Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Artur Gwozdowicz
Danil Kalland

Kuka - Robot Assistant

Bachelor's thesis in Automation and Intelligent Systems
Supervisor: Ottar Osen
Co-supervisor: Adam Leon Kleppe
June 2023

NTNU
Norwegian University of
Science and Technology

Artur Gwozdowicz
Danil Kalland

Kuka - Robot Assistant

Bachelor's thesis in Automation and Intelligent Systems
Supervisor: Ottar Osen
Co-supervisor: Adam Leon Kleppe
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

1 Abstract

This report is based on the project carried out as a bachelor thesis in the spring semester of 2023. Research, planning and implementation was conducted by Artur Andrzej Gwozdowicz and Danil Kalland, two students of 'Automatisering og intelligente systemer' at NTNU Ålesund.

The objective of the thesis was to utilize the industrial robotic arm KUKA LBR IIWA as an assisting tool, particularly, the workshop assistant. Conducted work revolved around developing a voice controlled pick-and-place solution which allows for functionality expansion. It's also shown how several systems ranging from industry to consumer-grade can be integrated and operate together.

Growing demand on precise and sophisticated tools such as robotic arms is more frequently being present outside the scope of industries and strictly professional applications. Having access to a vast spectrum of tools and systems, both professional and consumer sectors naturally seek advancements or improvements of quality of life in the form of process automation of any kind.

The idea was brought by the interest in managing the potential workplace and therefore increasing the effectiveness of work.

2 Summary

The purpose of the project was to develop a tool-retrieval robot operated by voice commands. Crucial part of the design was to involve uncertainty and avoid linear operations by implementing non-static positioning of the tools which then the robot is asked to pick-and-place. The structure of the robot assistant revolved around several non-related systems integrated together instead of utilizing designated or existing solutions.

The solution of the project includes computer vision for tool recognition and speech recognition. Computer vision part is built using the "OpenCV" python library that can result in both being a reliable and simple solution. Speech recognition part is built using the "SpeechRecognition" python library that uses Google Web Speech API. At the core of the project, a robotic arm, KUKA LBR IIWA is used. The arm is a very versatile tool which allows for precise and flexible control. The components are bound in the primary control unit Beckhoff CX5120, a powerful IPC that supervises and commands any ongoing process in the system.

In the working process the team used different methods and techniques to achieve results for different parts of the project. There were some technical challenges that occurred that slowed down the working process. However, at the end, the project managed to provide a voice-operated pick-and-place solution of multiple tools.

3 Sammendrag

Hensikten med prosjektet var til å utvikle en robot som kunne hente verktøy ved å høre på stemme kommandoer. Avgjørende delen av design var å involvere usikkerhet og unngå lineære operasjoner. Det kunne oppnås ved å implementere ikke statisk posisjonering av verktøy som roboten kan plukke opp hvis han blir bedt om det. Strukturen av robot assistent dreide seg om flere ikke relaterte systemer som er integrert sammen i stedet for å bruke målrettede eller eksisterende løsninger. Prosjektet er administrert til å gi stemmestyrt plukk-og-legg-på-plass løsninger.

Løsningen til prosjektet består av datamaskin syn for verktøy gjenkjenning og tale gjenkjenning. Datamaskin syn delen er laget ved bruk av OpenCV python bibliotek som kan gi både pålitelig og enkel løsning. Tale gjenkjenning delen er bygd ved bruk av "SpeechRecognition" python bibliotek som bruker Google Web Speech API. KUKA LBR IIWA robot armen er brukt i kjernen av dette prosjektet. Robot armen er veldig allsidig verktøy som gir mulighet til presis og fleksibel kontroll. Komponentene are bundet i den primære enheten Beckhoff CX5120 som er et kraftig industriell PC som overvåker og styrer enhver pågående prosess i systemet.

Under arbeidsprosessen, prosjektgruppen brukte ulike metoder og teknikker til å oppnå resultater for forskjellige deler av prosjektet. Det var noen tekniske utfordringer som oppsto og som forsinket arbeids prosessen. Men på slutten klarte gruppen å lage et prosjekt som leverer stemme styrt, plukke-og-plasser løsning av flere verktøy.

Table of Contents

List of Figures

List of Tables

4 Introduction

4.1 Project background

Industrial robots today are used for various tasks in industry. Most of the industrial robots are used for assembly of vehicles, construction and palletizing. There are still some areas in which industrial robots are not commonly used. For example, robot assistants that are working in a lab or a workshop by helping workers with picking up, placing tools by listening to voice commands.

4.2 Project task

This thesis will describe a project in which a Kuka LBR IIWA industrial robot is used as a workshop and lab assistant. The project has a goal to create a robot assistant that will be able to recognize different workshop tools such as hammer, screwdriver and pliers, and perform various actions by receiving different voice commands from the user. There are different methods that are used in this project to achieve this goal. In order to see different objects, the Kuka robot indirectly uses a web camera. The commands from the user will be received through the microphone. More details of the methods and techniques used in this project will be described in next chapters.

4.3 Definitions and abbreviations

IDE - Integrated Development Environment

CV - Computer Vision

RPI - Raspberry Pi

RTU - Remote Terminal Unit

CRC - Cyclic Redundancy Check

POU - Program organization unit

PLC - Programmable logic controller

UINT - unsigned integer

INT - Integer

IO - Input/Output

IPC - Industrial Personal Computer

GUI - Graphical user interface

RAD - Radians

4.4 Structure of the thesis

Theory

Theory part presents the technical foundation for different parts of the project. It covers all the necessary theory of concepts and techniques that are useful for understanding of methods used in the project.

Methods

This part of the report covers methods used to achieve a result for the project that will satisfy all the requirements. It explains how different techniques are implemented and how independent parts are integrated in a bigger system.

Results

This part of the report presents the final results achieved by the team for this project. It includes both successful results and difficulties that occurred during the working process.

Discussion

This part presents the discussion around the results of the project's work. It includes analysis of the system's flaws and strengths, expectations, limitations and challenges that either were at the starting phase of the project or occurred during the development process. It also includes how communication worked between team members during the development process.

Conclusion

Conclusion presents the summary of the work done in the project by the team. It discusses the further work that could be done in the future such as upgrades of some parts of the system.

5 Materials and devices

Table 1: List of devices used in the project

Hardware and Software	Quantity	Short description
Raspberry Pi 4	1	Low cost computer that can be used as microcontroller or as a computer for web browsing, programming etc.
Local PC	1	A PC used for running software and programming
Logitech Webcamera	1	Webcamera that can be plugged in a computer using USB port
Kuka iiwa robot	1	- Robot developed by Kuka and mostly used for industrial applications
Beckhoff PLC CX5120	1	- Embedded PC with Intel Atom processor
EL6695 Beckhoff Module	1	- EtherCat bridge module for Beckhoff PLC
TwinCAT 3	-	Automation software
Sunrise Workbench	-	Software for Kuka robot programming
WorkVisual	-	Software used for system configuration of Kuka robot
Thonny	-	Python IDE which is pre installed on Raspberry Pi

6 Theory

6.1 Raspberry Pi board

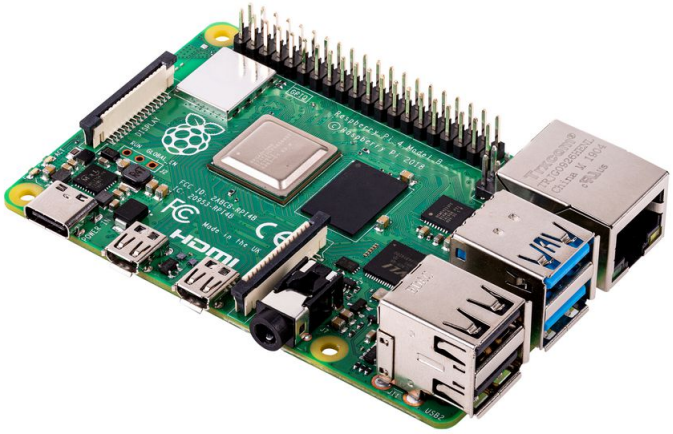


Figure 1: Raspberry Pi 4 - a single board computer

Raspberry Pi, shown in figure 1, is a low cost, single board computer that can be used for programming, computing or web browsing . Programming languages that are supported on the board are C/C++, Scratch and Python. Since Raspberry Pi is a single board computer, it can be plugged into a computer monitor or a TV screen. Mouse and keyboard can also be connected through USB ports on the Raspberry Pi board. This board is widely used by hobbyists in digital projects such as detectors, weather stations, birdhouses with infra-red cameras etc. Like any other computer, it requires an operating system. Raspberry Pi OS is the operating system that is used by all the boards from the Raspberry family [13].

6.2 Speech Recognition

"Speech Recognition" or also called "speech-to-text" is the ability of a program to identify words spoken by the user and convert them into text. Some of the simple speech recognition software is able to recognize words only when they are spoken clearly. With more sophisticated software users can convert speech with different accents and languages into text.

In order to process and interpret spoken words and convert them into text, speech recognition systems use computer algorithms. The sound that is recorded by the microphone is turned into text by a software program that both humans and computers can understand. The process of of sound analysis can be divided in four steps:

1. Analysis of the audio input
2. Division into multiple parts
3. Digitization into computer-readable format
4. Use of algorithm to create a most suitable text representation

The algorithms that are used in speech recognition software are trained on different speaking styles, dialects and accents. Speech recognition software is able to separate speech audio from the background noise that is often present under audio recording. Speech recognition systems use two types of models, acoustic models and language models. Acoustic models represent the relationship

between linguistic units of speech and audio signals. While in language models sounds are matched with word sequences to distinguish between words that sound similar [23].

There are many ways of using speech recognition to understand recorded voices. For example, speech recognition applications can be built using programming languages such as C, Python or Java. Some of the programming languages already provide libraries that make the use of speech recognition applications easier and doesn't require training a model from ground up.

6.3 Computer Vision

6.3.1 Definition of computer vision

Computer vision is the field of computer science which enables computers to replicate the human visual system. Computer Vision is also a subset of artificial intelligence which can collect information from digital videos or images and use that information to processes them and define attributes. The process can involve such techniques as image acquiring, image screening, identifying and analyzing.[10] The general idea and foundation of computer vision is to instruct computers to interpret and comprehend images on a pixel-by-pixel basis. In the field of artificial intelligence, computer vision is dedicated to development of automated systems that can interpret visual data as closely as people can [34]. Computer vision applications can be used for face recognition, pose estimation, object tracking, color detection, security and surveillance.

6.3.2 Color detection using computer vision

Color detection is an image process that involves differentiation between objects based on their color. If an image contains multiple objects with different colors, the color detection methods can return a binary image of the particular color where only the parts with relevant color are white, while the rest is black. There exist multiple ways to detect color. [35]

6.3.3 Color detection in BGR color model

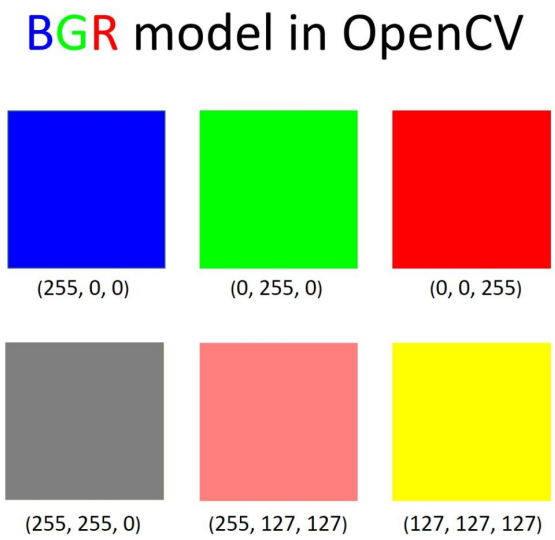
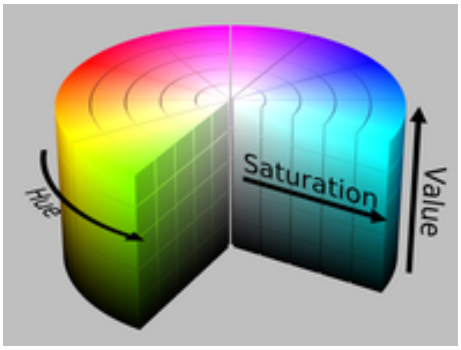


Figure 2: Colors represented in BGR format and example of colors which are the combination of blue, red and green colors

In the concept of color detection in BGR-format, an image has each pixel in the image processing a set of values for each of the 3 color channels. If there is a need for detection of a specific color and it is possible to define the range of R-G-B values for that particular color, then software or a program will look only for those pixels which have R-G-B values in the range that are defined [24]. Figure 2 demonstrates how different combinations of BGR values can create different colors.

6.3.4 Color detection in HSV color model



(a) Cylindrical model of HSV color space

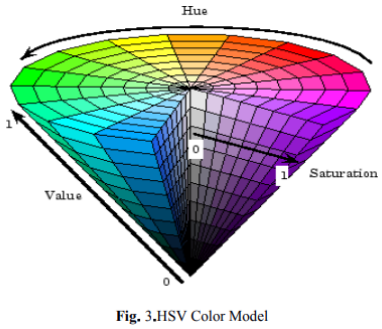


Fig. 3.HSV Color Model

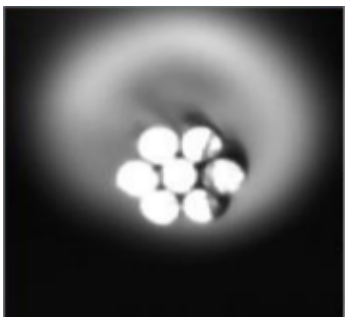
(b) Conical model of HSV color space

Figure 3: Two models of representing HSV color space

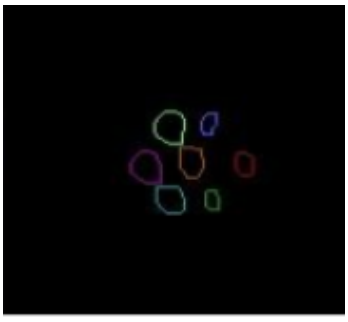
HSV is an abbreviation of hue, saturation and value of the color. In the HSV color model, the value represents intensity of the color, which is decoupled from the color information in the image. The hue and saturation components are related to the way the human eye perceives a color [2]. Hue can vary from 0 to 1 and the corresponding colors vary from red, through yellow, green, cyan, blue, and magenta, back to red. When saturation varies from 0 to 1, the corresponding colors vary from unsaturated colors to saturated colors. Value can be seen as the brightness of the color and varies from 0 to 1. When value is increasing, the colors become increasingly brighter. HSV color space is more intuitive to how people experience color than the RGB color space. HSV is the best use for color choice when a user is selecting color interactively [4]. HSV color space can also have a range from 0 to 255 instead of 0 to 1.

HSV color space can be viewed as a cylinder, where the angle in interval from 0 to 360 degrees, represents the hue. The distance from the center of the cylinder to the outer part of the cylinder corresponds to saturation. The central vertical axis of the cylinder is value, ranging from black on the bottom to fully white at the top [15]. Figure 3 demonstrates HSV color space in two different models.

6.4 OpenCV - Contours and contour tracking



(a) Original image



(b) Contours tracked on the original image

Figure 4: Original image and contours tracked on the image

Contours have a big role in computer vision and are useful for any kind of image processing, shape analysis or object detection. Contours can be seen as curves that join all the continuous points along the boundary of the image [26]. In computer vision applications contour tracking is an edge tracking process that traverses the border of a region completely. By applying contour tracking, it obtains a boundary points sequence as the edge points which could be tracked. Contour tracking is used in the fields of image recognition and libraries like "OpenCV" offer "cvFindContours()" function that perform contour tracking [40]. Figure 4a and figure 4b show original image and contours of the same image, respectively.

6.5 OpenCV - Masking

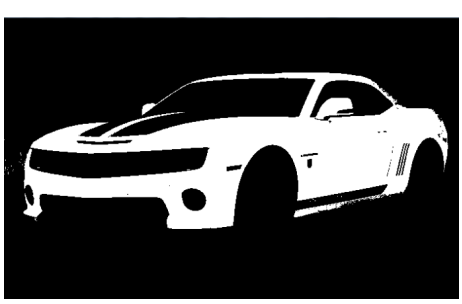
Masking is a useful technique in the field of computer vision. It allows highlighting of a specific object from an image that interests the most [1]. To track a color on an image, the mask can be defined in HSV color space using "cv2.inRange()" command that passes lower and upper limits of color values in HSV. To apply a mask to an image, "cv2.bitwise_and()" bitwise operation can be computed between mask and image [16]. Figure 5 shows an original image of a yellow car as an example.

Figure 5: Image example of a yellow car

Figure 6 shows a python code that imports an image, defines lower and upper limits of a color, creates a mask and shows the result using "bitwise-AND" operation. Figure 7a shows a mask which tracks color and figure 7b color shows tracked yellow color in the input image.

```
img = cv2.imread('car.jpg')

# Convert BGR to HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# define range of blue color in HSV
lower_yellow = np.array([15,50,180])
upper_yellow = np.array([40,255,255])

# Create a mask. Threshold the HSV image to get only yellow colors
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

# Bitwise-AND mask and original image
result = cv2.bitwise_and(img,img, mask= mask)
```

Figure 6: Example code of setting up a mask for an image



(a) Mask made for original image

(b) Mask of original image that tracks yellow color

Figure 7: Mask made for original image and mask that tracks the yellow color

6.6 Robotiq 2F-85 gripper

Figure 8: Rootiq 2F-85 gripper used for industrial robots

Robotiq 2F-85 gripper on figure 8 is an adaptive gripper produced by Robotiq and is used for industrial robots. Robotiq collaborates with a team from Universal Robots where the Robotiq 2F-85 gripper is widely used. Finger design of the gripper makes it possible to do both internal and external gripping of objects. Robotiq 2f-85 is a robust gripper that has high pinch force and payload. Such devices as wrist camera and force feedback modules can be attached to the gripper to allow it perform more tasks on the assembly line [31]. The fingers on the Robotiq 2F-gripper are under-actuated, which means they have fewer motors than the total number of joints. Such configuration makes it easier for the gripper fingers to adapt to the shape of the object that they grasp. The gripper can be controlled by Modbus RTU protocol over USB port using "ACC-ADR-USB-RS485" converter and controlled through "Robotic User Interface".[32].

6.7 KUKA LBR IIWA 7 R800

The Kuka LBR IIWA, shown at figure 9, is a collaborative robot designed for safe and efficient human-robot interaction manufactured by KUKA Robotics. LBR stands for 'Leichtbauroboter' which translates to 'Lightweight robots' from German. IIWA is a code for 'Intelligent industrial work assistant'. The Robot, by its purpose, presents a lightweight and compact design. Its key feature is the Seven Degrees of Freedom movement space which makes it agile and flexible enough to conduct the majority of movement related operations. Being a precise and safe tool, KUKA is equipped with sensitive joint torque sensors which make it possible to detect and respond to collisions and other forces. Paired with precise control which allows sub-millimeter adjustments, it makes a very versatile and safe product [21].

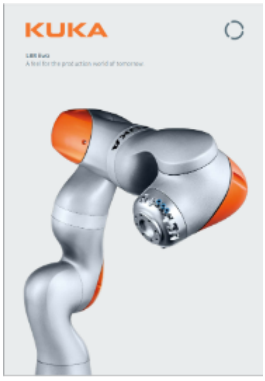


Figure 9: Kuka LBR iiwa 7 R800 industrial robot used in this project

KUKA LBR IIWA is only a part of a greater system, which requires integration with KUKA Sunrise Cabinet, a powerful controller, which holds the necessary power, control and communication interfaces for the robot. Sunbrise cabinet is shown in figure 10. Cabinet provides several interfaces for external I/O devices such as sensors and communication protocols like Ethernet and Fieldbus. It supports various safety implementations like the stop buttons, safety relays and safety circuits which ensure safe operation and compliance with safety standards. KUKA Sunrise Cabinet being the primary control unit of the robot, contains Control software which is in principle, a user-friendly programming environment. The two main Software are the Sunrise.Workbench and the WorkVisual [20].

Sunrise.Workbench is a software that can be installed, typically on Windows-based computers and communicate with the robot through network connection. Its main task revolves around programming, particularly in the Java language. With built in libraries, users are able to use the robot to its full potential [18].

WorkVisual is a stand-alone software providing more configurational features than Sunrise.Workbench. Its primary feature is the system configuration and network communication of the robot. It allows users to configure I/O interfaces, set up safety parameters and integrate external hardware into the system. Workvisual also serves a purpose as a project management tool where one can simulate the models and organize the structure of the project [19]. The software is complementary to other products provided by KUKA as it relies on Import/Export features in order to deliver a solution.

Figure 10: Kuka sunrise cabinet

Sunrise Cabinet is additionally complemented by the KUKA SmartPAD which is shown in figure 11. It enables real-time monitoring of the robot, manual calibration and control of the robot. It is a handheld control device with touchscreen interface connected directly to the Cabinet. In addition

to being the absolute control unit of the robot, it packs several safety features like buttons and keys [22].

Figure 11: Kuka smartPAD used for movement control of industrial Kuka robot

6.8 Programmable Logic Controller

Abbreviated as PLC, is an industrial processing unit utilizing modular components designed primarily for process automation. Controllers and its modules often provide inputs and outputs for a variety of simple end devices, as well as specially tailored industrial equipment. PLCs are choice devices for performing repetitive tasks and its robustness makes it viable for deployment in most environments such as production plants, elevators or even smart homes.

Overview over PLC system is presented in figure 12:

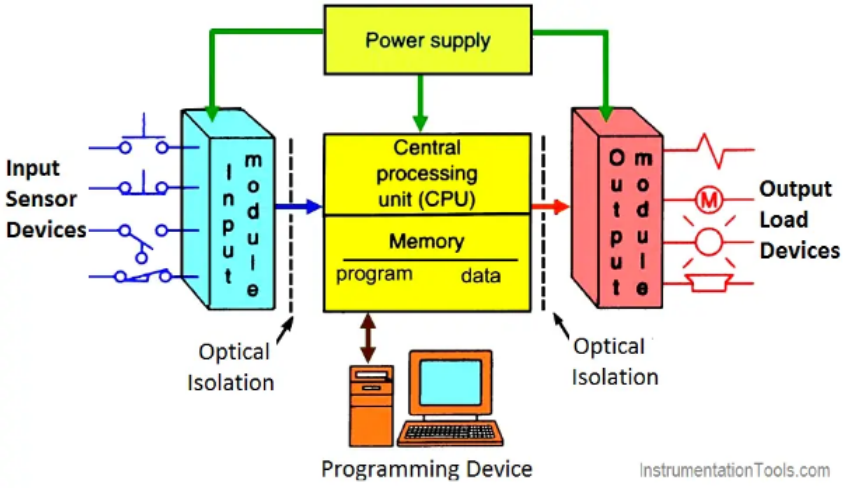


Figure 12: PLC system showing parts it can contain

Typical PLC system consists of :

- Power supply - Electrical device that supplies electric power to an electrical load.
- Input modules - Components which receive signals from external devices such as sensors.
- Output modules - Components which send signals to external devices such as sensors.
- Central processing unit (CPU) - Supplied by the memory, runs the operating system on the device and executes tasks in the form of a program.
- Programming device - External device, such as PC, which allows the user to create a program to be run on the PLC [36].

6.8.1 IPC and Embedded PC

IPCs, short for Industrial Personal Computers, are devices specifically designed to be used in industrial environments. They offer a flexible architecture similar to a standard computer compared to specifically tailored PLCs. They are more resilient than typical computers and cover areas typically troubled by dust, abnormal temperature and vibration. Usually IPCs can be considered Embedded PCs, which are computer systems integrated into a larger device or system. In terms of industrial controllers, they are built to perform specific tasks, usually executed by their PLC counterparts. However, such systems provide possibilities for running software applications, visualization, data processing and communication which extends its use to areas such as data acquisition, human-machine-interfaces or data logging. IPC offers a general-purpose CPU, memory and storage, therefore they can run operating systems such as Windows or Linux which provide a wide range of tools and programming languages for its users [14].

6.8.2 Beckhoff CX5120

Beckhoff CX5120, shown in figure 13 is an IPC or an Embedded PC which is part of Beckhoff CX5100 series. It presents a compact, yet durable design fulfilling the industrial device requirements. It is powered by the low-power and low-cost general purpose CPU Intel Atom which finds its usage in such embedded systems. CX5120 offers a variety of communication interfaces like the Ethernet, USB and serial ports allowing it to communicate and integrate with a wide range of devices and networks. Additionally, it can be supplied with extension modules for I/O control and beyond. The CX5120 can be considered an IPC due to industrial-grade features and capabilities and likewise as an embedded system for its special purpose and optimization within the industrial environment [5].

Figure 13: Beckhoff PLC used in this project

6.8.3 EL6695 - Beckhoff module

The beckhoff EL6695 is a bridge module which is a part of Beckhoff's EtherCAT Terminal System designed to provide real-time data exchange between strands with different masters. Beckhoff EL6695 module is shown in figure 14.

Figure 14: EL6695 - Beckhoff PLC module

The module acts as an interface between EtherCAT and other fieldbus systems, allowing for integration and communication between masters. It Provides a flexible CoE (CANopen over EtherCAT) interface, which is used for parameter management of EtherCAT devices. The interface allows for the integration of CANopen devices into EtherCAT network, therefore CANopen devices can benefit from the high-speed communication and real-time performance of EtherCAT [6].

In AMS NetID, AMS stands for Automation Device Specification and NetID stands for network identifier. It is a local address of the device in the TwinCAT network. AMS NetID consists of 6 bytes and must be assigned by the project planner. NetID should not be repeated in the TwinCAT environment [7].

6.9 Software

6.9.1 Multithreading and multiprocessing

Multithreading, particularly in Java language, refers to the ability to execute multiple threads simultaneously in a program sharing the same resources. Using such a method, it's possible to achieve concurrent and efficient program execution. Java supports multithreading through the Thread class which allows for certain operations like Creating the thread, Starting, Scheduling and synchronization [28].

Multiprocessing is another tool to achieve concurrent execution in a program with some key differences in its architecture compared to multithreading. Multiprocessed tasks rely on their own memory space and resources and may communicate with each other through more complex communication methods like pipes or sockets. As a result of separate resources, multiprocessing can efficiently use multiple CPU cores allowing for true parallelism [9].

Both multithreading and multiprocessing are utilized to achieve program parallelism and efficient execution. Depending on the nature of the task, either one should be used as they both excel in different environments.

6.9.2 Modbus RTU

Modbus RTU is an open serial industrial protocol which uses master/slave architecture and was developed by Modicon. This serial protocol is widely used in industry because of its ease and

reliability. It can be used in "Industrial Automation Systems" or within "Building Management Systems". RTU stands for "remote terminal unit" and is a version of Modbus that uses client/server technique to communicate between devices. If there are devices or applications that use RTU protocol then there will be at least one server and one client. [3].

Messages in an Modbus RTU protocol are based on data read or data write from the master to the slave. Packets that read data include a slave node address, which is a node address that the data will be read from, start address with the number of coils or registers to be read, and a 16 bit CRC, which stands for Cyclic Redundancy Check. Reply from the server will contain a message with the slave node number, command, number of bytes of data, data itself and 16 bit CRC [12]. Coils and registers are data types in Modbus. Coils can be seen as single bits which have value of 0 or 1. Coils can hold a state or a status of some physical input or output signal. Registers are 16 bit unsigned data that can have value from 0 to 65535 or 0 to FFFF in hexadecimal [3]. Modbus RTU can be used in combination with such programming languages as C, java or Python to create communication between devices.

6.9.3 TwinCAT 3

TwinCAT 3 is a software developed by Beckhoff Automation for controlling industrial automation systems and programming them. It is used for controlling mainly the Beckhoff hardware and managing its resources and capabilities. It also serves a purpose of visualization and simulation environment.

Twincat 3 is based on several programming languages typical for programmable logic controllers (PLC). The main languages are the Structure text(ST), Function Block Diagrams (FBD), and ladder diagrams(LD) [8].

Structured text is a text-based language which resembles the syntax of C programming language. It is designed to be intuitive to write and read [11].

Function block diagram is a construct or a modular unit of code which allows for creating graphical representations of Function blocks and functions in the form of boxes. They are designed to promote code reusability, modularity and scalability [38].

Ladder diagram is a programming method which in its form resembles the rungs of a ladder. The 'rungs' contain various control elements of choice. Additionally, the resemblance of rung representation to contact relay circuits was intended to aid programming by specialists familiar with electrical diagrams [39].

6.9.4 EtherCAT

Abbreviated Ethernet for Control Automation Technology is an industrial Ethernet-based fieldbus system developed by Beckhoff Automation widely used in industrial automation and control applications. EtherCAT utilizes the Slave-Master communication model with key features being its high performance. Industrial Automation and control systems are considered real-time systems, where low latency and data priority is the key to obtain an efficient and safe environment. EtherCAT does that by the support of distributed clocks, ensuring synchronized operation of devices and precise timing control across the network and its flexible topology which allows for easy network expansion [30].

7 Materials and methods

This chapter presents methods that are used in the project to achieve a desired result. Methods are presented as a guide on how the necessary software, components and libraries should be set up. All the information is described using simple examples that would be referenced to in later chapters.

7.1 Planning and design

The development process, as conducted by a two-man team required scrupulous planning, design, milestone management and communication in order to be carried out correctly. Certain aspects of the project structure were unexpectedly altered and demanded great team effort to counter.

7.2 Initial meeting

The first meeting was an introductory meeting at the NTNU University, where we got to know each other and learn about each other's backgrounds. We discussed what technologies and types of work are of interest and decided on the future project idea. Thanks to early stage meetings, either alone or with project coordinators, we were clearly able to establish a project culture and team dynamics. Furthermore, it helped to mitigate certain risks as different visions for the project development were presented. We managed to address any uncertainties and came to common conclusions.

7.3 Pre-project

During the very early stage of the project planning, the pre-project plan was essential for going on forward. Future milestones and expectations on how the project should proceed were required for assigning the tasks and creating an overall roadmap.

The key parts of the pre-project were allocation and utilization of the resources and the risk management. The group expected to require various hardware and technical support regarding safety of the workplace. Moreover, it transpired to be significant to identify potential risks and occurrences which can influence the development. Due to the nature of the project, where a great number of physical hazards can take place, risk evaluation was indispensable for the beginning of the work. The pre-project plan acted as a guidance for future plans for the project and greatly helped the group achieve the intended outcome.

7.4 System design

The project required designing a system where several not-fully related components are supposed to communicate and cooperate with each other. Design process has gone through several iterations, where the most favorable version was picked. The final composition of the system consists of the following parts:

- KUKA LBR IIWA robot
- Beckhoff CX5120 paired with the Beckhoff EL6695 bridge module
- Raspberry Pi 4
- Robotiq 2F-85 gripper
- Stationary Computer

Each component has their own distinctive communication methods and utilize various communication protocols. Roles, assignments and design procedures, will be described in further sections of the report.

7.4.1 KUKA LBR IIWA and CX5120

KUKA LBR IIWA is undoubtedly at the core of the project. There are several reasons as to why it was deployed and it proved itself superior to any other explored solution. Without any prior knowledge to the system, the group was able to quickly identify the needs and possibilities of the system. One of the key features of interest was the high precision and accuracy. It proves itself very consistent in performing tasks and requested movements are very persistent. Another important factor was flexibility and versatility. The robot utilizes seven degrees of freedom which is a very important feature when picking up randomly placed objects. Lastly, KUKA Sunrise Cabinet allows easy integration of external devices and seemingly simple programming through Java language.

To prepare a robot for external control and make it a part of a greater ecosystem with better scalability , it requires a control device such as Beckhoff CX5120. The family of Beckhoff controllers are irreplaceable when it comes to ease of integration with KUKA robots. The given model is equipped with an etherCAT coupler, which paired with EL6695 module allows easy connection to KUKA. The project was oriented around industrial appliances, therefore an industrial control unit was necessary. The CX5120 shows superior modularity and PLC-like programming capabilities, therefore was a perfect choice for the project.

7.4.2 SpeechRecognition v3.10.0 library

There are several ways of creating a speech recognition application. Speech recognition application for this project will be established using the Python programming language. The application itself is run on a stationary PC. “SpeechRecognition” library has been chosen for the project because of its simplicity and easy setup in Python IDEs such as PyCharm. The application uses voice input that comes from a microphone connected via USB port, and transforms it into a text. The text is a string which can be sent as voice command to other devices. ”SpeechRecognition” library allows to use all the functionality that is required for the project without using lots of hardware resources.

