UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Programmierung und Implementierung des Regelalgorithmus und Überarbeitung der Mensch-Maschine-Schnittstelle für das „Wippe-Experiment“

Diplomarbeit

zur Erlangung des Grades eines
Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Andreas Stahlhofen

Betreuer: Prof. Dr. Dieter Zöbel, Institut für Informatik: AG Echtzeitsysteme
Erstgutachter: Prof. Dr. Dieter Zöbel, Institut für Informatik: AG Echtzeitsysteme
Zweitgutachter: Alexander Hug, Arbeitsgebiet Fachdidaktik Informatik (FB4)

Koblenz, im August 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Echtzeitsysteme spielen in der heutigen Zeit in vielen Anwendungsbereichen eine bedeutende Rolle. Beispiele hierfür finden sich in der Robotik, der Fabrikautomation, der Medizintechnik, aber auch in Bereichen aus dem täglichen Leben, wie im Automobilbereich oder in der Mobilkommunikation. Trotz der ständig wiederkehrenden Begegnung mit ihnen im Alltag, sind Echtzeitsysteme häufig innerhalb von größeren Systemen eingebettet und können somit nur selten vom jeweiligen Nutzer bewusst als eigenständig wahrgenommen werden. Um spezifische Eigenschaften des Gebiets Echtzeitsysteme im Schulunterricht oder in der Ausbildung an Hochschulen den Lernenden sichtbar zu machen, existiert im Labor der Arbeitsgruppe Echtzeitsysteme der Universität Koblenz-Landau ein Versuchsaufbau mit dem Namen „Wippe“, bei dem eine Kamera die Bewegung einer Kugel auf einer ebenen Fläche aufnehmen und vermessen kann. Durch die Neigung der Fläche in zwei Achsen ist die Wippe prinzipiell in der Lage, die Kugel zu bewegen und zum Halten zu bringen. Insbesondere soll verhindert werden, dass die Kugel von der Fläche fällt.
Die vorliegende Arbeit beschäftigt sich mit der Analyse und Überarbeitung des Wippe-Systems. Außerdem wird ein Regelalgorithmus für eine automatische Steuerung des Systems realisiert. Das Wippe-Experiment soll für didaktische Zwecke eingesetzt werden und ein offenes System darstellen, welches die Merkmale und Eigenschaften eines Echtzeitsystems exemplarisch verdeutlicht.

1.2 Zielsetzung der Arbeit

Die Hauptaufgabe der Arbeit besteht zunächst darin, einen funktionierenden Regelalgorithmus zum Balancieren der Kugel im Zentrum der Platte zu implementieren und in das bereits vorhandene Softwaresystem zu integrieren. Durch neue Anforderungen, welche sich während des Bearbeitens der Aufgabe ergeben, entwickelt sich die Zielsetzung der Arbeit weiter. Die Hauptaufgabe verlagert sich dahingehend, unter Berücksichtigung des bisher verwendeten Softwaresystems zur Steuerung der

Wippe, eine überarbeitete Version zu entwickeln und anschließend bezüglich den Eigenschaften eines Echtzeitsystems zu analysieren. Die gewonnenen Erkenntnisse werden schließlich verwendet, um den Regelalgorithmus zu realisieren und sauber in das Softwaresystem zu integrieren. Dadurch ist es prinzipiell möglich, die Kugel auf der ebenen Platte im Zentrum zu balancieren.
Als bereits vorhanden gilt der technische Aufbau der Wippe, welcher aus einer Kamera mit einer entsprechenden Grafikkarte und zwei Schrittmotoren besteht, die über eine Kartensteuerung angesprochen werden können. Des Weiteren wird ein Verfahren innerhalb des alten Softwaresystem vorausgesetzt, welches aus dem Kamerabild die Position der Kugel berechnet. Als Referenz wird die Bachelor Arbeit von Sefa Usta [Ust10] verwendet, in welcher der technische Aufbau sowie wichtige Programmfragmente des Wippe-Experiments genauer dokumentiert sind.
Die verschiedenen Aufgaben, welche im Laufe der Arbeit bearbeitet werden, lassen sich folgendermaßen zusammenfassen:

- Überarbeitung der Hardwarekomponenten des Wippe-Experiments,
- Erneuerung des Softwaresystems inklusive der Mensch-Maschine-Schnittstelle,
- Analyse des neuen Systems bezüglich der Eigenschaften eines Echtzeitsystems,
- Implementierung und Integration des Regelalgorithmus.

Zunächst werden alte Hardwarekomponenten ausgetauscht, um somit einer mechanischen Fehlfunktion vorzubeugen und das System an später erhobene Anforderungen anzupassen. Anschließend werden große Teile des Programms zur Steuerung der Wippe unter Verwendung bekannter Praktiken aus dem Gebiet der Softwaretechnik analysiert und implementiert. Dazu zählt unter anderem die Erneuerung der grafischen Benutzeroberfläche, um eine intuitive Bedienung des Systems zu ermöglichen. Das daraus entstandene, aktualisierte System wird bezüglich den Eigenschaften eines Echtzeitsystems analysiert und ausgewertet. Anhand der daraus abgeleiteten Erkenntnisse wird ein Regelalgorithmus entwickelt, welcher auf der Basis der durch die Kamera ermittelten Daten und der Neigung der Platte die Reaktion zum Ausbalancieren der Kugel berechnet. Ziel ist es, dass die Kugel im automatischen Betrieb nicht von der Platte fällt und im Zentrum balanciert wird.
Der Aufbau dieser Arbeit orientiert sich an der Abarbeitung der einzelnen Aufgaben. Zunächst werden in Kapitel 2 verwandte Arbeiten genannt und erläutert. In einem weiteren Abschnitt wird der Stand des Versuchsaufbaus Wippe vor Beginn des Überarbeitungsprozesses erläutert. Kapitel 3 gibt eine Einführung in die verschiedenen Themengebiete, welche für den Überarbeitungsprozess des Wippe-Experiments und die Implementierung des Regelalgorithmus relevant sind. Dabei werden wichtige Grundlagen erläuternd dargestellt, welche für das Verständnis der Arbeit von Bedeutung sind. In Kapitel 4 wird der Überarbeitungsprozess des alten Wippe-Systems beschrieben. Dieser umfasst den Umbau der verwendeten Hardware und die Erneuerung des Softwaresystems. Anschließend erfolgt in Kapitel 5 eine

Analyse des überarbeiteten Systems. Dabei werden zunächst Messungen bezüglich der Latenz der einzelnen Softwarekomponenten und der Kamera durchgeführt und zusammengefasst. Im Folgenden werden hierauf basierend die Eigenschaften in Bezug auf ein Echtzeitsystem ausgearbeitet und erörtert. Darauf aufbauend schließt sich in Kapitel 6 der Entwurf und die praktische Umsetzung des Regelalgorithmus zur automatischen Steuerung der Wippe an. Kapitel 7 beinhaltet eine Zusammenfassung der vorliegenden Arbeit und neben der Vorstellung weiterer Ideen zur Verbesserung und Weiterentwicklung des Systems wird ein abschließendes Fazit gezogen.

Kapitel 2

Der Versuch Wippe

In den folgenden Abschnitten werden zunächst verwandte Arbeiten vorgestellt, welche einen groben Überblick über den aktuellen Stand in Bezug auf die Hauptthematik dieser Arbeit liefern. Daraufhin schließt sich die Erläuterung des Zustands des Versuchsaufbaus Wippe zu Beginn der vorliegenden Arbeit an. Zunächst wird der mechanische Aufbau sowie die verwendete Hardware betrachtet. Daraufhin werden die wichtigsten Eigenschaften und Komponenten der bislang verwendeten Software aufgegriffen und kurz erläutert.

2.1 Verwandte Arbeiten

In dem Buch „Control System Design“ [CGGS00] erläutern Goodwin et. al den Aufbau und Entwurf von Regelsystemen. Als Beispiel wird unter anderem ein *Ball on a Beam System* aufgeführt. Dieses soll mit Hilfe eines komplexen Regelalgorithmus gesteuert werden. Eine entsprechende Simulation in Form eines Java Applets findet sich auf der Internetseite des Buches[1].
In [FZT04] wird von Fan et. al eine Lösung zur automatischen Steuerung eines *Ball on a Plate Systems* vorgestellt. Insbesondere soll die Kugel einen vorgegebenen Pfad verfolgen und die benötigte Zeit als Indikator für die Güte des Kontrollsystems verwendet werden. Um die Bewegung der Kugel und die Auslenkung der Platte entsprechend zu planen, wird das System basierend auf Fuzzylogik entworfen. Das Testen des Endproduktes erfolgt lediglich mit Hilfe einer Computer-Simulation, wodurch zwar die Lauffähigkeit des Systems, jedoch nicht die Korrektheit unter realen Bedingungen nachgewiesen ist.
Awtar et. al beschreiben in [ABB+02] den Entwurf eines weiteren Ball on a Plate Systems. Dieses wurde von Studenten innerhalb eines Kurses des *Rensselaer Polytechnic Institute* (RPI) entwickelt. Im Vordergrund stehen hierbei jedoch die Planung und Umsetzung eines Projekts im Gebiet der Mechatronik. Als Platte wird ein Touchscreen verwendet, mit dessen Hilfe die Kugelkoordinaten bestimmt werden

[1] `http://csd.newcastle.edu.au/simulations/ball_sim.html`, Abgerufen am 15.06.2011

können. Entsprechend muss die Kugel ein ausreichendes Minimalgewicht besitzen. Der Regelalgorithmus wurde in MATLAB umgesetzt und ermöglicht die statische Ansteuerung von vorgegebenen Koordinaten, sowie das Verfolgen einer kreisförmigen Bahn durch die Kugel. Eine ausführlichere und verbesserte Version der Arbeit wird in [ACH04] von Andrew, Colasuonno und Hermann vorgestellt. Als Intention für dieses Projekt werden der kommerzielle Vertrieb sowie der Einsatz in der Lehre genannt.
Die tschechische Firma Humusoft® bietet auf ihrer Internetseite[2] ein fertiges *Ball on a Plate System* zum Verkauf an. Ähnlich zum Versuchsaufbau „Wippe“, wird die Kugelposition mit Hilfe einer Kamera erfasst. Das Auslenken der Platte übernehmen dabei ebenfalls zwei Schrittmotoren. Zum Umfang des Angebotes gehören zusätzliche Interface-Bibliotheken, geschrieben in der Borland C Sprache und ein Softwarepaket zur Demonstration verschiedener Regelalgorithmen. Hinsichtlich der Kosten werden keine Angaben gemacht.
Bei weiteren Recherchen im Internet finden sich vor allem auf der Video Plattform YouTube Filmaufnahmen zu dem hier behandelten Thema. Gezeigt werden dort Beispiele für *Ball on a Plate Systeme*, welche die Kugel an eine bestimmte Position steuern können oder einem vorgegebenen Pfad folgen lassen[3]. Die dort vorgestellten Projekte sind weder ausführlich dokumentiert noch erläuternd beschrieben und können somit einer wissenschaftlichen Verwendung nicht zugeführt werden.

2.2 Mechanischer Aufbau

Die Arbeitsgruppe Echtzeitsysteme der Universität Koblenz-Landau verfügt über einen Versuchsaufbau „Wippe“. Dieses im Labor der Arbeitsgruppe aufgebaute Experiment, kann als Erweiterung eines *Ball on a Beam Systems* aufgefasst werden. Hierbei wird eine Kugel nur eindimensional auf einer Strecke bewegt, mit dem Versuch diese auszubalancieren. Die Kugel läuft in einer Art Rinne, welche über den

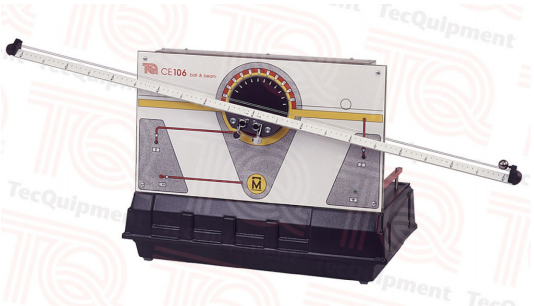

Abbildung 2.1: The CE106 Ball and Beam System.

[2] `http://www.humusoft.cz/produkty/models/ce151/`, `Abgerufenam19.07.2011.`
[3] `http://www.youtube.com/watch?v=uERF6D37E_o`, Abgerufen am 20.04.2011.

Mittelpunkt ausgelenkt werden kann, mit dem Ziel sie in der Mitte beziehungsweise als Erweiterung an einer beliebigen Stelle zu balancieren. Eine Darstellung des Systems erfolgt in Abbildung 2.1. Im Gegensatz zum Ball on a Beam System wird

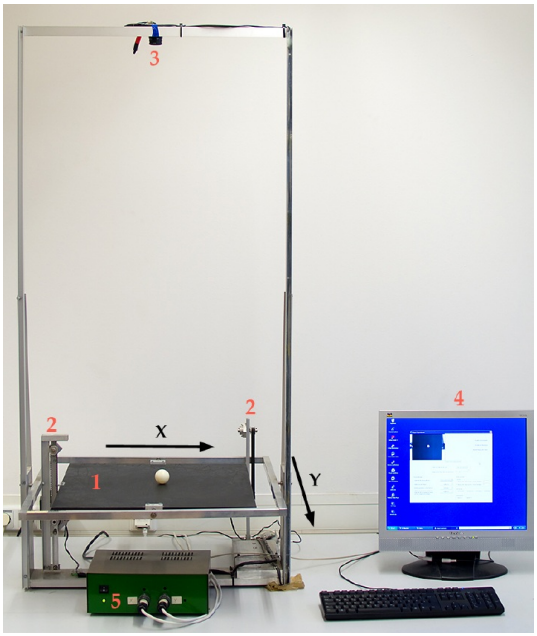

Abbildung 2.2: Der Versuchsaufbau des Wippe-Experiments.

beim Wippe-Experiment anstatt der Rinne eine Platte verwendet, welche in zwei Achsen geneigt werden kann. Entsprechend bewegt sich die Kugel nun auf einer Ebene und stellt somit ein komplexes System dar. Ziel ist es, die Kugel im Zentrum oder an eine anderen beliebigen Stelle der Platte balancierend zu positionieren.
Abbildung 2.2 gibt einen Überblick über die Zusammensetzung der einzelnen Hardwarekomponenten. Die Kugel befindet sich auf einer schwarzen, quadratischen Platte (1) mit einer Seitenlänge von 50 Zentimetern. Mit Hilfe von zwei Schrittmotoren (2) kann diese in x- und y-Richtung ausgelenkt werden. Die Steuerung wird mittels des TMCM-310 der Firma Trinamic (5) realisiert, ein 3-Achsen Schrittmotor Steuerungs- und Treibermodul [Tri11], welches von einem Desktop Computer (4) über die serielle Schnittstelle (RS232) betrieben wird. Der Computer verfügt über eine eingebaute TV-Karte, welche über einen Cinch-Eingang mit einer CCD Kamera verbunden ist. Diese ist senkrecht über dem Zentrum der Platte montiert (3). Die Software zum Steuern der Wippe wird auf dem Computer ausgeführt. Diese verwendet die von der Kamera gelieferten Bilder, um die Kugelbewegung zu bestimmen und anschließend die Motoren entsprechend anzusteuern. Somit ist das System prinzipiell in der Lage, die Kugel auf der Platte zu balancieren.

2.3 Aufbau des bisher verwendeten Softwaresystems

Mit Hilfe der auf dem Computer ausgeführten Software ist die Steuerung der Wippe möglich. Dabei wird zwischen manuellem und automatischem Modus unterschieden. Für die Variante der manuellen Steuerung stehen aktuell zwei Möglichkeiten zur Verfügung. Die erste wird mittels der Pfeiltasten einer üblichen Computer-Tastatur realisiert. Eine zweite Alternative bietet die Fernbedienung Wii-Remote der Firma Nintendo dar. Dabei kann die Wippe durch einfaches Neigen der Wii-Remote ausgelenkt werden. Für die automatische Steuerung existiert zu Beginn der Arbeit lediglich ein einfacher Ansatz eines Regelalgorithmus, welcher jedoch beim praktischen Test versagt und somit als nicht zufriedenstellend beurteilt werden muss.

2.3.1 Softwarekomponenten

Abbildung 2.3 zeigt die wichtigen Softwarekomponenten der initialen Version des Regelungssystems in Form eines Klassendiagramms. Im Folgenden werden die einzelnen Klassen beschrieben und in den Gesamtkontext eingeordnet [4].

EZVideoCaptureDlg Die Klasse EZVideoCaptureDlg stellt die Schnittstelle zur Benutzeroberfläche dar. Sie definiert die Reaktion des Programms auf eine vom Nutzer getätigte Interaktion, wie zum Beispiel das Anklicken eines Buttons. In ihrer Initialisierungsmethode OnInitDialog() instanziiert und startet sie den MainController.

Thread Die abstrakte Klasse Thread bildet die Oberklasse für die verschiedenen Komponenten, welche für die Regelung der Wippe zuständig sind. Jede von Thread abgeleitete Klasse muss die Methode step() implementieren. Darin werden Berechnungen oder Aktionen der entsprechenden Komponente definiert. Die Bezeichnung Thread ist in diesem Fall irreführend, da mittels dieser Klasse keine nebenläufigen Prozesse initiiert werden.

MainThread Die Klasse MainThread ist eine von Thread abgeleitete Klasse. Sie erhält bei ihrer Initialisierung eine Referenz auf die Instanz der Klasse MainController. Innerhalb der Methode step() wird lediglich die Methode doExecute() der Instanz der Klasse MainController aufgerufen.

MotorControlUnit Die von Thread abgeleitete Klasse MotorControlUnit beschreibt die Schnittstelle zum Treibermodul TMCM-310 für die Steuerung der Schrittmotoren. Sie instanziiert eine Verbindung über die serielle Schnittstelle und konfiguriert das Treibermodul mit spezifischen Standardwerten. Mittels der Methode setTarget()

[4] Eine detaillierte Dokumentation des Systems wird in [Ust10] vorgenommen.

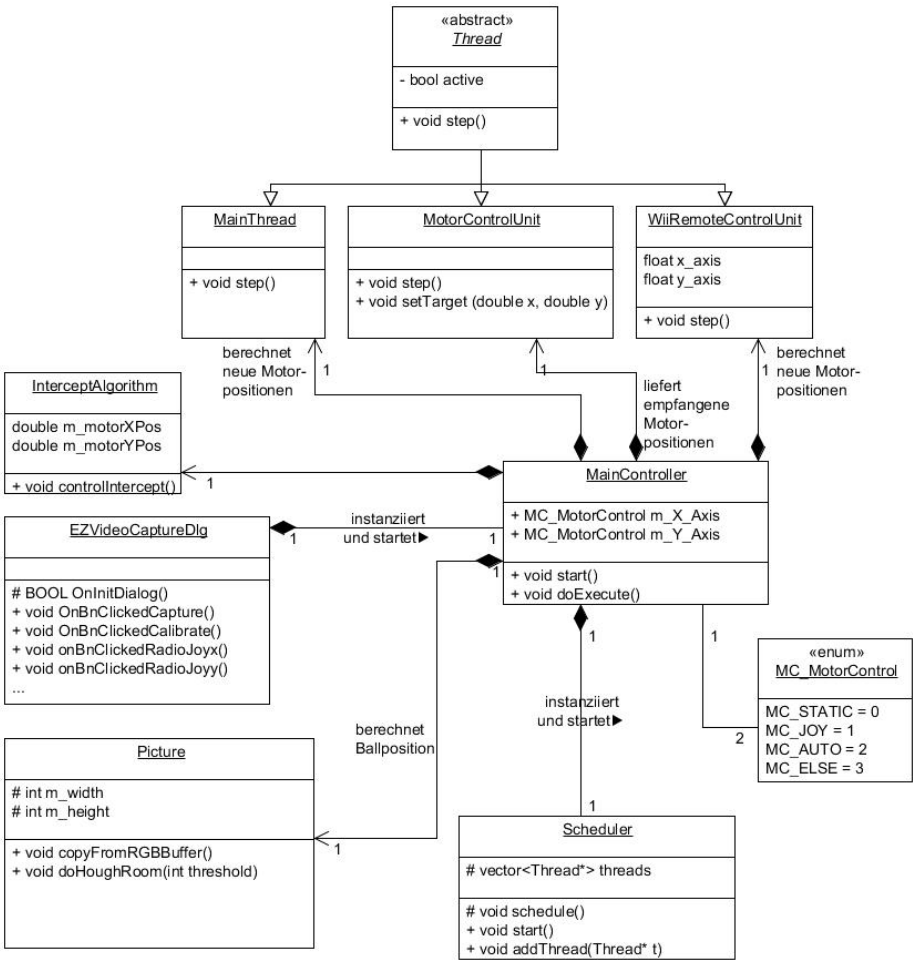

Abbildung 2.3: Klassendiagramm des Softwaresystems.

kann die Position der zwei Motoren verändert werden. Das entsprechende Kommando wird jedoch erst in der Methode step() ausgeführt.

WiiRemoteControlUnit Mit Hilfe der Klasse WiiRemoteControlUnit lässt sich die Steuerung mittels Wii-Remote realisieren. Durch spezifische Berechnungen werden in der Methode step(), geerbt von der Klasse Thread, die neuen Motorpositionen anhand der Neigung der Wii-Remote bestimmt und in den Variablen x_axis und y_axis gespeichert. Diese können dann verwendet werden, um die Motoren entsprechend zu steuern [5].

Picture Die Klasse Picture stellt Methoden zur Bildverarbeitung zur Verfügung. Durch den Aufruf der Methode copyFromRGBBuffer() empfängt sie das aktuelle

[5] Eine detaillierte Beschreibung bezüglich der Steuerung mit Hilfe der Wii-Remote findet sich in der Arbeit von Yohan Humbert [Hum11].

Kamerabild und speichert es in einen Byte-Puffer. Innerhalb der Methode doHoughRoom() wird mittels der Hough Transformation [6] die aktuelle Kugelposition ermittelt und in einen statischen Speicher geschrieben.

InterceptAlgorithm Innerhalb der Klasse InterceptAlgorithm befindet sich der Ansatz zur automatischen Regelung der Wippe. Die Methode controlIntercept() bezieht die von der Bildverarbeitung zur Verfügung gestellten Daten aus einem statischen Speicher und berechnet anhand der Kugelkoordinaten die neuen Motorpositionen. Dabei wird die Motorposition für die jeweilige Achse über die Differenz von Mittelpunkt der Platte und der entsprechenden Kugelkoordinaten gebildet. Diese werden in den Variablen m_motorXPos und m_motorYPos gespeichert und zur Weiterverwendung zur Verfügung gestellt. Dieser Ansatz funktioniert jedoch nicht korrekt.

Scheduler Die Aufgabe der Klasse Scheduler ist es, den Ablauf der verschiedenen Prozesse zur Steuerung der Wippe zu synchronisieren. Innerhalb der Methode start() wird ein nebenläufiger Prozess gestartet, welcher die Methode schedule() aufruft. Dort wird in einer Endlosschleife über den Vektor thread iteriert, welcher die vom MainController hinzugefügten Instanzen der Klasse Thread enthält und in jedem Iterationsschritt die Methode step() des aktuellen Elements aufruft.

MainController Der MainController kann als Kernklasse der Software betrachtet werden. Die Eigenschaften m_X_Axis und m_Y_Axis ermöglichen die Art der Steuerung der Wippe zu definieren. Unterschieden wird allgemein zwischen automatischem und manuellem Modus.
Innerhalb des Konstruktors wird der Scheduler instanziiert und gestartet. Dabei wird für jede von Thread abgeleitete Klasse eine Instanz erzeugt und mittels der Methode addThread() dem Scheduler hinzugefügt. Zunächst wird die Instanz der Klasse MainThread mit der Referenz auf den MainController der Methode addThread() übergeben, danach die Instanz der Klasse MotorControlUnit. Das Einhalten der Reihenfolge ist wichtig für den Datenfluss, welcher zum Teil vom MainController kontrolliert wird.
Innerhalb der Methode doExecute() wird im Fall der automatischen Steuerung der Wippe zunächst mit Hilfe der Bildverarbeitungsmethoden der Klasse Picture die Kugelposition bestimmt und daran anschließend folgt mittels der Klasse InterceptAlgorithm die Berechnung der neuen Motorpositionen. Diese werden am Ende der Instanz der MotorControlUnit übergeben. Im Falle der manuellen Steuerung, beispielsweise durch die Wii-Remote, werden zunächst die neuen Motorpositionen aus der Instanz der WiiRemoteControlUnit gelesen und anschließend der Instanz der MotorControlUnit übermittelt.

[6] Methode der Bildverarbeitung, um Linien und Kreise innerhalb eines Binärbildes zu erkennen (vgl. [FPWW03]).

Die Methode start() ist aufgrund der Vollständigkeit im Diagramm aufgeführt. Diese startet lediglich einen nebenläufigen Prozess, welcher genau einmal die Methode doExecute aufruft. Die Kenntnis, dass die Klasse MainThread diese Aufgabe bereits durch den Aufruf der step()-Methode innerhalb der Endlosschleife im Scheduler erfüllt, weist die Methode start() als überflüssig aus.

2.3.2 Programmablauf und Datenfluss

Dieser Abschnitt hat eine erläuternde Darstellung des Programmablaufs der Software zum Inhalt. Hierzu vermittelt Abbildung 2.4 einen groben Überblick zum Programmrahmen. Im Besonderen wird auf den Ablauf des automatischen Betriebs der Wippe eingegangen und dieser nachfolgend detailliert aufgezeigt.

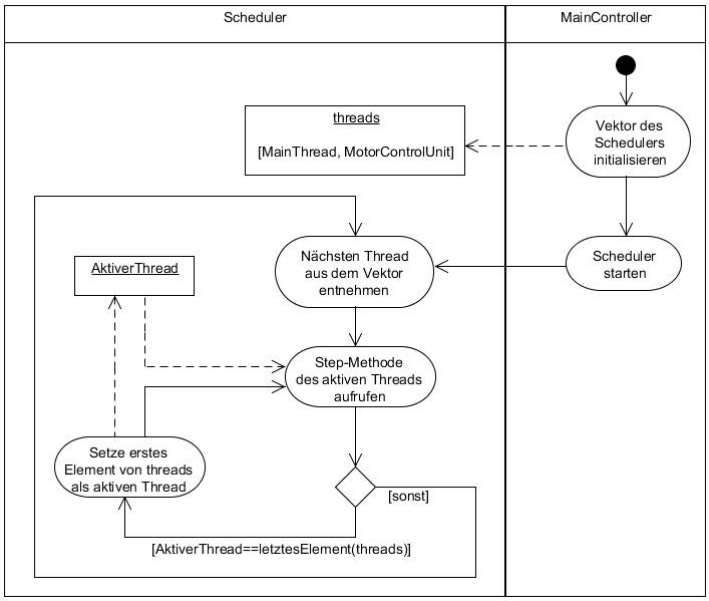

Abbildung 2.4: Allgemeiner Programmablauf der Kontrollschleife des Schedulers im automatischen Betrieb.

In der Anfangsphase instanziiert der MainController den Scheduler und initialisiert dessen Vektor mit Thread-Instanzen, indem zuerst eine Instanz des MainThreads und daran anschließend eine Instanz der MotorControlUnit hinzugefügt werden. Die Reihenfolge orientiert sich dabei ausschließlich an dem logischen Ablauf des Regelprozesses. Zunächst werden anhand der Kugelkoordinaten, bestimmt durch ein Verfahren der Bildverarbeitung, innerhalb des MainThreads neue Motorpositionen für die Schrittmotoren berechnet. Anschließend werden diese mit Hilfe der MotorControlUnit an das Treibermodul gesendet, was schließlich die Auslenkung der Motoren bewirkt. Durch den Aufruf der Methode start() des Schedulers, wird die Kontrollschleife betreten. Dabei wird bei jedem Durchlauf der nächste Thread aus dem Vektor threads entnommen und dessen step-Methode aufgerufen. Es schließt

sich dann eine Wiederholung des gesamten Vorgangs an. Wurde der Vektor komplett durchlaufen, wird wieder mit dem ersten Element begonnen.

