RFS-Projekt 2023

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

Gruppe 1



|  |  |
| --- | --- |
| Oliver Steck  Hutteneichenweg 10  70565 Stuttgart | Markus Kellner  Goerdelerstraße 10  71034 Böblingen |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc128955649)

[1.1 Ziel des Projektes 3](#_Toc128955650)

[1.2 Erwartete Herausforderungen 3](#_Toc128955651)

[1.3 Beschaffung der benötigten Ressourcen 4](#_Toc128955652)

[2 Technologien 5](#_Toc128955653)

[2.1 Python 5](#_Toc128955654)

[2.2 Bonus: Java 5](#_Toc128955655)

[3 Python-API 6](#_Toc128955656)

[3.1 Kommunikation mit dem Roboter 6](#_Toc128955657)

[3.2 Ansteuern des Roboters 7](#_Toc128955658)

[4 Bilderkennung 9](#_Toc128955659)

[4.1 Erkennen von Formen 9](#_Toc128955660)

[4.2 Koordinatentransformationsmatrix 10](#_Toc128955661)

[5 Kommunikation mit dem Frontend 12](#_Toc128955662)

[6 Bonus: Java-API 13](#_Toc128955663)

[6.1 Initialisieren und Ansteuern des Roboters 13](#_Toc128955664)

[6.2 Erstellen von Programmen und Unterprogrammen 15](#_Toc128955665)

[6.3 Betrieb im Online-Modus 15](#_Toc128955666)

[6.4 Demonstrationsprogramm ADV Jubiläum 15](#_Toc128955667)

[7 Fazit 18](#_Toc128955668)

[7.1 Bewertung der erwarteten Herausforderungen 18](#_Toc128955669)

[7.2 Ausblick 19](#_Toc128955670)

[8 Quellen 21](#_Toc128955671)

# 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

**Verbinden mit dem Roboter:** Eine große Hauptsorge war, dass die Verbindung nicht zustande kommt oder unterbricht. Genau genommen war das Wissen darüber, wie man den Controller anspricht überhaupt nicht gegeben. Daher kam eine gewisse Sorge, ob es überhaupt funktioniert.

**Kommunikation mit dem Roboter:** Neben der Verbindung mit dem Roboter ergab sich gleichzeitig auch die Frage nach der Kommunikation. Wie kann man dem Roboter Befehle geben? In welcher Form erfolgt eine Antwort und wie wird man benachrichtig, wenn eine Bewegung vollendet ist oder gar schiefläuft, da zum Beispiel die Koordinate nicht angefahren werden kann?

**Integration der Bilderkennung:** Eine ursprüngliche Idee war es, eine externe Bilderkennung miteinzubinden. Eine, wie sich später herausstellte berechtigte, Sorge war es, dass dieses Programm nicht kompatibel mit der eigenen Anwendung oder im schlimmsten Fall gar nicht durch Code aufrufbar ist.

**Kommunikation mit dem Frontend:** Da das Frontend von einer anderen Gruppe erstellt wurde, mit der zwar oft kommuniziert wurde, aber das Potential von Falschinformationen in beide Richtungen zumindest gegeben ist, ergab sich die Gefahr, dass die beiden Anwendungen am Ende nicht zusammenpassen könnten.

## 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) 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.

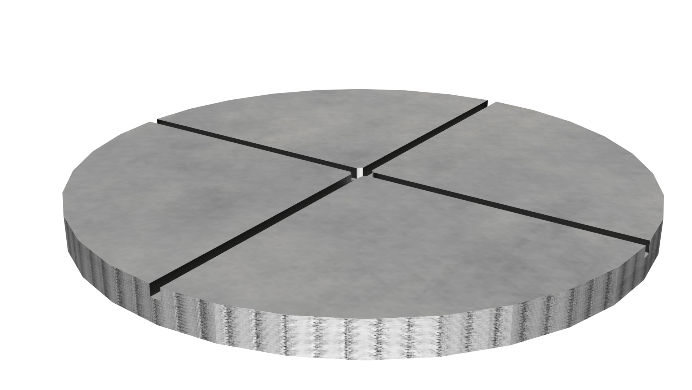
Ein Bild, das Text, Visitenkarte enthält.

