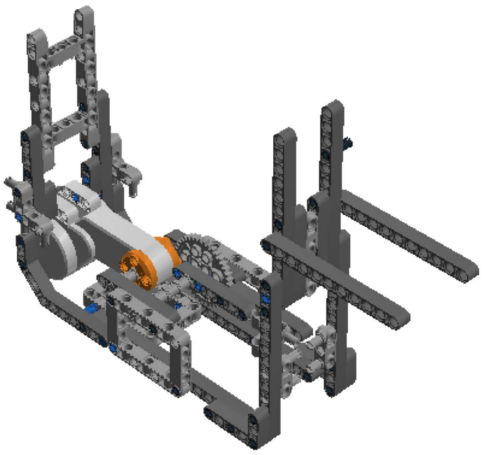
1 x



Step 104 of 121

2 x

2 x

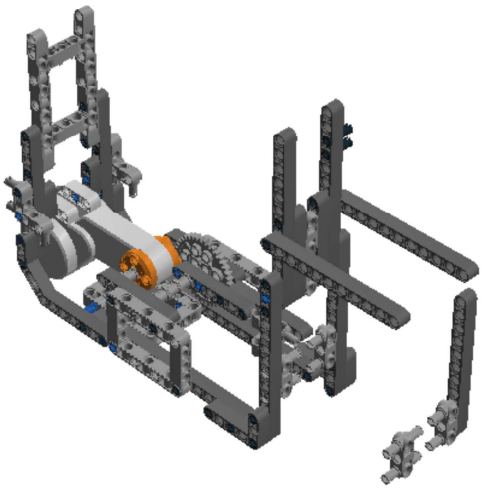


Step 105 of 121

1 x

1 x

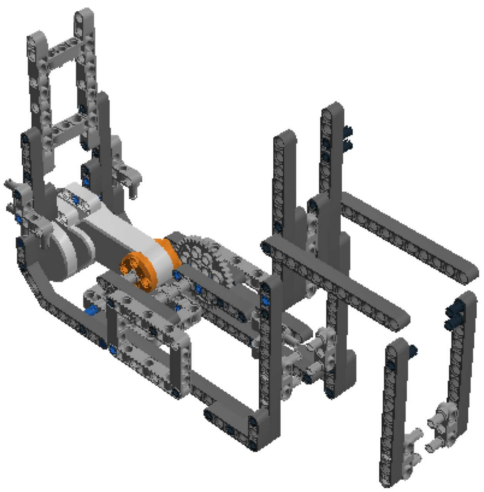
2 x



Step 106 of 121

1 x

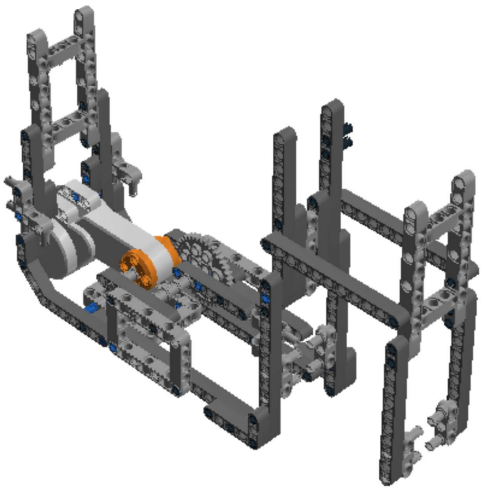
3 x



Step 107 of 121

3 x

1 x

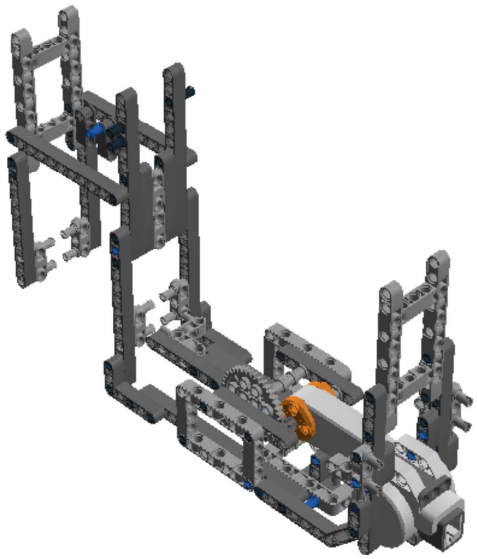


Step 108 of 121

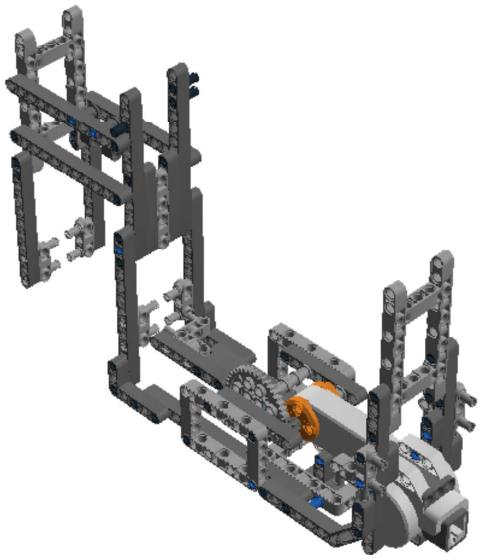
1 x

1 x

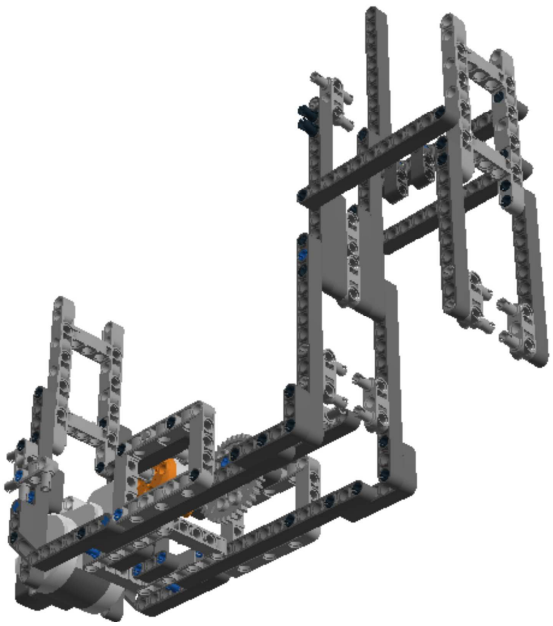
2 x



Step 109 of 121



Step 110 of 121

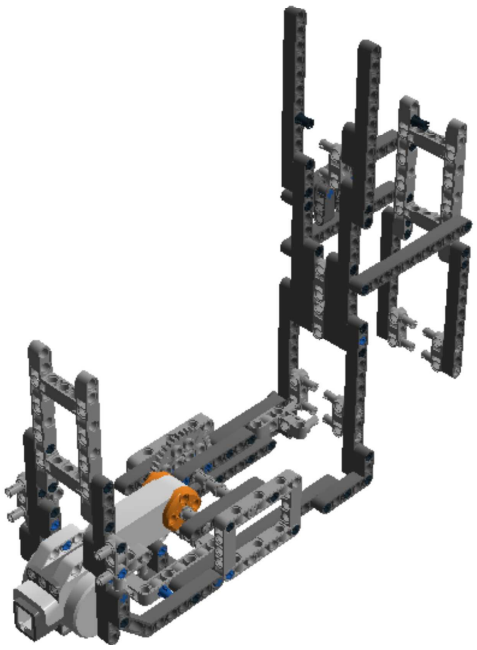


Step 111 of 121

1 x

1 x

2 x

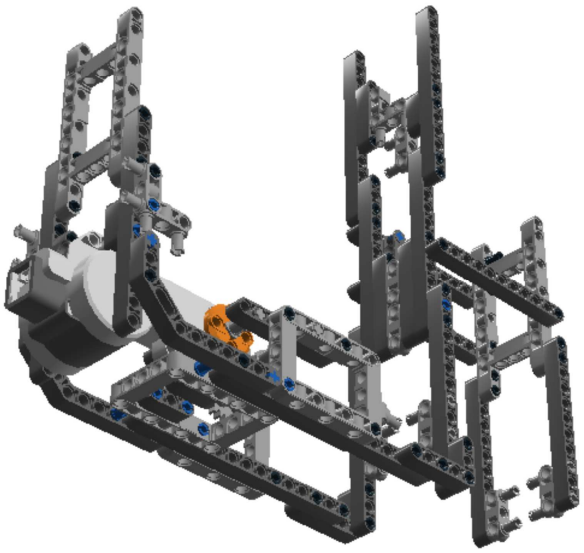


Step 112 of 121

1 x

2 x

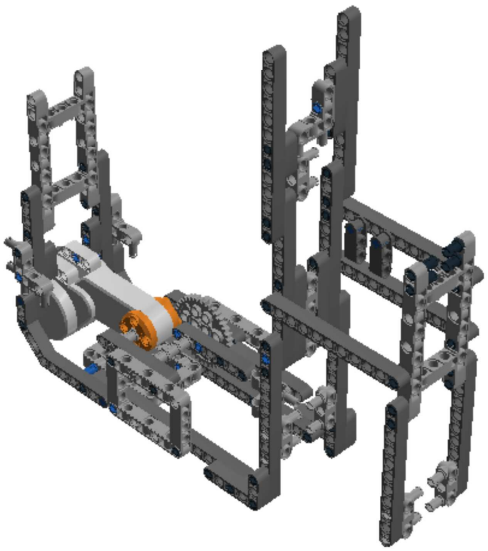
1 x



Step 113 of 121

3 x

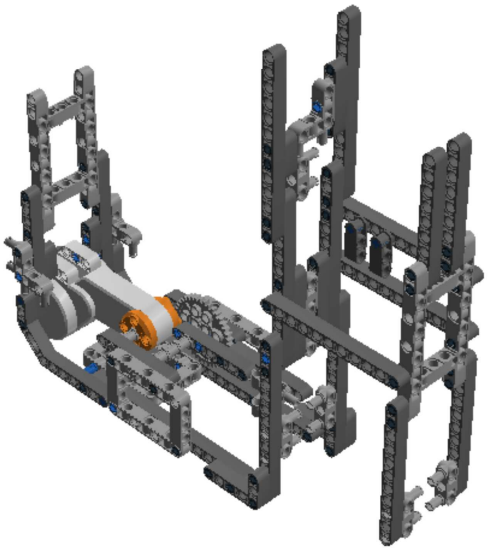
1 x



Step 114 of 121

2 x

1 x



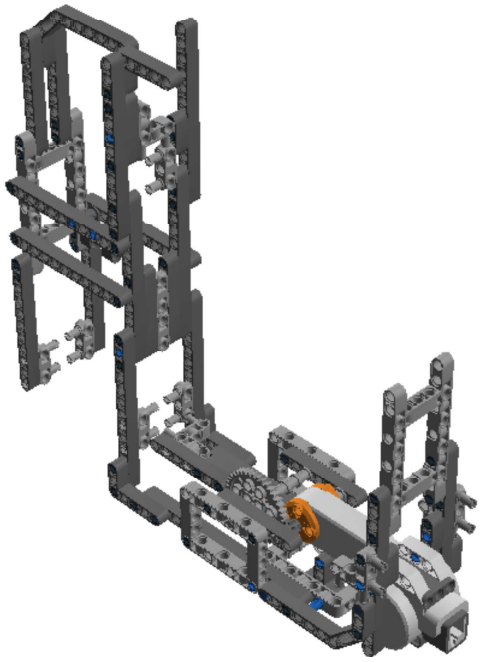
Step 115 of 121

1 x 

1 x 

1 x 

1 x 

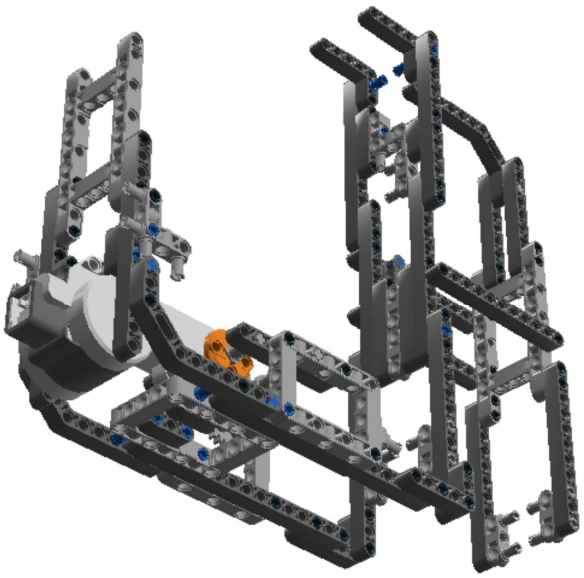


Step 116 of 121

1 x 

1 x 

2 x 

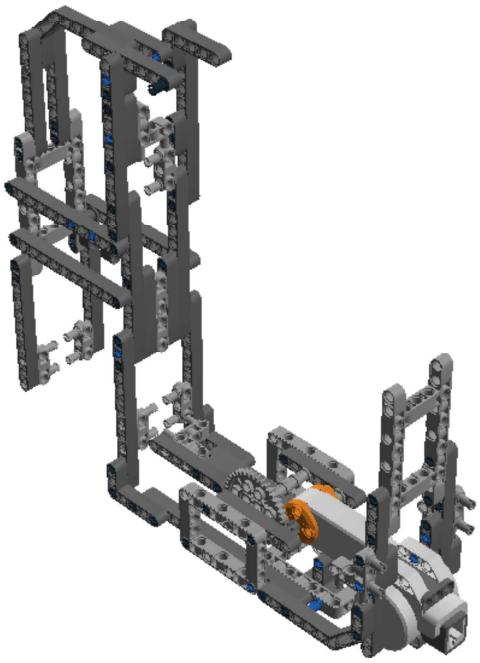


Step 117 of 121

2 x

1 x

1 x

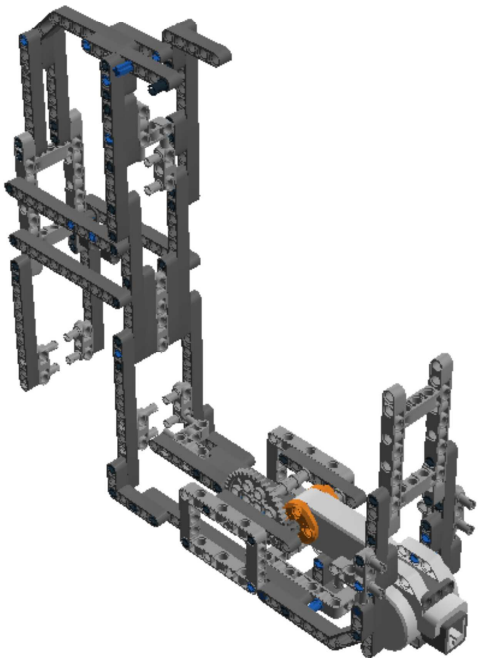


Step 118 of 121

1 x

1 x

2 x

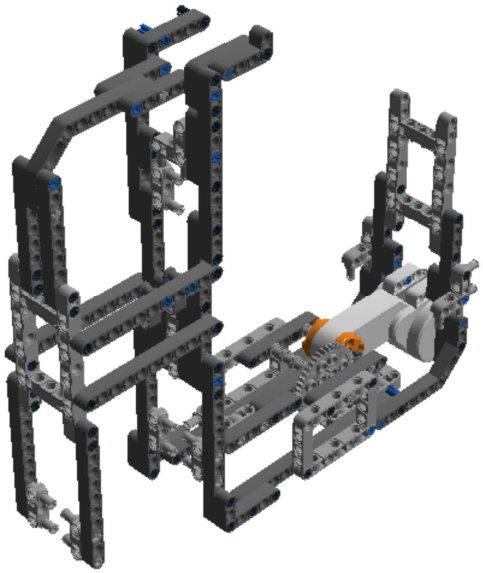


Step 119 of 121

1 x

2 x

1 x

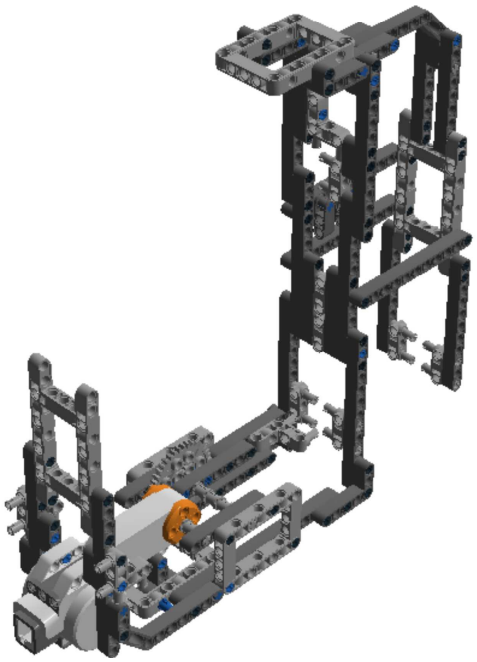


Step 120 of 121

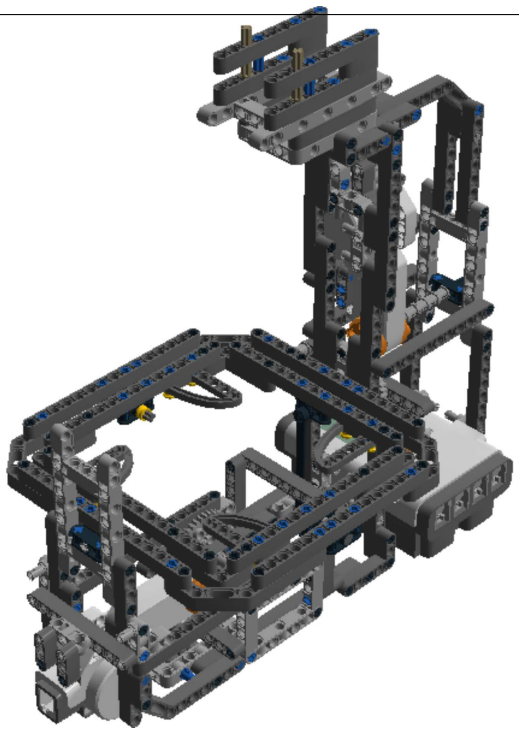
1 x

2 x

1 x



Step 121 of 121



Anhang B

Programmcode

B.1 Latenzmessung des Regelalgorithmus

```
1  /***********************************/
2  /* Latenzmessung                   */
3  /* des                             */
4  /* Regelalgorithmus                */
5  /***********************************/
6