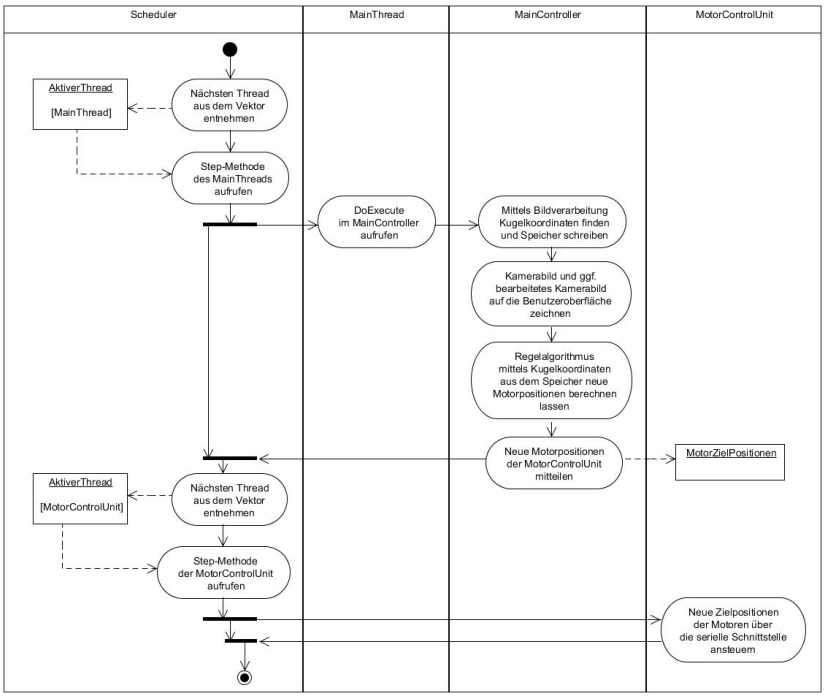

Abbildung 2.5: Programmablauf eines Zyklus innerhalb des Schedulers im automatischen Betrieb.

Abbildung 2.5 zeigt den Ablauf eines Regelungszyklus im automatischen Betrieb der Wippe. Zu Beginn wird die Instanz des MainThreads als aktiver Thread gesetzt. Mit dem Aufruf der step-Methode wird die Methode doExecute() des MainControllers ausgeführt. Dort werden mittels der Bildverarbeitungsalgorithmen der Klasse Picture die aktuellen Kugelkoordinaten bestimmt und in einen statischen Speicher geschrieben. Das von der Kamera aufgenommene Bild wird im nächsten Schritt auf die Benutzeroberfläche gezeichnet. Sofern vom Nutzer vorab eine entsprechende Auswahl getroffen ist, kann auch das bearbeitete Bild angezeigt werden. Im weiteren Verlauf wird der Regelalgorithmus in der Klasse InterceptAlgorithm aufgerufen, welcher mit Hilfe der Kugelkoordinaten aus dem statischen Speicher die Motorpositionen berechnet, um die Kugel auf der Platte zu balancieren. Die abschließende Aufgabe des MainControllers ist es, diese an die MotorControlUnit weiterzuleiten. Im Scheduler wird nun der nächste Thread aus dem Vektor entnommen, in diesem Fall die Instanz der MotorControlUnit. Der Aufruf der step-Methode bewirkt das Senden der Befehle über die serielle Schnittstelle, durch welche die Schrittmotoren in die zuvor berechneten Positionen ausgelenkt werden. Jetzt ist der Regelungszyklus beendet und wird innerhalb der Kontrollschleife erneut durchgeführt.

2.3.3 Grafische Benutzerschnittstelle

Es folgt eine Erläuterung zur grafischen Benutzerschnittstelle der Software. Abbildung 2.6 zeigt einen Screenshot dieser Benutzerschnittstelle, wie sie zu Begin der Diplomarbeit vorgelegen hat. Im Bereich der Voreinstellungen (1) müssen unterschiedliche Parameter vor dem ersten Start der Regelung konfiguriert werden. Mit dem Button „Videoquelle festlegen“ kann der Cinch-Eingang der eingebauten TV-Karte ausgewählt werden. Der Button „Kalibrieren“ öffnet einen Dialog, welcher es ermöglicht, die Wippe vor dem Start der Regelung mit der Tastatur in eine initiale Position zu steuern. Die beiden weiteren Buttons aus dem Bereich (1) werden für das Verbinden und Kalibrieren der Wii-Remote verwendet. Im Bereich (2) der Benutzerschnittstelle kann der Steuerungsmodus ausgewählt werden. Als Optionen stehen die manuelle Steuerung mittels Tastatur oder Wii-Remote und die automatische Steuerung mit Hilfe des Regelalgorithmus zur Verfügung. Durch Anklicken des Buttons „Start des Versuchs“ (3) wird die Wippe zur Steuerung freigegeben. In der linken oberen Ecke wird das aktuelle Kamerabild angezeigt (4). Das durch die Bildverarbeitungsalgorithmen bearbeitete Kamerabild kann durch Aktivieren des Buttons „Zeige transformierte Ansicht“ (5) rechts neben dem ursprünglichen Bild sichtbar gemacht werden.

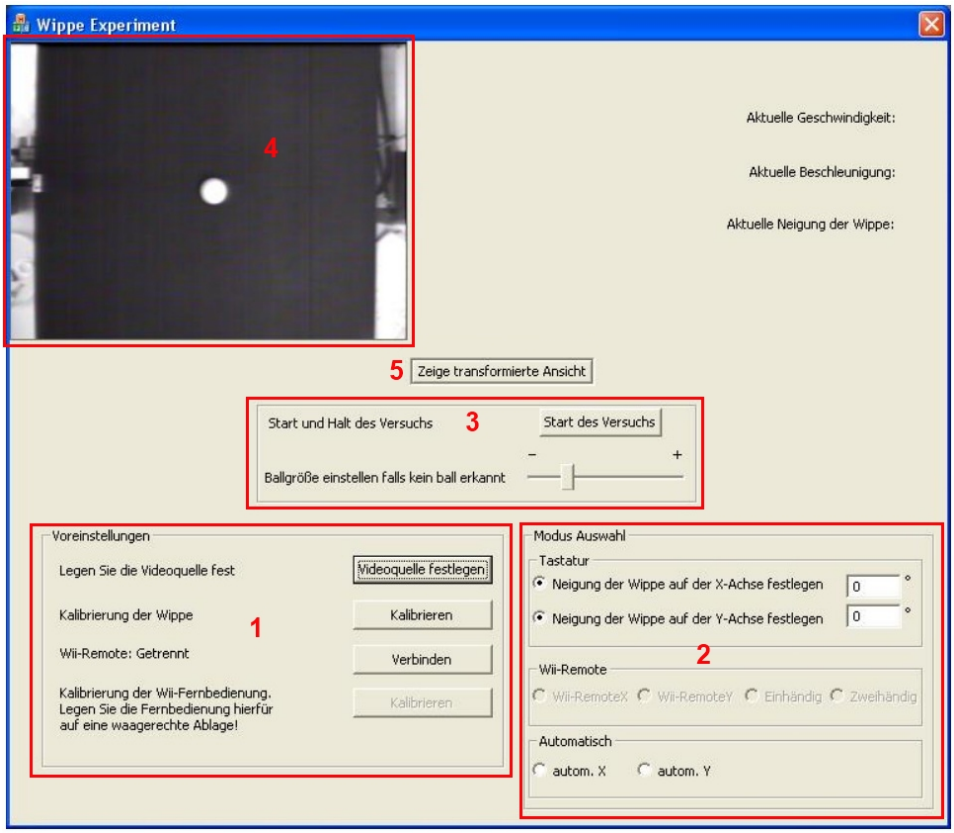

Abbildung 2.6: Benutzerschnittstelle der Software

Kapitel 3

Grundlagen

Um den Prozess der Überarbeitung des Systems Wippe verständlich beschreiben zu können, müssen einführend theoretische Grundlagen aus verschiedenen Bereichen thematisiert werden. Zunächst werden die physikalischen Aspekte bezüglich des Wippe-Experiments betrachtet. Im Anschluss daran wird ein Überblick über das Themengebiet Echtzeitsysteme gegeben und auf die spezifischen Eigenschaften eines solchen Systems eingegangen. Im letzten Abschnitt werden softwaretechnische Grundlagen erläutert, welche innerhalb des überarbeiteten Softwaresystems ihre Anwendung finden.

3.1 Physikalische Eigenschaften des Wippe-Experiments

Im Folgenden werden die physikalischen Eigenschaften des Wippe-Experiments erarbeitet und erläutert. Diese spielen bei der Entwicklung des Regelalgorithmus zur automatischen Steuerung der Wippe eine tragende Rolle. Zudem werden sie für die Analyse des Systems in Bezug auf die Eigenschaften eines Echtzeitsystems benötigt. Um die Komplexität des physikalischen Modells einzuschränken, werden folgende vereinfachende Annahmen getroffen:

- Es wird davon ausgegangen, dass die Kugel während des Versuchs kontinuierlich den Kontakt zur Platte hält, so dass eine Sprungbewegung ausgeschlossen werden kann.
- Die Reibung zwischen Kugel und Platte wird innerhalb der folgenden Rechnungen vernachlässigt, wodurch die Bestimmung einer vom Material der Platte und der Kugel abhängigen Reibungskonstanten nicht nötig ist.
- Einhergehend mit dem Vernachlässigen der Reibung wird ebenso der Schlupf der Kugel innerhalb der folgenden Rechnungen nicht betrachtet. Der Schlupf spezifiziert das auftretende Durchdrehen oder Gleiten der Kugel, falls durch eine große Beschleunigungs- beziehungsweise Bremskraft die maximale Haftreibung zwischen Kugel und Platte überschritten wird.

Die daraus folgend zu vernachlässigenden Eigenschaften sind in der praktischen Anwendung für eine korrekte Ausführung des Wippe-Experiments durchaus relevant. Deren Berücksichtigung würde sich allerdings in einem erheblichen Umfang auf die Komplexität des physikalischen Modells auswirken und damit den Rahmen dieser Diplomarbeit sprengen. Dies soll, wie bereits vorab erwähnt, vermieden werden.

3.1.1 Geschwindigkeit - Beschleunigung

Durch das Einwerfen der Kugel auf die Platte oder die Neigung der Platte in zwei Achsen wird die Kugel in Bewegung gesetzt und erhält somit eine Geschwindigkeit. Der Vektor der Gesamtgeschwindigkeit lässt sich in eine x- und eine y-Komponente unterteilen. Ist die Platte im Lot, handelt es sich um eine gleichförmige Bewegung. Durch beliebige Neigung der Platte wirkt zusätzlich noch eine beschleunigende Kraft auf die Kugel. In diesem Fall ergibt sich eine gleichmäßig beschleunigte Bewegung. Äquivalent zum Geschwindigkeitsvektor wird auch der Beschleunigungsvektor in eine x- und y-Komponente zerlegt (siehe Abbildung 3.1).

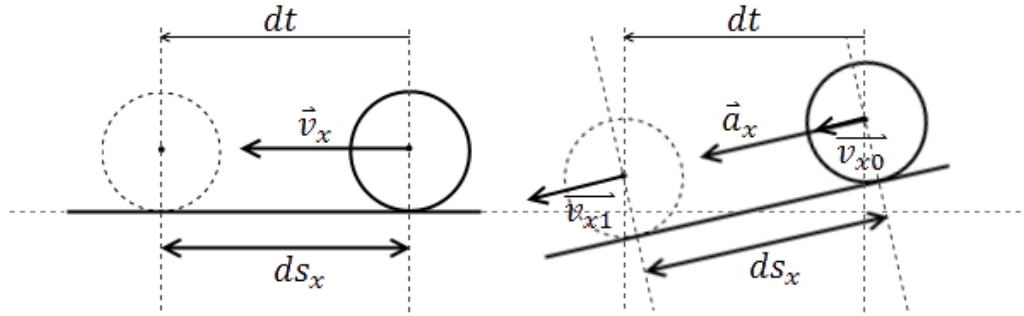

Abbildung 3.1: Bewegungsformen beim Wippe-Experiment (x-Achse).

Für die gleichförmige Bewegung (a) lassen sich die Vektoren der einzelnen Geschwindigkeitskomponenten folgendermaßen ausdrücken (siehe [BD00, S.16f.]):

$$\text{x-Komponente:} \quad |\vec{v_x}| = \frac{ds_x}{dt} \qquad \text{und somit} \qquad \vec{v_x} = \begin{pmatrix} |\vec{v_x}| \\ 0 \end{pmatrix} \tag{3.1}$$

$$\text{y-Komponente:} \quad |\vec{v_y}| = \frac{ds_y}{dt} \qquad \text{und somit} \qquad \vec{v_y} = \begin{pmatrix} 0 \\ |\vec{v_y}| \end{pmatrix} \tag{3.2}$$

Durch die Addition von $\vec{v_x}$ und $\vec{v_y}$ erhält man den Vektor, welcher die Gesamtgeschwindigkeit darstellt.

$$\text{Gesamtgeschwindigkeit:} \quad \vec{v} = \vec{v_x} + \vec{v_y} = \begin{pmatrix} |\vec{v_x}| \\ |\vec{v_y}| \end{pmatrix} \tag{3.3}$$

Im Fall einer gleichmäßig beschleunigten Bewegung (b) gibt die Beschleunigung die Geschwindigkeitsänderung in Abhängigkeit zu der Zeit an. Startet ein Körper aus der Ruhe, so gilt das folgende Zeit-Geschwindigkeit-Gesetz (siehe [BD00, S.26f.]):

$$v = a \cdot t \qquad \text{und somit} \qquad a = \frac{v}{t} \tag{3.4}$$

Ein Zusammenhang zwischen Beschleunigung und zurückgelegtem Weg lässt sich aus der Tatsache herleiten, dass die Geschwindigkeit die Wegänderung pro Zeit ist. Somit ist folgende Vorgehensweise legitim:

$$\dot{s} = v \;\Leftrightarrow\; s = \int a \cdot t \, dt \;\Leftrightarrow\; s = \frac{1}{2} \cdot a \cdot t^2 \tag{3.5}$$

In Bezug auf Abbildung 3.1 lässt sich die x- und y-Komponente der Beschleunigung folgendermaßen berechnen:

$$\begin{aligned} \text{x-Komponente:} \quad |\vec{a_x}| &= \frac{dv}{dt} = \frac{|\vec{v_{x1}}| - |\vec{v_{x0}}|}{dt} \\ \text{und somit} \quad \vec{a_x} &= \begin{pmatrix} |\vec{a_x}| \\ 0 \end{pmatrix} \end{aligned} \tag{3.6}$$

$$\begin{aligned} \text{y-Komponente:} \quad |\vec{a_y}| &= \frac{dv}{dt} = \frac{|\vec{v_{y1}}| - |\vec{v_{y0}}|}{dt} \\ \text{und somit} \quad \vec{a_y} &= \begin{pmatrix} 0 \\ |\vec{a_y}| \end{pmatrix} \end{aligned} \tag{3.7}$$

Zur Bildung des Vektors für die Gesamtbeschleunigung müssen nun $\vec{a_x}$ und $\vec{a_y}$ addiert werden:

$$\text{Gesamtbeschleunigung:} \quad \vec{a} = \vec{a_x} + \vec{a_y} = \begin{pmatrix} |\vec{a_x}| \\ |\vec{a_y}| \end{pmatrix} \tag{3.8}$$

3.1.2 Hangabtriebskraft

Wird die Platte ausgelenkt, wirkt eine beschleunigende Kraft auf die Kugel. Die Möglichkeit zur Berechnung der Beschleunigung anhand der Geschwindigkeitsänderung und des zugehörigen Zeitintervalls wird in Abschnitt 3.1.1 vorgestellt. Eine weitere Alternative lässt sich über die konstante Kraft formulieren, welche die Kugel beschleunigt, in diesem Fall die Hangabtriebskraft. Abbildung 3.2 veranschaulicht die Abhängigkeit der verschiedenen Kraftkomponenten anhand des Beispiels

der schiefen Ebene. Die Gewichtskraft $\vec{F_G}$ wird dabei in zwei Kraftkomponenten zerlegt, wobei $\vec{F_H}$ die Hangabtriebskraft und $\vec{F_N}$ die Normalkraft darstellt. Der Betrag der beiden Kraftvektoren ist dabei abhängig vom Winkel α, welcher übertragen auf das Wippe-Experiment die Auslenkung der Platte für eine Achse angibt.

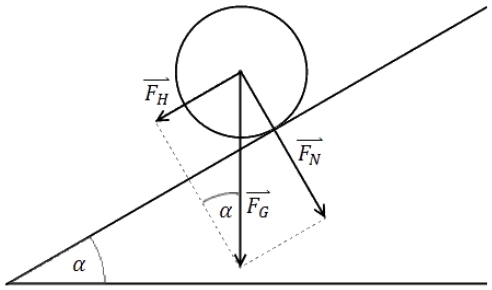

Abbildung 3.2: Darstellung der Hangabtriebskraft in der schiefen Ebene.

Folgende Formel gilt für eine gleichmäßig beschleunigte Bewegung, wobei F die konstante Kraft ist, welche den Körper beschleunigt (siehe [BD00, S.34f.]).

$$F = m \cdot a \qquad \text{und somit} \qquad a = \frac{F}{m} \tag{3.9}$$

Speziell für das Beispiel der schiefen Ebene wird die Beschleunigung durch die Hangabtriebskraft erzeugt. Somit gilt auch Folgendes:

$$F_H = m \cdot a \qquad \text{und somit} \qquad a = \frac{F_H}{m} \tag{3.10}$$

Die Gewichtskraft F_G lässt sich mittels der folgenden Formel berechnen (siehe [BD00, S.12f.]:

$$F_G = m \cdot g \qquad \text{mit} \qquad g = 9,81\frac{m}{s^2} \quad \text{(Gravitationskonstante)} \tag{3.11}$$

Mit Hilfe des Winkels α lässt sich folgender trigonometrischer Zusammenhang zwischen Gewichtskraft und Hangabtriebskraft erschließen:

$$\sin\alpha = \frac{F_H}{F_G} \qquad \text{und somit} \qquad F_H = F_G \cdot \sin\alpha \tag{3.12}$$

Durch Einsetzen von (3.10) und (3.11) in (3.12) ergibt sich folgende Abhängigkeit:

$$F_H = F_G \cdot \sin\alpha \;\Leftrightarrow\; m \cdot a = m \cdot g \cdot \sin\alpha \;\Leftrightarrow\; a = g \cdot \sin\alpha \tag{3.13}$$

Unter Beachtung der Trägheitsgesetze gilt speziell für die Beschleunigung einer gefüllten Kugel auf einer schiefen Ebene laut [Zö98] ein leicht modifizierter Zu-

sammenhang:

$$a = \frac{5}{7} \cdot g \cdot \sin \alpha \tag{3.14}$$

Die erarbeitete Formel für die Beschleunigung der Kugel a in Abhängigkeit zum Auslenkungswinkel der Platte α gilt für die x- und y-Achse. Entsprechend können die Teilkomponenten für x- und y-Achse gebildet und durch Addition der Vektor für die Gesamtbeschleunigung berechnet werden.

$$\text{x-Komponente:} \quad \vec{a_x} = \frac{5}{7} \cdot g \cdot \begin{pmatrix} \sin \alpha_x \\ 0 \end{pmatrix} \tag{3.15}$$

$$\text{y-Komponente:} \quad \vec{a_y} = \frac{5}{7} \cdot g \cdot \begin{pmatrix} 0 \\ \sin \alpha_y \end{pmatrix} \tag{3.16}$$

$$\text{Gesamtbeschleunigung:} \quad \vec{a} = \vec{a_x} + \vec{a_y} = \frac{5}{7} \cdot g \cdot \begin{pmatrix} \sin \alpha_x \\ \sin \alpha_y \end{pmatrix} \tag{3.17}$$

3.1.3 Zeitaspekt

Für die Analyse des Wippe-Experiments in Bezug auf das Themengebiet der Echtzeitsysteme ist der Zeitaspekt ein wichtiges Kriterium. Um eine obere Grenze für die Reaktionszeit des Wippe-Systems zu bestimmen, welche beim Einhalten garantiert, dass die Kugel nicht von der Platte fällt und sich im Zentrum balanciert, wird folgendes Gedankenexperiment bezüglich einer Achse betrachtet. Analog kann das Experiment auf die Ebene abgebildet werden.

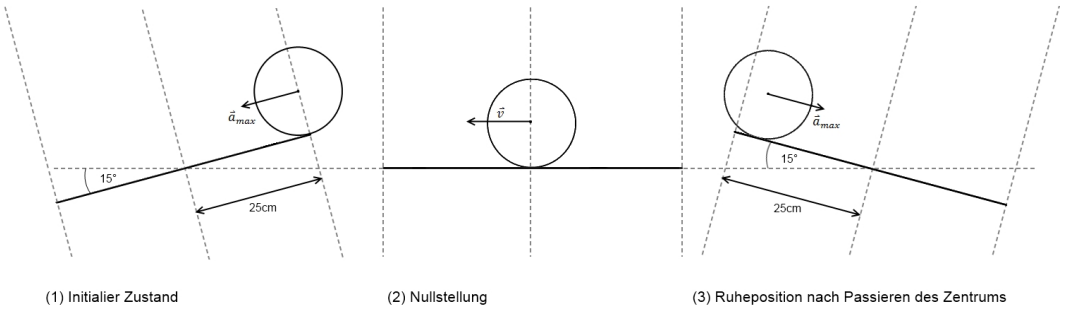

Abbildung 3.3: Beschreibung des Vorgangs zum Auspendeln der Kugel.

Die Kugel wird aus der Ruhe von dem Plattenrand ausgehend durch die maximale Auslenkung der Platte ($\alpha = 15°$) mit der resultierenden Beschleunigung a in Richtung des Mittelpunktes bewegt (1). Somit ergibt sich ein Abstand $s = 25cm$ zum Mittelpunkt der Platte. t ist die Zeit, welche die Kugel benötigt, um den Mittelpunkt zu erreichen. Innerhalb dieser Zeitspanne soll die Platte in die Nullstellung abgesenkt werden (2). Anschließend wird durch weiteres Auslenken über die Nullstellung hinaus das vollständige Abbremsen der Kugel garantiert. Durch eine

zusätzliche Bremseinwirkung des Luftwiderstands wird der Abstand von der neuen Ruheposition zum Mittelpunkt der Platte verringert (3). Die Wiederholung des Vorgangs hat das Auspendeln der Kugel im Zentrum zur Folge.
Durch die erworbenen Kenntnisse aus Abschnitt 3.1.1 und Abschnitt 3.1.2 lässt sich zunächst die Beschleunigung a_{max} der Kugel bestimmen.

$$a_{max} = \frac{5}{7} \cdot g \cdot \sin 15° \approx 1,81 \frac{m}{s^2} \tag{3.18}$$

Da die Strecke s bis zum Zentrum der Platte bekannt ist, kann mit Hilfe von Formel (3.5) die Zeit t bestimmt werden, welche die Kugel bis zum Mittelpunkt benötigt:

$$s = \frac{1}{2} \cdot a_{max} \cdot t^2 \;\Leftrightarrow\; t = \sqrt{\frac{2 \cdot s}{a_{max}}} = \sqrt{\frac{2 \cdot 0,25m}{1,81 \frac{m}{s^2}}} \approx 526ms \tag{3.19}$$

Die Verringerung der Beschleunigung durch das Auslenken der Platte hat bewirkt, dass der Geschwindigkeitszuwachs abnimmt und somit die Kugel mehr Zeit bis zum Erreichen des Zentrums benötigt. Durch die Annahme, dass auf die Kugel die konstante Beschleunigung a_{max} wirkt, werden die $526ms$ somit als eine minimale Zeitspanne betrachtet (schlechtester Fall).
Eine Erweiterung des Gedankenexperiments stellt das Einwerfen der Kugel am Plattenrand dar, wodurch sie eine Anfangsgeschwindigkeit v_0 erhält. Die resultierende Bewegung kann nun als Überlagerung einer gleichförmigen Bewegung, erzeugt durch die konstante Geschwindigkeit v_0, und einer durch a_{max} beschleunigten Bewegung betrachtet werden. Formel (3.5) muss demnach folgendermaßen erweitert werden:

$$s = \frac{1}{2} \cdot a_{max} \cdot t^2 + v_0 \cdot t \tag{3.20}$$

Durch das Auflösen nach t lässt sich für jede beliebige Anfangsgeschwindigkeit v_0 die entsprechende Zeit berechnen, welche die Kugel benötigt um das Plattenzentrum zu erreichen [1].

$$t = \frac{\sqrt{v_0^2 + 2 \cdot a_{max} \cdot s} - v_0}{a_{max}} \tag{3.21}$$

3.2 Echtzeitsysteme

Arbeitet ein System unter Einhaltung von vorgegebenen Zeitbedingungen, so spricht man von einem Betrieb in Echtzeit beziehungsweise einem Echtzeitsystem. Beispiele für ein solches System finden sich in der Robotik, der Fabrikautomation, der Medizintechnik, aber auch in Bereichen aus dem täglichen Leben wie in der Fahrzeugtechnik oder der Mobilkommunikation. Der Begriff Echtzeitbetrieb ist nach

[1] Da die Formel (3.20) eine quadratische Gleichung darstellt, existieren zwei Lösungen für t, wobei für eine der Lösungen $t \leq 0$ gilt und somit speziell für diesen Anwendungsfall nicht in Betracht kommt.

