



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Aufbau und Inbetriebnahme eines neuen Versuchsstands "Ball on Plate" mit optischer Positionerkennung

Abschlussarbeit

zur Erlangung des akademischen Grades
Master of Engineering

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich 1: Ingenieurwissenschaften - Energie und Information
Studiengang Computer Engineering

Erstprüferin Prof. Dr.-Ing. Heide Brandstädter

Zweitprüfer Prof. Dr. rer. nat. Frank Bauernöppel

Eingereicht von: Oliver Koepp

Matrikelnummer: s0559122

Abgabe: 24.03.2022

Inhaltsverzeichnis

1	Einleitung	2
2	Grundlagen	3
2.1	Mechanik	3
2.2	Hardwareboards	5
2.3	Kamera	7
2.4	Motoren	8
2.5	Motortreiber	9
2.6	Bildverarbeitung	10
2.7	Beschleunigungs-/ Gyroskop-sensor	11
3	Lösungsansatz	12
3.1	Teilbereich - Mechanischer Aufbau	12
3.2	Teilbereich - Optische Positionsbestimmung	12
3.3	Teilbereich - Positionsregelung	13
3.4	Teilbereich - Motorsoftware	13
4	Realisierung	15
4.1	Raspberry Pi Kamera	15
4.2	Software Bildverarbeitung	15
4.3	Mechanischer Aufbau	17
4.4	Software Motorsteuerung	21
4.5	Regler	26
5	Messungen	28
5.1	Detektion des Tischtennisballs	28
5.2	Software-Bildwiederholfrequenz	32
5.3	Geschwindigkeit Bildverarbeitungssoftware	33
5.4	Kamerabildentzerrung	33
5.5	Einschwingverhalten der Acrylglasplatte	34
5.6	Genauigkeit der Bildverarbeitungssoftware	35
6	Mathematische Herleitungen	40
6.1	Moment	40
6.2	x und y -Koordinaten	40
6.3	Winkelberechnung	41
6.4	Hebelarm Übersetzung	43
6.5	Newton Verfahren	47
6.6	Modellierung des Systems	48

7	Probleme	50
7.1	Herstellungszeit der Mechanik	50
8	Verbesserungen	51
8.1	Mechanischer Aufbau	51
8.2	Optische Positionsbestimmung	51
8.3	Motorsoftware	51
9	Erreichter Stand	52
9.1	Pflichtenheft-Kriterien	52
9.2	Wunsch-Kriterien	53
10	Ausblick	54
10.1	Mechanischer Aufbau	54
10.2	Optische Positionsbestimmung	54
10.3	Positionsregelung	54
10.4	Motorsoftware	54
	Eidesstattliche Erklärung	55
	Literaturverzeichnis	56

Abbildungsverzeichnis

1	Prinzipbild des mechanischen Aufbaus	2
2	Zeitbedarf des Tischtennisballs bei verschiedenen Längen / Winkeländerungen	4
3	Strukturbild mit seinen vier Freiheitsgraden	5
4	Vorderseite des NUCLEO-L476RG Board	6
5	Radiale- /Tangentiale-Verzerrung [13]	8
6	Ermittlung der Kameraverzerrung	8
7	Teilbereiche des Gesamtsystems	12
8	Zollstock in 11 cm Entfernung zur Kamera	12
9	Untere & Obere Gate-Driver- An/ Aus Schwellspannung	14
10	Programmablaufplan	17
11	Motorhalterung gefertigt und als 3D-Modell	18
12	Kamerahalterung mit Raspberry Pi 3 Kamera	19
13	Teile des Hebelarms	20
14	Hebelarm Aufhängung Z4 und Z5	20
15	Simulation eines Hebelarms in einer Dimension	21
16	Skizze des Zusammenhangs zwischen Regler, Motorsoftware und Motortreiber	22
17	Strukturbilder der Motorsoftware	25
18	Verhalten der Acrylglasplatte beim Anfahren einer neuen Position	26
19	Modellierung des Systems in Simulink realisiert	27
20	Kamerabild mit dem entsprechenden 3D-Modell	28
21	<i>Cb</i> Kanal vom <i>YCrCb</i> Farbraum mit dem dazugehörigen Histogramm	29
22	Diffuse Lichtverhältnisse	29
23	Gerichtete Lichtverhältnisse	30
24	Blauer Ball in zwei Farträumen	31
25	Skizze des Versuchsaufbaus	32
26	Geschwindigkeit der Bildverarbeitungssoftware ohne Entzerrung	33
27	Kariertes Blatt auf der Acrylglasplatte	34
28	Einschwingverhalten bei verschiedenen Geschwindigkeiten	35
29	Genauigkeit der <i>x</i> , <i>y</i> und <i>z</i> -Position vor und nachdem die Fehler behoben worden sind	37
30	Genauigkeit der <i>x</i> , <i>y</i> und <i>z</i> -Position vor und nachdem die Fehler behoben worden sind	38

31	Genauigkeit der x , y und z -Position vor und nachdem die Fehler behoben worden sind	39
32	Simulinkmodell der Winkelberechnung in der y -Achse	42
33	Skizze der Winkelveränderung für Formel 6.9 [26]	43
34	Skizze der Übersetzung des Hebelarms	44
35	Messwertkurve von -180° bis 180°	46
36	Interessanter Funktionsabschnitt	47
37	Vergleich verschiedener Polynomfunktionen	48
38	Skizze der Modellierung	49
39	Anfahren verschiedener Winkelpositionen	53

Abkürzungsverzeichnis

MCU Micro-Controller Unit

FPS Frames Per Second

ARM Advanced RISC Machine

I^2C Inter-Integrated Circuit

SPI Serial Peripheral Interface

GPIO General-Purpose Input/Output

UART Universal Asynchronous Receiver Transmitter

SoC System on Chip

RTOS Real Time Operating System

OpenCV Open Computer Vision

CSI Camera Serial Interface

Vorwort

Bei der vorliegenden Masterarbeit bedanke ich mich bei der Firma *DIGALOG Industrie – Mikroelektronik GmbH* [1] mit Sitz in Berlin. Diese haben freundlicherweise Quellcode für die Motorsteuerung für diese Arbeit bereitgestellt, die in Teilen wiederverwendet worden sind. Es wird an dieser Stelle darauf hingewiesen, dass eine unerlaubte Verbreitung der Motorsoftware rechtliche Konsequenzen nach sich ziehen würde.

Ich wünsche Ihnen viel Freude beim Lesen dieser Masterarbeit.

1 Einleitung

Die vorliegende Arbeit hat das Ziel, einen Versuchsstand "Ball on Plate" aufzubauen. Dabei dient das Video [2] als Vorbild. Sie soll die Grundlage für Modellbasierte Regelung (Reglerentwurf) sein, wobei der Regler dabei eine untergeordnete Rolle spielt. Versuchsstände mit Ball on Plate gibt es viele – allerdings besitzen die Motorstellung und Positionsbestimmung dort eine zu große Unschärfe, um genaue Ergebnisse zu liefern. Ziel ist es, ein mobiles eingebettetes System für das Labor zu entwickeln – mit dem Fokus auf genaue Messwerte und somit auf eine geringe Fehlerfortpflanzung. Für die komplette Funktion muss dann eine geeignete Regelkreisarchitektur entworfen werden. Es sollte einfach erweiterbar sein, um der zukünftigen Entwicklung Schritt zu halten. Die Validierung erfolgt durch einen prototypischen Aufbau. Zurzeit ist geplant, für den prototypischen Aufbau des Systems ein Tischtennisball mit einer Acrylglasplatte zu verwenden – dies ist vorgegeben. Abbildung 1 zeigt ein Prinzipbild des mechanischen Versuchsaufbaus. Der Tischtennisball soll sich auf der x und y Achse bewegen können. Weiterhin soll eine Genauigkeit bei der Winkeländerung auf $0,5^\circ$ erfolgen – bei einem maximalen Zeitbedarf von 500 ms. Als Motoren kommen NEMA 23 Schrittmotoren zum Einsatz. Diese stehen zur Verfügung. Das Preisbudget ist auf 600 € festgelegt.

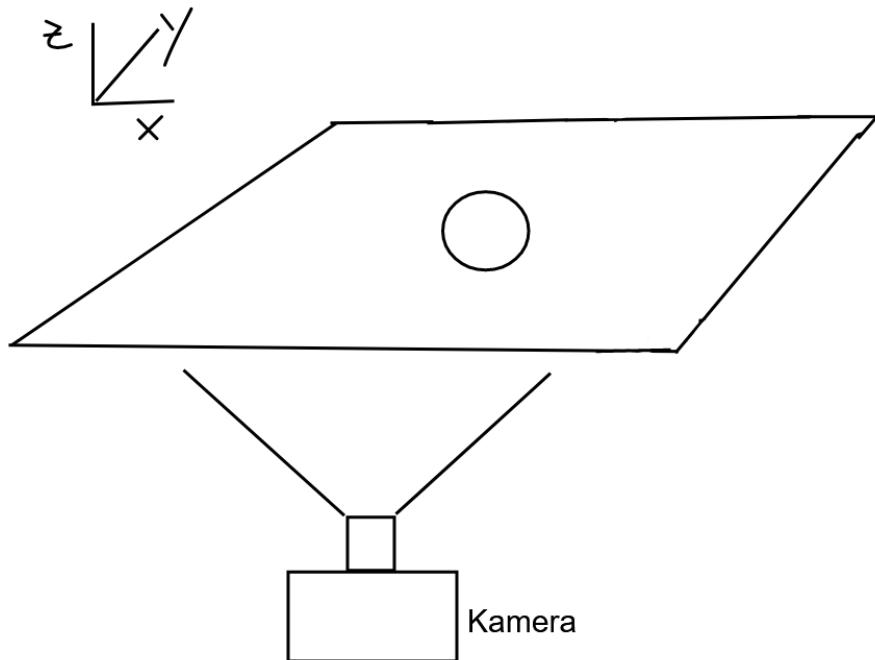


Abbildung 1: Prinzipbild des mechanischen Aufbaus

2 Grundlagen

2.1 Mechanik

Grundplatte

Die Größe der Grundplatte wird in Abhängigkeit von dem Tischtennisball ermittelt. Ein Tischtennisball hat einen Durchmesser von $d = 40 \text{ [mm]}$ und ein Gewicht von $g = 2,7 \text{ [g]}$. Bei einem maximalen Neigungswinkel der Platte, hat die Regelung nur begrenzt Zeit (Formel 2.1), bevor der Ball herunterfällt. Die Größe der Platte ist entscheidend, wie stark der Ball sich bewegen darf. Abbildung 2 visualisiert die Formel 2.1 bei verschiedenen Winkeln (α) und Längen der Acrylglasplatte für die Bewegung des Tischtennisballs. Je näher der Winkel $\alpha 0^\circ$ kommt, desto länger ist der Zeitbedarf des Tischtennisballs für die Länge s . Bei $\alpha = 0^\circ$ haben wir keine Bewegung des Tischtennisballs.

$$\begin{aligned}\alpha &= \text{Neigungswinkel} \\ m &= 2.7 \cdot 10^{-3} \text{ [kg]} \\ g &= 9.8 \left[\frac{m}{s^2} \right] = \text{Erdbeschleunigung} \\ s &= \text{Breite der Grundplatte}\end{aligned}$$

$$\begin{aligned}F_a &= m \cdot g \cdot \sin(\alpha) \\ F_a &= 2.7 \cdot 10^{-3} \text{ [kg]} \cdot 9.8 \left[\frac{m}{s^2} \right] \cdot \sin(\alpha)\end{aligned}\tag{2.1}$$

$$\begin{aligned}F_a &= m \cdot a \\ a &= \frac{F_a}{m}\end{aligned}$$

$$\begin{aligned}s &= \frac{1}{2} \cdot a \cdot t^2 \\ t &= \sqrt{\frac{2 \cdot s}{a}}\end{aligned}$$

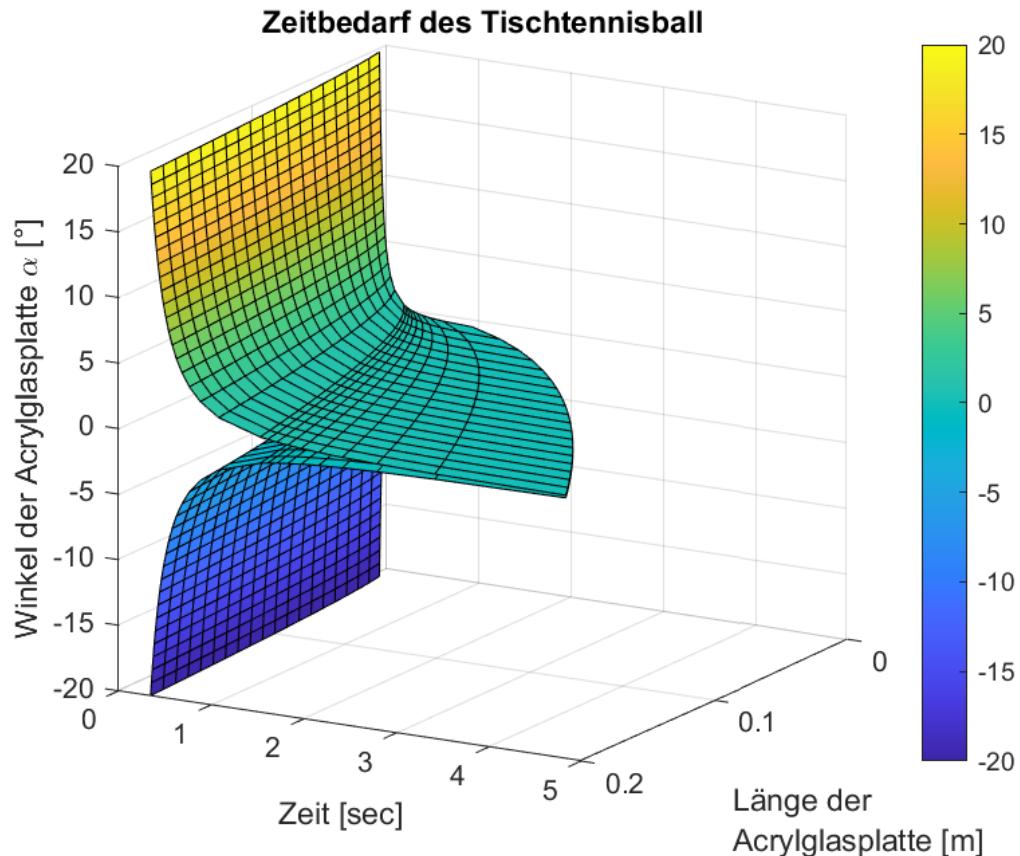


Abbildung 2: Zeitbedarf des Tischtennisballs bei verschiedenen Längen / Winkeländerungen

Hebelarme

Es können drei oder mehr Hebelarme verwendet werden. Für das Material kann Plastik oder ein anderes strukturstarkes Material verwendet werden. Darauf hinaus muss ein Hebelarm vier Freiheitsgrade unterstützen, wie aus der Strukturabbildung 3 entnommen werden kann. Der vierte Freiheitsgrad soll eine in das Bild hinein gerichtete Bewegung darstellen. Diese ist notwendig, damit die Grundplatte eine Pitch- und Rollbewegung nachgehen kann.

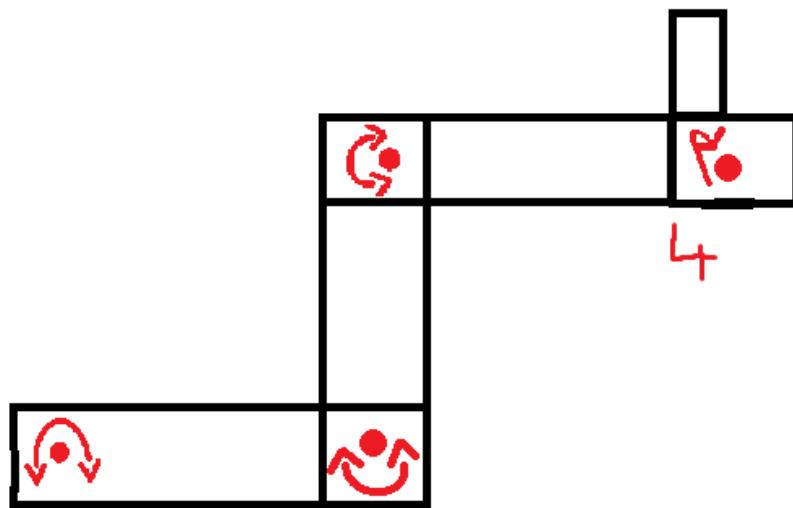


Abbildung 3: Strukturbild mit seinen vier Freiheitsgraden

Motor

Die Motoren müssen eine hohe Genauigkeit aufweisen um ihre Position mit einem hohen Drehmoment halten zu können, sodass die Hebelarme sich nicht absenken. Darüber hinaus ist eine schnelle Bewegung zur Zielposition notwendig, um eine hohe Dynamik aufweisen zu können.

2.2 Hardwareboards

Es gibt mehrere Controller, die zur Auswahl stehen. Dabei haben es die untenstehenden in die engere Auswahl geschafft.

NUCLEO-L476RG

Das von ST Microelectronics entwickelte Board "NUCLEO-L476RG" [3] besitzt einen ARM Cortex-M4 Prozessor. Für dieses Board gibt es eine Simulink Unterstützung. Mit den vorhandenen GPIO Schnittstellen kann das Board verschiedene Sensoren und Aktoren ansteuern. Eine eigene Kameraschnittstelle wird nicht von Hause aus unterstützt. Damit wird das NUCLEO-L476RG zu einem universell einsetzbaren Board, speziell geeignet für den Embedded Bereich. Die Abbildung 4 zeigt das NUCLEO-L476RG Board von der Vorderseite. Dieses Board besitzt kein eigenes Betriebssystem. Es wird ohne Betriebssystem ausgeliefert, allerdings können Betriebssysteme aufgespielt werden. Für die Programmierung dieses Boards wird die Entwicklungsumgebung Eclipse verwendet, in der ein STM Plugin installiert ist.

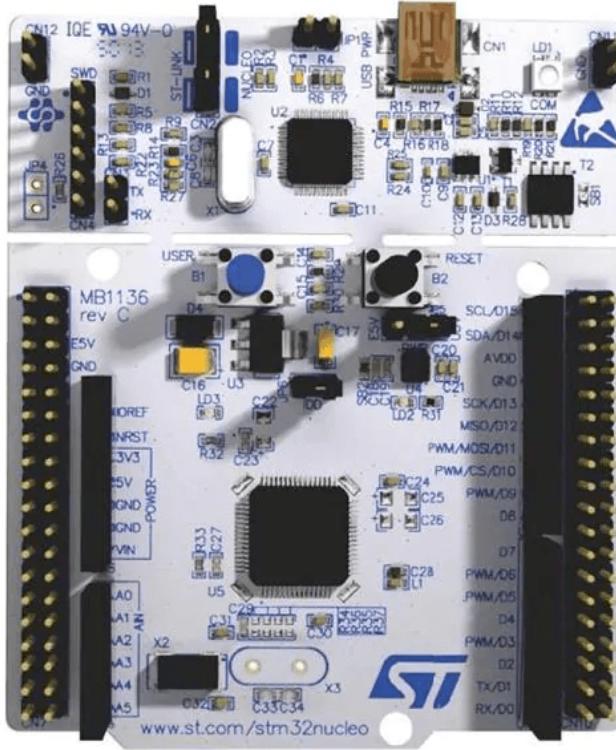


Abbildung 4: Vorderseite des NUCLEO-L476RG Board

Raspberry Pi 3

Ein Raspberry Pi 3 [4] ist ein Einplatinencomputer mit einer internen Grafikunterstützung. Dieser Controller ist weit verbreitet und bietet eine große Community. Weiterhin ist eine Grafikunterstützung von Hause aus integriert. Als Schnittstellen bietet der Raspberry Pi 3 GPIOs für Sensoren und Aktoren, vier USB 2.0, ein Ethernetanschluss und einen CSI-Anschluss für eine externe Kamera. Das Board wird üblicherweise mit dem Hersteller Betriebssystem *Raspberry Pi OS* bespielt.

NVIDIA Jetson TX1 Developer Kit

Das von NVIDIA entwickelte Embedded-Board [7], besitzt ein ARM Quad-Core und eine umfangreiche Grafikunterstützung auf Hardwareebene. Der Support wurde im Januar 2021 eingestellt. Es besitzt die Schnittstellen von einem Gigabit Ethernetport, USB 3.0, GPIOs, und einen Kamera-Expansion-Header. Allerdings befindet sich darunter keine CSI-Schnittstelle für eine Kamera. Das Board wird üblicherweise mit dem Hersteller Betriebssystem, welches ein Ableger des Ubuntu OS ist, bespielt.

