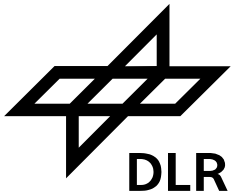

Studienarbeit

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

Thema

Entwicklung eines Systems zur Verfolgung von Objekten mit Hilfe einer Kamera

---

Julius Klodt

19. April 2022

Bearbeitungszeitraum	01.10.2021 - 19.04.2022
Matrikelnummer, Kurs	8431855, TINF19IT1
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer	Jürgen Schultheis
Unterschrift des Betreuers	

Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Studienarbeit mit dem Thema *Entwicklung eines Systems zur Verfolgung von Objekten mit Hilfe einer Kamera* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, 18. April 2022

_____

Julius Klodt

Zusammenfassung

Die vorliegende Studienarbeit befasst sich mit der Entwicklung eines Systems, um eine Kamera so zu bewegen, dass diese ein bewegendes Gesicht im Bild verfolgt. Bei der zu bewegenden Kamera handelt es sich um eine 1.1 kg schwere Kamera-Objektiv Kombination. Die Bewegung der Kamera wird durch eine Pan- und Tilt-Drehung, sowie einer Bewegung im Raum auf einem Slider durchgeführt. Die benötigte Hardware wird konstruiert und mit Hilfe von 3D-Druckern ausgedruckt. Die Hardwareansteuerung erfolgt mit einem Arduino Nano. Die für die Verfolgung benötigte Gesichtserkennung wird mit Hilfe der Software-Bibliothek OpenCV realisiert und wird auf einem Raspberry Pi ausgeführt. Dieser Kommuniziert über eine serielle Schnittstelle mit dem Arduino Nano. Es wird gezeigt, dass das entwickelte System funktionsunfähig und in der Lage ist Gesichter zu verfolgen. Die Verfolgung ist dabei jedoch ruckelig und könnte in einem nächsten Schritt verbessert werden.

Abstract

This thesis deals with the development of a system which can move a camera in a way to track moving faces. The weight of the used camera with the lens is 1.1 kg. The system can exectue a pan and tilt rotation. Additionaly the system can move the camera on a slider. The hardware is engineered and printed with a 3D-printer. An Arduino Nano is used for the hardware-control. An Raspberry Pi is used for realizing the face detection with the software library OpenCV. Both devices are comunicating with each other over a serial interface. It is shown that the system is full functioning and capable of tracking faces. The tracking it self is a little bit jerky and can be improved in a next step.

Inhaltsverzeichnis

Abkürzungsverzeichnis

DHBW	**D**uale **H**ochschule **B**aden-**W**ürttemberg
IDE	**I**ntegrated **D**evelopment **E**environment
CAD	**C**omputer **A**ided **D**esign
PLA	**P**oly**l**actic **a**cid

Abbildungsverzeichnis

1. Einleitung

In Zeiten der Covid-19-Pandemie wurden Videokonferenzen immer populärer. In Schulen, bei der Arbeit, beim Studium und selbst beim Sport werden Videokonferenzen mittlerweile genutzt. Dabei kommt es nicht selten vor, dass sich die Position einer Person verändert und nur noch schlecht im Bild zu sehen ist. Bei Verwendung mehrerer Bildschirme entsteht zusätzlich das Problem, dass je nach dem auf welchen Bildschirm geguckt wird, nur das Profil und nicht die Frontalansicht des Gesichts zu sehen ist.
Im Rahmen einer Studienarbeit der **D**uale **H**ochschule **B**aden-**W**ürttemberg (DHBW) Mannheim ist daraus die Idee entstanden, ein System zu entwickeln, welches die genutzte Kamera automatisch so bewegt, dass die gefilmte Person immer mittig und frontal im Bild zu sehen ist.
Dabei soll sowohl Hardware als auch Software erstellt werden. Um die benötigte Präzision der Hardware zu gewährleisten kam die Idee, das System 3D zu drucken. Alle Komponenten sollen dafür modelliert und gedruckt werden. Die Software soll die Motor und Peripherie Ansteuerung beinhalten. Des Weiteren eine Gesichtserkennung und ein Algorithmus zur Weiterverarbeitung der Gesichtsposition zu Motorbewegungen.

2. Aufgabenstellung

Die Studienarbeit wurde in folgende Aufgaben unterteilt:

Aufgabe 1: Hardware

1.1. Präzise Anforderungen an die Hardware stellen

1.2. Vorhandene Projekte sichten und Umsetzungsideen sammeln

1.3. Konzepte erstellen und Hardware konzipieren

1.4. Hardware konstruieren

1.5. Benötigte Hardware einkaufen

1.6. Hardware drucken und zusammenbauen

Aufgabe 2: Software

2.1. Anforderungen an die Software stellen

2.2. Hardwareansteuerung implementieren

2.3. Gesichtserkennung implementieren

2.4. Algorithmus zur Verarbeitung der Position des Gesichts zu Motorbewegungen

Aufgabe 3: Testen

3.1. Hardware auf Anforderungen testen

3.2. Software auf Anforderungen testen

3. Methoden und Verfahren

In diesem Kapitel werden die zugrunde liegenden Theoretischen Kenntnisse, die Methodik und Verfahrensgrundlage erörtert. Zu dem wird erklärt, warum diese genutzt wurden.

3.1. C

Die Hardwareansteuerung erfolgt mit der Programmiersprache C, welche nativ auf dem verwendeten *Arduino nano* unterstützt wird. C wurde ursprünglich von Dennis Ritchie in den frühen 1970er Jahren an den Bell Laboratories entwickelt und basiert auf der von Ritchie und Ken Thompson entwickelten Programmiersprache B [8].
Die Möglichkeit auf Hardwarekomponenten direkt zuzugreifen, macht C sehr hardwarenah [3]. Dies bringt auch den Vorteil mit sich, dass C sehr schnell ist. Um mit zu programmieren und den fertigen Code auf den Ardunio nano zu übertragen wurde die **I**ntegrated **D**evelopment **E**environment (IDE) Arduino-Code mit der Version 1.8.13 verwendet.

3.2. Python

Für die Gesichtserkennung und die Weiterverarbeitung der Gesichtspositionsdaten wird die Programmiersprache Python verwendet. Python ist eine objektorientierte, höhere Programmiersprache [6] und wird häufig für Anwendungen im Bereich des Maschinellen Lernens verwendet.
Im Rahmen dieser Arbeit wird die Python Version 3.9.7 verwendet.

3.2.1. OpenCV

OpenCV ist eine Open-Source Bibliothek welche für digitale Bildverarbeitung genutzt wird und in den Programmiersprachen Java, C, C++ und Python verwendet werden kann [**OpenCV**]. Die OpenCV Bibliothek bietet unter anderem Algorithmen zur Gesichtserkennung an und wird daher für diese Arbeit mit der Version 4.5.5 genutzt.

3.3. FreeCAD

FreeCAD ist ein Open-Source **C**omputer **A**ided **D**esign (CAD)-Programm. Es wird für die Konstruktion und Modellierung von Objekten genutzt [2]. In Rahmen dieser Arbeit wird FreeCAD mit der Versionsnummer 0.19 genutzt, um die benötigten Teile für das Verfolgungssystem als 3D-Modelle zu konstruieren.

3.4. Ultimaker Cura

"Ultimaker Cura ist eine Open-Source Slicer-Software zur Umwandlung von 3D Modellen in G-Code (Druckerbefehle für 3D-Drucker)"[11].
Für diese Arbeit wurde Ultimaker Cura mit der Version 4.12.1 genutzt. Das Programm bietet vielseitige Möglichkeiten an, wie ein 3D-Modell gedruckt werden soll. Zum Beispiel kann manuell eingestellt werden, wie viel Prozent des Innenraums eines 3D-Modells mit Material gefüllt werden soll. Diese Einstellung wird automatisch mit einer einstellbaren Inneren Struktur umgesetzt. Im folgenden sind zwei unterschiedliche innere Strukturen zu sehen, welche automatisch von Ultimaker Cura produziert worden sind. Beide Abbildungen zeigen die 44. Druckschicht eines Zahnrads mit 20% Füllung des Innenraums.

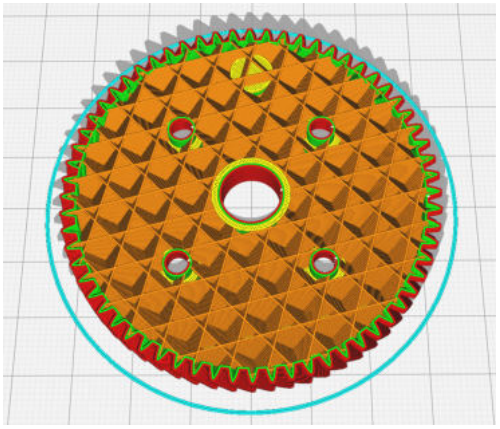

Abbildung 3.1.: "Cubic" - Strukur

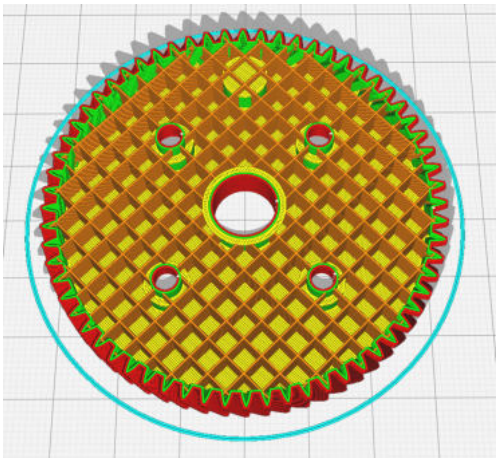

Abbildung 3.2.: "Grid" - Strukur

3.5. 3D-Drucker

3D-Drucken ist ein additives Verfahren, in welchem ein dreidimensional beweglicher Druckerkopf, Material Schicht für Schicht übereinander lagert.
Präzises, schnelles, unkompliziertes und kostengünstiges 3D-Drucken ist für den Privatanwender erst in dem letztem Jahrzehnt möglich geworden.

Im Rahmen dieser Arbeit wurden drei verschiedene 3D-Drucker genutzt, um selbst konstruierte Objekte mit dem Kunststoff **P**oly**l**actic **a**cid (PLA) zu drucken.

1. Ultimaker 2+
2. Renkforce RF2000v2
3. Creality Ender 3 V2

4. Durchführung

Dieses Kapitel erörtert die Durchführung der Aufgabenstellung. Die Strukturierung gleicht dem Aufbau der Aufgabenstellung.

- Hardware
- Software
- Testen

4.1. Hardware

In diesem Abschnitt wird beschrieben, wie die Hardware entwickelt wurde.

4.1.1. Präzise Anforderungen an die Hardware stellen

Damit das System eine Person im Bild einer Kamera verfolgen kann, muss gewährleistet sein, dass das System die Kamera drehen und neigen kann. Diese Drehbewegungen werden als Pan und Tilt bezeichnet.

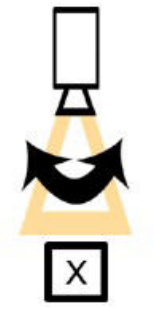

Abbildung 4.1.: Pan-Bewegung[9]

Abbildung 4.2.: Tilt-Bewegung[10]

Die Pan-Bewegung soll dabei 360° durchführbar sein damit das System möglichst vielseitig genutzt werden kann. Die Tilt-Bewegung soll aus der Horizontalen mindestens 30° nach unten und oben ausführbar sein. Sowohl die Pan, als auch die Tilt Bewegung soll eine Mindestverstellgenauigkeit von einem halben Grad aufweisen. Die Geschwindigkeit der Pan-Bewegung soll mindestens 180° pro 3 Sekunden betragen. Die Geschwindigkeit der

Tilt-Bewegung soll mindestens 90° pro 2 Sekunden betragen.

Um zu gewährleisten, dass immer die Frontalansicht einer Person zu sehen ist, muss sich die Kamera mindestens in einer Achse im Raum bewegen können. Diese Arbeit beschränkt sich dabei auf eine gerade Achse. Die Bewegung entlang der Achse soll mindestens 50 cm betragen. Die Verstellgenauigkeit soll mindestens 1 mm betragen. Die Geschwindigkeit soll mindestens 10 cm pro Sekunde betragen.

Das System soll die beschrieben Anforderungen mit der vorhanden Kamera-Objektiv Kombination Sony A7 RII und Tamron 28-75 mm F2.8 Di III RXD erfüllen. Diese Kombination wiegt zusammen ca. 1,1 kg.

Hier eine tabellarische Übersicht der Anforderungen:

Anforderungsbeschreibung	**Wert**
Pan-Bewegung, Drehbereich	$\geq 360°$
Pan-Bewegung, Verstellgenauigkeit	$\leq 0.5°$/ Schritt
Pan-Bewegung, Geschwindigkeit	$\geq 180°$/ 3 Sekunden
Tilt-Bewegung, Drehbereich aus der horizontalen	$\geq \pm 30°$
Tilt-Bewegung, Verstellgenauigkeit	$\leq 0.5°$/ Schritt
Tilt-Bewegung, Geschwindigkeit	$\geq 90°$/ 1 Sekunden
Bewegung im Raum, Strecke	$\geq$ 50cm
Bewegung im Raum, Verstellgenauigkeit	$\leq$ 1mm / Schritt
Bewegung im Raum, Geschwindigkeit	$\leq$ 10cm / Sekunde
Gewicht der zu bewegenden Kamera	$\geq$ 1.1kg

4.1.2. Vorhandene Projekte sichten und Umsetzungsideen sammeln

Um Ideen für das Projekt zu sammeln, wurde recherchiert was für Projekte schon vorhanden sind und welche hilfreich bei der Entwicklung der Hardware sein könnten. Bei der Recherche sind verschiedene Konzepte öfter aufgetaucht.

1. Einseitige Befestigung der Kamera entlang der auszuführenden Tilt-Bewegung-Achse

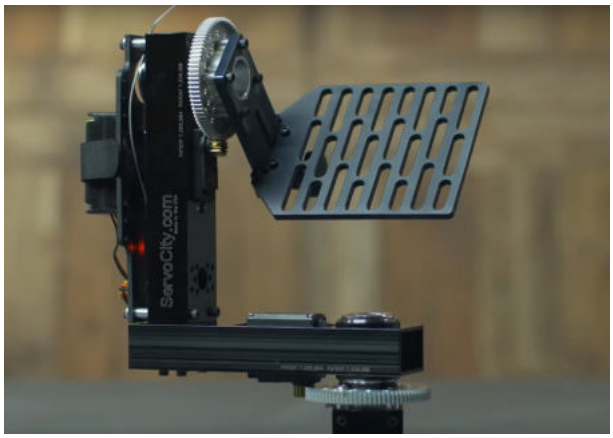

Abbildung 4.3.: Pan-Tilt Einheit der Firma ServoCity[7]

In Abbildung 4.3 ist zu sehen, dass die drehende Kamera Befestigung nur an der sich drehenden Seite befestigt ist. Dadurch wird ein starker einseitiger Druck aufgebaut, wodurch es notwendig ist, mit sehr stabilen Materialien wie Metallen zu arbeiten. Gleiches ist bei der Pan-Achse ersichtlich.

2. 3D-gedruckter Pan-Tilt-Kopf mit Differenzial Getriebe.

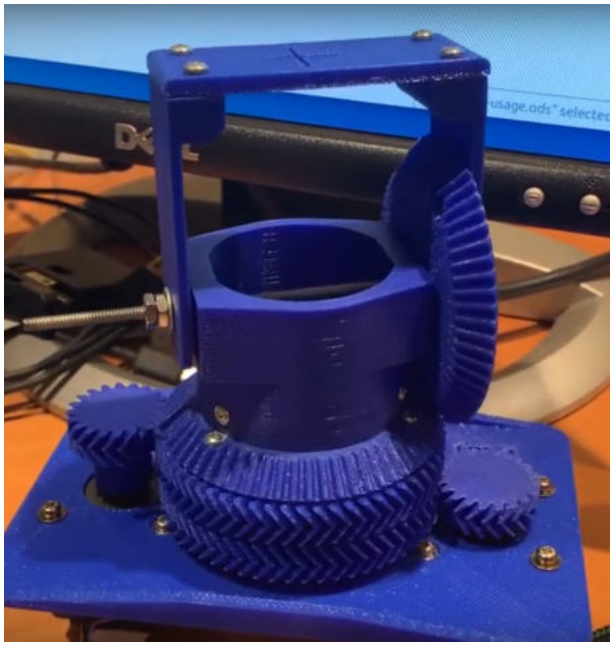

Abbildung 4.4.: 3D-gedruckter Pan-Tilt-Kopf mit Hilfe von einem Differential-Getriebe[1]

Der in Abbildung 4.4 zusehende Pan-Tilt-Kopf nutzt ein Differenzialgetriebe, um sowohl die Pan, als auch die Tilt Bewegung mit Motoren auszuführen, welche beide in der gleichen Achse montiert werden können.

3. Beidseitige Befestigung der Kamera entlang der auszuführenden Tilt-Bewegung-Achse

Abbildung 4.5.: 3D-gedruckte Pan-Tilt Einheit mit Spiegelreflexkamera[4]

Abbildung 4.5 zeigt ein funktionierendes Pan-Tilt System mit einer montierten Spiegelreflexkamera. Zu sehen ist, dass im Gegensatz zu dem System auf Abbildung 4.3, die Kamera beidseitig entlang der Tilt-Achse befestigt ist. Dadurch verteilt sich das Gewicht der Kamera gleichmäßiger.

Nach Betrachtung aller Konzepte, bietet sich für die Umsetzung dieser Studienarbeit der grundsätzliche Aufbau des dritten gezeigten Systems in Abbildung 4.5 am besten an. In dem Video, aus welchem das Bild entnommen wurde, wird gezeigt, dass das System mit einer Spiegelreflexkamera tadellos funktioniert und mit Hilfe von einem billigen 3D-Drucker für Privatanwender gebaut werden kann. Somit müssen keine hochpräzisen Metallarbeiten erledigt werden, welche voraussichtlich deutlich teurer wären. Das Differential-System hat den Konstruktions-Nachteil, dass bei Abschaltung der Motoren, die Kamera aufgrund des Gewichts, unkontrolliert nach vorne fallen würde.

4.1.3. Konzepte erstellen und Hardware konzipieren

Nach der Recherche wurde sich dazu entschieden, die Grundidee des Projekts aus Abbildung 4.5 als Orientierung zu nutzen. Ein bereits vorhandenes 3D-Modell des System, hat dabei geholfen, das entwickelte System und die dahinterstehenden Konzepte näher zu betrachten und zu verstehen[5].

