

Diplomarbeit 2023/2024

**Intelligenter Zeichenroboter
DrAI**

ausgeführt an der

Höheren Technischen Bundeslehranstalt für Maschinenbau,
Ausbildungsschwerpunkt Automatisierungstechnik
in Neufelden

von

Rene Schwarz (KN.:20191028)
Samuel Nösslböck (KN.: 20191019)

Betreuer
Dipl. -Ing. Peter Rachinger

Vorwort

Danksagung

Wir möchten all jenen danken, die uns bei der Abwicklung der Diplomarbeit tatkräftig unterstützt haben und uns bei jeglichen technischen sowie persönlichen Fragen weitergeholfen haben. Besonderer Dank gilt unserem Projektbetreuer Dipl.-Ing. Peter Rachinger, für seine Unterstützung und sein Vertrauen in unsere Fähigkeiten, dieses Projekt umzusetzen.

Ebenso möchten wir uns bei den Ars Electronica Mitarbeitern, Ali Nikrang und Gregor Woschitz bedanken, die uns zu Beginn des Projektes mit ihrem Einfallsreichtum und Wissen im Bereich Künstlicher Intelligenz, auf die finale Umsetzung gebracht haben. Auch gilt unser Dank den Mitarbeitern Manuela Hillmann und Gerold Hofstadler, die sich die Zeit genommen haben, unser Projekt in das Ars Electronica Center einzubinden.

Herzlichen Dank auch an alle Lehrkräfte der HTL-Neufelden, die ebenso immer für Fragen bereitstanden und deren praktisches wie auch theoretisches Wissen in unser Projekt eingeflossen ist.

Zu guter Letzt möchten wir uns bei der Firma Igus bedanken, die uns einen Teil unserer Komponenten finanziert hat.

Gender Erklärung

Um eine bessere Lesbarkeit der Diplomarbeit zu garantieren, wird im Laufe des Dokuments die Sprachform des generischen Maskulinums angewendet.

An dieser Stelle möchten wir unbedingt darauf hinweisen, dass diese gesonderte Verwendung der männlichen Form als geschlechtsunabhängig verstanden werden soll.

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht haben.

Neufelden, am

Samuel Nösslböck

Rene Schwarz

Zusammenfassung

Ziel dieses Projekts ist es, die Fähigkeiten von KI bei der Erledigung kreativer Aufgaben in Zusammenarbeit mit Menschen zu demonstrieren und menschliche Kreativität nachzuahmen.

Der Roboter ist außerdem als Ausstellungsstück für das Machine Learning Studio des Ars Electronica Center Museums geplant.

Das Endergebnis wird ein interaktiver Roboter sein, der mit einer KI ausgestattet ist, die darauf trainiert ist, vorgezeichnete Skizzen des Benutzers zu interpretieren und sie mit ihrer eigenen "Kreativität" zu vervollständigen. Sobald sich das Blatt im Roboter befindet, ist es sein Ziel, sein Potenzial zu entfalten, indem er zusätzliche Linien auf das Papier des Benutzers zeichnet. Die Möglichkeiten sind hier grenzenlos, die KI kann z. B. einen Wald neben einem Häuschen erschaffen, oder einen Strand neben einem Meer und so weiter.

Da der Roboter in einem Museum ausgestellt wird, sollte das Design möglichst attraktiv wirken und die Verletzungsgefahr minimiert werden. Auch sollte der Zeichenvorgang nicht zu viel Zeit beanspruchen, um die Aufmerksamkeit des Besuchers nicht zu verlieren. Deshalb wird die Steuerung auf das maximal mögliche ausgelegt. Jedoch sorgen hohe Geschwindigkeiten auch für eine hohe Verletzungsgefahr, weshalb der ganze Vorgang abgeschirmt hinter Glas stattfindet. Die gesamte Konstruktion ist kompakt gebaut und fertigt die Zeichnungen auf einem A5 Papier an.

Abstrakt

The aim of this project is to demonstrate the capabilities of AI in performing creative tasks in collaboration with humans and to imitate human creativity. The robot is also planned as an exhibit for the Ars Electronica Center Museum's Machine Learning Studio.

The end result will be an interactive robot equipped with an AI that is trained to interpret the user's pre-drawn sketches and complete them with its own "creativity". Once the sheet is in the robot, its goal is to unfold its potential by drawing additional lines on the user's paper. The possibilities here are limitless, the AI can, for example, create a forest next to a cottage, or a beach next to a sea and so on.

As the robot will be exhibited in a museum, the design should look as attractive as possible and the risk of injury should be minimized. The drawing process should also not take too much time so as not to lose the visitor's attention. This is why the control system is designed for the maximum possible speed. However, high speeds also pose a high risk of injury, which is why the entire process takes place behind glass.

1 Kontaktdaten

1.1 Diplomanden

Samuel Nösslböck

Schulstraße 11
4142 Hofkirchen
20191019@atn.ac.at
Elektronik, Steuerungssoftware, Konstruktion

Rene Schwarz

Hauptstraße 11
4151 Oepping
20191028@atn.ac.at
KI, Konstruktion

1.2 Projektbetreuer

Dipl. -Ing. Peter Rachinger
pe.rachinger@htl-neufelden.at



1.3 Schule

HTL-Neufelden
Höferweg 47
A-4120 Neufelden
07282-5955
info@htl-neufelden.at



Höhere Technische
Bundeslehranstalt Neufelden

1.4 Auftraggeber

Die Ars Electronica Linz GmbH & Co KG analysiert und kommentiert seit 1979 die Digitale Revolution. Gemeinsam mit Künstler*innen, Wissenschaftler*innen, Technolog*innen, Designer*innen, Entwickler*innen, Entrepreneuers und Aktivist*innen aus aller Welt befasst er sich mit den zentralen Fragen unserer Zukunft. Hierbei stehen neue Technologien und wie sie unser Zusammenleben / Zusammenarbeit ändern, im Mittelpunkt.

Ars Electronica Linz GmbH & Co KG

Ars-Electronica Straße 1

4040 Linz

+43 732 7272 0

center@ars.electronica.art

1.5 Ansprechpartner der Firma



Ali Nikrang

Gregor Woschitz

Gerold Hofstadler

Manuela Hillmann

E-Mail Adressen der Ansprechpartner:

- Ali.Nikrang@ars.electronica.art
- Gregor.Woschitz@ars.electronica.art
- Gerold.Hofstadler@ars.electronica.art
- Manuela.Hillmann@ars.electronica.art

Inhaltsverzeichnis

Vorwort.....	2
Danksagung.....	2
Gender Erklärung.....	2
Eidesstattliche Erklärung.....	3
Zusammenfassung.....	4
Abstrakt.....	5
1 Kontaktdaten.....	6
1.1 Diplomanden.....	6
1.2 Projektbetreuer.....	6
1.3 Schule.....	6
1.4 Auftraggeber.....	7
1.5 Ansprechpartner der Firma.....	7
Inhaltsverzeichnis.....	8
2 Einleitung.....	12
2.1 Produktentstehung.....	12
2.2 Finales Konzept und Aufgabenstellung.....	13
2.3 Projektplanung und Meilensteine.....	13
2.4 Kosten.....	15
2.5 Projektkoordination.....	16
2.5.1 Kompetenzen des Teams.....	16
2.5.2 Kooperation und Betreuung.....	17
2.6 Reifung und Lizenzierung.....	18
2.7 Recherche und Innovation.....	19
2.8 Wirtschaftlichkeit.....	20
3 Steuerung.....	21
3.1 R-D-S System.....	21
3.2 Robot.....	21
3.2.1 Schrittmotoren-Steuerung.....	21
3.2.2 Tools-Datenbank und Höhenkalibrierung.....	22
3.3 Descriptor.....	22
3.4 Station.....	22
3.5 Die Programmiersprache Rust.....	22
4 Konstruktion.....	24
4.1 Rahmen.....	24
4.1.1 Grundrahmen.....	24
4.1.2 Befestigung für X-Achse.....	24
4.1.3 Achteck.....	25
4.1.4 Gesamtrahmen.....	25
4.2 X,Y-Achse.....	26
4.3 Z-Achse.....	27

4.4 Befestigungstisch.....	28
4.5 Stifthalter.....	28
4.6 Sicherheitstechnik.....	29
4.7 Gesamtkonstruktion.....	29
4.8 Kamera / Papiererkennung.....	29
5 KI.....	30
5.1 Generelles.....	30
5.1.2 Ziel.....	30
5.1.3 Einzigartigkeit.....	30
5.1.4 Kontrolle.....	30
5.1.5 Ergebnisse.....	30
5.2. Entstehung / Experimente.....	32
5.2.1. Eigenes Neuronales Netzwerk.....	32
5.2.2. StablePipe.....	34
5.2.2.1. Stable Diffusion.....	34
5.2.2.1. StablePipe Pipeline.....	34
5.2.3. InterStablePipe.....	35
5.2.3.1. Clip-Interrogator.....	35
5.2.3.2. Line Extraction Software (LES).....	35
5.2.3.3 InterStablePipe Pipeline.....	36
5.2.4 InterStableCloud.....	38
5.2.4.1 Wordcloud.....	38
5.2.4.2 InterStableCloud Pipeline.....	39
5.2.5 InterStableLLM Pipeline.....	42
5.2.5.1 Motivation.....	42
5.2.5.2 Large Language Model.....	42
5.2.5.3 InterStableLLM Pipeline.....	43
5.2.6 Finale Umsetzung, InterStableLLMRLLine.....	45
5.2.6.1 R-Line.....	45
5.2.6.2 InterStableLLMRLLine Pipeline.....	45
5.4.3 Time / Memory Management.....	46
5.3 pdars - Paper detection and reconversion Software.....	48
5.3.1 Ablauf.....	48
5.3.1.1 Marker Detection.....	49
5.3.1.2 Entzerrung.....	50
5.3.1.3 Papier Eckenerkennung.....	51
5.3.1.4 Stifterkennung.....	51
5.3.2 MarkerWebserver.....	52
5.3.2.1 Starten des MarkerWebservers.....	52
5.3.2.2 Unabsichtliches Speichern von Werten.....	52
5.3.2.3 Software Aufbau und funktionsweise.....	52
6 Versionierung und Aufbau der Software.....	56
6.1 Versionierung der AI.....	56

6.2 Aufbau der KI-Software.....	57
6.2.1 KI-Implementierung.....	59
6.2.2 Layers.....	61
6.2.2.1 Interrogator.....	61
6.2.2.2 RandomLineDrawer.....	62
6.2.2.3 SimpleString.....	63
6.2.2.4 Merger.....	63
6.2.2.5 LLM.....	64
6.2.2.6 StableDiffusion.....	65
6.2.2.6.1 Stable Diffusion Config Datei.....	65
6.2.2.7 Overlapper.....	66
6.2.2.8 Model.....	66
6.2.3 Netzwerk Kombination.....	67
6.2.4 Auto-Dokumentation.....	69
6.3 Aufbau der Steuerungssoftware.....	71
6.3.1 Aufbau eines Rust-Projekts.....	71
6.3.2 sylo-Library.....	71
6.3.3 syact-Library.....	72
6.3.4 sybot-Library.....	72
7 Kommunikation der Schnittstellen.....	73
7.1 Das IOT des Projekts.....	73
7.2 Kommunikationskette.....	74
8 Betriebsanleitung.....	75
8.1 Installation der Software.....	75
8.2 Kalibrierung und Tests.....	75
8.3 Konfiguration.....	76
8.3.1 Allgemeine Konfiguration.....	76
8.4 Stiftwechsel.....	77
8.5 Aufsetzen der Elektronik.....	78
9 Elektronik.....	79
9.1 Spannungsversorgung.....	79
9.2 Achsen.....	80
9.3 Zeichentisch.....	81
10 Allgemeines.....	82
10.1 Häufige Fehler.....	82
10.1.1 Kommunikation mit der Kamera schlägt fehl.....	82
10.1.2 Die Serververbindung fehlt.....	82
10.1.3 Fehler bei der Bildaufnahme.....	82
10.1.4 Servos nicht erkannt.....	82
10.1.5 Error mit der pdards Software.....	82
10.1.6 Error in der AI-Pipeline.....	83
10.1.7 Marker-Webserver funktioniert nicht.....	83
10.1.8 Weights von Neuronalen Netzwerken fehlen.....	83

10.1.9 NSFW-Content erkannt.....	83
10.1.10 Prozess startet nicht.....	83
Abbildungsverzeichnis.....	84

2 Einleitung

2.1 Produktentstehung

Von klein auf waren wir das, was umgangssprachlich "Bastler" genannt wird, in den letzten Jahren vor allem in den Bereichen Robotik und KI. Da wir uns schon seit der Unterstufe kennen und uns gegenseitig immer wieder Fotos und Fortschritte unserer Basteleien zeigten, war die Findung eines Projektteams keine Aufgabe von großer Schwierigkeit.

Das Ziel war es, ein Projekt umzusetzen, das wirklich an die Grenzen geht, deshalb wurde sich für eine Zusammenarbeit mit einem Museum entschieden. Durch die guten Kontakte des Beratungslehrers, Herrn Dipl.-Ing. Peter Rachinger, konnte ein Meeting mit dem FutureLab des *Ars Electronica Centers (AEC)* organisiert werden. Die Aufgabe seitens des *AEC* nach der Auflistung persönlicher Interessen lautete:

Es sollte ein Projekt erstellt werden, welches den Besuchern Künstliche Intelligenz näherbringt.

In Absprache mit dem Projektpartner und des Projektbetreuers wurde über zahlreiche Konzepte diskutiert. Da die Möglichkeiten äußerst vielfältig sind, standen Systeme zur Debatte, die selbst Musik komponieren und Marionetten, die Gesten imitieren, bis sich schlussendlich auf die Idee geeinigt wurde, analoge Zeichnungen mit neuen KI-Technologien zu erstellen. Nach einigen weiteren Überlegungen entstand die finale Idee: ein von KI unterstützter, zeichnender Roboter.

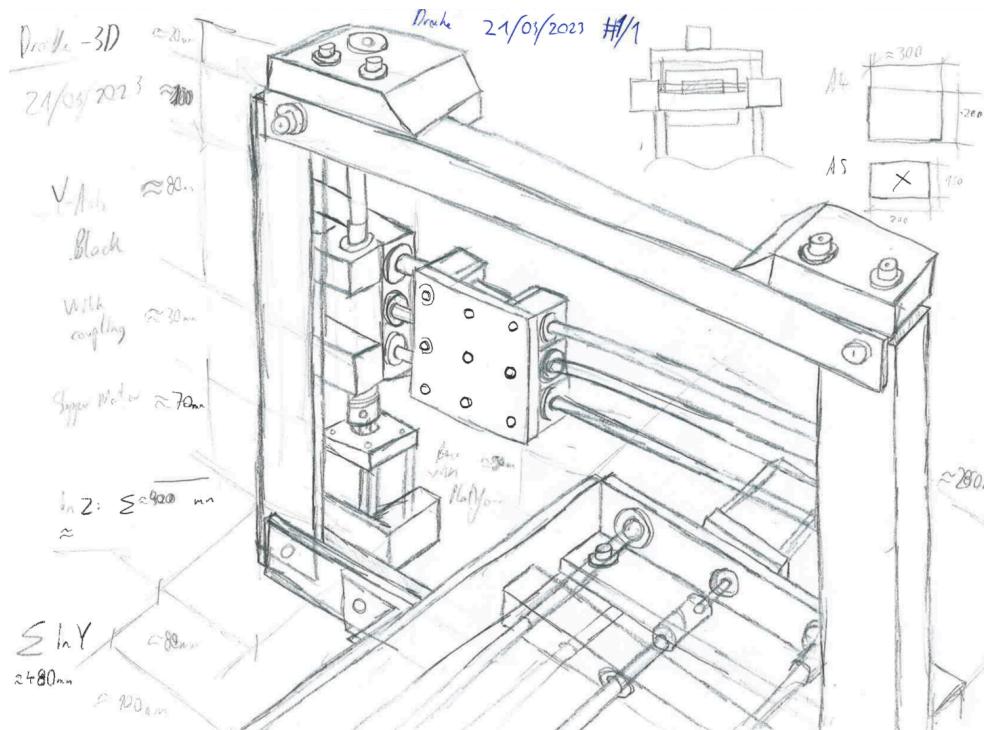


Abbildung 2.1.1 Erste Konzeptskizze

2.2 Finales Konzept und Aufgabenstellung

Die Aufgabe des Projektes ist es, den Besucher*Innen im *Ars Electronica Center* Künstliche Intelligenz näherzubringen. Hierzu wurde final folgender Ablauf definiert:

1. Der User zeichnet eine Zeichnung mit einem Stift auf ein Blatt Papier
2. Der Roboter erstellt mithilfe einer Kamera eine digitale Version davon
3. Die Künstliche Intelligenz erkennt, was gezeichnet wurde, und fügt kreativ Dinge hinzu
4. Das Generierte wird zu einer Liste an Linien konvertiert und diese werden vom Roboter gezeichnet
5. Dieser Prozess kann nach Belieben wiederholt werden, beide Parteien können immer weiter Dinge hinzufügen

Dies führt dazu, dass der*die Besucher*In in den Prozess integriert und mit der Künstlichen Intelligenz gemeinsam eine Zeichnung entwickelt. Die Kreativität beider Parteien soll in den Vordergrund gerückt werden.

Unser Projekt wurde in zwei Teilbereiche unterteilt: Roboter (Hardware) und Künstliche Intelligenz (Software). Hierbei ist die Künstliche Intelligenz die wesentliche Innovation des Projektes, weshalb besonderer Fokus auf diesen Teil gelegt wurde.

2.3 Projektplanung und Meilensteine

In eigenem Interesse und zugunsten des *AEC* wird das Projekt öffentlich auf [Github](#) in folgendem [Repository](#) (QR-Code auf der nächsten Seite) geführt.

The screenshot shows the GitHub repository for 'DrAI' (Public). The repository was created by SamuelNoesslboeck on 12eb779 · 3 hours ago, with 96 commits. The master branch has 5 branches and 1 tag. The commit history lists various updates across different folders like code, construction, documentation, and sketches, along with specific files like LICENSE-CERN, README.de.md, and assembly files. The repository includes standard GitHub features such as unpinning, watching, and forks. It also displays project metadata like CERN-OHL-W-2.0 licenses and contributors (SamuelNoesslboeck and SchwarZrene).

File/Folder	Description	Last Commit
code	new images	2 weeks ago
construction	new comp	4 days ago
documentation	sync	3 hours ago
drake_printer_files @ a1731e9	sync	3 hours ago
electronics	improved docs	3 months ago
export	added export folder	9 months ago
sketches	new images	2 weeks ago
standard	updated covery	4 months ago
.gitignore	better documentation	2 weeks ago
.gitmodules	updated submodules	4 months ago
LICENSE-CERN	Update licenses	6 months ago
LICENSE-MIT	Update licenses	6 months ago
README.de.md	sync	3 weeks ago
README.md	better documentation	2 weeks ago
a1_drake_without_cover.asm	Hole distance of H8-Lager changed	4 months ago
a_drake.asm	merging branches for current cad status	4 months ago
a_drake.cfg	merging branches for current cad status	4 months ago

Abbildung 2.1.4.1 Github-Website

Des weiteren wurden folgende Meilensteine definiert:

1. **Planung:** Das Grundkonzept sollte gut durchdacht und dadurch mögliche Fehler gleich am Anfang vermieden werden. Aufgrund der hohen Komplexität des Projektes, besonders in der Software, wurde hier viel Zeit eingeplant.
2. **Bau:** Der Bau und die Fertigung des Roboters sind durch die hohe Anzahl der 3D-Druckteile eher eine anspruchsvolle Aufgabe für gute Versionierung. Unsere Fertigungstoleranzen sind hoch, was uns zeitlich viel Spielraum gibt.
3. **Kombination:** Bis zu dieser Phase waren die beiden Teile Software und Hardware nahezu völlig getrennt, was sich in dieser Phase drastisch ändert. Jetzt wird alles kombiniert und fertiggestellt, um erste Tests durchzuführen.
4. **Tests:** In der Testphase werden mögliche Fehler und Probleme ausgeglichen und behoben. Am Ende dieser Phase ist der Roboter fertig und bereit für die Ausstellung.

Der Zeitplan wird ebenso über ein *Github*-Projekt organisiert. Um diesen grob zu veranschaulichen, dient die folgende Grafik:

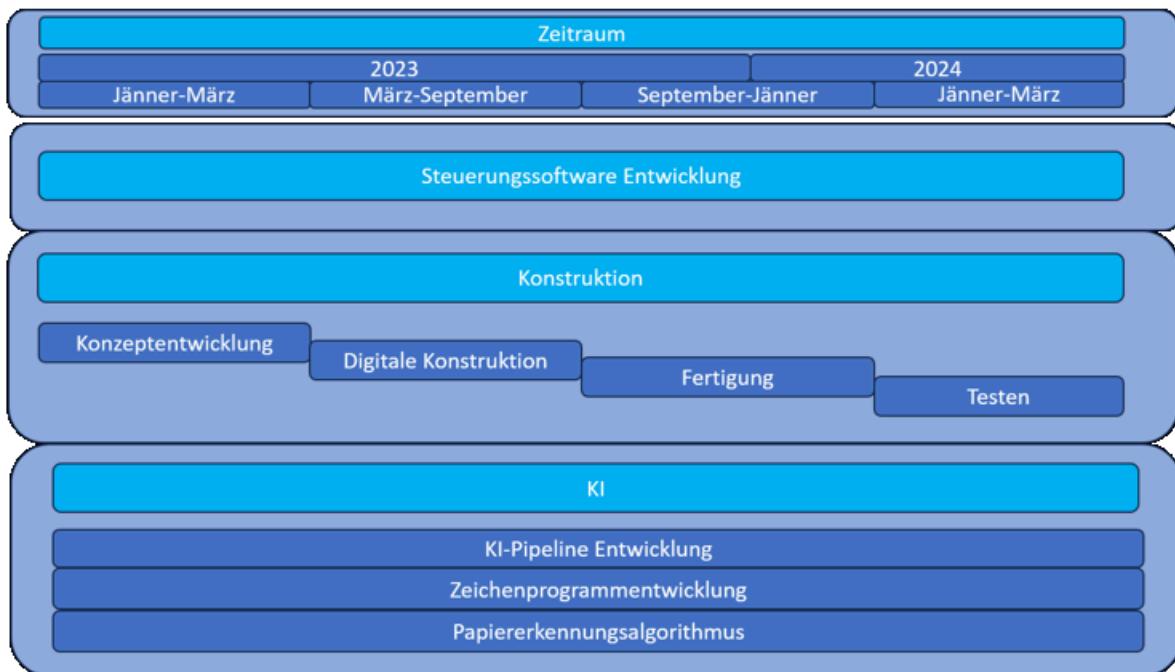
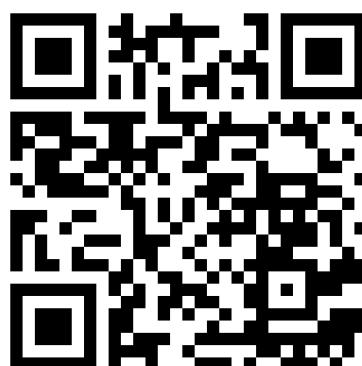


Abbildung 2.1.5.1 Zeitplan



2.4 Kosten

Die Kosten des Roboters belaufen sich auf rund 1500€ und werden vollständig von unserem Schulverein und dem *AEC* getragen.

Kosten

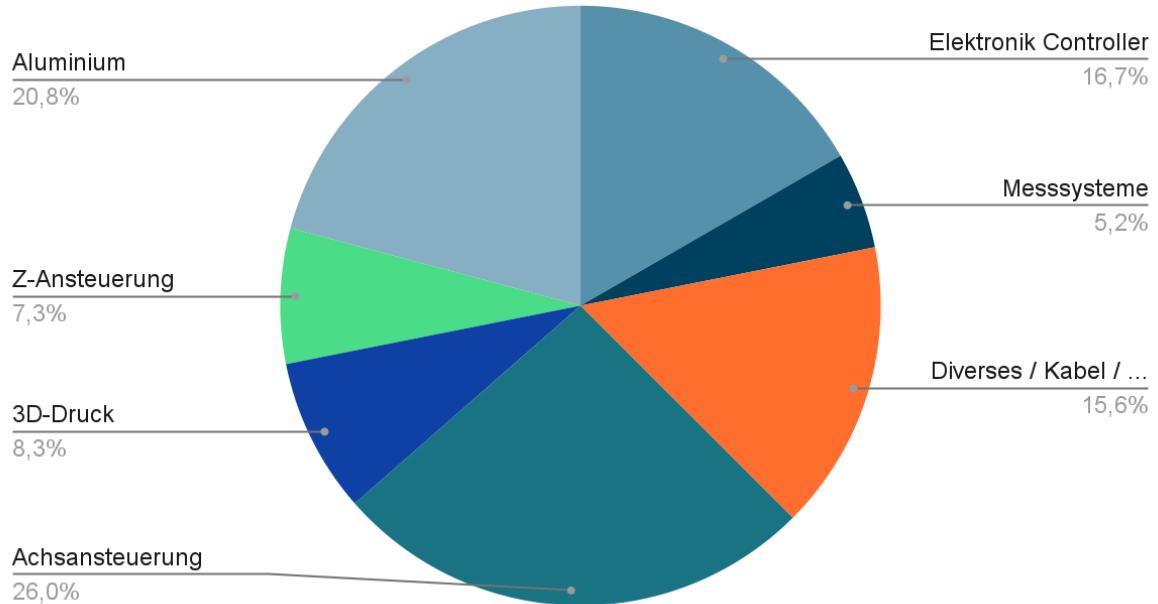


Abbildung 2.4.1 Kostendiagramm

2.5 Projektkoordination

2.5.1 Kompetenzen des Teams

Wie schon im Absatz Ideenfindung beschrieben, haben wir eine Ausbildung im Bereich Automatisierungstechnik. Da der Wissensdurst der Projektentwickler durch diese Ausbildung nicht gestillt wird, beschäftigen wir uns auch in der Freizeit mit diversen Themen, die über das Schulwissen hinausragen.