7  /***********************************/
8  /* Definition benötigter Konstanten */
9  /* und                             */
10 /* Deklaration benötigter Variablen */
11 /***********************************/
12
13 // Variablen zur Zeitmessung
14 unsigned long time = 60000;
15 unsigned long n = 0;
16 long t_start;
17 long t_end;
18 long delta_t;
19 long sum = 0;
20 int t_max = 0;
21
22 // Aktuelle und alte x und y-Koordinate
23 float x = 0, y = 0;
24 float xOld = 0, yOld = 0;
25
26 // Variablen zur Simulation der Kameraerfassung
27 int left, right, top, bottom;
28
29 // Aktuelle und alte x- und y-Geschwindigkeit
30 float vx = 0, vy = 0;
```

```
31 float vxOld = 0, vyOld = 0;
32
33 // Zurückgelegte Distanz
34 float dsx = 0, dsy = 0;
35
36 // Strecke bis zum Zentrum
37 float dszx = 0, dszy = 0;
38
39 // Einstellwinkel
40 int alpha_x = 0, alpha_y = 0;
41
42 // Maximale Auslenkung der Platte
43 int alpha_max_pos = 25;     // entspricht 5°
44 int alpha_max_neg = -25;    // entspricht -5°
45
46 // Kontrollwerte
47 bool isBallFound = false;
48 bool isFirstRun = true;
49
50
51 /*******************************/
52 /* Angepasster Regelalgorithmus */
53 /*******************************/
54
55 task calculation_counter (){
56
57   while (true){
58
59   // Simulation der Kameraerfassung
60
61   left = right = top = bottom = 0;
62   while(right <= left){
63     left = Random(145) + 16;
64     right = Random(145) + 16;
65   }
66   while(bottom >= top){
67     top = Random(145);
68     bottom = Random(145);
69   }
70
71     // Zeitnahme (Start)
72     t_start = CurrentTick();
73