7.4.3 OpenCV, color band solution and Raspberry PI

To establish a computer vision for tool recognition, ”OpenCV” python library has been chosen for this project. ”OpenCV” is an easy to use open source library that can be used both for simple and complex tasks. For this project ”OpenCV” computer vision application is run on Raspberry Pi 4 board. The task of the project is the ability of web camera to detect tools and differentiate them from one another. In addition, the webcamera should be able to calculate position and orientation of each of the tools. For detection of the tools, color bands have been chosen as a possible solution for this problem. Color bands for each tool have one primary bottom color, which is common for all the tools, and upper secondary color, which is different for each of the tools. Tool detection can also be done by training the computer vision model to recognize tools by using multiple images of the objects. The reason color bands have been chosen as a solution is that it’s a simple solution that is only based on color recognition and doesn’t require any training of the recognition model. Computer vision application runs well on the Raspberry Pi 4 board and performs all the tasks that were required for the project. Precision of the color detection can vary based on where the source of light is.

7.4.4 Gripper, why Robotiq 2F-85 gripper

Robotiq 2F-85 adaptive gripper is selected as a gripper for Kuka iiwa robot. This gripper has all the functionalities that can be useful for the project. Gripper is easy to connect and can be

controlled from a python environment using serial commands. In addition, Robotiq gripper's finger design allows it to pick up and adapt to size and shape of the objects.

7.4.5 Data flow chart

Data flow chart depicts the sequence and logic of the data flow. In order to communicate several devices together, it required implementation of answer-request relation. Figure 15 presents the data flow chart which presents the basis on which the devices cooperate.

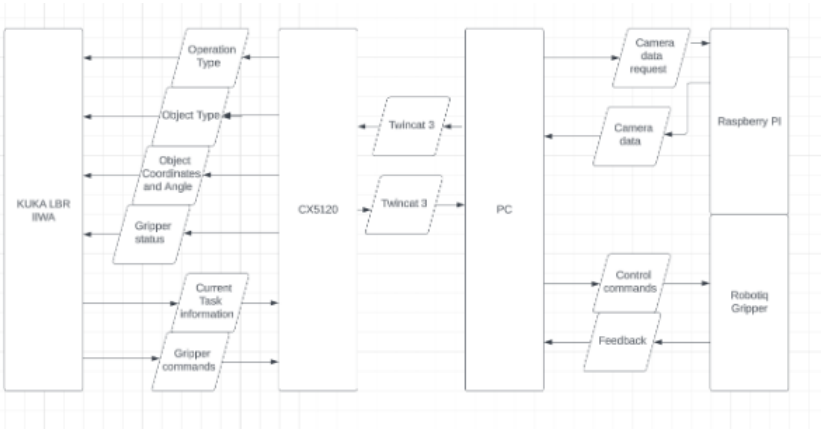


Figure 15: Data flow chart for the system

Planning part of the project demanded scrupulous design choices, where the system will be perfectly functional, yet not unnecessarily complicated. Following subsections go into detail about the structure of exchanged data.

7.4.6 Data exchange - KUKA LBR IIWA and Twincat 3

Data exchange between the Robot Environment and Twincat 3 is established by implementing the EL6695 module into the system. The logic and basis of the communication is solely based on the master-slave principle, where CX5120 takes the initiative of ordering the robot, based on its feedback, other external data and user inputs.

TwinCAT 3

In the project, the data type of choice was Unsigned Integer (UINT), a 16-bit data type which records values from 0 to 65535. The variables must be created as a part of the EL6695 module in Twincat 3 under 'IO Inputs' and 'IO outputs'. Variables are given either KUKA_ or PLC_ prefixes for better visibility and differentiability. The figure 16 shows variable declaration for EL6695 in TwinCAT 3.

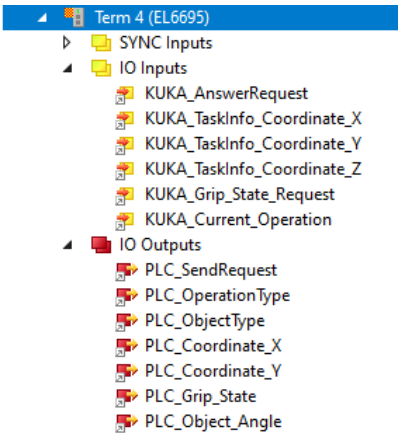


Figure 16: Variable declaration in EL6695 module in Twincat 3

Variables are further linked to their Global counterparts in the program in order to be handled. In order to be properly assigned, they require correct initialization as either inputs or outputs. Inputs are declared as 'AT %I*' and outputs are declared as 'AT %Q*' as shown on figure 17.

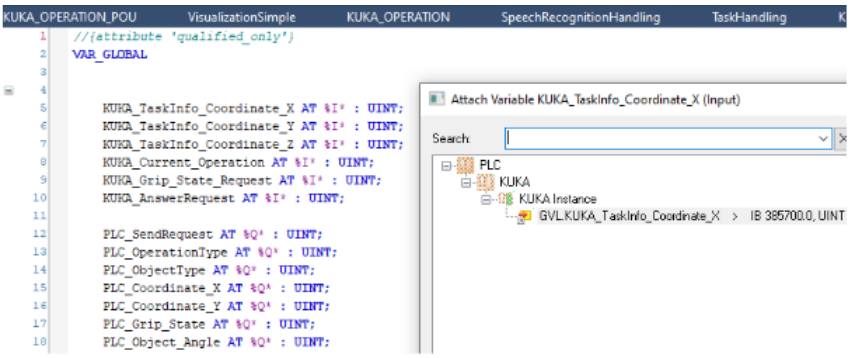


Figure 17: Declaration of the variables as global variables in TwinCAT 3, together with linking an exemplary variable

WorkVisual

Similar process must be conducted on the second side, namely Sunrise Cabinet. External variable declaration is handled by the software WorkVisual, which is the primary configuration tool for the system. Assuming that the EL6695 integration, described in the Communication section was carried out correctly, further configuration can take place.

First part of the setup requires prior knowledge of the future system design. WorkVisual requires a known number and names of the variables which are going to be handled. Inputs and outputs are declared as WORD, a 16-bit size, which matches with data size configured on the Twincat 3 side. Figure 18 demonstrates configuration of EL6695 settings.

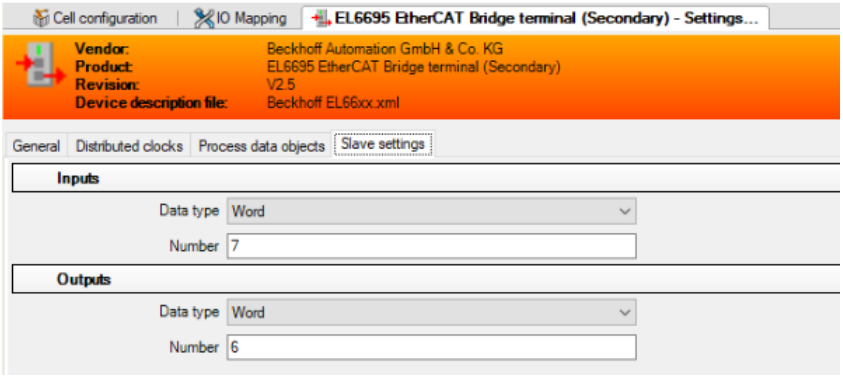


Figure 18: initial configuration of Slave (EL6695) settings

Proceeding to the IO mapping, a special variable group under Sunrise I/O groups is created. A Sunrise I/O group refers to the group of variables which are going to be utilized inside the Sunrise.OS software, where the main programming takes part. After naming the variables after their Twincat 3 counterparts, they can be assigned in the reverse manner - inputs as Twincat 3 outputs, and outputs as Twincat 3 inputs. Mapping of variables in WorkVisual is shown in figure 19 and signal editor of the EL6695 module is shown in figure 20.

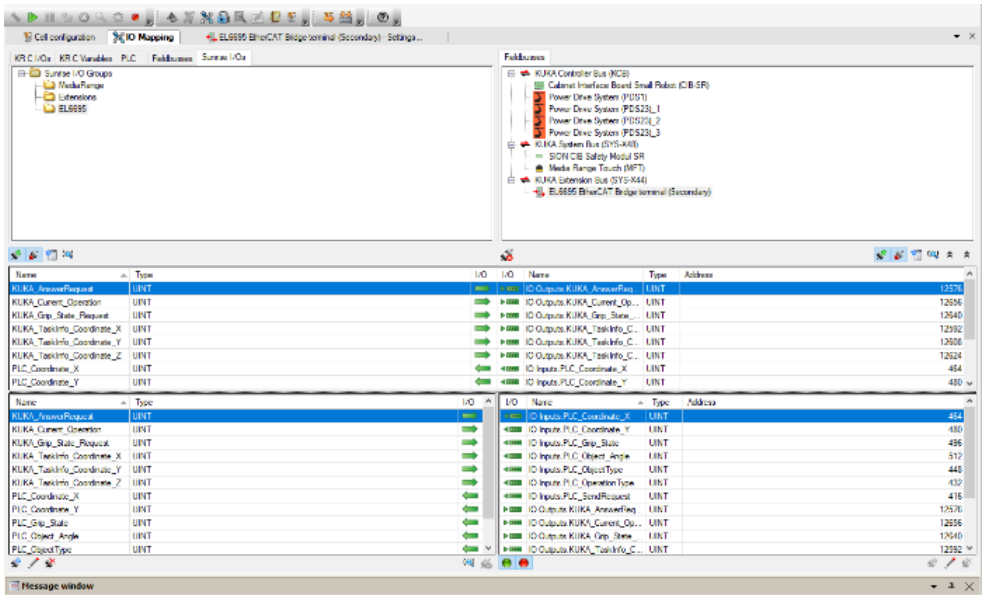


Figure 19: Variable mapping inside WorkVisual software

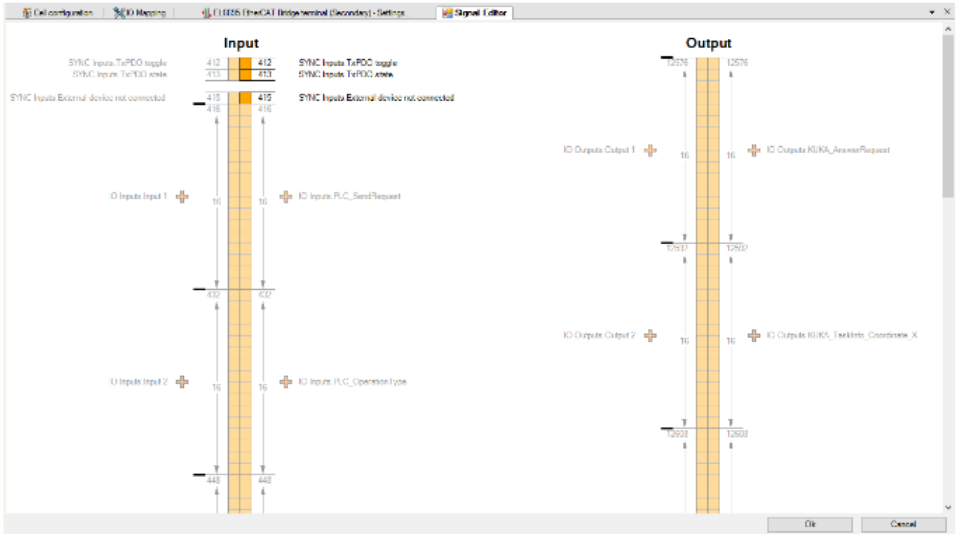


Figure 20: Signal editor of the EL6695, where proper variable names can be assigned

The WorkVisual project can be exported into the Sunrise.OS by utilizing the built in Import/Export function under the File prompt. On the side of Sunrise.OS, a declared variable group named 'EL6695IOGroup' can be imported and used as shown in figure 21.

```
public class RobotApplication extends RoboticsAPIApplication {
    public static boolean startInfoThread = true;
    private LBR robot;
    @Inject
    private EL6695IOGroup io;
```

Figure 21: Initialization of EL6695IOGroup as 'io' inside the Sunrise.OS software.

7.4.7 Data exchange - Twincat 3 and Python

A great portion of data exchange happens between the Twincat 3 and Python script. The script, run by choice, on the PyCharm software is responsible for receiving all of the data from the raspberry PI and the gripper. Additionally the script controls the Speech Recognition algorithm, as well as commanding the Robotiq Gripper. Code is executed on the PC due to better optimization of the hardware which relieves the computing load and due to better networking possibilities. All variables on the Python side contain the prefix 'KUKA_OPERATION_POU' which is an identifier referring to primary POU run in the TwinCAT 3 Software.

Speech Recognition algorithm is handled by the Boolean request sent from Twincat 3, namely KUKA_OPERATION_POU.RequestSpeech, which in return, after executing the algorithm overwrites the variable KUKA_OPERATION_POU.sWord in Twincat 3 with the String containing interpreted word.

The Gripper section of the python code is request based as well. The script receives in total two Gripper related boolean variables, namely KUKA_OPERATION_POU.RequestGripClose and KUKA_OPERATION_POU.RequestGripOpen. Based on those requests and execution, the code produces a feedback which informs the user if the gripper is closed, opened and if it has detected an object. In boolean variable form, they are named KUKA_OPERATION_POU.RPI_Grip_Closed, KUKA_OPERATION_POU.RPI_Grip_Opened, KUKA_OPERATION_POU.ObjectDetected. Figure 22 shows a use of plc variables in python code.

```
plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Opened", True)
plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Closed", False)
plc.write_by_name("KUKA_OPERATION_POU.objectDetected", objDetected)
```

Figure 22: Figure presents exemplary use of the variables within the actual code.

Object Detection is likewise handled by the boolean request call from Twincat named

KUKA_OPERATION_POU.RequestCoordinates. Upon receiving the call during a certain time frame, it will return coordinates supplied by the Raspberry PI. Variables contain strictly integer values as only such values can be passed further to the KUKA robot after conversion to UINT. In addition, each variable, excluding 'Angle'-related ones, corresponds to a position given in millimeters, therefore they behold the sufficient precision. Figure 23 shows variable that are passed to ther TwinCAT 3.

```
# Send data about objects to the PLC
if plc.read_by_name("KUKA_OPERATION_POU.RequestCoordinates"):
    plc.write_by_name("GVL.Raspberry_Screwdriver_X", int(ScrewdriverData[0]))
    plc.write_by_name("GVL.Raspberry_Screwdriver_Y", int(ScrewdriverData[1]))
    plc.write_by_name("GVL.Raspberry_Screwdriver_Angle", int(ScrewdriverData[2]))

    plc.write_by_name("GVL.Raspberry_Hammer_X", int(HammerData[0]))
    plc.write_by_name("GVL.Raspberry_Hammer_Y", int(HammerData[1]))
    plc.write_by_name("GVL.Raspberry_Hammer_Angle", int(HammerData[2]))

    plc.write_by_name("GVL.Raspberry_Pliers_X", int(PliersData[0]))
    plc.write_by_name("GVL.Raspberry_Pliers_Y", int(PliersData[1]))
    plc.write_by_name("GVL.Raspberry_Pliers_Angle", int(PliersData[2]))
```

Figure 23: Variables passed to TwinCAT 3.

Complete representation of data exchanged between Python script and Twincat is given as figure 24:

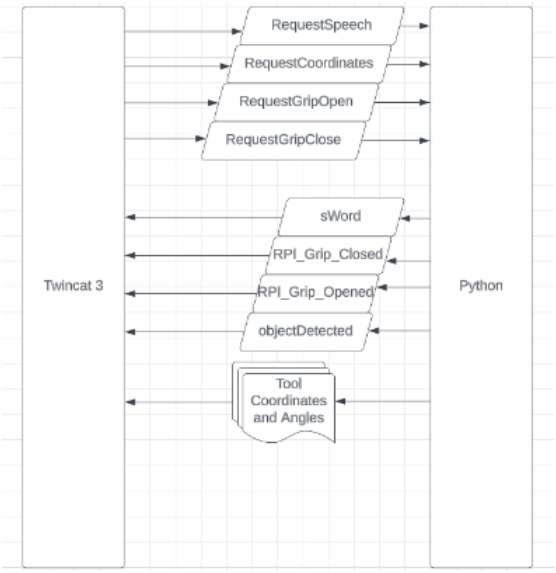


Figure 24: Data exchange between Python and TwinCAT 3.

7.4.8 Data Exchange - Raspberry PI and Python (PC)

Data exchange between Raspberry and PC occurs continuously and is completely non-request based. In order to avoid unnecessary complexity, only one, encoded type of data is being sent.

On the Raspberry side, the data is composed in the form of lists containing both X and Y coordinates and angles for each individual tool. Further, the list must be converted to string in order to pass it through the encode() function of the python. It is achieved by joining a symbol ';' for each data index as showed on figure 25.

```
objectData = [xData, yData, angleData]
objectDataString = ';'.join(objectData)
```

Figure 25: Creation of the objectData list and further converting it to string via joining ';' symbol.

Before passing the data, all separate stringed lists are merged into one with one more separator sign '%' in between the lists. Once that is done, the stringed list containing data for the tools can be encoded and sent through as shown in figure 26.

```
list = [RedBlueData, RedOrangeData, RedYellowData]
stringedList = "%".join(list)
conn.sendall(stringedList.encode())
```

Figure 26: The final list is created, encoded and sent for processing in Python (PC)

On the other side, a reverse process is conducted. Data is decoded, separated firstly by the '%' sign in order to produce three stringed lists and at last, each stringed list extracts the actual data by separating for the ';' sign. Indexes for the actual tools are predetermined in the Raspberry code upon creation of 'list' where for example, 'RedBlueData' represents the hammer. The process of data extraction is shown in figure 27

```
s.sendall('Recieving...'.encode())
data = s.recv(1024).decode()
dataSplit = data.split('%')

HammerData = dataSplit[0].split(';')
ScrewdriverData = dataSplit[1].split(';')
PliersData = dataSplit[2].split(';')
```

Figure 27: The process of extracting the data from the encoded stringed list.

7.5 Communication

Communication between devices is the primary requirement for a functional system. In the project, various communication methods and protocols were used, through either direct connection, like the robot to CX5120 or through the router. A router was utilized for several PCs conducting development on different branches of the project. Limited number of connection ports of certain devices were countered by utilizing DHCP routing through the router. Router setup is shown in figure 28.

Figure 28: Router setup.

7.5.1 Communication of PC to CX5120

Establishing a connection with the CX5120 is the backbone of the project. CX5120 is the main processing and logic unit utilized in the solution, therefore ensuring the proper communication is required. The Beckhoff IPC runs the Windows operating system with Twincat 3 runnable.

The CX5120 can be connected to the PC simply via one of two RJ45 ports allowing for TCP/IP communication. Each port can be configured manually on the machine for either static IP or DHCP. For ease of use and problem-free routing through the router, the method of choice was DHCP.

In the Twincat 3 software running on an external PC, the controller can be found using the routing table and further used in the software. Status of the connection between CX5120 and PC is shown in figure 29.

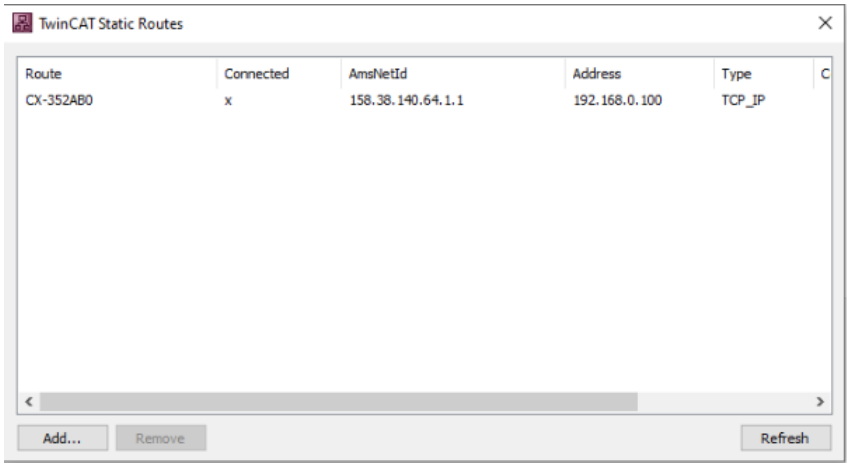


Figure 29: CX5120 IPC connected to the external PC.

7.5.2 Communication via Beckhoff EL6695

In order to create a communication between an external system and KUKA LBR IIWA, it is required to communicate two master systems with each other. After securing necessary hardware connections, EtherCAT communication can be established and data can be exchanged.

Hardware setup

Exact provided model of the EL6695 module is as shown in the picture below. It is a model provided by the Beckhoff with its primary side being destined for the E-Bus connection through the controller like CX5120. The secondary side of the module supplies two RJ45 connectors ready for EtherCAT communication. The EtherCAT module is shown in figure 30.

Figure 30: EtherCAT bridge module used in the project

Connection with KUKA LBR IIWA occurs by connecting the Sunrise Cabinet via Extension bus X65 to 'IN' port of the EL6695. Connector X65 allows for connection of EtherCAT slaves, such as peripheral devices and controllers [17]. Figure 31a and figure 31b show setup of Kuka cabinet and Beckhoff PLC, respectively.

(a) connection setup of the KUKA Sunrise Cabinet

(b) connection setup of the Beckhoff CX5120 with Beckhoff EL6695 bridge.

Figure 31: Setup of Kuka Sunrise cabinet and Beckhoff PLC

Software and configuration

First software initialization of the Beckhoff EL6695 bridge occurs in the Twincat 3, run on the PC. The device can be simply added by scanning for additional devices after the primary connection to the controller has been established. Device scan is shown on figure 32.

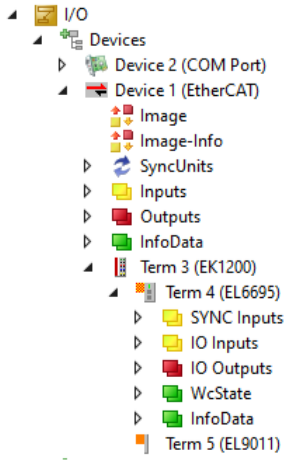


Figure 32: the effect of device scan performed on connected CX5120 IPC

Once the device is clearly integrated in the Twincat environment, input and output variables can be instantiated under the 'IO Inputs' and 'IO outputs'. It is important to note that inputs and outputs are created with reference to the controller and not the KUKA robot. Variables must be created according to a predetermined data type which is supported by the Sunrise Cabinet for receiving and sending. Once the proper variable setup is done, the configuration can be uploaded to the bridge.

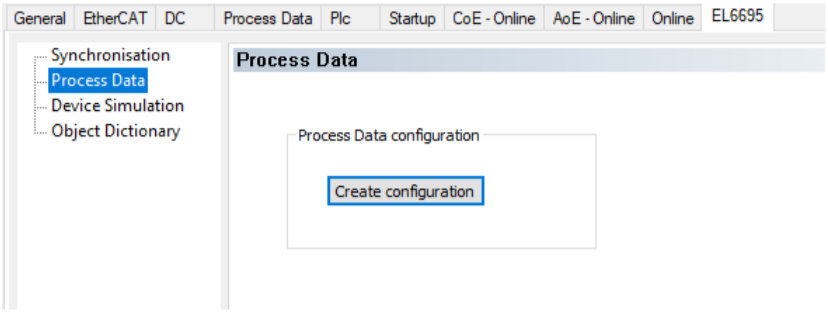


Figure 33: Process data synchronization with the module by creating a new configuration.

Second phase of the integration occurs on the KUKA side. In order to configure and integrate the bridge into the Sunrise Cabinet, it is necessary to modify or create a new IO configuration in the WorkVisual Development Environment as shown in figure 33.

It is achieved primarily by adding the correct controller in the software, namely KRC4 -8.3.0 in this particular project. Further, it is possible to utilize the X65 Extension Interface as described before, which directly gives access to SYS-X44 Extension Bus. Figure 34 presents setup of EL6695 module in WorkVisual.

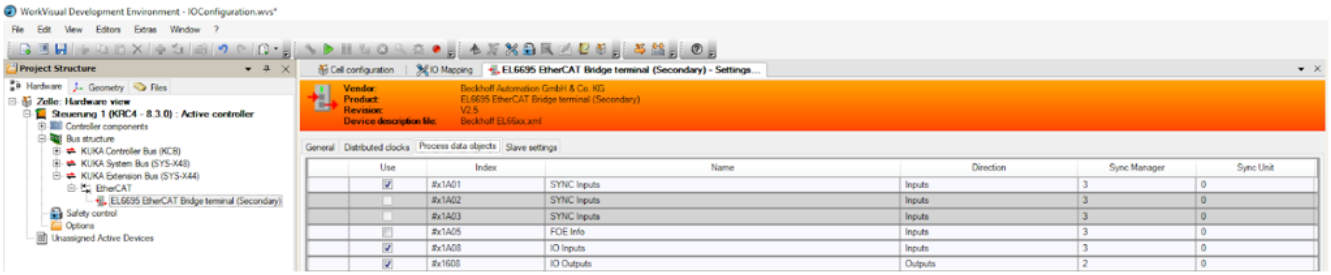


Figure 34: Setup of EL6695 in WorkVisual

After the proper IO exchange setup, which was described in section 3.2.3, Communication should be established properly.

7.5.3 Communication via Pyads

"PyADS" is a python library that is used for communication between the python environment and TwinCAT devices. Python code with "pyads" library is run on a stationary PC. The PC is running both python code and TwinCAT application in the background while being connected to the PLC via the router. Figure 35 shows python code that uses "pyads" library to communicate with TwinCAT devices:

```python
import pyads

AMSNETID = '158.38.140.64.1.1'

plc = pyads.Connection(AMSNETID, pyads.PORT_TC3PLC1)
plc.open()
print(f"Connected?: {plc.is_open}")
print(f"Local Address? : {plc.get_local_address()}")
print(plc.read_state())

plc.write_by_name("MAIN.iNumber", 11)

iNumber = plc.read_by_name("MAIN.iNumber")

print(f"iNumber State: {iNumber}")

plc.close()
```

Figure 35: Communication with PLC from python using PyADS library

At the beginning, the python code defines "AMSNETID" of the device it tries to communicate with. "AMSNETID" for the device is '158.38.140.64.1.1'. Variable 'plc' establishes connection to the TwinCAT device by using "AMSNETID" of the device and port. "plc" variable opens a connection and checks if it's connected, then prints the status of the connection. Connection to the PLC should be open before variable values can be changed.

By using the "plc.write_by_name()" command, python changes the value of the target variable "iNumber" in TwinCAT. Command "plc.write_by_name()" writes a new value to the variable given as "MAIN.iNumber" in Beckhoff PLC. On the other hand, "plc.read_by_name()" is used to read a value of the variable from the Beckhoff PLC. At the end, python reads the new status of the variable with command "plc.read_by_name()", prints the status of the variable and closes the connection.

7.5.4 Communication and control of Robotiq 2F-85 gripper

Robotiq 2F-85 gripper

Robotiq 2F-85 adaptive gripper is connected to the USB port on local PC through "ACC-ADR-USB-RS485" serial to USB converter. Figure 36 illustrate wiring schematics of the 2F-85 gripper, power supply and serial to signal converter:

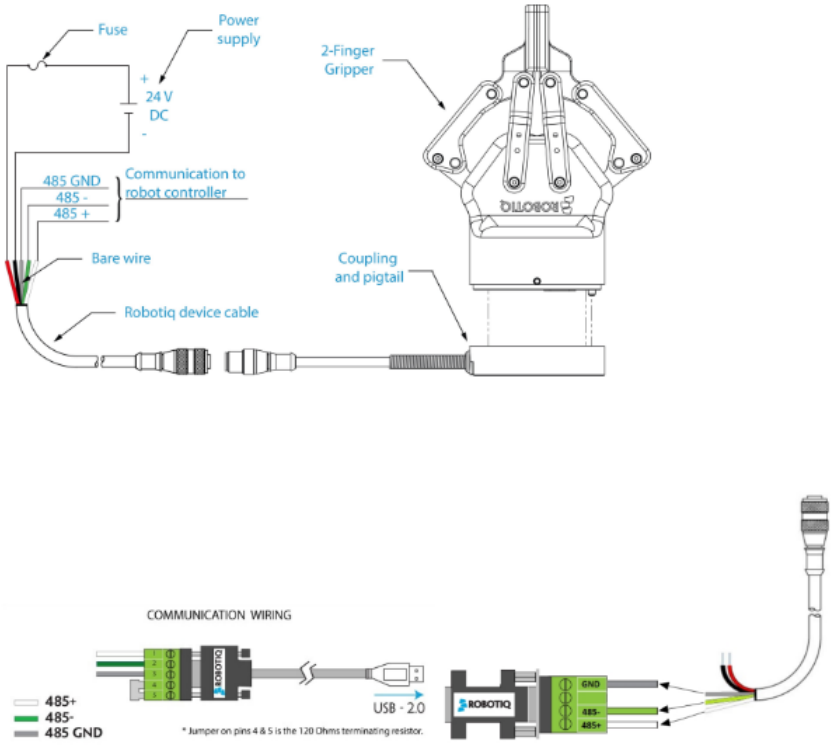


Figure 36: Connection of Robotiq gripper with serial to USB converter

White signal and green signal wires on the Robotiq device cable should be connected to the port "485+" and port "485-" respectively on the signal converter. Bare wire is connected to the "485 ground". Red and black wires are connected to the power supply, 24V and 0V respectively. [33].

Software

The communication method of choice for the Robotiq Gripper yielded to be the Modbus RTU. Sending and receiving direct serial commands proved itself sufficient and reliable enough to conduct any desired operation.

A portion of commands and control principles were selected with reference to the products manual, which contains necessary information about communicating through Modbus RTU.

The used commands were:

- Activation command - [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x73, 0x30]) - Commands the gripper to Activate.
- Open command - [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x09, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x72, 0x19] - Commands the gripper to open
- Close command - [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x09, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x42, 0x29] - Commands the gripper to close.

- Check command - [0x09, 0x03, 0x07, 0xD0, 0x00, 0x01, 0x85, 0xCF] - Command to receive a feedback from the gripper.
- readline() function provided by the 'serial' module for python - to read current feedback from the serial port.

7.6 Twincat 3 program

The CX5120 is without a doubt the center of the project. Combined with a complementary PC, it connects all the nodes in a system. To carry out all of the designed procedures within the project, an appropriate Twincat 3 program must be in place. It has shown that the request based solution in such a diversified environment is an appropriate method of implementation.

The main program (POU) named 'KUKA_OPERATION_POU' consists of 5 Function blocks, which handle all of the operations. Function blocks are distinct in their functionality and each of them was created with a separate operational scope in mind. The main program is shonw in figure 37

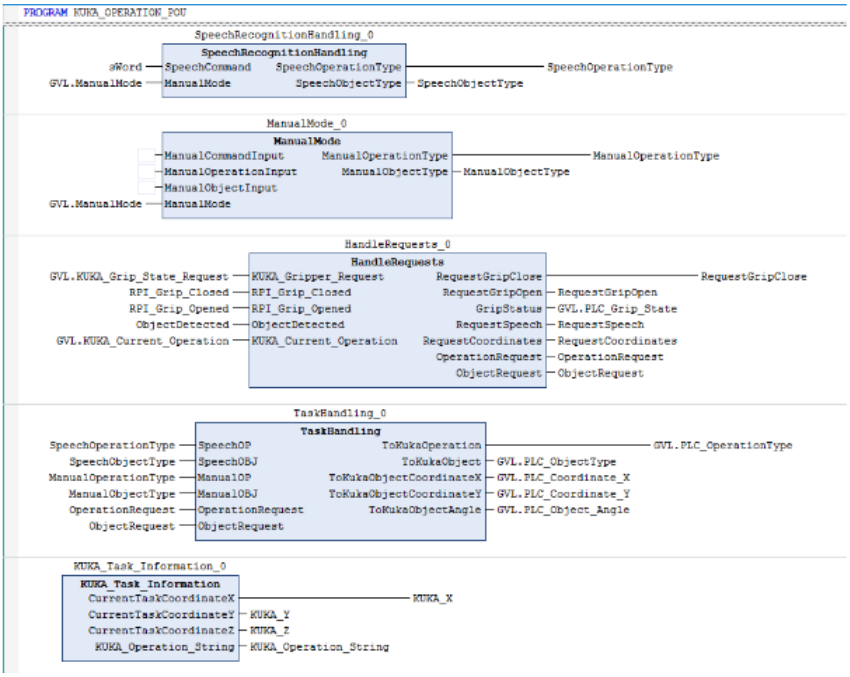


Figure 37: The main program, KUKA_OPERATION_POU

7.6.1 Request Handling

The request handling is implemented in the form of a Function Block which joins all necessary request variables across the system as shown in fiugure 38.