2.3 Kamera

Die Kamera benötigt eine hohe Framerate; allerdings spielt die Auflösung eine untergeordnete Rolle. Eine CSI-Schnittstelle ist von Vorteil, da diese einen hohen Durchsatz ermöglicht. Eine höhere Auflösung ist besser, allerdings schränkt das die Framerate ein. Da die Framerate entscheidender für das Projekt ist, ist die Auflösung zweitrangig. Da jede Kamera aufgrund ihres Aufbaus eine Kameraverzerrung produziert, ist eine Kamera mit einer integrierten Entzerrung vorteilhafter. Eine Kameraverzerrung ist eine geometrische Verzerrung des Bildes aufgrund der Form der Linse, welches dazu führt, dass Objekte eine Radiale (tonnenförmig) oder Tangentiale (kissenförmig) Verzerrung bekommen (Abbildung 5). Um zu ermitteln, ob bei der eingesetzten Kamera eine starke Verzerrung vorliegt, ist ein Foto (Abbildung 6) mit zwei Tischtennisbällen vor einer weißen Wand fotografiert worden. Dabei sitzt der erste Tischtennisball im Zentrum der Abbildung und der zweite Tischtennisball in der Ecke rechts unten. Danach wurde der Umfang von dem im Zentrum liegenden Ball eingezeichnet und mit dem Umfang des Balls in der Ecke verglichen. In Abbildung 6b erkennt man, dass der eingezeichnete Umfang von dem Tischtennisball in der Ecke abweicht. Da diese Abweichung einen Einfluss auf die Fläche des Tischtennisballs hat, sollte auf eine Entzerrung nicht verzichtet werden. Die Kamera muss den Tischtennisball tracken. Für die x, y Position kann die OpenCV Funktion *moments()* verwendet werden (Kapitel 6.1). Die Berechnung für die z Koordinate wird in Gleichung 2.2 erläutert. Dabei werden Beispielwerte angenommen.

$$A_{\text{Ball liegt auf der Platte}} = 1000 \text{ [px]}$$

$$h_{\text{Ball liegt auf der Platte}} = 100 \text{ [mm]}$$

$$A_{\text{Ball in der Luft}} = 500 \text{ [px]}$$

$$h_{\text{Ball in der Luft}} = ?$$

$$\begin{aligned} r_{A_{\text{Ball liegt auf der Platte}}} &= \sqrt{A_{\text{Ball liegt auf der Platte}} \text{ [px]}} \\ r_{A_{\text{Ball liegt auf der Platte}}} &= \sqrt{1000 \text{ [px]}} = 31.6228 \text{ [\sqrt{px}]} \\ r_{A_{\text{Ball in der Luft}}} &= \sqrt{500 \text{ [px]}} = 22.3607 \text{ [\sqrt{px}]} \\ \text{verhaeltnis} &= \frac{r_{A_{\text{Ball in der Luft}}}}{r_{A_{\text{Ball liegt auf der Platte}}}} \\ \text{verhaeltnis} &= \frac{22.3607 \text{ [\sqrt{px}]}}{31.6228 \text{ [\sqrt{px}]}} = 0.7071 \\ h_{\text{Ball in der Luft}} &= \frac{h_{\text{Ball liegt auf der Platte}}}{\text{verhaeltnis}} \\ h_{\text{Ball in der Luft}} &= \frac{100 \text{ [mm]}}{0.7071} = 141.4214 \text{ [mm]} \end{aligned} \tag{2.2}$$

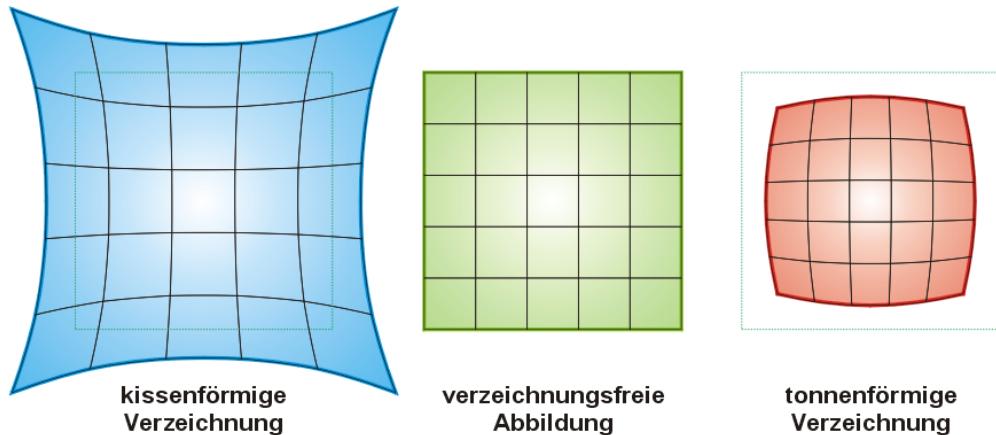


Abbildung 5: Radiale- /Tangentiale-Verzerrung [13]

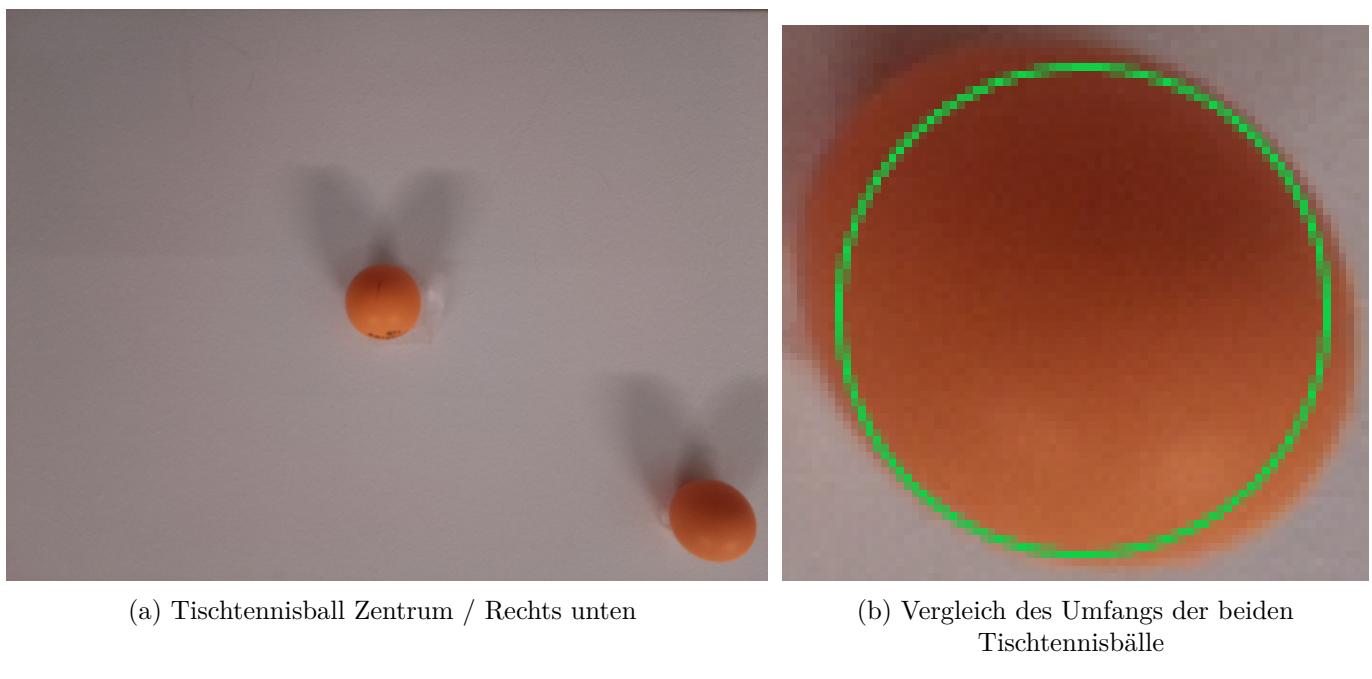


Abbildung 6: Ermittlung der Kameraverzerrung

2.4 Motoren

Als Motor kommen bipolare NEMA 23 Schrittmotoren zum Einsatz. Diese haben ein hohes Drehmoment und gleichzeitig ein Minimum an Leistungsaufnahme. Der Phasenwiderstand beträgt $1,2 \Omega$. Aus dem Datenblatt [8] kann entnommen werden, dass eine Phase maximal mit $2 A$ dauerhaft betrieben werden kann, bei einer Spannung von $2,4 V$. Der Schrittmotor kann auch mit höheren Spannungen betrieben werden (Zitat 2.4), obwohl im Datenblatt steht, dass eine Spannung von $2,4 V$ empfohlen wird.

However, as soon as the motor starts moving the combination of the inductance of the coils and the back-emf generated by the movement will prevent the nominal voltage from producing the rated current.

For this reason stepper motors are normally driven with a much higher voltage.[9]

Der Schrittmotor kann mit einer konstanten Geschwindigkeit betrieben werden, oder es wird eine Rampenfahrt für das Losfahren und Stoppen realisiert. Weiterhin sind diese im Labor schon öfters zum Einsatz gekommen und stehen bereits zur Verfügung. Um die NEMA 23 Schrittmotoren anzusteuern wird ein zusätzlicher Treiberbaustein benötigt.

Software für die Positionsregelung

Die kommerzielle Software Matlab ist vorrangig eine Mathematiksoftware, speziell für numerische Mathematik. Daneben gibt es noch Plugins, die Matlab mit verschiedenen Funktionen erweitern. Dazu zählt das Plugin Simulink. Simulink ist speziell für die Modellierung und Simulation eines Systems entwickelt worden. Darüber hinaus ist diese Software im Labor bereits im Einsatz.

2.5 Motortreiber

EVAL6480H

Das Evaluationsboard EVAL6480H [14] besitzt den Schrittmotorcontroller *L6480*, welches im industriellen Umfeld genutzt wird. Es unterstützt bis zu 256 Mikroschritte. Die zweifache H-Brücke, die für die Ansteuerung notwendig ist, liegt außerhalb des Controllers. Dadurch kann der NEMA 23 mit höheren Strömen betrieben werden, ohne dass der Controller wegen Überhitzung abschaltet. Mit diesem Baustein wurden bereits Erfahrungen im Rahmen meines Bachelor Praktikums gesammelt.

DRV8825

Das *DRV8825* Stepper Motor Driver Carrier Breakoutboard [15] besitzt den Treiberbaustein *DRV8825*. Dieser unterstützt bei zusätzlicher Kühlung einen maximalen Phasenstrom von 2,2 A. Es unterstützt eine Auflösung von bis zu 32 Mikroschritten und kann mit einer maximalen Schrittzahl von 250 kHz betrieben werden. Darüber hinaus ergaben Recherchen, dass dieser Motortreiber im Arduinoumfeld weit verbreitet ist.

DM542T

Der digitale Schrittmotortreiber von Stepperonline [10] unterstützt Ströme bis zu 4,2 A bei einer Spannung von 50 V. Eine Mikroschrittauflösung von 25600 Schritten pro Umdrehung, kann manuell eingestellt werden. Der Motortreiber besitzt drei Anschlüsse für den Controller – diese können jeweils entweder *LOW* für *AUS* und *HIGH* für *AN*, geschaltet werden:

den *ENABLE*-Pin, womit das Haltemoment aktiviert oder deaktiviert werden kann, den *PULSE*-Pin, der für die Schritte verantwortlich ist – liegt an diesen Pin eine steigende Flanke an, so macht der Motor ein Schritt, dabei sind die Timingbeschränkungen, die sich im Datenblatt [23] befinden zu beachten und zuletzt gibt es noch den *DIR*-Pin, der die Drehrichtung angibt. Der Schrittmotor wird über zwei Phasen automatisch vom Treiber angesteuert. Erste Tests haben gezeigt, dass der Treiber den Anforderungen gerecht wird.

2.6 Bildverarbeitung

Die Bildverarbeitung muss folgende Aufgaben erfüllen.

- Kameraentzerrung
- Erkennung des Tischtennisballs
- x und y Position bestimmen

Klassische Bildverarbeitung

Nach eingehender Recherche kann die Bildverarbeitungsbibliothek OpenCV verwendet werden. OpenCV ist eine Bildverarbeitungsbibliothek, die in C/C++ geschrieben ist. OpenCV eignet sich gut, da diese im Studium gelehrt wurde, eine weite industrielle Verbreitung hat und die Möglichkeit besteht, Algorithmen auf die Grafikkarte auszulagern. Darüber hinaus unterliegt die Bibliothek der Apache 2 Lizenz [5]. Für die Aufgabe der Kameraentzerrung bietet OpenCV die Funktion *undistort()* an. Diese benötigt nur die Kameramatrix, sowie die Verzerrungskoeffizienten, die vorher bestimmt werden müssen. Damit kann dann, das entzerrte Bild berechnet werden. Für die Erkennung des Tischtennisballs stehen mehrere Funktionen zur Verfügung. Zu einem die *cvtColor()*, *moments()* oder die *SimpleBlobDetector()* Funktion. Auch für die Extrahierung der x und y Positionen bietet OpenCV vorgefertigte Funktionen an. Alle diese Funktionen sind eine einfache Möglichkeit, um Kamerabilder schnell und effizient zu verarbeiten. Darüber hinaus bieten einige Funktionen ein Tuning für spezielle Algorithmen an. Daneben gibt es noch die Möglichkeit, die Aufgaben selber zu programmieren. Das macht den Code sehr effektiv. Allerdings besteht kein Problem mit dem Speicher und der Rechenleistung. Im Gegensatz dazu reichte der Zeitbedarf hierfür nicht aus, um die Aufgaben per Hand zu programmieren.

Neuronales Netz

Ein neuronales Netz ermöglicht es, die Aufgabe der Bildverarbeitung mithilfe von Mustererkennung zu erfüllen. Bevor ein neuronales Netz auf eine Problemstellung angewendet werden kann, muss es zuerst mit Trainingsdaten, die zuvor einem Labeling unterzogen worden sind, trainiert werden. Dies benötigt einen gewissen Zeitbedarf. Weiterhin ist das Anlernen von unterschiedlichen Bällen mit verschiedenen Farben möglich.

2.7 Beschleunigungs-/ Gyroskop-sensor

Als Beschleunigungs-/ Gyroskop-sensor kann der Sensor *MPU6050* [16] verwendet werden. Dieser wurde für den LowPower und High Performance Bereich entwickelt. Er besitzt eine *I2C* Schnittstelle zur Ansteuerung mit Fast Mode (400 kHz). Der Sensor ist ein beliebter Beschleunigungs-/ Gyroskop-sensor im Arduino Umfeld. Weiterhin kam dieser im Studium schon öfters zum Einsatz und es wurden erste Berührungs punkte gesammelt. Darüber hinaus, wird er für die Winkelberechnung im Regelungstechniklabor verwendet und hat sich als zuverlässig hervorgetan.

3 Lösungsansatz

Die Struktur des Systems ist in Abbildung 7 zu erkennen.

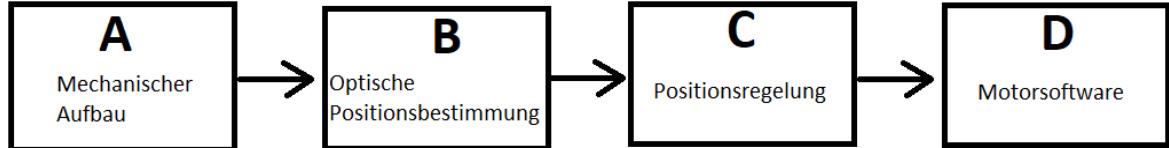


Abbildung 7: Teilbereiche des Gesamtsystems

3.1 Teilbereich - Mechanischer Aufbau

Die Größe der Grundplatte wird auf $20x20\text{ cm}$ bei einem maximalen Neigungswinkel von $\pm 20^\circ$ festgelegt. Bei der Größe wurde sich an dem Video [2] orientiert. Das Material für den mechanischen Aufbau ist Aluminium, da Plastik bei hohen Belastungen zu Verformungen führt. Darüber hinaus kann der mechanische Aufbau in der hauseigenen Zentralwerkstatt [6] hergestellt werden. Es werden vier Hebelarme eingesetzt, weil dies die Positionsregelung deutlich vereinfacht. Für den mechanischen Aufbau kommen haushaltsübliche Schrauben und Muttern zum Einsatz. Alle Abmessungen der mechanischen Teile sind im Anhang dargelegt. Alle Bauteile sind in *Fusion 360* angefertigt worden. Bei 11 cm Entfernung hat die Kamera eine Bildaufnahmegröße von $12.5 \times 8.5\text{ cm}$. Dies lässt sich anhand der Abbildung 8 erkennen.



(a) Zollstock in x Richtung

(b) Zollstock in y Richtung

Abbildung 8: Zollstock in 11 cm Entfernung zur Kamera

3.2 Teilbereich - Optische Positionsbestimmung

Für diese Arbeit wird die Raspberry Pi Kamera verwendet. Sie bietet eine hohe Framerate und hat eine CSI Schnittstelle. Des Weiteren kam sie im Fachbereich schon öfters zum Einsatz und

es konnten damit auch schon eigene positive Erfahrungen gesammelt werden. Als Controllerboard für die Bildverarbeitung wurde der Raspberry Pi 3 ausgewählt. Das NUCLEO-L476RG bietet mit Hilfe der OpenCV Bibliothek keine ausreichende Performance auf dem Gebiet der Bildverarbeitung [17], [18]. Das NVIDIA Jetson TX1 Developer Kit wird nicht mehr aktiv unterstützt und bietet keine Schnittstelle für die Raspberry Pi 3 Kamera. Der Raspberry Pi 3 wird aktiv weiterentwickelt und unterstützt die Kamera. Aufgrund mangelnder Erfahrung im Bereich "Neuronale Netze" und dem festen Zeitkontingent, kommt dieser Lösungsansatz für die Problemstellung nicht infrage. Stattdessen wird die klassische Bildverarbeitung verwendet. Bei der Bildverarbeitungsbibliothek OpenCV kommt die Version 3.5.1 zum Einsatz. Erste Messungen (5.1) haben ergeben, dass die Konvertierung des Kamerabildes nach $YCbCr$ gute Ergebnisse für einen blauen Tischtennisball liefert. Auf verschiedene Lichtverhältnisse wird an dieser Stelle nicht explizit eingegangen – genauere Angaben dazu befinden sich in Kapitel 5.

3.3 Teilbereich - Positionsregelung

Der Regler wird in Simulink gebaut. Dieses Plugin für Matlab ist genau für diesen Zweck gedacht. Weiterhin wird es im Regelungstechnik Labor gelehrt. Für diese Arbeit ist die Regelung in der z -Achse nicht berücksichtigt worden. Der Regler betrachtet nur die Achsen x und y .

3.4 Teilbereich - Motorsoftware

Als Treiberbaustein für den Schrittmotor kamen mehrere Controller zum Einsatz. Bei der Ansteuerung des EVAL6480H Board ist das Problem aufgetreten, dass der Controller immer in den *Undervoltage Lockout(UVLO)* Schutz [19] geht, obwohl nach persönlicher Einschätzung die Voraussetzung dafür nicht erfüllt ist. Das *UVLO* Bit wird vom Controller gesetzt, wenn die untere oder obere Gate-Driver-Versorgungsspannung unterschritten wird. Das *UVLOVAL* Bit ist auf 0 gesetzt worden; somit liegt die untere Gate-Driver-Versorgungsspannung bei 6,3 V und die obere bei 5,5 V (Abbildung 9). Eine Messung der unteren Gate-Driver-Versorgungsspannung ergibt $VCC_{thOff} = 7,5$ V. Die obere Gate-Driver-Versorgungsspannung kann über Gleichung 3.3 [20] berechnet werden. Beide Gate-Driver Spannungen liegen über der Anschaltschwelle, weswegen das *UVLO* Bit nicht automatisch auf 0 gesetzt werden dürfte. In dem Community-Forum von STMicroelectronics ist genau zu diesem Thema von dem Prüfling eine Frage gestellt worden [21].

$$\begin{aligned} V_{boot} - V_s &= \Delta VBOOT_{thOff} \\ 18V - 12V &= 6V \end{aligned} \tag{3.3}$$

Table 9. UVLO thresholds

Parameter	UVLOVAL	
	0	1
Low-side gate driver supply turn-off threshold ($V_{CC\text{thoff}}$)	6.3 V	10 V
Low-side gate driver supply turn-on threshold ($V_{CC\text{thon}}$)	6.9 V	10.4 V
High-side gate driver supply turn-off threshold ($\Delta V_{BOOT\text{thoff}}$)	5.5 V	8.8 V
High-side gate driver supply turn-on threshold ($\Delta V_{BOOT\text{thon}}$)	6 V	9.2 V

Abbildung 9: Untere & Obere Gate-Driver- An/ Aus Schwellspannung

Aus den oben genannten Gründen kommt das EVAL6480H nicht zum Einsatz. Der DRV8825 Controller schaltet sich unplanmäßig ab. Es ist somit keine kontrollierte Steuerung möglich. Mit dem DM542T sind diese Probleme nicht aufgetreten. Darüber hinaus besitzt er eine einfache Ansteuerung, womit der DM542T für die Arbeit bestens geeignet ist. Aufgrund dieser Information wird der DM542T für die Motoransteuerung verwendet. Für einen guten Regler wird weiterhin die Neigung der Acrylglasplatte benötigt. Dies wird mit dem Beschleunigungs-/ Gyroskop-sensor *MPU6050* gelöst. Die Motorsoftware wird auf dem NUCLEO-L476RG ausgeführt. Die Ansteuerung der Schrittmotoren mit einer konstanten Geschwindigkeit, erzeugt eine ruckartige Bewegung, die den Hebelarm in ungewollte Schwingungen versetzt. Eine Trapezfahrt dagegen, kann verwendet werden, allerdings ist der Programmieranteil dabei höher, als bei der Verwendung einer Dreiecksfahrt. Eine Dreiecksfahrt produziert keine ungewollten Schwingungen und ist einfach zu realisieren, weswegen für die Ansteuerung der Motoren eine Dreieckfahrt zum Einsatz kommt.

