Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Mustafe Kahin

# Robot Telemanipulation for Remote Maintenance

Master's thesis in Mechanical engineering
Supervisor: Gunleiv Skofteland
Co-supervisor: Christian Holden
June 2022

NTNU
Norwegian University of
Science and Technology

Mustafe Kahin

# Robot Telemanipulation for Remote Maintenance

Master's thesis in Mechanical engineering
Supervisor: Gunleiv Skofteland
Co-supervisor: Christian Holden
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

# Acknowledgements

First and foremost, I would like to thank my supervisor Gunleiv Skofteland, and co-supervisor Christian Holden, for guidance throughout this master's thesis. The assistance provided played a significant role, and is greatly appreciated. I am also thankful for Adam Leon Kleppe, for providing great guidance in anything from debugging and setup of hardware in the laboratory. I would also like to thank my lab partner Irfan Suvalija, for interesting discussions and a good collaboration. Last but not least, I would like to thank my friends and family for great support and keeping me motivated during this master's thesis.

# Abstract

This master's thesis investigates the possibility of conducting a filter change through the means of teleoperation. Teleoperation is often referred to as remote control of a robot and is mostly oriented towards inspection, repair, and maintenance in industry-related applications.

The task investigated in this thesis relates to changing a filter unit located at a gas dehydration skid on an oil and gas producing platform. The filter's purpose is to clean Triethylene glycol(TEG), used in the dehydration process. The filter change is normally performed manually. In this thesis, the teleoperation task is performed at the Norwegian Manufacturing Research Laboratory(MANULAB) at NTNU in Trondheim. The assigned robot for the task has six degrees of freedom and is named KUKA KR-16.

The teleoperation system allowed for end-effector control through MoveIts move_group node, and was successfully configured in ROS, but complications with connection to the joystick made it unsuccessful in the laboratory. Direct control through the robots teach pendant was used in the laboratory to get the exact positions for the filter change, and the whole cycle was implemented with a Python script. The implemented script, together with a computer mouse, allowed for command input of the complete filter change and teleoperation of the robot in RViz.

The tests showed that the filter change requires high precision when inserting and extracting the filter. This led to using the developed python code to pick up the old filter and place the new filter inside the container, while movements that did not require any precision were teleoperated with a computer mouse. Teleoperation with the joystick in ROS showed to be prone to delay when controlled in RViz. This delay led to a "move-and-wait" strategy in order to perform the wanted end-effector movements.

# Sammendrag

Denne masteroppgaven undersøker muligheten for å utføre et filterskifte ved bruk av teleoperasjon. Teleoperasjon blir ofte referert til som fjernstyring av en robot, og er for det meste orientert mot inspeksjon, reperasjon og vedlikehold i industrirelaterte applikasjoner.

Oppgaven som undersøkes i denne masteroppgaven er knyttet til å bytte en filterenhet plassert ved en gassdehydreringsenhet på en olje-og gassproduserende plattform. Formålet med filteret er å rense trietylenglykol(TEG), som brukes i dehydreringsprosessen. Filterbyttet utføres normalt manuelt. I denne oppgaven utføres teleoperasjonsoppgaven på MANULAB ved NTNU Trondheim. Den tildelte roboten for oppgaven har seks frihetsgrader og heter KUKA KR-16.

Teleoperasjonssystemet tillot endeffektor kontroll gjennom MoveIts move_group node, og ble vellykket konfigurert i ROS, men komplikasjoner med tilkobling av joysticken gjorde det mislykket i laboratoriet. Direkte kontroll gjennom robotens håndkontroll ble brukt i laboratoriet for å få de nøyaktige posisjonene for filterbyttet, og hele sykulusen ble implementert med et Python-skript. Det implementerte skriptet sammen med en datamus tillot kommandoinngang for hele filterbyttet, samt teleoperasjon av roboten i RViz.

Testene som ble utført viste at filterbyttet krever høy presisjon ved innsetting og uttak av filteret. Dette førte til å bruke den utviklede pythonkoden for å plukke opp det gamle filteret, og plassere det nye filteret inne i beholderen, mens bevegelser som ikke krevde noen presisjon ble teleoperert med en datamus. Teleoperasjon med joysticken i ROS viste seg å være utsatt for forsinkelser, når den ble kontrollert i RViz. Denne forsinkelsen førte til en “flytt-og-vent” strategi for å utføre de ønskede endeffektor bevegelsene.

# Contents

**Acknowledgements** i

**Abstract** ii

**Sammendrag** iii

**List of Figures** vii

**List of Tables** ix

**1. Introduction** 1
- 1.1. Problem objectives . . . 2
- 1.2. Contributions . . . 3
  - 1.2.1. Pick and Place . . . 3
  - 1.2.2. Simulation . . . 4
  - 1.2.3. Remote Maintenance . . . 4
- 1.3. Outline of the Thesis . . . 5

**2. Theoretical Background** 6
- 2.1. Teleoperation . . . 6
  - 2.1.1. Time delay . . . 7
- 2.2. Use cases of teleoperation . . . 8
  - 2.2.1. Hazardous operations . . . 8
  - 2.2.2. Teleoperation in Space . . . 9
  - 2.2.3. Telemedicine . . . 9
  - 2.2.4. CERN . . . 9
- 2.3. Control architectures . . . 10
  - 2.3.1. Direct control . . . 10
  - 2.3.2. Supervised Control . . . 11
  - 2.3.3. Shared control . . . 11
- 2.4. Robot Kinematics . . . 12
  - 2.4.1. Pose of rigid body . . . 12
  - 2.4.2. Rotation Matrix . . . 13

2.4.3. Homogeneous transformation Matrix . . . 14
2.4.4. Workspace . . . 15
2.4.5. Forward Kinematics . . . 15
2.4.6. Denavit-Hartenberg . . . 16
2.4.7. Inverse Kinematics . . . 17
2.4.8. Operational space & Joint space . . . 18
2.4.9. Trajectory planning . . . 19
2.4.10. Path & Trajectory . . . 19
2.4.11. Joint space trajectories . . . 20
2.4.12. Trajectories for Point to Point . . . 20
2.4.13. Sequence of points . . . 22

**3. Programming Tools** **23**
3.1. ROS . . . 23
3.1.1. ROS Computation Graph . . . 24
3.1.2. ROS control . . . 26
3.1.3. MoveIt . . . 26
3.1.4. RViz . . . 27
3.1.5. Gazebo . . . 27
3.2. URDF . . . 28
3.2.1. Xacro . . . 28

**4. Method** **29**
4.1. Software Implementation . . . 29
4.1.1. MoveIt Configuration . . . 29
4.1.2. Kinematics Configuration . . . 31
4.1.3. Simulation . . . 31
4.1.4. Nodes . . . 33
4.1.5. Summary . . . 35
4.2. Hardware setup . . . 37
4.2.1. Robot Hardware . . . 37
4.2.2. Gripper . . . 38
4.2.3. Filter Assembly . . . 39
4.3. Pick and place . . . 40
4.4. Testing . . . 42

**5. Results** **44**
5.1. Teleoperation . . . 44
5.2. Trajectories . . . 45
5.3. Plots . . . 50

**6. Discussion** **53**
6.1. Teleoperation . . . 53
6.2. Trajectories . . . 54
6.3. Plots . . . 55

**7. Conclusions and Future Work** **57**
7.1. Conclusion . . . 57
7.2. Future work . . . 58

**A. Digital Attachments** **65**
A.1. Running the teleoperation system in ROS . . . 65
A.1.1. Teleoperation with a joystick . . . 66
A.1.2. Setup in the laboratory . . . 67
A.2. Controllers . . . 67
A.3. Denavit Hartenberg implementation . . . 69
A.4. Hierarchical graph . . . 70

**B. Hardware** **72**
B.1. Gripper Machine Drawings . . . 72

# List of Figures


1.1. Filter Assembly . . . . . 2
1.2. Filter . . . . . 2
1.3. Taurob-Inspector . . . . . 5

2.1. Bilateral teleoperation [17] . . . . . 7
2.2. Telemax . . . . . 10
2.3. Control architectures [50] . . . . . 12
2.4. Position & Orientation of a rigid body [5] . . . . . 13
2.5. Workspace of KUKA KR-16 [12] . . . . . 15
2.6. Relationship between Forward and Inverse Kinematics [23] . . . . . 17

3.1. ROS Computational Graph Layer . . . . . 24
3.2. Publisher&Subscriber model . . . . . 25
3.3. ROS Control . . . . . 26
3.4. MoveIt . . . . . 27

4.1. Folder layout and KUKA KR-16 Visualized in RViz . . . . . 31
4.2. Pid tuning . . . . . 33
4.3. Control architecture . . . . . 36
4.4. Physical installation hardware . . . . . 37
4.5. The original gripper & Remodelled gripper mounted . . . . . 39
4.6. Complete filter assembly . . . . . 40
4.7. Laboratory workstation . . . . . 41

5.1. Joystick teleoperation . . . . . 44
5.2. Communication of nodes . . . . . 45
5.3. Trajectories visualized in RViz & robot executing in the laboratory 48
5.4. Sequence of points visualized . . . . . 49
5.5. P8 to P9 position plot(directly) . . . . . 50
5.6. P8 to P9 velocity plot(directly) . . . . . 50
5.7. P8 to P9 acceleration plot(directly) . . . . . 51
5.8. Position of joints from P8 to P9(sequence) . . . . . 51
5.9. P8 to P9 velocity plot(sequence) . . . . . 51

5.10. P8 to P9 acceleration plot(sequence) . . . . . 52

6.1. Failure due to precision error . . . . . 55

A.1. Hierarchical graph of KUKA KR-16 . . . . . 71

B.1. Gripper Machine Drawing(Opened) . . . . . 72
B.2. Gripper Machine Drawing(Closed) . . . . . 73

# List of Tables

2.1. Workspace dimensions [12] . . . . . 15

4.1. Mappings from controller to robot [30] . . . . . 35
4.2. Robot specifications [43] . . . . . 38
4.3. Filter& Filter container dimensions . . . . . 40
4.4. Joint angles for pick-and-place . . . . . 43
4.5. Corresponding Cartesian points . . . . . 43

5.1. Sequence of new points(joint angles) . . . . . 49
5.2. Sequence of new points(Cartesian) . . . . . 49

A.1. KR16 URDF in table format . . . . . 70

# Chapter 1.

# Introduction

Offshore plants are faced with many challenges regarding their exposure to harsh environments as they are installed in distant locations from the shore. The installations can be subject to toxic and corrosive atmospheres, which can pose a challenge to the human operators. Production of fossil fuels in these conditions has made the oil and gas industry one of the most heavily covered by safety, international, and organizational standards [19]. The O&G industry focuses on continuously reducing personnel risk and improving health, safety, and the environment(HSE). Addressing the stated issues requires new solutions to the conventional operation and maintenance of the installations.

Integrating robotic teleoperation is a possible solution for more efficiency and increased safety for the operator. Teleoperation means "Operating at a distance" and allows the human operator to perform tasks from a safe environment by the use of a human-machine interface [28]. The capabilities of the human operator are extended with this interface, making it a field of interest since it permits the interaction of environments that can be difficult to access or hazardous environments that pose a challenge to the human operator [28]. The operation of robots from a distance has the potential to solve the stated issues by using the robots on dull, dirty, distant, and dangerous work tasks. The integration of robotics in the industry is not something new, as there exist solutions for both inspection and maintenance tasks using Remotely operated vehicles(ROVs). As for this thesis, the use of a stationary robot is investigated.

The focus of this thesis is related to remotely operating a robot in order to conduct a filter change. The filter is located at a Tri-Ethylene Glycol(TEG) skid, whose primary purpose is to remove contaminants from the TEG. Today, the filter change is done manually by personnel, but by using a robot, the maintenance will be more automated and reduce the risk for personnel when it comes to hazardous work. Using robots for such tasks will also improve efficiency, as the robot stays at a

given location, implying that relocation of personnel is needless. Since the filter is remotely located, Equinor is investigating possible solutions to access the filter for maintenance from a distance.

A CAD figure of both the filter and its container can be seen in Figure 1.2 and Figure 1.1 respectively. The filter container can be divided into two parts, where the top section can be unscrewed to fit the filter inside, which means that the robot used must be able to conduct this motion. The assigned robot for this thesis is the KUKA KR-16, which has six degrees of freedom(DOF), and a spherical wrist, making it possible to solve the task at hand as it can replicate human motion. As for the actual task of exchanging the TEG filter on an unmanned platform, a mobile robot with a manipulator arm will be used e.g. the Taurob [20]. A demonstration of the filter change will be conducted in the laboratory MANULAB at NTNU, Gløshaugen.

The following objective in collaboration with NTNU Department of Mechanical and Industrial Engineering is given:

*Robot Telemanipulation for Remote Maintenance.*

Since teleoperation is a wide subject within telerobotics, a specification of the objectives in this thesis are presented in the next section.

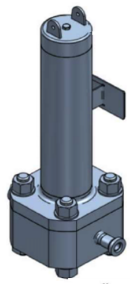


**Figure 1.1.:** Filter Assembly

**Figure 1.2.:** Filter

## 1.1. Problem objectives

The problem of remote operations of a robot is a subject that has been widely researched and covers a great specter of control architectures. This thesis covers the notion of investigating how the filter change can be conducted, which has led to the different parts presented below.

Teleoperation in offshore installations differs from other applications as the envi-

ronment is cluttered, and failures can damage both equipment and the environment. This is why a simulation of the robot's behavior to verify the interaction between it and its environment is highly relevant for this thesis. The simulation will be conducted using the 3D simulation program Gazebo, which can simulate realistic physics and has additional features such as programmable and graphical interfaces. The software setup will be conducted using the Robot Operating System(ROS) and tested at the laboratory, as mentioned in the previous section.

The main problem can be divided into smaller sections. In the first part of the thesis, a theoretical background study of teleoperation and its main properties is conducted. Further, the control architecture for remote operation of KUKA KR-16 is investigated. The objectives that are focused on in this project are as follows:

- Create a software setup of the KUKA KR-16
- Create a teleoperation control architecture
- Perform a pick and place of the filter in both simulation and in the laboratory

The filter change will be attempted with a PlayStation 4 controller, commonly referred to as *DualShock 4* since a haptic joystick is not accessible for this thesis. Both the filter and filter container is deployed to the laboratory by Equinor, and some changes regarding its structure have been applied, which is further explained in section 4.2.

## 1.2. Contributions

This section presents some of the main contributions to this thesis.

### 1.2.1. Pick and Place

Remotely operating a robot for a pick-and-place task is presented by Love et al. [25] and draws a relation to the investigated subject of this thesis. The authors investigated robot control algorithm's used for teleoperation with a long reach manipulator, where the task consisted of remotely moving a payload of 6.8kg between different stations. The pick-and-place cycle was initiated by having the robot manipulator at its home position and the payload in a predefined position. Following, the robot manipulator was moved to the object's position, and picked up before it was relocated to a new position. The paper's authors concluded that some form of trajectory or command filtering during task execution would increase teleoperated pick-and-place tasks. Inspiration is drawn from the paper when conducting the experiments in this thesis.

### 1.2.2. Simulation

Another essential part of teleoperation is the "situational awareness" provided to the operator. A simulation of the performed tasks increases this aspect and can be applied with the use of different software programs. Deng et.al [11] presented a procedure on how to perform a simulation of a mobile manipulation task. The authors used the *Robot Operating System*(ROS) in addition to the motion planning software MoveIt. An architecture that included the complete configuration of the robot, ranging from kinematics to both control and visualization, was demonstrated before implementing it on a mobile robot. The setup of this paper can be related to the same architecture in this thesis, further presented in section 4.1.

### 1.2.3. Remote Maintenance

The three main activities in a maintenance task include: disturbance handling, inspection, and planned maintenance [39]. The objective of this thesis falls in the last-mentioned category, as the filter change follows a pre-determined schedule. Teleoperation of robots for maintenance tasks referring to the oil and gas industry differs from other applications as the environment is harsh and less predictable, and considering the distance between the onshore operation center and the offshore installation.

There already exist solutions for both inspection and maintenance in the oil and gas industry, as mentioned previously in the introduction 1. The most common are Remotely Operated Vehicles(ROVs) for underwater operations and pipe inspections. ROVs are suitable for harsh environments and are usually controlled with a joystick along with video monitors [52] to perform tasks such as; repairing pipelines, well heads, communications cables, and platforms [39]. ROVs are also used for onsite inspections on platforms, such as the "Taurob-Inspector", which is the intended robot for the actual task of changing the filter. The ROV has a robotic arm with 5 degrees of freedom, and is equipped with sensors that gather data, detect gas leaks, and take high-definition photos [20]. The ROV can be seen below in Figure 1.3. Other use cases of teleoperation are further presented in the following chapter.


**Figure 1.3.:** Taurob Inspector[20]

## 1.3. Outline of the Thesis

This thesis will first present the theoretical background of teleoperation and robot kinematics in Chapter 2 before presenting the programming tools essential for developing the teleoperation control architecture in Chapter 3. The theory is mainly gathered from the specialization project [21] written in the fall semester of 2021, prior to this thesis. Chapter 4 presents the methodology part of the thesis and includes both implementation in ROS and the laboratory, as well as the developed control architecture. Chapter 5 and Chapter 6 present the results from the tests conducted in the laboratory and discussions, respectively. Chapter 7 rounds off the thesis with conclusions and suggestions for the project's future work.

# Chapter 2.

# Theoretical Background

This chapter presents the theoretical background covering the aspects of telerobotics and robot kinematics necessary to develop a teleoperation control architecture. The theory is mostly based on previous work done in the specialization project prior to this thesis [21], where some sections are gathered in its entirety.

## 2.1. Teleoperation

The term teleoperation derives from the Greek word tele(remote) and the Latin word operatio(operation, something done), which means the operation of tasks from some distance away [2]. This interface allows the human operator to perform tasks from a safe environment through a human-machine interface, and it is commonly associated with controlling a robot from a place far from the robot's location. The capabilities of the human operator are extended with this interface, which makes it a field of interest since it permits the interaction of environments that can be difficult to access or hazardous environments that pose a challenge to the human operator [28]. The application of teleoperation systems is wide and can be found in many different industries today.

A teleoperation system consists of a *master*(local) device and a *slave*(remote) device/robot, which are connected by a *communication channel*. The extension of the operator's capability is achieved by this system, and it can be divided into two main processes [41]:

1. Interaction between the operator and the master device
2. Interaction between the slave device and the environment

Figure 2.1 depicts a teleoperation system. A human operator is interacting with a master device which is located on the local side of the system. The master device can take different forms, such as a mouse, a joystick, a robotic arm, or a kinematic

replica of the slave robot. The desired commands are sent through this device to the slave robot, which in turn interacts with the remote environment.

Information is exchanged between the local and remote sites through the *communication channel*, such as a force or position signal and visual data. If the slave device possesses force sensors such that force feedback can be transmitted to the master device when a task is being performed, the system is said to be a bilateral teleoperation system [17]. When no force feedback is present, the system is referred to as a unilateral teleoperation system.

