Hochschule Kempten
University of Applied Sciences

Projektarbeit
Master Informatik

Data Science in der industriellen Produktion

Aufgabensteller/Prüfer:	Prof. Dr. Ulrich Göhner
Verfasser:	Lea Kuznik, Luca Neuburger, Alex Wagner, Christian Weiß, Matthias Wörz
Arbeit vorgelegt am:	13.02.2025
durchgeführt in der	Fakultät Informatik

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

(Matthias)

Die Motivation für unser Projekt bestand darin, dass ein Mitsubishi-Roboter so programmiert werden sollte, dass er in der Lage ist, Teile mit randomisierten Koordinaten aufzusammeln und sie an die Koordinaten zu legen, an denen sie vorgesehen sind. Die Basis hierfür sollte eine synthetische Zeichnung mit den passenden Koordinaten sämtlicher für das Modell vorgesehener Teile sein.
Im Zuge dessen musste zum einen mittels Computer Vision das Teil identifiziert, richtig kategorisiert und seine aktuellen Koordinaten erkannt werden. Des Weiteren musste anschließend anhand der synthetischen Zeichnung erkannt werden, wo das Teil platziert werden sollte.

Zudem musste im Zuge des Projekts ausfindig gemacht werden, wie der Roboter funktioniert und wie er von einer externen Quelle angesteuert werden kann. Letzteres ist insbesondere deswegen wichtig, da anschließend die Koordinaten aus dem ersten Schritt an den Roboter übergeben werden müssen.
Ein Projekt wie dieses könnte für die Industrie gerade deswegen interessant sein, da somit eine Komponente, die aus vielen Einzelteilen besteht, nicht mehr von einem Menschen von Hand zusammengebaut werden müsste. Diese Aufgabe könnte dann ein Roboter mittels Pick-and-Place quasi von alleine erledigen.
Zusätzlich sollte eine geordnete Projektorganisation aufgestellt werden, um den Betrieb in einem realen Unternehmen exakt nachbilden zu können.

1.2 Synthetische Zeichnung

(Lea)

Die CAD-Zeichnung (Abb. 1) beinhaltet alle Objekte, die der Roboter später greifen und in der dargestellten Form zusammensetzen wird. Diese Einzelteile wurden anschließend 3D-gedruckt. Unterscheiden lassen sich diese durch ihre individuellen Formen und Bohrungen.
Ablesbar aus der Legende lässt sich erkennen, dass einige Komponenten auch mehrfach vorkommen können.

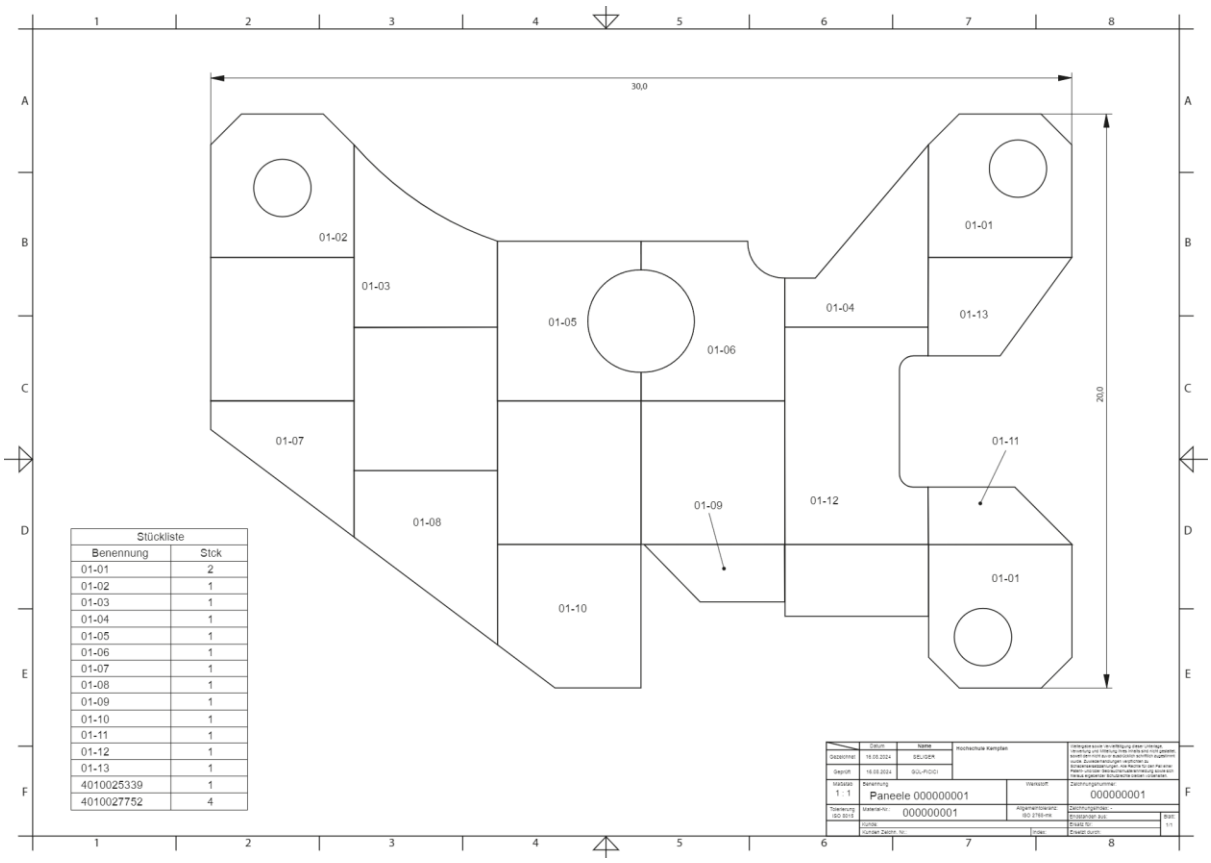

Abb. 1: Synthetische Zeichnung

2 Planung und Organisation

2.1 Aufgabenverteilung

Lea Kuznik

- **Projektmanagement**
  - Spezifikation
  - Unterstützung der Roadmap
  - Unterstützung der Trello-Organisation

- **Entwicklung**
  - Recherche der Kommunikationsmöglichkeiten und Schnittstellen
  - Entwicklung DataSheetParser
  - Entwicklung HuMomentsComparer
  - Unterstützung Docker Container
  - Unterstützung Rotationserkennung
  - Unterstützung Greifpunkt-Bestimmung
  - Refactoring Cleanup

- **Weiteres**
  - Zwischenpräsentation
  - Projektpräsentation
  - Plakatgestaltung
  - Aufsetzen des Discord-Servers
  - Finale Dokumentationsbearbeitung

Luca Neuburger

- **Projektmanagement**
  - Aufsetzen GitHub-Repository

- **Entwicklung**
  - Recherche: Ansteuerung Roboter per ROS
  - Recherche: Ansteuerung Roboter per TCP-Paketen
  - Einarbeiten und Kennenlernen von Docker und Docker-Containern
  - Recherche: Docker-Container in Kombination mit ROS
  - Entwicklung eines Docker-Container zur Ansteuerung des Roboters
  - Design und Konzeptionierung einer möglichen Schnittstelle Bilderkennung/Steuerung

- **Sonstiges**
    - Unterstützung bei der Erstellung des Plakats

Alex Wagner

- **Entwicklung**
    - Entwicklung: Anwendung euklidischer Distanztransformation (euclideanDistanceTransform.py)
    - Entwicklung: Greifpunktbestimmung
    - Entwicklung: Schwerpunktbestimmung (centroid.py)
    - Entwicklung/Unterstützung: Zusammensetzung aller Komponenten (main.py)
    - Unterstützung: Erkennung der Teile über HSV-Farbmaske
    - Unterstützung: Rotationsbestimmung durch kürzeste Kante der Kontur

- **Sonstiges**
    - Recherche: Erste Gedanken zu synthetischer Zeichnung machen
    - Recherche: Roboter mit CIROS ansteuern

Christian Weiß

- **Entwicklung**
    - Erkennung der Teile über HSV-Farbmaske
    - Bestimmung der inneren und äußeren Konturen
    - Rotationsbestimmung durch kürzeste Kante der Kontur
    - Erkennung der Position, Rotation und ID der Teile durch ArUco Marker
    - Entwicklung der Ankerpunkte, um Bildkoordinaten in Relative Koordinaten für den Roboter umwandeln zu können
    - Zusammensetzen des Codes für Erkennung über ArUco Marker in der main.py
    - Unterstützung: Finales Zusammensetzen aller Komponenten in der main.py
    - Unterstützung: Hu-Moments zur Identifizierung der Teile

- **Sonstiges**
    - Recherche: Roboter über Profibus ansteuern
    - Recherche: ArUco Marker
    - Recherche: IDS-Kamera
    - Ansteuerung der IDS-Kamera in Python
    - Anbringen der ArUco Marker an den Teilen

Matthias Wörz

- **Projektmanagement**
    - Aufbau und Verwaltung des Trello-Boards
    - Grobe Verteilung der Rollen
    - Aufbau einer Roadmap zur zeitlichen Übersicht

- **Entwicklung**
    - Ansteuerung des Roboters via CIROS
    - Manuelle Ansteuerung des Roboters zur Konzept-Demonstration
    - Recherche: ROS in Kombination mit Mitsubishi-Roboter
    - Recherche: Direkte Ansteuerung des Roboters
    - Docker Container
    - Konzept-Entwicklung Schnittstelle Bilderkennung/Steuerung

- **Sonstiges**
    - Evaluation Anbindungsmöglichkeiten Roboter
    - Plakatgestaltung

2.2 Tools

2.2.1 Trello

(Matthias)

Im Verlauf des Projekts hat Trello gleich zwei Aufgaben im Zuge des Projektmanagements übernommen. Zum einen half uns das Trello-Board dabei, unsere Aufgaben und deren Fortschritt im Überblick zu behalten. Zum anderen nutzten wir den Trello-Zeitstrahl, um eine Roadmap zu erstellen, um unseren Fortschritt in einem größeren zeitlichen Kontext zu betrachten

Das Trello-Board wurde in die Kategorien “Done”, “In Progress", "On Hold” und “Next Up” unterteilt. Es folgen die Erklärungen der einzelnen Kategorien.