4 Realisierung

Dieses Kapitel befasst sich mit der konkreten Realisierung des Gesamtsystems. Es werden die Teilbereiche (Abbildung 7), die Herangehensweise beschrieben und die Gründe erläutert, wieso bestimmte Entscheidungen getroffen wurden. Das System hat folgende Soll-Kriterien.

- Das System soll innerhalb von $500\ ms$ eine neue Winkelposition angefahren haben
- Das System soll zwei Achsen parallel anfahren
- Das System soll 0.5° Toleranz maximal für eine Winkelposition aufweisen
- Das System soll die drei Koordinaten des Tischtennisballs ausgeben

Die Wunschkriterien des Systems sind:

- Das System sollte eine Toleranz der Detektion des Tischtennisballs von $5\ mm$ aufweisen
- Das System sollte eine Bewegung in der z -Achse ermöglichen

4.1 Raspberry Pi Kamera

Die Kamerabilder stehen dem Linux Betriebssystem des Raspberry Pis über die Datei `/dev/video0` zur Verfügung. Darüber können die Bilder ausgelesen und weiterverarbeitet werden. Damit genug Kamerabilder zur Verfügung stehen, kann die Kamerabildaufnahme (Framerate) mit bis zu $200\ FPS$ [12] betrieben werden. Die praktische Implementierung zeigt, dass $90\ FPS$ zu erreichen sind, obwohl in Kapitel 5.2 eine höhere FPS, bei einer Bildauflösung von 640×480 Pixeln erreicht wird. Als Kamera-API kommt *Video4Linux* zum Einsatz. Darauf kann das Kamerainterface parametert werden. Weiterhin lässt es sich als Kernelmodul laden, womit die Ausführung nicht im Userland stattfindet. Das Kernelmodul muss dem System bekannt gemacht werden und die Kamerabildauflösung wird auf 640×480 verkleinert. Folgende Kommandos sind dafür notwendig:

```
1 $ sudo modprobe bcm2835-v4l2
2 $ v4l2-ctl --set-fmt-video=width=640,height
   =480, pixelformat=4 -d /dev/video0
```

4.2 Software Bildverarbeitung

Für die Positionsbestimmung des Tischtennisballs verhält sich die Software nach dem Programmablaufplan (Abbildung 10). Bevor die Software gestartet werden kann, muss eine Kalibrierung über eine zweite Software stattfinden. Dazu zählt die Größe des Kamerafensters, welches ausgeschnitten wird, die Grauwertschwelle, der Abstand der Kameralinse zur Acrylglasplatte in mm und der Durchmesser des Tischtennisballs in mm - wenn der Ball auf der

Acrylglasplatte liegt. Die Ergebnisse der Kalibrierung werden in eine YAML-Datei geschrieben und dann von der Bildverarbeitungssoftware eingelesen und weiter verarbeitet. Der Programmablaufplan (Abbildung 10) zeigt die einzelnen Schritte: von dem Einlesen des Kamerabildes bis zur Ausgabe der drei Koordinaten. Vor jeder Weiterverarbeitung des Kamerabildes wird üblicherweise eine Entzerrung vorgenommen. Allerdings wird aufgrund der Messung (Kapitel 5.4) darauf verzichtet. Danach wird ein Rechteck aus dem aufgenommenen Foto ausgeschnitten, die nur die Acrylglasplatte zeigt. Um den Tischtennisball von der restlichen Umgebung zu isolieren, wird das Kamerabild zuerst in den YCrCb-Farbraum gewandelt und nur das Cb-Foto weiter betrachtet. Alle Pixel, die über der Grauwertschwelle liegen, werden zu weißen Pixeln, die anderen zu schwarzen (genauere Untersuchung in Kapitel 5.1). Dadurch bekommen wir ein binarisches Bild. Um eventuelles Rauschen zu entfernen, kamen die morphologischen Transformationen [24] Erosion und Dilation zum Einsatz. Es hat sich gezeigt, dass aufgrund des guten Kontrastes zwischen Tischtennisball und Umgebung, die morphologischen Transformationen nicht nötig sind. Die Fläche des Tischtennisballs wird mittels des Momentum (Formel 6.6) berechnet. Bevor nun die Berechnung der x , y und z -Koordinate vorgenommen werden kann, brauchen wir für die z -Koordinate eine Referenzfläche. Diese ist in Gleichung 2.2 als Variable $A_{Ball \ liegt \ auf \ der \ Platte}$ gegeben. Diese Referenzfläche wird durch den Mittelwert der Fläche von den ersten zehn Bildern bestimmt. Dabei darf sich der Ball nicht bewegen. Nachdem die Referenzfläche ermittelt wurde, wird mit dem Momentum die x und y Koordinaten des Mittelpunktes bestimmt. Da diese Mittelpunkte noch in die Einheit mm überführt werden müssen (Kapitel 6.2), wird in der Software, die Variable $pxTomm$ berechnet. Diese gibt Auskunft darüber, wie viele Millimeter 1 [\sqrt{pixel}] entspricht. Diese Variable, welche für die Berechnung der z -Koordinate notwendig ist, wurde bis kurz vor Abgabe der Arbeit, falsch ermittelt, da die z -Koordinate einen anderen $pxTomm$ -Wert besitzt als für die x und y -Koordinaten. Dieser Fehler hat dazu geführt, dass die x und y -Koordinate nur auf ± 16 [mm] genau ist. Die Messtoleranz (Genauigkeit) nach der Behebung des Fehlers wird in Kapitel 5.6 näher erläutert. Die z Koordinate wird mittels der Formel 2.2 berechnet. Die berechneten Koordinaten werden über *UART* ausgegeben, für den Regelkreis in Teilbereich C.

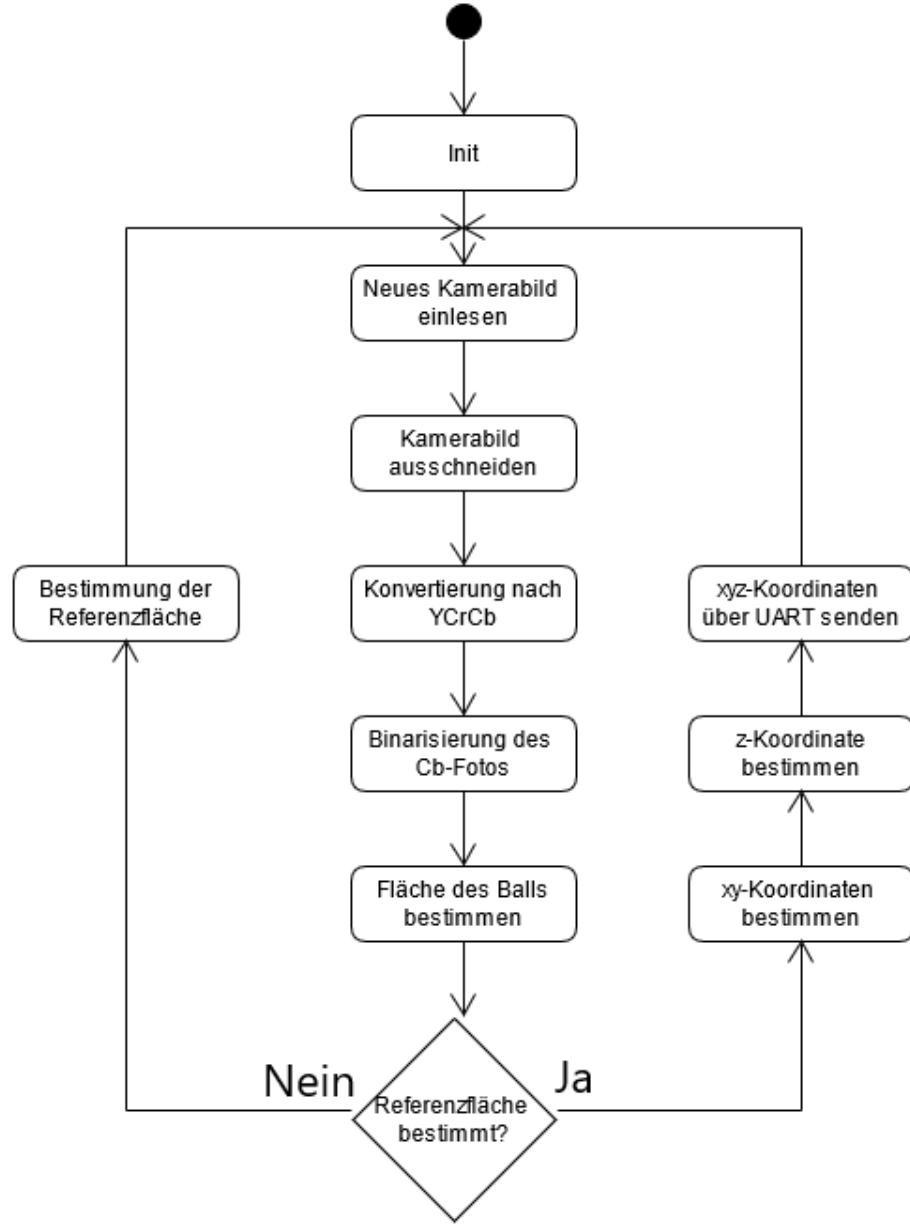
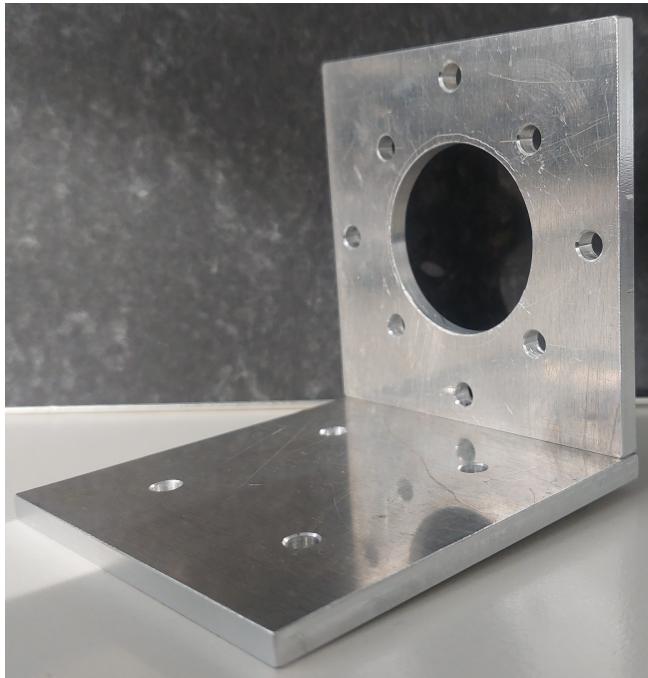


Abbildung 10: Programmablaufplan

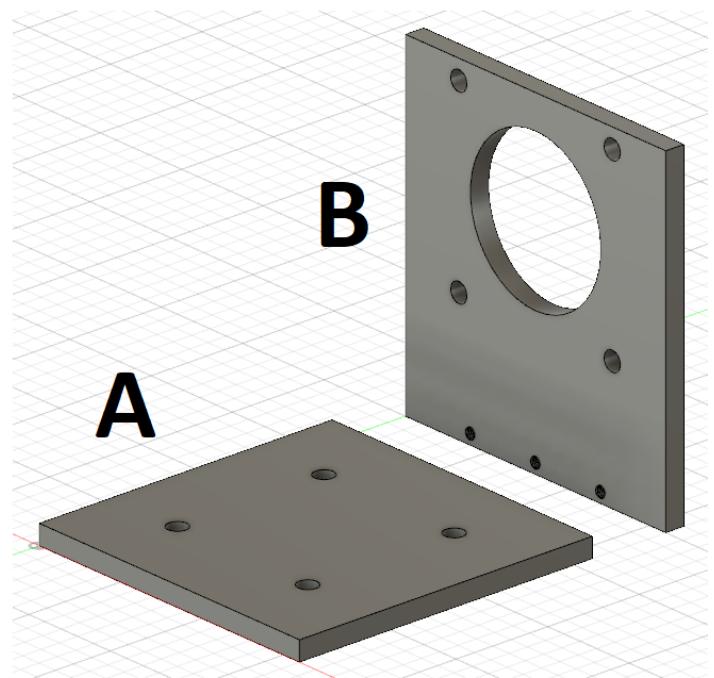
4.3 Mechanischer Aufbau

Motorhalterung

Bei der Halterung für den Schrittmotor wurde darauf geachtet, dass der Motor hoch genug hängt und der Hebelarm bei Bewegung nicht den Boden berührt. Der Motor besitzt vier M4-Schraubenlöcher für eine Verankerung. An diesen Löchern wird sich orientiert. Abbildung 11a zeigt die Motorhalterung ohne Motor. Die Halterung besteht aus zwei Bauteilen, wie Abbildung 11b darstellt. Dies ermöglicht eine einfachere Fertigung da bei der Verwendung eines anderen Motors nur Teil B (Abbildung 11b) erneut gefertigt werden muss.



(a) Motorhalterung



(b) 3D-Modell der Motorhalterung

Abbildung 11: Motorhalterung gefertigt und als 3D-Modell

Kamerahalterung

Für die Kamerahalterung wurde sich an den Maßen der Raspberry Pi 3 Kamera orientiert. Dabei ist zu beachten, dass die Kameraleiterplatte auf beiden Seiten mit Bauteilen bestückt ist. Die Kamera darf somit nicht auf der Kamerahalterung aufliegen, sondern muss mittels den vier Befestigungslöchern befestigt werden, sodass kein Kontakt mit der Kamerahalterung besteht. Weiterhin muss darauf geachtet werden, dass sich die Kamera bei Vibrationen der Motoren nicht verzieht. Abbildung 12 zeigt den Aufbau der installierten Kamera.



Abbildung 12: Kamerahalterung mit Raspberry Pi 3 Kamera

Hebelarm Z1 – Z6

Bei der Konstruktion des Hebelarms wurde auf geringes Gewicht, hohe Belastbarkeit und einfache Montage geachtet. Aus diesen Gründen besteht der Hebelarm Z_1 , Z_2 und Z_3 (Abbildung 13a) aus jeweils zwei Aluminiumteilen, bei denen es einen kleinen Spalt in der Mitte gibt. Dadurch verringern wir das Gewicht anstelle eines durchgehenden Aluminiumteils. Für die Befestigung der jeweiligen Teilarme kommen $M3$ -Schrauben zum Einsatz. Die Befestigung erfolgt immer am Drehpunkt, die Bohrungen sind alle ohne Gewinde. Ein Gewinde würde nur den Fertigungsaufwand erhöhen, ohne eine maßgebliche Verbesserung zu erzeugen. Diese drei Arme machen nur eine Bewegung in Pitch Richtung. Hebelarm Z_4 ist ein Quader mit zwei Befestigungslöchern und einer Bohrung in der Mitte, welches für die Aufhängung von Z_5 dient (Abbildung 14a). Die Aufhängung (rote Schraube in Abbildung 14a) soll eine Bewegung des Teilarms Z_5 in Roll-Richtung ermöglichen (blauer Pfeil in Abbildung 14a). Aus diesem Grund ist diese Bohrung um 90° zu den beiden Befestigungslöchern gedreht. Teilarm Z_5 existiert nur als Halterung für Z_6 (Abbildung 13b). Z_5 besitzt ebenfalls eine Quaderform, mit zwei Befestigungslöchern (für Z_6) und einer Bohrung für die Aufhängung an Z_4 . Die Acrylglasplatte wird von oben mit zwei Schrauben auf den Teilarm Z_6 (Abbildung 13b) geschraubt. Diese Verbindung der Acrylglasplatte mit dem Hebelarm hat sich als robust erwiesen. Die Gesamtkonstruktion wird auf einer Holzplatte befestigt. Abbildung 14b zeigt den Gesamtaufbau von einem Hebelarm.

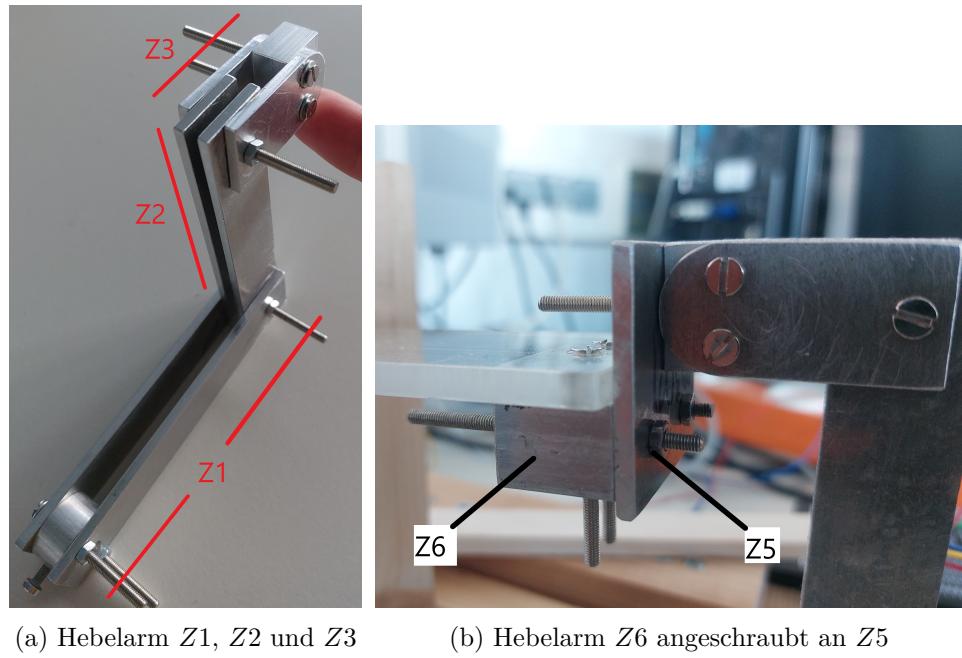
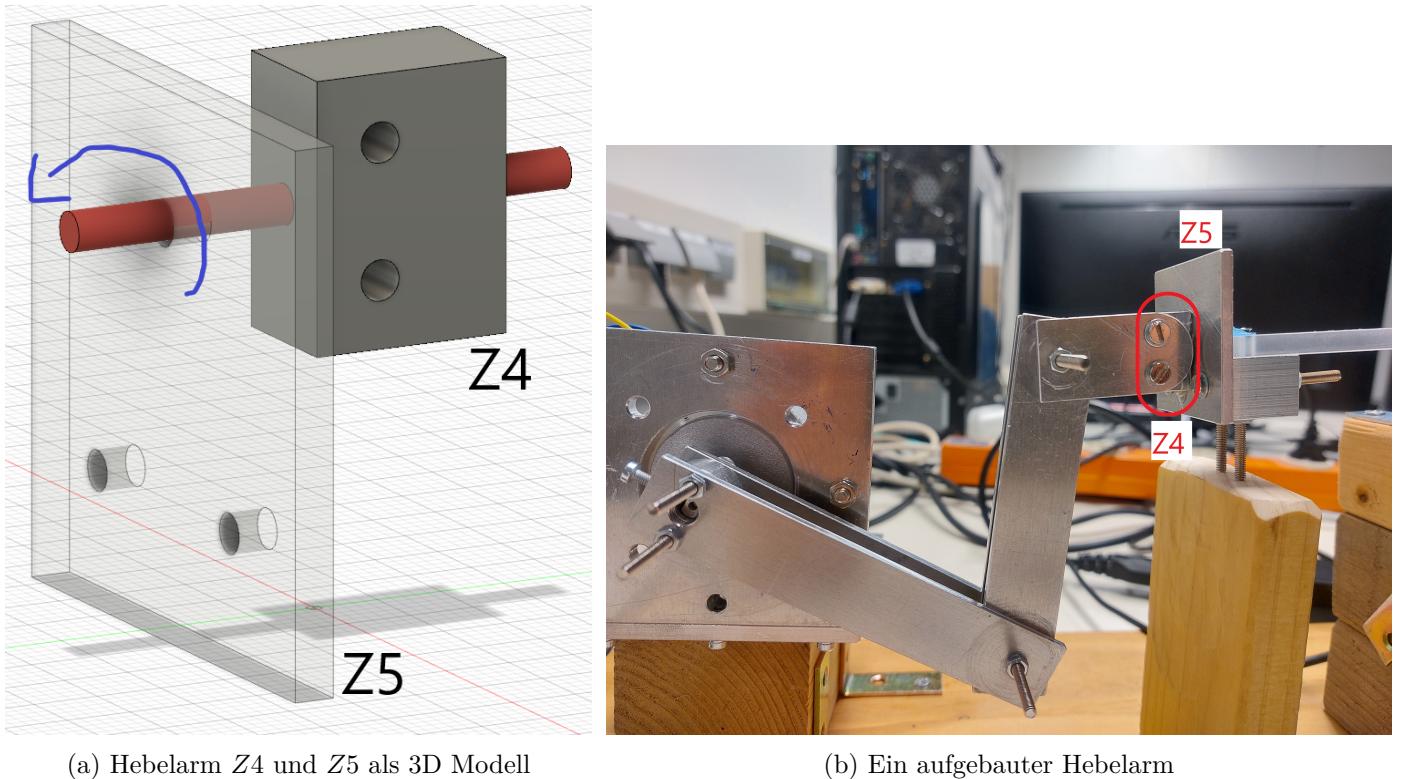