Tilt-Bewegung

Um die Tilt-Bewegung zu realisieren, wird das in Abbildung 4.6 gezeigte Prinzip übernommen. Zu sehen ist, dass die Kamera auf einer Fläche (4.6, 1.) unterhalb der Drehachse

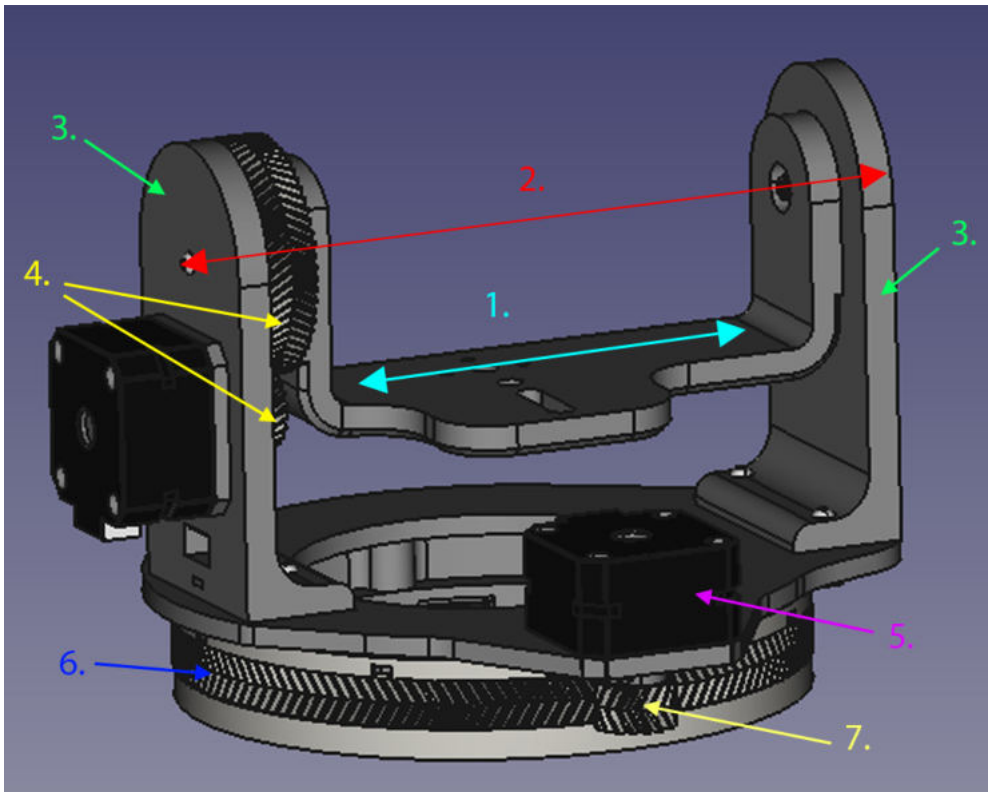

Abbildung 4.6.: 3D-Modell der Pan-Tilt Einheit von Isaac879

(4.6, 2.) montiert wird. Dadurch ergibt es sich, dass der Schwerpunkt der Kamera auf der Höhe der Drehachse ist. Des Weiteren fällt die Kamera bei Ausfall der Motoren somit immer wieder in die Grundposition zurück, wodurch keine ungewollten Effekte entstehen. Die groben Proportionen der seitlichen Befestigungsfüße (4.6, 3.) für die Tilt-Achse und die Zahnradübersetzung (4.6, 4.) werden aus dem Projekt von “Isaac879“ übernommen. Beides hat sich in dem dazugehörigen Video [4] als gut umsetzbar und druckbar dargestellt. Des Weiteren werden durch d

Pan-Bewegung

In Abbildung 4.6 ist zudem zu sehen, dass die Einheit die die Tilt-Bewegung durchführt und der frontal zu sehende Motor (4.6, 5.) , zusammen auf einer Plattform befestigt sind. Diese Plattform ist so gelagert, dass sich diese unabhängig von dem weiter unterliegendem Teil (4.6, 6.) drehen kann. Dieses Teil ist fixiert und an der außen Seite mit Zähnen bestückt. Dadurch kommt es zu einer Pan-Bewegung der ganzen oberen Plattform, sobald der Motor (4.6, 5.) sich dreht und das Zahnrad (4.6, 7.) um die fixierte Basis (4.6, 6.) fährt.

Dieses Konzept, dass sich eine Hauptplattform um eine statische Basis dreht, wird für diese Studienarbeit übernommen. Um zu gewährleisten, dass System weniger Fehleranfällig

durch Einflüsse von außen ist, wird der Motor (4.6, 5.) mit dem Zahnrad (4.6, 7.) die Pan-Bewegung nicht außerhalb der statischen Basis durchführen, sondern wie in Abbildung 4.7 zu sehen, innen-läufig.

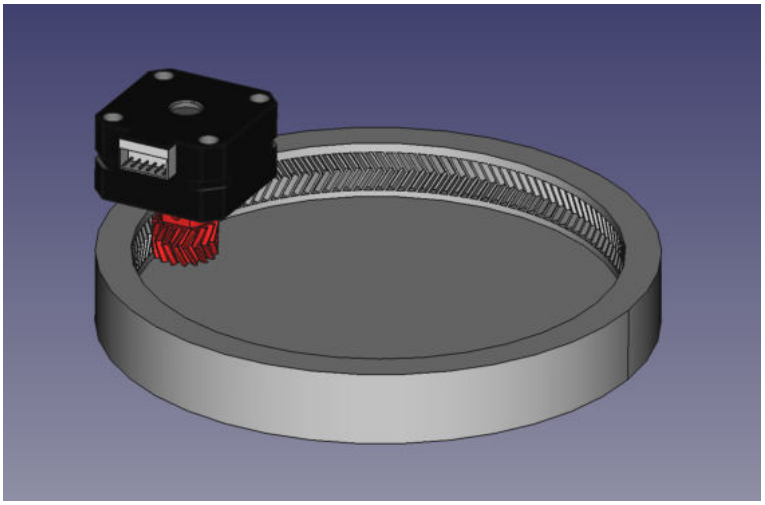

Abbildung 4.7.: Visualisierung des grundsätzlichen Aufbaus eines innen läufigen Zahnrads.

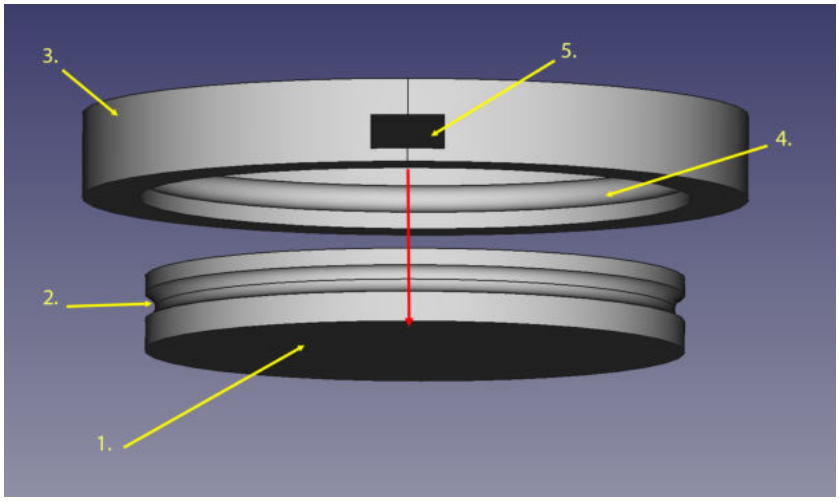

Abbildung 4.8.: Statische Basis (1.) mit Mulde an Außenseite (2.). Hauptplattform (3.) mit Mulde an Innenseite (4.). Loch für Kugeln in Hauptplattform für Befüllung der Kugeln (5.).

Kugellager

Damit die Drehbewegung der Hauptplattform um die statische Basis herum gewährleistet wird, verbindet ein Kugellager beide Teile miteinander. Für die Realisierung des Kugellager, wird die Hauptplattform (4.8, 3.) um die Basis (4.8, 1.) gestülpt und jeweils die Außenseite der statischen Basis (4.8, 2.) und die Innenseite der Hauptplattform (4.8, 4.) erhalten eine rundliche Mulde. Nach dem übereinander Stülpen sind die Mulden nun auf einer

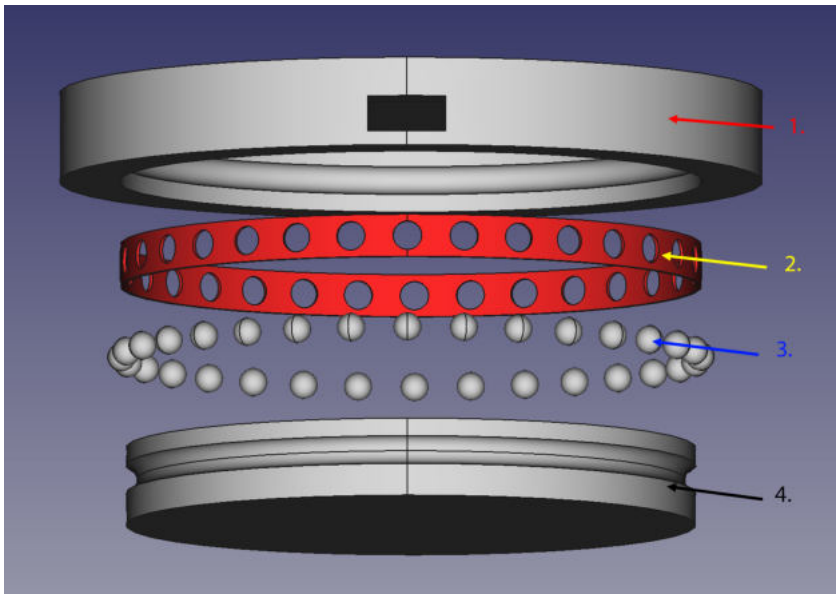

Abbildung 4.9.: Komponenten des Kugellagers. Hauptplattform (1.), Abstandhalter-Ring (2.), Kugeln (3.), statische Basis (4.)

Höhe, sodass durch ein Loch in der Hauptplattform (4.8, 5.) Kugeln in die Donut-förmige Aushöhlung eingeführt werden können.

Damit nicht die ganze Mulde mit Kugeln gefüllt werden muss, kommt ein Ring (4.9, 2.) zwischen Hauptplattform (4.9, 1.) und Basis (4.9, 4.). Dieser Ring ist dafür da, die eingeführten Kugeln (4.9, 3.) an ihrem Platz zu halten.

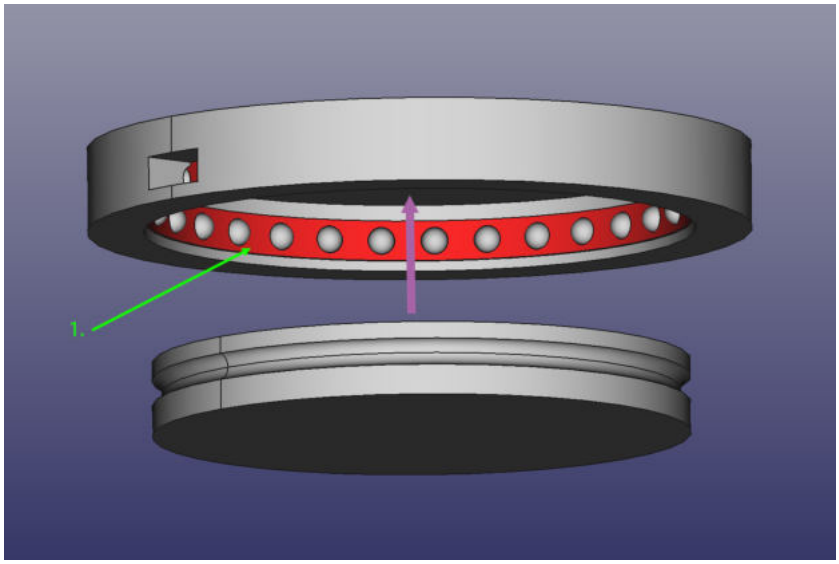

Abbildung 4.10.: Kugeln mit Abstandhalter-Ring (1.) in Hauptplattform. Statische Basis drüber.

In Abbildung 4.10 ist zu sehen, wie die Kugeln mit dem Abstandhalte-Ring (4.10, 1.) in der Hauptplattform liegen. Diese Abbildung dient nur der Illustration, da die Hauptplattform inklusive Kugeln nicht mehr über die Basis gestülpt werden könnte. Die Verbindung der

beiden Teile entsteht ausschließlich durch die Einführung von Kugeln durch das Loch in der Hauptplattform. Dies muss nach fertigem Einfüllen geschlossen werden.

Slider

Die Bewegung des System entlang einer Achse im Raum, wird mit einem sogenannten *Slider* durchgeführt. Der Slider wurde ohne Anregung anderer Projekte entwickelt. Besonderer Fokus wurde darauf gelegt, dass die Bewegung im Raum gleichmäßig und die Länge variabel ist. Um eine möglichst gleichmäßige und ruckelfreie Bewegung zu gewährleisten, wurde sich dazu entschieden, das Pan-Tilt-System mit Hilfe eines Zahnriemens über zwei Aluminiumrohre zu ziehen.

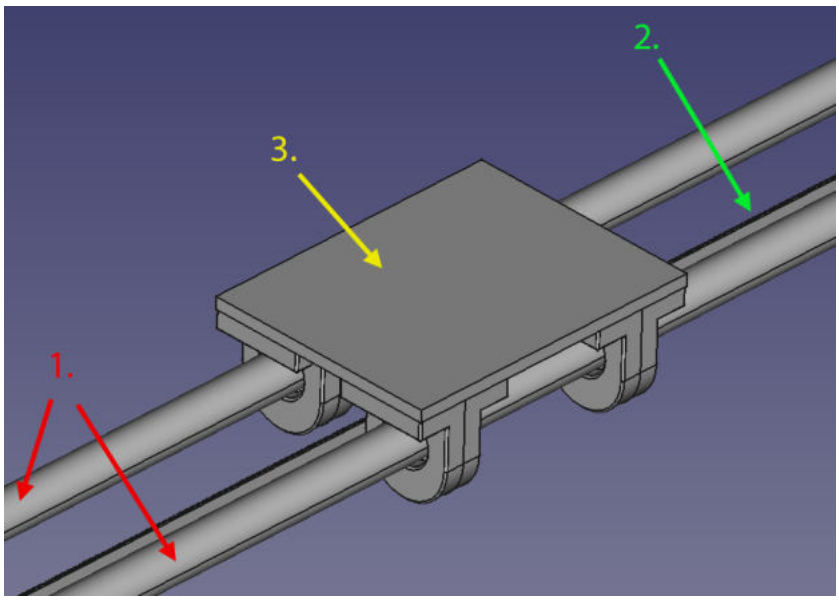

Abbildung 4.11.: Slider Aufbau: Bewegliche Plattform (3.) fährt mit Hilfe von starrem Zahnriemen (2.) über zwei Aluminiumrohre (1.)

In Abbildung 4.11 ist zu sehen, wie eine Plattform (4.11, 3.) an zwei parallele Aluminiumrohre (4.11, 1.) befestigt ist. Das Pan-Tilt-System wird auf die gezeigte Plattform montiert. Die Realisierung der Bewegung der Plattform entlang der Aluminiumrohre wird in Abbildung 4.12 gezeigt.

Der Motor welcher für die Bewegung des Sliders zuständig ist, bewegt sich immer mit dem restlichen Pan-Tilt-System. Daher kann die Länge der Aluminiumrohre und des starren Zahnriemens beliebig verändert werden.

Um die Bewegung möglichst gleichmäßig und ruckelfrei zu gestalten, werden die Befestigungen der Plattform an den Aluminiumrohren keinen direkten Kontakt zu den

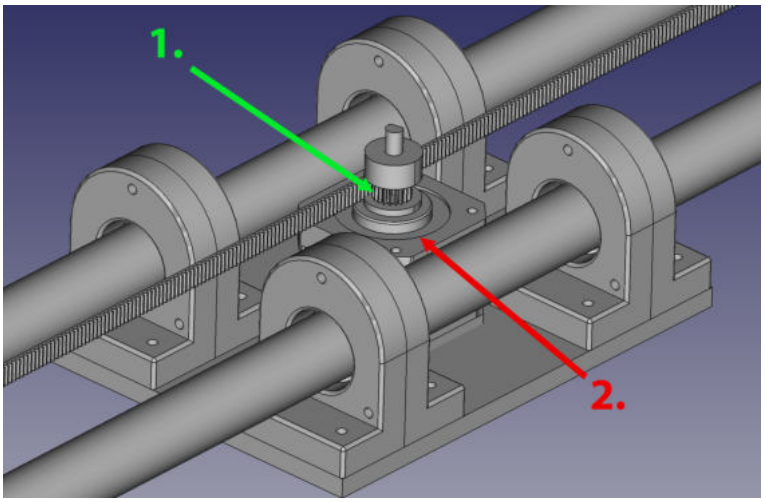

Abbildung 4.12.: Slider Aufbau: Motor (2.) mit Zahnrad (1.) ist an der Plattform befestigt. Der Zahnriemen liegt so, dass das Zahnrad sich bei Drehung entlang des Zahnriemens bewegt.

Aluminiumrohren haben. In die Befestigungen werden rund um die Aluminiumrohre Stahlkugeln platziert, welche sich zwar um sich selbst beliebig drehen können, allerdings nicht die Position verlassen können.

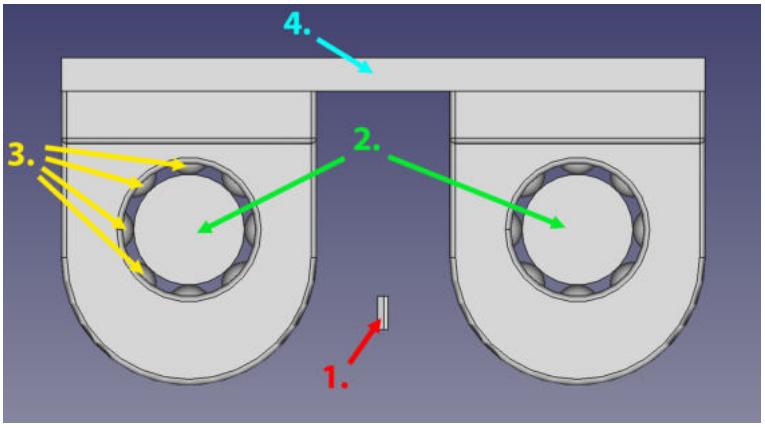

Abbildung 4.13.: Slider Aufbau Querschnitt: Plattform mit Befestigungen (4.). Stahlkugeln (3.) innerhalb der Befestigungen, um über Aluminiumrohre (2.) zu rollen. Zahnriemen (1.) mittig. Motor und Zahnrad wie in Abbildung 4.12 zur Übersichtlichkeit nicht zu sehen.

Elektronik-Management

Das Verarbeiten des Live-Bilds der Kamera erfolgt mit einem Raspberry Pi. Dieser ist in der Lage *Python*-Programme auszuführen und es kann für die Gesichtserkennung die bereits vorhandene Bibliothek *OpenCV* genutzt werden. Des Weiteren verfügt dieser über USB-B Schnittstellen, welche für den Eingang des Live-Bildes benötigt werden. Da der Raspberry Pi allerdings nicht für präzise Schrittmotor Ansteuerungen geeignet ist, wird zusätzlich ein Arduino nano verwendet. Über serielle Schnittstellen des Raspberry Pis

und des Arduino nanos können beide Einheiten bidirektional miteinander kommunizieren. Die Schrittmotoren arbeiten mit einer größeren Leitung als der Arduino nano bieten kann. Daher werden Motor-Treiber benötigt, welche mit Hilfe einer weiteren externen Stromquelle den Eingangsstrom des Arduino nanos verstärken. Um den Arduino nano und den Raspberry Pi nicht nur in einer Konsole über einen Computer zu steuern, werden ebenfalls ein kleines Display und ein Joystick in das System integriert. Darüber können die einzelnen Motoren angesteuert werden und das Geischts-Tracking aktiviert und deaktiviert werden.

Um die Elektronikteile möglichst kompakt in das System zu integrieren, wird eine Box auf die Hauptplattform montiert, welche die ganze Elektronik beinhaltet.

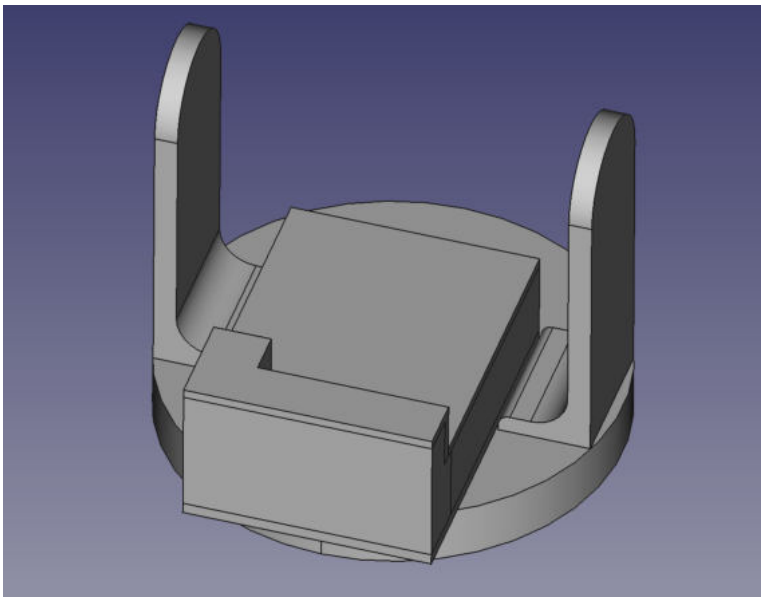

Abbildung 4.14.: Box für sämtliche Elektronik auf der Hauptplattform und neben den Füßen für die Tilt-Bewegung.