DIN44300 wie folgt definiert:

Definition 3.1 *Ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen [fN85].*

Eine Unterteilung von Echtzeitsystemen ergibt sich aus der Notwendigkeit, die geforderten Zeitbedingungen einzuhalten. Dabei wird zwischen harter und weicher Echtzeit differenziert. Arbeitet ein Echtzeitsystem unter weichen Zeitbedingungen, so genügt das Erfüllen dieser für einen überwiegenden Teil der Fälle und geringe Zeitüberschreitungen gelten als legitim. Ein anschauliches Beispiel findet sich in Computersystemen, welche auf die Eingabe eines Benutzers warten, um ein entsprechendes Ergebnis zu liefern. Dabei spielt die Bearbeitungs- und daraus resultierende Antwortzeit des Systems keine tragende Rolle, sondern kann als Komfort in Bezug auf die Schnelligkeit der Anwendung betrachtet werden. Leichte Zeitüberschreitungen eines vorbestimmten Mittelwerts haben lediglich eine längere Wartezeit für den Benutzer zur Folge. Die Korrektheit des Systems ist dadurch jedoch nicht gefährdet.
Harte Echtzeit zeichnet sich dadurch aus, dass die geforderten Zeitbedingungen eingehalten werden müssen. Schon eine geringe Überschreitung wird als Fehler eingestuft und das System arbeitet nicht korrekt. Ein Beispiel für ein Echtzeitsystem, welches unter harten Zeitbedingungen arbeitet, ist die Einparkhilfe bei Kraftfahrzeugen. Wird während des Parkvorgangs gegen eine Zeitbedingung verstoßen, wirkt sich dies fehlerhaft auf den Lenkvorgang aus und das Fahrzeug wird nicht korrekt eingeparkt. Ohne einen manuellen Eingriff können im schlimmsten Fall das Fahrzeug oder der Fahrer selbst Schaden davon tragen.
Gleichermaßen ordnet sich das Wippe-Experiment in die Gruppe der Echtzeitsysteme ein, welche unter harten Zeitbedingunen betrieben werden. Wird ein bestimmtes Zeitkriterium nicht erfüllt, fällt die Kugel von der Platte und der Versuch ist als fehlgeschlagen zu werten.
Aus der Sicht der Informatik lässt sich ein Echtzeitsystem in zwei Teilsysteme aufgliedern: in ein externes System, welches die anwendungsspezifischen Zeitbedingungen vorgibt; und in ein internes System, welches die vorgegebenen Zeitbedingungen einzuhalten hat. In der Praxis bildet ein technischer Vorgang typischerweise das externe System, welcher von einem internen Rechensystem erfasst, behandelt und gesteuert werden muss (vgl. [Zö08, S.3f]).
Bezogen auf das Wippe-Experiment verdeutlicht Abbildung 3.4 eine schematische Einordnung in die verschiedenen Komponenten eines Echtzeitsystems. Das technische beziehungsweise externe System stellt die mechanischen Bestandteile der Wippe dar. Zudem besitzt die Wippe ein Messsystem, welches je nach Betriebsart variiert. Im automatischen Betrieb wird die Webcam als Messsystem verwendet, im Fall des manuellen Betriebs wird es von der steuernden Person dargestellt. Das

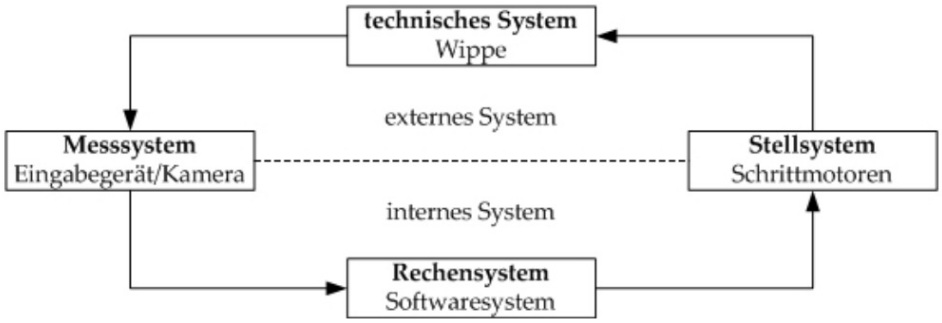

Abbildung 3.4: Grundmodell eines Echtzeitsystems bezogen auf das Wippe-Experiment.

Rechensystem oder interne System repräsentiert der Computer, welcher die Messdaten auswertet und weiter an das Stellsystem gibt. Dieses bildet das Treibermodul der Schrittmotoren einschließlich der Schrittmotoren selbst.
Durch die stark ausgeprägte Rolle der Zeit innerhalb von Echtzeitsystemen werden die wichtigen Anforderungen Rechtzeitigkeit, Vorhersagbarkeit und Determiniertheit erhoben. Dabei kann die Forderung der Determiniertheit als Voraussetzung für die Vorhersagbarkeit betrachtet werden. Wenn jeder Zustand exakt einem bekannten Folgezustand zugewiesen ist, so kann das Verhalten des Systems prognostiziert und damit auf Fehler reagiert werden.
Die Forderung der Rechtzeitigkeit bezieht sich auf das Einhalten der Zeitbedingung, so dass durch rechtzeitiges Handeln die Korrektheit des Systems gewährleistet wird. Eine formale Definition der Rechtzeitigkeit kann in Form einer bedingten Wahrscheinlichkeit ausgedrückt werden [Zö08], welche den Wert eins haben muss.

$$P(A \mid B) = 1 \tag{3.22}$$

Innerhalb der Formel bezeichnet B die Rahmenbedingungen, die als Voraussetzungen für den korrekten Betrieb gelten. Darunter ist zu verstehen, dass während der Ausführung des Wippe-Experiments keine technischen Ausfälle auftreten. A stellt die Zeitbedingung dar und wird durch [Zö08]

$$A \equiv r + \Delta e \leq d \tag{3.23}$$

beschrieben. Dabei sei r der Zeitpunkt, zu dem die Ausführung einer Aufgabe beginnen kann, Δe die Ausführungszeit und d der Zeitpunkt, zu dem die Aufgabe erfüllt sein muss. Die Forderung, dass die Ausführung der Aufgabe rechtzeitig beginnen und enden muss, wird als Rechtzeitigkeit bezeichnet.
Im Folgenden wird die Zeitbedingung für das Wippe-Experiment speziell für den automatischen Betrieb allgemein hergeleitet. Für das Abschätzen der Ausführungszeit Δe wird zunächst die Abfolge eines Arbeitszyklus betrachtet. Dieser setzt sich aus folgenden Teilschritten zusammen:

(A) Lesen des Kamerabildes und Berechnung der aktuellen Kugelposition.

(B) Berechnung der Auslenkung der Platte anhand der Kugelposition mittels eines Regelalgorithmus.

(C) Befehl zum Auslenken der Motoren entsprechend der Regelungsdaten an das Treibermodul senden.

Um daraus die Gesamtzeit des Arbeitszyklus abzuschätzen, muss lediglich jedem Teilschritt die entsprechende Ausführungszeit zugeordnet und anschließend die Summe gebildet werden. Bei der genaueren Betrachtung erweist sich jedoch die Einteilung nur anhand der Arbeitsschritte als unzureichend. Beispielsweise benötigt die Kamera eine gewisse Zeit, um ein Bild aufzunehmen und es an den Computer zu senden. Das Kamerabild hat folglich beim Eintreffen im Computerprogramm ein bestimmtes Alter. Es ergibt sich somit eine modifizierte Einteilung inklusive der Zuordnung der entsprechenden Ausführungszeiten:

1. Aufnahme des Kamerabildes ($\Delta e_K \widehat{=}$ Alter des Kamerabildes beim Eintreffen im Computer).
2. Lesen des Kamerabildes und Berechnung der aktuellen Kugelposition ($\Delta e_B \widehat{=}$ Rechenzeit für die Kugelerkennung, entsprechend Teilschritt (A)).
3. Berechnung der Auslenkung der Platte anhand der Kugelposition mittels eines Regelalgorithmus ($\Delta e_R \widehat{=}$ Rechenzeit des Regelalgorithmus für die Bestimmung der neuen Motorpositionen, entsprechend Teilschritt (B)).
4. Befehl zum Auslenken der Motoren entsprechend der Regelungsdaten an das Treibermodul senden ($\Delta e_S \widehat{=}$ Sendezeit des Befehls zum Auslenken der Motoren über die serielle Schnittstelle, entsprechend Teilschritt (C)).
5. Auslenkung der Motoren ($\Delta e_M \widehat{=}$ Zeit, die die Motoren benötigen um den gesetzten Winkel anzufahren).

Entsprechend der in Abbildung 3.4 vorgenommenen Einteilung in Messsystem, Rechensystem und Stellsystem, lassen sich drei verschiedene Aufgaben mit der Ausführungszeit Δe_i, $i \in \{0, 1, 2\}$ unterscheiden:

- Aufnahme des Kamerabildes (Messsystem):

$$\Delta e_0 = \Delta e_K \tag{3.24}$$

- Rechenzeit des internen Systems (Rechensystem):

$$\Delta e_1 = \Delta e_B + \Delta e_R + \Delta e_S \tag{3.25}$$

- Auslenkungszeit der Platte durch die Motoren (Stellsystem):

$$\Delta e_2 = \Delta e_M \tag{3.26}$$

Die Aufnahme des Bildes geschieht durch die Kamera und wird parallel periodisch ausgeführt. Die Periodizität ist dabei abhängig von der Bildfrequenz der Kamera. Durch eine diesbezüglich relative Betrachtung wird r_0 mit dem Wert Null initialisiert. Der Startzeitpunkt für die Berechnungen innerhalb des internen Systems r_1 und für die Auslenkung der Motoren r_2 ergibt sich jeweils aus dem Beenden der vorherigen Aufgabe. Somit gilt:

$$r_0 = 0\ , \quad r_1 = r_0 + \Delta e_0 \quad \text{und} \quad r_2 = r_1 + \Delta e_1 \tag{3.27}$$

Um die Rechtzeitigkeit für das Wippe-System festzulegen, müssen die entsprechenden Endzeitpunkte d_i, $i \in \{0, 1, 2\}$ für die Ausführung der Aufgaben definiert werden. Jedoch ergibt sich diesbezüglich nur die Bedingung, dass die Platte durch die Motoren rechtzeitig ausgelenkt sein muss, damit die Kugel gehalten und im Zentrum balanciert werden kann. Demnach ist nur d_2 relevant und es gilt:

$$r_2 + \Delta e_2 \leq d_2 \quad \Leftrightarrow \quad r_1 + \Delta e_1 + \Delta e_2 \leq d_2 \quad \Leftrightarrow \quad \Delta e_0 + \Delta e_1 + \Delta e_2 \leq d_2 \tag{3.28}$$

Die allgemeine Zeitbedingung, welche für den korrekten Betrieb des Wippe-Systems eingehalten werden muss, lautet demnach:

$$A \equiv (\Delta e_K) + (\Delta e_B + \Delta e_R + \Delta e_S) + (\Delta e_M) \leq d_2 \tag{3.29}$$

3.3 Softwaretechnische Grundlagen

Im Weiteren werden für die Arbeit wichtige Grundlagen der Softwaretechnik besprochen. Dabei geht es zunächst um Prinzipien der Softwareentwicklung und die daraus resultierenden Ansprüche an ein Softwaresystem. Nachfolgend werden zwei Entwurfsmuster erläutert, welche für den späteren Softwareentwurf eine tragende Rolle spielen.

3.3.1 Prinzipien der Softwaretechnik

Innerhalb des Gebiets der Softwaretechnik kann die Qualität eines Softwaresystems anhand bestimmter Kriterien bestimmt werden. Die folgende Auflistung zeigt eine Auswahl der wichtigsten Qualitätsmerkmale (siehe [Bal00]).

- **Erweiterbarkeit** - Kriterium für den benötigten Aufwand, um die Funktionalität des Systems anhand neuer Anforderungen zu erweitern.
- **Verständlichkeit** - Kriterium für das Maß der Verständlichkeit bezüglich der Architektur und der konkreten Umsetzung des Systems.

- **Wartbarkeit** - Kriterium für den Aufwand, welcher benötigt wird, um ein System an sich ändernde Umstände anzupassen.
- **Dokumentation** - Kriterium für die Qualität der Dokumentation des Systems.

Um die daraus resultierenden Anforderungen an ein Softwaresystem erfüllen zu können, bietet sich die Orientierung an innerhalb der Softwaretechnik definierten Prinzipien an. Die im Verlauf der Arbeit relevanten Prinzipien werden im Folgenden erläutert:

Prinzip der Abstraktion Unter dem Prinzip der Abstraktion versteht man eine Verallgemeinerung durch das Vernachlässigen spezifischer Eigenschaften. Beispielsweise wird im Fall einer funktionalen Abstraktion durch die Definition von Schnittstellen eine größere Allgemeingültigkeit erzielt. Folglich wird die Qualität der Erweiterbarkeit und Wartbarkeit erhöht, da die Implementierung einer Schnittstelle ohne tragende Folgen für andere Komponenten ausgetauscht werden kann. Zusätzlich wird die Verständlichkeit des Systems erleichtert (siehe [Bal00, S.37f.]).

Prinzip der Zerlegung und Strukturierung Durch eine sinnvolle Zerlegung und der damit einhergehenden Strukturierung eines Systems kann die Komplexität des Ganzen immens gesenkt werden. Hierbei lassen sich die Bestandteile einzeln betrachten und parallel entwickeln. Dies nimmt einen positiven Einfluss auf die Fehleranfälligkeit und die Verständlichkeit des Softwaresystems.
Eine sinnvolle Zerlegung basiert auf den Gliederungsprinzipien der **Hierarchisierung** und **Modularisierung**. Die Hierarchisierung umfasst das Anordnen von einzelnen Elementen anhand einer Rangordnung. Dabei bilden Elemente der gleichen Rangordnung eine Hierarchiestufe. Unter Modularisierung versteht man das Unterteilen in funktional abgeschlossene Bausteine, welche nur über eine definierte Schnittstelle mit ihrer Umwelt kommunizieren können. Einzelne Module bieten oftmals die Möglichkeit zur Wiederverwendung (siehe [Bal00, S.37f.]).

3.3.2 Entwurfsmuster

Das Erstellen und Entwerfen eines Softwaresystems anhand der in Abschnitt 3.3.1 vorgestellten Prinzipien kann ein komplizierter und mühseliger Prozess sein. Hilfreich ist dabei der Einsatz von Entwurfsmustern (engl. design patterns). Unter einem Entwurfsmuster versteht man einen vordefinierten Lösungsvorschlag für wiederkehrende Entwurfsprobleme innerhalb der Softwarearchitektur (siehe [GHJV94]). Es werden zwei Entwurfsmuster vorgestellt, welche im späteren Entwurf der Regelsoftware eine tragende Rolle spielen.

Beobachter Das Entwurfsmuster Beobachter (engl. Observer) findet dann Anwendung, wenn der Zustand eines zentralen Objektes für eine beliebige Menge anderer Objekte von Belang ist. Ein veranschaulichendes Beispiel zeigt Abbildung

3.5. Dabei wird ein spezifischer Datensatz auf verschiedene Art und Weise grafisch dargestellt. Die Veränderung des Datensatzes hat eine Aktualisierung der abhängigen Grafikkomponenten zur Folge. Üblicherweise wird das Beobachter-Muster folgendermaßen definiert:

Definition 3.2 *Das Beobachter-Muster definiert eine Eins-zu-viele-Abhängigkeit zwischen Objekten in der Art, dass alle abhängigen Objekte benachrichtigt werden, wenn sich der Zustand des einen Objekts ändert (siehe [Bat04, S.49]).*

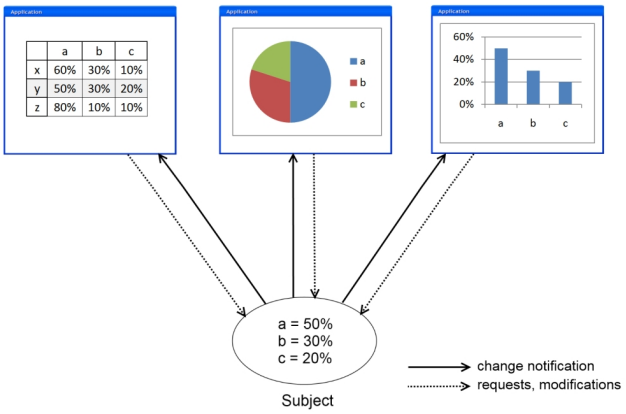

Abbildung 3.5: Anwendungsbeispiel des Entwurfsmusters Beobachter.

Das Klassendiagramm in Abbildung 3.6 zeigt eine Möglichkeit zur praktischen Umsetzung des Beobachter-Musters. Das zentrale Objekt wird durch die Schnitt-

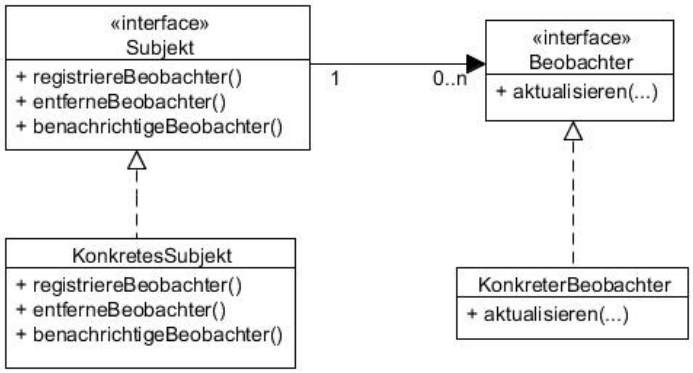

Abbildung 3.6: UML-Klassendiagramm des Entwurfsmusters Beobachter.

stelle Subjekt dargestellt. Diese bietet durch die Methoden registriereBeobachter() und entferneBeobachter() die Möglichkeit, eine Menge beliebig vieler Beobachter zu verwalten. Sofern sich der Zustand eines Subjekts ändert, kann die Methode benachrichtigeBeobachter() aufgerufen werden, um jeden registrierten Beobachter

über die Veränderung zu benachrichtigen. Dies geschieht in Form eines Aufrufs der Methode aktualisieren(), welche durch die Schnittstelle Beobachter definiert wird. Zusätzlich können die aktualisierten Daten in Form von Parametern übergeben und anschließend innerhalb des Methoden-Rumpfs verwertet werden.

Singleton Soll verhindert werden, dass von einer Klasse mehrere Instanzen gebildet werden können, so erweist sich das Entwurfsmuster Singleton (dt. Einzelstück) als hilfreich. Ein mögliches Anwendungsbeispiel ist das Protokollieren bestimmter Informationen während des Ausführens eines Programmes. Dabei existiert ein zentrales Protokoll-Objekt, welches die Informationen beispielsweise über eine Konsole ausgibt oder in eine Datei schreibt.

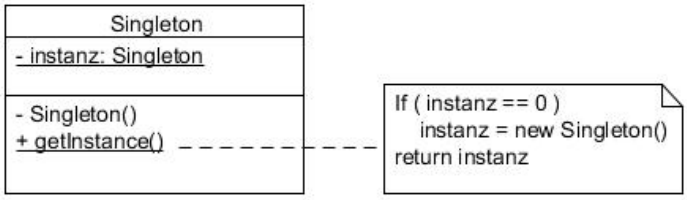

Abbildung 3.7: UML-Klassendiagramm des Entwurfsmusters Singleton.

Eine Möglichkeit zur praktischen Umsetzung des Singleton-Musters wird in Abbildung 3.7 aufgezeigt. Auffällig ist der **private Konstruktor** der Klasse Singleton. Dieser kann somit nur innerhalb der Klasse selbst aufgerufen werden, wodurch eine Instanziierung von außerhalb unmöglich ist. Zusätzlich besitzt die Klasse Singleton eine private statische Instanz von sich selbst, auf welche durch einen globalen Zugriffspunkt, gegeben durch die ebenfalls statische Methode getInstanz(), zugegriffen werden kann. Ein üblicher Teil der Implementierung ist die im Klassendiagramm als Notiz vermerkte Kontrollabfrage. Dabei wird überprüft, ob die private Singleton-Instanz bereits erzeugt wurde. Ist dies nicht der Fall, wird sie mit Hilfe des privaten Konstruktors initialisiert.

Kapitel 4

Überarbeitung des Wippe-Systems

In diesem Kapitel wird das vorhandene Wippe-Experiment einer umfassenden Überarbeitung unterzogen. Hierbei ist eine Unterscheidung zwischen der Überarbeitung des technischen Systems einschließlich des mechanischen Aufbaus und der Überarbeitung des Softwaresystems zur Steuerung der Wippe zu treffen.

4.1 Überarbeitung der Hardwarekomponenten

Das Wippe-Experiment hat inzwischen ein Alter von über zehn Jahren erreicht. Bedingt durch Verschleiß müssen die Hardwarekomponenten überprüft und gegebenenfalls erneuert werden. Dazu zählen der Computer, auf welchem die Software zur Steuerung der Wippe ausgeführt wird, und die Schrittmotoren, welche die Platte auslenken. Des Weiteren ist auch dem gesteigerten Anspruch an Mobilität Rechnung zu tragen. Die erforderliche Anpassung wird im Folgenden thematisiert. Da die Wippe in Zukunft in der Lehre eingesetzt und entsprechend an verschiedenen Orten (z.B. auf Messen) vorgeführt werden soll, ist dieser Aspekt zu berücksichtigen.

Austausch der Schrittmotoren Der Austausch der Schrittmotoren erfordert eine erneute Kalibrierung des Treibermoduls, so dass beispielsweise die korrekte Stromstärke für die neuen Motoren entsprechend eingestellt wird. Auch die Anfahrgeschwindigkeit einer Position sowie die Art der Beschleunigung und des Bremsens lassen sich konfigurieren. Die passende Einstellung lässt sich nur durch Probieren herausfinden. Vor allem muss dabei beachtet werden, dass bei zu hohen Werten die Mechanik überfordert wird. Mit der in Tabelle 4.1 (Erläuterungen aus [Tri11, S.27ff]) dargestellten Konfiguration kann jedoch der korrekte Betrieb der Wippe ermöglicht werden.

Austausch des alten Computers Damit ein einzelner Computer die Berechnungen für das Regeln des Wippe-Experiments durchführen kann, muss er eine Mindestleistung bereitstellen. Bei dem bisher eingesetzten Computer handelt es sich um

einen Desktop Rechner mit einem Intel Pentium IV Prozessor (3,2 GHz, 2 GB Arbeitsspeicher), auf welchem das Betriebssystem Windows XP installiert ist. Dieses Modell wird durch ein Notebook mit einem Intel Core i5 Prozessor (4 * 2,53GHz, 8GB Arbeitsspeicher) und installiertem Betriebssystem Windows 7 ersetzt, um aufkommenden Performanzproblemen vorzubeugen. Neben der Anforderung an die Leistung des Computers, wird durch den Einsatz eines mobilen Gerätes gleichzeitig der Aspekt der Mobilität erfüllt. Die Umrüstung hat jedoch zwingend den Austausch der Kamera zur Folge, da die bisher für den Empfang der Bilder verwendete TV-Karte von einem Notebook nicht verwendet werden kann. Eine praktikable Lösung stellt eine Webcam der Firma Logitech dar, welche mit einer Bildfrequenz von 30fps (frames per second) über eine USB-Verbindung Bilder liefert. Durch die veränderte Übertragungsart muss dementsprechend eine Änderung an dem Teil der Software vorgenommen werden, welcher für den Empfang der Bilder zuständig ist.

Tabelle 4.1: Einstellungen des Treibermoduls zum Ansteuern der neuen Schrittmotoren.

Befehl	**Erklärung**
SAP 4 <motorId> 2000	Legt die maximale Geschwindigkeit für den angegebenen Motor fest, welche beim Anfahren einer Position erreicht werden kann. $2000 \mathrel{\widehat{=}} 1,22 \cdot 10^5 \frac{\mu step}{s}$
SAP 5 <motorId> 200	Legt die maximale Beschleunigung für den angegebenen Motor fest, welche beim Anfahren einer Position erreicht werden kann. $200 \mathrel{\widehat{=}} 1,86 \cdot 10^5 \frac{\mu step}{s^2}$
SAP 6 <motorId> 600	Legt die Stromstärke für den angegebenen Motor fest: $600 \mathrel{\widehat{=}} 600mA$
SAP 138 <motorId> 1	Bestimmt die Art des Bremsens für das Anfahren einer Position (engl. ramp mode), welche bei dem angegebenen Motor angewendet wird. Der Wert 1 steht für den Soft-Modus, wodurch die Geschwindigkeit beim Bremsvorgang exponentiell abnimmt.
SAP 154 <motorId> 2	Bestimmt den Exponenten des Normierungsfaktors für das Generieren von Schritten (engl. pulse divisor). Folgende Formel zur Berechnung der Mikroschritt-Frequenz verdeutlicht die Bedeutung: $f = \frac{16 \cdot 10^6 Hz \cdot v_{max}}{2^{pulse\_div+16}}$

4.2 Überarbeitung des Softwaresystems

Für die Überarbeitung des Softwaresystems wird zunächst festgelegt, welche Funktionen das Softwaresystem dem Nutzer zur Verfügung stellen soll und wie diese schließlich umgesetzt werden. Dazu werden zunächst Anforderungen erhoben, welche in *funktional* (FA) und *nicht-funktional* (NFA) zu unterteilen sind. Es ist aufgrund des bereits geplanten Vorhabens, die Wippe für Unterricht und Lehre an Schulen und Universitäten einzusetzen, durchaus sinnvoll bei der Formulierung der Systemanforderungen auf dieses Vorhaben Rücksicht zu nehmen. Anhand der aufgestellten Kriterien wird einhergehend überprüft, inwieweit diese von der bisher verwendete Software unterstützt werden.

4.2.1 Anforderungen

Es werden zunächst die funktionalen Anforderungen definiert. Diese geben Aufschluss über die Funktionalität, welche von der Software geboten werden soll.

FA-1. **Steuerung** - Die Wippe ist mit Hilfe der Software steuerbar.

- FA-1.1. **Starten** - Der Betrieb der Wippe kann gestartet werden.
- FA-1.2. **Stoppen** - Der Betrieb der Wippe kann gestoppt werden.

FA-2. **Konfiguration** - Die Software bietet die Möglichkeit zur Konfiguration der Betriebsart des Wippe-Experiments.

- FA-2.1. **Automatisch** - Die Wippe kann automatisch betrieben werden, so dass die Kugel lediglich durch Berechnungen der Software auf der Platte balanciert wird.
- FA-2.2. **Manuell** - Die Wippe kann manuell betrieben werden, so dass die Platte mit Hilfe der Wii-Remote gesteuert werden kann.
- FA-2.3. **Kombination** - Die Wippe kann kombiniert betrieben werden, so dass eine Achse automatisch, die Verbleibende jedoch manuell betrieben wird.

FA-3. **Kalibrierung** - Die Software bietet die Möglichkeit zur Kalibrierung des Wippe-Experiments.

- FA-3.1. **Plattenauslenkung** - Die Platte kann vor dem Start des Versuchs manuell in die Nullstellung ausgelenkt werden.
- FA-3.2. **Wii-Remote** - Die Nullstellung der Wii-Remote kann kalibriert werden.
- FA-3.3. **Kamerabild** - Das Kamerabild kann so angepasst werden, dass lediglich der Ausschnitt, auf welchem die Platte zu sehen ist, für die Bildverarbeitung verwendet wird.

FA-4. **Laufzeitinformationen** - Die Software ist in der Lage, während des Betriebs der Wippe Laufzeitinformationen zur Verfügung zu stellen.

- FA-4.1. **Kugelinformationen** - Während des Betriebs der Wippe werden physikalische Informationen über die Kugel bereitgestellt (z.B. Kugelposition, Kugelgeschwindigkeit, Kugelbeschleunigung, etc.).
- FA-4.2. **Latenzmessung** - Während des Betriebs der Wippe werden Informationen über Latenzzeiten der für den Betrieb wichtigen Softwarekomponenten bereitgestellt.

Die bisher verwendete Software unterstützt einen Teil der erhobenen funktionalen Anforderungen. Die Steuerung und somit auch das Starten und Stoppen ist bereits möglich (vgl. FA-1). Auch die Möglichkeit zur Kalibrierung von Platte und Wii-Remote wird bereits angeboten (vgl. FA-3.1, FA-3.2). Lediglich die Funktion zum Anpassen des Kamerabildes wurde bisher nicht realisiert (vgl. FA-3.3).
Die Benutzeroberfläche stellt zum einen das von der Kamera aufgenommene Bild, aber auch das mittels Bildverarbeitung bearbeitete Bild dar. Daran lässt sich zwar die Kugelposition erkennen, jedoch fehlen Funktionen zum Anzeigen der Kugelkoordinaten beziehungsweise der weiteren Laufzeitinformationen, wie beispielsweise die Latenzzeiten der verschiedenen Berechnungsschritte (vgl. FA-4). Diese Informationen sind wichtig und werden für eine Analyse der Wippe in Bezug auf das Themengebiet Echtzeitsysteme benötigt.
Als Betriebsart können sowohl der automatische (vgl. FA-2.1) als auch der manuelle Modus (vgl.FA-2.2) ausgewählt werden, jedoch ist die Funktion zum automatischen Betrieb der Wippe noch nicht korrekt umgesetzt worden. Folglich wurde eine Kombination beider Modi beim Entwurf der bisher verwendeten Software nicht in Erwägung gezogen (vgl. FA-2.3).