74     x = ((left+right)/2 - 16)/1200;
75     y = (top+bottom)/2/1200;
76
77     isBallFound = true;
78
79     if(!isBallFound){
80       isFirstRun = true;
```

```
81          xOld = yOld = 0;
82          vxOld = vyOld = 0;
83        }
84        else{
85          if(isFirstRun){
86            isFirstRun = false;
87            xOld = x;
88            yOld = y;
89          }
90          else{
91            dszx = 0.06 - x;
92            dsx = x - xOld;
93            vx = dsx / 0.11;
94
95            dszy = 0.06 - y;
96            dsy = y - yOld;
97            vy = dsy / 0.11;
98
99            if((abs(vx) < 0.01)){
100             alpha_x = -(dszx / 0.06) * alpha_v0;
101           }
102           else{
103             alpha_x = 5 *(180 / 3.14) * asin(vx / ((5.0 /
                  7.0) * 9.81 * 0.443));
104             if(alpha_x > alpha_max_pos){
105               alpha_x = alpha_max_pos;
106             }
107             if(alpha_x < alpha_max_neg){
108               alpha_x = alpha_max_neg;
109             }
110           }
111
112           if((abs(vy) < 0.01)){
113             alpha_y = (dszy / 0.06) * alpha_v0;
114           }
115           else{
116             alpha_y_temp = - 5 *(180 / 3.14) * asin(vy /
                  ((5.0 / 7.0) * 9.81 * 0.443));
117             if(alpha_y > alpha_max_pos){
118               alpha_y = alpha_max_pos;
119             }
120             if(alpha_y < alpha_max_neg){
121               alpha_y = alpha_max_neg;
122             }
123           }
124
125           xOld = x;
126           vxOld = vx;
127
128           yOld = y;
```

```