4.1.4. Hardware konstruieren

Kamerahalterung

Die Größe des ganzen Systems hängt maßgeblich von der Größe der Kamera ab. Daher wird als erstes das Teil konstruiert, auf welches die Kamera montiert wird. Im Folgenden wird dieses Teil als *Kamerahalterung* benannt. Um die Maße der Kamerahalterung festzulegen, werden die ausschlaggebenden Maße der Kamera ermittelt.

Wie in Abbildung 4.16 illustriert, besitzt die Kamera mit eine 128 mm breite Auflagefläche. Daher muss die Auflagefläche der Kamerahalterung ebenfalls mindestens diese Breite

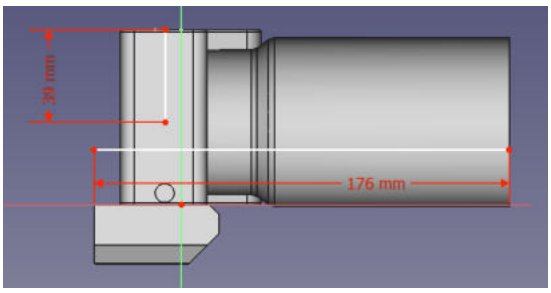

Abbildung 4.15.: Seitliche, auf dem Kopf stehende Ansicht eines Modells der Sony A7 RII. Länge der Kamera-Objektiv Kombination und Höhe der Sensormitte angegeben.

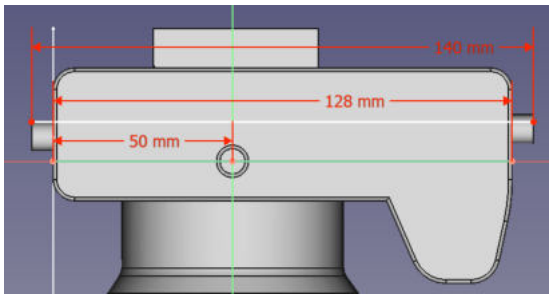

Abbildung 4.16.: Ansicht auf das Kameramodell von unten. Breite der Auflagefläche, Breite inklusive Zylinderförmige Befestigungen (Position in Abbildung 4.15 ersichtlich) und Abstand von Außenseite zu Mitte des Gewindelochs zur Befestigung der Kamera

aufweisen. Um einen kleinen Spielraum zu setzen, wird die Breite mit 131 mm definiert. In Abbildung 4.16 ist hinzu die Breite der Kamera inklusive zylinderförmige Halterungen ersichtlich. Die Halterungen sind nur oberhalb der Kamera vorhanden und in Abbildung 4.15 als Kreis erkenntlich. Diese haben zur Folge, dass die Breite oberhalb der Auflagefläche größer sein muss.

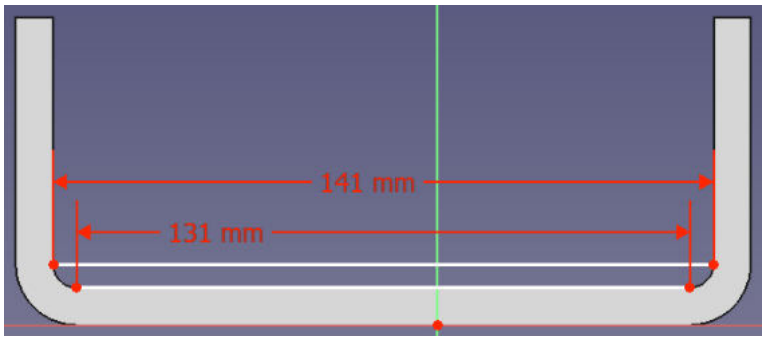

Abbildung 4.17.: Kamerahalterung: Auflagefläche für Kamera = 131 mm, komplette Breite für Kamera = 141 mm

Als nächstes wird die Form der Auflagefläche bestimmt. Zu beachten ist dabei, dass das Befestigungsgewinde der Kamera nicht mittig ist. Zudem hat die Kamera-Objektiv Kombination durch das Objektiv einen stark nach vorne verlagerten Schwerpunkt. Um das Gewicht besser zu verteilen, wird durch eine separate Objektivstütze das Gewicht des Objektives direkt auf die Auflagefläche verlagert.

Um den Schwerpunkt der Kamera-Objektiv Kombination auf die Tilt-Drehachse (4.18, 4.) zu verlagern, werden die Löcher zur Befestigung der Objektivstütze vor die Drehachse (4.18, 1.) und das Loch für die Befestigung der Kamera etwas hinter die Drehachse (4.18, 2.) verlagert. Die “Löcher“ sind dabei so geformt, dass die letztendliche Position der Kamera und der Objektivstütze für Feineinstellungen variiert werden können. Um zu gewährleisten, dass andere Objekte auf die Kamerahalterung montiert werden können, werden zusätzliche

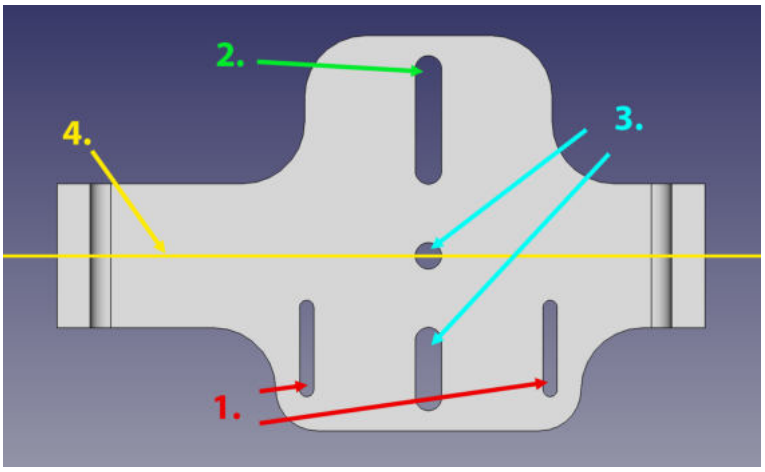

Abbildung 4.18.: Kamerahalterung Ansicht von oben: Löcher für Befestigung der Objektivstütze (1., r = 1,7 mm), Loch für Befestigung der Kamera (2., r = 3,2 mm), Optionale Löcher (3., r = 3,2 mm)

Löcher integriert (4.18, 3.).

Die Kamerahalterung wird an einer Seite an einem Fuß gehalten und auf der anderen Seite mit einem Zahnrad verbunden welches mit dem anderen Fuß verbunden ist (siehe Abbildung 4.6).

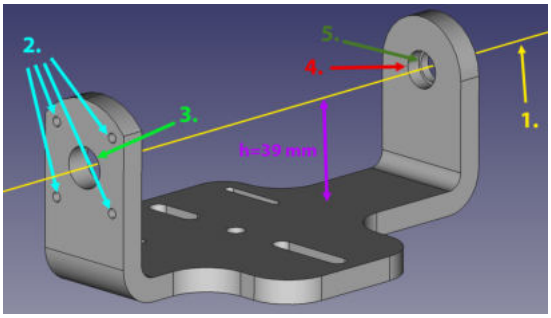

Abbildung 4.19.: Kamerahalterung mit Befestigungsvorrichtungen an den Seiten. Tilt-Drehachse (1.), Löcher für Befestigung des Zahnrads (2.), Loch in der Mitte um in späteren Schritten Kugellager im Zahnrad zu befestigen (3.), Loch mit größerem Durchmesser, sodass ein Kugellager rein passt (4.), Loch mit kleinerem Durchmesser, sodass Kugellager halt hat und fest gemacht werden kann (5.)

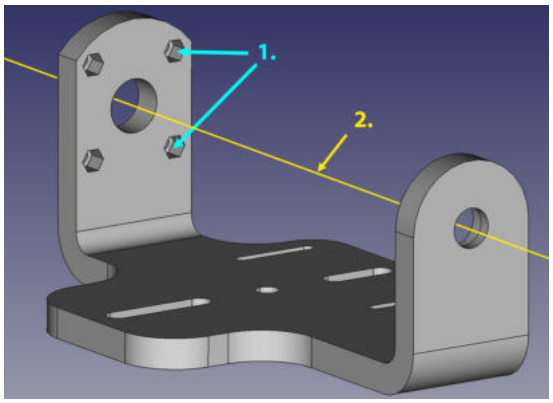

Abbildung 4.20.: Sechseckige Vertiefungen von der Innenseite für Muttern (1.), Tilt-Drehachse (2.)

In Abbildung 4.19 ist ersichtlich, dass auf der Seite an welcher die Kamerahalterung direkt an den Fuß montiert wird, das Loch zur Montage nicht durchgängig, sondern zwei verschiedene Durchmesser hat (4.19, 4. und 5.). Dies ist notwendig, um ein Kugellager in diesem Loch zu befestigen. Auf der anderen Seite ist ersichtlich, dass neben einem

großen Loch in der Mitte (4.19, 3.), 4 kleinere Löcher vorhanden sind (4.19, 2.). Diese werden benötigt um ein Zahnrad an die Kamerahalterung zu schrauben. Die Schrauben sollen dabei zuerst durch das Zahnrad, dann durch die Kamerahalterung gesteckt werden. Dadurch ergibt es sich, dass auf der inneren Seite der Kamerahalterung Muttern benötigt werden. Damit diese nicht durchdrehen, werden auf der Innenseite sechseckige Vertiefungen vorgenommen (4.20, 1.).

Objektivstütze

Die Objektivstütze wird benötigt, um das Gewicht der Kamera inklusive des langen Objektives auf zwei Stellen aufzuteilen.

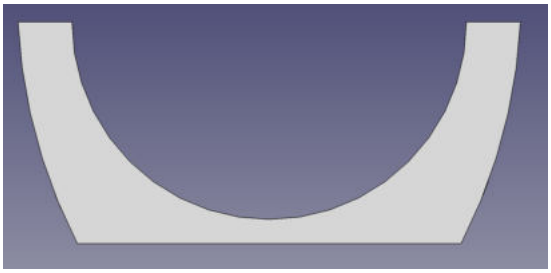

Abbildung 4.21.: Frontale Ansicht auf die Objektivstütze. Die Form ist an das benutzte *Tamron*-Objektiv angepasst.

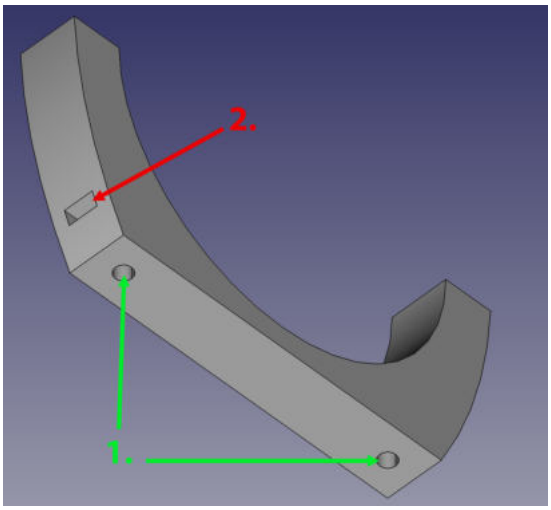

Abbildung 4.22.: Löcher von unten (1.) und Einschübe für Muttern an den Seiten (2.) ermöglichen eine Befestigung an den dafür vorhergesehenen Löchern an der Kamerahalterung (4.18, 1.)

Zahnrad an Kamerahalterung für Tilt-Bewegung

Damit die Kamerahalterung entlang der Tilt-Achse gedreht werden kann, wird ein Zahnrad an die Kamerahalterung montiert. Da das Zahnradgetriebe aus dem Projekt von Isaac879 gut funktioniert hat und gezeigt wurde, dass diese Zahnräder von herkömmlichen 3D-Druckern gedruckt werden können, wurde die gleiche Anzahl an Zähnen für diese Studienarbeit genutzt. Das kleinere Zahnrad, welches direkt am Motor befestigt ist (4.6, 4.), hat 21 Zähne. Das Zahnrad an der Kamerahalterung hat 64 Zähne. Das Zahnradgetriebe erfüllt des Weiteren auch die Anforderung, dass die minimale Schrittweite kleiner als 0.5° sein muss.
Die Schrittweite der Motoren liegt bei 1.8° pro Schritt. Allerdings kann ein Schritt mit

Hilfe der benutzten Motortreiber auf bis zu 16 Teilschritte geteilt werden. Dadurch ergibt sich folgende Rechnung:
$\frac{21}{64} * 1.8° \approx 0.59$ °pro **ganzem** Schritt.
Mit Hilfe des Motortreibers, kann dieser Wert beliebig durch 2, 4, 8 oder 16 geteilt werden. Dadurch wird die Anforderung der minimalen Schrittweite deutlich erfüllt.

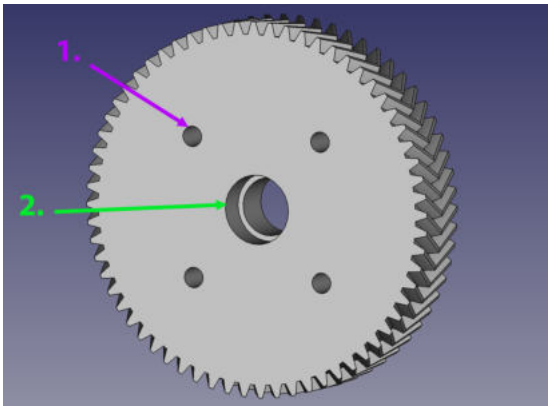

Abbildung 4.23.: Zahnrad für Tilt-Drehung an Kamerahaltung. Löcher zur Befestigung an Kamerahalterung (1.). Loch mit unterschiedlichem Durchmesser innerhalb des Zahnrads zur Befestigung eines Kugellagers (2.).

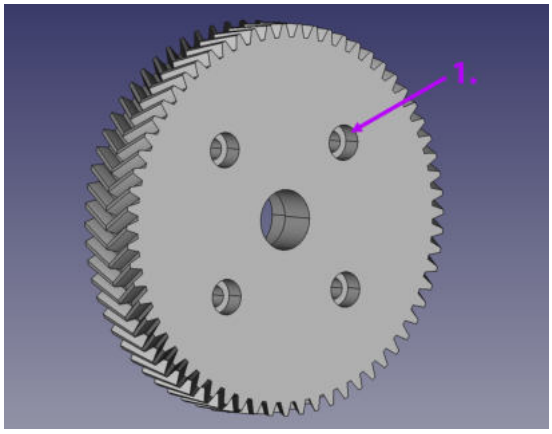

Abbildung 4.24.: Zahnrad für Tilt-Drehung andere Seite. Vertiefungen an Löchern für Schraubenköpfe (1.)

Wie in Abbildung 4.23 und 4.24 zu sehen, wird eine Doppelschräg-Verzahnungen für die Zähne verwendet. Durch die schräge Verzahnung werden die Kräfte gleichmäßiger übertragen. Es besteht im Gegensatz zu einfachen parallelen Zähnen immer Kontakt zwischen den Zähnen zweier Zahnräder. Eine einfache schräge Verzahnung würde allerdings zu ungewollten Axialkräften führe, welche die sich zu drehenden Zahnräder auseinander ziehen würde. Dies wird durch eine Doppelschräg-Verzahnung vermieden da somit immer zwei, sich ausgleichende, Axialkräfte entstehen.
In Abbildung 4.23 sind vier Löcher zur Befestigung des Zahnrads an der Kamerahalterung ersichtlich (4.23, 1.). Auf der anderen Seite des Zahnrads sind zusätzlich Vertiefungen an den Löchern vorhanden damit keine Schraubenköpfe aus dem Zahnrad hinaus ragen (4.24, 1.). Damit das Zahnrad mit der Kamerahalterung sich drehen kann, wird in das Zahnrad ein Kugellager befestigt (4.23, 2.). Die Befestigung basiert auf dem gleichen Prinzip wie die Befestigung des anderen Kugellagers in der Kamerahalterung (4.19, 4. und 5.).

Zahnrad an Motor für Tilt-Bewegung

Um die Kamerahalterung mit montiertem Zahnrad zu drehen, wird ein weiteres Zahnrad benötigt, welches mit einem Motor gedreht wird. Die Anzahl der Zähne beträgt 21. Das Zahnrad ist wie das andere Zahnrad doppelschräg-verzahnt.

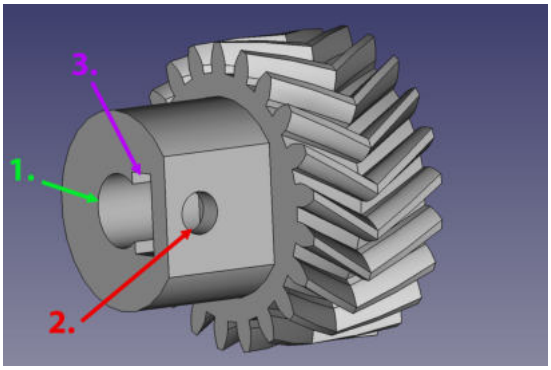

Abbildung 4.25.: Zahnrad an Motor für Drehung der Kamerahalterung. Mittiges Loch für Motorpin (1.). Loch für Befestigung des Zahnrads an Motorpin (2.). Einschub für Mutter damit Schraube gegen Motorpin geschraubt werden kann (3.).

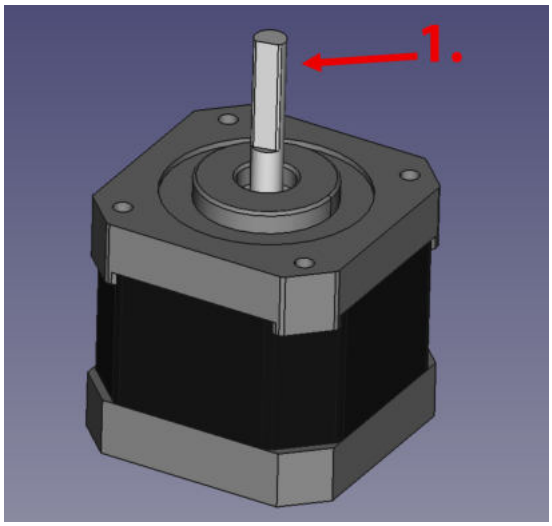

Abbildung 4.26.: Verwendeter Nema 17 42x42mm Schrittmotor. Motorpin (1.)

Mit einem Loch in der Mitte (4.25, 1.) wird das Zahnrad auf den Motorpin (4.26, 1.) gestülpt. Für die Fixierung wird zuerst eine Mutter in den vorgesehenen Einschub platziert (4.25, 3.), wodurch danach eine Schraube das Zahnrad an den Motorpin befestigen kann.

Rechte Befestigung der Kamerahalterung

Die Kamerahalterung, die Zahnräder und der Motor werden durch ein Befestigungsfuß miteinander verbunden und zusammen gehalten. Der Fuß wird dabei auf die Hauptplattform geschraubt.

Wie in Abbildung 4.27 und 4.28 ersichtlich, wird der Fuß mit drei Schrauben an die Hauptplattform geschraubt (4.27, 1. und 4.28, 2.). Für zusätzliche Stabilität wurden zwei kleine Erhebungen an den Fuß positioniert (4.28, 1.), welche Halt in vorgesehenen Löchern in der Hauptplattform finden. Die Löcher für die Befestigung des Motors mit Zahnrad, entsprechen langgezogenen Löchern (4.27, 4. und 3.). Dies hat den Vorteil, dass der Motor in der Höhe verstellt werden kann und somit unterschiedliche Zahnräder an den Motorpin geschraubt werden könnten. Durch ein seitliches Loch im Fuß (4.27, 6.) ist es möglich, die Schraube zur Befestigung des Zahnrads am Motorpin zu erreichen. Eine leichte Erhebung am Loch zur Befestigung der Kamerahalterung (4.27, 5.) führt dazu, dass das befestigte Zahnrad nicht mit der seitlichen Oberfläche an dem Fuß reibt. Die Schraube zur Befestigung wird mit einer Mutter fixiert, welche durch eine Hexagon-förmige Vertiefung im Fuß gehalten wird (4.28, 3.). Ein Loch mittig unterhalb der Befestigungsstelle des Motors (4.27, 2.) ermöglicht eine kurze Kabelführung der Motor-Kabel.

