Master's thesis

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

Irfan Suvalija

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

Irfan Suvalija

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

I would like to thank both my supervisor Gunleiv Skofteland and co-supervisor Christian Holden for great insight, and guidance under this semester. I want to give a special thanks to my lab-partner Mustafe Kahin for everything

Lastly i want to thank Valgrinda for completing the different modifications to both the gripper, and the experiment setup.

Abstract

This thesis will mainly focus on a filter change through the help of a KR16 robotic manipulator. This manipulator will be performing a simple pick and place, and this will be performed through commands. These commands will correspond to a specific point in the laboratory space. The manipulator will be able to calculate the trajectory to reach this point with the help of the MoveIt motion planner.

The theory is split into three main parts: robot, which will range from topics such as robots workspace to kinematics. Teleoperation, which will go through different control architectures, control systems, delay, and input/output devices. Lastly ROS where the different ROS libraries will be covered, and different programs used. RViz controls the manipulator through the end-effector will be tested. The test of a joystick/controller setup will be in a virtual simulation, and in the physical.

The experiment show promising results for the system developed. Since it is at its core a simple pick and place where we can control the robot by commands, these commands can be improved over each iteration to complete a smoother motion. The joystick/controller setup was performed, but because of the delay experienced by the hardware limitation. This was the drop and instead tested with a mouse.

Sammendrag

Denne oppgaven vil hovedsakelig fokusere på et filterbytte ved hjelp av en KR16 robotmanipulator. Denne manipulatoren vil utføre et enkel plukk og plasser, og dette vil bli utført gjennom kommandoer. Disse kommandoene vil tilsvare et spesifikt punkt i laboratorierommet. Manipulatoren vil være i stand til å beregne banen for å nå dette punktet ved hjelp av MoveIt-bevegelsesplanleggeren.

Teorien er delt inn i tre hoveddeler: robot, som vil spenne fra emner som robotens arbeidsområde til kinematikk. Teleoperasjon, som vil gå gjennom forskjellige styringsarkitekturer, styringssystemer, forsinkelse og inngangs-/utgangsenheter. Til slutt ROS hvor de ulike ROS-bibliotekene vil bli dekket, og forskjellige programmer brukt. RViz kontrollerer manipulatoren gjennom endeeffektoren vil bli testet. Testen av et joystick/kontrolleroppsett vil være i en virtuell simulering, og i den fysiske.

Eksperimentet viser lovende resultater for systemet utviklet. Siden det i kjernen er et enkelt valg hvor vi kan kontrollere roboten med kommandoer, kan disse kommandoene forbedres over hver iterasjon for å fullføre en jevnere bevegelse. Joystick/kontroller-oppsettet ble utført, men på grunn av forsinkelsen opplevd av maskinvarebegrensningen. Dette var dråpen og i stedet testet med en mus

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Robots are tools used to replace human labor. This replacement often takes place when the work task identifies with the three Ds. These Ds are defined as dull, dirty, and dangerous work tasks. These tasks often completed by human labor have over the century been replaced by robotics or some form of robot/human coexistence. With the advent of the internet, which introduced long distant communication. The central computer which monitors the operation, and execution does not need to be physically present. This has opened up the door for telerobotics, which translate directly into robotics at a distance. This makes it able for an operator to control robots from a distance and eliminates the fourth D. Equinor plans to use teleoperation, and robotics to operate their offshore platforms onshore.

1.1. Problem description

The main objective of this thesis is the maintenance of a filter depicted in Figure 1.1. This objective will be completed with the help of a virtual environment(RViz, gazebo), and in a realistic environment (laboratory). The robot that´s going to perform this operation is KR16, because of its accessibility to us in the NTNU robot laboratory on Gløshaugen, and for the existence of GitHub repository files containing the URDF files for the robot manipulator