Abbildung 13: Teile des Hebelarms

Abbildung 14: Hebelarm Aufhängung Z_4 und Z_5

Hebelarm Übersetzung

Um die Eigenschaften eines bewegten Hebelarms besser zu verstehen, ist die Übersetzung von der Winkeländerung der Motorwelle α bis zu der Winkeländerung der Acrylglasplatte γ in Kapitel 6.4 und 6.5 hergeleitet und anhand eines Beispiels berechnet worden. Da der Neigungswinkel der Acrylglasplatte auf $\pm 20^\circ$ begrenzt wird, sind alle anderen Neigungswinkel irrelevant, egal ob das Ergebnis dabei real oder komplex ist. Abbildung 15 zeigt zwei Simulationen der Bewegung der Acrylglasplatte in einer Dimension. Dabei entspricht Punkt x in Abbildung 15a der Motorwelle und Punkt T entspricht der Drehachse der Acrylglasplatte. Abbildung 15a zeigt die Simulation für eine vollständige Motordrehung. Dabei entspricht eine rote Linie in Abbildung 15a, dass die Lösung in der komplexen Ebene liegt. Abbildung 15b zeigt die Bewegung von $\gamma = -20^\circ$ bis 20° . Es ist zu erkennen, dass es dabei keine komplexe Lösung gibt, da ansonsten diese Winkelposition im praktischen Aufbau nicht möglich ist.

(a) Bewegung von 0° bis 360° (b) Bewegung von -20° bis $+20^\circ$

Abbildung 15: Simulation eines Hebelarms in einer Dimension

4.4 Software Motorsteuerung

Die Motorsoftware ist das Zwischenglied zwischen dem Regler und den Motortreibern (Abbildung 16). Neben der Motorsoftware wird noch der Sensor *MPU6050* angesteuert. Dieses Kapitel erläutert im Einzelnen die folgenden Subteilbereiche.

1. *MPU6050* Ansteuerung
2. NEMA 23 Ansteuerung
3. Motorsoftware

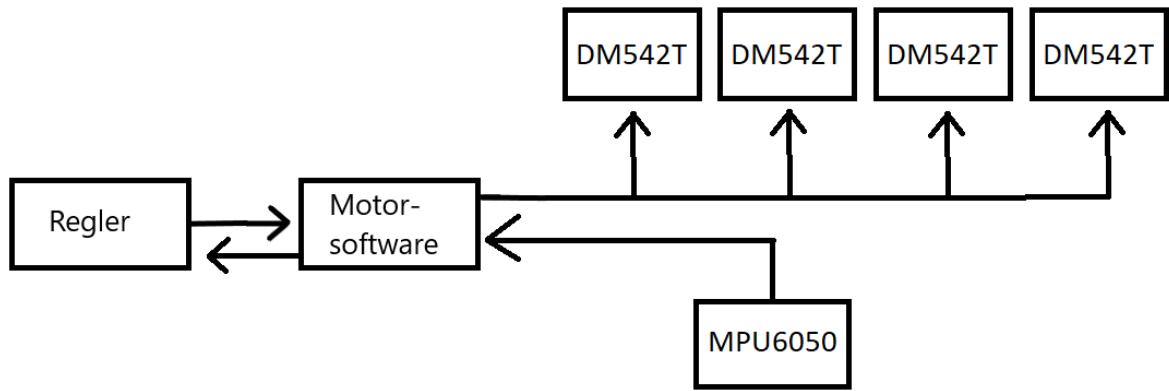


Abbildung 16: Skizze des Zusammenhangs zwischen Regler, Motorsoftware und Motortreiber

MPU6050 Ansteuerung

Der *MPU6050* Sensor wird benötigt, da er die Neigung der Acrylglasplatte misst und den Winkel dem Regler zur Verfügung stellt. Dafür ist der Sensor auf die Acrylglasplatte geklebt worden. Theoretisch ist das nicht nötig, da wir den Winkel aus der Drehung der Motorwelle berechnen können; allerdings kann dieses Ergebnis abdriften aufgrund einer Fehlerfortpflanzung. Für die Ansteuerung des *MPU6050* wird eine von GitHub bereitgestellte Bibliothek [22] verwendet. Diese ist minimal verändert worden, aufgrund dessen, dass das Auslesen im Fehlerfall (Source-Code Auschnitt 1) zu einem kompletten Absturz des Systems geführt hätte. Die Verbesserung findet sich in Source-Code Auschnitt 2.

Listing 1: Teil des Source-Codes, der im Fehlerfall zum Absturz des Systems führt

```
1 while( HAL_I2C_Master_Transmit ( . . . ) != HAL_OK) ;
```

Listing 2: Teil des Source-Codes der im Fehlerfall nicht zum Absturz des Systems führt

```
1 HAL_StatusTypeDef ret = HAL_I2C_Master_Transmit
    ( . . . ) ;
2 if( ret != HAL_OK)
3 {
4     return SD_MPU6050_Result_Error ;
5 }
```

Von dem Sensor werden die drei Beschleunigungsachsen und die drei Winkelgeschwindigkeiten über *I2C* ausgelesen. Das Auslesen geschieht alle 10 ms. Danach werden die Rohdaten noch in die Winkel in *x* und *y*- Richtung umgerechnet. Kapitel 6.3 erläutert dies näher.

NEMA 23 Ansteuerung

Der DM542T Motortreiber wird mit den *PULSE* und *DIR*-Pins an den Controller angeschlossen. Der *ENABLE*-Pin ist dagegen, wie es im Datenblatt empfohlen wird, nicht angeschlossen. Dadurch ist das Haltemoment ständig aktiv. Die Einstellung der Microschritte

erfolgt manuell. Diese ist auf 3200 Microschritte ($\frac{360}{3200} \cdot [\frac{\circ}{Schritte}] = 0,1125 [\frac{\circ}{Schritt}]$) eingestellt. Bei der Anfahrt der Motoren wird zuerst langsam mit einer konstanten Beschleunigung beschleunigt und, nachdem 50% der Schritte gefahren worden sind, wieder abgebremst. Dies hat zum einen den Vorteil, dass deutlich höhere Geschwindigkeiten erreicht werden können, der Hebelarm nicht in ungewollte Schwingung gerät und dass eine unterschiedliche Bremsbeschleunigung gewählt werden kann. Damit nicht zu viele Schritte absolviert werden, muss vorher die Schrittanzahl so berechnet werden, dass die Geschwindigkeit maximal wird, damit im richtigen Moment zwischen Beschleunigung und Bremsung umgeschaltet werden kann.

Motorsoftware

Die Motorsoftware muss folgende Aufgaben erfüllen:

1. Entgegennehmen neuer Kommandos vom Regler
2. Senden der Winkelinformationen
3. Parallele Ansteuerung von zwei Motoren pro Achse

Die Motorsoftware beinhaltet ein kleines statisches Betriebssystem, womit alle Abläufe periodisch ausgeführt werden (Batchbetrieb). Abbildung 17a zeigt ein grobes Strukturbild der Software. Das Auslesen, verarbeiten und übermitteln der zwei Winkel von der *MPU6050* geschieht unabhängig von den weiteren Aufgaben der Software. Für den Empfang von neuen Nachrichten ist, ein Interrupt verwendet worden, dieser überprüft jedes Zeichen auf seine Korrektheit und setzt die entsprechenden Variablen. Sollte eine Nachricht für eine Achse, die sich noch in einer Bewegung befindet empfangen werden, wird diese Nachricht ignoriert. Damit ist sichergestellt, dass eine Bewegung der Motoren nicht gestört wird, die ansonsten Schrittverluste bzw. falsche Motorpositionen zur Folge hätten. Eine Nachricht beinhaltet immer den Soll-Winkel von der *x* und *y*- Achse. Falls keine Motorbewegung ausgeführt und eine Nachricht mit neuen Sollwinkeln empfangen wird, müssen die Motorschritte für jeweils zwei Motoren berechnet werden. Da immer zwei Motoren eine Achse (gilt nicht für die *z*-Achse) kontrollieren, müssen pro Achse beide Motoren die gleiche Richtung aufweisen; nur die Motorwellenwinkeländerung ist für beide Motoren minimal unterschiedlich. Der Ablauf von dem Empfang einer Nachricht bis zum Start der Motoren zeigt Abbildung 17b. Zuerst wird die Nachricht entgegengenommen, und der jeweiligen Achse zugewiesen. Danach findet eine Überprüfung statt, ob diese angesteuerte Achse bereits in Bewegung ist. Bei Stillstand der jeweiligen Achse muss der Soll-Winkel ($Winkel_{Soll}$) in den Winkel des Hebelarms ($Winkel_{Hebelarm} = \gamma$) umgerechnet werden ($Winkel_{Hebelarm} = Winkel_{Soll} - Winkel_{aktuell}$). Dadurch weiß die Software, dass der $Winkel_{Soll}$ erreicht werden kann, indem der Hebelarm eine Winkeländerung von $Winkel_{Hebelarm}$ vornimmt. Eine weitere Umrechnung ist notwendig der gefahren werden muss für die Berechnung der Motorwellenwinkel ($Winkel_{Motor} = \alpha$). Dafür muss Gleichung 6.11 nach α gelöst werden. Aus Abbildung 17b erkennt man, dass die

Hebelarm-Übersetzung nicht linear verläuft und daher die Berechnung für beide Motoren separat erfolgen muss. Aus Abbildung 17b kann nun entnommen werden, dass die Berechnung der konkreten Schritte (Gleichung 4.4) für beide Motoren ansteht.

$$\begin{aligned} \text{winkel}_{\text{pro step}} &= \frac{360}{\text{mikroschritte}} \\ \text{steps} &= \frac{\text{winkel}_{\text{motor}}}{\text{winkel}_{\text{pro step}}} \end{aligned} \quad (4.4)$$

Zum Schluss wird noch für die Dreiecksfahrt, der Schritt zwischen Beschleunigung und Bremsen berechnet und die entsprechenden Timer und die dazugehörigen Flags gesetzt.

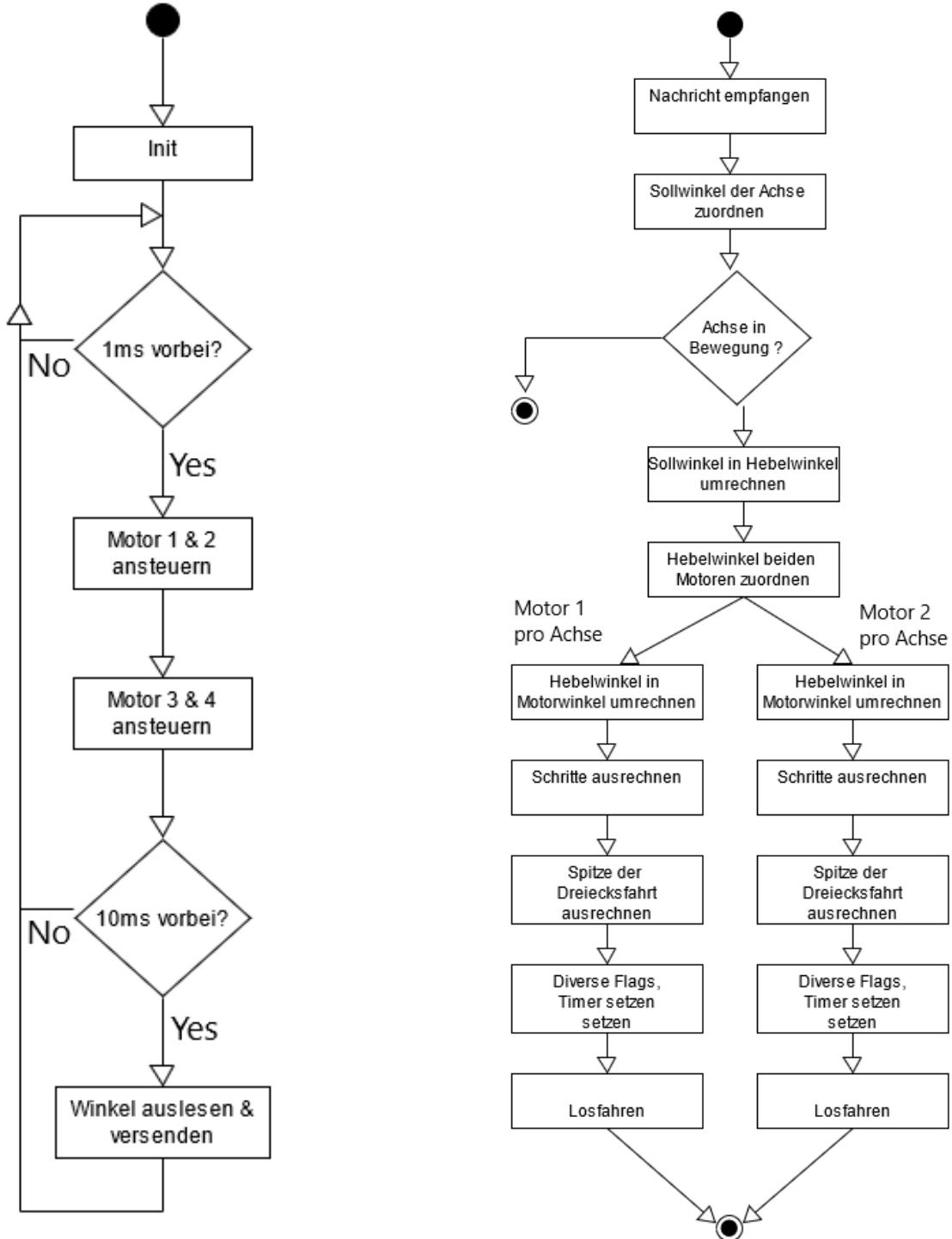


Abbildung 17: Strukturbilder der Motorsoftware

4.5 Regler

Um den Regler zu entwerfen, brauchen wir die Sprungantwort des Systems, sowie die mathematische Modellierung des Systems. Die Sprungantwort findet sich in Abbildung 18. Daraus kann man erkennen, dass die Anfahrt weniger als 500 ms dauert. Die Winkeländerung beim Anfahren senkt sich erst ab und steigt dann asymptotisch zu dem gewünschten Winkel an. Ein Überschwingen ist nicht vorhanden. Die mathematische Modellierung des Systems wird in Kapitel 6.6 näher erläutert. Aus Kapitel 6.6 bekommen wir die Formel $x = \int \sin(\alpha) \cdot (-9,81 \frac{m}{s^2})$. Abbildung 19 zeigt die Formel in Simulink. Für den Regler kommt ein PID-Regler zum Einsatz. Dieser verbindet die guten Eigenschaften von anderen Reglern. Der PID-Regler hat die folgende Form:

$$P + \frac{D \cdot N}{1 + N \cdot \frac{1}{s}} \quad (4.5)$$

Mithilfe des eingebauten Tuners in Simulink kann die Übertragungsfunktion bestimmt werden, bzw. die Koeffizienten P , D und N werden ausgegeben. Für die Koeffizienten ergeben sich die Werte $P = -1,9439$, $D = -1,8856$ und $N = 69,54$.

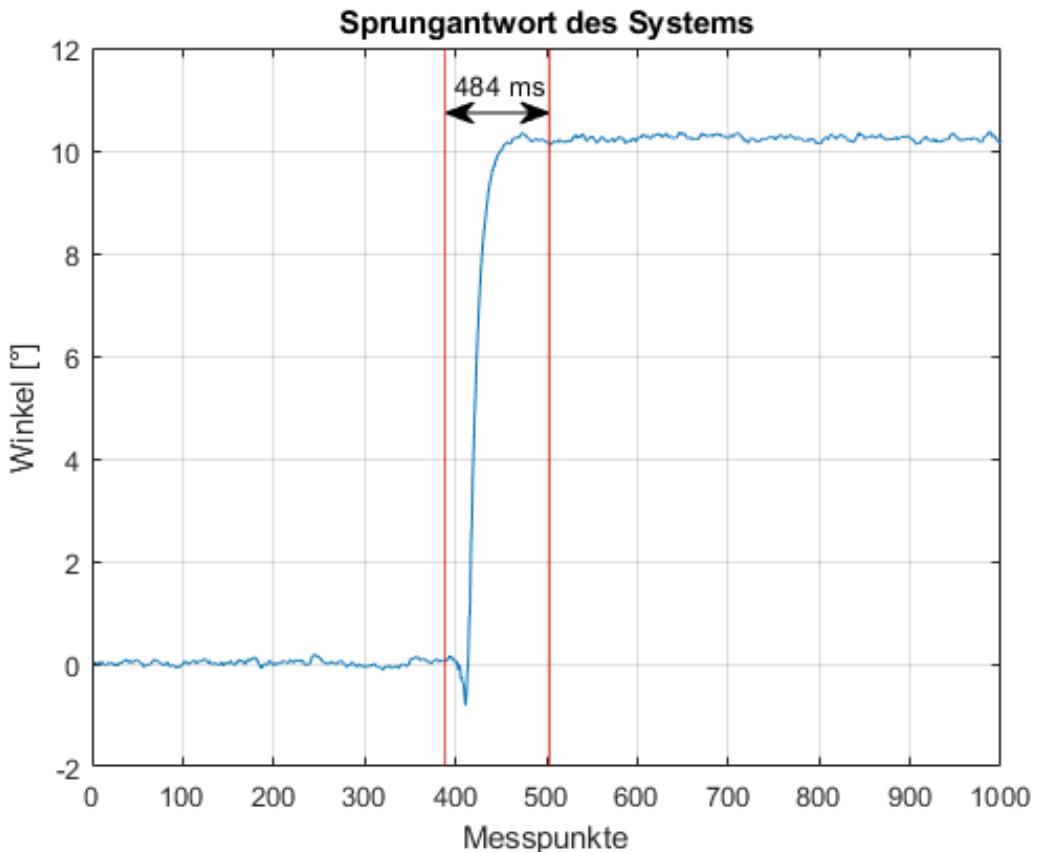


Abbildung 18: Verhalten der Acrylglasplatte beim Anfahren einer neuen Position

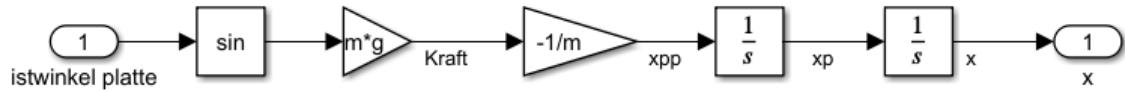


Abbildung 19: Modellierung des Systems in Simulink realisiert

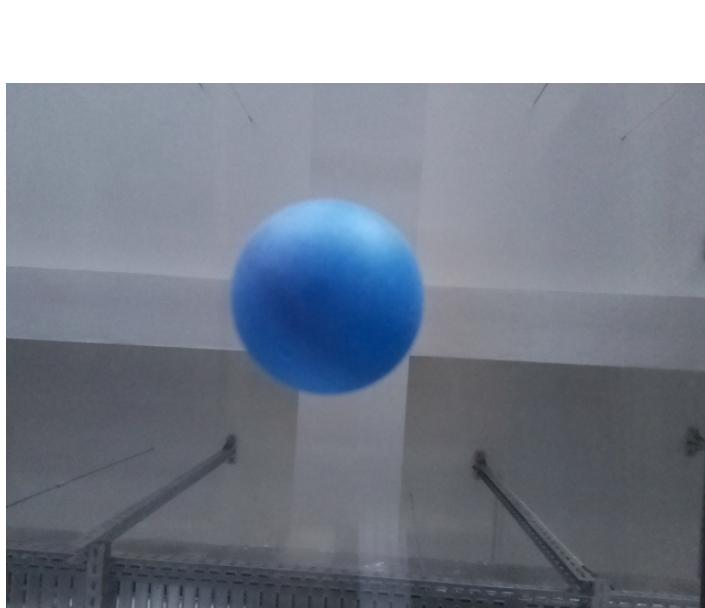
Bei der Realisierung ist zu beachten, dass der PID-Regler nichts von den Grenzen des physikalischen Systems weiß. Es kann also passieren, dass der PID-Regler eine Winkeländerung von $\gamma > 20^\circ$ oder $\gamma < -20^\circ$ vorschlägt, aber das physikalische System dies nicht realisieren kann. Dafür muss nach dem PID-Regler ein Saturation-Block eingefügt werden. Dieser verhindert durch *min* und *max* Werte, dass es Ausreißer gibt. Die Stellgröße wird über *UART* dem Teilbereich *D* zur Verfügung gestellt. Für die Messeinrichtung kommt die Bildverarbeitung zum Einsatz. Dies wird in Kapitel näher 4.2 erklärt. Die Schnittstelle zwischen der Bildverarbeitung und dem Regler wird mittels *UART* realisiert.

