

Projekt Kugellabyrinth

STUDIENARBEIT

des Studienganges Mechatronik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Max Zimmermann, Sascha Steger, Jan Koopmann

Abgabedatum 30.04.2020

Bearbeitungszeitraum 31 Wochen (30.09.2019 – 30.04.2020)

Matrikelnummern, Kurs 3402058, 1400512, 3663886

TMT17AM2

Ausbildungsfirma Pepperl+Fuchs AG,

Bundeswehr,

SCHENCK RoTec GmbH

Betreuer Herr Prof. Dr.-Ing. Stefan Werling

Unterschrift Betreuer

Ehrenwörtliche Erklärung

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die

Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg

vom 29.09.2017.

Wir versichern hiermit, dass wir diese Studienarbeit mit dem Thema:

„Projekt Kugellabyrinth“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Eppelheim, den 30.04.2020

Max Zimmermann

Sascha Steger

Jan Koopman

Vorwort

Diese Studienarbeit ist im Zeitraum vom 30.09.2019 bis zum 30.04.2020 entstanden. Der Projektaufbau und sämtliche Versuche zu dieser Studienarbeit wurden in den Laboren der Zweigstelle Eppelheim der DHBW Mannheim durchgeführt.

Aufgrund der raschen deutschlandweiten Ausbreitung des Corona-Virus im Frühjahr 2020 wurde am 15.03.2020 der Präsenzbetrieb an der DHBW Mannheim und somit auch an der Zweigstelle Eppelheim eingestellt.

Da diese Maßnahmen uns sehr abrupt getroffen haben, war es leider nicht möglich alle für die schriftliche Ausarbeitung notwendigen Daten sicher zu stellen, beziehungsweise für die Dokumentation relevante Fotos und Screenshots von dem Aufbau des Modells oder den Versuchen zu machen.

Aus diesem Grund sind die verwendeten Fotos des Modellaufbaus aus der Studienarbeit des Vorgängerjahrgangs entnommen und mit der entsprechenden Quellenangabe versehen worden. Des Weiteren sind viele visuelle Darstellungen von Ergebnissen und Versuchen auf den Rechnern der DHBW gespeichert, auf die kein Zugriff mehr möglich war.

Kurzfassung

In dieser Studienarbeit mit dem Titel „Projekt Kugellabyrinth“ wird die in den Ingenieurwissenschaften bekannte Thematik „Ball on Plate“ behandelt. Das Ziel der Arbeit ist sowohl die statische als auch die dynamische Regelung einer Kugel auf einer Platte. Die dynamische Regelung soll vorerst in Form einer Kreisbahn erfolgen. Hierfür wird ein bereits bestehender Modellaufbau aus einer vorangegangen Studienarbeit übernommen. Die gesamte softwaretechnische Umsetzung wird neu erstellt und erfolgt in der Programmiersprache Python. Der Schwerpunkt der Studienarbeit liegt auf der eigens entworfenen Regelung, sowie der Implementierung einer Bildverarbeitung zur zuverlässigen Detektion von Ball und Platte.

Zu Beginn wird der bereits fertige mechanische Aufbau genauer erläutert, welcher zusammen mit der Aufgabenstellung die Rahmenbedingungen festlegt. Mit übernommen werden zwei Servomotoren, welche die Platte steuern, ein Raspberry Pi zur Ansteuerung der Motoren, sowie eine Webcam zur Zustandserfassung. Hinzugekommen ist ein Desktop-PC, auf den die gesamten Berechnungen der Software ausgelagert werden.

Der anschließende Teil befasst sich mit der Theorie der Regelungstechnik und erläutert ausführlich, wie auf Basis einer hergeleiteten Differentialgleichung über ein Zustandsraummodell die verwendete Zustandsregelung berechnet wird. Mittels der Zustandsregelung wird auf Basis von Regelmatrix, Vorfilter, Zustandsvektor und Sollwertvorgabe der Plattenwinkel berechnet.

Der Fokus der Software liegt auf der Bildverarbeitung. Zu Beginn werden die verwendeten Bibliotheken erläutert und es wird ein Überblick über das Programm gegeben. Anschließend erfolgt in der Bildverarbeitung zu jedem Schritt eine detaillierte Erklärung der verwendeten Algorithmen oder Filter sowie die jeweilige Implementierung im Programmcode. Die Bildaufnahme erfolgt mittels einer selbst geschriebenen Webcam-Klasse, in welcher jedes von der Kamera gesendete Bild mit einem Zeitstempel versehen wird. Der anschließende Teil der Bildvorverarbeitung beschreibt die Bildglättung mittels eines Binomialfilters und die Transformation in den

HSV-Farbraum. Besonders das Kapitel der Plattendetektion soll einen tiefen Einblick in die Möglichkeiten der Bildverarbeitung bieten. Es wird erläutert, wie mithilfe von verschiedenen Bildverarbeitungsalgorithmen sowohl die Ecken der Platte als auch die Position des Balls präzise erkannt werden. Das Kapitel der perspektivischen Transformation erläutert, wie auf Basis der verzerrten Kameraperspektive die Ballkoordinaten relativ zur Platte bestimmt werden können.

Mit den aus der Bildverarbeitung gewonnenen Koordinaten und den Zeitstempeln aus der Webcam-Klasse kann die aktuelle Ballgeschwindigkeit berechnet werden. Sie bildet zusammen mit den Ballkoordinaten die Grundlage für die Umsetzung der Regelung.

Es folgt die mathematische Herleitung der Approximationsfunktion, die den Zusammenhang zwischen der Pulsdauer des PWM-Signals zur Steuerung der Servomotoren und dem Plattenwinkel erklärt. Das PWM-Signal wird von dem Raspberry-Pi erzeugt, welcher über den Desktop-PC ferngesteuert wird. Die PWM-Signalerzeugung sowie die Fernsteuerung werden ausführlich beschrieben. Da die Rechenkapazitäten des Raspberry Pis nicht ausreichen, werden die gesamte Bildverarbeitung und Regelung auf einen Desktop-PC ausgelagert. Die PWM-Signalerzeugung erfolgt weiterhin über den Raspberry Pi, welcher über den PC ferngesteuert wird. Es werden die Fernsteuerung des Raspberry Pis sowie die Erzeugung des PWM-Signals erläutert.

Da es zwischenzeitlich zu sprunghaften Änderungen der Plattenwinkel kam, werden verschiedene Schritte zur Performanceverbesserung behandelt, mit welchen das Problem gelöst wird.

Abstract

The present student research project titled "Project Ball Labyrinth" covers the topic "Ball on Plate", which is well known in the engineering sciences. The aim of the thesis is both the static and dynamic control of a ball on a plate. For the time being, the dynamic control should take the form of a circular path. For this purpose, an already existing model structure from a previous student research project is used. The entire software implementation is newly created and is done in the programming language Python. The focus of this project is the specially designed control system and the implementation of an image processing system for a reliable detection of the ball and the plate.

At first the already finished mechanical construction, which includes two servo motors, which control the plate, a Raspberry Pi for controlling the motors and a webcam for status detection, is explained in detail defining the basic conditions together with the task. A desktop PC, to which the entire software calculations are outsourced, has been added to the construction.

The following part approaches the theory of control engineering and explains in detail how the state control used is calculated on the basis of a derived differential equation via a state space model. By means of the state control, the plate angle is calculated using the control matrix, pre-filter, state vector and setpoint specification.

The software is focused on image processing. At the beginning the used libraries are explained and an overview of the program is given. Afterwards a detailed explanation of the used algorithms or filters as well as the respective implementation in the program code is given for each step in the image processing. The image acquisition is done by means of a self-written webcam class, in which each image sent by the camera is timestamped. The following part of the image preprocessing describes the image smoothing by means of a binomial filter and the transformation into the HSV color space. The chapter concerning the plate detection in particular is intended to provide a deep insight into the possibilities of image processing; it explains how different image processing algorithms are used to precisely detect the corners of the plate and the position of the ball. The chapter related to perspective transformation explains how the ball coordinates

in relation to the plate can be determined on the basis of the distorted camera perspective.

The current ball speed can be calculated with the coordinates obtained from image processing and the time stamps from the webcam class. Together with the ball coordinates it forms the basis for the implementation of the control.

Then the mathematical derivation of the approximation function, which explains the relationship between the pulse duration of the PWM signal for controlling the servo motors and the plate angle, is illustrated. The PWM signal is generated by the Raspberry Pi remotely controlled via the desktop PC. Since the computing capacities of the Raspberry Pi are not sufficient, the entire image processing and control are outsourced to a desktop PC. The PWM signal generation is still done by the Raspberry Pi. The remote control of the Raspberry Pi and the generation of the PWM signal are explained.

Since there have been some sudden changes in the plate angles, different steps to improve the performance will be discussed to solve the problem.

Inhaltsverzeichnis

VORWORT	III
ABBILDUNGSVERZEICHNIS.....	X
TABELLENVERZEICHNIS	XII
ABKÜRZUNGSVERZEICHNIS.....	XIII
EINHEITENVERZEICHNIS	XIV
1. EINFÜHRUNG.....	1
1.1. MOTIVATION.....	1
1.2. VORARBEITEN.....	1
1.3. AUFGABENSTELLUNG	2
1.4. STAND DER TECHNIK.....	2
2. MODELLAUFBAU.....	5
2.1. MECHANISCHER AUFBAU.....	5
2.2. ELEKTRISCHE KOMPONENTEN.....	8
2.2.1. Kamera	8
2.2.2. Servomotoren	9
2.2.3. Raspberry Pi.....	12
3. ZUSTANDSREGELUNG.....	16
3.1. AUFSTELLEN DES ZUSTANDSRAUMMODELLS	16
3.2. ENTWURF DES ZUSTANDSREGLERS	20
4. SOFTWARE	25
4.1. VERWENDETE BIBLIOTHEKEN.....	25
4.1.1. openCV	25
4.1.2. Numpy.....	26
4.1.3. pigpio	26
4.1.4. Matplotlib.....	26
4.1.5. Weitere verwendete Bibliotheken	27
4.2. PROGRAMMSTRUKTUR.....	27
4.3. PROGRAMMABLAUF	28
4.4. PROGRAMMINITIALISIERUNG	31
4.5. BILDVERARBEITUNG	33

4.5.1.	Bildaufnahme	35
4.5.2.	Bildvorverarbeitung	39
4.5.3.	Plattendetektion.....	49
4.5.4.	Balldetektion	69
4.5.5.	Perspektivische Transformation.....	71
4.6.	BERECHNUNG DER PLATTENSTELLUNG.....	79
4.6.1.	Berechnung der Ballgeschwindigkeit.....	79
4.6.2.	Programmimplementierung der Regelung	81
4.7.	ANSTEUERUNG DER SERVOMOTOREN.....	83
4.7.1.	Approximation	84
4.7.2.	Optimierung der Approximation.....	86
4.7.3.	Erzeugung des PWM-Signals per Fernzugriff	86
4.8.	PERFORMANCEVERBESSERUNG.....	93
5.	FAZIT UND AUSBLICK	98
5.1.	FAZIT	98
5.2.	AUSBLICK.....	99
	LITERATURVERZEICHNIS	102
	ANHANGSVERZEICHNIS.....	106

Abbildungsverzeichnis

Abbildung 1: Projektaufbau „Ball on Plate“ TU Darmstadt	3
Abbildung 2: CAD-Modell mit Beschriftung.....	5
Abbildung 3: Verbindung zwischen Servomotor und Balancierplatte	6
Abbildung 4: Kamerabefestigung mit Beleuchtung	8
Abbildung 5: Logitech Webcam C930e	9
Abbildung 6: Regelkreis eines Servomotors	10
Abbildung 7: Beispielhafte PWM-Signale	11
Abbildung 8: Raspberry Pi 3 mit Beschriftung	14
Abbildung 9: Freischnitt Ball auf Platte	17
Abbildung 10: Zustandsregelung.....	21
Abbildung 11: Flussdiagramm Programmablauf.....	29
Abbildung 12: Hierarchischer Ablauf der Bildverarbeitung	34
Abbildung 13: Anwendung eines Binomialfilters	39
Abbildung 14: Arbeitsweise eines Linearen Filter	40
Abbildung 15: Durch drehen der Filtermatrix um 180° macht aus einer Kreuzkorrelation in a) eine Faltung in b)	42
Abbildung 16: Pascalsches Dreieck.....	43
Abbildung 17: RGB Farbwürfel	45
Abbildung 18: HSV hexagonale Pyramide.....	47
Abbildung 19: Erosion mit einer 3x3 Maske; Links: Ausgangszustand; Rechts: Zustand nach Erosion, die entfernten Pixel sind rot markiert	50
Abbildung 20: Dilatation mit einer 3x3 Maske; Links: Ausgangszustand; Rechts: Zustand nach Erosion, die hinzugefügten Pixel sind grün markiert.....	51
Abbildung 21 Beispiel Öffnen: Links: Ausgangszustand, Mitte: Erosion mit einer 5x5 Maske; Rechts: Öffnen mit einer 5x5 Maske	52
Abbildung 22: Linienzug bestehend aus acht Punkten.....	58
Abbildung 23: Verbindungsline a0-f0 (Toleranzbereich ist blau hinterlegt): Größter Abstand zu f1 ist außerhalb der Toleranz; f1 wird der neue Springer.....	59
Abbildung 24: Verbindungsline a0-f1: Alle Abstände innerhalb Toleranz; Springer f1 wird neuer Anker a1 und vorheriger Springer f0 wird nächster Springer	59
Abbildung 25: Verbindungsline a1-f0: Größter Abstand zu f2 ist außerhalb der Toleranz	60

Abbildung 26: Verbindungslinie a1-f2: Größter Abstand zu f3 ist außerhalb der Toleranz	60
Abbildung 27: Verbindungslinie a1-f3: Alle Abstände innerhalb Toleranz; f3 wird zu a2	61
Abbildung 28: Verbindungslinie a2-f2: Alle Abstände innerhalb Toleranz; f2 wird zu a3	61
Abbildung 29: Verbindungslinie a3-f0: Keine Punkte zwischen Anker und Springer; f0 wird zu a4.....	62
Abbildung 30: Vergleich des vereinfachten Linienzuges (durchgezogen) zum originalen Linienzug (gestrichelt)	62
Abbildung 31: Konvexe Menge (links) und nichtkonvexe Menge (rechts)	64
Abbildung 32: Niveaulinien der Summe aus X- und Y-Koordinate	67
Abbildung 33: Niveaulinien der Differenz aus X- und Y-Koordinate	67
Abbildung 34: Modell der Lochkamera mit dargestellter Lochblende (Zentralpunkt) 73	73
Abbildung 35: Approximation Funktion der X-Achse	86
Abbildung 36: Messung des Jitters des Software-PWM-Signals	88
Abbildung 37: Messung des Jitters des Hardware-PWM-Signals.....	90
Abbildung 38: Berechnete Geschwindigkeiten bei 15 FPS und 0,6 Kreisumdrehungen pro Sekunde	95
Abbildung 39: Berechnete Geschwindigkeiten bei 10 FPS und 0,5 Kreisumdrehungen pro Sekunde	95
Abbildung 40: Berechnete Geschwindigkeiten bei 10 FPS und hoher Prozesspriorität	96

Tabellenverzeichnis

Tabelle 1: Typischer Zusammenhang zwischen Winkel und Pulsweite für Servomotoren	12
Tabelle 2: Liste der ausgelagerten Datein und ihre Funktionen	28
Tabelle 3: Aufgenommene Wertepaare zur Approximationsfunktion	86
Tabelle 4: Netzwerk-Konfiguration für Desktop-PC und Raspberry Pi.....	91

Abkürzungsverzeichnis

Abkürzung	Begriff
RC	Radio Control
PWM	Pulse Width Modulation
FPS	Frames per Second ¹
RAM	Random-Access Memory
CPU	Central Processing Unit
HDMI	High Definition Media Interface
USB	Universal Serial Bus
GPIO	General Purpose Input Output
GND	Ground
PID	Proportional-Integral-Derivativ
MIMO	Multi-Input, Multi-Output
SISO	Single-Input, Single-Output
API	Schnittstelle
pigpiod	pigpio-Daemon
IDE	Integrated Developer Environment

¹ FPS wird in dieser Arbeit sowohl für die Bildwiederholungsrate der Kamera, als auch für die Prozesszyklusrate (verarbeitete Frames pro Sekunde) verwendet

Einheitenverzeichnis

Einheit	Bezeichnung
cm	Zentimeter
mm	Millimeter
s	Sekunde
ms	Millisekunden
μs	Mikrosekunden
°	Grad (Winkelmaß)
N	Newton
V	Volt
GB	Gigabyte
px	Pixel

1. Einführung

1.1. Motivation

Bei dem Versuch, einen kleinen Ball in einer Kreisbahn auf einem Tischtennisschläger zu balancieren fungieren die Augen als Sensor zur Erkennung der Position des Balls und geben diese an das Gehirn weiter. Das Gehirn verarbeitet erhaltene Informationen und gibt wiederum an die Hände weiter, in welche Schräglage sie den Schläger bringen müssen. Diese Studienarbeit befasst sich mit der Automatisierung dieses Prozesses.

Die Wahl des Themas der Studienarbeit beruht auf dem Wunsch, ein Projekt zu realisieren, welches weitestgehend die Grundpfeiler der Mechatronik beinhaltet und einen realitätsnahen Bezug hat. Die drei Grundpfeiler sind der Maschinenbau, die Elektrotechnik und die Informatik. Das Einsatzgebiet der Mechatronik ist durch seine Vielseitigkeit fast überall in der Industrie anzutreffen, vor allem aber dort, wo Prozesse automatisiert werden. Dort, wo Maschinen die Arbeit von Menschen übernehmen und selbstständig Entscheidungen treffen anhand von Daten, die Sensoren für sie sammeln.

Das im Folgenden als Kugellabyrinth bezeichnete Projekt behandelt das automatisierte Balancieren eines Balls auf einer Platte und erfüllt somit die gewünschten Bedingungen.

Eine Kamera ersetzt das menschliche Auge und sendet Bilder an einen Computer, welcher als „Gehirn“ der Anlage fungiert. Dort angekommen werden die Daten verarbeitet und entscheiden, wie die Aktoren die Platte bewegen sollen. Der mechanische Aufbau benötigt Kenntnisse im Maschinenbau, die Verkabelung und Signalübermittlung benötigt Fachwissen in der Elektrotechnik und die Datenverarbeitung wird mithilfe der Informatik realisiert.

1.2. Vorarbeiten

Das Projekt einer automatisierten Vorrichtung, die mittels visueller Datenerfassung einen Ball auf einer Platte ausbalanciert, basiert auf einer Studienarbeit, welche schon im Jahr 2018 von den Herren Kurz, Ebert und Stadler angefangen wurde und den Titel „Projekt Kugellabyrinth“ trägt [1].

Diese vorangegangen Studienarbeit hat zwar das gleiche Ziel, legt dabei aber einen großen Schwerpunkt auf den mechanischen Aufbau und die Auslegung der einzelnen Bauteile (siehe Kapitel 2.1). Da aufgrund einer zu großen Verzögerung bei der Datenverarbeitung in den abschließenden Tests kein stabiler Zustand der Kugel erreicht werden konnte [1], galt das Projekt als nicht abgeschlossen und bietet die Möglichkeit einer Fortsetzung.

Zusammen mit dem Titel wird somit zwar der mechanische Aufbau des Vorgängerprojekts übernommen und dient als Grundlage für diese Studienarbeit, die programmier-technische Umsetzung wird allerdings grundlegend neu aufgebaut.

1.3. **Aufgabenstellung**

Ziel des Projektes „Kugellabyrinth“ ist es, eine Kugel voll automatisiert und zielsicher durch ein auf einer Platte montiertes Labyrinth zu führen. Hierfür soll eine Kamera die Position der Kugel erfassen und durch Neigung der Platte soll die Kugel gesteuert werden.

Da ein solches Projekt den Umfang einer einzelnen Studienarbeit übersteigt, beläuft sich das Ziel dieser Studienarbeit darauf, das von den Vorgängern bereits erstellte Modell soweit zu bringen, dass die Kugel in eine stabile Lage gebracht werden kann, beziehungsweise anschließend einer definierten Kreisbahn folgen kann.

1.4. **Stand der Technik**

Das Ausbalancieren von einem Ball auf einer Platte ist in der Ingenieurwissenschaft unter der Problemstellung „Ball on Plate“ angesiedelt. Bei der Online-Recherche finden sich viele Projekte zu diesem Thema, welche sich ausschließlich in der Detektion vom Ball unterscheiden. Eine weit verbreitete Methode ist die Detektion über die Kamera, welche einen hohen Aufwand der Bildverarbeitung mit sich bringt. Die Alternative ist ein kapazitives Touchpad, welches den Ball auf direkte Weise detektiert. Bei dieser Studienarbeit wird die Regelung durch eine Kamera realisiert, da dies Teil der Aufgabenstellung ist.

Das Projekt „Ball on Plate“ wird auch an vielen Hochschulen und Universitäten behandelt. Ein Projektteam um Dipl.-Ing. Jan Strubel und Dipl.-Ing. Daniel Labisch von der Technischen Universität Darmstadt (TU Darmstadt) arbeitet derzeit mit Studenten an dieser Aufgabenstellung².

Der wesentliche Unterschied im Projekt der TU Darmstadt liegt im mechanischen Aufbau. Die Aktoren wirken direkt an der Platte, wodurch eine direkte Verbindung zwischen den Stellmotoren und den Koordinatenachsen vorliegt [2].



Abbildung 1: Projektaufbau „Ball on Plate“ TU Darmstadt³

Durch den gewählten Aufbau muss keine Umrechnung zwischen der Winkelposition des Aktors und der Winkelposition der Platte vorgenommen werden, sodass die Komplexität minimiert wird.

Die Erkennung der Ballposition wird über eine handelsübliche Webcam realisiert. Durch die schwarze Platte und das Metallgerüst entsteht ein hoher Kontrast, welcher die Balldetektion auch bei schwierigen Lichtverhältnissen ermöglicht.

² Die zeitliche Angabe derzeit bezieht sich auf den Ausarbeitungszeitraum dieser Studienarbeit

³ Quelle: TU Darmstadt

Die Hochschule Arnheim hat das Projekt auch behandelt. Aus dem veröffentlichten Video [3] geht hervor, dass der Ball mit einer Kamera detektiert wird, die Bildverarbeitung sowie die Berechnung der Stellgrößen an einem Computer stattfinden und die Servomotoren über eine scheinbar selbst angefertigte Platine gesteuert werden. Der mechanische Aufbau scheint identisch mit dem in dieser Studienarbeit zu sein.

Das Video zeigt, dass mit der gegebenen Hardware der Ball präzise im Kreis balanciert werden und auch durch ein Kugellabyrinth verfahren kann.

In beiden Fällen gibt es keine veröffentlichten Papers oder Arbeiten über die Projekte, wodurch keine genauen Informationen über die Regelung und Bildverarbeitung vorliegen.

2. Modellaufbau

Dieses Projekt hat als Ziel, einen Ball auf einer Platte zu balancieren. Um das zu realisieren, erfasst eine Kamera Bilder des Balls auf der Platte und überträgt diese an einen Desktop-PC. Dort wird mittels Bildverarbeitung die Position und die Geschwindigkeit des Balls ermittelt und ausgerechnet, in welche Winkellage die Platte verfahren soll. Diese Information wird über einen Raspberry Pi, einen Minicomputer, an zwei Servomotoren gegeben, welche anschließend die Platte in die gewünschte Winkellage versetzen.

2.1. Mechanischer Aufbau

Zunächst wird der mechanische Aufbau des Modells erklärt, welcher von den Vorgängern übernommen wird. Hierfür gibt Abbildung 2 einen ersten Überblick über die vorhandenen Komponenten [1].

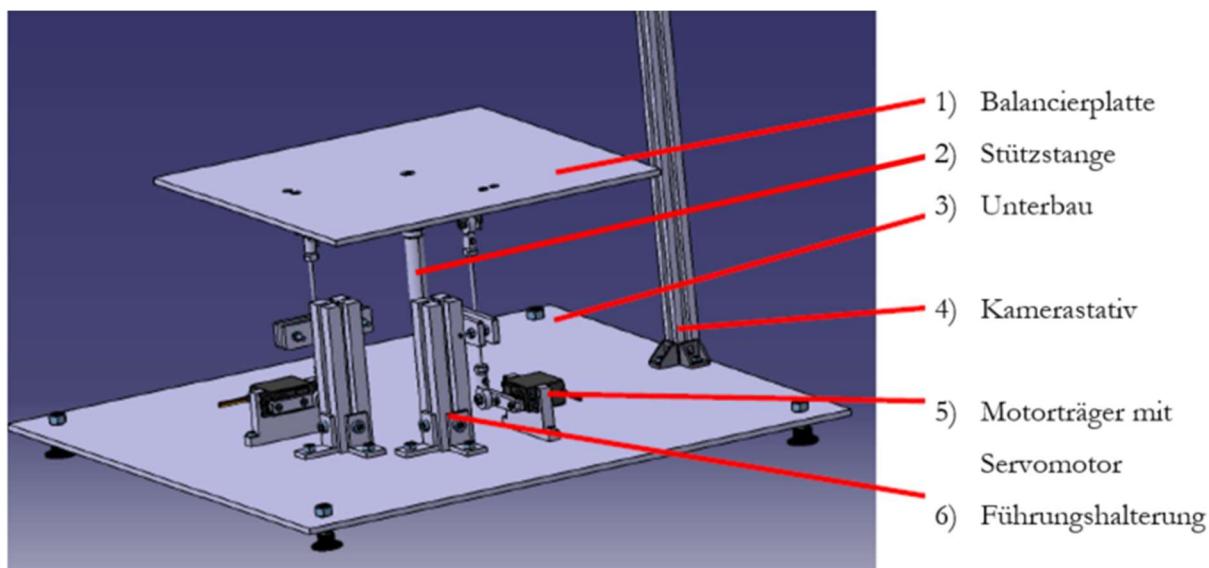


Abbildung 2: CAD-Modell mit Beschriftung⁴

Die quadratische Balancierplatte 1) mit einer Kantenlänge von 300 mm ist die Platte, auf welcher der Ball später balanciert werden soll. An ihrer Unterseite ist mittig ein Kugelgelenk befestigt. Das Kugelgelenk eliminiert die Translationsfreiheitsgrade und erlaubt somit nur die freie Rotation im Raum mit dem Mittelpunkt der Platte als starren

⁴ Quelle: Projekt Kugellabyrinth (2019) [1]

Fixpunkt. Die Leichtgängigkeit des Gelenks kann mithilfe einer Mutter variiert werden. An der festen Seite des Kugelgelenks befindet sich die 150 mm lange Stützstange 2). Sie verbindet die Balancierplatte mit dem Unterbau 3), welcher aus einer 500x500 mm, ebenfalls quadratischen, Metallplatte besteht. Mithilfe von vier Stellfüßen an den Eckpunkten der Grundplatte lässt sich das gesamte Modell austarieren.

Um zu veranschaulichen, wie die Servomotoren die Winkellage der Platte steuern können, zeigt Abbildung 3 einen detaillierten Einblick unter die Balancierplatte.

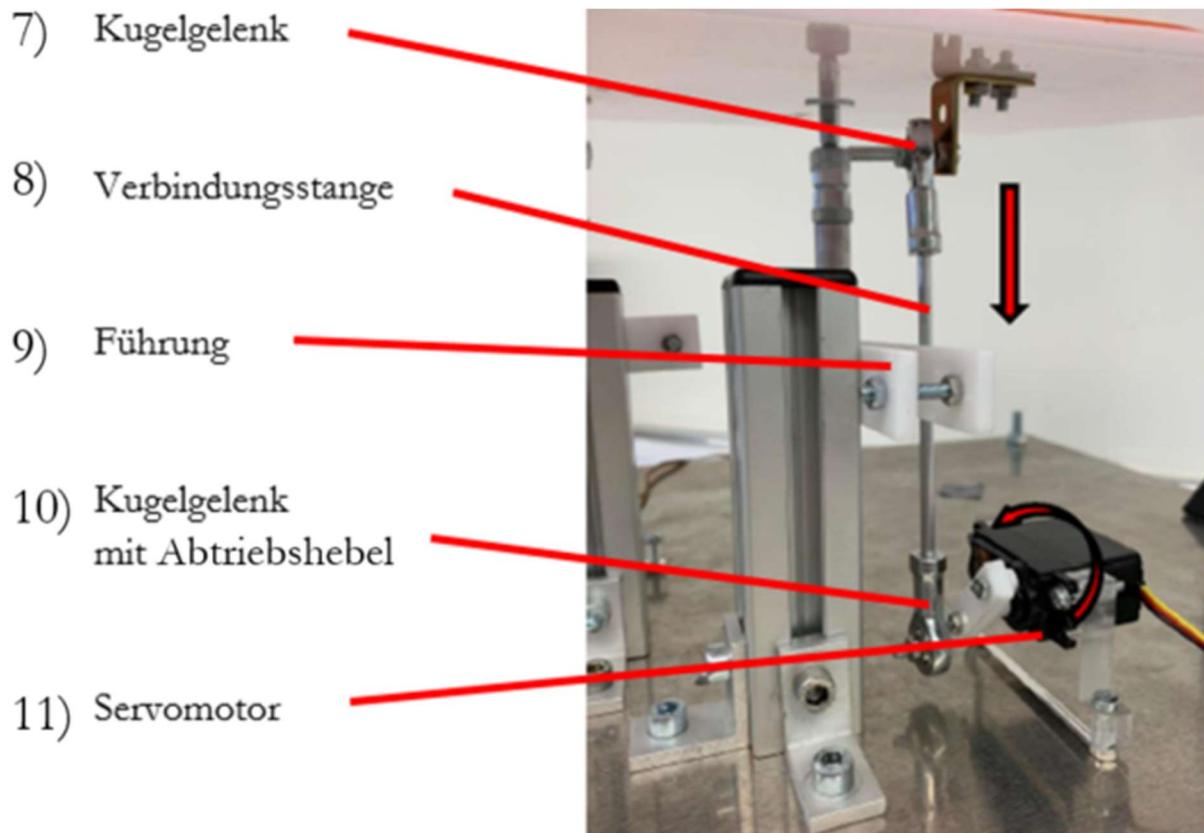


Abbildung 3: Verbindung zwischen Servomotor und Balancierplatte⁵

Zu sehen ist eine der beiden Verbindungen zwischen der Balancierplatte und den Servomotoren. Ziel ist es, die Rotationsbewegung des Servomotors 11) über eine Translation der Verbindungsstange 8) in eine Rotation der Balancierplatte umzuleiten (wie

⁵ Quelle: Projekt Kugellabyrinth (2019) [1]

Aufgrund der unerwartet eingeschränkten Zugangsmöglichkeit zum Projekt gegen Ende der Bearbeitungszeit der Studienarbeit konnten keine eigenen Bilder gemacht werden

durch die beiden roten Pfeile verdeutlicht). Um dies zu realisieren, ist die Verbindungsstange an beiden Enden mit einem Kugelgelenk versehen. Eines der Kugelgelenke ist mit der Platte 7) über einen Winkel verschraubt, das andere ist mit dem Abtriebshebel 10), welcher an der Motorwelle befestigt ist. Die Befestigung des Abtriebshebels an der Motorwelle ist formschlüssig mithilfe einer Verzahnung auf der Innenseite des Hebels und der Motorwelle. Dieser Formschluss verhindert nicht das Lösen des Abtriebshebels längs der Motorwellenachse, weshalb hier nachträglich eine Schraube, fluchtend mit der Motorwelle, angebracht wird. Der Servomotor 11) (Genaueres wird in Kapitel 2.2.2 erläutert) ist mithilfe einer Halterung aus Plexiglas mit der Grundplatte verschraubt. Die gesamte Verbindung ist so ausgelegt, dass bei waagerechter Stellung beider Abtriebshebel die Balancierplatte parallel zur Grundplatte ausgerichtet ist.

Um den dritten Rotationsfreiheitsgrad, gemeint ist die Rotation um eine mit der Stützstange fluchtende Achse, zu beseitigen, werden die Führungen 9) beider Verbindungen verwendet. Sie drücken seitlich, in entgegengesetzter Rotationsrichtung auf die Verbindungsstangen und eliminieren so besagten Freiheitsgrad. Durch Verändern des Abstands der beiden Kunststoffplatten lässt sich der Druck auf die Verbindungsstangen variieren. Der Abstand wird so gewählt, dass der Reibungswiderstand so gering wie möglich ist, die Verbindungsstange aber dennoch fixiert bleibt.

Die beiden beschriebenen Verbindungen befinden sich um 90° verdreht zueinander auf der Grundplatte, wie in Abbildung 2 angedeutet. Dies hat den Vorteil, dass bei der Regelung die beiden Koordinatenachsen getrennt voneinander betrachtet werden können, beziehungsweise entkoppelt sind (siehe Kapitel 3).

Bei Betrachtung von Abbildung 2 ist das Kamerastativ 4) nicht vollständig zu sehen⁶. Es besteht aus einem item-Profil aus Aluminium. Am oberen Ende (in Abbildung 2 nicht sichtbarer Teil) befindet sich ein horizontaler Ausleger 12), ebenfalls bestehend aus einem item-Profil. An diesem sind, wie in Abbildung 4 zu sehen, eine handelsübliche, batteriebetriebene Lampe 13) und die Kamera 16) montiert.

⁶ Aufgrund der unerwartet eingeschränkten Zugangsmöglichkeit zum Projekt gegen Ende der Bearbeitungszeit der Studienarbeit konnten keine eigenen Bilder gemacht werden

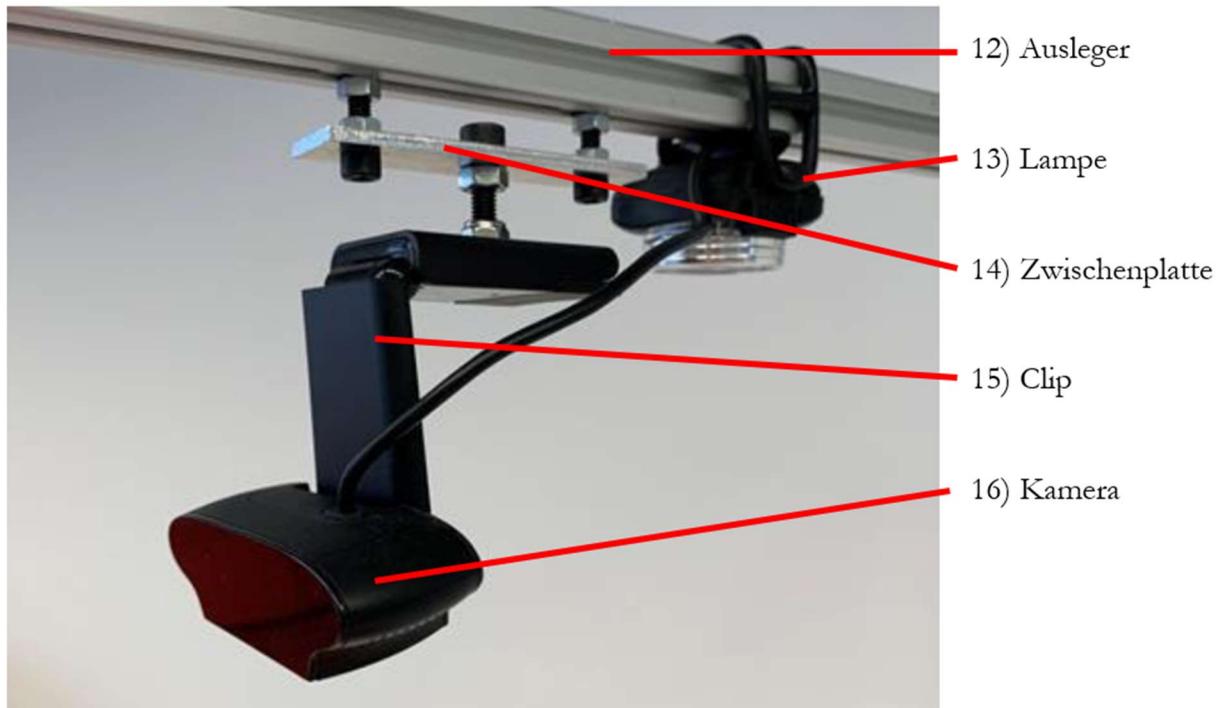


Abbildung 4: Kamerabefestigung mit Beleuchtung⁷

Die Position der Zwischenplatte 14) lässt sich durch die Aufhängung in den Nuten des item-Profil verschieben und der Winkel der Kamera mithilfe des Gelenks im Clip 15) variieren. Dank einer, in der Bildverarbeitung verwendeten, perspektivischen Transformation (siehe Kapitel 4.5.5) ist eine auf das Grad exakte Ausrichtung der Kamera auf die Balancierplatte nicht notwendig.