Automatisch generierte BeschreibungAuch 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.

# Technologien

## Python

## Bonus: Java



# Python-API

Als Grundlage für die Verbindung, Kommunikation und Ansteuerung des Roboters wurde eine API mit Python programmiert. Enthalten sind wesentliche Positionen, wie ein sicherer Punkt, an den der Roboter zu einem beliebigen Zeitpunkt gefahren werden kann, sowie Routinen und Funktionen zum Steuern und Bewegen.

Insgesamt besteht die API aus einer steuernden Komponente sowie einem Webserver, der später bei der Kommunikation mit dem Frontend zum Tragen kommt.

## Kommunikation mit dem Roboter

In der Datei `robotControl.py` sind alle Funktionen rund um die Kommunikation und Steuerung des Roboters enthalten. Grundlage dieser Kommunikation ist eine Socket, die die Verbindung von Computer und Controller darstellt. Sie wird initialisiert mit einem Host und einem Port, wobei logischerweise beide im gleichen lokalen Netzwerk sein müssen. Dazu werden zwei Netzwerkkarten benötigt: eine für das Netzwerk, die andere für die Verbindung direkt zum Controller. Im Endeffekt muss man also zwei LAN-Verbindungen nutzen.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4 | def rc\_connect(self, host: str, port: int) -> None:   |  |  | | --- | --- | |  | print(f’Connecting to {host}:{port}…’)  self.s.connect((host,port))  print(‘Connected’) | |

Diese Socket wird genutzt, um einen Input- und einen OutputStream aufzubauen, über die man Daten senden und empfangen kann. Über den OutputStream können Daten als byte-Array gesendet werden. Dabei werden reguläre Strings versendet, in denen die Befehle als Klartext stehen. Das bedeutet, dass alle Funktionalitäten zuerst in Klartext übersetzt werden müssen.

Der InputStream ist dagegen für das Empfangen zuständig, indem man die Antwort in ein vorinitialisiertes byte-Array speichert und dann zu einem String konvertiert. Für das simultane Senden und Empfangen ergibt sich also folgende Funktion:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | def send\_and\_receive(self, command: str, buff\_size: int = 4096,  premade: bool = True) -> str:   |  |  | | --- | --- | |  | if premade:  self.s.sendall(f’1;1;{command}’.encode())  else:  self.s.sendall(f’{command}’.encode())  time.sleep(0.1)  return self.s.recv(buff\_size).decode() | |

Die Funktion ‘sendall‘, die auf der Socket s aufgerufen wird, schickt dabei alle Daten, also den formatierten String an den Controller. Bei diesem Controller ist es essentiell bei jedem Befehl und jeder Abfrage den Suffix ‘1;1;‘ hinzuzufügen.

## Ansteuern des Roboters

Zum tatsächlichen Bewegen des Roboters wurden verschiedene Funktionen für normale MOV-Befehle und MVS-Befehle geschrieben. Der zugehörige Befehl im Befehlssatz des Controllers lautet dabei ‘EXEMOV‘ und ‘EXECMVS‘. Davon gefolgt ist ein Vektor, der die neue Position relativ zur momentanen angibt. Steht der Roboter also gerade auf der Koordinate (0.0, 0.0, 0.0, …) und soll sich zur Position (100.0, 50.0, 0.0, …) bewegen, lautet der Befehl ‘EXECMOV P\_CURR+(100.0, 50.0, 0.0, …)‘. Dieser Vektor muss davor berechnet werden und ergibt sich aus der Differenz zwischen Soll- und Ist-Koordinate.