Nachdem die angestrebte Funktionalität der Software definiert ist, werden in einem weiteren Schritt die nicht-funktionalen Anforderungen erhoben, welche vor allem softwaretechnische Aspekte betrachten und somit Hinweise zur Architektur und Realisierung andeuten.
Die nachfolgende Auflistung stellt die an das System erhobenen nicht-funktionalen Anforderungen dar:

NFA-1. **Bedienbarkeit** - Die Software soll eine Benutzerschnittstelle anbieten, mit deren Hilfe eine intuitive Bedienung der Wippe möglich ist.

NFA-2. **Verständlichkeit** - Der Aufbau und die Struktur des Systems soll sich an der zu lösenden Aufgabe orientieren und dementsprechend modularisiert und hierarchisiert sein, um somit Nachvollziehbarkeit und Verständlichkeit zu gewährleisten.

NFA-3. **Leistung und Effizienz** - Die Software muss bezüglich Leistung und Effizienz den Ansprüchen an die zu bewältigende Echtzeitaufgabe genügen.

NFA-4. **Analysierbarkeit** - Eine Möglichkeit zur Analysierbarkeit soll geboten werden, so dass beispielsweise Latenzzeiten verschiedener Softwarekomponenten leicht aufgezeichnet werden können.

NFA-5. **Änderbarkeit** - Die Softwarearchitektur soll so konzipiert werden, dass ein Austausch einer Softwarekomponente möglich ist. Beispielsweise soll der zu entwickelnde Regelalgorithmus eine austauschbare Komponente darstellen, so dass innerhalb des Einsatzes in der Lehre auch andere Implementierungen getestet werden können.

NFA-6. **Erweiterbarkeit** - Das System soll in begrenztem Maße erweiterbar sein, so dass eine nach einem bestimmten Muster aufgebaute Komponente leicht integriert werden kann. Zukünftig ist dadurch beispielsweise die Auswahl zwischen mehreren Regelalgorithmen möglich.

Wie in Abschnitt 2.3.3 bereits vorgestellt, besitzt die aktuelle Software eine Benutzerschnittstelle zur Steuerung der Wippe (vgl. NFA-1). Bisher sind die vom Nutzer auswählbaren Funktionen jedoch alle auf einem einzigen Panel platziert. Falls weitere Funktionen integriert werden sollen, wird der Platz für entsprechende Steuerelemente nicht ausreichen. In diesem Fall ist die Möglichkeit zur Erweiterung nicht gegeben (vgl. NFA-6).
Die geforderte Verständlichkeit des Systems ist bisher unzureichend (vgl. NFA-2). Eine Struktur der Software ist lediglich anhand eines Klassendiagramms erkennbar, wie es beispielsweise in Abbildung 2.3 dargestellt wird. Eine Einteilung auf einem höheren Abstraktionsniveau ist bei dem bisherigen Aufbau der Software nicht vorhanden. Folglich wurde eine Strukturierung in austauschbare Softwarekomponenten nicht vorgenommen (vgl. NFA-5). Beispielhaft belegt wird diese Tatsache bei der Betrachtung des Aktivitätsdiagramms in Abbildung 2.5. Innerhalb der Methode doExecute() der Klasse MainController wird das Kamerabild auf die Benutzeroberfläche gezeichnet. Somit verwendet die Klasse MainController, in welcher ein wichtiger Teil der Anwendungslogik umgesetzt ist, Bestandteile der Klasse EZVideoCaptureDlg, welche die Benutzerschnittstelle realisiert. Es ergibt sich eine gegenseitige Abhängigkeit, da der MainController gleichzeitig von der Klasse EZVideoCaptureDlg initialisiert und gestartet wird (vgl. Abbildung 2.3). Üblicherweise sollte eine einseitige Abhängigkeit garantiert werden und speziell in diesem Fall lediglich die Klasse EZVideoCaptureDlg abhängig von der Klasse MainController sein, damit die Benutzerschnittstelle eine gekapselte Komponente darstellt, welche bei Bedarf ausgetauscht werden kann.
Möglichkeiten zu Analysierbarkeit des Systems werden nur bedingt angeboten. Es existiert lediglich eine Helferklasse, mit deren Hilfe Zeichenketten in eine Datei geschrieben werden können. Diese Art des Aufzeichnens von Daten zur Analyse erscheint jedoch unzureichend (vgl. NFA-4). Entsprechend sind Aussagen bezüglich Leistung und Effizienz des Systems nur begrenzt möglich (vgl. NFA-3).
Zusammenfassend zeigt die Überprüfung anhand der definierten funktionalen und

nicht-funktionalen Anforderungen des bisher verwendeten Softwaresystems, dass diese nur teilweise beziehungsweise nicht erfüllt sind. Da vor allem das grundlegende Softwaredesign erhebliche Mängel aufweist, erscheint die Entwicklung eines neuen Systems effizienter, als eine Überarbeitung des alten Systems.

4.2.2 Modularisierung des Softwaresystems

Um die Anforderungen bezüglich Austauschbarkeit und Verständlichkeit zu erfüllen, wird das System in Softwarekomponenten unterteilt und hierarchisch strukturiert. Dabei soll sich die Einteilung vor allem an der zu lösenden Problemstellung orientieren.
Das Gesamtsystem besteht aus einem Softwarekern, welcher die Anwendungslogik zur Steuerung der Wippe enthält, und einer Benutzerschnittstelle, welche dem Nutzer die Bedienung der Wippe ermöglicht. Entsprechend werden zwei Softwarekomponenten gemäß Abbildung 4.1 unterschieden. Damit die Unabhängigkeit des Softwarekerns gegenüber anderen Komponenten gewährleistet ist, verwendet die Benutzerschnittstelle das bereitgestellte Interface des Softwarekerns mit Funktionen zur Steuerung der Wippe. Sofern zukünftig eine neue Benutzerschnittstelle entworfen werden soll, kann der Austausch mit wenig Aufwand und ohne Einfluss auf den Softwarekern realisiert werden.

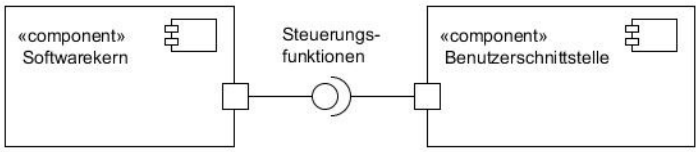

Abbildung 4.1: UML-Komponentendiagramm zur Darstellung der Einteilung des Gesamtsystems.

Die Benutzerschnittstelle ist nach dem Kompositum-Pattern aufgebaut. Eine weitere Einteilung in Softwarekomponenten über die bereits durch das Entwurfsmuster Kompositum vorgegebene Struktur, erscheint bei dem einfachen Aufbau der grafischen Benutzerschnittstelle des Wippe-Experiments nicht notwendig [1]. Die weitere Unterteilung des Softwarekerns hingegen erfordert eine detaillierte Beschreibung. Die Einteilung orientiert sich an der Echtzeitaufgabe, welche das Softwaresystems verwirklicht und insbesondere an den in Abschnitt 3.2 vorgestellten Teilaufgaben. Innerhalb von Abbildung 4.2 wird ein Überblick über die einzelnen Softwarekomponenten des Softwarekerns gegeben.

[1] Da diese Vorgehensweise bereits als Standard angesehen werden kann, wird auf das Entwurfsmuster Kompositum nicht explizit eingegangen. Weitere Informationen finden sich in entsprechender Fachliteratur, z.B. [GHJV94, S.163ff.].

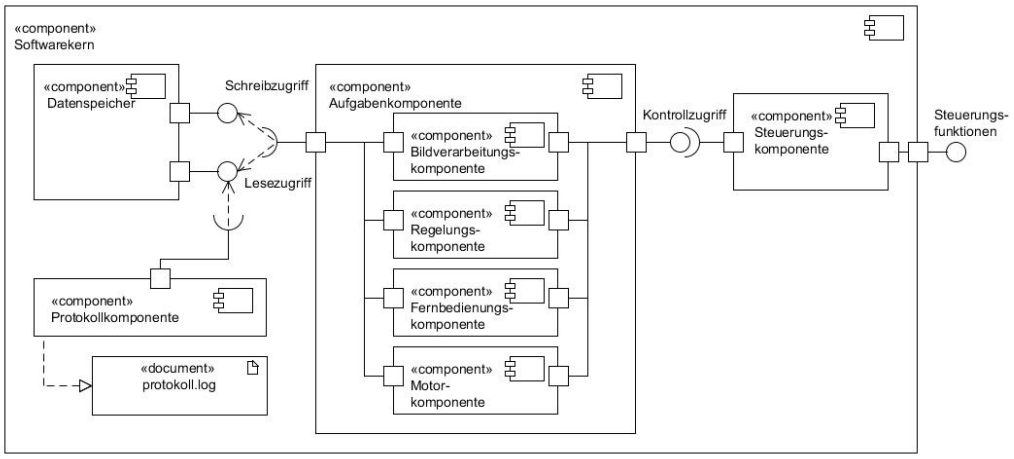

Abbildung 4.2: UML-Komponentendiagramm zur Beschreibung der Einteilung des Softwarekerns in Softwarekomponenten.

Steuerungskomponente Die Steuerungskomponente verkörpert zwei verschiedene Rollen. Zum einen kontrolliert sie die Aufgabenkomponente mit Hilfe eines bereitgestellten Interfaces. Dabei gewährleistet sie, dass bei der Abarbeitung der einzelnen Teilaufgaben die Reihenfolge eingehalten wird. Des Weiteren stellt die Steuerungskomponente selbst ein Interface mit Funktionen zur Steuerung der Wippe zur Verfügung. Beispielsweise existiert eine Funktion zur Konfiguration des Betriebsmodus der Wippe. Eine weitere dient dazu, die Wippe zu starten beziehungsweise zu stoppen. Das Interface ist über den Softwarekern ansprechbar und wird, wie in Abbildung 4.1 veranschaulicht, von der Benutzerschnittstelle verwendet.

Aufgabenkomponente Die Aufgabenkomponente verkörpert die einzelnen Teilaufgaben, welche beim Betrieb der Wippe anfallen. Dazu zählen die Bildverarbeitungskomponente, die Regelungskomponente, die Fernbedienungskomponente und die Motorkomponente. Jede einzelne Komponente stellt ein Interface zur Verfügung, welches Kontrollfunktionen anbietet. Mit Hilfe dieser Funktionen lässt sich in Erfahrung bringen, welche Komponente während des Betriebs der Wippe aktiv ist. Die Teilkomponenten der Aufgabenkomponente verwenden außerdem die vom Datenspeicher bereitgestellten Funktionen, um relevante Daten in den Speicher zu schreiben oder aus dem Speicher zu lesen. So schreibt die Bildverarbeitungskomponente die aktuelle Kugelposition in den Speicher, welche von der Regelungskomponente gelesen wird, um entsprechend neue Motorpositionen zu berechnen, damit diese wiederum für die Motorkomponente in den Speicher geschrieben werden.

Datenspeicher Der Datenspeicher stellt einen statischen Speicher dar, welcher ein Interface mit Funktionen zum Schreiben und Lesen anbietet. Relevante Daten werden während des Betriebs der Wippe zwischen den verschiedenen Aufgaben-

komponenten mit Hilfe des Datenspeichers ausgetauscht.

Protokollkomponente Die Protokollkomponente verwendet die Lesefunktion des Datenspeichers. Dadurch können relevante Daten protokolliert und anschließend ausgewertet werden. Als Beispiel wird innerhalb des Komponentendiagramms das Protokoll in Form einer Datei ausgegeben. Zusätzlich sind weitere Möglichkeiten denkbar, beispielsweise die Ausgabe über eine Anzeigekomponente der Benutzerschnittstelle, welche jedoch innerhalb des Komponentendiagramms nicht dargestellt werden.

Nachfolgend werden die nicht-funktionalen Anforderungen aufgezählt, welche durch den modularisierten Aufbau des Softwaresystems bereits erfüllt sind:

NFA-2 (Verständlichkeit) - Die Einteilung des Softwaresystems orientiert sich deutlich an der zu lösenden Echzeitaufgabe und gewährleistet ein hohes Maß an Verständlichkeit. Die Steuerungskomponente ist vergleichbar mit dem Scheduler innerhalb eines Echtzeitsystems, welcher für die Organisation und Synchronisierung der Aufgabenkomponente verantwortlich ist. Die Gliederung der Aufgabenkomponente richtet sich nach den Teilaufgaben, welche während des Betriebs der Wippe anfallen.

NFA-4 (Analysierbarkeit) - Durch die Protokollkomponente lassen sich relevante Daten protokollieren und anschließend auswerten. Zusätzlich bietet das Interface zum lesenden Zugriff auf den Datenspeicher die Möglichkeit, Daten der verschiedenen Teilaufgaben aus dem Speicher zu lesen und beispielsweise über die Benutzerschnittstelle zu visualisieren. Somit sind die Voraussetzungen zur Analysierbarkeit der Wippe vorhanden.

NFA-5 (Änderbarkeit) - Durch die Kapselung der einzelnen Teilaufgaben innerhalb der Aufgabenkomponente in weitere Komponenten, können diese ohne Auswirkungen auf das Gesamtsystem ausgetauscht werden. So muss eine neue Regelungskomponente lediglich das Interface mit Kontrollfunktionen zur Verfügung stellen und mit Hilfe der Daten der Bildverarbeitungskomponente die neu berechneten Motorpositionen in den Speicher schreiben. Des Weiteren ist prinzipiell der Austausch jeder Softwarekomponente innerhalb des Systems möglich. Die Anforderung in Bezug auf die Änderbarkeit des Systems ist somit erfüllt.

NFA-6 (Erweiterbarkeit) - Alle Teilkomponenten der Aufgabenkomponente sind nach einem vorgegebenen Muster aufgebaut, wodurch die Möglichkeit zur Erweiterung geboten wird. Somit muss bei dem Erstellen einer neuen Regelungskomponente nicht zwingend ein Austausch stattfinden, sondern es besteht die Möglichkeit diese in das System zu integrieren.

4.2.3 Der Softwarekern

Nachdem die grundlegende Struktur des Softwaresystems erarbeitet ist, wird nun auf die Implementierung der einzelnen Softwarekomponenten des Softwarekerns eingegangen.

Protokollkomponente Das Klassendiagramm der Protokollkomponente wird anhand Abbildung 4.3 dargestellt. Die Klasse CLogger verwendet das Entwurfsmuster Singleton. Jedoch muss der Methode GetInstance() der generische Parameter T übergeben werden, welcher die Klasse spezifiziert, für die eine Instanz der Klasse CLogger erstellt werden soll. Die Map mpLogger ordnet den Klassennamen die entsprechende Instanz zu. Mit Hilfe der Methode RegisterWriter() kann sich die Instanz einer Klasse, welche von der abstrakten Klasse CLogWriter abgeleitet ist, registrieren. Die zu implementierende Methode WriteLog() behandelt alle Nachrichten, welche an die Methoden Info(), Warning() und Error() übergeben werden. Innerhalb des Klassendiagramms wird ein Beispiel durch die Klasse CFileWriter dargestellt, welches ankommende Nachrichten in Form einer Datei ausgibt.

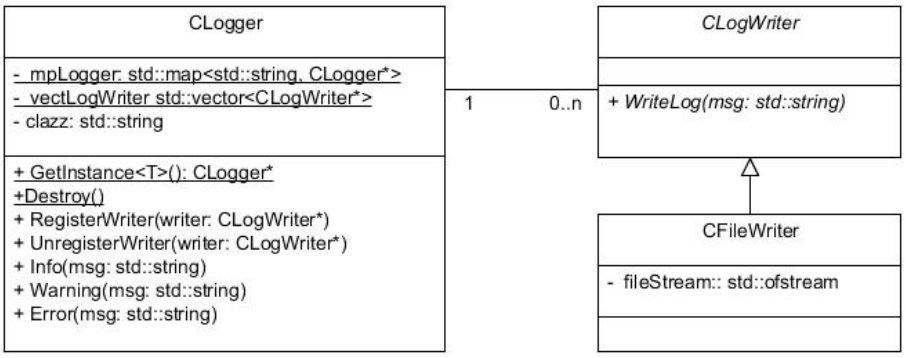

Abbildung 4.3: UML-Klassendiagramm der Protokollkomponente (weitere Dokumentation im Anhang B.1).

Datenspeicher Das in Abbildung 4.4 aufgeführte Klassendiagramm zeigt die Klassen der Softwarekomponente Datenspeicher. Den eigentlichen statischen Speicher stellt die Klasse CDataCache dar. Um zu gewährleisten, dass nur eine Instanz dieser Klasse existiert, wird das Entwurfsmuster Singleton angewendet. Die im Speicher verwalteten Objekte sind Instanzen der Klasse CControlledData, welche als Wert innerhalb einer Map abgelegt werden. Den Schlüssel stellt die Variable id der Klasse CControlledData dar.
Der Schreibzugriff wird mit Hilfe der Methode UpdateData() realisiert. Lesender Zugriff bietet der Speicher anhand der Methode GetData() an. In diesem Fall muss jedoch die Klasse des Rückgabeobjektes durch den generischen Parameter T angegeben werden. T stellt dabei eine von der Klasse CControlledData abgeleitete Klas-

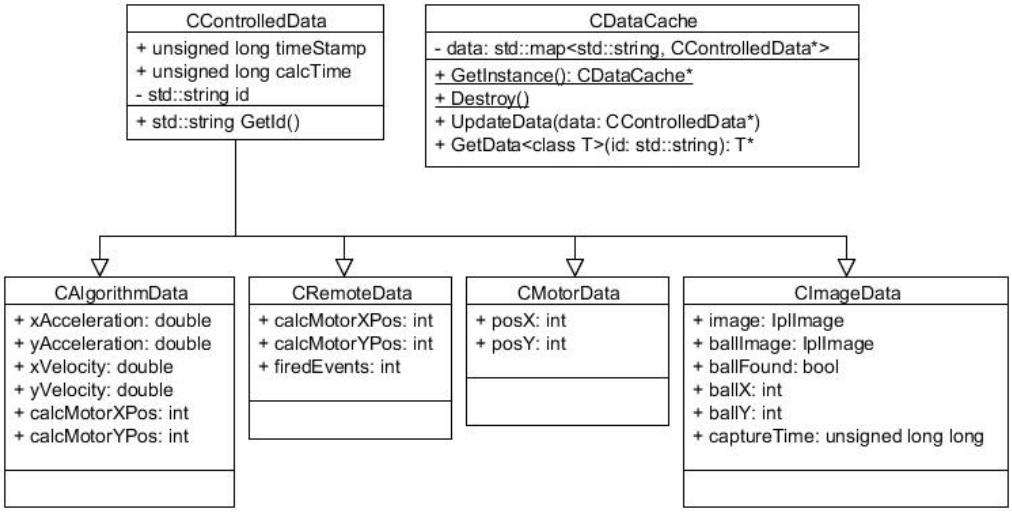

Abbildung 4.4: UML-Klassendiagramm des Datenspeichers (weitere Dokumentation im Anhang B.2).

se dar. Insgesamt werden vier verschiedene Kindklassen unterschieden, welche je einer Teilaufgabe innerhalb des Arbeitszyklus zugeordnet sind.

- **CAlgorithmData** - Enthält die vom Regelalgorithmus berechneten Daten, wie z.B. die aktuelle Geschwindigkeit der Kugel in x- beziehungsweise y-Richtung (xVelocity, yVelocity).
- **CRemoteData** - Enthält die anhand einer Fernbedienung (z.B. Wii-Remote) berechneten Motorpositionen.
- **CMotorData** - Enthält die zuletzt gesetzten Motorpositionen der Schrittmotoren.
- **CImageData** - Enthält Informationen, welche die Bildverarbeitung liefert, u.a. die aktuelle Kugelposition und das Kamerabild.

Aufgabenkomponente Das in Abbildung 4.5 aufgeführte Klassendiagramm veranschaulicht die Klassen, welche die Aufgabenkomponente bilden. Jede von den Klassen CThread und CObservable abgeleitete Klasse bildet eine Kontrolleinheit, welche der entsprechenden Teilkomponente der Aufgabenkomponente zugeordnet wird. So gehört beispielsweise die Klasse CImageProcessUnit zu der Bildverarbeitungskomponente. Jede Kontrolleinheit besitzt außerdem ein Attribut des entsprechenden Datentyps, so besitzt zum Beispiel die Klasse CAlgorithmControlUnit als Attribut eine Instanz der Klasse CAlgorithmData. Mit Hilfe der Klasse CObservable, welche Teil des Entwurfsmusters Beobachter ist, können diese Daten überwacht werden.

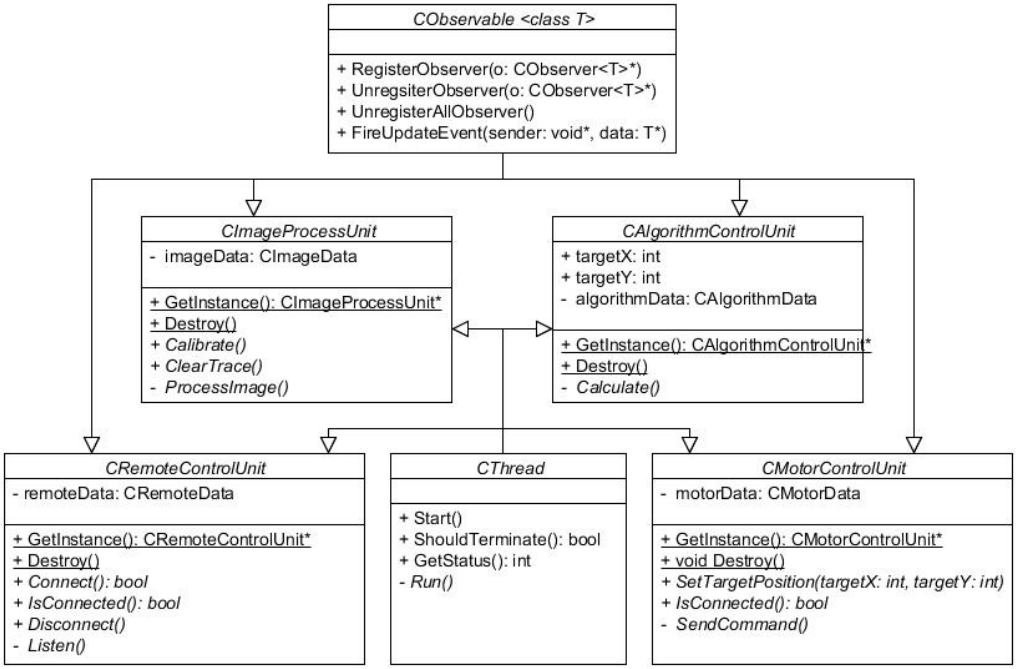

Abbildung 4.5: UML-Klassendiagramm der Aufgabenkomponente (weitere Dokumentation im Anhang B.3).

Für weitere Kontrollfunktionen sorgt die Klasse CThread, mit deren Hilfe eine Kontrolleinheit gestartet und gestoppt werden kann. Zudem lassen sich Informationen über den aktuellen Status erfragen. Der Aufruf der Methode Start() veranlasst den Start eines nebenläufigen Prozesses. Der auszuführende Code beinhaltet die abstrakte Methode der jeweiligen Kontrolleinheit (z.B. die Methode Calculate() der Klasse CAlgorithmControlUnit). Um beispielsweise einen konkreten Regelalgorithmus zu implementieren, muss eine Klasse erstellt werden, welche von CAlgorithmControlUnit abgeleitet wird. Der Grad der Generalisierung ist dabei durch die vorgegebenen abstrakten Methoden der entsprechenden Kontrolleinheit festgelegt. In diesem Fall müssen lediglich die Methode Calculate() implementiert und unter Verwendung der Beobachter-Methode FireUpdateEvent() die registrierten Beobachter über die Aktualisierung des mit Werten gefüllten CAlgorithmData-Objekts benachrichtigt werden. Mit Hilfe der Singleton-Methode GetInstance() wird die Instanz der abgeleiteten Klasse zurückgegeben. In Kapitel 6 findet dieses Beispiel Anwendung und kann als allgemeingültig für die verbleibenden Kontrolleinheiten angesehen werden.
Die abgeleitete Klasse der Kontrolleinheit CMotorControlUnit wird aus der Vorgängerversion des Softwaresystems übernommen, da sich die Ansteuerung der Schrittmotoren über die serielle Schnittstelle nicht verändert hat (siehe [Ust10]). Ebenso stammt die abgeleitete Klasse der CRemoteControlUnit für die manuelle Steuerung mittels der Wii-Remote aus der Arbeit von Yohan Humbert [Hum11]. Das Ver-

fahren zum Erkennen der Kugelposition, welches für die Arbeit als vorausgesetzt gilt, kann aufgrund der veränderten Übertragungstechnik des Kamerabildes (siehe Abschnitt 4.1) nicht weiterverwendet werden. Alternativ kann jedoch das Bild mit Hilfe der freien Bibliothek *OpenCV* (aktuell in der Version 2.1) [Ope11] über die USB-Schnittstelle gelesen werden. Zusätzlich bietet diese Bibliothek fertige Verfahren zur Bildverarbeitung, mit deren Hilfe das Erkennen der Kugelposition umgesetzt wird. Da diese Thematik nicht Gegenstand dieser Arbeit ist, soll hierauf im Einzelnen nicht weiter eingegangen werden.

Steuerungskomponente Abbildung 4.6 zeigt das Klassendiagramm der Steuerungskomponente. Die Klasse CMainController bietet die Funktionalität zur Steuerung der Wippe. Sie verwendet das Entwurfsmuster Singleton, um zu gewährleisten, dass zur Laufzeit lediglich eine Instanz dieser Klasse existiert.

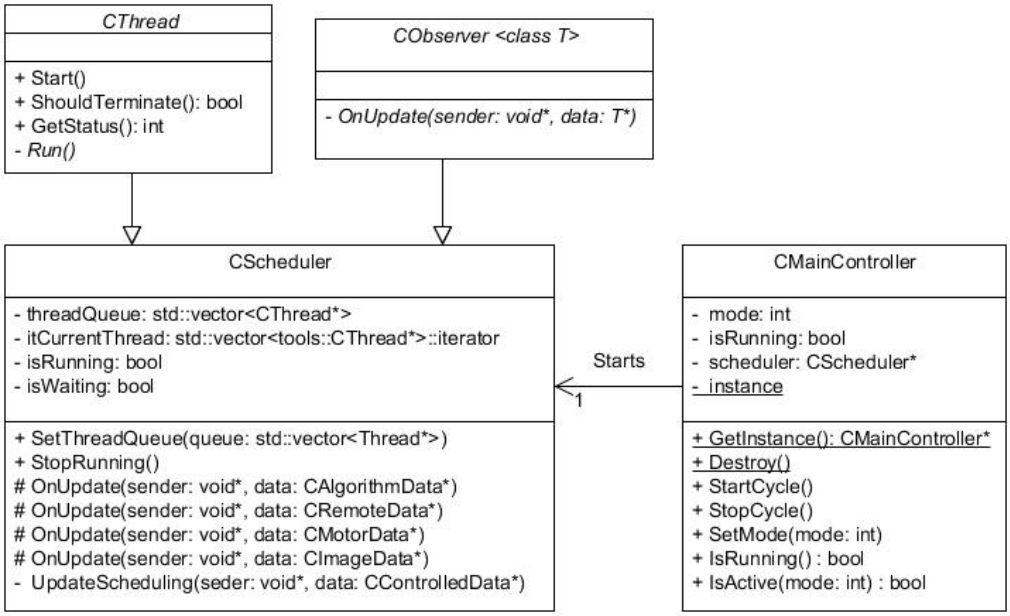

Abbildung 4.6: UML-Klassendiagramm der Steuerungskomponente (weitere Dokumentation im Anhang B.4).