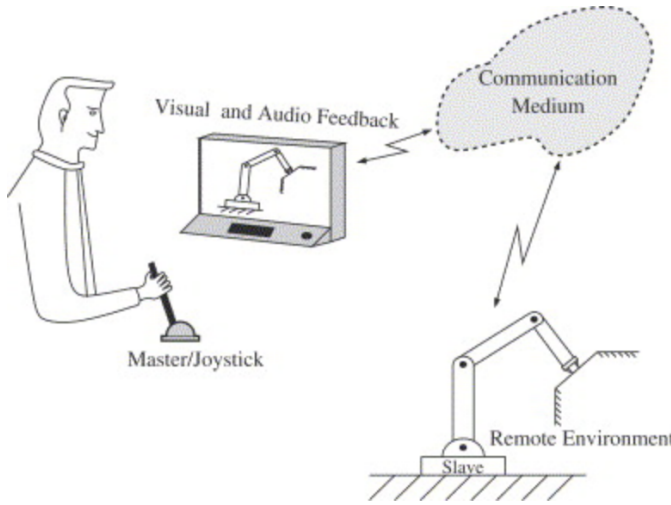



**Figure 2.1.:** Bilateral teleoperation [17]

Since teleoperation is usually used in the context of operating in an unstructured environment, the main goal would be to have the control system in a "steady state", *i.e*, when the slave velocity and force is similar to the masters. This type of control system is defined as *transparent* [28] and is often referred to as *Telepresence*, and is still far from achieved in most practical applications according to [4].

### 2.1.1. Time delay

A critical part of telerobotics is the issue of time delay that occurs because of the distance between the operator and the robot or limitations in the communication infrastructure. It can cause severe destabilizing consequences and poses a challenge when it comes to control. For bilateral systems where the objective is to let the operator feel the contact force of the remote robot, two basic approaches to solve the unstable consequences of time delay are available, which are based on

*passivity theory*, which states that a system is stable if it always has to dissipate and never increment its total energy, and *control theory* where a linear model of each element is proposed and block diagrams are constructed [40]. Research addressing the issue of stability for bilateral teleoperation systems has shown that passivity-based methods are widely accepted because of their robustness and ease of applicability [18]. Implementation of bilateral teleoperation controllers on both sides of the communication channel has also been proposed as this reduces the effects of packet delay, loss, and jitter [18]. This section was gathered from [21] in its entirety.

## 2.2. Use cases of teleoperation

Since its beginning of research in the nuclear fields, teleoperation systems have been highly motivated by securing human safety, which in turn has made its usage expand in many different applications. Teleoperation systems are used in many different fields where the environment is dangerous and difficult for humans to access, such as space exploration and underwater applications. Its use cases also extend to manipulation of smaller objects in situations where the area is limited such as in telerobotic surgery. This section presents early history of teleoperation and some of its main use cases.

### 2.2.1. Hazardous operations

The separation between the human operator and the manipulated environment is one of the main benefits of a teleoperation system, making it very desirable to employ in hazardous environments. The accomplishment of tasks in these environments depends on the nature and magnitude of the hazards, which may present themselves in the form of radiation, toxic contamination, falling objects, or potential explosions [55]. If human exposure to these hazards can cause life-threatening or long-term health consequences, some forms of remote operations that separate humans from the environment are highly relevant. An example of such operations is within the nuclear industry, where the roots of teleoperation can be traced back to research by Raymond C. Goertz in the 1940s and 1950s [55].

Raymond C.Goertz developed the first modern-master slave teleoperated system at Argonne National Laboratory [51]. The master(local) device represents the operators' environment and the slave(remote) device represents the operator at the remote environment [41]. The system was intended for nuclear operations, where the first system consisted of an array of on-off switches that activated motors and various axes [14]. According to Goertz, the electrical system caused the manip-

ulators to feel slow and awkward to operate, which led to Goertz building pairs of mechanically linked robots [14, 15, 34]. The new system allowed the operator to use natural hand motions where the forces and vibrations were transmitted through the structure it was connected to [34].

The research and advances made in teleoperation were, in later years, used in other areas as nuclear power activities began to decline in the 1980s and 1900s. Nuclear remote operations experience with teleoperation has today influenced its usage in other applications such as in space, medical systems, and other hazardous environments [34].

### 2.2.2. Teleoperation in Space

The utilization of teleoperation in space is highly motivated by the issue of non-breathable air, the risks and difficulties of sending a human to space, and the expenses that follow. Numerous teleoperation systems are used in space missions, such as *Canadar*, a remote manipulator system used to capture and redeploy defective satellites. The issue of time delay is prominent for teleoperation systems used for space applications. This is due to the distance between the information sent from earth to the receiving station in space [2].

### 2.2.3. Telemedicine

In the mid-'80s, the first robotic systems were introduced in medicine and make an impact today in various medical disciplines such as neurosurgery, surgery, and orthopedic surgery [3, 6, 56, 16] [1]. The usage of teleoperated robotic systems in medicine has the possibility to provide treatments across short or long distances, which eliminates the need of the physical presence of both the patient and physician at the same location.

The first successful transatlantic telesurgery was demonstrated by Computer Motion[1] in 2001 [34]. The operation was named "Lindberg", where a laparoscopic gall bladder operation was carried out by surgeons located in New York(USA) on a patient located in Strasbourg(France). The operation was performed using a Zeus robotic system, and it did not include force feedback which meant that the surgeons had to rely on visual feedback.

### 2.2.4. CERN

The European Council for Nuclear Research(CERN) is one of the largest centers for scientific research and is located in Geneva [8]. The primary research at CERN

[1]Computer Motion is a high-tech medical device company

is focused on particle physics, and they provide particle accelerators which have led to many international collaborations.

Teleoperated robots are used at CERN since some areas are prone to radiation contamination. Instead of waiting for long periods of time for the radiation to dissipate, humans are replaced by their counterparts, i.e., robots. This ensures the safety of personnel and improves the availability of CERN's accelerators, such as the Large Hadron Collider(LHC). One of the robots used at CERN is called "Telemax" and is mainly used for inspection and maintenance purposes. The robot is equipped with a seven degrees of freedom manipulator and has great mobility provided by tracked wheels. It is also equipped with cameras that ensure visual feedback when it is teleoperated [49]. The ROV can be seen in Figure 2.2 below.

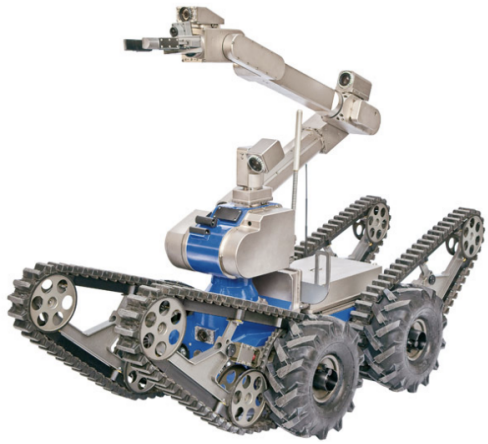


**Figure 2.2.:** Telemax [49]

## 2.3. Control architectures

The application of teleoperated robots has been concentrated on several operations modes, facilitated by the increasing development of sensor technologies, control, and computer technologies. The main operation modes are direct control, supervised control, and shared control [26]. The following sections are gathered from the specialization project [21] in its entirety.

### 2.3.1. Direct control

Direct control implies that the operator controls the robot's motion directly, which means the system has no intelligence or autonomy. Some systems involve direct

control where the operator commands are done via a joystick or a kinematic replica of the slave-robot. The joystick can be considered a robot itself(master-robot), and by providing direct control, difficulties in creating local autonomy in the system can be avoided [35, 21].

### 2.3.2. Supervised Control

With supervised control, the robots can execute tasks at the remote environment according to a predetermined program, where the operators only supervise the task execution [26]. This allows for more autonomy and intelligence for the robot system. Supervised control can deal with intermittent input, which also makes it suitable for commanding multiple robots [18].

An implementation of supervised control is "telesensor programming", which was developed for space applications. Telesensor programming allows the operator to test and adjust tasks while interacting with a robot simulation and remote environment. To implement such a control system's functionality, there exists a need for necessary tools. For the operator to debug the job execution, he/she needs to be provided with a simulation of the robot which includes sensory perception, and an efficient interface to set up task descriptions. In this way, the operator can configure the control parameters for the task, and decide the sensors and algorithms which will be used to execute the task. The control algorithm has advantages for significant time delays in the system, making it applicable for undersea and space applications. Since the operators are due to delayed feedback, a predictive simulation gives the advantage to teleoperating the robot [35, 21].

### 2.3.3. Shared control

Shared control systems combine the characteristics of direct and supervised control in the sense that the operator shares the control of the slave robot with an autonomous controller to achieve a common goal [50]. In this way, some degree of automated help is available to the human operator. Since some workload is transferred to the robot, complexities such as time/communication delay can be overcome by specifying gross path commands to the slave robot, which again can fine-tune the information, such as correcting motion commands or regulating sub tasks [50, 21].

.

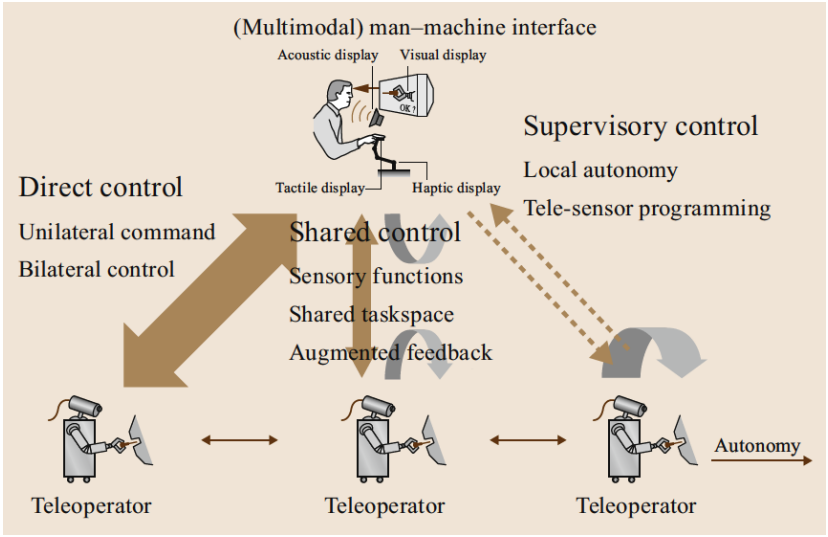



**Figure 2.3.:** Control architectures [50]

## 2.4. Robot Kinematics

The following sections present essential robot kinematics for the filter change. The kinematics are based on theory from the books *Robotics – Modelling, Planning and Control* by Siciliano et.al [5], *Robot Dynamics and Control* by Spong et.al [53], and *Robotics -Modelling, Planning and Control* by Lynch et.al [22]. The theory is gathered from [21] and modified, apart from Section 2.4.7 which is in its entirety.

### 2.4.1. Pose of rigid body

In order to manipulate an object in space, it is necessary to mathematically define the configuration of the robotic manipulator. In 3D space, this configuration is represented by its position and orientation with respect to a reference frame.

In Figure 2.4, $O$-$xyz$ is depicted as the reference frame, and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are the unit vectors of the axes. A point $O'$ on the rigid body can then be expressed by (2.1) with respect to frame $O$-$xyz$ as.

$$\boldsymbol{o}' = o'_x\boldsymbol{x} + o'_y\boldsymbol{y} + o'_z\boldsymbol{z} \tag{2.1}$$

The position of point $O'$ can also be written as a (3x1) matrix

$$\boldsymbol{o'} = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix} \tag{2.2}$$

The orientation of the rigid body can be found by attaching an orthonormal frame $O' - x'y'z'$ to the rigid body and expressing its unit vectors $\boldsymbol{x'}$,$\boldsymbol{y'}$,$\boldsymbol{z'}$ with respect to the reference frame $O$-$xyz$ as

$$\begin{aligned} \boldsymbol{x'} &= x'_x\boldsymbol{x} + x'_y\boldsymbol{y} + x'_z\boldsymbol{z} \\ \boldsymbol{y'} &= y'_x\boldsymbol{x} + y'_y\boldsymbol{y} + y'_z\boldsymbol{z} \\ \boldsymbol{z'} &= z'_x\boldsymbol{x} + z'_y\boldsymbol{y} + z'_z\boldsymbol{z} \end{aligned} \tag{2.3}$$

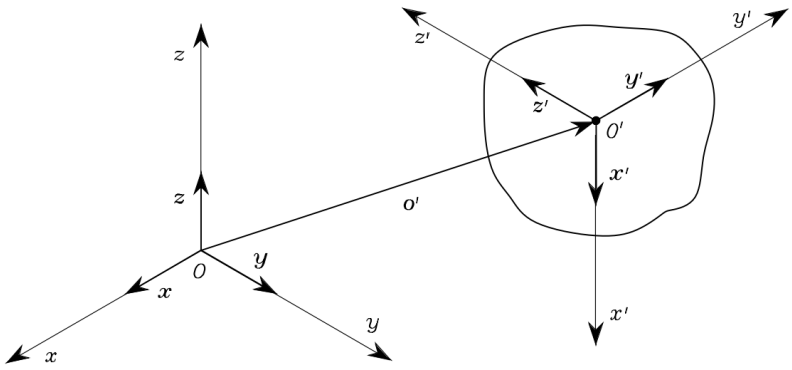



**Figure 2.4.:** Position & Orientation of a rigid body [5]

### 2.4.2. Rotation Matrix

The orientation of a frame can be described by a $(3 \times 3)$ matrix referred to as a *rotation matrix*. By combining (2.3) into a $(3 \times 3)$ matrix we get the rotation matrix $R$ as follows.

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{x'} & \boldsymbol{y'} & \boldsymbol{z'} \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{x'^T x} & \boldsymbol{y'^T x} & \boldsymbol{z'^T x} \\ \boldsymbol{x'^T y} & \boldsymbol{y'^T y} & \boldsymbol{z'^T y} \\ \boldsymbol{x'^T z} & \boldsymbol{y'^T z} & \boldsymbol{z'^T z} \end{bmatrix} \tag{2.4}$$

Continuing with the reference frame $O$-$xyz$, if it is rotated by an angle $\alpha$ about its $z$ axis, the new rotated frame $O - x'y'z'$ vectors are described as

$$\boldsymbol{x}' = \begin{bmatrix} \cos\alpha \\ \sin\alpha \\ 0 \end{bmatrix} \quad \boldsymbol{y}' = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \\ 0 \end{bmatrix} \quad \boldsymbol{z}' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The rotation matrix of the new frame $O - x'y'z'$ can then be expressed as

$$\boldsymbol{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Similarly, if the original frame *O-xyz* is rotated by an angle $\beta$ about its y axis, and an angle $\gamma$ about its x axis , they are respectively given by

$$\boldsymbol{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{R}_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.7}$$

These matrices are also referred to as the elementary rotation matrices, which represent rotation operations about coordinate frame axes.

### 2.4.3. Homogeneous transformation Matrix

The combined orientation and position of a rigid body is represented by the homogeneous transformation matrix, $T$. It allows the expression of coordinate transformations between two frames in a compact form by combining the rotation matrix $R$ in the Special Orthogonal Group $SO(3)$ and a translation vector $p \in \mathbb{R}^3$. The homogeneous transformation matrices in $\mathbb{R}^3$, is the set of all $4 \times 4$ real matrices T of the form

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

The major uses of the homogeneous transformation matrix include:

- To represent the position and orientation of a rigid body

- Change of reference frame in which a vector or a frame is represented
- The Displacement of a vector or a frame

### 2.4.4. Workspace

The reachable configurations of the robot's end-effector is commonly referred to as its *workspace*. The workspace depends on both the structure of the robot and mechanical joint limits, but it is independent of the task. A robot's workspace can be distinguished by its *dexterous workspace* i.e., the volume of space the end-effector can reach attaining different orientations, and its *reachable workspace*, which is the reachable workspace attained by the end-effector with at least one orientation. The workspace of KUKA KR-16 is shown in Figure 2.5. Marked in turquoise is the reachable volume of the robot. On the contrary, the area marked in white is unreachable for the robot. This can be due to mechanical limits of the joints or robot control software that prevents the arm from colliding or intersecting with other parts of the robot.

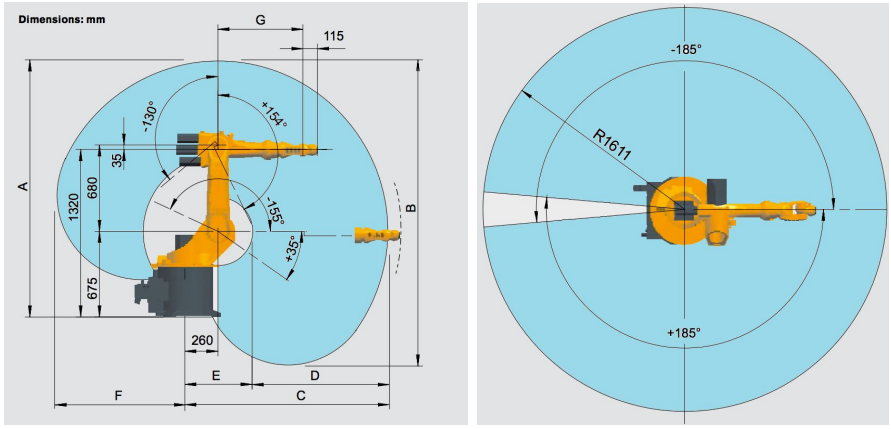



**Figure 2.5.:** Workspace of KUKA KR-16 [12]

| Work envelope | A | B | C | D | E | F | G | Volume |
|---|---|---|---|---|---|---|---|---|
| **KR 16** | 2026 mm | 2412 mm | 1611 mm | 1081 mm | 530 mm | 1027 mm | 670 mm | $14.5m^3$ |

**Table 2.1.:** Workspace dimensions [12]

### 2.4.5. Forward Kinematics

The forward kinematics problem is referred to as the calculation of the robots tool/end-effector position and orientation expressed by the robots joint variables.

The joint variables are defined as the angle between the links in case of a revolute joint, and the link extension in case of a prismatic joint [53], and the relationship between these variables can be expressed by $4 \times 4$ homogeneous transformation matrices, $T$. The degree of freedom of a robot can be determined by the number of homogeneous transformation matrices, and by multiplying these matrices, the final position of the end-effector can be located [23]. The forward kinematics of a manipulator can be derived using different models such as the *Denavit-Hartenberg Convention* presented in the following section.

### 2.4.6. Denavit-Hartenberg

The Denavit Hartenberg, or D-H convention, is a commonly used convention for selecting reference frames and computing the transformation matrices. Each rigid transformation has six degrees of freedom, three for rotation and three for translation, but with the D-H convention , each homogeneous transformation $A_i$ is represented as a product of four basic transformations

$$A_i = \text{Rot}_{z,\theta_i} \ \text{Trans}_{\ z,d_i} \ \text{Trans}_{\ x,a_i} \ \text{Rot}_{x,\alpha_i} \tag{2.9}$$

$$= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The parameters $a_i, \alpha_i, d_i$, and $\theta_i$ describe the relation between link $i$ and joint $i$, where

$\alpha$ and $\theta$ represent the rotations:

$\theta_i$ : is the angle between $x_{i-1}$ and $x_i$ measured in a plane normal to $z_{i-1}$

$\alpha_i$: is the angle between the axes $z_{i-1}$ and $z_i$ measured in a plane normal to $x_i$

If the joint is a revolute joint, the joint variable is included.
$a$ and $d$ represent linear distances of displacements.