Eine Funktion, die einen dieser Befehle an den Controller schickt, sieht beispielsweise folgendermaßen aus:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def move\_pos(self, pos: Point, safe\_travel = True) -> None:   |  |  | | --- | --- | |  | if safe\_travel:  cp = point.difference(pos, self.get\_current\_pos())  cp.z = cp.z + 50  diff = cp  else:  diff = point.difference(pos, self.get\_current\_pos())  cmd = f’EXECMVS P\_CURR+({diff.to\_string())’  self.send(cmd, premade = False)  self.move\_pause() | |

Die große Schwierigkeit war es den Roboter dazu zu bringen Befehle sequentiell abzuarbeiten und auf vorherige Antworte, also die Vollendung vorhergegangener Operationen abzuwarten. Genau diese Schwierigkeit bügelt die Funktion ‘move\_pause‘ aus, indem hier in einer Schleife dauerhaft abgefragt wird, ob sich der Roboter in einer Bewegung befindet. Erst, wenn diese Funktion den Wert ‘False‘ zurückgibt, geht es mit der Abarbeitung der Programms weiter und der nächste Befehl kann gesendet werden. In jedem Fall, auch bei Befehlen jenseits von Bewegungen (also beispielsweiße das Bedienen der Pumpe), bedarf es einem Timeout, der auf eine viertel Sekunde gesetzt wurde.

Diese Zeit reicht dem Controller im lokalen Betrieb, um eine einfache Aktion oder Abfrage auszuführen. Fehlt dieser Timeout, werden Befehle zufällig übersprungen oder ignoriert und das gesamte Programm passt nicht mehr zusammen.

*(Die Java-API, die später beschrieben wird, funktioniert auf dieser Ebene auf dieselbe Weise)*

# Bilderkennung

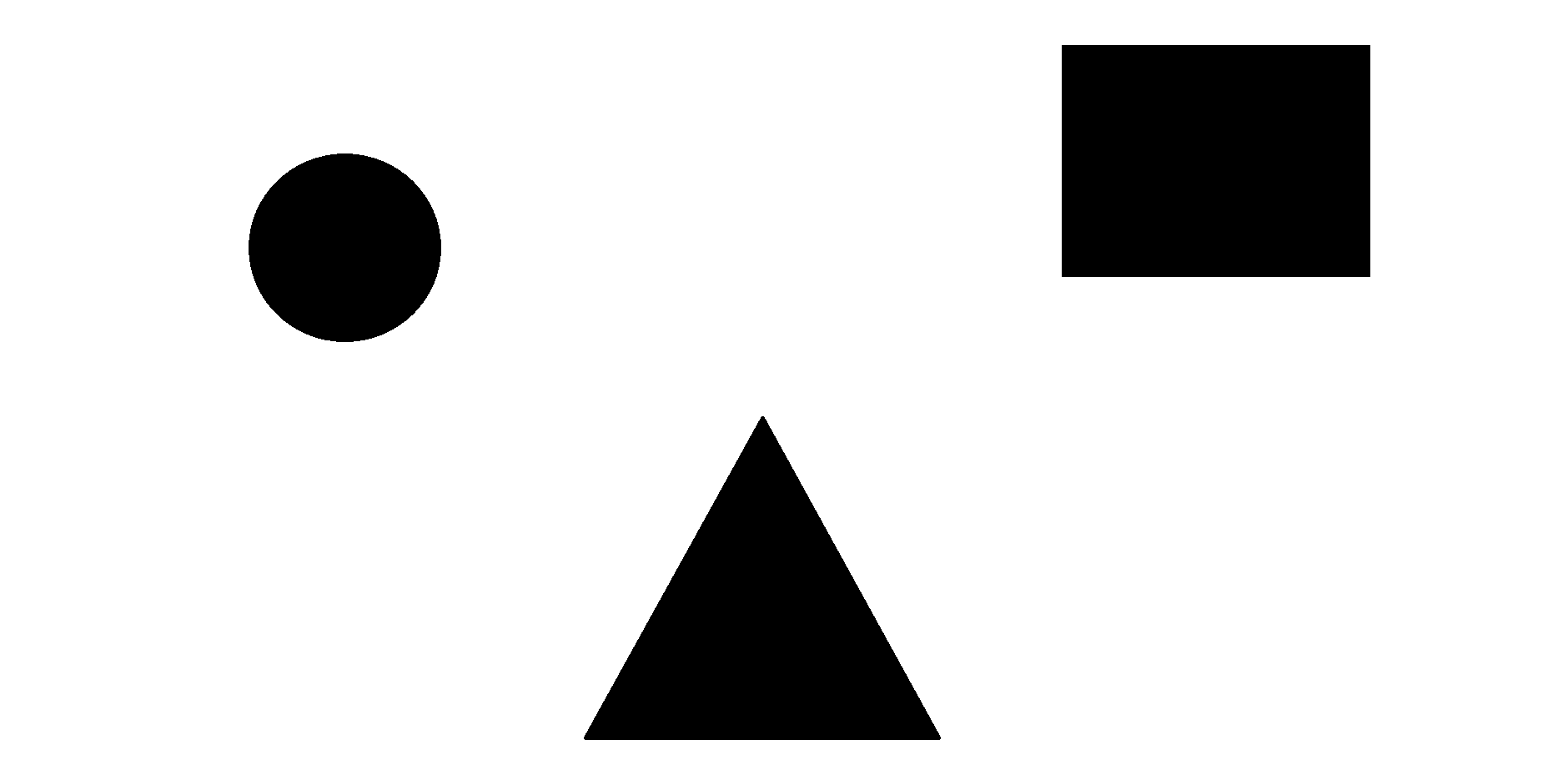
Eine essentielle Komponente des Projektes ist die Erkennung von Formen und Objekten, die auf der Arbeitsfläche des Roboters liegen. Im Programmablauf wird dauerhaft geprüft, ob sich entsprechende Formen auf der Fläche befinden, woraufhin sie klassifiziert werden und anschließend in Behältnisse an vordefinierten Positionen sortiert werden.