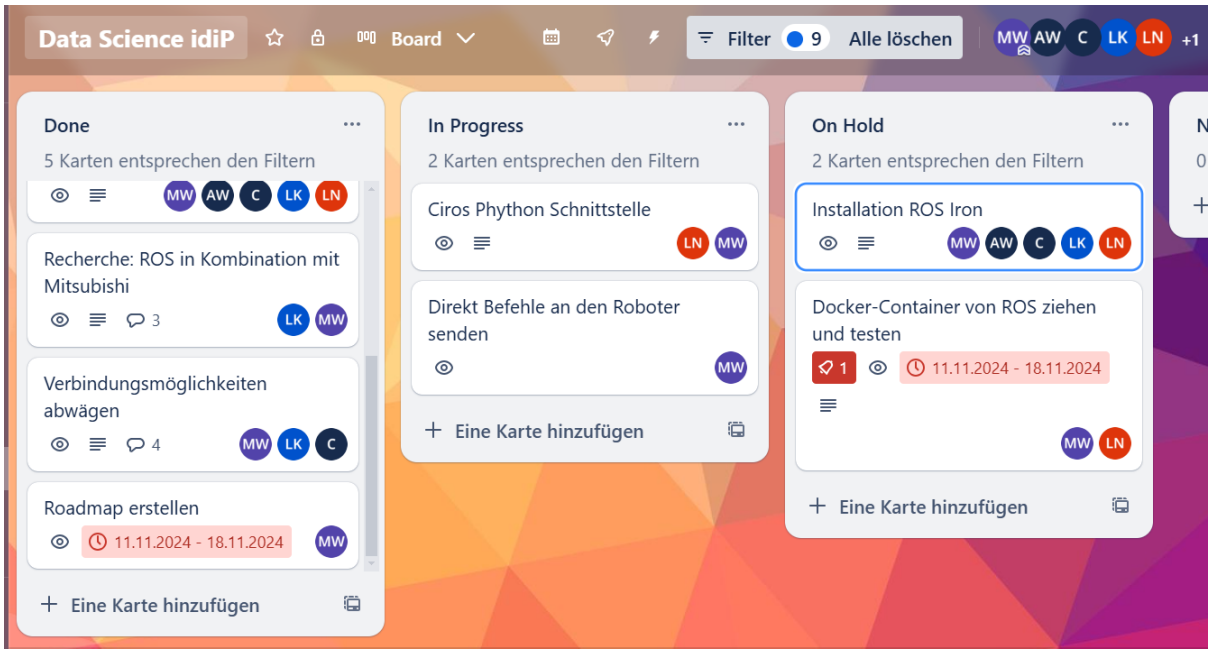

Abb. 2: Trello-Board Anzeige

- **Done**: Hier wurden Aufgaben abgelegt, die bereits erledigt waren oder vor kurzem erledigt wurden. Dank dem Aufbau von Trello konnten diese im Nachhinein trotzdem nochmals betrachtet werden
- **In Progress**: Aufgaben, die im Moment bearbeitet werden. Somit konnte jedes Teammitglied sehen, dass dort im Moment etwas gemacht wird.
- **On Hold**: Aufgaben, die derzeit nicht weiterbearbeitet werden, da sie auf Resultate einer anderen Aufgabe warten mussten oder noch zusätzlichen Klärungsbedarf hervorbrachten.
- **Next**: Aufgaben, die schon definiert waren, aber in Zukunft noch anstanden.

Für jede noch nicht zugeteilte Aufgabe wurden bei unseren wöchentlichen Meetings anschließend Verantwortlichkeiten zugeteilt, die im Bild an den Kreisen in der rechten oberen Ecke der Karte erkennbar sind.

Dank des Trello-Boards konnten wir somit stets alle unsere Aufgaben, den Fortschritt derer und die Verantwortlichkeiten hierfür im Auge behalten.
Wie bereits erwähnt wurde mittels der Trello-Zeitstrahl-Funktion auch eine Roadmap erstellt. Auch hier kamen die Tasks aus dem Trello-Board zum Einsatz und konnten auf einem großen Zeitstrahl platziert werden. Zusätzlich wurden im Trello-Board Karten ergänzt, die als Oberbegriffe dienten, um klarzustellen, wann ein ganzer “Projektblock” fertiggestellt sein sollte. Für jeden dieser so genannten Meilensteine war stets ein Zeitraum von zwei Wochen festgelegt. Dadurch konnte vom Projektstart bis zum Projektende ein zeitlicher Rahmen festgelegt werden, wann welche Aufgabe erfüllt werden sollte.

2.2.2 Git

(Luca)

Die Versionsverwaltung Git wurde im Jahr 2005 als Open-Source-Projekt von Linus Torvalds entwickelt. Zielsetzung war die Schaffung einer effizienten, dezentralen Codeverwaltung, die speziell für die Anforderungen großer Softwareprojekte geeignet ist. Im Gegensatz zu zentralisierten Versionskontrollsystemen ermöglicht Git jedem Entwickler, eine vollständige Kopie des Repositories zu erhalten. Dies erlaubt die Durchführung von Änderungen unabhängig voneinander und deren spätere Zusammenführung. Dies verbessert die Nachverfolgbarkeit von Änderungen und erleichtert die Zusammenarbeit in Entwicklerteams.
Zur Verwaltung des Quellcodes wurde GitHub genutzt, eine cloudbasierte Plattform, die auf Git aufbaut und zusätzliche Funktionen wie Issue-Tracking, Code-Reviews und eine benutzerfreundliche Weboberfläche bereitstellt. Dies führt zu einer effizienteren und zugänglicheren Versionskontrolle für Teams, die an verteilten Standorten arbeiten.
Die Entscheidung für die Nutzung von GitHub basierte auf mehreren Faktoren. Zum einen ist GitHub eines der am weitesten verbreiteten Tools zur Versionskontrolle und wird von einer Vielzahl von Open-Source- und Unternehmensprojekten genutzt. Zum anderen ist es plattformunabhängig und lässt sich einfach in verschiedene Entwicklungsumgebungen integrieren. Als besonders vorteilhaft hat sich die Unterstützung von Git-Branching durch GitHub erwiesen. Die Funktionalität des Git-Branching ermöglicht die parallele Verwaltung verschiedener Entwicklungszweige (Branches), den Test neuer Features und die sukzessive Integration von Änderungen in den Hauptzweig nach erfolgreicher Prüfung.

Die Implementierung dieser Funktionen trug maßgeblich zur Optimierung der Arbeitsweise innerhalb unseres Teams bei, indem sie eine effiziente und kollaborative Projektabwicklung förderte.

2.2.3 Discord

(Lea)

Neben der Organisation in Trello nutzten wir Discord als zentrale Plattform zur Kommunikation. Dieses Kommunikations-Tool ermöglicht es Communities und Teams, sich über Text- und Sprachkanäle auszutauschen, sowie Screen-Sharing und Videoanrufe zu nutzen.

Dank der vielseitigen Einsatzmöglichkeiten konnten wir Textkanäle für unterschiedliche Kategorien einrichten, beispielsweise für den schnellen Austausch von Code oder das Teilen von Bildern und Projektmaterialien. Wichtige Informationen und Dateien ließen sich dadurch einfach und schnell wiederfinden. Spontane Abstimmungen und längere Diskussionen fanden in Discord einen geeigneten Raum, sodass wir unsere Ergebnisse strukturiert festhalten konnten. Für Problembehandlungen oder Meetings nutzten wir häufig die Sprachkanäle, um effektiv zu kommunizieren.

2.3 Spezifikationen

(Lea)

Bevor die Entwicklung begonnen wird, ist es essentiell eine Spezifikation als zentrales Dokument anzulegen, Anforderungen klar zu definieren und wichtige Schritte zu beleuchten, damit diese nicht vergessen werden, um die Planung übersichtlicher zu gestalten.
Eine Spezifikation ist in den häufigsten Fällen ein “lebendiges” Dokument, welches während der Entwicklung ergänzt oder abgeändert wird. In unserem Fall erleichterte uns die Spezifikation die Erweiterung des Trello-Boards und die Verfassung der Roadmap.

Nr.	**Use Cases**	**Functional Requirements / System Responsibility**
00	Projekt starten	1. Roboter-Kommunikation starten 2. Kamera starten 3. Bildverarbeitung initialisieren
01	Teil auf der Fläche erkennen	1. In Startposition gehen 2. Fokus mit Kamera auf positioniertes Teil richten 3. Teil identifizieren
02	Vorbereitung des Pick-and-Place-Prozesses initialisieren	1. Position und Orientierung erfassen 2. Rotationsabweichung berechnen 3. Greifpunkt berechnen
03	Teil einer Zielposition zuweisen	1. Teil einer verfügbaren Position zuweisen 2. Position für andere Teile als belegt markieren
04	Ausgewähltes Teil anheben	1. Schwerpunkt des Teils berechnen 2. Vakuumsauger an Greif-Punkt ausrichten 3. Roboterarm zur Teilposition bewegen und absenken 4. Ansaugen starten
05	Teil an Zielposition platzieren	1. Roboterarm zur Zielfläche bewegen 2. Arm absenken und Teil platzieren 3. Ansaugen stoppen und Teil freigeben 4. Zurück an Startposition bewegen

Aus den Use Cases lassen sich ebenfalls nicht-funktionale Anforderungen über die Performance, Genauigkeit, Zuverlässigkeit und Sicherheit ableiten, welche in einer weiteren Tabelle zusammengefasst werden:

Use Case Nr.	**Non-Functional Requirements**
UC00	Der Roboter muss eine stabile Verbindung zu allen angeschlossenen Geräten herstellen können. Die Kommunikation muss in Echtzeit erfolgen. Die Kamera muss eine ausreichende Auflösung haben, um die Teile zu erkennen.
UC01	Die Bildverarbeitung muss in Echtzeit erfolgen. Die Erkennung muss mit unterschiedlichen Lichtverhältnissen und Hintergrundrauschen umgehen.
UC02	Die Bildverarbeitung muss den Schwerpunkt des Teils präzise berechnen. Die Orientierung des Teils muss korrekt erfasst werden, um einen optimalen Greifpunkt zu gewährleisten.
UC03	Die Zielposition muss eindeutig identifizierbar sein. Die Zuordnung muss in Echtzeit aktualisiert werden, um Konflikte zu vermeiden.
UC04	Der Vakuumsauger muss eine ausreichende Saugkraft haben, um das Teil sicher zu halten. Die Bewegung des Arms muss präzise und vorsichtig sein.
UC05	Die Platzierung des Teils muss präzise und vorsichtig sein. Der Roboter muss in der Lage sein, die Startposition wiederzufinden.