2.2. Elektrische Komponenten

2.2.1. Kamera

Bei der Kamera, wie sie in Abbildung 5 zu sehen ist, handelt es sich um das Webcam-Modell C930e von Logitech mit einer Auflösung von bis zu 1920x1080 Pixel und einer Bildrate von bis zu 30 Bildern pro Sekunde (Frames Per Second, kurz: FPS). Sie wird

⁷ Quelle: Projekt Kugellabyrinth (2019) [1]

Aufgrund der unerwartet eingeschränkten Zugangsmöglichkeit zum Projekt gegen Ende der Bearbeitungszeit der Studienarbeit konnten keine eigenen Bilder gemacht werden

wie in Kapitel 2.1 beschrieben montiert und anschließend über eine USB-Schnittstelle mit einem Computer verbunden [4].



Abbildung 5: Logitech Webcam C930e⁸

2.2.2. Servomotoren

Um den gewünschten Plattenwinkel anzufahren, werden zwei Servomotoren verwendet. „Ein Servomotor ist ein Motor, der es ermöglicht, die genaue Position der Motorwelle sowie die Drehzahl und/oder die Beschleunigung zu kontrollieren.“ [5] Das bedeutet, dass eine sogenannte Servoregelung integriert ist, welche auf die von außen angegebene Führungsgröße reagiert, wie in Abbildung 6 gezeigt [6].

⁸ Quelle: www.logitech.com

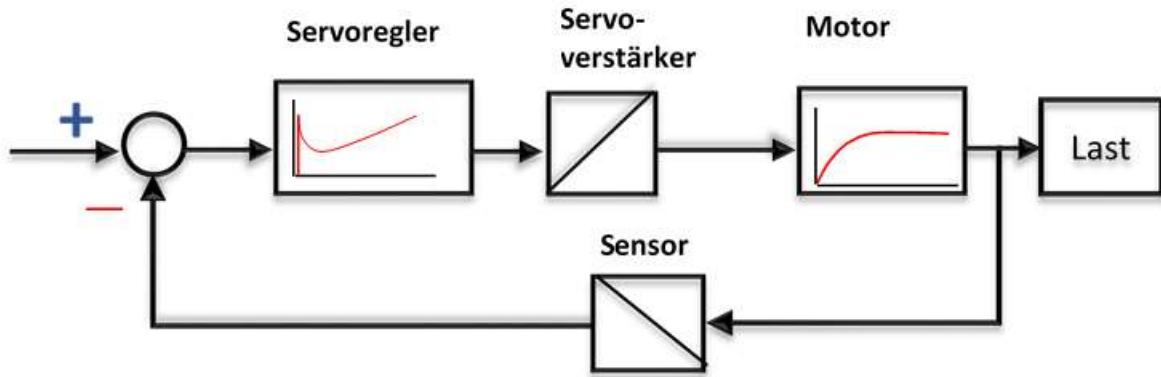


Abbildung 6: Regelkreis eines Servomotors⁹

In der Abbildung von links kommend ist die Führungsgröße, oder auch Sollwert genannt, zu sehen. Auf den Servomotor bezogen handelt es sich dabei um das Pulsweitenmodulations-Signal (Pulse Width Modulation, kurz: PWM), welches zuerst in die Sollgröße umgewandelt werden muss. Von dieser wird der Ist-Wert abgezogen, welcher durch die integrierten Sensoren aufgenommen wird. Es handelt sich hierbei, abhängig von der Anwendung und vom Servomotor, um Positions-, Geschwindigkeits- und/oder Drehmomentsensoren [7] (im Fall der im Projekt verwendeten Servomotoren erfolgt die Zustandsrückführung der Position über ein Potentiometer [1]). Die so gewonnene Differenz, auch Regelfehler oder Regelabweichung genannt, wird vom Servoregler, mithilfe des Servoverstärkers, in die Stellgröße umgewandelt. In den meisten Fällen befindet sich nach dem Motor noch ein Getriebe, um aus einer hohen Drehzahl das erforderliche Drehmoment zu generieren.

Bei den beiden Motoren des Modells handelt es sich um digitale RC (Radio Control) Servomotoren des Typs HS-5485HB von HiTEC, welche häufig Verwendung im Modellbau finden, da sie gezielt gewünschte Winkelpositionen anfahren und halten können. Sie können ein Drehmoment von $64 \frac{N}{cm}$ und eine Geschwindigkeit von bis zu $0,17 \frac{s}{60^\circ}$ aufbringen [8]. Sie benötigen eine Gleichspannung von 4,8 bis 6 V, welche durch das Netzgerät geliefert wird, und ein Steuersignal, um den gewünschten Drehwinkel anzusteuern, welches durch eine Pulsweitenmodulation vom Raspberry Pi kommt (siehe Kapitel 2.2.3). Des Weiteren besitzen sie einen Anschlag bei einem

⁹ Quelle: <http://learnchannel-tv.com/de/drives/servomotor/> (16.04.2020)

Drehwinkel von ca. plus und minus 45° , was für Servomotoren aus dem Modellbaubereich unüblich ist, da diese meistens einen Gesamtwinkel von 180° abdecken, für ihre Funktion im Modell aber ausreicht.

Bei einem PWM-Signal handelt es sich um ein Pulssignal mit einer gleichbleibenden Frequenz. Die Information liegt nicht in der Höhe der Spannung, sondern in dem Verhältnis zwischen Pulslänge und Periodendauer [9], wie Abbildung 7 beispielhaft zeigt. In diesem Fall wird der prozentuale Anteil des Pulses von der Periode angegeben, in der praktischen Anwendung wird allerdings bei bekannter Frequenz die Pulslänge in Millisekunden angegeben. Das heißt, der Drehwinkel der Servomotoren wird durch die Zeitangabe der Pulsdauer gesteuert.

Von besonderer Relevanz sind bei dieser Art der Informationsübermittlung die Flanken des Signals, welche möglichst steil sein sollten.

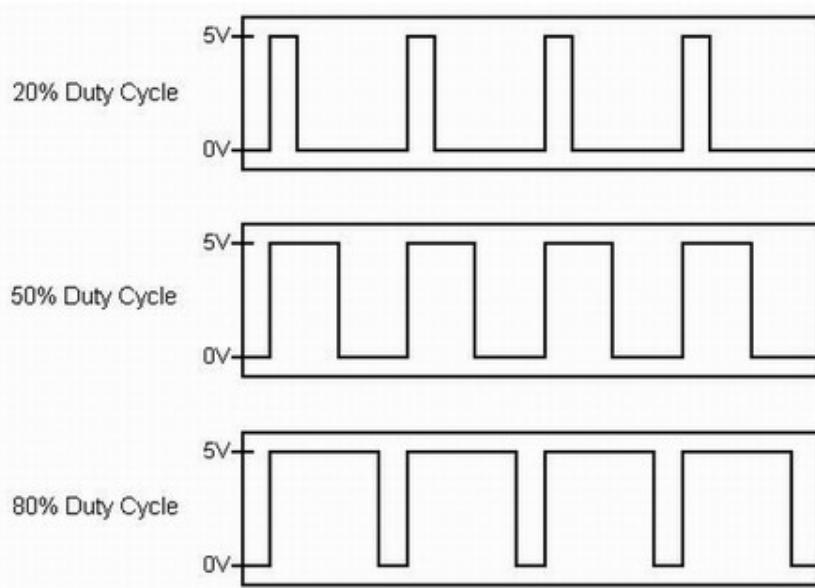


Abbildung 7: Beispielhafte PWM-Signale¹⁰

Die in dieser Studienarbeit verwendete Frequenz des PWM-Signals beträgt 50 Hz, was der für im Modellbau verwendeten Servomotoren üblichen Frequenz entspricht [10].

¹⁰ Quelle: <http://www.ni.com/tutorial/2991/en/> (15.04.2020)

Standardmäßig stellt sich bei RC-Servos die Mittelposition bei einer Pulsdauer von 1,5 Millisekunden ein. Eine längerer Pulsdauer fährt eine Winkelstellung entgegen dem Uhrzeigersinn an und eine kürzerer Pulsdauer mit dem Uhrzeiger (bei Blick auf die Motorwelle) [7]. Tabelle 1 zeigt beispielhaft einige solcher Pulslängen, wie sie häufig bei Servomotoren verwendet werden [11], wobei hier die Mittelstellung mit 0° angegeben wird.

-90°	-60°	-30°	0°	30°	45°	90°
1 ms	1,167 ms	1,33 ms	1,5 ms	1,667 ms	1,75 ms	2 ms

Tabelle 1: Typischer Zusammenhang zwischen Winkel und Pulsweite für Servomotoren

Eine genauere Betrachtung der exakten Motorwinkel-Pulslängen-Abhängigkeit ist nicht notwendig, da in dieser Studienarbeit nur der resultierende Neigungswinkel der Balancierplatte betrachtet wird. Das zugrundeliegende Modell wird in Kapitel 4.7.1 genauer erläutert.

2.2.3. Raspberry Pi

Der Raspberry Pi ist ein kompakter Einplatinencomputer, der weit verbreitet und in diversen Anwendungen vielseitig einsetzbar ist. Die in dieser Studienarbeit verwendete Variante 3B+ verfügt über 1 GB Arbeitsspeicher (Random-Access Memory, kurz: RAM) und eine ARM-CPU (Central Processing Unit) mit vier Kernen [12]. Neben seiner Rechenleistung bietet der Raspberry Pi viele Schnittstellen. In dieser Studienarbeit werden davon die HDMI- (High Definition Media Interface), USB- (Universal Serial Bus) und die Ethernet-Schnittstellen verwendet. Des Weiteren bietet der Raspberry Pi mit seinen 40 Pins, von denen 26 programmierbar sind, sogenannte GPIO-Pins (General-Purpose-Input Output), diverse Möglichkeiten zur elektrischen Ansteuerung [13]. Entwickelt und hergestellt wird der Minicomputer von der britischen Raspberry Pi Foundation, mit dem Ziel, den Erwerb von Programmier- und Hardware-Kenntnissen zu erleichtern, indem er kostengünstig angeboten wird [14].

Programmiert wird der Raspberry Pi mithilfe der Programmiersprache Python, welche durch ihre Benutzerfreundlichkeit den leichten Einstieg in das Programmieren ermöglicht. Ein weiterer Vorteil von Python ist das extrem breite Spektrum an bereits existierenden robusten Bibliotheken [13], wobei eine genauere Erläuterung der in dieser Studienarbeit verwendeten Bibliotheken in Kapitel 4.1 vorgenommen wird.

Die bereits erwähnten GPIO-Pins sind mitunter einer der Gründe, weshalb der Raspberry Pi viel Verwendung in verschiedensten Projekten findet. Mit nur wenigen Programmzeilen können die GPIO-Pins für Anwendungen, wie die Buskommunikation oder die Motoransteuerung über ein PWM-Signal konfiguriert werden.

Abbildung 8 zeigt den Raspberry 3B+ und die in dieser Studienarbeit relevanten Schnittstellen. Rot markierte Komponenten stehen für die beim fertigen Projekt verwendeten Schnittstellen.

- Unten links im Bild befindet sich die Micro-USB-Buchse, welche den Raspberry Pi mit der nötigen 5 V Spannung versorgt.
- Unten rechts im Bild ist der Ethernet-Anschluss zu sehen, welcher in diesem Projekt die Kommunikation mit dem Desktop-PC ermöglicht.
- Oben im Bild ist die Pin-Leiste mit ihren 40 Pins zu sehen. Die beiden Pins mit den Nummern 38 und 40 sind direkt mit den beiden Servomotoren verbunden und übertragen das PWM-Signal. Der Pin 34 verbindet die Massepotentiale (Ground, kurz: GND) von Raspberry Pi und Servomotoren.

Bei den blau markierten Schnittstellen handelt es sich zum einen um die vier USB-Schnittstellen, welche im ursprünglichen Projekt für die Kamera verwendet wurden und die weiterhin die Möglichkeit bieten, eine Maus und eine Tastatur an den Raspberry Pi anzuschließen. Zum anderen ist im unteren Teil der Abbildung die HDMI-Schnittstelle

zu sehen, welche das Anschließen eines Bildschirms zum Betrieb des Raspberry Pis ermöglicht.

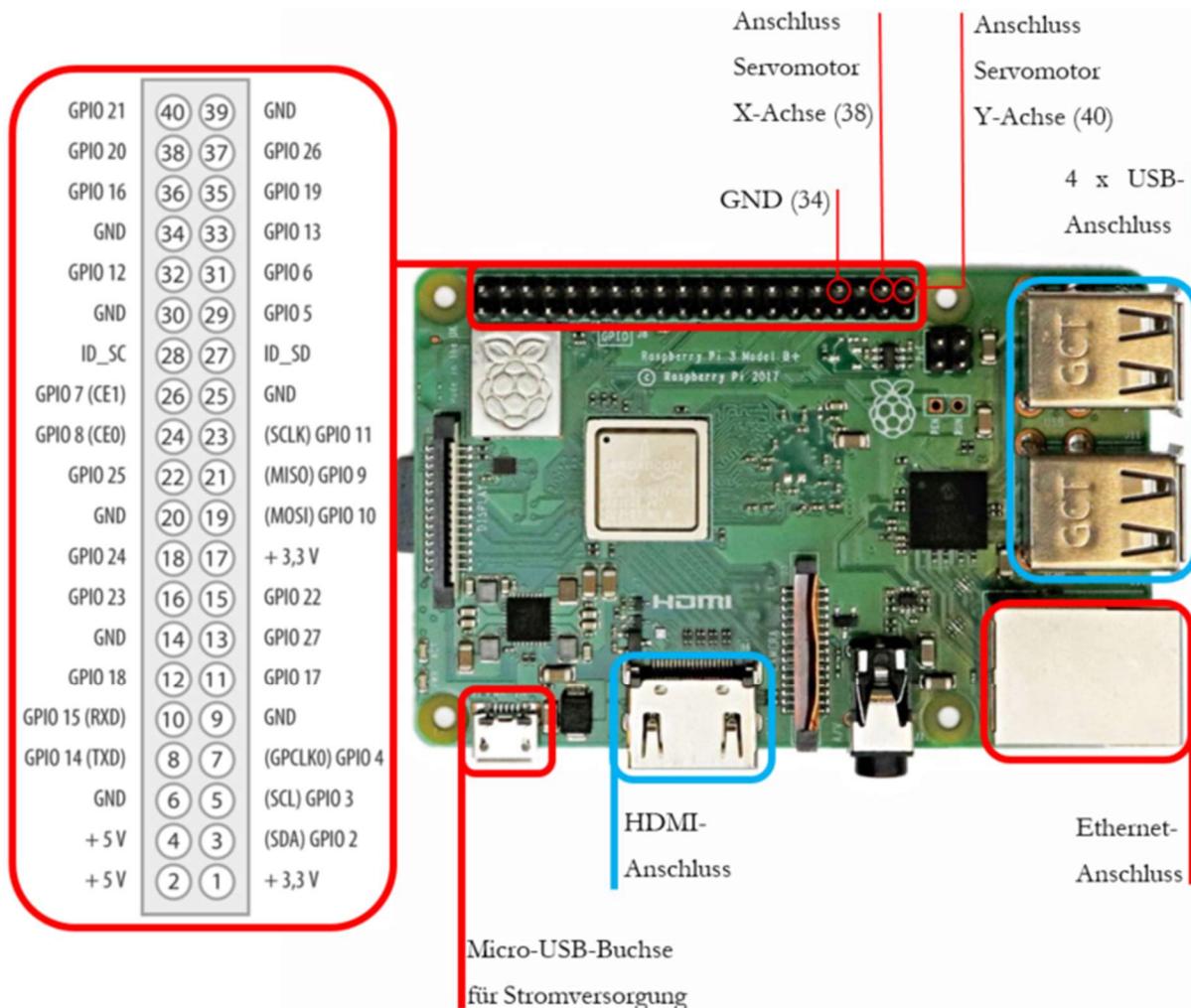


Abbildung 8: Raspberry Pi 3 mit Beschriftung¹¹

In der vorangegangenen Studienarbeit wurde der Raspberry Pi für die Bildverarbeitung und die Regelung verwendet. Die Berechnungen verursachten beim Raspberry Pi aufgrund des nicht ausreichenden internen Puffers starke Verzögerungen, welcher somit nicht das gewünschte Ergebnis liefern konnte. Als Lösung wird ein Leistungsstärkerer Computer empfohlen [1].

Die niedrige FPS und die starken Verzögerungen ließen sich selbst bei Nutzung von unkomplizierten Bildverarbeitungsalgorithmen nicht beheben, weshalb der Vorschlag

¹¹ Quelle Ursprungsbild: <https://www.makeyourschool.de/material/raspberrypi/> (17.04.2020)

des leistungsstarken Computers (in dieser Studienarbeit als Desktop-PC bezeichnet) umgesetzt wird. Die Bildverarbeitung und die Regelung werden somit ausgelagert und ausschließlich die PWM-Signalerzeugung findet noch auf dem Raspberry Pi statt (siehe Kapitel 4.7.3).

3. Zustandsregelung

Die Zustandsraumdarstellung (»State-Space Representation«) ist eine aus der Regelungstechnik stammende Methode, um dynamische Systeme im Zeitbereich darzustellen und zu modellieren [15]. Die Anwendung dieser Darstellung stammt aus den 1960er Jahren und führt zu einer deutlichen Erweiterung der Regelungstheorie, weshalb heute zwischen der „klassischen“ und „modernen“ Regelungstechnik unterschieden wird [16]. Der wesentliche Vorteil des Zustandsraummodells ist die besonders effiziente Behandlung von Mehrgrößensystemen, nichtlinearen und zeitvariablen Übertragungssystemen. Es eignet sich besonders gut für numerische Simulationen und im Gegensatz zur klassischen PID-Regelung werden alle für den Prozess relevanten dynamischen Größen des Prozesses mit einbezogen. Durch die geringe mechanische Komplexität des Problems „Ball on Plate“ wird das Zustandsraummodell gewählt, um eine Regelung zu entwerfen [16].

Festlegen des Koordinatensystems

Für die Implementierung des Zustandsraummodells in Python wird ein Koordinatensystem definiert. Durch dieses Koordinatensystem wird die Position des Balls einem eindeutigen X- und Y-Wert auf der Platte zugewiesen. Dieses „Plattenkoordinatensystem“ wird aus Symmetriegründen in das Zentrum der Platte gelegt, wobei der Koordinatenursprung eine untergeordnete Rolle spielt.

3.1. Aufstellen des Zustandsraummodells

Grundlage des Zustandsraummodells

Um ein Zustandsraummodell aufzustellen, muss im ersten Schritt die Differentialgleichung des zu regelnden Systems ermittelt werden. Anschließend wird diese Differentialgleichung n -ter Ordnung mittels Substitution in ein Differentialgleichungssystem mit n Differentialgleichungen erster Ordnung umgewandelt. Durch eine Vektor-Matrix-Darstellung wird dieses Differentialgleichungssystem als Zustandsraummodell dargestellt.

Ermittlung der Differentialgleichung

Durch die „*Newtonsche Bewegungsgleichung der klassischen Mechanik*“ ergibt sich die Differentialgleichung des Balles auf der Platte. Im Folgenden wird die Herleitung der Differentialgleichung erläutert.

Im ersten Schritt ergeben sich über das paarweise „Freischneiden“ die auf den Körper wirkende Kräfte.

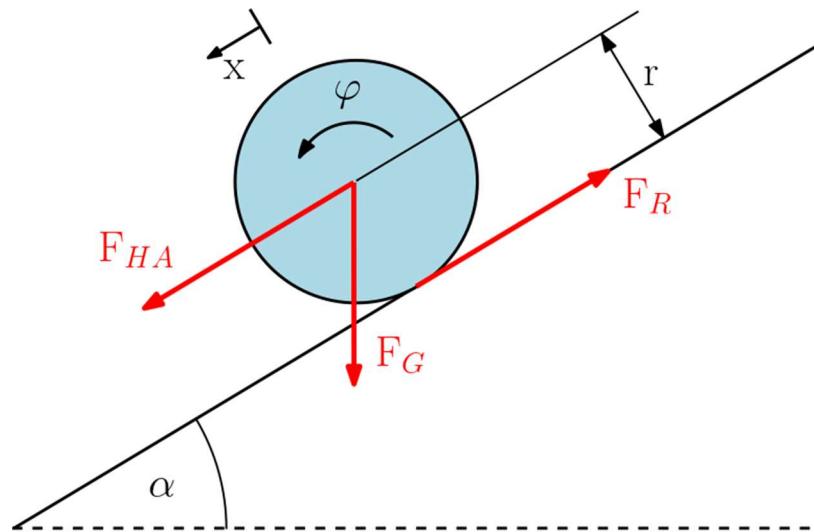


Abbildung 9: Freischnitt Ball auf Platte

Es ergibt sich die:

$$\text{Gewichtskraft: } F_G = m * g \quad (3.1)$$

$$\text{Hangabtriebskraft: } F_{HA} = F_G * \sin \alpha = m * g * \sin \alpha \quad (3.2)$$

$$\text{Haftriebkraft: } F_{HR}$$

$$\text{Kugelradius: } r$$

$$\text{Plattenwinkel: } \alpha$$

$$\text{Trägheitsmoment der Kugel [17]: } J = \frac{2}{5} * m * r^2 \quad (3.3)$$

Kräftegleichgewicht in x-Richtung:

$$m * \ddot{x} = F_{HA} - F_{HR} \quad (3.4)$$

Momentengleichung um den Ballmittelpunkt:

$$J * \ddot{\varphi} = F_{HR} * r$$

Mit Trägheitsmoment der Kugel : $J = \frac{2}{5} * m * r^2$ (3.3)

aufgelöst nach F_{HR} :

$$F_{HR} = \frac{J * \ddot{\varphi}}{r} = \frac{2 * m * r * \ddot{\varphi}}{5} \quad (3.5)$$

Kinematische Beziehung:

$$\ddot{\varphi} = \frac{\ddot{x}}{r} \quad (3.6)$$

Hangabtriebskraft: $F_{HA} = F_G * \sin \alpha = m * g * \sin \alpha$ (3.2),

(3.5) und (3.6) in (3.4):

$$\begin{aligned} m * \ddot{x} &= m * g * \sin \alpha - \frac{2 * m * r * \ddot{\varphi}}{5} \\ \ddot{x} &= g * \sin \alpha - \frac{2 * m * r * \ddot{x}}{5 * r} \\ \ddot{x} \left(1 + \frac{2}{5}\right) &= g * \sin \alpha \\ \ddot{x} &= \frac{5}{7} * g * \sin \alpha \end{aligned}$$

Durch den mechanischen Aufbau sind maximale Plattenwinkel von ca. $\pm 10^\circ$ möglich, weshalb die Kleinwinkelnäherung angewendet wird.

$$\ddot{x} = \frac{5}{7} * g * \alpha \quad (3.7)$$

Die Bewegung in Y-Richtung folgt denselben kinematischen Gesetzen, auf Basis des Plattenwinkels β :

$$\ddot{y} = \frac{5}{7} * g * \beta \quad (3.8)$$

Um das mathematische Modell anzuwenden, muss die Erdbeschleunigung noch in $\frac{\text{Pixel}}{\text{s}^2}$ umgerechnet werden, da die Position des Balls in Pixeln und nicht in Metern angegeben wird. Die Position des Balls bezieht sich auf einen Frame, welcher die 300x300 mm große Platte in 480x480 Pixel darstellt. Für die Umrechnung wird die Erdbeschleunigung g durch die Hilfsvariable g_{px} ersetzt. Es ergibt sich folgende Formel für g_{px} :

$$g_{px} = \frac{480 \text{ Pixel}}{300 \text{ mm}} * 9810 \frac{\text{mm}}{\text{s}^2} = 1.6 * 9810 \frac{\text{Pixel}}{\text{s}^2} \quad (3.9)$$

Aufstellen des Zustandsraummodells

Durch eine Substitution wird die Differentialgleichung 2-ter Ordnung zu einem Differentialgleichungssystem mit 2 Differentialgleichungen 1-ter Ordnung umgeformt.

Substitution:

$$x_1 = x$$

$$x_2 = \dot{x}_1 = \dot{x}$$

$$\dot{x}_2 = \ddot{x}$$

$$x_3 = y$$

$$x_4 = \dot{x}_3 = \dot{y}$$

$$\dot{x}_4 = \ddot{y}$$

Durch den Ansatz der Substitution wird nun das Differentialgleichungssystem 1-ter Ordnung aufgestellt.

$$\dot{x}_2 = \frac{5}{7} * g_{px} * \alpha \quad (3.10)$$

$$\dot{x}_4 = \frac{5}{7} * g_{px} * \beta \quad (3.11)$$

Durch Umschreiben dieses Differentialgleichungssystems in Vektor-Matrix-Schreibweise entsteht die Zustandsraumdarstellung.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \frac{5}{7} * \begin{bmatrix} 0 & 0 \\ g_{px} & 0 \\ 0 & 0 \\ 0 & g_{px} \end{bmatrix} * \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Da die Position des Balls als Ausgangsgröße geregelt wird und das System nicht sprungfähig ist, ergibt sich für die Messgleichung des Zustandsraummodells:

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Vereinfachung durch separate Betrachtung der Achsen

Bei dem aufgestellten Zustandsraummodell handelt es sich um ein Multi-Input, Multi-Output (MIMO) System. Um die mathematische Komplexität zu minimieren wird die X- und Y- Achsen separat betrachtet. Dies ist möglich, weil das Kugelgelenk der Stützstange im Koordinatenursprung sitzt und die Aktoren über die Verbindungstangen direkt auf den Achsen des Plattenkoordinatensystems wirken. Durch diese mechanische Anordnung der Verbindungsstangen wird bei einer Stellgrößenänderung der Aktoren nur der Plattenwinkel α beziehungsweise β beeinflusst. Durch diese vereinfachte Betrachtungsweise des MIMO-Systems entstehen zwei Single-Input, Single-Output- (SISO) Systeme.

3.2. Entwurf des Zustandsreglers

Die Vorteile der Zustandsraumdarstellung wird bei Betrachtung des Zustandsreglers deutlich. Im Gegensatz zur klassischen Regelung wird nicht nur die Ausgangsgröße y zurückführt, sondern auch der interne Zustand der gesamten Regelstrecke [18]. In der folgenden Abbildung 10 wird das Prinzip der Zustandsregelung verdeutlicht. Der Zustandsvektor x beinhaltet die internen Zustände der Regelung. Bei \underline{A} , \underline{b} und \underline{c}^T handelt

es sich Matrizen und Vektoren, welche die Regelstrecke beschreiben¹². Über die Rückkopplung mittels des transponierten Regelvektors \underline{r}^T und des Vorfilters V wird auf Basis einer Sollwertvorgabe w die Stellgröße u berechnet. Die Stellgröße u wirkt auf die Regelstrecke und bewirkt eine Änderung der Ausgangsgröße y ¹³. Diese Zusammenhänge werden im Folgenden mathematisch erläutert.

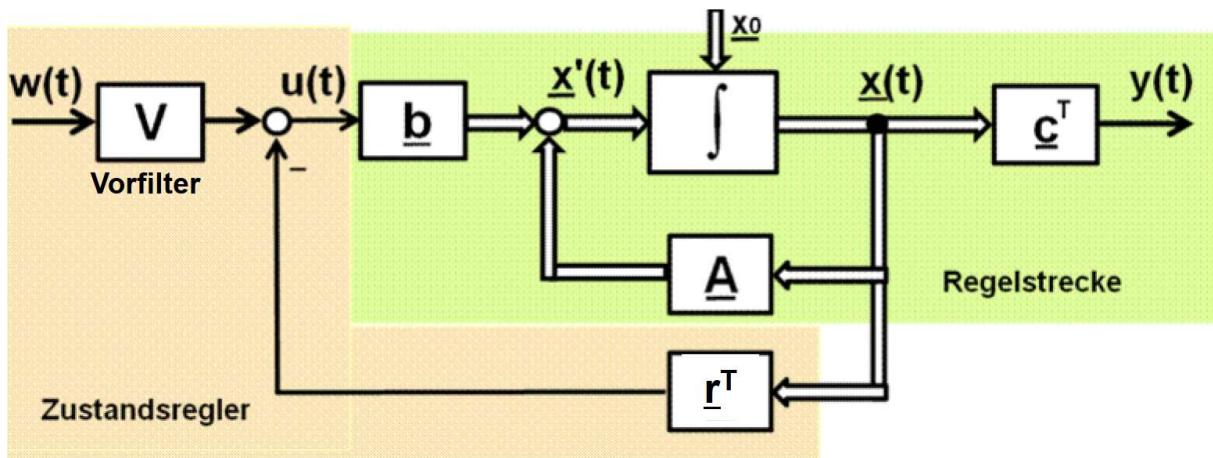


Abbildung 10: Zustandsregelung¹⁴

Aus der klassischen Regelstrecke folgt:

$$\text{Zustandsgleichung: } \dot{\underline{x}} = \underline{A} * \underline{x} + \underline{b} * u \quad (3.12)$$

$$\text{Messgleichung: } y = \underline{c}^T * \underline{x} \quad (3.13)$$

Es folgt die Zustandsrückführung und das Regelgesetz:

$$\text{Zustandsrückführung: } \underline{r}^T * \underline{x} \quad (3.14)$$

$$\text{Regelgesetz: } u = V * w - \underline{r}^T * \underline{x} \quad (3.15)$$

$$\text{(Regelgesetz: } u = V * w - \underline{r}^T * \underline{x}$$

$$(3.15)) \text{ in (Zustandsgleichung: } \dot{\underline{x}} = \underline{A} * \underline{x} + \underline{b} * u$$

$$(3.12)) \text{ eingesetzt: }$$

¹² Bei groß geschrieben Buchstaben handelt es sich um Matrizen

¹³ Die Erläuterung bezieht sich auf ein SISO System

¹⁴ Quelle: Prof Dr. Stefan Werling Skript, RT_I 1 Zustandsregelung

$$\dot{\underline{x}} = [\underline{A} - \underline{b} * \underline{r}^T] * \underline{x} + \underline{b} * V * w \quad (3.16)$$

Es entsteht die Systemmatrix \underline{A}_R der Regelung:

$$\underline{A}_R = \underline{A} - \underline{b} * \underline{r}^T \quad (3.17)$$

$$\underline{A}_R = \begin{bmatrix} 0 & 1 \\ -\frac{5}{7} * g * r_1 & -\frac{5}{7} * g * r_2 \end{bmatrix} \quad (3.18)$$

Wahl der Polstellen

Die Polstellen werden zunächst mit negativem Realteil frei gewählt, um ein stabiles System zu gewährleisten. Durch eine empirische Beobachtung ergaben sich die Polstellen [-2, -2].

Bestimmung des Regelvektor

Zur Bestimmung des Regelvektors wird die charakteristische Gleichung aufgestellt. Dies liefert das charakteristische Polynom C_g mit den Koeffizienten $a_{r,i}$.

$$C_g = \det(s * \underline{I} - \underline{A}_R) = a_{r,0} + a_{r,1} * s + \dots + a_{r,n-1} * s^{n-1} + s^n$$

$$\det(s * \underline{I} - \underline{A}_R) = \det \begin{pmatrix} s & -1 \\ \frac{5}{7} * g_{px} * r_1 & s + \frac{5}{7} * g_{px} * r_2 \end{pmatrix}$$

Daraus folgt:

$$C_g = s^2 + s * \frac{5}{7} * g_{px} * r_2 + \frac{5}{7} * g_{px} * r_1 \quad (3.19)$$

Anschließende werden die gewünschten Polstellen s_i festgelegt. Es entsteht ein neues Polynom $c_{g,wunsch}$ mit den Koeffizienten \bar{a}_i :

$$\begin{aligned} c_{g,wunsch} &= \bar{a}_0 + \bar{a}_1 * s + \dots + \bar{a}_{n-1} * s^{n-1} + s^n \\ c_{g,wunsch} &= (s - s_1)(s - s_2) = s^2 - s * (s_1 + s_2) + s_1 * s_2 \end{aligned} \quad (3.20)$$

Durch Koeffizientenvergleich beider Polynome ((3.19) und (3.20)) können die Koeffizienten $a_{r,i}$ des Regelvektors \underline{r}^T bestimmt werden.

$$s^0: r_1 = s_1 * s_2 * \frac{7}{5 * g_{px}}$$

$$s^1: r_2 = -(s_1 + s_2) * \frac{7}{5 * g_{px}}$$

Weil $s_1 = s_2$ ist, ergibt sich:

$$r_1 = s_1^2 * \frac{7}{5 * g_{px}}$$

$$r_2 = -2 * s_1 * \frac{7}{5 * g_{px}} \quad (3.21)$$

Aufstellen der Regelmatrix

Bei der Vereinfachung des Zustandsraummodells durch die Separierung der Achsen werden beide SISO-Systeme in einer Vektor-Matrix-Darstellung beschrieben. Die gemeinsame Darstellung zweier Systeme in einer Vektor-Matrix-Darstellung hat zur Folge, dass die Regelvektoren beider Systeme in einer gemeinsamen Matrix zusammengefasst werden müssen. Es gilt, wie bei der Differentialgleichung, dass beide Regelvektoren auf den gleichen kinematischen Gesetzen basieren, weshalb sie identisch sind. Es ergibt sich die Regelmatrix R :

$$R = \begin{bmatrix} -r_1 & 0 \\ -r_2 & 0 \\ 0 & -r_1 \\ 0 & -r_2 \end{bmatrix}$$

Berechnung des Vorfilters

Der Vorfilter wird so dimensioniert, dass für die Zeit $t \rightarrow \infty$ die Regelabweichung $w - y$ gleich Null wird. Daraus resultiert $\dot{x} = 0$. Aus Gleichung (3.12) folgt:

$$0 = [\underline{A} - \underline{b} * \underline{r}^T] * \underline{x} + \underline{b} * V * w$$

$$\underline{x} = -[\underline{A} - \underline{b} * \underline{r}^T]^{-1} * \underline{b} * V * w$$

In Gleichung (4.5) einsetzen:

$$y = -\underline{c}^T * [\underline{A} - \underline{b} * \underline{r}^T]^{-1} * \underline{b} * V * w$$

Aus $y = w$ folgt:

$$-\underline{c}^T * [\underline{A} - \underline{b} * \underline{r}^T]^{-1} * \underline{b} * V = I$$

Es ergibt sich für den Vorfilter:

$$V = \left[-\underline{c}^T * [\underline{A} - \underline{b} * \underline{r}^T]^{-1} * \underline{b} \right] \quad (3.22)$$

Die konkrete Berechnung vom Vorfilter und des Regelvektors wird Programmcode realisiert (siehe Kapitel 4.6.2 Programmimplementierung der Regelung).