129           vyOld = vy;
130         }
131       }
132
133       // Zeitnahme (Ende)
134       t_end = CurrentTick();
135
136       // Zeitberechnung
137
138       delta_t = t_end - t_start;
139       sum += delta_t;
140       if(delta_t > t_max){
141         t_max = delta_t;
142       }
143       n++;
144     }
145
146 }
147
148
149 /***************/
150 /* Kontrolltask */
151 /***************/
152
153 task timer(){
154
155   Wait(time);
156   StopTask(calculation_counter);
157
158   PlayTone(880, 100); Wait(110);
159   PlayTone(880, 100); Wait(890);
160
161   // Ausgabe der Daten
162
163   string msg;
164   ClearScreen();
165
166   float seconds = time / 1000;
167   float t_avg = sum / n;
168
169   msg = "t(s):     ";
170   msg += NumToStr(seconds);
171   TextOut(5, LCD_LINE1, msg, false);
172
173   msg = "n:        ";
174   msg += NumToStr(n);
175   TextOut(5, LCD_LINE2, msg, false);
176
177   msg = "Sum:      ";
178   msg += NumToStr(sum);
```

```
179   TextOut(5, LCD_LINE3, msg, false);
180
181   msg = "t_avg:    ";
182   msg += NumToStr(t_avg);
183   TextOut(5, LCD_LINE4, msg, false);
184
185   msg = "t_max:    ";
186   msg += NumToStr(t_max);
187   TextOut(5, LCD_LINE5, msg, false);
188
189   Wait(30000);
190
191 }
192
193
194 /*************/
195 /* Main Task */
196 /*************/
197
198 task main(){
199
200   //Starte Regelalgorithmus und Kontrolltask
201   PlayTone(440, 100); Wait(1000);
202   PlayTone(440, 100); Wait(1000);
203   PlayTone(440, 100); Wait(1000);
204   PlayTone(880, 500);
205   Precedes(timer, calculation_counter);
206
207 }
```