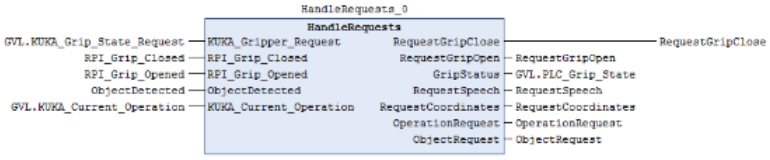


Figure 38: 'HandleRequests' Function block

Requests to be handled for the gripper are fairly trivial. The program should determine whether to open or close the gripper based on the feedback coming from the Python script. Solution is

purely conditionary as KUKA LBR IIWA is responsible for knowing and informing the system whether the gripper should be manipulated. Incoming request from the robot is an integer, either 0 or 1 contained in variable KUKA_Gripper_Request. Based on that, HandleRequests commands a Python script to Close or Open the gripper by triggering boolean variables RequestGripOpen or RequestGripClose.

As LBR IIWA is responsible for requesting the gripper action, it must receive feedback over the GripStatus variable further linked to PLC_Grip_State. Handling of requests is shown in figure 39.

```
//KUKA TO RPI GRIPPER REQUESTING
IF KUKA_Gripper_Request = 1 AND NOT RPI_Grip_Opened THEN
    RequestGripOpen := TRUE;
    RequestGripClose := FALSE;
    ELSIF KUKA_Gripper_Request = 0 AND NOT RPI_Grip_Closed THEN
    RequestGripOpen := FALSE;
    RequestGripClose := TRUE;
    ELSE
    RequestGripOpen := FALSE;
    RequestGripClose := FALSE;
END_IF

IF RPI_Grip_Closed THEN
    GripStatus := 0;
ELSIF RPI_Grip_Opened THEN
    GripStatus := 1;
END_IF
```

Figure 39: 'HandleRequests' Function block

Requesting the Speech Recognition input along with Tool coordinates from Raspberry PI is triggered only after receiving IDLE status from KUKA robot. The IDLE status is sent straight from the robot in the form of an UINT value named KUKA_Current_Operation, where value 0 depicts Idle state. Request handling for coordinates and speech recognition is shown in figure 40.

```
IF KUKA_Current_Operation = 0 THEN

    RequestCoordinates := TRUE;
        IF GVL.ManualMode THEN
            RequestSpeech := FALSE;
        ELSE
            RequestSpeech := TRUE;
        END_IF
ELSE
    RequestSpeech := FALSE;
    RequestCoordinates := FALSE;
END_IF
```

Figure 40: Speech Recognition and Coordinates request handling.

Similarly to handling external devices, the inner Twincat 3 program relies heavily on two particular requests, requesting the object and requesting the operation to be performed. Both are considered user inputs, however certain situations require only one or both of those inputs, therefore the program must react accordingly and request proper inputs. In the program, both inputs are requested during the IDLE state of the robot, however ObjectRequest additionally relies on Gripper's object detection in order to avoid commanding pick-up operations while already wielding a tool. Task requesting is shown in figure 41.

```
//TASK REQUEST

IF NOT ObjectDetected AND KUKA_Current_Operation = 0  THEN
    ObjectRequest := TRUE;
ELSE
    ObjectRequest := FALSE;
END_IF
IF KUKA_Current_Operation = 0 THEN
    OperationRequest := TRUE;
ELSE
    OperationRequest := FALSE;
END_IF
```

Figure 41: Task requesting in structured text

7.6.2 User input handling

In the project, users can provide inputs through two separate methods. One of them is Speech Recognition, which is the main idea of the project. Second option is manual input which yields the solution operational as long as necessary components are running. It's also a more convenient way for debugging, testing and operating in loud environments.

For the purpose of the project, Speech recognition recognizes four different operation types and three different tools and considers them inputs. The python script passes a string value to Twincat 3 for further processing. The value has a structure of sentence and a program should pick potential input values and separate them into operations and objects. The method used to select a word of interest is FIND() function, used in Structured Text language as shown in figure 42.

```
PickUp := FIND(SpeechCommand, 'pick up');
Drop := FIND(SpeechCommand, 'drop');
Give := FIND(SpeechCommand, 'give');
Hold := FIND(SpeechCommand, 'hold');


Screwdriver := FIND(SpeechCommand, 'screwdriver');
Hammer := FIND(SpeechCommand, 'hammer');
Pliers := FIND(SpeechCommand, 'pliers');
```

Figure 42: Program finding a particular word within a string.

Further, the program assigns correct OperationType value which ranges from 1 to 4 as shown in figure 43. Additionally, Pick-up operation requires existing object input.

- 1 represents Pick-Up operation
- 2 represents Drop operation
- 3 represents Give operation
- 4 represents Hold (IDLE) operation

For objects:

- 1 represents screwdriver
- 2 represents hammer
- 3 represents pliers

```
IF PickUp > 0 AND (Screwdriver + Hammer + Pliers) > 0THEN
    SpeechOperationType := 1;
ELSIF Drop > 0 THEN
    SpeechOperationType := 2;
ELSIF Give > 0 THEN
    SpeechOperationType := 3;
ELSIF Hold > 0 THEN
    SpeechOperationType := 4;
END_IF

IF Screwdriver > 0 THEN
    SpeechObjectType := 1;
ELSIF Hammer > 0 THEN
    SpeechObjectType := 2;
ELSIF Pliers > 0 THEN
    SpeechObjectType := 3;
END_IF

IF SpeechCommand = '' THEN
    SpeechObjectType := 0;
    SpeechOperationType := 0;
END_IF
```

Figure 43: A program assigning values to operations and objects.

Implementation of manual mode proceeds in a similar fashion, with mild modifications of the code. It was considered that manual mode should be split into three separate inputs. One as a main input for the users in the form of a single input string and two separate string inputs for operations and objects for debugging and testing purposes. The three inputs were integrated together, such that the user input splits and feeds two other inputs. This is done to maintain code clarity as no unintended interferences were observed.

User input with a proper command, looks for a separator of a blank space using previously mentioned FIND() function, then the two values are determined based on their symbol index positions in the string with respect to the separator. Operation type is assigned by looking left of the separator and Object type by the right. Naturally, if the separator does not exist, the full command will be passed, as not every command requires an attached object. Figure 44 shows how program splits command input into operation and object inputs.

```
IF LEN(ManualCommandInput) > 1 THEN

Separator := FIND(ManualCommandInput, ' ');

    IF Separator > 0 THEN
        ManualOperationInput := MID(ManualCommandInput, Separator-1, 1);
        ManualObjectInput := MID(ManualCommandInput, LEN(ManualCommandInput)-Separator+1, Separator+1);
    END_IF

    IF Separator = 0 THEN
        ManualOperationInput := ManualCommandInput;
    END_IF
```

Figure 44: The program splitting command input into Operation and Object inputs.

Furthermore, figure 45 shows how string outputs are determining corresponding operation and object types.

Figure 45: The program assigning values based on corresponding string inputs.

Additionally, for users convenience and easier input handling, the inputs and outputs will reset to empty strings and values of 0 as shown on figure 46. The robot does not follow commands while performing a task, however once the command and necessary data are passed, it is crucial to not order the same commands after successful execution. The solution of this safety related problem is based on a timer which will reset inputs and outputs of the program after two seconds.

```
IF LEN(ManualOperationInput) > 1 OR LEN(ManualObjectInput) > 1 THEN
    Delay(IN:=TRUE, PT:=T#2S);


    IF Delay.Q THEN
    ManualOperationType := 0;
    ManualObjectType := 0;
    Delay(IN := FALSE);
    ManualOperationInput := ' ';
    ManualObjectInput := ' ';
    ManualCommandInput := ' ';


    END_IF

END_IF
```

Figure 46: The reset logic of inputs and outputs within manual mode.

7.6.3 Task Handling

In the center of the Twincat 3 solution, as well as being the final step before commanding the Robot is the task handling. Requesting, inputs and logical operations meet at the last Function block, the 'TaskHandling'. This particular Function block, serves a purpose of directly commanding the robot, sending appropriate data and determining whether the inputs should be processed. It involves certain security measures and makes sure that unintended events do not occur.

First feature of TaskHandling is to once again ensure that the Pick-up operation cannot be conducted without an assigned object. In the request handling section, it is explained that the object should be requested at the idle state without detected object. However in this situation, the program makes sure that any input regarding picking up object operations, will be dismissed until the object request comes first. Figure 47 shows fail-proofing of the code.

```
IF OperationRequest THEN
    IF (ManualOP = 1 OR SpeechOP = 1) AND NOT ObjectRequest THEN
        ToKukaOperation := 0;
        ELSE

        IF GVL.ManualMode THEN
            ToKukaOperation := ManualOP;
        ELSE
            ToKukaOperation := SpeechOP;
        END_IF

    END_IF

END_IF


IF NOT OperationRequest THEN
    ToKukaOperation := 0;
END_IF
```

Figure 47: Fail-proofing of the code regarding operation inputs

Sending the object parameters is likely treated by fail-proofing. The code differentiates between input types and ensures that the object type sent towards KUKA robot will reset if the picking up operation is unsuccessful. It happens due to the fact that the robot sets its operation to IDLE after performing the task and during that time, request checks are performed, which allows the code to reset requested objects unless they were gripped properly. Figure 48 shows how code passes through object information.

```
IF ObjectRequest THEN
    ToKukaObject := 0;
    IF GVL.ManualMode THEN

        IF ManualOBJ > 0 THEN
            MsetOBJ := TRUE;
            IF MsetOBJ = TRUE THEN
                ToKukaObject := ManualOBJ;
                MsetOBJ := FALSE;
            END_IF
        END_IF
        ObjectCoordinates := UINT_TO_INT(ManualOBJ);
    ELSE

        IF SpeechOBJ > 0 THEN
            setOBJ := TRUE;
            IF setOBJ = TRUE THEN
                ToKukaObject := SpeechOBJ;
                setOBJ := FALSE;
            END_IF
        END_IF
        ObjectCoordinates:= UINT_TO_INT(SpeechOBJ);
    END_IF

END_IF
```

Figure 48: The code which passes through object information.

Lastly, the ObjectCoordinates switch-case, presented in figure 49, is responsible for extracting and passing through the object coordinates and angle. Based on the object type chosen, the case will assign correct values to the outputs which will be passed to KUKA robot. Coordinates and angles are required to be only positive integers to be sent through, therefore range shift is performed. Coordinates never exceed values over 900, therefore a 1000 is added to their original value and angles can range from -180 to 180, therefore 180 is added to the value before sending.

```
CASE ObjectCoordinates OF
    1:  //screwdriver

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Screwdriver_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Screwdriver_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Screwdriver_Angle)+180;

    2:

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Hammer_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Hammer_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Hammer_Angle)+180;

    3:

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Pliers_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Pliers_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Pliers_Angle)+180;

    0:

    ToKukaObjectCoordinateX := 0;
    ToKukaObjectCoordinateY := 0;
    ToKukaObjectAngle := 0;


END_CASE
```

Figure 49: ObjectCoordinates Switch case code.

7.6.4 Visualization and GUI

Visualization plays a big role in end-user experience. It represents all necessary data in one place which allows users to take actions more efficiently and be more informed about the proceedings. Together with input possibilities it forms a control panel from which the user can utilize and observe the prime aspects of the project.

Feedback of the KUKA LBR IIWA contains several monitoring variables. These are mainly the information about robots current coordinates X, Y, Z and the currently performed task. The incoming coordinate data requires range shift as well. On the side of the robot, value is increased by 1000 and decreased by 1000 on the Twincat 3 side. Furthermore, the current operation is assigned corresponding string values for better understanding for the user. Figure 50 shows the function block about task information.

```
CurrentTaskCoordinateX:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_X)-1000;
CurrentTaskCoordinateY:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_Y)-1000;
CurrentTaskCoordinateZ:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_Z)-1000;

IF GVL.KUKA_Current_Operation = 1 THEN
    KUKA_Operation_String := 'Pick Up';
ELSIF GVL.KUKA_Current_Operation = 2 THEN
    KUKA_Operation_String := 'Drop';
ELSIF GVL.KUKA_Current_Operation = 3 THEN
    KUKA_Operation_String := 'Give';
ELSIF GVL.KUKA_Current_Operation = 0 THEN
    KUKA_Operation_String := 'Hold';
END_IF
```

Figure 50: The code of KUKA_Task_Information Function block

Graphical representation of the project, shown in the figure 51, serves the purpose of a compact control panel with necessary control data and inputs. Control data displays real-time robot coordinates, current performing tasks and grasped objects. In addition, it is paired with real-time tracking of tools coordinates supplied by Raspberry PI and gripper information. Two remaining panels are for observing and declaring inputs in either Speech Dashboard or Manual Mode.

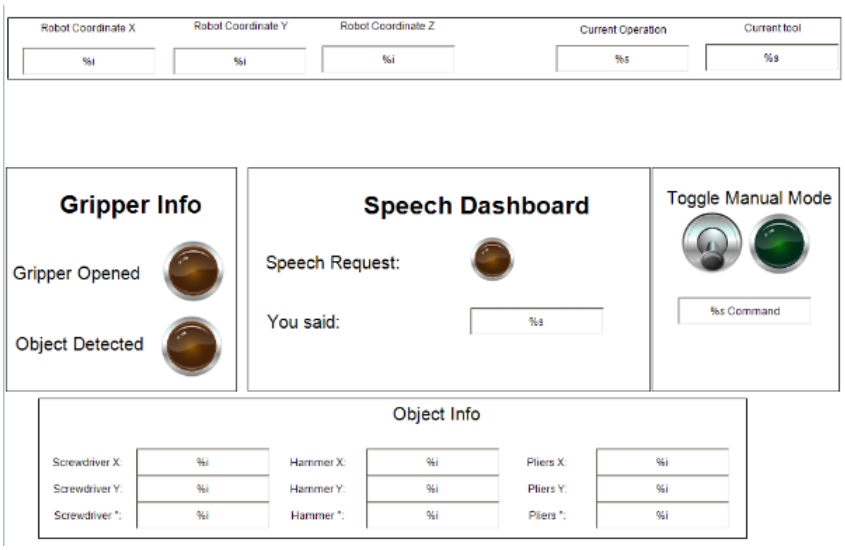


Figure 51: Visualization of data and GUI of the Project.

7.7 KUKA LBR IIWA PROGRAM

KUKA LBR IIWA is the main subject of control within the project, as well as the end receiver of commands put together by the rest of external devices. The main idea of the programming part is for the robot to be fully under control of the user through the Twincat 3, yet have its own safety measures and automatized internal controls and reactions.

The use of the robot is purely movement based. In the project, two different methods of executing moves were used.

- PTP - point to point movement, utilized mainly to manipulate joints into certain configurations or achieve specific coordinate position. Executed via built in ptp() functions of the Sunrise Workbench
- Lin - linear motion, used to move linearly along one axis in cartesian space.

In order to ensure safety, motions are velocity limited on the software side. It is done through setJointVelocityRel() method which is an attribute of movements.

As Java is an Object-oriented programming language, code is separated into classes based on their area of use. The division of classes is as follows:

- Class RobotApplication - main program which controls the robot.
- Class RobotCurrentInfo - provides real-time information for Twincat 3.
- Class RobotUtil - contains movement sets and utility tools.
- Class RobotTestUtil - contains calibration, testing tools and experimental movement sets.

7.7.1 Initialization

One of the requirements for a safe operation is an initialization of the robot after closing the program or fully shutting down the robot. A shutdown can occur at any position, during any performed task while grasping any object. To counteract any potential hazard regarding the topic, a method initializeRobotPosition() is always executed upon the start of the program.

The code in figure 52 creates a frame, which is a cartesian representation of a robot's position in space. The position is measured at the tip of the flange (gripper). Based on that frame it detects any potentially hazardous position and commands the robot to move into safe space and into a mechanical zero position. The movements are executed in slow linear fashion. Once the robot is at mechanical zero position, all its joints are free of angle rotations.

```
public static void initializeRobotPosition(LBR robot){
    Frame startFrame = robot.getCurrentCartesianPosition(robot.getFlange());
    System.out.println("Robot start position: " + startFrame);
    Frame ColisionStartFrame = startFrame.copyWithRedundancy();

    if (startFrame.getY() > 400){ //If robot is initialized from Drop position
        ColisionStartFrame.setY(500);
        robot.move(lin(ColisionStartFrame).setJointVelocityRel(0.1));
    }
    if (startFrame.getY() < -400){ //If robot is initialized from pickup position
        ColisionStartFrame.setZ(350);
        robot.move(lin(ColisionStartFrame).setJointVelocityRel(0.1));
        PTP HOLDPrepareJointPosition = ptp(0,0.715,0,-1.17,0,0,0).setJointVelocityRel(0.2);
        robot.move(HOLDPrepareJointPosition);
    }
    PTP ptpToMechanicalZeroPosition = ptp(0,0,0,0,0,0,0);
    ptpToMechanicalZeroPosition.setJointVelocityRel(0.25);
    robot.move(ptpToMechanicalZeroPosition);
```

Figure 52: InitializeRobotPosition() method within RobotUtil class.

7.7.2 Real-time information

Programming of LBR IIWA in the Java environment opens up possibilities of extracting important information without interrupting any operations. Method of choice is Threading, which can be used to provide the data for exchange and execute the main program at the same time.

Extracting the data is based on constantly requesting the current cartesian position of the flange and further extracting specific axes. It is done by utilizing the method getCurrentCatesianPosition(robot.getFlange()) and further passing it to EL6695 variables. Additionally, thread can be suppressed after each execution, to not unnecessarily consume resources.

RobotCurrentInfo, shown in figure 53 is a separate class which extends the Thread class. It is enabled during the initialization phase of the program.

```
public class RobotCurrentInfo extends Thread{
    private LBR robot;
    private EL6695IOGroup io;

    RobotCurrentInfo(LBR robot, EL6695IOGroup io){

        this.robot = robot;
        this.io = io;
    }
    public void run(){
            System.out.println("Robot info thread running...");
            System.out.println("-------------------------------------");

        while(RobotApplication.startInfoThread){

            int Transform_Robot_X = (int)robot.getCurrentCartesianPosition(robot.getFlange()).getX()+1000;
            int Transform_Robot_Y = (int)robot.getCurrentCartesianPosition(robot.getFlange()).getY()+1000;
            int Transform_Robot_Z = (int)robot.getCurrentCartesianPosition(robot.getFlange()).getZ()+1000;

            io.setKUKA_TaskInfo_Coordinate_X(Transform_Robot_X);
            io.setKUKA_TaskInfo_Coordinate_Y(Transform_Robot_Y);
            io.setKUKA_TaskInfo_Coordinate_Z(Transform_Robot_Z);

            try{
                Thread.sleep(300);
            }
            catch(Exception err){

            }
        }
    }


}
```

Figure 53: Implementation of RobotCurrentInfo class.

7.7.3 Main program

The main program is a part of RobotApplication class and is based on a constantly checked and looped switch-case. It heavily uses other classes and builds logic around them. The program is mostly scripted movement arrangements, as it's expected from industrial robots. Each case represents an operation type which the robot can perform.

Case 1: Pick up Operation

The pick up operation in figure 54 starts with a countermeasure for falsely passed object information. It checks if the coordinates are within the area of pick up operation and if not, it cancels the execution. Once the object is validated, it can proceed to the 'PickupPrepare()' position, which is a predetermined joint setup over tools.

```
public static void PickupPrepare(LBR robot){
    PTP PickupPrepareJointPosition = ptp(0, 0.523, 0, -1.57,0, 1.047, 0).setJointVelocityRel(0.2);
    robot.move(PickupPrepareJointPosition);
    PTP PickupPrepareJointPosition2 = ptp(-1.57, 0.523, 0, -1.57,0, 1.047, 0).setJointVelocityRel(0.25);
    robot.move(PickupPrepareJointPosition2);
```

Figure 54: PickupPrepare() method.

During the procedure, the program requires certain actions from the gripper. Until the synchronization and proper feedback is received, the robot stalls to avoid unintended behavior. Upon continuation, object position and parameters are passed for a new movement set in which the robot moves over the object and adjusts the angle for perfect grasp. Figure 55 shows a pick up case code.

```
switch(io.getPLC_OperationType()){

case 1:  //pickup command
    io.setKUKA_AnswerRequest(0);
    io.setKUKA_Current_Operation(1);

    int ObjectX = io.getPLC_Coordinate_X()-1000;
    int ObjectY = io.getPLC_Coordinate_Y()-1000;
    double ObjectAngleRad = Math.toRadians(io.getPLC_Object_Angle()-180);

    if (ValidateCoordinates(ObjectX,ObjectY)){
        RobotUtil.PickupPrepare(robot);

        while(GripperClosed(io.getPLC_Grip_State())){   //while grip is closed
            io.setKUKA_Grip_State_Request(1);   //open grip
            ThreadUtil.milliSleep(500);
        }
        System.out.println(ObjectAngleRad*0.0174532);
        RobotUtil.PickupPrepareObjectPosition(robot, ObjectX, ObjectY, 285, ObjectAngleRad);

        while(GripperOpened(io.getPLC_Grip_State())){   //while grip is opened
            io.setKUKA_Grip_State_Request(0);  //close grip
            ThreadUtil.milliSleep(500);
        }
        RobotUtil.afterPickup(robot);
    }
    ThreadUtil.milliSleep(3000);
        io.setKUKA_Current_Operation(0);

    break;
```

Figure 55: Pick up case.

Random or not predicted angle of rotated object poses certain issues. The flange of the robot is not able to rotate full 360 degrees around the axis. Instead, the range of rotation is -175 degrees to 175 degrees. The gripper must take the long path of 350 degrees in order to rotate from one end to another. Additionally, once the robot moves to specific coordinates, its flange already rotates the flange x degrees with respect to tool position. That being said, a tool, whose angle is evaluated only with respect to the pick-up area, will create an offset angle for the flange based on its position with respect to the robot's base . The problem cannot be addressed by always placing the flange at 0-angle rotation with respect to the pickup area and then rotating the flange with respect to the tool, as it can lead to axis rotation violations. Figure 56 shows a representation of the angle

problem.

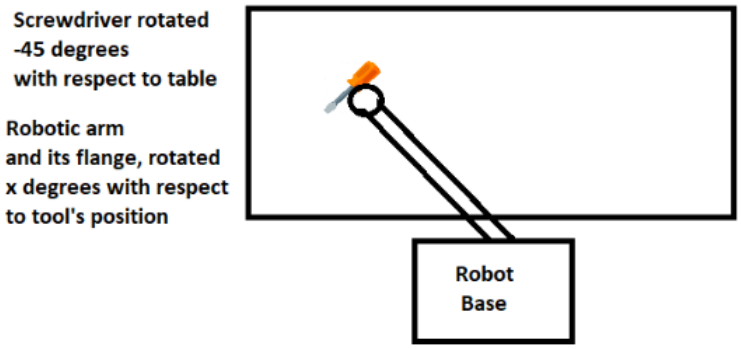


Figure 56: Simple representation of the angle problem

In order to address the problem, the offset angle must be known. By commanding the KUKA robot to set its flange 0 degrees with respect to the pickup-area after placing it over the object, it's possible to read the value of Joint rotation (offset angle) responsible for rotating the flange. With compensating for the offset angle, the calculation of the final rotation is Offset Angle + Tool Angle. Only that poses a risk of rotating the flange under -175 degrees or over 175 degrees. To counter that, a special algorithm shown in figure 57 will transform the tool's angle with respect to the robot's offset angle and determine which path of rotation is shorter and does not violate axis limits. Generally, it is impossible to avoid 180 or -180 degree rotations in unfortunate cases, however 5 degree error is negligible.

```
double J1= robot.getCurrentJointPosition().get(0);
double J2= robot.getCurrentJointPosition().get(1);
double J3= robot.getCurrentJointPosition().get(2);
double J4= robot.getCurrentJointPosition().get(3);
double J5= robot.getCurrentJointPosition().get(4);
double J6= robot.getCurrentJointPosition().get(5);
double Joint7Offset= robot.getCurrentJointPosition().get(6);

double angleFix = 0;
if (a + Joint7Offset > 3.05){
    angleFix = -((Math.PI - Joint7Offset) + (Math.PI - a));
        if (-angleFix > a + Joint7Offset){
            angleFix = a+Joint7Offset;
        }
}
else if (a + Joint7Offset < -3.05){
    angleFix = ((-Math.PI - Joint7Offset) - (Math.PI + a));
        if (-angleFix < a + Joint7Offset){

            angleFix = a + Joint7Offset;
        }
}
else {
    angleFix = a+Joint7Offset;
}


if (angleFix > 3.05) {
    angleFix = 3.05;
}
else if (angleFix < -3.05) {
    angleFix = -3.05;
}

robot.move(ptp(J1,J2,J3,J4,J5,J6,angleFix).setJointVelocityRel(0.1));
Frame OverObject = robot.getCurrentCartesianPosition(robot.getFlange());
System.out.println("OverO: " + OverObject);

Frame DownToObject = OverObject.copyWithRedundancy();
DownToObject.setZ(z);
robot.move(lin(DownToObject).setJointVelocityRel(0.10));
```

Figure 57: Angle transformation algorithm along with movements performed to prepare for object grasping.

2: Drop (placing) operation

The drop operation is fundamentally based on an internal switch case, where different objects are separate cases. Such design is based on the fact that different tools benefit from different movement sets, which allow for safer and more elegant procedure of placing. Twincat 3 program is responsible for determining whether the tool is ready to be placed and will not allow for the operation to be triggered without an object. The dropping sequence consists of moving the robot to neutral and safe space and slowly approaching its destined place. Upon reaching the place, KUKA orders the gripper to be opened and safely returns to neutral position. The code for drop operation is shown in figure 58.

```
case 2:  //drop command
        //prepare
    io.setKUKA_AnswerRequest(0);
    io.setKUKA_Current_Operation(2);
    while (io.getKUKA_Current_Operation() == 2){
        RobotUtil.DropPrepare(robot);

        switch(io.getPLC_ObjectType()){

        case 1:  //drop Screwdriver
            RobotUtil.DropScrewdriverPrepare(robot);
            while(GripperClosed(io.getPLC_Grip_State())){   //while grip is closed
                io.setKUKA_Grip_State_Request(1);  //open grip
                ThreadUtil.milliSleep(500);
            }

            ThreadUtil.milliSleep(4000);
            RobotUtil.AfterDropPosition(robot);
            io.setKUKA_Current_Operation(0);
            break;
        case 2: //drop Hammer
            RobotUtil.DropHammerPrepare(robot);

            while(GripperClosed(io.getPLC_Grip_State())){   //while grip is closed
                io.setKUKA_Grip_State_Request(1);  //open grip
                ThreadUtil.milliSleep(500);
            }

            ThreadUtil.milliSleep(4000);
            RobotUtil.AfterDropPosition(robot);
            io.setKUKA_Current_Operation(0);

            break;
        case 3: //drop pliers
            RobotUtil.DropPliersPrepare(robot);

            while(GripperClosed(io.getPLC_Grip_State())){   //while grip is closed
                io.setKUKA_Grip_State_Request(1);  //open grip
                ThreadUtil.milliSleep(500);
            }
            ThreadUtil.milliSleep(4000);
            RobotUtil.AfterDropPosition(robot);
            io.setKUKA_Current_Operation(0);

            break;
        }
    }
```

Figure 58: The case which controls the dropping sequence.

3: Give

Give is a simple conditional command which orders the robot to release the tool when in neutral position. It can only be triggered when the object is present. Give command is shown in figure 59.

```
case 3:  //give command
    System.out.println("give object");
    io.setKUKA_Grip_State_Request(1);  //open grip
    ThreadUtil.milliSleep(500);
    io.setKUKA_Current_Operation(0);
    break;
```

Figure 59: The code of Give operation

0: HOLD (IDLE)

Hold is the neutral position of the robot, from where any task can be performed. Only during the time of IDLE, the robot will receive any commands and will allow for request handling in the TwinCAT 3 environment. Case for Hold operation is shown in figure 60.

```
case 0: //HOLD
    io.setKUKA_Current_Operation(0);
    ThreadUtil.milliSleep(500);
    PTP HOLDPrepareJointPosition = ptp(0,0.715,0,-1.17,0,0,0).setJointVelocityRel(0.2);
    robot.move(HOLDPrepareJointPosition);
    //waiting for new operation
    break;
```

Figure 60: The code of hold (idle) operation.

7.8 Speech Recognition application

This part of the chapter will describe how Speech Recognition application in the Python environment should be established. In the project it will be used to recognize words spoken by the user. The hardware that is used for Speech Recognition application is stationary PC and an USB headset with microphone.

7.8.1 PyAudio SR libraries

In order to start using a Speech Recognition application in Python, first of all, the installation of python libraries should be done. There are three libraries that are required for this application:

SpeechRecognition package - this package gives an opportunity to recognize speech from audio files that are recorded in ".wav" or ".mp3" formats [41].

PyAudio package - this package is required for application to be able to recognize voice from the connected microphone in real time [29].

7.8.2 Speech Recognition Setup

For speech recognition application PyCharm IDE is used. The package installation can be done through the terminal in PyCharm. Before installation, it's necessary to check which Python version is active in the Python IDE. By printing the command "print(sys.version)" the output provides information about which Python version is in use. Example of checking version of python is shown in figure 61. Python versions 3.8 and above are required for speech recognition library.

Figure 61: Printing python version in PyCharm IDE

Now that required version of Python is active, all necessary libraries can be downloaded for the future application. All the packages can be installed through the terminal by typing specific commands. "SpeechRecognition" package can be installed by running "pip install SpeechRecognition" in the terminal as shown at figure 62.

Figure 62: Installing SpeechRecognition library in python

“SpeechRecognition” - package could be enough if the task is to recognize voice recording from audio files and translate them into text. However this package alone is not sufficient for real time speech recognition. “PyAudio” - package gives an opportunity to recognize voice from the microphone and print it out in the terminal in real time. Installing “PyAudio” is done in the same way as installing “SpeechRecognition”. By running the command “pip install pyaudio” in the terminal as shown in the figure 63, ”PyAudio” will be installed for Python 3.

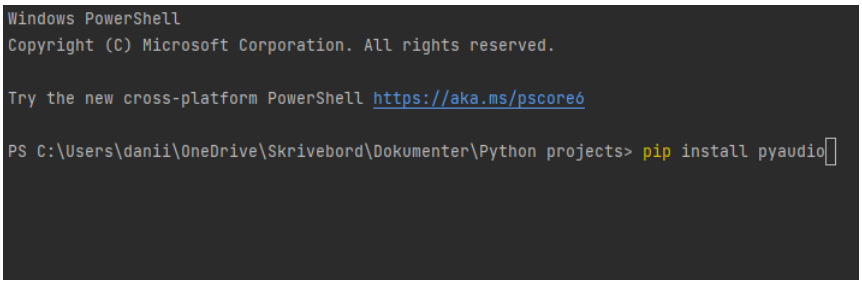


Figure 63: Installing PyAudio library in python

Now that all the necessary packages are installed, the Speech Recognition application can be used in Python IDEs.

7.8.3 Speech Recognition application

There are many possible IDEs that can be used for Python development. For speech recognition application ”PyCharm v2022.1.3” IDE is used. ”Speech recognition” python library has all the necessary commands to create simple application of speech recognition.