Über das Regelgesetz (Regelgesetz:

$$u = V * w - r^T * \underline{x}$$

(3.15)) lässt sich nun der Plattenwinkel bei gegebenem Zustandsvektor und gegebener Führungsgröße ermittelt. Die Datenerhebung für den Zustandsvektor wird in Kapitel 5 erklärt.

Regelgesetz:

$$u = V * w - r^T * \underline{x} \quad (\text{Re-})$$

gelgesetz:

$$u = V * w - r^T * \underline{x} \quad (3.15))$$

Die Lösung u des Regelgesetzes beschreibt die Plattenwinkel.

4. Software

Das folgende Kapitel behandelt den Schwerpunkt dieser Studienarbeit: Die Software. Zur Programmierung wird die Programmiersprache Python verwendet. Die Gründe dafür sind, wie bereits erwähnt, die zahlreich verfügbaren Bibliotheken und die Benutzerfreundlichkeit von Python. Die in dieser Arbeit benutzten Bibliotheken werden in Kapitel 4.1 vorgestellt. Des Weiteren sind bereits Vorkenntnisse im Bereich der Programmierung mit Python vorhanden. Es wird sich für die kostenfreie Programmierumgebung (IDE, Integrated Developer Environment) PyCharm in der Community-Edition entschieden, da sie viele nützliche Funktionen, wie zum Beispiel ein Debugging-Tool, Syntaxprüfung und Codevervollständigung besitzt [19].

Um effizient mit mehreren Leuten an einem gemeinsamen Programmcode zu arbeiten, wird die Plattform GitHub verwendet. GitHub basiert auf dem Versionsverwaltungssystem Git und ermöglicht das gleichzeitige Bearbeiten von Programmcode an verschiedenen Stellen. Änderungen können zu einer Version zusammengeführt werden und durch gesetzte Checkpoints können vorherige Versionen wiederhergestellt werden. PyCharm besitzt zudem ein Plugin, mit dem Git und GitHub in die IDE integriert werden können [20].

4.1. Verwendete Bibliotheken

In dieser Studienarbeit werden eine Vielzahl von Bibliotheken verwendet, die bereits programmierte Algorithmen, Funktionen, Befehle und Klassen zur Verfügung stellen [21]. Diese Bibliotheken und deren Aufgaben sind im Folgenden kurz erläutert.

4.1.1. openCV

openCV ist eine Bibliothek, die ein riesiges Repertoire an Möglichkeiten in der Bildverarbeitung bietet. Sie findet Anwendung in verschiedensten Bereichen, darunter medizinische Bildanalyse, Zusammenfügen von Straßennetzbildern, Detektion und Erkennung von Gesichtern, Verfolgung von Objekten, Extraktion von 3D-Modellen und viele mehr [13].

In dieser Projektarbeit werden unter anderem Bildverarbeitungsoperatoren (Filter und Transformationen) sowie Methoden zur Anzeige aus der Bibliothek verwendet. Des Weiteren wird die openCV-Kameraschnittstelle in einer eigenen Webcam-Klasse integriert, was in Kapitel 4.5.1 genauer erläutert wird.

4.1.2. **Numpy**

Der Name „Numpy“ steht für Numerisches Python. Dementsprechend bietet die Bibliothek zahlreiche Funktionen und Klassen die das Rechnen und Arbeiten mit mathematischen Operatoren erleichtern, aber auch effizienter machen [22].

Im Programmcode der Studienarbeit werden beispielsweise die trigonometrischen Funktionen Sinus, Cosinus und Tangens sowie die von Numpy bereitgestellten Arrays (Numpy-Arrays) verwendet.

4.1.3. **pigpio**

pigpio ist eine Python-Bibliothek, die die Kontrolle der GPIO-Pins des Raspberry Pis ermöglicht [23]. In der Studienarbeit ersetzt sie die zuvor genutzte Bibliothek “RPi.GPIO” und wird zur PWM-Signalerzeugung und zur Fernsteuerung der GPIO-Pins genutzt. In Kapitel 4.7.3 wird dies genauer erläutert.

4.1.4. **Matplotlib**

Matplotlib ist eine Bibliothek zum Visualisieren von Daten. Mit Hilfe von Matplotlib lassen sich Graphen und Diagramme in beliebiger Form darstellen. Verwendung hat dies im Zuge der Studienarbeit hauptsächlich zur Fehleranalyse durch die graphische Darstellung der Ergebnisse. Im funktionalen Programmcode findet die Bibliothek jedoch keine Anwendung.

4.1.5. Weitere verwendete Bibliotheken

Aus der Bibliothek imutils wird ein FPS-Zähler (FPS = Frames Per Second) importiert. Dieser wird lediglich benutzt, um Fehlerursachen zu ermitteln und Lösungsvorschläge zu validieren.

Die drei Importe von win32api, win32process und win32con erlauben es, die Prozesspriorität unter Windows, also die Priorität des ausgeführten Programmcodes, zu ändern. Durch Festlegen einer höheren Prozesspriorität wird das Programm von weniger anderen Prozesses unterbrochen, was eine bessere Performance ermöglicht [24].

Der Import der time-Bibliothek stellt Befehle zur exakten Zeiterfassung zur Verfügung, was bei der Zyklusdauer- und der Geschwindigkeitsberechnung von Bedeutung ist.

4.2. Programmstruktur

Um das Programm zu strukturieren, gibt es eine Hauptdatei, aus welcher das Programm gestartet wird, und mehrere ausgelagerte separate Dateien. In der Hauptdatei wird der grundlegende Programmzyklus bearbeitet. Um diese Datei übersichtlich zu halten, werden die größeren Teile Balldetektion, Plattendetektion und Plattenregelung sowie die Klasse der Webcam in separaten Dateien ausgelagert. Tabelle 2 zeigt eine Übersicht der separaten Dateien sowie eine Liste der enthaltenen Funktionen beziehungsweise Inhalte.

Dateiname	Inhalt/Funktion
ball_detection.py	- Detektion des Balls
plate_detection.py	- Detektion der Platte - Perspektivische Transformation
plate_control.py	- Berechnung von Regelmatrix und Vorfilter - Berechnung der Ballgeschwindigkeit - Berechnung der Aktorstellung mithilfe der Approximation
Devices.py	- Webcam Klasse

Tabelle 2: Liste der ausgelagerten Dateien und ihre Funktionen

4.3. Programmablauf

In diesem Kapitel wird der Programmablauf anhand eines Flussdiagramms dargestellt und erklärt. Eine detaillierte Erläuterung der verwendeten Algorithmen, und weshalb diese angewandt werden, erfolgt in den kommenden Kapiteln.

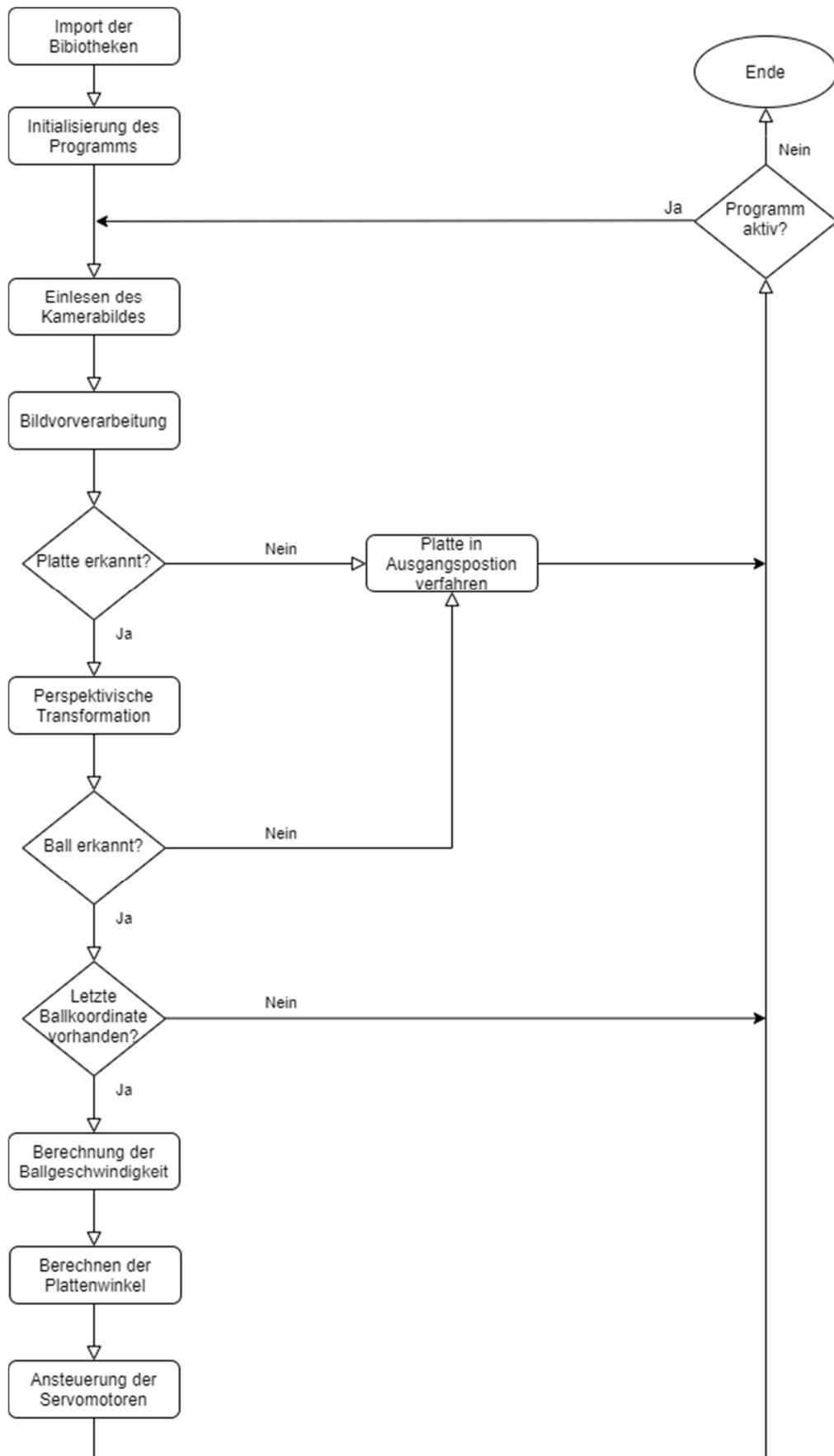


Abbildung 11: Flussdiagramm Programmablauf

In der Hauptdatei und den ausgelagerten Dateien werden im ersten Schritt die benötigten Bibliotheken importiert. Im nächsten Schritt der Initialisierung werden die Parameter und Konstanten definiert, der Vorfilter und die Regelmatrix berechnet, sowie die Prozesspriorität erhöht.

In einer While-Schleife, dem sogenannten Prozesszyklus, wird im ersten Schritt das Kamerabild eingelesen und mit einem Zeitstempel versehen. In der Bildvorverarbeitung werden hochfrequente Störanteile mittels eines Tiefpassfilters minimiert.

Anschließend wird das Bild in den HSV-Farbraum transformiert, in dem über Farbsegmentierung mittels zweier Schwellwertbereiche zwei Binärbilder erzeugt werden. In diesen Binärbildern werden die Platte und der Ball über das Finden der größten Kontur detektiert. Sollten Platte oder Ball nicht erkannt werden, verfährt die Platte in ihre Ausgangsstellung (Plattenwinkel in X- und Y-Achse auf 0°).

Die Position des Balls wird zunächst im Kamerakoordinatensystem ermittelt und anschließend über eine perspektivische Transformation in das Plattenkoordinatensystem umgerechnet. Im nächsten Schritt wird die Ballgeschwindigkeit auf Basis der Zeitstempel und der letzten Ballkoordinate berechnet. Sollte die letzte Ballkoordinate nicht vorhanden sein, erfolgen keine weiteren Schritte und der Prozesszyklus beginnt von vorne. Auf Grundlage der ermittelten Ballposition und Ballgeschwindigkeit werden mit Hilfe des Regelgesetzes die Plattenwinkel berechnet. Die Plattenwinkel werden anschließend über Approximationsfunktionen in die jeweiligen PWM-Signale der Servomotoren umgerechnet. Diese PWM-Signale werden von dem Raspberry-Pi erzeugt, welcher von dem Desktop-PCs ferngesteuert wird. Solange der Befehl zum Programmabbruch nicht initialisiert wird, durchläuft das Programm den Prozesszyklus.

4.4. Programminitialisierung

Bei Programmstart werden zu Beginn alle benötigten Bibliotheken mithilfe des `import`-Befehls importiert.

```
# Bibliotheken imports
import cv2
from imutils.video import FPS
import time
import numpy as np
import matplotlib.pyplot as plt
import pigpio
import win32api, win32process, win32con
```

Eine Beschreibung der Inhalte der Bibliotheken ist in Kapitel 4.1 aufgeführt. Anschließend werden mit dem gleichen Befehl alle Funktionen der ausgelagerten Dateien importiert. Auch hier ist eine Dokumentation mit Inhalt und Aufgabe in Kapitel 4.2 vorhanden.

```
# Projekt imports
import ball_detection
import plate_detection
import plate_control
from devices import Webcam
```

Die folgenden Zeilen erhöhen die Prozesspriorität unter Windows, um die Performance des Programms zu verbessern. Der Grund für die Erhöhung der Prozesspriorität wird in Kapitel 4.8 beschrieben.

```
# Setze Prozesspriorität auf "Real-Time"
process_id = win32api.GetCurrentProcessId()
handle = win32api.OpenProcess(win32con.PROCESS_ALL_ACCESS, True, process_id)
win32process.SetPriorityClass(handle, win32process.REALTIME_PRIORITY_CLASS)
```

Im Anschluss werden alle äußerlichen Parameter, wie Seitenlänge der Platte und Naturkonstanten, gesetzt. Des Weiteren werden veränderliche Parameter, wie die Wunschpolstellen der Regelung, der Radius der vorgegebenen Kreisbahn und die Bahngeschwindigkeit (in Kreisbahnen pro Sekunde), festgelegt.

```
# Systemparameter
pole = [-2.0, -2.0]
n = 0.5
r = 200

# Konstante Parameter
L = 300.
g_mm = 9810.
omega = 2*np.pi * n
```

Bevor eine Regelung beginnen kann, müssen mit den festgelegten Polstellen die Regelmatrix und der Vorfilter berechnet werden. Hierfür wird die, in der ausgelagerten Datei „plate_control.py“ geschriebene, Funktion `get_state_space` benutzt.

```
R, V = plate_control.get_state_space(g_mm, pole)
```

Damit im Bereich der Bildverarbeitung der Ball (blau) auf der Platte (orange) erkannt werden kann, müssen die Farbbereiche definiert sein. Die Farbbereiche werden jeweils durch zwei eingrenzende Schwellwerte beschrieben. Diese Schwellwertbereiche werden in der Initialisierung in Tripeln fest definiert, können jedoch nach Bedarf schnell über Schieberegler an sich verändernde Lichtverhältnisse angepasst werden.

```
activate_slider = False
if activate_slider:
    cv2.namedWindow("Slider")
    cv2.resizeWindow("Slider", 600, 600)
    cv2.createTrackbar("Lower H", "Slider", 0, 180, do_nothing)
    cv2.createTrackbar("Lower S", "Slider", 0, 255, do_nothing)
    cv2.createTrackbar("Lower V", "Slider", 0, 255, do_nothing)
    cv2.createTrackbar("Upper H", "Slider", 180, 180, do_nothing)
    cv2.createTrackbar("Upper S", "Slider", 255, 255, do_nothing)
    cv2.createTrackbar("Upper V", "Slider", 255, 255, do_nothing)
else:
    orangeLower = (13, 170, 127)
    orangeUpper = (17, 255, 255)
```

Als nächstes wird die Kamera gestartet, indem eine Instanz der Webcam-Klasse erzeugt wird (genauere Details zu dieser sind in Kapitel 4.5.1). Die anschließende Wartezeit durch den Befehl `time.sleep(0.1)` gibt der Kamera Zeit zum Starten und Senden eines ersten Bildes. Somit wird verhindert, dass der Programmzyklus ohne Anfangsbild startet, was einen Fehler durch nicht initialisierte Variablen verursachen würde.

```
cam = Webcam(src=0, use_thread=False)
time.sleep(0.1)
```

Um die Servos zu steuern, muss vom Desktop-PC aus auf den Raspberry Pi zugegriffen werden. Wie bereits in Kapitel 2.2.3 erwähnt, werden dafür die GPIO-Pins 20 und 21 als Output verwendet. Zur Übersichtlichkeit werden die beiden Pins den jeweiligen Servomotoren per Variablenamen mit enthaltener Koordinatenrichtung zugeordnet. Der Zugriff erfolgt mithilfe der pigpio Bibliothek. Im Detail wird dies in Kapitel 4.7.3 beschrieben.

```
ServoX = 20
ServoY = 21
pi = pigpio.pi("192.168.0.3")
pi.set_mode(ServoX, pigpio.OUTPUT)
pi.set_mode(ServoY, pigpio.OUTPUT)
```

Im letzten Schritt der Initialisierung werden die im Programmzyklus verwendeten Zeit- und Positionsvariablen auf ihre Startwerte gesetzt, um auch hier einen Fehler durch nicht initialisierte Variablen zu verhindern. Dies ist nur für den ersten Programmdurchlauf notwendig, da die Variablen in jedem Programmzyklus nach ihrer Verwendung überschrieben werden.

```
frame_time_last = time.time()
ball_coords_last = None
```

4.5. Bildverarbeitung

Die digitale Bildverarbeitung beinhaltet eine Vielzahl an Vorgängen, mit dem Ziel der Informationsgewinnung aus einem Bild oder einer Folge von Bildern. Die Bildverarbeitung umfasst unter anderem [25]:

- Bildaufnahme
- Bildbearbeitung
- Bildtransformation
- Bildauswertung
- Bildkomprimierung

Ziel der Bildverarbeitung in dieser Projektarbeit ist es, die Platte und den Ball zu detektieren und dem Ball eine Position auf der Platte zuzuordnen.

Die Bildverarbeitung findet hierfür in mehreren Prozessen statt. Zur Extraktion der relevanten Daten wird eine Kette von Bildverarbeitungsoperationen durchgeführt.

Dadurch entsteht ein hierarchisches Schema (siehe Abbildung 12). Es gibt einen Überblick über die durchgeführten Schritte der Bildverarbeitung zur Extraktion der relevanten Informationen, welche für die Datenerhebung des Zustandsvektors erforderlich sind [26].

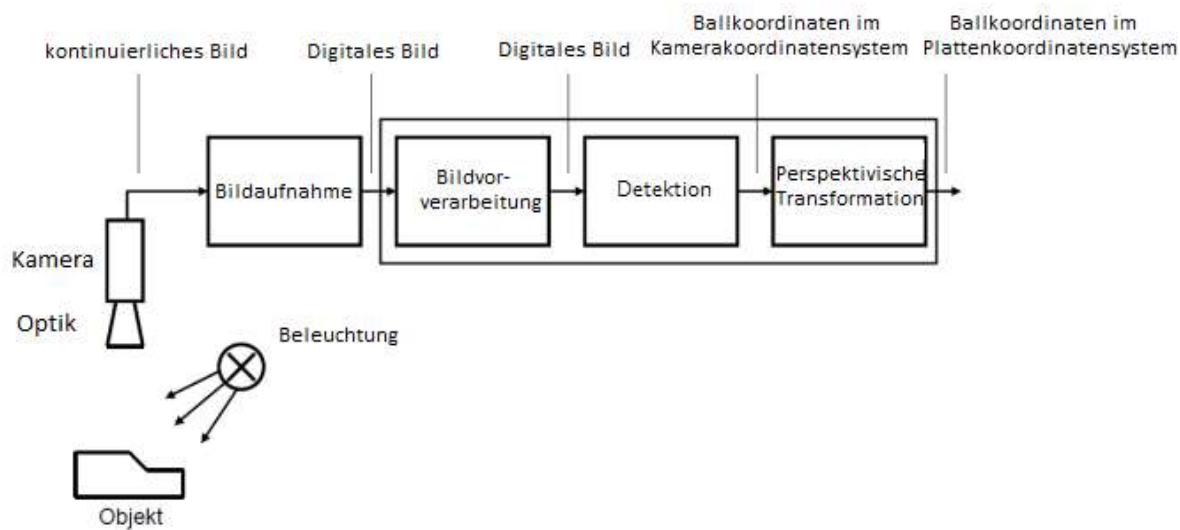


Abbildung 12: Hierarchischer Ablauf der Bildverarbeitung¹⁵

Zu Beginn jeder Bildverarbeitung steht die Bildaufnahme. Um die interessanten Objekteigenschaften abzubilden, werden ein geeignetes Bildaufnahmesystem, die Beleuchtung, sowie der Wellenlängenbereich und viele weitere Optionen gewählt [26]. Nach der Bildaufnahme folgt die erste Verarbeitungsstufe, die Bildvorverarbeitung. Durch einen Binomialfilter wird gleichmäßig verteilt Rauschen entfernt. Anschließend wird das Ergebnisbild in den HSV-Farbraum transformiert, um bestimmte Flächen und Elemente im Bild mittels ihrer Farbwerte zu erkennen.

In der Detektion werden zwei Binärbilder über eine Farbsegmentierung mittels Schwellwertvorgabe der Farbwerte im HSV-Farbraum erzeugt. Diese Binärbilder stellen jeweils nur die Platte beziehungsweise nur den Ball dar. Durch morphologische Operationen

¹⁵ Abbildung angelehnt an Präsentation: Messen mit optischen Systemen, Teil 1: Bildverarbeitung von Prof. Dr. –Ing. Jörn Korthals (2018) Folie 5 [57]

werden Fehler im Bild minimiert und übriges Salz- und Pfeffer-Rauschen entfernt. Anschließend wird die größte Kontur mittels eines Boundary-Follower-Algorithmus⁴ und des Satzes von Green erkannt und somit Platte beziehungsweise Ball detektiert. Um die exakte Position des Balls zu ermitteln, wird der Mittelpunkt des Balls über die Bestimmung von Momenten berechnet. Nach der Bestimmung des Mittelpunktes sind die Ballkoordinaten im Kamerakoordinatensystem bekannt.

Um dem Ball eine genaue Position relativ zur Platte zuzuweisen, wird über eine perspektivische Transformation ein neuer Frame erzeugt, welcher die Platte entzerrt darstellt. Über eine Transformationsmatrix wird die im aufgenommenen Frame bestimmte Position des Balls in das Plattenkoordinatensystem umgerechnet. Nach der Bildverarbeitung wird auf Basis der Ballkoordinaten im Plattenkoordinatensystem und den Zeitstempeln der einzelnen Frames die Geschwindigkeit des Balls bestimmt. Mit diesen Informationen wird dann durch das Zustandsraummodell der Plattenwinkel bestimmt. Die einzelnen Schritte der Bildverarbeitung werden im Folgenden näher erläutert.

4.5.1. **Bildaufnahme**

Die Bildaufnahme erfolgt mit Hilfe einer handelsüblichen Webcam von Logitech (Modell: C930e), die über eine USB-Schnittstelle an den PC angeschlossen ist (vgl. 2 Modellaufbau). Die Webcam arbeitet im Wellenlängenbereich des sichtbaren Lichts.

Der Zugriff auf die Webcam im Programmcode lässt sich über die VideoCapture Schnittstelle (API) der Bibliothek openCV realisieren [27]. Diese API bietet grundlegende Kommandos an, wie zum Beispiel das Einlesen eines Frames (Einzelbildes) oder das Einstellen von verschiedenen Kamera-Parametern. Die Auflösung der Kamera wird auf 640x480 Pixel (Breite x Höhe) gesetzt, da dies für diese Anwendung ausreichend ist. Die Bildwiederholungsrate ist auf 30 Bilder pro Sekunde (FPS) festgesetzt. Da sich die Platte immer im gleichen Entfernungsbereich zur Kamera befindet, wird der Autofokus deaktiviert.

Um die grundlegende Schnittstelle von openCV um mehrere, für dieses Projekt relevante, Funktion zu erweitern, wird eine neue „Webcam“-Klasse erstellt, welche eine

Erweiterung der openCV-Klasse darstellt. Die Webcam-Klasse befindet sich in der ausgelagerten Datei „devices.py“. Dort können zu einem späteren Zeitpunkt Klassen für weitere Geräte (z.B. Servomotoren) hinzugefügt werden.

Die Erweiterungen der Webcam-Klasse gegenüber der openCV-Schnittstelle belaufen sich auf das Hinzufügen eines Zeitstempels und die Option, die Bildaufnahme auf einem separaten Thread auszuführen.

Der Zeitstempel ordnet jedem Frame zusätzlich zu den Bildinformationen eine zeitliche Information zu. Direkt nach Ankommen des Frames an dem PC merkt sich das Programm den Zeitpunkt, welcher in Form des Zeitstempels an den Frame angehängt wird. Der Zeitstempel ist für die Regelung relevant und dient unter anderem zur Geschwindigkeitsberechnung.

Der Grund für die Nutzung eines separaten Threads zur Bildaufnahme wird im Folgenden erläutert. Der Zugriff auf die jeweiligen Frames lässt sich als ein Warten auf ein externes Event (in diesem Fall ein Event von der Webcam) beschreiben und dauert vergleichsweise lange. In dieser Zeit steht das Programm still und führt nichts anderes aus. Ein Thread ist in diesem Kontext ein separater Arbeitsschritt, der parallel zu dem restlichen Programmcode abgearbeitet werden kann. Durch Auslagern der Bildaufnahme auf einen Thread ist nur der Thread mit dem Warten beschäftigt, während der Rest des Programms mit anderen Berechnungen fortfahren kann [28].

Der Vorteil dabei ist eine schnellere Durchlaufzeit des Programmzyklus' durch die parallele Abarbeitung, was prinzipiell mehr Verarbeitungsschritte pro Sekunde ermöglicht. Der Nachteil ist, dass beide Prozesse asynchron zueinander ablaufen und in bestimmten Fällen ein Frame benutzt wird, auf den bereits vor einer längeren Zeit zugriffen wurde.

Implementierung im Programmcode

Die Implementierung der Webcam-Klasse im Programmcode wird im Folgenden beschrieben. Bei Initialisierung der Klasse wird der Konstruktion aufgerufen. Im Konstruktor werden für die Klasse relevanten Variablen definiert und Parameter übergeben.

In Python ist der Konstruktor die Methode `__init__`, welche im Folgenden für die Webcam-Klasse dargestellt ist:

```
class Webcam:  
    def __init__(self, src=0, use_thread=False):  
        self.stream = cv2.VideoCapture(src)  
        _, self.frame = self.stream.read()  
  
        self.use_thread = use_thread  
        self.frame_time = None  
        self.stopped = False  
  
        if self.use_thread:  
            t = Thread(target=self.update, name="Webcam-Thread", args=())  
            t.start()
```

Die Klasse wird initialisiert mit den Parametern `src` und `use_thread`. `src` gibt den Index der Kamera an, auf welche zugegriffen wird. Da nur die Logitech-Webcam angeschlossen ist, ist dieser standardmäßig `0`. Falls mehrere Kameras angeschlossen sind, werden diese in aufsteigender Reihenfolge indiziert. Die zweite Übergabevariable ist `use_thread`. Wenn `use_thread=True` ist, wird die Bildaufnahme auf einem separaten Thread ausgeführt, indem im unteren Teil des Konstruktors die Instanz `t` der Thread-Klasse von Python initialisiert wird. Diese Thread-Instanz führt die Funktion `self.update` aus, was durch den Parameter `target` festgelegt wird. Diese Methode läuft parallel zum Rest des Programmes ab.

In den oberen Zeilen der Initialisierung wird über die bereits erwähnte openCV-Schnittstelle ein Kamera-Klasse initialisiert. Auf diese Kamera-Klasse kann über die Variable `self.stream` zugegriffen werden, so zum Beispiel in der nächsten Zeile mit `self.stream.read()`. An dieser Stelle wird ein erster Frame eingelesen.

Die Methode `update` wird in der Initialisierung der Thread-Klasse aufgerufen und läuft parallel zum Rest des Programms dauerhaft im Hintergrund. In der Methode ist eine Dauerschleife definiert, in der kontinuierlich neue Frames eingelesen werden. Zusätzlich wird zu jedem Frame der erwähnte Zeitstempel in Form der Variable `self.frame_time` über die Funktion `time.time()` eingelesen. Zu beachten ist, dass an dieser Stelle weder Frame noch Zeitstempel von der Klasse ausgegeben wird. Falls die

Klassenvariable `self.stopped` gesetzt wird, wird die Schleife unterbrochen und die Methode beendet.

```
def update(self):
    while True:
        if self.stopped:
            return

        _, self.frame = self.stream.read()
        self.frame_time = time.time()
```

Die Methode `read` gibt bei Aufruf den aktuellen Frame und den Zeitstempel zurück. Falls ein separater Thread genutzt wird, werden an dieser Stelle der aktuelle Frame und der aktuelle Zeitstempel ausgegeben. Falls kein zusätzlicher Thread genutzt wird, erfolgt zuvor das direkte Einlesen des Frames und des Zeitstempels.

Die Methode `stop` dient lediglich zum Beenden der `update`-Methode, indem die Variable `self.stopped` auf `True` gesetzt wird.

```
def read(self):
    if not self.use_thread:
        self.frame = self.stream.read()
        self.frame_time = time.time()

    return self.frame, self.frame_time
```

Im Hauptprogramm wird die Webcam-Klasse über folgenden Befehl initialisiert. Dort werden die Parameter zur Kameraauswahl und zur Threadnutzung festgelegt:

```
cam = Webcam(src=0, use_thread=False)
```

Am Desktop-PC konnten ohne die Nutzung eines separaten Threads bessere Ergebnisse erzielt werden. Im Programmzyklus werden mit folgendem Aufruf

```
frame, frame_time = cam.read()
```

regelmäßig neue Frames und Zeitstempel eingelesen. Bei Beenden des Programms wird die Methode `stop()` aufgerufen. Dies hat nur einen Effekt, falls zuvor die `update`-Methode durch einen separaten Thread genutzt wurde.

```
def stop(self):
    self.stopped = True
```

4.5.2. Bildvorverarbeitung

4.5.2.1 Anwendung des Binomialfilters

Nachdem der Frame eingelesen ist, wird im ersten Schritt der Bildvorverarbeitung das gleichmäßig verteilte Bildrauschen minimiert. Beim Bildrauschen handelt es sich um Störungen im Bild, welche keinen Bezug zum eigentlichen Bildinhalt haben [29]. Die störenden Pixel sind Anteile, welche sich bei einer Fouriertransformation des Bildes in den hohen Ortsfrequenzen niederschlagen und heißen deshalb hochfrequente Anteile.

Zur Minimierung des Rauschens wird ein Tiefpassfilter verwendet. Hierbei handelt es sich um einen linearen Glättungsoperator, welcher das Bild glättet. Das heißt es werden Grauwertkanten und -spitzen, sowie Rauschen im Bild reduziert. Übrig bleiben niederfrequente Anteile, in denen es weniger Schwankungen im Grauwert gibt [25] . Das folgende Bild verdeutlicht, wie durch die Anwendung eines linearen Glättungsoperators das sich in den hohen Ortsfrequenzen niederschlagende Bildrauschen minimiert wird.

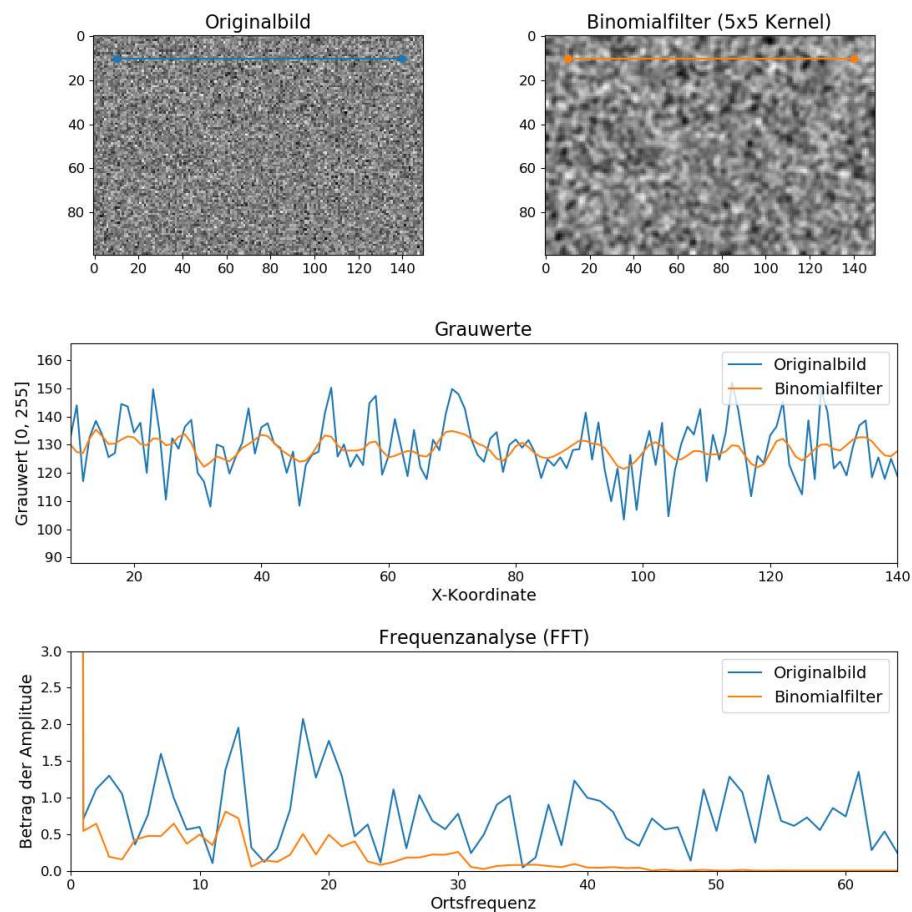


Abbildung 13: Anwendung eines Binomialfilters

Lineare Glättungsoperatoren werden über die Größe und Form der Filterregion und der „Gewichte“ der einzelnen Pixel beschrieben. Die Filterregion bestimmt, welche Pixel im Eingangsbild $g(x, y)$ zur Berechnung des neuen Pixelwert $g'(x, y)$ im Ereignisbild beitragen. Damit bestimmt die Filterregion das räumliche Ausmaß des Filters.