Aus der Spezifikation lassen sich folgende Risiken und Annahmen schließen, welche während der Entwicklung berücksichtigt werden müssen:

- Risiken, wie eine fehlerhafte Objekterkennung aufgrund beispielsweise schlechter Lichtverhältnisse oder Kommunikationsfehler beim Greifen und Platzieren der Teile und
- Annahmen über eine geeignete Form der Teile zum Ansaugen oder eine vorab definierte und zugängliche Zielposition für den Roboter.

3 Technologien

3.1 Mitsubishi RV-2SDB

(Luca)

Für die Aufgabe sollte ein 6-Achs-Roboter für das Greifen der Teile und das korrekte Positionieren der Teile genutzt werden. Der von uns genutzte Roboter-Arm für diese Aufgabenstellung wurde durch die Hochschule Kempten uns zur Verfügung gestellt. Es handelt sich genauer gesagt um einen Mitsubishi RV-2SDB. Dieser kann in Kombination mit einem Controller, in diesem Fall der Controller CR1DA-771-S15. Dieser Controller kann über verschiedene Wege Daten empfangen und somit mit dem Roboter kommunizieren.

Manuelle Steuerung:

- Um den Roboter per manueller Steuerung anzusteuern, ist es zum einen wichtig, dass der Controller in dem manuellen Modus ist. Dabei empfängt der Controller keine Daten über die automatischen Schnittstellen.
- Die Steuerung des Roboters erfolgt im manuellen Modus erfolgt über das Hand-Terminal. Dabei muss das Hand-Terminal, zusätzlich zum manuellen Modus, aktiviert werden und an dem Hand-Terminal der Tot-Mann-Schalter getriggert werden. Dieser Tot-Mann-Schalter “aktiviert” bzw. schält das Hand-Terminal für die Robotersteuerung frei.

Automatische Steuerung:

- Neben der manuellen Steuerung hat der Controller noch die Möglichkeit in einen automatischen Modus geschalten zu werden. Wenn der manuelle Modus aktiv ist, empfängt der Controller keine Befehlseingaben mehr vom Hand-Terminal. Stattdessen empfängt der Controller die Steuerungseingaben von einer der drei folgenden Schnittstellen: RS232, Profibus, TCP/IP. Die Vor- und Nachteile dieser Schnittstellen werden in Kapitel 5.1 genau diskutiert.

3.2 IDS-Industriekamera

(Christian)

Für die Erkennung der Teile wurde eine IDS-Kamera verwendet, die über dem Roboterarm am Rahmen des Sicherheitsglases befestigt wurde.

Um die Kamera verwenden zu können, muss der entsprechende Treiber des Herstellers installiert werden. Zusammen mit dem Treiber kommt auch das IDS-Cockpit, womit auf die Kamera ohne Code zugegriffen werden kann. Der Treiber lässt sich auf der Webseite des Herstellers finden.

Um die Kamera im Python Code aufzurufen, müssen zuerst die benötigten Packages installiert werden.

```
!pip install ids_peak_ipl
!pip install ids_peak
```

3.3 ArUco Marker

(Christian)

Für die Erkennung der Teile wurde entschieden, Arco-Marker zu verwenden. Vereinfacht gesagt können sich Arco-Marker als eine Art QR-Code vorgestellt werden. Ein ArUco-Marker enthält eine individuelle ID. Außerdem lassen sich durch in OpenCV vorhandene Funktionen auch die Eckpunkte der ArUco-Marker auslesen. ArUco-Marker haben noch weitere Funktionalitäten, die wir im Rahmen dieses Projektes aber nicht verwendet haben.

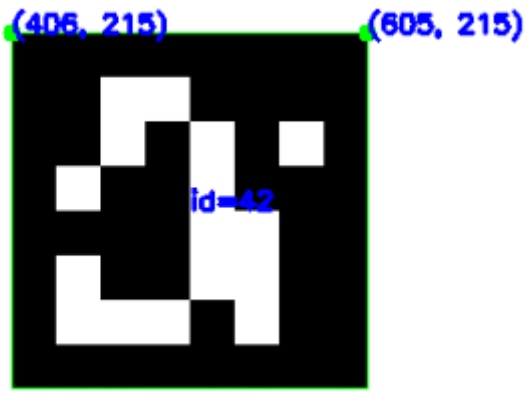

Abb. 3: ArUco Marker mit ID

3.4 OpenCV

(Alex)

Bei OpenCV (Open Source Computer Vision Library) handelt es sich, wie der Name bereits vermuten lässt, um eine freie Bibliothek mit Algorithmen, welche hauptsächlich in der Computer Vision, aber auch in der Bildverarbeitung verwendet werden.

Obwohl OpenCV in C++ geschrieben ist, bietet das Python-Paket "opencv-python" die gleiche Funktionalität für die Sprache Python an.

Im Folgenden werden ein paar Funktionen, welche in diesem Projekt zur Objekterkennung verwendet wurden, aufgezählt und kurz beschrieben:---

- cv.contourArea(): Berechnet eine Konturfläche. Mithilfe eines Schwellwertes sollen Anomalien im Binärbild (Maske) nicht als Teile erkannt werden, wenn dessen Konturflächen unterhalb des Schwellwertes liegen.

- cv.cvtColor(): Konvertiert ein Bild von einem Farbraum in einen anderen. In unserem Fall konvertieren wir das RGB-Kamerabild in den HSV-Farbraum, da die Definition des zulässigen Bereiches für die Farbe “Rot” zur Erkennung der Teile im HSV-Farbraum intuitiver erscheint, als im RGB-Farbraum.

- cv.distanceTransform(): Berechnet die (euklidische) Distanz zum nächstgelegenen Null-Pixel für jedes Pixel des Quellbildes. Wird zur Greifpunktbestimmung für Teile mit Bohrung verwendet.

- cv.findContours(): Findet Konturen in einem Binärbild (Maske).

- cv.HuMoments(): Berechnet die sieben Hu-Invarianten (Hu-Momente).

- cv.inRange(): Prüft, ob Array-Elemente zwischen den Elementen von zwei anderen Arrays liegen. Wird in unserem Anwendungsfall für die Erzeugung des Binärbildes / der Maske aus einem Bild, welches in den HSV-Farbraum konvertiert wurde, verwendet.

- cv.matchShapes(): Vergleicht zwei Formen unter Verwendung der Hu-Menge invarianter Momente (Hu-Momente).

- cv.minMaxLoc(): Findet das globale (Minimum und) Maximum in einem Array. Wird zur Greifpunktbestimmung für Teile mit Bohrung verwendet.

- cv.moments(): Berechnet alle Bild-Momente bis zur dritten Ordnung eines Polygons oder einer gerasterten Form. Wird für die Berechnung der Hu-Momente verwendet.

3.5 Docker

(Luca)

3.5.1 Was ist Docker?

Docker ist ein Programm, mit dem man Anwendungen in sogenannten Containern laufen lassen kann. Diese Container enthalten alles, was eine Anwendung zum Funktionieren braucht. Dadurch ist es einfacher, Programme auf verschiedenen Systemen zu nutzen, ohne sich um Abhängigkeiten kümmern zu müssen.

Anders als bei virtuellen Maschinen, die ein komplettes Betriebssystem nachbilden, nutzt Docker das vorhandene Betriebssystem und trennt nur die Anwendungen voneinander. Ursprünglich wurde Docker für Linux entwickelt, aber mittlerweile funktioniert es auch auf Windows und macOS.

3.5.2 Was sind die Vorteile und die Nachteile von Docker?

Ein großer Vorteil von Docker ist seine Plattformunabhängigkeit. Da Container immer gleich funktionieren, unabhängig davon, auf welchem System sie ausgeführt werden, kann eine Anwendung problemlos von der Entwicklungsumgebung in die Produktionsumgebung übertragen werden. Dies vermeidet das klassische "Es funktioniert auf meinem Rechner"-Problem. Zudem sind Docker-Container leichtgewichtig und starten schneller als virtuelle Maschinen.

Allerdings gibt es auch einige Nachteile. Docker-Container teilen sich den Kernel des Host-Systems, was bedeutet, dass nicht alle Betriebssysteme gleichzeitig in Containern ausgeführt werden können – zum Beispiel kann ein Windows-Container nicht direkt auf einem Linux-Host laufen. Dieser Nachteil kann in diesem Anwendungsfall vernachlässigt werden, weil ROS, was für die Kommunikation genutzt werden soll, auf einem Linux-Container basiert und somit der Container als ein Linux-Container aufgesetzt werden kann.