The figure 64 bellow shows simple Speech Recognition application that can recognize voice from the microphone:

```python
import speech_recognition as sr

def speech():
    while True:
            r = sr.Recognizer()
            # Speech Recognition part:
            with sr.Microphone() as source:
                print("Say something!")
                audio = r.listen(source)
                # recognize speech using Google Speech Recognition
                try:
                    text = r.recognize_google(audio)
                    print(f"You said: {text}")
                except sr.UnknownValueError:
                    print("Google Speech Recognition could not understand audio")
                except sr.RequestError as e:
                    print(f"Could not request results from Google Speech Recognition service; {e}")

```

Figure 64: Recognizing voice commands from microphone

In the beginning the "speech_recognition" package is imported. Next step is to create a function named "speech()" that will recognize spoken words. Inside a function is a "while"-loop that will run the code inside a function continuously. In "while"-loop an instance will be created which contains a "Recognizer()" class. The purpose of the "Recognizer()" class is to recognize speech. The code is listening to the audio from the microphone, which is a source, and then using "Google Speech Recognition" - algorithm to recognize the voice and transform it into text which will be printed out in the output terminal. Figure 65 shows output where the application is waiting for the user's input from the microphone. Figure 66 shows a sentence that has been said by the user and recognized by the "speech()" function. The sentence that is said by the user is a command to pick up a hammer. Output in figure 66 also shows alternative results of speech recognition and which of the alternatives have highest confidence. Users can say different commands such as "pick up screwdriver", "pick up pliers", "give" and "drop" a tool. It is important to note that this package require an internet connection while the script is running, otherwise the program will not work properly.

```
"C:\Program Files\Python39\python.exe" "C:/Users/danii/Desktop/Dokumenter/My Python/main.py"
Say something!
```

Figure 65: Speech recognition application waiting for the user to say something

```
"C:\Program Files\Python39\python.exe" "C:/Users/danii/Desktop/Dokumenter/My Python/main.py"
Say something!
result2:
{   'alternative': [   {'confidence': 0.82457, 'transcript': 'pick up hammer'},
                       {'transcript': 'pick-up Hummer'},
                       {'transcript': 'pick-up hammer'},
                       {'transcript': 'make-up hamper'},
                       {'transcript': 'pick-up Hama'}],
    'final': True}
You said: pick up hammer
```

Figure 66: Speech recognition application output with recognized words

7.9 Server/Client communication between Raspberry and PC

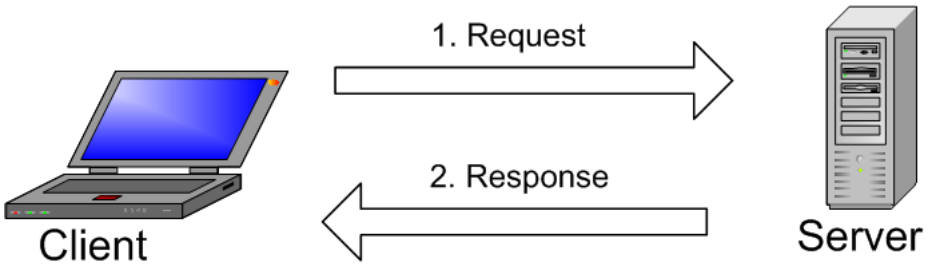


Figure 67: Simplified demonstration of server client communication

In order to send information between devices, a "Server/Client" application can be created by using socket programming. The devices that should communicate are Raspberry Pi and a local PC.

"Server/Client" communication will be established between those devices. In this application Raspberry Pi will act as a server while a local PC will have a role of a client as shown on figure 67. First step is to assign a static IP address to both Raspberry Pi and PC. It is important to note that IP addresses should be on the same subnet in order for communication to happen. There are two ways of setting up a connection. Raspberry Pi and PC can be connected together by using Ethernet cable and assigned with an IP address which is on the same subnet . Another method is to connect Raspberry Pi and PC to a common router using Ethernet cables and choosing DHCP server connection.

Raspberry Pi's assigned static IP address is "192.168.0.101". For PC a static IP address is "192.168.0.103". Python code for a "Server"-part is created first. This "Server" part will be run as a python script on Raspberry Pi. The complete code shown below on figure 68:

```python
import time
import socket


HOST = '192.168.0.103'  # IP address of the Raspberry Pi
PORT = 8000  # Port used for connection

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)

# Waiting for a client to connect to the Raspberry Pi
print('Waiting for a client to connect to the server...')
conn, addr = s.accept()
print('Connected by', addr)

data = conn.recv(1024)
print(data.decode())

while True:
    myData = 'tool data'
    conn.sendall(str(myData).encode())
    time.sleep(0.1)

```

Figure 68: Python code for server that is run on Raspberry Pi

The code first imports an built-in "socket" library that is needed for socket programming in python. The "HOST" is an IP of a device that will act as a server. The "PORT" for connection is chosen to be "8000" and it's important that both server and client use the same port number. When the

"Server" script is run, it will wait for client devices to connect to the server and give the following message at the output as shown on figure 69:

```
C:\Users\danii\AppData\Local\Programs\Python\Python39\python.exe
Waiting for a client to connect to the server...

Process finished with exit code -1
```

Figure 69: Raspberry Pi waiting for client device to connect

"Client" part of the application can be created for a local PC that will connect to the Raspberry Pi server. Figure 70 demonstrates an example of client application made in python:

```python
import socket as socket


def ObjectDetectionRPI():
        HOST = '192.168.0.101'  # This is the Raspberry Pi's hostname or IP address
        PORT = 8000  # The same port as the server

        # Create a socket object
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # Connect to the server
        s.connect((HOST, PORT))
        while True:
            # Send a message to the server
            s.sendall('Recieving...'.encode())
            data = s.recv(1024).decode()
            print(data)

ObjectDetectionRPI()

```

Figure 70: Python code for client that is run on local PC

The "Client" script uses Raspberry Pi's IP address which is an IP address the client tries to connect to. The "PORT" in "Server/Client" communication is common - "8000". PC establishes connection to the server using information about IP address and port. The server is waiting for the external connection from client devices. When the "Client" script is run on PC, it connects to the Raspberry Pi. The output on Raspberry Pi tells that client device has been connected and client is receiving data from the server as shown on figure 71:

```
C:\Users\danii\AppData\Local\Programs\Python\Python39\python.exe
Waiting for a client to connect to the server...
Connected by ('192.168.0.102', 8727)
Recieving...
```

Figure 71: Output on Raspberry Pi showing that client is connected and client receiving data

The PC receives tool data from Raspberry Pi after the connection is established. The message PC gets back is "tool data" and is showed as an output as in figure 72 below:

```
tool data
tool data
tool data
tool data
tool data
tool data
tool data
tool data
```

Figure 72: Output on local PC showing that message is received

Such a simple way of communication can be established between more than two devices. Raspberry Pi can still be used as a server, but more devices could be connected as clients that send information back and forth to the server.

7.10 Raspberry Pi and Tool recognition

This part of the section presents the method of using "OpenCV" computer vision library in Python to detect the position of the tools lying on a flat surface. Python script is run on Raspberry Pi 4 board.

7.10.1 Practical setup

The Python script presented in this section is able to recognize the position and orientation of three different tools that were selected for this project: hammer, screwdriver and pliers. To recognize tools that have different shapes and sizes, color bands are used. Each band contains two colors: one primary-bottom color and secondary upper color. By having a band with primary bottom color and secondary upper color it is possible to determine the orientation of the tool that lies on the surface. Figure 73 shows different color bands selected for the hammer screwdriver and pliers.

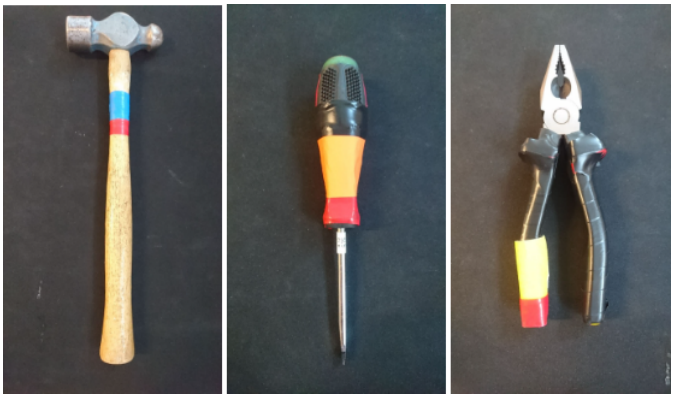


Figure 73: Three tools with color bands: hammer, screwdriver and pliars

The surface on which tools are lying, reflects light which in return could affect how the camera recognizes colors on the color bands. To minimize the effect of light reflection from the surface, a computer mouse pad of black color is used under the tools. Figure 74 demonstrates tools on the surface of the table with black mouse pad.

Figure 74: Tools with mouse pad lying on table surface

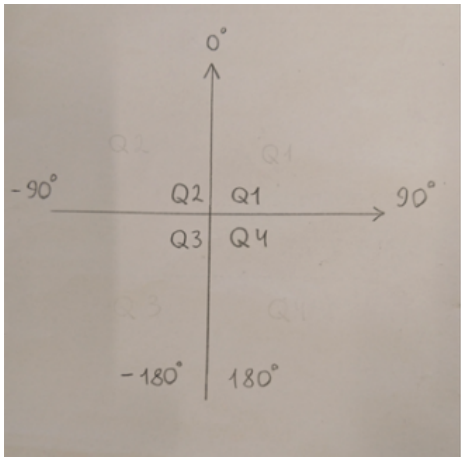
A USB Web camera that is used for tool recognition is placed above the surface of the table by

using aluminium profiles as shown on figure 75.

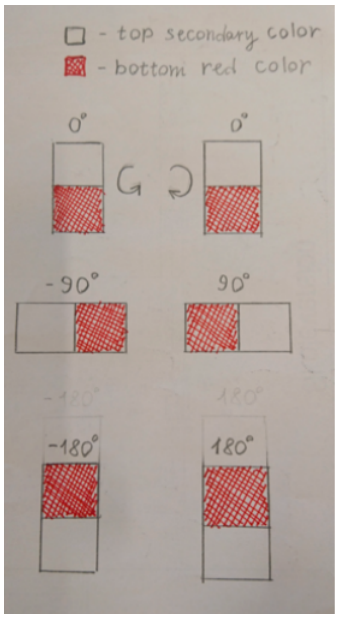
Figure 75: Camera stand made from aluminium profiles

7.10.2 Method of calculating angle for color bands

The comparison of coordinates of the bottom and top rectangles is needed for angle calculation of the color band. For this project it has been chosen that an object can have both positive and negative angles. Figure 76a shows a coordinate system with positive angles on the right side and negative angles on the left side.



(a) Camera stand made from aluminium profiles



(b) Change of angle for color bands in rotation

Figure 76: Sketch of angle calculation for tools

Coordinate system from figure 76a has four quadrants. Quadrant 1 (Q1) is from 0 to 90 degrees. Quadrant 2 (Q2) is from 0 to -90 degrees. Quadrant 3 (Q3) is from -90 to -180 degrees. Quadrant

4 (Q4) is from 90 to 180 degrees. The figure 76b demonstrates how color bands change angles when they are rotated in different directions. By rotating color bands clockwise, the angle changes from 0 to 180 degrees, while rotation in counter clockwise direction changes angle from 0 to -180 degrees.

- **Rotation from 90 to 180 degrees** - x coordinate of the top rectangle should be greater than x coordinate of red rectangle, while y coordinate of top rectangle is greater than y coordinate of red rectangle
- **Rotation from 0 to -90 degrees** - x coordinate of the top rectangle should be less than x coordinate of red rectangle, while y coordinate of top rectangle is less than y coordinate of red rectangle
- **Rotation from -90 to -180 degrees** - x coordinate of the top rectangle should be less than x coordinate of red rectangle, while y coordinate of top rectangle is greater than y coordinate of red rectangle

7.10.3 Python Code: Variable setup

First step is to define the lower and upper values for colors that should be detected. Packages that are required for this part are “cv2”, “time”, “numpy” and math. ”Cv2” and ”time” libraries are used for computer vision and time delays respectively. ”Numpy” is used to create arrays and ”math” is used for calculation of the tool’s angle. Figure 77 shows upper and lower limits for red, blue, orange and yellow colors as well as some color and string variables.

```
import cv2
import time
import numpy as np
import math

BlueColor = (255, 0, 0)
OrangeColor = (0, 200, 255)
GreenColor = (0, 255, 0)
RedColor = (0, 0, 255)
YellowColor = (0, 255, 255)

blueStr = 'Blue'
orangeStr = 'Orange'
redStr = 'Red'
yellowStr = 'Yellow'


# Define the lower and upper color thresholds for each color of the color band
lower_red = np.array([0, 100, 70])     #[166, 150, 189]
upper_red = np.array([15, 250, 170])    #[186, 250, 209]

lower_blue = np.array([90, 145, 205])  #[102, 176, 176]
upper_blue = np.array([120, 255, 255])  #[122, 255, 255]

lower_orange = np.array([10, 140, 180])   #[70, 25, 145]
upper_orange = np.array([25, 255, 255]) #[98, 125, 245]

lower_yellow = np.array([30, 70, 180])  #[21, 87, 157]
upper_yellow = np.array([50, 160, 255]) #[41, 187, 255]

```

Figure 77: Initialization of variables

Figure 78 shows a setup of a web camera with parameters such as USB port, weight and height of the displayed image, and framerate of the image.

```
camera_port = '/dev/video0'  # this is the camera port number (This can vary from, 0 - 10 from pc to pc)
cam = cv2.VideoCapture(camera_port)  # Set the camera capture device

width = 1024
height = 800

cam.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
cam.set(cv2.CAP_PROP_FPS, 30)
cam.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
```

Figure 78: Initialization of variables

7.10.4 Creating color detection function

For detection of the color band for each tool a function can be created. Figure 79 shows the function "MyTool()" with input arguments and a part that creates an "object box" around the color band.

```
42  def MyTool(myFrame, contours, topContours, color, colorName, Offset):
43      global xR, yR, angle
44      if contours:
45          for cnt in contours:
46              contour_area = cv2.contourArea(cnt)
47              if contour_area > 10:
48                  rect = cv2.minAreaRect(cnt)
49                  objectBox = cv2.boxPoints(rect)
50                  objectBox = np.int0(objectBox)
51
52                  # Get the parameters of the rotated bounding box
53                  x = int(rect[0][0])
54                  y = int(rect[0][1])
```

Figure 79: Creating function MyTool() that will recognize objects

"MyTool()" function uses five arguments that determine which color band should be detected. "myFrame" argument is a frame that is captured by a USB web camera. "contours" are contours that are created around the whole color band. "topContours" are contours that are created around the secondary upper color. "color" argument is a color of the "objectbox" that is created by the function. "Offset" argument tells a function if it needs to create an offset for the tool's coordinates. In figure 79 the box is created by using command "cv2.contourArea()" to calculate the area of the created contours. If the area of the contour area is greater than 10, a bounding rectangle with minimum area is created around the color band with command "cv2.minAreaRect()" [37]. In addition, coordinates x and y of the bounding rectangle are created as shown on lines 53 and 54 in figure 79.

Next step is to define an area with minimum and maximum coordinate values as shown on Figure 80.

```
56
57          black_area_x_min = -312
58          black_area_x_max = 312
59          black_area_y_min = -720
60          black_area_y_max = -401
61
62          scale_x = (black_area_x_max - black_area_x_min) / 1024
63          scale_y = (black_area_y_max - black_area_y_min) / 648
64
65          xR = int((black_area_x_max - (x * scale_x)) * 1.8) - 235
66          yR = int((black_area_y_min + (y * scale_y)) * 2.2) + 846
67
68          center = (x, y)
69          angle = int(rect[2])
70
71          robotCenterText = "Xr:" + str(xR) + " Yr:" + str(yR)
72          cv2.putText(myFrame, robotCenterText, (center[0], center[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1, cv2.LINE_AA)
73          cv2.circle(myFrame, (center[0], center[1]), 2, RedColor, 2)
74          cv2.drawContours(myFrame, [objectBox], 0, color, 2)
75
```

Figure 80: Definition of limits for an area of coordinates

Two variables are created for x coordinate, namely “black_area_x_min” and “black_area_x_max”. Same for y coordinates, namely “black_area_y_min” and “black_area_y_max”. These variables for x and y coordinate will be used to transform coordinates of the detected object into robot coordinates. Two more variables are needed in order to complete transformation from object- to robot coordinates, namely ”scale_x” and ”scale_y”. ”scale_x” and ”scale_y” coordinates will scale object coordinates in such a way that they will have a limit which is set by “black_area_x” and ”black_area_y” variables. To calculate ”scale_x” the difference between ”black_area_x_max” and ”black_area_x_min” is divided by frame width, 1024. In the same way to calculate ”scale_y” the difference between ”black_area_y_max” and ”black_area_y_min” is divided by frame height, 648.

Robot coordinates “xR” and yR are calculated by using variables “black_area_x_max”, ”black_area_y_min”, object coordinates x and y, and scaling variables ”scale_x ” and ”scale_y”. Robot coordinates ”xR” and ”yR” do not get best precision so scaling and offset can be adjusted to meet required minimum and maximum values for coordinate limits. ”xR” and ”yR” will result as robot coordinates that are transformed coordinates of the object within the black area. Black area that is used for object detection is shown at figure 81:

Figure 81: Black area for tool picking

On lines 68 and 69 on figure 80, “center” variable is created that stores x and y object coordinates, and “angle” variable which is a part of the rectangle list. “robotCenterText” is a string that is used to display robot coordinates of the object on the frame when the program is run. Command ”cv2.putText()” puts the text in the center of the detected object. The text will be displayed as robot coordinates calculated from scaling factors. ”cv2.FONT_HERSHEY_SIMPLEX” and

"cv2.LINE_AA" parameters are a font and line type of the text respectively. "color" parameter represents the color of the text. Command "cv2.circle()" on line 73 creates a dot in the center of the captured object. Finally, "cv2.drawContours()" draws a line around an object that forms a bounding box.

Every tool that should be recognized by a camera has a color band with red as primary bottom color and blue, orange, yellow as secondary top colors. Program using if-statement to check if the size of the red_contours and "topContours" are greater than 0. Then it creates bounding box around primary red color and secondary top color as shown on lines from 75 to 85 on figure 82:

```
75          if len(red_contours) > 0 and len(topContours) > 0:
76              red_contour = red_contours[0]
77              topContour = topContours[0]
78
79              xOffset = 25
80              yOffset = 85
81
82              # Get the bounding rectangles of the contours
83              red_rect = cv2.minAreaRect(red_contour)
84              top_rect = cv2.minAreaRect(topContour)
85
86              # If blue rectangle is on the top
87              if red_rect[0][1] > top_rect[0][1] and red_rect[0][0] <= top_rect[0][0]:
88                  angle = angle
89                  angleText = "Angle: " + str(angle)
90                  cv2.putText(myFrame, angleText, (center[0] - 100, center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 1, cv2.LINE_AA)
91                  if Offset is True:
92                      center_x = int(top_rect[0][0])
93                      center_y = int(top_rect[0][1])
94                      rect_angle = math.radians(angle)  # Convert angle to radians
95
96                      right_endpoint_x = center_x + xOffset
97                      right_endpoint_y = center_y - yOffset
98
99                      # Rotate the right endpoint around the center
100                     rotated_right_endpoint_x = center_x + (right_endpoint_x - center_x) * math.cos(rect_angle) - (right_endpoint_y - center_y) * math.sin(rect_angle)
101                     rotated_right_endpoint_y = center_y + (right_endpoint_x - center_x) * math.sin(rect_angle) + (right_endpoint_y - center_y) * math.cos(rect_angle)
102
103                     # Round the coordinates if needed
104                     rotated_right_endpoint_x = round(rotated_right_endpoint_x)
105                     rotated_right_endpoint_y = round(rotated_right_endpoint_y)
106
107                     # Sciling tool offset from x and y coordinates to xR and yR robot coordinates
108                     xToffset = int((black_area_x_max - (rotated_right_endpoint_x * scale_x)) * 1.8) - 235
109                     yToffset = int((black_area_y_min + (rotated_right_endpoint_y * scale_y)) * 2.2) + 840
110
111                     xR = xToffset
112                     yR = yToffset
113                     offsetCoordinates = "Xt:" + str(xR) + " Yt:" + str(yR)
```

Figure 82: Python code part for creating a bounding box, angle calculation and offset coordinates

On line 87 on figure 82, IF-statement checks if the "topColor" rectangle is on the top by comparing x and y coordinates of both "topRectangle" and bottom rectangle.

The angle calculation in python is done by checking which part of the color band is on the top and which is on the bottom. If the information about center coordinates of the bottom red color and top color rectangles are available, it's possible to calculate angle by comparing their center coordinates with each other. Center x and y coordinates of the red rectangle are "red_rect[0][0]" and "red_rect[0][1]" respectively. Center x and y coordinates of the top rectangle are "top_rect[0][0]" and "top_rect[0][1]" respectively.

On line 87 on figure 82 if statement checks if the x (red_rect[0][0]) coordinate of red rectangle is less or equal to the x (top_rect[0][0]) coordinate of the top rectangle, and at the same time if y (red_rect[0][1]) coordinate of red rectangle is greater than y (top_rect[0][1]) coordinate of top rectangle. If that is the case, the color band will get from 0 to 90 degrees as it rotates clockwise. On line 89 on figure 82 it creates a string of "angleText" that will display angle using command on line 90, "cv2.putText()". The code on figure 82 creates angle for the first quadrant Q1 of rotation, which is when the color band is rotated from 0 to 90 degrees. In a similar way IF statements can be made to calculate the angle for other cases.

Figure 83, figure 84 and figure 85 demonstrate python codes for angle calculation for intervals [90, 180], [0, -90] and [-90, -180] respectively:

```
elif red_rect[0][1] < top_rect[0][1] and red_rect[0][0] <= top_rect[0][0]:
    angle = 90 + angle
    angleText = "Angle: " + str(angle)
    cv2.putText(myFrame, angleText, (center[0] - 100, center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
                0.7, (0, 0, 0), 1, cv2.LINE_AA)
    if Offset is True:
        # Assuming you have the top rectangle 'top_rect'
        center_x = int(top_rect[0][0])
        center_y = int(top_rect[0][1])
        rect_angle = math.radians(angle)  # Convert angle to radians

        right_endpoint_x = center_x + xOffset
        right_endpoint_y = center_y - yOffset

        # Rotate the right endpoint around the center
        rotated_right_endpoint_x = center_x + (right_endpoint_x - center_x) * math.cos(rect_angle) - (right_endpoint_y - center_y) * math.sin(rect_angle)
        rotated_right_endpoint_y = center_y + (right_endpoint_x - center_x) * math.sin(rect_angle) + (right_endpoint_y - center_y) * math.cos(rect_angle)

        rotated_right_endpoint_x = round(rotated_right_endpoint_x)
        rotated_right_endpoint_y = round(rotated_right_endpoint_y)

        # Sciling tool offset from x and y coordinates to xR and yR robot coordinates
        xToffset = int((black_area_x_max - (rotated_right_endpoint_x * scale_x)) * 1.8) - 235
        yToffset = int((black_area_y_min + (rotated_right_endpoint_y * scale_y)) * 2.2) + 840

        xR = xToffset
        yR = yToffset
        offsetCoordinates = "Xt:" + str(xR) + " Yt:" + str(yR)

        # Draw the line connecting the rotated endpoints
        cv2.line(myFrame, (center_x, center_y), (rotated_right_endpoint_x, rotated_right_endpoint_y), (0, 0, 255), 2)
        cv2.circle(myFrame, (rotated_right_endpoint_x, rotated_right_endpoint_y), 2, GreenColor, 2)
        cv2.putText(myFrame, offsetCoordinates, (rotated_right_endpoint_x, rotated_right_endpoint_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1, cv2.LINE_AA)
    time.sleep(0.05)
```

Figure 83: Angle calcualtion 90 to 180 degrees

```
elif red_rect[0][1] > top_rect[0][1] and red_rect[0][0] >= top_rect[0][0]:
    angle = -(90 - angle)
    angleText = "Angle: " + str(angle)
    cv2.putText(myFrame, angleText, (center[0] - 100, center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 1, cv2.LINE_AA)
    if Offset is True:
        # Assuming you have the top rectangle 'top_rect'
        center_x = int(top_rect[0][0])
        center_y = int(top_rect[0][1])
        rect_angle = math.radians(angle)  # Convert angle to radians

        right_endpoint_x = center_x - xOffset
        right_endpoint_y = center_y - yOffset

        # Rotate the right endpoint around the center
        rotated_right_endpoint_x = center_x - (right_endpoint_x - center_x) * math.cos(rect_angle) - (right_endpoint_y - center_y) * math.sin(rect_angle)
        rotated_right_endpoint_y = center_y - (right_endpoint_x - center_x) * math.sin(rect_angle) + (right_endpoint_y - center_y) * math.cos(rect_angle)

        # Round the coordinates if needed
        rotated_right_endpoint_x = round(rotated_right_endpoint_x)
        rotated_right_endpoint_y = round(rotated_right_endpoint_y)

        # Sciling tool offset from x and y coordinates to xR and yR robot coordinates
        xToffset = int((black_area_x_max - (rotated_right_endpoint_x * scale_x)) * 1.8) - 235
        yToffset = int((black_area_y_min + (rotated_right_endpoint_y * scale_y)) * 2.2) + 840

        xR = xToffset
        yR = yToffset
        offsetCoordinates = "Xt:" + str(xR) + " Yt:" + str(yR)

        # Draw the line connecting the rotated endpoints
        cv2.line(myFrame, (center_x, center_y), (rotated_right_endpoint_x, rotated_right_endpoint_y), (0, 0, 255), 2)
        cv2.circle(myFrame, (rotated_right_endpoint_x, rotated_right_endpoint_y), 2, GreenColor, 2)
        cv2.putText(myFrame, offsetCoordinates, (rotated_right_endpoint_x, rotated_right_endpoint_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1, cv2.LINE_AA)
    time.sleep(0.05)
```

Figure 84: Angle calcualtion 0 to -90 degrees

```
157             elif red_rect[0][1] < top_rect[0][1] and red_rect[0][0] >= top_rect[0][0]:
158                 angle = -(180 - angle)
159                 angleText = "Angle: " + str(angle)
160                 cv2.putText(myFrame, angleText, (center[0] - 100, center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
161                             0.7, (0, 0, 0), 1, cv2.LINE_AA)
162                 if Offset is True:
163                     # Assuming you have the top rectangle 'top_rect'
164                     center_x = int(top_rect[0][0])
165                     center_y = int(top_rect[0][1])
166                     rect_angle = math.radians(angle)  # Convert angle to radians
167 
168                     right_endpoint_x = center_x - xOffset
169                     right_endpoint_y = center_y - yOffset
170 
171                     # Rotate the right endpoint around the center
172                     rotated_right_endpoint_x = center_x - (right_endpoint_x - center_x) * math.cos(rect_angle) - (right_endpoint_y - center_y) * math.sin(rect_angle)
173                     rotated_right_endpoint_y = center_y - (right_endpoint_x - center_x) * math.sin(rect_angle) + (right_endpoint_y - center_y) * math.cos(rect_angle)
174 
175                     # Round the coordinates if needed
176                     rotated_right_endpoint_x = round(rotated_right_endpoint_x)
177                     rotated_right_endpoint_y = round(rotated_right_endpoint_y)
178 
179                     # Sciling tool offset from x and y coordinates to xR and yR robot coordinates
180                     xToffset = int((black_area_x_max - (rotated_right_endpoint_x * scale_x)) * 1.8) - 235
181                     yToffset = int((black_area_y_min + (rotated_right_endpoint_y * scale_y)) * 2.2) + 840
182 
183                     xR = xToffset
184                     yR = yToffset
185                     offsetCoordinates = "Xt:" + str(xR) + " Yt:" + str(yR)
186 
187                     # Draw the line connecting the rotated endpoints
188                     cv2.line(myFrame, (center_x, center_y), (rotated_right_endpoint_x, rotated_right_endpoint_y), (0, 0, 255), 2)
189                     cv2.circle(myFrame, (rotated_right_endpoint_x, rotated_right_endpoint_y), 2, GreenColor, 2)
190                     cv2.putText(myFrame, offsetCoordinates, (rotated_right_endpoint_x, rotated_right_endpoint_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1, cv2.LINE_AA)
191                 time.sleep(0.05)
```

Figure 85: Angle calcualtion -90 to -180 degrees

Every part of angle calculation code includes an IF-statement that checks if the "Offset" parameter is true. The offset is needed specifically for calculation of the coordinates for the pliers. The reason pliers need an offset in coordinates is that the colorband for the pliers is attached to a handle, while it should be picked on the top by the robot. Without an offset parameter the Kuka robot will pick up pliers by the handle, which is not the best solution since the gripper will struggle to pick it up correctly. Offset part which is included under every IF-statement for angle calculation is presented in figure 86:

```
161                 if Offset is True:
162                     # Assuming you have the top rectangle 'top_rect'
163                     center_x = int(top_rect[0][0])
164                     center_y = int(top_rect[0][1])
165                     rect_angle = math.radians(angle)  # Convert angle to radians
166 
167                     right_endpoint_x = center_x - xOffset
168                     right_endpoint_y = center_y - yOffset
169 
170                     # Rotate the right endpoint around the center
171                     rotated_right_endpoint_x = center_x - (right_endpoint_x - center_x) * math.cos(rect_angle) - (right_endpoint_y - center_y) * math.sin(rect_angle)
172                     rotated_right_endpoint_y = center_y - (right_endpoint_x - center_x) * math.sin(rect_angle) + (right_endpoint_y - center_y) * math.cos(rect_angle)
173 
174                     # Round the coordinates if needed
175                     rotated_right_endpoint_x = round(rotated_right_endpoint_x)
176                     rotated_right_endpoint_y = round(rotated_right_endpoint_y)
177 
178                     # Sciling tool offset from x and y coordinates to xR and yR robot coordinates
179                     xToffset = int((black_area_x_max - (rotated_right_endpoint_x * scale_x)) * 1.8) - 235
180                     yToffset = int((black_area_y_min + (rotated_right_endpoint_y * scale_y)) * 2.2) + 840
181 
182                     xR = xToffset
183                     yR = yToffset
184 
185                     offsetCoordinates = "Xt:" + str(xR) + " Yt:" + str(yR)
186 
187                     # Draw the line connecting the rotated endpoints
188                     cv2.line(myFrame, (center_x, center_y), (rotated_right_endpoint_x, rotated_right_endpoint_y), (0, 0, 255), 2)
189                     cv2.circle(myFrame, (rotated_right_endpoint_x, rotated_right_endpoint_y), 2, GreenColor, 2)
190                     cv2.putText(myFrame, offsetCoordinates, (rotated_right_endpoint_x, rotated_right_endpoint_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1, cv2.LINE_AA)
191                 time.sleep(0.05)
```

Figure 86: Offset part of the code that creates offset coordinates

Figure 86 shows that new variables "center_x" and "center_y" are created that are center s and y coordinates of the top rectangle. Right endpoint is created on lines 167 and 168 which will be a new center with offset in x and y coordinates. Variables "xOffset" and "yOffset" that are used for right endpoint calculation are shown at figure 82. Then on lines 175 and 176 on figure 86, "rotated_right_endpoint" variables are created that will make the offset point rotate along the center of a rectangle as it moves. Rotated right endpoint coordinates should be transformed

to the new offset robot coordinates. It can be done by applying a scaling factor as shown on lines 179 and 180 on figure 86. Robot coordinates “xR” and “yR” will be assigned as robot offset coordinates “xToffset” and “yToffset”. From line 185 to 191, the function will create a string “offsetCoordinates”, create a line from the center of the rectangle to the offset point using “cv2.line()” command, create a dot for offset point using “cv2.circle()” command and display “offsetCoordinates” string as text using ”cv2.putText()” command.