## Erkennen von Formen

Für das Erkennen von Formen wurde ein Python-Skript geschrieben, das mit der Bibliothek OpenCV Objekte anhand von Kanten und Ecken identifiziert.

Hierbei wird zunächst das Bild eingelesen und so bearbeitet, dass die Bilderkennung zuverlässig arbeiten kann. Was man als Test schlecht simulieren kann, aber leider Fakt ist, ist, dass die Kamera über dem Roboter aufgrund der doch sehr ungünstigen Hintergrundbeleuchtung Bilder liefert, auf denen Farbübergänge als Objekte identifiziert werden.

Um zu gewährleisten, dass die gefundenen Objekte tatsächlich Werkstücke sind, wird das Bild zuerst in schwarz-weiß konvertiert und dann werden die Farbübergänge eliminiert. Alle Pixel in einem gewissen dunklen Spektrum werden auf schwarz gesetzt, der Rest weiß. Daraus ergibt sich dann ein ähnliches Bild wie das folgende, das zum Testen des Skripts genutzt wurde.



Von hier an werden die Farbübergänge, die nun ausschließlich die Kanten von Formen sind, dazu genutzt, Koordinaten zu erfassen und Ecken zu zählen. Diese sind durch einen einfachen Methodenaufruf abrufbar. Anhand der Koordinaten der Ecken kann dann der Formmittelpunkt bestimmt werden. Das Skript gibt am eine Liste von Formen kategorisiert in Dreiecke, Kreise und Rechtecke mitsamt den Mittelpunkten zurück. Lässt man sich das Ergebnis grafisch ausgeben, entsteht folgendes Bild. Dies kann zum Kontrollieren der Bilderkennung genutzt werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