$a$ : is the link length, and is measured by the distance between axes $z_{i-1}$ and $z_i$ along the $x_i$

$d_i$ : is the distance between the origin $O_0$ and the intersection of the $x_i$ axis with

$z_{i-1}$ measured along $z_i$ axis

The homogeneous transformation matrices $A_i$ are then formed by substituting the above parameters into (2.9), which are then used to derive the position and orientation of the end-effector frame expressed in base coordinates

$$T_n^0 = A_1 \cdots A_n \tag{2.10}$$

To apply the D-H conventions approach to the forward kinematics, a certain set of rules needs to be followed, a more derived description can be found in [5].

### 2.4.7. Inverse Kinematics

As forward kinematics is about figuring out the end-effector position given joint variables, the inverse kinematics problem is the opposite. Inverse kinematics determines the joint variables corresponding to a given end-effector position and orientation. The relationship between forward kinematics and the inverse kinematic problem can be seen in Figure 2.6.

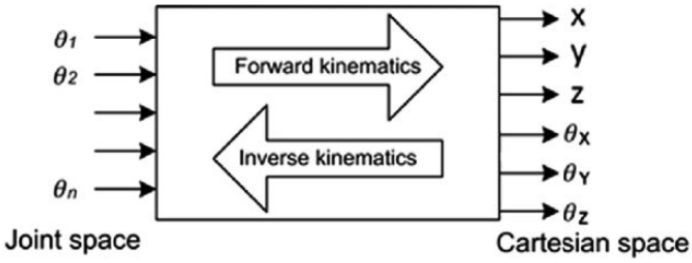



**Figure 2.6.:** Relationship between Forward and Inverse Kinematics [23]

Solving the inverse kinematics problem can be more complex, the reason being that the equations are, in general nonlinear, which means that it is not always possible to find a *closed-form solution*. Multiple solutions may exist depending on the number of DOFs of the manipulator and the number of non-null elements in the DH parameter table. Infinite solutions may also exist in the case of a kinematically redundant manipulator,*i.e.* when the number of DOFs is greater than the variables that are necessary to describe a given task, or no solutions may exist because of the manipulator's kinematic structure [21].

The inverse kinematics of KUKA KR-16 can be solved by decoupling the problem into position and orientation subproblems. The robot consists of an anthropomorphic arm where the initial $\theta-$ values of joints 1, 2, and 3 define the position

of the end effector while the $\theta$ values for joints 4, 5, and 6 define the orientation of the end-effector. The last three joints of the robot intersect at a common point and form a spherical wrist. Solving the inverse kinematics analytically can be done by the following:

1. Finding the wrist point $\boldsymbol{p}_W\,(x_w, y_w, z_w)$
2. Calculate $\theta_1, \theta_2, \theta_3$ from $\boldsymbol{p}_W$
3. Calculate the transformation matrix $\boldsymbol{T}_0^3$
4. Calculate the Euler angles $\theta_4, \theta_5, \theta_6$ which can be found from $\boldsymbol{T}_6^3 = \left(\boldsymbol{T}_0^3\right)^{\mathrm{T}} \boldsymbol{T}_6^0$

A more derived description on how to solve the inverse kinematics problem for manipulators with spherical wrist can be found in [5], as for this project, an approximate solution from the motion planner(MoveIt) 3.1.3, is used.

### 2.4.8. Operational space & Joint space

Section 2.4.6 described how the forward kinematics of a robot manipulator could be computed with the D-H convention, which describes the position and orientation of the end-effector frame expressed in base coordinates. It is also necessary to assign this expression as a function of time i.e a trajectory, which can be described in *operational space* and in *joint space*.

The end-effector pose can be expressed with a $\mathbb{R}^m$ vector, by describing the rotation of the end-effector frame with respect to the base frame with *Euler angles*, $\phi = \begin{bmatrix} \varphi & \theta & \psi \end{bmatrix}^T$, and the position with a minimal number of coordinates with regard to the geometry of the manipulator [5]

$$\boldsymbol{x}_e = \begin{bmatrix} \boldsymbol{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix} \in \mathbb{R}^m \tag{2.11}$$

where $m \leq n$, $\boldsymbol{p}_e$ is the end-effector position and $\phi_e$ describes the orientation.

The position and orientation of the end-effector is described by independent parameters, where the vector $\boldsymbol{x}_e$ is defined in space, also referred to as the *operational space*.

The *joint space* is denoted by the space in which the joint variables are defined by a vector $\mathbb{R}^n$

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \in \mathbb{R}^n \tag{2.12}$$

where $q_i = \theta_i$ if a joint is revolute and $q_i = d_i$ if a joint is prismatic. The forward kinematics can then be written in another form than (2.10) as:

$$\boldsymbol{x}_e = \boldsymbol{k}(\boldsymbol{q}) \tag{2.13}$$

where $\boldsymbol{k}$ is a non-linear vector function with dimension $\mathbb{R}^m$ , which allows the calculation of operational space variables from the knowledge of the joint space variables[5].

### 2.4.9. Trajectory planning

As a robot manipulator moves from its initial posture to its final posture, the robot controller is provided with a stream of goal positions and velocities. These positions as a function of time are referred to as *trajectories*, and in order to ensure that the robot manipulator executes the planned trajectories smoothly, it is necessary to use planning algorithms. The trajectories can be completely specified by the task that needs to be executed, such as when the end-effector is required to track a moving object, or in other cases where less constraints are applied, more freedom to design the trajectories are given [22].

### 2.4.10. Path & Trajectory

The terms *path* and *trajectory* are frequently used when it comes to trajectory planning, and they are often used synonymous, but different. Their differences is explained in [5] where a *path* is described as "the locus of points in the joint space, or in the operational space which the manipulator has to follow in the execution of the assigned motion". Which means that a path is a geometric description of a motion. In mathematical terms, a path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{final}}$ is defined as a continuous map, $\tau : [0, 1] \rightarrow \mathcal{Q}$, with $\tau(0) = \boldsymbol{q}_{\text{init}}$ and $\tau(1) = \boldsymbol{q}_{\text{final}}$ , where $\mathcal{Q}$ is described as the set of all possible configurations of the robot(configuration space), $\boldsymbol{q}_{\text{init}}$ is the robots initial position, and $\boldsymbol{q}_{\text{final}}$ is the robots final position [53].

A path is in some cases specified by a sequence of end-effector poses, and in this case a inverse kinematics solutions must be used for the conversion to joint configurations. The desired motion of an industrial robot is often achieved through

the use of a teach pendant by specifying paths. In this case, the motion is recorded as a set of joint angles, which means that the path will not serve as a trajectory for the robot and that there is no need to calculate the inverse kinematics [53].

A trajectory on the other hand, is described by [5] as "a path on which a timing law is specified, for instance in terms of velocities and/or accelerations at each point". This means that $\boldsymbol{q}(t_0) = \boldsymbol{q}_{\text{init}}$ and $\boldsymbol{q}(t_f) = \boldsymbol{q}_{\text{final}}$, where $t_f - t_0$ is the amount of time it takes for the robot manipulator to execute its trajectory.

### 2.4.11. Joint space trajectories

Trajectory planning for a robot manipulator is typically done in the *operational space* as it allows different constraints such as obstacles and forbidden areas of the workspace to be accounted for. This is due to the fact that the constraints are better described in this space as their corresponding points in the joint space can be difficult to compute [5].

When trajectory planning is done in the *joint space*, the manipulators joint values must first be obtained from the end-effector position and its orientation. An inverse kinematics algorithm must then be used in order to find its corresponding points in the operational space. There are some features that are required for this type of planning algorithm cited from Siciliano [5] as:

- The joint positions and velocities should be continuous functions of time
- The generated trajectories should not be very demanding from a computational viewpoint
- undesirable effects should be minimized, e.g, nonsmooth trajectories interpolating a sequence of points on a path.

The following sections present cases where only the initial and final position of the end-effector are given(point-to-point) and when points along the path are specified(sequence of points).

### 2.4.12. Trajectories for Point to Point

In a *point-to-point* motion, a robot manipulator is planned with a trajectory from $q(t_0)$ to $q(t_f)$, where its path is only dependent on its initial and final configuration. This type of trajectory planning is often used in material handling tasks, and in "*pick and place tasks*" where one object is picked from one location and placed in another, and can be related to the filter change investigated in this thesis. The point-to-point motion has infinitely many trajectories, stemming from its finite constraint on the end point. A common way of dealing with this issue is

to choose trajectories from polynomials of degree $n$, where $n$ is dependant of the number of constraints [53].

If we want to generate a smooth trajectory between two configurations $q_i$(initial) and $q_f$(final) within a time $t_f$, a cubic polynomial of the form

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{2.14}$$

can be chosen. It is imposed that $q(t_0) = q_0$, $q(t_f) = q_f$, $\dot{q}(t_0) = v_0$ and $\dot{q}(t_f) = v_f$. Its velocity is then given by its derivative

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{2.15}$$

the acceleration is then

$$\ddot{q}(t) = 2a_2 t + 6a_3 t \tag{2.16}$$

Combining Equation 2.14 and 2.15 gives the set of equations

$$q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \tag{2.17}$$
$$v_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 \tag{2.18}$$
$$q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \tag{2.19}$$
$$v_f = a_1 + 2a_2 t_f + 3a_3 t_f^2. \tag{2.20}$$

where the initial and final joint velocity values($\dot{q}_i$ and $\dot{q}_f$) are also specified. Solving these equations will then give a specified trajectory. These equations can also be combined into a single matrix equation on the form

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix} \tag{2.21}$$

In the case of assigning an initial value $\ddot{q}_i$ and $\ddot{q}_f$ for the acceleration, a polynomial of degree $n \geq 5$ is needed, since six degrees have to be satisfied . The polynomial is then given by the following equation

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{2.22}$$

### 2.4.13. Sequence of points

In a pick and place task it might also be suitable to assign more points other than the initial and final point of the end-effector. For instance, by assigning suitable points between the initial and final points, the object is imposed to reduced velocities when transferred. Assigning a sequence of points can also certify better monitoring of the trajectories executed [5].

# Chapter 3.

# Programming Tools

This chapter contains the most relevant ROS concepts and programming tools used for this project. Parts of the theory are gathered from the specialization project [21] and modified for this thesis, apart from sections that are taken in their entirety and cited with [21]. Most of the theory regarding ROS is gathered from the book *Mastering ROS for Robotics Programming* [24], as well as the Roswiki pages online [46].

## 3.1. ROS

The robot operating system(ROS) is an open-source framework for developing and building robotic applications. The framework is equipped with various tools and libraries, making it advantageous when developing, which has led to high-end robotics companies implementing their software through ROS [24]. The ROS project was started in 2007 as part of a Stanford robot project and has gained a vast community of researchers and developers since then. ROS allows for creation applicable for broad classes of robot hardware and software pipelines [42] and comes with "ready-to-use" capabilities, such as MoveIt, which can be used for motion planning of robot manipulators.

ROS is also equipped with tools for both visualization and simulation of robots, which has made it the framework of choice for this thesis. ROS comes with different distributions and currently has two main versions, ROS1 and ROS2, the latter being the newest. As for this thesis, ROS1 is the preferred version since it has more available packages. The distribution of the used ROS1 is called *Noetic Ninjemys* [48], which is the newest and recommended.

### 3.1.1. ROS Computation Graph

The computation in ROS is done through a network of processes, which together construct the *Computation graph* as can be seen in Figure 3.1. The main ROS concepts in the computation graph are; ROS **Nodes**, **Master**, **Parameter Server**, **messages**, **topics**, **services**, and **bags**, all of which provide data to the computation graph in distinct ways.

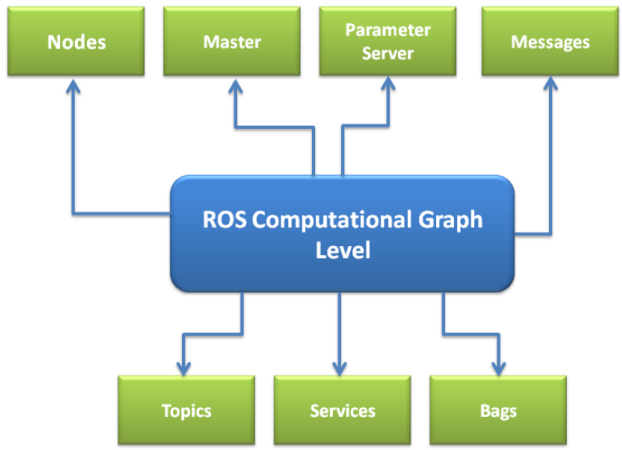



**Figure 3.1.:** ROS Computational Graph Layer [24]

#### Nodes

Nodes are software processes that perform computations or tasks. Each node usually has one single purpose and can perform its task independently, but it can also communicate with other nodes. A robot system might consist of many different nodes; for example, one node can calculate the end-effector's position while another node can perform path planning, and so on. By dividing code into smaller sub-modules, it facilitates the communication between the different parts of the robot system through nodes. Having more simple processes rather than one extensive process is the aim of ROS nodes. This provides the benefit to the overall system and makes it fault-tolerant as it isolates a fault in the system to the node in question [24].

#### Topics

Nodes exchange messages through named buses called *Topics* [24]. When a node sends a message through a topic, it is said to *publish*, while if a node receives a message through a topic, it is said to *subscribe* to the topic. Each topic has its

unique name, and the publish/subscribe model is anonymous, i.e., one node does not know whether another node is publishing or subscribing to the topic.

### Messages

Each ROS node usually has one single purpose and can perform its task independently, but it can also communicate with other nodes through *Messages*. Messages hold a set of data and can be of different types, such as an integer, a floating-point, and a boolean. Figure 3.2 shows an example of communication through messages. From the left, a node publishes a message over the topic called `/example`. The topic will define the type of messages sent, which in this case is a message of type `String`. These transmitted messages are then received by two nodes that subscribe to the `/example` topic [7].

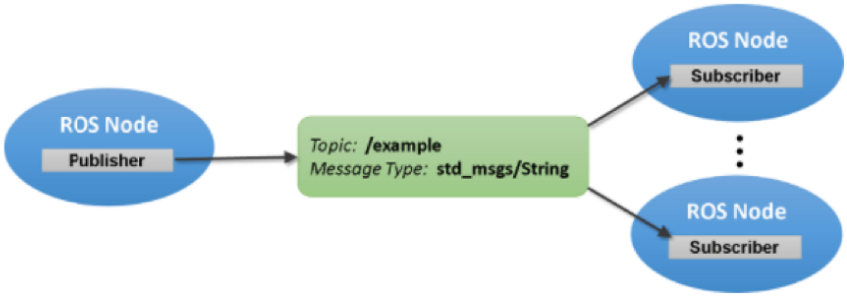



**Figure 3.2.:** ROS Publisher & Subscriber model [27]

### Services and Clients

Another way to pass data between nodes in ROS is through services. Services allow for single direct communication between nodes, *i.e.* one node can call a function that executes in another node. The inputs and outputs of the server function are defined in the same way message types are defined. Service calls can be used for quick actions such as turning on a sensor, taking a picture with a camera, or enabling/disabling a robot actuator[42, 21]

### ROS Parameter Server

The ROS master provides a parameter server used by nodes for configuration, storing, and retrieving data. The nodes can access the stored data globally since it is connected to the ROS master. Each parameter has a name and a data type, and the command-line tool for the parameter server is called: `rosparam`[42, 21].

### 3.1.2. ROS control

ROS control is a package for hardware drivers used to implement and manage robot controllers. It contains controllers and hardware interfaces and works as a "bridge" between the software and hardware. ROS control allows for mix and match of controllers since they are decoupled from the robot hardware structure, which means that the controllers can be developed independently and in isolation. There are many available controllers developed in the `ros_controllers` meta-packages such as `joint_position_controller`, which receives a position input and sends an effort output, using a PID controller [9]. The data flow of the ROS controllers can be seen in Figure 3.3 [21].

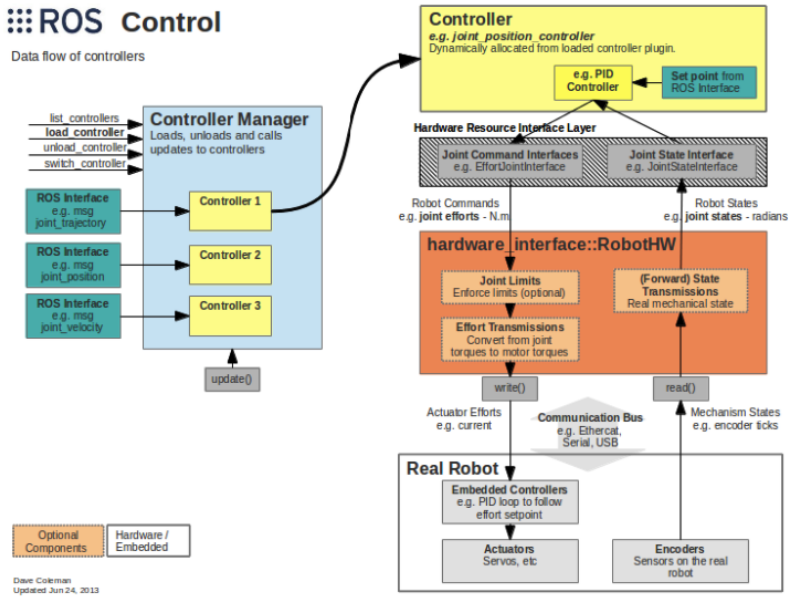



**Figure 3.3.:** Dataflow of controllers [47]

### 3.1.3. MoveIt

Motion planning in robotics is often referred to as the problem of enabling a robot to execute an assigned task in the presence of obstacles without colliding with them[5]. In ROS, the main motion planning software is called MoveIt and is the most widely used software for manipulation of robots. According to its main website, MoveIt "incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control, and navigation"[54]. MoveIt has been integrated with many robots, and its target application is industrial, commercial, and research environments. Motion planning algorithms are not directly run;

instead, MoveIt uses plugins for most of its functionality, such as kinematic plugins(KDL) for forward and inverse kinematics, motion planning plugins(OMPL), and collision detection [10]. It also comes with a GUI tool called *Setup Assistant*, which is used to configure a robot from its URDF.

MoveIt uses a concept called `move_group` which is the primary node. The `move_group` provides actions and services which can be interfaced with the use of `C++`, `Python` or through a `GUI`. The MoveIt architecture is depicted in Figure 3.4 below. It can be seen that the `move_group` node collects information about the robot's state in the form of topics and services. It also collects the robot kinematic description provided by the URDF through the ROS parameter server.

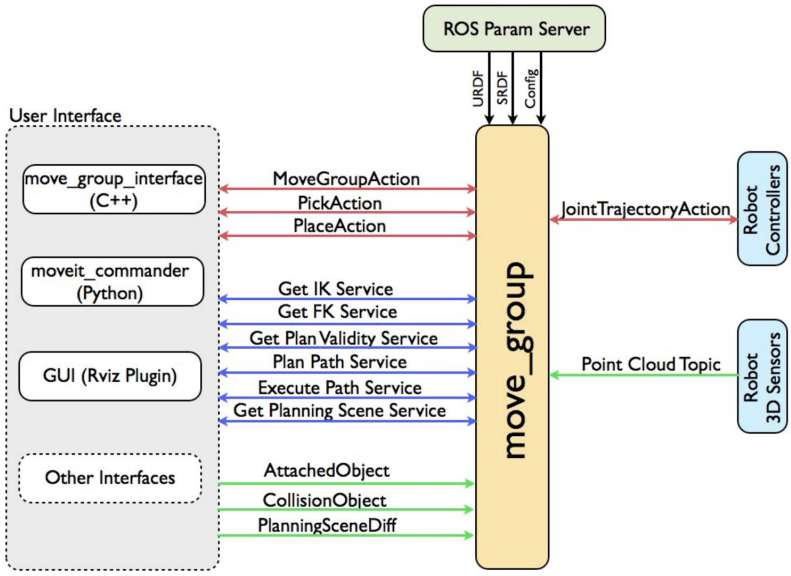



**Figure 3.4.:** MoveIt architecture [29]

### 3.1.4. RViz

ROS visualization, also referred to as RViz, is a 3D visualization tool for ROS and provides a view of the robot model, sensor information, and algorithms. RViz can be used for any robot and is a good tool for debugging a robot application [21].