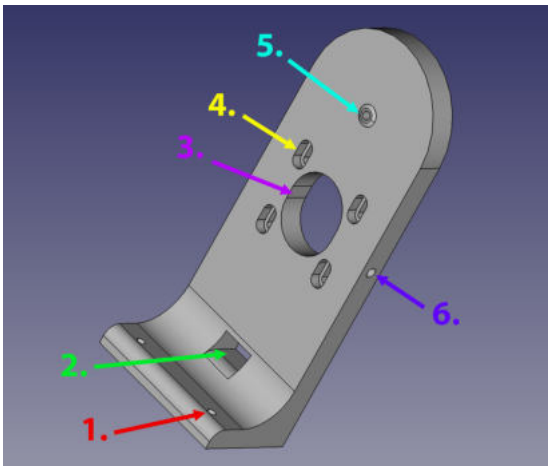

Abbildung 4.27.: Rechte Befestigung der Kamerahalterung. Löcher für Befestigung an Hauptplattform (1.). Loch um Kabel des Motors für die Tilt-Bewegung hindurch zu führen (2.). Langgezogenes Loch um Motorpin mit Zahnrad hindurch zu stecken (3.). Vier Langgezogene Löcher für die höhenverstellbare Befestigung des Motors (4.). Leichte Erhebung mit Loch um Kamerahalterung mit Zahnrad am Fuß zu befestigen und das Zahnrad bei Drehung nicht überall am Fuß Kontakt hat (5.). Seitliches Loch um Zahnrad an montiertem Motor festzuschrauben (6.).

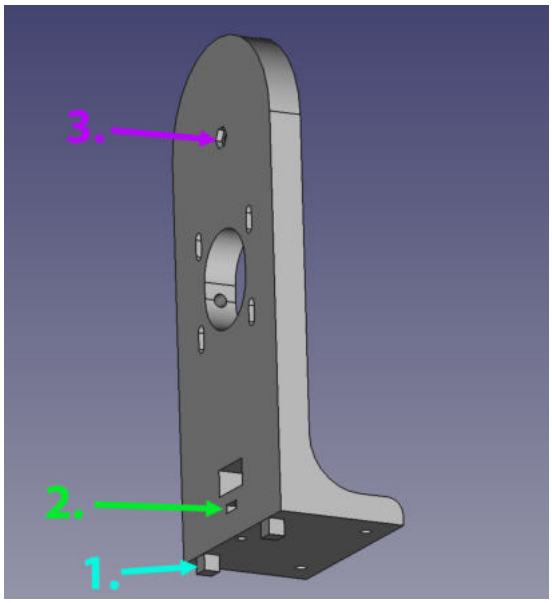

Abbildung 4.28.: Rechte Befestigung der Kamerahalterung. Ansicht von hinten-unten. Zwei Erhebungen in Hauptplattform für zusätzliche Stabilität (1.). Einschub für Mutter um Fuß mit weiterer Schraube an Hauptplattform zu schrauben (2.). Hexagonförmige Vertiefung für Mutter, welche Schraube zur Befestigung der Kamerahalterung fixiert (3.)

Linke Befestigung der Kamerahalterung

Die linke Befestigung der Kamerahalterung ist der rechten Befestigung nachempfunden, siehe Abbildung 4.29. Allerdings fehlen die Löcher zur Befestigung eines Motors und das Loch für die Kabelführung.

Hauptplattform

Die Hauptplattform verbindet die Befestigungen mit der Kamerahalterung. Sie dient des Weiteren für die Befestigung des Pan-Motors und einer Box für das Elektronik Management. Zudem ist die Hauptplattform um die statische Basis drehbar.

Die Hauptplattform ist kreisförmig aufgebaut. Auf der oberen Oberfläche befinden sich Löcher zur Befestigung anderer Teile. Die Löcher zur Befestigung der Befestigungsfüße sind jeweils auf der linken und rechten Seite gleich. Drei runde Löcher, um mit Hilfe von Schrauben den Fuß an die Hauptplattform zu schrauben und zwei quadratische Aushöhlungen für die unteren Erhebungen des Fußes (4.30, 1.). Das rechteckige Loch in

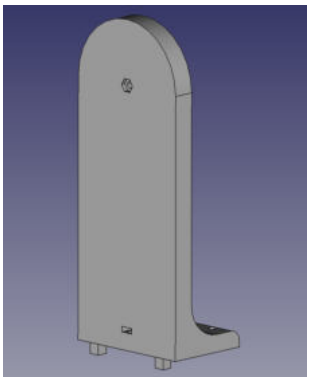

Abbildung 4.29.: Linker Befestigungsfuß. Vereinfachte Kopie der rechten Befestigung der Kamerahalterung 4.1.4.

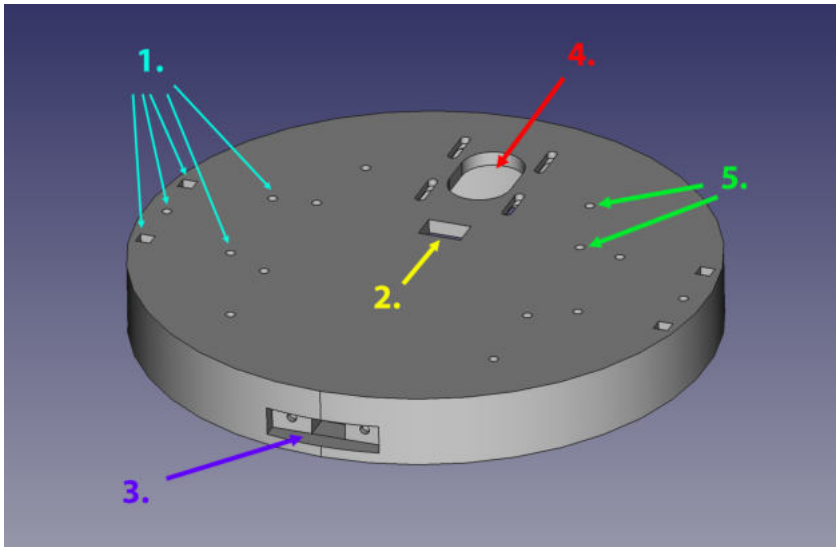

Abbildung 4.30.: Hauptplattform, Ansicht von oben. Befestigungslöcher für Befestigungsfuß (1.). Loch um Kabel des Slider-Motors von unten in die Elektronikbox zu führen (2.). Aushöhlung um Kugeln zwischen statische Basis und Hauptplattform einzufüllen (3.). Loch um Pan-Motor von oben auf die Hauptplattform zu montieren (4.). Alle weitere Löcher dienen zur Befestigung der Box für das Elektronik Management (5.).

der Mitte der Hauptplattform ermöglicht es die Kabel des Slider-Motors mittig durch die statische Basis und die Hauptplattform in die Elektronik-Box zu führen (4.30, 2.). An der frontalen Seite befindet sich ein Loch, um Kugeln zwischen die Hauptplattform und die statische Basis zu führen (4.30, 3.). Neben dem Loch befinden sich zwei Löcher und eine Aushöhlung, um einen Stopfen in das Loch zu führen und an die Hauptplattform zu schrauben. Des Weiteren befinden sich Löcher für die Befestigung des Pan-Motors auf der Hauptplattform (4.30, 4.). Der dort befestigte Motor ist der in Abbildung 4.7 zu sehende Motor, welcher die Hauptplattform durch Drehung eines Zahnrads und Zähnen auf der Innenseite der statischen Basis bewegt. Alle weiteren Löcher auf der Hauptplattform, dienen der Befestigung der Box für das Elektronik-Management (4.30, z.B. 5.).

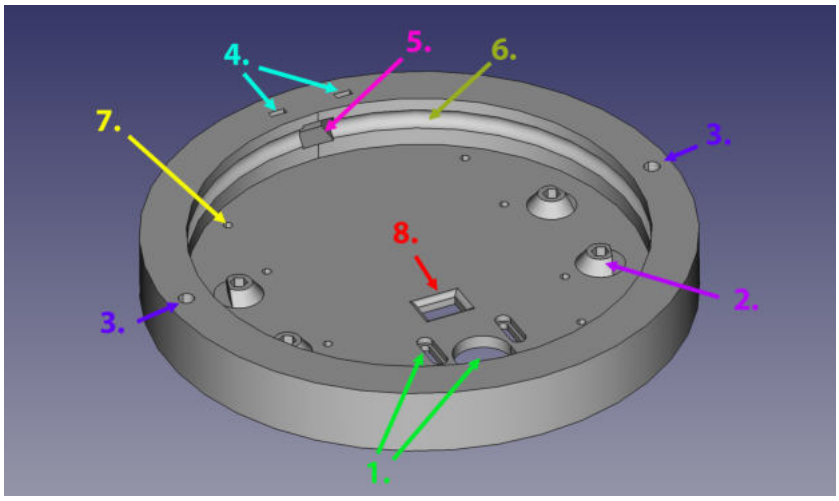

Abbildung 4.31.: Hauptplattform, Ansicht von unten. Löcher zur Montage des Pan-Motors (1.). Erhebungen mit sechseckigen Aushöhlungen für Muttern zur Befestigung der Befestigungsfüße (2.). Löcher zur Befestigung der Befestigungsfüße (3.). Einschübe für Muttern zur Befestigung des Kugeleinführungsstopfens (4.). Loch zum Einführen von Kugeln zwischen statische Basis und Hauptplattform (5.). Halbkreis-förmige Mulde für Kugeln zwischen statischer Basis und Hauptplattform (6.). Löcher zur Befestigung der Box für das Elektronik Management (7.).

Unterhalb der Hauptplattform ist zu erkennen, dass die seitlichen Löcher zur Befestigung des Pan-Motors ab der Mitte der Hauptplattform einen größeren Radius haben (4.31, 1.). Dadurch ragen die Schraubenköpfe nicht aus der Hauptplattform raus. Dies ist notwendig, da die statische Basis sich im Inneren der Hauptplattform befindet und die Hauptplattform sich um diese drehen soll. Dafür sind an den Erhebungen zur Fixierung der Muttern zur Befestigung der Füße ebenfalls nach außen gerichtete Einkerbungen ersichtlich (4.31, 3.). Um den Kugeleinführungsstopfen zu befestigen, befinden sich zwei Einschübe für Muttern auf der Unterseite der Hauptplattform (4.31, 4.). Dadurch kann der Stopfen an die Hauptplattform geschraubt werden und füllt das in Abbildung 4.31, 5. zusehende Loch, um Kugeln zwischen die statische Basis und die Hauptplattform zu befüllen. Der Stopfen selber ist dabei so aufgebaut, dass sich die Form der Mulde in der Hauptplattform (4.31, 6.) anpasst, siehe Abbildung 4.32.

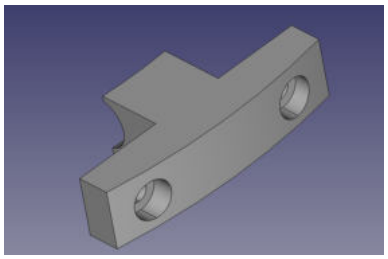

Abbildung 4.32.: Kugeleinführungsstopfen. Zwei Löcher an den Seiten um den Stopfen an die Hauptplattform zu befestigen (4.30, 3.).

Des Weiteren sind die Löcher zur Befestigung der Box für das Elektronik Management (4.31, z.B. 7.) und das Loch zur Kabelführung des Slider-Motors (4.31, 8.) ersichtlich.

Statische Basis

Die statische Basis dient als starres Element, um welches sich die Hauptplattform drum herum dreht.

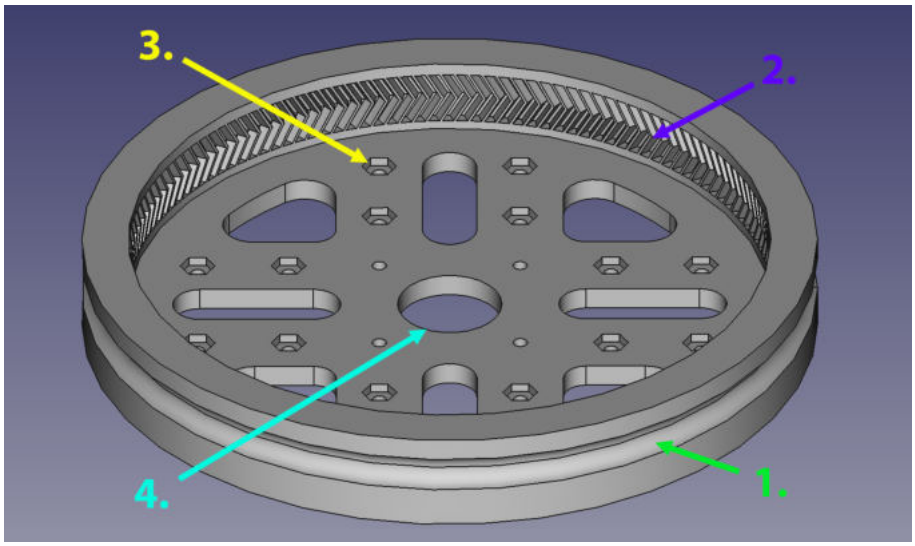

Abbildung 4.33.: Statische Basis. Mulde für Kugeln zwischen statischer Basis und Hauptplattform (1.). Innere doppelschräg Verzahnung (2.). Löcher mit sechseckiger Vertiefung für Befestigung des Sliders (3.). Loch zur Führung der Kabel des Slider-Motors (4.).

Um die Drehung der Hauptplattform zu ermöglichen, ist an der äußeren Seite der statischen Basis eine Mulde vorhanden (4.33, 1.). Diese ermöglichst es, Kugeln in den Platz zwischen der Mulde der Hauptplattform und der Mulde der statischen Basis zu platzieren. In der inneren Seite der statischen Basis befindet sich eine doppelschräg Verzahnung (4.33, 2.). Mit Hilfe der Verzahnung kann die Drehung der Hauptplattform durchgeführt werden. Auf die Details der Verzahnung wird in der nachfolgenden Erörterung des Zahnrads am Motor für die Pan-Bewegung eingegangen. Löcher in der Unterseite ermöglichen eine flexible Befestigung der statischen Basis auf dem Slider (4.33, 3.). Da ein Zahnrad an der Innenseite entlang fährt (vgl. Abbildung 4.7), ist es notwendig, dass keine Teile von unten heraus ragen. Damit die Befestigung trotzdem erfolgen kann, werden um die Löcher sechseckige Vertiefungen platziert. In diese Aushöhlungen können die Muttern gelegt werden, ohne dass diese nach oben hin raus ragen. Damit die Kabel des Slider-Motors zu der Box für die Elektronik geführt werden können, wird in die statische Basis mittig ein großes Loch positioniert (4.33, 4.).

Zahnrad an Motor für Pan-Bewegung

Die Drehung der Hauptplattform um die statische Basis wird mit einem Motor realisiert, welcher an der Hauptplattform fixiert ist. Dafür wird an den Motorpin ein Zahnrad befestigt, welches den Motor und somit auch die ganze Hauptplattform bei Drehung entlang der Verzahnung der statischen Basis dreht. Die Anzahl der Zähne der Verzahnung im Inneren der statischen Basis und die Anzahl der Zähne des Zahnrads am Motor werden aus dem Projekt von Isaac879 übernommen. Die Anzahl der Zähne der statische Basis beträgt 144. Die Anzahl der Zähne des Zahnrads am Motor beträgt 17. Daraus ergibt sich folgende Berechnung für die minimale Schrittweite:
$\frac{17}{144} * 1.8° \approx 0.21$ °pro **ganzem** Schritt.
Zu beachten ist, dass dieser Schritt zusätzlich mit Hilfe des Motortreibers beliebig durch 2, 4, 8 oder 16 geteilt werden kann. Das Zahnrad selber ist genau so aufgebaut wie das in Abbildung 4.25 zu sehende Zahnrad. Nur die Anzahl der Zähne wurde von 21 auf 17 reduziert.

Slider Plattform

Die Slider Plattform ist dafür zuständig, die benötigten Teile um auf den zwei Aluminiumrohren hin und her zu fahren, zu verbinden. Zudem wird die statische Basis auf die Plattform montiert.

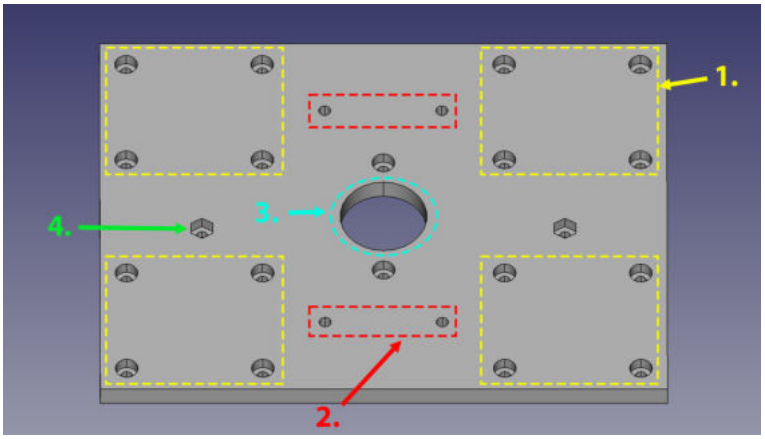

Abbildung 4.34.: Slider Plattform, Ansicht von oben. Löcher zur Fixierung von Befestigungen an den Aluminiumrohren (1.). Löcher zur Befestigung der statischen Basis auf der Plattform (2.). Loch um Kabel des Slider-Motors durch statische Basis und Hauptplattform in die Elektronikbox zu führen (3.). Alle weiteren Löcher dienen der Fixierung der Slider-Motor Befestigung (z.B. 4.).

In Abbildung 4.34 werden die Löcher zur Fixierung der Befestigungen an die Aluminiumrohre mit gelb-gestrichelten Rechtecken markiert (4.34, 1.). Da die statische Basis auf die Plattform drauf geschraubt wird, befinden sich an den Löchern Vertiefungen für

Schraubenköpfe. Die Befestigung der statischen Basis erfolgt mit den Löchern in den roten Rechtecken (4.34, 2.). Diese benötigen keine Vertiefungen für Schraubenköpfe, da die Schrauben von der Unterseite aus durch die Löcher geschraubt werden. In der Mitte der Plattform befindet sich ein Loch, um die Kabel des Slider-Motors durch die statische Basis und die Hauptplattform durch in die Elektronikbox zu führen (4.34, 3.). Alle weiteren Löcher dienen der Befestigung einer Einrichtung um den Motor für die Slider-Bewegung zu fixieren (4.34, z.B. 4.).

Befestigungen an Aluminiumrohren

Damit die Slider Plattform mit der montierten statischen Basis über die Aluminiumrohre gleiten kann, werden Befestigungen benötigt, welche die Plattform mit den Aluminiumrohren dynamisch verbindet. Damit das Gleiten möglichst gleichmäßig und mit geringem Widerstand abläuft, wird kein direkter Kontakt zwischen den selbst konstruierten Teilen und den Aluminiumrohren hergestellt. Es wird ein selbst entwickeltes statisches Kugellager verwendet. Für die Realisierung des statischen Kugellagers, wird ein Teil entwickelt, welches durch zweifaches Ausdrucken zu einem Objekt zusammengeschraubt werden kann.