Samuel Nösslböck hat sein Wissen im Bereich Roboterbau, Elektronik und Kinematik erweitert und auch Projekte wie einen eigenen [Roboterarm](#) oder diverse ferngesteuerte mobile Roboter umgesetzt.

Schwarz Rene hat sich in seiner Freizeit mit Künstlicher Intelligenz beschäftigt, da er schon immer Interesse am menschlichen Bewusstsein und Denken gezeigt hat. Auch hat er Projekte wie eine eigene Spracherkennung oder [Light-Weight Objekterkennung](#) umgesetzt.

Entsprechend der Kompetenzen wurden die Aufgaben folgendermaßen verteilt:

1. Samuel Nösslböck
 - a. *Steuerungssoftware und Elektronik*
 - b. *Entwicklung des ersten Konzeptes und Idee*
 - c. *Fertigung mit 3D-Druck*
 - d. *Finale Montage / Zusammenbau*
 - e. *Kommunikation nach Außen*
2. Schwarz Rene
 - a. *Entwicklung der KI*
 - b. *Design des digitalen Roboters*
 - c. *Diverse Fertigungen*
 - d. *Schriftliche Dokumentation*

2.5.2 Kooperation und Betreuung

Die effiziente Arbeitsteilung hat einen Großteil der benötigten Kooperation hinfällig gemacht, der Rest wurde durch Github ziemlich erleichtert. Andere Angelegenheiten wie zum Beispiel das Networking wurden durch eine frühe Einigung auf ein einheitliches Konzept, Definition der zu benützenden Schnittstellen, Ports etc. geregelt.

Die HTL Neufelden ist eine eher auf Maschinenbau spezialisierte Schule, weshalb die Betreuung im Bereich KI wenig unterstützend war. Auch der Projektpartner, das AEC, sieht uns als unabhängige Künstler, weshalb die Betreuung auf die Definition der Rahmenbedingungen hinausläuft.

Ziel dieses Projekts ist es, die Fähigkeiten von KI bei der Erledigung kreativer Aufgaben in Zusammenarbeit mit Menschen zu demonstrieren und menschliche Kreativität nachzuahmen. Es beschäftigt sich außerdem mit der Frage, wie weit Kreativität überhaupt menschlich ist. Die Demonstration gelingt durch das Einbauen der KI in einen kleinen Roboter für eine neue Museumsstation des Ars Electronica Center Linz.

Für diese Beschreibung wechsle ich nun bewusst zur Ich-Form:

Rene und ich sind schon seit Ewigkeiten Schulfreunde und haben uns gegenseitig immer wieder Inspiration gegeben, mit unseren Interessen weiterzumachen, nicht auf die zu hören, die zweifeln und Dinge sagen wie "Das wird eh nix!". Gemeinsam haben wir uns die verrücktesten Ideen ausgedacht und nun als Abschluss unserer Ausbildung ein Projekt in diesem Maß auszuführen, hat uns sehr, sehr gefreut.

Wir haben die Arbeit aufgeteilt in unsere klaren Stärken und Schwächen und haben uns gegenseitig perfekt ausgeglichen. Es war überhaupt keine Schande zu sagen, dass man ansteht, oder mehr Zeit oder Hilfe braucht. Es herrschte immer gegenseitiger Respekt und Wertschätzung, dafür bin ich meinem Projektpartner sehr dankbar.

Ich schätze seine Fähigkeiten sehr wert und da ich als Koordinator eher der bin, der schreibt und kommuniziert, bin ich geehrt, dass ich seinen Ideen etwas Leben einhauchen darf.

2.6 Reifung und Lizenzierung

Im Laufe des Projekts und mehreren Meetings mit dem *AEC* nahm das Projekt vor allem in Design-Aspekten Form an. Eine dieser visuellen Anpassungen ist zum Beispiel der Rahmen, der durch ein Oktagon ersetzt wurde.

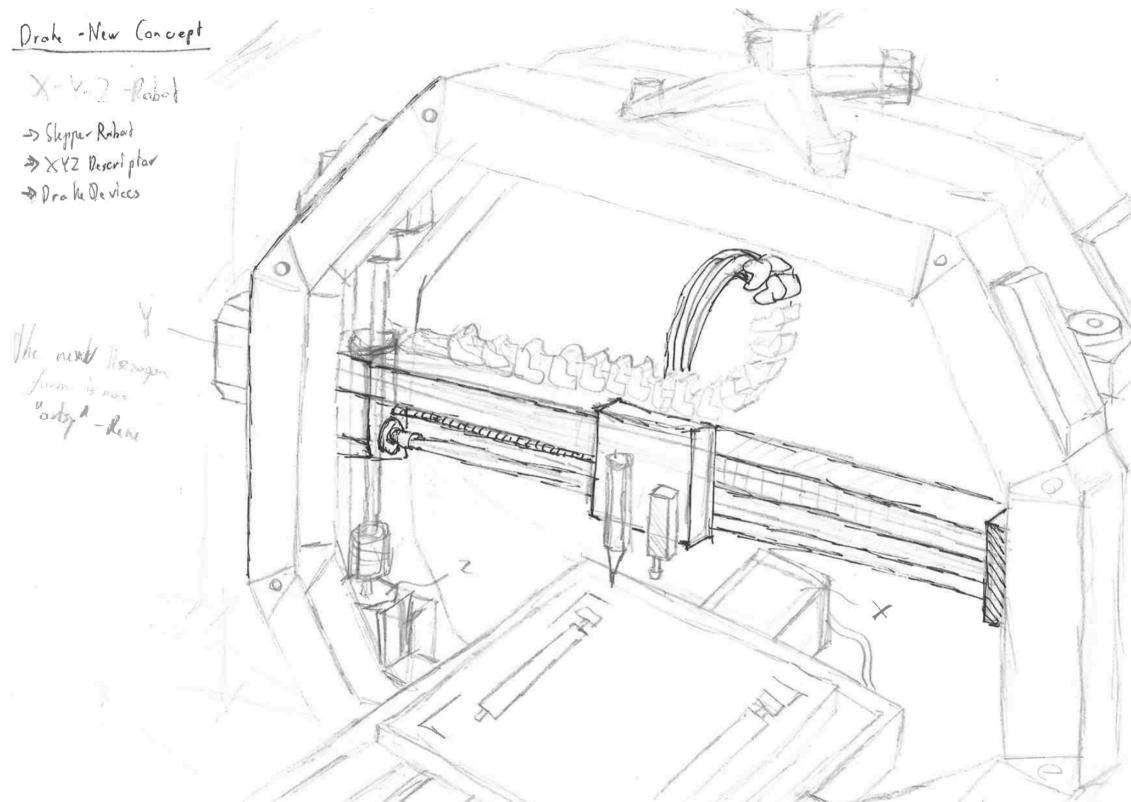


Abbildung 2.1.3.1 Skizze Octagon

Mit einem relativ unberührten Grundkonzept entstand dann im September 2023 ein museumstauglicher Roboter, der im Sommer 2024 ein Teil der Ausstellung werden soll. Das anfänglich eher nebensächliche KI-Projekt entwickelte sich zu einer intensiven Auseinandersetzung mit den Grenzen der KI und den vielen philosophischen Fragen zum Thema.

Das gesamte Projekt sowie das Konzept sind öffentlich und stehen der Allgemeinheit frei zur Verfügung. Die Dokumentation auf *Github* wird auf Englisch geführt, aus eigenem Interesse sowie auf Wunsch des *AEC*.

2.7 Recherche und Innovation

Eine Internetrecherche in Zusammenarbeit mit dem Projektleiter ergab, dass es zwar schon Zeichenroboter gab, die auf gewisse Datensätze trainiert wurden, jedoch funktionierten diese nur auf kleine Datensätze oder Zeichnungen und wiesen nicht die benötigte gewünschte Vielfältigkeit und Kreativität auf. Auch waren viele der existierenden Konzepte nicht für den Museumsbetrieb geeignet.

Die anfängliche Idee des Projektes war, zu einer bestehenden Zeichnung etwas Kreatives hinzuzufügen und die Besucher*innen mit der KI kollaborieren zu lassen. Das Projekt sollte zur Demonstration von kreativer Zusammenarbeit von Künstlicher Intelligenz und Mensch dienen.

Zu Beginn des Projektes existierte die Meinung, künstliche Intelligenz werde in den nächsten Jahren zu einem neuen Werkzeug für die Menschheit, wie die Erfindung des Taschenrechners. Bereits zu dieser Zeit existierten diverse Algorithmen, mit denen sich durch menschliches Einwirken Herausragendes entwickeln lässt. Durch Programme wie *Github Copilot* lassen sich komplexe Programme in beliebigen Programmiersprachen durch eine Textbeschreibung entwickeln. Oft genügt eine einfache Beschreibung des Lösungsansatzes und schon liefert ein *Large Language Model (LLM)* wie *GPT-4* den Programmcode. Was jedoch hierbei wichtig ist, ist die Beschreibung des Lösungsansatzes. Neuronale Netzwerke sind nur so gut wie ihre Trainingsdaten. Befindet sich das Problem nicht in den Datensätzen, so fällt es dem Programm meist schwer, eine komplexe Lösung zu finden, die auch wirklich funktioniert. Hierbei wird die menschliche Kreativität benötigt, um einen Lösungsansatz zu finden, der dann beschrieben und in Code umgesetzt wird.

Mit Bildern ist es das gleiche. Es wird menschliche Kreativität benötigt, um kreative Textbeschreibungen für Bilder zu finden, die im Anschluss gezeichnet werden. Je weiter die Forschung des Algorithmus fortschritt, desto mehr festigte sich die Erkenntnis, dass diese These inkorrekt ist. Der Programmcode hat einerseits die Möglichkeit, aus Bestehendem etwas Neues zu erzeugen und andererseits aus einem weißen Blatt Papier ohne textliche Beschreibungen oder menschliches Einwirken kreative Bilder zu malen. Es ist eine philosophische Frage, ob Maschinen kreativ sein können oder nicht, jedoch sehen die Resultate sehr vielversprechend aus. Die Frage, ob KI kreativ sein kann, wird eher zur Frage, wann KI kreativer wird, als die Menschheit es derzeit ist.

Der zunächst beschriebene Algorithmus versucht die menschliche Kreativität nachzuahmen, indem er generative Künstliche Intelligenz mit zufällig in das Eingangsbild generierten Inhalten kombiniert. Die Innovation liegt in dieser Interpretation menschlicher Kreativität als genau dieser Kombination von zufälligen Impulsen in der menschlichen Wahrnehmung, abgebildet durch die Zufallsgenerator, und der Erkennung von Mustern, abgebildet durch KI.

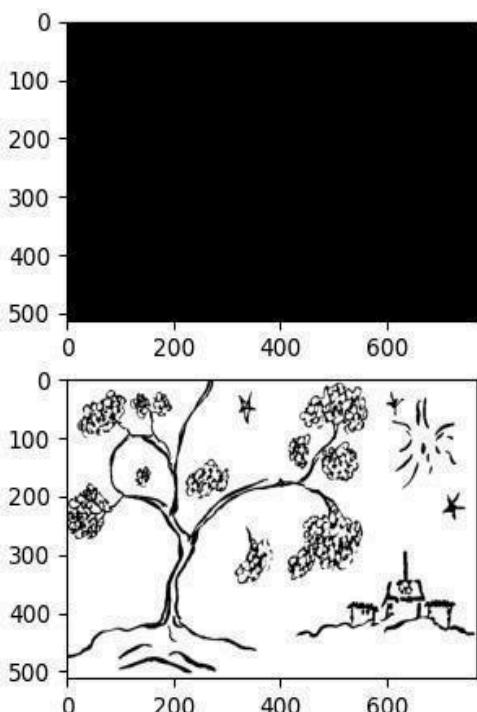
2.8 Wirtschaftlichkeit

Durch den Projektpartner und die Positionierung unseres Projektes wird die Erkenntnis, beschrieben im Absatz Neuheitsgrad, an viele Menschen weitergegeben und es kann breite Masse inspiriert werden, dies ist genau das Ziel dieses Projektes.

Das Konzept lässt sich jedoch enorm erweitern und einsetzen:

Die Pipeline kann mit verschiedenen Formen von Medien gekoppelt und trainiert werden, das heißt, es muss nicht wie in dieser Anwendung eine Bild-zu-Bild Pipeline verwendet werden. Es kann auch Audio-zu-Audio, Video-zu-Video oder Kreuzungen daraus verwendet werden. Die KI wird überall ihren kreativen Beitrag leisten.

Auch könnte man die Software an Künstler oder Menschen, die kreative Ideen benötigen, verkaufen. Die Projektteilnehmer kennen Phasen im Leben, in denen man den menschlichen



Drang verspürt, etwas zu bauen, jedoch keine Idee hat, was umgesetzt werden könnte. Einen genialen Einfall erhält man meistens durch externe Inspiration. Zum Beispiel sieht man eine Spinne, so kann man diese als Inspiration für einen Roboter nutzen. Wäre jemand, wie ein Science-Fiction Drehbuchautor in eben der Situation, so könnte er sich ein paar Bilder generieren lassen, welche er als Aufenthaltsort in Filmen nutzen kann.

Es ist einerseits möglich, zu schon bestehende Bilder etwas hinzuzufügen, andererseits können auch ganz neue Bilder ohne irgendeinen menschlichen Einfluss entstehen. Die Software kann nicht nur von Science-Fiction-Drehbuchautoren genutzt werden. Es wäre nutzbar für jeden Menschen, der sich durch Bilder inspirieren lässt. Auch das repräsentative Bild dieses Projektes wurde durch unsere Software generiert.

Abbildung 2.9.1 White Picture

Würde man die Software weiterentwickeln, so könnte es zu dem Punkt kommen, an dem sie auch auf End-Devices wie Mobiltelefone übertragbar wäre und so könnte man sich jederzeit inspirieren lassen.

Jedoch kann man sich auch selbst mit der KI spielen, da es eine großartige Beschäftigung ist. Es wäre auch einsetzbar für Apps wie Snapchat. Hierbei könnten die Gesichter von Menschen auf die kreativste und unvorhersehbare Weise verändert werden. Die Software hat eine breite Anwendbarkeit. Einerseits zur Unterhaltung und andererseits, um Menschen zu unterstützen.

3 Steuerung

3.1 R-D-S System

Die Steuerung kategorisiert den Roboter in ein *R-D-S-System*, welches den Code strukturiert und eine übersichtliche Gruppierung der Instruktionen ermöglicht. Dieses System setzt sich zusammen aus:

- *Robot*: Beinhaltet alle Instruktionen, die benötigt werden, um die Schrittmotoren zu bewegen, sie zu kalibrieren und gezielt mit ihnen zeichnen zu können
- *Descriptor*: Ist für die Orientierung des Roboters im Raum zuständig was das Management von verschiedenen Stiften oder das Speichern oft verwendeter Positionen
- *Station*: Beschreibt die Umgebung des Roboters und seine Interaktionen mit ihr

In den nachfolgenden Kapiteln werden die einzelnen Teile ausführlich behandelt.

3.2 Robot

3.2.1 Schrittmotoren-Steuerung

Um eine leise und effiziente Bewegung mit geringen Vibrationen zu gewährleisten, wird *Microstepping* verwendet. Dies ermöglicht es den Controller den Strom in den Phasen der Schrittmotoren so zu regulieren, dass ein ganzer Schritt in Halbe, Viertel, Achtel oder sogar Sechzehntelschritte aufgeteilt werden kann.

Des Weiteren wird eine Anlauf- und Ablauf- Kurve benötigt, um korrektes Starten und Stoppen des Motors zu garantieren. Diese Kurven berücksichtigen die maximalen Beschleunigungen, die die Motoren in einem bestimmten Betriebspunkt aufbringen können (Siehe Abbildung 3.2.1 Steppermotorkurve). Außerdem werden die Kurven im Voraus berechnet und bleiben konstant, da der Roboter über keine dynamischen Lasten verfügt.

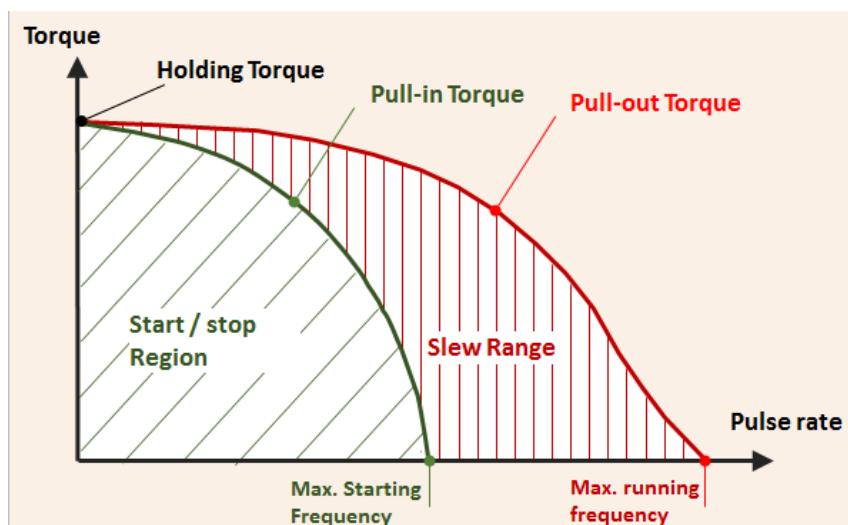


Abbildung 3.2.1 Steppermotorkurve

3.2.2 Tools-Datenbank und Höhenkalibrierung

Außerdem ermöglicht die *Robot*-Datenstruktur die Auswahl verschiedener *Tools*, in diesem Anwendungsfall verschiedene Stifte. Zwar könnten theoretisch verschiedene Stiftlängen mit Hilfe dieser Datenbank und Konfiguration geregelt werden, jedoch wird empfohlen, alle Stifte auf dieselbe Höhe zu kalibrieren und ein einziges Standard-*Tool* zu verwenden.

Die Höhenkalibrierung geschieht entweder automatisch durch den *BL-Touch*, der in den Zeichenkopf integriert wurde oder manuell. Zur Handhabung beider Möglichkeiten siehe 8.2.1 “Kalibrierung des Zeichentisches”.

3.3 Descriptor

Der *Descriptor* konvertiert die gesamten Positionen im Raum in die für Maschine verwendbaren Positionen. Bei einem klassischen Roboterarm mit sechs Freiheitsgraden würde dieser 3D-Koordinaten mit einer entsprechenden *Achsenkonfiguration* in sechs Winkel für die Gelenke umwandeln. Da bei diesem Robotertyp alle Achsen direkt den entsprechenden X-Y-Z-Koordinaten entsprechen und keine zusätzlichen Freiheitsgrade vorhanden sind, fällt der Descriptor simpel aus.

3.4 Station

Die *Station* eines Roboters regelt die gesamte Umgebung und zusätzliche Komponenten des Roboters, wie zum Beispiel den Zeichtisch oder den BL-Touch. Die *Station*-Datenstruktur verwaltet außerdem den gesamten Ablauf mit dem Benutzer.

3.5 Die Programmiersprache Rust

Die gesamte Steuerungssoftware wurde in der eher neuen Systemprogrammiersprache Rust angefertigt. Sie ist eine aus dem Unternehmen Mozilla als Freizeitprojekt eines Mitarbeiters entsprungene Sprache, die zuerst rein für die neue Webengine [Servo](#) gedacht war.

Rust verfügt über einige neue Ansätze was Datenverwaltung, *Runtime-Errors* und mehr betrifft, was die Sprache vollständig memory-safe, type-safe und extrem robust macht. Deshalb eignet sie sich hervorragend für Embedded-Programming, besonders auf dem Raspberry-Pi. Genauerer Aufbau der Libraries und generell von Rust-Projekten wird im [Abschnitt 6.3](#) erklärt.

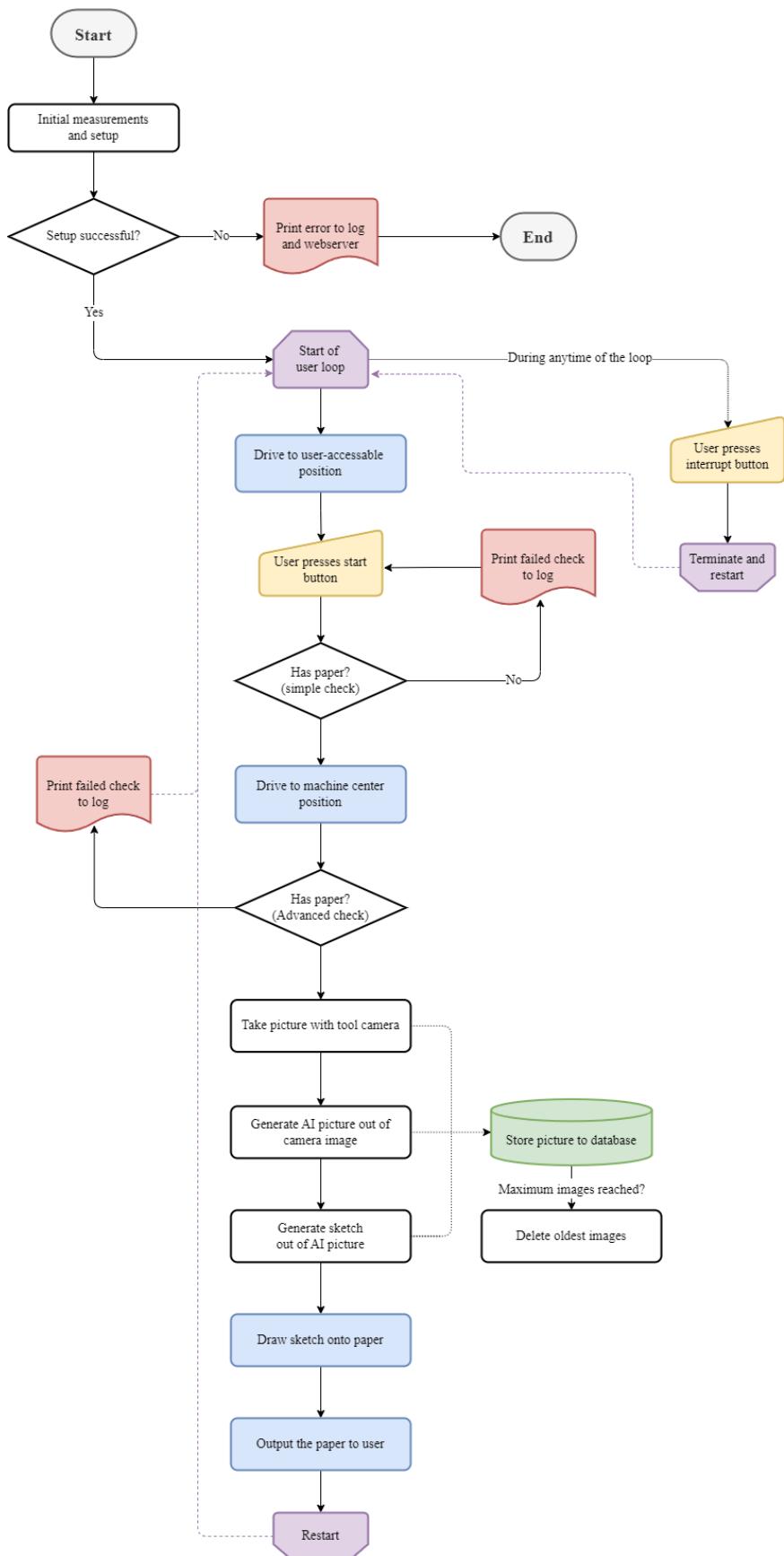


Abbildung 3.5.1 Ablaufdiagramm

4 Konstruktion

4.1 Rahmen

4.1.1 Grundrahmen

Der Grundrahmen hat die rechteckigen Maße $L = 700$, $B = 300$, $H = 80$ [Millimeter] und setzt sich aus Alu-80x40x4 U-Profilen zusammen. Hierbei wurden die U-Profile angewinkelt, um eine bessere Optik zu erzielen und eine längere Schweißnahtlänge zu erhalten. Um die Fertigung zu erleichtern, wurden zuerst die U-Profile bearbeitet und erst danach wurde geschweißt. Jedoch muss nun eine gewisse Längsverziehung berücksichtigt werden.

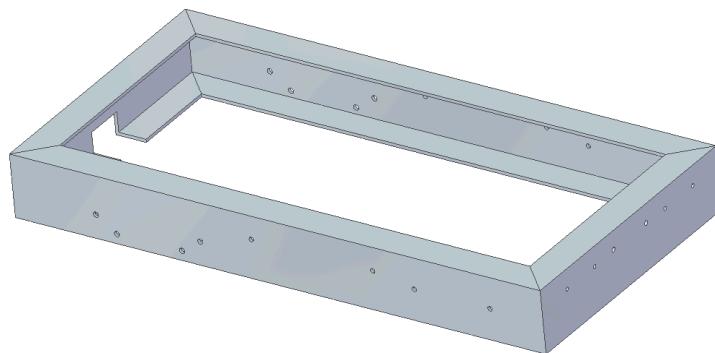


Abbildung. 4.1.1 Rahmen Grundgerüst

4.1.2 Befestigung für X-Achse

Um die X-Achse befestigen zu können, wurden auf der Unterseite des Grundrahmens 40x5x300 Alu Flachstähle angeschweißt.

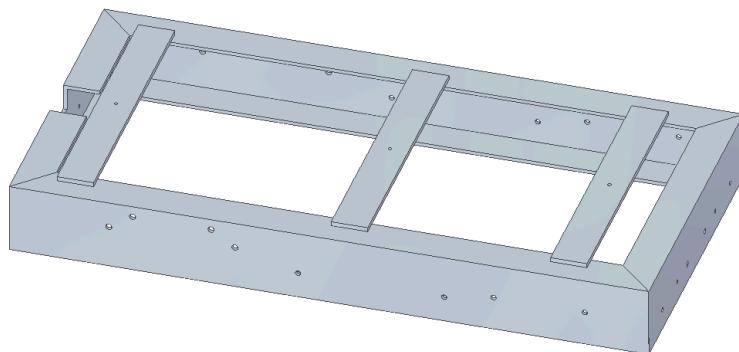


Abbildung 4.1.2 Befestigung X-Achse

Hierbei entsteht kein Nachteil für die Optik, da der gesamte Rahmen verkleidet wird.