### 3.1.5. Gazebo

Gazebo is a 3D simulation program integrated with ROS as its primary simulation tool. Gazebo allows for the creation of a 3D environment for the robot and uses a physical engine for gravity, inertia, illumination, etc. Gazebo provides a

realistic environment for the simulated robot, which allows testing the robot in difficult or dangerous scenarios without harming the real robot. It has multiple features and controllers. With `gazebo_ros_control` a lot of the underlying control mechanisms of the robot can be achieved [21].

## 3.2. URDF

A Universal Robot Description Format(URDF) file describes a robot's physical description in ROS. The URDF is in XML format and includes the robot's kinematics and dynamics and 3D models for visualization. The description of a robot consists of joints and links. The joints describe the kinematics of the robot and connect two links: a parent link and a child link. Different types of joints can be represented, such as revolute, prismatic, continuous, and fixed. Torque, velocities, and limits for the joints are also supported. The links in the URDF describe the mass properties of the robot[22, 21]. The first joint of KUKA KR-16 is shown in URDF format below.

**Listing 3.1:** First joint of KUKA KR16

```
<joint name="${prefix}joint_a1" type="revolute">
      <origin xyz="0 0 0.675" rpy="0 0 0"/>
+ <parent link="${prefix}base_link"/>
      <child link="${prefix}link_1"/>
      <axis xyz="0 0 -1"/>
      <limit effort="0" lower="${radians(-185)}" upper="${radians
          (185)}" velocity="${radians(156)}"/>
    </joint>
```

### 3.2.1. Xacro

Xacro is an XML macro language that extends the URDF. The xacro reduces the code length in the URDF by creating and reusing macros inside the robot description, simplifying the URDF. It also supports simple programming statements such as mathematical expressions, conditional statements, and variables [24].

# Chapter 4.

# Method

This chapter presents the methodology part of the thesis. It showcases both the software as well as the hardware implementation to conduct the filter change. The chapter also explains the control architecture developed and modifications made to the filter assembly provided by Equinor.

## 4.1. Software Implementation

In this section, the software setup is presented. Implementations in ROS to get a visualized model of the robot in RViz and a simulation in Gazebo are covered. The implemented nodes and the control architecture are also presented.

### 4.1.1. MoveIt Configuration

The first step of the process is to create a configuration of the KUKA robot in ROS. Packages for KUKA manipulators within ROS-industrial have already been implemented and can be found in [44]. The packages include a complete description of the KUKA robot in addition to hardware interfaces, but for the sake of motion planning, the setup assistant GUI provided by MoveIt is used. As mentioned in Section 3.1.3, the setup assistant is used to set up a robot configuration in MoveIt. It semantically describes the robot by creating an SRDF file, configuration files, launch files, and scripts from the robot's URDF to use the `move_group` node. After input of the KUKA robots URDF in the GUI, a set of variables that describe the system are defined. The main variables include;

- Collision Matrix Generation
- Adding virtual joints
- Adding planning groups

- Gazebo Simulation
- ROS control

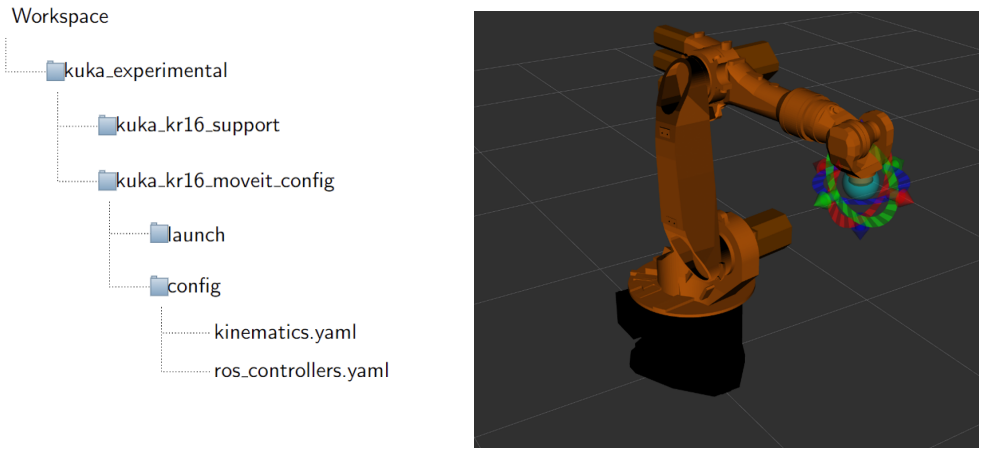
The collision matrix generation's main objective is to decrease the motion planning processing time. This is done by disabling collision checking for some links of the robot, i.e., they are disabled when they are always in collision, never in collision, in collision when in the manipulator's default position, or when the links are adjacent to each other on the kinematic chain [32]. The sampling density is also declared where the default value of 10,000 collision checks is implemented. More computation is required for higher densities, but it ensures that link pairs are not disabled.

The purpose of adding a virtual joint is to establish a connection between the robot base(*base_link*) to the world-frame and is done as shown in Listing 4.1. Further, planning groups are added, which constitute the parts of the robot manipulator and the end-effector. Another benefit of MoveIt setup assistant is the generation of Gazebo files which are used to simulate the robot; although the default values of the file are not enough to get a complete working simulation, it constitutes a good starting point. This is further explained in section 4.1.3.

**Listing 4.1:** Attaching base_link to the world frame

```
<virtual_joint name="virtual_joint" type="floating" parent_frame="
    world" child_link="base_link"/>
```

The setup of the generated workspace can be seen in Figure 4.1a. After completing the setup, the robot can be launched in the visualization program RViz, depicted in Figure 4.1b. The figure shows the robot with an interactive marker on its tool link. This marker can be used to move the robot around with a joystick or mouse in joint space. The visualization program also allows the joints to be specified in the graphical user interface.




**(a)** Folder layout **(b)** KUKA KR-16 in RViz

**Figure 4.1.:** Folder layout and KUKA KR-16 Visualized in RViz

### 4.1.2. Kinematics Configuration

Configuration of kinematic parameters are done in the *kinematics.yaml* file found in the config folder 4.1a. As mentioned in section 2.4.7, the inverse kinematics for KUKA KR-16 is solved with MoveIt. MoveIt uses the package called *Kinematic and Dynamics Library(KDL)* and solves the inverse kinematics based on a numerical Jacobian algorithm provided by OROCOS [38].

The parameters include:

- *Kinematics_solver*: Which is the kinematics solver plugin used, and in our case is: `kdl_kinematics_plugin/KDLKinematicsPlugin`
- *Kinematics_solver_search_resolution*: The quality of searches in the redundant space of the robot is specified, and is set to 0.005.
- *Kinematics_solver_timeout*: This is a timeout between each iteration of inverse kinematics and is specified in seconds:0.005
- *Kinematics_solver_attempts*: This is the number of random restarts performed.

### 4.1.3. Simulation

The Gazebo simulation files provided by the setup assistant are modified in order to get a working simulation. It is intended that the robot manipulator's motion

will be initialized with the `move_group` node, visualized in RViz, then simulated in Gazebo. Actuating the robot joints is done through the use of *ros_control* packages as mentioned in section 3.1.2. A controller is added to the config folder, which allows manipulation of the robot, and can be found in Appendix(A.2). The controller is of type `position_controllers/JointTrajectoryController` which contains the joint names and PID parameters. It allows the execution of joint-space trajectories, which can be specified as a set of waypoints consisting of positions, velocities, and accelerations.

The controller also needs to be compatible with the hardware interface added in the transmission tag of the robots URDF. The added transmissions models the output of a motor and the joint it is attached to. The transmissions are empty with a gear reduction of 1, where the transmission of the first joint can be seen in Listing 4.2. A `gazebo_ros_control` plugin is also added, as this plugin identifies which hardware interface to load. The plugin used is a position hardware which also includes a list of controller_manager services used to stop, switch or start controllers.

**Listing 4.2:** Attaching base_link to the world frame

```
<transmission name="a1_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint_a1">
      <hardwareInterface>hardware_interface/PositionJointInterface<
          /hardwareInterface>
    </joint>
    <actuator name="shoulder_pan_motor">
      <hardwareInterface>hardware_interface/PositionJointInterface<
          /hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>
```

In summary, the controller receives a desired joint angle and uses a control loop feedback mechanism that adjusts the output that is sent to the actuators. In contrast, the hardware interface acts as a bridge between the controller and Gazebo.

The ROS setup includes the visualization in RViz and a simulation of the robot movements in Gazebo. During the setup phase of a proper simulation, it was noted that a bug with the feature `PositionJointInterface` for the joint transmission hardware interface(Listing 4.2) made the robot unstable and fell due to gravity in the simulator.

A configuration of the PID parameters shown in the controllers(A.2) was attempted, such that the robot listens to the commands sent. The PIDs are a part

of the robot controllers file and are tuned with the use of rqt in ROS. Figure 4.2 shows the PID tuning. The blue graph represents a sinusoidal wave, and the red graph represents the actual position of the joint. A sinusoidal wave was sent to the base of the robot(joint_a1) to induce movement(back and forth), but as shown in the graph(red), it does not correspond to the same signal. Several Hz values were tested, resulting in the same slow movements. It was noted that the simulated robot had almost no inertia, which can be seen from a snippet of its URDF in Listing 4.3. This means the robot is massless and has no energy to move the limbs. A quick fix was then introduced by turning off the gravity for each link of the robot in Gazebo, such that the same movements from RViz with the motion planner are shown in Gazebo. The `PositonJointInterface` is also changed to a `EffortJointInterface`. Since gravity is turned off in the simulator, the focus is more shifted towards visualization in RViz and is used as the main verification tool.

**Listing 4.3:** Snippet of the inertia tag

```
<inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz="0.01"/>
```

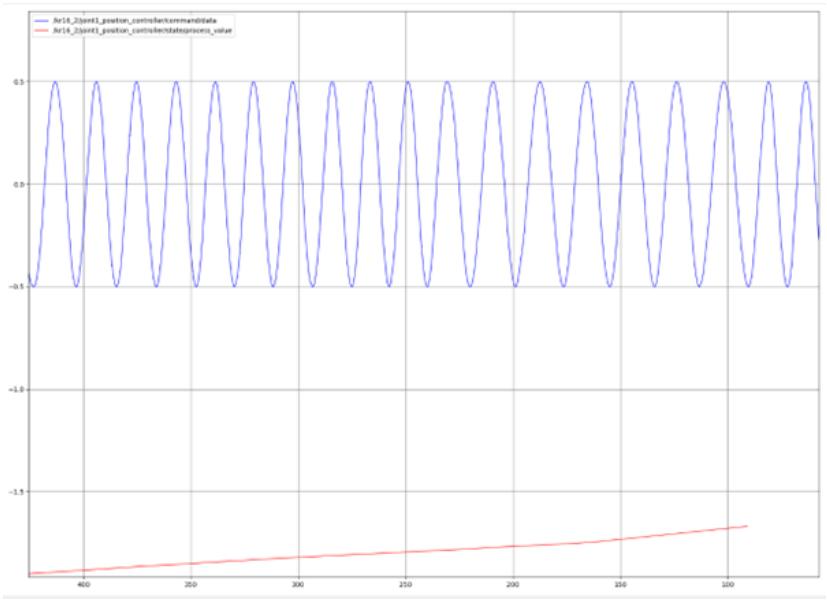


**Figure 4.2.:** Pid tuning

### 4.1.4. Nodes

This section presents the primary nodes of the system. This is to get an overview of the different nodes' main functions and describe some of their properties.

### Move_group node

The `move_group` node is considered the main node of the system, as it encompasses most of the functionalities. As presented in Section 3.1.3, the `move_group` node can also be interfaced through Python.

### Jointspace node

This node encompasses the complete pick-and-place cycle. The node is implemented in Python with the use of MoveIts *Move Group Python Interface*, where the original code can be found on Github [33]. The move group interface includes many functionalities, such as defining joint goals and creating objects in the RViz environment.

### Joint State Publisher

This nodes main function is to publish the joint states of the robot to the system as `sensor_msgs/JointState` messages. It is complemented with the `robot_state_publisher` node.

### Robot State Publisher

The `robot_state_publisher` nodes main property is to publish the state of the robot(from the Joint State Publisher) to `tft2`. This includes the position and orientation of each coordinate frame of the robot, and it creates a kinematic model.

### Gazebo node

The `gazebo_node` is the node responsible for simulating the robot in Gazebo. The robot's motions are first planned with the motion planner in Rviz, which generates a trajectory and communicates with the robot controller through the `FollowJointTrajectoryAction` interface. This trajectory is then simulated in Gazebo.

### Joystick node

The "master" device used in this thesis is a PS4 controller also named "DualShock4". This controller was picked as there were no other options for teleoperating the KUKA robot. Although the controller does not have haptic functions other than vibration, the benefits are that an implementation of the `move_group`

node with the controller is available through MoveIts webpages. The controller mappings are presented below in Table 4.1.

| Command | PS4 Controller |
|---|---|
| +-x/y | left analog stick |
| +-z | L2/R2 |
| +-Yaw | L1/R1 |
| +-Roll | Left/Right |
| +-Pitch | up/down |
| Change planning group | Select/start |
| Change end effector | Triangle/Cross |
| Plan | Square |
| Execute | Circle |

**Table 4.1.:** Mappings from controller to robot [30]

#### Robot Sensor Interface

The `KUKA_RSI_node` is the node that is responsible for communication between ROS and hardware setup. It has already been implemented and is a part of the `kuka_experimental` packages available on Github [44].

### 4.1.5. Summary

This section presented the software architecture. The primary nodes have been introduced and explained. Figure 4.3 shows an overview of how the communication between the different parts of the system occurs. The joystick node allows for manipulation of the robot's end-effector through the `move_group` node while being visualized in RViz. The `move_group` node is considered the primary mode of the system, as it consists of the most fundamental processes.

The current state of the robot is published by the `KUKA_RSI` node over the `/joint_states` topic, where the `move_group` node subscribes to. In addition, the `move_group` node monitors the different robot coordinate frames through the *tf* library. The `move_group` also monitors the *Planning Scene*, which provides a view of the current world in RViz. The planning scene includes the robot's current state as well as any objects added, such as a representation of the filter for the pick and place task.

After a desired movement of the robot has been initiated, the motion planner GUI in RViz is used to plan the specified motion, and collisions are checked. The desired trajectory, which obeys the velocity and acceleration limits, is then

generated by the `move_group`, which in turn moves the robot arm. The visualized movement in RViz is then simulated by Gazebo(`gazebo_node`). Communication between ROS and the hardware is done through the RSI node, which is also used for obtaining the robot's current position.

It is noted that this is not a bilateral teleoperation system. Recall that in a bilateral teleoperation system, the operator sends velocity, force, or position commands from the master device to the slave device, and information is exchanged bidirectionally in real-time. In our case, the master device does not directly actuate the real robot and does not provide force feedback. The master device is used to move the robot to its desired position in joint space; the motion is then planned and executed in RViz, which in turn actuates the real robot; thus the system is unilateral. It is also intended that the master device will be used for movements that are large and do not require precision, while the `joint_space_node` contains the actual and precise end-effector positions, which can be set in motion through the computer.

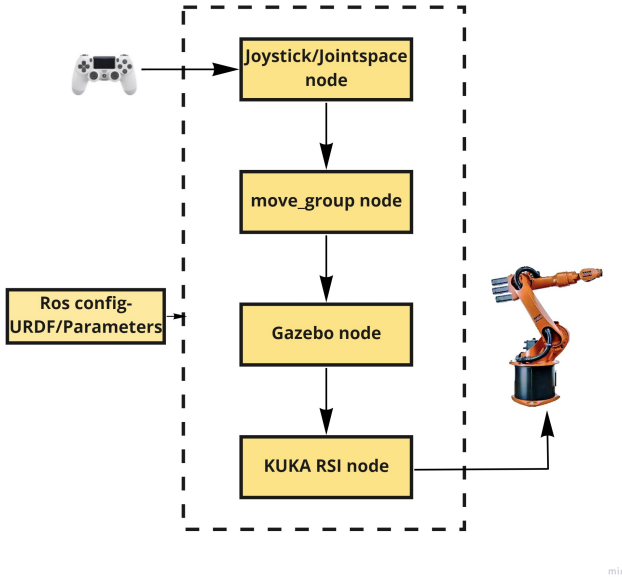



**Figure 4.3.:** Control architecture

## 4.2. Hardware setup

This section presents an overview of the hardware used to perform the manipulation tasks in the laboratory.

### 4.2.1. Robot Hardware

The robot used for the filter change is the KUKA KR-16, stationed at MANULAB. This particular robot is used as it has the ability to reproduce the human motions needed for the pick-and-place task, as mentioned in the problem description(1).

KUKA KR-16 is an industrial robot used in many applications such as assembly, pick and place, material handling, and arc welding. The robot has six degrees of freedom with a spherical wrist, implying that three of the revolute joints axes intersect at a common point. This feature allows the decoupling of position and orientation of the end-effector, where the manipulator arm positions the mentioned point and the wrist determines the end-effector orientation. Figure 4.4 shows the hardware that comes with the robot manipulator, and Table 4.2 shows additional robot specifications. The hardware includes a teach pendant(4.4a), and a robot controller(4.4b), apart from the robot manipulator(4.4c). Communication between the robot controller(KRC4) and the external computer is done through the *Robot Sensor Interface*(RSI). The exchanged data is transmitted via UDP/IP or TCP/IP protocol as XML strings by connection to Ethernet. The data packets sent from the KRC4 must be responded to within 12ms by the computer [13].

The teach pendant, also named *KUKA Control Panel*(KCP), is used for manual control of the robot. It comes with a touch screen and is equipped with control and display functions for operating and programming the KUKA robot. The primary programming language used for KUKA robots is called *KUKA Robot Language*(KRL) and runs on the KCP.