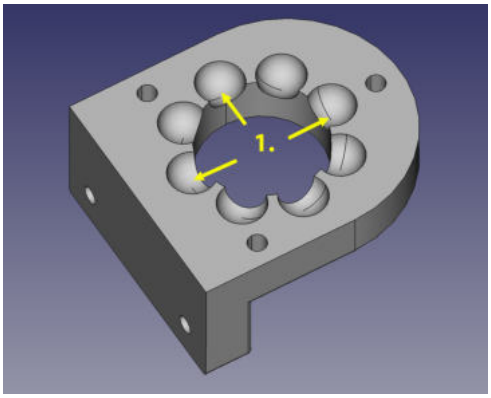

Abbildung 4.35.: Halbe Befestigung an Aluminiumrohr. Kugelmulden mit Radius $r = 4,5mm$, für Kugeln mit Radius $r = 4mm$ (1.).

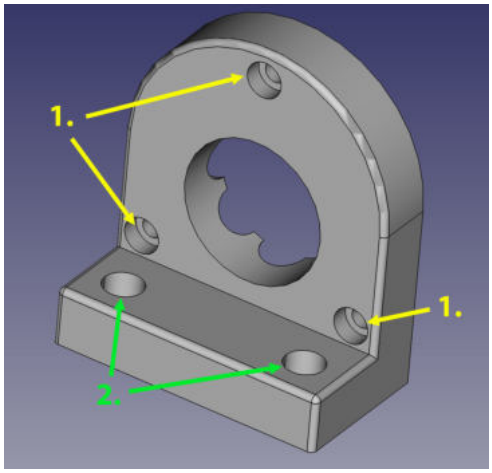

Abbildung 4.36.: Halbe Befestigung an Aluminiumrohr. Seitliche Löcher zur Fixierung zweier identischen Hälften aneinander (1.). Löcher zur Befestigung an Slider-Plattform (2.).

In Abbildung 4.35 sind auf der Oberseite des Objekts kugelförmige Mulden erkennbar (4.35, 1.). Diese haben einen Radius von 4,5 mm. In die Mulden werden Kugeln mit einem Radius von 4 mm positioniert. Dadurch ergibt es sich, dass die Kugeln etwas Raum haben um sich minimal zu auf der eigenen Stelle zu bewegen und sich zu drehen. Die Mulden sind dabei zur Mitte hin geöffnet, allerdings nur so, dass die Kugeln raus ragen, aber nicht zur Seite hin raus rollen können. Die Löcher neben den Mulden auf der Oberfläche in

Abbildung 4.35, sind die selben wie in Abbildung 4.36 mit der Nummer 1. betitelten. Sie dienen dazu, zwei der gezeigten Hälften, zu einer Befestigung mit Kugeln in den Mulden zusammen zu schrauben. Die anderen beiden Löcher (4.36, 2.) werden genutzt, um die Befestigung an die Slider-Plattform zu schrauben.

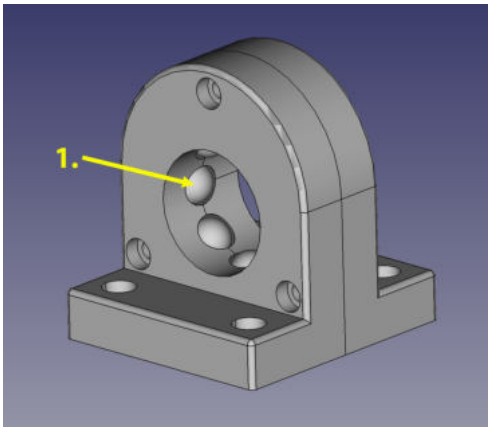

Abbildung 4.37.: Befestigung an Aluminiumrohr durch zwei identische Hälften. Modellhafte Stahlkugeln in den vorhergesehenen Mulden (1.).

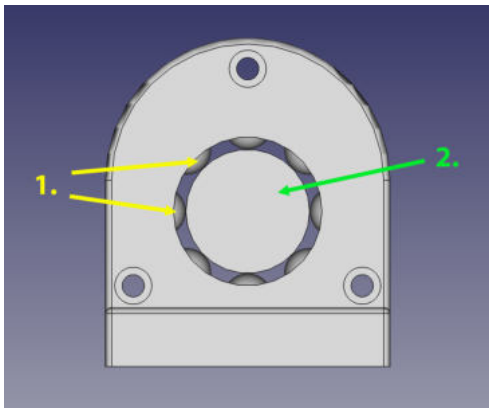

Abbildung 4.38.: Befestigung an Aluminiumrohr durch zwei identische Hälften. Modellhafte Stahlkugeln in den vorhergesehenen Mulden (1.). Alurohr zur Demonstration in der Mitte (2.).

In Abbildung 4.37 ist erkennbar, dass zwei identische Hälften nebeneinander positioniert sind und in den Mulden Kugeln liegen (4.37, 1.). Abbildung 4.38 zeigt den Aufbau frontal von vorne mit einem Aluminiumrohr (4.38, 2.) in der Mitte zwischen den Kugeln (4.38, 1.).

Slider-Motor Befestigung

Die Bewegung der Slider-Plattform mit den Befestigungen an den Aluminiumrohren ergibt sich dadurch, dass ein Motor mit einem Zahnrad entlang eines straffen Zahnriemens entlang fährt, siehe Abbildung 4.12. Diese Abbildung zeigt allerdings nur das grobe Konzept und die Positionierung des Motors ist aufgrund des vorhandenen Platzes nicht so realisierbar wie gezeigt. Des Weiteren ist es notwendig, dass das Zahnrad am Motor und der Zahnriemen gegeneinanderdrücken. Ansonsten würde das Zahnrad beim Drehen am Zahnriemen abrutschen. Um diese Anforderungen zu erfüllen wird ein weiteres Teil konzipiert, welches wie die Befestigungen an den Aluminiumrohren von unten an die Slider-Plattform montiert wird.

In Abbildung 4.39 ist die Slider-Motor Befestigung mittig ersichtlich. Es erfüllt hauptsächlich zwei Funktionen. Den Motor für die Slider-Bewegung an den vorhergesehenen Löchern (4.39, 3.) zu befestigen und mit Hilfe von zwei Umlenkrollen (4.39, 1.) den gespannten Zahnriemen gegen das Zahnrad am Motor (4.39, 2.) zu drücken. Der Zahnriemen, die

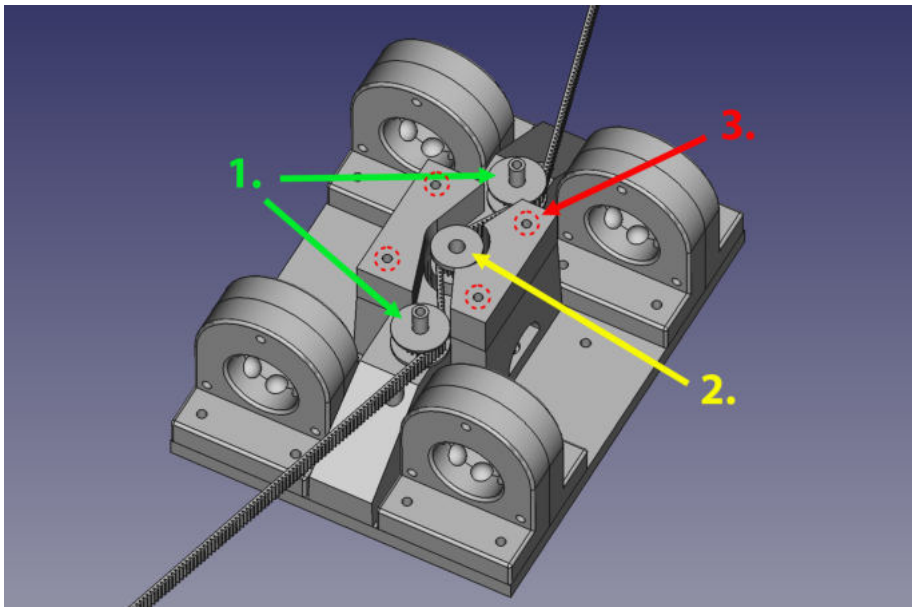

Abbildung 4.39.: Slider-Motor Befestigung in der Mitte auf Slider-Plattform. Vier Befestigungen an Aluminiumrohren. Zwei Umlenkrollen (1.) um Zahnriemen gegen Zahnrad am Motor zu drücken (2.).

Umlenkrollen und das Zahnrad am Motor werden dabei nicht selber entwickelt, sondern eingekauft.

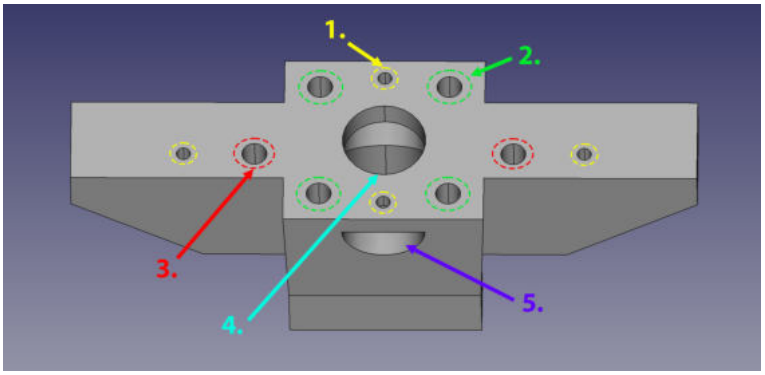

Abbildung 4.40.: Slider-Motor Befestigung, Ansicht von unten. Löcher zur Befestigung auf Slider-Plattform (1.). Löcher um Slider-Motor von unten zu befestigen (2.). Löcher um Umlenkrollen von unten zu befestigen (3.). Kabel des Slider-Motors werden durch seitliches halbkreis-Loch (5.) und dann durch mittiges großes Loch (5.) durch die Slider-Plattform, statische Basis und Hauptplattform in die Elektronikbox geführt.

In Abbildung 4.40 ist die konstruierte Slider-Motor Befestigung allein stehend von unten ersichtlich. Es befinden sich vier Löcher zur Befestigung an der Slider-Plattform auf der Unterseite (4.40, 1.). Die grün markierten etwas größeren Löcher (4.40, 2.) sind für die Befestigung des Slider-Motors zuständig. Sie sind größer, da sich der Schraubenkopf im Objekt befindet. Die zwei rot markierten Löcher (4.40, 3.) werden für die Befestigung der Umlenkrollen verwendet. Um die Kabel des Slider-Motors durch die Hauptplattform, die statische Basis und die Slider-Plattform in die Elektronikbox zu führen, befindet sich auf

der Unterseite ein mittiges großes Loch (4.40, 4.). Durch die halbkreisförmige Öffnung an der Seite (4.40, 5.) können somit die Kabel durch das große Loch geführt werden.

Füße zur Befestigung der Aluminiumrohre und des Zahnriemens

Der letzte Teil des Sliders besteht darin, die Aluminiumrohre und den Zahnriemen zu fixieren.

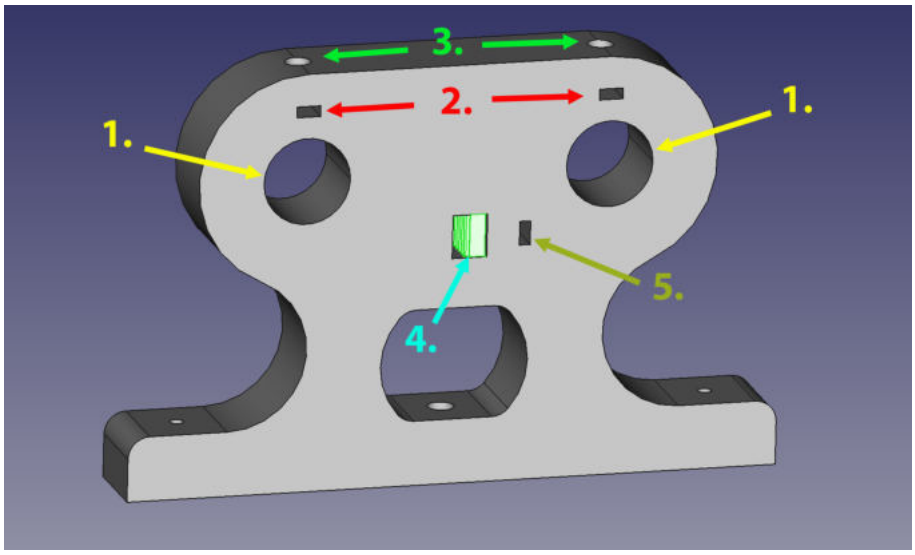

Abbildung 4.41.: Fuß zur Befestigung zweier Aluminiumrohre und eines Zahnriemens. Zwei Löcher um Aluminiumrohre mit einem Durchmesser von $d = 20mm$ hindurch zu schieben (1.). Fixierung der Rohre mit Hilfe von Schrauben, welche durch Löcher von oben (3.) und Muttern in Einschüben (2.) auf die Röhren drücken. Loses Objekt (4.), welches mit Hilfe einer Schraube und einer Mutter in einem Einschub (5.) gegen die linke innere Wand des Loches drückt und somit einen Zahnriemen fixieren kann. Anschaulicher in Abbildung 4.42
.

In Abbildung 4.41 ist das für die Fixierung konstruierte Objekt zu sehen. Im oberen Teil befinden sich zwei Löcher, um Aluminiumrohre mit einem Durchmesser von $d = 20mm$ hindurch zu schieben (4.41, 1.). Die Fixierung der Rohre erfolgt mit Schrauben, welche von oben gegen die Röhren drücken. Sie werden durch Löcher auf der Oberseite hindurch gesteckt (4.41, 3.) und bauen den Druck mit Hilfe von Muttern auf, welche mit seitlichen Einschüben in Position gebracht werden (4.41, 2.). Die Fixierung des Zahnriemens wird durch ein ähnliches Prinzip realisiert. Allerdings drückt nicht eine Schraube selber gegen den Zahnriemen. Die Schraube drückt gegen ein loses Objekt, welches ebenfalls mit Zähnen bestückt ist (4.41, 4.). Dieses drückt dann gegen den Zahnriemen.

In Abbildung 4.42 wird ersichtlich, dass die Schraube von der Seite aus durch ein Loch geschoben (4.42, 1.) und dann durch die Mutter und gegen das lose Objekt geschraubt wird.

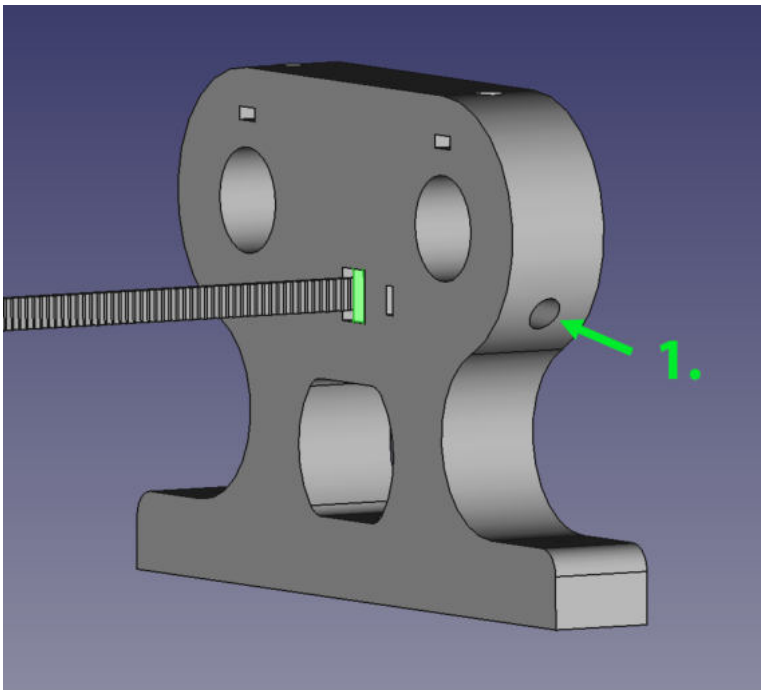

Abbildung 4.42.: Fuß zur Befestigung zweier Aluminiumrohre und eines Zahnriemens. Seitliches Loch für die Fixierung des Zahnriemens

.

Elektronikbox

Die Elektronikbox beinhaltet sämtliche Elektronik-Komponenten und verbindet alle Motoren miteinander. Wie in Abbildung 4.14 gezeigt, wird die Elektronikbox auf die Hauptplattform drauf geschraubt. Um das ganze System zu steuern, wird an die Elektronikbox ein kleines Display sowie ein Joystick montiert.

Die Elektronikbox besteht insgesamt aus fünf verschiedenen Teilen.

1. Boden mit Wänden, siehe Anhang A.1
2. Front mit Befestigung eines Displays, siehe Anhang A.2
3. Deckel der Front mit Befestigung für Joystick, siehe Anhang A.3
4. Trennungsplattform zwischen Raspberry Pi und Arduino Nano, siehe Anhang A.4
5. Deckel der restlichen Elektronikbox, siehe Anhang A.5

Da die Details der Teile nicht von größerer Bedeutung sind, befinden sich Bilder der einzelnen Objekt im Anhang im Kapitel A.

Auf dem Boden der Elektronikbox befinden sich leichte Erhöhungen, wodurch der Raspberry Pi (4.43, 1.) und eine Platine zur Verbindung der Motortreiber (4.43, 5.) direkt auf den Boden geschraubt werden können. Für die Befestigung des Arduino Nano's (4.43, 8.), wird

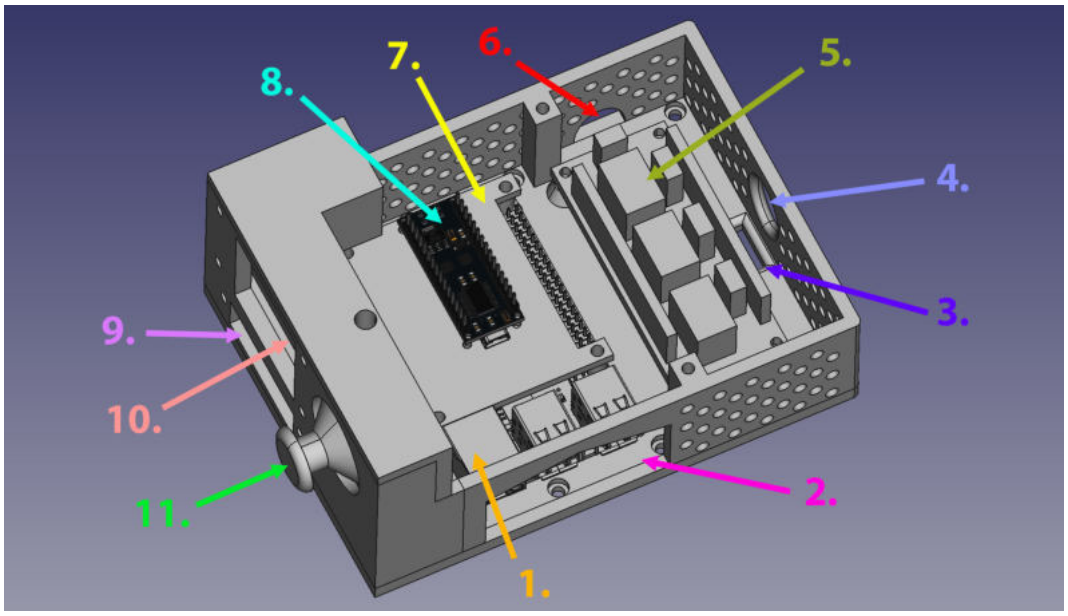

Abbildung 4.43.: Elektronikbox, Ansicht von oben ohne Deckel. Die Box ist in fünf Objekte geteilt. Boden mit Wänden. Front mit Löchern für einen Joystick (11.), ein Display (10.) und Zugang zu den seitlichen Eingängen (9.) des innen drin montierten Raspberry Pi (1.). Deckel über Display und Joystick. Plattform über Raspberry Pi für Montage des Arduino Nanos (8.). Nicht sichtbar auf dem Bild: Deckel über restlicher Elektronik. Die Wände sind für gute Luftzirkulation und Kühlung mit vielen Löchern bestückt. An der vorderen Seite ist ein großes Loch (2.), um an die USB-B Schnittstellen des Raspberry Pi's (1.) zu gelangen. Für die Kabel des Slider-Motors durch den Boden der Hauptplattform, ist ebenfalls in der Elektronikbox ein Loch vorhanden (3.). Für die Kabel des Pan-Motors ist ein Loch in der hinteren-seitlichen Wand positioniert (4.). Die Motortreiber zur Steuerung werden auf eine extra Platine gelötet und auf den Boden der Elektronikbox geschraubt (5.). Für die Kabel des Tilt-Motors ist an der hinteren Wand ein Loch so positioniert, dass es mittig des Befestigungsfußes des Tilt-Motors liegt (6.). Über dem montierten Raspberry Pi, wird zusätzlich zur Motor-Steuerung ein Arduino Nano (8.) verwendet. Dieser wird auf eine kleine Plattform (7.) geschraubt, welche auf den Raspberry Pi fixiert wird.

eine Plattform (4.43, 7.) zwischen den Arduino und den Raspberry positioniert. Um Zugriff auf die USB-B Schnittstellen des Raspberry Pi's zu erhalten, wird in die vordere Wand ein breites Loch konstruiert (4.43, 2.). Für die Kabel des Pan-Motors (4.43, 4.) und die Kabel des Tilt-Motors (4.43, 6.) wird an den anderen Wänden ebenfalls ein größeres Loch gelassen. Die restlichen kleinen Löcher in den Wänden sind für eine gute Luftzirkulation und der daraus resultierenden Kühlung zuständig. Da die Kabel des Slider-Motors durch ein Loch in der Hauptplattform kommen, wird im Boden der Elektronikbox ebenfalls ein Loch an der gleichen Position konstruiert (4.43, 3.). Das Display für die Steuerung wird an der Front befestigt (4.43, 10.). Unter dem Display befindet sich ein Loch um an die seitlichen Anschlüsse des Raspberry Pi's zu gelangen (4.43, 9.). Der Joystick wird an dem Deckel der Front befestigt (4.43, 11.).