3.5.3 Wieso setzen wir Docker ein?

Die Entscheidung, Docker in unserem Projekt einzusetzen, wurde aus mehreren Gründen getroffen. Ein zentraler Aspekt war die Plattformunabhängigkeit von Docker. Da unser Team auf unterschiedlichen Betriebssystemen arbeitete, wollten wir eine Umgebung schaffen, die überall identisch funktioniert – unabhängig davon, ob jemand mit Windows, macOS oder Linux arbeitet. Durch den Einsatz von Docker konnten wir sicherstellen, dass alle Teammitglieder mit der gleichen

Softwarekonfiguration arbeiten, ohne individuelle Anpassungen für jedes System vornehmen zu müssen.
Ein weiterer wichtiger Grund war das Umgehen einer direkten Installation von ROS (Robot Operating System) auf einem Windows-Gerät. ROS ist primär für Linux entwickelt worden, und die Installation unter Windows ist entweder kompliziert oder mit Einschränkungen verbunden. Mit Docker konnten wir eine vorgefertigte ROS-Umgebung bereitstellen, die ohne tiefgreifende Systemänderungen auch auf Windows lauffähig war. Dies erleichterte die Entwicklung erheblich und reduzierte mögliche Inkompatibilitätsprobleme.
Zusätzlich spielte auch der Lernaspekt eine Rolle. Keiner im Team hatte zuvor Erfahrung mit Docker, und wir sahen die Gelegenheit, uns mit einer modernen Technologie vertraut zu machen. Indem wir Docker in unser Projekt integrierten, konnten wir nicht nur unser eigentliches Ziel umsetzen, sondern auch wertvolle Erfahrungen im Umgang mit Container-Technologien sammeln. Die Entscheidung für Docker war somit nicht nur technisch motiviert, sondern auch eine bewusste Wahl, um als Team Neues auszuprobieren und unser Wissen zu erweitern.

3.6 CIROS Studio

(Matthias)

Mittels CIROS Studio gelang es uns, den Roboter auch von einem PC aus anzusteuern. Ciros Studio ist eine erweiterte Version der Ciros-Simulationssoftware, die speziell für die Entwicklung, Simulation und virtuelle Inbetriebnahme komplexer Automatisierungssysteme entwickelt wurde. Es ermöglicht die Erstellung und Anpassung detaillierter 3D-Modelle von Robotern, Maschinen und Produktionsumgebungen. Mit der Anwendung können Nutzer Steuerungsprogramme direkt in der virtuellen Umgebung testen und optimieren, bevor sie in der realen Produktion implementiert werden. Die Software unterstützt verschiedene Kommunikationsprotokolle, sodass sie mit externen Steuerungssystemen, wie SPSen und Industrierobotern interagieren kann. Zudem bietet Ciros Studio erweiterte Analyse- und Debugging-Tools, um die Effizienz und Sicherheit von Automatisierungsprozessen zu verbessern.
In Ciros gab es zwei Möglichkeiten den Roboter anzusteuern. Zum einen konnten in einem 3D-Modell des Roboters sowohl die einzelnen Achsen als auch das Objekt aus einer Werkzeug-Perspektive manuell bewegt werden. Dies war hilfreich, um ein Gefühl für die Beweglichkeit des Roboters zu entwickeln und zudem erste Probe-Versuche durchzuführen.

Außerdem war es möglich MELFA-Basic Code in den Controller einzuspielen. Dies war insbesondere deswegen hilfreich, da dieser sonst über das Pad des Controllers eingegeben werden musste. Dieses funktioniert wie eine alte Handy-Tastatur (also einen Knopf dreimal drücken, um den dritten Buchstaben dieses Knopfs zu erreichen). Somit versuchten wir unter anderem auch den Code einzuspielen, der die Schnittstelle für den Docker-Container öffnet. Oder in anderen Worten: Der Code, der den Roboter empfänglich für Schnittstellen von externen Quellen macht.
Das Problem bei CIROS ist, dass es aufgrund seiner gezielten Funktionalität keine Schnittstellen zu anderen Programmiersprachen wie beispielsweise Python unterstützt. Daher war es zwar zur Simulation der Bewegungen des Roboters geeignet, aber für den praktischen Gebrauch nicht.

4 Objekterkennung

4.1 Erkennung über HSV-Farbmaske

(Lea)

Erfolgt die Objekterkennung ohne die Wahl von ArUco-Marker, wird die Maskierung des Bildes zur Hervorhebung von Objekten in der Farbe Rot verwendet. In der erstellten Maske als binäres Bild werden dann die Konturen bestimmt und in einer Liste gespeichert. Es wird zwischen den äußeren und inneren Konturen entschieden. Falls eine innere Kontur vorhanden ist, erkennt das Programm diese als Bohrungen in einem Teil.
Die Werte der Konturen werden anschließend in Hu Momente konvertiert.

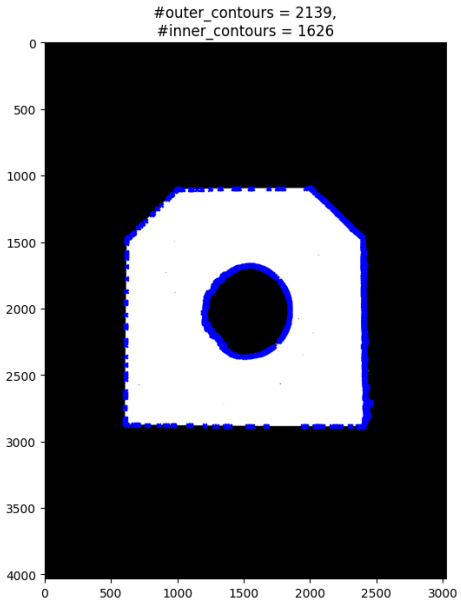

Abb. 4: Konturerkennung bei Teil mit Bohrung

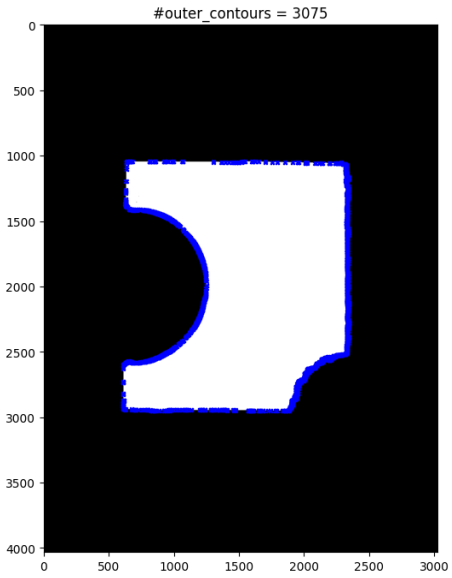

Abb. 5: Konturerkennung bei Teil ohne Bohrung

4.1.1 Hu Momente und Konturen

(Lea)

Hu Momente werden verwendet, um die einzelnen Teile über ihre äußeren und inneren Konturen zu identifizieren und dienen als Formvergleich. Diese Hu Momente werden in der Regel aus sieben zentralen (Bild-) Momenten, unabhängig von Translation, Rotation oder Skalierung einer Form oder eines Objekts.

Aus der “Datasheet”-JSON-Datei im Projekt lassen sich Eigenschaften und Informationen aller Teile herauslesen, wie beispielsweise die ID, Koordinaten der Teile, Anzahl der Bohrungen, etc. Die Datei kann man als eine verschachtelte Tabelle verstehen, deren Struktur sich aus Schlüssel-Wert-Paaren, Objekten bestehend aus Schlüssel-Wert-Paaren und Arrays, bestehend aus Werten oder weiteren Verschachtelungen zusammensetzt.


```
{
  "materialId": "0000000001",
  "tiles": [
    {
      "id": 1,
      "tileId": "0a7890b7-391a-442a-89c8-72bbf82bcbe1",
      "tileLabel": "01-01",
      "number": 2,
      "characteristics": {
        "numberCutouts": 1,
        "huMomentsOutlines": [
          0.16389115020927977,
          5.07031811491243e-07,
          6.551170355902457e-05,
          1.946901415487922e-06,
          -2.198747647329342e-11,
          -1.3863138590303904e-09,
          3.0688553922955915e-18
        ]
      }
    }
  ]
}
```


Abb. 6: Code-Schnipsel von "datasheet.json"

In der Abbildung 6 wird ein verkürzter Auszug der JSON-Datei dargestellt. In der Datei selbst interessieren wir uns hauptsächlich für die Teile-ID, die Charakteristiken, die Anzahl der Bohrungen und die äußeren Hu Momente.
Diese Schlüssel-Werte werden anschließend im “JSON-Parser” herausgefiltert. Die Funktion “find_values_by_key” ermöglicht die Durchsuchung über ein rekursives Verfahren über die Verschachtelung der JSON-Struktur hinaus zu finden und die dazugehörigen Werte zu speichern. “get_tile_info” extrahiert im Anschluss die wichtigsten Merkmale und damit auch den Bezug zwischen den Hu Momente der Teile-ID und gibt diese zurück.

Die Hu Momente aus der JSON-Datei sollen nun mit den erfassten Hu Momenten der Maske verglichen werden, um die prozentuale Übereinstimmung zu bestimmen. Der Referenzsatz sind die Hu Momente der Maske, welche als Eingabe erwartet werden. Eine Funktion durchläuft dann die entnommene Liste aus der JSON-Datei und identifiziert den ähnlichsten Wert mittels “Shape-Matching”, eine Methode aus OpenCV. Dafür wird über diese Liste iteriert und alle Daten jedes Einzelteils überprüft und anschließend berechnet, wie hoch die sogenannte “Match-Value” zur Referenz ist.

Die Funktion aktualisiert kontinuierlich den besten Match, basierend auf dem niedrigsten Ähnlichkeitswert und gibt dann die ID des Teils zurück, dessen Match am höchsten ist.

Es ist nicht notwendig, die inneren Hu Momente der Teile mit Bohrungen zu berücksichtigen, um sie zu unterscheiden und die richtige ID zuzuweisen. Die äußeren Hu Momente der Teile mit Bohrungen unterscheiden sich, sodass im “Shape-Matching”-Prozess keine Verwechslungsgefahr besteht.

Das Programm weiß nun, welches der Teile der Roboter in den nächsten Schritten ansaugen wird und kann die Zielposition bestimmen.

4.1.2 Rotationserkennung

(Lea)

Die Rotationserkennung ist ausschlaggebend, um im Prozess der Positionierung die Rotation zu korrigieren. Es gibt verschiedene Herangehensweisen, eine Ausrichtung eines Bauteils zu ermitteln. Wir entschieden uns dazu, die kleinste Kante eines Teils zu bestimmen und daraus die Abweichung des Wertes des Winkels nach der synthetischen Zeichnung zu nutzen. Die kürzesten Kanten erweisen sich als vorteilhafter, da
a) alle Winkel einen vereinfachten Wert wie beispielsweise von 45° oder 90° besitzen
b) in (fast) allen Fällen jedes Teil nur eine kürzeste Kante enthält.