B.2 Latenzmessung der Kameraaufnahme

```
1  /**********************************/
2  /* Latenzmessung                  */
3  /* der                            */
4  /* Kameraaufnahme                 */
5  /**********************************/
6
7  /**********************************/
8  /* Benötigte Importe              */
9  /**********************************/
10
11 #define CAMADDR 0x02
12 #include "nxtcamlib-default.nxc"
13
14
15 /**********************************/
16 /* Definition benötigter Konstanten */
17 /* und                            */
18 /* Deklaration benötigter Variablen */
19 /**********************************/
20
21 // Konstanten um Eingang anzusprechen
22 const byte camPort  =  IN_1;
23
24 // Variablen zur Zeitmessung
25 unsigned long time = 60000;
26 int n = 0;
27 long t_start;
28 long t_end;
29 long delta_t;
30 long sum = 0;
31 long delta_t_max = 0;
32 int n_max;
33
34 // Arrays und Integer zum erfassen der Kameradaten
35 int bc[10];      // Farbwert des Objekts
36 int bl[10];      // Bounding Box Links
37 int bt[10];      // Bounding Box Oben
38 int br[10];      // Bounding Box Links
39 int bb[10];      // Bounding Box unten
40 int nblobs;      // Anzahl der gefundenen Objekte
41
42
43 /*****************/
44 /* Kameraaufnahme */
45 /*****************/
46
47 task camera_counter (){
```

```
48
49    while (true){
50      t_start = CurrentTick();
51      NXTCam_GetBlobs(camPort, nblobs, bc, bl, bt, br, bb);
52      t_end = CurrentTick();
53      delta_t = t_end - t_start;
54      sum += delta_t;
55      if(delta_t > delta_t_max){
56        delta_t_max = delta_t;
57        n_max = n;
58      }
59      n++;
60    }
61
62  }
63
64
65  /****************/
66  /* Kontrolltask */
67  /****************/
68
69  task timer(){
70
71    Wait(time);
72    StopTask(camera_counter);
73
74    PlayTone(880, 100); Wait(110);
75    PlayTone(880, 100); Wait(890);
76
77    // Ausgabe der Daten
78
79    string msg;
80    ClearScreen();
81
82    float seconds = time / 1000;
83    float average = sum / n;
84
85    msg = "t(s):   ";
86    msg += NumToStr(seconds);
87    TextOut(5, LCD_LINE1, msg, false);
88
89    msg = "n:      ";
90    msg += NumToStr(n);
91    TextOut(5, LCD_LINE2, msg, false);
92
93    msg = "Sum:    ";
94    msg += NumToStr(sum);
95    TextOut(5, LCD_LINE3, msg, false);
96
97    msg = "Avg:    ";
```

```
98    msg += NumToStr(average);
99    TextOut(5, LCD_LINE4, msg, false);
100
101   msg = "Max:    ";
102   msg += NumToStr(delta_t_max);
103   TextOut(5, LCD_LINE5, msg, false);
104
105   msg = "n_Max: ";
106   msg += NumToStr(n_max);
107   TextOut(5, LCD_LINE6, msg, false);
108
109   Wait(10000);
110
111 }
112
113
114 /*************/
115 /* Main Task */
116 /*************/
117
118 task main(){
119
120   // Initialisierung der Kamera
121   int init=0;
122   init = NXTCam_Init(camPort, CAMADDR);
123
124   // Abbruch bei Initialisierungsfehler
125   if (init != 1) {
126     PlayTone(330, 100); Wait(200);
127     PlayTone(330, 100); Wait(200);
128     PlayTone(330, 100); Wait(200);
129     StopAllTasks();
130   }
131
132   // Starte Ksameraaufnahme und Kontrolltask
133   PlayTone(440, 100); Wait(1000);
134   PlayTone(440, 100); Wait(1000);
135   PlayTone(440, 100); Wait(1000);
136   PlayTone(880, 500);
137   Precedes(timer, camera_counter);
138
139 }
```