4.1.3 Achteck

Um eine bessere visuelle Darstellung zu erzielen, wurde für die Z-Achse ein 8-eckiger Rahmen gefertigt. Damit eine höhere Stabilität erreicht wird, wurde der Rahmen aus 40x40x2 L-Profilen gefertigt. Für die gesamte Konstruktion wurde der Rahmen 2x gefertigt, jedoch ist eine Fertigung eine gespiegelte Version der anderen. Langlöcher im oberen Bereich des Rahmens sollten Fertigungs Ungenauigkeiten ausgleichen und dienen zur Befestigung für die Z-Achse.

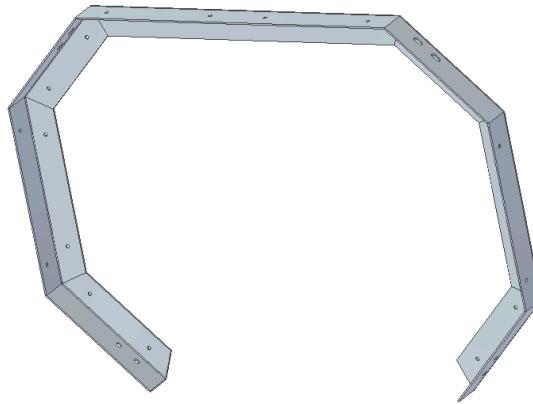
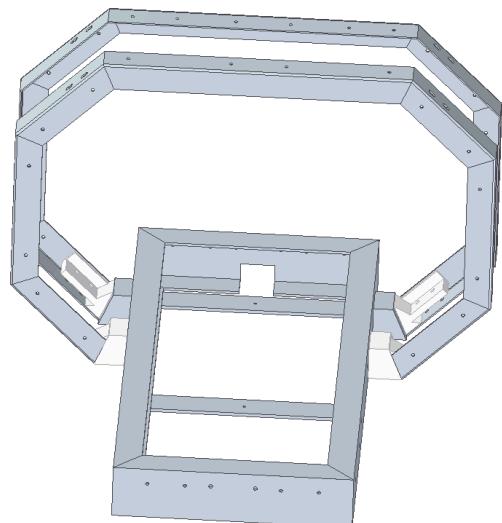


Abbildung 4.1.3 Alu-Achteck

4.1.4 Gesamtrahmen

An die achteckigen Rahmenteile (Absatz 4.1.3) sollten Alu Quader angeschweißt werden, welche mit Schrauben an den Grundrahmen (Absatz 4.1.1) befestigt werden können. Dies dient dazu, den Rahmen leicht und schnell zusammenbauen und zerlegen zu können, um den Roboter transportfähiger zu machen. Aus fertigungstechnischen Gründen war es jedoch nicht möglich, die Alu Quader anzuschweißen, da sie eine zu dicke Wandstärke haben und somit nicht erwärmbar sind. Da sie schon gefertigt waren, wurde ein Hilfsteil 3D gedruckt, welches an den Quader und an den Rahmen angeklebt wurde. Somit blieb die ursprüngliche geplante Zerlegbarkeit intakt ohne großen neu entstandenen Fertigungsaufwand. Zu sehen sind die 3D Druckteile in Abb 4.1.4, in weiß eingezeichnet.

Abbildung 4.1.4 Gesamtrahmen



4.2 X,Y-Achse

Um Beweglichkeit in X und Y Richtung zu ermöglichen, wurden Zahnriemenantrieben von Igus verbaut. Diese sind maßgefertigt und wurden für die X-Achse in der Länge 600 angeschafft und für die Y-Achse in der Länge 400. Die X-Achse wurde auf den in Absatz 4.1.2 beschriebenen Befestigungen angeschraubt. Die Y-Achse ist, wie bei einem 3D-Drucker in Z-Richtung, beweglich.

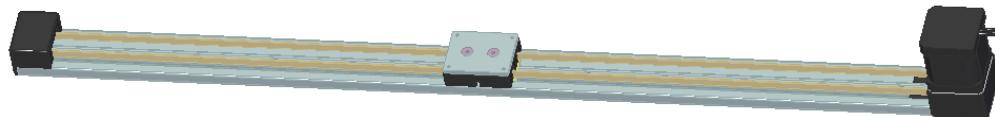


Abbildung 4.2.1 X,Y-Achse

4.3 Z-Achse

Um eine Beweglichkeit in Z-Richtung zu ermöglichen wurden links und rechts Gewindespindeln gebaut. Um die Z-Achse zu führen, wurde jeweils auf beiden Seiten eine Passstange verbaut.

Beide Gewindespindeln sind mittels Stepper Motor angetrieben. Durch separate Antriebe der beiden Gewindespindeln ist es möglich, Fertigungs Ungenauigkeiten auszugleichen. Des weiteren wurden die Gewindespindeln mit einer Klauenkupplung an den Stepper Motor befestigt.

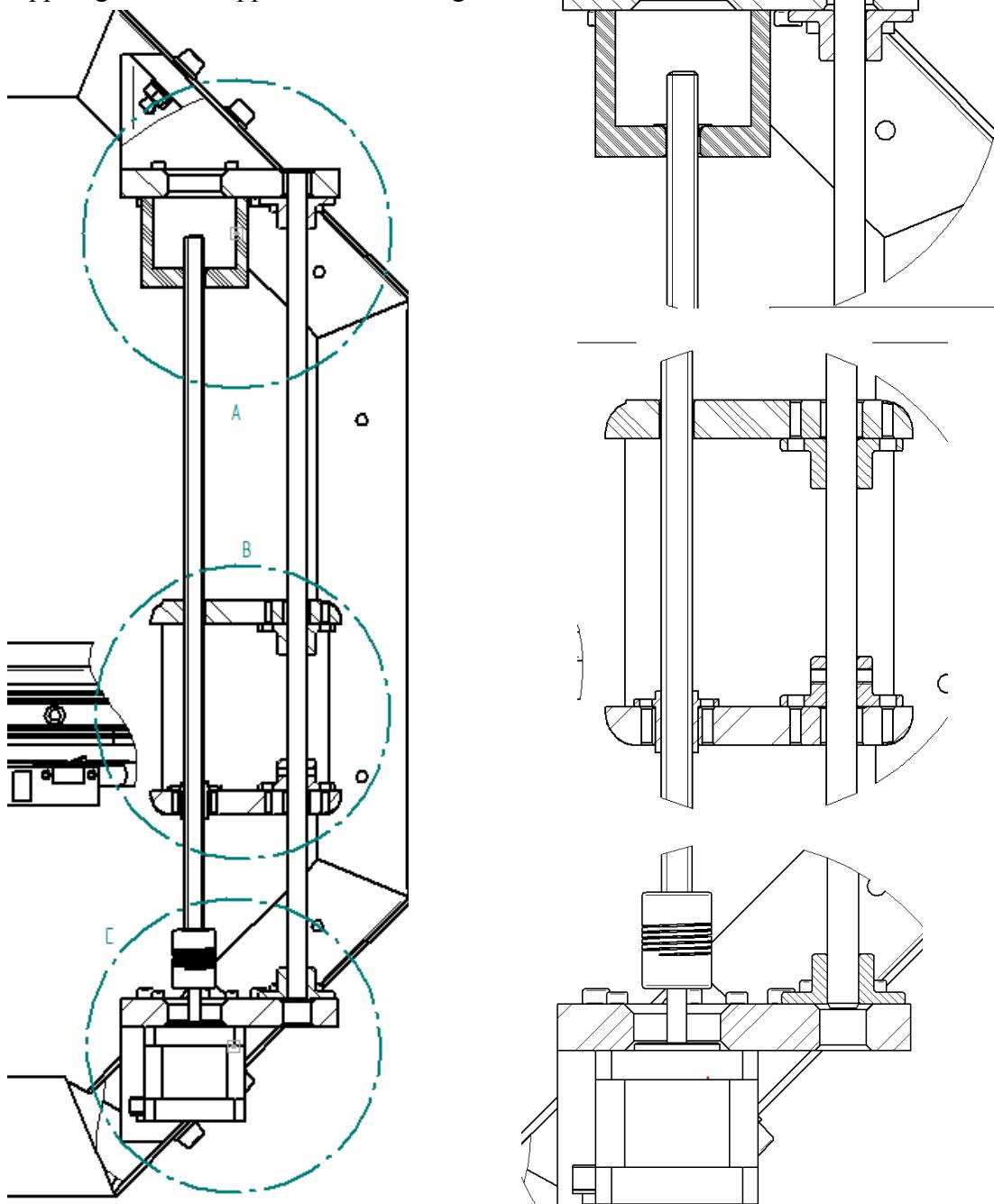


Abbildung 4.3.1 Rechte Z-Achse

4.4 Befestigungstisch

Damit das Papier während des Zeichenvorgangs nicht verrutscht, wird es mittels von Micro-Servos gesteuerten Klemmen auf den Tisch gepresst. Hierbei wurden 8 verstellbare Klemmen verbaut. Zu sehen sind die Klemmen in Abb 4.4.1 in grau. Durch die Vielzahl an Klemmen ist es möglich, eine Klemme zu heben, um an der gewünschten Position zu zeichnen und noch genug Anpressdruck durch die restlichen Klemmen zu erzeugen, damit es nicht verrutscht. Der gesamte Tisch wurde aus 3D-Druck gefertigt. Um für die Kamera Referenzierungen zu liefern wurden auf den Tisch sogenannte Marker gezeichnet (Näher beschrieben in Kapitel ...)

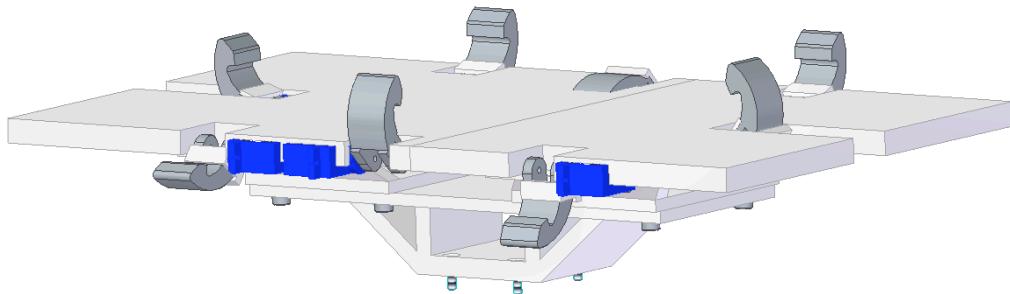


Abbildung 4.4.1 Befestigungstisch

4.5 Stifthalter

Der Stifthalter dient als Einspannvorrichtung für den zeichnenden Stift. Der Stift wird mittels beweglicher Backe (Abb 4.5.1 in rot) festgespannt. Die bewegliche Backe kann mittels den Verstellschrauben (Abb 4.5.1 in blau) verstellt werden. Um den Roboter kalibrieren zu können, wird ein 3D-Touch Sensor verwendet (Abb 4.5.1 in grau).

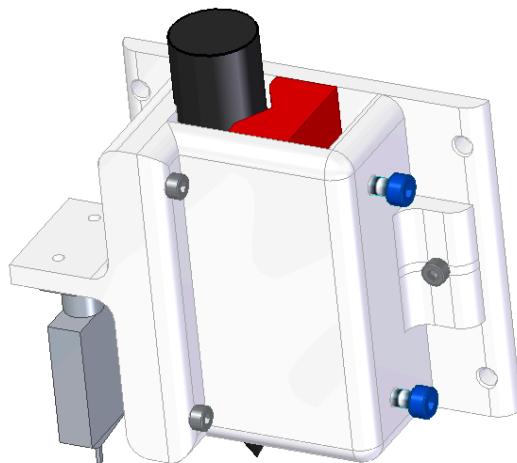


Abbildung 4.5.1 Stifthalter

4.6 Sicherheitstechnik

Da der Roboter Museums Tauglichkeit vorweisen muss, muss der Roboter so konstruiert werden, dass niemand hineingreifen und sich verletzen kann. Dies ist der Grund, warum die X-Achse die 600 Millimeter Länge besitzt, beschrieben im Absatz 4.2. Da sich der Roboter mit einer hohen Geschwindigkeit bewegt und so einige Möglichkeiten bietet für ein kleines Kind, sich zu verletzen, wird der gesamte Zeichenvorgang mit Plexiglas abgeschirmt. Die Vorderseite des Plexiglases wurde so eingebaut, dass sie schnell und leicht zu entfernen ist, da sie entfernt werden muss, wenn der Stift gewechselt wird. Stift wechseln, beschrieben in Absatz 9.2.2.

4.7 Gesamtkonstruktion

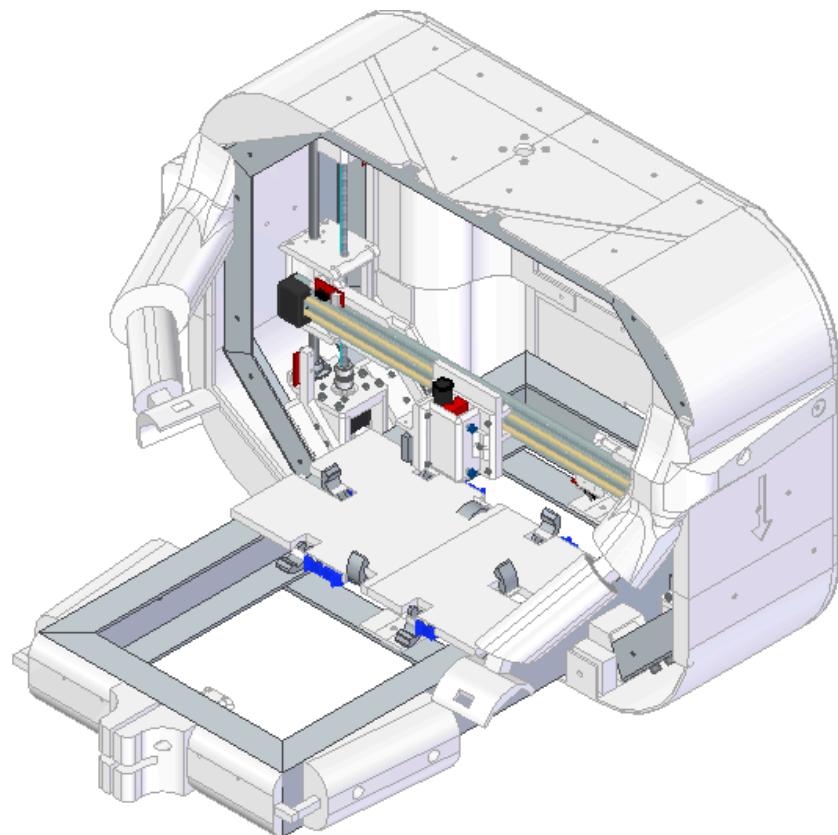


Abbildung 4.7.1 Gesamtkonstruktion

4.8 Kamera / Papiererkennung

Durch eine Kamera montiert in der rechten Hand im rechten Arm wird das Papier fotografiert. Da die Kamera nicht aus dem Zentrum heraus fotografiert. Muss das Papier in die Vogelperspektive transformiert werden. Hierzu wurden auf der Papierplattform Marker eingebaut, welche es ermöglichen, sich zu referenzieren. Beschreiben im Absatz 4.4. Da das Papier durch Servomotoren an den Tisch gepresst wird, werden mehrere Fotos gemacht. Bei jedem Foto wird ein anderer Servomotor geöffnet. Dies dient dazu, dass kein Teil der Zeichnung bedeckt bleibt.

5 KI

5.1 Generelles

5.1.2 Ziel

Das Ziel der Künstlichen Intelligenz besteht darin, das existierende Bild zu erkennen, zu interpretieren und schlussendlich einen kreativen Beitrag zu leisten. Wie bereits im Kapitel 2.2.1. erwähnt, ist der Prozess zeitlich etwas begrenzt und sollte deshalb effizient und schnell sein.

5.1.3 Einzigartigkeit

Bestehende Algorithmen erzielen herausragende Resultate, wenn es darum geht, aus einer Textbeschreibung ein Bild oder Text zu generieren. Der *DrAI* Algorithmus ermöglicht es, dass diese ursprüngliche Textbeschreibung nicht zwingend notwendig ist, um weiteres Material zu erstellen. Es ist einerseits möglich, auf einer “leeren Leinwand” etwas zu erzeugen, andererseits kann, basierend auf einem bestehenden Bild, etwas generiert werden. Algorithmen dieser Art könnten auch zur eigenen Inspiration dienen, indem andere Denkweisen der KI offenbart werden. In manchen Fällen ist es eine Herausforderung, die Kreativität und Einzigartigkeit der Bilder zu erkennen, bei eindeutigen Skizzen des Users oder der Userin fällt jedoch das Ergebnis klar und deutlich aus.

5.1.4 Kontrolle

Mithilfe diverser Parameter lässt sich effektiv in den Algorithmus eingreifen. Vereinfacht gesagt gibt es Parameter wie den Grad der Kreativität, Abstraktheit, Menge, Dichte und viele weitere.

Die Parameter wurden vorerst dem Auftraggeber entsprechend angepasst, jedoch können sie jederzeit abgeändert werden. Im Konzept des Algorithmus sind auch Möglichkeiten wie dynamische Parametrisierung enthalten, das heißt Anpassung bestimmter Parameter wie zum Beispiel Menge und Dichte des zu Generierenden auf Basis der bereits vorhandenen Menge in einem bestimmten Bereich des Bildes.

5.1.5 Ergebnisse

In den folgenden Abbildungen sind Tests und Experimente mit dem *DrAI* Algorithmus dargestellt. Das obere Bild ist von uns oder Freund*innen digital angefertigt worden, das jeweils untere das Ergebnis nach der Verarbeitung durch den Algorithmus.

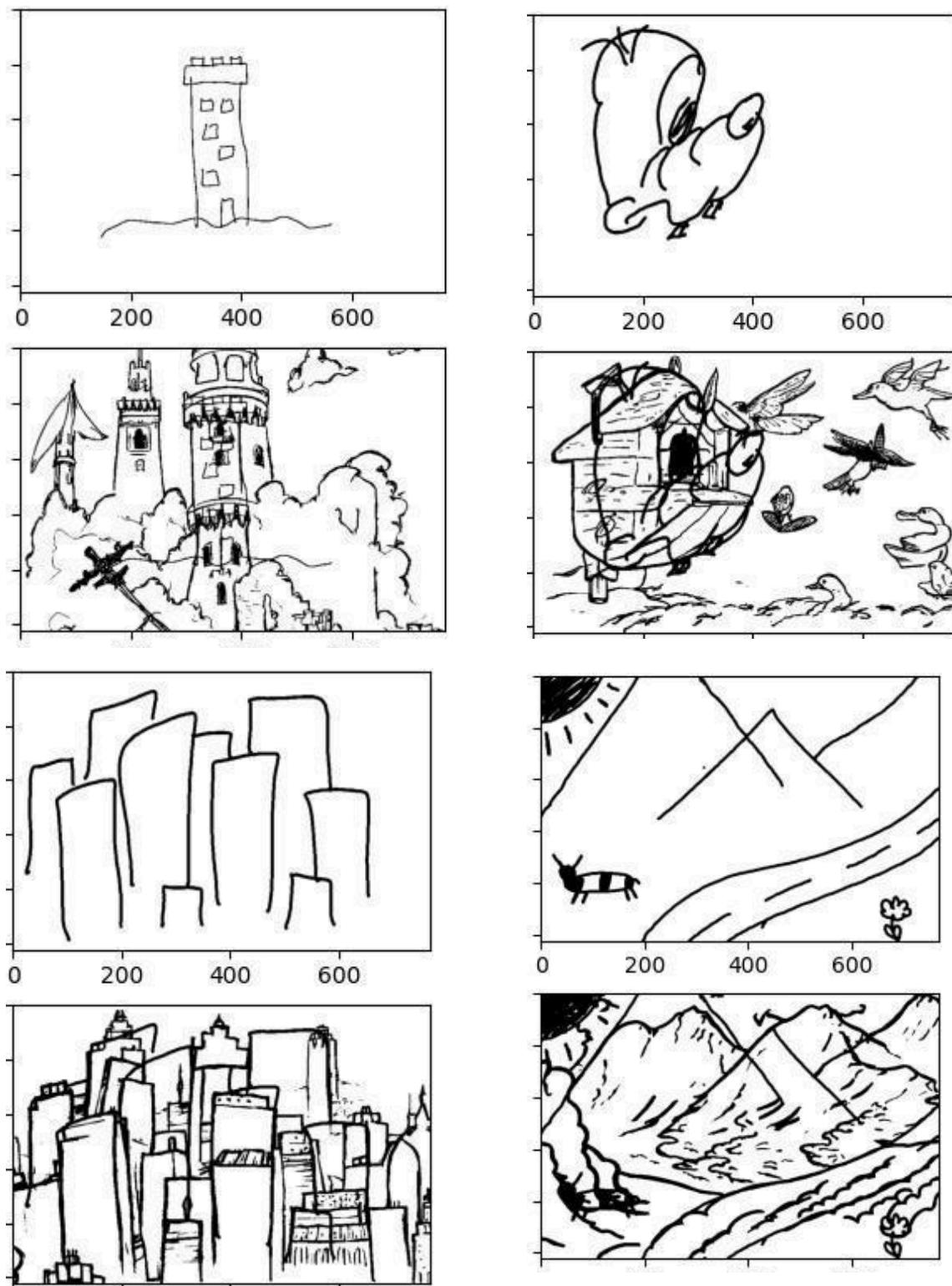


Abbildung 5.1.5.1 KI-Resultate

5.2. Entstehung / Experimente

Die nachfolgenden Kapitel behandeln bestehende Algorithmen und schlussendlich die verwendete Kombination und Verbesserung von ihnen für den benötigten Anwendungsfall.

5.2.1. Eigenes Neuronales Netzwerk

Da in diesem Anwendungsfall keine Flächen mit Farbe bemalt werden sollten, wurde die Idee geboren, ein eigenes *Generative Adversarial Network* (Wikipedia, 2017) zu trainieren. Dies ist eines der ersten vielversprechenden Konzepte im Bereich Generative AI. Hierbei werden zwei Netzerke trainiert: der Generator und der Discriminator. Der Generator erzeugt aus einem Random Noise oder einem Eingangsbild ein modifiziertes Ausgangsbild, der Discriminator hingegen erhält als Eingangsparameter ein echtes oder ein generiertes Bild und wird darauf trainiert, zu erkennen, ob das Eingangsbild echt oder generiert ist. Im besten Fall lernt der Generator, realitätsnahe Bilder zu erzeugen, damit ein Discriminator nicht mehr zwischen real und generiert unterscheiden kann.

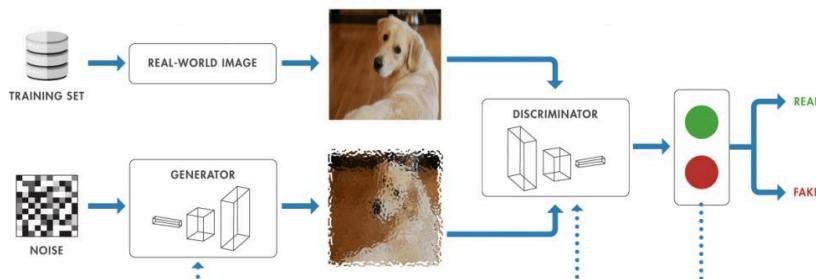
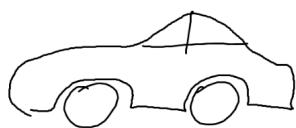


Abbildung 5.2.1 GAN-Architektur

Da nur beschränkt Rechenleistung zur Verfügung steht, wurde vereinbart, nur Landschaftsbilder als Trainingsdaten zu verwenden. Die Eingaben des Benutzers oder der Benutzerin werden als reine Umrisse betrachtet. Um die Trainingsdaten der Eingangsdaten des Benutzers / der Benutzerin anzupassen, wurden aus den Landschaftsbildern die Kanten mithilfe des *Holistically-Nested Edge Detection* (Saining Xie, Zhuowen Tu, 2015) Algorithmuses Kanten extrahiert.

Da es die Aufgabe des Roboters ist, Konturen zur Originalzeichnung hinzuzufügen, wurden Bildbereiche / Linien aus dem Originalbild entfernt. Der Generator sollte lernen, die entfernten Linien zu rekonstruieren. Um nun die menschliche Kreativität nachzuahmen, werden zufällig normalverteilte Werte dem Bild hinzugefügt.



Originalbild (Output für den User)



Eingangsbild (Input KI)

Abbildung 5.2.2 GAN-Trainingsdaten

Die Vorteile dieses Algorithmuses wären:

- Geringer Rechenaufwand im Betrieb
- Gute *Parametrisierung*, besonders im benötigten Anwendungsfall

Die Nachteile jedoch wären:

- Hoher rechenaufwand in der Trainingsphase
- Aufwendige Beschaffung von Daten
- Geringer Grad an Kreativität
- Hohe Spezialisierung notwendig (z.B. nur Landschaftsbilder)

Die Resultate erwiesen sich als wenig vielversprechend. Es war kein Hauch von Kreativität zu erkennen und der Algorithmus drohte, rein schwarze Bilder zu generieren, da der Discriminator zu gut erkannte, welches Bild echt und welches generiert wurde. Dies führte dazu, dass der Generator nicht mehr effizient lernen konnte und sich die Ergebnisse wenig bis kaum verbesserten.