Mit Hilfe der Methode SetMode() lässt sich der Betriebsmodus einstellen, welcher als Zahl übergeben wird. Für jeden Modus existiert ein vordefinierter Wert, bei welchem genau ein Bit gesetzt ist (vgl. Listing 4.1). Durch Addition der Werte ergibt sich eine Zahl, welche den entsprechenden Modus eindeutig definiert.

Listing 4.1: Definition der Zahlenwerte für die verschiedenen Betriebsmodi.

```
#define MODE_AUTOMATIC_X_AXIS 0x01
#define MODE_AUTOMATIC_Y_AXIS 0x02
#define MODE_WII_X_AXIS 0x04
```

```
#define MODE_WII_Y_AXIS 0x08
```

Innerhalb der Methode StartCycle() wird die Variable scheduler instanziiert. Die Aufgabe des MainControllers ist es, den Vektor threadQueue über die Methode SetThreadQueue() des Schedulers zu initialisieren. Die Reihenfolge der eingefügten Objekte und die Objekte selbst, welche Instanzen der Kontrolleinheiten aus der Aufgabenkomponente darstellen und somit von der Klasse Thread abgeleitet sind, hängen von dem konfigurierten Modus ab (vgl. Reihenfolge des Arbeitszyklus in Abschnitt 3.2). Zum Beispiel werden im Falle des automatischen Modus zunächst eine Instanz der Klasse CImageProcessUnit, anschließend eine Instanz der Klasse CAlgorithmControlUnit und schließlich eine Instanz der Klasse CMotorControlUnit hinzugefügt (vgl. Anhang A.1).
Die Methode Run() der Klasse CScheduler enthält die Hauptkontrollschleife (vgl. Anhang A.2). Dort wird mit Hilfe der Laufvariable itCurrentThread über den Vektor threadQueue mit den Instanzen der Kontrolleinheiten iteriert. Jedoch wird ein Iterationsschritt nur bedingt durch den Wert der Variable isWaiting durchgeführt. Hat diese den Wert „true“, wartet der Scheduler, bis sich der Wert auf „false“ ändert. Daraufhin wird isWaiting wieder auf „true“ gesetzt und der Iterationsschritt durchgeführt. Anschließend wird die Start()-Methode der vom Iterator itCurrentThread aktuell referenzierten Kontrolleinheit aufgerufen. Eine Kontrollabfrage überprüft, ob das Ende des Vektors erreicht ist. Trifft dieser Fall zu, wird wieder mit der ersten Kontrolleinheit begonnen.

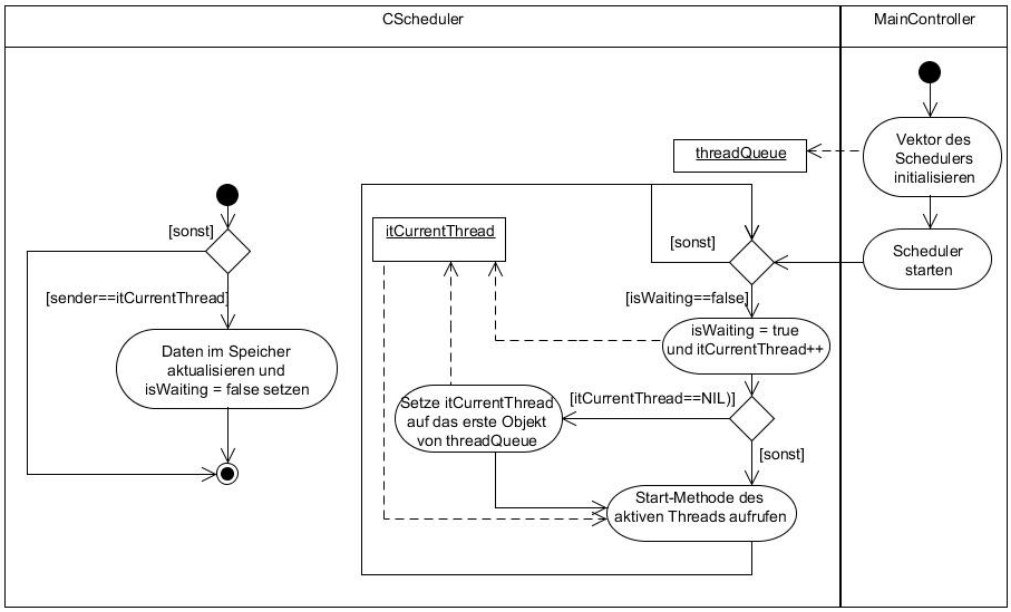

Abbildung 4.7: Aktivitätsdiagramm zur Veranschaulichung des Kontroll- und Datenflusses innerhalb der Steuerungskomponente.

Mit Hilfe des Entwurfsmusters Beobachter organisiert die Klasse CScheduler den

Datenfluss zwischen der Steuerungskomponente und dem Datenspeicher. Entsprechend dem aktiven Modus der Wippe registriert er sich als Beobachter bei relevanten Kontrolleinheiten, um über Aktualisierungen des entsprechenden Datentyps benachrichtigt zu werden. Demnach ist die Methode OnUpdate() für jeden möglichen Datentyp (CAlgorithmData, CRemoteData, CMotorData, CImageData) implementiert. Beim Eintreffen einer Aktualisierung werden die übergebenen Parameter an die private Methode UpdateScheduling() durchgereicht. Dort wird der Sender, welcher eine Kontrolleinheit referenziert, mit der durch die Variable itCurrentThread referenzierten Kontrolleinheit des Vektors threadQueue verglichen. Ist der Vergleich erfolgreich, so wird das übergebene Datenobjekt im Speicher aktualisiert und mit dem Setzen des Wertes der Variable isWaiting auf „false“ der nächste Iterationsschritt innerhalb der Hauptkontrollschleife eingeleitet. Durch Aufruf der Methode Start() wird die nächste Kontrolleinheit zum Arbeiten angestoßen (vgl. Anhang A.3).
Der beschriebene Kontrollfluss innerhalb der Steuerungskomponente wird durch das Aktivitätsdiagramms in Abbildung 4.7 dargestellt. Der rechte Startpunkt bezeichnet den Anfang der Methode StartCycle() innerhalb des MainControllers. Der linke Startpunkt hingegen wird beim Eintritt in die Methode UpdateScheduling() erreicht und damit auch entsprechend der Endpunkt beim Verlassen der Methode.

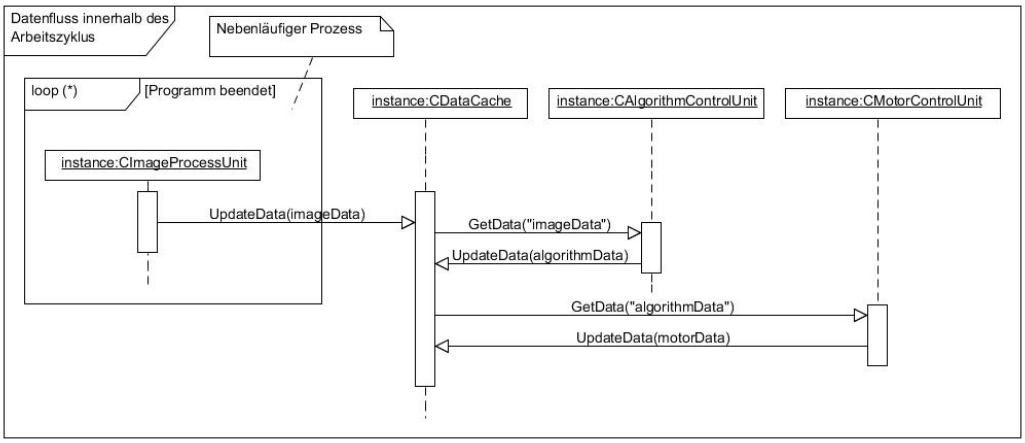

Abbildung 4.8: Datenfluss zwischen den verschiedenen Kontrolleinheiten während des automatischen Betriebs, welcher vom Scheduler gesteuert wird.

Das Sequenzdiagramm in Abbildung 4.8 veranschaulicht den vom Scheduler organisierten Datenfluss zwischen der Steuerungskomponente und dem Datenspeicher im automatischen Betrieb der Wippe. Der Übersicht wegen ist die Kommunikation der Kontrolleinheiten mit dem Datenspeicher vereinfacht dargestellt. Für die Erneuerung der Daten ist der Scheduler zuständig, welcher über die Methode FireUpdateEvent() benachrichtigt wird und dementsprechend eine Aktualisierung im Speicher vornimmt. Durch die folgende Erläuterung wird der genaue Ablauf deutlich.

Die Kontrolleinheit CImageProcessUnit empfängt in einer Endlosschleife, welche in einem nebenläufigen Prozess ausgeführt wird, periodisch Bilder der Kamera und verarbeitet diese. Anschließend benachrichtigt sie alle registrierten Beobachter über eine Aktualisierung der Bilddaten. Der Scheduler wartet zu diesem Zeitpunkt auf eine Benachrichtigung der Kontrolleinheit CImageProcessUnit. Trifft diese schließlich ein, werden die übergebenen Bilddaten im Datenspeicher aktualisiert, die Start()-Methode der CAlgorithmControlUnit durch den in der Hauptkontrollschleife erfolgten Iterationsschritt aufgerufen und anschließend auf eine Aktualisierung der Regelungsdaten gewartet. Auf ähnliche Art und Weise wird fortgefahren, bis der Scheduler schließlich wieder auf die Aktualisierung der Bilddaten wartet und die Prozedur von vorne beginnt. Prinzipiell ist es möglich, dass aktualisierte Bilddaten eintreffen, während der Scheduler auf die Benachrichtigung einer anderen aktiven Kontrolleinheit wartet. In diesem Fall werden die Bilddaten ignoriert.

4.2.4 Beschreibung der grafischen Benutzerschnittstelle

Die grafische Benutzerschnittstelle des bisher verwendeten Softwaresystems erscheint durch den Einsatz von Grafikkomponenten der MFC (Microsoft Foundation Classes) aus dem Jahr 2003[2] bezüglich des Aussehens und des Designs nicht aktuell. Zudem sind alle Kontrollelemente innerhalb eines Containers untergebracht, was im Falle einer Ausweitung der Funktionalität des Softwaresystems möglicherweise Platzprobleme verursacht. Somit wird eine neue grafische Benutzerschnittstelle benötigt, welche nicht nur vom Aussehen überzeugt, sondern ebenso eine Möglichkeit zur Erweiterung bietet.
Für die Entwicklung einer grafischen Benutzerschnittstelle in der Programmiersprache C++ (vgl. NFA-1) bietet sich das von der Firma Nokia entwickelte Framework Qt [Nok11] an. Neben einer großen Auswahl an grafischen Komponenten stellt Qt ebenso Softwarekomponenten für weitere Anwendungsgebiete wie beispielsweise Netzwerkprogrammierung oder Parallelprogrammierung zur Verfügung. Zudem wirbt es mit dem Aspekt der Plattformunabhängigkeit, was jedoch für das in dieser Arbeit entworfene Softwaresystem keine tragende Rolle spielt.
Um die Erweiterbarkeit der grafischen Benutzerschnittstelle zu gewährleisten, eignet sich die Verwendung von Registerkarten. Gleichzeitig wird dadurch zur Verbesserung der Übersichtlichkeit beigetragen, da in Bezug auf den Kontext zusammenhängende Grafikkomponenten innerhalb einer Registerkarte gruppiert werden können. Im Falle einer Erweiterung des Softwaresystems um eine bestimmte Funktionalität, kann, falls nötig, eine weitere Registerkarte hinzugefügt werden. In der folgenden Auflistung werden die vier verschiedenen Registerkarten der neuen Benutzerschnittstelle benannt und den in Abschnitt 4.2.1 erhobenen funktionalen Anforderungen zugeordnet:

[2]Weitere Informationen finden sich unter `http://msdn.microsoft.com/en-us/library/d06h2x6e%28v=vs.71%29.aspx`, Abgerufen am 20.07.2011.

- **Startseite** - Eine Registerkarte mit einer Startseite, welche kurz einen Überblick über das Wippe-Experiment gibt.
- **Automatischer Betrieb** - Eine Registerkarte zur Darstellung von Informationen und konfigurierbaren Optionen des automatischen Betriebs.
  - **Kamerabild** - Zeigt das aktuelle Kamerabild.
  - **Bearbeitetes Kamerabild** - Zeigt ein weiteres bearbeitetes Kamerabild, in welchem die Kugelposition und der von der Kugel zurückgelegte Pfad eingezeichnet sind.
  - **Bildkalibrierung** - Enthält einen Button für die Kalibrierung des Kamerabildes (vgl. FA-3.3).
  - **Kugelinformationen** - Koordinaten, Geschwindigkeit und Beschleunigung der Kugel werden durch Labels dargestellt (vgl. FA-4.1).
- **Wii-Remote Steuerung** - Eine Registerkarte zur Darstellung von Informationen und konfigurierbaren Einstellungen des manuellen Betriebs.
  - **Kalibrierung** - Enthält einen Button für die Kalibrierung der Wii-Remote (vgl. FA-3.2).
  - **Verbinden** - Enthält einen Button, um die Verbindung mit einer Wii-Remote herzustellen.
  - **Verbindung trennen** - Enthält einen Button, um die Verbindung mit der Wii-Remote zu trennen.
  - **Auslenkung** - Enthält eine grafische Komponente, welche die Auslenkung der Wii-Remote visualisiert.
- **Konfiguration** - Eine Registerkarte zum Konfigurieren globaler Einstellungen.
  - **Sprache** - Beinhaltet ein Auswahlmenü zum Einstellen der Sprache (Englisch, Deutsch).
  - **Platte** - Enthält Komponenten, mit deren Hilfe die Möglichkeit zur Kalibrierung der Platte besteht (vgl. FA-3.1).
  - **Serielle Verbindung** - Enthält zwei Auswahlmenüs zur Einstellung des Ports und der Baudrate.
  - **Latenzgraph** - Enthält einen Button, mit dessen Hilfe die Latenzgraphen der einzelnen Kontrolleinheiten angezeigt werden (vgl. FA-4.2).

Neben dem generischen Aspekt des Registerkartenmenüs besitzt die grafische Benutzerschnittstelle zwei statische Komponenten, die unabhängig von der ausgewählten Registerkarte präsent sind. Zum einen eine Statusbar, welche mit Hilfe von Icons

den Status der Verbindung zur Wii-Remote und zum Treibermodul der Schrittmotoren anzeigt. Des Weiteren ein Panel, welches die Möglichkeit zur Auswahl der Betriebsart anbietet und einen Button enthält, mit dem sich der Versuch starten und stoppen lässt (vgl. FA-1). Durch den Einsatz von Radio-Buttons, welche jeweils für eine Achse innerhalb einer Button-Gruppe zusammengefasst sind, so dass nur eine Betriebsart pro Achse wählbar ist, können die verschiedenen Modi beliebig miteinander kombiniert werden (vgl. FA-2). Abbildung 4.9 zeigt repräsentativ einen Screenshot der neuen grafischen Benutzerschnittstelle, bei welchem die Registerkarte für die Steuerung mittels Wii-Remote ausgewählt ist. Weitere Screenshots finden sich im Anhang C dieser Arbeit.

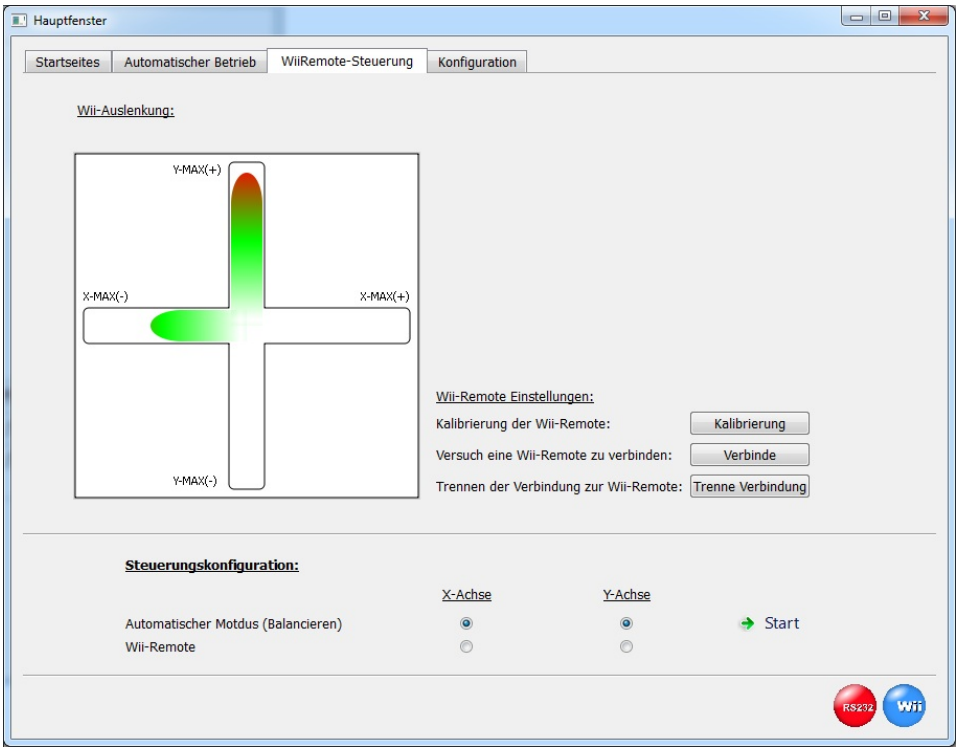

Abbildung 4.9: Repräsentativer Screenshot der grafischen Benutzerschnittstelle.

Kapitel 5

Analyse des überarbeiteten Systems

Die folgenden Abschnitte beschäftigen sich mit der Analyse des überarbeiteten Wippe-Systems. Dabei erfolgt zunächst die Messung der Arbeitszeiten der verschiedenen Komponenten innerhalb des Softwaresystems. Den Anschluss bildet die Betrachtung der Latenzzeiten, welche durch das externe System verursacht werden. Diesbezüglich wird vor allem der Aspekt hinsichtlich des Alters der Daten erläutert. Abschließend wird das Wippe-Experiment anhand der in Abschnitt 3.2 aufgestellten Zeitbedingung verifiziert.

5.1 Latenzzeiten des Softwaresystems

Damit die Erfüllung der Zeitbedingung überprüft werden kann, müssen zunächst die Zeiten der einzelnen Teilaufgaben des Arbeitszyklus innerhalb des Softwaresystems gemessen werden.
In Abschnitt 4.2.3 wird erläutert, dass jeder Kontrolleinheit eine Instanz der von CControlledData abgeleiteten Klasse zugeordnet ist. Das Attribut calcTime dieser Datenklasse stellt die Arbeitszeit der zugehörigen Kontrolleinheit in Mikrosekunden dar. Durch die Verwendung des Beobachter Patterns, kann sich eine von CObservable abgeleitete Klasse bei einer Kontrolleinheit registrieren und wird über Aktualisierungen der entsprechenden Instanz der Datenklasse benachrichtigt (vgl. Klasse CScheduler in Abschnitt 4.2.3).
Eine Analyseklasse, welche von den Klassen CObserver<CImageData> und CObserver<CMotorData> abgeleitet ist, wird speziell zum Protokollieren der Latenzzeiten verwendet. Dafür registriert sie sich bei den Kontrolleinheiten CImageProcessUnit und CMotorControlUnit als Beobachter. Bei einer Aktualisierung des entsprechenden Datums wird die Analyseklasse benachrichtigt und schreibt die Latenzzeiten in eine Datei.
Ein Testlauf ist erforderlich, um einen repräsentativen Datensatz an Werten zu sammeln. Da die automatische Steuerung noch nicht realisiert ist, wird die Wippe statt-

dessen im manuellen Modus betrieben und mit Hilfe der Wii-Remote gesteuert[1]. Währenddessen arbeitet die Bildverarbeitungskomponente und wertet die Bilddaten aus, welche über die Analyseklasse protokolliert werden. Durch die Auslenkung der Wii-Remote werden die Motorpositionen von der Fernbedienungskomponente berechnet und an die Motorkomponente weitergeleitet, um die entsprechenden Befehle an das Treibermodul zu senden. Diese Daten werden ebenfalls anhand der Analyseklasse dokumentiert. Tabelle 5.1 zeigt einen Ausschnitt aus dem Datensatz eines 5-minütigem Testlaufs. Die Arbeitszeit der Regelungskomponente, welche erst im späteren Verlauf der Arbeit beschrieben wird, kann vernachlässigt werden, da die Berechnung der neuen Motorpositionen nicht rechenintensiv ist.

Tabelle 5.1: Ausschnitt aus dem Datensatz des Testlaufs im manuellen Betrieb. **ZBK** - Rechenzeit der Bildverarbeitungskomponente, **KX/KY** - Kugelposition, **ZMK** - Rechenzeit der Motorkomponente, **MX/MY** - Eingestellte Motorenauslenkung.

ZBK	KX	KY	ZMK	MX	MY
29311	73	77	11510	506	7650
28627	73	78	11181	0	-480
31410	74	77	11419	482	482
42857	74	79	11490	0	-444
29978	74	78	5662	0	491
29839	75	79	11234	487	-487
30766	75	80	11158	300	-490
31706	75	80	5514	300	7500
28961	75	81	5708	300	-487
29908	76	81	11708	488	7450
31159	76	82	11487	360	-480
30219	76	82	5870	360	7400
29914	76	83	5648	360	-497
28290	77	84	11172	483	-483
31857	77	84	11454	420	7300
45681	77	85	5575	420	-218
30806	77	85	5744	420	7250

Weitere Testläufe zeigen, dass die Arbeitszeit der Bildverarbeitungskomponente stark von den Lichtbedingungen beeinflusst wird. Der sich aus dem angesprochenen Datensatz berechnete Mittelwert von ungefähr 31 Millisekunden ergibt sich bei guten Lichtverhältnissen.
Die Arbeitszeit der Motorkomponente beträgt durchschnittlich 10 Millisekunden. Dieser Wert stellt lediglich die Zeit dar, welche benötigt wird, um die neuen Motor-

[1] Durch die in Abschnitt 6.2 gemessenen Werten bei einem Testlauf im automatischen Betrieb, wird die hier verwendete Vorgehensweise legitimiert.

positionen an das Treibermodul über die serielle Schnittstelle zu senden, inklusive der Wartezeit für die Antwort. Somit kann davon ausgegangen werden, dass die Auslenkung der Motoren nach der Hälfte der Sendezeit beginnt (5 Millisekunden).

5.2 Latenzzeiten des externen Systems

Die Latenzzeiten des externen Systems müssen bei der Regelung der Wippe berücksichtigt werden. Speziell handelt es sich zum einen um das Alter des Kamerabildes beim Eintreffen innerhalb des Rechensystems und des Weiteren um die Geschwindigkeit, mit welcher die Schrittmotoren einen Winkel ansteuern.
Letzteres lässt sich aus den Angaben der in Abschnitt 4.1 aufgeführten Tabelle 4.1 ablesen und beträgt $1,22 \cdot 10^5 \frac{\mu steps}{s}$. Der Rechenweg, um von dem eingestellten Wert für die Geschwindigkeit (2000) auf die Drehfrequenz zu schließen, ist in der Bedienungsanleitung des Treibermoduls [Tri11] beschrieben. Innerhalb der Arbeit von Mehmet-Sefa Usta [Ust10] haben Messungen ergeben, dass für beide Achsen $\pm 1000 \mu steps$ die Platte entsprechend um $1°$ auslenken und eine Maximalauslenkung von $\pm 15°$ möglich ist. Es ergibt sich somit eine Winkelgeschwindigeit von $\frac{122}{s}$.
Um das Alter des Kamerabildes ermitteln zu können, steht bisher keine anwendbare Verfahrensweise zur Verfügung. Die Notwendigkeit zur Erarbeitung eines entsprechenden Verfahrens im Rahmen dieser Arbeit ergibt sich damit zwangsläufig. Ausgangspunkt der Überlegungen hierbei ist die Grundidee, mit der oberhalb der Platte installierten Webcam, eine LED zu filmen, welche sich auf der Platte befindet. Auf dem Computer wird in der Folge ein Programm ausgeführt, das anhand des gelieferten Kamerabildes den Zustand der LED erkennen kann. Schaltet man die LED an, wird die Zustandsänderung verzögert vom Programm wahrgenommen. Diese in Zeiteinheiten gemessene Verzögerung liefert die notwendigen Daten um das Alter des Kamerabildes zu bestimmen.

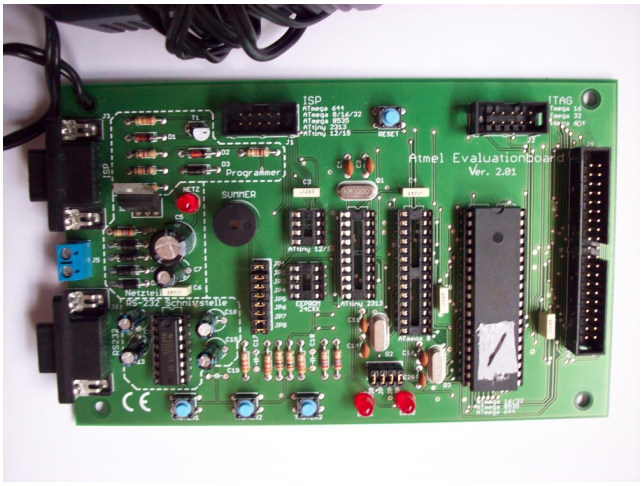

Abbildung 5.1: Atmel Evaluations-Board V.2.0.1 mit Atmega16.

Um die vorab beschriebene Verfahrensweise im Rahmen eines Versuchs überprüfen zu können, ist es erforderlich den Zeitpunkt zu kennen, zu welchem die LED angeschaltet wird. Dies wird durch die Fähigkeit des Computersystems, die LED an- und auszuschalten, ermöglicht. Zur Anwendung kommt hierbei das Atmel Evaluations-Board [Atm11b], auf welchem sich bereits zwei LEDs befinden, die mit Hilfe eines Atmega16 Mikrocontroller [Atm11a] angesteuert werden (vgl. Abbildung 5.1). Die Kommunikation zwischen dem Atmel-Board und dem Computer wird über die serielle Schnittstelle realisiert. Das zu entwickelnde Programm kann demnach parallel zur Erkennung des Zustands der LED diesen mit Hilfe entsprechender Befehle beeinflussen.
In Abbildung 5.2 wird der Programmablauf anhand eines Aktivitätsdiagramms gezeigt. Dabei wird zwischen dem parallelen Prozess für die Zustandserkennung und dem Hauptprogramm zum Steuern des Zustands der LED unterschieden.

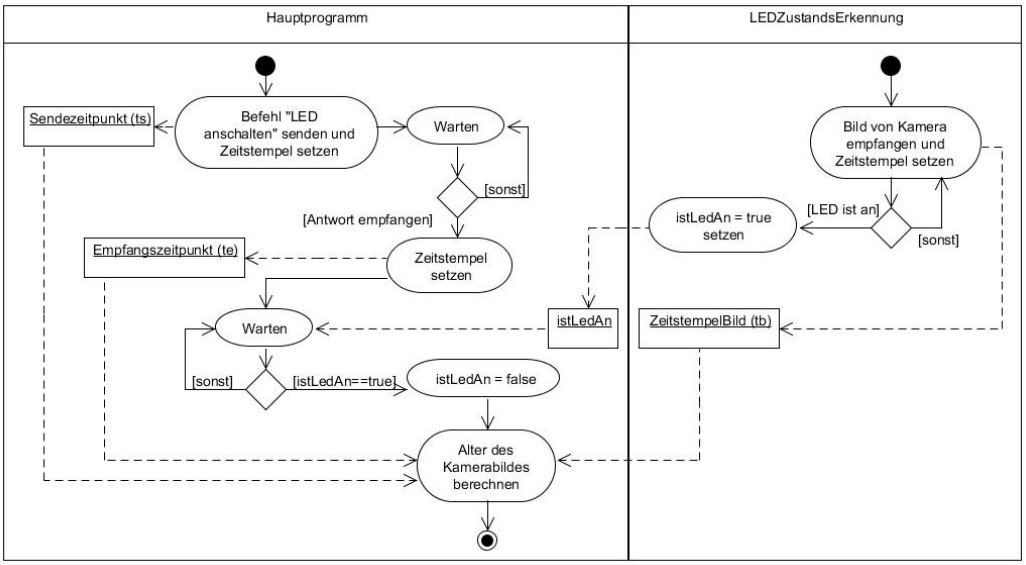

Abbildung 5.2: Aktivitätsdiagramm zur Veranschaulichung des Programmablaufs, um das Alter des Kamerabildes zu bestimmen.

