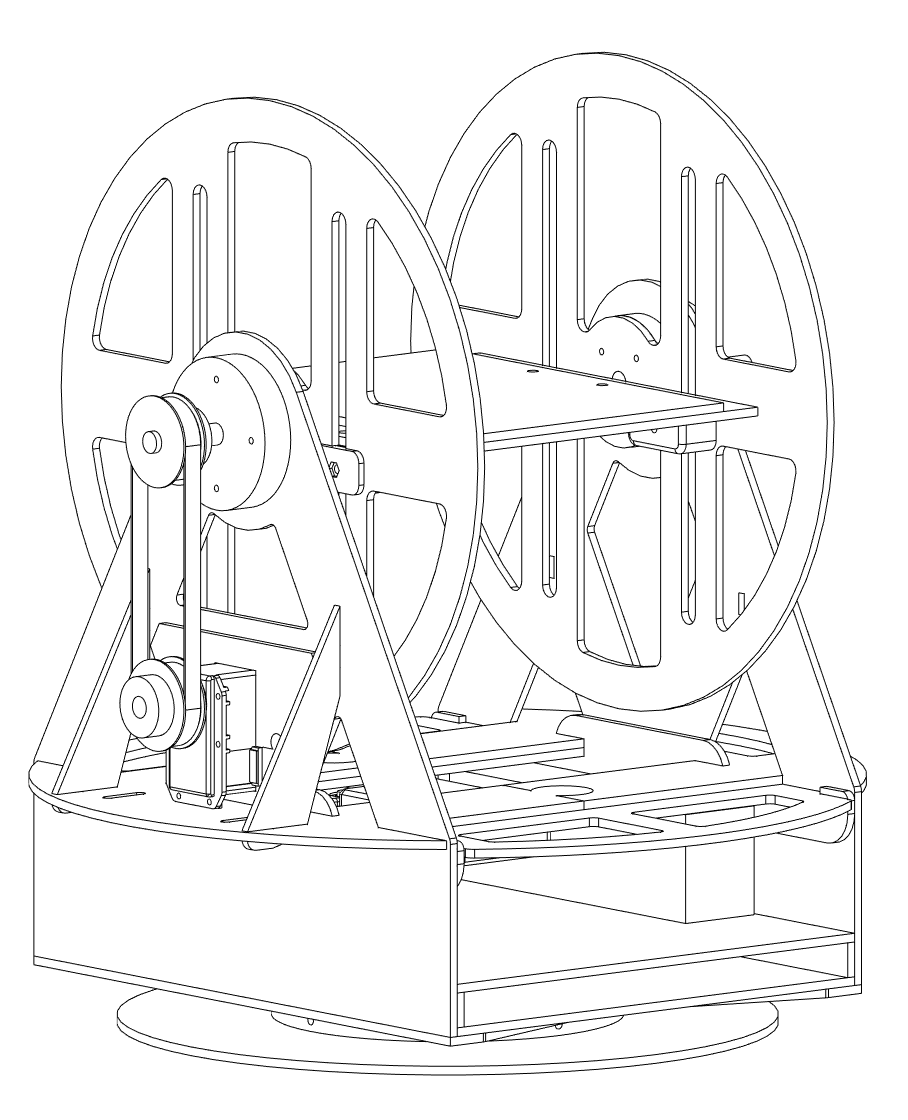
|  |
| --- |
| Eine Projektarbeit von: Luca Gloor G3A, Gianluca Imbiscuso G3A und Kay Obermeier G3A Unter Betreuung von Martin Stangl  22.1.2020 |

|  |
| --- |
| Dokumentation |
| Zum Bau eines autonomen, objektverfolgenden Bilderkennungs- und Kamerasystems mithilfe einer Künstlichen Intelligenz. |



Abstract

Diese Projektarbeit befasst sich mit der Entwicklung und der Konstruktion eines autonomen Kamerasystems. Dieses soll in der Lage sein, vom Benutzer ausgewählte Objekte automatisch zu erkennen und zu verfolgen. Das System soll sich in X-, und Z-Richtung drehen können. Gesteuert wird die Konstruktion hierbei mithilfe einer Künstlichen Intelligenz. Um den Aufwand zu verringern, wird die KI eines Open-Source-Projektes verwendet und für unsere Zwecke angepasst.

Die gesamte Arbeit haben wir in einen Konstruktions- und einen Programmier-Teil gegliedert. Das ermöglichte uns, gleichzeitig und unabhängig voneinander bereits an der Konstruktion sowie am Programm zu arbeiten.

Mit unseren Ergebnissen sind wir zufrieden. Natürlich gibt es noch hunderte Dinge, die man an diesem Projekt weiterführen, verfeinern oder neu hinzufügen könnte.

Vorwort

Wir sind an einer Flugshow auf diese Idee gestossen. Da es sehr anstrengend ist einen gesamten Nachmittag die Kamera zu halten, kamen wir auf die Idee ein Gerät zu bauen. Da zu diesem Zeitpunkt gerade die Projektarbeit bevorstand, entschieden wir uns dieses Projekt zu verwirklichen.

Wir wollen uns hiermit bei allen Personen bedanken, die uns bei der Arbeit unterstützt haben.

Besonderer Dank geht an Andreas Obermeier der für uns den Raspberry Pi, die Pi Cam, die Dynamixel Motoren, die Zahnriemen, Wellen- und Axiallager sowie das Holz zur Verfügung stellte. Auch für die stetige Hilfsbereitschaft für Tipps und Vorschläge sind wir sehr dankbar.