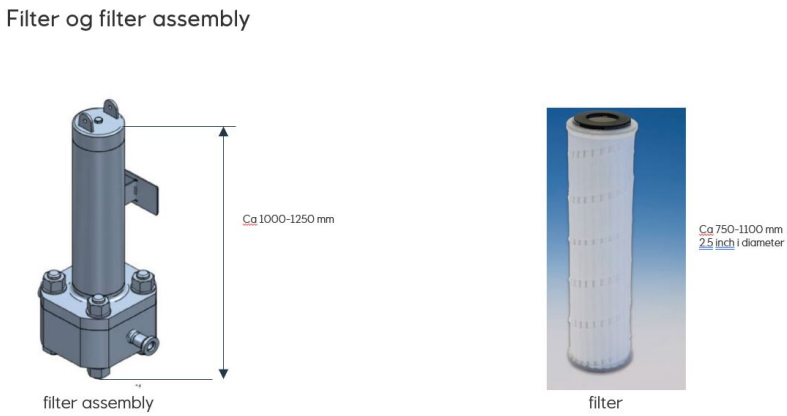


Figure 1.1.: Filter and filter Assembly

This filter is used to clean the TEG fluid. The TEG fluid is mostly used for the separation of water vapor from the gas stream. This filter collects these impurities and therefore needs periodical checks and replacement. Equinors wants to replace the worker that needs to complete this task with a mobile robot through teleoperation.

1.2. Related work

The part from the ROV and AUVs are taken directly from the specialization project [22].

1.2.1. Telerobotic applications

Surgical

In the 1995 intuitive surgical inc. used several concepts such as haptic augmentation, and teleoperation to lead in the construction of the "Da Vinci" telesurgical systems introduced into the market in 1999. Some of the basic capabilities were used in constructing the ZEUS system, which in 2001 was used to perform the first transatlantic surgery. When the surgeon in New York (US), performed a surgery on a patient located in Strasbourg (France).[12]

Space teleoperation

In 1993 the first telerobotic system was developed by the german Spacelab mission D2 called ROTEX. This mission was flown into space equipped with a robot

that could work in four operational modes: Automatic, teleoperation on board, teleoperation from the ground, and tele-sensor-programming. [12], [6]

1. Automatic has the functionality that the robot can be preprogrammed from the ground
2. Teleoperation on board the space shuttle, where the astronauts could use stereo TV monitors to control the robot.
3. Teleoperation from ground which where built with predictive computer graphics.
4. The tele-sensor-programming was learning by showing in a completely simulated world on the ground which includes the sensory perception with sensor-based execution later on board.

The main control architecture concept was that these robots will be using a shared autonomy approach. Where this included the shared control, and share intelligence that is based on local autonomy loops on board with high bandwidth.[12], [6]

The time delay for this system was about 6-7 seconds for the ROTEX to work it used predictive computer graphics. In the predictive computer graphics, the operator issues a command to a predictive model and this then gets translated to the robot onboard.[12], [6]

1.2.2. ROV and AUVs

Remotely Operated Vehicles (ROVs) which are underwater robots that have been given the task of replacing human divers. As the name implies is remotely controlled by an operator onshore, and is connected by a cable that supplies information, and power between the ROV, and the operator. [22]

Autonomous Underwater Vehicles (AUVs), is an Autonomous version of the ROV, and is yet not available in the consumer market [17]. The AUVs have generally been classified into three different classes: [22]

1. **AUVs for survey** is a version of the autonomous underwater vehicle that's built for underwater exploration, cartography of the seafloor, and probe sampling. These vehicles are not connected to an operator but instead move autonomously. This property makes it able to execute missions over a great distance without the need for a support vessel by its side. [17], [2], [22]
2. **Hybrid ROVs/AUVs** as the name implies is a combination between Remotely operated and Autonomously operated. According to [17], AUV is

used to transport the ROV to the sea-bed. Once the AUV has arrived it docks automatically to a docking station which has a link to the surface through the cable that supplies the AUV its energy and communication. At this point, the ROV can be controlled in real-time from the surface. [17], [2], [22]

3. **IAUVs**: Typical missions are the activation of valves, the deployment of components, or the inspection during the installation and maintenance of subsea wellheads. Since umbilicals are practicable for depth of more than 3000 meters. IAUVs can be used to replace ROVs in deep-sea applications. [17], [2], [22]

Chapter 2.

Theory

2.1. Robot

Information used in this chapter is mostly taken from siciliano et. al [20], and from the specialization project written last semester, the part for DH-convention, and supervised control architecture is directly taken from the specialization project [22].

2.1.1. Robot manipulator

The mechanical structure of a robotic manipulator consists of serial of rigid body called links, that is interconnected by joints. A manipulator is characterized as an arm that ensures mobility, a wrist that confers dexterity, and an end-effector that performs the task required of the robot.