5 Messungen

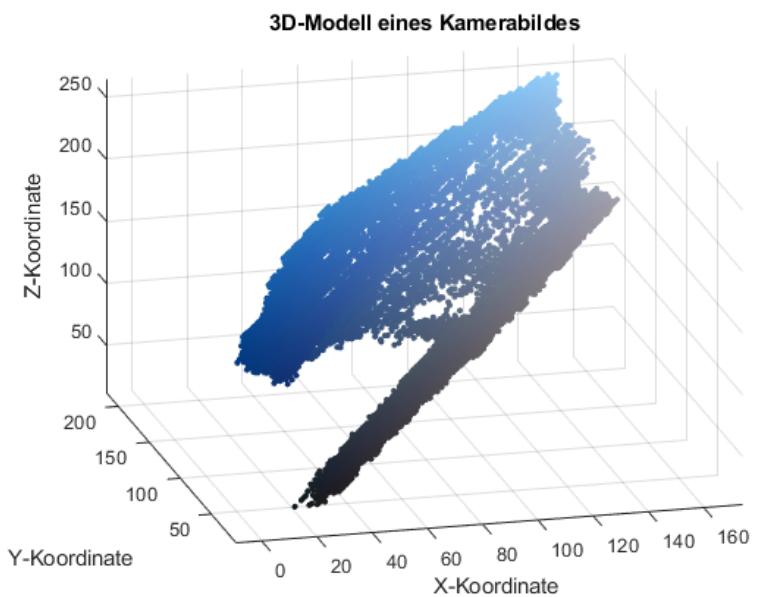
Messungen des Systems werden benötigt, um die Genauigkeit sowie die Korrektheit zu überprüfen.

5.1 Detektion des Tischtennisballs

Bei der Detektion des Tischtennisballs gibt es das Problem, dass es nicht immer einen hohen Kontrast zwischen Hintergrund und dem Tischtennisball gibt, insbesondere, wenn gerichtetes Licht auf den Tischtennisball trifft. Für die Detektion des Tischtennisballs kamen verschiedene Verfahren mit unterschiedlichen Tischtennisfarben zum Einsatz. Eine diffuse Lichtquelle ist gegeben. Aus Abbildung 20b, welches eine 3D-Darstellung des Kamerabildes 20a ist, erkennt man, dass es nicht möglich ist, mittels eines Schwellwerts für alle drei Kanäle nur den Farbanteil des Tischtennisballs zu isolieren. Ein Schwellwert entspricht einer Ebene im 3D-Modell. Eine Konvertierung in einen anderen Farbraum $YCbCr$ hat sich als beste Lösung hervorgetan. Hierbei hat sich die Konvertierung in den Cb Kanal als vorteilhaft erwiesen. Dort entsteht ein Kontrast zwischen dem Tischtennisball und dem Hintergrund, welcher mit einem Schwellwert unterschieden werden kann (Abbildung 21). Werden nun verschiedene Lichtverhältnisse berücksichtigt (gerichtetes Licht), funktioniert dieses Verfahren nicht mehr, denn der Kontrast verfließt. Abbildung 22 und 23 verdeutlichen es. Diese Methode der Balldetektion funktioniert aber nicht in der rauen Umgebung, daher und aufgrund des begrenzten Zeitbudgets wurde auf veränderbare Lichtverhältnisse keine Rücksicht genommen. Der Kontrast bei einem blauen Tischtennisball ist im Vergleich zu anders farbigen Bällen am größten (Abbildung 24). Aus diesem Grund wird ein blauer Tischtennisball verwendet.

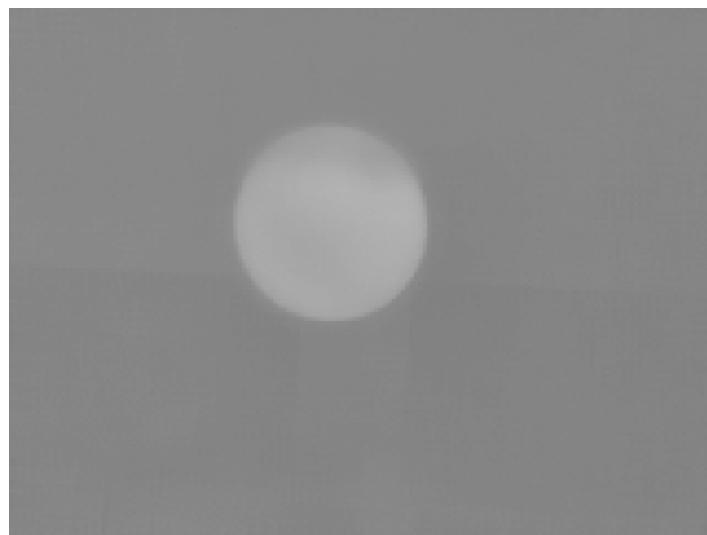
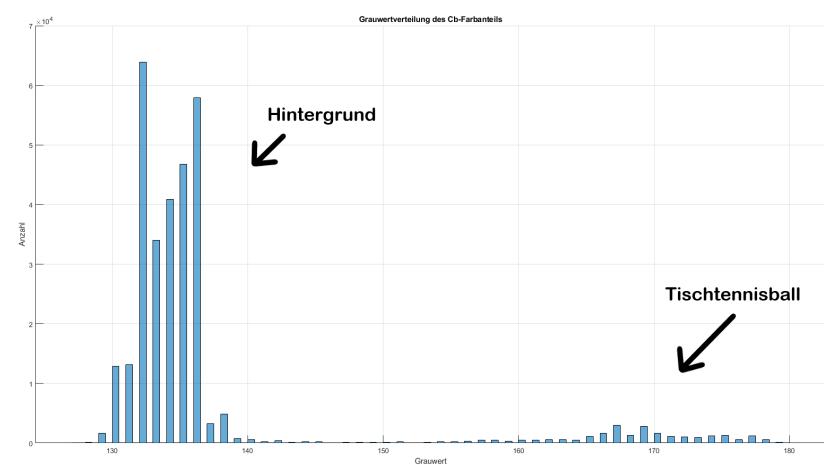


(a) Kamerabild von einem blauen Tischtennisball



(b) 3D-Modell

Abbildung 20: Kamerabild mit dem entsprechenden 3D-Modell

(a) Cb Kanal vom $YCrCb$ Farbraum(b) Histogramm des Cb Kanals (Abbildung 21a)Abbildung 21: Cb Kanal vom $YCrCb$ Farbraum mit dem dazugehörigen Histogramm

(a) Diffuse Lichtverhältnisse (RGB Foto)

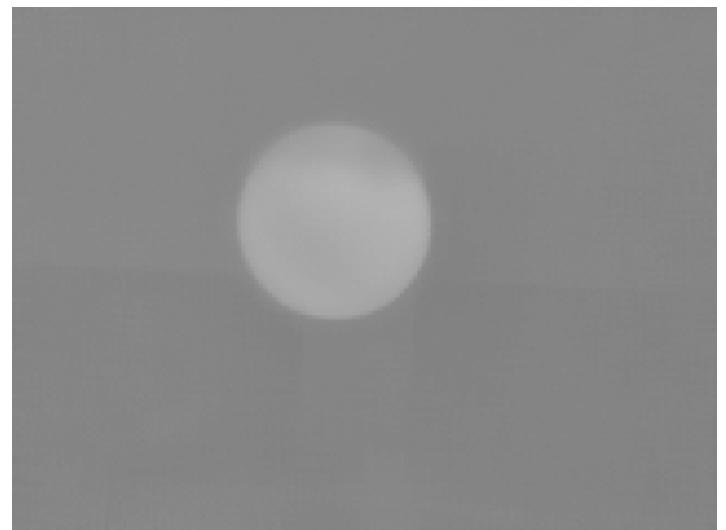
(b) Diffuse Lichtverhältnisse ($YCrCb$ Foto)

Abbildung 22: Diffuse Lichtverhältnisse



(a) Gerichtete Lichtverhältnisse (RGB Foto)



(b) Gerichtete Lichtverhältnisse (YCrCb Foto)

Abbildung 23: Gerichtete Lichtverhältnisse

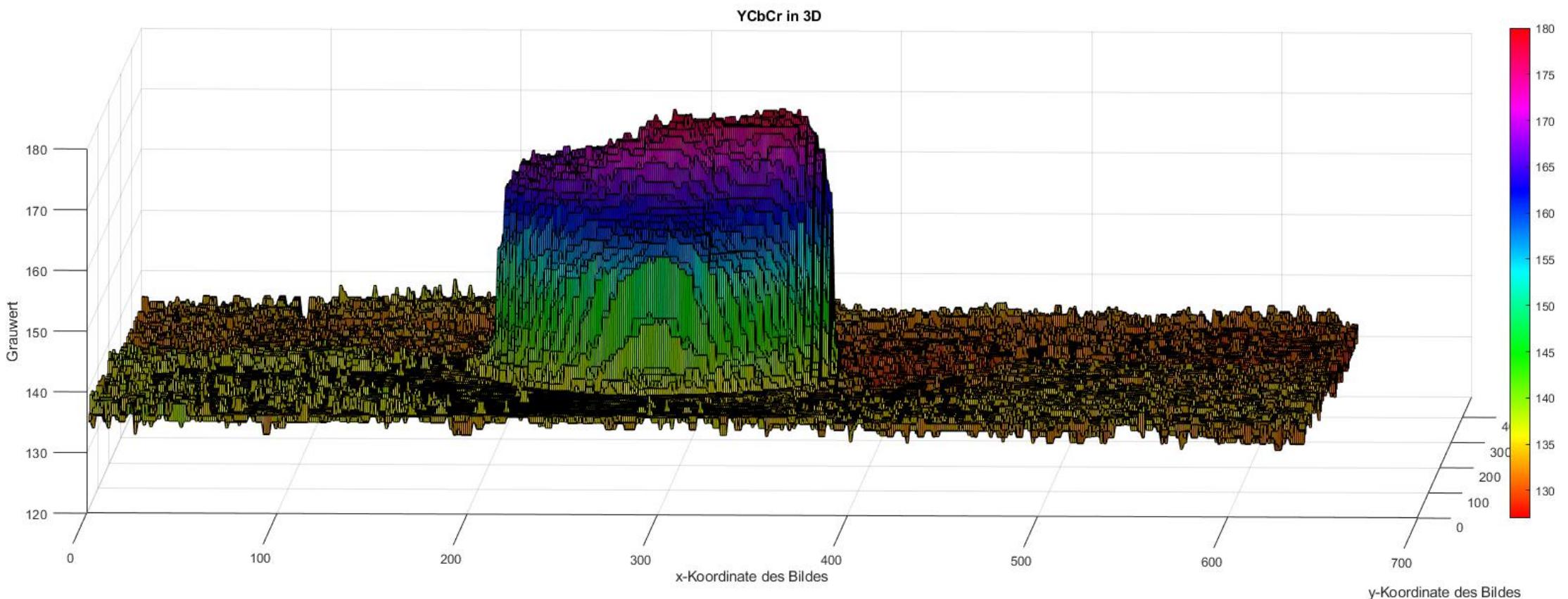
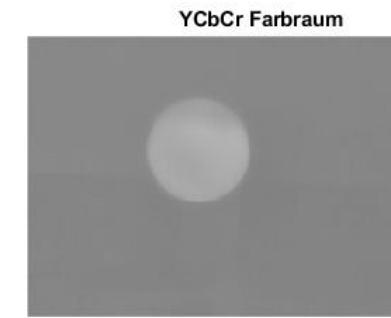
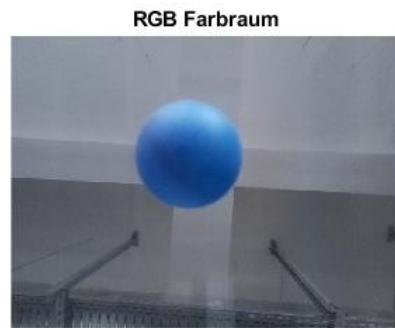


Abbildung 24: Blauer Ball in zwei Farträumen

5.2 Software-Bildwiederholfrequenz

Bei dieser Messung wird untersucht, wie schnell eine Veränderung des Videobildes von der OpenCV Software wahrgenommen wird. Dabei wird die Kamera mit einem Gegenstand abgedeckt und dieser wird ruckartig entfernt (Abbildung 25). Die Software berechnet den Durchschnittswert aller Pixel pro Farb-Kanal, die nicht maskiert sind. Dies ist notwendig, um die ruckartige Bewegung festzustellen. Mithilfe von Unix-Epoch-Timestamps kann die Verzögerung in Mikrosekundenbereich gemessen werden. Bei der Messung (Listing 3) sind alle ungeraden Zeilennummern das Ergebnis der `cv :: mean()`-Funktion. Die geraden Zeilennummern geben Auskunft über den Timestamp in *ms*. Die ruckartige Entfernung des Gegenstandes startet in Zeile drei und ist in Zeile fünf vorbei. Es müssen also für eine Zeitdifferenz der Timestamp in Zeile sechs mit Zeile vier verglichen werden. Dabei entsteht eine Verzögerung von $2259977319\text{ ms} - 2259977284\text{ ms} = 35\text{ ms}$. Wir haben damit eine Framerate von *28 FPS*. Diese Geschwindigkeit reicht aus, um Bewegungen in OpenCV flüssig darzustellen bzw. zu verarbeiten.

Listing 3: Output der Messung: Software-Bildwiederholfrequenz

```

1 [0.102549, 2.60023, 10.7588, 0]
2 2259977251
3 [0.128343, 3.16085, 13.6267, 0]
4 2259977284
5 [204.067, 212.904, 231.04, 0]
6 2259977319

```

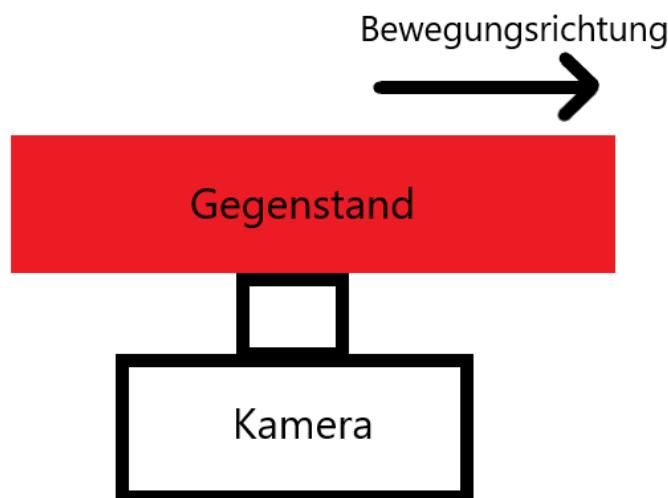


Abbildung 25: Skizze des Versuchsaufbaus

5.3 Geschwindigkeit Bildverarbeitungssoftware

Diesmal wird der Zeitbedarf ermittelt, von dem Einlesen des Kamerabildes bis zur Ausgabe der drei Koordinaten (x, y, z). Abbildung 26 zeigt den Zeitbedarf bei mehreren Durchläufen und bei aktiver und inaktiver Entzerrung. Der Durchschnitt liegt bei $35,2488\text{ ms}$, ohne dass die Software eine Entzerrung vornimmt. Bei aktiver Entzerrung erhöht sich der Zeitbedarf auf $163,3476\text{ ms}$. Dabei ist zu erkennen, dass es hierbei Ausreißer gibt. Diese sind darauf zurückzuführen, dass der Scheduler die Software nicht periodisch aufruft, weil andere Software höher priorisiert ist. Ohne aktive Entzerrung beläuft sich der Geschwindigkeitsvorteil auf den Faktor $\frac{163,3476\text{ ms}}{35,2488\text{ ms}} = 4,63$.

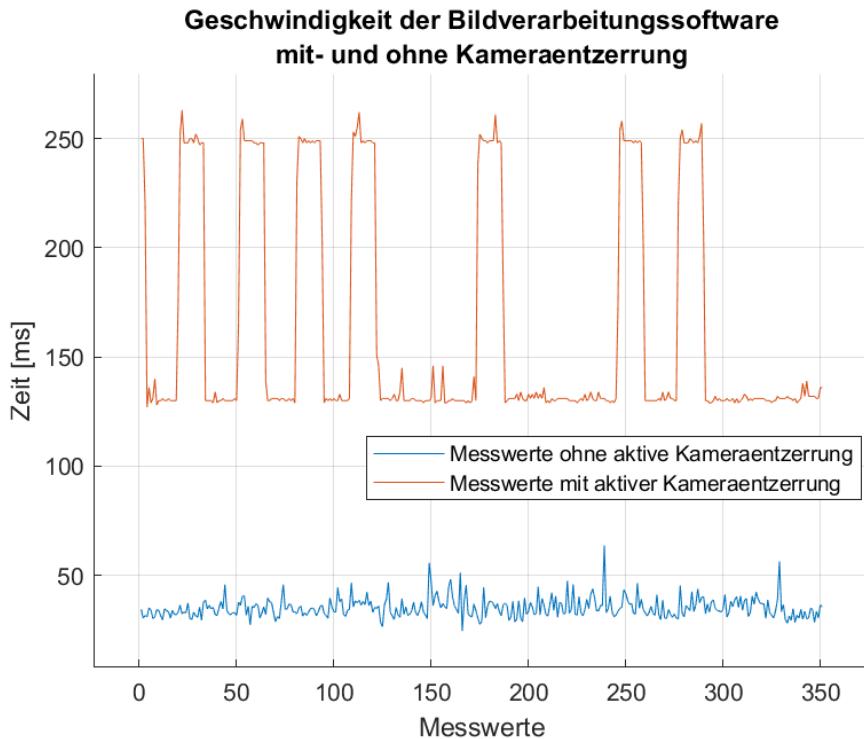


Abbildung 26: Geschwindigkeit der Bildverarbeitungssoftware ohne Entzerrung

5.4 Kamerabildentzerrung

In diesem Kapitel wird untersucht, welchen Einfluss eine Kameraentzerrung auf das Ergebnis hat. Dabei wurde ein kariertes Blatt auf die Acrylglasplatte gelegt (Abbildung 27) und ein Kamerabild aufgenommen. Mithilfe selbstgeschriebener Software können die Koordinaten der Kreuzungspunkte bestimmt werden. Wiederholt man dies für vertikale Linien, die gewölbt erscheinen, erkennt man, dass die Koordinaten in x -Richtung nur

minimal abweichen (Tabelle 4). Das liegt daran, dass der User den Kreuzungspunkt aufgrund einer minimalen Verzerrung des Kamerabildes nicht mit der Maus exakt trifft. Die gleichen Ergebnisse erzielt man auch mit horizontalen Linien. Aus diesem Grund ist ein entzerrtes Kamerabild nur minimal besser.