Ebenso danken wir auch unserer Betreuungsperson Martin Stangl für den reibungslosen Start in unser Projekt.

Inhaltsverzeichnis

[1. Einleitung 6](#_Toc30630348)

[1.1 Hauptziel 6](#_Toc30630349)

[1.2 Eigene Zusatzkriterien 6](#_Toc30630350)

[Universell brauchbare Plattform für den Gebrauch verschiedener Kameratypen 6](#_Toc30630351)

[Benutzerfreundliche Bedienungsoberfläche (GUI) 6](#_Toc30630352)

[Sauberer und übersichtlicher Code 7](#_Toc30630353)

[2. Theoretische Grundlagen 8](#_Toc30630354)

[2.2 Künstliche Intelligenz 8](#_Toc30630355)

[OpenCV 8](#_Toc30630356)

[Tracker Algorithmen 9](#_Toc30630357)

[2.3 Motoren 9](#_Toc30630358)

[Schrittmotoren 9](#_Toc30630359)

[Dynamixel Motoren 9](#_Toc30630360)

[2.4 Raspberry Pi 10](#_Toc30630361)

[2.5 FLTK Fast Light Toolkit 10](#_Toc30630362)

[2.6 Makefile 10](#_Toc30630363)

[2.7 TCP/IP 10](#_Toc30630364)

[2.8 mDNS 11](#_Toc30630365)

[3. Methode 12](#_Toc30630366)

[3.1 Bau des Drehturms 12](#_Toc30630367)

[Konstruktion des Grundgerüsts 12](#_Toc30630368)

[Mechanische Konstruktion 14](#_Toc30630369)

[Elektrische Strom- und Kabelverwaltung 14](#_Toc30630370)

[3.2 Applikationsentwicklung 15](#_Toc30630371)

[Anpassungen an der Künstlichen Intelligenz 15](#_Toc30630372)

[Ansteuerung der Motoren 15](#_Toc30630373)

[Entwerfung der GUI 16](#_Toc30630374)

[Verbindungsprobleme und ihre Lösungen 17](#_Toc30630375)

[4. Produkt und Applikation 18](#_Toc30630376)

[4.1 Drehturm 18](#_Toc30630377)

[4.2 Programm 18](#_Toc30630378)

[5 Ergebnisse und Schlussfolgerungen 19](#_Toc30630379)

[5.1 Konstruktion 19](#_Toc30630380)

[5.2 Software 19](#_Toc30630381)

[6. Zusammenfassung 20](#_Toc30630382)

[6.1 Selbstreflektion 20](#_Toc30630383)

[6.2 Gesamtprodukt 20](#_Toc30630384)

[7. Quellenverzeichnis 21](#_Toc30630385)

[8. Anhang 23](#_Toc30630386)

[8.1 Source Code 23](#_Toc30630387)

[C++ Source Code Server 23](#_Toc30630388)

[C++ Source Code Client 23](#_Toc30630389)

[Makefile Server 23](#_Toc30630390)

[Makefile Client 23](#_Toc30630391)

[8.2 Programm 23](#_Toc30630392)

# 1. Einleitung

Autonomes Objekttracking ist an sich keine neue Idee. Oftmals werden aber die zu trackenden Objekte speziell markiert, damit ein Gerät diese leicht erkennen kann. Bei unserer Arbeit haben wir eine Künstliche Intelligenz angewendet, die das Objekt anhand seiner Form erkennt und somit die Motoren ansteuern kann. Solche Systeme sind auch heutzutage bereits existent, nur findet man solche Systeme nur in der Rüstungsindustrie. Wir versuchen eine «Low-Budget» Variante in Marke Eigenbau nachzubauen.

## 1.1 Hauptziel

Das Hauptziel der Arbeit ist der Bau eines Drehturmes, welcher eine Kamera um die eigene X-Achse, wie auch die Z-Achse drehen kann. Da dieser autonom angesteuert werden soll, wurde die Verwirklichung des Projektes in 2 Teile aufgeteilt: Die Konstruktion des Drehturmes und die Einbettung der Künstlichen Intelligenz für die Ansteuerung der Motoren, die den Turm bewegen.

## 1.2 Eigene Zusatzkriterien

### Universell brauchbare Plattform für den Gebrauch verschiedener Kameratypen

Das Ziel dieses Kriteriums ist es, dass nicht nur wir, sondern auch andere Menschen mit anderen Kameras das Kamerasystem verwenden können. Auch mit unterschiedlich grossen Objektiven soll der Drehturm nicht aus dem Gleichgewicht gebracht werden.

### Benutzerfreundliche Bedienungsoberfläche (GUI)

Damit das Kamerasystem von anderen Menschen benutzt werden kann, ist wichtig, dass die Bedienung auf den ersten Blick klar erscheint. Der Benutzer soll das Produkt einfach und ohne zu raten bedienen können.

### Sauberer und übersichtlicher Code

Das Projekt soll nicht einfach funktionieren, sondern auch nachvollziehbar sein. Da alles durch ein Programm gesteuert wird, ist das Ziel dieses Kriteriums, dass dessen Code sauber geschrieben, kommentiert und übersichtlich gestaltet ist.

# 2. Theoretische Grundlagen

## 2.2 Künstliche Intelligenz

«Künstliche Intelligenz ist ein Teilgebiet der Informatik, welches sich mit der Automatisierung befasst. Der Begriff ist nicht eindeutig, weil eine genaue Definition fehlt.