Mittels der Gewichte der Pixel lässt sich der Einfluss einzelner Pixel in der Filterregion auf den neuen Pixelwert $g'(x, y)$ variieren.

Diese Parameter werden allein durch eine Matrix von Filterkoeffizienten spezifiziert, der sogenannten „Filtermatrix“ oder dem „Filterkern“. Aus Symmetriegründen ist der Filterkern meist quadratisch mit einer ungeraden Kantenlänge.

Bei Linearen Filteroperationen handelt es sich um diskrete Funktionen, welche immer von einem Eingangsbild $g(x, y)$ in ein Ergebnisbild $g'(x, y)$ wirkt.

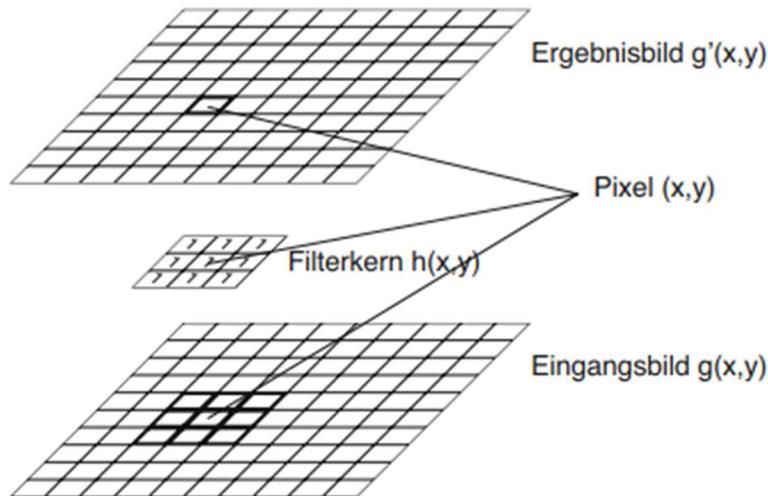


Abbildung 14: Arbeitsweise eines Linearen Filter¹⁶

Lineare Operatoren haben ihren Namen durch die lineare Verknüpfung von Pixelwerte innerhalb der Filterregion [3]. Das heißt für jedes Pixel im Ergebnisbild $g'(x, y)$ werden alle, unter der Filtermatrix sich befindenden, Pixel im Eingangsbildes $g(x, y)$ mit den Filterkoeffizienten gewichtet und aufsummiert [2].

¹⁶Quelle: Angelika Erhardt, Einführung in die Digitale Bildverarbeitung (2008) [25]

Zur Berechnung der Pixel im Ergebnisbild $g'(x, y)$ wird eine Faltung angewendet. Bei einer Faltung handelt es sich um eine mathematische Operation, welche zwei Funktionen gleicher Dimension kontinuierlich oder diskret miteinander verknüpft [30].

Sowohl Eingangsbild als auch Filtermatrix können als zweidimensionale diskrete Funktionen beschrieben werden. Die Faltung zur Berechnung der Pixel im Ergebnisbild wird deshalb mittels einer Faltungssumme und nicht durch ein Faltungsintegral durchgeführt. Hierbei handelt es sich um die diskrete Faltung.

Anschaulich bedeutet die Faltung von Pixel des Eingangsbild mit der Filtermatrix, dass der Pixelwert vom Eingangsbild $g(x, y)$ durch eine Art gewichtetes Mittel von $h(x, y)$ ersetzt wird [30].

Bei der Faltung handelt es sich um eine Kreuzkorrelation mit einer um 180° gedrehten Filtermatrix [25]. Die Faltung einer Bildfunktion g mit einer Filterfunktion h wird im Folgenden näher erläutert.

Im ersten Schritt der Faltung wird die Filterfunktion h über dem Eingangsbild g positioniert, sodass der Koordinatenursprung der Filterfunktion bei $h(0,0)$ auf dem zu ersetzenen Pixel des Eingangsbildes $g(x, y)$ liegt [30]. Anschließend wird die Filtermaske horizontal und vertikal gedreht (Drehung um 180°) und alle Koeffizienten $h(x, y)$ werden mit den unterliegenden Pixeln des Eingangsbildes multipliziert und anschließend aufsummiert. Dieser Vorgang wird für alle Pixel im Eingangsbild durchgeführt. Es entsteht das Ergebnisbild. Die Folgende Abbildung 15 verdeutlicht den Vorgang der Faltung.

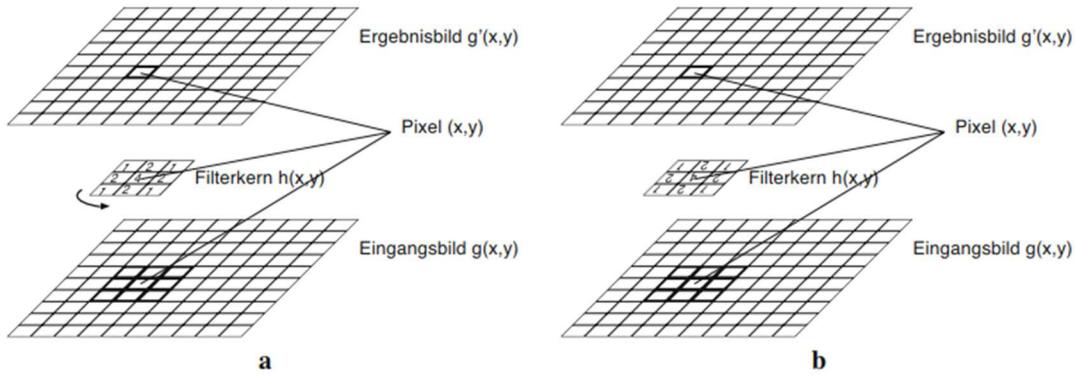


Abbildung 15: Durch drehen der Filtermatrix um 180° macht aus einer Kreuzkorrelation in a) eine Faltung in b)¹⁷

Das Ergebnis einer diskreten Faltung ist die sogenannten „Faltungssumme“. Dieseersetzt im Ergebnisbild den Pixelwert vom Eingangsbild. Bei der Faltung einer 2-dimensionale Bildfunktion mit einer Filtermatrix der Größe $(2m + 1) \times (2m + 1)$ ergibt sich die Faltungssumme [25]:

$$g'(x, y) = \sum_{u,v=-m}^m \sum_{u=-m}^m g(x+u, y+v)h(-u, -v) \quad (4.1)$$

Die Bedeutung einer Faltung basiert auf den mathematischen Eigenschaften. Im „Ortsraum“ oder „Bildraum“ gelten grundlegende Eigenschaften, wie die Kommutativität, Linearität und Assoziativität. Die Eigenschaften und deren Bedeutung näher zu erläutern würde den Rahmen dieser Projektarbeit überschreiten. Es wird auf das Buch „Digitale Bildverarbeitung“ von W. Burger [30], siehe Kapitel 5.3.2, verwiesen.

Als linearer Glättungsoperator wird ein zweidimensionaler Binomialfilter gewählt. Seine Filtermatrix entspricht einer diskreten, zweidimensionalen Gaußfunktion [30]:

$$H^{G,\sigma}(x, y) = e^{\frac{-x^2+y^2}{2\sigma^2}} \quad (4.2)$$

Standardabweichung: σ

¹⁷ Quelle: Angelika Erhardt, Einführung in die Digitale Bildverarbeitung (2008) [25]

Aus diesem Grund wird der Binomialfilter auch Gaußfilter genannt. Der Gaußfilter besitzt im Gegensatz zu anderen Filtern, wie z.B. dem Rechteck-Filter, eine geeignete Gestalt zur homogenen Behandlung aller Frequenzen [31]. Die Gaußfunktion ist die einzige Funktion, welche im Orts- und im Ortsfrequenzraum dieselbe Form hat. Durch diese Eigenschaft hat ein Binomialfilter im Ortsfrequenzraum ein fast perfektes Tiefpassverhalten [25]. Durch das hintereinander Ausführen zweier eindimensionaler Binomialfilter wird zudem eine effiziente Berechnung des Ergebnisbild ermöglicht [30]. Aus den genannten Gründen wird zur Minimierung des gleichmäßig verteilten Rauschens ein Binomialfilter verwendet.

Eine unmittelbare Folge des Assoziativität der Faltung ist, dass sich der zweidimensionalen Binomialfilter durch eine diskrete Faltung eines horizontalen mit einem vertikalen 1D-Binomialfilter berechnen lässt [26]. Durch die separate Betrachtung des zweidimensionalen Filters als zwei eindimensionale Filter h_x und h_y wächst die Zahl der Operationen bei der Berechnung anstatt quadratisch zur Seitenlänge nur linear.

Bei einem 1D-Binomialfilter handelt es sich um eine Filtermaske mit den Werten der diskreten Binomialverteilung. Die resultierende Filtermaske lässt sich nach dem Berechnungsschema des Pascalschen Dreiecks berechnen.

R	f		σ^2
0	1	1	0
1	1/2	1 1	1/4
2	1/4	1 2 1	1/2
3	1/8	1 3 3 1	3/4
4	1/16	1 4 6 4 1	1
5	1/32	1 5 10 10 5 1	5/4
6	1/64	1 6 15 20 15 6 1	3/2
7	1/128	1 7 21 35 35 21 7 1	7/4
8	1/256	1 8 28 56 70 56 28 8 1	2

Abbildung 16: Pascalsches Dreieck¹⁸

¹⁸ Quelle: Bernd Jähne, Digitale Bildverarbeitung (2005) [26]

Hierbei ist R die Ordnung des Binoms, f der Skalierungsfaktor 2^{-R} und σ^2 die Varianz [26]. Der einfachste zweidimensionale Binomialfilter lässt sich durch die Faltung zweier eindimensionaler Binomialfilter der Ordnung $R = 2$ berechnen. Es ergibt sich eine 3×3 Binomialmaske:

$$h = h_x * h_y = \frac{1}{4} [1 \quad 2 \quad 1] * \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.3)$$

Die Filtermatrix kann eine beliebige Größe annehmen. Desto größer die Filtermatrix gewählt wird, umso stärker ist die Glättung, jedoch verschmieren die Kanten stärker und der Rechenaufwand erhöht sich. Da die Berechnung des Ergebnisbildes einen maßgeblichen Einfluss auf die Durchlaufzeit des Prozesszyklus und damit auf die Prozesszyklusrate hat, wird eine kleine 5×5 Binomialmaske angewendet.

Der Binomialfilter wird auf ein Eingangsbild angewendet, welches sich im RGB-Farbraum befindet. Bei dem RGB-Farbraum handelt es sich um einen additiven Farbraum, welcher die Farben als Linearkombination der drei Primärfarben (Rot, Grün, Blau) darstellt [32]. Der RGB-Farbraum kann in Form eines Würfels in einem dreidimensionalen Koordinatensystem visualisiert werden. Siehe folgende Abbildung 17:

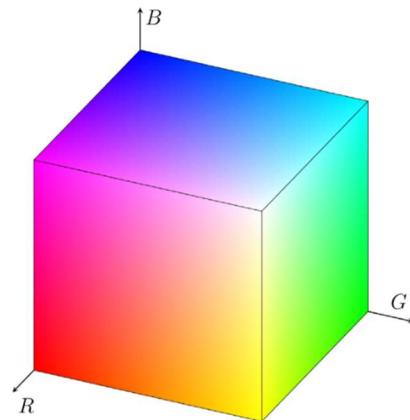


Abbildung 17: RGB Farbwürfel¹⁹

Der Binomialfilter wird auf jede einzelne Dimension, die sogenannten „Farbkanäle“, angewendet und anschließend werden die Farbkanäle wieder zu einem Bild zusammengefügt.

4.5.2.2 Implementierung im Programmcode

Im Programmcode wird die Anwendung des Binomialfilters über die openCV Bibliothek realisiert. Mit der Funktion

```
blurred = cv2.GaussianBlur(frame, (5, 5), 0)
```

lässt sich der Binomialfilter mit einer 5x5 Filtermatrix auf den von der Webcam aufgenommen Frame anwenden. Auf Basis dieses neuen Frames findet die weitere Bildverarbeitung statt.

4.5.2.3 Transformation in den HSV-Farbraum

Zur Detektion der Kontur von Platte und Ball müssen im ersten Schritt Binärbilder erzeugt werden, in den nur die Platte beziehungsweise der Ball dargestellt wird. Zum Erzeugen der Binärbilder wird das Bild über eine sogenannte Farbsegmentierung in Farbklassen eingeteilt. Diese Klassifikation wird über die Definition von Schwellwertbereichen der Parameter im Farbraum realisiert.

¹⁹ Quelle: Vladimir Chernov, Jarmo Alander, Vladimir Bochko, Integer-based accurate conversion between RGB and HSV color spaces (2015) [32]

Der RGB-Farbraum ist aus programmiertechnischer Sicht eine besonders unkomplizierte Darstellungsform, die sich an RGB-Anzeigegeräten orientiert, jedoch ist die Metrik des Farbraums für eine Farbklassifikationsaufgabe nicht geeignet. Diese Darstellung von Farben entspricht nicht der menschlichen Wahrnehmung, wodurch die Interpretation von Schwellwertbereichen erschwert wird. Beim Verschieben einzelner Farbpunkte im RGB Farbraum um eine bestimmte Distanz, abhängig vom Farbbereich, ändert sich die subjektiv wahrgenommene Farbe stark [30]. Ein weiteres Problem besteht darin, dass sich Farbton, Farbsättigung und Helligkeit bei jeder Bewegung im Koordinatensystem gleichzeitig ändern, wodurch bei unterschiedlicher Beleuchtung und sich änderndem Schatten eine hohe Varianz der einzelnen Parameter im RGB Farbraum entsteht. So ist die manuelle Auswahl eines subjektiv wahrgenommenen Farbspektrums sehr schwer und wenig intuitiv [30]. Zudem fallen die Schwellwertbereiche groß aus, wodurch die Robustheit der Klassifikation sinkt.

Der von Alvy Ray Smith entwickelte HSV-Farbraum erleichtert die Aufgabe der Farbsegmentierung. Hierbei handelt es sich um einen Farbraum, welcher dem menschlichen Sehen sehr ähnelt. Der HSV-Raum wird durch die Parameter *Hue*, *Saturation* und *Value* (Farbton, Sättigung, Helligkeit) definiert [30].

Der Farbton H repräsentiert die wahren Farben, z.B. Blau, Rot, Gelb, Magenta, und spezifiziert aus physikalischer Sicht die dominierende Wellenlänge der Farbe [32].

Die Sättigungskomponente (S) steht für die Farbreinheit. Sie ist ein Maß dafür, wie stark die wahre Farbe durch Zumischung von purem Weiß verdünnt wird. Ein geringer Prozentsatz des S-Kanals erzeugt eine gräuliche Farbe, während ein hoher Prozentsatz eine tiefe Farbe erzeugt. Bei abnehmenden S-Anteil wird die Farbe zunehmend Weiß [32].

Der Helligkeitswert V misst die Abweichung eines Farbtöns von Schwarz [33]. Der Helligkeitswert stellt damit den Gesamtenergieinhalt dar, beziehungsweise die maximale Amplitude des Lichts.

In der Ursprungsaarbeit von Smith wird die Visualisierung des HSV-Raums als hexagonale Pyramide beschrieben [33].

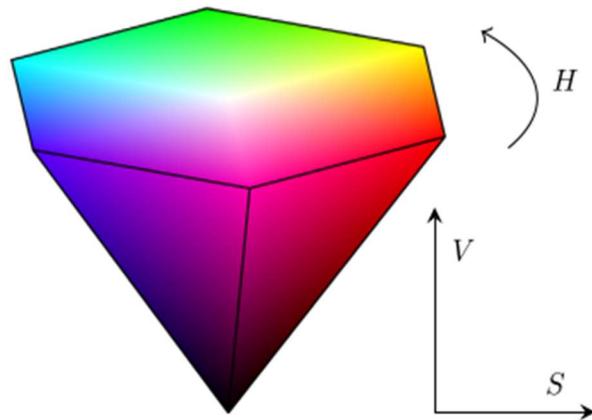


Abbildung 18: HSV hexagonale Pyramide²⁰

Aus mathematischer Sicht lässt sich die Ähnlichkeit zum menschlichen Sehen und die damit verbundene intuitive Farbsegmentierung des HSV-Raums damit erklären, dass die sogenannte „Farbdifferenz“ proportional zur menschlich subjektiv wahrgenommenen Differenz ist [32].

Die Farbdifferenz zweier Farbpunkte $c_1(h_1, s_1, v_1)$ und $c_2(h_2, s_2, v_2)$ ist definiert als der euklidische Abstand [30]:

$$\begin{aligned} \text{ColorDist}(c_1, c_2) = \\ \|c_1 - c_2\| = \sqrt{(h_1 - h_2)^2 + (s_1 - s_2)^2 + (v_1 - v_2)^2} \end{aligned} \quad (4.4)$$

Diese substanziale Eigenschaft besitzt der RGB-Raum nicht, weshalb eine Transformation vom RGB in den HSV-Raum, anhand des im Folgenden erklärten Algorithmus, stattfindet.

²⁰ Quelle: Vladimir Chernov, Jarmo Alander, Vladimir Bochko, Integer-based accurate conversion between RGB and HSV color spaces (2015) [32]

Smith beschreibt in seiner Arbeit den Transformationsalgorithmus vom RGB in HSV in den folgenden Schritten [33]:

Vorbedingung $R, G, B \in [0, 1]$

1. Finden des Maximums (M) und Minimums (m) von R, G , und B .

$$M = \max(R, G, B)$$

$$m = \min(R, G, B)$$

2. Setzen der Helligkeit $V = \text{Maximum } M$

3. Berechnen der Differenz (d) zwischen M und m

$$d = M - m$$

4. Wenn $d = 0$ dann wird $S = 0$ gesetzt und die Transformation beendet. H ist in diesem Fall undefiniert.

5. Berechnen von S als Quotient von d und M :

$$S = \frac{d}{M}$$

6. Berechnen von H in Abhängigkeit von M und m :

$$H = \frac{1}{6} \begin{cases} \frac{G - B}{d} & \text{wenn } M = R \text{ und } m = B \\ 1 + \frac{R - B}{d} & \text{wenn } M = G \text{ und } m = B \\ 2 + \frac{B - R}{d} & \text{wenn } M = G \text{ und } m = R \\ 3 + \frac{G - R}{d} & \text{wenn } M = B \text{ und } m = R \\ 4 + \frac{R - G}{d} & \text{wenn } M = B \text{ und } m = G \\ 5 + \frac{B - G}{d} & \text{wenn } M = R \text{ und } m = G \end{cases}$$

Nachbedingung: $H, S, V \in [0, 1]$

4.5.2.4 **Implementierung im Programmcode**

Die Transformation des RGB-Raums in den HSV-Raum wird im Programmcode durch die openCV Funktion

```
# Transformation in den HSV-Farbraum  
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

implementiert. Durch die Transformation des RGB-Raums in den HSV-Raum entsteht ein neuer Frame, auf dessen Basis zwei Binärbilder entstehen, welche als Grundlage der Detektion von Ball und Platte dienen.

4.5.3. **Plattendetektion**

4.5.3.1 **Farbfilter**

Für die Detektion von Platte und Ball müssen im ersten Schritt, auf Basis des Frames im HSV-Farbraum, mittels Farbsegmentierung zwei Binärbilder erstellt werden. In diesen Binärbildern werden alle Pixel innerhalb des Schwellwertbereiches durch eine 1 (True) und außerhalb durch eine 0 (False) dargestellt. Zur Ermittlung der Schwellwertbereiche werden die Parameter über Schieberegler an die gegebenen Lichtverhältnisse angepasst. Durch die Wahl von Schieberegelern können die Parameter schnell auf schwankende Lichtverhältnisse angepasst werden, wodurch gute Ergebnisse in der Detektion erzielt werden. Um die Schwellwerte nicht bei jedem Programmstart neu zu ermitteln und ein schnelles Arbeiten zu ermöglichen, werden die Farbparameter in einem Tripel (dreifach Tupel) gespeichert und fest im Programmcode implementiert (Siehe Kapitel 4.4).

4.5.3.2 **Morphologische Operationen**

Im vorherigen Schritt wird durch den Farbfilter aus Bild mit drei Farbkanälen ein Binärbild erzeugt. Bei dieser Schwellwertbildung entstehen durch übriggebliebenes Rauschen und die Textur der Objekte Fehler. In diesem Fall bedeutet das, dass neben den Pixeln der Platte noch weitere Pixel den Wert 1 erhalten und andersrum einige Pixel der Platte fälschlicherweise mit 0 markiert werden.

Um solche Fehler zu beseitigen, kommen in der Technik häufig morphologische Operationen zum Einsatz. Diese morphologischen Operationen – auch Nachbarschaftsoperationen genannt – bearbeiten Objekte in Binärbildern, indem sie Pixel zu einem Objekt hinzufügen oder von einem Objekt entfernen [26]. Dadurch wird die Form der Objekte verändert, sodass die Fehler aus der Schwellwertbildung beseitigt werden sollen [34]. Zwei grundlegende morphologische Operationen sind die Erosion und die Dilatation. Beide Operationen werden in diesem Projekt angewendet.

Erosion

Bei der Erosion werden - wie bei der natürlichen Erosion auch - die Ränder der Objekte im Bild abgetragen. Dies geschieht, indem eine Maske in Form einer Matrix mit Hilfe einer 2-dimensionalen Faltung über das Bild geschoben wird (vgl. Kapitel 4.5.2). Ein Pixel ist im erodierten Bild nur dann 1, wenn alle Pixel unterhalb der Maske im Eingangsbild 1 sind, ansonsten ist der Pixel 0. Die Größe der Maske bestimmt somit, wie stark die Ränder der Objekte erodiert werden. In folgender Abbildung ist die Erosion mit einer 3x3 Maske an einem Beispiel dargestellt.

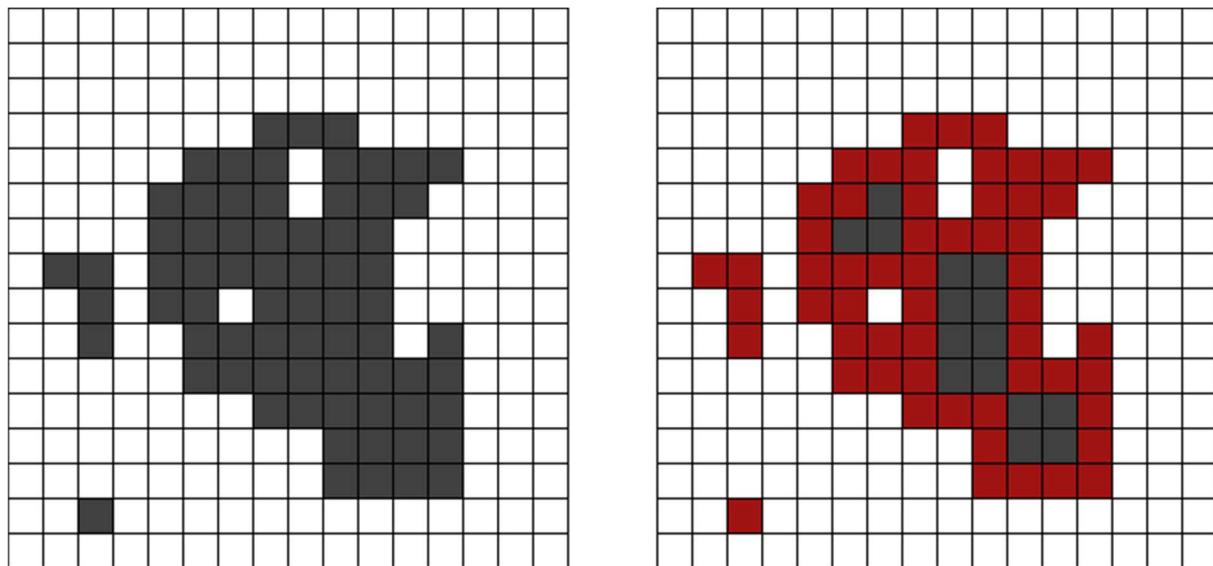


Abbildung 19: Erosion mit einer 3x3 Maske; Links: Ausgangszustand; Rechts: Zustand nach Erosion, die entfernten Pixel sind rot markiert

Dabei fällt auf, dass bei kleineren Objekten nicht nur die Ränder abgetragen werden, sondern aufgrund der geringen Größe diese Objekte komplett entfernt werden. Somit

eignet sich die Erosion gut, um kleine Störungen wie zum Beispiel Salz- und Pfeffer-Rauschen zu eliminieren. Zudem eliminiert die Erosion auch an größeren Objekten charakteristische Bereiche, wie zum Beispiel spitz zulaufende Randbereiche, dünne Stege oder Bereiche um Löcher innerhalb von Objekten. Dies ist in der Abbildung 19 an mehreren Stellen des großen Objekts sichtbar.

Dilatation

Die Dilatation beschreibt das Gegenteil der Erosion und stellt eine Art „Erweitern“ dar. Wie bei der Erosion auch wird eine Maske 2-dimensional über das Bild gefaltet. Der Unterschied ist, dass im dilatierten Bild der Pixel bereits 1 ist, sobald mindestens 1 Pixel im Eingangsbild unterhalb der Maske 1 ist. Dadurch werden Objekte im Bild vergrößert, was in folgender Abbildung 20 zu sehen ist. Auch in diesem Beispiel wird ein 3x3 Maske verwendet.

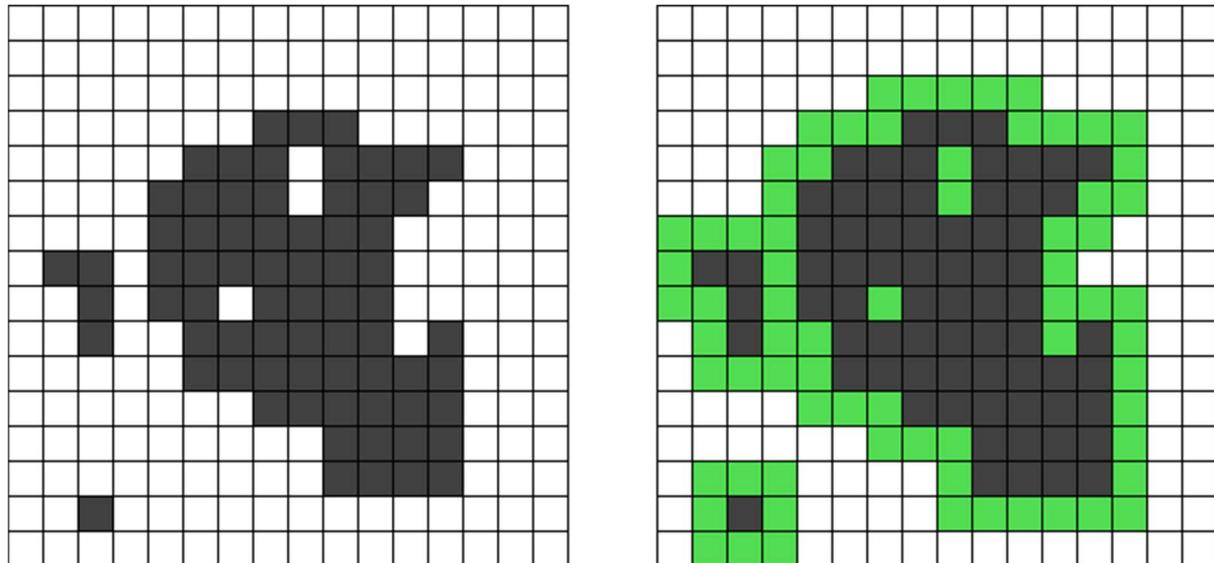


Abbildung 20: Dilatation mit einer 3x3 Maske; Links: Ausgangszustand; Rechts: Zustand nach Erosion, die hinzugefügten Pixel sind grün markiert

Erkenntlich ist, dass alle Objekte vergrößert werden. Zusätzlich werden kleine Löcher oder Risse innerhalb von Objekten gefüllt und nahe gelegene Objekte werden miteinander verbunden.

Öffnen

Die Erosion ist gut zur Entfernung von kleinen Objekten geeignet, hat allerdings den Nachteil, dass alle anderen Objekte verkleinert werden. Um dies zu vermeiden, kann anschließend eine Dilatation mit einer gleichgroßen Maske benutzt werden. Die Kombination aus Erosion und Dilatation wird Öffnen genannt [26].

Das Öffnen entfernt alle Objekte, die kleiner als die Maske sind und lässt dabei größere Objekte unverändert. In folgender Abbildung 21 ist das Öffnen mit einer 5x5-Maske beispielhaft dargestellt.



Abbildung 21 Beispiel Öffnen: Links: Ausgangszustand, Mitte: Erosion mit einer 5x5 Maske; Rechts: Öffnen mit einer 5x5 Maske²¹

Im ersten Schritt wird das Bild erodiert, im zweiten Schritt wird das erodierte Bild dilatiert. Das „J“-förmige Hauptobjekt bleibt beim Öffnen unverändert, wohingegen die kleinen Objekte entfernt werden.

Das Öffnen wird in diesem Projekt folgendermaßen implementiert:

```
kernel = np.ones((5, 5))

eroded = cv2.erode(mask, kernel)
opened = cv2.dilate(eroded, kernel)
```

In der ersten Zeile wird die Maske beziehungsweise der Faltungskern in Form einer 5x5-Matrix, gefüllt mit Einsen, definiert. Im nächsten Schritt wird mit der Erosions-Funktion

²¹ Quelle: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html (27.03.2020).

aus der openCV-Bibliothek das Binärbild (`mask`) durch den Faltungskern erodiert. Anschließend wird das erodierte Bild mit dem gleichen Faltungskern dilatiert.

4.5.3.3. Konturerkennung

In diesem Abschnitt wird die Erzeugung von Objektrepräsentation kurz erläutert. Eine weit verbreitete Methode ist die Konturverfolgung, welche meistens auf Basis von Binärbildern entsteht [32]. Bei einer Kontur handelt es sich um eine geschlossene Linie, welche vollständig am Rand eines Objektes verläuft.

Zur Detektion der Kontur wird die Funktion `cv2.findContours()` aus der openCV-Bibliothek angewendet. Die Funktion basiert auf dem von Suzuki und Ade entwickelten Boundary-Follower-Algorithmus [33], welcher in dieser Arbeit nicht näher erläutert wird, da er sehr komplex ist und nur wenige der dort behandelten Spezialfälle in dieser Arbeit Anwendung finden. Sollte mittels des Algorithmus²² keine Konturen gefunden werden, erfolgt keine weitere Bildverarbeitung und die Platte wird in ihre Ausgangsposition verfahren²².

4.5.3.4. Auswahl der größten Kontur

Da bei der Konturerkennung des geöffneten Schwellwertbildes mehrere Konturen gefunden werden, muss die richtige Kontur der Plattenränder ausgewählt werden. Im Regelfall gibt es neben der Kontur der Platte mindestens noch die Kontur um den Ball auf der Platte, die im Binärbild ein Loch in der Platte darstellt. Weitere Konturen können durch fälschlich erkannte orangene Gegenstände entstehen.

Da die Platte den Großteil des Bildbereichs der Kamera ausmacht, kann gesagt werden, dass die Kontur der Plattenränder flächenmäßig immer die größte Kontur darstellt. Das heißt, im Programm wird der Flächeninhalt von jeder Kontur miteinander verglichen und die Kontur mit dem größten Flächeninhalt wird ausgewählt.

²² In der Ausgangsposition sind die Plattenwinkel in X- und Y-Achse 0°

Die Konturen in openCV sind in Form einer Liste von miteinander verbundenen Punkten gespeichert und nehmen in der Regel komplexe Formen an. Deshalb ist eine herkömmliche Berechnung des Flächeninhalts (z.B. die Aufteilung in einfache geometrische Formen) nur mit großem Aufwand möglich. Die Berechnung des Flächeninhalts erfolgt stattdessen mit Hilfe des Satzes von Green [35].

Satz von Green

Der Satz von Green (1.1) ist ein Sonderfall des Integralsatzes von Stokes. Er ermöglicht es, das Integral über eine ebene Fläche D durch das Kurvenintegral ihres einschließenden Randes C auszudrücken [36].

$$\oint_C (f(x, y)dx + g(x, y)dy) = \iint_D (g_x - f_y) dxdy \quad (4.5)$$

Satz von Green

(wobei g_x die partielle Ableitung von g nach x und f_y die partielle Ableitung von f nach y ist)

Da der Flächeninhalt A der Fläche D durch

$$A = \iint_D 1 dxdy \quad (4.6)$$

gegeben ist, lässt sich dieser mit dem Satz von Green bestimmen, wenn f und g so gewählt werden, dass

$$g_x - f_y = 1$$

ist. Diese Bedingung wird unter anderem erfüllt bei:

$$f(x, y) = 0 \text{ und } g(x, y) = x \quad (4.7)$$

Durch Einsetzen von (4.7) in (4.5) folgt, dass der Flächeninhalt der Fläche D über folgendes Kurvenintegral berechnet werden kann: [37]

$$A = \oint_C x \, dy \quad (4.8)$$

Die einschließende Kurve ist in diesem Fall die Kontur, welche aus n miteinander verbundenen Punkten besteht und als Polygon charakterisiert werden kann. Da ein Polygon nur abschnittsweise glatte Ränder hat, muss die Kurve in n einzelne Verbindungsstücke C_k aufgeteilt werden. Die entsprechenden Kurvenintegrale können aufsummiert werden:

$$A = \int_{C_1} x \, dy + \int_{C_2} x \, dy + \cdots + \int_{C_n} x \, dy \quad (4.9)$$

Die einzelnen Verbindungsstücke können mit einer Geradengleichung beschrieben werden. Für das k -te Verbindungsstück, welches die Punkte (x_k, y_k) und (x_{k+1}, y_{k+1}) verbindet, ergibt sich folgende Parametrisierung: [37]

$$C_k: \vec{r} = ((x_{k+1} - x_k)t + x_k, (y_{k+1} - y_k)t + y_k), \quad 0 \leq t \leq 1 \quad (4.10)$$

Substitution von x und dy mit (4.10) im Kurvenintegral für C_k ergibt:

$$\begin{aligned} \int_{C_k} x \, dy &= \int_0^1 ((x_{k+1} - x_k)t + x_k) (y_{k+1} - y_k) \, dt \\ &= \frac{(x_{k+1} + x_k)(y_{k+1} - y_k)}{2} \end{aligned} \quad (4.11)$$

Die Summe aller Kurvenintegrale entspricht dem Flächeninhalt der Fläche, welche von der Kontur eingeschlossen wird [37]:

$$A = \frac{1}{2} \sum_{k=1}^n (x_{k+1} + x_k)(y_{k+1} - y_k) \quad (4.12)$$

Der Vorteil an dieser Formel ist, dass nur die Eingabe der bereits vorhandenen Koordinaten der einzelnen Punkte erforderlich ist [37] und somit auf komplexe Konturformen anwendbar ist.

Implementierung im Programmcode

Die oben hergeleitete Berechnung des Flächeninhalts über den Satz von Green wird von openCV verwendet. Der Flächeninhalt der Fläche, die von einer Kontur eingeschlossen wird, kann über die Funktion