**(a)** Teach pendant **(b)** KRC4 **(c)** KUKA KR-16

**Figure 4.4.:** Physical installation hardware

| Robot Specification | |
|---|---|
| Description | Value |
| Axes | 6 |
| Payload | 16kg |
| H-Reach | 1610 mm |
| Repeatability | ±0.1 mm |
| Robot Mass | 235kg |
| Structure | Articulated |
| Mounting | Floor |

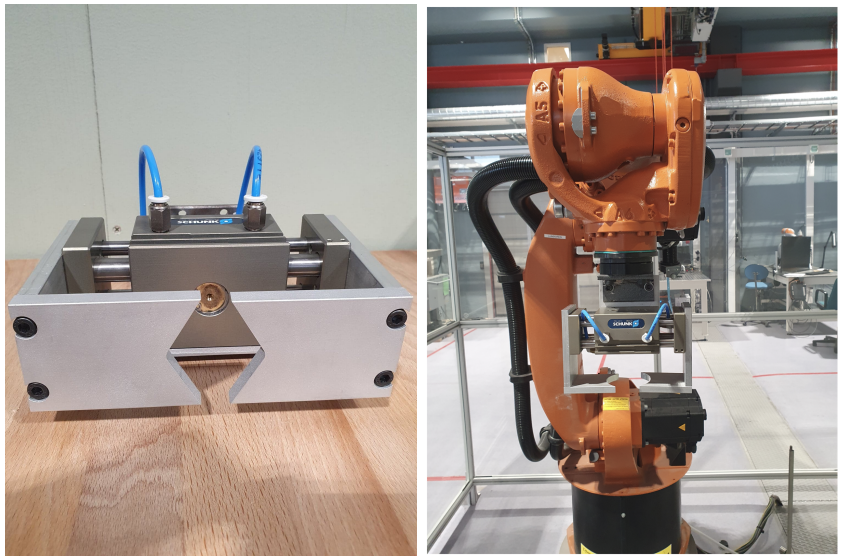
**Table 4.2.:** Robot specifications [43]

### 4.2.2. Gripper

The development of a suitable gripper for the pick and place task was not taken into consideration in this thesis due to time restrictions. Fortunately, there was an available gripper in the laboratory, but not intended for the task. Some appropriate changes were made, which included;

1. Change of the gripper claw from its original sharp edges to a circular shape to fit the filters.
2. Reducing the gripper's open-close length.

A model of the gripper was also created in the CAD program *Solidworks* by my lab partner. Solidworks has a convenient tool for exporting parts to URDF-format and was used to attach the gripper to the robot tool link. It is also worth mentioning that the gripper does not have a system to activate it. This means that it is manually opened and closed when used in the experiments; this is further explained in Section 4.3. The robot's complete hierarchical structure, including the gripper, can be found in Appendix A.4 and its machine drawing in Appendix B.1. The original gripper can be seen in Figure 4.5a, and the new gripper is seen mounted on the robot in Figure 4.5b.



**(a)** Original gripper **(b)** Gripper mounted on KR16

**Figure 4.5.:** The original gripper & Remodelled gripper mounted

### 4.2.3. Filter Assembly

The filter assembly used in the experiments was deployed by Equinor to the robotics laboratory. The complete assembly consists of the following parts;

1. Four filters
2. Filter container
3. Two Filter rods

The complete assembly can be seen in Figure 4.6, with its dimensions in Table 4.3. The filter rod is placed inside the filter and tightened at its top side before it is placed inside the filter container. The filter container has threads on its top side and consists of two parts that can be disassembled, as seen in the figure.

Some changes were also applied to the assembly for the filter container to maintain stable during the pick-and-place experiment. Construction of steel bars was welded on the bottom and bolted on the container. This increases the stability of the whole part during task execution. The bottom section was also sealed off with a plate.

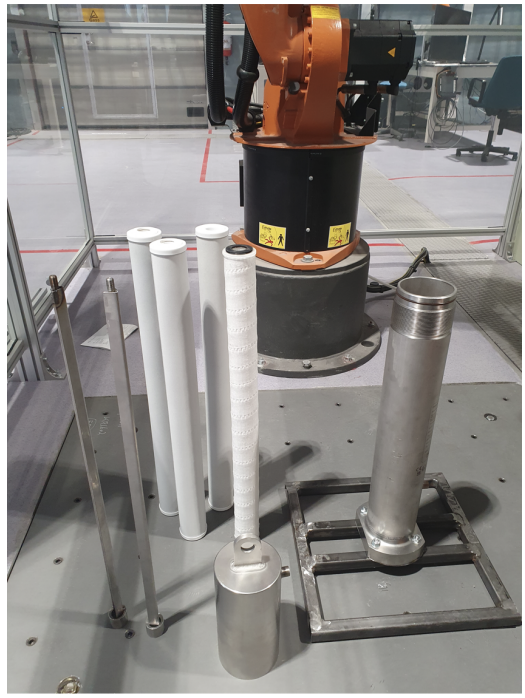


**Figure 4.6.:** Complete filter assembly

| | **Filter Container** | **Filter** |
|---|---|---|
| **Height** | 660mm | 765 mm |
| **Diameter** | 100 mm | 75 mm |

**Table 4.3.:** Filter& Filter container dimensions

## 4.3. Pick and place

As mentioned throughout the thesis, the main objective is to conduct a filter change, and the cycle is presented as follows:

1. Perform end-effector movement towards the filter container
2. Unscrew the top lid of the filter container and place it at a specified location
3. Pick up the old filter and place it at a drop-off location
4. Pick up the new filter
5. Place the new filter inside the filter container

The gripper presented in section(4.2.2) is not intended for the filter change, im-

plying that point 3 presented in the pick-and-place cycle is disregarded. This means that the top lid is taken off before the cycle is initiated. The gripper was previously used with a pneumatic system prior to this thesis, as can be seen from the blue cables attached to it in Figure 4.5a, but a new pneumatic system was not developed for this thesis. This leads us to open the gripper manually, pre-grasp of the filter, stop the cycle, and manually close the gripper before the robot continues with the complete cycle.

The complete workstation is seen in Figure 4.7. It can be noted that the robot's workspace is slightly reduced and differs from the one presented in section 2.4.4. The workspace is compromised on the robot's left side, leading us to choose the right side for the drop-off location. The filter container is positioned such that it is capable of the movement needed to pick up the filter.

Other changes to the system also included the filter assembly, as mentioned in the previous section. The idea was to keep the filter pole centered in the middle of the filter container and insert the rod into a steel plate at the bottom, such that the pole is stable and does not move through the process of the filter being picked up and placed back again. Some miss understandings with the workshop ended up with a fully sealed bottom. This means that the stability problem for the filter still exists and causes the filter to lean towards one side of its container as there is a 25mm gap.

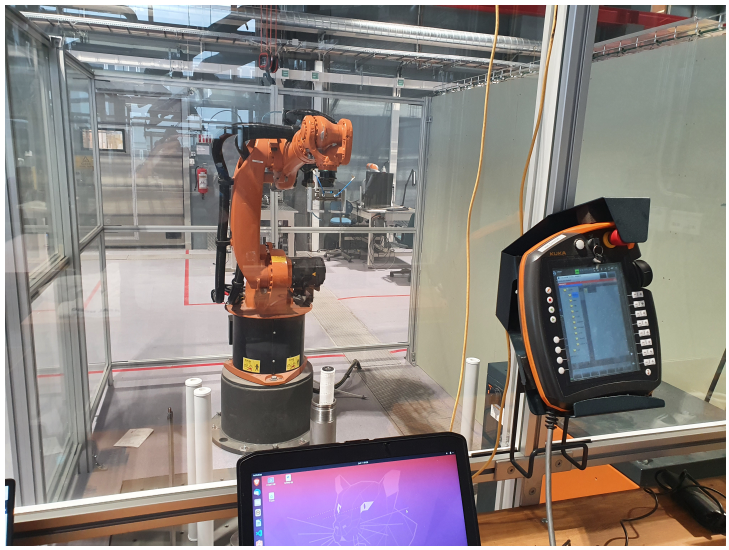


**Figure 4.7.:** Laboratory workstation

## 4.4. Testing

Testing the control architecture presented in Section 4.1.5 consisted of a mixture of ROS and tests in the laboratory. The implementation of the `joint space node` included using the KCP(teach pad), as it can control the robot directly. The KCP has different control modes: T1 and T2, where T1 is used for testing new programs as it has reduced velocity(max 250mm/s) [13].

The KCP was used to get the right positions and orientations of the end-effector. Ten positions were then gathered, which constitute the whole pick-and-place cycle. The joint angles presented in Table 4.4 represent six joint values $\theta_n$, where $n = 1, \ldots, 6$, for each position of the cycle in degrees. The corresponding operational space coordinates are presented in Table 4.5, in meters. The latter values are provided by *tf*, where output is shown in the RViz GUI. The complete cycle can then be presented as follows:

1. P0: The robot is at its home position(*home*)
2. P1: Movement towards old filter position(*filter1*)
3. P2: The old filter is picked up(*filter1up*)
4. P3: Movement towards drop location(*filter1predrop*)
5. P4: The old filter is dropped(*filter1drop*)
6. P5: Movement to position above the new filter(*filter2up*)
7. P6: New filter position(*filter2*)
8. P7: The new filter is lifted up(*filter2up*)
9. P8: Movement towards position above the filter container(*home*)
10. P9: New filter is placed inside the filter container(*center + filter1*)
11. P10: The robot moves back to home position(*home*)

The developed python code allows for executing robot movements towards the different points presented. This is done by specifying the desired position in the terminal, where the positions are shown in parentheses in the pick-and-place cycle above. By typing `home`, the robot will move to its home position, etc.

An implementation of the Denavit Hartenberg convention was also performed, following the presented theory in Section 2.4.6. A parameter table consisting of the four parameters was constructed from the robots URDF, and used to derive the transformation matrices. This is to verify the different parameters, and the operational space coordinates provided. The transformation matrices for the complete cycle, and a table version of the URDF can can be found in Appendix A.3.

| Positions | Joint angles | | | | | |
|---|---|---|---|---|---|---|
| | $\theta_1$ | $\theta_2$ | $\theta_3$ | $\theta_4$ | $\theta_5$ | $\theta_6$ |
| Home | -4 | -72 | 62 | 180 | -99 | -67 |
| $P_1$ | -4 | -51 | 95 | 180 | -45 | -67 |
| $P_2$ | -4 | -72 | 62 | 180 | -99 | -67 |
| $P_3$ | -48 | -75 | 65 | 180 | -98 | -67 |
| $P_4$ | -48 | -51 | 97 | 180 | -44 | -67 |
| $P_5$ | -42 | -48 | 21 | 180 | -117 | -61 |
| $P_6$ | -42 | -35 | 72 | 180 | -53 | -61 |
| $P_7$ | -42 | -48 | 21 | 180 | -117 | -61 |
| $P_8$ | -4 | -72 | 62 | 180 | -99 | -67 |
| $P_9$ | -4 | -51 | 95 | 180 | -45 | -67 |

**Table 4.4.:** Joint angles for pick-and-place

| Positions | Coordinates($m$) | | |
|---|---|---|---|
| | $x$ | $y$ | $z$ |
| $Home$ | 1.13 | 0.08 | 1.40 |
| $P_1$ | 1.14 | 0.08 | 0.71 |
| $P_2$ | 1.13 | 0.08 | 1.40 |
| $P_3$ | 0.74 | 0.82 | 1.41 |
| $P_4$ | 0.75 | 0.84 | 0.70 |
| $P_5$ | 0.99 | 0.89 | 1.45 |
| $P_6$ | 0.99 | 0.89 | 0.63 |
| $P_7$ | 0.99 | 0.89 | 1.45 |
| $P_8$ | 1.13 | 0.08 | 1.40 |
| $P_9$ | 1.14 | 0.08 | 0.71 |

**Table 4.5.:** Corresponding Cartesian points

# Chapter 5.

# Results

This chapter presents the results from the experiments conducted in the laboratory.

## 5.1. Teleoperation

This section presents a visual representations of the teleoperation system in ROS. The setup presented in Section 4.1.5 is run in ROS with the `joint_space_node` which allows for commands as input in the terminal while teleoperating the robot in RViz. Figure 5.1 shows a visualization of the robot in RViz and simulated in Gazebo. The joystick is connected to the computer and allows for manipulation of the end-effector, which can be seen as an interactive marker on its tool link. The robot's joints follow the end-effector position, and the motion can be executed through the joystick. The implemented setup with the joystick did not get tested in the laboratory; this is further discussed in Section 6.1. The communication between the nodes can be seen from the node graph in Figure 5.2.

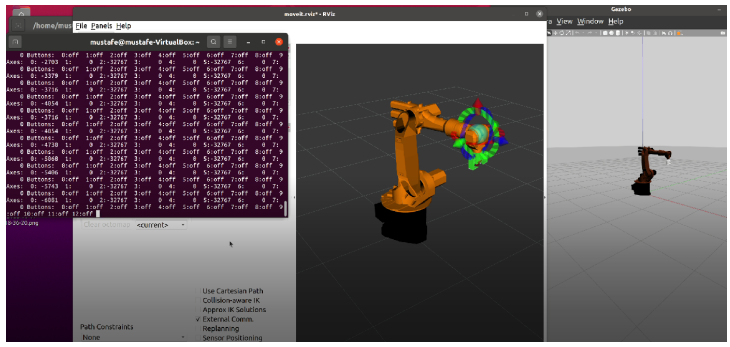


**Figure 5.1.:** Joystick teleoperation

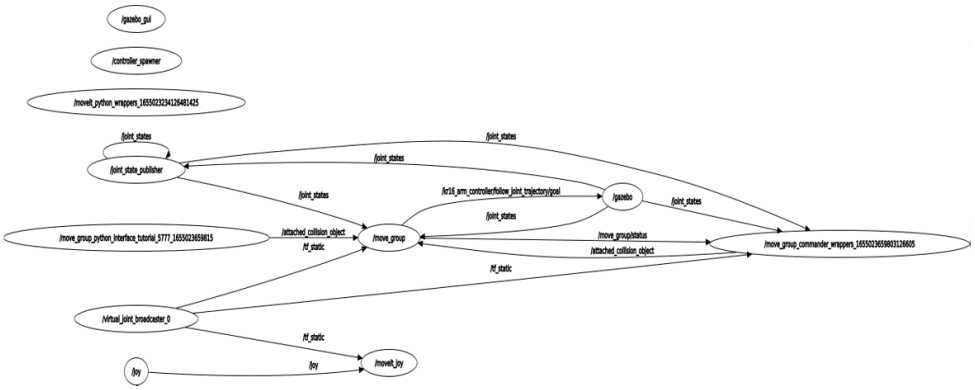
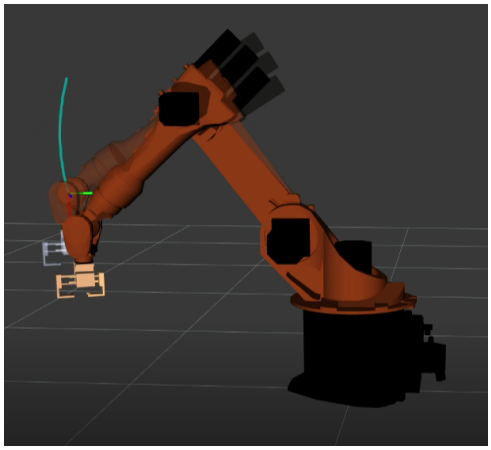


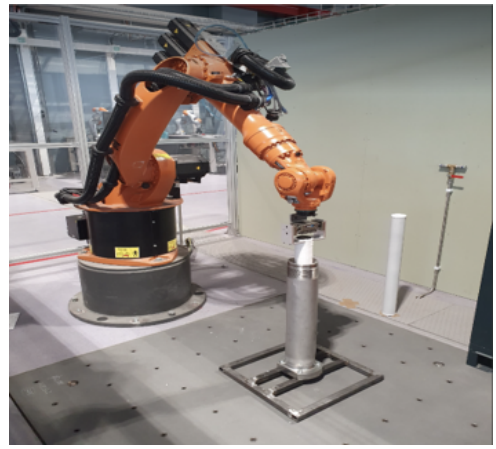

**Figure 5.2.:** Communication of nodes

## 5.2. Trajectories

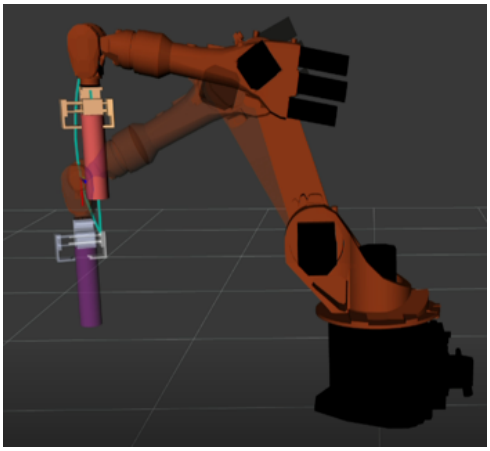
This section presents the generated trajectories from the tests conducted. The trajectories are visualized in the visualisation tool RViz, where the figures presented are a representation of the filter change. It also shows the robot in the laboratory performing the motions visualised. The transparent robot shows the path followed, while the other robot shows when it is at the final position.



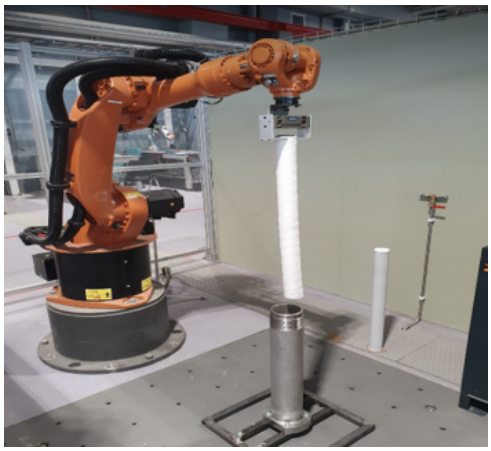
**(a)** *Home* to $P1$



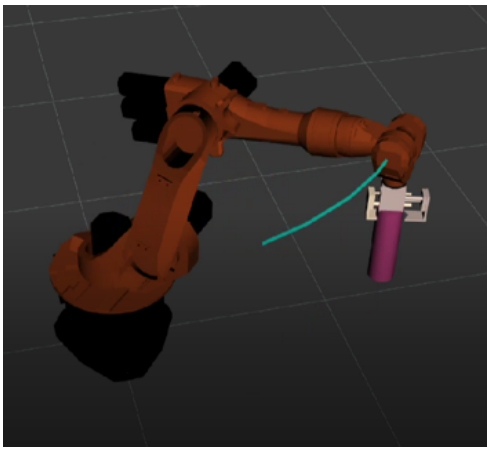
**(b)** *Home* to $P1$ in the laboratory