Ist ein Computerprogramm intelligent, wenn es jedes Sudoku lösen, oder die besten menschlichen Spieler im Schach schlagen kann? Es stellt sich heraus, dass die AI-Komponente für viele komplizierte Probleme schlicht in besonders schlauen und effizienten algorithmischen Lösungen oder Abschätzungen besteht. Solch klassische Herangehensweisen werden oft mit der Abkürzung GOFAI (good oldfashioned AI) bezeichnet. Sie erfordern, dass der Programmierer, aufgrund seiner Intelligenz, einen Lösungsweg für das gegebenen Problem entwickelt und diesen in Programmcode umsetzt.» (Oinf.ch, Künstliche Intelligenz)

GOFAI ist ja schön und gut, doch wirklich intelligent ist das noch nicht. Immer wichtiger wird daher maschinelles Lernen, bei dem der Mensch keinen Lösungsalgorithmus vorgibt. Es basiert viel mehr auf dem Prinzip von Trial und Error. Vorgegeben werden der KI hierbei nur die Eingabe-Daten, ein neuronales Netz und ein Endergebnis. Das neuronale Netz besteht aus mehreren Schichten von Knoten, die miteinander verbunden sind. Je nach Eingabewert werden die Daten im neuronalen Netz verarbeitet und ausgegeben. Wenn das Endergebnis stimmt, werden, auf Basis dieser Generation, kleine Veränderungen in den Verbindungen gemacht, die den Prozess verfeinern. Ebenso wenn das Endergebnis falsch ist. Dann sind die Anpassungen dementsprechend höher und können in eine ganz andere Richtung gehen. So lernt die KI, wo und wie sie sich verbessern kann. Genauso wie wir immer besser werden, wenn wir etwas viel üben oder repetieren, verbessert sich auch die KI entsprechend der Repetitionen, die es macht. Je länger eine KI trainieren kann, desto besser wird sie.

### OpenCV

OpenCV (Open Computer Vision) ist die wohl bekannteste Open Source Bibliothek mit grösstenteils Algorithmen für Echtzeit-Bildverarbeitung. Sie kann auf allen herkömmlichen Betriebssystemen wie Linux, Windows oder Mac OSX kompiliert und verwendet werden. Geschrieben ist die Bibliothek in C/C++, doch sie unterstützt unter anderem auch Python und Java.

### Tracker Algorithmen

Das Ziel des «Object Trackings» ist es, Objekte über viele Bilder hinweg zu verfolgen. Der Tracker generiert ein Aussehens-Modell des ausgewählten Objekts. Hinzu wird ein Bewegungsmodell kreiert. Dieses misst die Geschwindigkeit und die Richtung über mehrere Bilder hinweg und kann so voraussehen, wo das Objekt sein wird. In diesem Bereich sucht der Tracker dann nach einem Objekt, das dem Aussehens-Modell gleicht. Es wird zwischen «Online» und «Offline» unterschieden. «Online» bedeutet, dass der Tracker ein Aussehens-Modell berechnet, während das Programm läuft. Es passt das Modell den Umständen an. «Offline» bedeutet, dass bevor das Programm gestartet wird, der Tracker schon weiss, wie das Aussehens-Modell aussieht. Solche Tracker werden zum Beispiel genutzt, um Gesichter zu erkennen. Diese Tracker sind viel schneller und in ihrer trainierten Disziplin sehr gut, doch können nur diese spezifischen Objekte verfolgen.

## 2.3 Motoren

### Schrittmotoren

Schrittmotoren sind in der Robotik immer anzutreffende Motoren. Durch ihre Einfachheit und hohen Präzision eignen Sie sich stark für solche Projekte. Auch sind sie sehr preisgünstig und Energieeffizienz. Jedoch haben Schrittmotoren wegen ihrer hohen Präzision im Vergleich zu anderen Motoren ein tiefes Drehmoment.

### Dynamixel Motoren

Dynamixel Motoren sind professionelle Motoren für die Robotik. Sie sind sozusagen die Königsklasse was Motoren betrifft. Durch ihre kompakte Grösse haben sie im Vergleich zu anderen Motoren ein enorm hohes Drehmoment. Ebenfalls lassen sich diese Motoren in Serie schalten, welches die Kabelführung bei Robotern extrem vereinfacht. Durch Ihre hohe Leistung, Qualität und Anwendungsmöglichkeiten fallen solche Motoren oft sehr teuer aus. Speziell bei diesen Motoren ist eine kompliziertere Ansteuerung. Um die Motoren über eine USB-Schnittstelle anzusteuern, braucht man einen Dongle. Für Dynamixel Motoren heisst dieser USB2Dynamixel.

## 2.4 Raspberry Pi

Ein Raspberry Pi ist ein kompakter Computer, nicht viel grösser als ein DIN A7 Papier. Er eignet sich wunderbar als Rechner für selbstgebaute Geräte. Durch seine Einfachheit und Modularität findet man einen Raspberry Pi in allen möglichen Schulprojekten. Ebenfalls ist dieser Computer sehr preisgünstig und vergleichsweise leistungsstark.

## 2.5 FLTK Fast Light Toolkit

FLTK ist eine auf C++ basierende GUI (General User Interface), die auf allen herkömmlichen Betreibsystemen wie Linux, Mac OSX und Windows funktioniert. Sie ist klein, schnell, portabel, benutzerfreundlich und braucht wenige Ressourcen. Das hauseigene Programm FLUID kann benutzt werden, um eine GUI aufzubauen.

## 2.6 Makefile

Ein wichtiger Teil des Kompilierens sind sogenannte Makefiles. Makefiles sind Linux-spezifische Dateien, in denen Parameter für den Kompilierer hineingeschrieben werden. Dies erleichtert das Kompilieren. Man schreibt einfach «*$make*» in den Terminal und das ganze Projekt wird kompiliert.