The fundamental structure of a manipulator can be separated into two different categories, open and closed kinematic chains. The open kinematic chain is when the manipulator is closed in the base, and free in the other. When the sequence of links forms a loop this will be categorized as a closed kinematic chain.

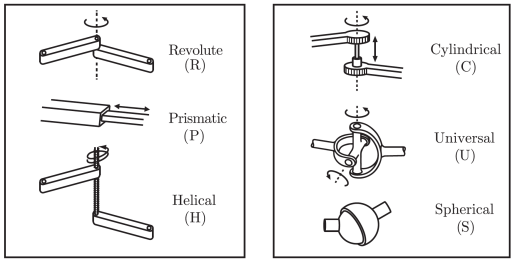


Figure 2.1.: Common joint types
[8]

A robotic manipulator has ensured mobility through the presence of joints. These joints can come in different forms as can be seen in Figure 2.1. For an industrial manipulator which we will be working with the most common form of joints are the revolute denoted by R, and prismatic denoted by P. The revolute ensures rotational movement along the joint axis, while the prismatic ensure translational movement with the joint axis. Since in an open kinematic chain both the prismatic and the revolute joints will each give one DOF (degree of freedom). The amount of joint will then determine the amount of DOF the kinematic system will have.

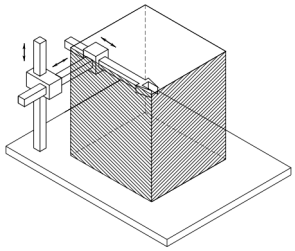
Joint type	dof f	Constraints c between two planar rigid bodies	Constraints c between two spatial rigid bodies
Revolute (R)	1	2	5
Prismatic (P)	1	2	5
Helical (H)	1	N/A	5
Cylindrical (C)	2	N/A	4
Universal (U)	2	N/A	4
Spherical (S)	3	N/A	3

Figure 2.2.: DOF for joint types [8]

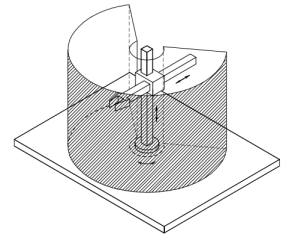
For KR16 which is 6 revolute joints, the DOF will then be 6, and if we add that the robot manipulator is not stationary. This will add another 2 DOF to the system.

2.1.2. Workspace

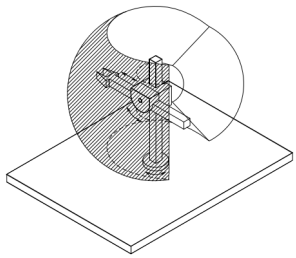
For a robot manipulator, the workspace represents the portion of the environment the manipulator's end-effector can access. Workspace's shapes and volume are determined by the manipulator's structure as well as the presence and type of mechanical joint limits. This makes it possible to categorize the manipulator into different categories. Some of the categories are Cartesian, Cylindrical, spherical, SCARA, and anthropomorphic. These manipulator types have different workspaces based on the possible movement the manipulator can achieve as seen in Figure 2.3.



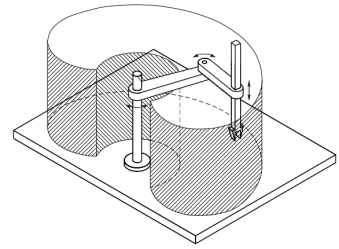
(a) Carthesian workspace



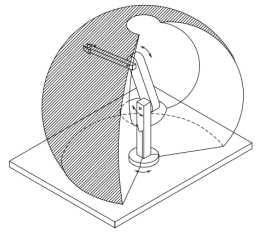
(b) Cylindrical workspace



(c) Spherical workspace



(d) SCARA workspace



(e) Anthropomorphic workspace

Figure 2.3.: Workspace for different manipulator types [20]

As can be seen from Figure 2.3, the category that the KR16 manipulator falls into is the anthropomorphic manipulator type. This type of manipulator is flexible and can perform the lifting of heavy objects. Figure 2.4 shows the KR16 manipulators workspace.