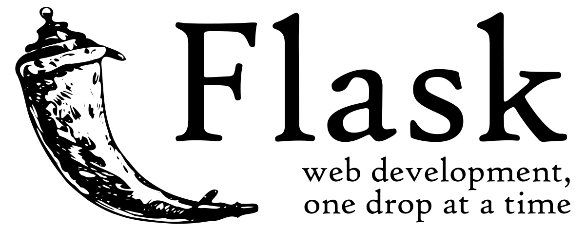
Zu erwähnen ist, dass diese Koordinaten relativ zur Bildgröße sind. Das bedeutet, dass die Punkte noch nicht den Koordinaten des Koordinatensystems des Roboters entsprechen und damit noch unbrauchbar sind. Um die Punkte aus der Bilderkennung nutzen zu können müssen diese erst in Punkte des Koordinatensystems konvertiert werden. Diese Konvertierung findet in einer entsprechenden Koordinatentransformationsmatrix statt, die im nächsten Kapitel beschrieben wird.

## Koordinatentransformationsmatrix

Zunächst war die Idee, dass ein externes Programm Formen erkennt und die Koordinaten der Mittelpunkte direkt als Punkte im Koordinatensystem des Roboters zurückgibt. Dazu wurden auch an einem Nachmittag in einem Termin mit einer externen Firma, die den Roboter einrichtet, verschiedene Möglichkeiten erörtert. Da offenbarte sich dann allerdings das große Problem, dass keines dieser Programme über Python abrufbar sei. Man müsse aktiv ein anderes Programm öffnen und manuell sich bis zu den gefundenen Koordinaten durchklicken.

Aus diesem Grund wurde dann nach der eigenen Bilderkennung eine Koordinatentransformationsmatrix eingeführt, die die gefundenen Koordinaten in ansteuerbar konvertieren sollte.

# Kommunikation mit dem Frontend

Die programmierte API mit Python kommt im Projekt mithilfe der Flask-API zum Einsatz. Sie stellt einen Webserver zur Verfügung, über welchen auf die benötigten Funktionen zugegriffen werden kann. Benötigt werden dabei sowohl Host als auch Port. Ein zugriff funktioniert per Link und könnte zum Beispiel folgendermaßen aussehen: http://127.0.0.1:5000/servoon. Dieser Befehl würde beispielsweiße die Servo des Roboters anschalten.

Das Frontend kann dann einfach über einen REST-Aufruf entweder Befehle an das Backend schicken, oder sich von dort Informationen holen. Nachdem der Webserver gestartet hat, hängt also die gesamte Kommunikation von den Aktionen der Frontend-Komponente ab. Backendseitig stehen alle Funktionen jederzeit zur Verfügung. Grundlage des Informationsaustauschs sind JSON- oder XML-Formate.