## 2.7 TCP/IP

TCP/IP ist eine Protokoll-Familie, welche genutzt wird, um mit anderen Computern über ein Netzwerk zu kommunizieren. Das TCP (Transmission Control Protocol) nimmt Daten von Anwendungen an und teilt sie in Pakete auf. Diese übergibt es dann dem IP (Internet Protocol). Dieses Protokoll sorgt dafür, dass die Pakete vom TCP adressiert und über das Netzwerk gesendet werden.

## 2.8 mDNS

Das Konzept des mDNS (Multicast DNS) löst Hostnamen zu IP Adressen auf und folgt dem herkömmlichen Prinzip des DNS. Dies hat jedoch einen Haken, denn es muss ein DNS Server im Netzwerk aufgesetzt sein. Dies ist nicht oft der Fall. Genau hier setzt mDNS an. Durch mDNS präsentiert sich der Computer im Netzwerk und sendet seine IP Adresse, wenn ein Computer seinen Namen «ruft».

# 3. Methode

## 3.1 Bau des Drehturms

### Konstruktion des Grundgerüsts

Für den Bau des Drehturms haben wir uns entschieden, das gesamte Produkt zuerst in einem CAD-Programm zu entwerfen und zu designen. Wir haben uns hierbei auf Fusion 365 geeinigt. Das CAD-Programm ist für Privatpersonen gratis zu gebrauchen. Da wir bereits in früheren, privaten Projekten dieses CAD benutzten, fiel es uns leicht mit diesem zu arbeiten.

Als Grundidee des Drehturms nahmen wir die Kameradrehtürme aus Drohnen zum Vorbild. Diese bestehen aus zwei motorisierten Achsen, eine für die Höhen- und die andere für die Seitenverstellung. Grob gesehen ist es ein Riesenrad auf einem Karussell.

Am Anfang begrenzten wir uns auf eine Grösse für die Kameras. Im Durchschnitt ist eine Fotokamera etwa 15cm breit. Damit wir genug Spielraum hatten, setzten wir unsere Breite für den Hauptlift auf 20cm. Von da aus konnten wir anfangen. Als Erstes entstand der Hauptlift. Dabei haben wir darauf geachtet, möglichst viele Verstellmöglichkeiten miteinzubauen, um nachher eine Kamera im Schwerpunkt ausbalancieren zu können. Man kann auf der Seite die Höhen-, und von oben die Tiefenlage der Kamera einstellen. Als Nächstes kamen die Stützen. Da die Stützen die grössten Bauteile sind, mussten wir auf unsere Möglichkeiten achten und die Grösse gemäss den möglichen Dimensionen unseres Lasercutters anpassen. Relativ schnell hatten wir dann bereits ein «Riesenrad».

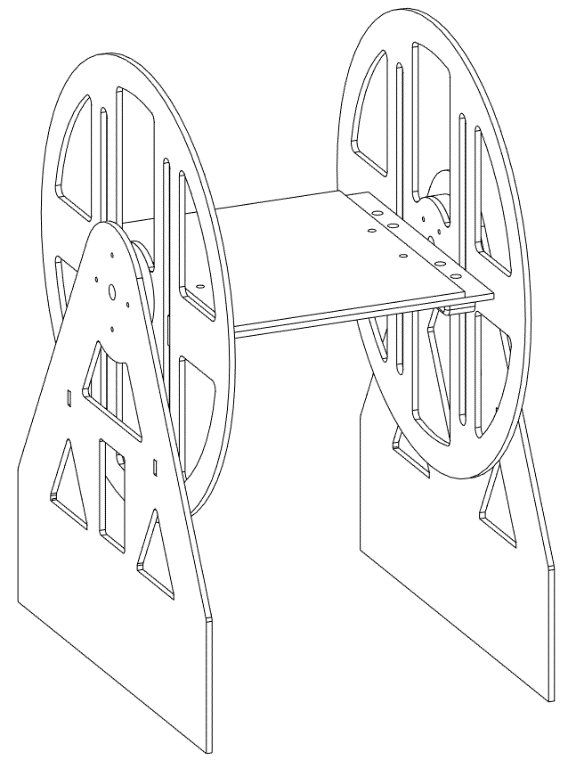
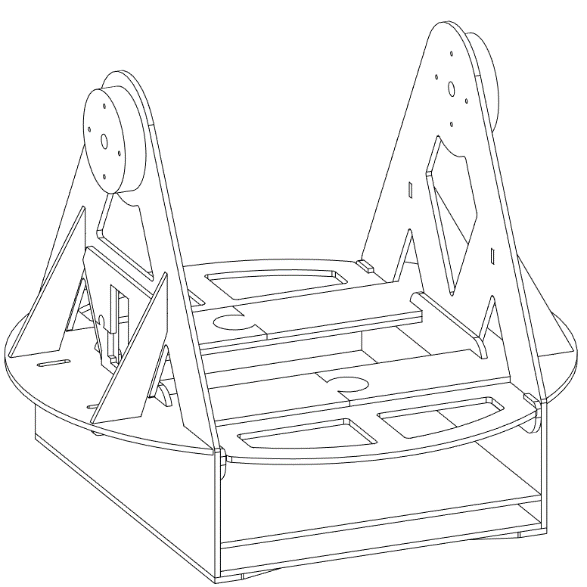
Um die Kamera auch auf die Seite schwenken zu können, brauchten wir das «Karussell». Dafür mussten wir zuerst eine grundlegende Entscheidung treffen. Ob wir die Elektronik mit dem «Riesenrad» mitdrehen, welches zu einem höheren Kraftaufwand führt, da mehr Masse bewegt wird, oder ob wir nur das «Riesenrad» drehen, ohne der Elektronik, welches zu möglichen Problemen mit verdrehten Kabeln führen könnte. Der Einfachheit wegen entschieden wir uns, die Elektronik mitsamt dem «Riesenrad» zu drehen. Hier fiel das Design recht einfach aus. Durch das Hinzufügen von mehreren Etagen, konnten wir eine praktische Plattform errichten und durch unseren Entscheid, alles mitzudrehen, muss nur ein Kabel der Kamera um eine Achse gedreht werden.