Zum Startzeitpunkt im Hauptprogramm wird davon ausgegangen, dass der parallele Prozess für die Zustandserkennung der LED bereits aktiv ist. Dessen Aufgabe besteht darin, den Zeitpunkt des Eintreffens eines Bildes t_b aufzuzeichnen und anschließend zu überprüfen, ob die LED auf diesem Bild eingeschaltet ist.
Im Hauptprogramm wird währenddessen der Befehl zum Anschalten der LED über die serielle Schnittstelle gesendet. Um später bei der Berechnung des Bildalters die Zeit berücksichtigen zu können, welche benötigt wird, um den Befehl an den Mikrocontroller zu senden, wird der Sendezeitpunkt durch den Zeitstempel t_s festgehalten. Das Hauptprogramm wartet nun, dass die korrekte Ausführung des Befehls bestätigt wird und setzt beim Eintreffen der Antwort den Zeitstepmel t_e. Die LED auf dem Atmel-Board ist nun angeschaltet und es muss abgewartet werden, bis die

Bildverarbeitung das erste Bild empfängt, bei dem die Zustandsänderung erkannt wird. Das Hauptprogramm wird darüber durch den booleschen Wert istLedAn von der LEDZustandsErkennung benachrichtigt. Ist der Wert „true“, wird er zunächst wieder mit „false“ initialisiert. Anschließend wird die Berechnung des Alters des Kamerabildes Δt_a anhand der folgenden Formel vorgenommen:

$$\Delta t_a = t_b - \left(t_s + \frac{t_e - t_s}{2} \right) \tag{5.1}$$

Für die Bestimmung des Zeitpunktes, zu dem sich die LED in der realen Welt einschaltet, wird zum Sendezeitpunkt des entsprechenden Befehls die Hälfte der Sendezeit addiert.
Durch Ausschalten der LED am Ende des Versuchs, kann im Anschluss daran ein neuer Durchgang beginnen. Durch mehrmaliges Wiederholen kann ein Mittelwert gebildet werden, welcher eine gute Abschätzung für das Alter des Kamerabildes darstellt. Zudem wird durch das resultierende Blinken der LED eine Art visuelle Darstellung der verzögerten Zustandserkennung des Computers erzeugt. Der Screenshot in Abbildung 5.3 zeigt die Statistik eines 5-minütigen Tests bei Tageslichtverhältnissen. Insgesamt konnte der Versuch 2995 mal durchgeführt werden

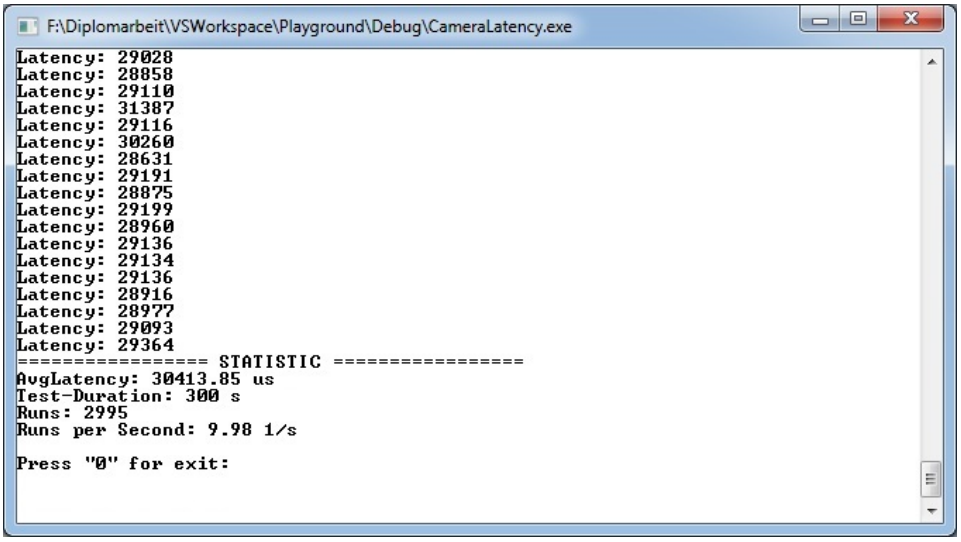

Abbildung 5.3: Screenshot des Programms, um das Alter des Kamerabildes zu bestimmen.

und der Mittelwert des Bildalters beträgt ungefähr 30 Millisekunden. Wird der gleiche Test in einem leicht abgedunkelten Raum durchgeführt, so können lediglich 974 Durchgänge erreicht werden. Der Mittelwert des Bildalters beträgt dabei 120 Millisekunden. Diese Abhängigkeit von den Lichtverhältnissen wurde bereits bei den Messungen der Bildverarbeitungskomponente in Abschnitt 5.1 festgestellt und lässt sich anhand der automatischen Regelung der Belichtungszeit durch die Webcam begründen. Bei guten Verhältnissen fällt diese gering aus und die Framerate

der Kamera beträgt annähernd $30\frac{frames}{s}$. Entsprechend hoch fällt die Belichtungszeit bei schlechten Lichtverhältnissen aus, folglich leidet darunter die Framerate der Kamera.

5.3 Verifizierung des Wippe-Experiments

In Abschnitt 3.2 wird die Zeitbedingung für den korrekten Betrieb der Wippe hergeleitet. Anhand der gewonnenen Erkenntnisse durch die Messung der verschiedenen Latenzzeiten, lässt sich nun überprüfen, ob das Balancieren der Kugel im Zentrum der Platte grundsätzlich möglich ist. Gleichzeitig wird dadurch die nicht-funktionale Anforderung in Bezug auf Leistung und Effizienz (vgl. NFA-3) nachgewiesen.
Die Zeitbedingung lautet:

$$A \equiv (\Delta e_K) + (\Delta e_B + \Delta e_R + \Delta e_S) + (\Delta e_M) \leq d_2 \tag{5.2}$$

Die zuvor durchgeführten Messungen bei Tageslichtverhältnissen haben folgende Werte für die Ausführungszeiten der einzelnen Teilaufgaben ergeben:

$$\Delta e_K = 30ms, \quad \Delta e_B = 31ms, \quad \Delta e_S = 5ms, \quad \Delta e_M = \frac{\alpha}{122} \cdot 10^3 ms \tag{5.3}$$

Die Zeit Δe_M ist dabei abhängig von dem Winkel α, um welchen die Motoren ausgelenkt werden. Wie bereits in Abschnitt 5.1 angedeutet, kann die Zeit Δe_R zur Berechnung der Motorpositionen mit Hilfe eines Regelalgorithmus vernachlässigt werden. Für das in Abschnitt 3.1.3 erläuterte Gedankenexperiment gilt:

$$d_2 = 526ms, \quad \alpha = 15° \quad \text{und somit} \quad \Delta e_M = \frac{15}{122} \cdot 10^3 ms = 123ms \tag{5.4}$$

Durch Einsetzen in die Formel für die Zeitbedingung ergibt sich folgende Berechnung:

$$(30ms) + (31ms + 5ms) + (123ms) = 189ms \tag{5.5}$$

$$A \equiv 189ms \leq 526ms \tag{5.6}$$

Dadurch ist gewährleistet, dass die Platte in der Lage ist, die vom Plattenrand aus der Ruhe gestartete Kugel zu balancieren.
Es ist im Weiteren denkbar, dass der Ball bereits eine Anfangsgeschwindigkeit v_0 besitzt. Ein interessanter Aspekt ist in diesem Zusammenhang die Bestimmung der maximalen Anfangsgeschwindigkeit, welche die Kugel besitzen darf. Dabei wird davon ausgegangen, dass erst im zweiten Durchlauf der Kontrollschleife eine Auslenkung der Motoren stattfindet, da zwei Kamerabilder benötigt werden, um Bewegungsrichtung und Geschwindigkeit der Kugel zu bestimmen. Somit gilt für die Reaktionszeit des Systems folgender Zusammenhang:

$$2 \cdot (30ms) + 2 \cdot (31ms + 5ms) + (123ms) = 255ms \tag{5.7}$$

Demzufolge muss die Kugel mindestens eine Zeit von $255ms$ bis zum Erreichen des Zentrums beanspruchen. Sofern sie die errechnete Minimalzeit unterschreitet, wird sie mit hoher Wahrscheinlichkeit von der Platte fallen. Unter Beachtung der maximalen Auslenkung a_{max} (vgl. Formel (3.18)) lässt sich anhand der Formel (3.20) aus Abschnitt 3.1.3 die maximale Anfangsgeschwindigkeit v_{0M} berechnen:

$$s = \frac{1}{2} \cdot a_{max} \cdot t^2 + v_{0M} \cdot t \quad \Leftrightarrow \quad v_{0M} = \frac{s}{t} - \frac{a_{max} \cdot t}{2} \tag{5.8}$$

$$v_{0M} = \frac{0,25m}{255 \cdot 10^{-3}s} - \frac{1,81\frac{m}{s^2} \cdot 255 \cdot 10^{-3}s}{2} = 0,75\frac{m}{s} \tag{5.9}$$

Um eine Zeitbedingung innerhalb des Programmcodes des Regelalgorithmus für die Verifizierung des Versuchs verwenden zu können, wird der kanonische Ansatz zur Herleitung von Echtzeitbedingungen verwendet, welchen Dieter Zöbel in [Zö05] vorstellt. Im Folgenden wird das dafür verwendete Modell erläutert und anschließend mit Hilfe des Wippe-Experiments instanziiert.
Abbildung 5.4 zeigt das Grundmodell eines Regelsystems. Ähnlich wie bereits in Abschnitt 3.2 wird eine Unterteilung in ein technisches System und ein Rechensystem vorgenommen. Mit der Unterscheidung zwischen Sensor und Aktuator findet eine senkrechte Einteilung statt. Die physikalischen Eigenschaften des technischen Systems $V = \{V_1, ..., V_n\}$ werden innerhalb des Rechensystems in Form von Bildwerten $OV = \{OV_1, ..., OV_n\}$ abgebildet. Physikalische Werte den entsprechenden Bildwerten zuzuordnen und somit eine Verbindung zwischen technischem System und Rechensystem herzustellen, ist Aufgabe des Sensors, so wie in der folgenden Relation dargestellt wird:

$$SR_x \subseteq V_x \times OV_x \tag{5.10}$$

Analog ordnet der Aktuator den Bilddaten wiederum die physikalischen Werte zu:

$$AR_y \subseteq OV_y \times V_y \tag{5.11}$$

Innerhalb des Rechensystems werden aus den vom Sensor gelieferten Bildwerten neue Bildwerte für den Aktuator berechnet. Dieser Vorgang wird in Abbildung 5.4 mit CA (engl. control action, dt. Kontrolleinfluss) bezeichnet. Der grundlegende, zeitgesteuerte Programmablauf innerhalb des Rechensystems sieht wie folgt aus (siehe [Zö05, S.3]):

```
while (true) {
   wait_until(t);
   OVx = sensor_x.get();
   if (C(OVx)) OVy = A(OVx, OVy);
   actuator_y.set(OVy);
   t = t + period;
}
```

Der durch das Regeln angestrebte Zustand des Systems wird mit Hilfe des Prädikats $I_{RT}\,(V)$ dargestellt. Im Kontrast dazu basiert das aufgeführte Programm auf formalen Ausdrücken, beispielsweise die Bedingung C(OVx) in Zeile 4.

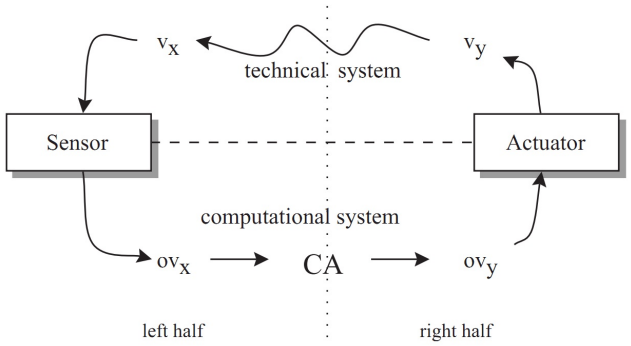

Abbildung 5.4: Grundmodell eines Regelsystems.

Bedingt durch die Ungenauigkeit des Sensors muss davon ausgegangen werden, dass zwei verschiedene physikalische Werte $v1_x$ und $v2_x$ auf den gleichen Wert ov_x abgebildet werden. Analog gilt, dass der gleiche physikalische Wert v_x einmal auf $ov1_x$ und ein anderes mal auf $ov2_x$ abgebildet werden kann. Daher ist es erforderlich für einige Werte $M \subset OV_x$ die ursprünglichen physikalischen Werte zu kennen:

$$DOM(SR_x, M) = \{v_x | (v_x, ov_x) \in SR_x \wedge ov_x \in M\} \tag{5.12}$$

Gleichermaßen ist es hilfreich für eine Menge physikalischer Werte $M \subset V_x$ die durch den Sensor abgebildeten Werte zu kennen:

$$IMG(SR_x, M) = \{ov_x | (v_x, ov_x) \in SR_x \wedge v_x \in M\} \tag{5.13}$$

Es muss berücksichtigt werden, dass die Bildwerte vergänglich sind und im Laufe der Berechnungen innerhalb des Rechensystems altern. Hierzu wird zunächst festgelegt, dass ein Wert ov_x eine absolute zeitliche Konsistenz Δatc_x besitzt, wenn in einem Intervall $[t - \Delta atc_x, t]$ ein v_x existiert hat, wobei t den Zeitpunkt widerspiegelt, zu dem die CA durchgeführt ist. Zudem muss folgender Zusammenhang gelten:

$$(v_x, ov_x) \in SR_x, \quad \text{mit} \quad v_x \in V_x \tag{5.14}$$

Die Zeitspanne Δbtc_x bildet sich aus der Zeit, welche der Sensor zum Messen benötigt und aus der vom internen System benötigten Rechenzeit, welche bis zum Zeitpunkt t vergangen ist. Ein neues Intervall $[t - \Delta atc_x, t - \Delta btc_x]$ grenzt somit den Zeitpunkt, zu dem v_x existiert hat, weiter ein.

Für eine verfeinerte Beschreibung des Regelsystems - die bisherige Beschreibung beschränkt sich lediglich auf die linke Hälfte (vgl. Abbildung 5.4) - werden folgende Kenntnisse vorausgesetzt:

- das minimale Alter Δsl_x und maximale Alter Δsh_x des Bildwertes bei Ausführung des Befehls sensor_x.get() bezüglich der Relation SR_x,
- die Periode Δp_i des Prozesses i, welcher den Wert v_x auf den Bildwert ov_x abbildet und die Bedingung C(OVx) auswertet,
- die Entwicklung des technischen Systems über ein Zeitintervall $\Delta\tau \in \Delta T$, welche mit der folgenden Funktion ausgedrückt wird:

$$TS_x : 2^{V_x} \times \Delta T \rightarrow 2^{V_x} \tag{5.15}$$

Anhand der getroffenen Definitionen kann zum Zeitpunkt der Auswertung von C(OVx) die Menge an Werten $SSV_x \subset V_x$ bestimmt werden, aus welchen der Bildwert ov_x entstanden sein kann:

$$SSV_x = \bigcup_{\Delta\tau \in [\Delta sl_x, \Delta sh_x + \Delta p_i]} TS(DOM(SR_x, \{ov_x\}), \Delta\tau) \tag{5.16}$$

Bisher muss jeder Wert $v_x \in SSV_x$ entsprechend eines Wertes ov_x, für welchen die Bedingung C(OVx) zutrifft, auch I_{RT} erfüllen.
Um die gesamte Entwicklung des Systems ausdrücken zu können, muss nun noch der Zeitpunkt von der Übermittlung von ov_y an den Aktuator bis hin zum Einsetzen der Regelung durch den Wert v_y berücksichtigt werden. Analog zur linken Hälfte des Systems beansprucht die Ausführung des Befehls actuator_y.set(OVy) eine minimale Zeit Δal_y und eine maximale Zeit Δah_y, bis das technische System entsprechend beeinflusst wird. Zudem muss, vom Zeitpunkt der Auswertung der Bedingung C ausgehend, die ausgeführte Aktion gültig sein, bis sie erneut in der nächsten Periode ausgeführt wird.
Für die Berechnung der minimalen Zeit, welche die Kontrollschleife beansprucht, ist nicht die Ausführungszeit Δe_i des Prozesses i maßgebend, sondern eine reduzierte Ausführungszeit Δre_i, welche beim Überschreiben des Wertes OVx beginnt und mit dem Setzen der Ausführung des Befehls actuator_y.set(OVy) endet.
Die Berechnungen der Intervallgrenzen ergeben sich laut [Zö05] folgendermaßen:

$$\Delta l_{xy} = \Delta sl_x + \Delta re_i + \Delta al_y \tag{5.17}$$

$$\Delta h_{xy} = \Delta sh_x + 2\Delta p_i + \Delta ah_y \tag{5.18}$$

Innerhalb dieses Intervalls wird nun das gesamte Regelsystem berücksichtigt und entsprechend die Definition der Menge SSV_x modifiziert:

$$SSV_x = \bigcup_{\Delta\tau \in [\Delta l_{xy}, \Delta h_{xy}]} TS(DOM(SR_x, \{ov_x\}), \Delta\tau) \tag{5.19}$$

Darauf bezogen ist die Korrektheit von C(OVx) äquivalent dazu, dass I_{RT} für jeden Wert $v_x \in SSV_x$ gültig ist.

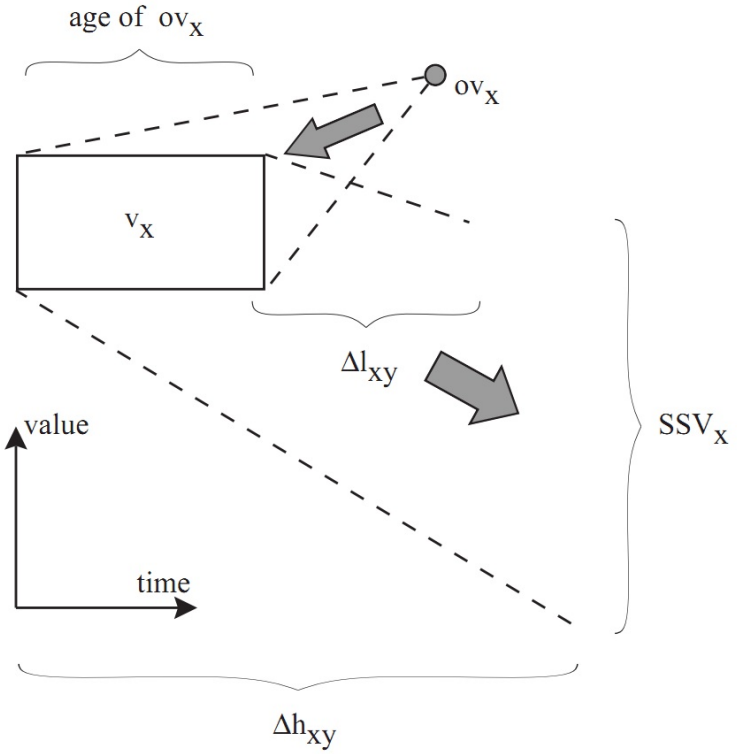

Abbildung 5.5: Veranschaulichung des Entstehungsprozesses der Menge SSV_x.

Abbildung 5.5 stellt die Entstehung der Menge SSV_x in Form eines Diagramms dar. Ein gegebener Wert ov_x wird anhand eines v_x einer Menge an möglichen Werten, welche aus Messfehlern des Sensors und dem möglichen Alter der Daten resultieren, abgebildet. Die möglichen Werte stellen die Entwicklung des technischen Systems dar, welches sich im Laufe der Zeit verändert und schließlich in der Menge SSV_x zusammengefasst wird. Das Diagramm deutet an, dass der Wertebereich von SSV_x im weiteren Verlauf anwächst, ausgelöst durch Abschätzungen und sich addierenden Messfehlern. Durch Optimierung der Abschätzung kann dieses Wachstum gegebenenfalls eingeschränkt werden, jedoch verbleibt stets eine Restunsicherheit.
Bei der Programmentwicklung ist I_{RT} üblicherweise vorgegeben und die Bedingung C(OVx) muss implementiert werden. Per der Definition der Menge SSV_x wird nun schrittweise ausgehend vom Prädikat I_{RT} die Bedingung C(OVx) abgeleitet [Zö05, S.5]:

1. Anhand von I_{RT} wird die Menge V'_x definiert, welche alle möglichen physikalischen Werte ausdrückt, für die I_{RT} gültig ist:

$$V'_x = \{v'_x \in V_x | I_{RT}(v'_x)\} \tag{5.20}$$

2. Zu einem Zeitpunkt $t-\Delta\tau$, mit $\Delta\tau \in [\Delta l_{xy}, \Delta h_{xy}]$ misst der Sensor das technische System. Es wird nun die Untermenge V_x'' betrachtet, welche innerhalb des Zeitintervalls $\Delta\tau$ durch TS Werten $v_x' \in V_x'$ zugeordnet wird:

$$V_x'' = \bigcup_{\Delta\tau\in[\Delta l_{xy},\Delta h_{xy}]} \{v_x'' \in V_x | TS(\{v_x''\}, \Delta\tau) \subseteq V_x'\} \tag{5.21}$$

3. In Bezug auf das Rechensystem wird eine Untermenge $OV_x' \subseteq OV_x$ gebildet, so dass die Werte ov_x' von Werten $v_x'' \in V_x''$ stammen, für welche I_{RT} gültig ist:

$$OV_x' = \{ov_x' \in OV_x | DOM(SR_x, \{ov_x'\}) \subseteq V_x''\} \tag{5.22}$$

4. Abschließend wird aus der Menge OV_x' die Bedingung C(OVx) abgeleitet.

Die beschriebene Vorgehensweise wird nun auf das Wippe-Experiment angewendet. Dabei wird auf das in Abschnitt 3.1 vorgestellte Gedankenexperiment zurückgegriffen. Analog beschränkt sich die Modellierung nur auf eine Achse. Damit das Alter des Kamerabildes und damit einhergehend die Ausführungszeit der Bildverarbeitung als konstant betrachtet werden können, wird von gleichbleibenden guten Lichtverhältnissen ausgegangen.
Die erste Aufgabe besteht darin, das Prädikat I_{RT} bezüglich des Wippe-Experiments zu definieren. Die textuelle Beschreibung lautet folgendermaßen:

Bewegt sich die Kugel auf das Zentrum der Platte zu und wird in gleicher Richtung beschleunigt, so muss die Platte, unter Berücksichtigung der aktuellen Position Δs_Z, der aktuellen Geschwindigkeit v_{Akt} und der aktuellen Beschleunigung a_{Akt} der Kugel, vor dem Erreichen des Zentrums der Kugel in Nullstellung zu bringen sein.

Diese Beschreibung muss nun in einen logischen Ausdruck überführt werden, welcher innerhalb des Regelalgorithmus überprüfbar ist. Es ist möglich abzuschätzen, wie lange die Kugel bis zum Erreichen des Zentrums benötigt. Durch Vergleichen dieses Wertes mit einem Zeitwert, welcher garantiert, dass die Platte innerhalb dieser Zeit in Nullstellung gebracht werden kann, lässt sich I_{RT} ausdrücken. Im ersten Durchgang des Kontrollzyklus werden keine neuen Motorpositionen berechnet, da beispielsweise zum Bestimmen der Kugelgeschwindigkeit die aktuelle Position x_{Akt} und die alte Position der Kugel x_{Alt} benötigt werden. Innerhalb des zweiten Durchgangs wird von einer Auslenkung der Platte um 15° ausgegangen. Die Dauer der Periode Δp_i, welche sich aus der Summe der Ausführungszeiten Δe_B und Δe_S ergibt, beträgt $36ms$. Zusätzlich beansprucht die Auslenkung der Platte um $15°$ eine Zeit von $123ms$ (vgl. Abschnitt 5.2). I_{RT} lässt sich demnach wie folgt formulieren:

$$I_{RT} \equiv v_x \geq 36ms + 36ms + 123ms = 195ms \tag{5.23}$$

v_x entspricht in diesem Zusammenhang nicht direkt einem durch das externe System gemessenen Wert, sondern der Zeit Δt_Z, welche sich mit Hilfe der durch die Kamerabilder bestimmten Kugelposition ergibt und die Zeit beschreibt, welche die Kugel bis zum Zentrum benötigt. Mit den Formeln aus Abschnitt 3.1 lässt sich Δt_Z wie folgt berechnen:

$$\Delta s_Z = 0,25m - x_{Akt}, \quad v_{Akt} = \frac{x_{Akt} - x_{Alt}}{\Delta p_i}, \quad a_{Akt} = \frac{v_{Akt} - v_{Alt}}{\Delta p_i} \tag{5.24}$$

$$\Delta t_Z = \frac{\sqrt{v_{Akt}^2 + 2 \cdot a_{Akt} \cdot s_Z} - v_{Akt}}{a_{Akt}} \tag{5.25}$$

Die Intervallgrenzen Δl_{xy} und Δh_{xy} werden zur Vereinfachung zu Δt_{xy} zusammengefasst, da bei den Messungen der Ausführungs- und Latenzzeiten keine Maximal- beziehungsweise Minimalwerte betrachtet werden:

$$\Delta t_{xy} = \Delta e_K + \Delta e_B + \Delta e_S = 30ms + 31ms + 5ms = 66ms \tag{5.26}$$

Alle Voraussetzungen für die Herleitung der Bedingung C(OVx) sind nun erfüllt und die erforderlichen Schritte können durchgeführt werden.

1. Zunächst wird die Menge $V_x' \subseteq V_x$ aller Zeitwerte definiert, für die I_{RT} gültig ist:
$$V_x' = \{v_x' \in V_x | v_x' \geq 195ms\} \tag{5.27}$$

2. Die Menge V_x'' beinhaltet alle Werte, welche zum Zeitpunkt t I_{RT} erfüllen, jedoch unter Berücksichtigung des Alters der Werte. Zum Zeitpunkt t wird die Bedingung C(OVx) überprüft. Es kann davon ausgegangen werden, dass die Kamera das Bild innerhalb des Intervalls $[t - \Delta t_{xy}, t]$ aufgenommen hat. Somit gilt:
$$V_x'' = \{v_x'' \in V_x | v_x'' \geq 195ms + 66ms = 261ms\} \tag{5.28}$$
Die Abschätzung, dass die Kugel nach $66ms$ auch entsprechend weniger Zeit benötigt, um das Zentrum der Platte zu erreichen, setzt voraus, dass die Beschleunigung in dieser Zeitspanne konstant bleibt. Da davon ausgegangen wird, dass diese durch das Auslenken der Platte in Richtung Nullstellung abnimmt, kann die Abschätzung als großzügig interpretiert werden.

3. Die Messfehler bei der Bestimmung der Kugelkoordinaten durch die Bildverarbeitungskomponente sind bisher nicht untersucht worden. Obwohl diese sicherlich vorhanden sind, wird der Einfachheit halber davon abgesehen. Alle Werte $ov_x' \in OV_x'$, welche von den physikalischen Werten $v_x \in V_x''$ stammen, erfüllen somit die Bedingung $ov_x' \geq 261ms$.