**(c)** $P1$ to $P2$

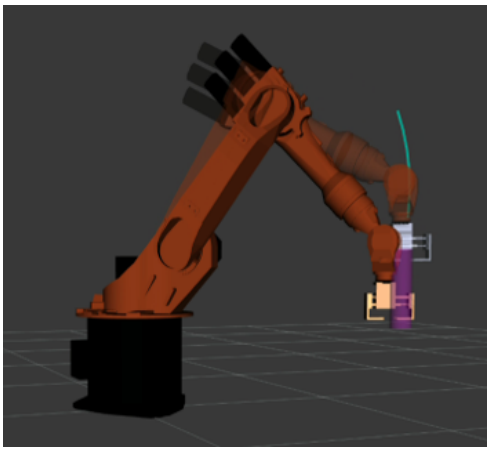


**(d)** $P1$ to $P2$ in the laboratory



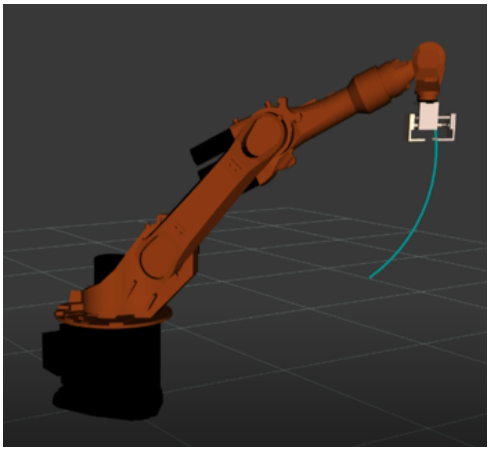
**(e)** $P2$ to $P3$

**(f)** $P2$ to $P3$ in the laboratory



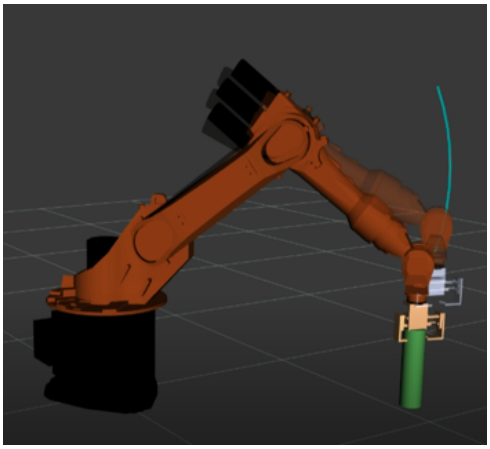
**(g)** $P3$ to $P4$

**(h)** $P3$ to $P4$ in the laboratory



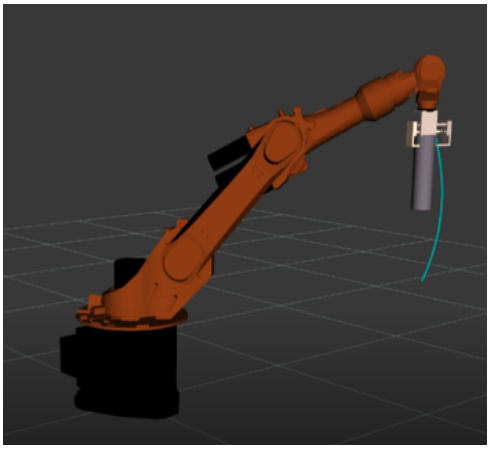
**(i)** $P4$ to $P5$

**(j)** $P4$ to $P5$ in the laboratory

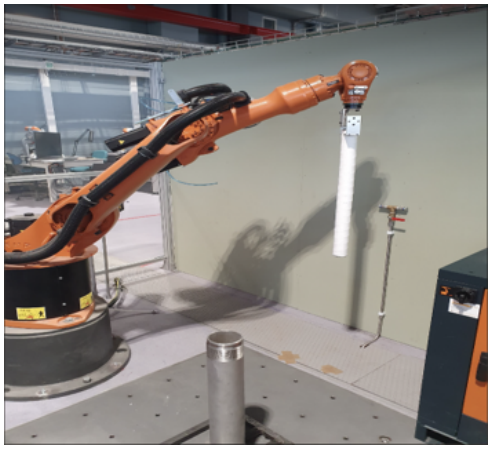


**(k)** $P5$ to $P6$

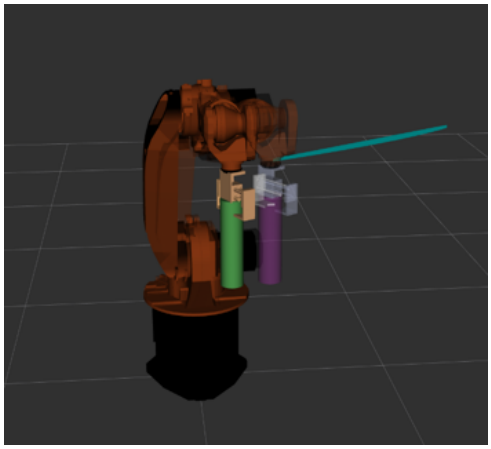
**(l)** $P5$ to $P6$ in the laboratory



**(m)** $P6$ to $P7$

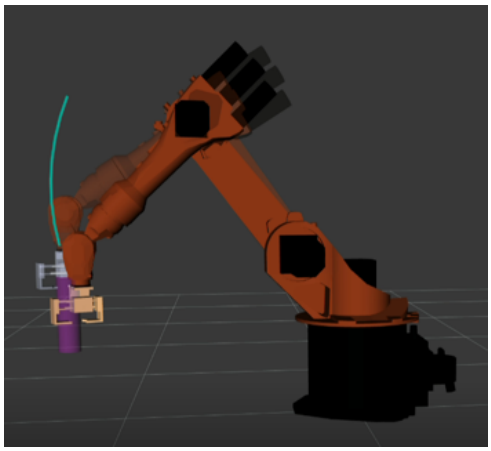


**(n)** $P6$ to $P7$ in the laboratory

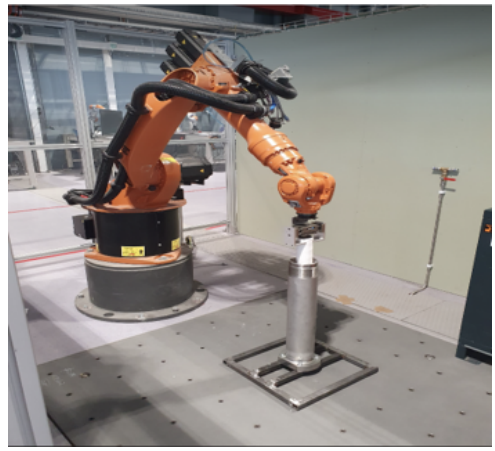


**(o)** *P*7 to *P*8

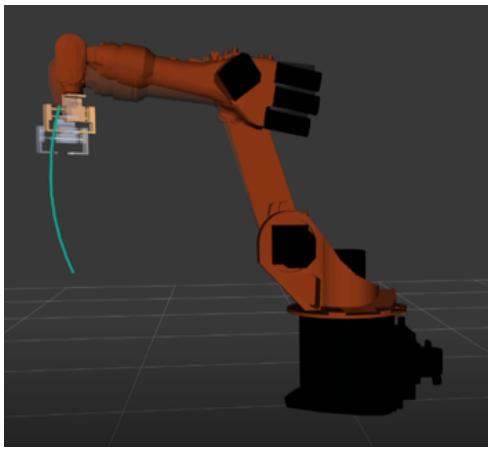
**(p)** *P*7 to *P*8 in the laboratory



**(q)** *P*8 to *P*9



**(r)** *P*8 to *P*9 in the laboratory



**(s)** *P*9 to *Home*

**(t)** *P*9 to *Home* in the laboratory

**Figure 5.3.:** Trajectories visualized in RViz & robot executing in the laboratory

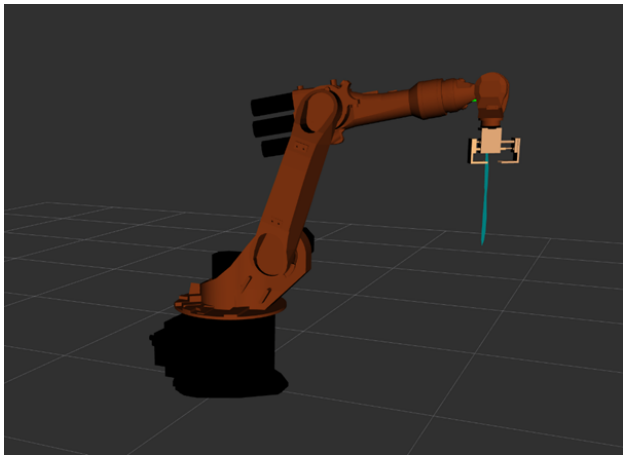


**Figure 5.4.:** Trajectory generated with a sequence of points

The following points shown in Table 4.4, and Table 4.5, are new points added to the cycle with reference to Figure 5.4.

| New points | | | | | | |
|---|---|---|---|---|---|---|
| Positions | $\theta_1$ | $\theta_2$ | $\theta_3$ | $\theta_4$ | $\theta_5$ | $\theta_6$ |
| *new_point*1 | -4° | -71° | 82 | 180 | -78 | -68 |
| *new_point*2 | -4 | -65 | 90 | 180 | -65 | -68 |
| *new_point*3 | -4 | -54 | 95 | 180 | -49 | -68 |
| *new_point*4 | -4 | -51 | 95 | 180 | -45 | -68 |

**Table 5.1.:** Sequence of new points(joint angles)

| New points | | | |
|---|---|---|---|
| Positions | $x$ | $y$ | $z$ |
| *new_point*1 | 1.13 | 0.08 | 1.16 |
| *new_point*2 | 1.14 | 0.08 | 0.98 |
| *new_point*3 | 1.14 | 0.08 | 0.76 |
| *new_point*4 | 1.14 | 0.08 | 0.71 |

**Table 5.2.:** Sequence of new points(Cartesian)

## 5.3. Plots

The following plots show the position, velocity, and accelerations when the filter is being inserted into its container. The plots are with reference to movement from P8 to P9 directly, and with a sequence of points between P8 and P9, which is seen on Figure 5.3q, and Figure 5.3r.

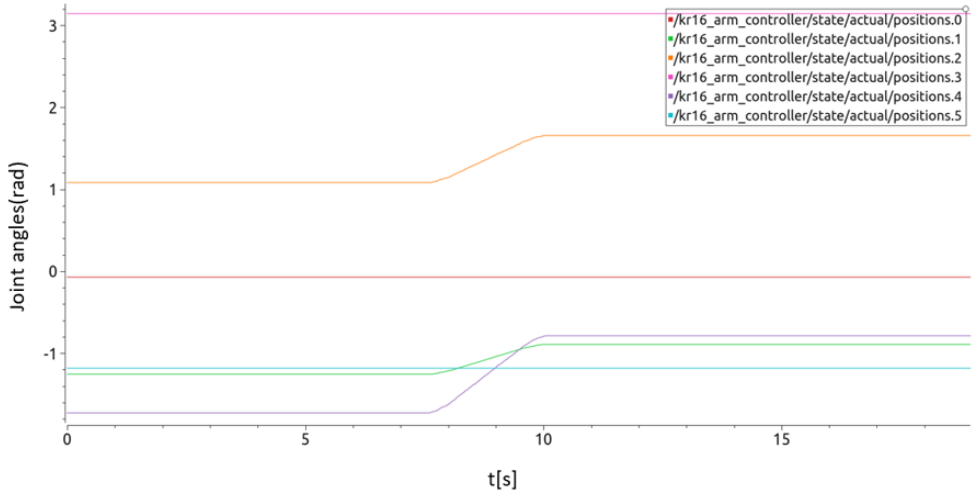



**Figure 5.5.:** Position of joints from P8 to P9(directly)

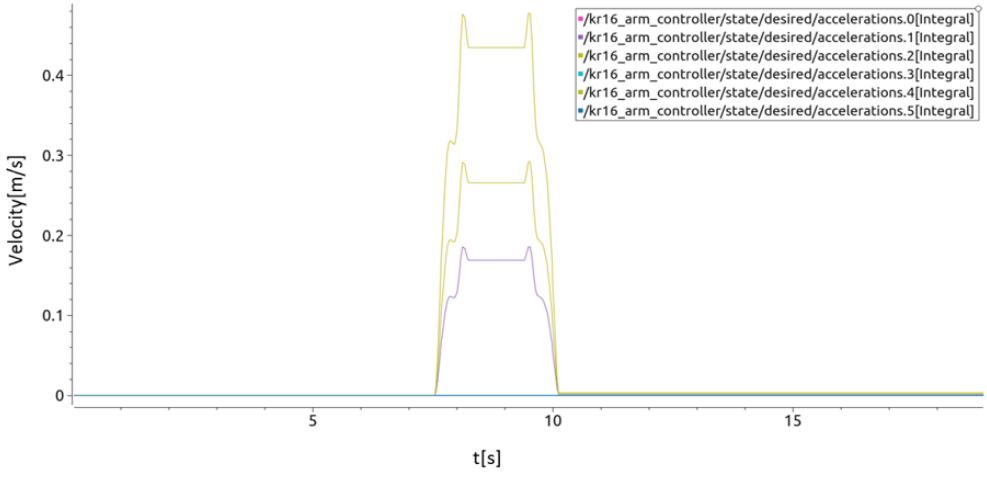



**Figure 5.6.:** Velocity from P8 to P9(directly)

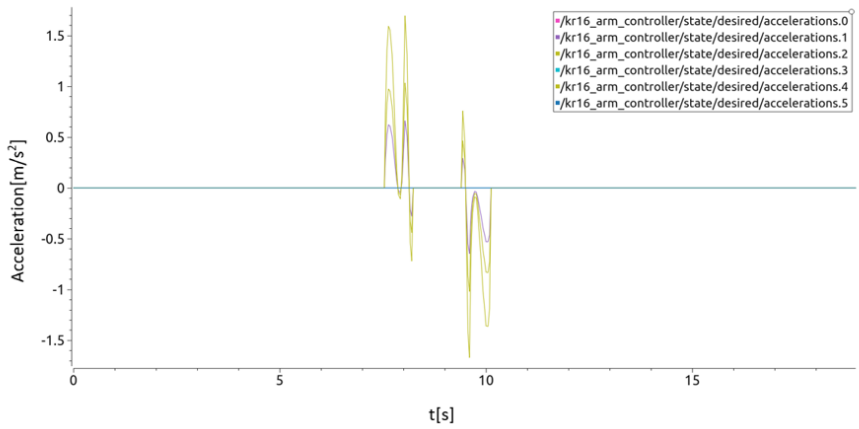



**Figure 5.7.:** Acceleration from P8 to P9(directly)

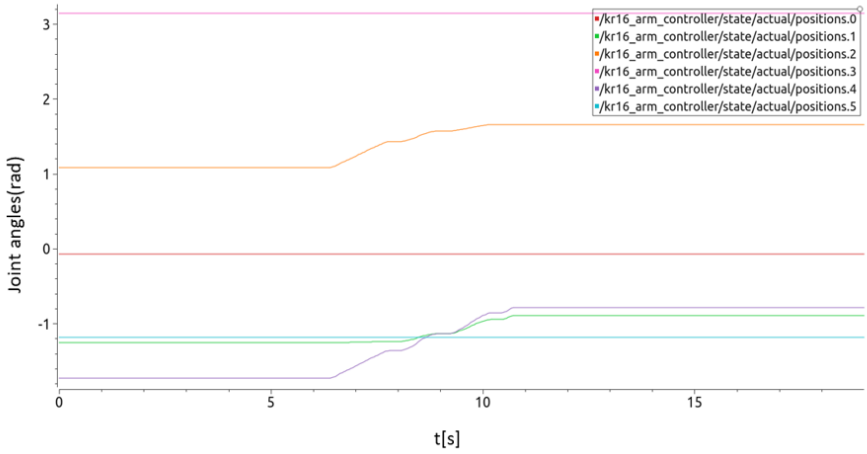



**Figure 5.8.:** Position of joints from P8 to P9(sequence)

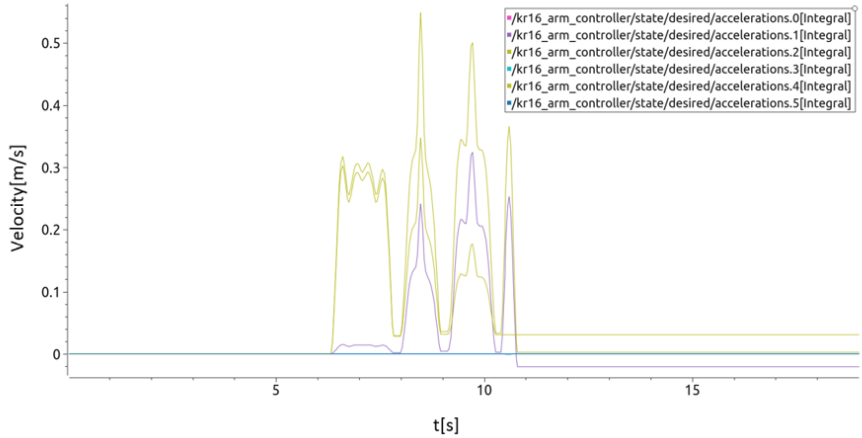



**Figure 5.9.:** Velocity from P8 to P9(sequence)

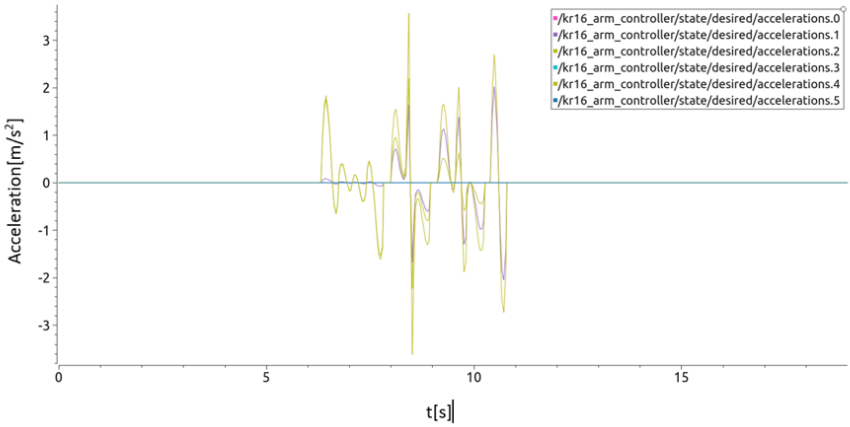



**Figure 5.10.:** Acceleration from P8 to P9(sequence)

# Chapter 6.

# Discussion

This chapter presents a discussion of the results showcased in the previous chapter.

## 6.1. Teleoperation

Teleoperation with the joystick in the laboratory did not get tested, unfortunately. My lab partner and I had some issues connecting the joystick that we could not resolve, which led us to focus more on the filter change directly. As the robot can be moved with joystick and mouse in ROS, it was tested with the mouse in the laboratory to check how the control of the robots end-effector would take place. The end-effector was controlled with the computer mouse through RViz, and after a position was confirmed, a visualisation of the movement could be seen in RViz before it was executed on the real robot. This was done to investigate how close we could get to the real positions, since we can monitor the joint values in the RViz GUI. It was also done with the developed Python code, which includes all the accurate positions of the pick-and-place cycle. This method resulted in a very tedious process, since it is very cumbersome to control the robot only through its end-effector. The other joints of the robot strictly follow the end-effector, making it difficult to get a correct position of the individual joints.

Regarding the issue of time-delay in a teleoperation system, our system does not have time-delay as it is directly connected to the robot, but when connected to the joystick in ROS, the issue of time-delay occurs. A digital attachment is added to this thesis, which shows a recording of the delay when the robot is controlled with a joystick. The upper left terminal shows the joystick movements, while the lower terminal shows the developed script running, and allows for commands input. This delay indicates how the delay can occur over long distances between the master device and remote robot. Since we are visualizing the robot's movement in RViz , we can display future movements of the robot before confirming the

motion. This acts as a predictive display where the operator can see how the robots motion will take place, before it is initiated. This is a benefit regarding the delay between the master device and the robot. A known strategy when the delays are very noticeable is the "move-and-wait" strategy, where the operator moves the robot, waits to see the response, then moves again. This solution takes a lot of time to position the robot to its correct location since the operator must stop the robot every time before moving it, and it also requires additional concentration which can be very demanding.

## 6.2. Trajectories

With reference to the figures presented in Section 5.2, the `joint_space` node is run through the terminal in ROS and shows the visualized trajectories the end-effector follows during task execution, where MoveIt interpolates with a $n \geq 5$ polynomial, as in (2.22). From 5.3c, it can be seen that the path the end-effector follows when picking up the filter is curved. This is because two points are defined, i.e., the end-effector's initial and final position. The problem with predetermined points is that the system becomes "stiff", and small changes to the setup can cause damage to the filter. Since the robot directly follows the joint angle configurations specified in the node, it can be exposed to precision errors.