Abbildung 1: Die fertige Konstruktion des «Karussells»

Abbildung 2: Die fertige Konstruktion des «Riesenrads»

Als Nächstes mussten wir uns Gedanken darüber machen, wie wir die entsprechenden Achsen ansteuern und drehen können. Da gewöhnliche Zahnräder ziemlich viel Umkehrspiel haben, fiel die Entscheidung leicht, Zahnriemen zu verwenden. Der Grund: Zahnreimen sind relativ preisgünstig, präzise und erlauben es, Kräfte über weite Strecken mit wenig Platzaufwand zu übertragen. Ein Nachteil, den wir nicht in Betracht gezogen haben, ist, dass es nötig ist die Zahnriemen recht stark zu spannen.

Mit all diesen Voraussetzungen ging es nun darum, die Konstruktion auch umsetzbar zu machen. Wir entschieden uns für vier Millimeter dickes Sperrholz, welches wir mit dem Lasercutter bearbeiten konnten. Man könnte das Sperrholz in Zukunft auch mit stärkeren Aluminiumplatten ersetzen. Hierbei mussten wir uns achten, Verstärkungen dort einzusetzen, wo die Konstruktion zu brechen drohte. Gleichzeitig mussten wir auch schauen, dass wir an stabilen Orten, bei denen eher das Eigengewicht des Teiles zum Problem wurde, kleine Teile ausschneideten, sodass eine gute Balance zwischen Stabilität und Gewicht gefunden werden konnte. Verstärkt wurden vor allem die untere Hälfte, um die oberen Aufbauten halten zu können. Rausgeschnitten wurde vor allem am «Riesenrad», um die aufzuwendende Kraft gering zu halten. Das Ziel war, eine möglichst selbsttragende Konstruktion zu haben, um Klebstellen zu entlasten. Hier haben uns oft mit ineinander gehenden Zungen (Puzzleteilprinzip) geholfen. Mit dem fertigen Entwurf ging es dann an die Produktion.

Innerhalb von einer Woche wurden alle Teile mit dem Lasercutter geschnitten. PLA-Plastikteile, wie Klemmen und Lager-, sowie Motorhalterungen wurden mit dem 3D-Drucker gedruckt. Der Lasercutter schnitt eine Woche lang, durchgehend mit einer Geschwindigkeit von 120mm/min (mit je zwei Durchgängen pro Schnitt), alle Teile aus. Der 3D-Drucker benötigte etwas weniger als 50 Stunden, um alle Bauteile fertig zu stellen.

Damit das Sperrholz stabiler war, wurden alle Teile drei Mal mit Nitro-Hartgrund bestrichen. Durch das Bestreichen wurde das Holz trockener, härter und länger haltbar. Nach dem Auftragen wurden alle Teile nochmals geschliffen.

Für den Zusammenbau wurden Klebungen für Holz-zu-Holz-Verbindungen vorgesehen und für die Plastikteile hauptsächlich Schrauben der Grösse M3. Ausnahmen findet man bei den Motoren, da die standardmässig bereits M2,5 Bohrungen hatten.

### Mechanische Konstruktion

Beide Motoren werden von jeweils einem Motor angetrieben. Da der Motor nicht direkt auf der Achse liegt, mussten wir die Kraft umleiten. Dabei halfen uns die Zahnriemen. Für alle drehende Teile wurden nur, abgesehen von den Zahnriemen aus Glasfaser und Gummi, Aluminiumteile gebraucht. Dazu gehören:

2x 80mm Aluminiumachsen für das «Riesenrad»

4x 40mm Zahnriemenscheiben T5

2x passende Zahnriemen T5

2x Wellenkugellager 26x10x8mm

1x Axialkugellager 76x100x19mm

Natürlich passte nicht alles direkt zusammen. Beinahe alle Teile mussten überarbeitet werden. Die Aluminiumachsen wurden aus einem Stabrohling gedreht. Der Durchmesser musste von circa 10.03mm auf ein leichtes Untermass von 9.98mm gedreht werden, damit die Achse perfekt auf das aufgebohrte Loch der Zahnriemenscheibe und des Wellenkugellagers passte. Die Zahnriemenscheiben für die Motoren und für die Achse mussten auch angepasst werden. Dafür wurden in zwei Zahnriemenscheiben entsprechende Löcher für die Motorenaufnahme gebohrt, in die beiden anderen Zahnriemenscheiben die Löcher der Achsen aufgebohrt und seitlich in einem Loch ein Gewinde hinzugefügt. Für alle drei Kugellager wurde aus PLA-Plastik eine Halterung gedruckt. Alles zusammengebaut bietet uns einen leichten und sauberen Lauf der Achsen.