Listing 4: Output der Kreuzungskoordinaten einer in y -Richtung gewölbten gelben Linie

	x	y
2	598	7
3	604	81
4	602	130
5	609	178
6	609	229
7	609	280
8	611	333
9	612	378
10	612	433

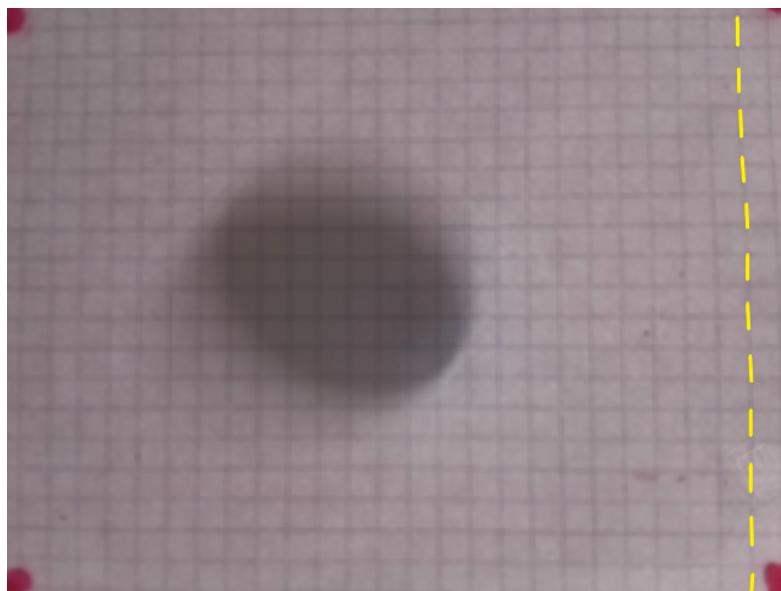


Abbildung 27: Kariertes Blatt auf der Acrylglasplatte

5.5 Einschwingverhalten der Acrylglasplatte

Die Acrylglasplatte besitzt ein Einschwingverhalten, nachdem sie in eine neue Position gefahren wurde. Es wird dabei untersucht, wie lange die Schwingung andauert. Es kommen dabei verschiedene Motorgeschwindigkeiten (Hz) zum Einsatz. Dabei sind immer

50 Schritte bei einem Schrittneigungswinkel von $0,9 \cdot [\frac{\circ}{step}]$ gefahren worden. Die Geschwindigkeit war über den gesamten Zeitraum einer Messung konstant. Abbildung 28 zeigt die Winkelveränderung über die Zeit für alle verwendeten Geschwindigkeiten. Es ist zu erkennen, dass für die Geschwindigkeit 1000 Hz und 2000 Hz weniger Schritte gefahren werden als bei den anderen Geschwindigkeiten. Eine Vermutung ist, dass es zu Schrittverlusten kommt, da der Motor aufgrund der Beschaltung nicht schnell genug das elektrische Feld für einen Schritt stellen kann.

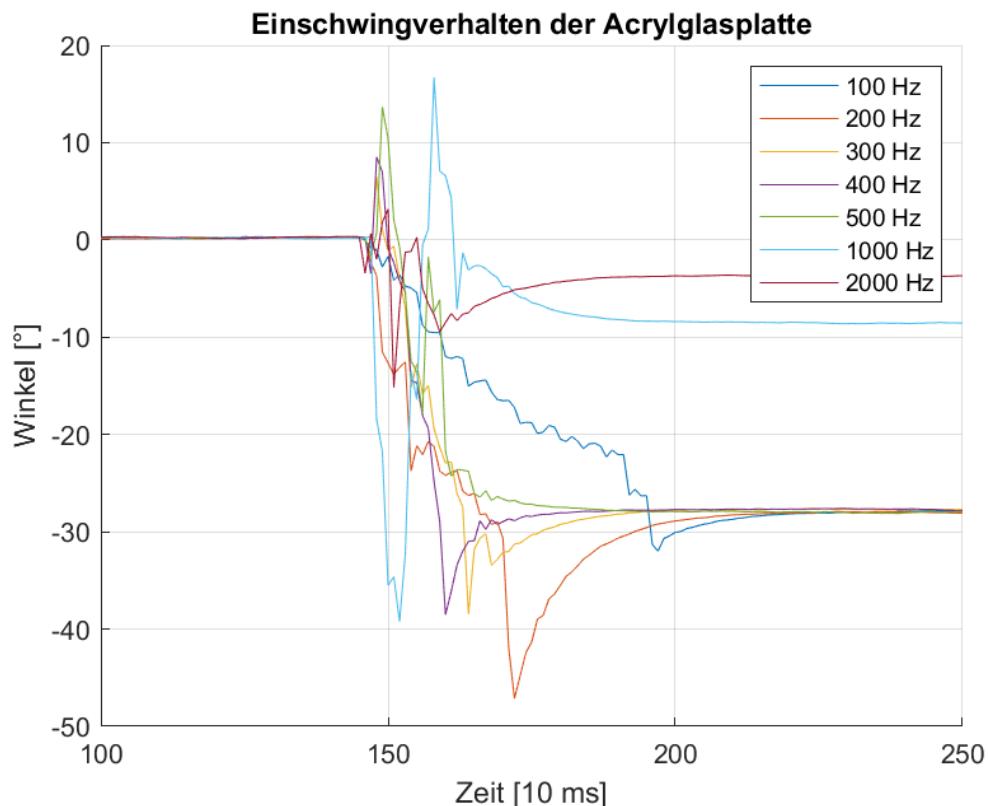


Abbildung 28: Einschwingverhalten bei verschiedenen Geschwindigkeiten

5.6 Genauigkeit der Bildverarbeitungssoftware

Dieses Kapitel untersucht, welche Genauigkeit die Bilderkennung anbietet, nachdem der Fehler mit der Variable *pxTomm* (Kapitel 4.2) behoben worden ist. Dafür sind drei Messungen gemacht worden. Der Tischtennisball liegt auf der Acrylglasplatte an drei unterschiedlichen Positionen – einmal in der Mitte-, oben links- und unten rechts- am Kamerabild. Aus allen drei Abbildungen 29, 30 und 31 erkennt man, dass die *z*-Koordinaten gleich sind. Das liegt daran, dass die Kamerabilder zuvor in einem Video aufgezeichnet

wurden und die Berechnung der z -Achse nicht verändert wurde. Weiterhin zeigen die Abbildungen, dass es einen signifikanten Positionsunterschied in den x und y -Koordinaten gibt. Es ist ersichtlich, dass die Streuung der x und y -Koordinaten gering ist. Die Genauigkeit zum theoretischen Messwert liegt bei $\pm 2,5\text{ mm}$, nachdem der Fehler behoben worden ist.

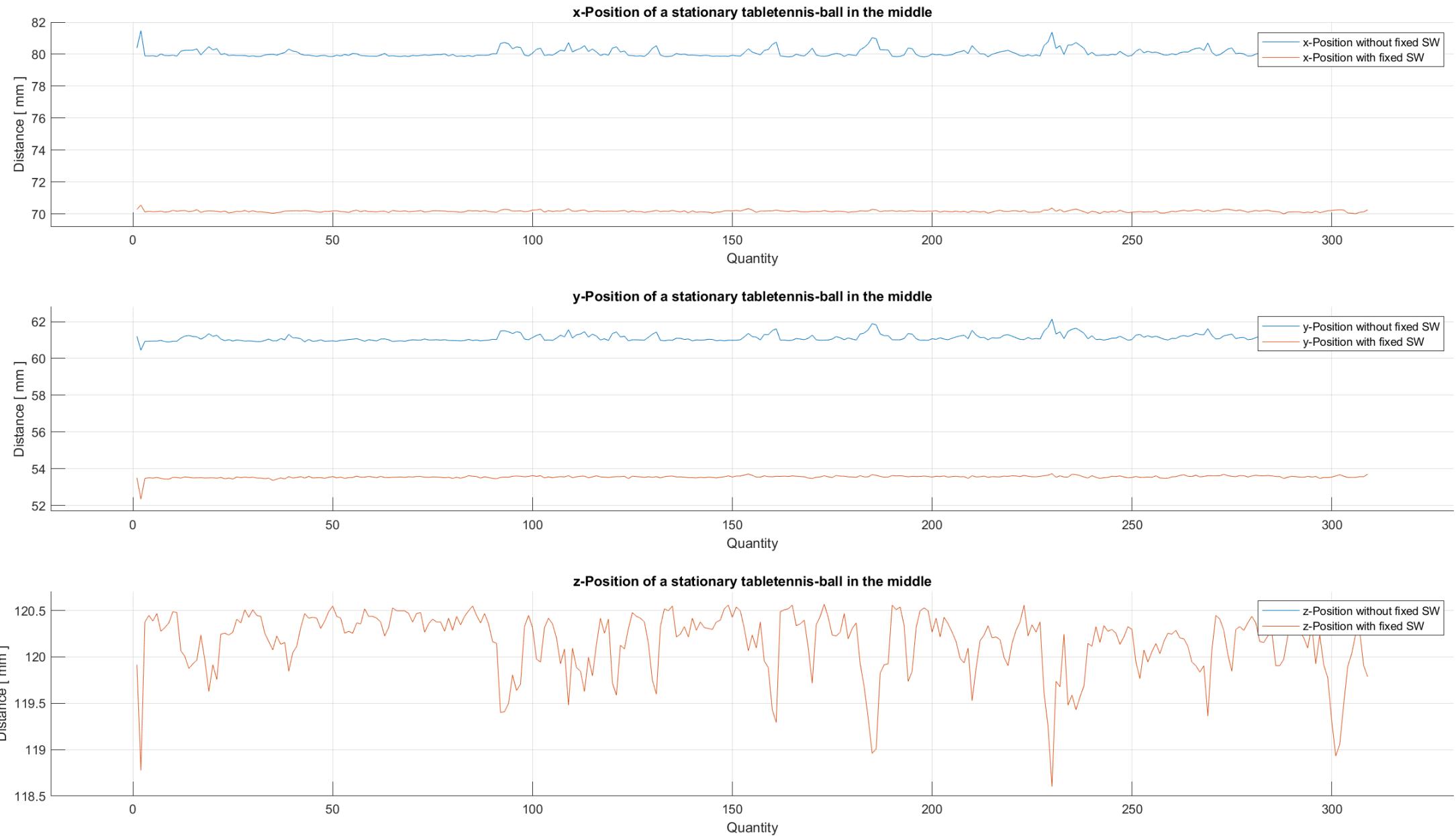
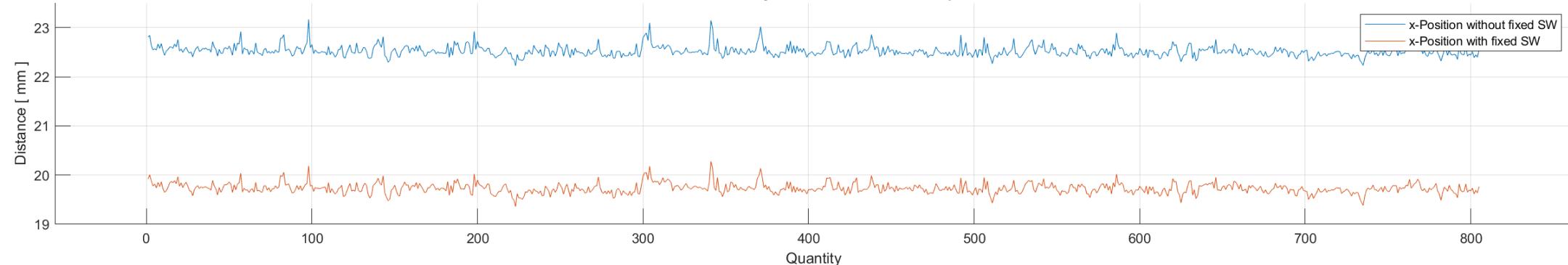
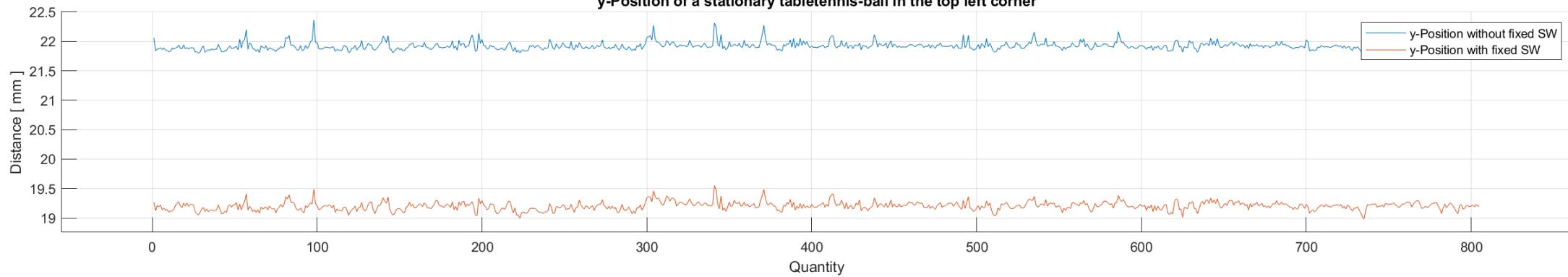


Abbildung 29: Genauigkeit der x , y und z -Position vor und nachdem die Fehler behoben worden sind

x-Position of a stationary tabletennis-ball in the top left corner



y-Position of a stationary tabletennis-ball in the top left corner



z-Position of a stationary tabletennis-ball in the top left corner

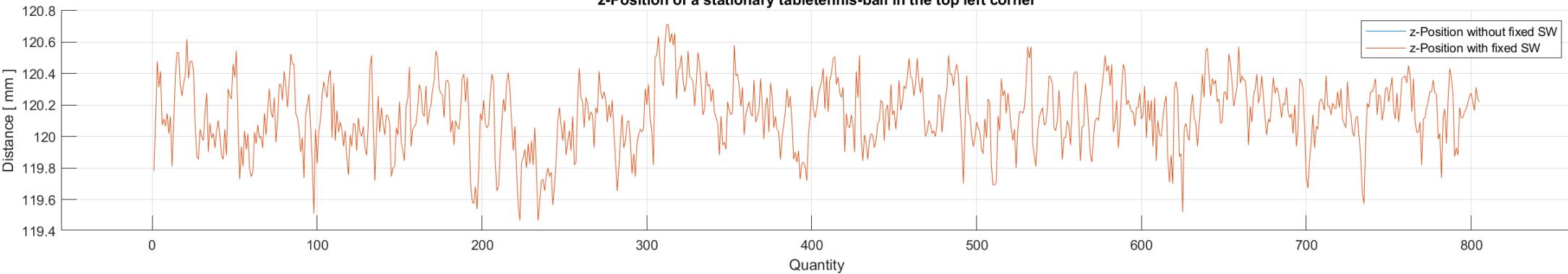


Abbildung 30: Genauigkeit der x , y und z -Position vor und nachdem die Fehler behoben worden sind

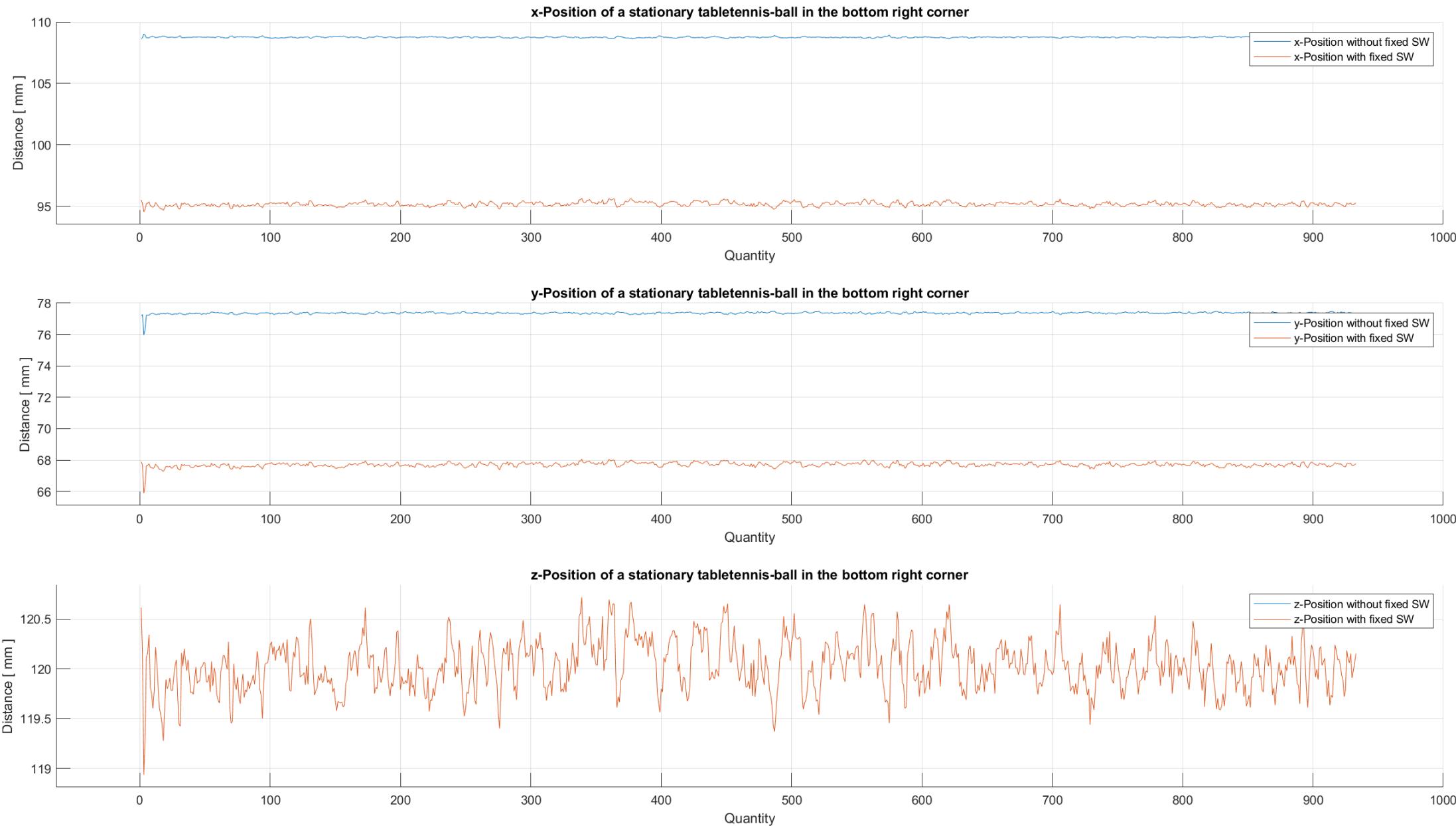


Abbildung 31: Genauigkeit der x , y und z -Position vor und nachdem die Fehler behoben worden sind

6 Mathematische Herleitungen

6.1 Moment

Die Verteilung um einen Punkt wird Moment genannt. Bei der OpenCV Bibliothek steht die Funktion für den Durchschnitt der Intensitäten der Pixel eines Bildes. Das Moment wird mit Formel 6.6 [11] berechnet. Dabei steht $I(x, y)$ für die Intensität des Pixels an Stelle (x, y) . Das Zentrum kann durch Formel 6.7 [11] berechnet werden.

$$M_{p,q} = \sum_x \sum_y (x - \bar{x})^p \cdot (y - \bar{y})^q \cdot I(x, y) \quad (6.6)$$

$$\begin{aligned} x &= \frac{M_{1,0}}{M_{0,0}} \\ y &= \frac{M_{0,1}}{M_{0,0}} \end{aligned} \quad (6.7)$$

6.2 x und y -Koordinaten

Dieses Kapitel erläutert die Berechnung der x - und y -Koordinaten in mm die für die Bedeutung der Variable $pxTomm$ wichtig ist. Das Momentum (Kapitel 6.1) gibt die Koordinaten in der Einheit \sqrt{pixel} aus. Bei Anwendung des Dreisatzes (Formel 6.8), bekommen wir das Ergebnis in mm , welches für die Weiterverarbeitung verwendet wird. Die Variable $Ball_{durchmesser}$ [mm] steht über die YAML-Datei zur Verfügung. Für die Berechnung werden Beispielwerte angenommen.