```
cv2.contourArea(contour)
```

berechnet werden.

Um aus einer Liste von mehreren Konturen die flächenmäßig größte Kontur zu finden, wird in diesem Programm der eingebaute `max`-Operator von Python benutzt. Diese ermöglicht es, neben der Liste, aus der das größte Element zurückgegeben werden soll, einen „Key“ zu übergeben, nachdem sortiert werden soll [38]. Als key wird oben genannte Funktion übergeben:

```
max_contour = max(contours, key=cv2.contourArea)
```

4.5.3.5. Konturapproximation zu einem Vieleck

Im vorherigen Schritt wird bereits die richtige Kontur der Platte identifiziert und ausgewählt. Da die Platte quadratisch und eben ist, stellt sie nach dem Modell der Lochkamera im Kamerabild ein allgemeines Viereck dar [26]. Die zugehörige Kontur sollte im Idealfall ebenfalls ein Viereck darstellen, gebildet aus den vier Eckpunkten der Platte.

Da die Kamera optisch nicht perfekt ist und immer minimale Verzerrungen auftreten, ist die Kontur der Platte kein ideales Viereck und besteht aus weitaus mehr Eckpunkten. Für die weiteren Verarbeitungsschritte ist es allerdings vorteilhaft, die Platte nur mit ihren vier Eckpunkten zu beschreiben, weshalb in diesem Schritt die erkannte Kontur an ein Viereck angenähert wird. Dies geschieht mit Hilfe des Douglas-Peucker-Algorithmus.

Die Konturapproximation mit dem Douglas-Peucker-Algorithmus kann in diesem Fall mit der herkömmlichen Eckendetektion mit Hilfe des Haris-Operators verglichen werden und stellt einen alternativen Lösungsweg dar. Beide Verfahren liefern ähnliche Ergebnisse, wobei bei der Konfiguration in diesem Projekt die Konturapproximation die

Ergebnisse minimal besser waren. Eine spätere Implementierung des Haris-Operators mit optimierten Parametern ist vorstellbar.

Douglas-Peucker Algorithmus

Der Douglas-Peucker-Algorithmus ist eine weit verbreitete Methode zur Kurvenglättung beziehungsweise zur Vereinfachung von Linienzügen. Ein Linienzug – auch Polygon- oder Streckenzug genannt – ist der Zusammenschluss aus Verbindungslienien die mehrere Punkte miteinander verbinden. Konturen in openCV sind ebenfalls Linienzüge, weshalb sich dieser Algorithmus auf Konturen anwenden lässt. Der Douglas-Peucker-Algorithmus vereinfacht Linienzüge, indem er die Anzahl an Stützpunkten innerhalb des Linienzuges reduziert, ohne die grobe Gestalt des Linienzuges zu verändern. Unerwünschte Details sollen eliminiert werden [39]. Ein Beispiel des Douglas-Peucker-Algorithmus ist im Folgekapitel dargestellt.

Das Grundprinzip des Douglas-Peucker-Algorithmus wird im Folgenden beschrieben. Gegeben ist ein Linienzug bestehend aus mehreren Punkten und eine Toleranz ε , die bestimmt wie stark der Linienzug vereinfacht werden soll.

Der erste Punkt des Linienzuges wird als „Anker“ und der letzte Punkt als „Springer“ bezeichnet. Beide Punkte werden durch eine gerade Linie verbunden und der senkrechte Abstand zu allen dazwischenliegenden Punkten wird berechnet. Wenn der Abstand aller dieser Punkte kleiner als die Toleranz ε ist, wird die Verbindungslienie als eine geeignete Näherung betrachtet und der Algorithmus ist abgeschlossen.

Falls das Toleranzkriterium nicht erfüllt ist, wird der Punkt mit dem größten senkrechten Abstand als neuer Springer ausgewählt und der Vorgang wird zwischen dem Anker und dem neuen Springer wiederholt. Es wird eine neue Verbindungslienie gezogen und es werden erneut die Abstände von allen dazwischenliegenden Punkten berechnet. Wenn mindestens einer dieser Abstände größer als die vorgegebene Toleranz ist, wird der Punkt mit dem größten Abstand der Springer im nächsten Schritt.

Die Aufteilung in immer kleinere Verbindungsstrecken wird solange wiederholt, bis das Toleranzkriterium erfüllt ist. Wenn es erfüllt ist, wird der aktuelle Springer zum nächsten Anker und der vorherige Springer wird zum nächsten Springer.

Der gesamte Prozess wird fortgeführt, bis der letzte Punkt des Linienzuges Ankerpunkt ist. Im letzten Schritt werden alle jemals ausgewählten Ankerpunkte durch gerade Linien miteinander verbunden. Dies entspricht dem resultierenden vereinfachten Linienzug.

Wird eine große Toleranz ε gewählt, werden viele Punkte entfernt und nur wenige Details bleiben erhalten. Andersrum werden bei kleinem ε weniger Punkte entfernt und somit bleiben mehr Details erhalten [40] [41].

Beispiel Douglas-Peucker Algorithmus

Mit diesem Beispiel soll der Douglas-Peucker-Algorithmus verdeutlicht werden. Jeder Vorgehensschritt des Algorithmus wird knapp beschrieben und durch eine Abbildung visualisiert. Anker und Springer sind jeweils durch a_k und f_k gekennzeichnet, wobei k eine fortlaufende Nummer ist. Gegeben ist ein Linienzug bestehend aus acht Punkten und eine feste Toleranz ε . Das Beispiel ist an verschiedenen Illustrationen des Douglas-Peucker-Algorithmus angelehnt [40] [42].

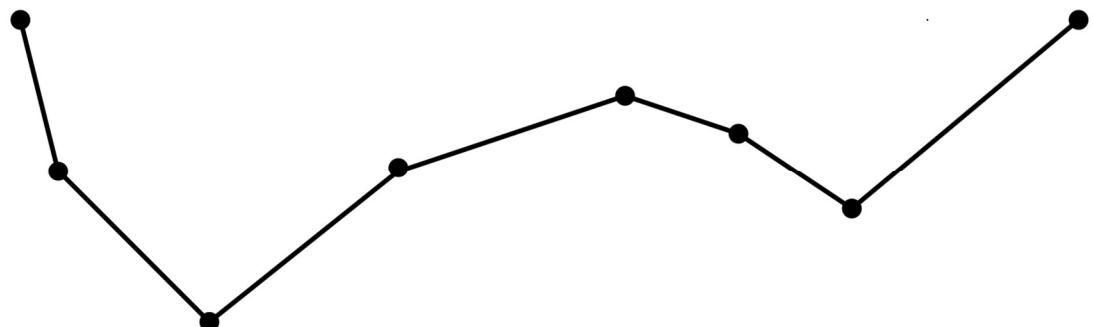


Abbildung 22: Linienzug bestehend aus acht Punkten

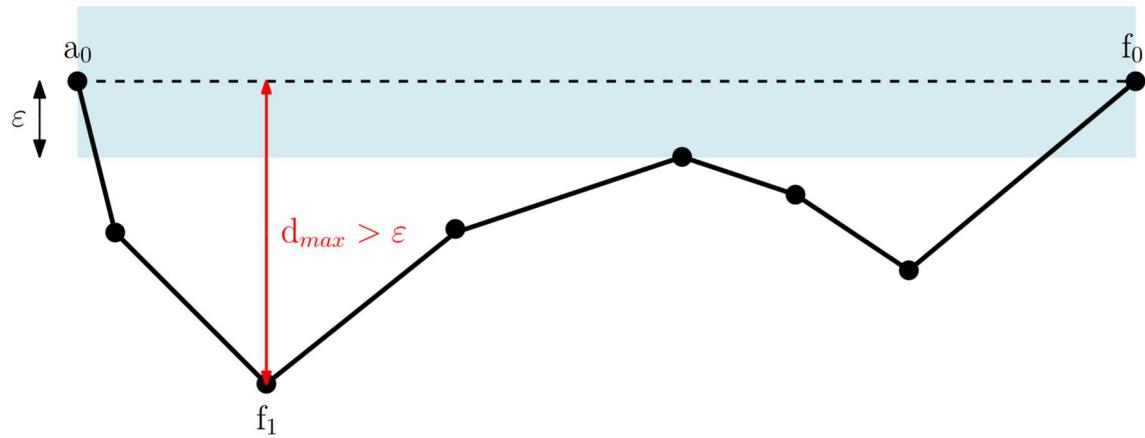


Abbildung 23: Verbindungsleitung a_0-f_0 (Toleranzbereich ist blau hinterlegt): Größter Abstand zu f_1 ist außerhalb der Toleranz; f_1 wird der neue Springer

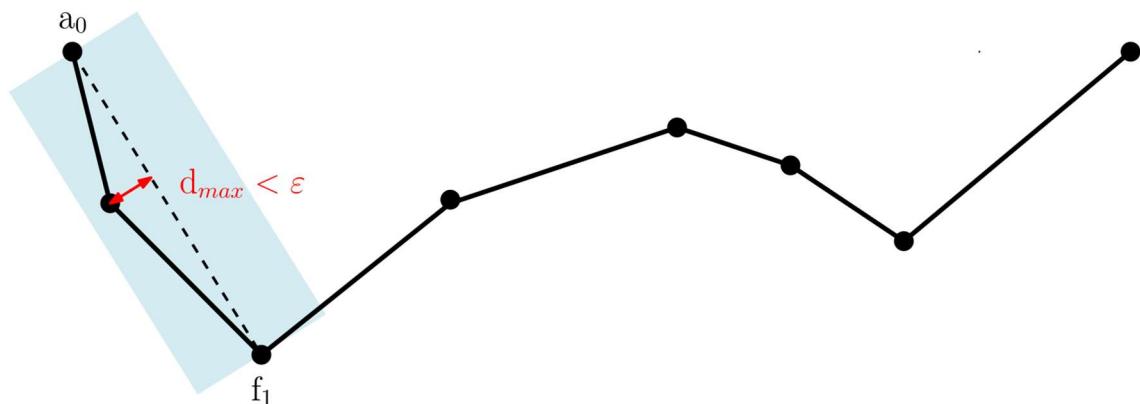


Abbildung 24: Verbindungsleitung a_0-f_1 : Alle Abstände innerhalb Toleranz; Springer f_1 wird neuer Anker a_1 und vorheriger Springer f_0 wird nächster Springer

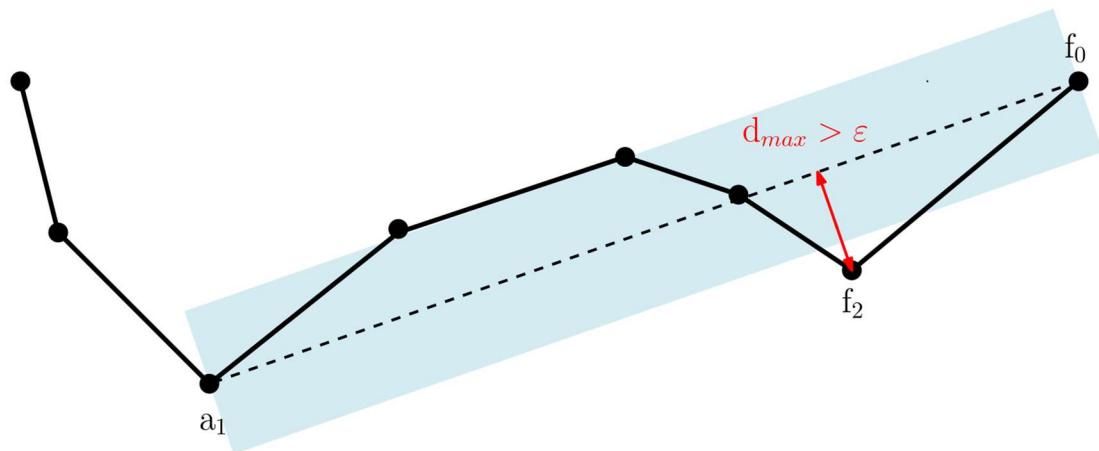


Abbildung 25: Verbindungsleitung a_1-f_0 : Größter Abstand zu f_0 ist außerhalb der Toleranz

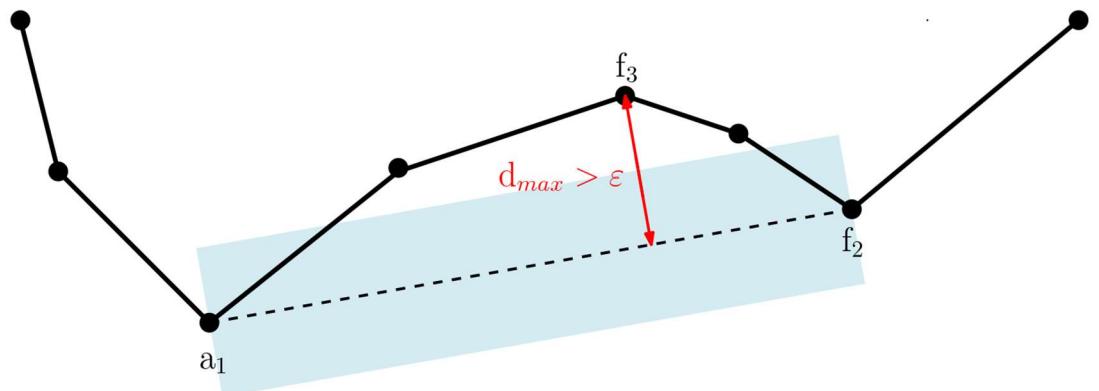


Abbildung 26: Verbindungsleitung a_1-f_2 : Größter Abstand zu f_2 ist außerhalb der Toleranz

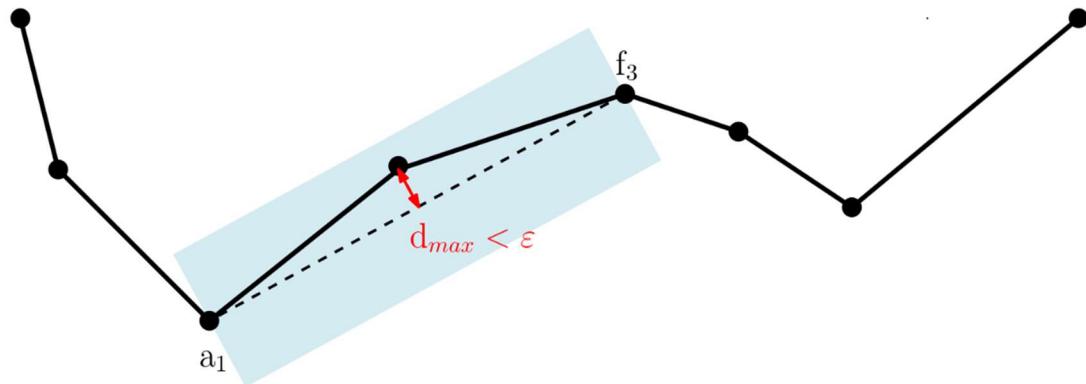


Abbildung 27: Verbindungsleitung a₁-f₃: Alle Abstände innerhalb Toleranz; f₃ wird zu a₂

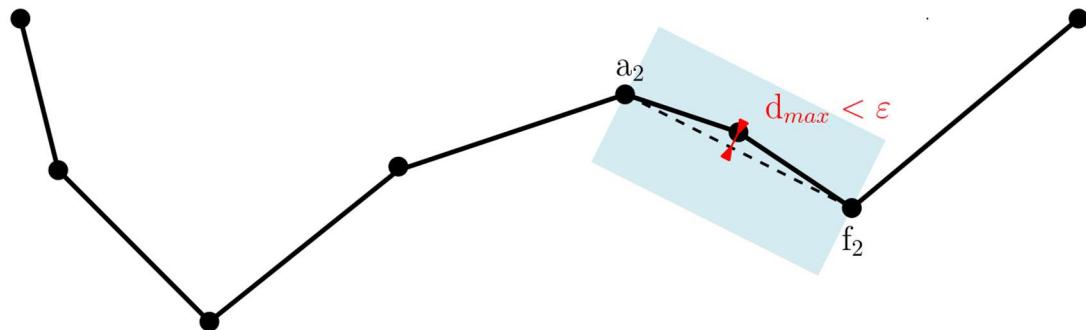


Abbildung 28: Verbindungsleitung a₂-f₂: Alle Abstände innerhalb Toleranz; f₂ wird zu a₃

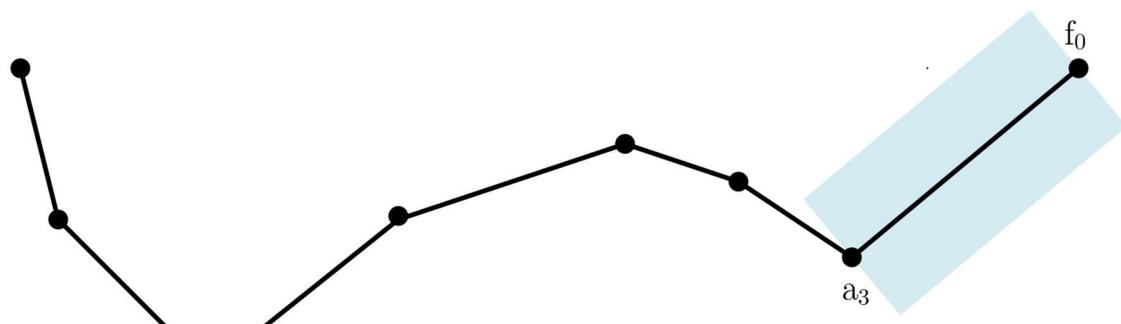


Abbildung 29: Verbindungslinie a_3-f_0 : Keine Punkte zwischen Anker und Springer; f_0 wird zu a_4

Da in diesem Schritt der letzte Punkt des Linienzuges zum Anker a_4 wird, ist die Vereinfachung des Linienzuges abgeschlossen. Der vereinfachte Linienzug wird durch Verbinden der Anker mit geraden Linien aufgebaut, siehe folgende Abbildung 30.

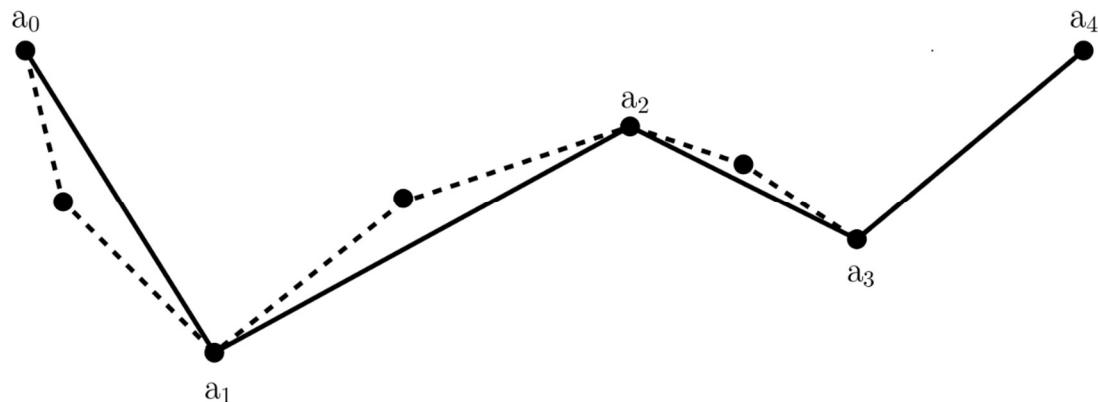


Abbildung 30: Vergleich des vereinfachten Linienzuges (durchgezogen) zum originalen Linienzug (gestrichelt)

Wie zu sehen ist, wird der Linienzug von ursprünglich acht Punkten auf fünf Punkte reduziert. Dabei bleibt die grobe Gestalt des Linienzuges erhalten.

Anwendung des Douglas-Peucker-Algorithmus im Programmcode

Im Programmcode wird der Douglas-Peucker-Algorithmus angewendet, um die Kontur der Platte auf ein viereckiges Polygon, bestehend aus den vier Eckpunkten der Platte, zu reduzieren.

Die Funktion `cv2.approxPolyDP` aus der openCV-Bibliothek implementiert den Douglas-Peucker-Algorithmus und wird folgendermaßen benutzt:

```
polygon = cv2.approxPolyDP(curve=max_contour, epsilon=20, closed=True)
```

Als Linienzug wird die größte Kontur aus dem vorherigen Schritt übergeben, die Toleranz ε wird auf 20 Pixel gesetzt und mit dem Parameter `closed=True` wird festgelegt, dass die Kontur geschlossen sein soll (erster und letzter Punkt werden miteinander verbunden) [35].

Beachtet werden muss, dass der Douglas-Peucker-Algorithmus die vorhandene Kontur nicht zwangsweise auf ein Viereck vereinfacht. Dennoch lässt sich durch geeignete Wahl der Toleranz ε die vorhandene Kontur der Platte zuverlässig auf ein Viereck reduzieren. Bei einer Bildauflösung von 640x480 Pixel hat sich ein ε von 20 Pixel bewährt.

Trotzdem wird zur Sicherheit nachfolgend das Polygon auf ein Viereck überprüft:

```
if polygon.shape[0] == 4:
```

Da `polygon` in Form eines Numpy-Arrays vorliegt, kann mit Hilfe des Shape-Operators in erster Dimension die Anzahl der Stützpunkte ermittelt werden.

Fehlervermeidung beim Legen der Hand auf die Platte

Beim Positionieren des Balles kann es vorkommen, dass die Hand des Benutzers die Sicht der Kamera auf die Platte teilweise verdeckt. Die erkannte Form der Platte ändert sich signifikant, sodass trotz gut gewählter Toleranz ε bei der Konturapproximation kein Viereck entsteht. Stattdessen entsteht ein viereck-ähnliches Polygon, welches an der Stelle der Hand in Kamerabild eingeschnitten ist.

Für einen bestimmten Zeitraum kann die Position des Balls auf der Platte nicht berechnet werden und es besteht die Gefahr, dass der Ball von der Platte fällt. Besonders am Anfang und beim dynamischen Platzieren des Balls (zum Beispiel schnelles Draufrollen) ist diese Gefahr groß.

Das Problem kann gelöst werden, indem die konvexe Hülle der von der approximierten Kontur eingeschlossenen Fläche berechnet wird.

Die konvexe Hülle ist die kleinste konvexe Menge, die die Ausgangsmenge (in diesem Fall die eingeschlossene Fläche) enthält [43]. Eine konvexe Menge ist dadurch definiert, dass die Verbindungsstrecke von zwei beliebigen Punkten der Menge ebenfalls komplett in der Menge liegt [44]. Anschaulich im 2-dimensionalen Fall beschrieben heißt das, dass die konvexe Menge weder Einbuchtungen noch Löcher besitzt. Beim Bilden der konvexen Hülle aus einer Menge werden Einbuchtungen und Löcher entfernt.

Beispiele für eine konvexe und für eine nichtkonvexe Menge sind in folgender Abbildung zu sehen. Illustriert ist zusätzlich das Kriterium für eine konvexe Menge - die Verbindungsstrecke zwischen zwei beliebigen Punkten a und b .

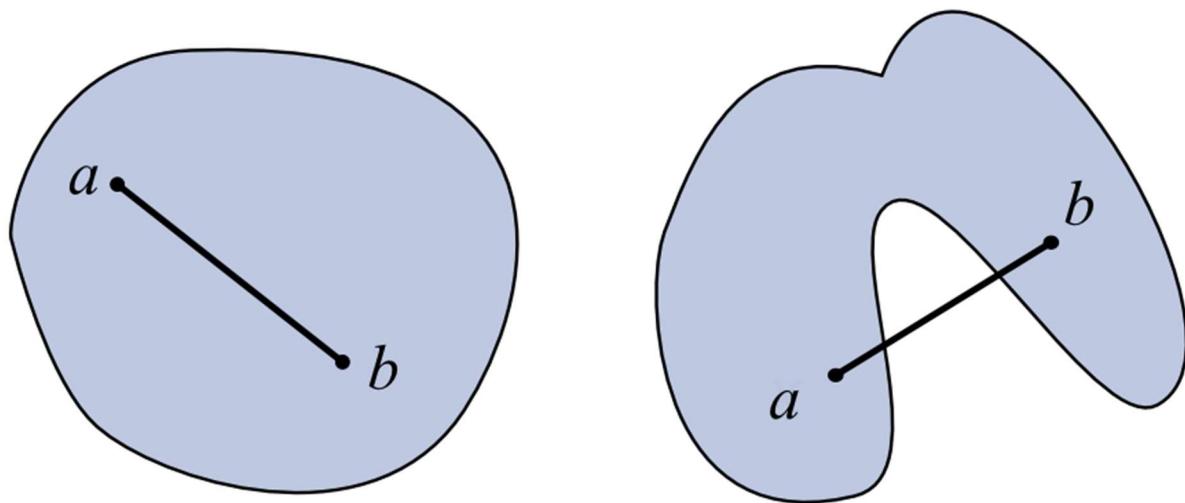


Abbildung 31: Konvexe Menge (links) und nichtkonvexe Menge (rechts)²³

Durch Bilden der konvexen Hülle kann der Einschnitt im Vieleck „aufgefüllt“ werden, sodass die Form des zugrundeliegenden Vierecks entsteht.

²³ Quelle: https://de.wikipedia.org/wiki/Konvexe_Menge (13.04.2020)

Zu beachten ist, dass die hier vorgestellte Methode nur funktioniert, wenn keine Ecke der Platte verdeckt ist. Wenn eine Ecke verdeckt ist, kann die Form der Platte nicht mehr über die konvexe Hülle rekonstruiert werden.

Zur Berechnung der konvexen Hülle eines einfachen Polygons wird in dieser Arbeit der Algorithmus von J. Sklansky [45] verwendet. Die konkrete Beschreibung des Algorithmus würde den Rahmen dieser Arbeit überschreiten. Die genaue Funktionsweise kann im Originaldokument von 1982 nachgelesen werden.

Implementierung der konvexen Hülle im Programmcode

Im Programm ist der Algorithmus über die openCV-Funktion

```
convex_polygon = cv2.convexHull(points=polygon)
```

implementiert. Als Übergabeparameter ist lediglich eine Punktemenge notwendig – in diesem Fall das approximierte Vieleck. Zurückgegeben wird die konvexe Hülle des Polygons.

Da beim Bilden der konvexen Hülle meist an der Stelle des Einschnitts die zwei Eckpunkte erhalten bleiben, muss erneut mit Hilfe des Douglas-Peucker-Algorithmus‘ das Polygon auf ein Viereck angenähert werden:

```
convex_polygon = cv2.approxPolyDP(curve=convex_polygon, epsilon=20,  
closed=True)
```

Verknüpft mit dem ursprünglichen Douglas-Peucker-Algorithmus lautet der gesamte Programmcode dieses Kapitels wie folgt:

```
polygon = cv2.approxPolyDP(curve=max_contour, epsilon=20, closed=True)  
convex_polygon = cv2.convexHull(points=polygon)  
convex_polygon = cv2.approxPolyDP(curve=convex_polygon, epsilon=20,  
closed=True)  
  
if convex_polygon.shape[0] == 4:
```

4.5.3.6. Anordnung der Eckpunkte

Die vier Eckpunkte der approximierten Kontur sollen in diesem Schritt in eine definierte Reihenfolge gebracht werden. Die Anordnung der Punkte ist eine notwendige Grundlage für den nächsten Verarbeitungsschritt (die perspektivische Transformation) und ist allgemein für eine eindeutige Zuordnung von mehreren Punktemengen erforderlich.

Dazu wird ein Algorithmus verwendet, welcher im Online-Blog „pyimagesearch“ vor gestellt wird²⁴ [46].

Die Punkte des Vierecks sollen der Reihe nach im Uhrzeigersinn angeordnet werden, beginnend mit dem oberen linken Punkt. Somit ergibt sich folgende Reihenfolge:

- Punkt oben links,
- Punkt oben rechts,
- Punkt unten rechts,
- Punkt unten links.

Es werden von allen vier Punkten die Summen und die Differenzen aus den jeweiligen X- und Y-Koordinaten berechnet. Das heißt:

$$\text{sum}(x_i, y_i) = y_i + x_i \quad (4.13)$$

$$\text{diff}(x_i, y_i) = y_i - x_i$$

Dabei ist die Auslegung des zugrundeliegenden Koordinatensystems maßgeblich. Bei digitalen Bildern und in der Bildverarbeitung ist es üblich, den Koordinatenursprung in die obere linke Ecke zu legen. Die positive Y-Richtung verläuft vertikal nach unten und die positive X-Richtung verläuft horizontal nach rechts. Dies ist in den Abbildung 32 und Abbildung 33 zu sehen.

Mit dieser Auslegung des Koordinatensystems können die Summe und die Differenz aus den Koordinatenpaaren qualitativ beschrieben werden.

²⁴ Der Autor des Blogs „pyimagesearch“ ist Adrian Rosebrock, ein im Internet bekannter Informatiker, welcher bereits mehrere Bücher zum Thema „Computer Vision“ verfasst hat.

Die Summe aus X- und Y-Wert nimmt in Richtung der Winkelhalbierenden zwischen X- und Y-Achse zu und ist immer größer gleich Null. Die Differenz aus X- und Y-Wert ist auf der Winkelhalbierenden zwischen X- und Y-Achse gleich Null. Unterhalb der Winkelhalbierenden ist sie größer Null, oberhalb der Winkelhalbierenden kleiner Null. In den beiden folgenden Abbildungen ist dies visualisiert, indem die Niveaulinien, jeweils von Summe und Differenz der X- und Y-Koordinaten, dargestellt sind.

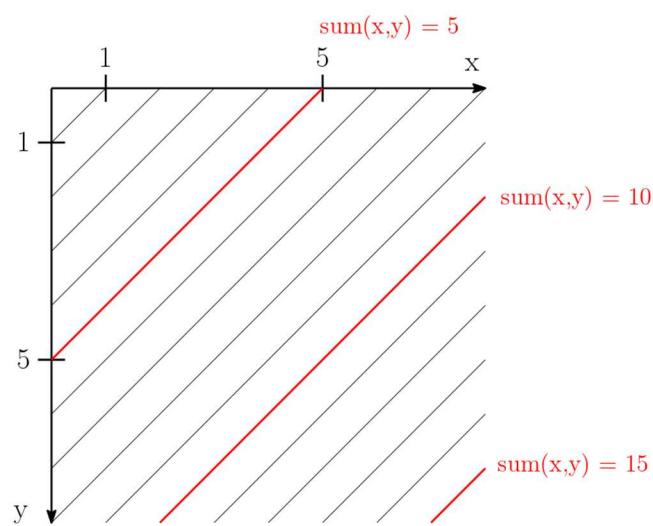


Abbildung 32: Niveaulinien der Summe aus X- und Y-Koordinate

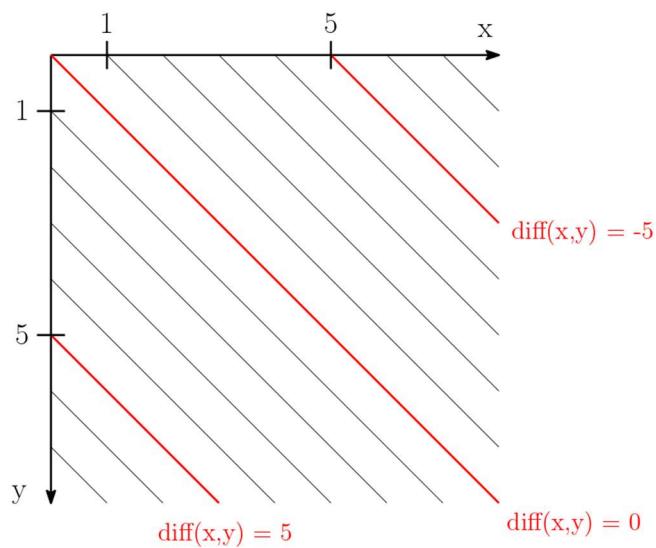


Abbildung 33: Niveaulinien der Differenz aus X- und Y-Koordinate

Bei geeigneter Form des Vierecks können die vier Punkte eindeutig im Uhrzeigersinn angeordnet werden:

- Der obere linke Punkt besitzt die kleinste Summe aus X- und Y-Wert
- Der obere rechte Punkt besitzt die kleinste Differenz aus X- und Y-Wert
- Der untere rechte Punkt besitzt die größte Summe aus X- und Y-Wert
- Der untere linke Punkt besitzt die größte Differenz aus X- und Y-Wert

Dazu muss gelten: Nur ein Punkt darf jeweils die größte/kleinste Summe/Differenz haben. Sobald zwei oder mehr Punkte zum Beispiel die gleiche größte Summe aus X- und Y-Koordinate haben, schlägt der Algorithmus fehl [46].

Bereits die Einteilung der Punkte in zum Beispiel „oben links“ verrät gewisse Anforderungen an das Viereck. Beispielsweise kann einem um 45 Grad gedrehtes Quadrat kein „oberer linker“ Punkt zugeordnet werden. Die Anordnung der Punkte dieses Quadrats würde mit dem Algorithmus fehlschlagen.

Die approximierte Kontur der Platte stellt zwar ein verzerrtes Viereck dar, allerdings ist sie durch die relative Position der Kamera zur Platte nie verdreht und nie so stark verzerrt, dass mehrere Punkte eines der Extrema erfüllen. Aus diesem Grund ist eine falsche Zuordnung ist bei der approximierten Plattenkontur praktisch unmöglich, was auch im Praxisbetrieb bestätigt wird. Damit ist der Algorithmus für diese Anwendung komplett ausreichend.

Implementierung im Programmcode

Zur Implementierung des Algorithmus‘ im Programmcode muss zunächst die Form des Numpy-Arrays `convex_polygon` aus dem vorherigen Schritt angepasst werden:

```
pts = convex_polygon.reshape(4, 2)
```

Zuvor kommen zusätzliche Dimensionen durch die Verarbeitung mit openCV hinzu. Da das Array vier Punkte, bestehend aus X- und Y-Koordinate enthält, wird eine Struktur, bestehend aus vier Zeilen und zwei Spalten, gewählt.

Im nächsten Schritt wird ein Array für die geordneten Punkte mit gleicher Form definiert:

```
pts_ordered = np.zeros(shape=(4, 2))
```

Danach werden mit den folgenden Funktionen die Summen und Differenzen für jeden Punkt gebildet:

```
sum = pts.sum(axis=1)
diff = np.diff(pts, axis=1)
```

`axis=1` bedeutet dabei, dass die Summe beziehungsweise die Differenz entlang einer Zeile, also für jeden Punkt einmal gebildet wird. Die zurückgegeben Arrays `sum` und `diff` bestehen somit aus jeweils vier Einträgen.

Durch Verwendung der Numpy-Funktionen `np.argmax` und `np.argmin` wird der Index der kleinsten beziehungsweise der größten Summe und Differenz zurückgegeben. Nach bereits erläutertem Prinzip werden dann die einzelnen Indizes des Arrays `pts_ordered` überschrieben:

```
pts_ordered[0] = pts[np.argmin(sum)]
pts_ordered[1] = pts[np.argmin(diff)]
pts_ordered[2] = pts[np.argmax(sum)]
pts_ordered[3] = pts[np.argmax(diff)]
```

Damit enthält das Array die Punkte in richtiger Reihenfolge und kann für die weiteren Verarbeitungsschritte genutzt werden.

4.5.4. **Balldetektion**

Die Balldetektion verläuft weitestgehend analog zur Plattendetektion. Im ersten Schritt wird eine Farbsegmentierung auf Basis der Farbparameter des Balles durchgeführt. In dem entstehenden Binärbild werden Bildfehler und Salz- und Pfeffer-Rauschen über Morphologisch Operationen entfernt. Durch den Boundary Flower Algorithmus von Suzuki und Ade werden alle Konturen erkannt. Anschließend wird, mittels des Satz von Green wird die Größte Kontur ermittelt²⁵. Auf Basis der Kontur vom Ball wird sein Schwerpunkt und somit seine genaue Position bestimmt.

²⁵ Diese Schritte wurden ausführlich in Kapitel 4.5.3 erläutert

Schwerpunktberechnung über Momente

Zur exakten Bestimmung der Ballposition wird die Kontur k als Verteilung von Koordinatenpunkten im zweidimensionalen Raum betrachtet. Die Kontur stellt eine sogenannte „binäre Region“ im Eingangsbild dar. Die Ballposition wird über den Schwerpunkt $\bar{x} = (\bar{x}, \bar{y})^T$ dieser binären Kontur bestimmt. Zur Schwerpunktberechnung einer binären Region wird das arithmetische Mittel der darin enthaltenen Punktkoordinaten $x_i(x, y)$ gebildet [30].

$$\bar{x} = \frac{1}{|k|} \sum_{x_i \in k} x_i \quad (4.14)$$

Die Formulierung des Schwerpunktes ist ein spezieller Fall eines allgemeinen Konzeptes aus der Statistik, der sogenannten „Momente“ [30].: Das Moment der Kontur in der Bildfunktion $I(x, y)$ wird durch den Ausdruck

$$M_{i,j}(k) = \sum_{x,y} x^i * y^j * I(x, y) \quad (4.15)$$

beschrieben. Zur Berechnung des Schwerpunktes wird das Gesamtmoment²⁶ $M_{0,0}$ sowie die Momente in X- $M_{1,0}$ und Y-Richtung $M_{0,1}$ bestimmt. Der Schwerpunkt bestehend aus

X- und Y-Koordinate wird anschließend durch folgende Formel bestimmt:

$$\bar{x} = \frac{M_{1,0}}{M_{0,0}} \quad (4.16)$$

$$\bar{y} = \frac{M_{0,1}}{M_{0,0}} \quad (4.17)$$

Implementierung im Programmcode

Die Momente der Ballkontur werden im Programmcode durch die Funktion

²⁶ Da es sich um ein Binärbild handelt, wird die Anzahl der Pixel gezählt

```
moments = cv2.moments(max_contour)
```

berechnet. Der Schwerpunkt wird über ein Tupel definiert. Die Berechnung basiert auf der oben beschriebenen Theorie.