4.1.5. Benötigte Hardware einkaufen

Für die Realisierung des Systems werden neben den selbstkonstruierten Teilen auch Teile benötigt, welche eingekauft werden müssen. Folgend eine tabellarische Auflistung der zusätzlich eingekauften Teile.

Bezeichnung	**Anzahl**
Nema 17 Schrittmotor 42Ncm 1.5A	2
Nema 17 Schrittmotor 16Ncm 1A	1
GT2 Zahnriemen, 6 mm Breite	min. 50 cm
GT2 Umlenkrollen, für 6 mm Zahnriemen	2
GT2 Zahnrad 20 Zähne, für 6 mm Zahnriemen	1
3x10x4 mm Flanschlager F623 (Kugellager)	2
TMC2208 Schrittmotortreiber	3
ST7735 1,8 Zoll Display	1
Joystick KY-023	1
Aluminiumrohr d = 20 mm, l = min. 50 cm	2
M3 Muttern, ISO 4035	min. 110
M3 Schrauben, DIN 912 in Längen 6-30 mm	min. 110

Zusätzlich wurde auf der online Plattform “https://easyeda.com“ eine Platine entwickelt, geätzt und bestellt. Diese ist dafür zuständig die Verkabelung der Motortreiber zu vereinfachen und somit redundante Kabel zu verhindern. Da dieser Teil nicht im Fokus der Arbeit liegt, befinden sich Bilder des zugehörigen Schaltplans (B.1) und die Anordnung der Teile (B.2) im Anhang.

4.1.6. Hardware drucken und zusammenbauen

Nach dem Konstruieren aller benötigten Teile, werden diese mit Hilfe von 3D-Druckern ausgedruckt. Die Objekte wurden in FreeCAD konstruiert und einzeln als **.stl* Datei exportiert. Da die **.stl* Dateien lediglich die Oberfläche der Objekte beschreiben, werden die Dateien in einem nächsten Schritt in dem Slicer *Ultimaker Cura* eingeladen und in Anweisungen für 3D-Drucker weiter verarbeitet. Mit den daraus entstehenden **.gcode* Dateien können 3D-Drucker die konstruierten Objekte ausdrucken. Alle Zahnräder, die statische Basis, die Hauptplattform, der Kugeleinführungsstopfen, der Abstandhalterin für die Kugeln zwischen statischer Basis und Hauptplattform, die Slider-Motor Befestigung und

die Befestigung an den Aluminiumrohren werden für eine möglichst hohe Präzision mit dem *Ultimaker 2+* gedruckt. Die Slider-Plattform, die Befestigungsfüße für die Kamerahaltung und der Boden der Elektronikbox werden mit dem *Renkforce RF2000v2* gedruckt. Alle anderen Teile der Elektronikbox werden mit dem *Creality Ender 3 V2* gedruckt.

Nach dem Ausdrucken der Komponenten werden alle Teile zusammengebaut. Dabei wird ein Modell aller konstruierten Teile als Vorlage genutzt. Für den Slider wurden 50 cm lange Aluminiumrohre verwendet.

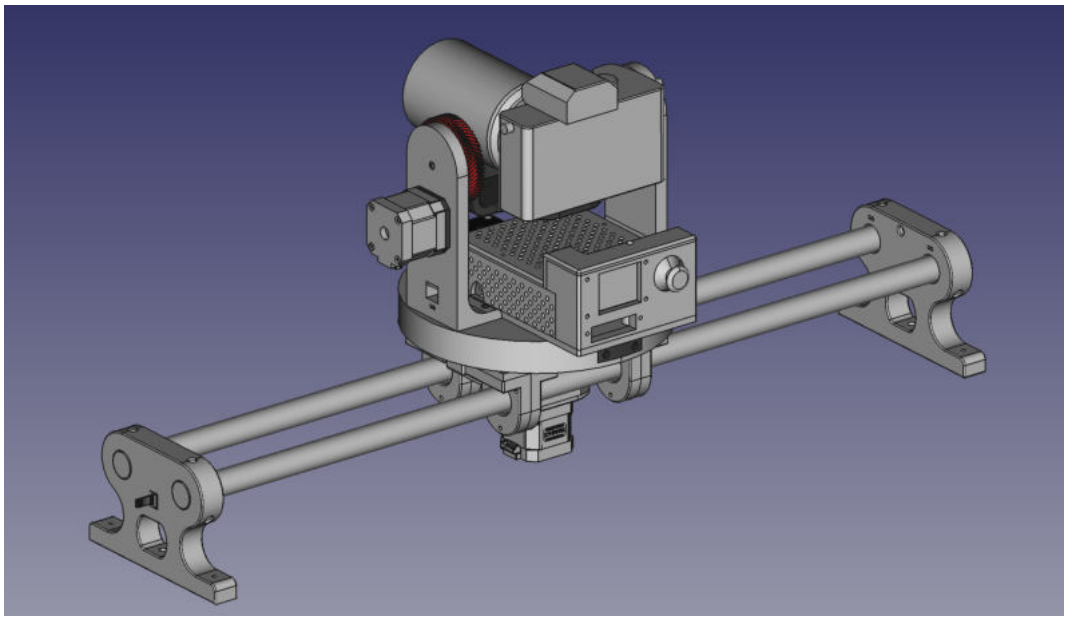

Abbildung 4.44.: Simulation des zusammengebauten Systems in FreeCAD

Abbildung 4.44 zeigt ein simuliertes Modell in FreeCAD. Abbildung 4.45 zeigt das fertige, zusammengebaute System von der Seite. Weitere Bilder des reellen Systems befinden sich im Anhang C.

4.2. Software

Der folgende Abschnitt erörtertet die Umsetzung der benötigten Software für das System. Die Software ist in zwei Bereiche unterteilt.

1. Hardwareansteuerung
2. Bildverarbeitung

Die Motoren, das Display und der Joystick werden über den Arduino Nano gesteuert. Dafür wird eine Hardwareansteuerung in der Programmiersprache C entwickelt. Da für diese Studienarbeit nur die Ansteuerung der Motoren relevant ist, wird auf die Ansteuerung der

Abbildung 4.45.: Zusammengebautes Systems, Ansicht von der Seite.

anderen Peripheriegeräte nicht eingegangen. Für das Einlesen des live-Kamerabildes und die Weiterverarbeitung der Bilddaten in Motor-Bewegungen, wird ein leistungsstärkerer Raspberry Pi 3B+ und die Programmiersprache *Python* verwendet. Im folgenden wird die Gesichtserkennung, sowie der prinzipielle Ansatz der Weiterverarbeitung eines erkannten Gesichts in Motor-Bewegungen erörtert. Das vollständige Python-Programm ist im Anhang D.3 ersichtlich.

4.2.1. Anforderungen an die Software stellen

Anforderungen an die Hardwaresteuerung auf dem Arduino Nano

1. Führe Pan-, Tilt- und Slider-Bewegung in bestimmter Zeit mit definierter Anzahl an Schritten pro Motor durch.
2. Eingegrenzte Motorbewegungen, sodass Motoren nicht weiter drehen wenn es nicht mehr sinnvoll ist.
3. Serielle Kommunikation zwischen Arduino und Raspberry Pi.
4. Befehle über serielle Schnittstelle erhalten und in Motor-Bewegungen umsetzen.

Anforderungen an die Bildverarbeitung auf dem Raspberry Pi

1. Einlesen des Live-Kamerabildes.
2. Gesichter in Bild erkennen und Position des nachzuverfolgenden Gesichts bestimmen.
3. Position des nachzuverfolgenden Gesichts in Motorbewegungen umwandeln.
4. Serielle Kommunikation zwischen Arduino und Raspberry Pi.
5. Motorbewegungen über serielle Schnittstelle an Arduino senden.

4.2.2. Hardwareansteuerung implementieren

Die Hardwareansteuerung wird mit der Programmiersprache C in der Arduino IDE mit der Versionsnummer 1.8.13 implementiert. Die Software ist in zwei Teile gegliedert. Der erste Teil umfasst den ausführbaren Hauptteil als Arduino-C-Programm, welcher die Initialisierung, die Endlosschleife des Programms sowie die serielle Kommunikation mit dem Raspberry Pi beinhaltet. Der zweite Teil realisiert die Motorsteuerung.

Im folgenden wird die Endlosschleife des Arduino Programms erörtert. Innerhalb der Schleife wird in jedem Durchgang überprüft, ob ein Signal an den Arduino gesendet wird (vgl. 4.1, Zeile 4). Ist dies der Fall, wird das Signal als *String* eingelesen. Der String wird dabei solange eingelesen, bis das Symbol “E“ gesendet wird (vgl. 4.1, Zeile 5). Dass das Symbol “E“ das Ende eines Befehls angibt, ist hierbei eine eigens gewählte Konvention. Des Weiteren wurde festgelegt, dass erstmal nur Befehle für die Pan- und Tilt-Bewegung übertragen werden sollen. Eine Motorbewegung wird dabei durch die Anzahl der auszuführenden Schritte und die Zeit in Millisekunden in welcher diese ausgeführt werden sollen bestimmt. Beide Werte werden als fünfstellige Zahlen jeweils für die Pan- und Tilt-Bewegung übertragen (vgl. 4.1, Zeilen 6-9). Für die Weiterverarbeitung innerhalb der Motorsteuerung werden die Werte in Arrays gespeichert (vgl. 4.1, Zeilen 11-12) und folgend mit einem Funktionsaufruf übergeben (vgl. 4.1, Zeile 13). Zu beachten ist, dass in den Arrays jeweils eine dritte Angabe mit dem Wert 0 hinterlegt ist. Dieser wäre für die auszuführenden Schritte des Slider-Motors und die dazu verwendete Zeit zuständig.

```
1 void loop() {
2   String command_raw;
3
4   if (Serial.available()) {
5     command_raw = Serial.readStringUntil("E");
6     int steps_pan = command_raw.substring(0, 5).toInt();
7     int steps_tilt = command_raw.substring(5, 10).toInt();
```

```
8       int time_pan = command_raw.substring(10, 15).toInt();
9       int time_tilt = command_raw.substring(15, 20).toInt();
10
11      int steps[3] = {steps_pan, steps_tilt, 0};
12      long times[3] = {time_pan, time_tilt, 0};
13      moveall(steps, times);
14    }
15  }
```

Listing 4.1: Main-Loop aus dem Arduino Code

Die Motorsteuerung ist in vier Teile gegliedert.

1. motor_setup()
2. moveall(...)
3. convert_steps(...)
4. single_motor_step(...)

Sobald das Hauptprogramm ausgeführt wird, wird das *motor_setup* der Motorsteuerung ausgeführt. Dieses ist dafür zuständig die Pin-Belegung der Motoren am Arduino Nano zu definieren. Jeder Motor hat ein Pin, um einen Motorschritt durchzuführen. Ein Schritt wird durchgeführt durch Wechseln des anliegenden Potentials von *HIGH* zu *LOW* oder umgekehrt. Die Richtung des Schritts wird mit einem anderen Pin und dessen Potential definiert.
Sobald in der Endlosschleife des Hauptprogramms ein Befehl angekommen ist, wird die *moveall(...)* Methode aufgerufen und die Motorbewegungen werden übergeben. Innerhalb der *moveall(...)* Methode wird als erstes die Anzahl der auszuführenden Schritte mit Hilfe der *convert_steps(...)* Methode überprüft und verarbeitet. Um die Motoren in beide Richtungen zu drehen, werden innerhalb der Befehle positive oder negative Werte für je eine Richtung verwendet. Da ein Motorschritt nicht negativ sein kann und die Richtung alleine durch den Pin-Zustand des Richtungspins definiert ist, wird in der Hilfsmethode überprüft, ob die Anzahl der Schritte negativ oder positiv ist. Je nachdem wird das Potential am Pin für die Richtung verändert und für alle weiteren Schritte der Betrag der Anzahl der Schritte verwendet. Des Weiteren wird anhand der aktuellen Position des Motors überprüft, ob durch das Ausführen der erhaltenen Schritte Grenzen des Bewegungsfreiraums überschritten werden. Ist dies der Fall, wird die Anzahl der Schritte so verkürzt, dass die Grenzen eingehalten werden. Nachfolgend wird ermittelt, wie lange welcher Motor zwischen einem Schritt warten muss, um die Zeit für die auszuführenden Schritte einzuhalten. In dem nächsten Schritt wird eine Schleife solange ausgeführt, bis alle Motoren Ihre auszuführenden Schritte erledigt haben. In dieser Schleife führt jeder

Motor innerhalb der davor ermittelten Frequenz einen Schritt aus. Das Ausführen eines Schrittes wird mit der Hilfsmethode *single_motor_step(...)* durchgeführt. Diese überprüft das momentane Potential des jeweiligen Motors und setzt es dann auf das Gegengesetze. Zudem wird innerhalb der Methode die aktuelle Postion des Motors festgehalten. Der Sourcecode der Motorsteuerung ist im Anhang D.2 verfügbar.

4.2.3. Gesichtserkennung implementieren

Die Gesichtserkennung ist eine Methode der implementierten Python-Software auf dem Raspberry Pi. Die Methode erhält als Parameter das aktuelle live-Bild der Kamera und gibt, falls vorhanden, die Position eines Gesichtes zurück. Für die Gesichtserkennung wird die open-source-Bibliothek *OpenCV* genutzt.

```
1 haar_cascade_face = cv2.CascadeClassifier('
      haarcascade_frontalface_default.xml')
2 eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
3
4 ...
5
6 # Applying classifiers to detect faces and eyes
7 faces_rect = haar_cascade_face.detectMultiScale(gray_image, ...)
8 eyes_rect = eye_cascade.detectMultiScale(gray_image)
```

Listing 4.2: Auszug aus der Methode zur Gesichtserkennung. Gesichts- und Augenerkennung mit Hilfe von OpenCV

Für die Gesichtserkennung wird der *haarcascade_frontalface_default.xml* Klassifizierer der OpenCV Bibliothek verwendet (vgl. 4.2, Zeile 1). Da die Klassifizierung von Gesichtern nicht zuverlässig ist und häufig Gesichter in Bildern erkannt werden wo keine sind, wird zusätzlich ein Klassifizierer verwendet, welcher Augen in einem Bild detektiert (vgl. 4.2, Zeile 2). Beide Klassifizierer verwenden in einem nächsten Schritt ein Graubild, um auf diesem Gesichter und Augen zu erkennen (vgl. 4.2, Zeile 7-8). Als Rückgabe wird jeweils eine Liste der erkannten Objektpositionsdaten im Bild erstellt.

Um die Gesichtserkennung präziser zu gestalten, wird als nächstes überprüft, ob sich mindestens zwei Augen in einem Gesicht befinden. Durch Testen wurde herausgefunden, dass in einem echten Gesicht häufig mehr als zwei Augen erkannt werden. Daher ist es notwendig auf *mindestens* zwei Gesichter zu überprüfen. Da es vorkommen kann, dass mehr als eine Person in dem Kamerabild ist, ist eine Entscheidung notwendig, welches erkannte Gesicht verfolgt werden soll. In diesem Fall wird immer das größte erkannte

Gesicht verwendet, um verfolgt zu werden.

Der Sourcecode der Gesichtserkennung ist im Anhang D.3 unter dem Methodennamen “detect_and_return_rectangles“ ersichtlich.

4.2.4. Algorithmus zur Verarbeitung der Position des Gesichts zu Motorbewegungen entwickeln und implementieren

Nachdem ein nachzuverfolgendes Gesicht erkannt und die Position bestimmt wurde, wird anhand dieser Information entschieden, wie sich die Motoren drehen müssen, um die Kamera so zu drehen, dass das Gesicht wieder mittig im Bild zu sehen ist.

Dafür wird als erstes überprüft, ob ein nachzuverfolgendes Gesicht erkannt wurde. Als nächstes wird kontrolliert, ob das Gesicht bereits mittig positioniert ist. Um ein hin und her Zittern des Systems zu vermeiden, wird dafür die Mitte nicht als einzelner Punkt sondern als kleiner Bereich definiert. Dadurch bewegt sich das System nur, wenn es wirklich notwendig ist. Falls das Gesicht nicht mittig ist, wird untersucht, ob das Gesicht näher dem rechten oder dem linken Bildrand ist. Ebenso ob es näher dem oberen oder dem unteren Bildrand positioniert ist. Diese Informationen sind für die Richtungsbestimmung der Motoren relevant. Für die Anzahl der Schritte der Motoren wird der Abstand des Gesichts zum Rand des Bildes, sowie die Zahnradübersetzung des jeweiligen Motors mitberücksichtigt.

Der Sourcecode der Weiterverarbeitung von Gesichts-Positionsdaten zu Motor-Bewegungen befindet sich im Anhang D.3 unter dem Methodenname “process_rectangle_to_steps“.

4.3. Testen

4.3.1. Hardware auf Anforderungen testen

Zuerst wird überprüft, ob die Anforderungen der Pan-Bewegung des Abschnitts 4.1.1 eingehalten wurden.

Das Testen der Hardware erfolgt durch eine Überprüfung der Anforderungen aus dem Abschnitt 4.1.1.