B.3 Latenzmessung der Motorsteuerung

```
1  /**********************************/
2  /* Latenzmessung                  */
3  /* der                            */
4  /* Motorsteuerung                 */
5  /**********************************/
6
7  /**********************************/
8  /* Definition benötigter Konstanten */
9  /* und                            */
10 /* Deklaration benötigter Variablen */
11 /**********************************/
12
13 // Konstanten um Ausgang anzusprechen (x- oder y-Achse)
14 const byte currentAxis = OUT_A;
15 //const byte currentAxis = OUT_B;
16
17 // Variablen zur Zeitmessung
18 unsigned long time = 60000;
19 long n=0;
20 long t_start;
21 long t_end;
22 long delta_t;
23 long sum = 0;
24 long t_max = 0;
25
26 // aktueller Einstellwinkel und Testwinkel
27 int alpha= 0;
28 int alpha_test = 25;
29
30
31 /*****************/
32 /* Motorsteuerung */
33 /*****************/
34
35 task rotation_counter (){
36
37   while (true){
38
39     // positive Rotation
40
41     alpha += alpha_test;
42     t_start = CurrentTick()
43     PosRegSetAngle(currentAxis, alpha);
44     while(MotorRotationCount(currentAxis) != alpha){
45
46     }
47     t_end = CurrentTick();
```

```
48      delta_t = t_end - t_start;
49      sum += delta_t;
50      if(delta_t > t_max){
51        t_max = delta_t;
52      }
53      n++;
54
55      Wait(10);
56
57      //negative Rotation
58
59      t_start = CurrentTick();
60      alpha -= alpha_test;
61      PosRegSetAngle(currentAxis, alpha);
62      while(MotorRotationCount(currentAxis) != alpha){
63
64      }
65      t_end = CurrentTick();
66      delta_t = t_end - t_start;
67      sum += delta_t;
68      if(delta_t > t_max){
69        t_max = delta_t;
70      }
71      n++;
72
73      Wait(10);
74
75      // negative Rotation
76
77      t_start = CurrentTick();
78      alpha -= alpha_test;
79      PosRegSetAngle(currentAxis, alpha);
80      while(MotorRotationCount(currentAxis) != alpha){
81
82      }
83      t_end = CurrentTick();
84      delta_t = t_end - t_start;
85      sum += delta_t;
86      if(delta_t > t_max){
87        t_max = delta_t;
88      }
89      n++;
90
91      Wait(10);
92
93      // positive Rotation
94
95      alpha += alpha_test;
96      t_start = CurrentTick()
97      PosRegSetAngle(currentAxis, alpha);
```

```
98      while(MotorRotationCount(currentAxis) != alpha){
99
100     }
101     t_end = CurrentTick();
102     delta_t = t_end - t_start;
103     sum += delta_t;
104     if(delta_t > t_max){
105       t_max = delta_t;
106     }
107     n++;
108
109     Wait(10);
110
111   }
112
113 }
114
115
116 /***************/
117 /* Kontrolltask */
118 /***************/
119
120 task timer(){
121
122   Wait(time);
123   StopTask(rotation_counter);
124
125   PlayTone(880, 100); Wait(110);
126   PlayTone(880, 100); Wait(890);
127
128   PosRegSetAngle(currentAxis, 0);
129
130   // Ausgabe der Daten
131
132   string msg;
133   ClearScreen();
134
135   float seconds = time / 1000;
136   float t_avg = sum / n;
137
138   msg = "t(s): ";
139   msg += NumToStr(seconds);
140   TextOut(5, LCD_LINE1, msg, false);
141
142   msg = "n:     ";
143   msg += NumToStr(n);
144   TextOut(5, LCD_LINE2, msg, false);
145
146   msg = "Sum   :";
147   msg += NumToStr(sum);
```

```
148   TextOut(5, LCD_LINE3, msg, false);
149
150   msg = "t_avg: ";
151   msg += NumToStr(t_avg);
152   TextOut(5, LCD_LINE4, msg, false);
153
154   msg = "t_max: ";
155   msg += NumToStr(t_max);
156   TextOut(5, LCD_LINE5, msg, false);
157
158
159   Wait(30000);
160
161 }
162
163
164 /*************/
165 /* Main Task */
166 /*************/
167
168 task main(){
169
170   // Initialisierung der Motoren
171   PosRegEnable(currentAxis, PID_5, PID_4, PID_5);
172   SetMotorRegulationTime(10);
173
174   //Starte Motorsteuerung und Kontrolltask
175   PlayTone(440, 100); Wait(1000);
176   PlayTone(440, 100); Wait(1000);
177   PlayTone(440, 100); Wait(1000);
178   PlayTone(880, 500);
179   Precedes(timer, rotation_counter);
180
181 }
```