5.2.2. StablePipe

5.2.2.1. Stable Diffusion

Um die Probleme eines selbst trainierten Algorithmus zu lösen, wurde *Stable Diffusion* benutzt, ein von *Stability AI* vor trainiertes Netzwerk. Dieser Algorithmus benutzt Text, um Bilder zu erstellen. Hierbei wird wie bei einem *Generative Adversarial Network* aus einem zufällig normalverteilten Bild ein für uns menschliches Bild erzeugt. Der Algorithmus wird trainiert indem zu einem für das menschliche Auge sinnhaften Bild normalverteilte Werte hinzugefügt werden, sodass das Bild über Zeit an Sinnhaftigkeit verliert. Ziel des Netzwerkes ist es nun, über den Verlauf der Zeit den *Diffusion*-Prozess umzukehren und basierend auf dem diffundierten Bild und dem Text Prompt das Originalbild wiederherzustellen.

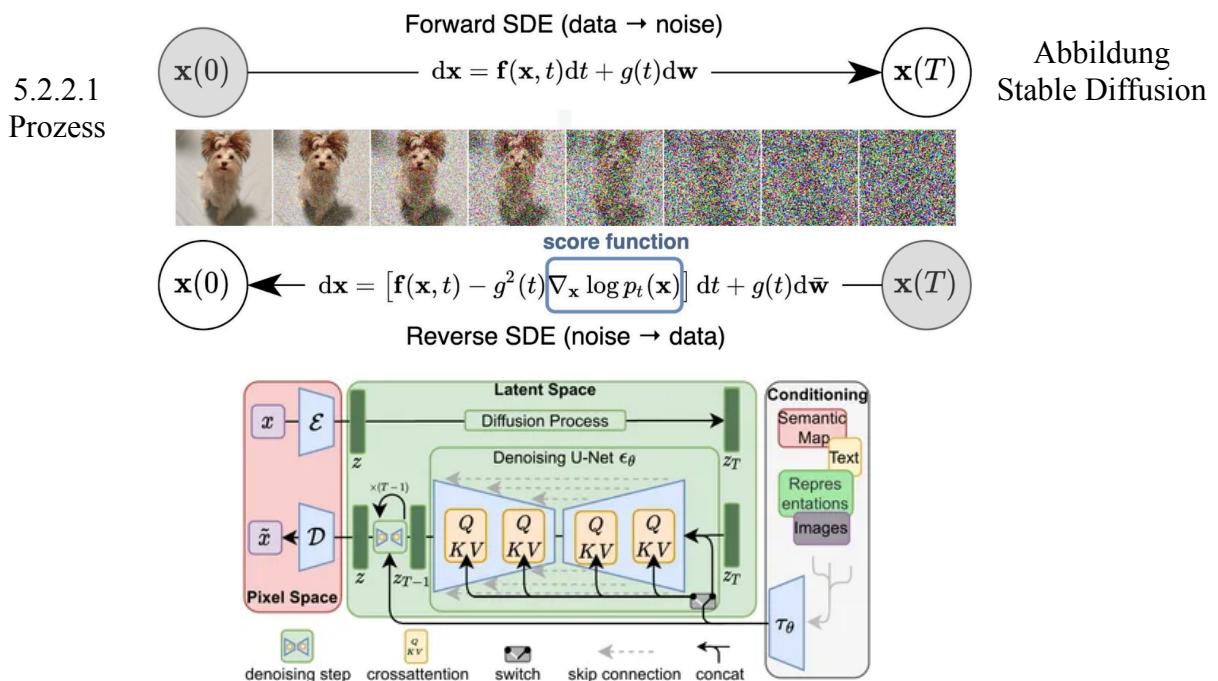


Abbildung 5.2.2.2 Stable Diffusion Architecture

Die erste Version dieses Netzwerkes wurde auf dem *Laion2B-en* (Laion, n.d.) Datensatz trainiert. Dies ermöglichte es dem Netzwerk eine Vielzahl an Bildern zu konstruieren und viele Bereiche, wie zum Beispiel Landschaften, Städte und vieles mehr, basierend auf Textbeschreibungen zu rekonstruieren. Je einfallsreicher die Textbeschreibung, desto einfallsreicher das generierte Bild.

5.2.2.1. StablePipe Pipeline

Mithilfe des *Stable Diffusion* Algorithmus, beschrieben im Absatz 5.2.2.1, ist es möglich, aus einem bestehenden Bild und einer Textbeschreibung ein neues Bild zu generieren. Die *Stable Pipeline* malt zusätzlich zufällig generierte Linien auf das Originalbild. Durch den Diffusionsprozess sollten die Linien verschwinden und nach dem *Forward Process* würden danach komplexe Zeichnungen entstehen. Die Resultate waren weniger vielversprechend, da der Diffusionsprozess keine konkrete Textbeschreibung besaß und so keine komplexen Formen entstanden.

5.2.3. InterStablePipe

5.2.3.1. Clip-Interrogator

Da der *Stable Diffusion Algorithmus* (Absatz 5.2.2.1) einen Text Prompt benötigt um kreative Bilder zu erzeugen und um zu wissen, wovon das Bild handelt, wird ein *Interrogator* (Pharmapsychotic, n.d.), benutzt, um aus dem vom Benutzer gezeichnetem Bild eine Beschreibung zu erhalten.

5.2.3.2. Line Extraction Software (LES)

Ein digitales Bild kann auf verschiedenste Weisen repräsentiert werden. Die bekanntesten Darstellungen sind *RGB*, *HSV* oder *Grayscale*, ersteres wird erzeugt von *Stable Diffusion*. Die *RGB*-Darstellung setzt sich aus einem roten, einem grünen und einem blauen Kanal zusammen und ergibt ein für das menschliche Auge in Farbe sichtbares Bild. Da nur Linien aus dem Bild extrahiert werden, wird es zu schwarz-weiß konvertiert. Jedes Pixel wird in dieser Darstellung entweder durch den Wert 0 (schwarz) oder 1 (weiß) repräsentiert. Im ersten Schritt wird ein Bild erstellt, welches die Pixel darstellt, die im Originalbild nicht bemalt wurden, im generierten Bild jedoch schwarz sind. Der Algorithmus extrahiert mithilfe des *Canny-Edge-Detection-Algorithmus* nun Kanten aus der Grafik. Diese werden nun zu Polygone mit Hilfe der *OpenCV2* Bibliothek verbunden, welche dann, mit einer definierten Stiftgröße in Pixel, in das Originalbild gezeichnet werden. Dieser Prozess wiederholt sich so oft, bis alle Pixel, die der *Stable Diffusion* Algorithmus generiert hat und im Originalbild nicht gezeichnet waren, gezeichnet wurden. Das Output wird dann an die Steuerungssoftware in der Form einer langen, meistens mit mehreren zehntausenden Einträgen, Liste an Linien weitergegeben.

Der Zeichenprozess wird dann optimiert, indem der kürzest mögliche Weg zwischen Linien kalkuliert wird und die rekursive Struktur ermöglicht sogar die effiziente Bemalung von Flächen.

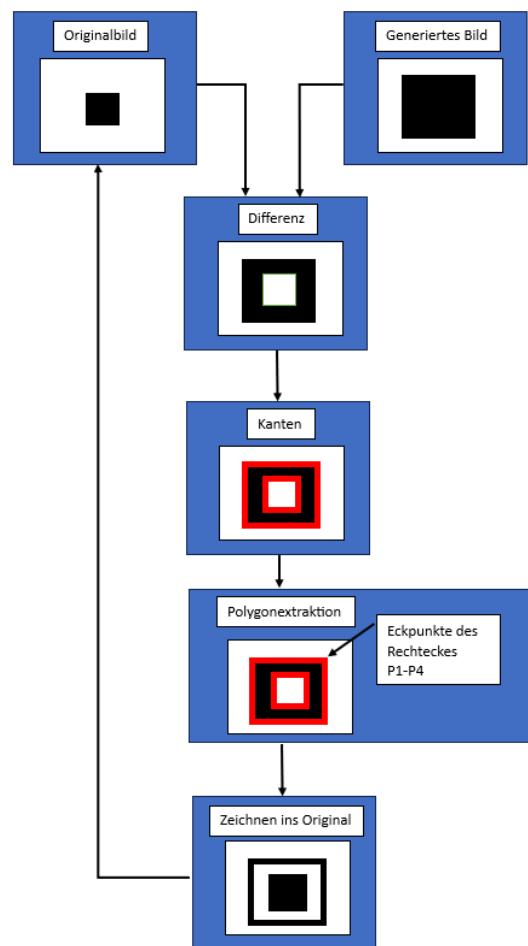


Abbildung 5.2.3.2.1 Line Extraction Software

5.2.3.3 InterStablePipe Pipeline

Um zu einer neuen Zeichnung zu kommen wird zuerst aus dem Bild eine Beschreibung mittels des *Interrogators* (Absatz 5.2.3.1) erzeugt. Diese wird dann in den *Stable Diffusion* Algorithmus (Absatz 5.2.2.1) gegeben und ein neues Bild wird generiert. Aus diesem werden nun Linien mittels der LES (Absatz 5.2.3.2) extrahiert, welche dann an den Roboter gesendet werden können.

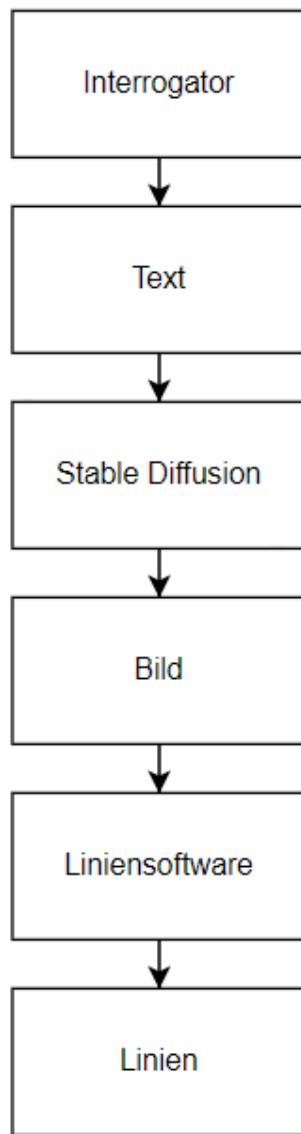


Abbildung 5.2.3.3 InterStablePipe Pipeline

Dieser Algorithmus hat zwei Hauptprobleme:

1. Es wird ein neues Bild generiert, welches mit dem Thema des Benutzers übereinstimmt, jedoch verschwinden die originalen Linien, welche initial gezeichnet
2. Der Algorithmus überlegt sich nichts Neues, was er hinzufügen kann

5.2.4 InterStableCloud

5.2.4.1 Wordcloud

Im Bereich des *Natural Language Processing* werden Wörter in Vektoren umgewandelt, damit mathematische Operationen mit ihnen durchgeführt werden können. Ein Effekt der dann auftritt ist, dass wenn Wörter sich ähneln, es auch ihre Vektoren tun. Ein Musterbeispiel der *Wordcloud* (Wikipedia, 2024) ist, wenn der Vektor des Wortes “King” mit dem Vektor des Wortes von “Women” addiert wird, so kommt der Vektor von “Queen” heraus. Um aus Wörtern Vektoren erzeugen zu können, wird ein Wort in einen *One-Hot Vektor* verwandelt. Ein *One-Hot Vektor* besteht vollständig aus Null-Werten bis auf eine Eins, deren Index die Position in einer Liste aus allen Wörtern des Trainingsmaterials repräsentiert.

Trainiert wird dieser Algorithmus, indem der *One-Hot Vektor* mit den Spalten “n” (n = Anzahl an bekannten Wörtern in den Trainingsdaten) mit einer zu trainierenden Matrix “n” x “m” multipliziert wird, “m” ist die hierbei Anzahl an Vektor Spalten der *Wortvektoren*. Nun werden “t” Wörter vor und nach dem Wort in Vektoren verwandelt und miteinander addiert. Als Summe sollte, wie in Abbildung 5.2.4.1 illustriert, das Original herauskommen,

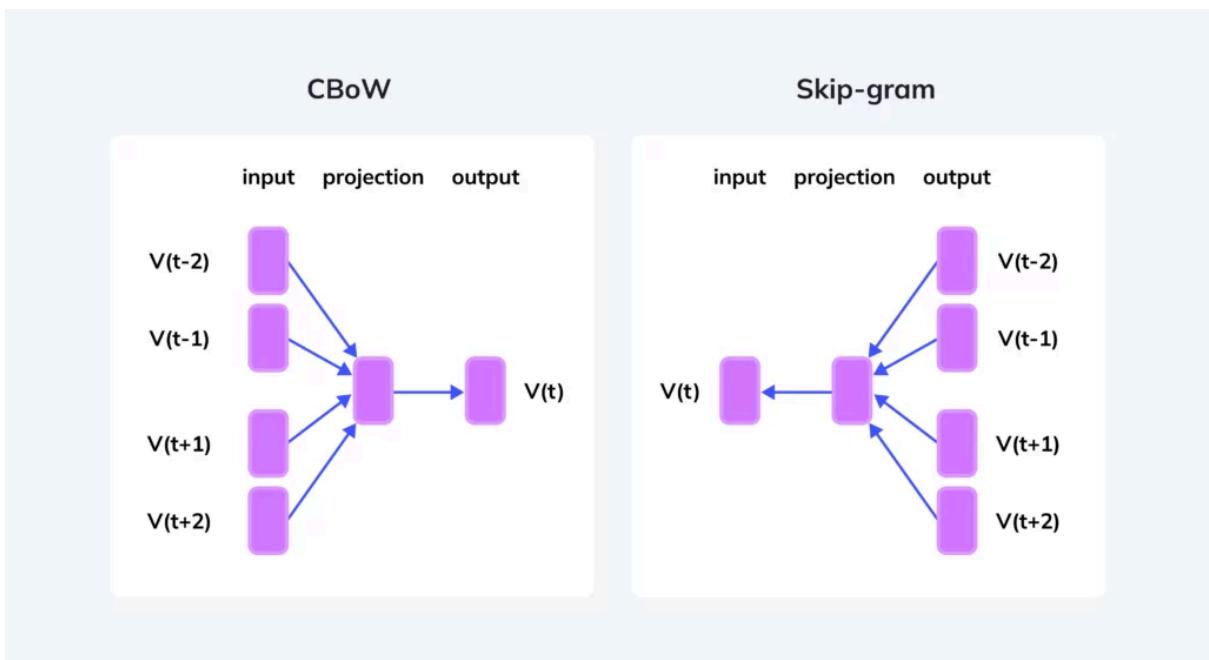


Abbildung 5.2.4.1 Word Cloud-Training

5.2.4.2 InterStableCloud Pipeline

Die *InterStableCloud Pipeline* ist eine Erweiterung der im Absatz 5.2.3 beschriebenen *InterStable Pipeline*. Die Probleme, die in 5.2.3.3 beschrieben werden, werden durch zwei Änderungen teilweise behoben.

1. Änderung bei WordCloud (Absatz 5.2.4.1)

Um das Problem der Kreativität zu beheben, wird der Text aus dem Bild mit Hilfe des *Interrogators* extrahiert. Nun werden aus dem generierten Text die Hauptwörter extrahiert. Zu jedem Hauptwort werden nun "n" ähnliche Wörter in der Wordcloud gesucht. Hierbei wird das Hauptwort in einen Vektor verwandelt. Nun kann zu jedem bekannten Wort die Vektor Distanz berechnet werden. Um nun ein ähnliches Wort zu bekommen, wird ein zufälliges Wort ausgewählt, welches im Rahmen einer definierten Vektor-Distanz liegt. Dies ermöglicht es, zum Beispiel aus dem Hauptwort "Haus", Wörter wie "Garten", "Türe", oder "Fenster" zu erhalten. Durch die zufällige Auswahl wird jedesmal etwas anderes gezeichnet und je kleiner die Vektor Distanz, desto mehr hat das Wort mit dem Hauptwort zu tun. Zu guter Letzt werden die Wörter zur Textbeschreibung des Originalbildes hinzugefügt.

2. Änderung bei Img2Img

Da die in Absatz 5.2.3.3 beschriebene Pipeline das Problem aufweist, das Originalbild nicht zu berücksichtigen, dient nicht nur die neu generierte Beschreibung als Input für den *Stable Diffusion* Algorithmus, sondern auch das Originalbild. Nun können durch *Strength* und *Prompt Guidance* die neu generierten Bilder kontrolliert werden. Je höher die beiden Werte sind, desto kreativer wird das Ausgangsbild. Der Effekt dieser beiden Werte ist illustriert in Abbildung 5.2.4.2.1. Zu sehen ist, dass durch Steigerung der beiden Werte mehr zum Originalbild generiert wird.

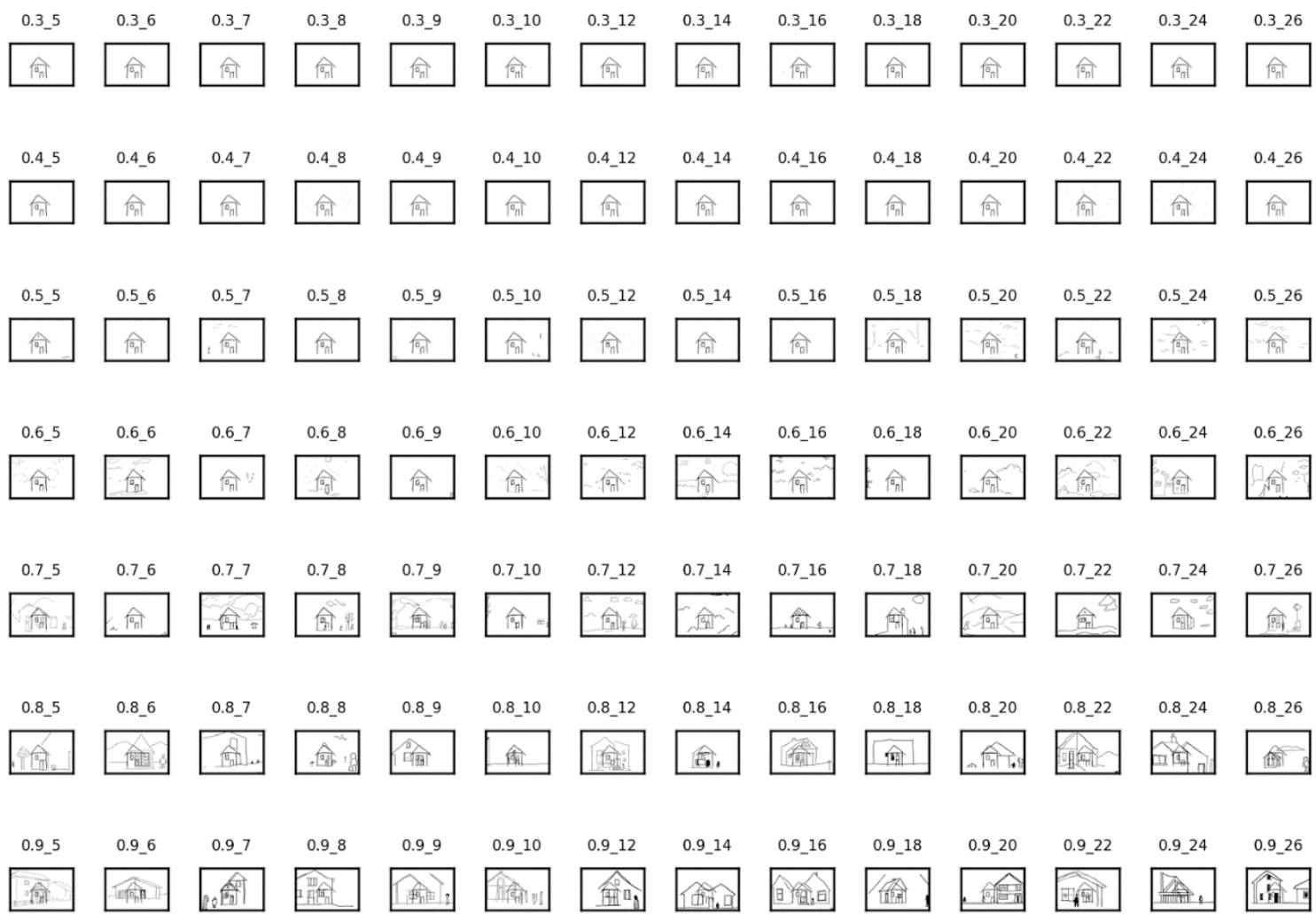


Abbildung 5.2.4.2.1 Guidance Scale

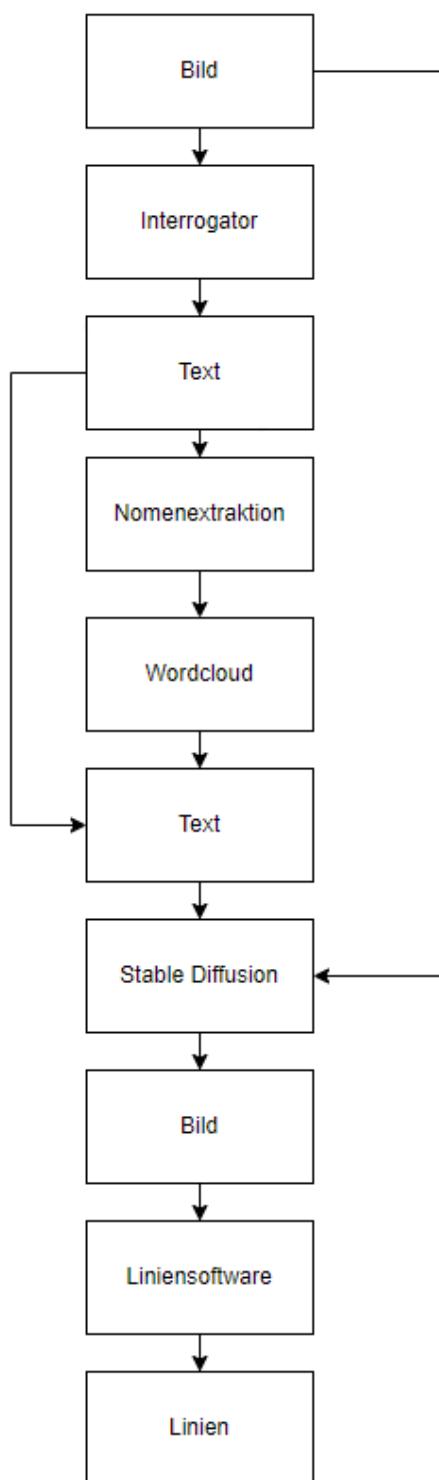


Abbildung 5.2.4.2.2 InterStableCloud Pipeline

Der Algorithmus liefert erste brauchbare Resultate, jedoch entsprechen die Resultate nicht dem gesetzten Ziel, da nur manche Bilder das Maß an Kreativität aufweisen, das vom Auftraggeber gewünscht wurde.

5.2.5 InterStableLLM Pipeline

5.2.5.1 Motivation

Bekommt ein Mensch die Aufgabe, ein Bild kreativ zu erweitern, so erfolgt dies in mehreren Schritten:

1. Er erkennt wovon das Bild handelt
2. Er überlegt sich, was hinzufügen werden könnte, oft durch zufällige Impulse in der Wahrnehmung
3. Er zeichnet dies auf das Papier mit dem Originalbild im Hinterkopf und verknüpft dabei die neuen Impulse durch Bekanntes, vertraute Bilder und Muster

Dieses Schema kann man dank neuester Technologie, welche zu Beginn dieser Experimente noch nicht im Trend lag, auch auf eine Maschine übertragen.

1. Ein *Interrogator* (Absatz 5.2.3.1) erkennt wovon das Bild handelt
2. Ein *Large Language Model* (Absatz 5.2.5.2) „überlegt“ sich kreative Beschreibungen
3. *Stable Diffusion* zeichnet das Neue auf das Originalbild

5.2.5.2 Large Language Model

Ein *Large Language Model* (Wikipedia, 2024) versucht, das nächste Satzzeichen / Wort vorherzusagen. *Language Models* bauen derzeit auf den *Transformer* Algorithmus auf. Zuerst verwandelt ein *Large Language Model* Wörter in Vektoren, danach werden diese Vektoren durch eine Reihe von Attention Layer und Feed Forward Layer verändert. Diese Vektoren werden so lange trainiert, bis das *LLM* lernt, das nächste Wort mit einer gewissen Wahrscheinlichkeit vorherzusagen. Forscher haben herausgefunden, dass, wenn diese Netzwerke auf Milliarden von Wörtern trainiert werden, sie den Text verstehen und so Texte in menschlicher Qualität schreiben können. Die bekanntesten *LLMs* sind *ChatGPT*, *Palm*, *LLama*, *Mistral*, und viele weitere.

Ein Problem dieser Netzwerke ist, dass sie hohen Rechenaufwand aufweisen und meist sehr viel *VRAM* benötigen. Je nach Anzahl der Parameter, welche in großer Zahl die Qualität verbessern, wird mehr von beidem benötigt.

Durch *Quantisierung* (geringere Anzahl an Bits pro Zahl) ist es möglich, solche Netzwerke auf leistungsschwächeren Computern auszuführen. Hierbei wird jedoch Geschwindigkeit und Performance eingebüßt.

In unserem Fall wurde zu Beginn *LLama* (Meta AI, 2024) verwendet, um *LLama 7B* zu quantisieren und auf den uns zur Verfügung gestellten Rechner laufen zu lassen. Hierbei ist jedoch das Problem, dass *LLama* den Prompt evaluieren muss, bevor die Wahrscheinlichkeit des nächsten Wortes errechnet wird. Dies dauert pro Token, also ein einziges Satzzeichen, 200

Millisekunden. In unserem Fall dauert dieser Prozess je nach Bildbeschreibung zwischen 5-25 Sekunden. Des Weiteren wurde *LLama* hauptsächlich auf der CPU exekutiert, was wiederum den Prozess verlangsamt.

Am 27.09.23 wurde *Mistral 7B* (Mistral AI, 2024) veröffentlicht. Dieses Netzwerk zeigt bessere Performance wie *Llama-7b*. Des Weiteren ist es möglich, durch die Python Library *Hugging Face*, gewisse Layer des Netzwerkes auf die GPU zu verlagern und so den Prozess zu beschleunigen. Ein weiterer Vorteil gegenüber *Llama-7b* ist, dass keine Prompt Evaluierung benötigt wird.