Es wurden alle Anforderungen erfüllt. Beachtenswert ist, dass der Drehbereich der Tilt-Bewegung lediglich durch die Größe der Kamera eingeschränkt ist. Ohne eine Kamera

Anforderungsbeschreibung	**Sollwert**	**Überprüfter Wert**
Pan-Bewegung, Drehbereich	$\geq 360°$	720°
Pan-Bewegung, Verstellgenauigkeit	$\leq 0.5°$/ Schritt	0.05°/ Schritt
Pan-Bewegung, Geschwindigkeit	$\geq 180°$/ 3 Sekunden	360°/ 3 Sekunden
Tilt-Bewegung, Drehbereich	$\geq \pm 30°$	$-31°$/ $+62°$
Tilt-Bewegung, Verstellgenauigkeit	$\leq 0.5°$/ Schritt	0.15°/ Schritt
Tilt-Bewegung, Geschwindigkeit	$\geq 90°$/ 1 Sekunden	360°/ 1 Sekunden
Bewegung im Raum, Strecke	$\geq$ 50cm	50cm
Bewegung im Raum, Verstellgenauigkeit	$\leq$ 1mm / Schritt	0.082 mm / Schritt
Bewegung im Raum, Geschwindigkeit	$\leq$ 10cm / Sekunde	$\leq$ 28cm / Sekunde
Gewicht der zu bewegenden Kamera	$\geq$ 1.1kg	1.1kg

könnte die Tilt-Bewegung ohne Einschränkungen immer weiter ausgeführt werden. Des Weiteren ist bei dem Testen der Hardware aufgefallen, dass die Pan-Bewegung nicht zu jeder Zeit flüssig läuft. Manchmal treten minimale Ruckler und Verzögerungen in der Bewegung auf. Diese sind vermutlich dem Kugellager zwischen statischer Basis und Hauptplattform geschuldet. Voraussichtlich dadurch, dass im Kugellager ein Loch zur Befüllung vorhanden ist und die Kugeln somit an den Kanten des Stopfens entlang reiben. Dadurch ergibt sich, dass die Verstellgenauigkeit vermutlich nicht zu jedem Zeitpunkt den angegeben Wert von 0.05° erfüllen kann. Der Wert ist dadurch entstanden, dass für eine komplette Umdrehung der Hauptplattform 6800 Schritte benötigt werden.

4.3.2. Software auf Anforderungen testen

Zuerst wurde die Hardwareansteuerung getestet. Motorbewegungen sind problemlos möglich und können mit einer bestimmten Anzahl an Schritten in einer gewissen Zeit durchgeführt werden. Grenzen der Bewegungen wurden implementiert und die Motoren stoppen sobald eine Grenze erreicht wurde. Voraussetzung dafür ist, dass das System mit der Kamera gerade an einer Seite des Sliders gestartet wurde. Die Kommunikation zwischen dem Arduino Nano und dem Raspberry Pi funktioniert und Befehle können übersetzt werden in Motor-Bewegungen.

Die Bildverarbeitung funktioniert und Gesichter werden durch das zusätzliche Kombinieren der Augenerkennung präzise erkannt. Als Beispiel ist in Abbildung 4.46 das erkannte

Gesicht mit den Augen erkannt worden und durch Rechtecke gekennzeichnet. Dabei erfolgte die Kennzeichnung durch die programmierte Software.

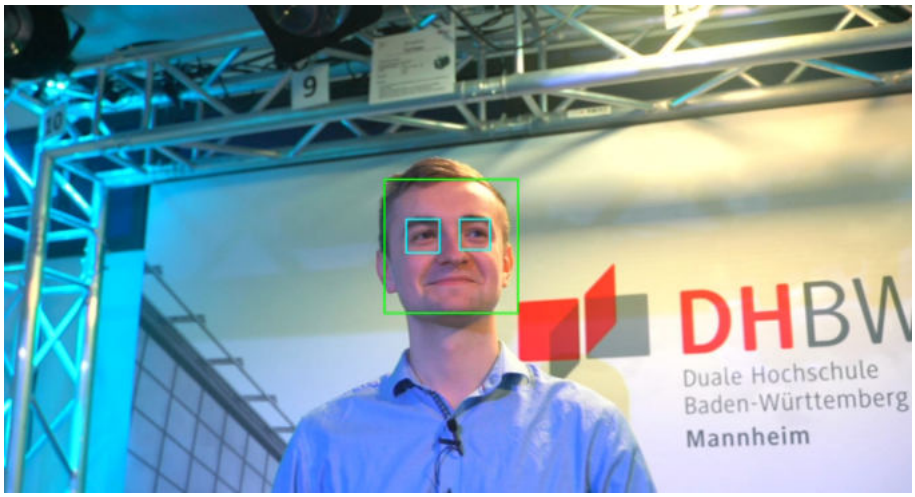

Abbildung 4.46.: Gesichtserkennung und Augenerkennung sichtbar durch eingezeichnete Rechtecke im Bild.

Die Gesichtspositionsdaten wurden erfolgreich in Motor-Bewegungen umgewandelt und über die serielle Schnittstelle an den Arduino gesendet.

5. Ergebnis

Es wurden alle Aufgaben der Aufgabenstellung erfolgreich bearbeitet. Die Tests des Systems haben gezeigt, dass das entwickelte System in der Lage ist, Gesichter zu erkennen und die Kamera so zu bewegen, dass ein Gesicht mittig im Bild gehalten wird. Das Verfolgen ist dabei jedoch recht ruckelig und könnte noch verbessert werden. Ursprünglich wurde überlegt, dass das System in der Lage sein soll, ein Gesicht mit Hilfe eines Sliders immer frontal zu filmen. Dies wurde aufgrund des sonst zu großen Aufwands nicht umgesetzt. Die benötigte Hardware ist jedoch vorhanden, sodass für die Umsetzung lediglich die Software dazu fehlt.

Alle Ergebnisse können unter Vorbehalt in folgendem Git-Repository eingesehen werden: *https://github.com/JuliusKlodt/TrackingSystem*

Nachfolgend eine Tabelle mit einer Gegenüberstellung der Aufgabenstellung und deren Bearbeitungsstatus.

Aufgabenbeschreibung	**Bearbeitungsstatus**
1. Hardware	
1.1 Präzise Anforderungen an die Hardware stellen	✓
1.2 Vorhandene Projekte sichten und Umsetzungsideen sammeln	✓
1.3 Konzepte erstellen und Hardware konzipieren	✓
1.4 Hardware konstruieren	✓
1.5 Benötigte Hardware einkaufen	✓
1.6 Konstruierte Teile mit 3D-Drucker drucken und nachbearbeiten	✓
1.7 Konzepte erstellen und Hardware konzipieren	✓
2. Software	
2.1 Anforderungen an die Software stellen	✓
2.2 Hardwareansteuerung implementieren Lösen des Würfels planen	✓
2.3 Gesichtserkennung implementieren	✓
2.4 Algorithmus zur Verarbeitung der Position des Gesichts zu Motorbewegungen	✓
3. Testen	
3.1 Hardware auf Anforderungen testen	✓
3.2 Software auf Anforderungen testen	✓

6. Reflektion

Die Studienarbeit war zusammenfassend ein voller Erfolg. Ich habe gelernt mit Hilfe der Software FreeCAD präzise Modelle zu erstellen und diese mit einem 3D-Drucker auszudrucken. Dabei habe ich viele Erfahrungen gesammelt und weiß nun was man bei dem Konstruieren von Objekten beachten muss. Bei dem Drucken von Objekten entsteht sehr häufig viel Müll des verwendeten Druck-Materials PLA. Daher habe ich mir Gedanken dazu gemacht, wie man diesen Müll möglichst nachhaltig verwerten kann und was mit dem Müll passieren würde. In diesem Zuge ist der Gedanke aufgekommen, dass es am besten wäre, wenn man den Müll oder sonstige alte Drucke recyclen und wiederverwenden könnte. Dies bietet zum Beispiel die Firma “Recycling Fabrik UG“ auf Ihrer Website *https://recyclingfabrik.com/* an. An diese Firma werden alle Fehldrucke und jeglicher anderer PLA-Müll welcher im Zuge dieser Studienarbeit entstanden ist, gesendet.

7. Ausblick

Im Rahmen dieser Studienarbeit wurde erfolgreich ein System entwickelt, welches eine Kamera so dreht, dass ein Gesicht immer mittig im Bild verfolgt wird. Im Laufe der Arbeit sind zusätzlich noch viele Möglichkeiten und Verbesserungen entstanden.

Es könnten Sensoren dazu verwendet werden, sodass das System sich selbst kalibriert und in einen definierten Anfangszustand fährt. Des Weiteren könnten Sensoren am System helfen, dass System in unebenen Untergründen zu verwenden und die Kamera trotzdem immer gerade zu halten.

Des Weiteren könnte die Motorsteuerung verbessert werden. Momentan wartet die Software zwischen den einzelnen Schritten der Motoren aktiv auf den nächsten Schritt. Somit kann die Software nicht auf andere Einflüsse in dieser Zeit reagieren. Dies könnte mit Hilfe von Interrupt-Routinen behoben werden.

Eine weitere Verbesserungsmöglichkeit besteht darin, das Verfolgen eines Gesichts flüssiger zu gestalten. Hinzu könnte das System dahingehend verbessert werden, dass es Gesichter nicht nur erkennt, sondern auch Gesichter zuordnen kann. Dadurch könnte gewährleistet werden, dass selbst wenn viele Personen im Bild sind, immer nur eine bestimmte Person verfolgt wird.

8. Danksagung

Zum Schluss bedanke ich mich ausdrücklich bei Jürgen Schultheis, welcher mich als Betreuer dieser Arbeit zu jeder Zeit in allen Angelegenheiten bestmöglich unterstützt hat. Des Weiteren bedanke ich mich bei Thomas Baumgärtner und Wassilij Kaiser, welche 3D-Drucker und Laborräume der DHBW Mannheim für mich zur Verfügung stellten.

Literatur

[1] Eyal Abraham. *Differential pan-tilt*. Mai 2020. URL: https://www.youtube.com/watch?v=LF1JT0Pmddc (besucht am 12. 03. 2022).

[2] FreeCAD. *FreeCAD - Offizielle Website*. URL: https://www.freecadweb.org/ (besucht am 07. 03. 2022).

[3] Tobias Häberlein. *Technische Informatik - Ein Tutorium der Maschinenprogrammierung und Rechnertechnik*. Vieweg+Teubner, 2011. ISBN: 978-3-8348-1372-5.

[4] Isaac879. *3D Printed DSLR Camera Pan Tilt Mount (Arduino/Stepper Driven) 2020*. Mai 2020. URL: https://www.youtube.com/watch?v=uJO7mv4-0PY (besucht am 12. 03. 2022).

[5] Isaac879. *Pan-Tilt-Mount, GitHub*. März 2021. URL: https://github.com/isaac879/Pan-Tilt-Mount (besucht am 16. 03. 2022).

[6] Python. *Python - Offizielle Website*. URL: https://www.python.org/about/ (besucht am 06. 03. 2022).

[7] ServoCity. *ServoCity Products: PT2645S Pan and Tilt System*. Juli 2017. URL: https://www.youtube.com/watch?v=EfovZizgfx8 (besucht am 12. 03. 2022).

[8] Wikipedia. *C-Programmiersprache*. URL: https://de.wikipedia.org/wiki/C_(Programmiersprache) (besucht am 06. 03. 2022).

[9] Wikipedia. *Pan*. URL: https://en.wikipedia.org/wiki/Panning_(camera) (besucht am 03. 03. 2022).

[10] Wikipedia. *Tilt*. URL: https://en.wikipedia.org/wiki/Tilt_(camera) (besucht am 02. 03. 2022).

[11] Wikipedia. *Ultimaker Cura - Wikipedia Artikel*. URL: https://www.freecadweb.org/ (besucht am 07. 03. 2022).

A. Elektronikbox Komponenten

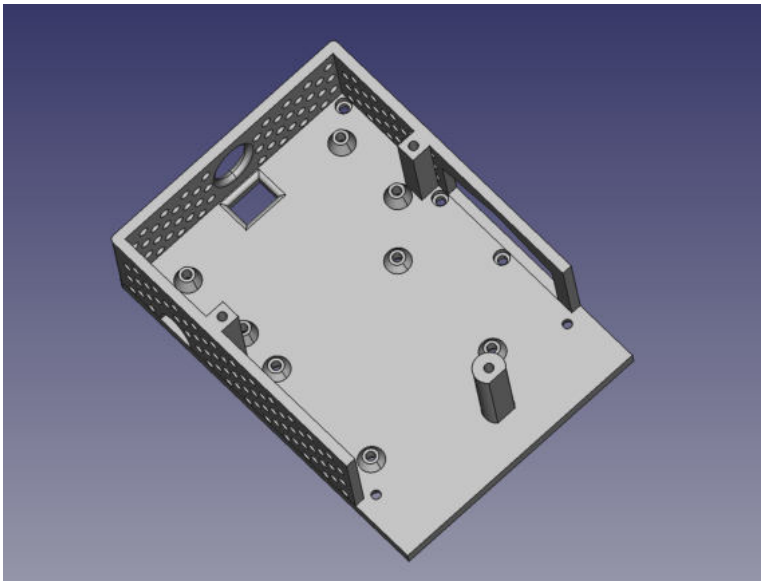

Abbildung A.1.: Elektronikbox - Boden mit Wänden

.

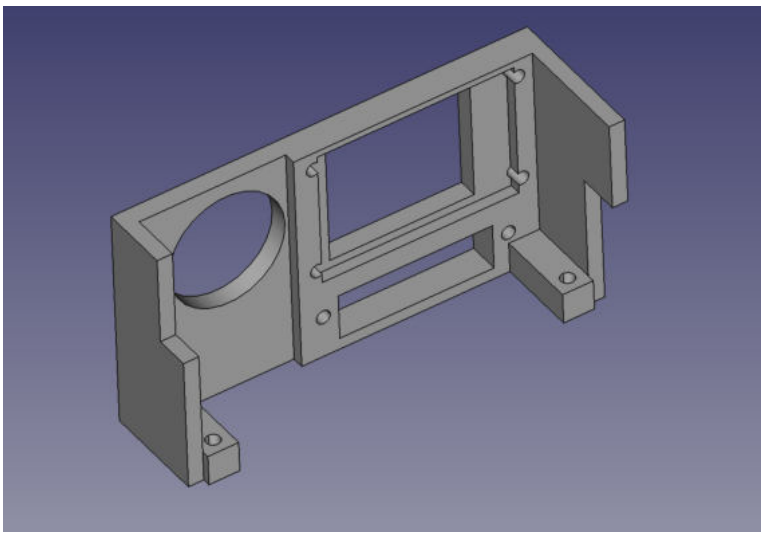

Abbildung A.2.: Elektronikbox - Front mit Löchern für Joystick, Display und Durchgang zu seitlichen Anschlüssen des Raspberry Pi's

.

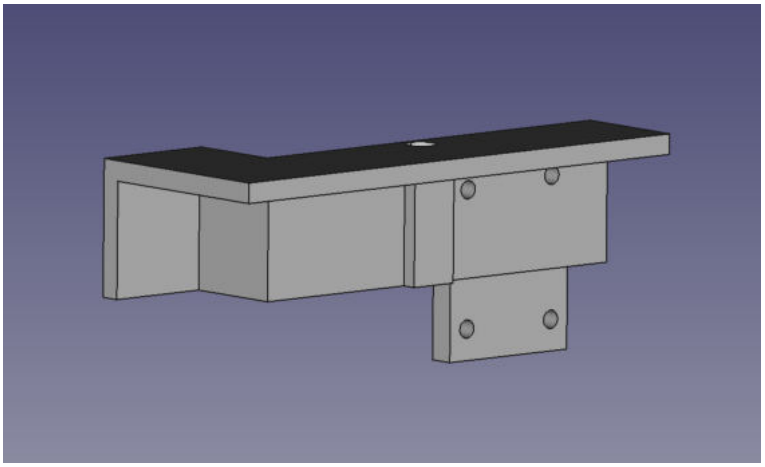

Abbildung A.3.: Elektronikbox - Deckel der Front mit Befestigungsmöglichkeit für Joystick

.

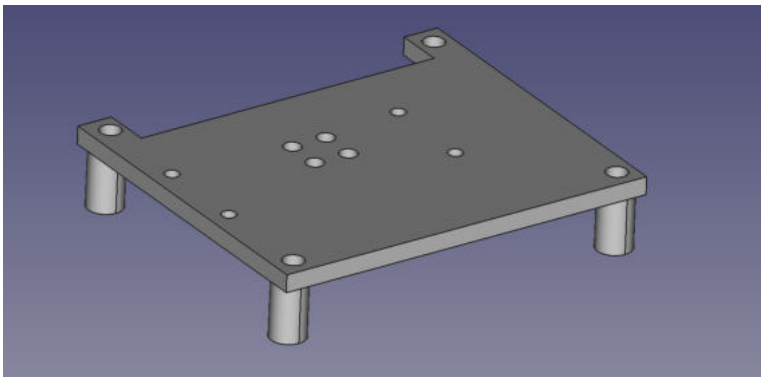

Abbildung A.4.: Elektronikbox - Trennungsplattform zwischen Raspberry Pi und Arduino Nano

.

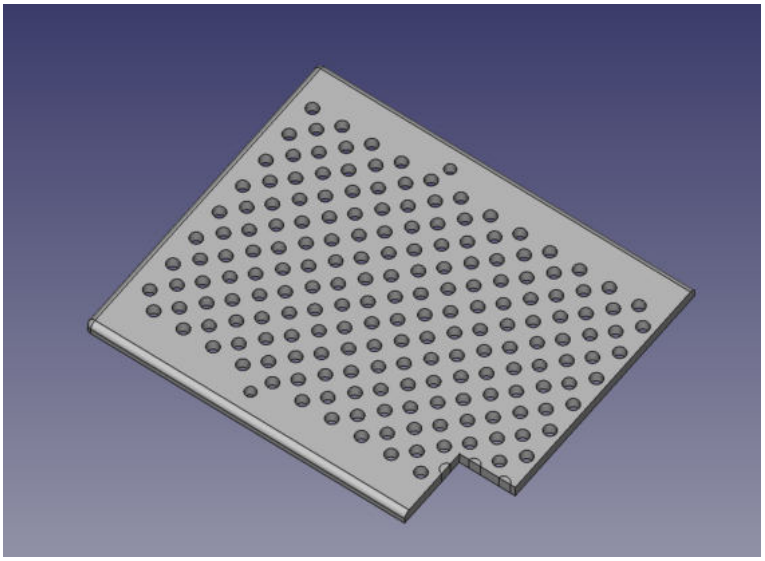

Abbildung A.5.: Elektronikbox - Deckel der restlichen Elektronikbox

.

B. Platine für Motortreiber

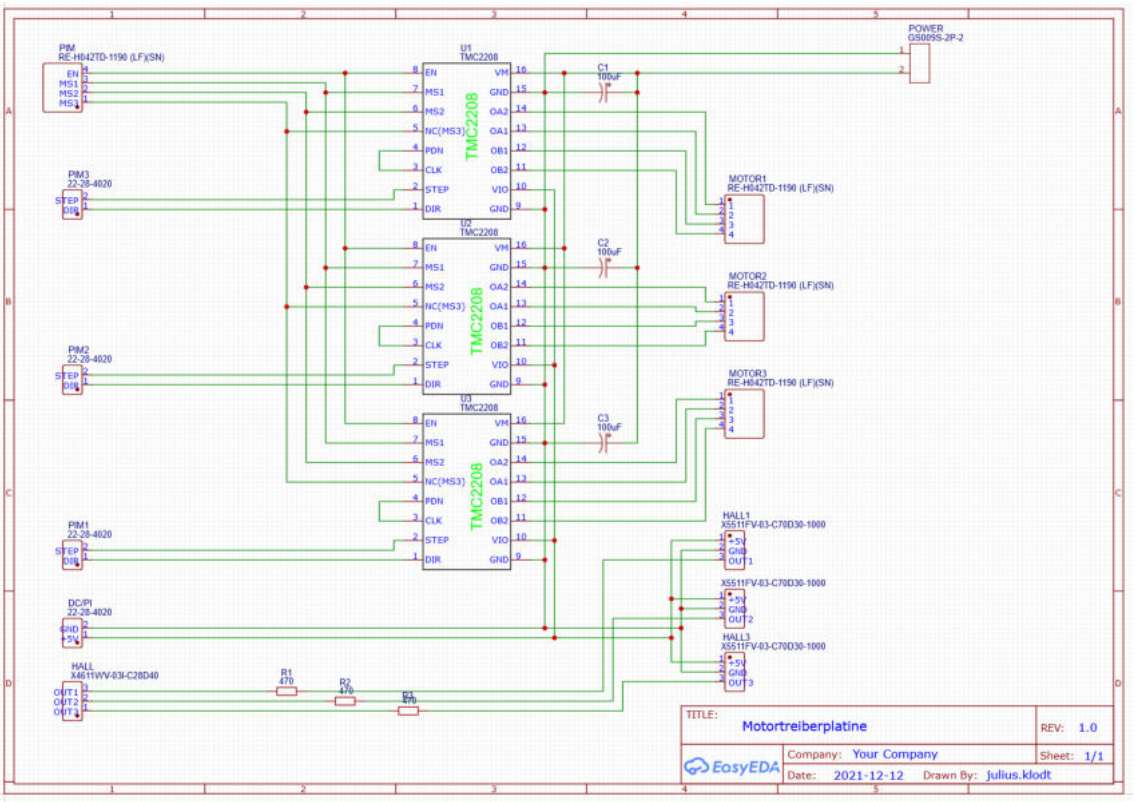

Abbildung B.1.: Schaltplan der entwickelten Platine zur Verbindung der Motortreiber. Als Erweiterungsmöglichkeit wurden zusätzlich drei Schnittstellen für A3144 Hall-Effect-Sensoren in die Platine integriert. Diese können mit Hilfe von Magneten dafür genutzt werden, das System autonom zu kalibrieren.