Dazu werden die Eckpunkte und Kantenlängen durch Konturapproximation von OpenCV markante Punkte ermittelt und in der Maske markiert. Die Kantenlängen zwischen den Eckpunkten werden berechnet und somit die kürzeste Kante bestimmt. Zusätzlich muss für die Steuerung des Roboters der Winkel von Radiant in Grad umgewandelt werden.

4.1.3 Greifpunktbestimmung

(Alex)

Bei der Greifpunktbestimmung ist zunächst zu unterscheiden, ob das zu greifende Teil eine Bohrung enthält oder nicht. Dies kann über die Liste der inneren Konturen bestimmt werden. Wenn das betroffene Teil eine Bohrung enthält, so enthält die Liste der inneren Konturen die Koordinaten der Punkte, welche die Lage der Bohrung beschreiben. Sollte das betroffene Teil andererseits keine Bohrung enthalten, so ist die Liste der inneren Konturen leer.

Enthält das zu greifende Teil keine Bohrung, so kann der Greifpunkt über den äußeren Schwerpunkt, welcher mithilfe der äußeren Konturen (aber auch alternativ über die äußeren Bildmomente) berechnet wird, bestimmt werden.

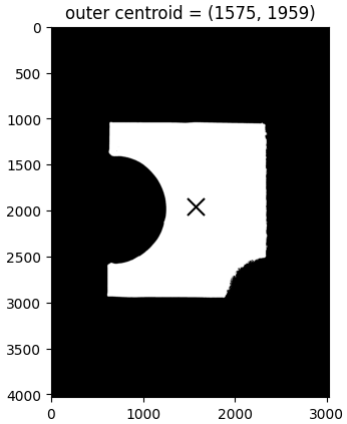

**Abb. 7: Binärbild des Teils "01-06"
mit eingezeichnetem äußerem Schwerpunkt**

Sollte das zu greifende Teil andererseits eine Bohrung enthalten, so muss der Greifpunkt anders als im Falle, dass das Teil keine Bohrung enthält, bestimmt werden, da der äußere Schwerpunkt wie zum Beispiel im Teil "01-02" sich mitten in der Bohrung befinden kann, und der Roboter somit wortwörtlich "ins Leere greifen" würde.

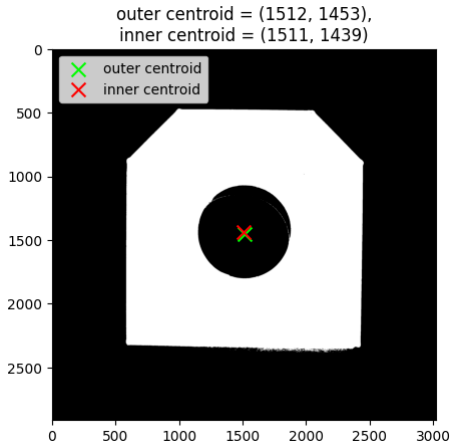

**Abb. 8: Binärbild des Teils "01-02" mit
eingezeichnetem äußerem und innerem Schwerpunkt**

Stattdessen wird unter Anwendung euklidischer Distanztransformation für jedes Pixel in der Maske die Distanz zum nächsten schwarzen Pixel berechnet und anstelle der Pixelwerte eingetragen. Für die schwarzen Pixel in der Maske ändert sich nichts, da die Distanz zum nächsten schwarzen Pixel, also zu sich selbst, gleich Null ist, was auch dem Ausgangswert der schwarzen Pixel entspricht.

Bei den weißen Pixeln in der Maske, welche das Teil darstellen, werden diejenigen Pixel, welche näher zum Rand des Teils, und somit näher an den schwarzen Pixel sind, zwar Werte größer Null haben; aber je weiter entfernt vom Rand die Pixel sind, desto höher wird ihr Wert sein.

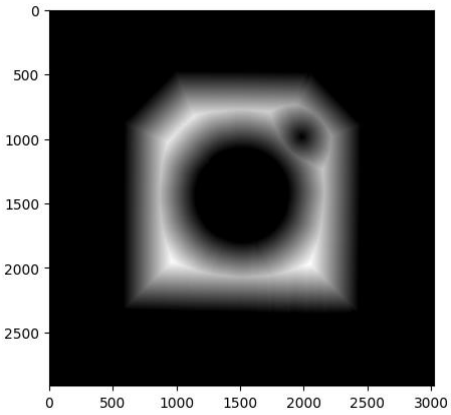

Abb. 9: Ergebnis einer euklidischen Distanztransformation (Zu beachten: Schwarzes "Loch", welches entsteht, wenn schwarze Pixel durch Rauschen o.ä. in den Bildbereich des Teils gelangen)

Nach Anwendung der euklidischen Distanztransformation wird das Pixel mit dem höchsten Wert, welches der größten Distanz zum nächsten schwarzen Pixel entspricht, bestimmt und dessen Position als Greifpunkt verwendet. Dieses Verfahren kann außerdem für Teile beliebiger Form verwendet werden.

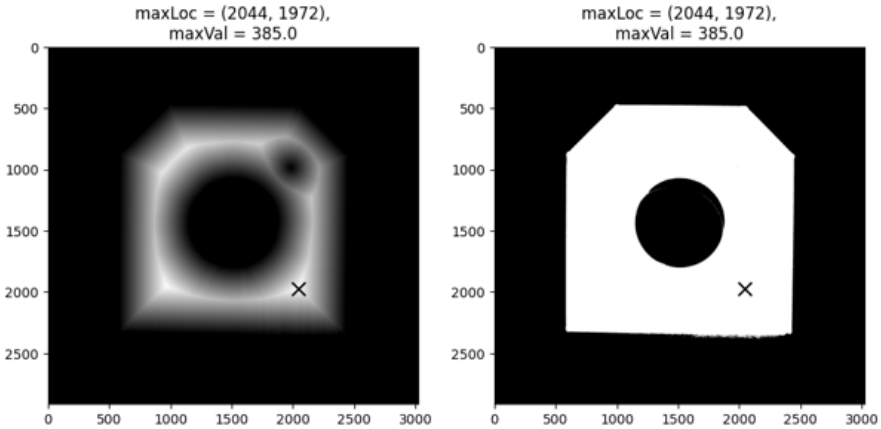

Abb. 10: Ergebnis euklidischer Distanztransformation und Binärbild mit, im Anschluss an euklidischer Distanztransformation ermitteltem, eingezeichnetem Greifpunkt

4.2 Erkennung über ArUco-Marker

(Christian)

Bei dieser Methode wurde auf alle Teile ein individueller ArUco-Marker geklebt. Dabei wurden die Marker so auf den Teilen positioniert, dass jeweils die obere linke Ecke ausreichend vom Rand entfernt ist, so dass der Roboter das Teil problemlos an diesem Punkt ansaugen kann.

4.2.1 Ablauf

Da jeder Marker eine individuelle ID besitzt, braucht es kein gesondertes Verfahren, um das jeweils vorliegenden Teil zu bestimmen. Durch den in OpenCV bereits vorhandenen ArUco Detector lassen sich die ID sowie die Bildkoordinaten der 4 Eckpunkte der Marker ganz einfach auslesen.

Um die Rotation eines Teiles zu bestimmen, werden die ersten beiden Ecken, das sind oben links und oben rechts, ausgelesen und mit folgender Formel die Steigung der durch beide Punkte verlaufenden Geraden berechnet. Mithilfe der Steigung wird zuerst der Winkel zur x-Achse in Radiant berechnet und anschließend in Grad umgerechnet.

Die Position des Greifpunktes wird, wie bereits beschrieben, durch die obere linke Ecke der ArUco-Marker vorgegeben. Um daraus die relative Position zu den Ankerpunkten zu berechnen, wird jeweils der Abstand der x- und y-Koordinaten in Prozent ausgerechnet. Bei den Ankerpunkten werden ebenfalls die Koordinaten der oberen linken Ecke verwendet (siehe Abb. 11).