Figure 87 shows that at the end of the function, robot coordinates with object angle are printed at the output. Else-sentence returns robot coordinates and angle as 0 if there are no objects present in the frame.

```
228
229                 print(f'{colorName} coordinates:', xR, yR, angle)
230
231     else:
232         xR = 0
233         yR = 0
234         angle = 0
235
236
237     return xR, yR, angle
238
```

Figure 87: Part of the function that prints coordinates at the output

7.10.5 Python Code: While loop

While-loop will contain all the parts of the code that should be run continuously. Figure 88 shows the first part of the loop for the color band detection program.

```
240 while True:
241     # Detect object coordinates part
242     ret, frame = cam.read()
243     frameROI = frame[100:435, 215:875]
244
245     # Convert the frame to the HSV color space
246     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
247     hsvROI = cv2.cvtColor(frameROI, cv2.COLOR_BGR2HSV)
248
249     # Perform color segmentation to detect the red color in the frame
250     red_mask = cv2.inRange(hsvROI, lower_red, upper_red)  # Change hsv to hsvROI if you want tracking in region of interest!
251     # Perform color segmentation to detect the blue color in the frame
252     blue_mask = cv2.inRange(hsvROI, lower_blue, upper_blue)  # Change hsv to hsvROI if you want tracking in region of interest!
253     # Perform color segmentation to detect the blue color in the frame
254     orange_mask = cv2.inRange(hsvROI, lower_orange, upper_orange)  # Change hsv to hsvROI if you want tracking in region of interest!
255     # Perform color segmentation to detect the blue color in the frame
256     yellow_mask = cv2.inRange(hsvROI, lower_yellow, upper_yellow)  # Change hsv to hsvROI if you want tracking in region of interest!
257
258     # Combine the red and blue masks
259     maskRedBlue = red_mask + blue_mask
260     maskRedOrange = red_mask + orange_mask
261     maskRedYellow = red_mask + yellow_mask
262
263     # Apply a series of morphological operations to remove noise and fill gaps
264     kernel = np.ones((5, 5), np.uint8)
265     maskRedBlue = cv2.morphologyEx(maskRedBlue, cv2.MORPH_OPEN, kernel)
266     maskRedBlue = cv2.morphologyEx(maskRedBlue, cv2.MORPH_CLOSE, kernel)
267
268     maskRedOrange = cv2.morphologyEx(maskRedOrange, cv2.MORPH_OPEN, kernel)
269     maskRedOrange = cv2.morphologyEx(maskRedOrange, cv2.MORPH_CLOSE, kernel)
270
271     maskRedYellow = cv2.morphologyEx(maskRedYellow, cv2.MORPH_OPEN, kernel)
272     maskRedYellow = cv2.morphologyEx(maskRedYellow, cv2.MORPH_CLOSE, kernel)
273
274     # Find contours of the detected regions in the mask
275     contoursRedBlue, _ = cv2.findContours(maskRedBlue, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
276     contoursRedOrange, _ = cv2.findContours(maskRedOrange, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
277     contoursRedYellow, _ = cv2.findContours(maskRedYellow, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 88: First part of the while-loop

First, the frame variable is created from the "cam.read()" command that captures a frame that is read from a web camera. Then a new frame is created that is called "frameROI". "frameROI" is a frame that has a region of interest which is a black area shown in figure 81. That frame with region of interest is made from the original frame by setting lower and upper limits of pixels in x and y direction. Next, an image is converted to the HSV color space. Red, blue, orange and yellow masks are created using lower and upper values of the colors. To create a combination of masks for color bands, "RedBlue", "RedOrange" and "RedYellow" masks are created by simply adding masks together. Additionally, by using commands "cv2.morphologyEx()" noise can be removed from masks by applying morphological transformations [27].

On lines 275, 276 and 277 on figure 88, the program finds contours of the detected regions in the mask.

```
279     # Angle calculation part
280
281     # Find contours in the masks
282     red_contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
283     blue_contours, _ = cv2.findContours(blue_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
284     orange_contours, _ = cv2.findContours(orange_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
285     yellow_contours, _ = cv2.findContours(yellow_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
286
287     RedBlue_contours, _ = cv2.findContours(maskRedBlue, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
288     RedOrange_contours, _ = cv2.findContours(maskRedOrange, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
289     RedYellow_contours, _ = cv2.findContours(maskRedYellow, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
290
291     # Sort the contours by area in descending order
292     red_contours = sorted(red_contours, key=cv2.contourArea, reverse=True)
293     blue_contours = sorted(blue_contours, key=cv2.contourArea, reverse=True)
294     orange_contours = sorted(orange_contours, key=cv2.contourArea, reverse=True)
295     yellow_contours = sorted(yellow_contours, key=cv2.contourArea, reverse=True)
296
297     RedBlueData = MyTool(frameROI, contoursRedBlue, blue_contours, BlueColor, blueStr, Offset=False)
298     RedOrangeData = MyTool(frameROI, contoursRedOrange, orange_contours, OrangeColor, orangeStr, Offset=False)
299     RedYellowData = MyTool(frameROI, contoursRedYellow, yellow_contours, YellowColor, yellowStr, Offset=True)
300
301     # Display the frame with bounding rectangles
302     cv2.imshow('My WEBcam', frame)
303     cv2.moveWindow('My WEBcam', 500, 100)
304
305     # Check for key press
306     if cv2.waitKey(1) & 0xFF == ord('q'):
307         break
308
309 # Release the video capture object and close windows
310 cam.release()
311 cv2.destroyAllWindows()
```

Figure 89: Second part of the while-loop

In the second part of the loop, which is shown on figure 89, python code finds color contours in the masks. Then color contours are sorted by area. On lines 297, 298 and 299 on figure 89, three variables called "RedBlueData", "RedOrangeData" and "RedYellowData" are created and assigned as "MyTool()" functions. Each of those three functions will detect different color bands based on the parameters that were provided. At the end of the second part of the while loop, the program shows a frame as presented on line 302 in figure 89.

7.11 3D models of tool stands

Main task of the project is to make a Kuka LBR IIWA robotic arm to place tools on tool stands. Tool stands for hammer, pliers and screwdriver are 3D modeled in "Fusion 360" software. Models of the tool stands are shown on the figure 90.

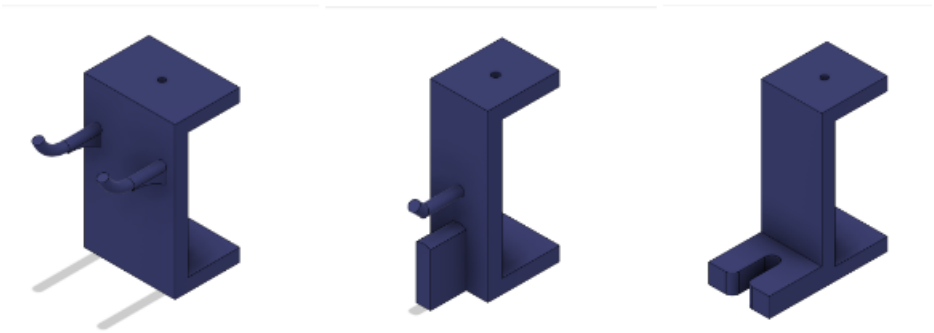


Figure 90: 3D models of the tool stands made in Fusion 360 software

By measuring thickness and length of each tool, the stands can be designed to fit and hold them in place. The tool stands are modeled in such size and shape that they could simply be attached on the tree beams that are located on the wall. After the modeling of tool stands, they can be printed out on a 3D printer and can look as shown on figure 91.

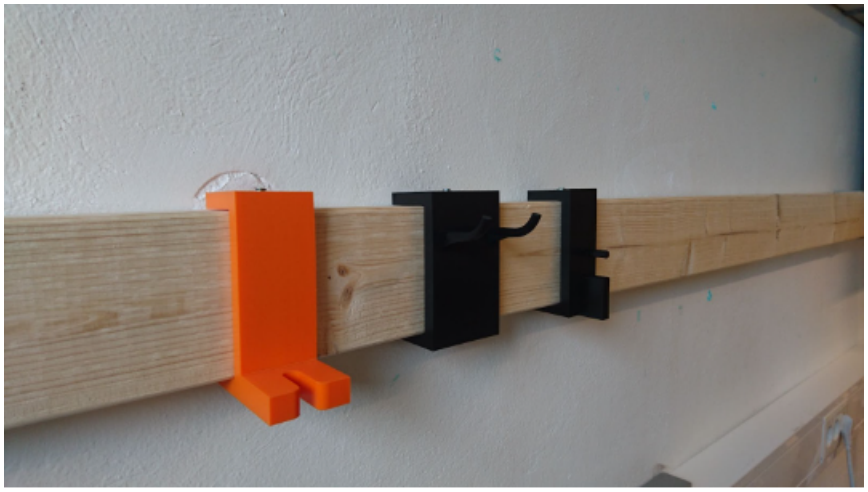


Figure 91: Tool stands made on 3D printer and mounted on the wall

8 Results

This chapter presents the results and achievements of the project based on the implemented methods. It will also present the administrative results as an outcome of the teamwork.

8.1 Engineering result

8.1.1 Integration of devices

One of the key objectives of the project was to integrate several devices together, ranging from industry to consumer grade. The methods utilized several different communication protocols which proved to be more than sufficient for delivering the final solution. It was feared that certain implementations could hold back or bottleneck the system. A good example is the gripper, which might have been lacking necessary or reliable feedback, like the object detection. However, all the sub-systems utilized in the project gave an opportunity for developing a troubleless, consistent and reliable integration solution.

8.1.2 Safety

Safety measurements were the first objective and initial stage of development. In order to conduct any tests or use the robot, it was necessary to develop an extensive risk evaluation and strictly follow it. Initially, the robot was meant to be used only when paired with a relay system, connected to the proximity sensor. This idea was scrambled, as the responsible personnel were not able to carry out the correct configuration over a long period of time.

Based on the risk evaluation and recommendations of experienced personnel and teachers, the environment had to be adapted. The measures were strictly visual warnings and marked lines depicting robots' potential reach. One hardware-based measure and potentially the most important one, was KUKA SmartPAD Stop button, which was at all time within users reach.

Lastly, the software measures, such as very careful programming and fail-proofing of the code, resulted in no hazardous situations at all stages of development. Velocity of the robot was bottlenecked down to maximum 25% on any move, which gave enough reaction time for the users to manually shut down the operation.

In summary, the safety rules were successfully followed and no hazardous situations occurred. However, the planned implementation is not entirely fulfilled.

8.1.3 Commanding the KUKA LBR IIWA

Commanding KUKA LBR IIWA and executing the tasks in a predictable manner was the main objective of the project. The solution required voice operated pick-and-place procedure to be viable and ready to be used. It is safe to say that the main goal has been achieved and in fact, the group's work has produced a fruitful result in the topic. Voice commanding the robot has a very fail-safe implementation. It is not, under any circumstances expected to pass wrong and unwanted commands or trigger unintended behavior. The robot follows strictly scripted paths which makes sure it's collision free. There are no deployed collision detection features as explained previously in the Safety section. The reliability of placing the items on designed stands is very high once the correct calibration is done. As long as the robot is firmly installed in the place, the pick-and-place works flawlessly.

8.1.4 Speech recognition - results

Speech Recognition application is one of the important parts of this project. The primary task of speech recognition was to recognize voice commands precisely and use them later to command a robot to do specific tasks. Speech recognition applications are able to recognize both simple words or sentences said by the user. There are five voice commands used in this project. The voice commands are:

- **Pick up hammer** - sends robot a command to pick up hammer
- **Pick up screwdriver** - sends robot a command to pick up screwdriver
- **Pick up pliers** -sends robot a command to pick up pliers
- **Give** - if robot holds a tool, sends command to a robot that it should release gripper and give tool to a user
- **Drop** - sends command to a robot that it should place tool on a tool stand which is located on the wall.

All of the commands above are recognized by the speech recognition application. However, sometimes it struggles to recognize all of the words for the first try. The user sometimes has to say the voice command multiple times before the application recognizes it correctly. For example, when a user asks to "pick up the hammer", sometimes the application chooses words that are close enough to what the user has said but not exactly correct. Figure 92 and figure 93 demonstrate the correct and incorrect transcript for the hammer.

```
Say something!
result2:
{   'alternative': [   {   'confidence': 0.83532494,
                           'transcript': 'pick up hammer'},
                       {'transcript': 'pick-up hammer'},
                       {'transcript': 'pick-up Hummer'},
                       {'transcript': 'make-up hamper'},
                       {'transcript': 'pick-up Hama'}],
    'final': True}
You said: pick up hammer
```

Figure 92: Correct recognition of hammer command

```
Say something!
result2:
{   'alternative': [   {   'confidence': 0.48389059,
                           'transcript': 'make-up hamper'},
                       {'transcript': 'pick up hammer'},
                       {'transcript': 'pick-up camper'},
                       {'transcript': 'pick-up hammer'},
                       {'transcript': 'speak up hammer'}],
    'final': True}
You said: make-up hamper
```

Figure 93: Incorrect recognition of hammer command

Voice command “pick up screwdriver” is easiest of all of the pick up commands to recognize with transcript confidence of approximately 0.88. The transcript of the correct screwdriver command is shown on the figure 94.

```
Say something!
result2:
{   'alternative': [   {   'confidence': 0.88841063,
                           'transcript': 'pick up screwdriver'},
                       {'transcript': 'pick-up screwdriver'},
                       {'transcript': 'big up screwdriver'},
                       {'transcript': 'pick up a screwdriver'},
                       {'transcript': 'pick-ups screwdriver'}],
    'final': True}
You said: pick up screwdriver
```

Figure 94: Correct recognition of screwdriver command

“Pick up pliers” command is the hardest for speech recognition application to transcript correctly. The application struggles the most to recognize the word “pliers” and often confuses it with other words that sound similar. Figure 95 shows the correct transcript of the “Pick up pliers” command, while figure 96, figure 97 and figure 98 show incorrect transcript.

```
Say something!
result2:
{   'alternative': [   {   'confidence': 0.86337477,
                           'transcript': 'pick up pliers'},
                       {'transcript': 'pick up players'},
                       {'transcript': 'pick-up pliers'},
                       {'transcript': 'pick up tyres'},
                       {'transcript': 'pickup pliers'}],
    'final': True}
You said: pick up pliers
```

Figure 95: Correct recognition of pliers command