$$\begin{aligned} Ball_{durchmesser_{mm}} &= 35 \text{ [mm]} \\ Ball_{durchmesser_{\sqrt{\text{pixel}}}} &= 2 \cdot \sqrt{\frac{A_{ball}}{\pi}} \text{ [\sqrt{\text{pixel}}]} \\ Ball_{durchmesser_{\sqrt{\text{pixel}}}} &= 100 \text{ [\sqrt{\text{pixel}}]} \end{aligned}$$

$$Ball_{durchmesser_{mm}} = Ball_{durchmesser_{\sqrt{\text{pixel}}}}$$

$$\begin{aligned} 35 \text{ [mm]} &= 100 \text{ [\sqrt{\text{pixel}}]} \\ \frac{35}{100} &= 1 \text{ [\sqrt{\text{pixel}}]} \\ 0,35 \text{ [mm]} &= 1 \text{ [\sqrt{\text{pixel}}]} \end{aligned}$$

$$pxTomm = 0,35 \tag{6.8}$$

$$\begin{aligned} x &= 56 \text{ [\sqrt{\text{pixel}}]} \\ y &= 23 \text{ [\sqrt{\text{pixel}}]} \end{aligned}$$

$$\begin{aligned} x &= 56 \text{ [\sqrt{\text{pixel}}]} \cdot pxTomm \\ y &= 23 \text{ [\sqrt{\text{pixel}}]} \cdot pxTomm \\ x &= 56 \text{ [\sqrt{\text{pixel}}]} \cdot 0,35 \\ y &= 23 \text{ [\sqrt{\text{pixel}}]} \cdot 0,35 \\ x &= 19,60 \text{ [mm]} \\ y &= 8,05 \text{ [mm]} \end{aligned}$$

6.3 Winkelberechnung

Dieses Kapitel erklärt, wie aus Beschleunigungs- und Winkelgeschwindigkeiten der Neigungswinkel berechnet werden kann. Dies geschieht anhand eines Komplementärfilters. Ein Komplementärfilter verwendet unterschiedliche Sensordaten und verrechnet diese (Sensorfusion), um den Neigungswinkel zu berechnen. Um eine höhere Genauigkeit zu erreichen werden die unterschiedlichen Sensordaten unterschiedlich gewichtet. Abbildung 32 zeigt die Berechnung des Winkels in einer Achse. In Teilbereich A (Abbildung 32) kommen die Rohwerte vom Sensor an. Für die anschließende Berechnung des Winkels in y -Richtung ist nur die Beschleunigung X und Z und die Winkelgeschwindigkeit der y -Achse notwendig. Diese Rohdaten werden für die Weiterverarbeitung zuerst mit einem

Offset subtrahiert, um einen stabilen Nullpunkt zu erreichen. Um aus Beschleunigungsdaten den Winkel zu berechnen, wird Formel 6.9 angewandt [25]. Abbildung 33 zeigt dies grafisch. Anstatt des Arkustangens, wird in Teilbereich *B* (Abbildung 32) der Arkustangens zwei verwendet. Dieser gibt Werte zwischen $\pm 90^\circ$ aus, welches für die weitere Berechnung von Vorteil sind. Danach wird das Ergebnis noch in Grad umgewandelt. In Teilbereich *D* (Abbildung 32) wird eine Skalierung des Messwertes $gyroY$ vorgenommen. Dieser Wert entspricht der Empfindlichkeit des Gyroskop. Die *Y*-Messwerte vom Gyroskop müssen schrittweise auf integriert werden. Dies wird erreicht, indem wir vorher eine Skalierung durchführen (Teilbereich *E* in Abbildung 32) und danach mit dem alten Messwert (Teilbereich *F* in Abbildung 32) addieren. Der Wert $0,01\ s$ in Teilbereich *E* (Abbildung 32) entspricht $10\ [ms] = 0,01\ [s]$ – hier bekommen wir alle $10\ ms$ einen neuen Messwert. Teilbereich *C* (Abbildung 32) sorgt dafür, dass die beiden Ergebnisse aus Teilbereich *B* und *E*, *F* (Abbildung 32) unterschiedlich gewichtet werden. Dadurch erhalten wir einen Winkel, der sich zu 90% vom Gyroskop-Messwert und nur 10% vom Beschleunigungswert orientiert.

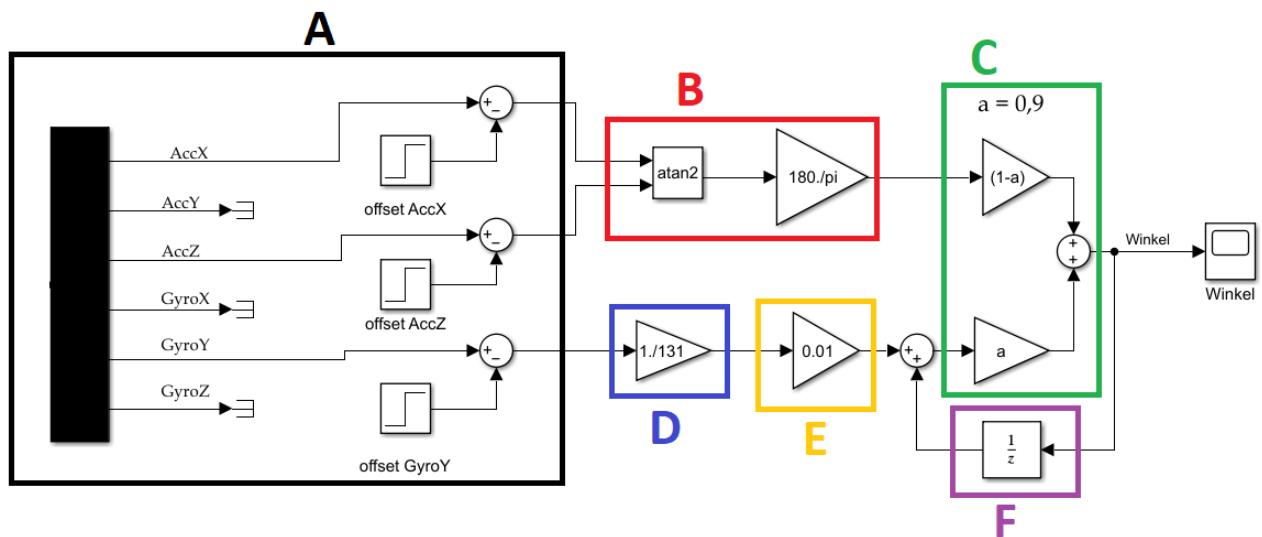


Abbildung 32: Simulinkmodell der Winkelberechnung in der *y*-Achse

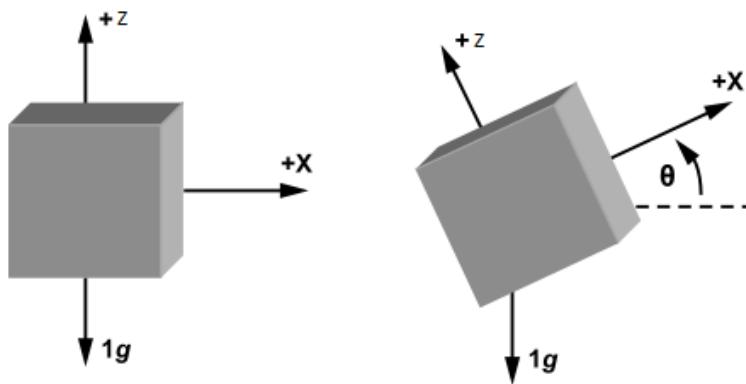


Abbildung 33: Skizze der Winkelveränderung für Formel 6.9 [26]

$$g = 9,81 \left[\frac{m}{s^2} \right]$$

$$\frac{AccX}{AccZ} = \frac{1 \cdot g \cdot \sin(\phi)}{1 \cdot g \cdot \cos(\phi)} = \tan(\phi) \quad (6.9)$$

$$\phi = \arctan\left(\frac{AccX}{AccZ}\right)$$

6.4 Hebelarm Übersetzung

Die Übersetzung des Hebelarms kann mittels Dreieckberechnung (Formeln 6.10) bestimmt werden. Die Längen L_1 , L_2 , L_3 können der Abbildung 34 entnommen werden. Der Winkel α (Motorwinkel) hat als Beispiel 10° . Winkel γ ist der Winkel der Acrylglasplatte. Anhand Abbildung 34 können die mathematischen Schritte verfolgt werden. Diese sind

1. Y-Koordinate bestimmen
2. Strecke L_x bestimmen
3. Winkel β bestimmen
4. Winkel T_β bestimmen nach dem Kosinussatz
5. Ausgangswinkel γ bestimmen

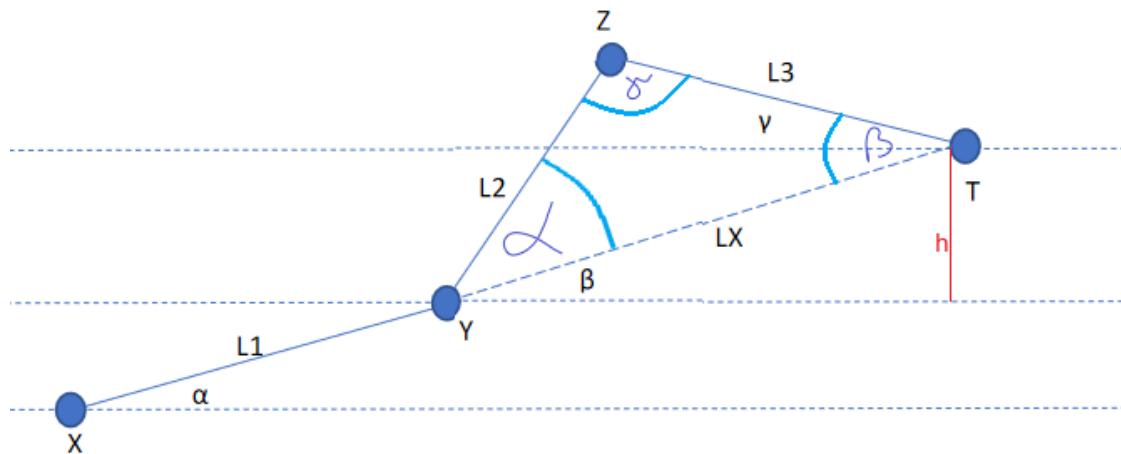


Abbildung 34: Skizze der Übersetzung des Hebelarms

$$L_1 = 9 \text{ cm} \quad L_2 = 8 \text{ cm} \quad L_3 = 13,3 \text{ cm} \quad \alpha = 10^\circ$$

1.

$$Y_x = r \cdot \cos(\alpha)$$

$$Y_y = r \cdot \sin(\alpha)$$

$$Y_x = 9 \text{ cm} \cdot \cos(10^\circ) = 8,8632 \text{ cm}$$

$$Y_y = 9 \text{ cm} \cdot \sin(10^\circ) = 1,5628 \text{ cm}$$

2.

$$T_x = L_1 + L_3 = 22,3 \text{ cm}$$

$$T_y = L_2 = 8 \text{ cm}$$

$$L_x = \sqrt{(T_x - Y_x)^2 + (T_y - Y_y)^2}$$

$$L_x = \sqrt{(22,3 \text{ cm} - 8,8632 \text{ cm})^2 + (8 \text{ cm} - 1,5628 \text{ cm})^2} = 14,8991 \text{ cm}$$

3.

$$h = T_y - Y_y$$

$$h = 8 \text{ cm} - 1,5628 \text{ cm} = 6,4372 \text{ cm} \quad (6.10)$$

$$\beta = \arcsin\left(\frac{h}{L_x}\right)$$

$$\beta = \arcsin\left(\frac{6,4372 \text{ cm}}{14,8991 \text{ cm}}\right) = 25,5978^\circ$$

4.

$$L_2^2 = L_3^2 + L_x^2 - 2 \cdot L_3 \cdot L_x \cdot \cos(T_\beta)$$

$$T_\beta = \arccos\left(\frac{L_2^2 - L_3^2 - L_x^2}{-2 \cdot L_3 \cdot L_x}\right)$$

$$T_\beta = \arccos\left(\frac{(8 \text{ cm})^2 - (13,3 \text{ cm})^2 - (14,8991 \text{ cm})^2}{-2 \cdot (13,3 \text{ cm})^2 \cdot (14,8991 \text{ cm})^2}\right)$$

$$T_\beta = \arccos(0,8450)$$

$$T_\beta = 32,3318^\circ$$

5.

$$\gamma = T_\beta - \beta$$

$$\gamma = 32,3318^\circ - 25,5978^\circ$$

$$\gamma = 6,7340^\circ$$

Führt man den Rechenweg in 6.10 für $\alpha = -180^\circ$ bis 180° aus, erhält man Abbildung

35. Die Abbildung 35 verdeutlicht in Abhängigkeit von Winkel α den Winkel γ . Es ist zu erkennen, dass kein linearer Zusammenhang zwischen beiden Winkeln besteht. Vielmehr kommt es ab einem bestimmten Winkel α zu komplexen Winkeln γ . Das ist darauf zurückzuführen, dass die Längen L_1, L_2, L_3 (Abbildung 34) starr sind und keine "Dehnung" zulassen. Da die Acrylglasplatte einen maximalen Neigungswinkel von $\gamma = \pm 20^\circ$ besitzt, vereinfacht sich die Messwertkurve (Abbildung 36a) zu einer scheinbar einfachen Messwertkurve (Abbildung 36b).

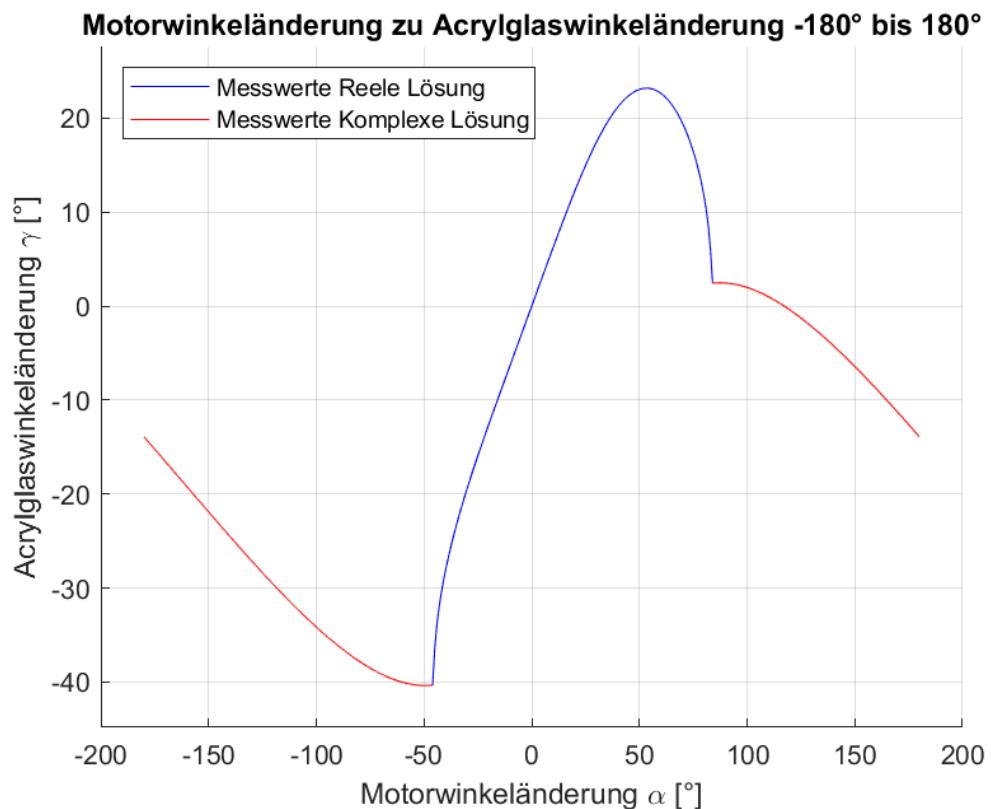


Abbildung 35: Messwertkurve von -180° bis 180°

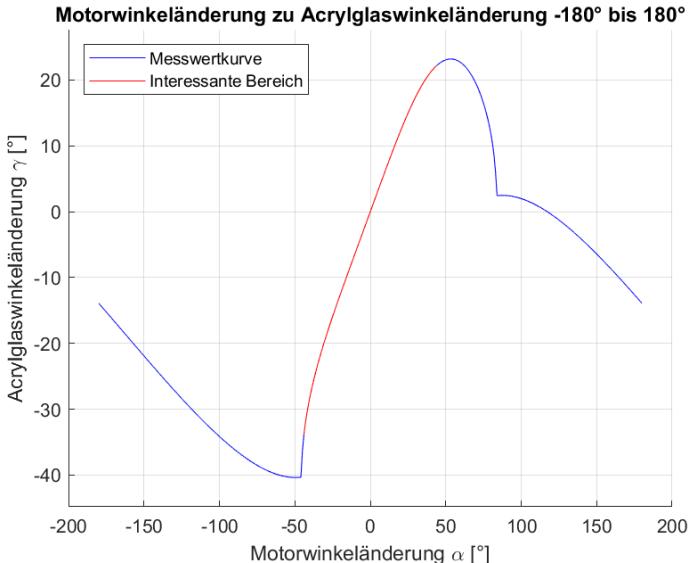
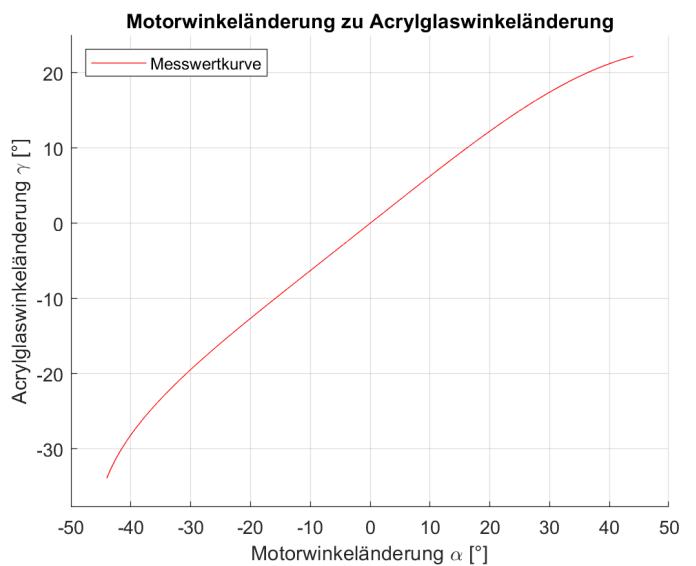
(a) Messwertkurve von -180° bis 180° (b) Messwertkurve für $\gamma = -20^\circ$ bis 20°

Abbildung 36: Interessanter Funktionsabschnitt

6.5 Newton Verfahren

Dieses Kapitel behandelt die Methode, wie aus der Messwertkurve (Abbildung 36b) eine Polynomfunktion bestimmt werden kann. Weiterhin soll aus einem gegebenen Winkel γ der dazugehörige Winkel α bestimmt werden. Das Newton Verfahren ist ein numerisches Verfahren für das Finden einer Nullstelle. Mithilfe der internen Matlabfunktionen (*polyfit* und *polyval*) kann die Messwertkurve (Abbildung 36b) in eine Polynomfunktion beliebigen Grades zerlegt werden. Dabei zeichnet sich ab, dass je höher der Grad ist, desto genauer wird die Messwertkurve repräsentiert. Allerdings muss ein Kompromiss zwischen der Genauigkeit und der Anwendung des Newton Verfahrens gefunden werden, da mit steigendem Grad die Anwendung des Newton Verfahrens den Zeitbedarf erhöht. Abbildung 37 vergleicht verschiedene Polynomfunktionen unterschiedlichen Grades miteinander. Dabei schneidet die Funktion mit Grad sieben am Besten ab. Die Funktion vom Grad sieben ist ein guter Kompromiss aus Genauigkeit und dem daraus resultierenden Zeitbedarf des Newton Verfahrens. Aus diesem Grund wird die Funktion vom Grad sieben für die weitere Verarbeitung verwendet. Dazu muss Gleichung 6.11 zu einem Nullstellenproblem umgeformt werden. Zusammen mit der ersten Ableitung von Funktion 6.11 kann die Newtonfunktionsvorschrift $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ angewandt werden. Als Startwert wird $x_0 = 0$ gesetzt. Die Praxis hat gezeigt, dass zehn Iterationen für eine Genauigkeit auf zwei Nachkommastellen und für die Einhaltung des Timing ausreichen.