```
center = (int(moments["m10"] / moments["m00"]),
           int(moments["m01"] / moments["m00"]))
return center
```

Der Schwerpunkt wird an die Hauptdatei als Rückgabewert übergeben, in welcher auf Basis von der Position und der Geschwindigkeit des Balls der Zustandsvektor definiert wird.

4.5.5. Perspektivische Transformation

Mit den bisherigen Bildverarbeitungsschritten wird die Platte bereits als Viereck beschrieben. Dieses Viereck ist jedoch durch die Perspektive der Kamera verzerrt. Zusätzlich ändert sie die Verzerrung der Platte regelmäßig, da die Platte um zwei Achsen gedreht beziehungsweise gekippt werden kann.

Das macht es schwer, eine quantitative Aussage über die Position des Balls relativ zur Platte zu treffen. Die genaue Bestimmung der Position ist jedoch eine grundlegende Voraussetzung für die Regelung. Zur exakten Positionsbestimmung des Balls im Plattenkoordinatensystem sind weitere Verarbeitungsschritte notwendig.

Eine Bestimmung der Ballkoordinaten ist theoretisch möglich, wenn die Platte als quadratisches Bild vorhanden ist. Der Fall, dass die Platte im Kamerabild quadratisch erscheint, kommt nur unter einem exakten Winkel vor und ist äußerst selten. Die Näherung, die Platte immer als quadratisch anzusehen, wird als unzureichend eingestuft, da die Platte während der Regelung Neigungswinkel von bis zu 10 Grad erreicht und dabei deutlich verzerrt auf dem Kamerabild erscheint.

Um die Ballkoordinaten im Plattenkoordinatensystem zuverlässig bei allen Neigungswinkeln bestimmen zu können, wird eine Koordinatentransformation von Kamera- in Plattenkoordinatensystem durchgeführt. Die Platte, welche im Kamerabild als ein ver-

zerrtes Viereck zu sehen ist, soll bei dieser Transformation entzerrt und somit als Quadrat dargestellt werden. Das bedeutet, dass die Platte in ihrem eigenen Koordinatensystem angezeigt wird, was sich visuell als ein direkter Blick von oben auf die Platte vorstellen kann.

Es werden folgenden Annahmen getroffen. Die Platte mit Ball wird als ebene, quadratische Fläche angenommen, die schräg im dreidimensionalen Raum liegt. Auf der Kamera ist eine zweidimensionale perspektivische Projektion der Platte zu sehen. Da selbst bei großen Neigungswinkeln die gesamte Platte in der Nähe der Brennweite beziehungsweise im Bild als scharf erscheint, wird das vereinfachte Modell der Lochkamera angenommen. Das Modell der Lochkamera beschreibt auch, weshalb die quadratische Platte als verzerrtes Viereck angezeigt wird.

Im Lochkameramodell verlaufen alle Projektionsstrahlen durch die Lochblende, den Zentralpunkt, und werden auf eine zweidimensionale Bildebene projiziert (siehe Abbildung 34). Das hat zur Folge, dass jede Gerade, die nicht durch das Projektionszentrum verläuft, auf eine Gerade in der Bildebene abgebildet wird. Der Grund dafür ist, dass die Ebene, auf der die Projektionsstrahlen einer Geraden im Raum verlaufen, die Bildebene ebenfalls in einer Geraden schneidet. Dabei bleiben in vielen Fällen Längenverhältnisse und Winkel zwischen verschiedenen Geraden nicht erhalten. Die Platte, welche oben als eine quadratische Ebene charakterisiert wird, verliert bei der Zentralprojektion die rechten Winkel und die Seitenverhältnisse, sodass ein allgemeines Viereck auf die Bildebene projiziert wird [26].

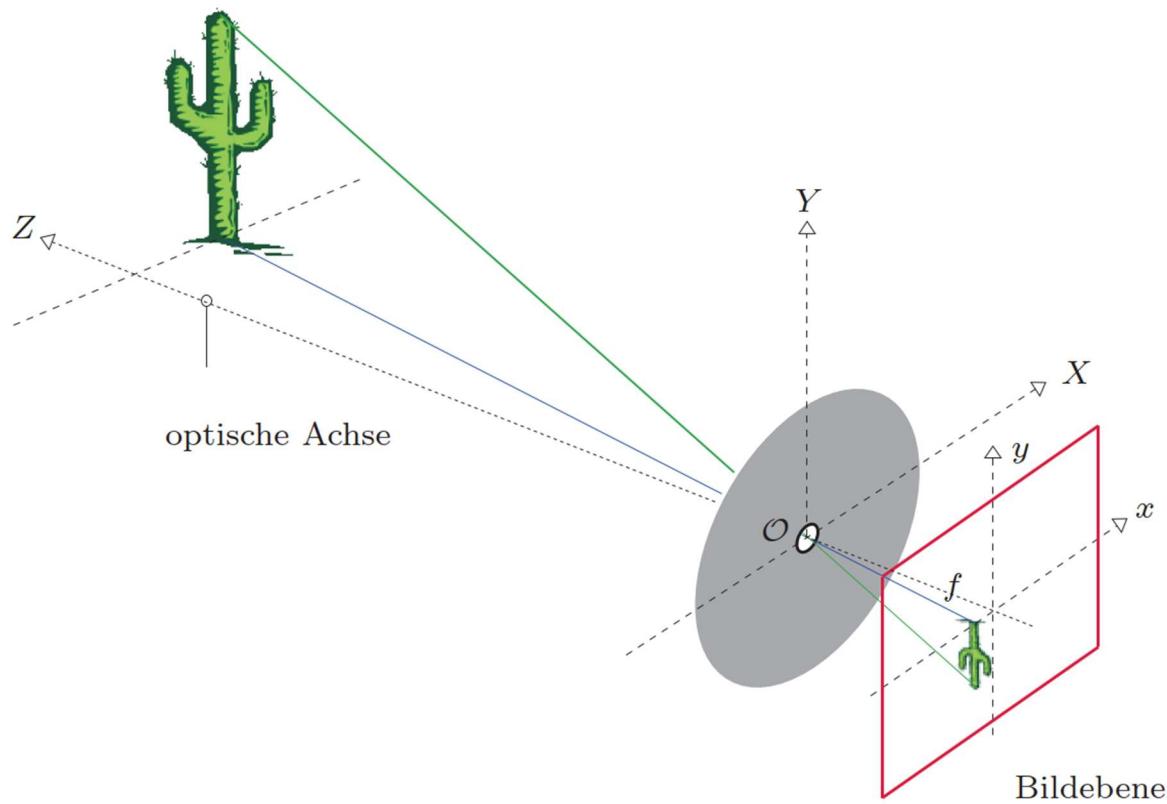


Abbildung 34: Modell der Lochkamera mit dargestellter Lochblende (Zentralpunkt)²⁷

Um allgemeine Vierecke zu verformen, eignet sich die perspektivische Transformation [30]. Damit kann die Projektion der Platte, das allgemeine Viereck, wieder zurück in ein Quadrat transformiert werden. Die perspektivische Transformation erfolgt mit Hilfe einer 3×3 Transformationsmatrix, welche somit die Punkte aus dem Kamerakoordinaten- system in das Plattenkoordinatensystem transformiert. Durch Erweiterung der Standardkoordinaten in homogene Koordinaten kann die perspektivische Transformation von einer ursprünglich nichtlinearen auf eine lineare Transformation vereinfacht werden. Das Prinzip der homogenen Koordinaten wird im folgenden Kapitel erklärt.

Homogene Koordinaten

Die homogenen Koordinaten sind eine Erweiterung der Standardkoordinaten und dienen zur einheitlichen Beschreibung von verschiedenen Transformationen, wie Translation,

²⁷ Quelle: Wilhelm Burger, Mark James Burge, Digitale Bildverarbeitung (2015) [30]

Rotation und perspektivischer Transformation [26]. Das heißt, es können mehrere Operationen, z.B. eine Translation und eine Rotation in eine einzige Transformationsmatrix zusammengefasst werden. Dies reduziert den Rechenaufwand und vermeidet Rundungsfehler durch mehrere Einzel-Transformationen [47].

Einem Punkt P mit den Koordinaten (x, y) werden die homogenen Koordinaten

$$\begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \begin{pmatrix} x_h \\ y_h \\ w \end{pmatrix}$$

zugeordnet, wobei die homogenisierende Koordinate w im allgemeinen Fall ungleich Null ist. Um aus homogenen Koordinaten x_h, y_h die Standardkoordinaten x, y zu ermitteln, muss durch w geteilt werden [48].

Zu beachten ist, dass mehrere homogene Koordinaten den gleichen Punkt beschreiben können. Zum Beispiel repräsentieren sowohl $(3, 2, 1)$ als auch $(6, 4, 2)$ den Punkt mit den Standardkoordinaten $(3, 2)$.

Berechnung der Transformationsmatrix

Wenn sowohl Kamera- als auch Plattenkoordinaten als homogene Koordinaten vorliegen, lässt sich die perspektivische Transformation wie folgt darstellen [26]:

$$\begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \quad \text{oder } X' = PX \quad (4.18)$$

Dabei ist die Transformationsmatrix P nicht eindeutig. Durch die Homogenisierung sind Kamera- und Plattenkoordinaten mit dem Faktor w beliebig skalierbar. Mit der Annahme $a_{33} \neq 0$ lassen sich beide Seiten immer so skalieren, dass $a_{33} = 1$ ist. Das Problem mit ursprünglich neun Variablen/Freiheitsgraden lässt sich auf eines mit acht reduzieren: [49]

$$\begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \quad (4.19)$$

Ausmultipliziert ergibt dies:

$$w'x' = a_{11}wx + a_{12}wy + a_{13}w$$

$$w'y' = a_{21}wx + a_{22}wy + a_{23}w$$

$$w' = a_{31}wx + a_{32}wy + w$$

Durch Substitution von w' kann die dritte Gleichung eliminiert werden:

$$(a_{31}wx + a_{32}wy + w)x' = a_{11}wx + a_{12}wy + a_{13}w$$

$$(a_{31}wx + a_{32}wy + w)y' = a_{21}wx + a_{22}wy + a_{23}w$$

\Leftrightarrow

$$a_{31}xx' + a_{32}yx' + x' = a_{11}x + a_{12}y + a_{13}$$

$$a_{31}xy' + a_{32}yy' + y' = a_{21}x + a_{22}y + a_{23}$$

Isolieren von x' und y' ergibt (4.20).

$$x' = a_{11}x + a_{12}y + a_{13} - a_{31}xx' - a_{32}yx' \quad (4.20)$$

$$y' = a_{21}x + a_{22}y + a_{23} - a_{31}xy' - a_{32}yy' \quad (4.21)$$

Dieser Ausdruck kann in ein lineares Gleichungssystem umgeschrieben werden: [49]

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad (4.22)$$

Für dieses Gleichungssystem mit acht Variablen werden acht Gleichungen benötigt. Pro Punktpaar $((x, y)$ und (x', y')) ergeben sich zwei Gleichungen. Das heißt, für vier Punktpaare lässt sich das Gleichungssystem (4.22) eindeutig lösen (siehe Formel (4.23)). Es bieten sich die vier Eckpunkte der Platte an. Im Kamerakoordinatensystem werden die Eckpunkte bereits gefunden. Da die Platte eben und quadratisch ist, können

im Plattenkoordinatensystem vier beliebige Punkte gewählt werden, die ein Quadrat bilden. Ausgewählt werden die Punkte $(0, 0)$, $(480, 0)$, $(480, 480)$, $(0, 480)$.

Um die perspektivische Transformation exakt zu bestimmen, werden mindestens vier Punktpaare benötigt. Deshalb wird die perspektivische Transformation auch Vierpunkt-Abbildung genannt [26].

Mit den vier Punktpaaren ergibt sich folgendes Gleichungssystem:

$$\begin{bmatrix} x_o & y_o & 1 & 0 & 0 & 0 & -x_o x'_o & -y_o x'_o \\ 0 & 0 & 0 & x_o & y_o & 1 & -x_o y'_o & -y_o y'_o \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 x'_1 & -y_1 x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 y'_1 & -y_1 y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 x'_2 & -y_2 x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 y'_2 & -y_2 y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 x'_3 & -y_3 x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 y'_3 & -y_3 y'_3 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x'_o \\ y'_o \\ x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} \quad (4.23)$$

Alle Koordinaten (x_k, y_k) und (x'_k, y'_k) , $k = \{1, 2, 3, 4\}$ sind gegeben, sodass sich das Gleichungssystem für alle Unbekannten a_{11} bis a_{32} lösen lässt. Damit liegt die Transformationsmatrix P komplett vor [26].

Durch die perspektivische Transformation entsteht eine eindeutige Beziehung zwischen den Abmessungen der Platte ($300 \times 300 \text{ mm}$) und der Abbildung der Platte ($480 \times 480 \text{ px}$). Somit lassen sich digitale Größen, angegeben in Pixel, in reale Größen, angegeben in Millimetern, umrechnen und umgekehrt. Dies wird sich bei der Regelung (vgl. Kapitel 3 und 4.6.2) zu Nutze gemacht.

Perspektivische Transformation von Punkten und Frames

Um einen Punkt von Kamerakoordinaten in Plattenkoordinaten zu transformieren, muss dieser zunächst in homogene Koordinaten umgewandelt werden ($w = 1$ bietet sich an). Der homogenisierte Punkt in Vektorform kann dann mit der berechneten Transformationsmatrix multipliziert werden. Der resultierende Punkt ist ebenfalls in homogenen Koordinaten angegeben und muss mittels Division durch w wieder zurück in Standardkoordinaten umgewandelt werden. Die Ballkoordinaten werden nach diesem Prinzip in das

Plattenkoordinatensystem transformiert und stellen damit das Endergebnis der gesamten Bildverarbeitung dar.

Zusätzlich besteht durch openCV die Möglichkeit einen ganzen Kameraframe perspektivisch zu transformieren. Da durch die Transformation Pixel aus dem Eingangsbild nicht auf ganze Pixel des Ausgangsbildes fallen, ist zusätzlich eine Interpolation notwendig. Da diese Anwendung nur zur visuellen Veranschaulichung genutzt wird und keinen funktionalen Nutzen für diese Arbeit hat, wird die perspektivische Transformation von Frames nicht ausführlich beschrieben. Details zum sogenannten Resampling und zur Interpolation sind in „Digitale Bildverarbeitung“ [30] von W. Burger in den Kapiteln 21.2 und 22 ausführlich erklärt.

Implementierung im Programmcode

Zur Anwendung der perspektivischen Transformation werden im Programmcode in der Datei „plate_detection.py“ drei Methoden definiert (pt = perspektivische Transformation):

```
get_pt_matrix(pts, pt_size)
pt_point(point, matrix)
pt_image(img, matrix, pt_img_size)
```

Mit der Methode get_pt_matrix wird die Transformationsmatrix der perspektivischen Transformation berechnet:

```
def get_pt_matrix(pts, pt_size):
    pts = pts.astype("float32")
    pt_x = pt_size[0]
    pt_y = pt_size[1]

    pts_pt = np.array([[0, 0],
                      [pt_x, 0],
                      [pt_x, pt_y],
                      [0, pt_y]],
                      dtype="float32")

    matrix = cv2.getPerspectiveTransform(pts, pts_pt)

    return matrix
```

Dazu werden zum einen die vier Eckpunkte der approximierten Plattenkontur (`pts`) und zum anderen die beiden Seitenlängen des resultierenden Rechtecks als Tupel (`pt_size`) übergeben. In diesem Fall soll das resultierende Rechteck ein Quadrat mit der Seitenlänge von 480 Pixel sein, sodass bei Funktionsaufruf beide Seitenlängen des Rechtecks auf 480 Pixel festgelegt werden. Innerhalb der Methode wird die openCV-Funktion `cv2.getPerspectiveTransform` benutzt, in welcher die oben beschriebene Berechnung stattfindet. Vorbereitend dafür werden aus den beiden Seitenlängen die vier Eckpunkte des Rechtsecks erzeugt und `pts` sowie `pts_pt` in den Datentyp `float32` umgewandelt. Abschließend wird die berechnete Transformationsmatrix zurückgegeben.

Um einen einzelnen Punkt, wie zum Beispiel die Ballkoordinaten, zu transformieren, wird die Methode `pt_point` angewendet:

```
def pt_point(point, matrix):
    point_h = np.array([point[0], point[1], 1])

    point_pt_h = np.dot(matrix, point_h)

    point_pt = np.array([point_pt_h[0]/point_pt_h[2],
                        point_pt_h[1]/point_pt_h[2]])
    point_pt = np.around(point_pt).astype("int")

    return point_pt
```

In der Methode wird die oben beschriebene Matrix-Vektor-Multiplikation von Transformationsmatrix und homogener Koordinate mit Hilfe der Numpy-Funktion `np.dot` implementiert. Zunächst muss der übergebene Punkt (`point`) in homogene Koordinaten transformiert werden. Nach Berechnung des transformierten Punktes werden mittels Division durch die homogenisierte Koordinate die Standardkoordinaten berechnet. Diese werden abschließend mit `np.around` gerundet und in den Integer-Datentyp umgewandelt. Bei Transformation der Ballkoordinaten ist die zurückgegebene Variable `point_pt` der Grundstein für alle weiteren Schritte im Bereich der Regelung und Geschwindigkeitsberechnung.

Zur perspektivischen Transformation eines ganzen Bildes dient die Methode `pt_image`:

```
def pt_image(img, matrix, pt_img_size):
    pt_img = cv2.warpPerspective(img, matrix, pt_img_size)

    return pt_img
```

Diese Funktion ruft direkt die openCV-Funktion `cv2.warpPerspective` auf, in welcher zusätzlich mittels Interpolation das perspektivisch transformierte Bild berechnet wird. Neben Kamerabild und Transformationsmatrix müssen die Seitenlängen des resultierenden Bildes übergeben werden (`pt_img_size`). Auch in diesem Fall werden bei Funktionsaufruf beide Seitenlängen auf 480 Pixel festgelegt.

4.6. Berechnung der Plattenstellung

4.6.1. Berechnung der Ballgeschwindigkeit

Die Zustandsgrößen sind für die Regelung und für die Berechnung der Plattenstellung unverzichtbar. Sie beinhalten die Position und die Geschwindigkeit der Kugel relativ zur Platte. Beide Größen werden jeweils als Vektor, bestehend aus X- und Y-Koordinate, angegeben (wie in Kapitel 3.1 ausführlich erläutert).

Die Berechnung der Geschwindigkeit erfolgt, analog wie in der klassischen Physik, getrennt für beide Achsen mit den Formeln:

$$v_x \approx \bar{v}_x = \frac{S_x}{T} = \frac{\Delta X}{\Delta t} \quad (4.24)$$

$$v_y \approx \bar{v}_y = \frac{S_y}{T} = \frac{\Delta Y}{\Delta t} \quad (4.25)$$

Die gesuchten Geschwindigkeiten v_x und v_y werden hierbei durch die mittleren Geschwindigkeiten \bar{v}_x und \bar{v}_y angenähert, welche mit dem Quotienten aus der zurückgelegten Strecke S entlang der jeweiligen Koordinatenachse und der vergangene Zeit T berechnet wird. Da es sich bei Dividend und Divisor jeweils um eine Differenz handelt, wird die mittlere Geschwindigkeit von Bild zu Bild ausgerechnet.

Die aktuelle Position des Balls wird, wie in Kapitel 4.5.4 beschrieben, erfasst und mithilfe der perspektivischen Transformation in das Plattenkoordinatensystem umgerechnet (siehe Kapitel 4.5.5). Die hier verwendeten Ballkoordinaten sind somit bereits in das Plattenkoordinatensystem umgerechnet und können direkt zur Geschwindigkeitsberechnung verwendet werden.

Die Zeit der Bilddurchlaufzeit und somit der zur Berechnung verwendeten Position, geht aus dem Zeitstempel hervor, welcher jedem Bild angehängt wird (siehe Kapitel 4.5.1).

```
frame, frame_time = cam.read()
```

Jeder Zyklus beginnt mit einer Abfrage der Differenz zwischen der aktuellen Zeit und der Zeit der Erstellung des vorherigen Frames. Die minimale verstrichene Zeit muss 100 Millisekunden betragen, damit der neue Zyklus durchlaufen werden kann, was die maximale FPS des Prozesszyklus auf 10 beschränkt. Grund hierfür sind auffällig hohe Geschwindigkeitssprünge und die damit verbundenen Störungen bei der Regelung, die in den ersten Testläufen verzeichnet werden. Die dieser Maßnahme zugrundeliegende Fehleranalyse wird im Kapitel 4.8 erläutert.

```
# Neuer Frame soll erst 0.1s nach dem letzten aufgenommen werden  
# (fixe 10 FPS)  
while time.time() - frame_time_last < 0.1:  
    pass
```

Der Zeitpunkt und die Ortskoordinate des vorherigen Frames werden bereits am Ende des vorangehenden Zyklus auf die dafür vorgesehene Variable geschrieben. Eine Ausnahme bildet der erste Zyklus, hier werden Zeit und Ort den Initialisierungsdaten entnommen (siehe Kapitel 4.4).

```
# Speichere die Frame Time für den nächsten Schleifendurchlauf  
frame_time_last = frame_time  
  
ball_coords_last = ball_coords_pt_cnt
```

Wenn sowohl die aktuellen Ballkoordinaten als auch die letzten Ballkoordinaten vorhanden sind, werden die vier Variablen der Geschwindigkeitsberechnung übergeben.

```

if ball_coords_last is not None:
    # Geschwindigkeit
    if frame_time != frame_time_last:
        v = plate_control.calc_velocity(frame_time,
                                         frame_time_last,
                                         ball_coords_pt_cnt,
                                         ball_coords_last)

```

Die Funktion hierfür ist ausgelagert in der Datei „plate_control.py“ und beinhaltet die Formeln (4.24) und (4.25).

```

def calc_velocity(time, time_last, coords, coords_last):
    vx = (coords[0] - coords_last[0]) / (time - time_last)
    vy = (coords[1] - coords_last[1]) / (time - time_last)

    return (vx, vy)

```

Da sich die Ortsangabe auf eine Pixelkoordinate bezieht, hat auch die berechnete Geschwindigkeit die Einheit Pixel pro Millisekunde. Für die Berechnung der Plattenwinkel ist dies aber unerheblich, da im Bereich der perspektivischen Transformation eine feste Zuordnung zwischen Plattenlänge in Pixel und in Millimeter aufgestellt wird.

4.6.2. Programmimplementierung der Regelung

Auf Basis der in Kapitel 3 theoretisch erläuterten Regelung wird nun die programmtechnische Implementierung erklärt. Im ersten Schritt werden die, durch die perspektivische Transformation errechneten, Ballkoordinaten auf den verschobenen Koordinatenursprung umgerechnet. Der definierte Koordinatenursprung befindet sich in der Mitte des 480x480 Pixel großen Frames. Die Ballkoordinaten nach der perspektivischen Transformation haben ihren Ursprung in der oberen linken Ecke, weshalb die Koordinaten vom Plattenmittelpunkt von den Ballkoordinaten subtrahiert werden.

```

plate_center_pt = (240, 240)
ball_coords_pt_cnt = (ball_coords_pt[0] - plate_center_pt[0],
                      ball_coords_pt[1] - plate_center_pt[1])

```

Der folgende Programmcode zeigt, wie auf Basis der berechneten Ballposition und seiner Geschwindigkeit der Zustandsvektor definiert wird.

```

x = np.array([ball_coords_pt_cnt[0],
              v[0],
              ball_coords_pt_cnt[1],
              v[1]]).T

```

Um den Ball in einer Kreisbahn definierten Kreisbahn zu verfahren, wird eine Funktion erstellt, welche auf Basis des Zeitstempels (`frame_time`) die aktuelle die aktuelle Sollposition des Balls berechnet. Alternativ kann auch eine feste Koordinate als Sollwertvorgabe gewählt werden.

```
# Führungsgrößen (Mittelpunkt oder Kreisbahn)
w = np.array([r*np.sin(omega * frame_time),
              r*np.cos(omega * frame_time)])
# w = np.array([0, 0]).T
```

Um mittels Regelgesetz‘ den Plattenwinkel zu berechnen, wird in der ausgelagerten Datei „plate_control.py“ die Funktion `get_state_space` definiert. In dieser Funktion wird das Zustandsraummodell aufgestellt und die Regelmatrix sowie der Vorfilter berechnet. Der Vorfilter und die Regelmatrix werden als Rückgabewert an die Hauptdatei übergeben.

```

def get_state_space(g_mm, pole):
    g_px = 480/300 * g_mm
    b = 5 / 7 * g_px

    # Matrizen des Zustandsmodells
    A = np.array([[0., 1., 0., 0.],
                  [0., 0., 0., 0.],
                  [0., 0., 0., 1.],
                  [0., 0., 0., 0.]))

    B = np.array([[0., 0.],
                  [b, 0.],
                  [0., 0.],
                  [0., b]]))

    C = np.array([[1., 0., 0., 0.],
                  [0., 0., 1., 0.]))

    # Berechnung Regelmatrix
    r1 = 7 / (5*g_px) * pole[0] ** 2
    r2 = -2 * 7 / (5*g_px) * pole[1]

    R = np.array([[-r1, 0.],
                  [-r2, 0.],
                  [0., -r1],
                  [0., -r2]]).T