5.2.5.3 InterStableLLM Pipeline

Um nun noch kreative Textbeschreibungen zu bekommen, wird anstelle der in *InterStableCloud Pipeline* (Absatz 5.2.4.2) verwendeten *Wordcloud* ein *Large Language Model* verwendet. Hierbei ist die Textbeschreibung für das *LLM* wie folgt:

```
You are a very creative artist.  
Think of something extraordinary.  
Think of something what you can add to this image prompt to make it  
magic and special  
Think in a way that inspires people and which makes them happy  
Do not describe the style of the new image mention that no areas  
should be filled and the style is black and white linestyle  
Be very creative for example to a house you could add a garden or to  
a car a trailer  
You can even draw a truck out of a car  
The description should be something which makes sense  
The image description is
```

Durch die Kombination aus verschiedenen Netzwerken ist es nun möglich, kreative Textbeschreibungen zu erhalten, jedoch zeichnet der *Stable Diffusion* Algorithmus nur dort, wo der Benutzer gezeichnet hat. Zeichnet der Besucher nur auf einer Hälfte des Papiers, so zeichnet auch der Algorithmus nur auf dieser Seite.

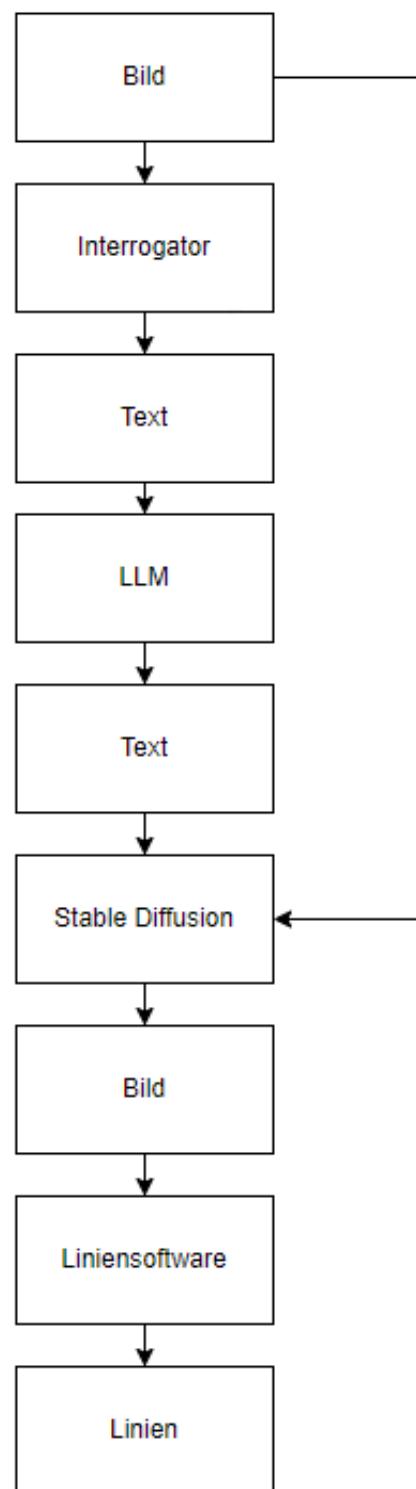


Abbildung 5.2.5.3 InterStableLLM Pipeline

5.2.6 Finale Umsetzung, InterStableLLMRLLine

5.2.6.1 R-Line

Die *InterStableLLM Pipeline* erfüllt bis auf das eher monotone Ergebnis, beschrieben in Absatz 5.2.5.3 alle Kriterien. *InterStableLLMRLLine* versucht die menschliche Kreativität nachzuahmen, indem er generative Künstliche Intelligenz mit zufällig in das Eingangsbild generierten Inhalten kombiniert. Die Innovation liegt in dieser Interpretation menschlicher Kreativität als genau dieser Kombination von zufälligen Impulsen in der menschlichen Wahrnehmung, abgebildet durch die Zufallsgeneratoren, und der Erkennung von Mustern, abgebildet durch KI.

5.2.6.2 InterStableLLMRLLine Pipeline

Die *InterStableLLMRLLine Pipeline* basiert auf dem gleichen Prinzip wie die *InterStableLLM Pipeline* beschrieben in Absatz 5.2.5.3 nur wird hierbei das Originalbild durch den *R-Line Algorithmus* beschrieben im Absatz 5.2.7.1 verändert. Zu sehen ist die Pipeline in der Abbildung 5.2.7.2. Sie kann noch durch *Hyperparameter Tuning* und *Prompt-Engineering* verbessert werden, jedoch liefert die derzeitige Version bereits angemessene Resultate.

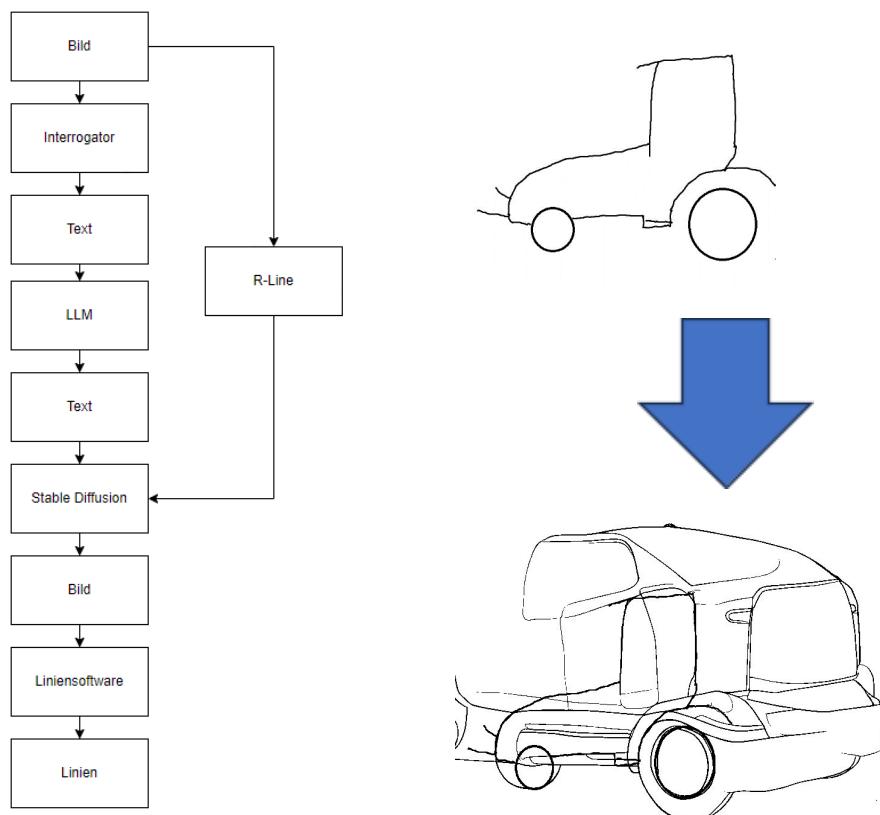


Abbildung 5.2.7.2 InterStableLLMRLLine Pipeline

5.4.3 Time / Memory Management

Um die Ausführzeit von neuronalen Netzwerken zu vermindern, werden die Rechenprozesse auf die GPU des Computers verlagert. Die neuronalen Netzwerke der Pipeline sind:

- der Interrogator
- das LLM (Mistral 7b)
- Stable Diffusion

Der Rechner, auf dem die Experimente ausgeführt wurden, hat eine Ryzen 5900 CPU, 32 GB Ram, eine RTX3070Ti mit 8 GB VRAM.

Je nach Bildgröße, in unserem Fall 512x768 Pixel, wird mehr RAM benötigt, da es mehr Rechenleistung erfordert. Stable Diffusion ist ein iterativer Prozess, und somit herrscht ein linearer Zusammenhang zwischen Zeit und Anzahl an Diffusionsprozessen. In der Regel sind 20 bis 50 Steps notwendig, um ein qualitatives Bild zu erzeugen, wobei eine geringere Diffusion Step Anzahl einer schlechteren Bildqualität entspricht.

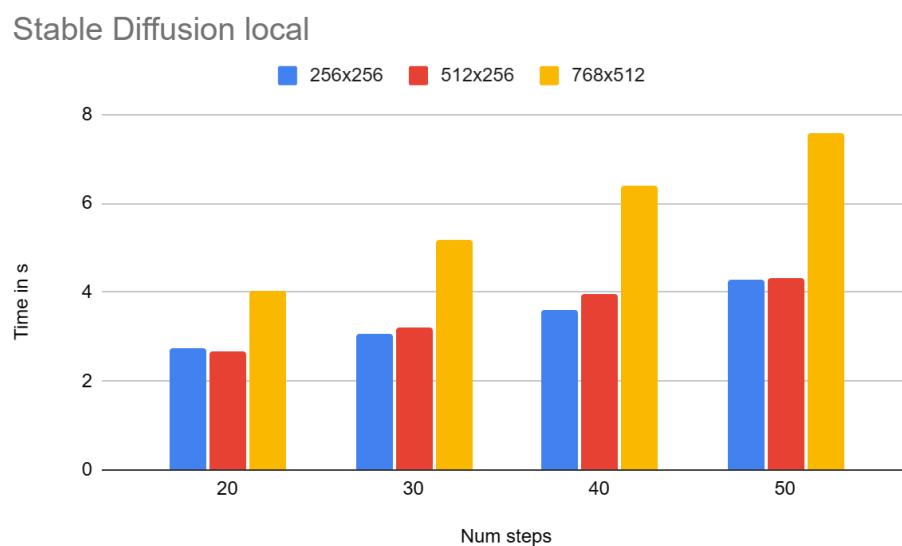


Abbildung 5.4.3.1 Stable Diffusion Time Management

Wie bei dem Stable Diffusion Algorithmus hängt auch die Zeit, die der Interrogator benötigt, von der Bildgröße ab. Da dieser Prozess vergleichsweise mit den anderen Algorithmen wenig Rechenleistung benötigt, ist es möglich, ihn auf der CPU laufen zu lassen, jedoch wird dadurch die Laufzeit verdoppelt.

Wie schon in Absatz 5.3.4.2 erwähnt, ist es möglich, einen Teil des LLMs auf die GPU zu verlagern. Auch ist es möglich durch verschiedene quantisierungs Methoden (Gewichte durch weniger Bits zu repräsentieren) die Inferenzdauer zu minimieren, jedoch wird hierbei die Qualität und Performance der Texte schlechter.

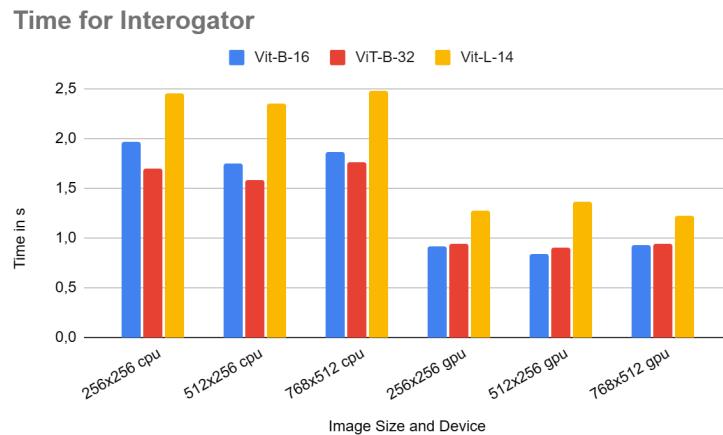


Abbildung 5.4.3.2 Interrogator Time Management

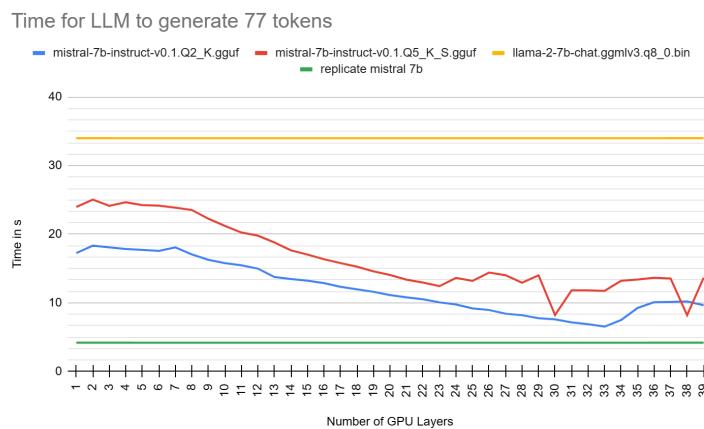


Abbildung 5.4.3.3 LLM Time Management

Konkrete Aufteilung auf einer RTX 3070 Ti:

- *Stable Diffusion*:
Da der Stable Diffusion Prozess der rechen aufwendigste ist, wird Stable Diffusion zur Gänze auf die GPU geladen.
- *Mistral 7b*:
Die Übrigen 5,3 GB V-Ra werden mit Mistral 7b belegt
Hierbei lassen sich 20 Layer auf der GPU rechnen, wenn maximal 77 neue Wörter generiert werden. Werden mehr neue Wörter generiert, so wird quadratisch mehr VRam benötigt und der Pipeline läuft der Speicher aus.
- *Interrogator*
Der Interrogator wird zur Gänze auf der CPU berechnet.

5.3 pdars - Paper detection and reconversion Software

5.3.1 Ablauf

Da das Papier nicht aus der Vogelperspektive fotografiert wird, muss das Papier erkannt und transformiert werden. Dies erfolgt in folgenden Schritten:

1. *Marker Detection*
2. *Entzerrung*
3. *Transformation*
4. *Paper Eckenerkennung*
5. *Transformation*
6. *Stifterkennung*



Abbildung 5.3.1.1 Original



Abbildung 5.3.1.2
Marker Detection



Abbildung 5.3.1.3
Entzerrung



Abbildung 5.3.1.4
Transformiert



Abbildung 5.3.1.5
Papier Eckenerkennung

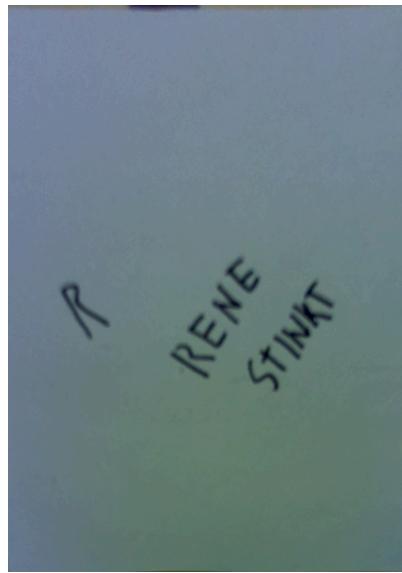


Abbildung 5.3.1.6
Transformiert

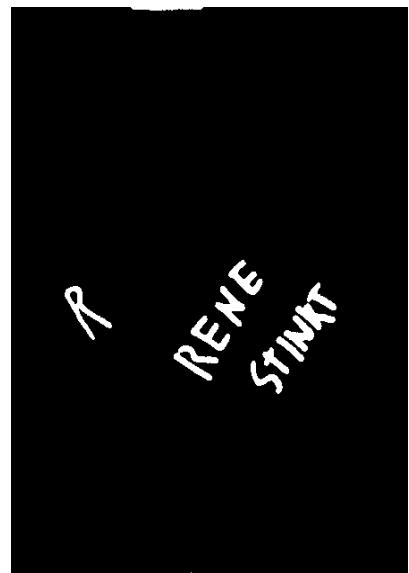


Abbildung 5.3.1.7
Stifterkennung

5.3.1.1 Marker Detection

Um die Marker zu erkennen, wird das Bild vom RGB-Format in das HSV-Format konvertiert. Dadurch ist es möglich, die Marker trotz Helligkeitsänderungen zu erkennen. Nun wird jeder Pixel überprüft, ob er zwischen den definierten Grenzen der einzelnen Channelns liegt. Liegt er zwischen diesen Grenzen, so kommt dieser Pixel in Frage als ein Marker Pixel. Diese Grenzen werden experimentell durch den MarkerWebserver (Absatz 5.3.2) ermittelt.

Im nächsten Schritt werden Gruppen an infrage kommenden Pixel extrahiert, indem Konturen erkannt werden. Kleinere Konturen werden entfernt, wenn deren Fläche kleiner als 2000 Pixel beträgt.

Um nun zu erkennen, welche Kontur der Form eines Rechtecks ähnelt, werden die Konturen durch Polygone approximiert und somit durch weniger Punkte repräsentiert. Nun wird ein Rechteck um die Kontur gelegt, welches den minimal Rechteckigen Flächeninhalt der Kontur repräsentiert. Zuletzt wird der Wert R ausgerechnet, welcher repräsentativ für ein Rechteck steht. Je höher dieser Wert, desto wahrscheinlicher ist es, dass ein Rechteck vorhanden ist. Dieser Wert setzt sich aus den Verhältnissen des Umfangs des minimalen Rechtecks, zum Umfang der Kontur, multipliziert mit dem Flächenverhältnis des minimalen Rechtecks zum Flächenverhältnis der Kontur. Die vier Konturen, deren V-Wert am höchsten ist, kommen als Marker infrage.

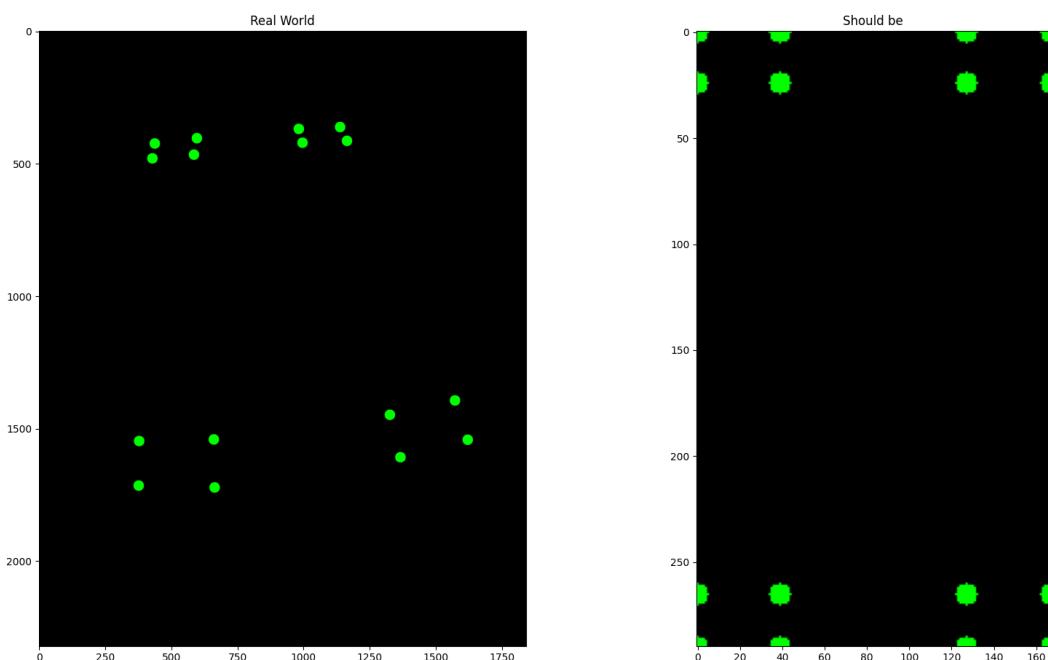
$$R = 1 - |1 - \text{boxarea} / \text{contourarea}| * |1 - \text{scope} / \text{contourScope}|$$

Um nun die äußereren Eckpunkte der Marker zu erkennen, wird die Euklidische Distanz zum Mittelpunkt berechnet und die Punkte mit der höchsten Distanz werden als Eckpunkte angenommen.

5.3.1.2 Entzerrung

Durch die Kameralinse wird das Bild verzerrt und somit werden Punkte auf einer Linie zu Punkten auf einem Kreisbogen. In Abbildung 5.3.1.2.1. zu sehen. Hierbei ist zu erkennen, dass die Punkte, welche im rechten Bild in einem Rechteck angeordnet sind, eine andere Anordnung aufweisen. Durch die Verzerrung wird die Software um bis zu 10 mm ungenau, was zu Abständen in der Zeichnung führt.

Um das Bild zu entzerren, werden die Marker Positionen im erkannten Bild mit dem Marker an ihrer Sollposition verglichen. Da das Bild aus einem gewissen Verdrehwinkel aufgenommen wird, müssen die Markerpunkte den Punkten zugeordnet werden. Hierbei wird, um den äußersten und innersten Punkt eines Markers zu erkennen, die Euklidische Distanz zum Papier Mittelpunkt benutzt. Um den Linken / Rechten Punkt eines Markers zu erkennen, wird der Seitenmittelpunkt des Papiers berechnet. Der Punkt mit der kürzesten Distanz zum Rechten / Linken Seitenmittelpunkt wird als Rechter / Linker Markerpunkt definiert. Mithilfe dieser Punkte und der OpenCV Library kann das Bild nun entzerrt werden.



5.3.1.2.1 Entzerrung Real World

5.3.1.3 Papier Eckenerkennung

Um nun das Papier zu erkennen, wird das transformierte Bild wieder im HSV-Format repräsentiert. Liegt ein Pixel zwischen definierten Grenzen, so wird er als mögliches Papier Pixel betrachtet. Diese Grenzen werden experimentell durch den MarkerWebservice (Absatz 5.3.2) ermittelt. Nun werden Konturen erkannt, um die möglichen Pixel zu gruppieren.

5.3.1.4 Stifterkennung

Die Stifterkennung erfolgt wie die Papiererkennung (Absatz 5.3.1.3) nur, wird hierbei der V-Wert des HSV Bildes als ausschlaggebendes Kriterium benutzt.

5.3.2 MarkerWebserver

Der im Absatz 5.3 beschriebene pdras Algorithmus benötigt zur vollständigen Funktionsweise kalibrierte Grenzen des Bildpunkt. Da sich diese Grenzen basierend auf den Lichtverhältnissen verändern können, wurde dem Auftraggeber eine Software zur Verfügung gestellt, welche es ermöglicht, diese Werte neu zu kalibrieren.

5.3.2.1 Starten des MarkerWebservers

Der MarkerWebserver wird gestartet durch das Ausführen der piCameraMarkerWebserver.py Datei welche sich nach korrekter Installation, beschrieben in Absatz 10.3.1, am Raspberry Pi befindet. Wie man Zugang zum Raspberry Pi erlangt, ist im Absatz 10.3.2 beschrieben.

5.3.2.2 Unabsichtliches Speichern von Werten

Wird der Apply Button unbeabsichtigt betätigt, so können die originalen Werte aus folgender Datei gelesen werden. DrAi/code/ai_pipeline/raspberrypi/original.json

5.3.2.3 Software Aufbau und funktionsweise

Die Software gliedert sich in 5 Teilbereiche

Wird ein Teilbereich geändert, so ändern sich alle Teilbereiche, welche unterhalb angeordnet sind. Da dies ein rechenaufwendiger Prozess ist, kann der Raspberry Pi das Python Script beenden, wenn zu viele Requests vorhanden sind. Ist dies der Fall, den Server neu starten, wie es in Absatz 5.3.2.1 beschrieben ist.

Die Werte einer Abteilung werden gespeichert, sobald der Knopf Apply gedrückt wird.

Teilbereiche

1. Kamerabild aufnehmen (Abbildung 5.3.2.3.1 Original Image)

Das Kamerabild wird durch Drücken des Knopfes Capture aufgenommen. Hierbei kann Kontrast, Helligkeit und Schärfe des Bildes durch die Schieberegler bzw. den Text Input verändert werden. Diese sollten so verändert werden, dass die Marker und das Papier sich in den unteren Teilbereichen leicht von der Umwelt unterscheiden lassen, dies muss nicht unbedingt bedeuten, dass das Bild eine gute Qualität für das menschliche Auge besitzt. Das aufgenommene Bild wird über dem Einstellbereich angezeigt. Dies kann mehrere Sekunden dauern, da der Raspberry Pi Zeit benötigt, um das Bild aufzunehmen. Falls sich das Bild nach 10 Sekunden nicht aktualisiert hat, ist zu empfehlen, den Capture Knopf nochmals zu betätigen. Aktualisiert sich das Bild immer noch nicht, ist es zu empfehlen, den Log auf Fehler zu kontrollieren bzw. zu kontrollieren, ob der Webserver noch läuft und die Kamera korrekt angeschlossen wurde.

2. Marker Detection

Durch Verändern der Werte S-Min / S-Max / H-Min / H-Max werden mehr oder weniger Pixel als Marker Pixel definiert. Hierbei ist darauf zu achten, dass der Min Wert den Max Wert nicht überschreitet. Das Linke Bild in diesem Abschnitt repräsentiert alle Pixel, die als Marker Pixel in Frage kommen in der Farbe weiß. Das zweite Bild von links repräsentiert den H-Anteil, gefolgt von dem dritten Bild mit dem S-Anteil. Im rechten Bild sollten die äußeren Markerecken mit einem Grünen Kreis gekennzeichnet sein. Ist dies nicht der Fall, so müssen die Parameter geändert werden.

Um das adjustieren der Werte zu vereinfachen, kann mit der Maus auf das Bild gezeigt werden und unterhalb des Bildes wird nun der Pixelwert des Pixels angezeigt.

3. Paper Detection

Der Papiererkennungs-Teilbereich ist gleich aufgebaut wie der Marker Detection Teilbereich.

4. Pen Detection

Pen Detection ist vom Prinzip gleich aufgebaut wie der Marker und Paper Detection Teilbereich, jedoch repräsentiert hierbei das Linke Bild die Schrift und somit das Resultat. Die restlichen 3 Bilder sind der H / S und der V-Anteil des Bildes.

5. Pen Size

Da unterschiedliche Stifte eingesetzt werden können, besteht im untersten Teilbereich die Möglichkeit, unterschiedliche Stiftbreiten einzustellen. Da der LES-Algorithmus (Absatz 5.3.2.2) mittels Pixel rechnet, wird die Stiftbreite in Pixel repräsentiert wobei 1 Pixel ungefähr 0.3mm repräsentiert. Somit hat ein 2 mm breiter Stift eine Siftbreite von $2 / 0.3 = 6.6$ Pixel.

