RFS-Projekt 2023

Ansteuern eines Mitsubishi RV-3SB und sortieren von Formen mit Bilderkennung

Gruppe 1



|  |  |
| --- | --- |
| Oliver Steck  Hutteneichenweg 10  70565 Stuttgart | Markus Kellner  Keine Ahnung  Irgendwas Böblingen |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc128667041)

[1.1 Ziel des Projektes 3](#_Toc128667042)

[1.2 Erwartete Herausforderungen 3](#_Toc128667043)

[1.3 Beschaffung der benötigten Ressourcen 3](#_Toc128667044)

[2 Technologien 5](#_Toc128667045)

[2.1 Python 5](#_Toc128667046)

[2.2 Bonus: Java 5](#_Toc128667047)

[3 Python-API 6](#_Toc128667048)

[3.1 Kommunikation mit dem Roboter 6](#_Toc128667049)

[3.2 Ansteuern des Roboters über den Browser 6](#_Toc128667050)

[4 Bilderkennung 7](#_Toc128667051)

[4.1 Erkennen von Formen 7](#_Toc128667052)

[4.2 Koordinatentransformationsmatrix 7](#_Toc128667053)

[5 Kommunikation mit dem Frontend 8](#_Toc128667054)

[6 Bonus: Java-API 9](#_Toc128667055)

[6.1 Initialisieren und Ansteuern des Roboters 9](#_Toc128667056)

[6.2 Erstellen von Programmen und Unterprogrammen 11](#_Toc128667057)

[6.3 Betrieb im Online-Modus 11](#_Toc128667058)

[6.4 Demonstrationsprogramm ADV Jubiläum 12](#_Toc128667059)

[7 Fazit 14](#_Toc128667060)

[7.1 Bewertung der erwarteten Herausforderungen 14](#_Toc128667061)

[7.2 Ausblick 14](#_Toc128667062)

# Einführung

## Ziel des Projektes

Als Ziel des Projektes wurde festgelegt, dass die entwickelte Anwendung Werkstücke der Form nach sortieren und in dafür vorgesehene Behälter bringen soll. Dafür sollte ein Backend, eine Bilderkennung und eine Schnittstelle zu einem Frontend, welches von einer anderen Gruppe erstellt werden sollte, entwickelt werden. Sowohl das Backend mit Schnittstelle als auch die Bilderkennung sollten mit Python programmiert werden.

## Erwartete Herausforderungen

Die Herausforderungen beschränkten sich in diesem Projekt auf folgende Hauptpunkte:

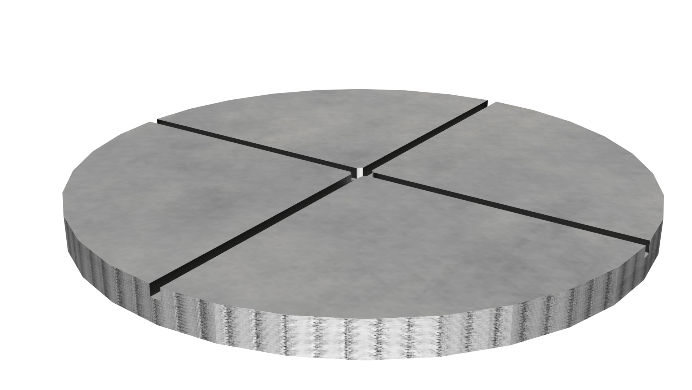
1. Verbindung mit dem Roboter
2. Kommunikation mit dem Roboter
3. Integration der Bilderkennung
4. Kommunikation mit dem Frontend

## Beschaffung der benötigten Ressourcen

Als Ressourcen für das Projekt wurden Kreise und Rechtecke benötigt, wobei an dieser Stelle die Hilfe der 3D-Drucker-Gruppe benötigt wurde. Zunächst wurden einfache Kreise mit einem Durchmesser von fünf Zentimetern und einer Höhe von drei Millimetern gedruckt, ebenso wie gleichhohe Quadrate mit einer Breite von fünf Zentimetern. Als diese mit der Vakuumpumpe getestet wurden, fiel schnell auf, dass etwas angepasst werden muss, da die Werkstücke nach Abschalten der Pumpe noch lange an der Saugvorrichtung hingen, wegen des Unterdrucks.