4. Schließlich lautet die Bedingung C (OVx >= 261).

Unter Berücksichtigung des Alters der Daten und der Reaktionszeit des Systems muss die Zeit, welche die Kugel bis zum Zentrum benötigt und innerhalb des Regelalgorithmus anhand der Kugelposition berechnet wird, größer oder gleich $261ms$ sein. Bei Unterschreitung der Zeitgrenze ist das System nicht in der Lage, die Kugel abzubremsen und der Versuch schlägt fehl.

Kapitel 6

Umsetzung des Regelalgorithmus zur automatischen Steuerung

Der folgende Teil beschäftigt sich mit der Umsetzung des Regelalgorithmus zur automatischen Steuerung der Wippe. Ziel ist es dabei, den Ball im Zentrum der Platte zu balancieren. Zunächst wird ein Entwurf des Regelalgorithmus entwickelt, welcher die Vorgehensweise des Regelungsprozesses definiert. Im Anschluss daran folgt die Erläuterung bezüglich der Implementierung und der Integration innerhalb des Softwaresystems der Wippe. Diese Vorgehensweise kann als allgemeingültiges Beispiel für das Erweitern beziehungsweise Ändern der Aufgabenkomponente angesehen werden.

6.1 Entwurf

Die Arbeitsweise des Regelalgorithmus wird im weiteren Verlauf anhand von Pseudocode verdeutlicht. Die Betrachtung der x-Achse ist ausreichend, da sich die entsprechende Vorgehensweise für die y-Achse analog darstellt. Vorausgesetzt wird, dass sich die Platte zu Beginn des Versuchs in Nullstellung befindet. Abbildung 6.1 zeigt, wie die Position der Kugel bestimmt wird.

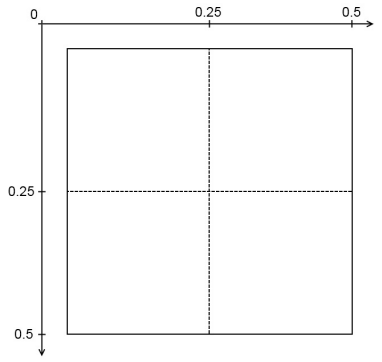

Abbildung 6.1: Koordinatensystem zur Bestimmung der Kugelposition.

Für die Berechnung der Kugelgeschwindigkeit wird die Kugelposition aus zwei aufeinander folgenden Durchgängen des Kontrollzyklus benötigt. Innerhalb des initialen Durchgangs können demnach lediglich die Variablen, welche die alte Kugelposition darstellen, initialisiert werden.

```
Wenn (istErsterDurchgang)
   istErsterDurchgang = false
   x_alt = x
```

Befindet sich das Programm bereits im zweiten Durchgang, so werden die Entfernung zum Zentrum der Platte (dszx), die Differenz der alten und aktuellen Kugelposition (dsx), sowie die Geschwindigkeit (vx) bestimmt.

```
Sonst
   dszx = 0.25 - x
   dsx = x - x_alt
   vx = dsx / 0.036
```

Für die Berechnung der Beschleunigung wird der bekannte Wert der Geschwindigkeit aus dem letzten Durchgang benötigt. Da dieser während des zweiten Durchganges noch nicht initialisiert ist, wird davon ausgegangen, dass die Kugel nicht beschleunigt wird. Diese Annahme ist legitim, da zu Beginn des zweiten Durchganges die Platte noch immer in Nullstellung ist.

```
   Wenn (istZweiterDurchgang)
      istZweiterDurchgang = false
      ax = 0
   Sonst
      ax = vx - vx_alt / 0.036
```

Anhand der in Abschnitt 5.1 hergeleiteten Zeitbedingung wird die Zeit dtz_x, welche die Kugel bis zum Erreichen des Plattenzentrums benötigt, verifiziert, so dass ein Abbremsen der Kugel realisierbar ist. Zunächst erfolgt jedoch die Überprüfung der Bewegungsrichtung. Sind die Vorzeichen von Geschwindigkeit und Abstand zum Plattenzentrum gleich, so bewegt sich die Kugel in die Richtung des Mittelpunktes und die Berechnung der Zeit kann korrekt durchgeführt werden.

```
   Wenn ( (vx > 0 && dszx > 0) || (vx < 0 && dszx < 0))
      Wenn (ax != 0)
         dtz_x = (sqrt( sqr(vx) + 2 * ax * dsz_x) - vx) / a_x
      Sonst
         dtz_x = dsz_x / vx
      Wenn (dtz_x < 0.266)
         Fehler "Kugel kann nicht balanciert werden!"
```

Wirkt keine Beschleunigung auf die Kugel, so wird dtz_x mit Hilfe des Geschwindigkeit-Zeit-Gesetzes einer gleichförmigen Bewegung berechnet (vgl. Abschnitt 3.1), im Übrigen mit der in Abschnitt 5.3 verwendeten Formel. Daran anschließend wird überprüft, ob die Zeit, welche die Kugel bis zum Zentrum der Platte benötigt, kleiner als die hergeleitete Zeitgrenze ist. Trifft dieser Fall zu, so ist das System nicht

in der Lage, die Kugel zu balancieren und der Versuch schlägt fehl.
Anders als bei dem in Abschnitt 3.1.3 vorgestellten Gedankenexperiment ist es erstrebenswert, die Kugel bereits vor dem Passieren des Zentrums abzubremsen. Der Winkel, welcher ein Abbremsen nahe des Mittelpunktes bewirkt, ist jedoch schwer zu bestimmen. Demnach gilt es zunächst einen nahezu vollkommenen Ruhezustand der Kugel an einer beliebigen Position der Platte zu erreichen. Erst wenn dieser Zustand realisiert ist, wird die Kugel in Richtung des Plattenzentrums gesteuert.
Innerhalb des Regelalgorithmus wird eine Geschwindigkeit, deren Betrag kleiner als $0,02\frac{m}{s}$ ist, als angenäherter Ruhezustand gewertet. In diesem Zustand soll lediglich eine Positionskorrektur vorgenommen werden. Dafür wird zunächst das Verhältnis des aktuellen Abstands und des maximal möglichen Abstands $(0,25m)$ zum Plattenzentrum gebildet. Das daraus resultierende Verhältnis wird mit einem noch zu bestimmenden Winkel multipliziert. Da für das Manövrieren der Kugel zum Plattenzentrum nur kleine Auslenkungen nötig sind, erscheint die Wahl des maximalen Winkels mit 15° als zu groß. Testläufe haben gezeigt, dass 5° ein geeigneter Wert ist, welcher die gegebene Zeitbedingung erfüllt. Damit die Auslenkung der Platte die Kugel in die korrekte Richtung beschleunigt, wird mit -5° multipliziert.

```
Wenn (abs(vx) < 0.02)
   alpha_x = (dszx / 0.25) * -5;
```

Um den Winkel zu bestimmen, welcher die Kugel zunächst abbremst, wird auf die Formel aus Abschnitt 3.1 zurückgegriffen. In Abschnitt 5.3 wurde bereits die maximale Anfangsgeschwindigkeit berechnet, welche die Kugel zu Beginn des Versuchs besitzen darf. Es wird angenommen, dass sie in diesem Fall unter Einwirkung der maximalen Beschleunigung $255ms$ bis zum Plattenzentrum benötigt. Die maximale Geschwindigkeit v_{max}, welche die Kugel beim Passieren des Zentrums besitzt, lässt sich folgendermaßen bestimmen:

$$v_{max} = a_{max} \cdot t + v_0 = 1,81\frac{m}{s^2} \cdot 255 \cdot 10^{-3}s + 0,75\frac{m}{s} = 1,21\frac{m}{s} \tag{6.1}$$

Dieser Wert kann als die Geschwindigkeit betrachtet werden, welche das System durch die maximale Auslenkung der Platte um 15° abbremsen kann. Geringeren Geschwindigkeiten können folglich kleinere Winkel zugeordnet werden. Denkbar ist eine lineare Zuordnungsvorschrift; jedoch widerlegt dies die folgende Herleitung:

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

Durch Einsetzen der bekannten Werte $\alpha = 15°$, $v_{max} = 1,21\frac{m}{s}$ und $t = 255ms$, ergibt sich folgender Zusammenhang:

$$k \cdot \arcsin\left(\frac{7 \cdot 1,21\frac{m}{s}}{5 \cdot 9,81\frac{m}{s^2} \cdot 255 \cdot 10^{-3}}\right) = k \cdot 42,62° = 15° \tag{6.3}$$

Für die Konstante k ergibt sich somit der Wert $0,35$. Mit der hieraus resultierenden Zuordnungsvorschrift lässt sich nun jede Geschwindigkeit $v \in [-1,21, 1,21]$ einem Winkel $\alpha \in [-15°, 15°]$ zuordnen, so dass die Kugel durch die Auslenkung der Platte entsprechend abgebremst wird.

$$\alpha = \arcsin\left(\frac{7 \cdot v}{5 \cdot g \cdot t}\right) \cdot 0,35 \tag{6.4}$$

Der folgende Code zeigt die Berechnung des Winkels anhand der hergeleiteten Formel und das abschließende Setzen von alter Geschwindigkeit und Kugelposition zur weiteren Verwendung innerhalb des nächsten Durchgangs.

```
   Sonst
      alpha_x = asin( vx / ( (5.0/7.0) * 9.81 * 0.255) ) * 0.35
   x_alt = x
   vx_alt = vx
```

6.2 Integration des Regelalgorithmus

Nachdem die prinzipielle Arbeitsweise des Regelalgorithmus spezifiziert ist, kann dieser innerhalb des Softwaresystems integriert werden. Speziell dafür muss die Aufgabenkomponente des Softwarekerns erweitert werden. Die im Folgenden beschriebene Vorgehensweise kann analog auf die verschiedenen Kontrolleinheiten angewendet werden. Der einzige Unterschied liegt in den zu implementierenden Methoden der abstrakten Klassen, welche voneinander abweichen.
Zunächst wird eine von der Klasse CAlgorithmControlUnit abgeleitete Klasse mit dem Namen CNewAlgorithmController erstellt. Aufgrund der Anwendung des Entwurfsmusters Singleton, muss ein Makro verwendet werden, welches die neue Klasse als Singleton definiert. Das Grundgerüst der Headerdatei sieht folgendermaßen aus:

```
class CNewAlgorithmController : public CAlgorithmControlUnit
{
   DECLARE_CHILD_SINGLETON(CNewAlgorithmController);

   private:
      CNewAlgorithmController(void);
      ~CNewAlgorithmController(void);

      void Calculate() = 0;
      bool m_isFirstRun, m_isSecondRun;
}
```

Das Makro DECLARE_CHILD_SINGLETON (vgl. Anhang A.4) generiert anhand des angegebenen Klassennamens die Deklaration der Singleton-Methoden und Singleton-Instanz:

```
public:
   static CNewAlgorithmController *GetInstance();
   static void Destroy();
private:
   static CNewAlgorithmController *m_instance;
```

Nun soll die statische Instanz der Klasse CNewAlgorithmController über die Singleton-Methode der Elternklasse verfügbar sein. Hierfür existiert ein weiteres Makro namens DECLARE_PARENT_SINGLETON (vgl. Anhang A.4), welches innerhalb der Headerdatei der Klasse CMotorControlUnit angewendet werden muss:

```
class CAlgorithmControlUnit : public tools::CThread, public
    CObservable<CAlgorithmData*>
{
   DECLARE_PARENT_SINGLETON(CAlgorithmControlUnit,
       CNewAlgorithmController, NewAlg);
   ...
```

Durch die Verwendung des Makros erhält man mittels einer Methode CAlgorithmControlUnit::GetInstanceNewAlg() eine Instanz der Klasse CAlgorithmControlUnit, welche intern jedoch eine Instanz der Klasse CNewAlgorithmController darstellt. Der generierte Quellcode sieht folgendermaßen aus:

```
friend class CNewAlgorithmController;
public:
   static CAlgorithmControlUnit *GetInstanceNewAlg();
   static void DestroyNewAlg();
private:
   static CAlgorithmControlUnit *m_instanceNewAlg;
```

Die Implementierung der Methoden und die Initialisierung der Singleton-Instanz wird durch den Aufruf des Makros IMPLEMENT_PARENT_SINGLETON (vgl. Anhang A.4) innerhalb der cpp-Datei der Klasse CAlgorithmControlUnit generiert:

```
IMPLEMENT_PARENT_SINGLETON(CAlgorithmControlUnit,
    CNewAlgorithmController, NewAlg)
```

Der letzte Schritt ist die Implementierung der Methoden, welche durch die abstrakte Klasse der Kontrolleinheit vorgegeben werden. Im Fall der Klasse CAlgorithmControlUnit muss lediglich die Methode Calculate() mit Befehlen gefüllt werden, welche dem Pseudocode aus Abschnitt 6.1 entsprechen (vgl. Anhang A.5). Durch die Verwendung des Makros IMPLEMENT_CHILD_SINGLETON (vgl. Anhang A.4) innerhalb der cpp-Datei der Klasse CNewAlgorithmController werden zusätzlich die Implementierung der Singleton-Methode, sowie die Initialisierung der Singleton-Instanz generiert.
Um den Regelalgorithmus nun anwenden zu können, muss für den entsprechenden Modus innerhalb des MainControllers eine Abfrage eingefügt werden, so dass der Vektor mit Instanzen der verschiedenen Kontrolleinheiten innerhalb des Schedulers an passender Stelle mit der Instanz der Klasse CAlgorithmControlUnit gefüllt wird. Dies kann beispielsweise wie folgt aussehen:

```
std::vector<tools::CThread*> threadQueue;
...
if (IsActive(MODE_AUTOMATIC_X_AXIS + MODE_AUTOMATIC_Y_AXIS))
{
    CAlgorithmControlUnit* newAlg = CAlgorithmControlUnit::
        GetInstanceNewAlg();
    threadQueue.push_back(newAlg);
    // Scheduler als Beobachter registrieren.
    newAlg->RegisterObserver(scheduler);
}
...
scheduler->SetThreadQueue(threadQueue);
scheduler->Start();
```

Um verschiedene Regelalgorithmen anzubieten, müssen entsprechend neue Definitionen für den zusätzlichen Modus angelegt werden (vgl. Listing 4.1).
Damit die in den Kapiteln zuvor getroffenen Annahmen bezüglich der Ausführungszeit der CAlgorithmControlUnit und der Analogie der Messwerte in Hinblick auf den in Abschnitt 5.1 durchgeführten Test im manuellen Betrieb der Wippe zu bestätigen, zeigt Tabelle 6.1 einen Ausschnitt der Messdaten eines 5-minütigen Tests des automatischen Betriebs unter Einsatz des erarbeiteten Regelalgorithmus. Dabei beträgt die durchschnittliche Rechenzeit der Regelungskomponente $111\mu s$.

Tabelle 6.1: Ausschnitt aus dem Datensatz des Testlaufs im automatischen Betrieb. **ZBK** - Rechenzeit der Bildverarbeitungskomponente, **KX/KY** - Kugelposition, **ZRK** - Rechenzeit der Regelungskomponente, **ZMK** - Rechenzeit der Motorkomponente, **MX/MY** - Eingestellte Motorenauslenkung.

ZBK	KX	KY	ZRK	ZMK	MX	MY
48408	184	86	94	-866	-2404	-600
30081	179	84	90	-2326	-3009	-1200
29286	176	83	100	-1523	-1801	-600
29214	172	82	215	-1822	-2404	-600
31635	168	81	106	-2036	-2404	-600
33421	165	80	88	-1125	-1801	-600
27925	162	79	91	-1684	-1801	-600
27420	158	78	82	-1988	-2404	-600
28080	156	77	90	-1100	-1200	-600
27269	153	76	85	-1487	-1801	-600
29591	151	75	107	-996	-1200	-600
31930	148	74	87	-1360	-1801	-600
28212	146	73	204	-1061	-1200	-600
45085	144	72	84	-436	-1200	-600

Kapitel 7

Zusammenfassung und Ausblick

Mit dieser Diplomarbeit ist das Wippe-Experiment der Arbeitsgruppe Echtzeitsysteme an der Universität Koblenz-Landau durch die Entwicklung eines neuen Softwaresystems und den Austausch von Hardwarekomponenten verbessert worden.
Zunächst werden der zu Beginn der Arbeit angetroffene Entwicklungsstand des Versuchsaufbaus Wippe und deren Funktionsweise sowie Informationen zu verwandten Systemen ausführlich vorgestellt. Die Darstellung der Funktionsweise umfasst neben der Beschreibung des technischen Aufbaus auch die Erläuterung der Struktur und Arbeitsweise des ursprünglich zum Steuern der Wippe eingesetzten Softwaresystems. In einem ersten Schritt zur Verbesserung des Wippe-Experiments wird die Wartung des technischen Systems vorgenommen. Dazu zählt der Austausch der Schrittmotoren zum Auslenken der Platte, die Portierung des Softwaresystems auf ein aktuelles Computersystem und damit einhergehend der Austausch der Kamera zur Aufnahme der Bilder.
In der Folge schließt sich die analytische Bewertung des Softwaresystems an. Es wird zunächst eine Liste mit funktionalen und nicht-funktionalen Anforderungen erarbeitet, um die bisher verwendete Software anhand der daraus resultierenden Kriterien zu analysieren und zu bewerten. Dabei werden, in Anbetracht der geplanten Verwendung im Rahmen der Ausbildung an Schulen und Universitäten, softwaretechnische Aspekte, wie beispielsweise Erweiterbarkeit und Änderbarkeit, berücksichtigt. Die Analyse des ursprünglichen Wippe-Systems führt zur Feststellung erheblicher Mängel, die die Notwendigkeit zur Konzipierung und Realisierung eines neuen Softwaresystems, angepasst an die erhobenen Anforderungen, begründet.
Die neue überarbeitete Software wird daraufhin einer Analyse im Hinblick auf die Eigenschaften eines harten Echtzeitsystems unterzogen. Es werden Ausführungszeiten der verschiedenen Arbeitsschritte des Rechensystems gemessen. Danach werden Zeitverzögerungen bestimmt, welche durch das externe System entstehen. Hierbei wird ein in dieser Arbeit neu entwickeltes Verfahren vorgestellt, um das Alter des Kamerabildes, durch welches die Position der Kugel bestimmt wird, zu messen. Auf der Basis dieser Analyse wird das Wippe-Experiment mit Hilfe der erarbeiteten Echtzeitbedingung verifiziert.
Abschließend wird unter Anwendung der gesammelten Informationen über das Sys-

tem der Entwurf eines Regelalgorithmus zur automatischen Steuerung der Wippe präsentiert. Ziel der Regelung ist es, die Kugel im Zentrum der Platte zu balancieren. Es folgt die Implementierung des Entwurfs und die Integration in das Softwaresystem.
Die zu Beginn aufgestellten Ziele sind im Verlauf dieser Arbeit vollständig gelöst worden. Mit dem Einsatz softwaretechnischer Mittel ist ein neues Softwaresystem entstanden, welches im Gegensatz zu seinem Vorgänger eine klare Struktur aufweist. Insbesondere ist eine neue grafischen Benutzerschnittstelle entstanden, welche hinsichtlich ihres Erscheinungsbildes und ihres softwaretechnischen Aufbaus dem gegenwärtigen Stand der Technik entspricht.
Aufbauend auf der Analyse der Wippe wird der Entwurf und die Implementierung eines Regelalgorithmus vorgestellt, welcher mit wenig Aufwand in das neue Softwaresystem integriert wird. Obwohl hierbei mit einem vereinfachten physikalischen Modell gearbeitet wird und die Regelung zum Teil auf Abschätzungen basiert, kann die Kugel automatisch im Zentrum balanciert werden. Die daraus gewonnene Erkenntnis ist, dass trotz einer verbleibenden Ungewissheit, die ihre Ursache im Vereinfachen und Abschätzen hat, ein funktionierender Regelungsprozess umgesetzt werden kann.
Aus physikalischer Sicht wäre eine Erweiterung des Gesamtmodells unter Einbeziehung der in dieser Arbeit vernachlässigten Eigenschaften (Reibung, Schlupf) denkbar. Eine spannende Aufgabe ergibt sich im Bereich der Softwaretechnik - besonders im Hinblick auf den Einsatz an Schulen und Universitäten - durch die Möglichkeit, die einzelnen Softwarekomponenten sprachunabhängig zu implementieren. Voraussetzung hierfür ist es eine allgemeingültige Kommunikationsschnittstelle (z.B. Datenaustausch basierend auf JSON[1] oder XML[2]) zwischen den Komponenten zu schaffen. Durch die zu erwartende Verschlechterung der Leistung, verursacht durch einen aufwändigeren Informationsaustausch zwischen den Komponenten, stellt sich die Frage, ob die hergeleitete Echtzeitbedingung eingehalten werden kann.
Bedingt durch seine Offenheit und leicht verständliche und vermittelbare Arbeitsweise stellt das Wippe-Experiment ein Beispiel dar, mit welchem die spezifischen Eigenschaften eines Echtzeitsystems dargelegt und verdeutlicht werden können.

[1] JavaScript Object Notation
[2] Extensible Markup Language

Anhang A

Quellcode

A.1 Initialisierung der Variable threadQueue des Schedulers innerhalb des MainControllers

Das aufgeführte Listing zeigt die Initialisierung der Variable threadQueue der Klasse CScheduler innerhalb der Methode StartCylce() der Klasse CMainController.

Listing A.1: Initialisierung der Variable threadQueue des Schedulers innerhalb des MainControllers.

```
void CMainController::StartCycle()
{
   std::vector<tools::CThread*> threadQueue;

   CImageProcessUnit *imageProcessUnit =
     CImageProcessUnit::GetInstance();

   CAlgorithmControlUnit *newAlgUnit =
     CAlgorithmControlUnit::GetInstanceNewAlg();

   CRemoteControlUnit *remoteControlUnit =
     CRemoteControlUnit::GetInstance();

   CMotorControlUnit *motorControlUnit =
     CMotorControlUnit::GetInstance();

   // Neue Instanz des Schedulers erstellen, falls noch nicht initialisiert.
   if (!m_scheduler)
      m_scheduler = new CScheduler();

   if (IsActive(MODE_AUTOMATIC_X_AXIS + MODE_AUTOMATIC_Y_AXIS))
   {
      // Falls automatischer Modus augewählt ist, Bildverarbeitungs-
      // komponente starten und zur threadQueue hinzufügen.
      imageProcessUnit->Start();
      threadQueue.push_back(imageProcessUnit);
      // Scheduler als Beobacheter registrieren.
      imageProcessUnit->RegisterObserver(scheduler);

      // Regelungskomponente zur threadQueue hinzufügen und
      // Scheduler als Beobachter registrieren.
      threadQueue.push_back(newAlgUnit);
```

```
        simplAlgUnit->RegisterObserver(scheduler);
    }

    if (IsActive(MODE_WII_X_AXIS + MODE_WII_Y_AXIS))
    {
        // Bei manuellem Modus Fernbedienungskomponente der threadQueue
        // hinzufügen und Scheduler als Beobachter registrieren.
        threadQueue.push_back(remoteControlUnit);
        remoteControlUnit->RegisterObserver(scheduler);
    }

    // Unabhängig vom Moduls zuletzt die Motorkomponente zur
    // threadQueue hinzufügen und Scheduler als Beobachter registrieren.
    motorControlUnit->RegisterObserver(scheduler);
    threadQueue.push_back(motorControlUnit);

    // threadQueue im Scheduler setzen und den Scheduler starten.
    scheduler->SetThreadQueue(threadQueue);
    scheduler->Start();
}
```

A.2 Hauptkontrollschleife innerhalb des Schedulers

Das aufgeführte Listing zeigt die Hauptkontrollschleife innerhalb der Methode Run() der Klasse CScheduler.

Listing A.2: Hauptkontrollschleife innerhalb des Schedulers.

```
void CScheduler::Run()
{
    CTimer timer;
    timer.Start();

    // Initialisierung.
    m_itCurrentThread = m_threadQueue.begin();
    (*m_itCurrentThread)->Start();
    m_isWaiting = true;
    m_isRunning = true;

    while (m_isRunning)
    {
        // waitTime für Timeout initialisieren. Dient der Deadlockverhinderung.
        ULONGLONG waitTime = 0;
        ULONGLONG now = CTimer::SystemTimeInMicros();
        while (m_isWaiting && m_isRunning && waitTime < SCHEDULER_TIMEOUT)
        {
            // Wartet, bis die Daten von der aktuellen Kontrolleinheit eintreffen.
            waitTime = CTimer::SystemTimeInMicros() - now;
        }
        // isWaiting wieder auf true setzen.
        m_isWaiting = true;

        // Iterationsschritt durchführen und nächsten Thread starten.
        BEGIN_CRITICAL_SECTION(m_semChangeThread)
        m_itCurrentThread++;
        if (m_itCurrentThread == m_threadQueue.end())
        {
            m_itCurrentThread = m_threadQueue.begin();
            timer.Stop();
```

```
            timer.Start();
        }
        (*m_itCurrentThread)->Start();
        END_CRITICAL_SECTION(m_semChangeThread)
    }
}
```

A.3 Verhalten des Schedulers beim Eintreffen einer Aktualisierung

Das gezeigte Listing veranschaulicht, wie die Klasse CScheduler auf die Aktualisierung eines Datenobjektes reagiert.

Listing A.3: Verhalten des Schedulers beim Eintreffen einer Aktualisierung.

```
void CScheduler::OnUpdate(void *sender, CImageData *data)
{
    UpdateScheduling(sender, data);
}

void CScheduler::OnUpdate(void *sender, CMotorData *data)
{
    UpdateScheduling(sender, data);
}

void CScheduler::OnUpdate(void *sender, CRemoteData *data)
{
    UpdateScheduling(sender, data);
}

void CScheduler::OnUpdate(void *sender, CAlgorithmData *data)
{
    UpdateScheduling(sender, data);
}

void CScheduler::UpdateScheduling(void *sender, CControlledData *data)
{
    // Ist der Scheduler nicht aktiv, Updates ignorieren.
    if (!m_isRunning) return;
    unsigned int senderPtr = (unsigned int) sender;
    unsigned int currentThrdPtr;

    BEGIN_CRITICAL_SECTION(m_semChangeThread)
    currentThrdPtr = (unsigned int) *m_itCurrentThread;
    END_CRITICAL_SECTION(m_semChangeThread)

    // Sender mit der aktiven Kontrolleinheit vergleichen.
    if (senderPtr == currentThrdPtr)
    {
        // Daten im Datenspeicher aktualisieren und nächsten Iterationsschritt
        // einleiten.
        BEGIN_CRITICAL_SECTION(m_semUpdateData)
        CDataCache::GetInstance()->UpdateData(data);
        m_isWaiting = false;
        END_CRITICAL_SECTION(m_semUpdateData)
    }
}
```

A.4 Definitionen der Makros bezüglich des Entwurfsmusters Singleton

Listing A.4: Definitionen der verschiedenen Makros bezüglich des Entwurfmusters Singleton.

```
// Deklariert Methoden und Attribut innerhalb der h-Datei einer Klasse, um diese
// als Singleton zu definieren.
#define DECLARE_CHILD_SINGLETON(Singleton) \
public: \
   static Singleton *GetInstance(); \
   static void Destroy(); \
private: \
   static Singleton *m_instance;

// Implementiert die Methoden und initialisiert das Attribut innerhalb der
// cpp-Datei einer Klasse, welche durch das Makro DECLARE_CHILD_SINGLETON
// deklariert wurden.
#define IMPLEMENT_CHILD_SINGLETON(Singleton) \
Singleton *Singleton::m_instance=0; \
\
Singleton *Singleton::GetInstance() \
{ \
   if (!m_instance) \
      m_instance = new Singleton(); \
   return m_instance; \
}

// Deklariert Methoden und Attribut innerhalb der h-Datei einer Klasse, um
// die Singleton-Instanz einer abgeleiteten Klasse über eine statische
// Methode zur Verfügung zu stellen, welche anhand einer Id unterscheidbar ist.
#define DECLARE_PARENT_SINGLETON(PrntSingleton, ChldSingleton, Id) \
friend class ChldSingleton; \
public: \
   static PrntSingleton *GetInstance ## Id(); \
   static void Destroy ## Id(); \
private: \
   static PrntSingleton *m_instance ## Id;

// Implementiert die Methoden und initialisiert das Attribut innerhalb der
// cpp-Datei einer Klasse, welche durch das Makro DECLARE_PARENT_SINGLETON
// deklariert wurden.
#define IMPLEMENT_PARENT_SINGLETON(PrntSingleton, ChldSingleton, Id) \
PrntSingleton *PrntSingleton::m_instance ## Id =0; \
\
PrntSingleton *PrntSingleton::GetInstance ## Id() \
{ \
   if (!m_instance ## Id) \
      m_instance ## Id= ChldSingleton::GetInstance(); \
   return m_instance ## Id; \
} \
\
void PrntSingleton::Destroy ## Id() \
{ \
   if (m_instance ## Id) \
      ChldSingleton::Destroy(); \
   m_instance ## Id = 0; \
}
```

A.5 Implementierung der Methode Calculate() der Klasse CNewAlgorithmController

Aufgrund der Übersichtlichkeit, wird nur die Implementierung der x-Achse dargestellt. Analog müssen lediglich Variablen und Berechnunen für die y-Achse eingefügt werden.