B.4 Automatische Steuerung der Wippe

```
1  /*********************************/
2  /* Steuerung                     */
3  /* der                           */
4  /* LEGO(R) Mindstorms(R) NXT Wippe  */
5  /*********************************/
6
7  /*********************************/
8  /* Benötigte Importe             */
9  /*********************************/
10
11 // Importiere Kamera-Bibliothek
12 #define CAMADDR     0x02
13 #include "nxtcamlib-default.nxc"
14
15
16 /*********************************/
17 /* Definition benötigter Konstanten */
18 /*********************************/
19
20 // Konstanten um Ein- und Ausgänge anzusprechen
21 const byte camPort = IN_1;
22 const byte xAxis = OUT_A;
23 const byte yAxis = OUT_B;
24
25 // Mutexvariable
26 mutex m;
27
28 // Maximale Geschwindigkeit und Beschleunigung der
       Motoren (0...100)
29 int max_speed = 100;
30 int max_acceleration = 100;
31
32 // PID-Werte (PID_0...PID_7)
33 const byte PID_P = PID_5;
34 const byte PID_I = PID_4;
35 const byte PID_D = PID_5;
36
37 // Maximale Auslenkung der Platte
38 int alpha_max_pos = 25;     // entspricht 5°
39 int alpha_max_neg = -25;    // entspricht -5°
40
41 // Korrektur bei Schieflage der Platte
42 int x_offset = -6;
43 int y_offset = -8;
44
45 // Maximaler Einstellwinkel bei Unterschreitung der
       Grenzgeschwindigkeit
```

```
46 int alpha_v0 = 25;
47
48 // Arrays und Integer zum erfassen der Kameradaten
49 int color[1];      // Farbwert des Objekts
50 int left[1];       // Bounding Box Links
51 int top[1];        // Bounding Box Oben
52 int right[1];      // Bounding Box Links
53 int bottom[1];     // Bounding Box Unten
54 int nblobs;        // Anzahl der gefundenen Objekte
55
56
57 /*************************************/
58 /* Initialisierung benötigter Variablen */
59 /*************************************/
60
61 // Aktuelle und alte x und y-Koordinate
62 float x = 0, y = 0;
63 float xOld = 0, yOld = 0;
64
65 // Aktuelle und alte x- und y-Geschwindigkeit
66 float vx = 0, vy = 0;
67 float vxOld = 0, vyOld = 0;
68
69 // Zurückgelegte Distanz
70 float dsx = 0, dsy = 0;
71
72 // Strecke bis zum Zentrum
73 float dszx = 0, dszy = 0;
74
75 // Einstellwinkel
76 float alpha_x = 0, alpha_y = 0;
77 float alpha_x_temp = 0, alpha_y_temp = 0;
78
79 // Kontrollwerte
80 bool isBallFound = false;
81 bool isFirstRun = true;
82
83
84 /*****************************/
85 /* Initialisierung der Kamera */
86 /*****************************/
87
88 task initCam(){
89
90      // Initialisierung der Kamera
91      int init=0;
92      init = NXTCam_Init(camPort, CAMADDR);
93
94      // Abbruch bei Initialisierungsfehler
95      if (init != 1) {
```

```
96          PlayTone(330, 100); Wait(200);
97          PlayTone(330, 100); Wait(200);
98          PlayTone(330, 100); Wait(200);
99          StopAllTasks();
100       }
101
102       Release(m);
103 }
104
105
106 /*****************************/
107 /* Initialisierung der Motoren */
108 /*****************************/
109
110 task initPosReg(){
111   bool isMotorOverload;
112   int power;
113   int newRotationCount, oldRotationCount;
114   int xMax, xMin, yMax, yMin, position;
115
116   //x-Achse
117
118   isMotorOverload = false;
119   newRotationCount = 0, oldRotationCount = 0;
120   while(!isMotorOverload){
121     OnFwd(xAxis, 15);
122     Wait(100);
123     newRotationCount = MotorRotationCount(xAxis);
124     if(oldRotationCount == newRotationCount){
125       isMotorOverload = true;
126     }
127     oldRotationCount = newRotationCount;
128   }
129   xMax = newRotationCount;
130
131   isMotorOverload = false;
132   newRotationCount = 0, oldRotationCount = 0;
133   while(!isMotorOverload){
134     OnRev(xAxis, 15);
135     Wait(100);
136     newRotationCount = MotorRotationCount(xAxis);
137     if(oldRotationCount == newRotationCount){
138       isMotorOverload = true;
139     }
140     oldRotationCount = newRotationCount;
141   }
142   xMin = newRotationCount;
143   position = (xMax-xMin)/2;
144
```

```
145     RotateMotorPID(xAxis, 15, position + x_offset, 30, 50,
            90);
146     Wait(100);
147     PosRegEnable(xAxis, PID_P, PID_I, PID_D);
148     PosRegSetMax(xAxis, max_speed, max_acceleration);
149
150     //y-Achse
151
152     isMotorOverload = false;
153     newRotationCount = 0, oldRotationCount = 0;
154     while(!isMotorOverload){
155       OnFwd(yAxis, 15);
156       Wait(100);
157       newRotationCount = MotorRotationCount(yAxis);
158       if(oldRotationCount == newRotationCount){
159         isMotorOverload = true;
160       }
161       oldRotationCount = newRotationCount;
162     }
163     yMax = newRotationCount;
164
165     isMotorOverload = false;
166     newRotationCount = 0, oldRotationCount = 0;
167     while(!isMotorOverload){
168       OnRev(yAxis, 15);
169       Wait(100);
170       newRotationCount = MotorRotationCount(yAxis);
171       if(oldRotationCount == newRotationCount){
172         isMotorOverload = true;
173       }
174       oldRotationCount = newRotationCount;
175     }
176     yMin = newRotationCount;
177     position = (yMax-yMin)/2;
178
179     RotateMotorPID(yAxis, 15, position + y_offset, 30, 50,
            90);
180     Wait(100);
181     PosRegEnable(yAxis, PID_P, PID_I, PID_D);
182     PosRegSetMax(yAxis, max_speed, max_acceleration);
183
184     SetMotorRegulationTime(10);
185
186     Release(m);
187
188 }
189
190
191 /*******************/
192 /* Regelalgorithmus */
```

```
193 /********************/
194
195 task getPosition(){
196
197   while(true){
198     NXTCam_GetBlobs(camPort, nblobs, color, left, top,
          right, bottom);
199     x = ((((right[0] + left[0]) / 2) - 16) / 1200);
200     y = ((bottom[0] + top[0]) / 2) / 1200;
201
202     if(nblobs == 0){
203       isFirstRun = true;
204       xOld = yOld = 0;
205       vxOld = vyOld = 0;
206       alpha_x = alpha_y = 0;
207       PlayTone(110, 10);
208     }
209     else{
210       if(isFirstRun){
211         isFirstRun = false;
212         xOld = x;
213         yOld = y;
214       }
215       else{
216         dszx = 0.06 - x;
217         dsx = x - xOld;
218         vx = dsx / 0.032;
219
220         dszy = 0.06 - y;
221         dsy = y - yOld;
222         vy = dsy / 0.032;
223
224         if((abs(vx) < 0.01)){
225           alpha_x = -(dszx / 0.06) * alpha_v0;
226         }
227         else{
228           alpha_x = 5 *(180 / 3.14) * asin(vx / ((5.0 /
                7.0) * 9.81 * 0.404));
229           if(alpha_x > alpha_max_pos){
230             alpha_x = alpha_max_pos;
231           }
232           if(alpha_x < alpha_max_neg){
233             alpha_x = alpha_max_neg;
234           }
235         }
236
237         if((abs(vy) < 0.01)){
238           alpha_y = (dszy / 0.06) * alpha_v0;
239         }
240         else{
```

```
241             alpha_y = - 5 *(180 / 3.14) * asin(vy / ((5.0 /
                    7.0) * 9.81 * 0.404));
242             if(alpha_y > alpha_max_pos){
243               alpha_y = alpha_max_pos;
244             }
245             if(alpha_y < alpha_max_neg){
246               alpha_y = alpha_max_neg;
247             }
248           }
249
250           xOld = x;
251           vxOld = vx;
252
253           yOld = y;
254           vyOld = vy;
255
256           if((abs(dszx) >= 0.005) || (vx >= 0.005)){
257             PosRegSetAngle(xAxis, alpha_x);
258           }
259           else{
260             PosRegSetAngle(xAxis, 0);
261           }
262           if((abs(dszy) >= 0.005) || (vy >= 0.005)){
263             PosRegSetAngle(yAxis, alpha_y);
264           }
265           else{
266             PosRegSetAngle(yAxis, 0);
267           }
268
269         }
270       }
271
272   } // while
273
274 } // task
275
276
277 /*************/
278 /* Main Task */
279 /*************/
280
281 task main (){
282
283   //Initialisiere die Motoren
284   Acquire(m);
285   StartTask(initPosReg);
286
287   // Initialisiere Kamera
288   Acquire(m);
289   StartTask(initCam);
```

```
290
291     // Starte Regelalgorithmus
292     Acquire(m);
293     StartTask(getPosition);
294
295 }
```