### Elektrische Strom- und Kabelverwaltung

Für die Motoren mussten noch Kabel gecrimpt werden. Dynamixel verwendet für ihre Motoren standardmässige Molex Stecker. Diese sind im Detailhandel einfach erhältlich. Wir mussten für das zusätzliche Kabel zwischen den beiden Motoren nur die vorgecrimpten Kabel mit den Buchsen zusammenstecken. Für den Anschluss am Akku mussten wir dann die beiden Data-Pins mit einem 120 Ohm Widerstand verbinden und die beiden Stromphasen an den Akku anstecken. Den Akku besassen wir bereit, da er für Flugmodelle gebraucht wurde. Für unser Projekt passt er ebenso gut. Es ist ein 4S 5000mAh Lipo Akku. Da er bereits die nötige Spannung von 14.8V bringt, können wir die Motoren ohne Bedenken direkt anschliessen.

Der Raspberry Pi benötigt auch Strom, aber nicht die direkten 14.8V, sondern 5V. Deshalb haben wir uns einen Spannungswandler gekauft, der konstante 5V mit 3A ausgibt. Diesen haben wir mit den Kabeln der Motorenspeisung zusammen in einen EC5 Stecker für an den Akku gelötet.

## 3.2 Applikationsentwicklung

### Anpassungen an der Künstlichen Intelligenz

Für die Applikationsentwicklung fingen wir zuerst mit der KI an. Wir entschieden uns, OpenCV zu benutzen. Da wir noch nie mit OpenCV arbeiteten, brauchten wir einen Anfang. OpenCV stellt einfache C++ Beispiele auf ihrer Webseite zur Verfügung. Als erster Schritt galt es, jede Variabel und jede Funktion zu verstehen. Dies war der wichtigste Schritt von allen, denn er legte die Basis, um die sehr abstrakte und komplexe OpenCV-Dokumentation verstehen zu können. Das Beispiel konnte lediglich ein Objekt verfolgen und ein Rechteck im Fenster zeichnen, welches den Tracker visualisierte. Beim Aufstarten wurde ein Standbild der Kamera gezeigt, in welchem man daraufhin ein Objekt zum Tracken auswählen konnte. Wir erweiterten das Programm, sodass es Daten in einem zweiten Fenster ausgeben konnte. Diese Daten waren zum Beispiel die Auflösung und das Zentrum des Rechtecks. Wir befassten uns genauer mit dem Tracker und schafften es, den Tracker zu glätten, sodass er nicht mehr herumspringt oder zittert. Diese Massnahme sollte später den Motoren helfen, gleichmässige und ruckelfreie Bewegungen auszuführen.

### Ansteuerung der Motoren

Der nächste grosse Abschnitt des Projekts war es, die Dynamixel-Motoren in unser Programm einzubinden. Wieder bei null anfangend, suchten wir nach Anleitungen und Bespielen. Auf der offiziellen GitHub Seite von Dynamixel wurden wir fündig und schafften es, die Bibliothek zu installieren. Mit dem Paket kamen simple C++ Beispiele. Durch die Dokumentation und mehrere Anleitungen fingen wir an, die Funktionsweise zu verstehen. Leider funktionierte nicht einmal das Beispiel. Wir versuchten, das ganze Beispiel bis auf das Nötigste zu kürzen, doch dies funktionierte ebenfalls nicht. Wir nahmen die Dynamixel-GUI in Windows zur Hilfe, um zu testen, ob die Motoren überhaupt betriebsbereit waren. Das waren sie und uns wurde klar, dass etwas an diesem Code falsch sein musste. Ausserdem wurde uns klar, dass der Motor eine anderen ID besass, als im C++ Beispiel geschrieben war. Doch dies half herzlich wenig. Beim Durchsuchen der Dokumentation wurden wir auf verschiedene Sendefunktionen aufmerksam. Zu den Motoren wird ein Array aus Bytes über den USB2DYNAMIXEL-Adpater gesendet. Beim Ansteuern definiert man, welcher Teil vom Array angesprochen wird. Dieser spezifische Teil wird dann geändert. Es gibt sowohl R (ReadOnly) als auch RW (Read&Write) Werte. Die «ReadOnly» Werte können nur gelesen werden, während die «Read&Write» Werte auch verändert werden können.

Zu jedem Motor muss eine ID definiert werden, um die Motoren untereinander unterscheiden zu können. Die Motoren sind in Serie nacheinander gekoppelt. Dies bedeutet, dass die Signale durch alle Motoren verlaufen, bis sie ihren Zielmotor erreichen.

Eine der Funktionen funktionierte und der Motor drehte sich das erste Mal. Wieso der Motor mit den anderen Sendefunktionen nicht funktionierte, ist unklar. Wahrscheinlich ist der Motor schon zu alt und die Beispiele sind für die neueren Motoren ausgelegt.

Dynamixel Motoren unterstützen generell zwei verschiedene Modi; den Wheel-Mode und den Joint-Mode. Der Joint-Mode ist dabei vorinstalliert. Dieser Modus funktioniert mit Grad. Die Motoren können jedoch nur circa 270° drehen. Eine volle Umdrehung ist also nicht möglich. Der Vorteil ist, dass die Position (Winkel in °) jederzeit abgefragt werden kann. Der Wheel-Mode unterscheidet sich darin, dass er nicht mit einer Position funktioniert, sondern mit einer Geschwindigkeit. Die Motoren können unendlich drehen, doch haben keine Angaben an welchen Positionen sie sich gerade befinden. Gewählt wurde für dieses Projekt der Wheel-Mode, denn der Turm soll sich unendlich drehen können. Hinzu kommt, dass der Tracker von OpenCV die Positionsabfrage übernimmt. Er instruiert die Motoren, sich so lange zu bewegen, bis sich das verfolgte Objekt in der Mitte befindet. Problem hierbei ist, dass wenn ein Motor instruiert wird, sich mit voller Leistung zu drehen, dann tut er dies, bis er einen anderen Wert für die Geschwindigkeit zugesendet bekommt. Bei einem Absturz des Programms dreht er also unendlich, bis der Strom gekappt wird. Auf der X-Achse ist dies kein grosses Problem, doch bei der Z-Achse würde es die RPi IR-CUT Camera mitreissen. Als Sicherheitsmassnahme dafür wurde eingerichtet, dass, falls die Verbindung abgebrochen würde, der Raspberry Pi neu startet und das Programm erneut von selbst ausführt.