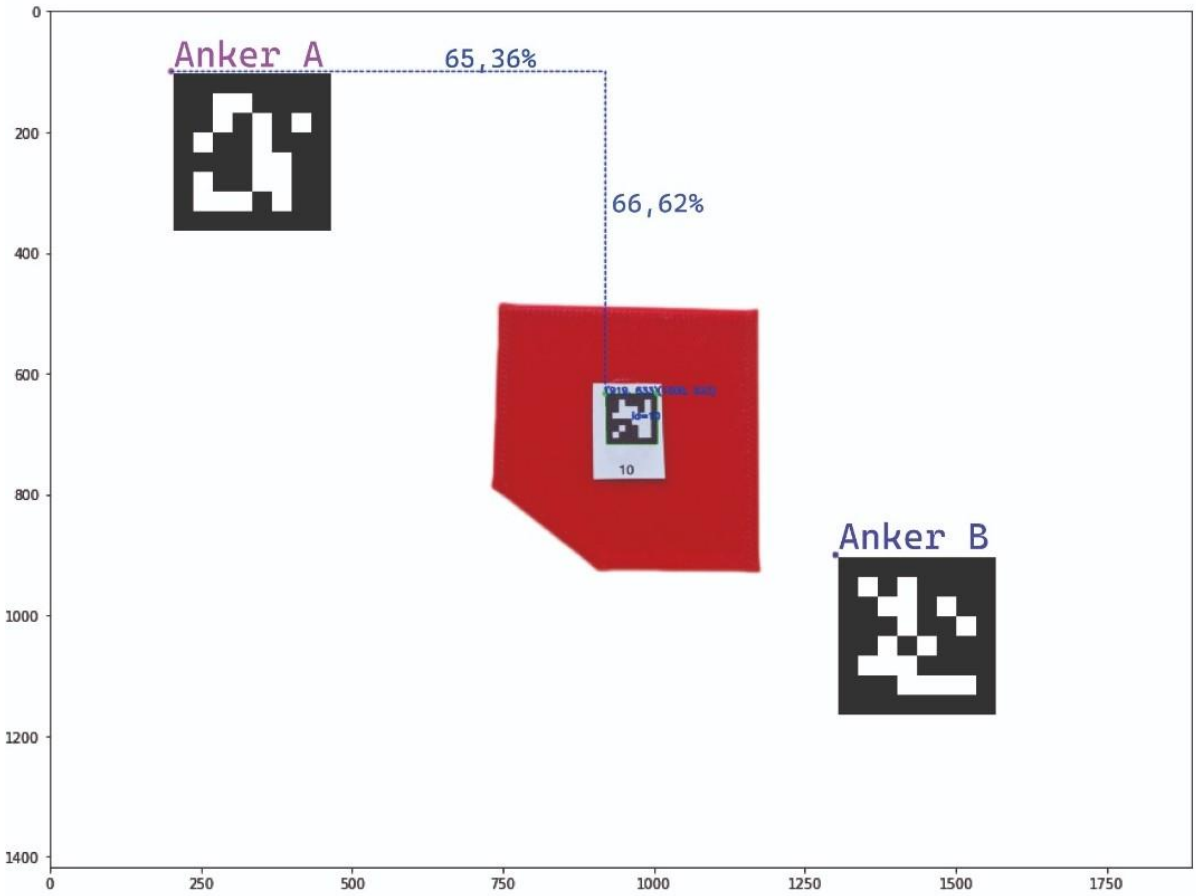

Abb. 11: Teil zwischen zwei Ankerpunkten

4.3 Bedingungen

(Christian)

Damit die Berechnung funktioniert und keine Exception erzeugt, müssen eine Reihe von Bedingungen erfüllt werden. Für die Erkennung über die HSV-Farbmaske muss gelten:

- Die Ankerpunkte müssen beide gesetzt sein. Das wird jeweils abgefragt dadurch, dass die x-Koordinate des jeweiligen Ankers nicht None ist.

Für die Erkennung über ArUco-Marker muss außerdem gelten:

- Erkannte ID darf nicht 42 oder 69 sein, da dies die ID's der Ankerpunkte sind.

Weitere Abfragen:

- Das aktuell erkannte Teil darf nicht das zuletzt erkannte Teil sein.

- Der Delay muss größer 30 sein, das dient dazu, dass man genug Zeit hat, das Teil in Position zu legen, und es nicht bereits erkannt wird, wenn es gerade erst im Bild auftaucht.

4.4 Problematik in der Objekterkennung

4.4.1 Handling, wenn mehrere Teile von der Kamera erfasst werden

(Alex)

Bislang wurden für die Implementierung der Objekterkennung Bilder verwendet, welche beispielsweise mit einer Handykamera geschossen wurden. Bei der Objekterkennung wurden in diesem Zusammenhang einige Annahmen getroffen. Seitdem die Bilder in Zukunft jedoch von einer Kamera (IDS-Industriekamera), welche mittlerweile am Gehäuse befestigt ist, stammen werden, müssen die bisherigen Annahmen überdacht und überarbeitet werden.

Ein erstes Problem ist die Tatsache, dass im Kamerabild neben dem zu greifenden Teil, ein weiteres rotes Objekt im Bilde ist, welches mit der aktuellen Implementierung der Objekterkennung, und zwar der, dass immer nur ein rotes Objekt im Bild erwartet wird, in Konflikt gerät.

Ein möglicher Lösungsvorschlag wäre, den für die Objekterkennung sichtbaren Bildbereich einzuschränken. Folgende Abbildungen zeigen zum einen die bisher beschriebene Problematik und den Bereich, welcher für die Objekterkennung in Zukunft beispielsweise in Frage kommen könnte.

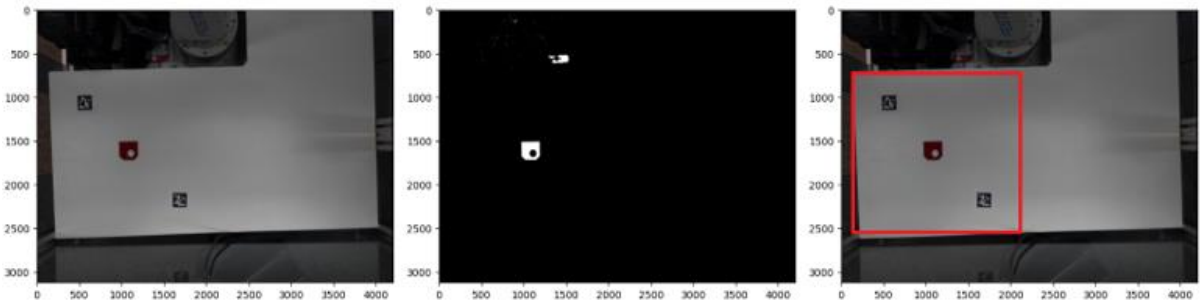

Abb. 12: Kamerabild und Binärbild des Kamerabildes, jeweils mit rotem, die Objekterkennung störendem, Objekt im Bilde; Vorschlag für die Objekterkennung in Frage kommenden, eingeschränkten Bereich

Ein weiteres Problem ergibt sich daraus, dass mit den bislang für die Berechnung der Maske verwendeten Abgrenzungen (Boundaries) des HSV-Farbraumes, die Teile mit den neuen Kamerabildern nicht mehr erkannt werden.

Die folgenden beiden Lösungsvorschläge können in Betracht gezogen werden: Zum einen könnten die Abgrenzungen des HSV-Farbraumes, insbesondere der Value- bzw. der Helligkeitswert, für die neuen Kamerabilder angepasst werden. Andererseits könnte darüber nachgedacht werden, das Kamerabild mit einer passenden Lichtquelle zu beleuchten.

Die folgenden beiden Bilder zeigen zum einen, wie ein Kamerabild mit den bisherigen Abgrenzungen für den HSV-Farbraum aussieht und zum anderen, wie es beispielsweise aussehen sollte.

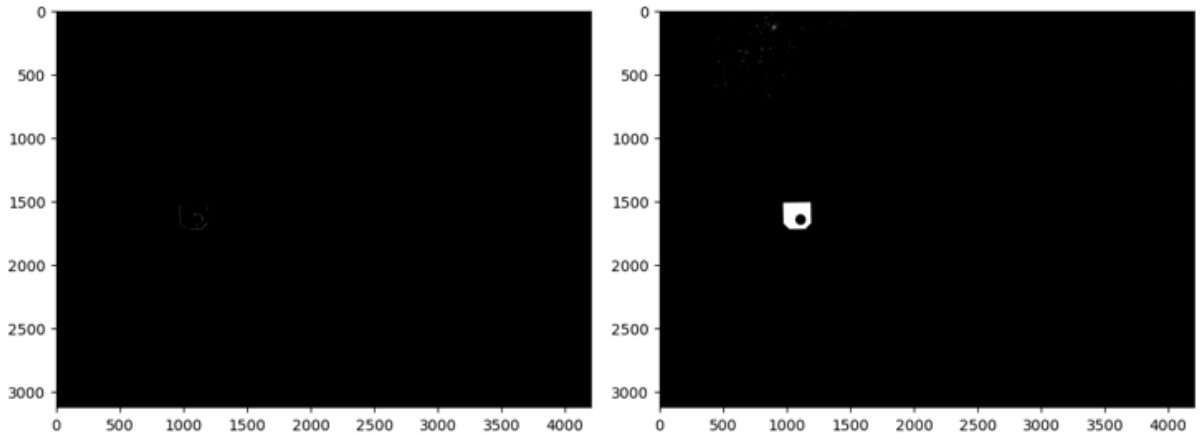

Abb. 13: Binärbild des Kamerabildes mit bisherigen Abgrenzungen des HSV-Farbraumes; und gewünschtes Binärbild des Kamerabildes

4.4.2 Verwackelte Aufnahmen

(Lea)

Wie bereits erwähnt, ist die Industriekamera am Gehäuse befestigt worden. Durch Auslösen des Kompressors, der ein Vakuum am Sauger des Roboters erzeugt, gerät das Gehäuse in Schwingung. Die Schwingungen übertragen sich auf das Kamerabild. Dieses wird optisch verzerrt, sodass die Bilderkennung ohne Korrekturmaßnahmen nicht mehr ordnungsgemäß erfolgen kann.

Ein möglicher Lösungsansatz wäre, die Objekterkennung in gewählten Zeitabständen auszulösen, die nicht während der Laufzeit des Kompressors fallen.

4.4.3 Rotationserkennung für andere synthetische Zeichnungen & Teile

(Lea)

Die Rotationserkennung funktioniert im Falle für dieses Projekt bezogen auf die synthetische Zeichnung und deren Einzelteile. Sollte die Anzahl der kürzesten Kanten variieren, stellt die Methodik der Rotationserkennung eine weitere Herausforderung dar.

Ein Lösungsansatz könnte über den Vergleich der Position des Schwerpunkts eines Teils in Betracht gezogen werden. Falls ein Objekt in einer zufälligen Ausrichtung liegt, ist auch die relative Position seines Schwerpunktes verschoben.

5 Steuerung

5.1 Kommunikationsmöglichkeiten

5.1.1 RS232

(Matthias)

RS232 bietet eine einfache Implementierung und ist aufgrund seiner langen Etablierung und guter Dokumentation gut geeignet für grundlegende Kommunikationsaufgaben. Die Technologie ist stabil und zuverlässig, auch bei längeren Kabelstrecken von bis zu 15 Metern, und kostengünstig im Vergleich zu modernen Alternativen (auch wenn der Kostenaspekt für unser Projekt keine größere Rolle gespielt hat). Insbesondere für einfache Steuerbefehle und das Abrufen von Statusinformationen kann RS232 ausreichen.