Als Lösung wurde ein kleines (zwei Millimeter Durchmesser breites) Loch in die Mitte der Formen gebohrt. Die Teile vielen nun zwar ab, hatten jedoch das große Problem, dass die Arbeitsfläche unter den Werkstücken ebenfalls angesaugt wurde. Das bedeutet, dass man auf diese Weise wahrscheinlich die Plexiglasfläche beschädigt oder vollkommen zerstört hätte.

Auch hier wurde eine Lösung gefunden: Dass sichergestellt ist, dass die Werkstücke angesaugt werden, die darunterliegende Arbeitsfläche unangetastet bleibt und sich die Formen nach Abschalten der Pumpe wieder lösen, wurden Kanäle an die Unterseite der Objekte angelegt. Diese gewährleisteten eine Luftzufuhr nach außen, auch wenn die Formen flach auf einer Oberfläche liegen. Gleichzeitig ließen sie aber gemeinsam mit dem Loch in der Mitte nicht zu viel Luft durch, sodass die Werkstücke trotzdem angehoben werden konnten.



Ein Bild, das Text, Visitenkarte enthält.

Automatisch generierte Beschreibung

# Technologien

## Python

## Bonus: Java

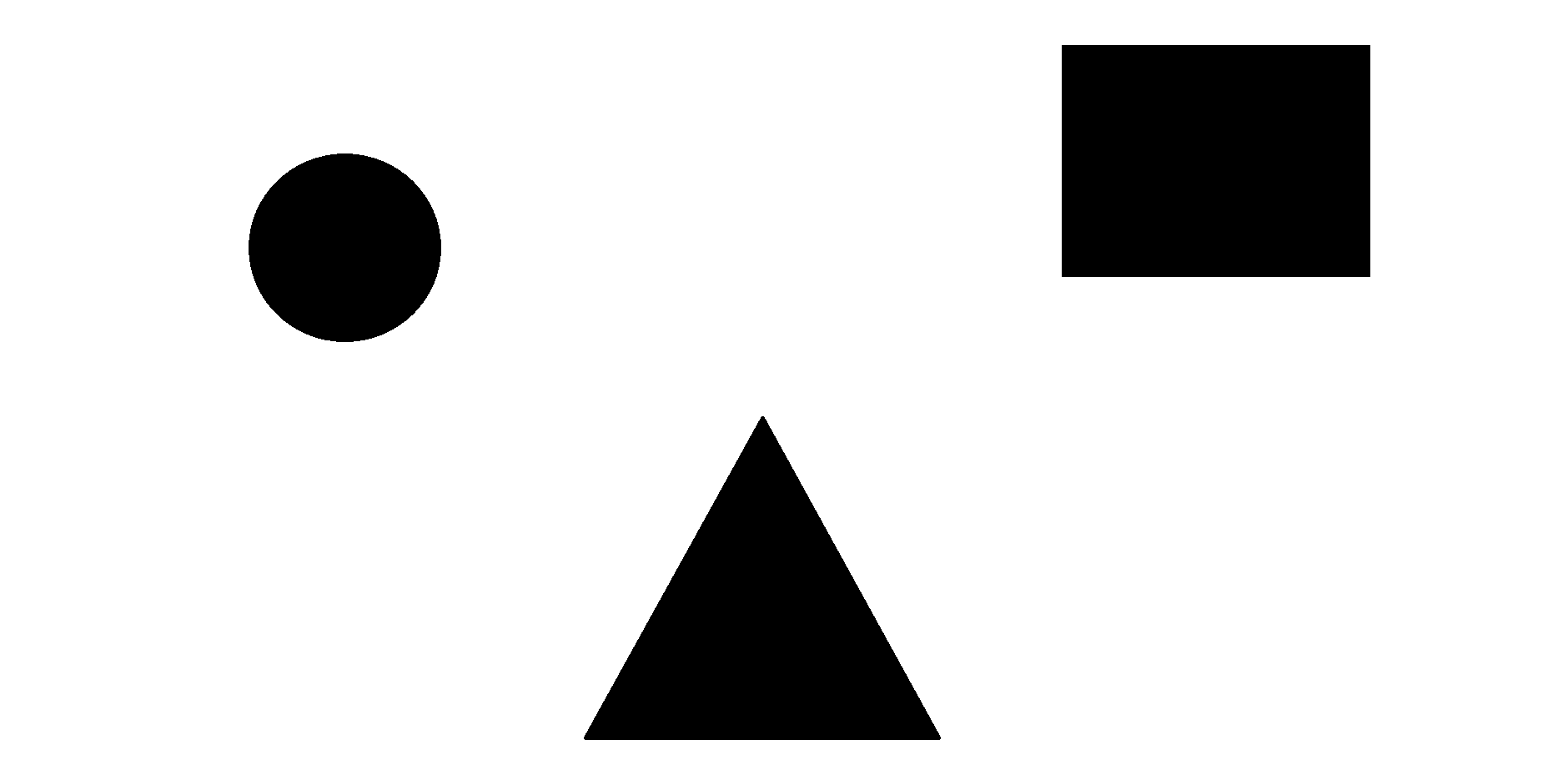
# Python-API

## Kommunikation mit dem Roboter

## Ansteuern des Roboters über den Browser

# Bilderkennung

## Erkennen von Formen



## Koordinatentransformationsmatrix

# Kommunikation mit dem Frontend

# Bonus: Java-API

Als Bonus (sozusagen als kleines ‚Schmankerl‘) wurde in diesem Projekt auch eine Java-API (‚RobotAPI‘) geschrieben. Das Ziel war es eine sogenannte Utility-Bibliothek zu entwickeln, welche entweder als gebaute Jar-Datei oder komplett in den eigenen Quellcode importiert werden kann. Sie ist nicht als REST-Anwendung gedacht wie die Python-API, sondern dient lediglich der Ansteuerung des Roboters und kann einfach ins eigene Projekt integriert werden.

Genutzt wurde dabei entfernt die von der Lehrkraft als Vorlage gegebene Java-Anwendung, die vor einigen Jahren Inhalt eines ähnlichen Projektes war. Alles, was davon genutzt wurde, war allerdings nur die Methode, die Daten an den Roboter schickt und sie empfängt. Der ganze Rest der API wurde von der diesjährigen Gruppe selbstständig programmiert und hat nichts mehr mit dem Altprojekt zu tun.