Literaturverzeichnis

[1] Aberkane, Fouad ; Du, To–Anh ; Lawton, Anika ; Nguyen-Pham, Tai-Khoa ; Seriyüz, Nermin: *PDV Projekt NXT - Steuerung NXT & Bildverarbeitung Webcam.* Projektarbeit, Fachhochschule Wiesbaden, 2008. `http://www.cs.hs-rm.de/~linn/vpdv0708/nxt/downloads/pdv_dokumentation.pdf`. – Letzter zugriff am 25.10.2012

[2] Agoston, Max K.: *Computer Graphics and Geometric Modelling: Implementation & Algorithms.* 1. Auflage. Springer-Verlag, London, 2005

[3] Bader, Franz ; Dorn, Friedrich: *Physik - Gymnasium Gesamtband - Sek II.* 1. Auflage. Schroedel-Verlag GmbH & Co. KG, Hannover, 2000

[4] Berns, Karsten ; Schmidt, Daniel: *Programmierung mit LEGO® Mindstorms® NXT.* 1. Auflage. Springer-Verlag, Heidelberg, Dordrecht, London, New York, 2010

[5] den Hartog, Leo: *Lego Mindstorms NXT Camera.* Semesterthesis, Eidgenössische Technische Hochschule Zürich, 2008. `ftp://ftp.tik.ee.ethz.ch/pub/students/2008-HS/SA-2008-18.pdf`. – Letzter Zugriff am 25.10.2012

[6] Deutsches Institut für Normung: *Informationsverarbeitung - Begriffe.* Beuth-Verlag, Berlin, Köln, 1985

[7] Deutsches Institut für Normung: *Regelungstechnik und Steuerungstechnik - Allgemeine Begriffe.* Beuth-Verlag, Berlin, Köln, 1994

[8] Koob, Gary M. ; Lau, Clifford G.: *Foundations of Dependable Computing: Paradigms for Dependable Applications.* 1. Auflage. Kluwer Academic Publishers, Dordrecht, 1994

[9] Kuhn, Thomas S.: *Die Struktur der wissenschaftlichen Revolutionen.* 2. Auflage. Suhrkamp-Verlag, Frankfurt, 1969

[10] Leimbach, Thorsten ; Trella, Sebastian: *LEGO Mindstorms NXT Programmiersprachen im Überblick.* Fraunhofer-Institut Intelligente Analyse- und Informationssysteme (IAIS), 2010. `http://www.roberta-home.de/sites/default/files/Vergleich%20Programmiersprachen%20NXT%20v-1.3.pdf`. – Letzter zugriff am 25.10.2012

[11] Litz, Lothar: *Grundlagen der Automatisierungtechnik: Regelungssysteme - Steuerungssysteme - Hybride Systeme.* Oldenburg Wissenschaftsverlag, München, 2005 (1. Auflage)

[12] Mindsensors.com: *NXTCam v4 User Guide.* 2012. `http://www.mindsensors.com/index.php?module=documents&JAS_DocumentManager_op=downloadFile&JAS_File_id=1044`. – Letzer Zugriff am 10.08.2012

[13] Mindstorms.lego.com: *Product Information - 9842 Interactive Servo Motor.* 2012. `http://mindstorms.lego.com/en-us/products/default.aspx#9842`. – Letzter Zugriff am 12.09.2012

[14] Moros, Ralf ; Luft, Frank L. ; Papp, Helmut: *Grundlagen Regelung - Regler - PID-Regler.* Fachinformationszentrum Chemie GmbH - ChemgaPedia. `http://www.chemgapedia.de/vsengine/vlu/vsc/de/ch/7/tc/regelung/grundlagen/regelung_grundlagen.vlu.html`. – Letzter Aufruf am 08.11.2012

[15] Niemann, Gustav ; Winter, Hans: *Maschinenelemente - Band 2: Getriebe allgemein, Zahnradgetriebe - Grundlagen, Stirnradgetriebe.* 2. Auflage. Springer-Verlag, Berlin, Heidelberg, 2003

[16] Schmidt, Irina: *Didaktische Analyse von Echtzeitsystemen unter besonderer Berücksichtigung des „Wippe-Experiments“ für den Informatikunterricht.* Bachelorarbeit, Universität Koblenz-Landau, Campus Koblenz, 2011,

[17] Schröder, Joachim ; Gockel, Tilo ; Dillman, Rüdiger: *Embedded Linux: Das Praxisbuch.* 1. Auflage. Springer-Verlag, Dordrecht, heidelberg, London, New-York, 2009

[18] Smuts, Jacques: *PID Controllers Explained.* Plant Automation Services (PAS), Inc. of Houston, Texas, 2002. `http://de.scribd.com/doc/7261195/PIDControllers-Explained`. – Letzter Zugriff am 25.10.2012

[19] Sourceforge.net: *Bricx Command Center 3.3.* 2009. `http://bricxcc.sourceforge.net/`. – Letzter Zugriff am 09.07.2012

[20] Sourceforge.net: *NXC Guide - Version 1.2.1 r5.* 2011. `http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf`. – Letzter Zugriff am 25.10.2012

[21] Stahlhofen, Andreas: *Programmierung und Implementierung des Regelalgorithmus und Überarbeitung der Mensch-Maschine-Schnittstelle für das „Wippe-Experiment“.* Diplomarbeit, Universität Koblenz-Landau, Campus Koblenz, 2011,

[22] Stankovic, John A.: *Real-Time and Embedded Systems.* In: *ACM Computing Surveys, Vol. 28, No. 1, March 1996*

[23] Wallentowitz, Henning ; Reif, Konrad: *Handbuch Kraftfahrzeugelektronik.* 1. Auflage. Springer-Vieweg-Verlag, Wiesbaden, 2006

[24] Wörn, Heinz: *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen.* 1. Auflage. Springer-Verlag, Berlin, Heidelberg, 2005

[25] ZÖBEL, Dieter: *Echtzeitsysteme - Grundlagen der Planung.* 1. Auflage. Springer-Verlag, Berlin, Heidelberg, 2008

[26] ZÖBEL, Dieter: *A Versatile Real-Time Experiment: Balancing a Ball on a Flat Board.* In: *Third IEEE Real-Time Systems Education Workshop (RTEW'98), Poznan, Poland, November, 1998*