Hierbei werden allerdings auch schon erste Nachteile von RS232 erkennbar. Die Übertragungsgeschwindigkeit ist mit maximal 115.200 bps relativ gering, was bei großen Datenmengen problematisch sein kann. Zudem ist die Reichweite auf etwa 15 Meter begrenzt, was in bestimmten Szenarien suboptimal sein könnte. Auch die Unterstützung komplexer Steuerungsprotokolle ist bei RS232 begrenzt, was in modernen Anwendungen oft nicht ausreicht. Die Technologie wird zunehmend von schnellen und vielseitigen Alternativen wie USB oder Ethernet abgelöst.

Insgesamt ist RS232 eine geeignete Wahl, wenn die Datenübertragung zwischen PC und Roboter nicht umfangreich ist, der Roboter relativ nah am PC steht und eine einfache, stabile Verbindung ausreicht. In komplexeren Szenarien wären jedoch andere Technologien wahrscheinlich die bessere Wahl.

5.1.2 Profibus

(Christian)

Der Profibus, ausgeschrieben Process Field Bus.

Der Vorteil vom Profibus ist, dass die Verbindung Bidirektional ist, das heißt, das Signal kann in beide Richtungen gesendet werden, wodurch es möglich ist, Feedback vom Roboter zu empfangen.
Der Nachteil vom Profibus ist, dass es eher für mehrere Geräte ausgelegt ist, genauer ist es ein Multi-Master-System Die Master wären in diesem Kontext Steuergeräte. Außerdem ist der Profibus eher für die Verwendung durch ein Steuergerät gedacht und nicht durch einen Laptop, wie im Fall des Projekts. Ein weiterer Nachteil von Profibus ist, dass dieser ein spezielles Kabel verwendet.

Zusammenfassend lässt sich also sagen, dass der Profibus zwar theoretisch geeignet ist, aber durch die in den Nachteilen genannten Eigenschaften unnötig komplex ist, um im Rahmen dieses Projekts verwendet zu werden.

5.1.3 TCP/IP

(Lea)

TCP/IP ist ein standardisiertes und weit verbreitetes Netzwerkprotokoll. Sowohl SPS-Systeme als auch Automatisierungssysteme unterstützen diese Art von Kommunikation. Außerdem ist es sehr flexibel auf verschiedenen physischen Netzwerken wie beispielsweise Ethernet oder WLAN-Protokoll- und geräteunabhängig. Über LAN kann dann der Roboter im lokalen Netzwerk überwacht werden. Durch Integration in Industrial Internet of Things (IIoT) und in Industrie 4.0 Anwendungen können Betriebsdaten in “Echtzeit” an Systeme gesendet werden und in der Theorie auch auf mehrere Geräte an einem Netzwerk.
Allerdings besitzt es kein Echtzeitprotokoll, da keine deterministischen Übertragungszeiten existieren und somit Verzögerungen und Latenzen verursacht werden können. Es herrschen Sicherheitsrisiken wie beispielsweise Cyberangriffe oder Möglichkeiten von unerlaubten Zugriffen. Durch Nutzung einer VPN-Verbindung lässt sich dies beheben. Sollte das Netzwerk überlastet sein oder ein erhöhter Datenverkehr herrschen, kann die Leistung beeinträchtigt werden und das Worst-Case-Szenario könnte zu Verlust von Datenpaketen führen. Es gibt keine Garantie der vollständigen Überlieferung aller Datenpakete und somit ist eine Beeinträchtigung der Steuerung erdenklich.
Daher ist die Nutzung einer physikalischen Ethernet-Verbindung empfehlenswert.

5.2 Problematik in der Kommunikation

(Matthias)

Letztendlich fanden wir auch nach langer Recherche und viel “Trial-and-Error ” keine funktionierenden Möglichkeiten, diesen speziellen Roboter über eine externe Schnittstelle anzusteuern. Es existiert höchstwahrscheinlich noch keine allgemeine/ in der Industrie verwendete und öffentlich zugängliche Lösung hierfür.
Ein Docker-Container, der die Kommunikation mittels dem Open-Source-Framework für die Entwicklung und Steuerung von Robotersystemen “ROS” ermöglichen sollte und öffentlich zugänglich war, war letztlich nur spezifisch für einen anderen Mitsubishi-Roboter kompatibel.
Zudem versuchten wir unter anderem auch MELFA Basic - Code in einem Python-Code in Visual Studio Code zu verpacken und diesen direkt an den Roboter zu versenden. Auch dies funktionierte nicht, da der Roboter nicht in der Lage war, diesen zu lesen und zu verwerten.
Wie bereits im Kapitel zu CIROS erwähnt, konnte man den Roboter zwar mittels CIROS steuern, allerdings erlaubt dieses Programm keine Schnittstelle zu anderen Programmiersprachen, so dass es für unsere Zwecke ungeeignet war.

Es stellte sich im Verlauf des Projekts immer klarer heraus, was wir befürchtet hatten: Eine Schnittstelle zur externen Ansteuerung des Roboters müsste komplett von null auf neu entwickelt werden. Dies hätte den Rahmen unseres Projekts deutlich gesprengt, könnte jedoch eine Idee für spätere Projekte in diesem Bereich sein, wenn diese von Beginn an wissen, worauf sie hinarbeiten. Daher entschieden wir uns für die konzeptuelle Schnittstelle, so dass unser Projekt schon sämtliche Grundlagen für spätere Bearbeiter erstellt und diese “nur” noch den Anschluss an den Roboter herstellen müssen.

5.3 Konzeption einer Schnittstelle

(Luca)

Das vorliegende Konzept beschreibt die Schnittstelle zwischen einer Bilderkennungssoftware und einem Roboter-Controller. Ziel dieses Systems ist die Identifizierung von Formen auf einem Bild, die Bestimmung ihrer Koordinaten sowie die Weitergabe dieser Informationen an einen Roboter, der die erkannten Objekte greift und an eine vordefinierte Zielposition setzt. Die Kommunikation zwischen der Bilderkennung und dem Roboter erfolgt über eine Schnittstelle, die eine reibungslose Datenübertragung sowie eine zuverlässige Steuerung des Roboters gewährleistet.

Die Entwicklung dieser Schnittstelle erfolgte als Konzept mit dem Ziel, eine Alternative zur Ansteuerung per Docker-Container zu bieten. Die gewählte Herangehensweise ermöglicht eine direkte Interaktion zwischen der Bilderkennungssoftware und dem Roboter, ohne dass eine zusätzliche Virtualisierungsschicht genutzt werden muss.

Die Schnittstelle setzt sich aus drei Hauptkomponenten zusammen: der Bilderkennung, der Schnittstelle selbst und dem Roboter-Controller. Die Bilderkennung identifiziert Objekte im Bild und bestimmt die entsprechenden Koordinaten. Die Schnittstelle konvertiert die erkannten Bildkoordinaten in das vom Roboter verarbeitbare Koordinatensystem und gewährleistet die Kommunikation zwischen Bilderkennung und Roboter. Der Roboter-Controller empfängt die Koordinaten, wandelt sie in Joint-Werte um und führt die Greif- und Bewegungsbefehle aus.

Das System arbeitet sequentiell. Zunächst wird durch die Schnittstelle eine Verbindung mit dem Roboter-Controller hergestellt, woraufhin die Software das Bild analysiert und relevante Objekte erkennt. Die so ermittelten Positionen in Bildkoordinaten werden in das Koordinatensystem des Roboters umgerechnet. Ein vorbereiteter Steuerbefehl in MELFA-BASIC wird mit den umgerechneten Koordinaten ergänzt und an den Roboter-Controller übermittelt. Der Roboter verarbeitet das Skript, führt die erforderlichen Bewegungen aus und bestätigt die erfolgreiche Durchführung. Falls weitere Objekte erkannt wurden, fordert das System neue Koordinaten aus der Bilderkennung an. Dieser Prozess wird wiederholt, bis keine neuen Objekte mehr vorhanden sind.