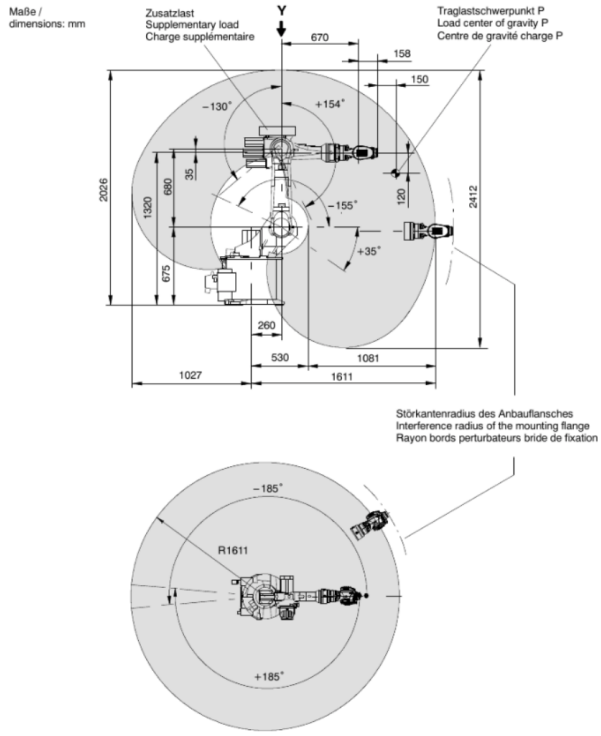


Figure 2.4.: KR16 workspace
[5]

2.2. Kinematics

Kinematics is the study of the motion of bodies without consideration for the forces or moments that cause the motion. When it comes to robotics is often separated into two main categories: forward kinematics(FK), and inverse kinematics(IK). Whereas forward has the goal of finding the end-effector positions when the position and orientation of each joint are known. The inverse is about finding the position and orientation when the end-effector point is given.

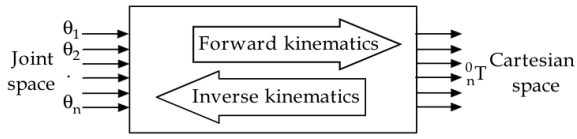


Figure 2.5.: Correlation between Forward, and inverse [7]

2.2.1. Forward Kinematics

As mentioned above forward kinematics refers to the motion of the robot manipulator when the end-effector is at its final position, and information such as cartesian position(x, y, z) and orientation is known for each joint. This information can then be used to find the rotation matrix and the homogenous transformation matrix for each joint. This is then used to find the end-effectors end-point. Some well-known way to calculate the end-point position is through the use of DH-convention.

Rotation matrix

A rotation matrix is defined as a transformation matrix that is used to perform a rotation of an axis. This rotation matrix belongs in the special orthonormal group SO(m) of the real (mxm) matrixes. For robotics this m is defined by the joints that will perform this rotation for spatial rotation it is m=3, and for planar rotation the m = 2.

Since the workload will be performed by KR16, this manipulator possesses only a rotational joint. Since the motion that will be performed by these joints are in a 3D environment. The focus will be on spatial rotation ergo SO3.

The special orthogonal group SO3, also known as the group of rotation matrixes, is the set of all 3x3 real matrices R that satisfy the following two conditions. $R^T * R = I$ and $det(R) = 1$ if the matrix if right-handed, and $det(R) = -1$ if its left-handed. The matrix will have the following setup shown Equation 2.1.

$$R(x, y, z) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.1}$$

Considered that the frames can be obtained via elementary rotations of the reference frame about one of the coordinate axes. These rotations are positive if they are made counter-clockwise about the relative axis.

$$R(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, R(y,\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{2.2}$$

$$R(z,\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

A visualization of rotation around a reference frame around the z-axis with the angle α. Shown in Figure 2.6.