$$\gamma = f(\alpha) = \frac{1,5896}{10^{11}} \cdot \alpha^7 - \frac{-4,7403}{10^{10}} \cdot \alpha^6 + \frac{-2,7900}{10^8} \cdot \alpha^5 + \frac{-3,9369}{10^7} \cdot \alpha^4 + \frac{-1,8075}{10^6} \cdot \alpha^3 + \frac{-3,7522}{10^4} \cdot \alpha^2 + 0,6266 \cdot \alpha + 0,0233 \quad (6.11)$$

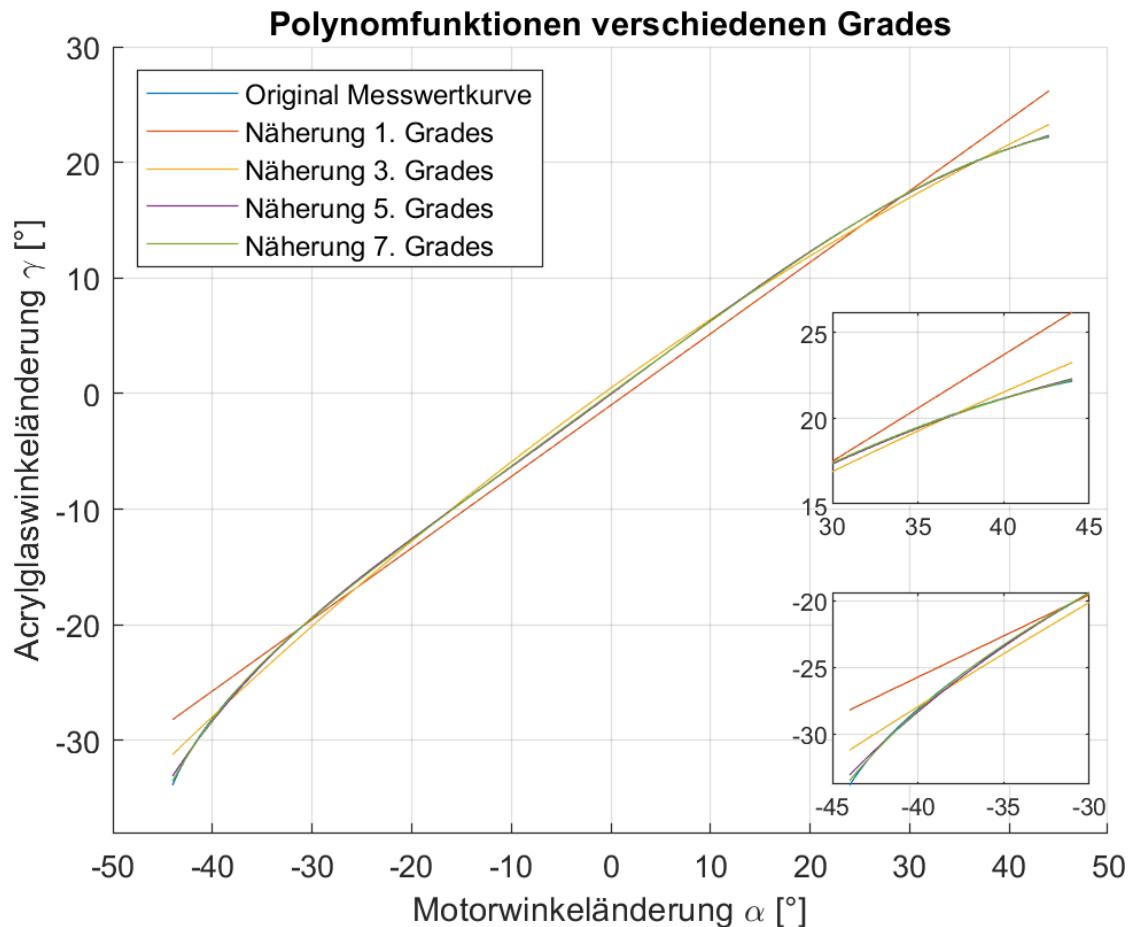


Abbildung 37: Vergleich verschiedener Polynomfunktionen

6.6 Modellierung des Systems

Der Tischtennisball soll in eine bestimmte Position gebracht werden. Dafür können wir die Acrylglasplatte in x und y Richtung neigen. Somit kann das System als Schiefe Ebene betrachtet werden, bei dem der Tischtennisball als Körper fungiert (Abbildung 38). Die Masse des Körpers beträgt 2,7 g. Daraus resultieren die Formeln:

$$\begin{aligned}
 F_G &= m \cdot a \\
 F_A &= m \cdot g \cdot \sin(\alpha) \\
 F_A &= F_G \\
 m \cdot g \cdot \sin(\alpha) &= m \cdot a
 \end{aligned} \tag{6.12}$$

Die Beschleunigung von F_A kann umgeschrieben werden zu der zweiten Ableitung des Weges ($a = \ddot{x}$). Ist der Winkel α klein genug, kann der Sinus vernachlässigt werden, weil für kleine Winkel gilt: $\sin(\alpha) = \alpha$. Daraus ergibt sich die resultierende Formel:

$$\begin{aligned}
 \ddot{x} &= \sin(\alpha) \cdot g \\
 \ddot{x} &= \sin(\alpha) \cdot (-9,81 \frac{m}{s^2}) \\
 x &= \int \int \sin(\alpha) \cdot (-9,81 \frac{m}{s^2})
 \end{aligned} \tag{6.13}$$

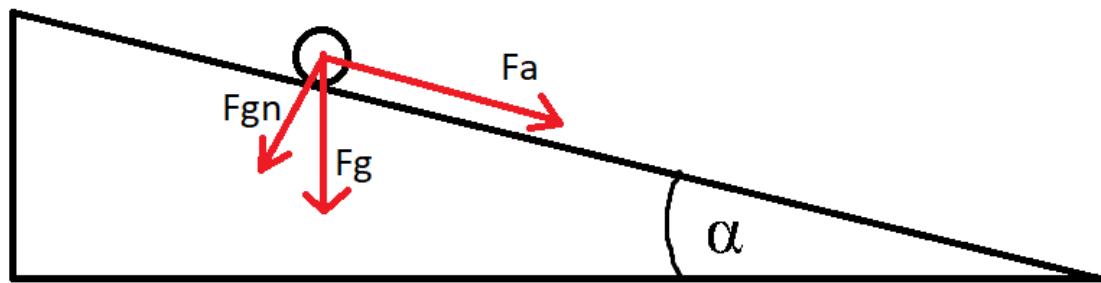


Abbildung 38: Skizze der Modellierung

7 Probleme

Dieser Abschnitt wendet sich den Problemen zu, die erst bei der Durchführung dieser Arbeit aufgetreten sind und vorher nicht abzuschätzen waren.

7.1 Herstellungszeit der Mechanik

Im Rahmen der Masterarbeit wurde öfters eine mechanische Konstruktion in Auftrag gegeben. Der Zeitbedarf für die Fertigung nahm aufgrund von Verzögerungen seitens der Zentralwerkstatt jeweils mehrere Wochen in Anspruch. Der Zeitplan konnte daraufhin oft nicht eingehalten werden, was dazu führte, das davon abhängige Komponenten nur mit enormer Verzögerung entwickelt werden konnten.

8 Verbesserungen

Im Folgenden werden Verbesserungsvorschläge für die einzelnen Module vorgestellt.

8.1 Mechanischer Aufbau

Bei der Mechanik kann der Hebelarm *Z1* verlängert werden. Dadurch kann mit geringen Schritten eine stärkere Neigung der Acrylglasplatte gewonnen werden. Weiterhin können die geometrischen Formen der Hebelarme verändert werden, sodass eine höhere Gewichtsreduktion erfolgt. Für eine höhere Schrittzahl kann ein Brushless-DC-Motor verwendet werden; allerdings wird dann der Neigungswinkel der Acrylglasplatte über das Moment des Motors gesteuert. Weitere Optimierungen der Hebelarme müssten durch mechanische Berechnungen erfolgen, die an dieser Stelle nicht geleistet werden können.

8.2 Optische Positionsbestimmung

Für die Detektion könnte anstatt einer handelsüblichen Farbkamera, eine Infrarotkamera verwendet werden. Der Tischtennisball müsste dafür Infrarotlicht reflektieren. Dadurch erreicht man einen höheren Kontrast zwischen der Umgebung und dem Tischtennisball, unabhängig von den vorherrschenden Lichtverhältnissen. Weiterhin kann anstatt eines Raspberry Pi 3 ein *x86* Computer verwendet werden. Dieser wäre leistungsstärker und man könnte damit die Positionserfassung und den Regler in einem Teilbereich zusammenfassen. Weiterhin ist es möglich, anstatt einer optischen Erfassung, ein Touchscreen mit einem Metallball zu verwenden. Damit kann die Genauigkeit gegenüber einer anfälligen optischen Positionerkennung erhöht werden. Des weiteren könnte auch ein neuronales Netzwerk zur Anwendung kommen, wenn zuvor untersucht wird, ob es den Anforderungen entspricht.

8.3 Motorsoftware

Die Motorsoftware wird auf einem Mikrocontroller ausgeführt. Dieser ist entsprechend langsam – verglichen mit herkömmlichen Computern. Vereint man die Motorsoftware und den Regler auf einer CPU, können die langsamen Übertragungsraten von *UART* vermieden werden. Weiterhin kann anstelle einer Dreiecksfahrt mit einer linearen Beschleunigung, eine *S*-Kurve für die Anfahrt genommen werden.

9 Erreichter Stand

Dieses Kapitel widmet sich der Verifizierung des Pflichtenhefts (Liste 4).

9.1 Pflichtenheft-Kriterien

Das System soll innerhalb von 500 ms eine neue Winkelposition angefahren haben

Diese Anforderung wird durch Abbildung 18 verifiziert. Diese Abbildung gilt nur für eine Achse. Da allerdings beide Achsen (und somit die Motoren) gleich angesteuert werden, besteht kein Zweifel daran, dass sich für eine andere Achse der Zeitwert von 484 ms verändert. Darüber hinaus kann die Beschleunigung und die Startgeschwindigkeit der Dreiecksbewegung, durch einen Eingriff in den Quellcode angepasst werden.

Das System soll zwei Achsen parallel anfahren

Echte Parallelität kann nur gewährleistet werden, indem ein Prozessor verwendet wird, der Multiprocessing unterstützt. Der verwendete Mikrocontroller unterstützt dies nicht. Allerdings startet der Controller die Bewegung für beide Achsen direkt nacheinander. Die Zeitdifferenz beträgt weniger als 1 ms. Als außenstehender Beobachter kann aufgrund der kurzen Zeitspanne nicht beobachtet werden, ob eine Achse nach der anderen startet. Somit ist dieser Punkt als erfüllt anzusehen.

Das System soll 0.5° Toleranz maximal für eine Winkelposition aufweisen

Um dieses Kriterium zu überprüfen, wurde mehrmals eine unterschiedliche Winkelposition angefahren. Die angefahrene Winkelposition waren 7.5° und -7.5° . Abbildung 39 zeigt die Messwertkurve mit einem Toleranzschlauch. Nur während einer Bewegung wird dieser Toleranzschlauch verletzt ansonsten liegen alle Messwerte innerhalb des Toleranzschlauchs. Somit ist dieses Kriterium erfüllt.

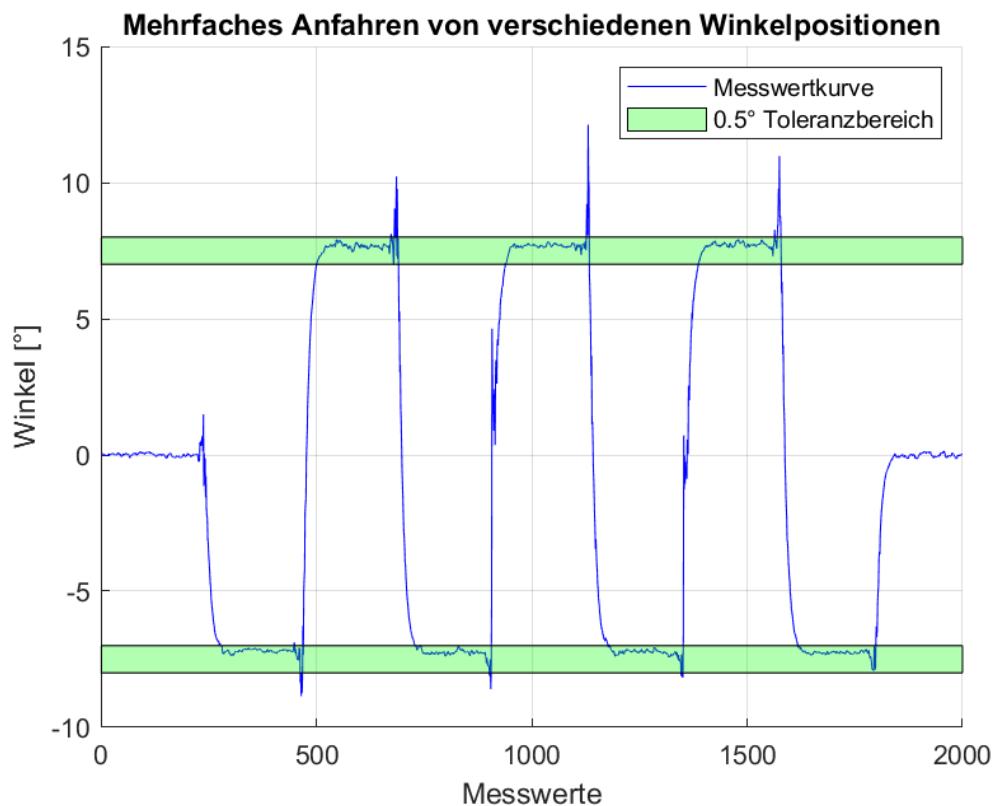


Abbildung 39: Anfahren verschiedener Winkelpositionen

Das System soll die drei Koordinaten des Tischtennisballs ausgeben

Im Zuge der Arbeit wurde auf eine Erweiterbarkeit des Systems Wert gelegt, so dass die Berechnung der z -Achse mit in die Software integriert wurde. Dies hat den Vorteil, dass bei einer Version 2 die Bildverarbeitung als fertige Komponente übernommen werden kann. Es liegt am Regler, ob die Position der z - Achse interpretiert wird.

9.2 Wunsch-Kriterien

Das System sollte eine Toleranz der Detektion des Tischtennisballs von 5 mm aufweisen

Dieses Wunschkriterium wird in Kapitel 5.6 näher erläutert. Dort wird ebenfalls erwähnt, dass dieses Wunschkriterium erfüllt wird.

10 Ausblick

Im Folgenden werden die Schritte erläutert, die für ein weiteres Vorgehen notwendig sind.

10.1 Mechanischer Aufbau

Für die korrekte Funktionsweise des mechanischen Aufbaus müssen noch zwei weitere Hebelarme, zwei Schrittmotoren, vier Motortreiber und eine Acrylglasplatte in der entsprechenden Größe in Auftrag gegeben oder bestellt werden. Danach ist es erforderlich, dass die Motoren und die Kamerahalterung auf einer bereits vorhandenen Holzplatte montiert werden.

10.2 Optische Positionsbestimmung

Nachdem die mechanische Installation abgeschlossen ist, muss eine neue Konfigurationsdatei durch die Parametersoftware erstellt werden. Danach sollte die eigentliche Software für die Positionsbestimmung noch in den Autostarter hinzugefügt werden. Die entsprechenden *UART*-Pins des Raspberry Pi 3 müssen mit dem Controller, auf dem die Reglersoftware ausgeführt wird, verbunden werden.

10.3 Positionsregelung

Der vorhandene Regler muss noch in Simulink mit den Ein- und Ausgabekomponenten verbunden werden. Darüber hinaus ist eine Exportierung auf einen eigenen Controller notwendig, denn Simulink ist nicht Echtzeitfähig.

10.4 Motorsoftware

Die Software muss noch ein Softwaretestverfahren durchlaufen, um unentdeckte Bugs zu entdecken und zu entfernen. Der integrierte Watchdog muss einkommentiert werden.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 24.03.2022

Oliver Koepp

Literaturverzeichnis

- [1] *DIGALOG Industrie-Mikroelektronik GmbH* - URL <https://www.digalog.de/> - abgerufen am 07.02.2022
- [2] *YouTube Video als Vorbild* - URL <https://www.youtube.com/watch?v=1YyAMDYzJQM> - abgerufen am 18.10.2021
- [3] *NUCLEO-L476RG* - URL <https://www.st.com/en/evaluation-tools/nucleo-l476rg.html> - abgerufen am 18.09.2021
- [4] *Raspberry Pi* - URL <https://www.raspberrypi.org/> - abgerufen am 18.09.2021
- [5] *OpenCV 4.5.2 Lizenz* - URL <https://github.com/opencv/opencv/blob/master/LICENSE> - abgerufen am 18.09.2021
- [6] *HTW Zentralwerkstatt* - URL <https://www.htw-berlin.de/einrichtungen/zentralwerkstatt/> - abgerufen am 18.09.2021
- [7] *NVIDIA Jetson TX1 Developer Kit* - URL <https://developer.nvidia.com/embedded/jetson-tx1-developer-kit> - abgerufen am 18.10.2021
- [8] *NEMA 23 Datenblatt* - URL <https://de.nanotec.com/fileadmin/files/Datenblaetter/Schrittmotoren/ST5918/X/ST5918X2008-A.pdf> - abgerufen am 22.10.2021
- [9] *Schrittmotortreiber; Höhere Spannung* - URL <https://forum.arduino.cc/t stepper-motor-basics/275223> - abgerufen am 22.10.2021
- [10] *Schrittmotortreiber DMT542T* - URL <https://www.omc-stepperonline.com/de/digitaler-schrittmotortreiber-1-0-4-2a-20-50vdc-fur-nema-17-23-24-schrittmotor.html> - abgerufen am 23.10.2021
- [11] *Formel für das Moment* - URL <https://www.pythontoolbox.com/opencv-moments/> - abgerufen am 29.10.2021
- [12] *200 FPS Bildaufnahme* - URL <https://forums.raspberrypi.com/viewtopic.php?t=206047> - abgerufen am 23.10.2021
- [13] *Radiale- /Tangentiale-Verzerrung* - URL <https://upload.wikimedia.org/wikipedia/commons/0/0a/Verzeichnung3.png> - abgerufen am 15.10.2021
- [14] *Evaluationboard EVAL6480H* - URL <https://www.st.com/en/evaluation-tools/eval6480h.html> - abgerufen am 02.02.2022

- [15] *DRV8825 Stepper Motor Driver Carrier* - URL <https://www.pololu.com/product/2133> - abgerufen am 08.03.2022
- [16] *Sensor MPU6050* - URL <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/> - abgerufen am 11.01.2022
- [17] *Performance des NUCLEO-L476RG bei OpenCV* - URL <https://www.embedded.com/benchmarking-opencv-on-stm32-mcus/> - abgerufen am 19.10.2021
- [18] *Bildverarbeitung auf Mikrocontrollern* - URL <https://stackoverflow.com/questions/5102484/image-processing-in-microcontroller/5113743> - abgerufen am 19.10.2021
- [19] *Datenblatt des L6480; UVLO Schutz, Seite 30* - URL <https://www.st.com/resource/en/datasheet/l6480.pdf> - abgerufen am 26.10.2021
- [20] *Datenblatt des L6480; Formel $\Delta V_{BOOTthOff}$; Seite 12* - URL <https://www.st.com/resource/en/datasheet/l6480.pdf> - abgerufen am 26.10.2021
- [21] *Frage im Community Forum von STMicroelectronics* - URL <https://community.st.com/s/question/0D53W000019FnKpSAK/16480-uvlobit-problem-cant-get-it-flip-to-1> - abgerufen am 26.10.2021
- [22] *Github Link zu der verwendeten MPU6050 Bibliothek* - URL https://github.com/sinadarvi/SD_HAL_MPU6050 - abgerufen am 1.1.2022
- [23] *Datenblatt des Motortreibers DM542T* - URL <https://www.omc-stepperonline.com/download/DM542T.pdf> - abgerufen am 09.1.2022
- [24] *Morphologische Transformationen (Erosion und Dilatation)* - URL https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html - abgerufen am 22.1.2022
- [25] *Formel für die Berechnung des Winkel* - URL <https://www.analog.com/en/app-notes/an-1057.html> - abgerufen am 28.12.2021
- [26] *Skizze für die Winkelberechnung 6.9* - URL <https://www.analog.com/en/app-notes/an-1057.html> - abgerufen am 28.12.2021