```
Say something!
result2:
{   'alternative': [   {   'confidence': 0.87952954,
                           'transcript': 'pick up players'},
                       {'transcript': 'pick up pliers'},
                       {'transcript': 'pick-up pliers'},
                       {'transcript': 'pick-up players'},
                       {'transcript': 'pick up liars'}],
    'final': True}
You said: pick up players
```

Figure 96: Incorrect recognition of pliers command

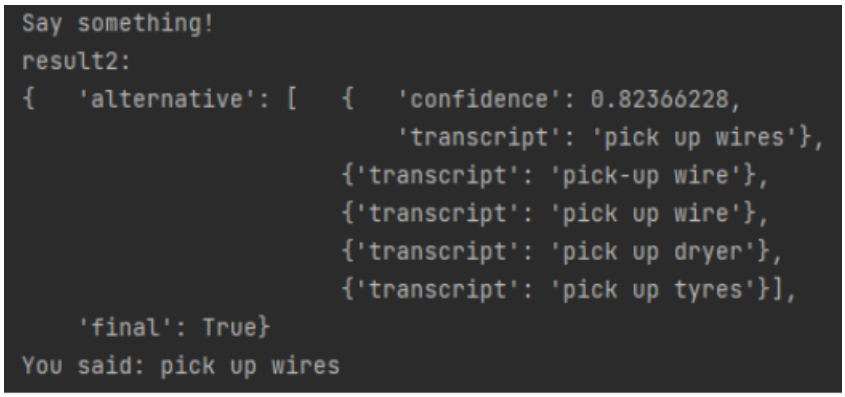


Figure 97: Incorrect recognition of pliers command

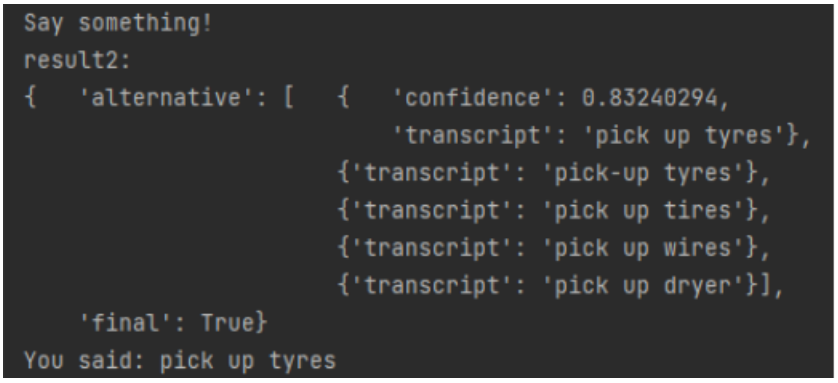


Figure 98: Incorrect recognition of pliers command

In summary, the speech recognition application works as expected and it delivers all the functionality required for the project. However, the application could be more precise in the transcript of the speech.

8.1.5 Computer Vision - results

The Computer vision part of this project that has a task of recognizing and differentiating tools works for the most part as expected. "OpenCV" python library provided the team with all the necessary functionality for the tool detection that was required for the project. By using different color bands for each tool, the computer vision script in python is able to recognize color bands, create coordinates for them and calculate orientation as shown on figure 99.

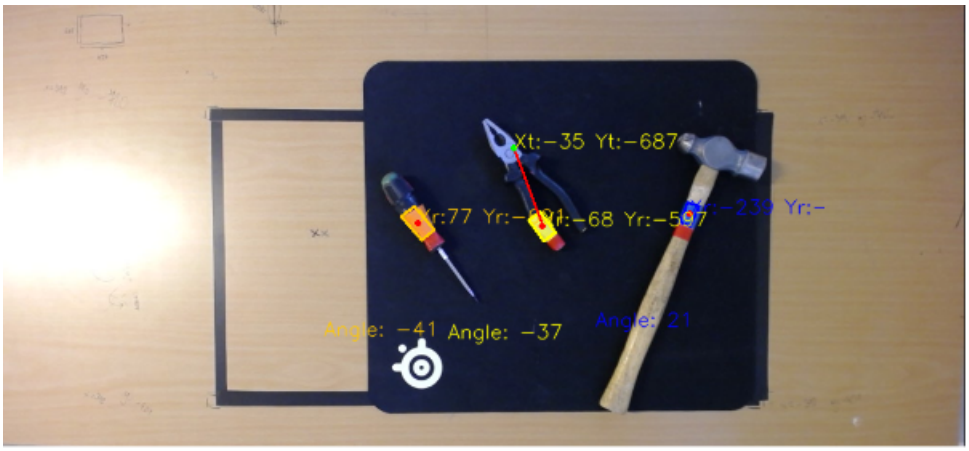


Figure 99: Example of tools recognized by webcamera using computer vision library "OpenCV"

However, sometimes difficulties can occur under the process of tool detection. For example, the program sometimes struggles to recognize color correctly which leads to wrong calculation of the angle or coordinates for some of the tools as shown on figure 100 and figure 101. On figure 100, the coordinates and orientation of the hammer and pliers are calculated correctly, while orientation of the screwdriver is incorrect and should be in the interval of [0, - 90] degrees.

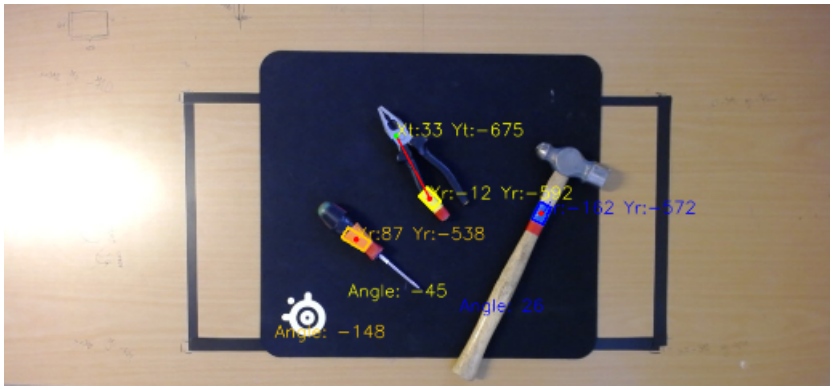


Figure 100: Example of computer vision application having difficulties to correctly calculate angle of screwdriver

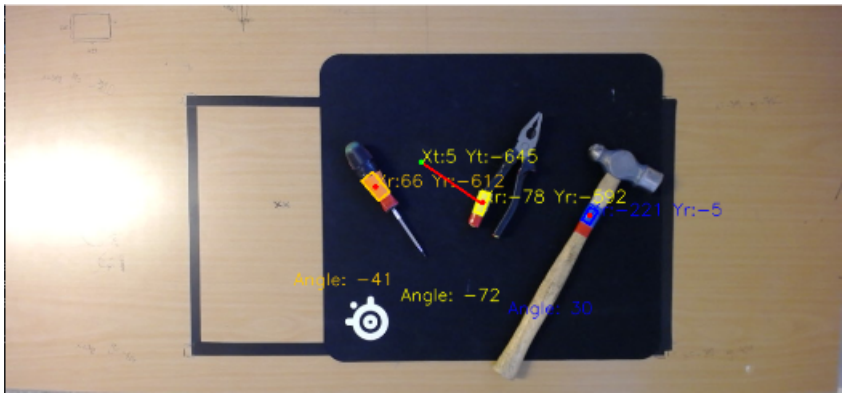


Figure 101: Example of computer vision application having difficulties to correctly calculate offset coordinates and angle for pliers

From figure 101 the program doesn't calculate the offset coordinates for pliers correctly. The offset

coordinates for pliers are shown as a green dot connected with the center of the yellow color band by a straight red line. When the computer vision application is run, it prints coordinates for each color band in the output as shown in figure 102.

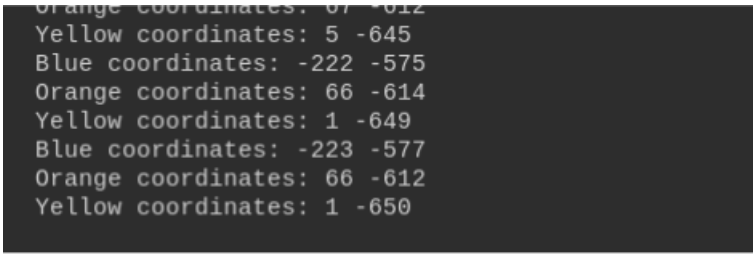


Figure 102: Computer vision application prints out tool coordinates as an output

With all the flaws the computer vision application has in this project, it delivers a simple solution that meets most of the requirements set by the team. The application has room for improvement by applying more advanced object detection techniques that could give better final results.

8.2 Administrative results

8.2.1 Teamwork

Teamwork was a big part in this project. Throughout the project process, team members worked together, reported the progress of their daily tasks and gave technical advice if some of the team members had difficulties with performing the tasks.

8.2.2 Sprints and planning

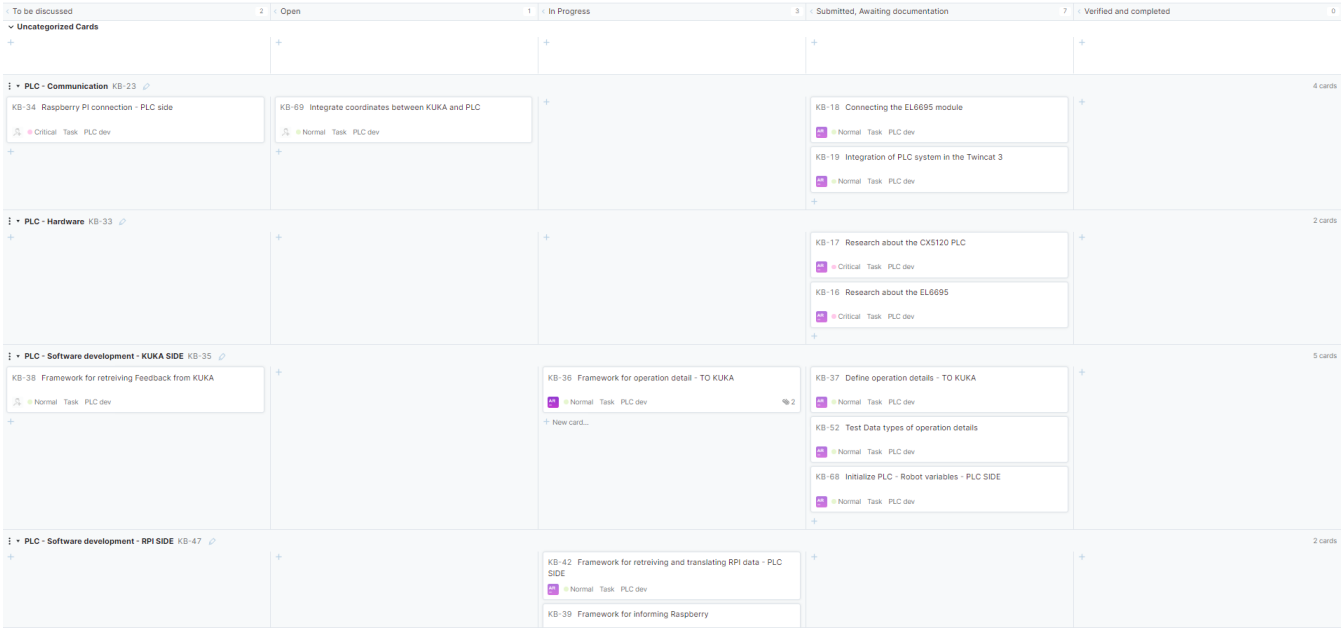


Figure 103: Example of planning tasks in YouTrack

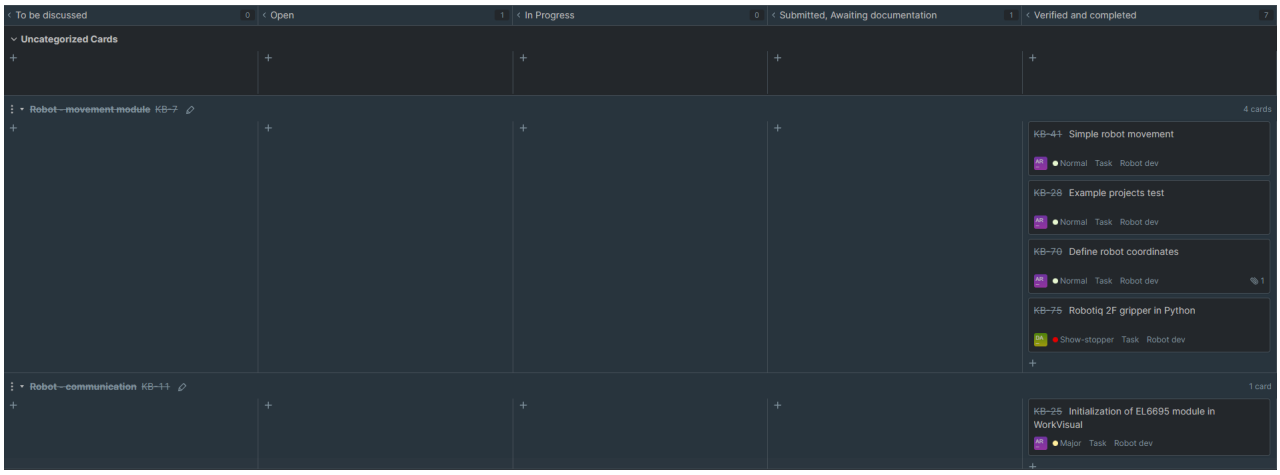


Figure 104: Example of planning tasks in YouTrack

At the beginning of the project the team had challenges with proper planning and assigning of the tasks to each member of the team. The team had difficulties with time estimation and planning deadlines. Later in the process, the team started to use collaborative tools such as “Confluence” and “YouTrack” that enabled team members to split project into smaller parts and keep track of each other’s work. Figure 103 and figure 104 shows an example of planning tasks in ”YouTrack”. With proper planning and task tracking, the team started to work more efficiently towards the project’s goal.

9 Discussion

This chapter is focused on the discussion, whether the group managed to meet personal and scientific expectations. It will also reveal the limitations and challenges that were faced.

9.1 Expectations vs reality

The expectations regarding the technological complexity in the project were definitely higher than what was presented. During the development, it has shown that an overcomplicated system does not yield better results, therefore a simpler vision for the project was created and certain limitations greatly delayed the development process resulting in inferior quality of particular aspects.

There were expectations about safety in the starting phase of the project. Initially the team expected safety to be present on the robot from the start in a proper working condition. However the safety didn't meet the team's expectations and it led to consume a lot of time to figure out how it should be reseted.

Initially, the robot was supposed to be utilized in a collaborative manner, but that idea needed to be put aside because of the limitations. The positive aspect of the expectations versus reality is that the project has reached fully operational state before the estimated deadline. The group has discussed a final roadmap including the final and definitive version of the system and that expectation was fully satisfied.

9.1.1 Computer Vision: Why some parts didn't work?

Computer Vision application overall works good enough and meets requirements set by the team members. However, the application meets some challenges that come from different sources. The color band solution for tool recognition in itself is not the best solution for the task where it needs to recognize and differentiate tools. Color detection of the color bands is based on the recognition of colors that have lower and upper HSV values. Sometimes an application can confuse colors it detects, since they can have HSV values that are in the same interval. It leads to computer vision application recognizing parts of blue color as red or orange color as yellow etc. In addition the light source and shadows affected the performance of the tool detection. The black area where the tools should be detected had places with shadows and light reflections which led to objects being not detected in the same way in different places.

9.1.2 Speech Recognition: Why didn't it work perfectly?

Overall result of the speech recognition application is good for the project. The "SpeechRecognition" python library and the whole application itself delivers all the functionality that is needed to achieve a good result for the project. For the most part the application is reliable and simple in use. Team's requirements is that the speech recognition application should be able to recognize the speech at first try. However the recognition of speech can sometimes not give the desired results. Application can happen to transcript the words incorrectly or not transcript some words at all, leaving empty space. One of the reasons is that a speech recognition application requires precision. Application gives better results if the user has a microphone of high quality and words pronounced loud and clear.

9.2 Limitations

The goals of the project were hindered by certain limitations.

9.2.1 Safety limitations

Undoubtedly the most influential limitation is safety. The group had to strictly follow the guidelines attached in the risk evaluation. One of the main resolutions was maintaining safe proximity from the operating robot. That alone ruled out any human-robot collaboration at close distance. The key element which would allow for certain interactions, was the safe-zone sensor based on the relay system. That was unfortunately not delivered by the responsible personnel and left the group improvising at maintaining a safe working environment.

The environment in which the robot was installed also led to minor limitation issues. KUKA LBR IIWA was placed directly in the corner of the room in order to leave the space for a safe-zone. Due to that fact, the robot had possibilities for only 180 degrees of movement and a portion of it was reserved for safety reasons, it left less space for additional functions.

9.2.2 Limited Hardware

Although the project utilizes a reasonable amount of hardware, the group has applied for additional equipment which was not provided or at least not what was required. The initial resolution of the project was to utilize separate Raspberry PI units to handle any Python related assignments. The objective was to create a master-slave based system, where the Object detection, Speech recognition and Gripper controls would be distributed among the Raspberry units for optimal resource consumption. The final implementation deploys only one unit of Raspberry PI handling just the Object Detection due to the computational limits.

The provided gripper is not purely designed for KUKA LBR IIWA thus induces certain limitations in form of delays in the solution. The robot must await the feedback which travels through several subsystems which results in not instant reactions and hindered precision.

9.3 Challenges

The safety also produced an indirect limitation in the form of enormous time delays.

9.3.1 Safety

Safety setup that was present at the starting phase of the project didn't work properly and didn't allow the team members to progress through the project without limitations. Safety problems that occurred made the project's tasks progress more slowly than the team had expected.

Before the safety implementation took place, the time schedule was already stretched to the maximum. The group did not start any work on the KUKA LBR IIWA until late April as it was not permitted to utilize it without it. Responsible personnel finally decided to remove any safety module and allowed the group to proceed. It is safe to say that the final shape of the robots code and overall performance is lacking due to that fact. The massive delay not only influenced the robot, but generally any aspect of the project since every component required general overhaul to properly integrate it with the robot. The group was granted an additional month to the deadline due to that delay, however the immobility that was induced by it, created a far greater setback.

9.3.2 Computer vision

During development of computer vision application some challenges occurred in the process. One of the ideas of developing a computer vision application was to use pictures of color bands. Instead of tracking colors, camera would use reference pictures of color bands, compare with real time image and detect colorbands which are attached to the tools. However, this solution didn't work as it was expected and the team decided to switch to color tracking instead.

9.3.3 Speech Recognition

When the team developed the speech recognition solution it was planned to use the "Whisper" library provided by OpenAI. There were some problems and errors during setup and use of the Whisper library in PyCharm python IDE. The team later decided to use an alternative library called "SpeechRecognition" to establish a simple speech recognition application.

9.4 Communication

9.4.1 Internal

Along the progression of the project, the team has built great relations in the professional space. The communication in the team was mostly built on common problem solving and achieving the same goal, rather than disagreeing and being frustrated. The group has managed to mitigate potential disagreements regarding meetings, by introducing a more loose working relations, where no strict schedule points were made. Thanks to this working method, the communication took place as soon as both parties were available and all members were satisfied.

Another key factor which resulted in smooth relations in the team, was offering help to each other. Even though the team worked mostly on separate assignments, the objectives were positively influenced by others in the form of providing expertise or ideas for implementation.

9.4.2 How the team worked

The project team worked on several independent parts that needed to be integrated into one system. From the starting period of the project work, team members decided to work on each part separately and independently from each other. Each of the team members had freedom to try out different methods and techniques for the part they worked on.

During the working process, team members didn't have a fixed time at which they should meet up at school. Without fixed meeting time, team members had a flexible working process in which they could meet at any time that was suitable for them. Such choice of working style didn't affect the working progress much and didn't lead to any conflicts between team members.

9.4.3 Project planning

The planning process during the development was dynamic and changing. The group had a prepared plan and goals, but certain changes were made along the way. The original version of the plan included different methods and equipment.

The team decided that an ongoing communication, updating each other on results, predictions and limitations was the key to successful project delivery. Although the team worked mostly on separate milestones, the collaboration occurred daily on the campus where all the information could be exchanged on the site. It helped enormously to achieve common goals and plans knowing that the members are well motivated and engaged in the project.

Besides the physical meetings, occasional web-meetings were held, where the members exchanged their thoughts and ideas outside the professional restrictions.

9.4.4 Collaboration using YouTrack and Confluence

At the beginning of the project the team didn't use any collaboration tools and didn't have any proper planning of the work that should be done. After some time, the team saw that in order to have more progress in the project, collaboration and planning tools are needed. Collaboration tools such as "YouTrack" and "Confluence" were chosen for this project.

"YouTrack" tool enabled team members to split the project into smaller tasks. Every team member could work on every task and at the same time keep track of the work of other team members. "Confluence" made it easy to document the work during the process that later could be used in the report. Team members have never used these collaboration tools before but after some time of use it was clear that it helped the project to progress more quickly in development.

10 Conclusion and further work

10.1 Conclusions

The goal of this project was to develop a Workshop assistant - a robotic arm which is going to follow voice commands to pick-and-place randomly distributed tools. The solution also aimed to show how industrial grade equipment can be complemented by consumer grade devices like Raspberry PI.

In order to meet the requirements, the group picked the necessary hardware and designed working principles from scratch. The work included delving deep into priorly unknown areas like the collaborative robots and putting to work experience gained throughout the studies. By utilizing several programming languages a great ecosystem was created, allowing for complex data exchange and integrity between devices.

By combining different independent software, programming languages and experience from various technical fields the team successfully created a solution to the problem described in the project and achieved an overall good result.

10.2 Further work (upgrade ideas)

This section will explain how the project can be expanded and upgraded. There are many things which would benefit the solution and make it more versatile and robust.

10.2.1 More organized hardware

The requirement for the PC is simply said, not existent. The design already packs a powerful CX5120 IPC which can easily be configured to do what the PC does in the project. The upgrade aspect is based purely on adapting the environment of the IPC to run potential Python code and scripts.

In addition to removing the PC from the ecosystem, it would be advised to deploy additional Raspberry PI units to control speech recognition and gripper code. By doing that the devices would handle one assignment at a time and would potentially be less devastating upon failure of either device yielding the system still operational.

10.2.2 Collaborative robot

The system would greatly benefit from actually utilizing collaborative functions of the robot. As a workshop assistant, the current solution lacks direct touching interaction with humans. In order to combat that, a new risk evaluation would have to be conducted and new reasonable safety measures would have to take place. The robot did not pose a threat during the development process and together with understandable programming methods, it did not end up moving unexpectedly. The future safety measures do not require drastic changes.

10.2.3 Task queueing

Task queueing was one of the points on the roadmap. The idea ended up being dropped due to limited time and not fully thought-through potential implementation. The upside of task queueing is that the robot can perform tasks in series, without constant observation and commanding the robot. However the downside is that the users can either change their minds or trigger unintended behaviors. The group believes that Task queueing would still be a superior solution, however it requires more in depth analysis whether it is actually beneficial in a given environment.

10.2.4 Better tool recognition

Computer vision part has room for improvements and potential to produce better results. Tool recognition by color band detection is a simple and working solution for this project. However the solution is not completely reliable since it doesn't produce accurate results all the time and has calculation flaws. In computer vision application the color of the color bands are affected by light and shadows from the surroundings which makes it difficult for camera to produce the same recognition result for all places on the surface.

The better solution would be to use shape recognition of the object. Python program could get training from multiple pictures of the tools and based on those pictures learn the shape of the object it needs to detect. Such solution would be more advanced and challenging to execute but would give better results at the end.

10.2.5 Better speech recognition

Speech Recognition application does satisfy the requirements of this project by ability to recognize user's commands for Kuka robot. However the solution is not perfect. Speech recognition library used in this project does sometimes have difficulties understanding some words or commands on the first attempt. This is not a critical problem that affects the project much but the application could be improved. Other speech recognition libraries such as "Whisper" from "OpenAI" could be more reliable and give better and more precise speech recognition results.

Bibliography

[1] 2018sauravtelge. *OpenCV – Invert Mask*. URL: https://www.geeksforgeeks.org/opencv-invert-mask/.

[2] Youngeun An, Muhammad Riaz and Jongan Park. 'CBIR Based on Adaptive Segmentation of HSV Color Space'. In: *2010 12th International Conference on Computer Modelling and Simulation*. 2010, pp. 248–251. DOI: 10.1109/UKSIM.2010.53.

[3] Real Time Automation. *An introduction to Modbus RTU*. URL: https://www.rtautomation.com/technologies/modbus-rtu/.

[4] S. Kolkur; D. Kalbande; P. Shimpi; C. Bapat and J. Jatakia. *Human Skin Detection Using RGB, HSV and YCbCr Color Models*. URL: https://arxiv.org/ftp/arxiv/papers/1708/1708.02694.pdf.

[5] Beckhoff. *CX5120 — Embedded PC with Intel Atom® processor*. URL: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/cx5100-intel-atom/cx5120.html.

[6] Beckhoff. *EL6695 — EtherCAT Terminal, communication interface, EtherCAT bridge, extended functions*. URL: https://www.beckhoff.com/en-us/products/i-o/ethercat-terminals/el6xxx-communication/el6695.html?pk_campaign=AdWords-AdWordsSearch-EtherCat_dynamic_EN&pk_kwd=.

[7] Beckhoff. *Router*. URL: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_userinterface/3813966475.html&id=.

[8] Beckhoff. *TwinCAT 3 — eXtended Automation (XA)*. URL: https://www.interempresas.net/FeriaVirtual/Catalogos_y_documentos/171630/Beckhoff_TwinCAT3_042012_e.pdf.

[9] Jason Brownlee. *Python Multiprocessing: The Complete Guide*. URL: https://superfastpython.com/multiprocessing-in-python/.

[10] Marina Chatterjee. *What is Computer Vision? Know Computer Vision Basic to Advanced & How Does it Work?* URL: https://www.mygreatlearning.com/blog/what-is-computer-vision-the-basics/.

[11] Codesys. *Structured Text (ST), Extended Structured Text (ExST)*. URL: https://help.codesys.com/webapp/_cds_f_programming_language_st;product=codesys;version=3.5.13.0.

[12] Gerry Creech. *Black Channel Communication: What is it and how does it work?* URL: https://journals.sagepub.com/doi/pdf/10.1177/002029400704001003.

[13] Raspberry Pi Foundation. *Raspberry Pi OS*. URL: https://www.raspberrypi.com/software/.

[14] George T Hilliard. *What is an Industrial Embedded Computer?* URL: https://www.winsystems.com/what-is-an-industrial-embedded-computer/.

[15] D. Hema; Dr. S. Kannan. *Interactive Color Image Segmentation using HSV Color Space*. URL: http://site.mzu.edu.in/wp-content/uploads/2020/05/Interactive-Color-Image-Segmentation-using-HSV-Color-Space.pdf.

[16] Shahid Akhtar Khan. *How to mask an image in OpenCV Python?* URL: https://www.tutorialspoint.com/how-to-mask-an-image-in-opencv-python.

[17] KUKA. *KUKA Sunrise Cabinet*. URL: https://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/MA_KUKA_Sunrise_Cabinet_en.pdf.

[18] KUKA. *KUKA Sunrise.OS*. URL: https://www.kuka.com/en-de/products/robot-systems/software/system-software/sunriseos.

[19] KUKA. *KUKA.WorkVisual*. URL: https://www.kuka.com/en-us/products/robotics-systems/software/system-software/kuka_systemsoftware/kuka-work-visual.

[20] Kuka. *KUKA Sunrise Cabinet*. URL: https://www.kuka.com/en-de/products/robot-systems/robot-controllers/kuka-sunrise-cabinet.

[21] Kuka. *LBR iiwa*. URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa.

[22] Kuka. *The KUKA smartPAD: simply more freedom*. URL: https://www.kuka.com/en-de/products/robot-systems/robot-controllers/smartpad.

[23] Ben Lutkevich. *Speech recognition*. URL: https://www.techtarget.com/searchcustomerexperience/definition/speech-recognition.

[24] Hani Al-Mohair. 'Impact of Color Space on Human Skin Color Detection Using an Intelligent System'. In: Jan. 2013, p. 179.

[25] Department of Marine Technology NTNU. *IMT Software Wiki - LaTeX*. URL: https://www.ntnu.no/wiki/display/imtsoftware/LaTeX (visited on 15th Sept. 2020).

[26] OpenCV. *Contours : Getting Started*. URL: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

[27] OpenCV. *Image Filtering*. URL: https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html.

[28] Pankaj. *Multithreading in Java - Everything You MUST Know*. URL: https://www.digitalocean.com/community/tutorials/multithreading-in-java.

[29] Hubert Pham. *PyAudio 0.2.13*. URL: https://pypi.org/project/PyAudio/.

[30] Gabriele Ribichini. *What Is EtherCAT Protocol and How Does It Work?* URL: https://dewesoft.com/blog/what-is-ethercat-protocol.

[31] Robotiq. *2F-85 and 2F-140 Grippers*. URL: https://robotiq.com/products/2f85-140-adaptive-robot-gripper?ref=nav_product_new_button.

[32] Robotiq. *Robotiq 2F-85 & 2F-140 Instruction Manual*. URL: https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf.

[33] Robotiq. *Robotiq 2F-85 2F-140*. URL: https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_General_PDF_20210623.pdf?_ga=2.201990357.51230788.1653473411-1379069650.1649070508.

[34] Simplilearn. *What Is Computer Vision: Applications, Benefits and How to Learn It*. URL: https://www.simplilearn.com/computer-vision-article.

[35] Technology Robotix Society. *Colour Detection*. URL: https://medium.com/image-processing-in-robotics/colour-detection-e15bc03b3f61.

[36] Editorial Staff. *Components of PLC*. URL: https://instrumentationtools.com/components-of-plc/.

[37] TheAILearner. *OpenCV Minimum Area Rectangle*. URL: https://theailearner.com/tag/cv2-minarearect/.

[38] Wikipedia. *Function block diagram*. URL: https://en.wikipedia.org/wiki/Function_block_diagram.

[39] Ladder Logic World. *Ladder Logic Basics*. URL: https://ladderlogicworld.com/ladder-logic-basics/.

[40] Guobo Xie and Wen Lu. *Image Edge Detection Based On Opencv*. URL: http://www.ijeee.net/uploadfile/2013/0702/20130702104409134.pdf.

[41] Anthony Zhang. *SpeechRecognition 3.10.0*. URL: https://pypi.org/project/SpeechRecognition/.

Appendix

A Project video link

https://www.youtube.com/watch?v=sVem_W-k8hU&ab_channel=Artify

B Preliminary project report

FORPROSJEKT - RAPPORT
FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
KUKA – Robot Assistent

KANDIDATNUMMER(E):
Daniil Gladkikh – 10005 Artur Andrzej Gwozdowicz -

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
30.01.2023	**IE303612**	**Bacheloroppgave**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
AUTOMATISERING OG ROBOTIKK	12/12	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Ottar Laurits Osen, Adam Leon Kleppe

OPPGAVE/SAMMENDRAG:

I denne rapporten diskuteres det hva er problemstillingen til en prosjektoppgave som er valgt av studenter og hva er oppgavens mål på slutten av prosjektarbeidet. Det blir diskutert ulike metoder og utsyr som vil hjelpe å oppnå suksessfullt resultat til prosjektoppgaven.

I rapporten er det planlagt roller for alle studenter som er medlemmer av prosjektgruppen. Hver av gruppemedlemmer får sine egne deloppgaver. Det blir planlagt prosjekt bøter både med veiledere og prosjektmedlemmer for å ha oversikt over statusen på arbeidet. På slutten av rapporten vil det oppgitt prosjekt planer i form av Gantt-diagram og tekniske dokumenter som er relevante for prosjektoppgaven.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Kort innledning om bakgrunn – om valg av oppgave, oppdragsgiver, den grunnleggende problemstillingen og formålet med oppgaven.

I dette prosjektet ble det å bruke KUKA LBR iiwa som er tilgjengelig på NTNU labbygget til å lage en robot assistent som kan se rundt seg med kamera og høre etter stemme kommandoer. "Robot assistent" skal hjelpe med å holde og/eller sortere verktøy på et verkstad eller en lab. Det er en egendefinert oppgåve som vi har valgt fordi det var en interessant ide om å kombinere "Computer Vision" og "Voice Commands" til å styre roboten og få den til å utføre forskjellige oppgaver.

2 BEGREPER

Definisjoner som blir ofte brukt i prosjektet:

PLC – Programmable Logic Controller

CV – Computer Vision

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
Daniil Gladkikh – 539214, Sekretær Artur Andrzej Gwozdowicz – 509373, Prosjektleder

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget ID 302906

3.1.1 Oppgaver for prosjektleder - organisering

Gruppen består av to medlemmer som skal utføre prosjektet sammen. I denne tilfelle er begge studenter like ansvarlig for prosjektorganisasjon.

Ansvarsområde til prosjektleder:

- Bestemme tidspunkt til møte innkalling
- Dele og opprette oppgaver
- Ta avgjørelser for gruppen basert på felles diskusjon

- Sørge for at gruppen kommer i mål
- Løse problemer som oppstår under prosjektarbeid

Arbeidsoppgaver til prosjektleder:

- Føre oppmøte
- Ansvar for at prosjektarbeid er på vei mot et godt resultat
- Ansvar for at all dokumentasjon er på plass

3.1.2 Oppgaver for prosjektsekretær

Arbeidsområde for prosjektsekretær:

- Dokumentere arbeidsprosessen og føre logg
- Skrive møtereferat
- Administrativt følge opp avviksbehandling

Arbeidsoppgaver til prosjektsekretær:

- Hjelpe prosjekt leder med administrativt arbeid
- Lage avtaler og kontrakter for prosjektet
- Organisering av prosjektmøter

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Veiledere for dette prosjektet: Ottar Laurits Osen (hovedveileder) og Adam Leon Kleppe (veileder).

4 AVTALER

4.1 Avtale med oppdragsgiver

Ingen avtaler har blitt laget med oppdragsgiver. Det eneste som ble enighet om er kravet til løsningen som er vist nedenfor i del kapittel 5.2.

4.2 Arbeidssted og ressurser

Ressurser og utstyr er på Campus Ålesund og fagskolen Ålesund.

4.3 Gruppenormer – samarbeidsregler – holdninger

- Gruppemedlemmer er pliktige å møte til avtalt tid og eventuelt informere andre om endringer.
- Alle er ansvarlige for å utføre sine oppgaver, eller informere om problemer slik at oppgaven kan bli utført av andre.
- Medlemmer skal støtte hverandre og jobbe for felles mål ved å respektere andre og opptre profesjonelt.
- Under ukentlige møter skal alle gruppemedlemmer vise frem arbeidet de har gjort. Dersom oppgaver som skulle løses ikke ble løst før ukentlig status møte, bør gruppa sjekke problemet i lag.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Formuleringer av den grunnleggende problemstillingen og hva en skal komme fram til i løpet av prosjektet – hovedmål og evt. delmål. Gjerne med en inndeling eller beskrivelse som skjelner mellom effektmål (verdimål), resultatmål og prosessmål.

Problemstilling for dette prosjektet er å bruke "KUKA LBR iiwa" kollaborative robot til å lage en "robot assistent" som skal hjelpe med på verkstedet/labben. Roboten skal bruke web kamera til å se rundt seg og spore forskjellige objekter. Oppgavene til en robot kan blant annet være å holde objekter dersom en bruker er bedt om det, sette ønsket objekt på plass og liknende. Planen for dette prosjektet er å bruke både "Computer Vision" og "Voice Command" biblioteker til å styre roboten. Fremgangsmåten til å lage denne løsningen er:

1. Gjøre utforskning på eksisterende teknologier som kan være tilgjengelig og hjelpsomme i prosjektet.
2. Lage en liste av alle komponenter som blir brukt i prosjektet: mekanismer, sensorer, elektroniske/mekaniske komponenter, software osv. Vurdere om noen av komponenter kan lages på 3D printer for å ikke bruke mye tid på å vente på bestilling.
3. Lage en skisse av prosjektet som demonstrerer utseende og virkemåten (Valgfritt).
4. Studere om hvilken type bus-kommunikasjon som skal brukes for denne type roboten.
5. Jobbe med å sette opp bus kommunikasjonen slik at det er dannet grunnlag for fremtidige oppgaver.
6. Når bus-kommunikasjon er på plass, finne ut mer om Python biblioteker som skal brukes til å gi kommandoer til KUKA-roboten.
7. Sette kommunikasjon mellom PLC til roboten og Python for å kunne bruke både kamera og mikrofon til å gi roboten kommandoer. Gjøre research og testing på ulike metoder (Artificial Vision, Digital Twin etc.)
8. Teste ut hele systemet og regulere, fjerne feil som oppstår underveis og forbedre resultatet.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Kravet til løsning i denne oppgave er at robot KUKA LBR iiwa skal hente og sette verktøy på plass. Roboten skal bruke web-kamera til å kunne se hva som skjer rundt og registrere objekter/verktøy med hjelp av CV algoritme som blir kjørt i Python. Dersom robot hører bestemte stemme kommandoer, skal den hente eller sette på plass verktøy som f.eks. skrujern. Det skal designes et stativ som inneholder alle verktøy og det stativet skal roboten bruke til å hente fra og plassere verktøy.
For suksessfull slutt resultat bør roboten klare å hente ønsket verktøy når den hører stemme kommando og plassere den tilbake med et annet stemme kommando. I tillegg bør roboten registrere objekter/verktøy uten å miste dem ut av synet konstant. Det kan også være viktig at roboten klarer å utføre oppgave og ikke bli forstyrret dersom noe kommer i veien (menneske, veggen osv).

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Hele arbeidsprosessen blir registrert i Gantt diagram. Gantt diagram inneholder oppgaver, deloppgaver og hvilken tidsperiode er forventet til å løse de oppgavene. Gantt diagram gir en god oversikt av hva som må gjøres for å gjennomføre prosjektet.

5.4 Informasjonsinnsamling – utført og planlagt

Informasjonen skal samles fra forskjellige kilder. For å kunne etablere bus-kommunikasjon til KUKA roboten, skal informasjon samles både offisielle nettsider og dokumentasjoner/manualer som vi får fra NTNU. For å kunne bruke CV og stemme kontroll, bør det samles informasjon om ulike Python biblioteker som er tilgjengelig på nett og vurdere ulike typer av kameraer og mikrofoner som er mulig å bruke for å bygge en fungerende prototype.

5.5 Vurdering – analyse av risiko

Nedenfor er vist tabeller med situasjoner som kan oppstå under prosjekt arbeid og hindre arbeidsprosessen. De viser hvor stor sannsynlighet for at situasjonene skjer og hva blir konsekvenser.

Sannsynlighet	Frekvens
1. Lite sannsynlig	Skjer sjeldnere enn hvert 10. år
2. Mindre sannsynlig	Mellom en gang hvert 5. år og en gang hvert 10. år
3. Sannsynlig	Mellom en gang hvert år og en gang hvert 5. år
4. Meget sannsynlig	Mellom en gang i måneden og en gang i året
5. Svært sannsynlig	En gang i måneden eller oftere

Konsekvens	Helse	Materiell skade

1. Ufarlig	Ubehag og mindre skader som ikke krever tilsyn	< 100 kr
2. Mindre alvorlig	Uvarige skader. Kan kreve tilsyn	< 500 kr
3. Alvorlig	Alvorlig personskade med sykefravær	< 1000 kr
4. Svært alvorlig	Permanente skader og hemminger	< 10 000 kr
5. Kritisk	Død	> 10 000 kr

Risikomatrise					
Sannsynlighet	Konsekvens				
	1. Ufarlig	2. Mindre alvorlig	3. Alvorlig	4. Svært alvorlig	5. Kritisk
5. Svært sannsynlig					
4. Meget sannsynlig					
3. Sannsynlig					
2. Mindre sannsynlig					
1. Lite sannsynlig					

Risiko nivåer	
	Lav risiko, akseptert
	Middels risiko, hindrer ikke arbeids prosessen, men tiltak må vurderes
	Høy risiko, påvirker prosessen i stor grad, tiltak må gjennomføres

Situasjoner under prosjektet	Sannsynlighet	Konsekvenser	Risiko	Løsning
Kortslutning	5.	2.		Avkoble spennigskilde før arbeid
Kutt	3.	1.		-
Feilbruk av komponenter/utstyr	3.	2		Lese dokumentasjon/ Finne informasjon på nettet
Skade av en komponent/utstyr	2.	3.		Lese dokumentasjon/ Finne informasjon på nettet

Risikomatrise-mal hentet fra: https://www.uib.no/hms-portalen/137268/hms-risikovurdering-og-sikker-jobbanalyse-sja

5.6 Hovedaktiviteter i videre arbeid

Hovedaktiviteter i dette prosjektet er å etablere bus-kommunikasjon for KUKA-roboten, Python programmering og eventuelt noe 3D design/simulering dersom det blir behov for det. Arbeidsoppgaver blir fordelt etter kunnskap og erfaring til gruppemedlemmer. Dersom en av gruppemedlemmene står fast på en oppgave, skal andre gruppemedlemmer prøve å hjelpe med løsning.

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Bus-kommunikasjon er den delen av prosjektet som kommer til å ta mest tid. Det forventes at bus-kommunikasjon mellom Roboten og PLC blir satt opp før **3. mars**. Dersom det er noen ekstra deler som skal bestilles skal de bestilte deler komme frem før 1**5. Mars**. Montering og oppkobling av hele robot systemet sammen med bestilte deler forventes å bli ferdig før **5. april**. All kode kan videreutvikles og forberedes underveis, men kodeprototype bør være klar til **5. april**. Fra det tidspunktet robot-oppsettet er klar, gjøres det en testing av hele robot-prototype. Parallelt med testing av roboten, skal bachelor oppgaven skrives frem til **20. Mai**. Det kan foregå endringer i datoer.

5.7.2 Styringshjelpemidler

Møter med veileder(e) skal foregå minst en gang i to uker. Møter med veileder(e) skal gi oss nok informasjon, kunnskap og erfaring og hjelpe oss å oppnå ønsket resultat i prosjektet. I tillegg om det er mulig, kan vi kontakte tidligere faglærere å spørre om hjelp med noe konkret dersom prosjekt gruppe står fast.

5.7.3 Utviklingshjelpemidler

Hardware:

- Kuka LBR iiwa (finn nøyaktig model)
- Kuka control computer (finn nøyaktig model)
- Beckhoff PLC (finn nøyaktig model)
- Wago PLC (finn nøyaktig model)
- Raspberry PI/Arduino - **kanskje**
- Diverse tilbehør til mikrokontrollere

Software:

- E!Cockpit
- PyCharm
- ROS - **kanskje**
- RoboDK - **kanskje**
- Kuka Sunrise.OS
- Software for beckhoff (Må sjekkes), sikkert Codesys 3.5, TwinCAT
- Fusion 360

5.7.4 Intern kontroll – evaluering

Under prosjekt arbeid, skal gruppemedlemmer møttes en gang i uka utenom de dagene som er opptatt med andre fag/annet. Under møtene skal gruppemedlemmer diskutere fremdriften med deloppgaver/prosjektet og dersom det oppsto noen utfordringer, prøve å hjelpe hverandre å løse det sammen som gruppe. Dersom en deloppgave er løst, skal gruppemedlemmer vurdere om det skal gjøres noen forbedringer eller justeringer. Dersom oppgaveløsning tilfredsstiller alle krav, går gruppe videre til neste deloppgave.

5.8 Beslutninger – beslutningsprosess

Alle beslutninger som gjelder prosjektet, skal gruppemedlemmene utføre sammen. Dersom det kommer uenighet i gruppen, bør det tas kontakt med veileder for å ta opp diskusjon sammen med han/hun. Dersom det kommer noen ideer om radikale endringer i prosjektet som gjør store forandringer i prosjektet, bør gruppemedlemmene vurdere om slike endringer er nødvendig og eventuelt snakke med veilederen om det.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Under prosjektarbeidet skal det utarbeides dokumentasjon på hele prosjektet slik at hele prosessen kan repeteres av andre studenter eller bedrifter. Dokumenter og diverse filer kan distribueres via Discord server, GitHub, Wiki eller liknende. Typer dokumentasjon som skal utarbeides er følgende:

- Hardware - innebærer alle koblingsskjema av utstyr og komponenter slik som PLC, mikrokontrollere osv.
- Software – dokumentere arbeidet innenfor programmering. Det betyr å inkludere koder fra forskjellige programvarer sammen med kommentarer eller forklaringer rundt virkemåte.
- 3D/CAD - filer – dersom det blir gjort noe modellerings arbeid, skal alle 3D – modeller lagres og dokumentering rundt modellerings arbeid kan være forklaring på hva modellen er og hvor den hører til på fysisk system.
- Arbeidsprosess – logg av arbeidet og ukentlige framdriftsrapporter. Det kan også inkludere endringer i arbeidsprosessen og metoder.

7 PLANLAGTE MØTER OG RAPPORTER

Gantt diagram ligger nederst på rapporten som vedlegg og viser aktiviteter som skal foregå under prosjekt arbeidet.

7.1 Møter

Møter med veiledere skal foregå annen hver uke på torsdager kl 11:00 – 12:00. Dersom det oppstår noen utfordringer eller situasjoner som hindrer møter i dette tidspunktet, kan annet tidspunkt avtales.

7.2 Prosjektmøter

Møter med galle gruppe medlemmer skjer på fredag kl. 12. I gruppemøtene kan gruppemedlemmer diskuterere hva gruppa kom frem til, løsninger, og problemer som oppsto under arbeidet. Under møtene skal gruppa også planlegge oppgaver til kommende uke og skrive framdriftsrapport. Dersom det blir problematisk å møte opp fysisk, kan gruppemedlemmer møtes digitalt på Teams/Zoom eller Discord.

7.3 Periodiske rapporter

Periodiske rapporter skal skrives annenhver uke og skal inneholde en diskusjon om hva som gikk bra, hva som var problematisk å få til og hva er videre steg til å oppnå ønsket resultat.

8 PLANLAGT AVVIKSBEHANDLING

Dersom det oppstår noen avvik under prosjektarbeidet, skal gruppemedlemmer analysere situasjon sammen å vurdere hvordan avvik i arbeidet kan fjernes eller reduseres.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

For å gjennomføre denne bacheloroppgaven har vi g\behov for ulike typer utstyr. NTNU og Fagskole har både PLC-er og Kuka roboten fra før, så det er ingen behov i begynnelsen til å bestille kontrollere for roboten. Vi har tilgang til web-kamera fra før som vi skal bruke sammen med Python til å lage CV til Kuka roboten. Stativet til verktøy kan 3D –designes i programmet Fusion 360. Det som vi ikke har tilgang til nå og som skal bestilles senere er en mikrofon som skal brukes til stemme kommandoer.
Ellers så har vi tilgang til all utsyr for å kunne sette i gang, men det kan skje at det blir behov for noen ekstra deler og utstyr.

VEDLEGG

Alle dokumenter som Gantt diagram, risikovurderings matrise til bruk av Kuka-roboten og dokumentasjon ligger i vedlagt ZIP-filen.

C Progress report

Bachelor thesis	Project Kuka iiwa – lab and workshop robot assistant	Number of meeting this period: 5. 3 planned	Firma - Oppdragsgiver NTNU	Side 1 av 3
Progress report	Period/week(s) 17	Number of hours this period. Approx. 300	Prosjektgruppe (navn) Bachelorgruppe 11	Dato 25.04.23

Main goal/purpose for this periods work

- Get familiar with Kuka Sunrise Workbench and WorkVisual
- Use TwinCAT 3 software for communication between devices.
- Create an application that recognizes speech and sends recognized commands to PLC
- Program simple robot movements
- Get a gripper that can be used for Kuka robot

Planned activities this period

- Make an application with python that will use speech recognition library on Raspberry Pi to recognize voice of a user
- Extend Speech Recognition application so that it will try to look after particular words
- Make a Server/Client communication between PC and Raspberry Pi
- Make a communication between PC and PLC using TwinCAT 3
- Create a set of movements for the KUKA robot to be utilized as a framework for future goal
- Adapt the code of the Robot to handle variables sent and received by the PLC accordingly.
- Design and calibrate the workspace of the robot.
- Make Robotiq 2F gripper work with Python and communicate to PLC

Actually conducted activites this period

- Implemented a SpeechRecognition library and made an application that recognizes speech and uses it to control variables of the PLC.
- We created a simple Server/Client communication in order to connect Raspberry Pi, PC and Beckhoff PLC together. SpeechRecognition is run on Raspberry Pi which acts as a server, while PC as a client recieves recognized text and uses it to control variables in PLC through TwinCat 3.
- There were several ways of doing speech recognition with Python. We could've also used "Whisper" package from OpenAI which could be a better solution. When this package was tested, we came across multiple errors and decided to stick with the already established solution.
- 3D printed tool stands for each of the tools we want Kuka robot to pick up. The stands will be mounted on the wall next to the robot. Additionally, we made a dropbox that Kuka robot will use to drop tools it picked. The dropbox is made of acrylic material and we used a laser cutter to cut the material.
- Implemented communication between Beckhoff CX5120 and Kuka LBR IIWA robot via the beckhoff EL6695 module
- Designed the data exchange framework between devices.
- Determined robot workspace.
- Created a movement sequence for the robot based on the orders given by the PLC.
- Adapted variables exchanged between the devices to suit the end goal of the project.
- Created a Python/PLC control and communication of the Robotiq 2F gripper. Both Python and TwinCat can see the status of the gripper and if it has grasped an object.

Description of/ justification for potential deviation between planned and real activities

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

Bachelor thesis	Project Kuka iiwa – lab and workshop robot assistant	Number of meeting this period: 5. 3 planned	Firma - Oppdragsgiver NTNU	Side 2 av 3
Progress report	Period/week(s) 17	Number of hours this period. Approx. 300	Prosjektgruppe (navn) Bachelorgruppe 11	Dato 25.04.23

- At the start of the project, we had a plan to use only one Raspberry Pi for both speech recognition application and computer vision application. The plan was that Raspberry would communicate with PLC while PLC would send commands to Kuka robot. Now we see that one Raspberry Pi possibly will not handle SR and CV at the same time and they need to be run separately on two different Raspberry Pis. In our updated setup we make communication between Raspberries and PC using Server/Client technique. The Serve/Client communication was successfully tested and will be used later in the project.
- The biggest portion of the project was stuck because of not implemented safety by the responsible personnel. No tests, robot programming, variable exchange or using the robot manually via SmartPAD was conducted until late March. Since the beginning of the project, the safety proximity sensor was reconfigured multiple times and was brought to operational state in March. After consultations and multiple inspections, it showed that the safety was implemented incorrectly and required an internal overhaul by the safety personnel. This produced a massive setback in the project development, since any activity related to Kuka LBR IIWA robot and various communication parts were put on hold.
- The risk evaluation of the project had to include adaptation of the surroundings of the robot to its operational mode, which is supposed to be implemented by the personnel responsible for safety. To use the robot in software-controlled mode, the waterpipes which are found on the wall behind the robot require protection. This part of the safety has still not been implemented, and we have not received any assurance or date when it will be done.
- The dropbox that we designed for the tools will probably not be used since we created a workspace for a robot where it can drop tools.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- We will most likely remove safety from the whole system. The reason is that Omron sensor that is used on robot reacting on the changes on the environment and resetting itself every time it senses differences around the setup. It leads to Kuka robot constantly stopping in the middle of the action and resetting the robot every time it happens.
- We decided to use local PC for both speech recognition and gripper control while Raspberry Pi will be responsible for computer vision. PC and Raspberry will communicate with each other as server and client.

Main experience from this period

- We had troubles with communication between Beckhoff PLC, components and PC for quite a long time. When using TwinCat 3 software we couldn't add new devices like the bridge device EL6695 which was necessary for the project.
- We didn't have a gripper that we could test on our Kuka robot. Tried to borrow Robotiq gripper from Manulab but all grippers were already in use by other bachelor groups.
- We had problems with safety for the Kuka robot. Robot safety was not setup properly which led to spending a lot of time trying to fix the issues instead of focusing on project goals.

Main purpose/focus next period

- Make an application of Robot assistant that picks up tools with user's voice as an input

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

Bachelor thesis	Project Kuka iiwa – lab and workshop robot assistant	Number of meeting this period: 5. 3 planned	Firma - Oppdragsgiver NTNU	Side 3 av 3
Progress report	Period/week(s) 17	Number of hours this period. Approx. 300	Prosjektgruppe (navn) Bachelorgruppe 11	Dato 25.04.23

- Write the final report

Planned activities next period

- Make sure all communication parts work as the should/debugging
- Investigate and implement Computer Vision part
- Fine tuning of robot movements; implementing more move sets.
- Further Programming of the robot in Sunrise Workbench
- Strictly integrating devices together.
- Writing the final report

Other

Wish/need for counceling

Approval/signature group leader	**Signature other group participants**
Artur Andrzej Gwozdziz	Daniil Gladkikh

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

D Time list

Date	Hours	Contributors	Comments
15.01.2023 - 1.02.2023	60	Artur Gwozdowicz	Initial Project planning
	60	Daniil Kalland	Initial Project planning
1.02.2023 - 15.02.2023	55	Artur Gwozdowicz	Pre-project report
	55	Daniil Kalland	Pre-project report
15.02.2023 - 20.02.2023	30	Artur Gwozdowicz	Risk evaluation
	30	Daniil Kalland	Risk evaluation
20.02.2023 - 05.03.2023	8	Artur Gwozdowicz	Research required hardware
	8	Daniil Kalland	Research required hardware
	5	Artur Gwozdowicz	Obtain required hardware
	5	Daniil Kalland	Obtain required hardware
	3	Artur Gwozdowicz	Establish Collaborative tools
	3	Daniil Kalland	Establish Collaborative tools
	16	Artur Gwozdowicz	Establish Twincat 3 to PC connection
	8	Artur Gwozdowicz	Test Twincat 3 functionality
	30	Daniil Kalland	Setup Raspberry PI 4
	20	Daniil Kalland	Test Raspberry PI 4 functionallity
	16	Artur Gwozdowicz	Establish EL6695 connection
	8	Artur Gwozdowicz	Test EL6695 communication
	16	Artur Gwozdowicz	Solve safety-related issues
	16	Daniil Kalland	Solve safety-related issues
05.03.2023 - 30.03.2023	35	Artur Gwozdowicz	Solve safety-related issues
	35	Daniil Kalland	Solve safety-related issues
	30	Artur Gwozdowicz	Testing Environment of KUKA LBR IIWA
	30	Artur Gwozdowicz	Designing Data Exchange within the project
	25	Artur Gwozdowicz	Twincat 3 Programming
	40	Daniil Kalland	Designing and implementing Speech Recognition
	40	Daniil Kalland	Implementing Gripper
30.03.2023 - 10.04.2023	30	Artur Gwozdowicz	Documentation
	33	Daniil Kalland	Documentation
10.04.2023 - 25.04.2023	55	Daniil Kalland	Solve safety-related issues/Awaiting for robot to be operational
	55	Artur Gwozdowicz	Solve safety-related issues/Awaiting for robot to be operational
	20	Artur Gwozdowicz	Integration of Speech Recognition to Twincat 3 program
	20	Artur Gwozdowicz	Integration of Gripper control to Twincat 3 program
	30	Artur Gwozdowicz	Twincat 3 Programming, System integration
	45	Daniil Kalland	Designing and implementing of Object recognition
	5	Daniil Kalland	3D modelling
25.04.2023 - 15.05.2023	7	Daniil Kalland	Final Design and implementation of robots surroundings
	10	Artur Gwozdowicz	Final Design and implementation of robots surroundings
	90	Daniil Kalland	Designing and implementing of Object recognition
	96	Artur Gwozdowicz	KUKA LBR IIWA programming, Integration with Twincat 3
	30	Artur Gwozdowicz	Twincat 3 Programming, Implementing Manual Controls
15.05.2023 - 1.06.2023	40	Artur Gwozdowicz	KUKA LBR IIWA programming
	40	Artur Gwozdowicz	Twincat 3 Programming, Code restructurization
	65	Daniil Kalland	Designing and implementing of Object recognition
	5	Daniil Kalland	Correcting 3D Designs
1.06.2023 - 20.06.2023	15	Artur Gwozdowicz	Integrating Object Recognition to Twincat 3
	15	Artur Gwozdowicz	Final adjustments of KUKA LBR IIWA, Twincat 3 program
	15	Daniil Kalland	Final adjustments of Object Recognition program
	105	Artur Gwozdowicz	Report writing
	105	Daniil Kalland	Report writing

E Raspberry Pi - code