Wir veränderten anfänglich nur den Beispiel-Code, um den Dreh rauszukriegen. Das Kompilieren des Codes und das Verändern des Makefiles war und unklar. Es galt also, dies zuerst zu verstehen, bevor wir in der Lage waren, das Beispiel auf unser Programm anzuwenden. Durch Ausprobieren und Recherchieren war dies jedoch schnell gelernt. Wir schafften es, die beiden Projekte zu kombinieren.

Wir programmierten den Code, um die vom Tracker herausgegebenen Werte in Motoreninstruktionen umzuwandeln. Nun war das Programm in der Lage, die Motoren selbst zu steuern.

### Entwerfung der GUI

Ein weiterer Teil der Arbeit enthielt das Kreieren einer GUI (Benutzeroberfläche). FLTK stellte eine geeignete Oberfläche zur Verfügung, welche einfach anzupassen war. So wurde das OpenCV Fenster durch FLTK ersetzt.

Wir starteten auf GitHub ein neues Projekt, um das Transferieren der Files zu erleichtern. Diese sollen für alle zugänglich und reproduzierbar sein.

### Verbindungsprobleme und ihre Lösungen

Da der Raspberry Pi eine andere Architektur besitzt, als der Computer, auf dem alles entwickelt wurde, funktionierte das Programm auf dem Raspberry Pi zuerst nicht. Das Makefile wurde angepasst und nicht vorhandene Bibliotheken wurden installiert. Auf GitHub schrieben wir Anleitungen, wie alles installiert und kompiliert werden muss. Jedoch war es nicht, wie geplant, möglich, alles nur auf dem Raspberry Pi laufen zu lassen, denn nach dem portieren des Programms auf den Raspberry Pi lief der Verfolgungsalgorithmus nur mit zwei Bildern pro Sekunde. So konnte keine Echtzeit-Objekt-Verfolgung erreicht werden. Durch Datenübertragung über ein Netzwerkverbindung wird das Berechnen einem anderen Computer mit mehr Rechenleistung überlassen. So erhöhte sich die Zahl auf elf Bilder pro Sekunde.

Der Raspberry Pi ist mit dieser Lösung für das Steuern der Motoren und für die Übertragung des Bildes verantwortlich. Ein beliebiger anderer Computer empfängt das Bild und berechnet, durch den Verfolgungs-Algorithmus, Werte, die als Bewegungsinstruktionen für die Motoren zurückgeschickt werden.

Da wir in den meisten Netzwerken nicht über den Zugriff auf den Router oder den Switch verfügen, können wir den Computern keine statische IP zuweisen. Da der Raspberry Pi Raspberry Pi an keinem Bildschirm angeschlossen ist, kreiert dies die Herausforderung, die IP des Raspberry Pis herauszufinden. Gelöst wurde dies mit mDNS. So verbindet sich der Computer automatisch mit dem Raspberry Pi.

Unser Programm basiert auf einem «Call-Response» Schema zwischen dem Raspberry Pi und dem Computer. Das heisst, dass der eine Partner Daten schickt, während der andere wartet, bis alles angekommen ist. Danach werden die Positionen gewechselt. Der vorherigen Sender ist nun der Empfänger. Diese Abfolge wiederholt sich, bis einer der Partner die Verbindung abbricht.

Auf GitHub fanden wir ein Projekt, welches in der Lage war, Bilder über das Netzwerk an OpenCV zu senden. So kann der Raspberry Pi Bilder an den Computer senden, welcher die Bilder in Instruktionen für die Motoren umwandelt und diese zurücksendet. Der Raspberry Pi bewegt mit diesen Instruktionen die Motoren.

Somit funktioniert alles. Der Raspberry Pi kann vom Computer aus ausgeschalten oder neu gestartet werden und braucht somit selbst keine Peripheriegeräte.

# 4. Produkt und Applikation

## 4.1 Drehturm

Der Drehturm kommt in zwei Teilen, der Grundplatte und dem Turm. Beide Teile können auseinandergenommen und wieder verschraubt werden. Für den Zusammenbau braucht man einen Schlitz-Schraubenzieher. Die Schrauben zum Anziehen befinden sich im Tiefbau. Um eine Kamera zu montieren, braucht man einen Torx-Schraubenzieher. Die Höhe kann, mithilfe von vier Schrauben, unter dem Hauptlift eingestellt werden. Die Tiefe kann, von oben, ebenfalls mit vier Schrauben verstellt werden. Nach längerem Gebrauch sollte das Axiallager wieder mit normalem Schmieröl geölt werden.