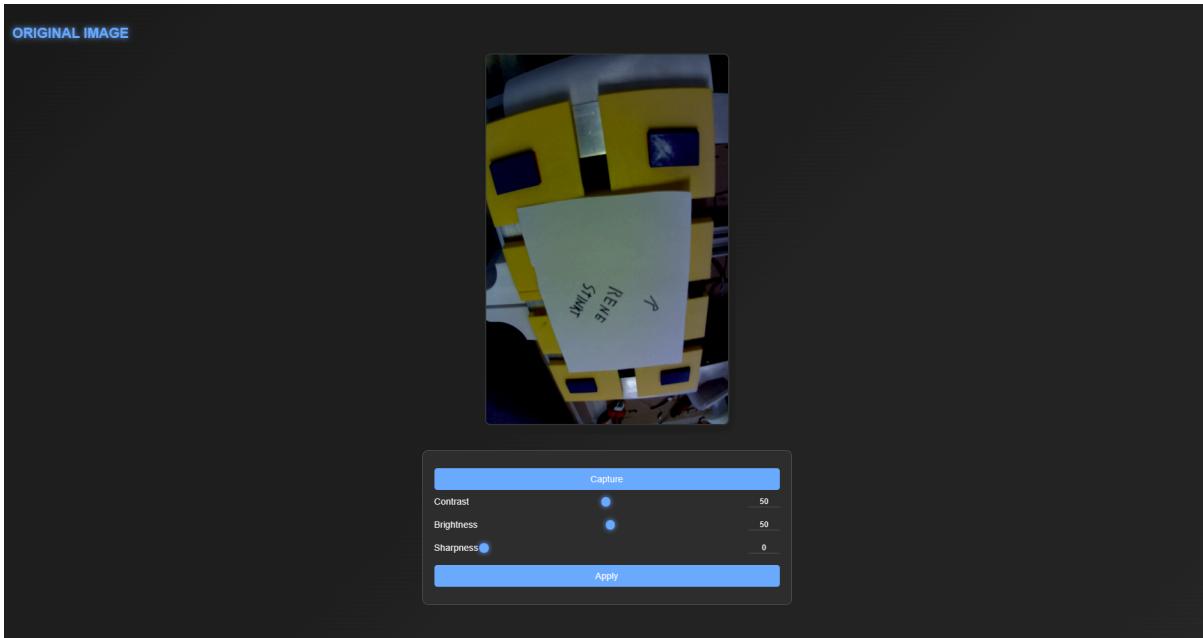


Abbildung 5.3.2.3.1 Original Image

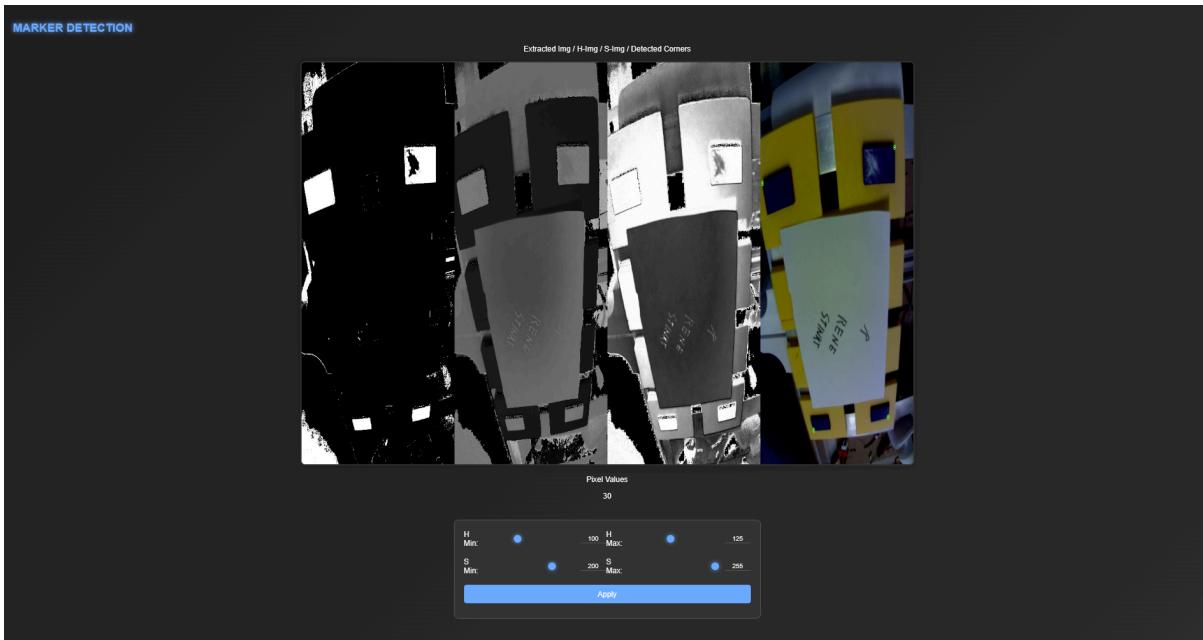


Abbildung 5.3.2.3.2 Marker Detection

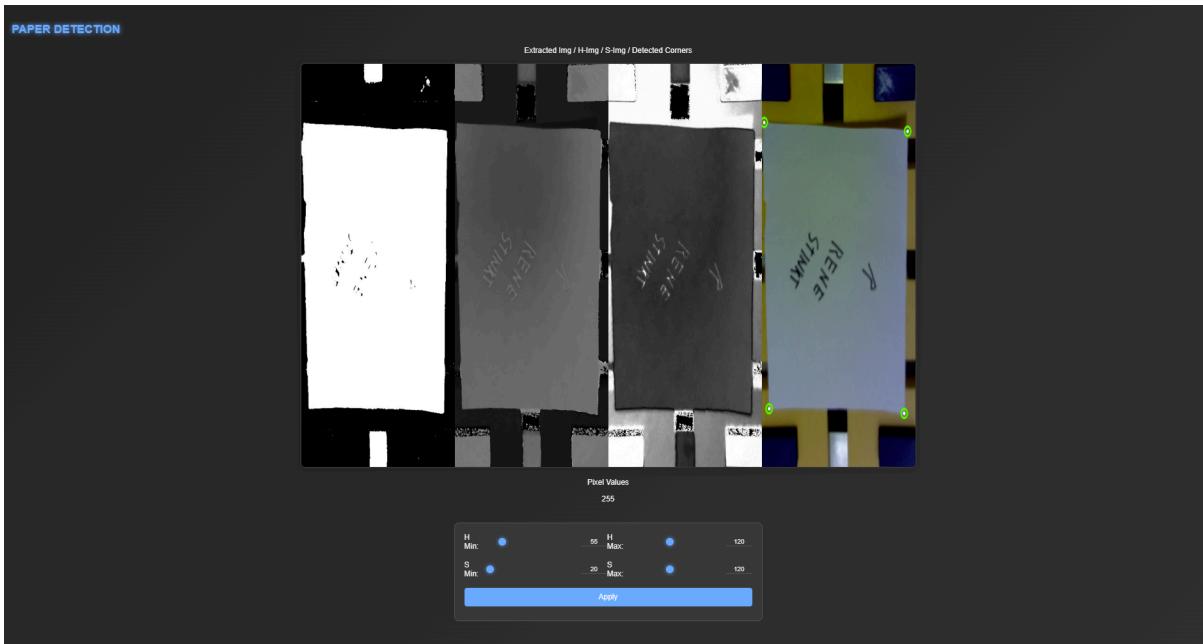


Abbildung 5.3.2.3.3 Paper Detection

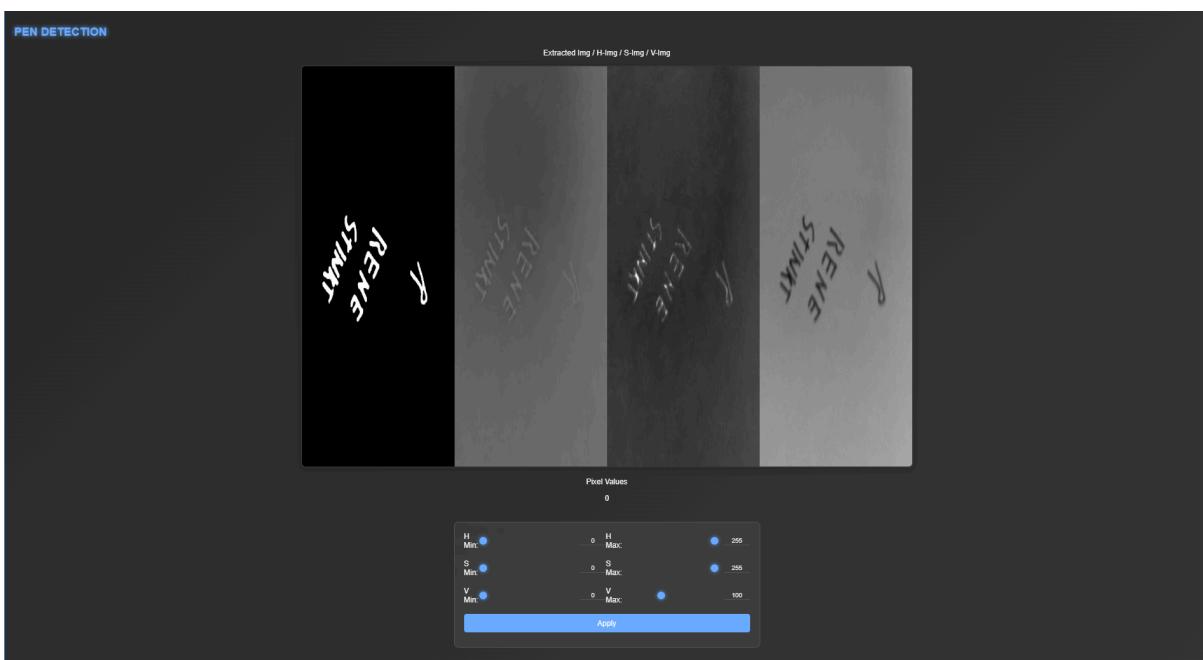


Abbildung 5.3.2.3.4 Pen Detection

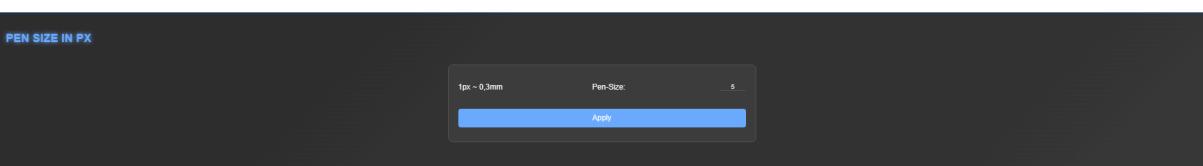


Abbildung 5.3.2.3.5 Pen Size

6 Versionierung und Aufbau der Software

6.1 Versionierung der AI

Die Künstliche Intelligenz, der Papier-Erkennungs- und Kanten Erkennungs-Algorithmus wurden vollständig in Python programmiert. Um die benötigten Libraries zu installieren, die ./requirements.txt Datei via pip ausführen. Dies installiert die benötigten Libraries in den benötigten Versionen. Anbei ist eine Liste mit den Versionen zu sehen.

Der Code wurde in der Python Version 3.9.13 geschrieben.

Als Large Language Model wurde TheBloke/Mistral-7B-v0.1-GGUF verwendet. Hierbei wurden die Weights über Huggingface heruntergeladen.

Stable Diffusion wurde in der Version 1.5 verwendet, die Weights wurden auch hier über Huggingface von runwayml/stable-diffusion-v1-5 heruntergeladen.

Für den Clip-Interrogator wurde ein Large Vision Transformer von Openai verwendet, abgekürzt ViT-L-14/openai.

Für den Algorithmus wurden weiters folgende Libraries mit folgender Version verwendet.

clip_interrogator	0.6.0
ctransformers	0.2.27
diffusers	0.16.0.dev0
Flask	3.0.2
langchain	0.1.11
matplotlib	3.4.2
numpy	1.22.2
opencv-python	4.7.0.72
Pillow	11.2.0
Requests	2.31.0
torch	2.0.0
tqdm	4.66.0

6.2 Aufbau der KI-Software

Die KI wurde vollständig in Python programmiert und ist unter folgendem Link zu finden.

DrAI/code/ai_pipeline/

Der nun zu findende Ordnerbaum sieht folgendermaßen aus.

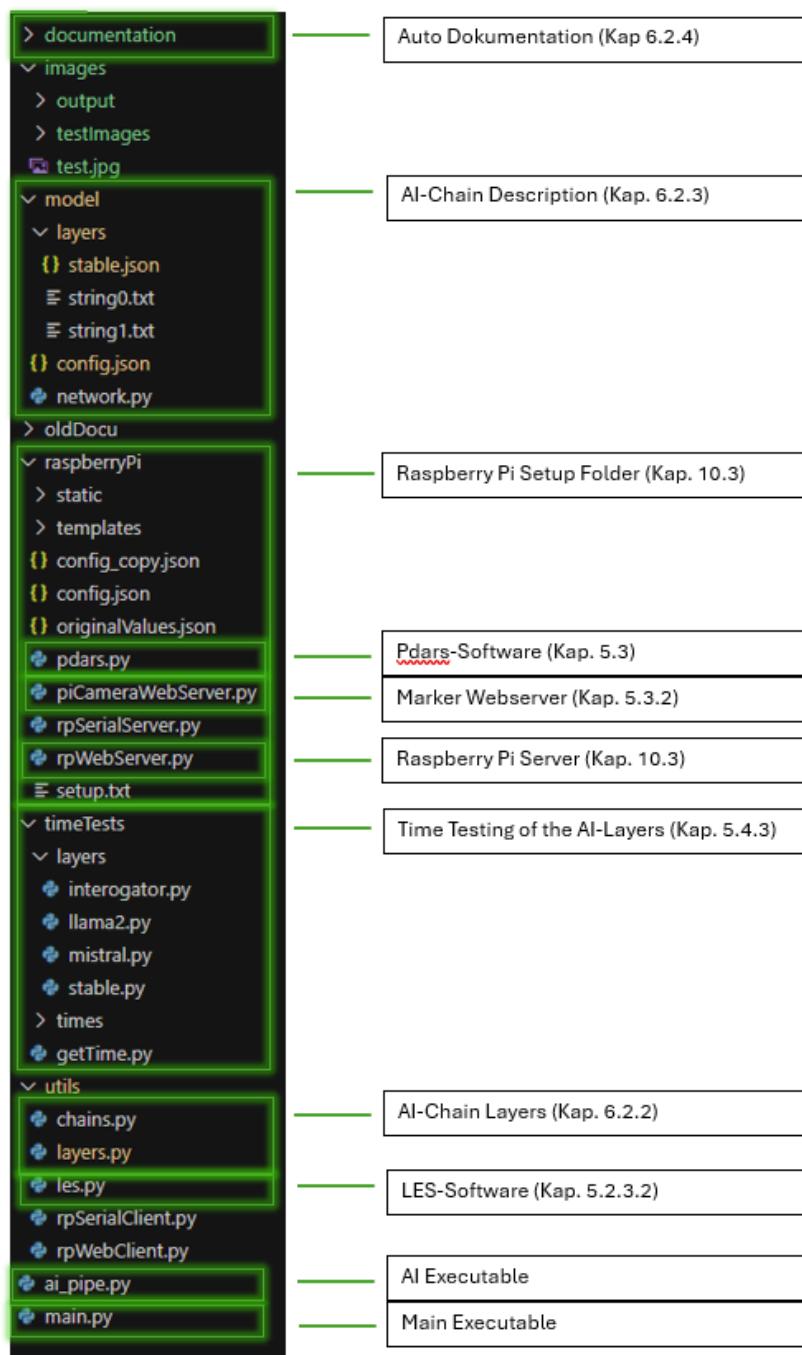


Abbildung 6.2.1 Ordnerstruktur

Der Server wird mithilfe der main.py file gestartet. Durch starten der ai_pipe.py Datei wird nur die KI gestartet und es kann getestet werden, ob diese richtig aufgesetzt wurde.

Die KI besteht aus diversen Deep Learning Algorithmen, welche bis zu 8GB Vram konsumieren können. Das Laden dieser Algorithmen ist sehr zeitaufwendig und oft ändert sich der Input nur wenig vom vorherigen Input.

Ein Vergleich wäre, ChatGPT jedes Mal neu in den RAM Speicher zu laden, wenn ein Benutzer eine Anfrage sendet. Hierbei ist es viel effizienter, die Netzwerk Gewichte einmal in den Speicher zu laden und die Eingabe variabel zu machen.

Während der KI-Entwicklung ändert sich jedoch nicht nur der Input, sondern auch die Struktur der Deep Learning Algorithmen ändert sich. Im Absatz 5 sind Beispiele für diverse Strukturen gelistet. Um nun unterschiedliche Netzwerk-Kombinationen zeiteffizient zu testen, wurde die AI-Chain Library programmiert.

6.2.1 KI-Implementierung

Um nun diverse Kombinationen effizient zu testen, wurden die folgende Layer in der Datei ./utils/layers.py implementiert. (Absatz 6.2.2 Layer):

- *Interrogator*
- *RandomLineDrawer*
- *SimpleString*
- *Merger*
- *LLM*
- *StableDiffusion*
- *Overlapper*
- *Model*

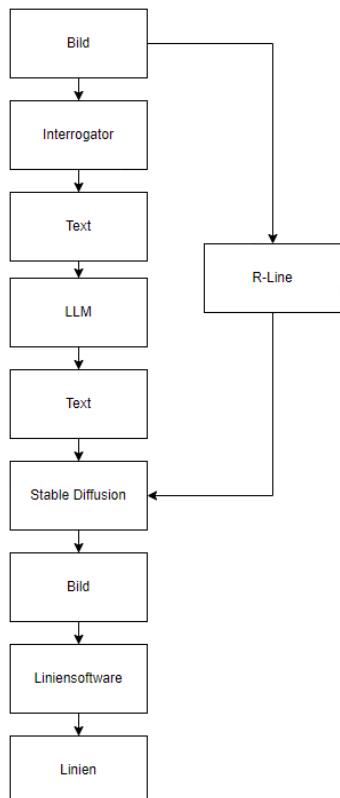
Die Netzwerkgewichte werden direkt nach dem Start des Algorithmuses geladen.

Um nun die Layer zu kombinieren, wird ein rekursiver Algorithmus verwendet, wobei jeder Layer den vorherigen Layer kennt und diesen zu Beginn des Selbstaufrufes aufruft.

Wird ein Layer aufgerufen, lädt dieser seine Einstellungen aus einer Config Datei, jedoch müssen die Netzwerk Gewichte nicht nochmals geladen werden.

Durch die Rekursivität ist es möglich, die verschiedenen Netzwerkgewichte nur ein Mal zu laden und später die Struktur und diverse Konfigurationsparameter während der Laufzeit zu ändern.

Im Ordner ./model/network ist nun die Kombination der diversen Layer beschrieben. Eine Beispiel Beschreibung für den StableDiffRLLine Algorithmus ist in Abbildung 6.2.1.1 StableDiffRLLine Implementation zu sehen. Wie Layer kombiniert werden, wird in Absatz 6.2.3 Netzwerkkombination beschrieben.



```
1 from utils.layers import *
2 import json
3
4     You, 1 second ago * Uncommitted changes
5 settings = json.load( open( "./model/config.json", "r" ) )
6
7 input_image = settings[ "original-image-path" ]
8
9
10 inp = Interrogator( "inter1", input_image )
11 rand = RandomLineDrawer( "rand0", inp, input_image, settings[ "original-image-path" ] )
12
13 simp0 = SimpleString( "simp0", settings[ "string0" ] )
14 simp1 = SimpleString( "simp1", settings[ "string1" ] )
15
16 merg = Merger( "mergl1", [ simp0, simp1 ] )
17
18 llm = LLM( "llm1", prompt = merg, prevElement = inp, prevElementKey = "<INPUT>" )
19
20 diff = StableDiffusion( "diff1", settings[ "stable-config-path" ], llm, rand )
21
22 over = Overlapper( "over0",
23                     diff,
24                     input_image,
25                     settings[ "original-image-path" ],
26                     mergedOutputFolder = settings[ "merge-output-folder" ],
27                     outputImageName = settings[ "output-image-name" ] )
28
29 model = Model( over )
```

6.2.1.1 StableDiffRLine Code Implementation und Ablaufdiagramm

6.2.2 Layers

Die Layer wurden in der Datei ./utils/layers.py programmiert.

6.2.2.1 Interrogator

Die Aufgabe des Interrogators ist es, aus einem Bild eine Textbeschreibung zu generieren. Dies erfolgt mithilfe des Pretrained Clip Algorithmus der Firma OpenAi. Hierbei wird die Library clip_interrogator verwendet, um das Laden der Netzwerk Gewichte zu vereinfachen. Hierbei wird ein Bild eingelesen und die Textbeschreibung an den nächsten Layer weitergegeben.

Abbildung 6.2.2.1.1 Interrogator Aufruf

Übergabeparameter:

name : str (wichtig für die Dokumentation, beschrieben in Absatz 6.2.4)

imagePath : str = Dateipfad zum Interpretierenden Bild

prevElement : [ChainElement | None] = Vorheriger Layer

Output Parameter:

str = Textliche Bildbeschreibung

Der Interrogator kann auch ein Element aufrufen, welches das Bild bearbeitet, bevor er es interpretiert. So wäre es beispielsweise möglich, einen RandomLineDrawer Layer vor dem Interrogator zu schalten und so eine andere Textbeschreibung zu erhalten.

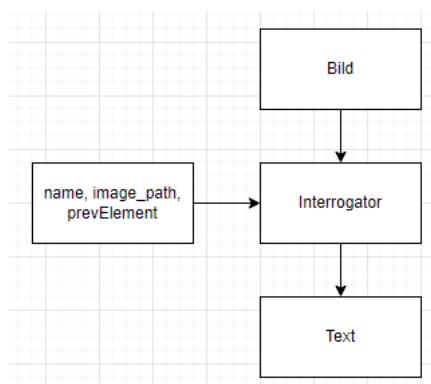


Abbildung 6.2.2.1.2 Interrogator Schema

6.2.2.2 RandomLineDrawer

Um dem Algorithmus “Kreativität” zu verleihen, werden dem Bild zufällige Linien hinzugefügt, welche für den Stable Diffusion Algorithmus als Inspiration dienen. Der RandomLineDrawer gibt nichts an den nächsten Layer weiter. Er liest ein Bild ein, bearbeitet dies und speichert das Überarbeitete in dem output Bild.

```

1 import utils.layers
2
3
4 interrogator = utils.layers.RandomLineDrawer( name = "rand0",
5                                         prevElement = None,
6                                         input_image = "./test.png",
7                                         output_image = "./test2.png",
8                                         num_random_lines = 40,
9                                         line_length = 40,
10                                        pen_size = 4 )
11
12
13

```

6.2.2.2.1 Initialisierung RandomLineDrawer

Übergabeparameter:

name : str = (wichtig für die Dokumentation, beschrieben in Absatz 6.2.4)
prevElement : [ChainElement | None] = Vorheriger Layer
input_image : str = Dateipfad des Bild werden
output_image : str = Dateipfad des Ausgabebildes
num_random_lines : int = Anzahl and zufälligen Linien
line_length : int = Pixellänge der zufälligen Linien
pen_size : int = Pixelbreite der zufälligen Linien

Die Übergabeparameter

- *num_random_lines*
- *line_length* = l
- *pen_size* = b

sind variabel. Je dicker die zufälligen Linien sind, desto weniger wird Wert auf das originale Bild gelegt. Die Stiftbreite des Users ist 8 Pixel.



6.2.2.2.1 Beispiel einer R-Line

6.2.2.3 SimpleString

Um Textbeschreibungen und Textkombinationen zu ermöglichen, wurde der SimpleString Layer implementiert. Hierbei wird eine Textdatei angelegt, aus der die Textbeschreibung gelesen wird. Wird der Layer aufgerufen, so wird aus der Textdatei der Inhalt gelesen und dem nächsten Layer weitergegeben. Dies ermöglicht, während der Laufzeit den Text zu ändern.

```
1 import utils.layers  
2  
3  
4 simp0 = utils.layers.SimpleString( name = "string0",  
5 | | | | | | | | | | | stringPath = "./model/layers/string0.txt" )  
6  
7
```

Abbildung 6.2.2.3.1 SimpleString Aufruf

Übergabeparameter:

name : str = (wichtig für die Dokumentation, beschrieben in Absatz 6.2.4)
stringPath : str = Dateipfad zur zu lesenden Datei

Output parameter:

str = Textdateiinhalt

6.2.2.4 Merger

Mehrere Textbeschreibungen können kombiniert werden, indem ein Merger-Layer eingebaut wird. Dieser kombiniert den Text-Output seiner vorherigen Layer durch ein definiertes Symbol.

```
1 import utils.layers
2
3 simp1 = utils.layers.SimpleString( name = "string0", stringPath = "./string1.txt" )
4 simp2 = utils.layers.SimpleString( name = "string0", stringPath = "./string1.txt" )
5 simp3 = utils.layers.SimpleString( name = "string0", stringPath = "./string1.txt" )
6
7
8 merger = utils.layers.Merger( name = "merger",
9 | | | | | elements = [ simp1, simp2, simp3 ],
10 | | | | mergingSymbol = " | ")
```

Abbildung 6.2.2.4.1 Merger Aufruf

Übergabeparameter

name : str = (Wichtig für die Dokumentation)

elements : list[ChainElements] = Liste aus Layers

mergingSymbol : str = Kombinationszeichen der Strings

Output parameter:

str = Kombinierte eingabe Strings

6.2.2.5 LLM

Dieser Layer ist die Implementierung eines Large Language Models, welcher aus einer Textbeschreibung weiteren Text generiert und diesen an das nächste Chain Element weitergibt.

```

1 import utils.layers
2
3 promptString = utils.layers.SimpleString( name = "simp0",
4                                         stringPath = "./prompt.txt" )
5
6 prevElement = utils.layers.SimpleString( name = "simp1",
7                                         stringPath = "./insertment.txt" )
8
9
10 llm = utils.layers.LLM( name = "llm0",
11                         prompt = promptString,
12                         prevElement = prevElement,
13                         prevElementKey = "{input}" )
14

```

Abbildung 6.2.2.5.1 LLM Aufruf

Übergabeparameter:

name : str = (wichtig für die Dokumentation)

prompt : ChainElement = Layerelement welches den Prompt liefert

prevElement : ChainElement = vorheriges LayerElement

prevElementkey : str = Zu ersetzendes zeichen im Prompt durch prevElement

Output Parameter:

str = Text generiert durch das LLM

Das Large Language Model generiert Text zu einer Textbeschreibung. Hierbei wird der Prompt aus einer Textdatei gelesen. Das LLM liest den Prompt und ersetzt im Prompt den prevElementKey durch den Output des prevElement. So ist es möglich, die Bildbeschreibung dem Prompt einzufügen, ohne einen Merger Layer.