```
import cv2
import time
import socket
import numpy as np
import math

HOST = '192.168.0.101'  # IP address of the Raspberry Pi
PORT = 8000  # Port used for connection

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)

BlueColor = (255, 0, 0)
OrangeColor = (0, 200, 255)
GreenColor = (0, 255, 0)
RedColor = (0, 0, 255)
YellowColor = (0, 255, 255)

blueStr = 'Blue'
orangeStr = 'Orange'
redStr = 'Red'
yellowStr = 'Yellow'

xZero = 0
yZero = 0
angleZero = 0
xZeroData = str(xZero)
yZeroData = str(yZero)
angleZeroData = str(angleZero)

objectZeroData = [xZeroData, yZeroData, angleZeroData]
objectRedDataString = ';'.join(objectZeroData)
objectGreenDataString = ';'.join(objectZeroData)
objectBlueDataString = ';'.join(objectZeroData)

# Define the lower and upper color thresholds for each color in the flag
lower_red = np.array([0, 100, 70])  # [166, 150, 109]
upper_red = np.array([15, 250, 170])  # [186, 250, 209]

lower_blue = np.array([90, 145, 205])  # [102, 176, 176]
upper_blue = np.array([120, 255, 255])  # [122, 255, 255]

lower_orange = np.array([10, 140, 180])  # [70, 25, 145]
upper_orange = np.array([25, 255, 255])  # [90, 125, 245]

lower_yellow = np.array([30, 70, 180])  # [21, 87, 157]
upper_yellow = np.array([50, 160, 255])  # [41, 187, 255]

camera_port = '/dev/video0'  # this is the camera port number (This can vary
↪  from, 0 - 10 from pc to pc)
cam = cv2.VideoCapture(camera_port)  # Set the camera capture device

width = 1024  # 1024
height = 800  # 648
```

```
cam.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
cam.set(cv2.CAP_PROP_FPS, 30)
cam.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))


def MyTool(myFrame, contours, topContours, color, colorName, Offset):
    global objectDataString
    if contours:
        for cnt in contours:
            contour_area = cv2.contourArea(cnt)
            if contour_area > 10:
                rect = cv2.minAreaRect(cnt)
                objectBox = cv2.boxPoints(rect)
                objectBox = np.int0(objectBox)

                # Get parameters of the rotated bounding box

                x = int(rect[0][0])
                y = int(rect[0][1])

                black_area_x_min = -312
                black_area_x_max = 312
                black_area_y_min = -720
                black_area_y_max = -401

                scale_x = (black_area_x_max - black_area_x_min) / 1024
                scale_y = (black_area_y_max - black_area_y_min) / 648

                xR = int((black_area_x_max - (x * scale_x)) * 1.8) - 235
                yR = int((black_area_y_min + (y * scale_y)) * 2.2) + 840

                center = (x, y)
                angle = int(rect[2])

                centerText = "X:" + str(x) + " Y:" + str(y)
                robotCenterText = "Xr:" + str(xR) + " Yr:" + str(yR)

                cv2.putText(myFrame, robotCenterText, (center[0], center[1]),
                ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1,
                           cv2.LINE_AA)
                cv2.circle(myFrame, (center[0], center[1]), 2, RedColor, 2)

                cv2.drawContours(myFrame, [objectBox], 0, color, 2)

                if len(red_contours) > 0 and len(topContours) > 0:
                    red_contour = red_contours[0]
                    topContour = topContours[0]

                    xOffset = 25
                    yOffset = 85

                    # Get the bounding rectangles of the contours
                    red_rect = cv2.minAreaRect(red_contour)
                    top_rect = cv2.minAreaRect(topContour)

                    # Compare the y-coordinate of the centers of the bounding
                    ↪  rectangles
```

```
# If blue rectangle is on the top
if red_rect[0][1] > top_rect[0][1] and red_rect[0][0] <=
↪  top_rect[0][0]:
    # orientation = "Blue is top, Red is bottom"
    angle = angle
    angleText = "Angle: " + str(angle)
    cv2.putText(myFrame, angleText, (center[0] - 100,
    ↪  center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
                0.7, (0, 0, 0), 1, cv2.LINE_AA)
    if Offset is True:
        # Assuming you have the top rectangle 'top_rect'
        center_x = int(top_rect[0][0])
        center_y = int(top_rect[0][1])
        rect_angle = math.radians(angle)  # Convert angle to
        ↪  radians

        right_endpoint_x = center_x + xOffset
        right_endpoint_y = center_y - yOffset

        # Rotate the right endpoint around the center
        rotated_right_endpoint_x = center_x +
        ↪  (right_endpoint_x - center_x) *
        ↪  math.cos(rect_angle) - (right_endpoint_y -
        ↪  center_y) * math.sin(rect_angle)
        rotated_right_endpoint_y = center_y +
        ↪  (right_endpoint_x - center_x) *
        ↪  math.sin(rect_angle) + (right_endpoint_y -
        ↪  center_y) * math.cos(rect_angle)

        # Round the coordinates if needed
        rotated_right_endpoint_x =
        ↪  round(rotated_right_endpoint_x)
        rotated_right_endpoint_y =
        ↪  round(rotated_right_endpoint_y)

        # Sciling tool offset from x and y coordinates to xR
        ↪  and yR robot coordinates
        xToffset = int((black_area_x_max -
        ↪  (rotated_right_endpoint_x * scale_x)) * 1.8) -
        ↪  235
        yToffset = int((black_area_y_min +
        ↪  (rotated_right_endpoint_y * scale_y)) * 2.2) +
        ↪  840

        xR = xToffset
        yR = yToffset

        offsetCoordinates = "Xt:" + str(xR) + " Yt:" +
        ↪  str(yR)

        # Draw the line connecting the rotated endpoints
        cv2.line(myFrame, (center_x, center_y),
        ↪  (rotated_right_endpoint_x,
        ↪  rotated_right_endpoint_y), (0, 0, 255), 2)
        cv2.circle(myFrame, (rotated_right_endpoint_x,
        ↪  rotated_right_endpoint_y), 2, GreenColor, 2)
```

```
                cv2.putText(myFrame, offsetCoordinates,
                ↪  (rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y),
                ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1,
                ↪  cv2.LINE_AA)
            time.sleep(0.05)

        elif red_rect[0][1] > top_rect[0][1] and red_rect[0][0] >=
        ↪  top_rect[0][0]:
            angle = -(90 - angle)
            angleText = "Angle: " + str(angle)
            # angleBox = cv2.rectangle(frame, (center[0] - 100,
            ↪  center[1] + 100), (center[0] + 80, center[1] + 125),
            ↪  (255, 255, 255), -1)
            cv2.putText(myFrame, angleText, (center[0] - 100,
            ↪  center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
                        0.7, (0, 0, 0), 1, cv2.LINE_AA)
            if Offset is True:
                # Assuming you have the top rectangle 'top_rect'
                center_x = int(top_rect[0][0])
                center_y = int(top_rect[0][1])
                rect_angle = math.radians(angle)  # Convert angle to
                ↪  radians

                right_endpoint_x = center_x - xOffset
                right_endpoint_y = center_y - yOffset

                # Rotate the right endpoint around the center
                rotated_right_endpoint_x = center_x -
                ↪  (right_endpoint_x - center_x) *
                ↪  math.cos(rect_angle) - (right_endpoint_y -
                ↪  center_y) * math.sin(rect_angle)
                rotated_right_endpoint_y = center_y -
                ↪  (right_endpoint_x - center_x) *
                ↪  math.sin(rect_angle) + (right_endpoint_y -
                ↪  center_y) * math.cos(rect_angle)

                # Round the coordinates if needed
                rotated_right_endpoint_x =
                ↪  round(rotated_right_endpoint_x)
                rotated_right_endpoint_y =
                ↪  round(rotated_right_endpoint_y)

                # Sciling tool offset from x and y coordinates to xR
                ↪  and yR robot coordinates
                xToffset = int((black_area_x_max -
                ↪  (rotated_right_endpoint_x * scale_x)) * 1.8) -
                ↪  235
                yToffset = int((black_area_y_min +
                ↪  (rotated_right_endpoint_y * scale_y)) * 2.2) +
                ↪  840

                xR = xToffset
                yR = yToffset

                offsetCoordinates = "Xt:" + str(xR) + " Yt:" +
                ↪  str(yR)
```

```
                # Draw the line connecting the rotated endpoints
                cv2.line(myFrame, (center_x, center_y),
                ↪  (rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y), (0, 0, 255), 2)
                cv2.circle(myFrame, (rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y), 2, GreenColor, 2)
                cv2.putText(myFrame, offsetCoordinates,
                ↪  (rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y),
                ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1,
                ↪  cv2.LINE_AA)
            time.sleep(0.05)

        # If the red rectangle is on the top
        elif red_rect[0][1] < top_rect[0][1] and red_rect[0][0] >=
        ↪  top_rect[0][0]:
            angle = -(180 - angle)
            angleText = "Angle: " + str(angle)
            cv2.putText(myFrame, angleText, (center[0] - 100,
            ↪  center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
                        0.7, (0, 0, 0), 1, cv2.LINE_AA)
            if Offset is True:
                # Assuming you have the top rectangle 'top_rect'
                center_x = int(top_rect[0][0])
                center_y = int(top_rect[0][1])
                rect_angle = math.radians(angle)  # Convert angle to
                ↪  radians

                right_endpoint_x = center_x - xOffset
                right_endpoint_y = center_y - yOffset

                # Rotate the right endpoint around the center
                rotated_right_endpoint_x = center_x -
                ↪  (right_endpoint_x - center_x) *
                ↪  math.cos(rect_angle) - (right_endpoint_y -
                ↪  center_y) * math.sin(rect_angle)
                rotated_right_endpoint_y = center_y -
                ↪  (right_endpoint_x - center_x) *
                ↪  math.sin(rect_angle) + (right_endpoint_y -
                ↪  center_y) * math.cos(rect_angle)

                # Round the coordinates if needed
                rotated_right_endpoint_x =
                ↪  round(rotated_right_endpoint_x)
                rotated_right_endpoint_y =
                ↪  round(rotated_right_endpoint_y)

                # Sciling tool offset from x and y coordinates to xR
                ↪  and yR robot coordinates
                xToffset = int((black_area_x_max -
                ↪  (rotated_right_endpoint_x * scale_x)) * 1.8) -
                ↪  235
                yToffset = int((black_area_y_min +
                ↪  (rotated_right_endpoint_y * scale_y)) * 2.2) +
                ↪  840

                xR = xToffset
                yR = yToffset
```

```
                offsetCoordinates = "Xt:" + str(xR) + " Yt:" +
                ↪  str(yR)

                # Draw the line connecting the rotated endpoints
                cv2.line(myFrame, (center_x,
                ↪  center_y),(rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y), (0, 0, 255), 2)
                cv2.circle(myFrame, (rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y), 2, GreenColor, 2)
                cv2.putText(myFrame,
                ↪  offsetCoordinates,(rotated_right_endpoint_x,
                ↪  rotated_right_endpoint_y),
                ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1,
                ↪  cv2.LINE_AA)
            time.sleep(0.05)

        elif red_rect[0][1] < top_rect[0][1] and red_rect[0][0] <=
        ↪  top_rect[0][0]:
            angle = 90 + angle
            angleText = "Angle: " + str(angle)
            cv2.putText(myFrame, angleText, (center[0] - 100,
            ↪  center[1] + 120), cv2.FONT_HERSHEY_SIMPLEX,
                        0.7, (0, 0, 0), 1, cv2.LINE_AA)
            if Offset is True:
                # Assuming you have the top rectangle 'top_rect'
                center_x = int(top_rect[0][0])
                center_y = int(top_rect[0][1])
                rect_angle = math.radians(angle)  # Convert angle to
                ↪  radians

                right_endpoint_x = center_x + xOffset
                right_endpoint_y = center_y - yOffset

                # Rotate the right endpoint around the center
                rotated_right_endpoint_x = center_x +
                ↪  (right_endpoint_x - center_x) *
                ↪  math.cos(rect_angle) - (right_endpoint_y -
                ↪  center_y) * math.sin(rect_angle)
                rotated_right_endpoint_y = center_y +
                ↪  (right_endpoint_x - center_x) *
                ↪  math.sin(rect_angle) + (right_endpoint_y -
                ↪  center_y) * math.cos(rect_angle)

                rotated_right_endpoint_x =
                ↪  round(rotated_right_endpoint_x)
                rotated_right_endpoint_y =
                ↪  round(rotated_right_endpoint_y)

                # Sciling tool offset from x and y coordinates to xR
                ↪  and yR robot coordinates
                xToffset = int((black_area_x_max -
                ↪  (rotated_right_endpoint_x * scale_x)) * 1.8) -
                ↪  235
                yToffset = int((black_area_y_min +
                ↪  (rotated_right_endpoint_y * scale_y)) * 2.2) +
                ↪  840
```

```
                            xR = xToffset
                            yR = yToffset

                            offsetCoordinates = "Xt:" + str(xR) + " Yt:" +
                            ↪  str(yR)

                            # Draw the line connecting the rotated endpoints
                            cv2.line(myFrame, (center_x, center_y),
                            ↪  (rotated_right_endpoint_x,
                            ↪  rotated_right_endpoint_y), (0, 0, 255), 2)
                            cv2.circle(myFrame, (rotated_right_endpoint_x,
                            ↪  rotated_right_endpoint_y), 2, GreenColor, 2)
                            cv2.putText(myFrame, offsetCoordinates,
                            ↪  (rotated_right_endpoint_x,
                            ↪  rotated_right_endpoint_y),
                            ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1,
                            ↪  cv2.LINE_AA)
                        time.sleep(0.05)

                xData = str(xR)
                yData = str(yR)
                angleData = str(angle)

                objectData = [xData, yData, angleData]
                objectDataString = ';'.join(objectData)
                print(f'{colorName} coordinates:', xR, yR)

    else:
        xR = 0
        yR = 0
        angle = 0

        xData = str(xR)
        yData = str(yR)
        angleData = str(angle)

        objectData = [xData, yData, angleData]
        objectDataString = ';'.join(objectData)

    return objectDataString


# Waiting for a client to connect to the Raspberry Pi
print('Waiting for a client to connect to the server...')
conn, addr = s.accept()
print('Connected by', addr)

data = conn.recv(1024)
print(data.decode())

while True:
    # Detect object coordinates part
    ret, frame = cam.read()
    frameROI = frame[100:435,
               215:875]  # [130:400, 80:560] Change values from here to affect
               ↪  boundaries of robot pick up area

    # Convert the frame to the HSV color space
```

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
hsvROI = cv2.cvtColor(frameROI, cv2.COLOR_BGR2HSV)

# Perform color segmentation to detect the red color in the frame
red_mask = cv2.inRange(hsvROI, lower_red,
                       upper_red)  # Change hsv to hsvROI if you want
                       ↪  tracking in region of interest!

# Perform color segmentation to detect the blue color in the frame
blue_mask = cv2.inRange(hsvROI, lower_blue,
                        upper_blue)  # Change hsv to hsvROI if you want
                        ↪  tracking in region of interest!

# Perform color segmentation to detect the blue color in the frame
orange_mask = cv2.inRange(hsvROI, lower_orange,
                          upper_orange)  # Change hsv to hsvROI if you want
                          ↪  tracking in region of interest!

# Perform color segmentation to detect the blue color in the frame
yellow_mask = cv2.inRange(hsvROI, lower_yellow,
                          upper_yellow)  # Change hsv to hsvROI if you want
                          ↪  tracking in region of interest!

# Combine the red and blue masks
maskRedBlue = red_mask + blue_mask
maskRedOrange = red_mask + orange_mask
maskRedYellow = red_mask + yellow_mask

# Apply a series of morphological operations to remove noise and fill gaps
kernel = np.ones((5, 5), np.uint8)
maskRedBlue = cv2.morphologyEx(maskRedBlue, cv2.MORPH_OPEN, kernel)
maskRedBlue = cv2.morphologyEx(maskRedBlue, cv2.MORPH_CLOSE, kernel)

maskRedOrange = cv2.morphologyEx(maskRedOrange, cv2.MORPH_OPEN, kernel)
maskRedOrange = cv2.morphologyEx(maskRedOrange, cv2.MORPH_CLOSE, kernel)

maskRedYellow = cv2.morphologyEx(maskRedYellow, cv2.MORPH_OPEN, kernel)
maskRedYellow = cv2.morphologyEx(maskRedYellow, cv2.MORPH_CLOSE, kernel)

# Find contours of the detected regions in the mask
contoursRedBlue, _ = cv2.findContours(maskRedBlue, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
contoursRedOrange, _ = cv2.findContours(maskRedOrange, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
contoursRedYellow, _ = cv2.findContours(maskRedYellow, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)

# Angle calculation part

# Find contours in the masks
red_contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
blue_contours, _ = cv2.findContours(blue_mask, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
orange_contours, _ = cv2.findContours(orange_mask, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
yellow_contours, _ = cv2.findContours(yellow_mask, cv2.RETR_EXTERNAL,
↪  cv2.CHAIN_APPROX_SIMPLE)
```

```
    RedBlue_contours, _ = cv2.findContours(maskRedBlue, cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)
    RedOrange_contours, _ = cv2.findContours(maskRedOrange, cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)
    RedYellow_contours, _ = cv2.findContours(maskRedYellow, cv2.RETR_EXTERNAL,
    ↪  cv2.CHAIN_APPROX_SIMPLE)

    # Sort the contours by area in descending order
    red_contours = sorted(red_contours, key=cv2.contourArea, reverse=True)
    blue_contours = sorted(blue_contours, key=cv2.contourArea, reverse=True)
    orange_contours = sorted(orange_contours, key=cv2.contourArea, reverse=True)
    yellow_contours = sorted(yellow_contours, key=cv2.contourArea, reverse=True)

    RedBlueData = MyTool(frameROI, contoursRedBlue, blue_contours, BlueColor,
    ↪  blueStr, Offset=False)
    RedOrangeData = MyTool(frameROI, contoursRedOrange, orange_contours,
    ↪  OrangeColor, orangeStr, Offset=False)
    RedYellowData = MyTool(frameROI, contoursRedYellow, yellow_contours,
    ↪  YellowColor, yellowStr, Offset=True)

    list = [RedBlueData, RedOrangeData, RedYellowData]
    stringedList = "%".join(list)
    conn.sendall(stringedList.encode())
    time.sleep(0.1)

    # Display the frame with bounding rectangles
    cv2.imshow('My WEBcam', frame)
    cv2.moveWindow('My WEBcam', 500, 100)
    cv2.moveWindow('Image Blue', 1550, 100)
    cv2.moveWindow('Image Green', 1650, 100)

    # Check for key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object and close windows
cam.release()
cv2.destroyAllWindows()
conn.close()
```

F PC - main code

```
import pyads as pyads
import socket as socket
import multiprocessing
import threading
import serial
import time
import speech_recognition as sr
import numpy as np

from GripperControl import GripperControl
from SpeechRecognition import SpeechRecognition

AMSNETID = '158.38.140.64.1.1' #AMS of the PLC

plc = pyads.Connection(AMSNETID, pyads.PORT_TC3PLC1)
plc.open()
print(f"Connected?: {plc.is_open}")
print(f"Local Address? : {plc.get_local_address()}")
print(plc.read_state())
#Connect to PLC and read it's state




def speech():
    while True:

            r = sr.Recognizer()
       # if plc.read_by_name("KUKA_OPERATION.RequestSpeech"):

            # Speech Recognition part:
            time.sleep(1)
            plc.write_by_name("KUKA_OPERATION_POU.RequestSpeech", False)
            with sr.Microphone() as source:
                print("Say something!")
                audio = r.listen(source)
                # recognize speech using Google Speech Recognition
                try:
                    text = r.recognize_google(audio)
                    print(f"You said: {text}")
                    plc.write_by_name("KUKA_OPERATION_POU.sWord", text)
                    time.sleep(2)
                    plc.write_by_name("KUKA_OPERATION_POU.sWord", '')
                except sr.UnknownValueError:
                    print("Google Speech Recognition could not understand audio")
                except sr.RequestError as e:
                    print(f"Could not request results from Google Speech
                    ↪  Recognition service; {e}")

def grip():
    activateCommand = bytearray(
        [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00,
        ↪  0x73, 0x30])
    checkCommand = bytearray([0x09, 0x03, 0x07, 0xD0, 0x00, 0x01, 0x85,
```

```
                        0xCF])  # GACT (Gripper Action) command for the
                        ↪  gripper activation

feedBackCommand = bytearray(
    [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x0B, 0x00, 0x03, 0x08, 0x00,
    ↪  0x00, 0x0C, 0x36])
deactivateForceControl = bytearray(
    [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00,
    ↪  0x00, 0xDE, 0xB9])
readForceFeedback = bytearray([0x09, 0x03, 0x07, 0xD0, 0x00, 0x01, 0x84,
↪  0x0F])

openCommand = bytearray(
    [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x09, 0x00, 0x00, 0x00, 0xFF,
    ↪  0xFF, 0x72, 0x19])
closeCommand = bytearray(
    [0x09, 0x10, 0x03, 0xE8, 0x00, 0x03, 0x06, 0x09, 0x00, 0x00, 0xFF, 0xFF,
    ↪  0xFF, 0x42, 0x29])

ser = serial.Serial(port='COM16', baudrate=115200, timeout=1,
                    parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
                    ↪  bytesize=serial.EIGHTBITS)
gripOpened = True
gripClosed = False
stateNow = gripOpened

objDetected = False
gripLow = 0
gripHigh = 1

printed = False
#ser.write(activateCommand)
time.sleep(1)



print('grip Init')
while True:
    ser.write(checkCommand)

    response = ser.readline()

    # print("Response:", response)
    status_byte = response[3]

    print(response)
    if plc.read_by_name("KUKA_OPERATION_POU.RequestGripClose"):
        ser.write(closeCommand)
        time.sleep(1)
        # Change the states for PLC
        plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Opened", False)
        plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Closed", True)
        plc.write_by_name("KUKA_OPERATION_POU.objectDetected", objDetected)
        # print("Gripper is closed")
        # print("Checking status...")


    elif plc.read_by_name("KUKA_OPERATION_POU.RequestGripOpen"):
```

```
            ser.write(openCommand)
            objDetected = False
            time.sleep(1)
            # Change the states for PLC
            plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Opened", True)
            plc.write_by_name("KUKA_OPERATION_POU.RPI_Grip_Closed", False)
            plc.write_by_name("KUKA_OPERATION_POU.objectDetected", objDetected)
            if not printed:
                print("Gripper is opened")
                print("Gripper lost the object")
                printed = True

        if status_byte == 0xb9:  # If gripper senses an object
            objDetected = True
            plc.write_by_name("KUKA_OPERATION_POU.objectDetected", objDetected)
            print("Gripper has successfully grasped an object")

        elif status_byte == 0xf9:  # If gripper doesn't sense any object
            objDetected = False
            plc.write_by_name("KUKA_OPERATION_POU.objectDetected", objDetected)
            #print("Gripper did not find any object")

def ObjectDetectionRPI():
        HOST = '192.168.0.101'  # This is the Raspberry Pi's hostname or IP
        ↪  address
        PORT = 8000  # The same port as the server

        # Create a socket object
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # Connect to the server
        s.connect((HOST, PORT))
        while True:
             #Send a message to the server
             # Send a message to the server
            s.sendall('Recieving...'.encode())
            data = s.recv(1024).decode()
            dataSplit = data.split('%')

            HammerData = dataSplit[0].split(';')
            ScrewdriverData = dataSplit[1].split(';')
            PliersData = dataSplit[2].split(';')




            # Send data about red object to the PLC
            if plc.read_by_name("KUKA_OPERATION_POU.RequestCoordinates"):
                plc.write_by_name("GVL.Raspberry_Screwdriver_X",
                ↪  int(ScrewdriverData[0]))
                plc.write_by_name("GVL.Raspberry_Screwdriver_Y",
                ↪  int(ScrewdriverData[1]))
                plc.write_by_name("GVL.Raspberry_Screwdriver_Angle",
                ↪  int(ScrewdriverData[2]))

                plc.write_by_name("GVL.Raspberry_Hammer_X", int(HammerData[0]))
                plc.write_by_name("GVL.Raspberry_Hammer_Y", int(HammerData[1]))
```

```
                plc.write_by_name("GVL.Raspberry_Hammer_Angle",
                ↪  int(HammerData[2]))

                plc.write_by_name("GVL.Raspberry_Pliers_X", int(PliersData[0]))
                plc.write_by_name("GVL.Raspberry_Pliers_Y", int(PliersData[1]))
                plc.write_by_name("GVL.Raspberry_Pliers_Angle",
                ↪  int(PliersData[2]))




#Gripper = GripperControl(port= 'COM16',baudrate=115200, plc=plc)
#SR = SpeechRecognition(plc=plc)
#ObjDetection = ObjectDetection(plc=plc, s=s)


if __name__ == '__main__':
        gr = multiprocessing.Process(target=grip)
        gr.start()
      #  sp = multiprocessing.Process(target=speech)
      #  sp.start()
        od = multiprocessing.Process(target = ObjectDetectionRPI)
        od.start()



#p2.start()
#p1.start()


#p1.join()
#p2.join()



#multiprocessing.Process(target = SR.start()).start()
#t3 = threading.Thread(target = ObjDetection())
```

G TwinCAT 3 code

GVL variables

```
{attribute 'qualified_only'}
VAR_GLOBAL


    KUKA_TaskInfo_Coordinate_X AT %I* : UINT;
    KUKA_TaskInfo_Coordinate_Y AT %I* : UINT;
    KUKA_TaskInfo_Coordinate_Z AT %I* : UINT;
    KUKA_Current_Operation AT %I* : UINT;
    KUKA_Grip_State_Request AT %I* : UINT;
    KUKA_AnswerRequest AT %I* : UINT;

    PLC_SendRequest AT %Q* : UINT;
    PLC_OperationType AT %Q* : UINT;
    PLC_ObjectType AT %Q* : UINT;
    PLC_Coordinate_X AT %Q* : UINT;
    PLC_Coordinate_Y AT %Q* : UINT;
    PLC_Grip_State AT %Q* : UINT;
    PLC_Object_Angle AT %Q* : UINT;


    Raspberry_Queue_Index: INT;
    Raspberry_Operation_Type: INT;
    Raspberry_Object_Type: INT;

    Raspberry_Screwdriver_X: INT;
    Raspberry_Screwdriver_Y: INT;
    Raspberry_Screwdriver_Angle: INT;

    Raspberry_Hammer_X: INT;
    Raspberry_Hammer_Y: INT;
    Raspberry_Hammer_Angle: INT;

    Raspberry_Pliers_X: INT;
    Raspberry_Pliers_Y: INT;
    Raspberry_Pliers_Angle: INT;


    ManualMode: BOOL;
    setManualOP: STRING;
    setManualObjType: STRING;

END_VAR
```

HandleRequests

```
FUNCTION_BLOCK HandleRequests
VAR_INPUT

    KUKA_Gripper_Request: UINT;
    RPI_Grip_Closed: BOOL;
    RPI_Grip_Opened: BOOL;
    ObjectDetected: BOOL;
    KUKA_Current_Operation: UINT;
END_VAR
VAR_OUTPUT
    RequestGripClose: BOOL;
    RequestGripOpen: BOOL;
    GripStatus: UINT;
```

```
//KUKA TO RPI GRIPPER REQUESTING
IF KUKA_Gripper_Request = 1 AND NOT RPI_Grip_Opened THEN
    RequestGripOpen := TRUE;
    RequestGripClose := FALSE;
    ELSIF KUKA_Gripper_Request = 0 AND NOT RPI_Grip_Closed THEN
    RequestGripOpen := FALSE;
    RequestGripClose := TRUE;
    ELSE
    RequestGripOpen := FALSE;
    RequestGripClose := FALSE;
END_IF

IF RPI_Grip_Closed THEN
    GripStatus := 0;
ELSIF RPI_Grip_Opened THEN
    GripStatus := 1;
END_IF


//SPEECH RECOGNITION AND COORDINATES REQUEST

IF KUKA_Current_Operation = 0 THEN

    RequestCoordinates := TRUE;
        IF GVL.ManualMode THEN
            RequestSpeech := FALSE;
        ELSE
            RequestSpeech := TRUE;
        END_IF
ELSE
    RequestSpeech := FALSE;
    RequestCoordinates := FALSE;
END_IF


//TASK REQUEST

IF NOT ObjectDetected AND KUKA_Current_Operation = 0  THEN
    ObjectRequest := TRUE;
ELSE
    ObjectRequest := FALSE;
END_IF
IF KUKA_Current_Operation = 0 THEN
    OperationRequest := TRUE;
ELSE
    OperationRequest := FALSE;
END_IF
```