## 4.2 Programm

Die Ansteuerung ist ebenfalls zweiteilig aufgebaut. Zum einen muss der Raspberry Pi und zum anderen ein externer Rechner gestartet werden. Hierbei muss darauf geachtet werden, dass der Raspberry Pi zuerst läuft. Das Programm wird auf dem externen Rechner gestartet und verbindet sich, per Ethernet, automatisch mit dem Raspberry Pi. Der Nutzer wird daraufhin aufgefordert, einen beliebigen Tracking Algorithmus zu wählen. Der letzte Schritt beinhaltet das Auswählen des Objekts mit der Maus. Daraufhin bewegt sich der Turm und verfolgt das zuvor gewählte Objekt. Dies ist so lange der Fall, bis der Nutzer das Programm schliesst, oder den Tracker stoppt und ein neues Objekt auswählt.

# 5 Ergebnisse und Schlussfolgerungen

## 5.1 Konstruktion

Der Drehturm ist insgesamt sehr robust herausgekommen. Die Schnittqualität der einzelnen Teile hat uns sehr überrascht. Im Grossen und Ganzen funktionierte der Zusammenbau einwandfrei. Die Teile passten ohne jegliche Nachbearbeitung. Auch die Bewegung verlief während dem Testen sehr flüssig. Während dem letzten Test verklemmte sich die Konstruktion und wir konnten sie nicht mehr drehen. Der Grund dazu ist uns unbekannt. Weitere Mängel mussten wir an der Z-Achse feststellen. Bei dieser fiel durch das leichte Gesamtgewicht der Turm oft aus dem Axiallager. Ebenso haben wir zu kleine Toleranzen für das Spannen der Zahnriemen mit einberechnet. Hier hätten ein paar Millimeter mehr geholfen. Insgesamt waren wir durch die hohe Stabilität des Sperrholzes sehr beeindruckt.

## 5.2 Software

Der Code an sich ist für uns überraschend kurz ausgefallen. Ebenso hat uns die Schnelligkeit und Präzision des Trackers beeindruckt. Was uns noch nicht gelungen ist, ist während des Betriebes auf verschiedene Tracker-Algorithmen umzusteigen, sprich, zwischen den Menüs zu wechseln. Ebenso hängt sich der Raspberry Pi in vereinzelten Fällen auf und die Motoren drehen unkontrolliert weiter. Dafür haben wir bis dato keine Lösung gefunden. Hingegen konnten wir bei Kommunikationsproblemen auf der Seite des externen Rechners ein unkontrolliertes Durchdrehen der Motoren verhindern, indem der Raspberry Pi von selbst neustartet.

# 6. Zusammenfassung

## 6.1 Selbstreflektion

Wir haben dank diesem Projekt unseren Horizont sehr erweitern können. Uns war am Anfang nicht bewusst, was auf uns zukommt und wie wir unsere Vision in Wirklichkeit umsetzten können. Die Konstruktion konnten wir selbständig konzipieren und ausarbeiten, da wir bereits Vorkenntnisse im Gebrauch von CAD-Programmen hatten. Dank unseren Vorkenntnissen im Programmieren konnten wir OpenCV mithilfe von kleinen Beispielen erlernen. Stein um Stein fingen wir an, das Programm zu erweitern. Der nächste grosse Schritt war, die Dynamixel Motoren einzubinden. Dies gelang uns auch, obwohl wir anfängliche Schwierigkeiten hatten. Viel Ausprobieren, Hinfallen und wieder Aufstehen war gefragt. Uns wurde bewusst, wie schwer es sein kann, ein Programm mit Technik zu verbinden. Wir sind sehr stolz auf das, was wir erreicht haben, doch wir sehen auch ein, dass noch viel Potential vorhanden ist. Dass es am Ende doch nicht ganz funktioniert hat, ist natürlich enttäuschend.

## 6.2 Gesamtprodukt

In Theorie hätte alles natürlich funktionieren können. Das Programm kann auch weiter genutzt werden. Dass am Ende nur wegen zu hoher Reibung die Z-Achse nicht funktionstüchtig war, ist sehr schade.

# 7. Quellenverzeichnis

Dokumentation Open CV: <https://docs.opencv.org/4.2.0/>

Abgefragt am 14.9.2019 21:37

Tutorial zum gebrauch von Verschiedenen Trackern:<https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/>

Abgefragt am 15.9.2019 17:04

FLTK Programm: <https://www.fltk.org/index.php>

Abgefragt am 8.11.2019 20:16

FLTK Dokumentation: <https://www.fltk.org/documentation.php>

Abgefragt am 8.11.2019 20:53

E-Manual für Dynamixel Motoren: <http://emanual.robotis.com/docs/en/dxl/rx/rx-64/>

Abgefragt am 10.10.2019 19:35

Jeremy Morgan - Provided code for reading output of a console command (getStdoutFromCommandFunc function in client.cpp): <https://www.jeremymorgan.com/tutorials/c-programming/how-to-capture-the-output-of-a-linux-command-in-c/>

Abgefragt am 21.11.2019 20:32

Grundlage der Makefiles: <https://github.com/ROBOTIS-GIT/DynamixelSDK/tree/master/c%2B%2B/example/protocol1.0/read_write/linux64>

Abgefragt am 13.12.2019 19:23

Steve Tuenkam - Provided idea behind sending video over TCP/IP using OpenCV: <https://gist.github.com/Tryptich/2a15909e384b582c51b5>

Abgefragt am 11.1.2020 18:14

Recherche für den Theorieteil über Künstliche Intelligenz: <https://oinf.ch/kurs/informationsgesellschaft/kuenstliche-intelligenz/>

Abgefragt am 22.1.2020 18:21

# 8. Anhang

## 8.1 Source Code

### C++ Source Code Server

### C++ Source Code Client

### Makefile Server

### Makefile Client

## 8.2 Programm

Das Programm ist aufrufbar unter: https://github.com/BE3dARt/CamTrackAI?files=1