Zu finden ist diese API mitsamt Erklärung unter: *https://github.com/TecuilaCat/RobotAPI.git*.

## Initialisieren und Ansteuern des Roboters

Für das Initialisieren eines neuen Roboter-Objektes wurde eine Klasse ‚RobotBuilder‘ erstellt, die es erlaubt Funktionen zu konfigurieren, die dann intern sortiert und beim Start in der richtigen Reihenfolge ausgeführt werden. Dies soll sicherstellen, dass der Roboter beim Start immer korrekt konfiguriert und an derselben sicheren Position steht.

Zunächst muss ein Host und ein Port mitgegeben werden, mit dem sich die API später verbinden kann. Beim Host handelt es sich um die IP-Adresse des Controllers, der sich im logischerweise im gleichen Netzwerk befinden muss. Des Weiteren braucht der Roboter ein Sprachpaket, welches im Falle des RV-3SB MelfaBasic4 ist. Danach werden die Operationen teils optional. Entweder man will den Roboter sofort ohne weitere Konfigurationen benutzen oder man konfiguriert munter weiter.

Im ersten Fall muss man dann die Methode ‚disableSecureStartup()‘ aufrufen, die alle weiteren Konfigurationen deaktiviert. An dieser Stelle ist ein sicheres Starten nicht gewährleistet. Im zweiten Fall können beliebig viele Operationen aufgerufen werden, wobei die Reihenfolge egal ist, da sie später beim Bauen des Objektes sortiert werden.

Das fertige einsatzfähige Roboter-Objekt erhält man mit der Methode ‚build()‘. Ein entsprechendes Konstrukt könnte wie folgt aussehen:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | Robot robot = new RobotBuilder(HOST, PORT)   |  |  | | --- | --- | |  | .setSafePosition(SAFE\_POSITION)  .setCommandSet(MelfaBasic4CommandSet.getCommandSet())  .enableCommunication()  .enableOperation()  .enableServo()  .setSpeed(25)  .setName(“RV-3SB”)  . build(); | |