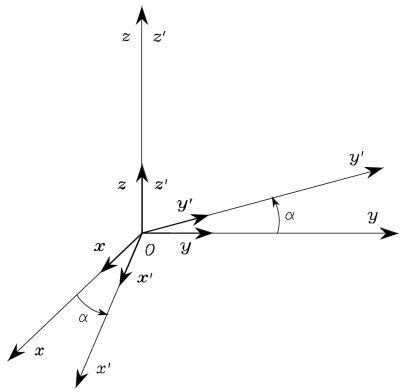


Figure 2.6.: Visualization of rotation of the reference frame around z-axis [20]

Homogenous transformation martix

The homogenous transformation matrix is a representation of both the orientation and position of the reference frame in the rigid body. The orientation of the reference frame is defined by the rotation matrix, and the position is defined by the position of the reference frame in the rigid body. As seen in the Equation 2.4

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{33} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The links are often labeled from the ground link 0 to the end-effector frame n in an open-chain robot. When combining the homogeneous transformation matrices in an open chain. The orientation for the chain and the last point in the chain can be found. As can be seen in Equation 2.5

$$T_{0n} = T_{01}T_{12}, ..., T_{(n-1)n} \tag{2.5}$$

General DH convention

Denavit-Hartenberg (DH) convention is the most common approach to forward kinematics for attaching reference frames to each link in an open chain. The links are often labeled from the ground link $\{0\}$ to the end-effector frame $\{n\}$ in an n-link open-chain robot. The forward kinematics of an n-link open-chain can be expressed as Figure 2.7

In DH-convention each reference frame to each link is not placed arbitrarily but instead designed to cut the number of free parameters required to specify the whole system. As shown in Figure 2.7. [20], [22]

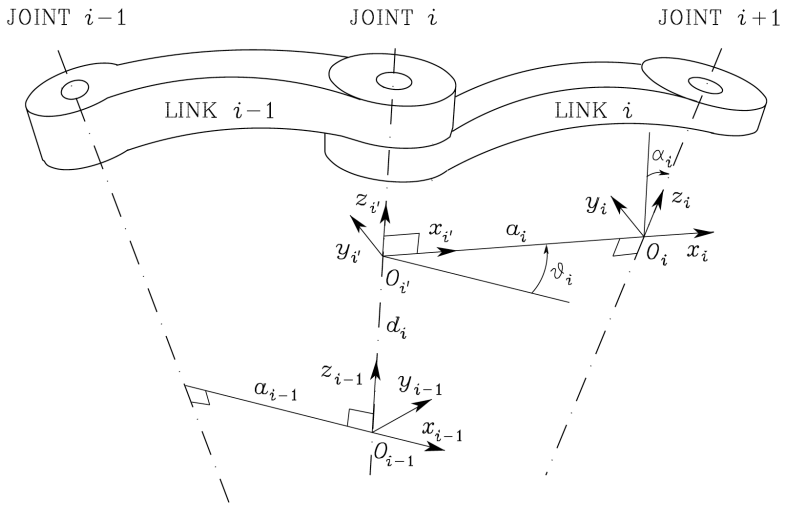


Figure 2.7.: Modified DH convention axes assignment and parameters [20]

1. d_i: Link offset, distance between X_{i-1} and X_i, measured along Z_{i-1}, variable in prismatic joints.
2. α_{i-1}: Angle between Z_{i-1} and Z_i, measured along X_i.
3. a_{i-1}: Link length, distance between Z_{i-1} and Z_i, measured along Z_{i-1}.
4. θ_i: Joint angle, Angle between X_{i-1} and X_i, measured along Z_i, variable in revolute. joints

The total transform between the links L_{i-1} and L_i can be thought of as a rotation by α_{i-1} along X_{i-1}, translation by α_{i-1} along X_{i-1}, rotation by θ_i along Z_i, and finally translation by d_i along Z_i. Shown below:

$$T_{i-1} = Rot(x_{i-1}, \alpha_{i-1}) Trans(x_{i-1}, a_{i-1}) Trans(z_i, d_i) Rot(z_i, \theta_i) = \tag{2.6}$$

$$\begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_{i-1} \\ sin(\theta_i)cos(\alpha_{i-1}) & cos(\theta_i)cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & -sin(\alpha_{i-1})d_i \\ sin(\theta_i)sin(\alpha_{i-1}) & cos(\theta_i)sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_i) * d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

2.2.2. Inverse Kinematics

Inverse kinematics as mentioned above is about finding the position and orientation of each joint in the manipulator. When the known factor is the position of the end-effector. The solution to this problem is complex for the following reasons [20]:

1. Equations generated by the inverse kinematics are often nonlinear and thus is not possible to find a closed-form solution.
2. The inverse kinematics can generate an infinite solution that can describe the robot manipulator's joint orientations related to the end-effector.
3. There might be no admissible solutions, in view of the manipulator kinematic structure