.

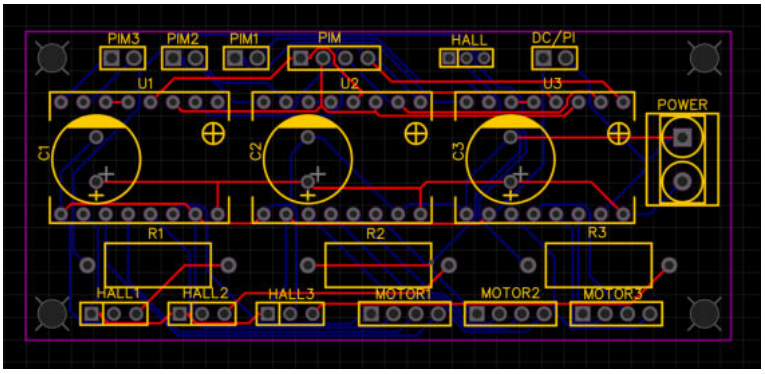

Abbildung B.2.: Anordnung der Bauteile auf der Platine für die Verbindung der Motortreiber.

.

C. Bilder des zusammengebauten Systems

Abbildung C.1.: Fertiges System mit Kamera, Ansicht von oben.

.

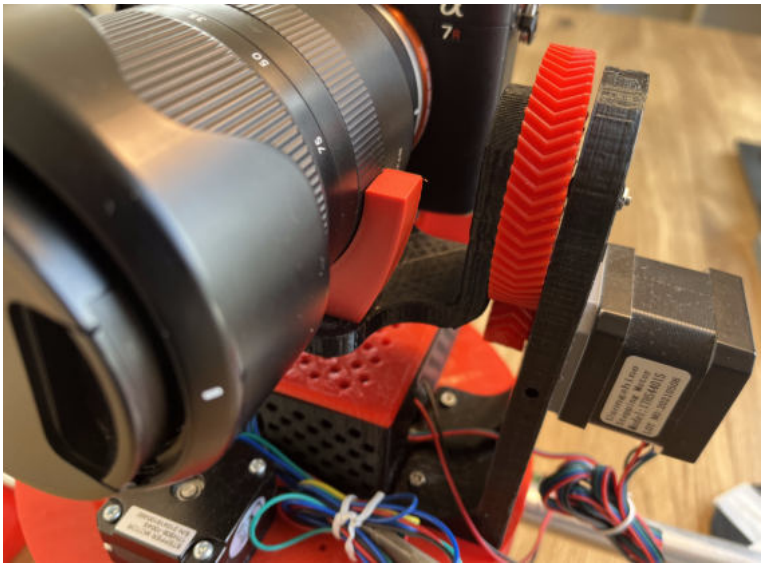

Abbildung C.2.: Fertiges System mit Kamera, Detailansicht von der Seite.

.

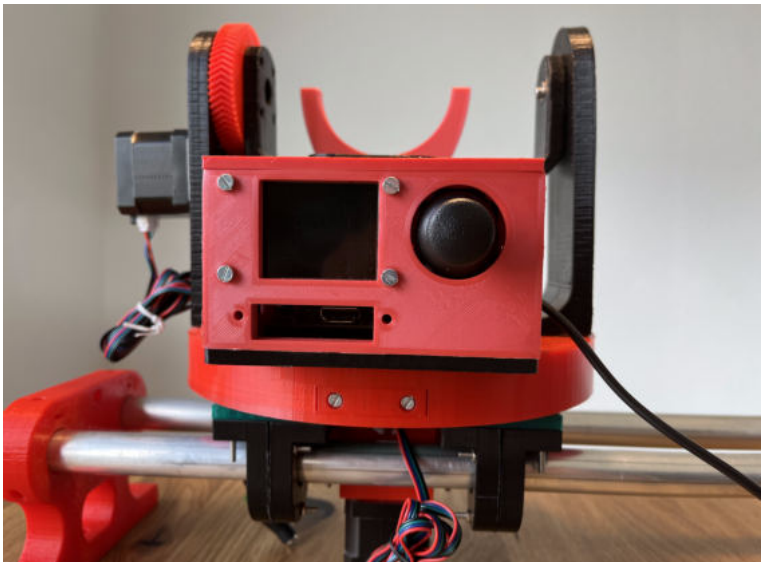

Abbildung C.3.: Fertiges System ohne Kamera, Ansicht von vorne.

.

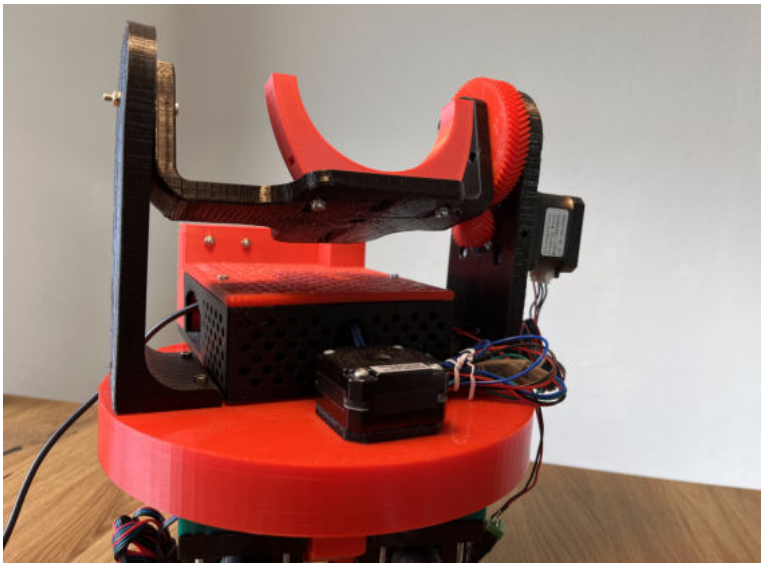

Abbildung C.4.: Fertiges System ohne Kamera, Ansicht von hinten.

.

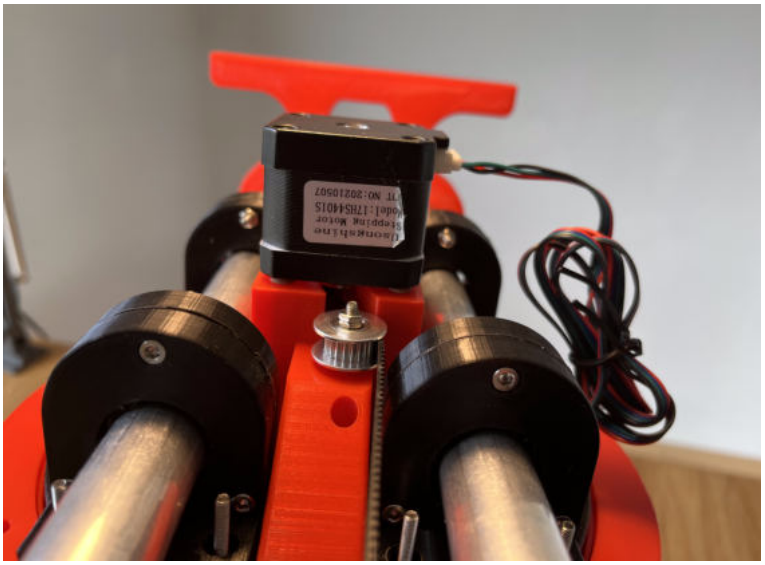

Abbildung C.5.: Fertiges System ohne Kamera, Ansicht von unten auf die Befestigung an die Aluminiumröhre.

.

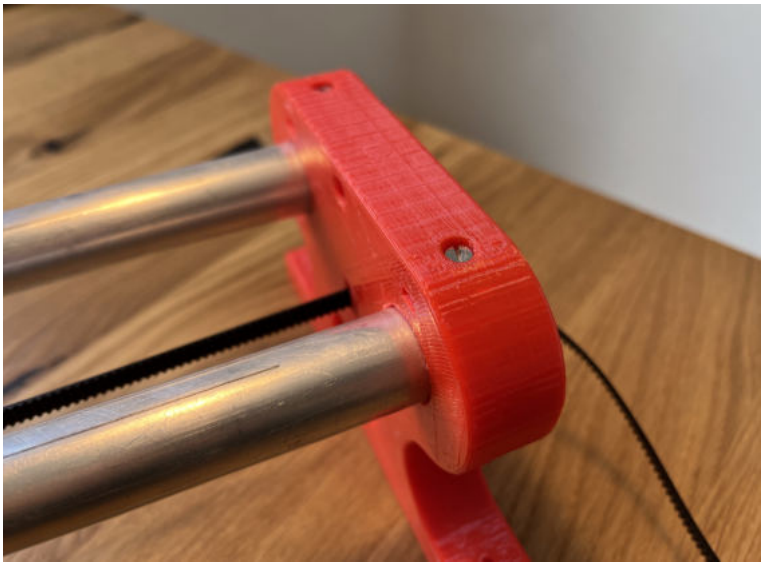

Abbildung C.6.: Detailansicht auf die Fixierung der Aluminiumröhre mit einem Befestigungsfuß

.

D. Software

D.1. Arduino-Programm

```
#include<stdio.h>
#include"Motors.h"

void setup() {
  Serial.begin(250000);
  Serial.setTimeout(1);
  motor_setup();
}

void loop() {
  String command_raw;

  if (Serial.available()) {
    command_raw = Serial.readStringUntil("E");
    int steps_pan = command_raw.substring(0, 5).toInt();
    int steps_tilt = command_raw.substring(5, 10).toInt();
    int time_pan = command_raw.substring(10, 15).toInt();
    int time_tilt = command_raw.substring(15, 20).toInt();

    int steps[3] = {steps_pan, steps_tilt, 0};
    long times[3] = {time_pan, time_tilt, 0};
    moveall(steps, times);
  }
}
```

Listing D.1: Arduino Programm

D.2. Motorsteuerung

```
#ifndef Motors
#define Motors

#include<math.h>

#define pin_motor_en 12

```

```
int pin_motors_dir[3] = {4, 16, 15};
int pin_motors_step[3] = {6, 7, 5};

int min_pos[3] = { -3400, -210, -6100};
int max_pos[3] = {3400, 420, 0};

int current_pos[3];
int dir_motors[3];
int step_status_motors[3];

int convert_steps(int, int);
void single_motor_step(int);


void motor_setup() {
  pinMode(pin_motor_en, OUTPUT);
  digitalWrite(pin_motor_en, LOW);

  for (int i = 0; i < 3; i++) {
    pinMode(pin_motors_step[i], OUTPUT);
    pinMode(pin_motors_dir[i], OUTPUT);
    dir_motors[i] = 1;
    digitalWrite(pin_motors_dir[i], HIGH);
  }
}

int convert_steps(int akt_motor, int steps) {
  if (steps < 0) {
    digitalWrite(pin_motors_dir[akt_motor], LOW);
    dir_motors[akt_motor] = 0;
    if (current_pos[akt_motor] + steps < min_pos[akt_motor]) {
      steps = min_pos[akt_motor] - current_pos[akt_motor];
    }
  } else {
    digitalWrite(pin_motors_dir[akt_motor], HIGH);
    dir_motors[akt_motor] = 1;
    if (current_pos[akt_motor] + steps > max_pos[akt_motor]) {
      steps = max_pos[akt_motor] - current_pos[akt_motor];
    }
  }
  return fabs(steps);
}

void single_motor_step(int akt_motor) {
  if (step_status_motors[akt_motor]) {
    digitalWrite(pin_motors_step[akt_motor], LOW);
    step_status_motors[akt_motor] = 0;
```

```
55    } else
56    {
57      digitalWrite(pin_motors_step[akt_motor], HIGH);
58      step_status_motors[akt_motor] = 1;
59    }
60
61    if (dir_motors[akt_motor]) {
62      current_pos[akt_motor]++;
63    } else {
64      current_pos[akt_motor]--;
65    }
66  }
67
68  void moveall (int steps_per_motor[], long time_per_motor[])
69  {
70    int steps_done[3] = {0, 0, 0};
71
72    // process raw count of steps (negative steps are possible before)
73    for (int i = 0; i < 3; i++) {
74      steps_per_motor[i] = convert_steps(i, steps_per_motor[i]);
75    }
76
77    // calculate time per step
78    int delay_time[3] = {time_per_motor[0]/steps_per_motor[0],
        time_per_motor[1]/steps_per_motor[1], time_per_motor[2]/
        steps_per_motor[2]};
79    for(int i = 0; i < 3;i++){
80      if(delay_time[i] == 0){
81        delay_time[i] = 1;
82      }
83    }
84
85    long last_millis = 0;
86    // motor movement
87    while (steps_done[0] != steps_per_motor[0] || steps_done[1] !=
        steps_per_motor[1] || steps_done[2] != steps_per_motor[2] ) {
88      // prohibit loop-passes while "millis()" gives the same current time
89      long akt_millis = millis();
90      while (last_millis == akt_millis)
91      {
92        akt_millis = millis();
93      }
94      last_millis = akt_millis;
95
96      for (int i = 0; i < 3; i++) {
97        if (steps_done[i] < steps_per_motor[i])
98          if (last_millis % delay_time[i] == 0) {
```

```
99            single_motor_step(i);
100           steps_done[i]++;
101         }
102     }
103   }
104 }
105
106 #endif
```

Listing D.2: Motorsteuerung als *header*-Datei

D.3. Python-Programm zur Gesichtserkennung und Weiterverarbeitung

```
1  from os import replace
2  from sys import _current_frames
3  import serial
4  import cv2 as cv2
5
6  arduino = serial.Serial(port='COM12', baudrate=250000, timeout=.1)
7  image_size_x = 1920
8  image_size_y = 1080
9
10 video_capture = cv2.VideoCapture(1)
11 video_capture.set(3, image_size_x)
12 video_capture.set(4, image_size_y)
13
14 pan_gear = (17 / 144) * (360 / 400)
15 tilt_gear = (21 / 64) * (360 / 400)
16
17 # process of face tracking, processing into motor movements, send motor
       movements and display live video
18 def tracking():
19     while True:
20         ret, current_image = video_capture.read()
21
22         real_faces, other_objects, tracked_face =
       detect_and_return_rectangles(current_image)
23         steps_x, speed_x, steps_y, speed_y = process_rectangle_to_steps(
       tracked_face)
24         send_movement_to_ardunio(steps_x, speed_x, steps_y, speed_y)
25         image = print_boxes_into_image(current_image, real_faces,
       other_objects, tracked_face)
26         show_image(image)
```

```
27
28 # detect faces and eyes in image and return positions and size of them
29 def detect_and_return_rectangles(image):
30         haar_cascade_face = cv2.CascadeClassifier('
       haarcascade_frontalface_default.xml')
31         eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
32
33         #convert the test image to gray scale as opencv face detector
       expects gray images
34         gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
35
36         # Applying the haar classifier to detect faces
37         faces_rect = haar_cascade_face.detectMultiScale(gray_image,
       1.3,5,0)
38
39         eyes_rect = eye_cascade.detectMultiScale(gray_image)
40
41         real_faces = []
42         other_objects = []
43         for (x,y,w,h) in faces_rect:
44             eyes = []
45             for (ex,ey,ew,eh) in eyes_rect:
46                 if(ex > x and ex+ew < x+w and ey > y and ey+eh < y+h):
47                     eyes.append((ex,ey,ew,eh))
48             if(len(eyes) >= 2):
49                 real_faces.append((x,y,w,h,eyes))
50             else:
51                 other_objects.append((x,y,w,h))
52
53         biggest_face = [0,0]
54         index = 0
55         for (x,y,w,h,eyes) in real_faces:
56             if(w*h> biggest_face[1]):
57                 biggest_face[0] = index
58                 biggest_face[1] = w*h
59             index = index + 1
60
61         tracked_face = ()
62         if(len(real_faces)>0):
63             tracked_face = real_faces[biggest_face[0]]
64
65
66         return real_faces, other_objects, tracked_face
67
68 # process rectangle position into motor steps
69 def process_rectangle_to_steps(tracked_face):
70         if(len(tracked_face) == 0):
```

```
71              return 0,0,0,0
72
73          speed_x = 400
74          speed_y = 400
75          steps_x = 0
76          steps_y = 0
77
78          x = tracked_face[0]
79          y = tracked_face[1]
80          w = tracked_face[2]
81          h = tracked_face[3]
82
83          mx = x+(w/2)
84          my = y+(h/2)
85
86          if abs((image_size_x/2)-mx) < 90 and abs((image_size_y/2)-my) <
        50:
87              return 0,0,0,0
88
89          gear_x = 13
90          gear_y = 0.8
91
92          distance_right = image_size_x - (x+w)
93          distance_down = image_size_y - (y+h)
94
95          if(distance_right > x):
96              steps_x = int((distance_right/x)*gear_x)
97          else:
98              steps_x = int(-((x/distance_right)*gear_x))
99
100         if(distance_down > y):
101             steps_y = int((distance_down/y)*gear_y)
102         else:
103             steps_y = int(-((y/distance_down)*gear_y))
104
105
106         return steps_x, speed_x, steps_y, speed_y
107
108 # print boxes of eyes and faces into an output image
109 def print_boxes_into_image(image, real_faces, other_objects,
        tracked_face):
110     for (x,y,w,h, eyes) in real_faces:
111         cv2.rectangle(image, (x, y), (x+w, y+h), (0, 200, 0), 2)
112         for (ex,ey,ew,eh) in eyes:
113             cv2.rectangle(image, (ex, ey), (ex+ew, ey+eh), (0, 200, 40),
         2)
114
```

```
115
116     for (ex,ey,ew,eh) in other_objects:
117         cv2.rectangle(image,(ex,ey),(ex+ew,ey+eh),(0,100,0),2)
118
119
120     if(len(tracked_face)>0):
121         cv2.rectangle(image, (tracked_face[0], tracked_face[1]), (
        tracked_face[0]+tracked_face[2], tracked_face[1]+tracked_face[3]),
        (0, 255, 0), 2)
122         for (ex,ey,ew,eh) in tracked_face[4]:
123             cv2.rectangle(image, (ex, ey), (ex+ew, ey+eh), (255, 255, 0)
        , 2)
124
125     return image
126
127 # display an image
128 def show_image(image):
129     dim = (1280, 720)
130     re_image = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
131     cv2.imshow("Live-feed", re_image)
132     cv2.waitKey(10)
133
134 # send command to Arduino
135 def send_movement_to_ardunio(steps_x, speed_x, steps_y, speed_y):
136     print("------")
137     print(steps_x)
138     print(speed_x)
139     print(steps_y)
140     print(speed_y)
141     string_length = 5
142     steps_x = str(steps_x).zfill(string_length)
143     steps_y = str(steps_y).zfill(string_length)
144     speed_x = str(speed_x).zfill(string_length)
145     speed_y = str(steps_y).zfill(string_length)
146
147     command = steps_x  + steps_y + speed_x + speed_y + "E"
148     print(command)
149     arduino.write(bytes(command, 'utf-8'))
150     data = arduino.readline()
151     print(data)
152
153 tracking()
```

Listing D.3: Vollständiger Python-Programm zur Gesichtserkennung und Weiterverarbeitung.