# 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, Tisch enthält.

Automatisch generierte Beschreibung

## Demonstrationsprogramm ADV Jubiläum

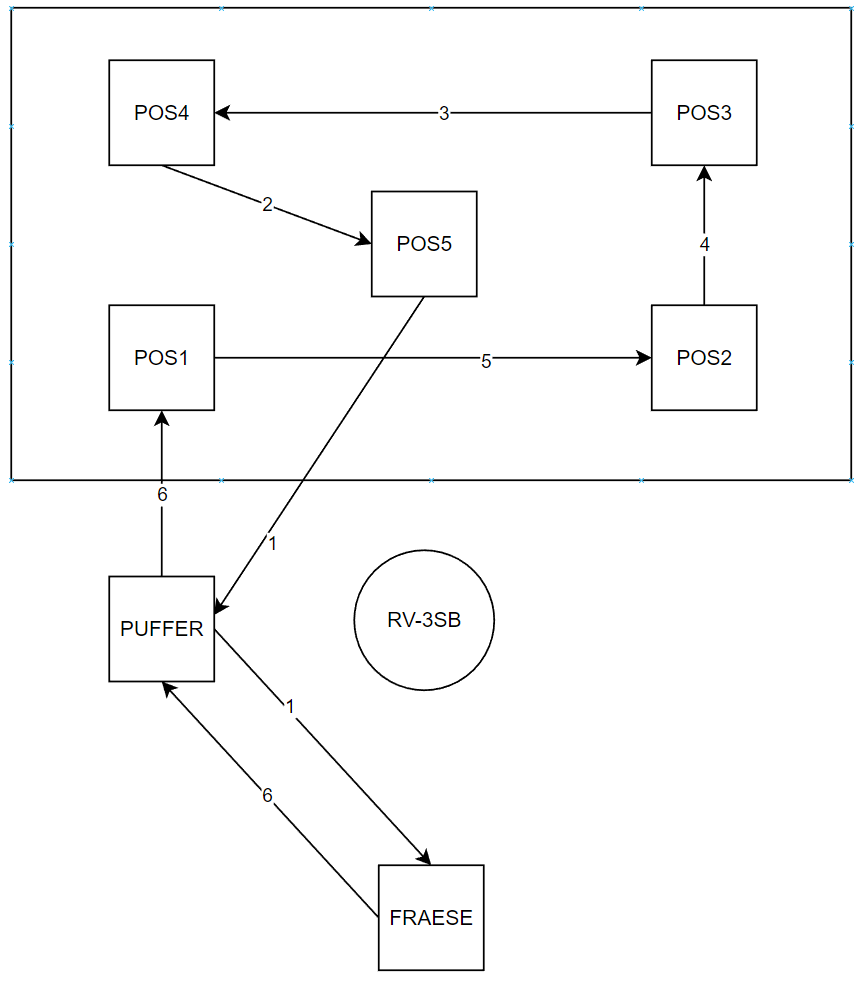
Für das Jubiläum der ADV am 16. März 2023 wurde mit der RobotAPI ein Demonstrationsprogramm geschrieben, welches die Arbeit am Roboter veranschaulichen sollte. Dieses beinhaltet die Simulation eines Arbeitsvorgangs, bei dem Werkstücke verschiedene Stationen durchlaufen, bevor sie zur Fräse gebracht werden. 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 Teach Panel 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 | private 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

Am Ende und nach, beziehungsweise nahe, der Fertigstellung des Projektes lassen sich einige Erkenntnisse und Ausblicke zusammenfassen. Alle Hauptpunkte des Projektes konnten umgesetzt werden, nur nicht immer auf die erhoffte Art und Weise.

## Bewertung der erwarteten Herausforderungen

**Verbinden mit dem Roboter:**

Die Verbindung zum Roboter konnte recht schnell aufgebaut werden, nachdem die IP-Adresse des Controllers und dessen Port bekannt waren. Da gleich die richtige Art der Verbindung, eine Socket, probiert wurde, war diese Sorge schnell vom Tisch und man konnte sich wichtigeren Dingen wie der tatsächlichen Kommunikation widmen.

**Kommunikation mit dem Roboter:**

An dieser Stelle stand man tatsächlich vor einer Herausforderung, da der Befehlssatz zunächst nicht ganz ersichtlich war und sich nicht mit dem erwarteten deckte. Es stand eine Beschreibung zur Verfügung auf deren ersten Seite der gesamte Befehlssatz in einer Tabelle zusammengefasst wurden – nur wurde dieser Teil der Beschreibung erst später gefunden, weshalb man zunächst etwas verwirrt war, da nichts funktionierte.

Nachdem man den richtigen Befehlssatz nutzte, hat sich der Roboter zwar bewegt, doch Abläufe in einem Programm waren schlichtweg unmöglich, da das Programm nicht auf die Vollendung des vorherigen Befehls wartete. Dazu gibt es noch immer keine optimale Lösung, aber zumindest eine, die funktioniert: Bei jeder Bewegung wird in einer Schleife geprüft, ob sich der Roboter bewegt und das Programm wird erst dann fortgesetzt, sobald diese Prüfung `false` ergibt.

Bei anderen Befehlen musste aus einem ähnlichen Grund ein Timeout eingeführt werden, damit der Befehl nicht ignoriert wird. Grund dafür war, dass der Controller wohl mit einer zu schnellen Abfolge von Befehlen etwas überfordert war und dann nur den letzten ausführte.