2.3. Telemanipulation

Telerobotics literally translates to robotics at a distance and refers to a robotic system controlled by a human operator through long-distance communication. The research material often refers to a "master" and "slave" relation. Where the "master" is defined as the human operator that encompasses elements such as a joystick, monitors, keyboard, or other input/output devices used for control. While the "slave" is often defined as the remote device that encompasses the robot, supporting sensors, and control elements. [12], [22]

Teleoperation which is the general subset of telerobotics is often used to replace human work tasks in a location that are hazardous, or unsuitable for humans. This idea was developed in the 50-60s as a way to control the nuclear power plant, but with the advent of the Internet is used in different industries like research, space exploration, and on the space station (ISS). Some of the challenges that telemanipulation encounters are delay, loss of packets, and limited bandwidth. These challenges can be solved or minimized through exposure with different methods listed below. [12], [22]

2.3.1. Delay

Delay is one of the most important aspects that need consideration when designing a teleoperated system. Delay or time delay is described by the difference in the communication link between the "master" and the "slave" device. Delay can surface when there is a large distance between the "master" and "slave" devices, material disturbance, hardware errors/limitations, or a combination. This is an

important problem when it comes to stability for teleoperation systems, and is there well documented.

From [15] tells us about some difficulties that are encountered when designing a teleoperation system in ground-space relation. One way that is proposed to minimize the delay is through a predictive display that shows the model of the environment and the "slave" device. The operator can then perform the task on the display by moving the "master" arm without any time delay. Those inputs given to the "master" arm can be transferred over to a virtual simulation of the "slave", and thereafter transferred over to the remote "slave" device. The issue that this documented was that a perfect model will not exist in practicality, and there introduce the Roseborough Dilemma. The Roseborough dilemma goes such as if a perfect model exists, why should it be teleoperated. Therefore it should be considered a tool that will reduce the amount of information and online mental modeling that the operator has to do. It helps bridge the time gap, offering approximate clues until the actual information is available. The difference between real and modeled environment has to be copied in real-time by the remote "slave" with the use of some local autonomy

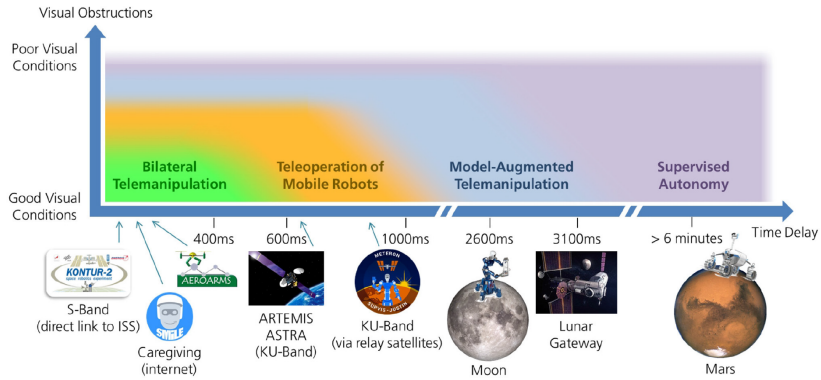


Figure 2.8.: Depiction of delay, and the different control architecture best suited [6]

Input/Output devices

Considering the devices that should be used for information collection in a teleoperation to work. From [15] tells about two features that have to be considered. These 2 are the input device properly(master arm, joystick, etc.), and the control mode to employ position and velocity. The reference listed the importance, and main reasons for picking between these features.

Two types of control modes that are considered:

1. **Position control:** Since the position of the manipulator will correspond to the position of the input device. The biggest disadvantage is the need for indexing when large or precise manipulator motions are in question
2. **Rate control:** is preferably used when the difference between the position and the manipulator is very large. It is thereby less intuitive than the position control but allows for improving controllable for simple tasks.

Common types of input devices considered by [15]:

1. **Mater arm:** The use of a master arm is very intuitive. 6 DOF can be used on a single grip. It can be tiring for slow movement and difficult to operate for precise positioning.
2. **Joystick:** Are less intuitive and two joysticks are needed for 6 DOF. They are very good for precise positioning and the operator does not get tired.
3. **Space mouse:** Force input device is not intuitive, but can integrate a 6 DOF in a single device.