The first tests included only the set points shown in Table 4.4 to determine if the filter can be withdrawn from its container without causing damage. As mentioned in Section 4.2.3, the complete assembly includes a filter rod inserted inside the filter. The filter rod is dependent on high accuracy when inserted inside the container, but since the bottom of the container is sealed, it led to the filter rod leaning towards the edge of the container after the first filter was picked up. This can have severe consequences if the robot misses its target when inserting the new filter. As the paths generated were first visualized in RViz, it indicated that the filter would most likely hit the inside of the container. For these reasons, the filter rod was excluded during the tests. Figure 6.1, shows an example of the mentioned problem. The filter was not positioned correctly, which caused it to gently touch the outer edge of its container, resulting in damaging the filter. The filter rod was fortunately removed prior to this incident, but the same outcome would occur.

Another issue that caused precision error was the gripper. The gripper was re-designed with a circular shape to fit the filters, but it did not grasp the filter enough to keep it stable. It is also noted that it can be caused by too high velocity, especially for movement from P7 to P8(Figure 5.3o), as this is the longest travel distance and represents the position before the filter is placed inside its container. This caused the bottom side of the filters to sway, which meant that it had to be positioned manually right before it was taken down in the container

if the error was very noticeable.

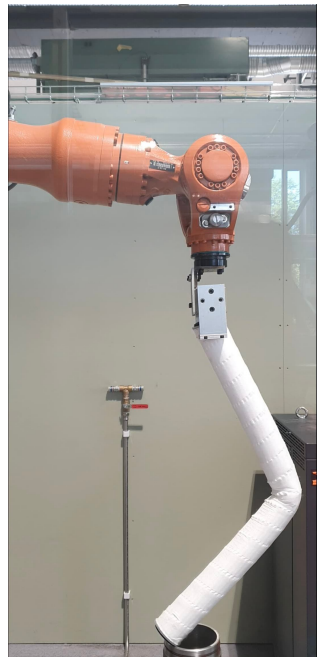


**Figure 6.1.:** Failure due to precision error

A sequence of points was applied between the initial and final point of the end-effector path in order for the filter to follow a linear path. The representation of the path can be seen visualized in Figure 5.4. Four points were assigned between the initial and final position, which can be seen in Table 4.4, and Table 4.5. By asserting these points, it made it easier to see if the new filter would miss the filter container, resulting in a more precise extraction of the old filter. These points were applied for the most crucial motions, i.e., picking up the old filter(P1 to P2) and inserting the new one(P8 to P9).

## 6.3. Plots

The plots presented in Section 5.3 show the movement of the joints(in radians), their velocity, and acceleration from point P8 to P9. Only these plots were presented as this is considered the most critical part of the filter change and represents the robot's motion when inserting the new filter. The desired position of the joints and the actual position of the joints were identical for all of the points in the cycle when tested in the laboratory; hence these plots were not included.

The first three plots, Figure 5.5, Figure 5.6, and Figure 5.7, show the position,

velocity, and acceleration from *Home* directly to P9, when only initial and final end-effector points are given. While the other three figures are related to the same start position *Home*, but with the extra points from Table 4.4 asserted. It can be observed that the maximum velocity when the robot moves directly to the filter container is 0.47 m/s, while for a sequence of points between P8 and P9, it is 0.54 m/s. The movement can be seen where the velocity increases towards the point, stops and does the same for the remaining points. If something were to happen between these points, it would be more noticeable as it stops and then moves again.

When the program was tested in the laboratory, we had issues that led to the robot stopping, and the program had to be rerun. The robots maximum velocity in T1 mode is set to 250 mm/s, which is 0.25 m/s, and if the velocity exceeds this maximum, the robot controller(KRC) will enter a protective stop. From Figure 5.6 and Figure 5.9, it can be seen that the velocity is above the maximum allowed velocity, which led us to switch from T1 mode to T2, as T2 has no velocity limits. The filter change was easier to monitor after the new points were added to the cycle since one of the points first aligns with the center of the container before the filter is inserted more linear as opposed to the curved path.

# Chapter 7.

# Conclusions and Future Work

## 7.1. Conclusion

This thesis began by introducing teleoperation and its common architectures before presenting the kinematics and the software tools needed to implement a teleoperation control architecture.

This thesis aimed to conduct a filter change through the means of teleoperation. Chapter 4 presented the control architecture and the different nodes implemented to get a simulation in Gazebo, as well as the possibility of teleoperating the KUKA KR-16 robot in RViz. A schematic of the control architecture was also presented, which shows the communication between the different nodes implemented. Using the open-source motion planning framework MoveIt, simplified many of the underlying processes to get a working setup of the robot. MoveIts setup assistant was used to create a configuration of the robot together with the gripper made in SolidWorks.

Direct control through the robot's teach pad(KCP) was used in the laboratory to get the exact positions for the filter change, and the whole cycle was implemented with a Python script. The teleoperation system allowed for end-effector control through MoveIts `move_group` node and was successfully configured in ROS, but complications with connection to the joystick made it unsuccessful in the laboratory. The implemented script, together with a computer mouse, allowed for command input of the complete filter change and teleoperation of the robot in RViz. Some elements led to simplifying the filter change, such as not including the filter container's top lid, since the gripper does not have the ability to grasp it, and removal of the filter rod, which is initially placed inside the filter.

The test performed showed that the filter change requires high precision as there is only a 25 mm gap between the filter and its container. Teleoperation of the KUKA KR-16 robot was performed through the movement of the end-effector,

which leads to the other joints following as these are chained in the configuration of the robot. This led to using the developed python code to pick up the old filter and place the new filter inside the container, while movements that did not require any precision were teleoperated with a computer mouse. Additional points were added between the initial and final end-effector positions, which led to more precise control of the insertion and extraction of the filter. Teleoperation with the joystick in ROS was prone to delay when controlled in RViz. This delay led to a *move-and-wait* strategy to perform the wanted end-effector movements.

Incorporating teleoperation for the filter change has the potential to increase both health and safety for the remote operator, and the need to enhance the operator's awareness during task execution is evident. As the project is still in its very early stage, some suggestions to further develop the system are presented in the following section

## 7.2. Future work

The project of conducting the filter change includes many aspects that should be further investigated to make the system more automated. One suggestion is to implement a camera system to give the operator more feedback. Since the developed script in this thesis moves the robot to a predetermined point, it causes the system to be non-compliant and stiff. With a mounted camera on, e.g., the robot gripper, the position of the filters can be obtained through object detection. There are plenty of developed scripts for object detection through the *Open Source Computer Vision Library*(OpenCV). OpenCV is a free open source computer vision and machine learning software library that includes over 2500 algorithms[36]. ArUco markers[37] can be placed on the filters to detect their position and orientation and will make the insertion of the new filter more manageable.

The gripper used was not intended for this project, which makes developing a new gripper highly recommended for the continuation of this project. The new gripper should be able to include dismounting of the filter container's top lid. This motion will require the gripper to withstand the forces needed to unscrew the lid, as well as the ability to pick it up by the cylinder hinge and place it at a drop-off location. Grasping filters in the simulation was simplified since the gripper did not have functionality in the laboratory. The simplification was done in ROS by attaching the filter on the gripper, but with a new gripper, the `moveit_grasping` [31] packages can be implemented to generate grasps of the filter.

The teleoperation aspect can be further developed by kinematically coupling the joystick and the robot so that individual joints of the robot can be controlled, not

just the end-effector as in this thesis. The teleoperation of a stationary robot will also differ from a mobile robot, which is the intended usage for the actual task on an unmanned platform. The laboratory has mobile robots available, which can be used to continue the project.

# References


[1] Sotiris Avgousti, Eftychios G Christoforou, Andreas S Panayides, Sotos Voskarides, Cyril Novales, Laurence Nouaille, Constantinos S Pattichis, and Pierre Vieyres. "Medical telerobotic systems: current status and future trends". In: *Biomedical engineering online* 15.1 (2016), pp. 1–44.

[2] Luis Basañez and Raúl Suárez. "Teleoperation". In: *Springer Handbook of Automation*. Ed. by Shimon Y. Nof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 449–468. ISBN: 978-3-540-78831-7. DOI: 10.1007/978-3-540-78831-7_27. URL: https://doi.org/10.1007/978-3-540-78831-7_27.

[3] Ryan A Beasley. "Medical robots: current systems and research directions". In: *Journal of Robotics* 2012 (2012).

[4] Henri Boessenkool, David Abbink, Cock Heemskerk, Frans van der Helm, and Jeroen Wildenbeest. "A Task-Specific Analysis of the Benefit of Haptic Shared Control During Telemanipulation". In: *IEEE Transactions on Haptics* 6 (May 2012), pp. 2–12. DOI: 10.1109/ToH.2012.22.

[5] Giuseppe Oriolo Bruno Siciliano Lorenzo Sciavicco Luigi Villani. *Robotics, Modelling planning and control*. Springer, 2009. ISBN: 978-1-84628-641-4.

[6] David B Camarillo, Thomas M Krummel, and J Kenneth Salisbury Jr. "Robotic technology in surgery: past, present, and future". In: *The American Journal of Surgery* 188.4 (2004), pp. 2–15.

[7] Dr. Thomas L. Harman Carol Fairchild. *ROS Robotics By Example - Second Edition*. Last accessed 11 November 2021. 2017. URL: https://subscription.packtpub.com/book/hardware_and_creative/9781788479592/1/%5C%5Cch01lvl1sec13/ros-nodes-topics-and-messages.

[8] Cern. *What is cerns misson?* Last accessed 20 May 2022. 2022. URL: https://home.cern/about/who-we-are/our-mission.

[9] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo. "ros_control: A generic and simple control framework for ROS". In: *The*

*Journal of Open Source Software* (2017). DOI: 10.21105/joss.00456. URL: http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf.

[10] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. "Reducing the barrier to entry of complex robotic software: a moveit! case study". In: *arXiv preprint arXiv:1404.3785* (2014).

[11] Hao Deng, Jing Xiong, and Zeyang Xia. "Mobile manipulation task simulation using ROS with MoveIt". In: *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2017, pp. 612–616. DOI: 10.1109/RCAR.2017.8311930.

[12] Eurobots. *Kuka kr16-2*. Last accessed 27 May 2022. 2022. URL: https://www.eurobots.net/kuka-robots-kr-16-2-p233-en.html.

[13] KUKA Roboter GmbH. *KUKA.RobotSensorInterface 2.3*. KST RSI 2.3 V1 en. 2009.

[14] Raymond C Goertz. "Fundamentals of general-purpose remote manipulators". In: *Nucleonics* 10.11 (1952), pp. 36–42.

[15] Raymond C Goertz. "Mechanical master-slave manipulator". In: *Nucleonics (US) Ceased publication* 12 (1954).

[16] Mathias Hoeckelmann, Imre J Rudas, Paolo Fiorini, Frank Kirchner, and Tamas Haidegger. "Current capabilities and development potential in surgical robotics". In: *International Journal of Advanced Robotic Systems* 12.5 (2015), p. 61.

[17] Peter F. Hokayem and Mark W. Spong. "Bilateral teleoperation: An historical survey". In: *Automatica* 42.12 (2006), pp. 2035–2057. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2006.06.027. URL: https://www.sciencedirect.com/science/article/pii/S0005109806002871.

[18] Thomas Hulin, Michael Panzirsch, Harsimran Singh, Andre Coelho, Ribin Balachandran, Aaron Pereira, Bernhard M. Weber, Nicolai Bechtel, Cornelia Riecke, Bernhard Brunner, Neal Y. Lii, Julian Klodmann, Anja Hellings, Katharina Hagmann, Gabriel Quere, Adrian S. Bauer, Marek Sierotowicz, Roberto Lampariello, Jörn Vogel, Alexander Dietrich, Daniel Leidner, Christian Ott, Gerd Hirzinger, and Alin Albu-Schäffer. "Model-Augmented Haptic Telemanipulation: Concept, Retrospective Overview, and Current Use Cases". In: *Frontiers in Robotics and AI* 8 (2021), p. 76. ISSN: 2296-9144. DOI: 10.3389/frobt.2021.611251. URL: https://www.frontiersin.org/article/10.3389/frobt.2021.611251.

[19] Bahadur Ibrahimov. "A cost-oriented robot for the Oil Industry". In: *IFAC-PapersOnLine* 51 (July 2018), pp. 204–209. DOI: https://doi.org/10.1016/j.ifacol.2018.11.287.

[20] Taurob robotic inspector. *Taurob*. Last accessed 3 December 2021. 2021. URL: https://taurob.com/taurob-inspector/.

[21] Mustafe Kahin. "Robot Telemanipulation for Remote Maintenance". In: (Dec. 2021), pp. 1–39.

[22] Frank C. Park Kevin M. Lynch. *Mechanics,Planning, and Control*. Cambridge Univeristy Press, 2017. ISBN: 978-1-1071-5630-2.

[23] Serdar Kucuk and Z. Bingul. "The inverse kinematics solutions of industrial robot manipulators". In: July 2004, pp. 274–279. ISBN: 0-7803-8599-3. DOI: 10.1109/ICMECH.2004.1364451.

[24] Jonathan Cacace Lentin Joseph. *Mastering ROS for Robotics Programming*. 2nd. Packt, 2018. ISBN: 9781788478953.

[25] Lonnie J Love, David P Magee, and Wayne John Book. "A comparison of joint control algorithms for teleoperated pick and place tasks using a flexible manipulator". In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Vol. 2. IEEE. 1994, pp. 1257–1262.

[26] Jing Luo, Wei He, and Chenguang Yang. "Combined Perception, Control, and Learning for Teleoperation: Key Technologies, Applications, and Challenges". In: *Cognitive Computation and Systems* 2 (Mar. 2020). DOI: 10.1049/ccs.2020.0005.

[27] Mathworks. *Exchange Data with ROS Publishers and Subscribers*. Last accessed 11 November 2021. 2021. URL: https://in.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html.

[28] Claudio Melchiorri. "Robotic telemanipulation systems: an overview on control aspects". In: *IFAC Proceedings Volumes* 36.17 (2003). 7th IFAC Symposium on Robot Control (SYROCO 2003), Wroclaw, Poland, 1-3 September, 2003, pp. 21–30. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)33365-7. URL: https://www.sciencedirect.com/science/article/pii/S1474667017333657.

[29] MoveIt. *Concepts*. Last accessed 20 May 2022. 2022. URL: https://moveit.ros.org/documentation/concepts/.

[30] MoveIt. *Joystick Control Teleoperation*. Last accessed 3 June 2022. 2022. URL: http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/joystick_control_%5C%5Cteleoperation/joystick_control_teleoperation_tutorial.html.

[31] MoveIt. *MoveIt grasps*. Last accessed 29 May 2022. 2022. URL: https://ros-planning.github.io/moveit_tutorials/doc/moveit_grasps/moveit_grasps_tutorial.html.

[32] MoveIt. *MoveIt Setup Assistant*. Last accessed 20 May 2022. 2022. URL: https://ros-planning.github.io/moveit_tutorials/doc/setup_assistant/setup_assistant_tutorial.html.

[33] Moveit. *Movegroup Python Interface Tutorial*. Last accessed 20 May 2022. 2022. URL: https://github.com/ros-planning/moveit_tutorials/blob/master/doc/move_group_python_interface/scripts/move_group_python_interface_tutorial.py.

[34] Günter Niemeyer, Carsten Preusche, and Gerd Hirzinger. "Telerobotics". In: vol. 25. May 2008, pp. 741–757. DOI: 10.1007/978-3-540-30301-5_32.

[35] Günter Niemeyer, Carsten Preusche, Stefano Stramigioli, and Dongjun Lee. "Telerobotics". In: *Springer handbook of robotics*. Springer, 2016, pp. 1085–1108.

[36] OpenCV. *About*. Last accessed 29 May 2022. 2022. URL: https://opencv.org/about/.

[37] OpenCV. *Detection of ArUco Markers*. Last accessed 29 May 2022. 2022. URL: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[38] OROCOS. *Orocos Kinematics and Dynamics*. Last accessed 20 May 2022. 2022. URL: https://www.orocos.org/kdl.html.

[39] Lynne E Parker and John V Draper. "Robotics applications in maintenance and repair". In: *Handbook of industrial robotics* 2 (1998), pp. 1023–1036.

[40] Luis F. Peñín and Kotaro Matsumoto. "Teleoperation with time delay: A survey and its use in space robotics". In: 2002.

[41] Péter Dr. Tamás Péter Dr. Korondi János Halas Krisztián Dr. Samu Attila Bojtos. *Internet based telemanipulation*. Last accessed 14 May 2022. 2021. URL: https://mogi.bme.hu/TAMOP/robot_applications/ch04.html#ch-4.2.

[42] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1449323898.

[43] RobotWorks. *KUKA Low Payload Robot Series*. Last accessed 27 May 2022. 2022. URL: https://www.robots.com/robots/kuka-kr-16.

[44] ros-industrial. *Kuka-experimental*. Last accessed 30 April 2022. 2021. URL: https://github.com/ros-industrial/kuka_experimental.

[45] ROS.org. *Configuring and Using a Linux-Supported Joystick with ROS*. Last accessed 3 June 2022. 2022. URL: http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick.

[46] Ros.org. *Documentation*. Last accessed 20 May 2022. 2022. URL: http://wiki.ros.org/ros_control.

[47] Ros.org. *Ros control*. Last accessed 11 November 2021. 2021. URL: http://wiki.ros.org/ros_control.

[48] ROSwiki. *ROS Noetic Ninjemys*. Last accessed 26 May 2022. 2021. URL: http://wiki.ros.org/noetic.

[49] Clare Saliba, Marvin K Bugeja, Simon G Fabri, Mario Di Castro, Alessandro Mosca, and Manuel Ferre. "A Training Simulator for Teleoperated Robots Deployed at CERN." In: *ICINCO (2)*. 2018, pp. 293–300.

[50] M. Selvaggio, P. Robuffo Giordano, F. Ficuciello, and B. Siciliano. "Passive Task-Prioritized Shared-Control Teleoperation with Haptic Guidance". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 430–436. DOI: 10.1109/ICRA.2019.8794197.

[51] T.B. Sheridan. "Telerobotics". In: *Automatica* 25.4 (1989), pp. 487–507. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(89)90093-9. URL: https://www.sciencedirect.com/science/article/pii/0005109889900939.

[52] Charlotte Skourup, John Pretlove, Neil Stembridge, and Mona Svenes. "Enhanced awareness for offshore teleoperation". In: *Intelligent Energy Conference and Exhibition*. OnePetro. 2008.

[53] Mark W Spong, Seth Hutchinson, and M Vidyasagar. "Robot Dynamics and Control". In: (2004).

[54] Ioan A. Sucan and Sachin Chitta. *MoveIt*. Last accessed 16 November 2021. 2021. URL: https://moveit.ros.org/.

[55] James Trevelyan, William R. Hamel, and Sung-Chul Kang. "Robotics in Hazardous Applications". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Cham: Springer International Publishing, 2016, pp. 1521–1548. ISBN: 978-3-319-32552-1. DOI: 10.1007/978-3-319-32552-1_58. URL: https://doi.org/10.1007/978-3-319-32552-1_58.