    # Berechnung Vorfilter
    V = np.linalg.inv(np.dot(np.dot(C, np.linalg.inv(np.subtract(
        np.dot(B, R), A))), B))

    return R, V

```

Auf Basis des Zustandsvektors, des Vorfilters, der Regelmatrix und der Sollwertvorgabe wird das Regelgesetz im Prozesszyklus angewendet.

`u_rad = np.dot(V, w) - np.dot(R, x)`

Die Lösung des Regelgesetzes gibt den Plattenwinkel in Radiant an. Für die weitere Berechnung des PWM-Signals der Servomotoren werden die Plattenwinkel in Grad umgerechnet:

`u_deg = 180/np.pi * u_rad`

4.7. Ansteuerung der Servomotoren

Die Position der Platte wird über die Ansteuerung der Servomotoren geregelt. Die Ansteuerung der Servomotoren erfolgt über ein PWM Signal (siehe Kapitel 4.7.3). Um den Zusammenhang zwischen Pulsdauer des PWM-Signals und dem Plattenwinkel zu bestimmen, wird ein mathematisches Modell und eine Approximationsfunktion berechnet.

Im Zuge der Studienarbeit hat sich gezeigt, dass die Abweichung von den vorgegebenen Sollwerten bei den Approximationsfunktionen, im Vergleich zu dem mathematischen Modell, deutlich geringer sind, weshalb die Approximationsfunktionen im fertigen Programmcode anwendet wird (Das mathematische Modell befindet sich in Anhang 1)

4.7.1. **Approximation**

Bei der Approximationsfunktion handelt es sich um eine Näherungsfunktion, welche den Zusammenhang zwischen Pulsdauer und Plattenwinkel beschreibt.

Es wird eine Approximation anstatt einer Interpolation gewählt, da ein lineares Verhalten zwischen Pulsdauer und Winkelposition beobachtet wird und die Daten aufgrund der Messmethode (siehe folgenden Absatz) Mess- und Beobachtungsfehlern unterliegen. Die Interpolation mittels Lagrange-Polynomen neigt zu starker Oszillation im Randbereich welches das Systemverhalten negativ beeinträchtigt und kubische Splines sind deutlich rechenaufwändiger und durch die ungenaue Messmethode nicht nötig.

Die Approximationsfunktion basiert auf einer aufgenommenen Datentabelle. Auf Basis von 10 Messwerten, welche den Zusammenhang zwischen Pulsdauer des PWM-Signals und Plattenwinkel beschreibt, wird die Approximationsfunktion berechnet. Zur Aufnahme der Messwerte wird der Raspberry Pi über die pigpio Bibliothek angesteuert und erzeugt ein PWM-Signal mit definierter Pulsdauer. Der resultierende Plattenwinkel wird über eine digitale Wasserwaage ermittelt.

Im Folgenden wird beschrieben, wie auf Basis von Wertepaaren (x_i, y_i) mit $i = 0 \dots m$ eine Approximationsfunktion $p(x)$ angenähert wird.

Bei einer Approximation wird die Funktion $p(x)$, ein Polynom n -ten Grades, $n < m$, so bestimmt, dass die Summe der Fehlerquadrate

$$\sum_{i=0}^m (p(x_i) - y_i)^2 \tag{4.26}$$

minimal wird [50].

Die Approximation erfolgt mittels eines Polynoms dritten Grades, da Polynome höheren Grades zu Oszillationen neigen und dieses bereits hinreichend genaue Annäherung erzielt. Die allgemeine Approximationsfunktion für ein Polynom $p(x)$ dritten Grades lautet:

$$p(x) = a_0 * x^3 + a_1 * x^2 + a_2 * x + a_3$$

Um die Koeffizienten a_j mit $j = 0 \dots 3$ der Funktion zu bestimmen, wird die Methode der kleinsten Fehlerquadrate (4.26) angewendet.

$$E(a_0, a_1, a_2, a_3) = \sum_{i=0}^m ((p(x_i) - y_i)^2 \rightarrow \min$$

Im nächsten Schritt wird der Gradient der Funktion $E(a_0, a_1, a_2, a_3)$ gleich Null gesetzt. Es entsteht ein Gleichungssystem. Durch Lösen des Gleichungssystems sind die Koeffizienten bekannt, mit denen die Approximationsfunktion vollständig bestimmt werden kann.

Es werden zwei Approximationsfunktionen (für X-Achse und Y-Achse) erstellt, weil bei gleicher Laufrichtung der Servomotoren die translatorische Bewegung der Verbindungsstangen zur Balancierplatte in entgegengesetzte Richtung stattfindet. Zudem ist eine Spiegelung der Approximationsfunktionen (Graph betrachtet) nicht möglich, da sich aufgrund der Fertigungstoleranzen bei gleicher Pulsdauer ein unterschiedlicher Stellwinkel einstellt.

Die verwendeten Approximationsfunktionen für X-Achse und Y-Achse werden durch den Onlinerechner der Website: <https://valdivia.staff.jade-hs.de/interpol.html> erstellt.

Die folgende Abbildung stellt die Approximationsfunktion

$$p(x) = f(x) = 0.03904x^3 + 0.00165x^2 + 35.81x + 1496$$

der X-Achse dar. Im Diagramm ist auf der X-Achse der Plattenwinkel α und auf der Y-Achse die Pulsdauer aufgetragen. Die Wertepaare, der aufgenommenen Messreihe sind in der nebenstehenden Tabelle 3 abzulesen.

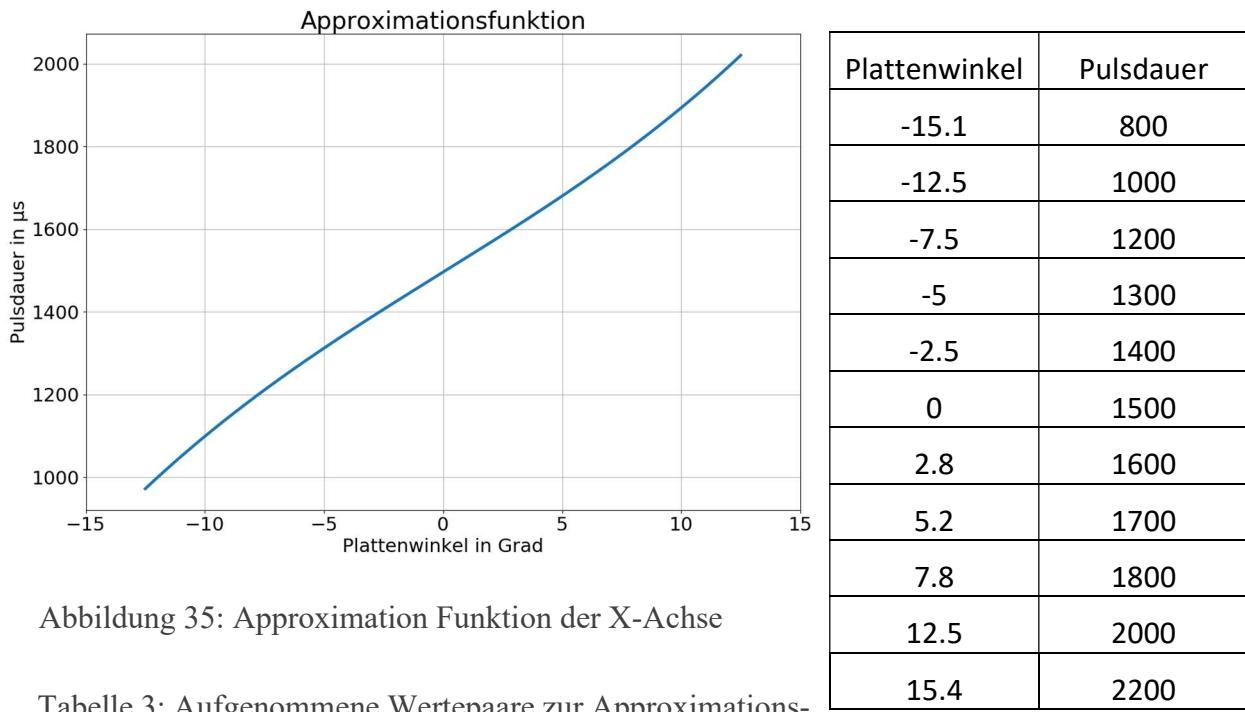


Abbildung 35: Approximation Funktion der X-Achse

Tabelle 3: Aufgenommene Wertepaare zur Approximationsfunktion

4.7.2. Optimierung der Approximation

Bei der Implementierung in dem Programm hat sich gezeigt, dass sich der Ball auf der Platte gut mittels der Approximationsfunktion regeln lässt. Die Kreisbahn des Balls verlief jedoch nicht symmetrisch um den Kreismittelpunkt, sondern in Bezug auf das Plattenkoordinatensystem positiv in X- und Y-Achse verschoben. Durch Einführen von zwei additiven Korrekturfaktoren konnte dieses Problem behoben werden.

4.7.3. Erzeugung des PWM-Signals per Fernzugriff

Wie bereits in Kapitel 2.2.2 beschrieben, werden die beiden Servomotoren über ein PWM-Signal angesteuert. Ein PWM-Signal ist im Allgemeinen ein periodisches Rechtecksignal, bei dem bei konstanter Frequenz die Pulsdauer moduliert wird [51]. In diesem Fall hat das PWM-Signal eine Frequenz von 50 Hz und Pulsdauern von ca. 1 ms bis 2 ms.

Das PWM-Signal wird vom Raspberry Pi erzeugt und über die GPIO-Pins ausgegeben.

Ursprünglich wurde das PWM-Signal auf Grundlage der vorherigen Arbeit mit der Bibliothek „RPi.GPIO“ erzeugt. Dort kann mit nur wenigen Programmzeilen die PWM-Frequenz, das Tastverhältnis und der jeweilige Ausgabe-Pin des PWM-Signals festgelegt werden.

Dies hat trotz korrekt gewählter Parameter zu einem „Zittern“ der Servomotoren geführt. Das Zittern war besonders unter Prozessorlast (zum Beispiel bei gleichzeitiger Bildverarbeitung) äußerst stark und hat die Regelung der Kugel auf der Platte praktisch unmöglich gemacht.

Der Grund für das Zittern scheint die Generierung des PWM-Signals per Software zu sein. In der offiziellen Dokumentation der „RPi.GPIO“-Bibliothek wird beschrieben, dass das PWM-Signal per Software erzeugt wird und nicht für zeitkritische Anwendungen geeignet ist. Ein Grund dafür ist, dass der Linux-Kernel, der Kern des Betriebssystems des Raspberry Pi, nicht für Echtzeit-Anwendungen geeignet ist, sondern auf Multitasking-Anwendungen ausgelegt ist [52]. Deswegen kann es dazu kommen, dass ein anderer Prozess Priorität über die PWM-Signalerzeugung erlangt und die zeitliche Ansteuerung der PWM-Pulse verzögert ist. Bei regelmäßigen, zeitlich schwankenden Verzögerungen entsteht eine Art Taktzittern, welches auch Jitter bezeichnet wird.

Um zu überprüfen, ob Jitter vorhanden ist und wie stark die jeweiligen Verzögerungen sind, wird das PWM-Signal mit einem Oszilloskop bei einer vorgegebenen Pulsdauer von $1,5\text{ ms}$ analysiert. Dabei wird mit einem Trigger und dem Persistence-Modus gearbeitet. Ausgelöst wird der Trigger bei der fallenden Flanke des Pulses, sodass ein stehendes Bild erzeugt wird, bei dem die fallende Flanke immer am Zeitpunkt $t = 0\text{ ms}$ ist. Der Persistence-Modus erlaubt es, neben dem aktuellen Puls noch ältere Pulse zu sehen, was bei 50 Pulsen pro Sekunde hilfreich ist. Eingestellt wird eine Persistence-Dauer von 10 Sekunden, sodass alle Pulse der letzten 10 Sekunden zu sehen sind. Das Ergebnis der Messung des Software-PWM-Signals ist in folgender Abbildung zu sehen.

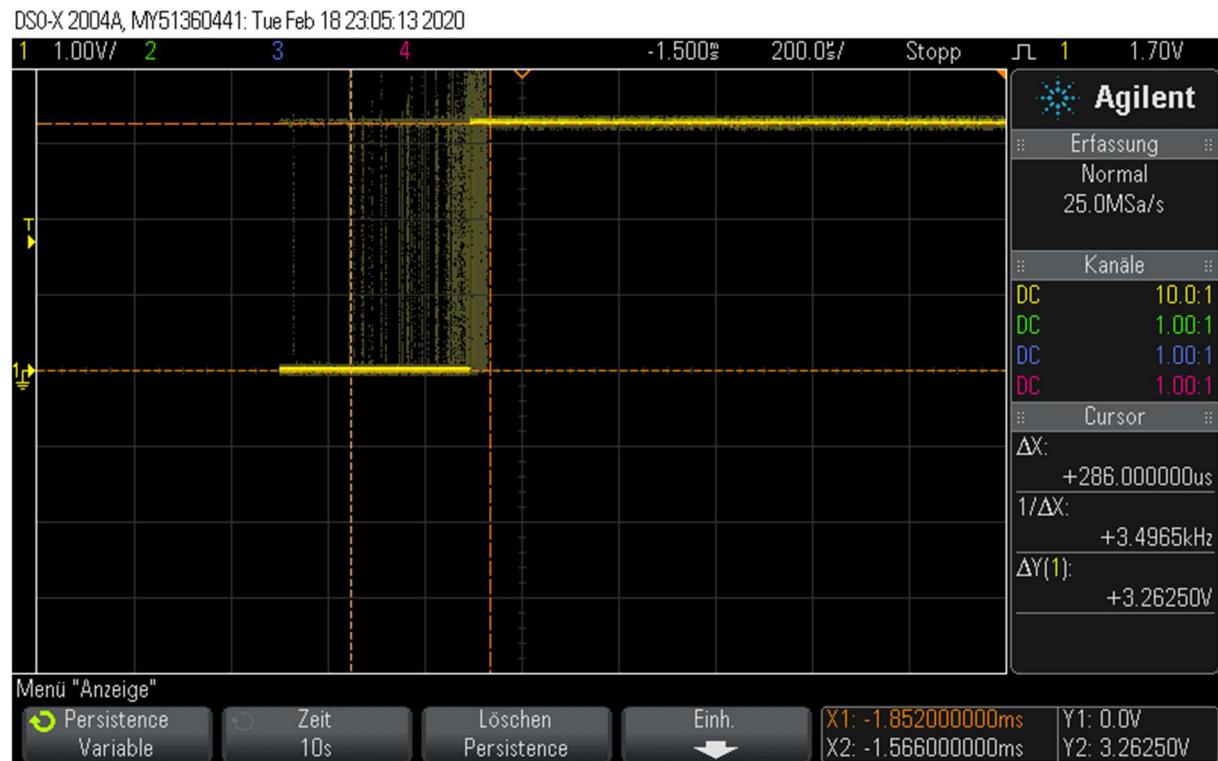


Abbildung 36: Messung des Jitters des Software-PWM-Signals

Im Diagrammbereich ist die steigende Flanke vergrößert dargestellt, um Details besser zu erkennen. Die beiden gelben horizontalen Linien sind die Spannungen vor und nach den steigenden Flanken. Es ist deutlich zu erkennen, dass die steigenden Flanken, die vertikalen grüngelben Linien, großflächig verteilt sind. Die beiden vertikalen Hilfslinien X_1 und X_2 sind auf den Anfang und das Ende des größten Teils der steigenden Flanken gelegt. Ihre Werte sind am unteren Bildschirmrand rechts zu sehen:

$$X_1 = -1,852 \text{ ms} \text{ und } X_2 = -1,566 \text{ ms}.$$

Das berechnete ΔX beträgt $286 \mu\text{s}$. Dies zeigt, dass starke Verzögerungen und Jitter vorhanden sind. Eine Änderung der Pulsdauer im Extremfall von $286 \mu\text{s}$ würde einer Winkeländerung des Servomotors von ca. 25° entsprechen. Selbst deutlich kleinere Schwankungen in der Pulsdauer haben starke Auswirkungen auf den Winkel der Servomotoren und sind nicht akzeptabel. Zusätzlich fällt auf, dass selbst die kürzesten Pulsdauern mit der steigenden Flanke 1,566 ms vor der fallenden Flanke noch deutlich län-

ger als vorgegebenen Pulsdauern von $1,5\text{ ms}$ sind. Somit ist die oben genannte Hypothese bestätigt und ein softwareseitiges PWM-Signal ist nicht für diese Anwendung geeignet.

4.7.3.1 **Hardware-PWM-Signalerzeugung**

In Internetforen wird für Servomotor-Ansteuerung mit dem Raspberry Pi statt dem Software-PWM häufig ein Hardware-PWM vorgeschlagen [53] [54]. Hardware-PWM beim Raspberry Pi bedeutet, dass das PWM-Signal von Hardware-Modulen erzeugt wird, die ausschließlich für die Steuerung der PWM-Signale verantwortlich sind. Diese werden nicht von anderen Prozessen unterbrochen und sollen ein stabiles PWM-Signal ohne Jitter erzeugen.

Eine weit verbreitete Bibliothek zur PWM-Signalerzeugung per Hardware ist „pigpio“. Diese Bibliothek bietet die Möglichkeit, auf dem Raspberry Pi Hardware-PWM-Signale zu erzeugen, ohne dabei auf zusätzliche Hardware zurückzugreifen [23].

Die Pin- und PWM-Konfiguration ist mit diesem Modul ebenfalls unkompliziert und wird in einem eigenen Unterpunkt beschrieben. Mit Verwendung von „pigpio“ ist kein Zittern der Servomotoren mehr zu erkennen auf und eine flüssige Regelung ist möglich.

Zur Überprüfung wird auch beim Hardware-PWM der Jitter mit dem Oszilloskop gemessen, siehe folgende Abbildung. Alle Versuchsparameter bleiben gegenüber der Messung des Software-PWM-Signals unverändert.

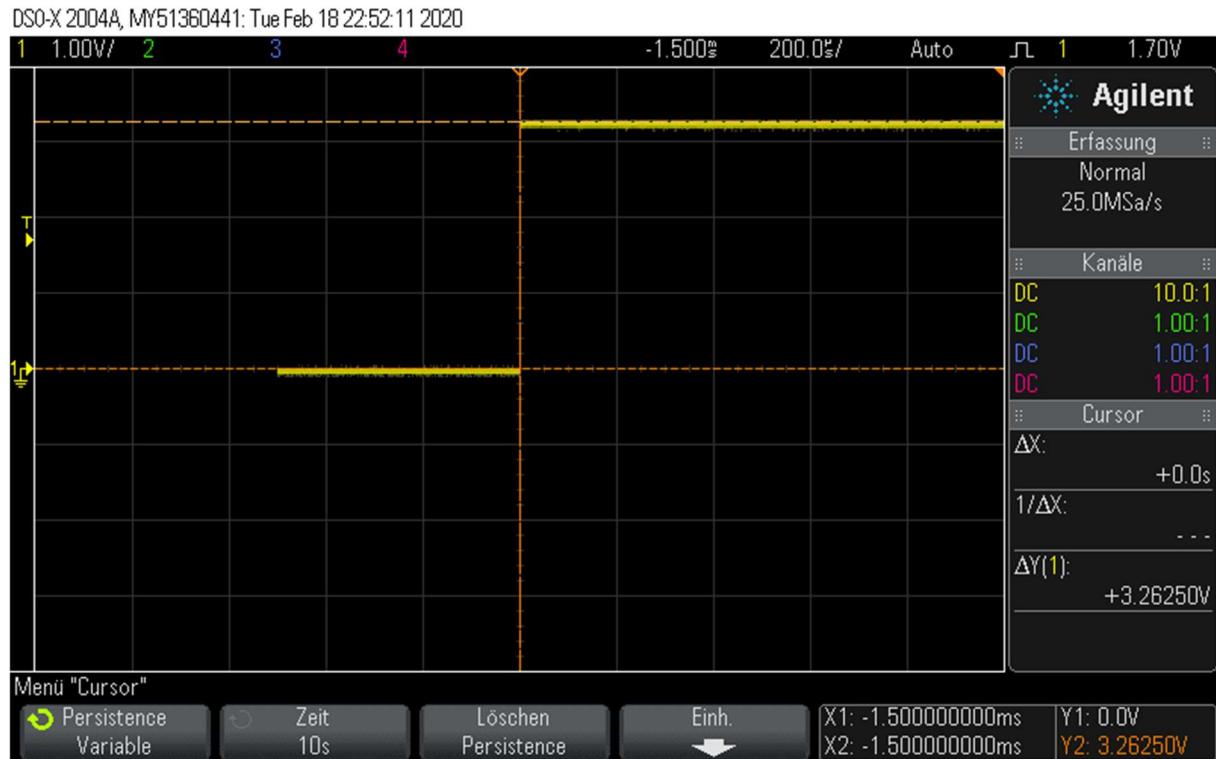


Abbildung 37: Messung des Jitters des Hardware-PWM-Signals

Der Unterschied zum Software-PWM ist deutlich zu sehen und selbst bei starker Vergrößerung sind keine zeitlichen Schwankungen der steigenden Flanke zu erkennen. Auch ist durch die vertikalen Hilfslinien $X1$ und $X2$ erkennlich, dass die Pulsdauern beim Hardware-PWM exakt bei $1,5 \text{ ms}$ liegen.

Damit ist bestätigt, dass die Hardware-PWM-Signalerzeugung mit der „pigpio“-Bibliothek kein Jitter verursacht und für die Servomotor-Ansteuerung geeignet ist.

4.7.3.2 Fernsteuerung des PWM-Signals über den PC

Die Bibliothek „pigpio“ bietet neben der Hardware-PWM-Signalerzeugung weitere Funktionen an. Ein Feature ist der „pigpio“-Daemon (kurz: „pigpiod“), ein Hintergrundprozess, der auf dem Raspberry Pi läuft und pigpio-Signale von anderen Geräten empfangen kann. Das heißt, mit dem pigpiod lassen sich die PWM-Signale auf dem Raspberry Pi zum Beispiel von einem PC fernsteuern.

Das bietet mehrere Vorteile. Da die GPIO-Pins des Raspberry Pi nun ferngesteuert werden können, kann das gesamte Programm auf einem Desktop-PC ausgeführt werden.

Die stärkere Rechenleistung des PCs kann ausgenutzt werden und die gewohnte Programmierumgebung PyCharm unter Windows steht zur Verfügung.

Damit der Raspberry Pi ferngesteuert werden kann, müssen Raspberry Pi und PC innerhalb eines Netzwerks miteinander verbunden sein. Die Verbindung wird mit einer Punkt-zu-Punkt-Verbindung mittels eines herkömmlichen Ethernet-Kabels realisiert. Für PC und Raspberry Pi werden jeweils folgende statische IP-Adressen konfiguriert mit einer Klasse C Subnetzmaske und somit gleicher Netzwerkadresse:

Gerät	IP-Adresse	Subnetzmaske
Desktop-PC	192.168.0.2	255.255.255.0
Raspberry Pi	192.168.0.3	255.255.255.0

Tabelle 4: Netzwerk-Konfiguration für Desktop-PC und Raspberry Pi

Der „pigpiod“ wird im Raspberry Pi automatisch bei jedem Bootvorgang ausgeführt, sodass eine Fernsteuerung direkt beim Start und ohne Kommandozeilenbefehle möglich ist.

Durch die Fernsteuerung mit dem PC sind keine Verzögerungen gegenüber der direkten Ansteuerung auf dem Raspberry Pi festzustellen. Durch die höhere Rechenleistung läuft das gesamte Programm inklusive Bildverarbeitung und Regelung auf dem PC sogar deutlich flüssiger als auf dem Raspberry Pi.

4.7.3.3 Konfiguration „pigpio“ im Programmcode

Die Konfiguration der Pins und des PWM-Signals ist mit „pigpio“ unkompliziert und kann in wenigen Codezeilen realisiert werden. Dazu müssen zunächst die Pins des Raspberry Pi-GPIOs konfiguriert werden.

Im ersten Schritt werden die beiden Variablen `ServoX` und `ServoY` initialisiert. Sie enthalten jeweils die GPIO-Nummer des Pins, an dem das PWM-Signal für den jeweiligen Servomotor ausgegeben werden soll:

```
ServoX = 20
ServoY = 21
```

Das PWM-Signal für den Servomotor, der die X-Achse steuert, wird an dem GPIO 20 ausgegeben und das PWM-Signal für den Servomotor der Y-Achse wird an dem GPIO 21 ausgegeben. Hierbei ist zu beachten, dass die GPIO-Nummerierung verwendet wird und nicht die allgemeine Pin-Nummerierung des Raspberry Pis. Das heißt GPIO 20 und GPIO 21 sind an den Pins 38 und 40 respektive [55].

Um Zugriff auf die GPIO-Pins des Raspberry Pi zu erhalten, muss folgende Funktion ausgeführt werden:

```
pi = pigpio.pi(host="192.168.0.3")
```

Als Host wird hier der Raspberry Pi durch Angabe der konfigurierten statischen IP-Adresse ausgewählt. Falls die Funktion nicht auf dem PC, sondern auf dem Raspberry Pi selbst ausgeführt wird, darf keine Host-IP-Adresse übergeben werden. Das bedeutet, dass Zugriff auf die geräteeigenen GPIO-Pins gewährt wird [56].

Im nächsten Schritt müssen mit der Funktion `pi.set_mode` die beiden Pins als Output konfiguriert werden:

```
pi.set_mode(ServoX, pigpio.OUTPUT)
pi.set_mode(ServoY, pigpio.OUTPUT)
```

Dazu wird die Pin-Nummer des jeweiligen Servos und die Variable `pigpio.OUTPUT` übergeben. Die Initialisierung der Pins ist hiermit abgeschlossen.

Die Winkelstellung der Servomotoren wird während der Dauerschleife eingestellt. Bei Nutzung der Funktion `pi.set_servo_pulsewidth` muss lediglich die GPIO-Nummer des jeweiligen Pins und die Pulsbreite des PWM-Signals angegeben werden. Die Frequenz von 50 Hz wird als allgemeingültig für Servomotoren angenommen und muss nicht angegeben werden. Somit lässt sich in einer Programmzeile die Stellung des jeweiligen Servomotors einstellen:

```
pi.set_servo_pulsewidth(ServoX, u_servo_x)
pi.set_servo_pulsewidth(ServoY, u_servo_y)
```

Die Variablen `u_servo_x` und `u_servo_y` enthalten die Pulsbreite des PWM-Signals in μs . Für die Mittelstellung sind sie z.B. `1500`, was $1500 \mu s = 1,5 ms$ entspricht. Durch Einstellen einer Pulsbreite von 0 können die Servomotoren ausgeschaltet werden [23].

Für den Fall, dass keine Kugel oder keine Platte erkannt wird, fahren die Servos und die Platte mit den gleichen Funktionen

```
pi.set_servo_pulsewidth(ServoX, 1500)
pi.set_servo_pulsewidth(ServoY, 1500)
```

zurück in ihre Ausgangslage.

Beim Beenden des Programms fahren die Servomotoren ebenfalls in ihre Mittelstellung und werden kurz danach ausgeschaltet.

4.8. Performanceverbesserung

Das folgende Kapitel beschreibt, wie und warum die Performance durch Begrenzen der FPS und erhöhen der Prozesspriorität verbessert wird.

Nach Beendigung der ersten Programmierphase, in welcher der Raspberry Pi noch die Aufgaben der Bildverarbeitung und der Regelung übernommen hat, zeigen die ersten Tests kein zufriedenstellendes Ergebnis (Zittern der Servomotoren wie in Kapitel 4.7.3 beschrieben). Daraufhin werden die Berechnungen auf den Desktop-PC ausgelagert, was eine Verbesserung der Performance zur Folge hat, aber das Problem nicht löst. Weiterhin fährt die Balancierplatte in unregelmäßigen Abständen in unüblich steile Winkelpositionen, was das System stark aus dem Gleichgewicht bringt.

Eine daraufhin folgende Fehleranalyse zeigt verhältnismäßig starke Sprünge in der berechneten Geschwindigkeit, welche über das Regelgesetz (siehe Kapitel 3.2) direkten Einfluss auf die Stellgrößen hat. Es wird überprüft und anschließend ausgeschlossen, dass weder die Positionserfassung noch die perspektivische Transformation die Ursache ist, was eine genauere Untersuchung der Zeitstempel nahelegt. Mithilfe des FPS-Zählers von der Bibliothek imutils (siehe Kapitel 4.1.5) kann eine Korrelation zwischen den

Geschwindigkeitssprünge und dem Auftreten zeitlicher Schwankungen zwischen den einzelnen Frames nachgewiesen werden, dem sogenannten Jitter [9]²⁸.

Da die verwendete Webcam maximal 30 FPS liefern kann, wird anfangs die minimale Zykluszeit begrenzt, um doppelte Geschwindigkeitsberechnungen auf dem gleichen Bild zu vermeiden. Der Abstand, der durch die Zeitstempel gespeicherten Zeiten demnach ca. 33 ms betragen sollte, schwankte aber stark auf bis zu 60 ms wobei die Geschwindigkeitssprünge genau in den Bereichen starker FPS-Schwankungen auftreten.

Bei Betrachtung der Geschwindigkeitsberechnung (siehe Kapitel 4.6.1) sollten die beschriebenen FPS-Schwankungen keinen Einfluss auf die Geschwindigkeit haben. Die für die Berechnung benötigte Zeitdifferenz wird aus den Zeitstempeln der Bilder berechnet, was zur Folge hat, dass ein verzögert aufgenommenes Bild auch einen ebenso verzögerten Zeitstempel haben sollte. Die Verzögerung sollte demnach in der berechneten Geschwindigkeit nicht mehr auftauchen.

Dies legt die Vermutung nahe, dass im Bereich der FPS-Schwankungen eine erhöhte Verzögerung zwischen dem Zeitpunkt der Bilddurchsuche und dem Erstellen des Zeitstempels entsteht, obwohl dieser direkt nach Ankunft des Frames im Programmcode erstellt wird. Die auf den Bildern abgebildeten Ballpositionen und die zugehörigen Zeitstempel würden also in diesen Fällen nicht zusammenpassen (stärkere Abweichung als im Regelfall). Diese Abweichungen, auch Jitteramplituden genannt [9], sind abhängig von der Zugriffsrate auf die Kamera, welche wiederum direkt mit der Programmzykluszeit zusammenhängt. Durch verringern der Zugriffsrate auf 10 FPS Vdie Jitteramplitude vernachlässigbar gering.

Zu beachten ist hierbei, dass sich eine zu starke Begrenzung der FPS negativ auf die Regelung auswirkt, da der Istzustand des Systems zu selten erfasst wird. Die Begrenzung der Bildwiederholungsrate auf 10 FPS, also 100 ms pro Zyklus, hat sich für die

²⁸ Grafiken welche den Zusammenhang deutlicher zeigen, können aufgrund der unerwarteten Zugriffseinschränkung auf die Computer der DHBW leider nicht gezeigt werden.

Regelung als ausreichend herausgestellt. Die so erzielte Verbesserung der Performance ist im Unterschied zwischen Abbildung 38 und Abbildung 39 zu sehen²⁹.

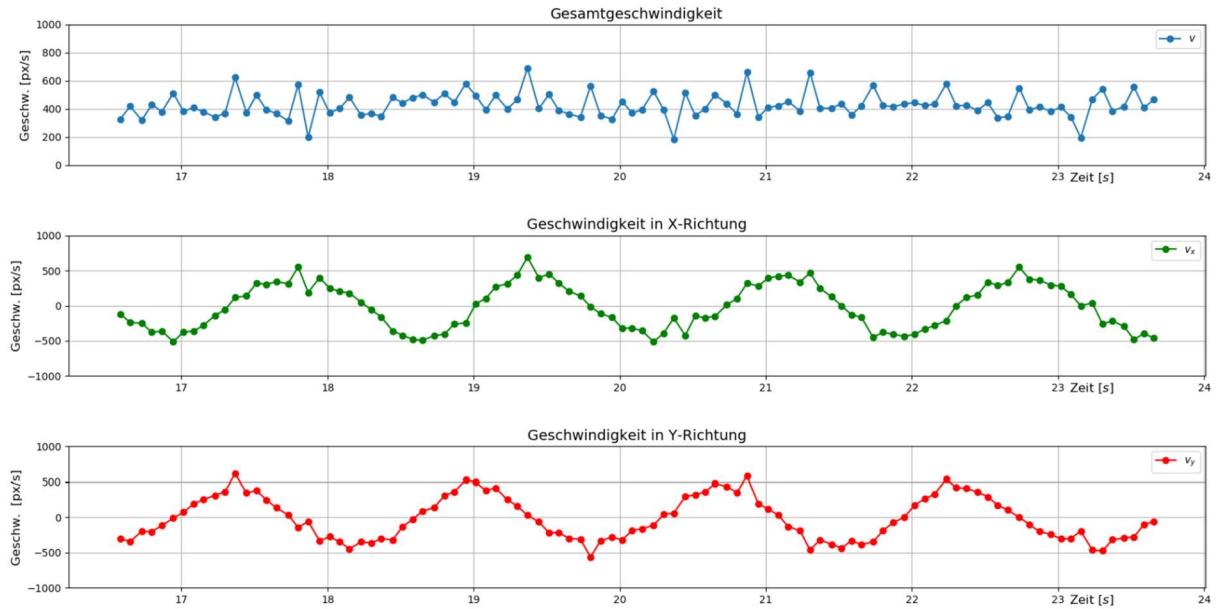


Abbildung 38: Berechnete Geschwindigkeiten bei 15 FPS und 0,6 Kreisumdrehungen pro Sekunde

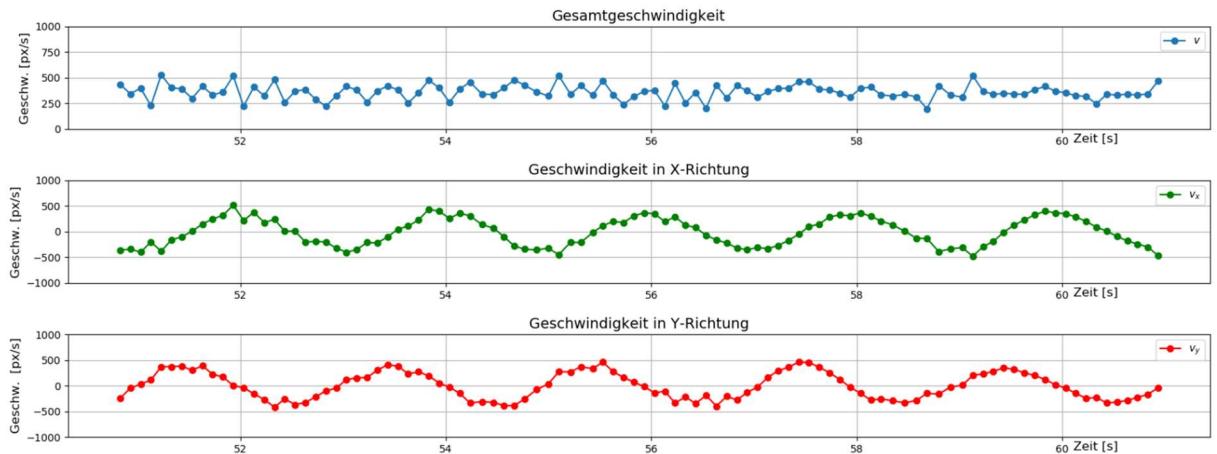


Abbildung 39: Berechnete Geschwindigkeiten bei 10 FPS und 0,5 Kreisumdrehungen pro Sekunde

Die hier blau dargestellte Gesamtgeschwindigkeit berechnet sich analog zur klassischen Physik mit der quadratischen Addition nach Formel (4.27).

²⁹ Weitere Grafiken und Bilder können leider aufgrund des eingeschränkten Zugriffs auf das Modell nicht abgebildet werden. Der Unterschied zwischen 30 FPS und 15 FPS ist wesentlich größer, die Visualisierungen hierzu aber leider nicht verfügbar.

$$v_{gesamt} = \sqrt{(v_x)^2 + (v_y)^2} \quad (4.27)$$

Die Änderung in relativer Höhe und in Varianz der Geschwindigkeitssprünge, sowie deren verringerte Häufigkeit, bestätigen die Korrelation zwischen Geschwindigkeitssprüngen und FPS und die damit verbundene Vermutung. Die Umsetzung erfolgt im Programmcode durch folgende Befehle zu Beginn des Programzyklus':

```
# Neuer Frame soll erst 0.1s nach dem letzten aufgenommen werden
# (fixe 10 FPS)
while time.time() - frame_time_last < 0.1:
    pass
```

Zur weiteren Performanceverbesserung wird die beschriebene Jitteramplitude durch Erhöhung der Prozesspriorität verringert. Dies hat zur Folge, dass weniger Interrupts anderer Prozesse den Programmablauf stören und zu Verzögerungen führen. Dieser Vorgang kann bei Windows über den Taskmanager manuell eingestellt werden. Um diesen manuellen Vorgang nicht bei jedem Programmstart erneut vornehmen zu müssen, wird dies durch die folgenden Befehle in der Programminitialisierung (siehe Kapitel 4.4) automatisiert.