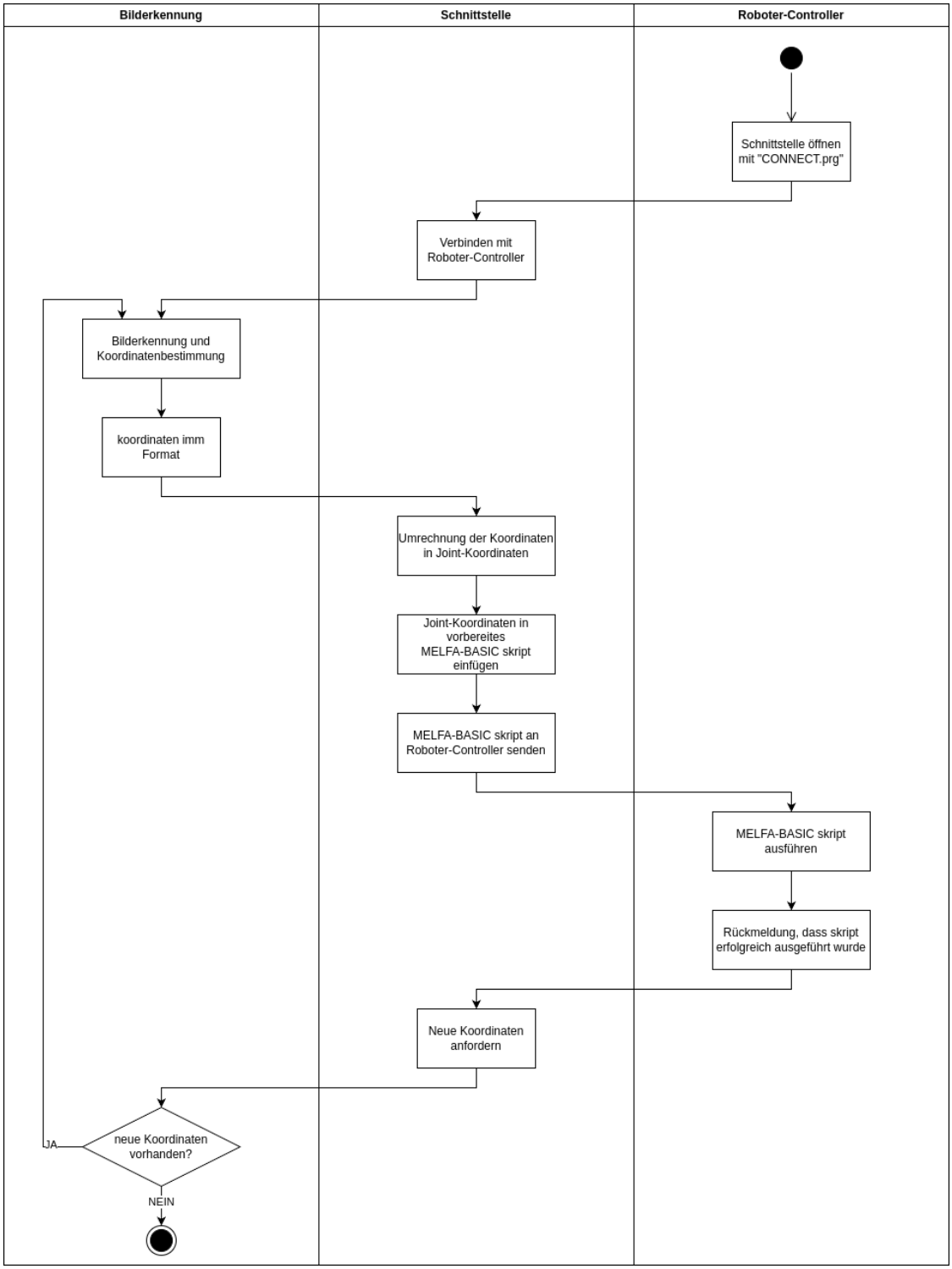

Abb. 14: Konzept einer Schnittstelle zwischen dem Bilderkennungsalgorithmus und dem Controller des Roboters

Die Kommunikation zwischen der Schnittstelle und dem Roboter erfolgt mittels eines MELFA-BASIC-Skripts, welches die Bewegungs- und Greifbefehle enthalten. Diese Skripte werden über eine serielle oder Netzwerkschnittstelle an den Roboter gesendet, wodurch eine direkte und zuverlässige Steuerung ermöglicht wird.

Das vorgestellte Konzept bietet verschiedene Vorteile, wie die Automatisierung des gesamten Prozesses von der Bilderkennung bis zur Robotersteuerung sowie die Minimierung manueller Eingriffe. Zudem ist das System durch seine modulare Struktur flexibel und kann für verschiedene Anwendungsfälle angepasst werden, was die Effizienz durch die direkte Kommunikation zwischen der Bilderkennung und dem Roboter steigert, da es zu Reduktion von Verzögerungen und Beschleunigung von Arbeitsabläufen kommt.

Die entwickelte Schnittstelle stellt eine effektive Lösung zur Integration von Bilderkennung und Robotersteuerung dar. Gemäß dem zugrundeliegenden Konzept ist es dem Roboter möglich, autonom auf visuelle Informationen zu reagieren und Objekte präzise zu platzieren. Das Konzept findet Anwendung in industriellen Anwendungen, beispielsweise in der automatisierten Montage oder Sortierung. Es ist jedoch zu konstatieren, dass der Ansatz der Schnittstelle noch einer Testung unterzogen werden muss, um seine Praxistauglichkeit und Leistungsfähigkeit zu validieren.

6 Fazit

In unserem Projekt bestand unsere Motivation darin, einen Roboter so zu programmieren, dass dieser Komponenten mit einer synthetischen Zeichnung abgleicht und diese präzise mittels Pick-and-Place auf einer Fläche positioniert.

In diesem Rahmen war es mir möglich, meine Fähigkeiten in Python zu vertiefen und erste Erfahrungen mit OpenCV sammeln. Als besonders wertvoll empfand ich es, theoretisches Wissen aus Vorlesungen praktisch anzuwenden und Parallelen zu realen Prozessen aus der Arbeitswelt wiederzufinden.

Auch wenn die Kommunikation mit der Steuerung des Roboters sich als eine zentrale Herausforderung entpuppte und uns vor unerwartete Probleme stellte, wurde deutlich, dass nicht immer alles perfekt funktioniert und Lösungen über geeignete Workarounds gefunden werden können. Die Entwicklung erforderte oft einen Trial-and-Error-Ansatz. Dies zeigte mir, wie wichtig fortlaufende Verbesserungen in einem Projekt sind.

Abschließend konnten wir wertvolle Erkenntnisse für zukünftige Teams gewinnen. Der Ausblick auf Optimierungsmöglichkeiten und ein Konzept einer Schnittstelle bieten eine gute Grundlage für die Weiterentwicklung des Projekts.
- Lea

Das Projekt hatte als Ziel einen Algorithmus zu bauen, der in einem Bild ein Bauteil identifizieren kann und dieses Bauteil in einem weiteren Schritt durch einen Roboter an die korrekte Position verschoben wird. Uns ist es gelungen, einen Algorithmus zu entwickeln und zu implementieren, der die Teile erkennt und die Zielposition ausgibt. Der Teil der Roboteransteuerung ist uns nicht wie gewünscht gelungen. Wir konnten allerdings ein Konzept für eine Schnittstelle entwickeln. Trotzdem war das Projekt für mich aus mehreren Gründen ein großer Erfolg.

Zum einen wurde mir wieder klar, wie wichtig gutes Projektmanagement für so ein großes Projekt ist. Wenn zu Beginn das Projektmanagement vernachlässigt wird, benötigt man unglaublich viel Energie, das später wieder nachzuholen. Das konnten wir uns glücklicherweise ersparen.
Trotz mehrerer Rückschläge in Sache Roboteransteuerung haben wir den Ansatz mit einem Docker-Container weiterverfolgt. Mir hat es zwar nicht immer Spaß gemacht, ich bin aber der Meinung, dass Ausdauer ein wichtiger Skill für das spätere Berufsleben ist. Diesen Skill konnte ich in diesem Projekt weiter trainieren.

Zusätzlich zu dem Learning der Ausdauer habe ich mich selbst im Bereich Docker weiterentwickeln können. Wir waren anfangs unsicher, ob der Ansatz, den Roboter per Docker-Container anzusteuern, für uns sinnvoll ist, weil wir alle keine Erfahrung im Bereich Docker hatten. Ich bin im Nachhinein dennoch sehr froh, dass wir diesen Ansatz gewählt haben. Ich bin der Meinung, dass Docker für mein Berufsleben in der Informatikbranche sehr wichtig ist.

Mit diesem Projekt ist für mich persönlich wieder einmal klar geworden, dass mir Projekte viel Spaß machen und ich dort auch immer viel Praktisches lernen kann.
- Luca

Das ursprüngliche Ziel des Projektes war es, den sechsachsigen Roboter dazu zu bringen, die uns vorgegebenen Teile selbstständig zu erkennen, anzuheben und an ihre Zielposition zu bringen. Dementsprechend wurde das Projekt in zwei Teilaufgaben aufgeteilt; nämlich die Entwicklung der Objekterkennung und die Entwicklung einer Schnittstelle zwischen der Objekterkennung und des Roboters.

Die Entwicklung der Objekterkennung verlief erfolgreich. Es ist uns gelungen, die Teile zu erkennen, klassifizieren und die für die anschließende Bewegung der Teile benötigten Daten zu ermitteln; nämlich um welches Teil es sich handelt, den Greifpunkt und die Rotation des Teiles.

Die Entwicklung der Schnittstelle verlief anders als ursprünglich erwartet, jedoch nicht erfolglos, wenn auch nicht ganz erfolgreich. Die Entwicklung einer Schnittstelle für den Roboter hätte den Rahmen dieses Projektes schlichtweg gänzlich überschritten. Stattdessen konnten wir im Verlauf des Projektes Untersuchungen über verschiedene Arten der Kommunikation mit dem Roboter anstellen, deren Ergebnisse den nachfolgenden Gruppen bei der Entwicklung einer Schnittstelle helfen sollen.

Bei der Mitentwicklung der Objekterkennung konnte ich das aus den Vorlesungen im Masterstudium bereits erlangte Wissen einsetzen, was mir bei den Herausforderungen erlaubte, mein Verständnis über die erlernten Konzepte durch praktische Anwendung zu vertiefen. Darüber hinaus stellte die Zusammenarbeit mit meinen Mitstudenten eine lehrreiche Erfahrung dar.
- Alex

Auch wenn wir nicht alles geschafft haben, was wir uns Anfangs vorgenommen haben, war das Projekt eine wertvolle Erfahrung. Ich konnte viele neue Methoden und Verfahren lernen, aber auch Wissen aus den Vorlesungen gleich an einer praktischen Aufgabe anwenden. Trotz einiger Schwierigkeiten gab es doch immer wieder auch Erfolgserlebnisse. Zusammenfassend lässt sich sagen, dass es eine interessante Erfahrung war, an dem Projekt zu arbeiten.
- Christian

Es hat sich herauskristallisiert, dass ein Projekt eben nicht immer nur steil bergauf verlaufen muss, sondern sowohl “Ups” als auch “Downs” haben kann. Auch wenn es schade ist, dass sich die Schnittstelle als nicht realisierbar erwies, war es trotzdem spannend zu sehen, wie es uns in anderen Bereichen gelang, Fortschritte zu erzielen und letztlich ein gut durchdachtes Konzept zu entwickeln, um die Idee hinter dem Projekt zu realisieren.
Zudem hat sich wieder bewiesen, wie wichtig gutes Projektmanagement und eine klare Verteilung und Definition von Aufgaben ist. Ich denke, dass wir die Rollen in unserem Team nicht hätten besser aufteilen können und dass jeder sehr gut in seinem Bereich aufgehoben war. Insgesamt war das Projekt sehr lehrreich für mich und ich bin gespannt, wie es sich vielleicht in den kommenden Semestern mit anderen Teams noch weiterentwickeln wird.
- Matthias

7 Abbildungsverzeichnis

8 Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.
Ich übernehme als Autorin oder Autor beim Einsatz von IT-/KI-gestützten Schreibwerkzeugen die Verantwortung für die inhaltliche Richtigkeit.

Datum 13.02.2025 Unterschrift

Lea Kuznik 

Luca Neuburger 

Alex Wagner 

Christian Weiß 

Matthias Wörz