[56] Yulun Wang, Steven E Butner, and Ara Darzi. "The developing market for medical robotics". In: *Proceedings of the IEEE* 94.9 (2006), pp. 1763–1771.

# Appendix A.

# Digital Attachments

The thesis includes three digital attachments:

1. A video of the filter change
2. A video of teleoperation in RViz
3. The specialization project written prior to this thesis [21]

## A.1. Running the teleoperation system in ROS

The developed system is implemented with ROS Noetic and Ubuntu 20.04.4 LTS (Focal Fossa). Follow the instructions to install ROS Noetic, and ensure that the latest packages are installed.

Create a new catkin workspace :
$ `source /opt/ros/noetic/setup.bash`
$ `mkdir -p ~/catkin_ws/src`
$ `cd ~/catkin_ws/`
$ `catkin_make`

Clone the repository to the src folder:
$ `cd /path/to/catkin_ws/src`
$ `git clone https://github.com/mustafekahin/Joystickteleop.git`
Check dependencies :
$ `cd ..`
$ `rosdep install --from-paths src --ignore-src`
Build the workspace:
$ `catkin_make`

### A.1.1. Teleoperation with a joystick

To run the implemented setup, open a new terminal and start an instance of roscore:

```
$ roscore
```
Open another terminal and source your workspace:

```
$ cd /path/to/catkin_ws/
$ source devel/setup.bash
$ roslaunch kuka_kr16_moveit_config demo_gazebo.launch use_gui:=false
```

RViz and Gazebo will now show the robot. To manipulate the robot with a mouse, simply change the planning group from *gripper*, to *kr16_arm*.

To use a joystick first follow the instructions to get a proper setup from the ROS.org page [45]. In RViz, check off the *External Comm*, and open up a new terminal.
In the new terminal:

```
$ cd /path/to/catkin_ws/
$ source devel/setup.bash
$ roslaunch kuka_kr16_moveit_config joystick_control.launch
dev:=/dev/input/js2
```

Be aware that specification of the joystick input `dev:=/dev/input/js2` varies, in my case, the joystick is input js2. The joystick can now be used to control the robot in RViz, and the mappings from controller to robot can be seen in Table 4.1.

To use the `joint_space_node` open up a new terminal, and type the following :

```
$ cd /path/to/catkin_ws/
$ source devel/setup.bash
$ cd src/Joystickteleop/kr16_control/src
```
Create an executable of jointspace_node.py:

```
$ cd chmod +x jointspace_node.py
$ python3 joint_space_node.py
```

The code can be found in the kr16_control package, where all the inputs can be seen. The complete system now allows for both teleoperation with the joystick, and command input in the terminal.

### A.1.2. Setup in the laboratory

The setup in the laboratory is ran on another workspace, and differs in the configuration. To run the setup in the laboratory, simply clone the following repository in your workspace:

$ `cd /path/to/catkin_ws/`
$ `git clone https://github.com/mustafekahin/Laboratory.git`
Check dependencies :
$ `rosdep install --from-paths src --ignore-src`
Build the workspace:
$ `catkin_make`
Run the following:
$ `source devel/setup.bash`
$ `roslaunch kuka_kr16_support start_robot.launch sim:=false`
To add the robot in RViz, simply press add in the RViz GUI and in the pop-up window select MotionPlanning.

To run the `joint_space_node` type the following in a new terminal:
$ `cd /path/to/catkin_ws/`
$ `source devel/setup.bash`
$ `roslaunch kuka_kr_16_support code.launch`

## A.2. Controllers

**Listing A.1:** Main controllers

```
kr16_arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:

      - joint_a1
      - joint_a2
      - joint_a3
      - joint_a4
      - joint_a5
      - joint_a6
  gains:
      joint_a1: {p: 1000.00, i: 0, d: 0}
      joint_a2: {p: 1000.00, i: 0, d: 0}
```

```
      joint_a3: {p: 1000.00, i: 0, d: 0}
      joint_a4: {p: 1000.00, i: 0, d: 0}
      joint_a5: {p: 1000.00, i: 0, d: 0}
      joint_a6: {p: 1000.00, i: 0, d: 0}
  constraints:
       goal_time: 2.0
  state_publish_rate: 25

gripper_controller:
    type: position_controllers/JointTrajectoryController
    joints:
        - left_gripper_joint
        - right_gripper_joint

    gains:
        left_gripper_joint: { p: 5, d: 3.0, i: 0, i_clamp: 1 }
        right_gripper_joint: { p: 5, d: 3.0, i: 0, i_clamp: 1 }

    state_publish_rate: 25

controller_list:
  - name: kr16_arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - joint_a1
      - joint_a2
      - joint_a3
      - joint_a4
      - joint_a5
      - joint_a6
  - name: gripper_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - left_gripper_joint
      - right_gripper_joint
```

## A.3. Denavit Hartenberg implementation

By using the Denavit Hartenberg convention presented in Section 2.4.6, the following matrices are computed. The matrices represent the final transformation(from base to end-effector), where the last column shows the position of the end-effector in operational space. T1 refers to the transformation of P1, which is the first position of the pick-and-place cycle, and so on.

$$T_1 = \begin{bmatrix} -0.45 & -0.89 & 0.01 & 1.14 \\ -0.89 & 0.45 & 0 & -0.07 \\ 0 & -0.01 & -1 & 0.71 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} -0.45 & -0.89 & 0.01 & 1.13 \\ -0.89 & 0.45 & 0 & -0.07 \\ 0 & -0.01 & -1 & 1.40 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} -0.94 & -0.32 & 0.02 & 0.73 \\ -0.32 & 0.94 & -0.02 & -0.81 \\ -0.01 & -0.03 & -1 & 1.41 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_4 = \begin{bmatrix} -0.94 & -0.32 & 0 & 0.75 \\ -0.32 & 0.94 & 0 & -0.83 \\ 0 & -0 & -1 & 0.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5 = \begin{bmatrix} -0.94 & -0.32 & 0 & 0.98 \\ -0.32 & 0.94 & -0 & -0.88 \\ 0 & 0 & -1 & 1.45 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_6 = \begin{bmatrix} -0.94 & -0.32 & 0 & 0.98 \\ -0.32 & 0.94 & 0 & -0.88 \\ 0 & -0 & -1 & 0.63 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_7 = \begin{bmatrix} -0.94 & -0.32 & 0 & 0.98 \\ -0.32 & 0.94 & -0 & -0.88 \\ 0 & 0 & -1 & 1.45 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_8 = \begin{bmatrix} -0.90 & -0.42 & 0.01 & 0.84 \\ -0.42 & 0.90 & -0.01 & -0.76 \\ 0 & -0.01 & -1 & 1.40 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_9 = \begin{bmatrix} -0.45 & -0.89 & 0.01 & 1.14 \\ -0.89 & 0.45 & -0 & -0.07 \\ 0 & -0.01 & -1 & 0.71 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{home} = \begin{bmatrix} -0.45 & 0.01 & 0.01 & 1.13 \\ -0.89 & 0.45 & 0.0 & -0.07 \\ 0 & -0.01 & -1 & 1.40 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

| Joint name | Parent Link | Child Link | $x$( m) | $y$( m) | z(m) | roll | pitch | yaw |
|---|---|---|---|---|---|---|---|---|
| Joint 1 | Base link | Link 1 | 0 | 0 | 0.675 | 0 | 0 | 0 |
| Joint 2 | Link 1 | Link 2 | 0.26 | 0 | 0 | 0 | 0 | 0 |
| Joint 3 | Link 2 | Link 3 | 0.68 | 0 | 0 | 0 | 0 | 0 |
| Joint 4 | Link 3 | Link 4 | 0.67 | 0 | −0.035 | 0 | 0 | 0 |
| Joint 5 | Link 4 | Link 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| Joint 6 | Link 5 | Link 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| Gripper joint | Link 6 | Gripper link | 0.158 | 0 | 0 | 0 | 1.5707 | 0 |

**Table A.1.:** KR16 URDF in table format

## A.4. Hierarchical graph

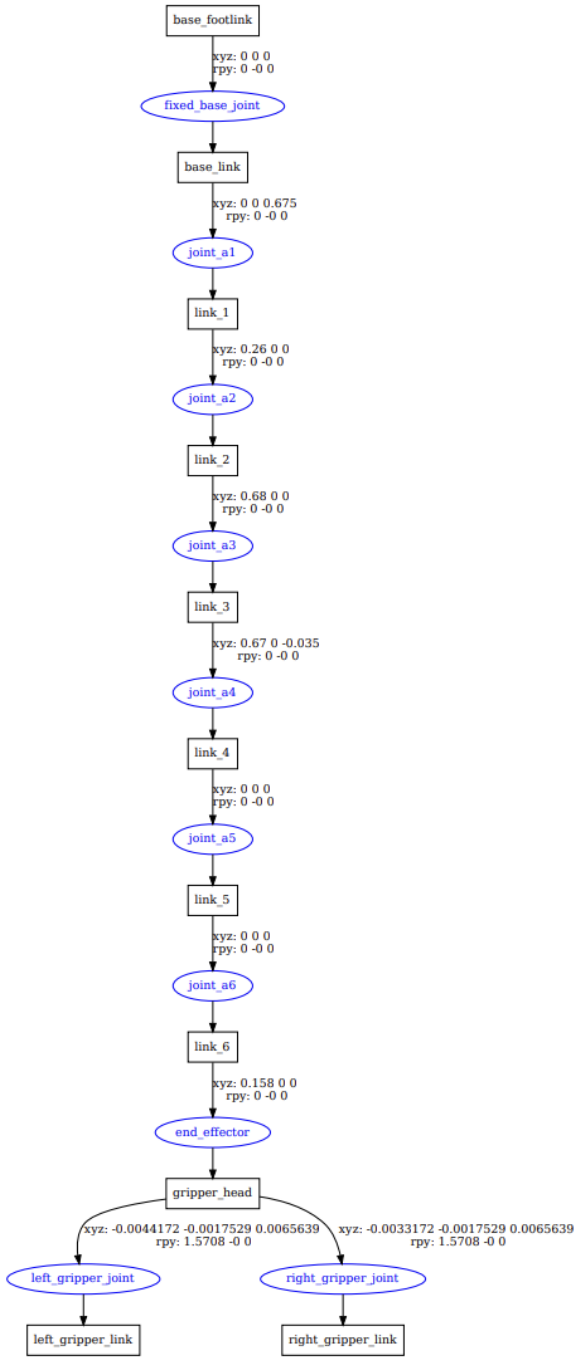



**Figure A.1.:** Hierarchical graph of KUKA KR-16

# Appendix B.

# Hardware

## B.1. Gripper Machine Drawings

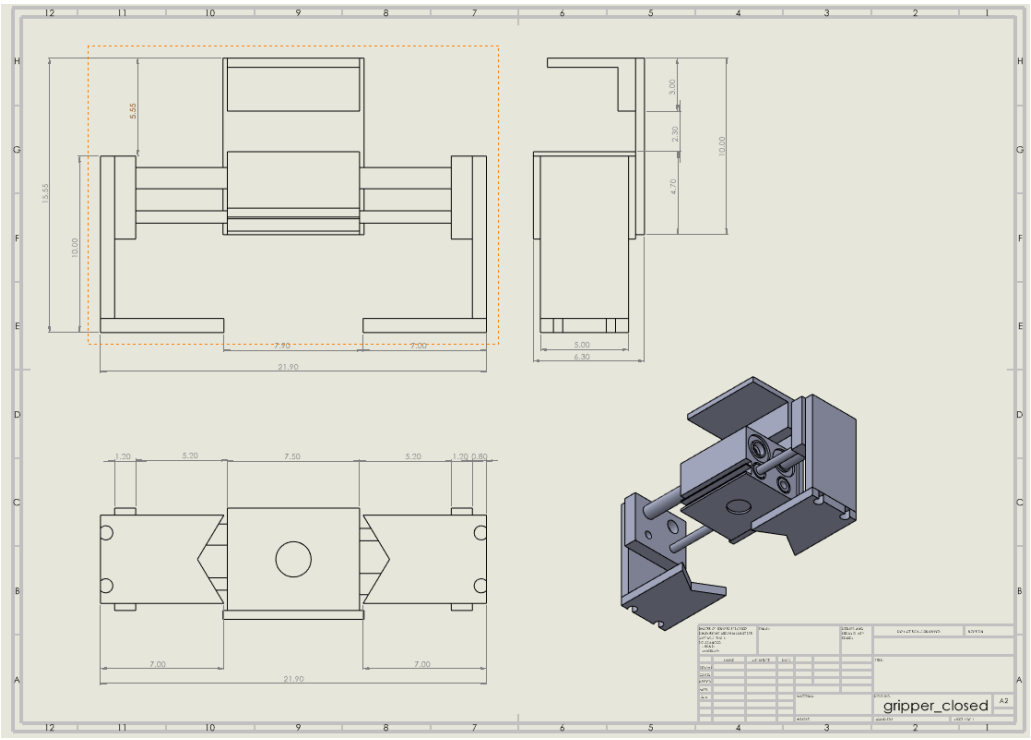



**Figure B.1.:** Opened gripper machine drawing

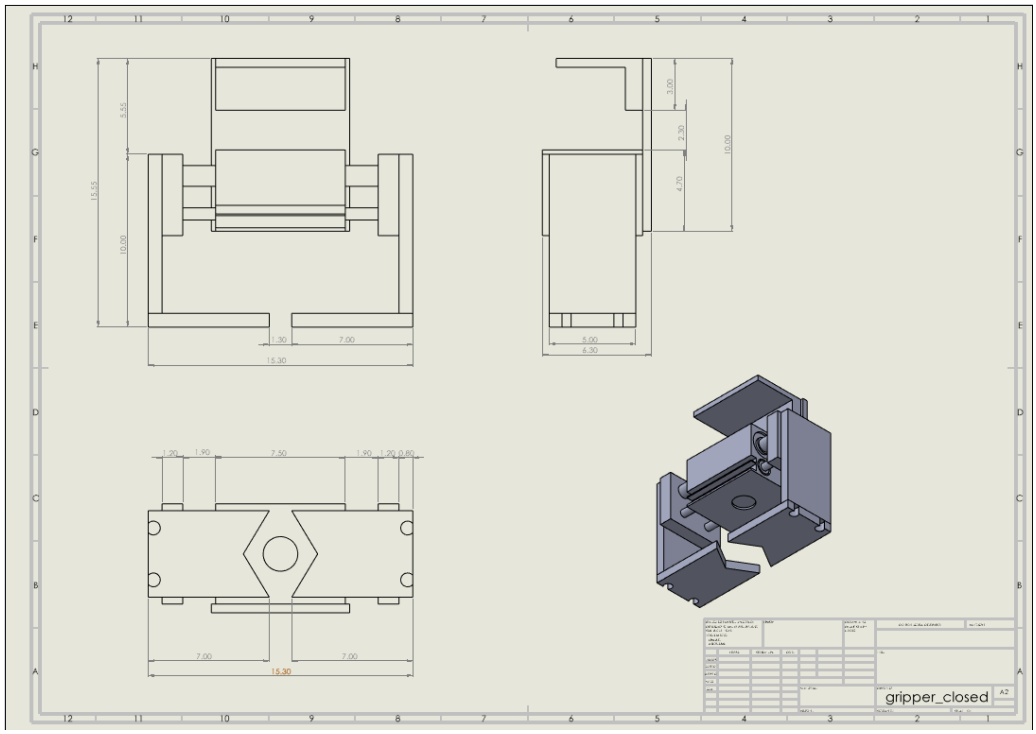
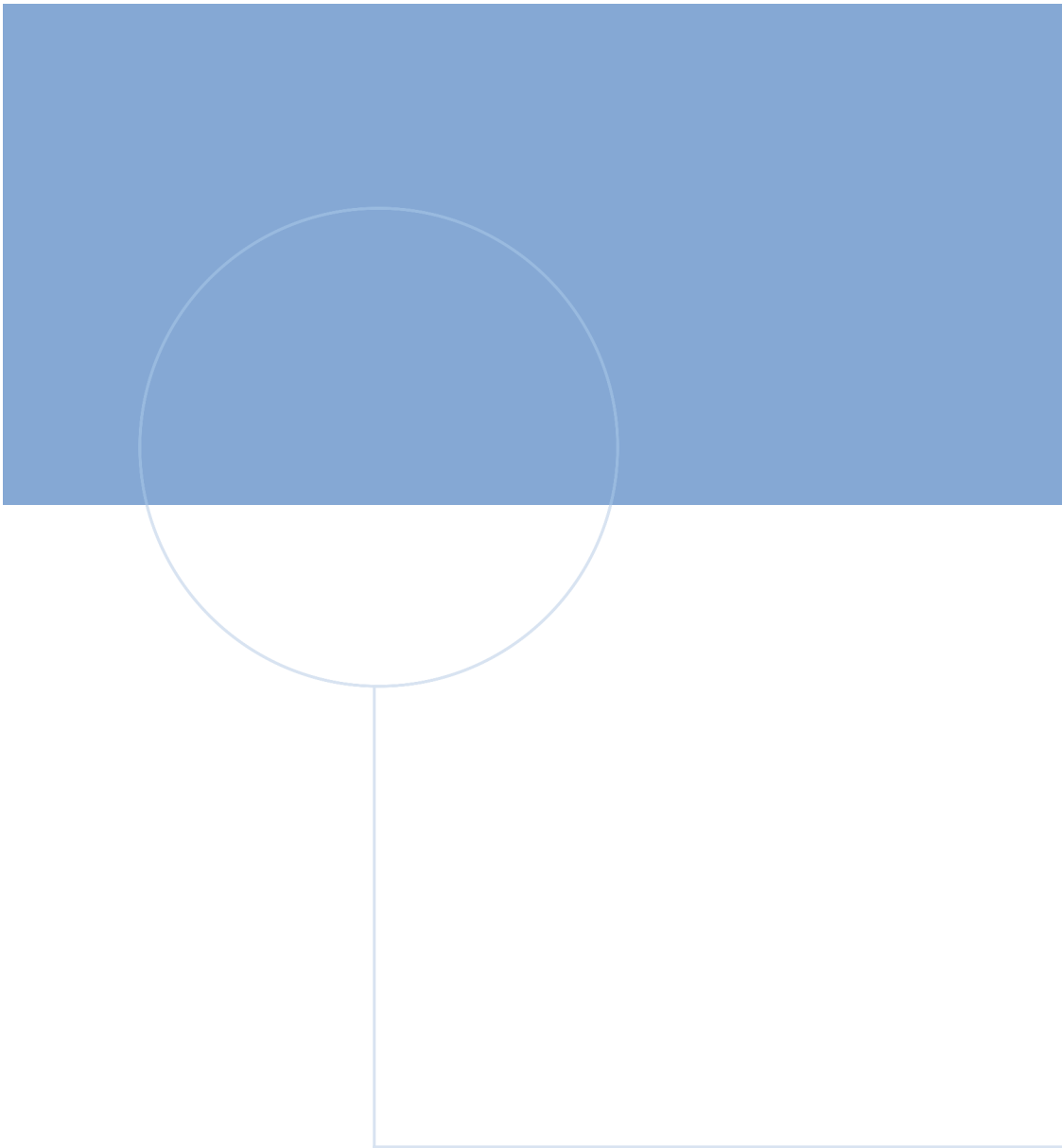



**Figure B.2.:** Closed gripper machine drawing



NTNU

Norwegian University of
Science and Technology