Hier wird nun zuerst die Verbindung zum physischen Roboter hergestellt, dann die Servo angeschaltet, die Geschwindigkeit auf fünfundzwanzig gesetzt und letzten Endes der Roboter zu einer vordefinierten sicheren Position ‚SAFE\_POSITION‘ gefahren. Von dort aus ist der Roboter einsatzbereit. Auf diesem Objekt ‚robot‘ können jetzt allerlei Operationen, welche vom Interface ‚RobotOperations‘ definiert werden, ausgeführt werden:

|  |  |
| --- | --- |
| enableServo() | Schaltet die Servo an |
| disableServo() | Schaltet die Servo aus |
| setSpeed(int speed) | Setzt die Geschwindigkeit auf einen neuen Wert |
| getCurrentPosition() | Gibt die momentane Position des Roboters zurück |
| getState() | Gibt den Status des Roboters zurück |
| movToPosition(Position pos) | Fährt den Roboter an die mitgegebene Position mit einem regulären MOV-Befehl |
| mvsToPosition(Position pos) | Fährt den Roboter an die mitgegebene Position mit einem regulären MVS-Befehl |
| movToSafePosition() | Fährt den Roboter an die vorkonfigurierte sichere Position |
| executeCustomCommand(String cmd) | Führt einen manuell eingegebenen Befehl aus |
| setSafePosition(Position pos) | Setzt eine neue sichere Position |
| runProgram(RunnableProgram rp) | Führt ein Programm aus (Erklärung im nächsten Kapitel) |
| grab() | Schaltet die Vakuumpumpe an |
| drop() | Schaltet die Vakuumpumpe aus |

## Erstellen von Programmen und Unterprogrammen

Um das Erstellen und den Umgang mit Programmen sowie Unterprogrammen so einfach wie möglich zu gestalten, wurde diese Funktionalität über das Interface ‚RunnableProgram‘ definiert. Damit eine Klasse die Anforderungen eines lauffähigen Programmes erfüllt, muss es dieses Interface implementieren, was die Klasse dazu zwingt die Methode ‚runProgram(RobotOperation ops)‘ zu überschreiben. Diese Methode kann auf einem Objekt der Klasse mit dem entsprechenden Roboter als Parameter ausgeführt werden. Mit diesem Konstrukt lassen sich auch verschachtelte Programme realisieren.

Gleichzeitig kann ein Roboter aber auch ein Programm aufrufen, also genau andersrum. Ein solcher Zugriff kann über die Methode ‚runProgram(RunnableProgram program)‘ geschehen. Auf diese Weise können Programme auf einfachste Art aufgerufen werden, was den Code schmaler, einfacher und leichter zu lesen gestaltet.

## Betrieb im Online-Modus

Im Rahmen des Projektes wurde in der RobotAPI auch ein Online-Modus erstellt, der die Ansteuerung des Roboters konsolenähnlich gestaltet. Wie in den Terminals von Linux oder Windows kann man Befehle direkt in eine Konsole eingeben.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

An dieser Stelle muss der Benutzer zum jetzigen Zeitpunkt allerdings noch wissen wie die Syntax der Befehle lautet. Vor allem bei den Befehlen ‚EXECMOV‘ und ‚EXECMVS‘ kann dies ein kleines Problem darstellen. An dieser Funktionalität wird allerdings noch gearbeitet.