Um verschiedene neuronale Netzwerke zu testen, wird in der Config.json Datei der HuggingFace Link des neuronalen Netzwerkes angehängt. Wird die LangchainLibrary zum Ausführen des Netzwerkes verwendet, so ist der llm-model Parameter "LLama2" ansonsten wird die Hugging Face Library zum Ausführen des Netzwerkes verwendet. Wird die Hugging Face Library verwendet, so ist es möglich, durch den llm-layers Parameter Teiles des Neuronalen Netzwerkes auf die Grafikkarte zu verlagern.

```

"llm-path": "./weights/llm/llama-2-7b-chat.ggmlv3.q8_0.bin",
"llm-model": "mistral",
"llm-layers": 15,

```

Abbildung 6.2.2.5.2 LLM Config

6.2.2.6 StableDiffusion

Stable Diffusion generiert aus einem Eingabebild und einer Textbeschreibung ein Ausgabebild (Absatz 5.2.2.1). Hierbei wird ein Bild eingelesen, die Textbeschreibung durch den vorherigen Layer generiert und das neu generiertes Bild abgespeichert.

```

1 import utils.layers
2
3 promptString = utils.layers.SimpleString( name = "simp0",
4                                         stringPath = "./prompt.txt" )
5
6
7 diff = utils.layers.StableDiffusion( name = "stable0",
8                                     parameterPath = "./model/layers/stable.json",
9                                     prevElement = promptString,
10                                    prevImageElement = None )
11
12

```

Abbildung 6.2.2.6.1 Stable Diffusion Aufruf

Übergabeparameter:

name : str = (Wichtig für die Dokumentation)

parameterPath : str = Dateipfad zur Config Datei (Abbildung 6.2.2.6.2)

prevElement : chainElement = Layer welcher den Prompt liefert

prevImageElement : [chainElement | None] = Layer welcher das Bild vorher bearbeitet

6.2.2.6.1 Stable Diffusion Config Datei

In der Config Datei des Stable Diffusion Layers ist der Pfad zum Eingabebild durch den Key “imagePath” definiert. Cache ist der Dateipfad zur Ausgabe des Stable Diffusion Algorithmuses. Guidance_scale ist ein Integer Wert zwischen 0-30, wobei Stable Diffusion bei 30 den Prompt so genau wie möglich umzusetzen und 0 der Prompt ignoriert wird. Durch den Strength Parameter, einen Float zwischen 0 und 1, kann mehr oder weniger hinzugefügt oder verändert werden. Wobei 0 nichts verändert und 1 die höchst mögliche Veränderung darstellt.

```

1 {
2   "imagePath": "./images/test.png",
3   "cache": "./images/test.png",
4   "guidance_scale": 7,
5   "strength": 0.6
6 }

```

Abbildung 6.2.2.6.2 Config File

6.2.2.7 Overlapper

Die Les Software (5.2.2.3 Line Extraction Software) benötigt eine Überlappung des originalen Bildes mit dem Generierten Bild, um daraus die Linien zu extrahieren. Dies wird durch den Overlapper Layer ermöglicht.

```

1  import utils.layers
2
3  over = utils.layers.Overlapper( name = "over0",
4                                image_1 = "./test1.png",
5                                image_2 = "./test2.png",
6                                mergedOutputFolder = "./",
7                                outputImageName = "output.png" )
8
9

```

Abbildung 6.2.2.7.1 Overlapper Implementierung

Übergabeparameter:

- name* : str = (Wichtig für die Dokumentation)
- image_1* : str = Dateipfad zum 1. Bild
- image_2* : str = Dateipfad zum 2. Bild
- mergeOutputFolder* : str = Dateipfad zum Ausgabebild
- outputImageName* : str = Name des Ausgangsbildes

6.2.2.8 Model

Die Layer Chain wird in der network.py File beschrieben. Um nun zu wissen, welcher Layer der letzte Layer der Chain ist, muss dieser definiert werden. Dies geschieht, indem dieser als Übergabeparameter in die Model Klasse gegeben wird. Die Model Klasse wird unter den Globalen Variablen erkannt und kann gestartet werden.

```

22  over = Overlapper( "over0",
23                      diff,
24                      input_image,
25                      settings[ "original-image-path" ],
26                      mergedOutputFolder = settings[ "merge-output-folder" ],
27                      outputImageName = settings[ "output-image-name" ] )
28
29  model = Model( over )      You, 3 months ago • clean and sync

```

Abbildung 6.2.2.8.1 Model Aufruf

Übergabeparameter:

- prevElement* : *chainElement* = Letzter Layer der Chain.

6.2.3 Netzwerk Kombination

Um verschiedene Netzwerkkombinationen zu programmieren, wird in der ./model/network.py Datei die Netzwerkstruktur definiert, nach ähnlichem Schema wie die Programmierung eines Neuronalen Netzwerkes mit Hilfe der Tensorflow / Pytorch library.

In der config.json Datei sind die Pfade zu den Netzwerkgewichten definiert und auch ist der Dokumentations Path definiert.

In der Hauptdatei (ai_pipe.py) wird nun die Chain initialisiert. Sobald diese Datei ausgeführt wird, werden die Gewichte geladen.

Durch die Parameter document, kann definiert werden, ob mitdokumentiert werden soll oder nicht.

Durch den Parameter verbose wird der Konsolenoutput definiert.

Ruft man die Funktion chain.run() auf, so startet die Chain und geht die Layer definiert in der network.py Datei rekursiv ab.

Beispiel:

Ein Bild soll durch den Interrogator interpretiert werden. Die Interpretation soll mit einem String kombiniert werden und daraus soll Stable Diffusion ein Bild generieren.

Ablaufdiagramm:

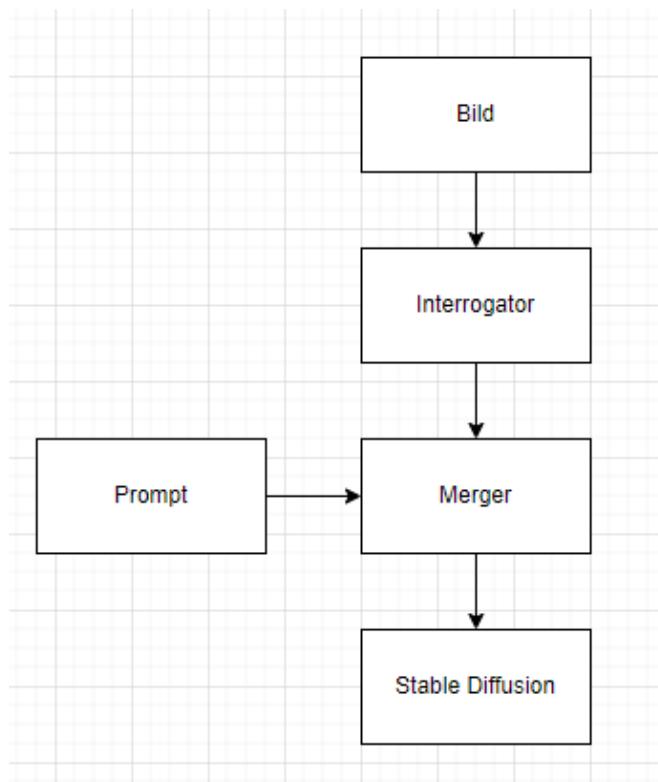


Abbildung 6.2.3.1 Ablaufdiagramm

Programmcode in der network.py Datei:

```

1  from utils.layers import *
2
3  ### Defining the Interrogator
4  inter = Interrogator( name = "inter0",
5  | | | | | imagePath = "./testImage.png"
6  | | | | | prevElement = None )
7
8  #Reading the prompt
9  prompt = SimpleString( name = "prompt0",
10 | | | | | stringPath = "./model/string.txt" )
11
12 #Combining the prompt with the interrogatorinput
13 merger = Merger( name = "merg0",
14 | | | | | elements = [ inter, prompt ],
15 | | | | | mergingSymbol = "|" )
16
17
18 #Generating Image
19 diff = StableDiffusion( name = "diff0", parameterPath = "./model/stableConfig.json",
20 | | | | | prevElement = merger,
21 | | | | | prevImageElement = None )
22
23 #Defining the model
24 model = Model( diff )

```

Abbildung 6.2.3.2 Network.py

Programmcode in der ai_pipe.py Datei:

```

1  from utils.chains import *
2
3  if __name__ == "__main__":
4      c = Chain( document = True, verbose = True )
5
6      print( "Start" )
7      while True:
8          c.run()
9          inp = input()
10         You... 2 weeks ago * finished version of th

```

Abbildung 6.2.3.3 ai_pipe.py

Programmcode in der Config.json Datei:

```

1  {
2      "stable-config-path": "./model/layers/stable.json",
3      "stable-path": "runwayml/stable-diffusion-v1-5",
4      "docu-path": "./documentation/"
5  }

```

Abbildung 6.2.3.4 Stable Config

Ordnerstruktur:

```

| ai_pipe.py
| testImg.png
\---cache
\---model
    | network.py
    | string.txt
    | stableConfig.json
    | config.json

```

6.2.4 Auto-Dokumentation

Um die KI besser analysieren zu können, wird automatisch der Input und Output jedes Layers mitdokumentiert, sowie die Netzwerkstruktur und die Konfigurationsparameter.

Wird ein Algorithmus ausgeführt, so wird automatisch die Netzwerkstruktur in den documentations Ordner kopiert. Der Dokumentationsordner kann in der ./model/config.json Datei mit dem Makro “./docu-path” geändert werden. Sobald die KI started, wird ein neuer Ordner im Documentations Ordner angelegt mit dem Zeitpunkt des Starts.

Die Ordnerstruktur sieht wie folgt aus:

```
\---documentation
  +---2024_01_13_10_43_27
    | +---history
    |   | diff1_history.txt
    |   | diff1_history_input_image.jpg
    |   | diff1_history_output_image.jpg
    |   | inter1_history.txt
    |   | inter1_history_input_image.jpg
    |   | llm1_history.txt
    |   | merg1_history.txt
    |   | over0_history.txt
    |   | over0_history_output_image.jpg
    |
    | \---model
    |   | config.json
    |   | network.py
    |   |
    |   \---layers
    |     stable.json
    |     string0.txt
    |     string1.txt
    |
\---2024_01_13_10_45_33
  +---history
    |   | llm1_history.txt
    |   | merg1_history.txt
```

```
|   | over0_history.txt  
|   | over0_history_output_image.jpg  
|   | rand0_history.txt  
|   | rand0_history_input_image.jpg  
|   | rand0_history_output_image.jpg  
|   | simp0_history.txt  
|   | simp1_history.txt  
  
|  
\---model  
| config.json  
| network.py  
|  
\---layers  
    stable.json  
    string0.txt  
    string1.txt
```

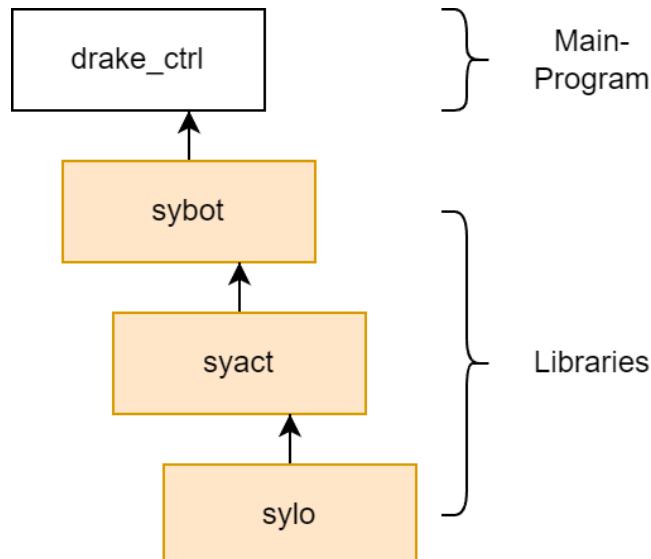
Hierbei wurde beispielsweise der erste Algorithmus 2024 am 13. Jänner um 10:45:33 ausgeführt.
Der zweite wurde 2024 am 13. Jänner um 10:43:27 ausgeführt.

Im history Unterordner ist nun der Input und Output eines jeden Layers zu sehen oder zu lesen.
Um eine ordentliche Dokumentation zu erhalten, ist es von oberster Priorität, die Layernamen
nur einmal zu verwenden, ansonsten überschreiben sich die Dateien. Die Dateien im History
ordner sind nach folgendem Schema benannt. Layername_history_in/output_typ. Durch diese
Auto-Dokumentation kann einfach überprüft werden, ob der ausgewählte Layer das gewünschte
Ziel erreicht.

Der ./model Unterordner wird in den Dokumentationsordner kopiert.

6.3 Aufbau der Steuerungssoftware

Die Steuerungssoftware ist in ein Hauptprogramm und drei *Libraries* aufgeteilt. Dabei basiert das jeweils eine Programm auf dem Nächsten in folgendem Aufbau:

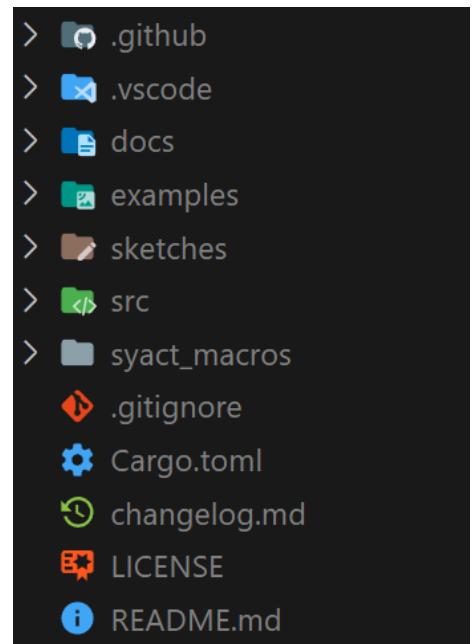


6.3 Rust Software Aufbau

6.3.1 Aufbau eines Rust-Projekts

Rust wird von einem zentralen Package-Manager namens *Cargo* verwaltet, der als Source-Control-Tool *Git* verwendet. Besondere Merkmale sind folgende:

- *Cargo.toml* - Hier sind alle Metadaten über das Projekt, wie auch seine *Dependencies*
- *README.md* - Eine klassische README Datei die einen schnellen Überblick über das Projekt verschafft
- *src* - In diesem Ordner befindet sich der allgemeine Code des Projekts
- *examples* - In diesem Ordner befinden sich *Examples*, welche in der Form von ausführbaren Binaries realisiert werden
- *bin* - Hier nicht vertreten, da es sich um eine *Library handelt*. Beinhaltet normalerweise die ausführbaren Programme eines Projekts



6.3.1 Rust Ordnerstruktur

6.3.2 sylo-Library

Die *sylo-Library* reguliert die Steuerung der logischen Signale auf der untersten Hardwareebene. Sie standardisiert digitale Ein- und Ausgänge und hilft beim Debugging verschiedener Signale.

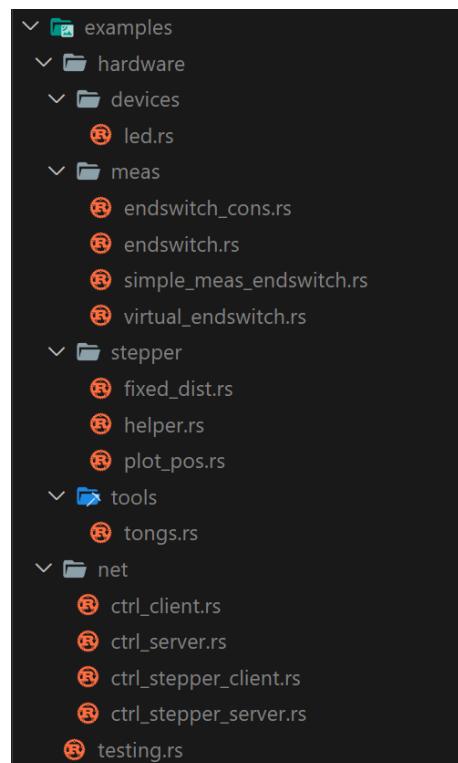
6.3.3 syact-Library

Die *syact-Library* ist mit Abstand der größte Teil der Steuerungssoftware, denn sie ist für die korrekte asynchrone Steuerung von Stepper-Motoren und anderen Komponenten wie zum Beispiel Endlagenschalter verantwortlich. Außerdem beinhaltet diese Library alle notwendigen Examples, die als Tests für verschiedene Komponenten dienen. Wichtige Tests sind zum Beispiel

- *hardware-meas-endswitch* - Zeigt an, ob ein Endlagenschalter an dem gegebenen Pin ausgelöst wird
- *hardware-stepper-fixed_dist* - Bewegt den gegebenen Stepper Motor eine fixe Distanz mit variabler Geschwindigkeit

6.3.4 sybot-Library

Die *sybot-Library* definiert das in [Abschnitt 3.1](#) angeführte R-D-S-System und reguliert das Zusammenspiel der Komponenten beim Roboter.



6.3.3 Syact Ordnerstruktur



sylo



syact

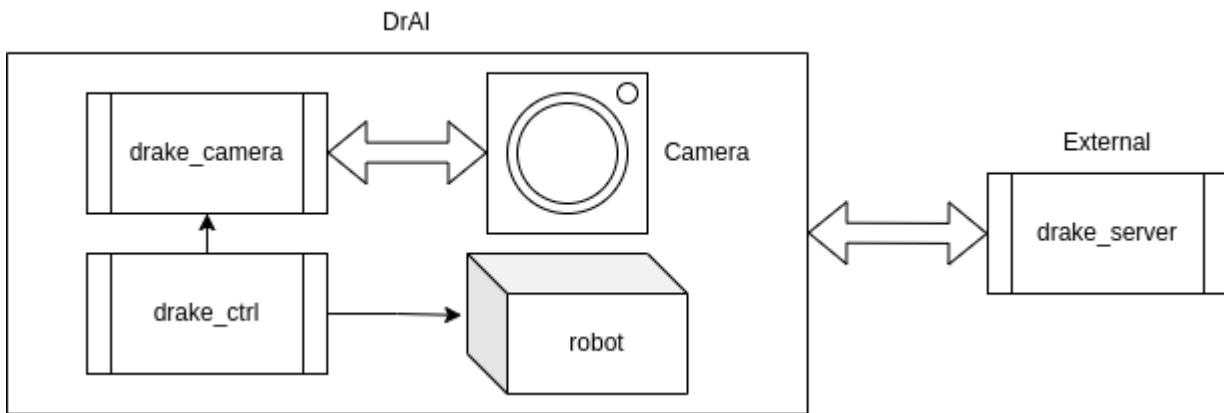


sybot

7 Kommunikation der Schnittstellen

7.1 Das IOT des Projekts

Das IOT besteht hauptsächlich aus zwei Raspberry Pis, einer für die Kamera im Arm und einer für die Robotersteuerung, und einem externen Rechner mit leistungsfähiger Grafikkarte.



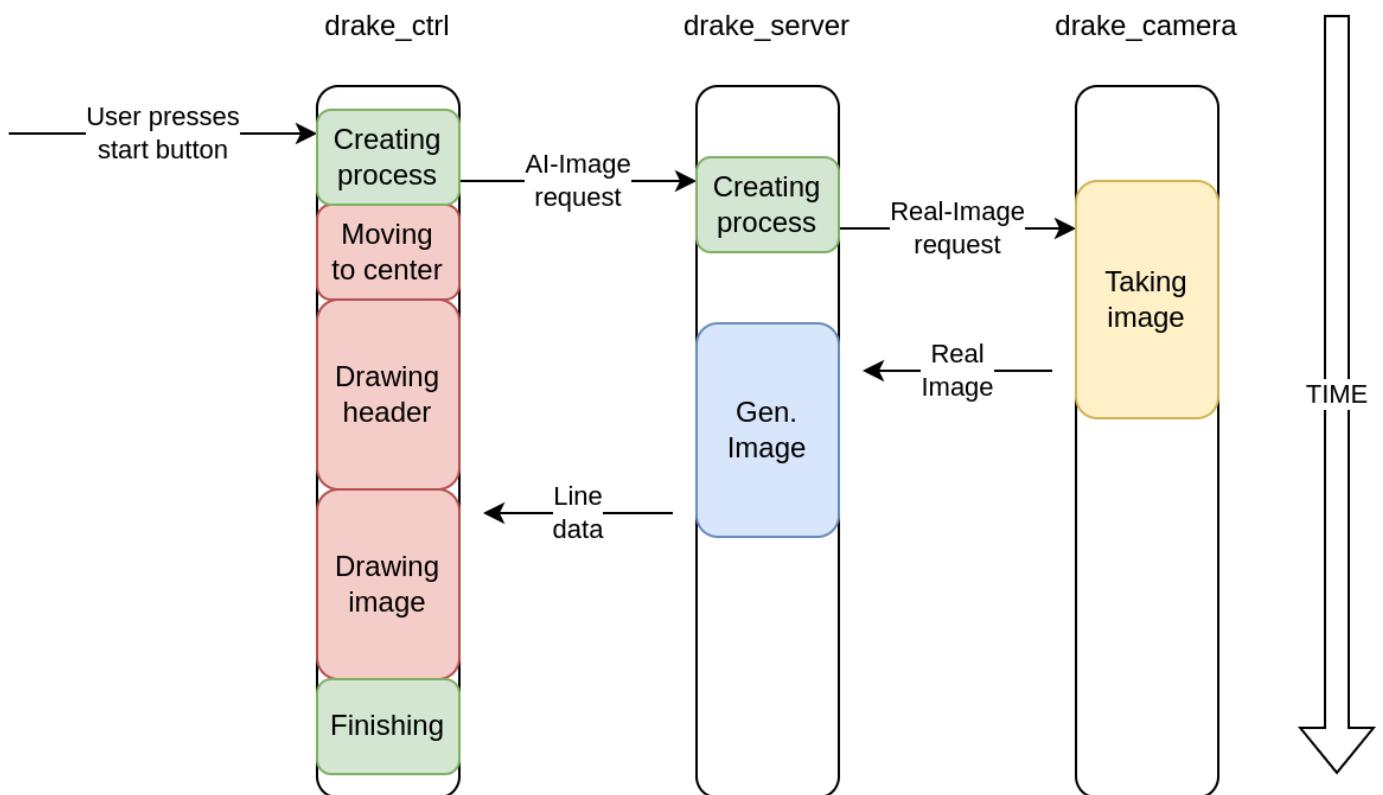
8.1 Iot Ablaufdiagramm

- *drake_camera*: Raspberry Pi mit integrierter Kamera
- *drake_ctrl*: Raspberry Pi zur Steuerung des Roboters
- *drake_server*: Externer Rechner zum Betreiben der KI

Die Geräte müssen sich alle im selben Netzwerk befinden, weshalb ein virtuelles Netzwerk, gehostet von *ZeroTier*, verwendet wurde. Sobald die Geräte über eine Internetverbindung verfügen, wird automatisch eine direkte Verbindung zwischen ihnen hergestellt.

7.2 Kommunikationskette

Der Ablauf wird vom Benutzer durch die Betätigung des Start-Knopfes gestartet, was sofort einen HTTP-Request der Steuerung an den Server auslöst. Die Steuerung fordert ein neues KI-Bild, beziehungsweise die Linien zum Zeichnen des Bildes an. Der Server fordert ein neues Bild von der Kamera an und generiert die KI-Version davon. Die daraus generierten Linien werden zurück an die Steuerungssoftware übertragen.



7.2 Kommunikationskette

8 Betriebsanleitung

Eine aktuelle Version der Tutorials und Anleitungen befindet sich online auf dem, am Anfang angeführten, *Github-Repository*.