KUKA_Task_Information

```
FUNCTION_BLOCK KUKA_Task_Information
VAR_INPUT

END_VAR
VAR_OUTPUT
    CurrentTaskCoordinateX: INT;
    CurrentTaskCoordinateY: INT;
    CurrentTaskCoordinateZ: INT;

    KUKA_Operation_String : STRING;

END_VAR
VAR
END_VAR
```

```
CurrentTaskCoordinateX:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_X)-1000;
CurrentTaskCoordinateY:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_Y)-1000;
CurrentTaskCoordinateZ:= UINT_TO_INT(GVL.KUKA_TaskInfo_Coordinate_Z)-1000;

IF GVL.KUKA_Current_Operation = 1 THEN
    KUKA_Operation_String := 'Pick Up';
ELSIF GVL.KUKA_Current_Operation = 2 THEN
    KUKA_Operation_String := 'Drop';
ELSIF GVL.KUKA_Current_Operation = 3 THEN
    KUKA_Operation_String := 'Put Back';
ELSIF GVL.KUKA_Current_Operation = 0 THEN
    KUKA_Operation_String := 'Hold';
END_IF
```

ManualMode

```
FUNCTION_BLOCK ManualMode
VAR_INPUT
    ManualCommandInput: STRING;
    ManualOperationInput: STRING;
    ManualObjectInput: STRING;
    ManualMode: BOOL;
END_VAR
VAR_OUTPUT
    ManualOperationType: UINT;
    ManualObjectType: UINT;

END_VAR
VAR

Delay : TON;
    delayet: TIME;
    testvar: BOOL;
    delayETt: TIME;

    Separator: INT;
END_VAR
```

```
IF ManualMode THEN

    IF LEN(ManualCommandInput) > 1 THEN

    Separator := FIND(ManualCommandInput, ' ');

        IF Separator > 0 THEN
            ManualOperationInput := MID(ManualCommandInput, Separator-1, 1);
            ManualObjectInput := MID(ManualCommandInput, LEN(ManualCommandInput)-Separator+1, Separator+1);
        END_IF

        IF Separator = 0 THEN
            ManualOperationInput := ManualCommandInput;
        END_IF


    END_IF

IF (ManualOperationInput = 'pickup') OR (ManualOperationInput ='Pickup') OR ManualOperationInput ='pick up' OR ManualOperationInput = 'Pick up' AND ManualObjectType = NOT 0 THEN
    ManualOperationType := 1;
END_IF
IF (ManualOperationInput = 'drop' OR ManualOperationInput = 'Drop')  THEN
    ManualOperationType := 2;
END_IF
IF (ManualOperationInput = 'give' OR ManualOperationInput = 'Give')  THEN
    ManualOperationType := 3;
END_IF

IF ManualObjectInput = 'Screwdriver' OR ManualObjectInput = 'screwdriver' THEN
    ManualObjectType := 1;
END_IF
IF ManualObjectInput = 'Hammer' OR ManualObjectInput = 'hammer' THEN
    ManualObjectType := 2;
END_IF
IF ManualObjectInput = 'Pliers' OR ManualObjectInput = 'pliers' THEN
    ManualObjectType := 3;
END_IF

IF ManualOperationInput = '' OR ManualObjectInput  = '' THEN
    ManualOperationType := 0;
    ManualObjectType := 0;
END_IF
```

```
IF LEN(ManualOperationInput) > 1 OR LEN(ManualObjectInput) > 1 THEN
    Delay(IN:=TRUE, PT:=T#2S);

    IF Delay.Q THEN
    ManualOperationType := 0;
    ManualObjectType := 0;
    Delay(IN := FALSE);
    ManualOperationInput := ' ';
    ManualObjectInput := ' ';
    ManualCommandInput := ' ';

    END_IF

END_IF
END_IF
```

SpeechRecognitionHandling

```
FUNCTION_BLOCK SpeechRecognitionHandling

VAR_INPUT
    SpeechCommand: STRING;
    ManualMode: BOOL;
END_VAR
VAR_OUTPUT
SpeechOperationType: UINT;
SpeechObjectType: UINT;

END_VAR
VAR
    PickUp: INT;
    Drop: INT;
    Give: INT;
    Hold: INT;

    Screwdriver: INT;
    Hammer: INT;
    Pliers: INT;

    TonDelay : TON;
    et: TIME;
    TONTrig: BOOL;
END_VAR
```

```
IF NOT GVL.ManualMode THEN


PickUp := FIND(SpeechCommand, 'pick up');
Drop := FIND(SpeechCommand, 'drop');
Give := FIND(SpeechCommand, 'give');
Hold := FIND(SpeechCommand, 'hold');


Screwdriver := FIND(SpeechCommand, 'screwdriver');
Hammer := FIND(SpeechCommand, 'hammer');
Pliers := FIND(SpeechCommand, 'pliers');

IF PickUp > 0 AND (Screwdriver + Hammer + Pliers) > 0THEN
    SpeechOperationType := 1;
ELSIF Drop > 0 THEN
    SpeechOperationType := 2;
ELSIF Give > 0 THEN
    SpeechOperationType := 3;
ELSIF Hold > 0 THEN
    SpeechOperationType := 4;
END_IF


IF Screwdriver > 0 THEN
    SpeechObjectType := 1;

ELSIF Hammer > 0 THEN
    SpeechObjectType := 2;

ELSIF Pliers > 0 THEN
    SpeechObjectType := 3;

END_IF

IF SpeechCommand = '' THEN
    SpeechObjectType := 0;
    SpeechOperationType := 0;
END_IF

END_IF
```

TaskHandling

```
FUNCTION_BLOCK TaskHandling
VAR_INPUT
    SpeechOP: UINT;
    SpeechOBJ: UINT;
    ManualOP: UINT;
    ManualOBJ: UINT;

    OperationRequest: BOOL;
    ObjectRequest: BOOL;
END_VAR
VAR_OUTPUT
    ToKukaOperation: UINT;
    ToKukaObject: UINT;
    ToKukaObjectCoordinateX: UINT;
    ToKukaObjectCoordinateY: UINT;
    ToKukaObjectAngle: UINT;
END_VAR
VAR
    ObjectCoordinates: INT;
    setOBJ: BOOL;
    MsetOBJ: BOOL;
END_VAR
```

```
IF OperationRequest THEN
    IF (ManualOP = 1 OR SpeechOP = 1) AND NOT ObjectRequest THEN
        ToKukaOperation := 0;
        ELSE

        IF GVL.ManualMode THEN
            ToKukaOperation := ManualOP;
        ELSE
            ToKukaOperation := SpeechOP;
        END_IF

    END_IF

END_IF



IF NOT OperationRequest THEN
    ToKukaOperation := 0;
END_IF


IF ObjectRequest THEN
    ToKukaObject := 0;
    IF GVL.ManualMode THEN

        IF ManualOBJ > 0 THEN
            MsetOBJ := TRUE;
            IF MsetOBJ = TRUE THEN
                ToKukaObject := ManualOBJ;
                MsetOBJ := FALSE;
            END_IF
        END_IF
        ObjectCoordinates := UINT_TO_INT(ManualOBJ);
    ELSE

        IF SpeechOBJ > 0 THEN
            setOBJ := TRUE;
            IF setOBJ := TRUE THEN
                ToKukaObject := SpeechOBJ;
                setOBJ := FALSE;
            END_IF
        END_IF
        ObjectCoordinates:= UINT_TO_INT(SpeechOBJ);
    END_IF

END_IF
```

```
CASE ObjectCoordinates OF
    1:  //screwdriver

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Screwdriver_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Screwdriver_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Screwdriver_Angle)+180;

    2:

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Hammer_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Hammer_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Hammer_Angle)+180;

    3:

    ToKukaObjectCoordinateX := INT_TO_UINT(GVl.Raspberry_Pliers_X) + 1000;
    ToKukaObjectCoordinateY := INT_TO_UINT(GVl.Raspberry_Pliers_Y) + 1000;
    ToKukaObjectAngle := INT_TO_UINT(Raspberry_Pliers_Angle)+180;

    0:

    ToKukaObjectCoordinateX := 0;
    ToKukaObjectCoordinateY := 0;
    ToKukaObjectAngle := 0;


END_CASE
```

KUKA_Operation_POU

```
PROGRAM KUKA_OPERATION_POU
VAR
    SpeechRecognitionHandling_0: SpeechRecognitionHandling;
    sWord: STRING;
    SpeechOperationType: UINT;
    SpeechObjectType: UINT;
    RequestSpeech: BOOL;
    ManualMode_0: ManualMode;
    ManualOp: STRING;
    ManualObj: STRING;
    ManualOperationType: UINT;
    ManualObjectType: UINT;
    RequestHandling_0: RequestHandling;
    HandleRequests_0: HandleRequests;
    RPI_Grip_Closed: BOOL;
    RPI_Grip_Opened: BOOL;

    RequestGripClose: BOOL;
    RequestGripOpen: BOOL;
    ObjectDetected: BOOL;
    RequestCoordinates: BOOL;
    OperationRequest: BOOL;
    ObjectRequest: BOOL;
    TaskHandling_0: TaskHandling;
    KUKA_Task_Information_0: KUKA_Task_Information;
    KUKA_Operation_String: STRING;
    KUKA_Z: INT;
    KUKA_Y: INT;
    KUKA_X: INT;
END_VAR
```


1
SpeechRecognitionHandling_0
SpeechRecognitionHandling
sWord
SpeechCommand
SpeechOperationType
SpeechOperationType
GVL.ManualMode
ManualMode
SpeechObjectType
SpeechObjectType
2
ManualMode_0
ManualMode
ManualCommandInput
ManualOperationType
ManualOperationType
ManualOperationInput
ManualObjectType
ManualObjectType
ManualObjectInput
GVL.ManualMode
ManualMode
3
HandleRequests_0
HandleRequests
GVL.KUKA_Grip_State_Request
KUKA_Gripper_Request
RequestGripClose
RequestGripClose
RPI_Grip_Closed
RPI_Grip_Closed
RequestGripOpen
RequestGripOpen
RPI_Grip_Opened
RPI_Grip_Opened
GripStatus
GVL.PLC_Grip_State
ObjectDetected
ObjectDetected
RequestSpeech
RequestSpeech
GVL.KUKA_Current_Operation
KUKA_Current_Operation
RequestCoordinates
RequestCoordinates
OperationRequest
OperationRequest
ObjectRequest
ObjectRequest
4
TaskHandling_0
TaskHandling
SpeechOperationType
SpeechOP
ToKukaOperation
GVL.PLC_OperationType
SpeechObjectType
SpeechOBJ
ToKukaObject
GVL.PLC_ObjectType
ManualOperationType
ManualOP
ToKukaObjectCoordinateX
GVL.PLC_Coordinate_X
ManualObjectType
ManualOBJ
ToKukaObjectCoordinateY
GVL.PLC_Coordinate_Y
OperationRequest
OperationRequest
ToKukaObjectAngle
GVL.PLC_Object_Angle
ObjectRequest
ObjectRequest
5
KUKA_Task_Information_0
KUKA_Task_Information
CurrentTaskCoordinateX
KUKA_X
CurrentTaskCoordinateY
KUKA_Y
CurrentTaskCoordinateZ
KUKA_Z
KUKA_Operation_String
KUKA_Operation_String

H JAVA - code

RobotApplication

```
package application;
import javax.inject.Inject;

import com.kuka.common.ThreadUtil;
import com.kuka.generated.ioAccess.EL6695IOGroup;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;

import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.roboticsAPI.geometricModel.ObjectFrame;
import com.kuka.roboticsAPI.motionModel.CartesianPTP;
import com.kuka.roboticsAPI.motionModel.IMotionContainer;
import com.kuka.roboticsAPI.motionModel.PTP;
import com.kuka.roboticsAPI.motionModel.controlModeModel.PositionControlMode;

/**
 * Implementation of a robot application.
 * <p>
 * The application provides a {@link RoboticsAPITask#initialize()} and a
 * {@link RoboticsAPITask#run()} method, which will be called successively in
 * the application lifecycle. The application will terminate automatically after
 * the {@link RoboticsAPITask#run()} method has finished or after stopping the
 * task. The {@link RoboticsAPITask#dispose()} method will be called, even if an
 * exception is thrown during initialization or run.
 * <p>
 * <b>It is imperative to call <code>super.dispose()</code> when overriding the
 * {@link RoboticsAPITask#dispose()} method.</b>
 *
 * @see UseRoboticsAPIContext
 * @see #initialize()
 * @see #run()
 * @see #dispose()
 */

public class RobotApplication extends RoboticsAPIApplication {
        public static boolean startInfoThread = true;
    private LBR robot;
        @Inject
        private EL6695IOGroup io;
        public void initialize() {

                io.setKUKA_Current_Operation(0);
                robot = getContext().getDeviceFromType(LBR.class);
                RobotUtil.initializeRobotPosition(robot);
                RobotCurrentInfo info = new RobotCurrentInfo(robot, io);
                info.start();
        }

        @Override
        public void run() {
                io.setKUKA_Current_Operation(0);
                int RobotEnable = 1;
```

```
while(RobotEnable == 1){

        switch(io.getPLC_OperationType()){

        case 1:  //pickup command
                io.setKUKA_AnswerRequest(0);
                io.setKUKA_Current_Operation(1);

                int ObjectX = io.getPLC_Coordinate_X()-1000;
                int ObjectY = io.getPLC_Coordinate_Y()-1000;
                double ObjectAngleRad =
                ↪  Math.toRadians(io.getPLC_Object_Angle()-180);

                if (ValidateCoordinates(ObjectX,ObjectY)){
                        RobotUtil.PickupPrepare(robot);

                        while(GripperClosed(io.getPLC_Grip_State())){
                        ↪  //while grip is closed
                                io.setKUKA_Grip_State_Request(1);
                                ↪  //open grip
                                ThreadUtil.milliSleep(500);
                        }
                        System.out.println(ObjectAngleRad*0.0174532);
                        RobotUtil.PickupPrepareObjectPosition(robot,
                        ↪  ObjectX, ObjectY, 285,
                        ↪  ObjectAngleRad);

                        while(GripperOpened(io.getPLC_Grip_State())){
                        ↪  //while grip is opened
                                io.setKUKA_Grip_State_Request(0);
                                ↪  //close grip
                                ThreadUtil.milliSleep(500);
                        }
                        RobotUtil.afterPickup(robot);
                }
                ThreadUtil.milliSleep(3000);
                        io.setKUKA_Current_Operation(0);

                break;

        case 2:  //drop command
                     //prepare
                io.setKUKA_AnswerRequest(0);
                io.setKUKA_Current_Operation(2);
                while (io.getKUKA_Current_Operation() == 2){
                        RobotUtil.DropPrepare(robot);

                        switch(io.getPLC_ObjectType()){

                        case 1:  //drop Screwdriver
                                RobotUtil.DropScrewdriverPrepare(robot);
                                while(GripperClosed(io.getPLC_Grip_State())){
                                ↪  //while grip is closed
                                        io.setKUKA_Grip_State_Request(1);
                                        ↪  //open grip
                                        ThreadUtil.milliSleep(500);
                                }
```

```
                                        ThreadUtil.milliSleep(4000);
                                        RobotUtil.AfterDropPosition(robot);
                                        io.setKUKA_Current_Operation(0);
                                        break;
                                    case 2: //drop Hammer
                                        RobotUtil.DropHammerPrepare(robot);

                                        while(GripperClosed(io.getPLC_Grip_State())){
                                        ↪  //while grip is closed
                                            io.setKUKA_Grip_State_Request(1);
                                            ↪  //open grip
                                            ThreadUtil.milliSleep(500);
                                        }

                                        ThreadUtil.milliSleep(4000);
                                        RobotUtil.AfterDropPosition(robot);
                                        io.setKUKA_Current_Operation(0);

                                        break;
                                    case 3: //drop pliers
                                        RobotUtil.DropPliersPrepare(robot);

                                        while(GripperClosed(io.getPLC_Grip_State())){
                                        ↪  //while grip is closed
                                            io.setKUKA_Grip_State_Request(1);
                                            ↪  //open grip
                                            ThreadUtil.milliSleep(500);
                                        }
                                        ThreadUtil.milliSleep(4000);
                                        RobotUtil.AfterDropPosition(robot);
                                        io.setKUKA_Current_Operation(0);

                                        break;
                                    }
                            }
                            break;
                        case 3:  //give command
                            System.out.println("give object");
                            io.setKUKA_Grip_State_Request(1);  //open grip
                            ThreadUtil.milliSleep(500);
                            io.setKUKA_Current_Operation(0);
                            break;
                        case 0: //HOLD
                            io.setKUKA_Current_Operation(0);
                        ThreadUtil.milliSleep(500);
                        PTP HOLDPrepareJointPosition =
                        ↪  ptp(0,0.715,0,-1.17,0,0,0).setJointVelocityRel(0.2);
                        robot.move(HOLDPrepareJointPosition);
                            //waiting for new operation
                            break;



                    }

            }

            }
```

```
private static boolean ValidateCoordinates(int x, int y){
        if((x < -312) || (x > 312) || (y > -401) || (y<-720)) {
                return false;
        }
        else{
                return true;
        }

}

private static boolean GripperOpened(int state){
                if(state == 1){
                        return true;
                }
                else{
                        return false;
                }


        }
private static boolean GripperClosed(int state){
        if (state == 0){
                return true;
        }
        else{
                return false;
        }
}

}
```

RobotCurrentInfo

```
package application;

import com.kuka.common.ThreadUtil;
import com.kuka.generated.ioAccess.EL6695IOGroup;
import com.kuka.roboticsAPI.deviceModel.LBR;

public class RobotCurrentInfo extends Thread{
        private LBR robot;
        private EL6695IOGroup io;

        RobotCurrentInfo(LBR robot, EL6695IOGroup io){

                this.robot = robot;
                this.io = io;
        }
        public void run(){
                        System.out.println("Robot info thread running...");
                        System.out.println("-------------------------------------");

                while(RobotApplication.startInfoThread){
```

```
                        int Transform_Robot_X =
                        ↪  (int)robot.getCurrentCartesianPosition(robot.getFlange()).getX()+1000;
                        int Transform_Robot_Y =
                        ↪  (int)robot.getCurrentCartesianPosition(robot.getFlange()).getY()+1000;
                        int Transform_Robot_Z =
                        ↪  (int)robot.getCurrentCartesianPosition(robot.getFlange()).getZ()+1000;

                        io.setKUKA_TaskInfo_Coordinate_X(Transform_Robot_X);
                        io.setKUKA_TaskInfo_Coordinate_Y(Transform_Robot_Y);
                        io.setKUKA_TaskInfo_Coordinate_Z(Transform_Robot_Z);

                        try{
                                Thread.sleep(300);
                        }
                        catch(Exception err){

                        }
                }
        }
}
```

RobotTestUtil

```
package application;

import static com.kuka.roboticsAPI.motionModel.BasicMotions.ptp;

import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.roboticsAPI.motionModel.CartesianPTP;

public final class RobotTestUtil {


        public static void printout(){
                System.out.println("hej");
        }
        public static void PickupBoxCornering(LBR robot){
                Frame PickupBoxCorner1 = new Frame(312,-401,335,0,0,3.13);
                Frame PickupBoxCorner2 = new Frame(312,-720,335,0,0,3.13);
                Frame PickupBoxCorner3 = new Frame(-312,-720,335,0,0,3.13);
                Frame PickupBoxCorner4 = new Frame(-312,-401,335,0,0,3.13);

                CartesianPTP PickupBoxCorner1Prepare =
                ↪  ptp(PickupBoxCorner1).setJointVelocityRel(0.25);
                CartesianPTP PickupBoxCorner2Prepare =
                ↪  ptp(PickupBoxCorner2).setJointVelocityRel(0.25);
                CartesianPTP PickupBoxCorner3Prepare =
                ↪  ptp(PickupBoxCorner3).setJointVelocityRel(0.25);
                CartesianPTP PickupBoxCorner4Prepare =
                ↪  ptp(PickupBoxCorner4).setJointVelocityRel(0.25);
                robot.move(PickupBoxCorner1Prepare);
                robot.move(PickupBoxCorner2Prepare);
                robot.move(PickupBoxCorner3Prepare);
                robot.move(PickupBoxCorner4Prepare);
        }

}
```

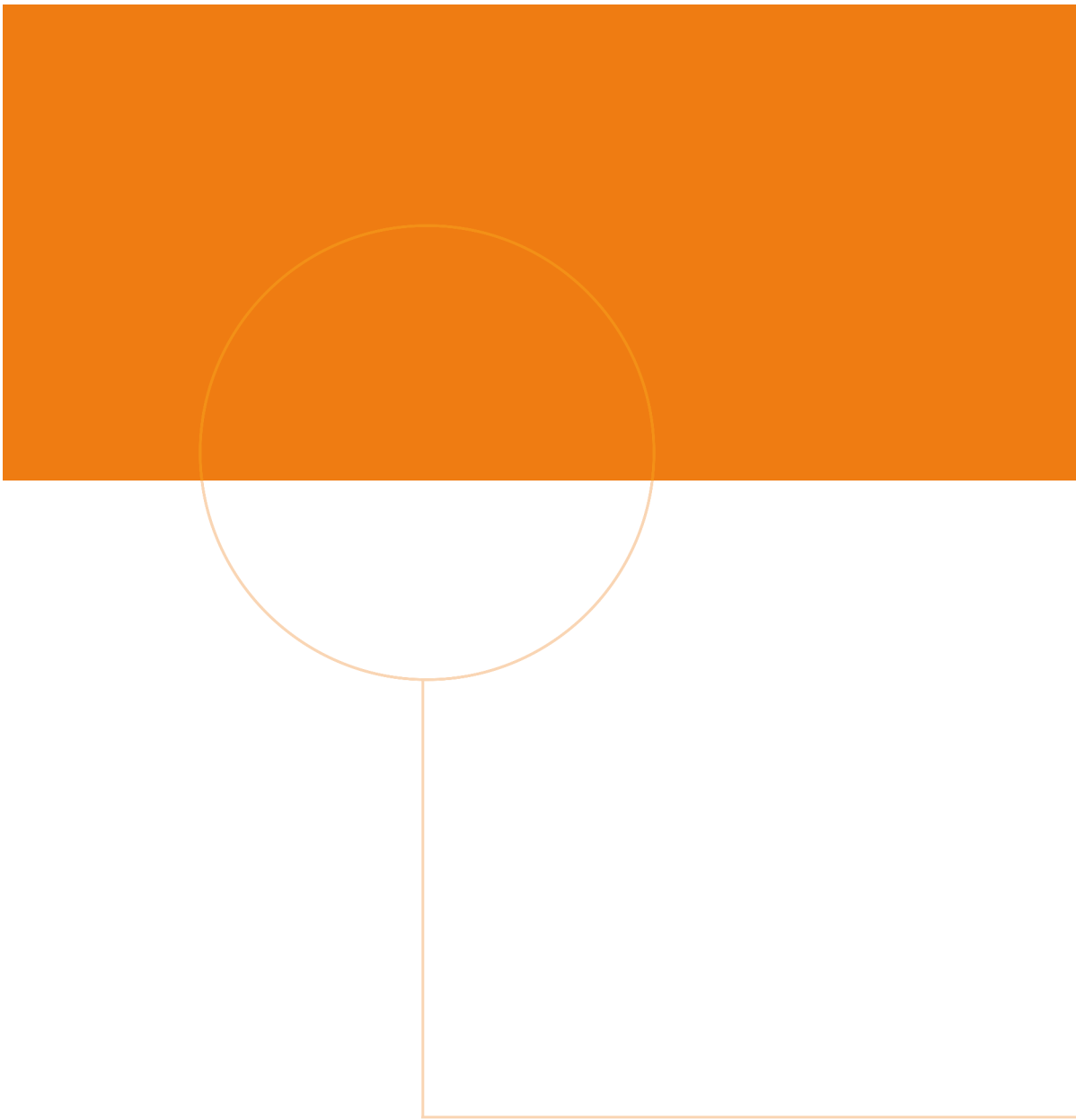
RobotUtil

```
package application;

import static com.kuka.roboticsAPI.motionModel.BasicMotions.lin;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.ptp;

import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.roboticsAPI.motionModel.CartesianPTP;
import com.kuka.roboticsAPI.motionModel.PTP;

public class RobotUtil {

	public static void PickupPrepare(LBR robot){
		PTP PickupPrepareJointPosition = ptp(0, 0.523, 0, -1.57,0, 1.047,
		↪  0).setJointVelocityRel(0.2);
		robot.move(PickupPrepareJointPosition);
		PTP PickupPrepareJointPosition2 = ptp(-1.57, 0.523, 0, -1.57,0,
		↪  1.047, 0).setJointVelocityRel(0.25);
		robot.move(PickupPrepareJointPosition2);
	}
	public static void PickupPrepareObjectPosition(LBR robot, double x,
	↪  double y, double z, double a){
		Frame AngledFrameOverObject = new
		↪  Frame(x,y,robot.getFlange().getZ(),robot.getCurrentJointPosition().get(6)+1.57
		robot.move(lin(AngledFrameOverObject).setJointVelocityRel(0.1));

		double J1= robot.getCurrentJointPosition().get(0);
		double J2= robot.getCurrentJointPosition().get(1);
		double J3= robot.getCurrentJointPosition().get(2);
		double J4= robot.getCurrentJointPosition().get(3);
		double J5= robot.getCurrentJointPosition().get(4);
		double J6= robot.getCurrentJointPosition().get(5);
		double Joint7Offset= robot.getCurrentJointPosition().get(6);

		double angleFix = 0;
		if (a + Joint7Offset > 3.05){
			angleFix = -((Math.PI - Joint7Offset) + (Math.PI - a));
				if (-angleFix > a + Joint7Offset){
					angleFix = a+Joint7Offset;
				}
		}
		else if (a + Joint7Offset < -3.05){
			angleFix = ((-Math.PI - Joint7Offset) - (Math.PI + a));
				if (-angleFix < a + Joint7Offset){

					angleFix = a + Joint7Offset;
				}
		}
		else {
			angleFix = a+Joint7Offset;
		}


		if (angleFix > 3.05) {
			angleFix = 3.05;
		}
```

```
        else if (angleFix < -3.05) {
                angleFix = -3.05;
        }

        robot.move(ptp(J1,J2,J3,J4,J5,J6,angleFix).setJointVelocityRel(0.1));
        Frame OverObject =
        ↪  robot.getCurrentCartesianPosition(robot.getFlange());
        System.out.println("Over0: " + OverObject);

        Frame DownToObject = OverObject.copyWithRedundancy();
        DownToObject.setZ(z);
        robot.move(lin(DownToObject).setJointVelocityRel(0.10));
}

public static void afterPickup(LBR robot){

        Frame afterPickupFrame =
        ↪  robot.getCurrentCartesianPosition(robot.getFlange());

        Frame MoveUpwardsFrame = afterPickupFrame.copyWithRedundancy();
        MoveUpwardsFrame.setZ(385);
        robot.move(lin(MoveUpwardsFrame).setJointVelocityRel(0.10));


        double J1= robot.getCurrentJointPosition().get(0);
        double J2= robot.getCurrentJointPosition().get(1);
        double J3= robot.getCurrentJointPosition().get(2);
        double J4= robot.getCurrentJointPosition().get(3);
        double J5= robot.getCurrentJointPosition().get(4);
        double J6= robot.getCurrentJointPosition().get(5);


        PTP rotateFlange =
        ↪  ptp(J1,J2,J3,J4,J5,J6,0).setJointVelocityRel(0.15);
        robot.move(rotateFlange);

}
public static void initializeRobotPosition(LBR robot){
        Frame startFrame =
        ↪  robot.getCurrentCartesianPosition(robot.getFlange());
        System.out.println("Robot start position: " + startFrame);
        Frame ColisionStartFrame = startFrame.copyWithRedundancy();

        if (startFrame.getY() > 400){ //If robot is initialized from Drop
        ↪  position
            ColisionStartFrame.setY(500);
            robot.move(lin(ColisionStartFrame).setJointVelocityRel(0.1));
        }
        if (startFrame.getY() < -400){ //If robot is initialized from
        ↪  pickup position
            ColisionStartFrame.setZ(350);
            robot.move(lin(ColisionStartFrame).setJointVelocityRel(0.1));
            PTP HOLDPrepareJointPosition =
            ↪  ptp(0,0.715,0,-1.17,0,0,0).setJointVelocityRel(0.2);
            robot.move(HOLDPrepareJointPosition);
        }
        PTP ptpToMechanicalZeroPosition = ptp(0,0,0,0,0,0,0);
        ptpToMechanicalZeroPosition.setJointVelocityRel(0.25);
```

```
        robot.move(ptpToMechanicalZeroPosition);
}

public static void DropPrepare(LBR robot){
        Frame DropPrepFrame = new Frame(-14,470,824,0,1.553,-1.56);

        CartesianPTP DropPrepare =
        ↪  ptp(DropPrepFrame).setJointVelocityRel(0.25);
        robot.move(DropPrepare);
}

public static void DropScrewdriverPrepare(LBR robot){

        Frame ScrewdriverDropPrepFrame = new
        ↪  Frame(12.9,550,840,0,1.553,-1.56);
        Frame ScrewdriverDropPrepFrame2 = new
        ↪  Frame(13.5,587.3,830,0,1.553,-1.56);
        Frame ScrewdriverDropPrepFrame3 = new
        ↪  Frame(13.5,587.3,832,0,1.553,-1.56);

        CartesianPTP ScrewdriverDropPrepare =
        ↪  ptp(ScrewdriverDropPrepFrame).setJointVelocityRel(0.1);
        CartesianPTP ScrewdriverDropPrepare2 =
        ↪  ptp(ScrewdriverDropPrepFrame2).setJointVelocityRel(0.10);
        CartesianPTP ScrewdriverDropPrepare3 =
        ↪  ptp(ScrewdriverDropPrepFrame3).setJointVelocityRel(0.10);
        robot.move(ScrewdriverDropPrepare);
        robot.move(ScrewdriverDropPrepare2);
}

public static void DropHammerPrepare(LBR robot){

        Frame HammerDropPrepFrame = new Frame(161,520,857,0,1.553,-1.56);
        Frame HammerDropPrepFrame2 = new
        ↪  Frame(161,579,857,0,1.553,-1.56);
        Frame HammerDropPrepFrame3 = new
        ↪  Frame(161,579,837,0,1.553,-1.56);
        CartesianPTP HammerDropPrepare =
        ↪  ptp(HammerDropPrepFrame).setJointVelocityRel(0.25);
        CartesianPTP HammerDropPrepare2 =
        ↪  ptp(HammerDropPrepFrame2).setJointVelocityRel(0.1);
        CartesianPTP HammerDropPrepare3 =
        ↪  ptp(HammerDropPrepFrame3).setJointVelocityRel(0.1);
        robot.move(HammerDropPrepare);
        robot.move(HammerDropPrepare2);
        robot.move(HammerDropPrepare3);

}
public static void DropPliersPrepare(LBR robot){

        Frame PliersDropPrepFrame = new
        ↪  Frame(299.5,564,883.2,0,1.553,-1.56);
        Frame PliersDropPrepFrame2 = new
        ↪  Frame(299.5,603.5,883.2,0,1.553,-1.56);

        CartesianPTP PliersDropPrepare =
        ↪  ptp(PliersDropPrepFrame).setJointVelocityRel(0.1);
```

```
            CartesianPTP PliersDropPrepare2 =
            ↪  ptp(PliersDropPrepFrame2).setJointVelocityRel(0.1);
            robot.move(PliersDropPrepare);
            robot.move(PliersDropPrepare2);


    }

    public static void AfterDropPosition(LBR robot){

            Frame AfterDropFrame =
            ↪  robot.getCurrentCartesianPosition(robot.getFlange());
            System.out.println("Robot start position: " + AfterDropFrame);

            Frame ColisionAvoidFrame = AfterDropFrame.copyWithRedundancy();
            ColisionAvoidFrame.setY(500);
            robot.move(lin(ColisionAvoidFrame).setJointVelocityRel(0.1));



    }


}
```



NTNU

Norwegian University of
Science and Technology