## Demonstrationsprogramm ADV Jubiläum

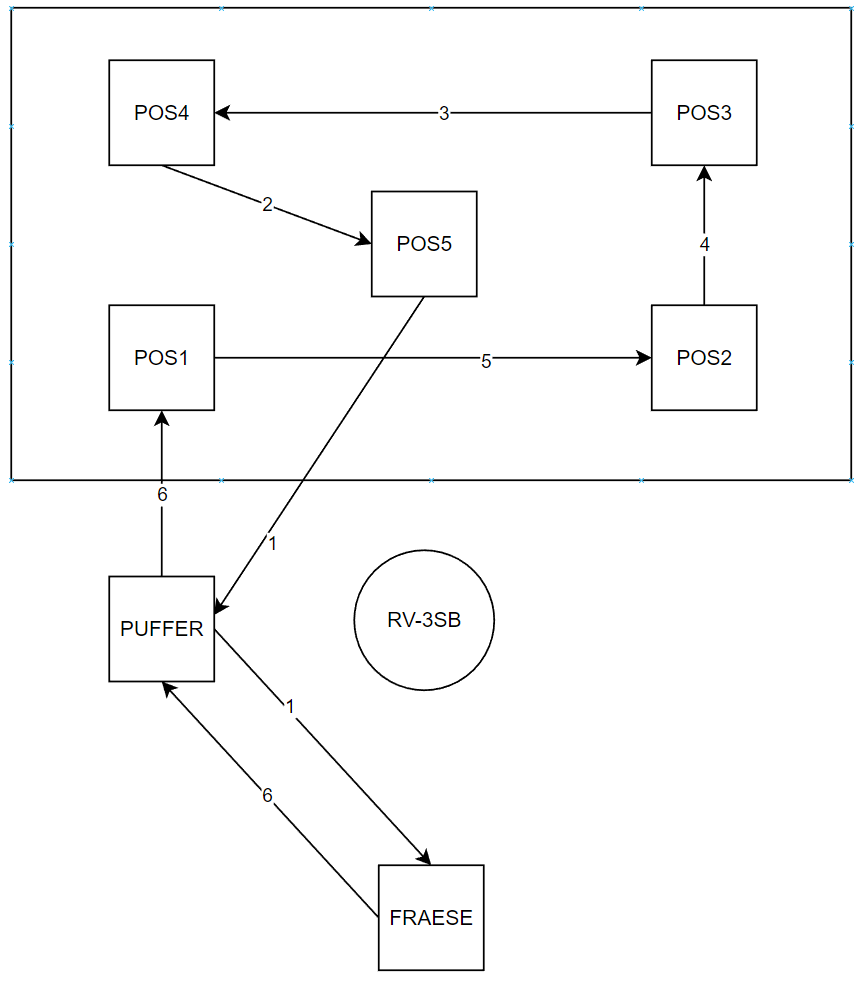
Für das Jubiläum der ADV am 16. März 2023 wurde mit der RobotAPI ein Demonstrationsprogramm geschrieben, welches die Arbeit am Roboter veranschaulichen sollte. Dieses beinhaltet die Simulation eines Arbeitsvorgangs, bei dem Werkstücke verschiedene Stationen durchlaufen, bevor sie zur Fräse gebracht werden. Insgesamt gibt es fünf Stationen und fünf Formen. Während ein Werkstück bei der Fräse verweilt, werden die anderen wie auf einem Band nachgerückt. Anschließend wird das verbleibende Werkstück wieder auf die erste Position gelegt. Daraus ergibt sich dann eine Art Warteschlange.

Zunächst wurden die benötigten Punkte hart implementiert, da zu diesem Zeitpunkt die Bilderkennung noch nicht integriert war. Diese wurden mühsam Schritt für Schritt live am Teachpanel des Roboters, beziehungsweise über RT-ToolBox3, herausgefunden und die Werte übertragen. So ergaben sich am Ende acht Positionen (POS1 – POS5, PUFFER, PICKUP, FRAESE). Der Puffer stellt sicher, dass der Roboter auf dem Weg von Arbeitsfläche zu Fräse nicht irgendwo hängen bleibt. Diese Position wird sowohl auf dem Hin- als auch auf dem Rückweg angefahren.

Das Hauptprogramm wurde so konzipiert, dass es in einer Endlosschleife laufen kann:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | public class MainProgram implements RunnableProgram {   |  |  |  |  | | --- | --- | --- | --- | |  | @Override  public void runProgram(RobotOperations ops) {   |  |  | | --- | --- | |  | ops.runProgram(new ToFraese());  ops.runProgram(new Rearrange());  ops.runProgram(new FromFraese()) |   } |   } |

Das Programm ‚ToFraese‘ transportiert das Werkstück an der letzten Position zur Fräse. Das Programm ‚FromFraese‘ transportiert es von der Fräse zur ersten Position. Das dritte dazwischen ausgeführte Programm ‚Rearrange‘ ordnet die verbleibenden Werkstücke auf der Arbeitsfläche neu und simuliert somit das Band. Daraus ergibt sich folgendes Schaubild:



Die Pfeile bedeuten jeweils, dass ein Teil von Pfeilanfang nach Pfeilende transportiert wird. Die Zahlen auf den Pfeilen beschreiben die Reihenfolge, in der dies geschieht.

# Fazit

## Bewertung der erwarteten Herausforderungen

## Ausblick