Im Endeffekt wurde ein weg gefunden die zahlreichen verschiedenen Antworten des Roboters zu interpretieren und zu verarbeiten. So weit, dass man nun sogar Informationen zu Geschwindigkeit, Position und Status herauslesen kann.

Sehr interessant war dann allerdings auch die Art, wie man mit dem Roboter interagieren kann. Implementiert wurde wie beschrieben eine API, die es dem Benutzer erlaubt den Roboter wie mit einem Teach Panel anzusprechen in Form einer Webseite. Zusätzlich zu der Oberfläche der anderen Gruppe wurde hier also eine backend- und auch hardwarenahe Steuerung angelegt und bildet gleichzeitig die Schnittstelle zum Frontend.

**Integration der Bilderkennung:**

Wie erwartet, das war nämlich das Erste, was programmiert wurde, war die einfache Bilderkennung die leichteste Übung. Die Übertragung zum Backend allerdings nicht. Hier war, wie schon beschrieben, vor allem das Problem mit den Koordinaten im falschen Format ein großes Problem. Natürlich ließ sich auch das im Endeffekt lösen, doch war es nicht so einfach wie erhofft.

Zu den sowieso schon gegeben Hindernissen kam noch dazu, dass die Kamera nicht immer gleich eingestellt war, wodurch die Transformationsmatrix teils wieder falsch war, obwohl sie schon richtig funktionierte. Die gleiche Arbeit musste also teils doppelt und dreifach erledigt werden. Dies stellte zwar nicht unbedingt ein Problem dar, doch da es sehr zeitintensiv ist eine solche Funktionalität zu testen, ging dadurch Arbeitszeit an anderen Problemen verloren.

Auch die Tatsache, dass der Roboter und der dazugehörige Computer mehr als die Hälfte der Zeit von externer Seite besetz waren, machte es nicht einfacher Neues zu implementieren und zu testen – gerade bei Themen, die die Bilderkennung und die entsprechende Kommunikation mit dem Roboter betrafen.

**Kommunikation mit dem Frontend:**

## Ausblick

Sowohl die Python- als auch die Java-API werden auch in der Zukunft und potentiell bei anderen Projekten oder Einsatzmöglichkeiten funktionieren, obwohl sie bei verschiedenen Use-Cases Vor- oder Nachteile haben. Die Python-API bietet neben der Steuerung des Roboters auch einen Webserver, auf den eine andere Anwendung (zum Beispiel ein Frontend) zugreifen kann. Die Java-API bietet dagegen eine sehr einfache Art Unterprogramme oder Routinen hinzuzufügen, allerdings keinen Webserver.

Was allerdings wahrscheinlich in der Zukunft angepasst werden muss, ist die Bilderkennung, da an dieser Stelle alles mit der Kamerapositionierung zusammenhängt. Aus Erfahrung wird diese immer wieder verändert und terminiert somit die Transformationsmatrix, die auf die Positionierung abgestimmt ist. Ändert man diese allerdings entsprechend ab, kann auch diese Funktionalität weiterhin genutzt werden.

Über das Projekt hinaus muss noch gesagt sein, dass es in kommenden Projekten gewährleistet sein muss, dass Schüler während des Projektblocks uneingeschränkten Zugriff auf den Roboter haben müssen. Wie oft kam man in den Raum und sah schon von Weitem den geöffneten TeamViewer und wusste direkt, dass man den Roboter an diesem Tag nicht würde nutzen können? Das verlangsamte den gesamten Prozess um ein Vielfaches und nervte zugegebenermaßen. Zumal sich an der Prämisse nichts wesentlich änderte – die Bilderkennung kam trotzdem von der eigenen Gruppe und nicht wie erhofft von der externen Firma.

# Quellen

* Flask-Icon  
  https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Flask\_logo.svg/1200px-Flask\_logo.svg.png