8.1 Installation der Software

Die Software muss auf allen drei Geräten des Netzwerkes installiert werden, um korrekt zu funktionieren. Dafür wird zuerst das Projekt mithilfe von *Git* im *Home*-Ordner des Benutzers heruntergeladen:

```
cd ~  
git clone "https://github.com/SamuelNoesslboeck/DrAI.git"
```

und die Setup-Datei ausgeführt werden:

```
cd DrAI  
sh "code/scripts/setup.sh"
```

Je nach Gerät muss ein zusätzlicher Command ausgeführt werden:

Kamera

```
sudo sh "code/scripts/setup/drai_camera.sh"
```

Steuerung

```
sudo sh "code/scripts/setup/drai_ctrl.sh"
```

Server

```
sudo sh "code/scripts/setup/drai_server.sh"
```

Damit ist die Software installiert und startet automatisch als Systemprozess, sobald das Gerät neu gestartet wird. Statt die Software gleich nach der Installation manuell zu starten, wird es empfohlen, dies über einen Reboot zu tun:

```
sudo reboot
```

8.2 Kalibrierung und Tests

Eine aktuelle Beschreibung aller Scripts und ihrer Handhabung befindet sich ebenfalls auf dem Github-Repository, sowie eine ausführliche Testroutine zur Gewährleistung der Funktionalität von Achsen und Zeichentisch. (documentation/testing.md)

8.3 Konfiguration

8.3.1 Allgemeine Konfiguration

Jedes der drei Programme auf den Geräten im IOT bezieht wichtige Daten und Konfigurationen aus einer Script-Datei “env.sh” bzw. “env.sh” im Verzeichnis “code/scripts”.

```
# General
export DRAI_DIR="~/DrAI"
export DRAI_LOG_DIR="/logs"

# Networking
export DRAI_CAMERA_PORT=40324
export DRAI_SERVER_PORT=40325

export DRAI_CAMERA_USER_FILE="/key/drai-camera/username.key"
export DRAI_CAMERA_PW_FILE="/key/drai-camera/password.key"
export DRAI_CTRL_USER_FILE="/key/drai-ctrl/username.key"
export DRAI_CTRL_PW_FILE="/key/drai-ctrl/password.key"
export DRAI_SERVER_USER_FILE="/key/drai-server/username.key"
export DRAI_SERVER_PW_FILE="/key/drai-server/password.key"

# Controls
## Stepper
### X-Axis
export DRAI_X_AXIS_STEP_PIN=12
export DRAI_X_AXIS_DIR_PIN=21
export DRAI_X_SWITCH_POS_PIN=5
export DRAI_X_SWITCH_NEG_PIN=0
```

Ausschnitt aus “env.sh” ohne Beschreibungen

Es wird nicht empfohlen, Werte in dieser Datei zu ändern, jedoch ist es theoretisch möglich, sollte zum Beispiel ein Pin des Raspberry Pis der Steuerung beschädigt werden. Genauere Beschreibungen der Variablen befinden sich in der Datei in der Form von Kommentaren.

8.4 Stiftwechsel

Um den Stift zu wechseln, zuerst das Plexiglas im Vorderbereich entfernen. Hierzu die beiden Passstifte herausziehen, zu sehen in Abbildung 11.2.2.1 Plexiglasstifte.

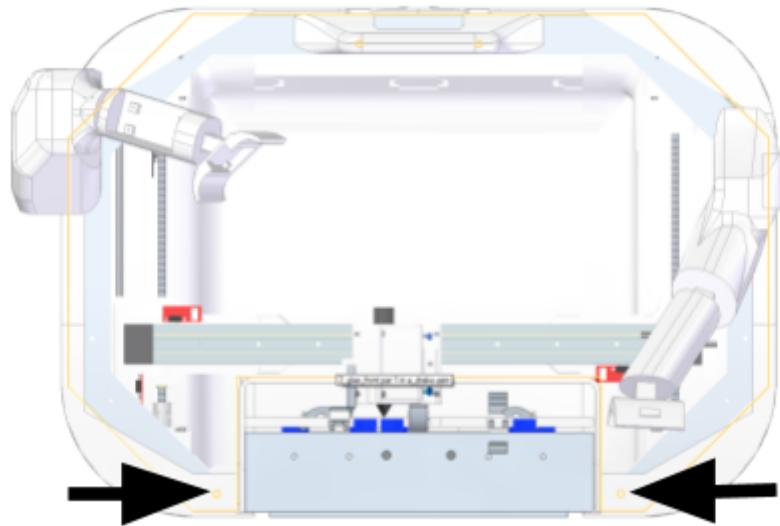


Abbildung 8.4.1 Plexiglasstifte

Das Plexiglas kann mit Hilfe des Griffes zu sehen in Abbildung 11.2.2.2 Plexiglasgriff nach oben gezogen werden. Nun können die Schrauben, die den Stift halten, zu sehen in Abbildung 4.5.1 in Blau, aufgeschraubt werden. Um das Glas wieder zu montieren, sind die Schritte in umgekehrter Weise auszuführen.

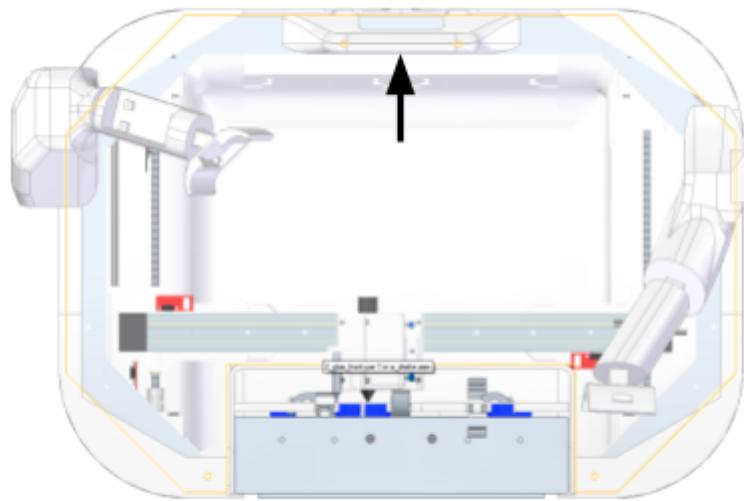


Abbildung 8.4.2 Plexiglasgriff

8.5 Aufsetzen der Elektronik

Die gesamte Steuerungselektronik des Projekts befindet sich extern in einem Plexiglas-Koffer zur besseren Darstellung der Abläufe. Beim Aufsetzen mit dem Roboter müssen die in folgenden Abbildung geschilderten Stecker verbunden werden:

- **Power supply**
 - (PSC): Power supply for the camera
- **Stepper motors**
 - (X): The X-Axis stepper motor
 - (Y): The Y-Axis stepper motor
 - (Z1): The left Z-Axis stepper motor (viewed from the front)
 - (Z2): The right Z-Axis stepper motor (viewed from the front)
- **Endswitches**
 - (X+): The positive X-Axis endswitch
 - (X-): The negative X-Axis endswitch
 - (Y+): The positive Y-Axis endswitch
 - (Y-): The negative Y-Axis endswitch
 - (Z+): The positive Z-Axis endswitch
 - (Z-): The negative Z-Axis endswitch
- **Extras**
 - (BL): Bed-Layer touch attached to the tool
 - (SC): The servo-controller board of the drawing table
 - (UT): The user terminal
- **Optional (might not be included)**
 - (H1): First servo of the head
 - (H2): Second servo of the head

Es wird empfohlen, wie in Abschnitt 8.2 beschrieben, nun alle Tests und Kalibrierungsmethoden durchzuführen.

9 Elektronik

Die Elektronik des Projekts gleicht der eines 3D-Druckers sehr stark. Die Hauptunterschiede liegen in der benötigten Rechenleistung, der Kamera und dem Zeichentisch.

9.1 Spannungsversorgung

Die Spannungsversorgung lässt sich in zwei einzelne Stromkreise unterteilen: Den Steuerstromkreis und den Arbeitsstromkreis.

Der Steuerstromkreis versorgt die Schrittmotoren-Treiber und die Servomotoren des Tisches, während der Steuerstromkreis die Raspberries und die Sensorik betreibt. Dabei verfügen beide Kreise über ein eigenes Netzteil.

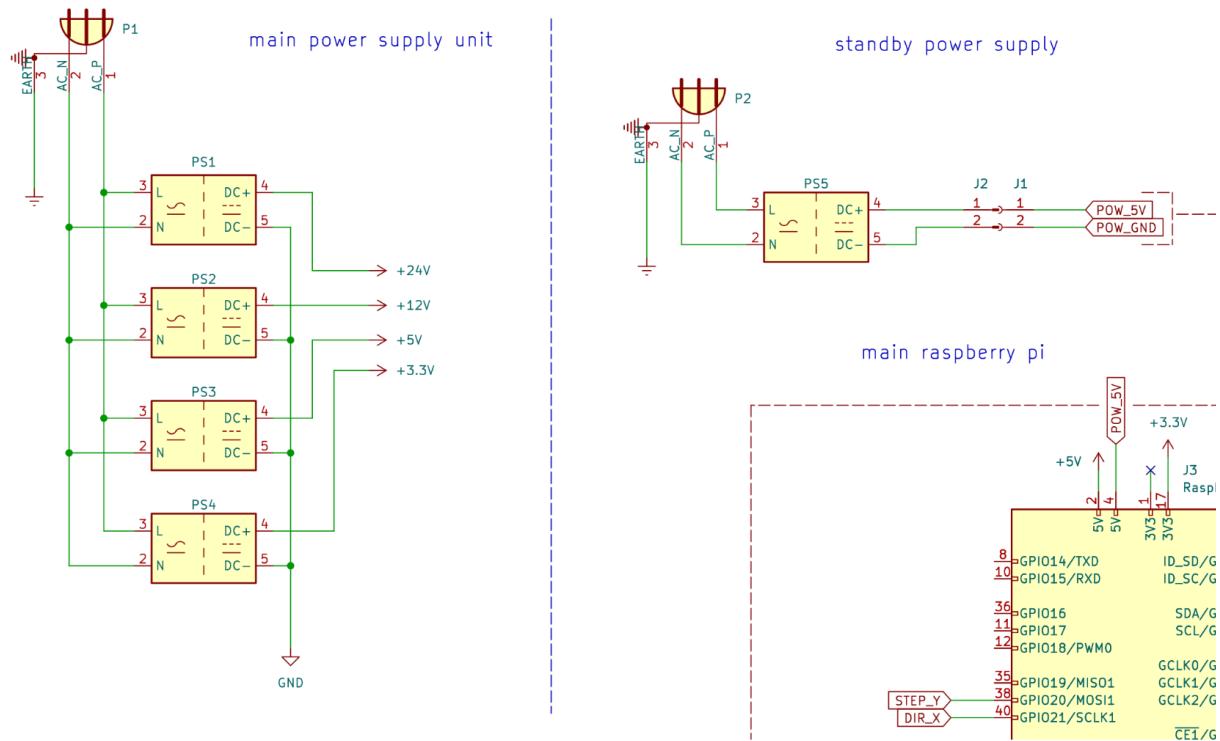


Abbildung 9.1 Spannungsversorgung

Der Steuerkreis verwendet fünf Volt als Versorgungsspannung, die dann im Raspberry Pi zu 3.3 Volt umgewandelt wird. Mit diesen 3.3 Volt wird dann die weitere Sensorik wie zum Beispiel die Endschalter der Motoren betrieben.

9.2 Achsen

Jede Achse ist mit einem, die Z-Achse sogar mit zwei, Schrittmotoren mit entsprechenden Treibern ausgestattet.

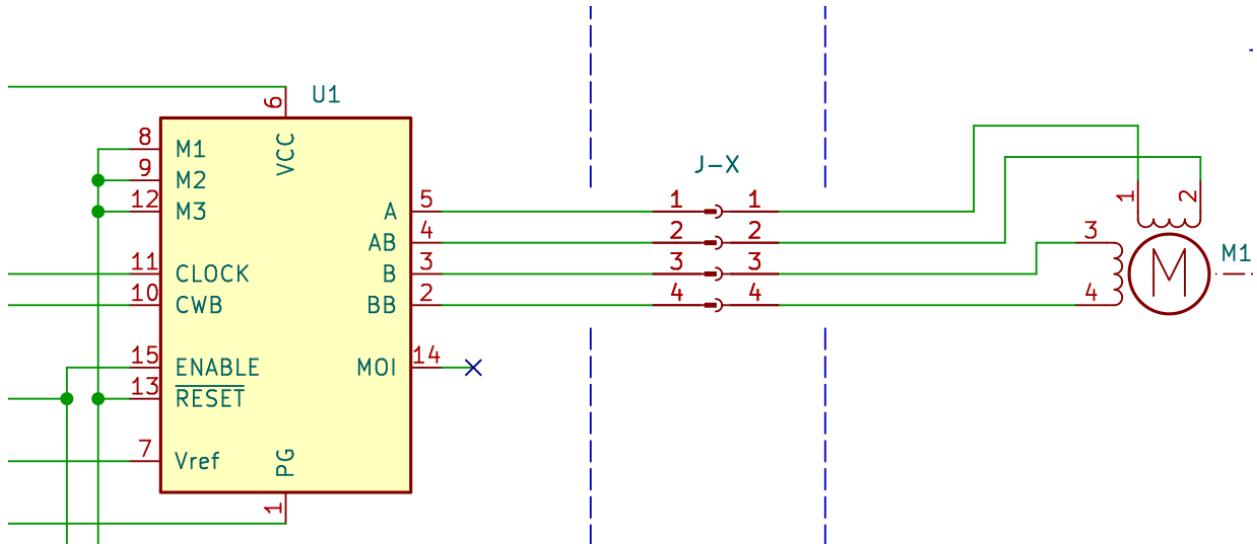


Abbildung 9.2.1 Achsen

Außerdem verfügt jede Achse über zwei Endschalter, einen am positiven Ende und einen am negativen.

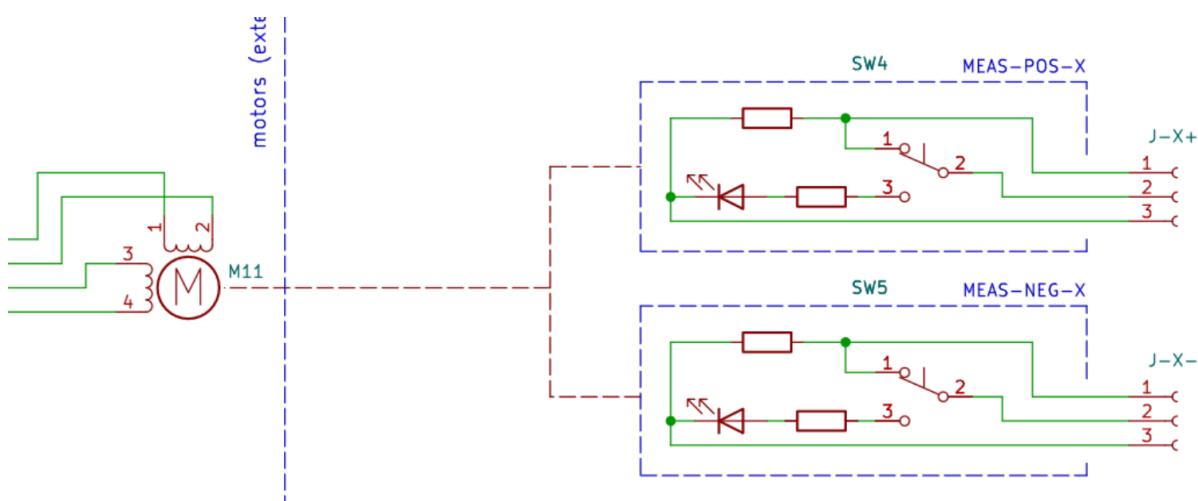


Abbildung 9.2.2 Endschalter

9.3 Zeichentisch

Der Zeichentisch des Projekts wird von insgesamt acht Micro-Servomotoren, die sich unabhängig voneinander schließen und öffnen, betrieben. Um ein effizientes Schließen und Öffnen mit einem qualitativen PWM-Signal zu gewährleisten, wird ein Servo-Treiber-Board verwendet. Statt acht GPIO Pins beim Raspberry werden somit nur zwei für einen I²C-Bus benötigt.

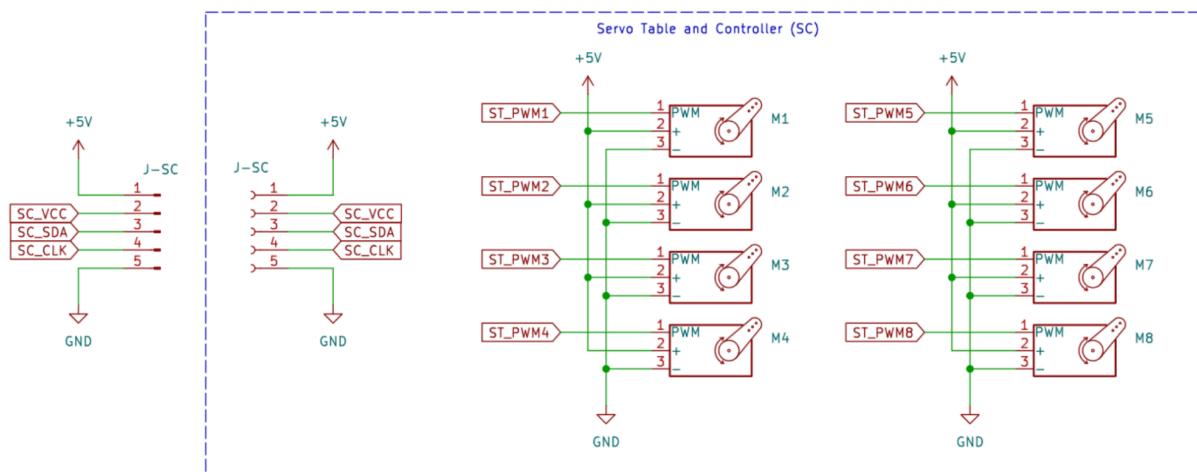
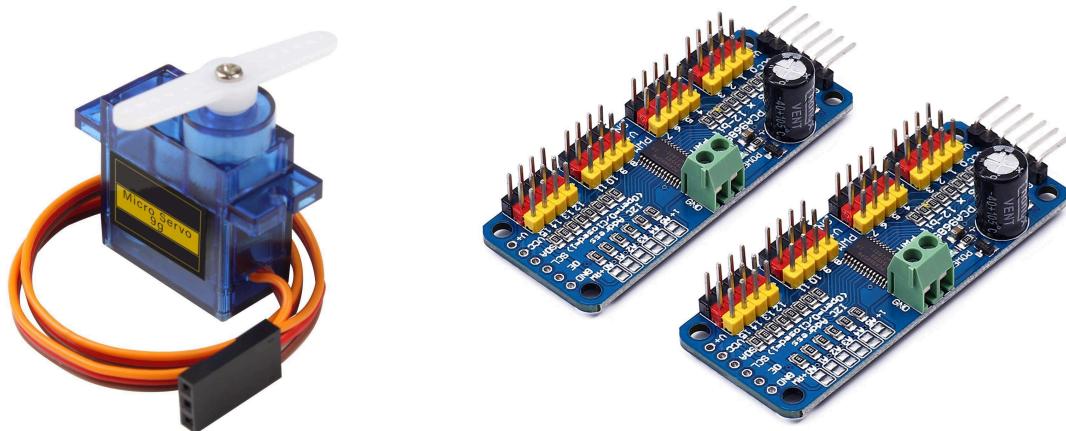


Abbildung 9.3 Servosteuerung

10 Allgemeines

10.1 Häufige Fehler

10.1.1 Kommunikation mit der Kamera schlägt fehl

Der Grund dieses Fehlers ist, dass der Kamerawebserver nicht im WLAN gefunden werden kann. Durch Putty kann überprüft werden, ob der Raspberry Pi im Wlan zu finden ist. Ist dieser nicht zu finden, so ist der Raspberry Pi nicht mit dem Wlan verbunden. Ist er jedoch aufzufinden, so liegt der Fehler auf der Serverseite.

10.1.2 Die Serververbindung fehlt

Kontrollieren, ob der Server eine aktive Verbindung zum Internet besitzt. Um zu kontrollieren ob der Server im Netzwerk auffindbar ist, kann er über die Console und den Windows Kommand ping angepingt werden und es wird kontrolliert ob es möglich ist, mit dem Server zu kommunizieren. Ist eine aufrechte Verbindung vorhanden, ist zu kontrollieren, ob die eingestellte IP-Adresse des Servers stimmt.

10.1.3 Fehler bei der Bildaufnahme

Kontrollieren, ob das Kamera-Kabel korrekt angeschlossen ist. Ist dies der Fall, so kann der Marker Webserver gestartet werden. Wird ein neues Bild aufgenommen durch das Drücken des Capture Buttons, so liegt ein Fehler im Code vor. Verändert sich das Bild jedoch nicht, so ist zu kontrollieren, ob die Kamera aktiviert und richtig angeschlossen wurde.

10.1.4 Servos nicht erkannt

Wurden nicht alle Servo Motoren erkannt, so startet der Prozess nicht. Hierbei kontrollieren, ob alle Kabeln der Servos an den Raspberry Pi angeschlossen sind. Bewegt sich ein Servo nicht, so kann dieser einfach ausgetauscht und neu angeschlossen werden. Hierbei auf die richtige Farbkombination der Kabeln achten. Wie die Servos verkabelt wurden, ist im Abschnitt Elektronik beschrieben.

10.1.5 Error mit der *pdards* Software

Um die genaue Ursache des Fehlers zu erkunden, den MarkerWebserver beschrieben in Absatz 5.3.2 starten. Je nachdem welcher Teilbereich unerkannt bleibt, die Werte in diesem Teilbereich ändern.

10.1.6 Error in der AI-Pipeline

Am Server kontrollieren, ob ein Fehler aufgeworfen wird. Liegt ein Fehler beim Laden eines Neuronalen Netzwerkes, so die Einstellungen in den config.json Dateien in dem Ordner model ändern. Je nachdem welcher Layer einen Fehler aufwirft, die zugehörige Datei ändern. Weitere Informationen zum einstellen der AI-Chain sind zu finden unter Absatz 6.2.3.

10.1.7 Marker-Webserver funktioniert nicht

Wird der Marker Webserver (Absatz 5.3.2) gestartet, aber das Bild aktualisiert sich nicht durch drücken des Capture Knopfes, so kann es entweder an Fehlern liegen die in Absatz 11.2.1.3 besprochen wurden, oder der Grund ist, dass die rpWebserver.py Datei dem Autostart hinzugefügt wurde. Ist dies der Fall, so erhält der rpWebserver automatisch die Kamerarechte und der Webserver hat keinen Zugriff mehr auf die Kamera. Um diesen Fehler zu beheben, muss die rpWebserver.py Datei aus dem Autostart entfernt werden.

10.1.8 Weights von Neuronalen Netzwerken fehlen

Den PC am Internet anschließen und den Algorithmus neu starten. Dies müsste die Weights automatisch herunterladen und in den richtigen Pfaden speichern. Da diese Dateien mehrere Gigabyte groß sind, kann dies eine Weile dauern.

Sind die Weights online nicht mehr verfügbar, so existiert ein Backupordner mit den Weights. Ist dieser auch nicht mehr verfügbar, so muss die Software umprogrammiert werden, bzw andere Weights gedownloadet werden. Einstellen der Gewichtspfade ist zu sehen in Abschnitt 6.2.3 bzw Abschnitt 6.2.2.5.

10.1.9 NSFW-Content erkannt

Zeichnet der Besucher etwas Unangemessenes, so erkennt dies Stable Diffusion automatisch und zeichnet nichts hinzu. Auch wird in die Log-Datei geschrieben, dass dies erkannt wurde.

10.1.10 Prozess startet nicht

Startet der Prozess aus unbekanntem Grund nicht, so kann man sich in den Raspberry Pi einloggen und sich die Log-Datei herunterladen. In dieser Datei wird der Grund beschrieben, warum die Maschine den Zeichenprozess nicht startet.

Abbildungsverzeichnis

2.1.1 Erste Konzeptskizze	12
2.1.4.1 Github-Website	13
2.1.5.1 Zeitplan	14
2.4.1 Kostendiagramm	15
2.1.3.1 Skizze Octagon	18
2.9.1 White Picture	20
3.2.1 Steppermotorkurve	21
3.5.1 Ablaufdiagramm	23
4.1.1 Rahmen Grundgerüst	24
4.1.2 Befestigung X-Achse	24
4.1.3 Alu-Achteck	25
4.1.4 Gesamtrahmen	25
4.2.1 X,Y-Achse	26
4.3.1 Rechte Z-Achse	27
4.4.1 Befestigungstisch	28
4.5.1 Stifthalter	28
4.7.1 Gesamtkonstruktion	29
5.1.5.1 KI-Resultate	31
5.2.1 GAN-Architektur	32
5.2.2 GAN-Trainingsdaten	32
5.2.2.2 Stable Diffusion Architecture	34
5.2.3.2.1 Line Extraction Software	36
5.2.3.3 InterStablePipe Pipeline	37
5.2.4.1 Word Cloud Training	38
5.2.4.2.1 Guidance Scale	40
5.2.4.2.2 InterStableCloud Pipeline	41
5.2.5.3 InterStableLLM Pipeline	44
5.2.7.2 InterStableLLMRLLine Pipeline	45
5.4.3.1 Stable Diffusion Time Management	46
5.4.3.2 Interrogator Time Management	47
5.4.3.3 LLM Time Management	47
5.3.1.1 Original	48
5.3.1.2 Marker Detection	48
5.3.1.3 Entzerrung	48
5.3.1.4 Transformiert	48
5.3.1.5 Papier Eckenerkennung	49
5.3.1.6 Transformiert	49
5.3.1.7 Stifterkennung	49
5.3.1.2.1 Entzerrung Real World	50
5.3.2.3.1 Original Image	54

5.3.2.3.2 Marker Detection	54
5.3.2.3.3 Paper Detection	55
5.3.2.3.4 Pen Detection	55
5.3.2.3.5 Pen Size	55
6.2.1 Ordnerstruktur	57
6.2.1.1 StableDiffRLine Code Implementation und Ablaufdiagramm	60
6.2.2.1.1 Interrogator Aufruf	61
6.2.2.1.2 Interrogator Schema	61
6.2.2.2.1 Initialisierung RandomLineDrawer	62
6.2.2.2.1 Beispiel einer R-Line	62
6.2.2.3.1 SimpleString Aufruf	63
6.2.2.4.1 Merger Aufruf	63
6.2.2.5.1 LLM Aufruf	64
6.2.2.5.2 LLM Config	64
6.2.2.6.1 Stable Diffusion Aufruf	65
6.2.2.6.2 Config File	65
6.2.2.7.1 Overlapper Implementierung	66
6.2.2.8.1 Model Aufruf	66
6.2.3.1 Ablaufdiagramm	67
6.2.3.2 Network.py	68
6.2.3.3 ai_pipe.py	68
6.2.3.4 Stable Config	68
7.3 Rust Software Aufbau	71
7.3.1 Rust Ordnerstruktur	71
7.3.3 Syact Ordnerstruktur	72
8.1 Iot Ablaufdiagramm	73
8.2 Kommunikationskette	74
8.4.1 Plexiglasstifte	77
8.4.2 Plexiglasgriff	77
9.1 Spannungsversorgung	79
9.2.1 Achsen	80
9.2.2 Endschalter	80
9.3 Servosteuerung	81