```
process_id = win32api.GetCurrentProcessId()
handle = win32api.OpenProcess(win32con.PROCESS_ALL_ACCESS, True,
                             process_id)
win32process.SetPriorityClass(handle, win32process.REALTIME_PRIORITY_CLASS)
```

Die somit erzielten Ergebnisse der Performanceoptimierungen sind in Abbildung 40 zu sehen.

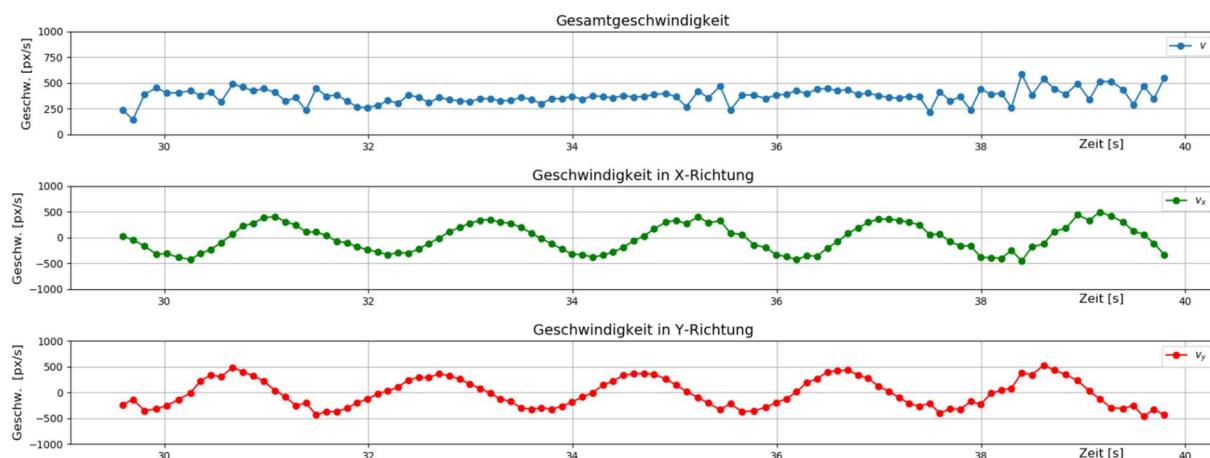


Abbildung 40: Berechnete Geschwindigkeiten bei 10 FPS und hoher Prozesspriorität

Im Vergleich zu der in Abbildung 39 zu sehenden berechneten Gesamtgeschwindigkeit (blau), werden mit der Prozesspriorisierung wesentlich geringere Schwankungen erzielt und die Regelung läuft sichtbar flüssiger.

5. Fazit und Ausblick

5.1. Fazit

Zusammenfassend kann diese Studienarbeit als ein großer Schritt in Richtung des fertigen Kugellabyrinths angesehen werden. In der Aufgabenstellung ist formuliert, dass die Kugel in eine stabile Lage gebracht werden und einer definierten Kreisbahn folgen soll. Dies konnte, hauptsächlich durch den kompletten Neuaufbau der Software, aber auch durch viele weitere Fortschritte, vollständig realisiert werden.

Zu Beginn wurde entschieden, den mechanischen Aufbau weitestgehend zu übernehmen und den Fokus dieser Arbeit auf die softwaretechnische Umsetzung zu legen. Trotzdem wurde der Raspberry Pi aufgrund seiner geringen Rechenleistung, zumindest für die Regelung und die Bildverarbeitung, durch einen leistungsstarken Desktop-PC ersetzt. Damit wurde die Grundlage für eine starke Performance-Verbesserung und eine schnelle Abarbeitung der Bildverarbeitungsalgorithmen geschaffen.

Die neu programmierte Software wurde bewusst in verschiedene Teile aufgeteilt, sowohl in der Dateistruktur als auch im Prozessablauf. Damit sind ein schneller Überblick und eine unkomplizierte Anpassung des Programmcodes möglich. Die in der Bildverarbeitung verwendeten Filter und Algorithmen ermöglichen eine stabile Erkennung von Ball und Platte. Dabei ist besonders die perspektivische Transformation zu erwähnen, die unabhängig von genauem Kamerablickwinkel eine zuverlässige Positionserkennung des Balls relativ zur Platte ermöglicht. Ein weiterer Aspekt ist die Bildaufnahme, die um die Möglichkeit der Auslagerung auf einen separaten Thread und das Hinzufügen eines Zeitstempels erweitert wurde. All dies wird in einer neuen Webcam-Klasse vereint, so dass die Webcam nach außen hin unverändert genutzt werden kann.

Die Regelung wurde ebenfalls von Grund auf überarbeitet. Mit Hilfe des Zustandsraummodells wurde ein gut funktionierender Regler berechnet, der auf Basis der zugrundeliegenden Bewegungsgleichung die Position der Kugel präzise regelt. Mit diesem Regler ist die Bewegung der Kugel auf einer Kreisbahn ohne Probleme möglich. Zudem

können die vorgegebene Bewegung und die Polstelle, welche maßgeblich für den Stabilitätsgrad des Systems verantwortlich ist, beliebig verändert werden. Die Berechnung der für den Regler benötigten Ballgeschwindigkeit erfolgt mit Hilfe der Zeitstempel aus der Bildaufnahme. An dieser Stelle werden die Bildverarbeitung und die Regelung kombiniert.

Eine weitere Verbesserung ist die Ansteuerung der Servomotoren. Durch die „pigpio“-Bibliothek kann auf dem Raspberry Pi ein fehlerfreies Hardware-PWM-Signal erzeugt werden, welches die Servomotoren ohne Taktzittern ansteuert. Zudem werden die GPIO-Pins des Raspberry Pi, über welche das PWM-Signal ausgegeben wird, von dem Desktop-PC aus ferngesteuert. Dies hat die Auslagerung der Berechnungen auf den PC deutlich vereinfacht. Um die aus der Regelung erzeugten Stellgrößen in das PWM-Signal umzurechnen, wird eine eigens ermittelte Approximationsfunktion verwendet, die sich im Praxisbetrieb bewährt hat.

Diese Studienarbeit beinhaltet viele weitere Verbesserungen und Umsetzungen gegenüber der vorangegangenen Arbeit, die den Erfolg dieser Arbeit ausmachen. Damit ist eine essentielle Grundlage für das Verfahren der Kugel in einem Kugellabyrinth geschaffen. Zudem bietet das in dieser Studienarbeit erstellte System ein fundamentales Grundgerüst, auf dem unkompliziert weitere Projekte aufgebaut werden können. Mögliche nächste Schritte und verschiedene Aspekte zur Erweiterung werden im Ausblick vorgestellt.

5.2. Ausblick

Mit Blick auf die abschließenden Ergebnisse der Studienarbeit sind die erwarteten Genauigkeitsanforderungen erfüllt. Die Kreisbahn, die sich nach dem Start des Programmes in nur wenigen Sekunden einstellt, ist stabil und gleichmäßig, unabhängig davon, ob der Ball anfangs ruht oder mit höherer Geschwindigkeit auf die Platte gerollt wird. Besonders im Bereich der Bildverarbeitung, bezogen auf die Detektion von Ball und Platte, werden alle Voraussetzungen für den Erfolg des Projekts Kugellabyrinth erfüllt.

Die Genauigkeitsanforderungen an die geforderte Kreisbahn sind zwar erfüllt, aber für die Regelung in einem Kugellabyrinth könnten die Anforderungen noch nicht ausreichen. Das Modell muss in der Lage sein, den Ball in kürzester Zeit präzise zu stoppen und in eine exakte Position zu verfahren. Um das zu realisieren, könnten die nächsten Schritte zum fertigen Kugellabyrinth einige der folgenden Punkte enthalten.

Beginnend mit einer allgemeinen Verbesserung: Für eine erfolgreiche Fehleranalyse und zur Evaluation von Ergebnissen wäre die Visualisierung der erzielten Kreisbahn und anderer Vorgänge der Regelung von großem Wert. Die Implementierung eines Live-Plots mithilfe von Matplotlib scheiterte aufgrund zu hoher Performanceeinbußen, da die von der Bibliothek zur Verfügung gestellten Methoden zu rechenaufwendig sind. Eine Alternative könnte hier die Bibliothek von PyQtGraph bieten. PyQtGraph stellt viele Möglichkeiten zum Anzeigen von Daten zur Verfügung und ist durch die Nutzung der Numpy- und PyQt-Frameworks besonders rechenarm und schnell. Dies ermöglicht das Anzeigen von Plots parallel zum Prozesszyklus, die keine Performance-Einbuße auf den Rest des Programms haben.

Des Weiteren kann die zugrundeliegende PyQt-Bibliothek verwendet werden, um eine Benutzeroberfläche zu programmieren. Als Vorbild hierfür dient das YouTube-Video der Hochschule Arnheim [3], welches zeigt, wie in der Benutzeroberfläche eine beliebige Trajektorie für die Soll-Position des Balls angegeben werden kann. Liveänderungen von Sollgeschwindigkeit oder Polstellen, sowie der beschriebene Live-Plot könnten hier implementiert werden.

Um die Regelung weiter zu verbessern, kann ein mehrschleifiger Zustandsregler eingesetzt werden. Bei diesem Regler werden mehrere Zustandsgrößen zurückgeführt, welche sich aus der Integration der jeweils vorherigen Zustandsgröße ergeben. [57]

Des Weiteren kann das sich im Anhang befindende mathematische Modell verfeinert und optimiert werden. Aktuell ist die Approximation zur Umrechnung der Plattenwinkel in Stellgröße der Servomotoren ausreichend, jedoch könnten in der Zukunft höhere Genauigkeitsanforderungen entstehen, bei denen ein optimiertes mathematisches Modell

bessere Ergebnisse als die Approximation liefert. Mit dem entsprechenden Zeitaufwand ließe sich ein verbessertes mathematisches Modell erstellen.

Die Ansteuerung der Balancierplatte durch die Servomotoren ist zwar für die Regelung einer Kreisbahn ausreichend, könnte aber, bedingt durch ein vorhandenes Spiel in den Lagern der Servomotoren, beim schnellen Anfahren von gezielten Punkten problematisch werden. Hier sollte untersucht werden, ob es sich um Verschleißerscheinungen handelt oder ob eventuell ein anderes Servomotor-Modell verwendet werden sollte.

Wie bereits beschrieben, reicht die Rechenleistung des Raspberry Pis für die Bildverarbeitung nicht aus, weshalb die Berechnungen auf einen Desktop-PC ausgelagert werden. Trotz der erhöhten Rechenleistung und schlanker Algorithmen hatte eine Bildrate von über 15 FPS starke Einbußen in der Gleichmäßigkeit der Regelung zur Folge. Dies hängt mit der verwendeten Kamera und der openCV-Schnittstelle zusammen. Auch wenn die Auflösung und die Bildwiederholungsrate des Webcam-Modells C930e ausreichen, sorgt die API für starke Jitteramplituden in den Abständen der gelieferten Bilder, was sich negativ auf die Regelung auswirkt. Da die Webcam für einen anderen Verwendungszweck gedacht ist, wäre hier beispielsweise eine Industriekamera die bessere Wahl.

Abhängig von der verwendeten Kamera muss auch die Beleuchtung betrachtet werden. Bei meist gleichen Lichtverhältnissen im Labor des Modells, war eine genauere Betrachtung der Lampe bisher nicht weiter notwendig, bei einem anderen Einsatzort oder einem anderen Kameramodell können sich ändernde Lichtverhältnisse allerdings starke Einflüsse auf die Farbsegmentierung und somit auf die Detektion haben.

Neben diesen, teilweise optionalen, Meilensteinen auf dem Weg zum fertigen Kugellabyrinth darf natürlich nicht das Labyrinth an sich vergessen werden. Denkbar wäre hier, neben fest verschraubten Barrieren, auch eine Realisierung bei welcher sich das Labyrinth anpassen lässt und gegebenenfalls mithilfe eines Pathfinder-Algorithmus die Trajektorie des Balls berechnet wird.

Literaturverzeichnis

- [1] M. Kurz, N. Ebert und A. Stadler, „Projekt Kugellabyrinth,“ DHBW Mannheim, Eppelheim, 2019.
- [2] TU Darmstadt, „Konstruktion eines Demonstrators für „Ball on Plate“,“ [Online]. Available: https://www.rtm.tu-darmstadt.de/media/rtm/rtm_mitarbeiterseiten/strubeljan/ps_ball_on_plate.pdf. [Zugriff am 20 02 2020].
- [3] Hochschule Arnheim, „Ball on plate, Balancing,“ 13 01 2009. [Online]. Available: https://www.youtube.com/watch?v=uERF6D37E_o. [Zugriff am 21 4 2020].
- [4] Logitech, „c930e-Webcam,“ 2020. [Online]. Available: <https://www.logitech.com/de-de/product/c930e-webcam>. [Zugriff am 17 04 2020].
- [5] SEW EURODRIVE, „Servomotoren,“ 2020. [Online]. Available: <https://www.sew-eurodrive.de/produkte/motoren/servomotoren-synchronmotoren.html>. [Zugriff am 17 04 2020].
- [6] J. Lunze, Regelungstechnik 1, 12 Hrsg., Heidelberger Platz 3, 14197 Berlin, Germany: Springer Vieweg, 2020.
- [7] Infotip.de, „Analoge Schaltungstechnik,“ 2017. [Online]. Available: <https://kompendium.infotip.de/servos.html>. [Zugriff am 16 04 2020].
- [8] HiTEC, „HS-5485HB Standard Karbonite Digital Sport Servo,“ 2020. [Online]. Available: <https://hitecrcd.com/products/servos/sport-servos/digital-sport-servos/hs-5485hb-standard-karbonite-digital-servo/product>. [Zugriff am 16 04 2020].
- [9] M. Meyer, Komunikationstechnik, Hausen b. Brugg: Vieweg, 2002.
- [10] Wikipedia, „Servo,“ 2019. [Online]. Available: <https://de.wikipedia.org/wiki/Servo>. [Zugriff am 16 04 2020].
- [11] P. Hüwe und S. Hüwe, IoT @ Home, München: Hanser, 2019.
- [12] R. Follmann, Das Raspberry 3 Kompendium, Kamp-Linfort: Springer Vierweg, 2018.
- [13] G. Guillen, Sensor Projects with Raspberry Pi, Ciudad de Mexico: Apress, 2019.
- [14] Wikipedia, „Raspberry Pi,“ 2020. [Online]. Available: https://de.wikipedia.org/wiki/Raspberry_Pi. [Zugriff am 17 04 2020].
- [15] P. D. K. Neusser, Zeitreihenanalyse in den Wirtschaftswissenschaften, Springer Verlag, 2006.
- [16] H. . Unbehauen, „Behandlung linearer kontinuierlicher Systeme im Zustandsraum,“ , 1997. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-322-94390-3_1. [Zugriff am 11 2 2020].

- [17] TU München, „Ferienkurs Theoretische Physik 1,“ [Online]. Available: https://www.ph.tum.de/academics/bsc/break/2012s/fk_PH0005_04_exercisesolution.pdf. [Zugriff am 27 02 2020].
- [18] Wikipedia, „Zustandsregelung,“ [Online]. Available: <https://de.wikipedia.org/wiki/Zustandsregelung>. [Zugriff am 25 04 2020].
- [19] jetbrains, „PyCharm,“ [Online]. Available: <https://www.jetbrains.com/de-de/pycharm/>. [Zugriff am 18 04 2020].
- [20] GitHub, „GitHub - features,“ [Online]. Available: <https://de.github.com/features.html>. [Zugriff am 18 04 2020].
- [21] E. Weitz, Konkrete Mathematik (nicht nur) für Informatiker, Hamburg: Springer Spektrum, 2018.
- [22] NumPy developers, „NumPy,“ 2020. [Online]. Available: <https://numpy.org/>. [Zugriff am 20 04 2020].
- [23] joan, „The pigpio library,“ 24 03 2020. [Online]. Available: <http://abyz.me.uk/rpi/pigpio/index.html>. [Zugriff am 14 04 2020].
- [24] B. Niederberger, „ActivateState Code >> Recipes,“ 02 06 2006. [Online]. Available: <http://code.activestate.com/recipes/496767-set-process-priority-in-windows/>. [Zugriff am 26 04 2020].
- [25] A. Erhardt, Einführung in die Digitale Bildverarbeitung, Springer Verlag, 2008.
- [26] B. Jähne, Digitale Bildverarbeitung, Springer Berlin Heidelberg, 2012.
- [27] openCV, „Getting Started with Videos; OpenCV-Python Tutorials 1 documentation,“ [Online]. Available: https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html. [Zugriff am 28 02 2020].
- [28] RealPython, „An Intro to Threading in Python – Real Python,“ [Online]. Available: <https://realpython.com/intro-to-python-threading/>. [Zugriff am 02 28 2020].
- [29] Wikipedia, „Bildrauschen,“ [Online]. Available: <https://de.wikipedia.org/wiki/Bildrauschen>. [Zugriff am 24 03 2020].
- [30] M. J. B. Wilhelm Burger, Digitale Bildverarbeitung, Springer Berlin Heidelberg, 2015.
- [31] D. H. Bergelt, „Filter in der Bildverarbeitung,“ [Online]. Available: <https://tuberlin.de/sites/default/files/media/institut-fuer-mechanik-und-fluiddynamik-15832/Lehre/lehrveranstaltungen/fluid/MT/filter.pdf>. [Zugriff am 2020 02 25].
- [32] J. A. V. B. Vladimir Chernov, „Integer-based accurate conversion between RGB and HSV color spaces,“ Elsevier, 2015.
- [33] A. R. Smith, „Color Gamut Transfrom Pairs,“ New York Institute of Technology.

- [34] School of Computer Science - The University of Auckland, „Lecture Notes: Morphological Image Processing,“ [Online]. Available: <https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm>. [Zugriff am 11 03 2020].
- [35] openCV, „OpenCV documentation: Structural Analysis and Shape Descriptors,“ [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. [Zugriff am 27 03 2020].
- [36] Wikipedia, „Satz von Green,“ [Online]. Available: https://de.wikipedia.org/wiki/Satz_von_Green. [Zugriff am 27 03 2020].
- [37] apnorton, „Stack Exchange Mathematics Blog: Green’s Theorem and Area of Polygons,“ 04 06 2014. [Online]. Available: <https://math.blogoverflow.com/2014/06/04/greens-theorem-and-area-of-polygons/>. [Zugriff am 28 03 2020].
- [38] Python Software Foundation, „Built-in Functions,“ [Online]. Available: <https://docs.python.org/3/library/functions.html#max>. [Zugriff am 29 03 2020].
- [39] S. X. Y. P. a. G. C. J. Ma, „A real-time parallel implementation of Douglas-Peucker polyline simplification algorithm on shared memory multi-core processor computers,“ in *International Conference on Computer Application and System Modeling (ICCASM 2010)*, Taiyuan, 2010.
- [40] J. D. a. W. P. R. Whyatt, „The Douglas-Peucker line simplification algorithm,“ Bulletin - Society of University Cartographers, Leicester, 1988.
- [41] J. D. W. M. Visvalingam, „The Douglas-Peucker Algorithm for Line Simplification: Re-evaluation through Visualization,“ in *Computer Graphics Forum* 9, 1990.
- [42] Mysid, „File: Douglas-Peucker animated.gif,“ 30 07 2012. [Online]. Available: https://commons.wikimedia.org/wiki/File:Douglas-Peucker_animated.gif. [Zugriff am 12 04 2020].
- [43] Wikipedia, „Konvexe Hülle,“ [Online]. Available: https://de.wikipedia.org/wiki/Konvexe_H%C3%BClle. [Zugriff am 13 04 2020].
- [44] Wikipedia, „Konvexe Menge,“ [Online]. Available: https://de.wikipedia.org/wiki/Konvexe_Menge. [Zugriff am 13 04 2020].
- [45] J. Sklansky, „Finding the convex hull of a simple polygon,“ North-Holland Publishing Company, 1982.
- [46] A. Rosebrock, „Ordering coordinates clockwise with Python and OpenCV,“ 21 03 2016. [Online]. Available: <https://www.pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/>. [Zugriff am 2020 04 24].
- [47] Uni Osnabrück, „Homogene Koordinaten,“ [Online]. Available: http://www-lehre.informatik.uni-osnabrueck.de/~cg/2000/skript/6_4_Homogene_Koordinaten.html. [Zugriff am 31 03 2020].

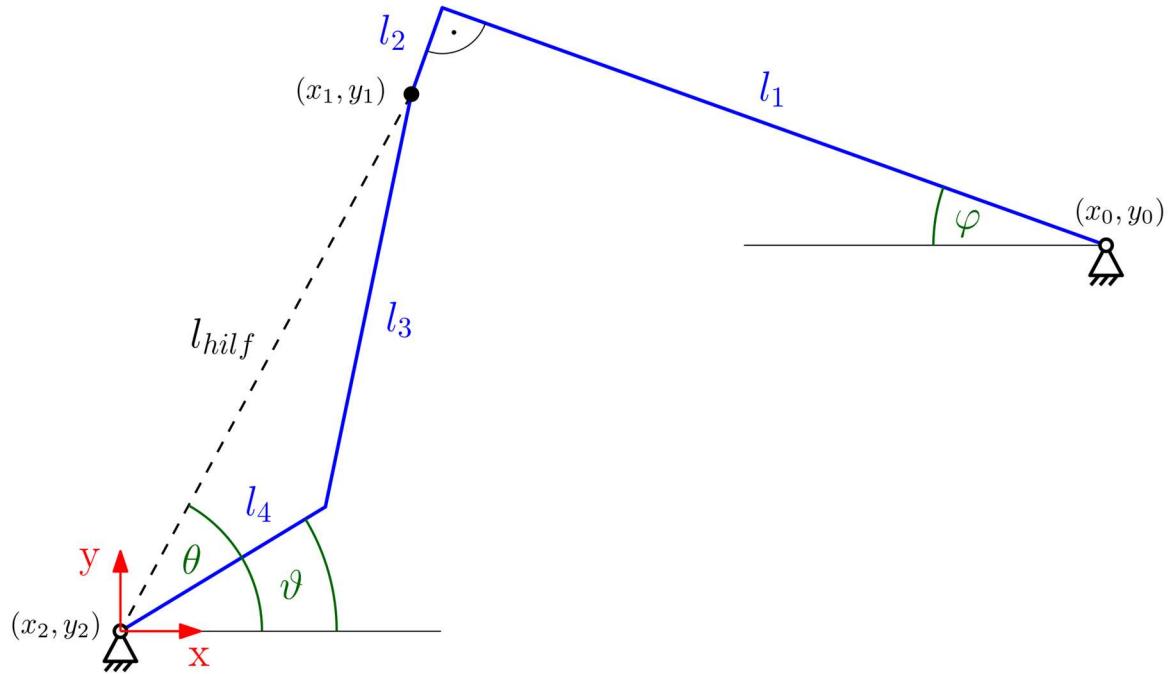
- [48] D. L. Josef Hoschenk, *Grundlagen der geometrischen Datenverarbeitung*, Darmstadt: B.G. Teubner Stuttgart, 1992.
- [49] F. Ta, „Computing CSS matrix3d transforms,“ 04 09 2014. [Online]. Available: <https://franklinta.com/2014/09/08/computing-css-matrix3d-transforms/>. [Zugriff am 02 04 2020].
- [50] E. Hausenblas und C. Brand, „7. Vorlesung Numerische Methoden,“ Leoben, 2006.
- [51] mikrocontroller.net, „Pulsweitenmodulation,“ [Online]. Available: <https://www.mikrocontroller.net/articles/Pulsweitenmodulation>. [Zugriff am 15 04 2020].
- [52] B. Croston, „RPi.GPIO 0.7.0,“ 21 07 2019. [Online]. Available: <https://pypi.org/project/RPi.GPIO/>. [Zugriff am 14 04 2020].
- [53] joan, „What's the difference between soft PWM and PWM,“ 14 07 2019. [Online]. Available: <https://raspberrypi.stackexchange.com/questions/100641/whats-the-difference-between-soft-pwm-and-pwm>. [Zugriff am 14 04 2020].
- [54] M. Booth, „Can I use the GPIO for pulse width modulation (PWM)?,“ 18 06 2018. [Online]. Available: <https://raspberrypi.stackexchange.com/questions/298/can-i-use-the-gpio-for-pulse-width-modulation-pwm>. [Zugriff am 14 04 2020].
- [55] Raspberry Pi Foundation, „GPIO,“ [Online]. Available: <https://www.raspberrypi.org/documentation/usage/gpio/>. [Zugriff am 16 04 2020].
- [56] joan, „pigpio python interface,“ [Online]. Available: <http://abyz.me.uk/rpi/pigpio/python.html>. [Zugriff am 16 04 2020].
- [57] W. Schneider, *Praktische Regelungstechnik*, Springer Berlin Heidelberg, 2008.
- [58] P. D. -I. J. Korthals, *Messen mit Optischen Systemen Teil 1: Bildverarbeitung*, Eppelheim, 2018.

Anhangsverzeichnis

ANHANG 1. MATHEMATISCHES MODELL	107
ANHANG 2. APPROXIMATION.....	111

Anhang 1. Mathematisches Modell

Umrechnung Winkel Abtriebshebel Servomotor – Winkel Platte



Gegeben:

Koordinaten Plattenmittelpunkt³⁰: (x_0, y_0)

Koordinaten Servomotor-Drehachse: $(x_2, y_2) = (0, 0)$

Abstand Plattenmittelpunkt zu Montagewinkel: l_1

Abstand Plattenoberfläche zu Drehgelenk 1: l_2

Länge Stützstange: l_3

Abstand Servomotor-Drehachse zu Drehgelenk 2: l_4

Vorgegebener Plattenwinkel: φ

Gesucht:

³⁰ Hier wird die Annahme getroffen, dass sich das Kugelgelenk, um welches sich die Platte dreht, in der Platte befindet und nicht wie im Modell unterhalb der Platte.

Winkel Abtriebshebel Servomotor: ϑ

Rechnung:

Hilfspunkt (x_1, y_1) in Abhängigkeit von (x_0, y_0) und φ :

$$x_1 = x_0 - l_1 * \cos(\varphi) - l_2 * \sin(\varphi) \quad (1)$$

$$y_1 = y_0 + l_1 * \sin(\varphi) - l_2 * \cos(\varphi) \quad (2)$$

Winkel θ zwischen l_{hilf} und y-Achse:

$$\theta = \arctan \left(\frac{y_1 - y_2}{x_1 - x_2} \right) \quad (3)$$

Länge l_{hilf} :

$$l_{hilf} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4)$$

Kosinussatz Dreieck $l_3 l_4 l_{hilf}$:

$$\begin{aligned} l_3^2 &= l_4^2 + l_{hilf}^2 - 2 * l_4 * l_{hilf} * \cos(\theta - \vartheta) \\ \cos(\theta - \vartheta) &= \frac{l_4^2 + l_{hilf}^2 - l_3^2}{2 * l_4 * l_{hilf}} \\ \vartheta &= \theta - \arccos \left(\frac{l_4^2 + l_{hilf}^2 - l_3^2}{2 * l_4 * l_{hilf}} \right) \end{aligned} \quad (5)$$

(3) in (5):

$$\vartheta = \arctan \left(\frac{y_1 - y_2}{x_1 - x_2} \right) - \arccos \left(\frac{l_4^2 + l_{hilf}^2 - l_3^2}{2 * l_4 * l_{hilf}} \right) \quad (6)$$

(4) in (6):

$$\vartheta = \arctan \left(\frac{y_1 - y_2}{x_1 - x_2} \right) - \arccos \left(\frac{l_4^2 - l_3^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2}{2 * l_4 * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \right) \quad (7)$$

Aus Gründen der Überschaubarkeit werden (1) und (2) nicht in (7) eingesetzt.

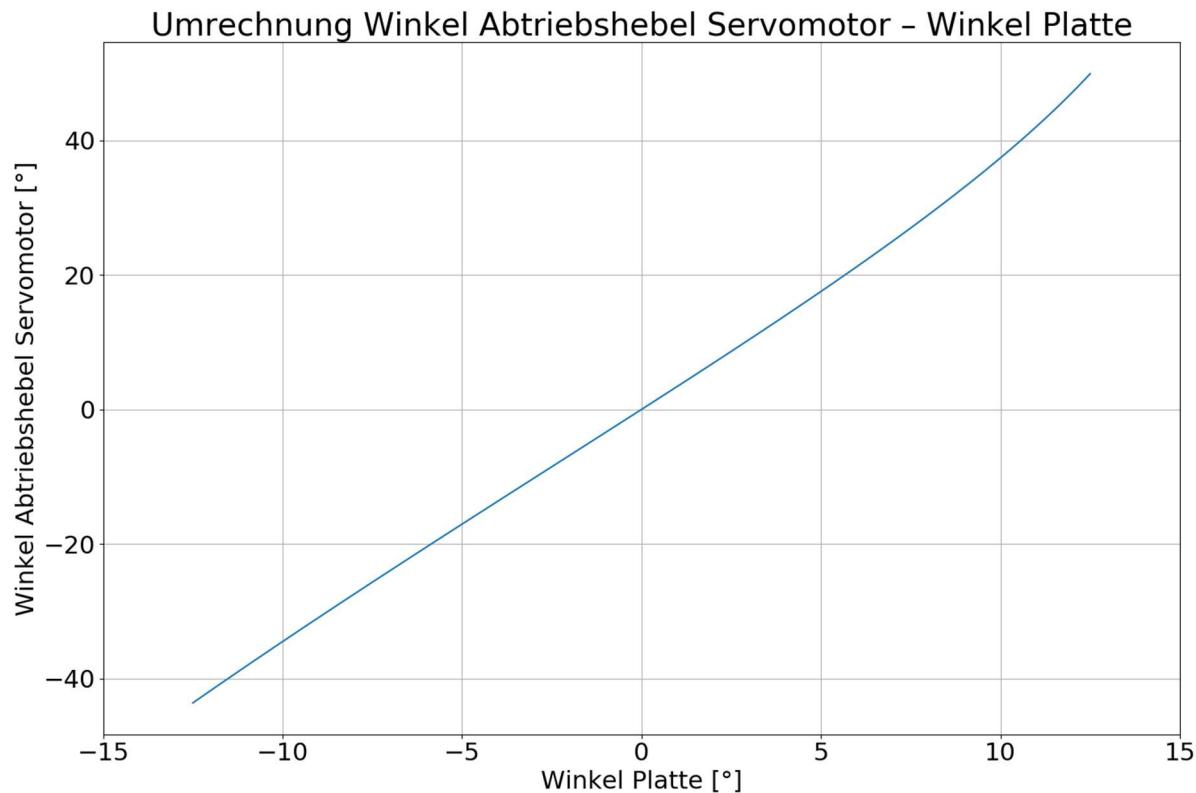
Diagramm

Abbildung 1 Zusammenhang zwischen Winkel Abtriebshebel Servomotor und Winkel Platte

Umrechnung PWM-Pulsdauer – Winkel Abtriebshebel Servomotor

Die Servomotoren werden nach der Anleitung aus der Dokumentation der Python-Bibliothek „Gpiozero“³¹ kalibriert. Dazu werden die Winkel der Servomotoren bei festgelegten Pulsdauern p gemessen. Damit können die Servomotoren über die Klasse „AngularServo“ von „Gpiozero“ durch Vorgabe des Winkels angesteuert werden.

	Servo X	Servo Y
Kennzeichnung am Motorgehäuse	1	2
min_angle ($p = 1 \text{ ms}$)	-40°	-41°
max_angle ($p = 2 \text{ ms}$)	+43°	+42°
Drehrichtung für positive Plattenwinkel	+	-

Tabelle 1 Gemessene Größen und Drehrichtung der beiden Servomotoren

³¹ Anleitung unter https://gpiozero.readthedocs.io/en/stable/api_output.html?highlight=angularservo#gpiozero.AngularServo verfügbar

Anhang 2. Approximation

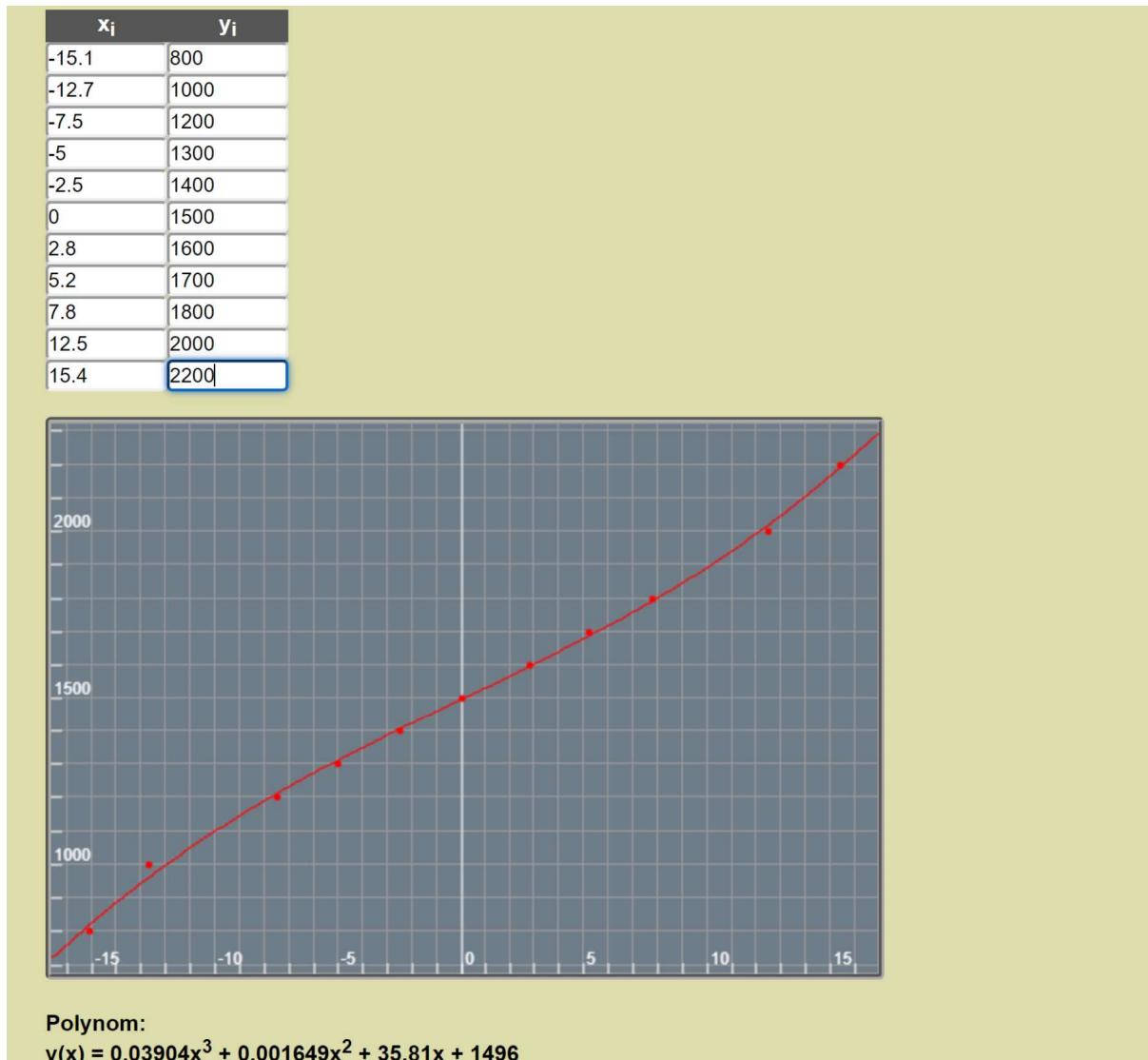


Abbildung 1 Approximationsfunktion ServoX

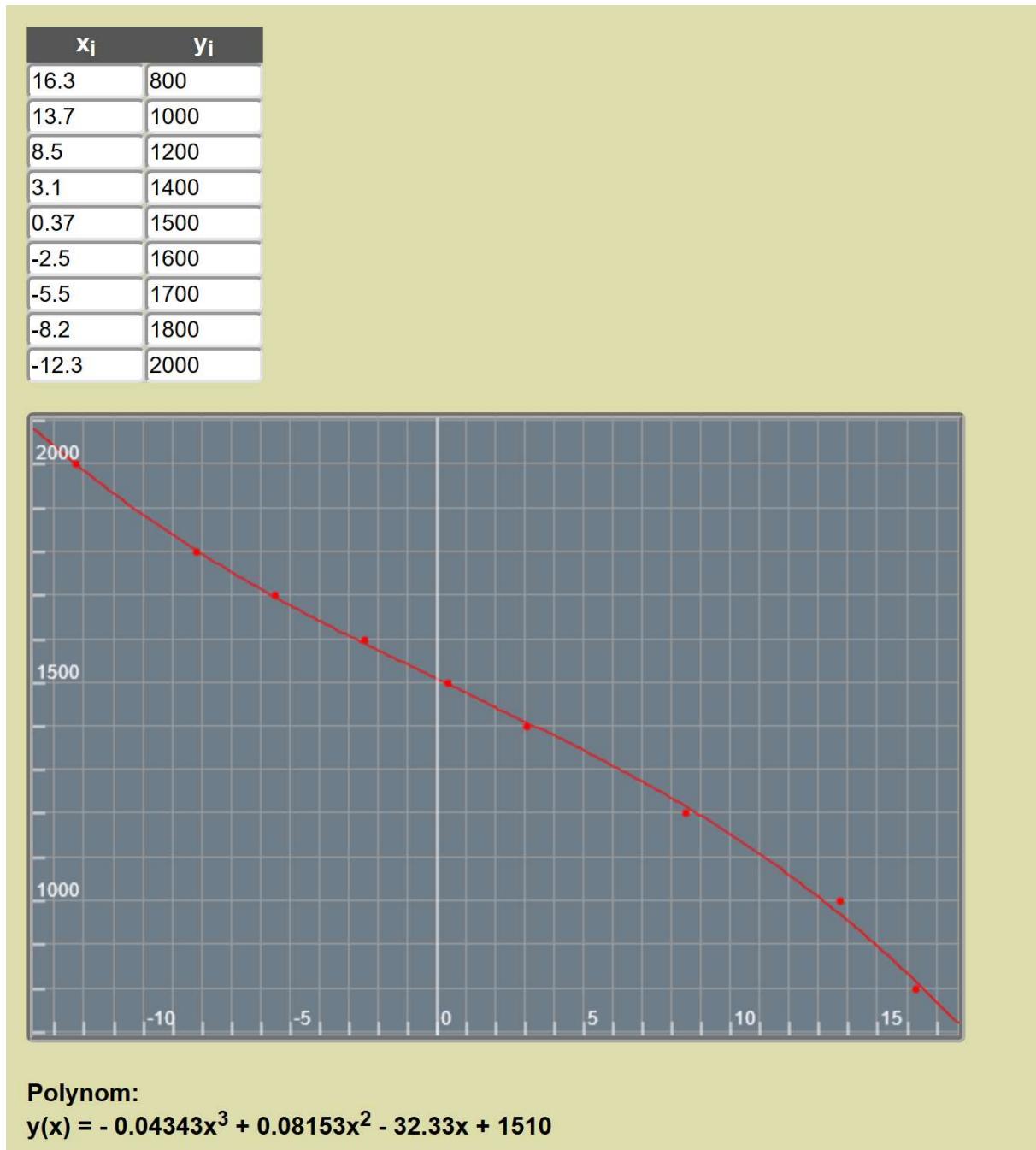


Abbildung 2 Approximationsfunktion ServoY