Listing A.5: Implementierung der Methode Calculate() der Klasse CNewAlgorithmController.

```
void CNewAlgorithmController::Calculate()
{
   // Initialisierung.
   bool isFailed = false;
   double x = 0, vx = 0, ax = 0, dsx = 0, dszx = 0, dtzx = 0, motorXPos = 0,
       alpha_x = 0;
   bool isBallFound = false;

   // Starten der Messung für die Berechnungsdauer.
   CTimer timer;
   timer.Start();

   // Bilddaten aus dem Datenspeicher lesen.
   CImageData *imageData = CDataCache::GetInstance()->GetData<CImageData>(
       ID_IMG_RETR_DATA);
   x = (double) (imageData->m_x / 260.0) * 0.5;
   y = (double) (imageData->m_y / 260.0) * 0.5;
   isBallFound = imageData->m_ballFound;

   // Wurde die Kugel nicht gefunden, wird nichts unternommen und die Werte neu
   // initialisiert.
   if (!isBallFound)
   {
      m_isFirstRun = m_isSecondRun = true;
      m_xOld = m_yOld = 0;
      m_vxOld = m_vyOld = 0;
   }
   else
   {
      // Unterscheidung des ersten Durchgangs.
      if (m_isFirstRun)
      {
         m_isFirstRun = false;
         m_xOld = x;
         m_yOld = y;
      }
      else
      {
         dszx = 0.25 - x;
         dsx = x - m_xOld;
         vx = dsx / 0.036;

         dszy = 0.25 - y;
         dsy = y - m_yOld;
         vy = dsy / 0.036;

         // Unterscheidung des zweiten Durchgangs, bei dem a=0 angenommen wird.
         if (m_isSecondRun)
         {
            m_isSecondRun = false;
```

```
            ax = 0;
        }
        else
        {
            // Berechnung der Beschleunigung anhand der Geschwindigkeitsänderung.
            ax = (vx - m_vxOld) / 0.036;
        }

        // Überprüfung, ob sich die Kugel in Richtung Plattenzentrum bewegt.
        if ( (vx > 0 && dszx > 0) || (vx < 0 && dszx < 0) )
        {
            if (ax != 0)
            {
                // Existiert eine Beschleunigung, dann die benötigte Zeit bis
                // zum Zentrum der Platte mit folgender Formel berechnen.
                dtzx = ( sqrt( pow(vx, 2.0) + 2 * ax * dszx) - vx) / ax;
            }
            else
            {
                // Gilt a=0 und v!=0, dann Zeit bis zum Zentrum mit der Formel
                // für eine gleichförmige Beweung berechnen.
                dtzx = dszx / vx;
            }
            if (dtzx < 0.266)
                // Ist die Kugel schneller als in 266ms im Plattenzentrum, so
                // schlägt der Versuch fehl.
                isFailed = true;
        }

        if (!isFailed)
        {
            if (abs(vx) < 0.02)
                // Bei geringer Geschwindigkeit Kugel Richtung Zentrum manövrieren.
                alpha_x = (dszx / 0.25) * -5;
            else
                // Bei höherer Geschwdindigkeit versuchen, Kugel zu bremsen.
                alpha_x = (180.0/3.14)*asin(vx/((5.0/7.0)*9.81*0.255))*0.35;
        }
    }
  }
  // Stoppen des Timers für die Messung der Rechenzeit.
  timer.Stop();

  // Die Winkel in Motorticks umrechnen.
  alpha_x = (alpha_x / 15.0) * 15000;

  // Füllen der Algorithmus-Daten und Beobachter benachrichtigen.
  if (!m_currentAlgData)
    m_currentAlgData = new CAlgorithmData(ID_SIMPL_ALG_DATA, xAcceleration,
        yAcceleration,
      xVelocity, yVelocity, (int) calcMotorXPos, (int) calcMotorYPos, calcTime)
          ;
  else
    m_currentAlgData->SetAlgorithmData(xAcceleration, yAcceleration, xVelocity,
        yVelocity,
      (int) calcMotorXPos, (int) calcMotorYPos, calcTime);
  FireUpdateEvent(this, m_currentAlgData);
}
```

Anhang B

Klassen-Dokumentation

B.1 Dokumentation der Klassen der „Protokollkomponente“

Tabelle B.1: Dokumentation der Klasse CLogger der Protokollkomponente.

Attribut/Methode	**Erläuterung**
mpLogger	Statische Map, welche als Schlüssel die String-Repräsentation einer Klasse enthält und als Wert die zugehörige Logger-Instanz.
vectLogWriter	Statischer Vektor, welcher die registrierten Log-Writer verwaltet.
clazz	String-Repräsentation der Klasse, zu welcher der Logger gehört.
GetInstance()	Liefert die Singleton-Instanz einer Klasse, welche durch den generischen Parameter T spezifiziert wird.
Destroy()	Ruft den Konstruktor aller Instanzen auf, welche in der Map mpLogger enthalten sind.
RegisterWriter()	Registriert einen LogWriter, welcher die Log-Meldungen ausgibt.
UnregisterWriter()	Löscht einen bereits registrierten LogWriter.
Info()	Gibt eine Informations-Nachrich über alle registrierten LogWriter aus.
Waring()	Gibt eine Warnungs-Nachrich über alle registrierten LogWriter aus.
Error()	Gibt eine Error-Nachrich über alle registrierten LogWriter aus.

Tabelle B.2: Dokumentation der Klasse CLogWriter der Protokollkomponente.

Methode	Erläuterung
WriteLog()	Zu implementierende Methode, um die Art der Ausgabe der übergebenen Log-Nachrichten zu definieren.

Tabelle B.3: Dokumentation der Klasse CFileWriter der Protokollkomponente.

Attribut	Erläuterung
fileStream	Datei-Stream, um ankommende Log-Nachrichten in einer Datei auszugeben.

B.2 Dokumentation der Klassen des „Datenspeichers“

Tabelle B.4: Dokumentation der Klasse CDataCache des Datenspeichers.

Attribut/Methode	Erläuterung
timeStamp	Zeitstempel, wann die Attribute des Datums zuletzt aktualisiert wurden.
calcTime	Zeitspanne die benötigt wurde, um die verschiedenen Attribute des Datums zu berechnen.
data	Map, in welchem die zu speichernden Objekte, abgeleitet von der Klasse CControlledData, als Wert abgelegt sind. Der Schlüssel ist dabei das Attribut id des entsprechenden Objekts.
GetInstance()	Liefert die Singleton-Instanz des CDataCache.
Destroy()	Ruf den Destruktor der Singleton-Instanz auf.
UpdateData()	Aktualisiert ein zu speicherndes CControlledData-Objekt innerhalb der Map data.
GetData()	Liefert ein Datenobjekt aus der Map data anhand der angegeben id.

Tabelle B.5: Dokumentation der Klasse CControlledData des Datenspeichers.

Attribut/Methode	**Erläuterung**
timeStamp	Zeitstempel, wann die Attribute des Datums zuletzt aktualisiert wurden.
calcTime	Zeitspanne die benötigt wurde, um die verschiedenen Attribute des Datums zu berechnen.
id	Identifikator, welcher den entsprechenden Datentyp eindeutig identifiziert (z.B. “algorithmData“ für den Datentyp CAlgorithmData).
GetId()	Liefert den Identifikator des Datentyps zurück.

Tabelle B.6: Dokumentation der Klasse CAlgorithmData des Datenspeichers.

Attribut	**Erläuterung**
xAcceleration	Beschleunigung der Kugel in x-Richtung.
yAcceleration	Beschleunigung der Kugel in y-Richtung.
xVelocity	Geschwindigkeit der Kugel in x-Richtung.
yVelocity	Geschwindigkeit der Kugel in y-Richtung.
calcMotorXPos	Berechnete Motorposition bezüglich der x-Achse.
calcMotorYPos	Berechnete Motorposition bezüglich der y-Achse.

Tabelle B.7: Dokumentation der Klasse CMotorData des Datenspeichers.

Attribut	**Erläuterung**
posX	Zuletzt eingestellte Motorposition bezüglich der x-Achse.
posY	Zuletzt eingestellte Motorposition bezüglich der y-Achse.

Tabelle B.8: Dokumentation der Klasse CRemoteData des Datenspeichers.

Attribut	**Erläuterung**
calcMotorXPos	Die mit Hilfe einer Fernbedienung (z.B. WiiRemote) berechnete Motorposition bezüglich der x-Achse.
calcMotorYPos	Die mit Hilfe einer Fernbedienung (z.B. WiiRemote) berechnete Motorposition bezüglich der y-Achse.
firedEvents	Integer-Wert, welcher bestimmte Events definiert, welche durch die Fernbedienung ausgelöst werden können (z.B. Starten des Versuchs).

Tabelle B.9: Dokumentation der Klasse CImageData des Datenspeichers.

Attribut	**Erläuterung**
image	Unbearbeitetes, aktuelles Kamerabild.
ballImage	Aktuelles Kamerabild mit eingezeichneter Kugelposition und dem Kugelpfad.
ballFound	Boolean, welcher angibt, ob im aktuellen Kamerabild die Kugel gefunden wurde.
ballX	x-Koordinate der Kugelposition.
ballY	y-Koordinate der Kugelposition.
captureTime	Zeitstempel, welcher den Zeitpunkt des Eintreffens des aktuellen Kamerabildes im Programm darstellt.

B.3 Dokumentation der Klassen der „Aufgabenkomponente“

Tabelle B.10: Dokumentation der Klasse CThread der Aufgabenkomponente.

Methode	**Erläuterung**
Start()	Startet einen nebenläufigen Prozess. Der Prozess führt die Befehle innerhalb der Methode Run() aus.
ShouldTerminate()	Versucht den Thread zu beenden. Falls der Vorgang erfolgreich war, wird „true“ zurückgegeben, ansonsten „false“.
GetStatus()	Liefert den Status des Threads in Form eines Integer-Wertes zurück. Dabei sind folgende Werte möglich: • 0 - Methode Run() verlassen und beendet. • -1 - Thread-Objekt ist erstellt, aber noch nicht gestartet. • >1 - Thread befindet sich innerhalb der Methode Run() und läuft.
Run()	Abstrakte Methode, welche die Befehle enthält, die beim Aufruf von der Methode Start() ausgeführt werden.

Tabelle B.11: Dokumentation der Klasse CObservable<T> der Aufgabenkomponente.

Attribut/Methode	**Erläuterung**
Generischer Parameter T	Gibt den Klassentyp des Datums an, welches beobachtet wird.
RegisterObserver()	Registriert einen Beobachter für die Benachrichtigung einer Aktualisierung eines bestimmten Datums.
UnregisterObserver()	Entfernt einen bereits registrierten Beobachter aus der Liste der registrierten Beobachter.
UnregisterAllObserver()	Entfernt alle bereits registrierten Beobachter aus der Liste der registrierten Beobachter.
FireUpdateEvent()	Benachrichtigt alle registrierten Beobachter über die Aktualisierung des beobachteten Datums.

Tabelle B.12: Dokumentation der Klasse CMotorControlUnit der Aufgabenkomponente.

Attribut/Methode	**Erläuterung**
motorData	Enthält die berechneten Motordaten.
GetInstance()	Liefert eine Instanz der Klasse, welche jedoch eine Instanz einer abgeleiteten Klasse ist.
Destroy()	Ruft den Destruktor der Instanz der abgeleiteten Klasse auf.
SetTargetPosition()	Abstrakte Methode, welche für die Implementierung zum Setzen der Motorpositionen verwendet wird.
IsConnected()	Liefert einen booleschen Wert zurück, welcher den Status der Verbindung ausdrückt.
SendCommand()	Abstrakte Methode, welche für die Implementierung zum Senden eines Befehls an die Motorenschnittstelle verwendet wird.

Tabelle B.13: Dokumentation der Klasse CAlgorithmControlUnit der Aufgabenkomponente.

Attribut/Methode	**Erläuterung**
targetX	Motorposition der x-Achse, welche beim nächsten Aufruf der Methode Start() der Elternklasse CThread an das Treibermodul gesendet wird.
targetY	Motorposition der y-Achse, welche beim nächsten Aufruf der Methode Start() der Elternklasse CThread an das Treibermodul gesendet wird.
algorithmData	Enthält die berechneten Regelungsdaten.
GetInstance()	Liefert eine Instanz der Klasse, welche jedoch eine Instanz einer abgeleiteten Klasse ist.
Destroy()	Ruft den Destruktor der Instanz der abgeleiteten Klasse auf.
Calculate()	Abstrakte Methode, in welche für die Implementierung zum Berechnen und Erstellen der Regelungsdaten benutzt wird.

Tabelle B.14: Dokumentation der Klasse CImageProcessUnit der Aufgabenkomponente.

Attribut/Methode	**Erläuterung**
imageData	Bilddaten, welche innerhalb der Klasse berechnet und erzeugt werden.
GetInstance()	Liefert eine Instanz der Klasse, welche jedoch eine Instanz einer abgeleiteten Klasse ist.
Destroy()	Ruft den Destruktor der Instanz der abgeleiteten Klasse auf.
Calibrate()	Abstrakte Methode, welche für die Implementierung in Bezug auf die Kalibrierung des Kamerabildes benutzt wird.
ClearTrace()	Abstrakte Methode, welche für die Implementierung für das Löschen des Kugelpfades verwendet wird, der auf dem bearbeiteten Bild der Bilddaten eingezeichnet wird.
ProcessImage()	Abstrakte Methode, welche für die Implementierung zum Berechnen und Erstellen der Bilddaten benutzt wird.

Tabelle B.15: Dokumentation der Klasse CRemoteControlUnit der Aufgabenkomponente.

Attribut/Methode	**Erläuterung**
remoteData	Enthält die berechneten Fernbedienungsdaten.
GetInstance()	Liefert eine Instanz der Klasse, welche jedoch eine Instanz einer abgeleiteten Klasse ist.
Destroy()	Ruft den Destruktor der Instanz der abgeleiteten Klasse auf.
Connect()	Abstrakte Methode, welche für die Implementierung zum Verbinden der Fernbedienung mit dem Computer genutzt wird.
IsConnected()	Liefert einen booleschen Wert zurück, welcher den Status der Verbindung ausdrückt.
Disconnect()	Abstrakte Methode, welche für die Implementierung zum Trennen der Verbindung zwischen Fernbedienung und Computer genutzt wird.
Listen()	Abstrakte Methode, welche für die Implementierung zum Berechnen und Erstellen der Fernbedienungsdaten genutzt wird.

B.4 Dokumentation der Klassen der „Steuerungskomponente“

Tabelle B.16: Dokumentation der Klasse CMainController der Steuerungskomponente.

Attribut/Methode	**Erläuterung**
mode	Definiert den Modus, in welchem die Wippe betrieben wird.
isRunning	Boolean, welcher angibt, ob die Wippe aktuell im Betrieb ist.
scheduler	Instanz des Schedulers.
instance	Singleton-Instanz.
GetInstance()	Liefert die Singleton-Instanz.
Destroy()	Ruft den Destruktor der Singleton-Instanz auf.
StartCycle()	Startet den Betrieb der Wippe.
StopCycle()	Stoppt den Betrieb der Wippe.
SetMode()	Setzt den übergebenen Betriebsmodus.
IsRunning()	Liefert den aktuellen Wert von isRunning.
IsActive()	Gibt true zurück, wenn der übergebene Modus als Betriebsmodus angegeben ist.

Tabelle B.17: Dokumentation der Klasse CScheduler der Steuerungskomponente.

Attribut/Methode	**Erläuterung**
threadQueue	Vektor mit Instanzen der Klasse CThread, welche die Kontrolleinheiten darstellen. Die Reihenfolge entspricht der Reihenfolge der Arbeitsschritte während der Regelung.
itCurrentThread	Referenziert die aktive Kontrolleinheit.
isRunning	Boolean, welcher angibt, ob der Scheduler aktiv ist oder nicht.
isWaiting	Boolean, welcher angibt, ob der Scheduler auf den Empfang der Daten einer Kontrolleinheit wartet.
SetThreadQueue()	Initialisiert den Vektor mit Threads.
StopRunning()	Stoppt den Scheduler.
OnUpate()	Update-Methoden, welche von der Oberklasse CObserver vorgegeben werden.
UpdateScheduling()	Überprüft, ob Daten der aktiven Kontrolleinheit empfangen wurden, um die nächste Kontrolleinheit zu aktivieren.

Anhang C

Screenshots der neuen grafischen Benutzerschnittstelle

Abbildung C.1: Screenshot der grafischen Benutzerschnittstelle mit ausgewählter Registerkarte „Startseite“.

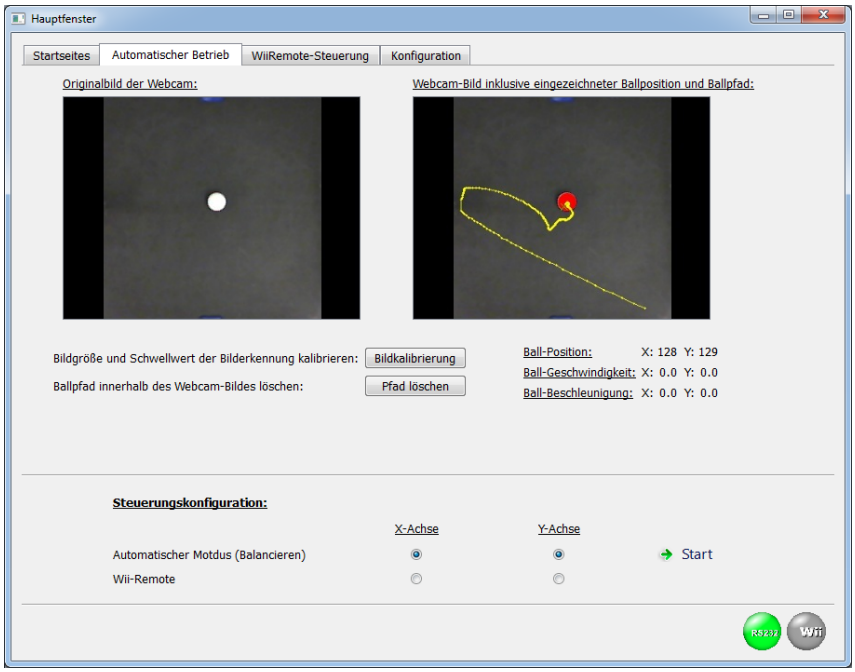

Abbildung C.2: Screenshot der grafischen Benutzerschnittstelle mit ausgewählter Registerkarte „automatischer Betrieb“.

Abbildung C.3: Screenshot des Dialogs zum Konfigurieren des Kamerabildes.

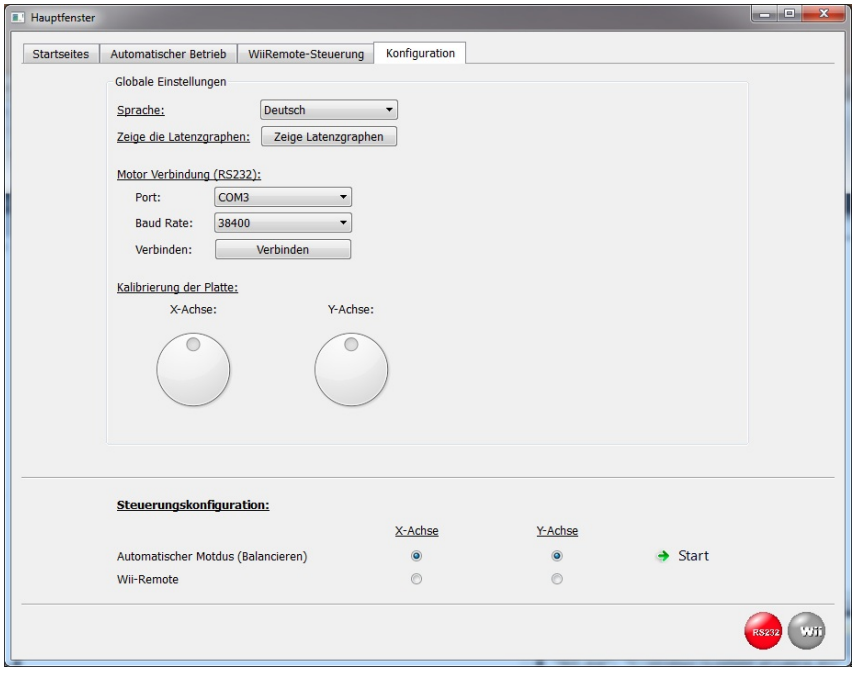

Abbildung C.4: Screenshot der grafischen Benutzerschnittstelle mit ausgewählter Registerkarte „Konfiguration“.

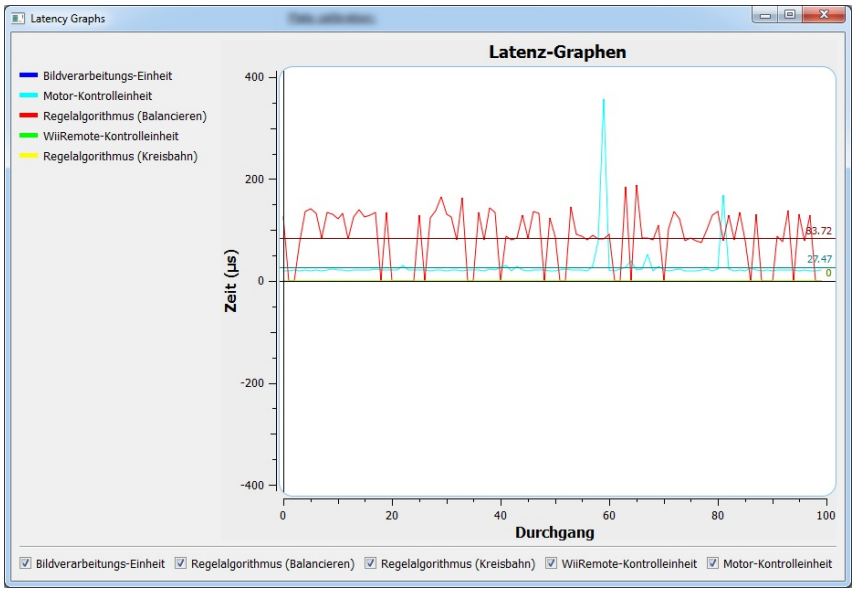

Abbildung C.5: Screenshot des Dialogs zum Darstellen von Latenzzeiten der einzelnen Kontrolleinheiten.

Literaturverzeichnis

[ABB+02] S. Awtar, C. Bernard, N. Boklund, A. Master, D. Ueda, and K. Craig. Mechatronic design of a ball-on-plate balancing system. *Mechatronics*, 12(2):217–228, 2002.

[ACH04] G. Andrews, C. Colasuonno, and A. Herrmann. Ball on plate balancing system progress report for ecse-4962 control systems design. 2004.

[Atm11a] Atmel. *Atmega16*, 2011. Erreichbar unter `http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf`, Abgerufen am 19.06.2011.

[Atm11b] Atmel. *Atmel Evaluations-Board Version 2.0.1*, 2011. Erreichbar unter `http://www.pollin.de/shop/downloads/D810074B.PDF`, Abgerufen am 19.06.2011.

[Bal00] Helmut Balzert. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag GmbH - Heidelberg/Berlin, 2000.

[Bat04] Bert Bates. *Head First - Design Patterns*. O'Reilly Media, 2004.

[BD00] F. Bader and F. Dorn. *Physik - Gymnasium Gesamtband, Sek II*. Schroedel Verlag GmbH, Hannover, 2000.

[CGGS00] G. C. Goodwin, S. Greabe, and M. Salgado. *Control System Design*. Addison Wesley Pub Co Incl, Englewood Cliffs, NJ, 2000.

[fN85] Deutsches Institut für Normung. Informationsverarbeitung - begriffe, din44300, 1985.

[FPWW03] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Hough Transform. Webseite, 2003. Erreichbar unter `http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm`, Abgerufen am 24.04.2011.

[FZT04] X. Fan, N. Zhang, and S. Teng. Trajectory planning and tracking of ball and plate system using hierarchical fuzzy control scheme. *Fuzzy Sets and Systems*, 144(2):297–312, 2004.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[Hum11] Yohan Humbert. Realisierung einer geeigneten Mensch-Maschine-Schnittstelle für die manuelle Steuerung des Wippe-Experiments. Bachelor-arbeit, Universität Koblenz-Landau, Institut für Informatik: AG Echtzeitsysteme, 2011.

[Nok11] Nokia. Qt - cross platform application und ui framework. Webseite, 2011. Erreichbar unter `http://opencv.willowgarage.com/wiki/`, Abgerufen am 19.07.2011.

[Ope11] OpenCV-Community. OpenCV. Webseite, 2011. Erreichbar unter `http://opencv.willowgarage.com/wiki/`, Abgerufen am 18.07.2011.

[Tec11] TecQuipment. CE106. Webseite, 2011. Erreichbar unter `http://www.tecquipment.com/Datasheets/CE106_1210.pdf`, Abgerufen am 20.05.2011.

[Tri11] Trinamic. *TMCM-310, 3-Achsen Schrittmotor Steuerungs- und Treibermodul*, 2011. Erreichbar unter `http://www.trinamic.com/tmc/media/Downloads/modules/TMCM-310/TMCM-310_TMCL_firmware_manual.pdf`, Abgerufen am 22.05.2011.

[Ust10] Mehmet-Sefa Usta. Modellierung, Implementierung und Dokumentation der Mensch-Maschine-Schnittstelle, sowie der Ein-/Ausgabe-Komponenten des Wippe-Experminents. Bachelor-arbeit, Universität Koblenz-Landau, Institut für Informatik: AG Echtzeitsysteme, 2010.

[Zö98] Dieter Zöbel. A versatile real-time experiment: Balancing a ball on a flat board. In *Third IEEE Real-Time Systems Education Workshop (RTEW'98)*, pages 98–105, Poznan, Poland, November 1998.

[Zö05] Dieter Zöbel. Canonical approach to derive and enforce real-time conditions. In *1st International ECRTS Workshop on Real-Time and Control (RTC 2005)*, Palma de Mallorca, July 2005.

[Zö08] Dieter Zöbel. *Echtzeitsysteme - Grundlagen der Planung*. Springer-Verlag, Berlin, 2008.