2.3.2. Classification of methods used

There a different control systems used to construct a teleoperated system some of the most common control systems are Bilateral, and Non-bilateral systems.[15]

Bilateral System

Bilateral systems is describes as a system where the operator (master) is directly coupled with the device (slave). The main objective with this control scheme is that the operator (master) can directly feel the contact forces from the device (slave) when commands are executed. [15]

From [15] tells that a classical bilateral schemes can be very unstable under time-delay. This is caused by the amount of packets that are sent over the communication channel. Some of the approach used to solve the issue of time-delay in a bilateral system are Passivity theory and control theory.

1. **Passivity theory** states that a system is stable if the system dissipate, and never increments its total energy.
2. **Control theory** is a more classical approach where the linear model of each element is proposed and block diagrams are constructed.

Non-bilateral system

For a non-bilateral system is a control scheme where the coupling between the operator (master), and the device (slave) only takes place in one direction. Since the coupling only takes place in one direction does not mean that this control scheme will exclude force feedback between the device (slave), and the operator (master). Some Ways that the force feedback have been constructed according to [15] is by:

1. **Virtual forces FR:** is where the operator can get feedback from the commands, but it does not have to be generated by the device (slave). It can be based on a model or simply used to display other kind of information.
2. **Indirect FR:** Is by directly sensing the contact forces in the passive hand that is, the hand that is not generating the command.

Other information

Tele-programming, which consist of performing the task manually in a simulator before the real operation to gather data of how the task must be carried-out. Afterwards, the commands are sent over to the device (slave) to be carried-out in the real environment.

Predictive-techniques, which are techniques that emplot a predictive simulator where the operator can carry out the task interactively, while the commands are being sent out in real time to the remote device (slave) for execution.

2.3.3. Control architecture

Compared to a normal robotic system, which can execute given motions of other programs without consultation by the user/operator. A telerobotic system is controlled by input/output devices that provide the user with control, and understanding. of the surroundings of the robot which can be described by the style and level of this connection. There are two categories that will be gone through in this thesis is Direct control, shared autonomy(Shared control, and supervisory control). Where the separation is made is the intelligence and autonomy that the robot system has.

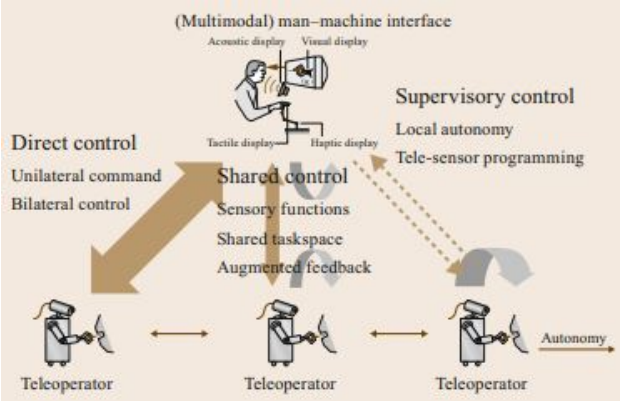


Figure 2.9.: Control architecture depiction
[12]

2.3.4. Direct control

Direct control implies that there no intelligence or autonomy in the system. Thereby making the operator via a master interface responsible for all movement and motions done by the slave device. This control architecture may incorporate sensory feedback, or other haptic augmentations to improve the operators control of the slave device in a bilateral control scheme.

Some drawbacks of direct control are their need for visual, haptic or other augmentations to help the operator preform precise and controlled movements. This can increase the amount of packets needed to be sent, and help increase the time delay for the telerobotic system.

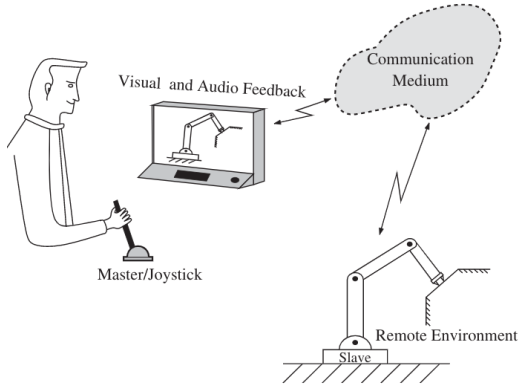


Figure 2.10.: Direct control architecture depiction
[12]

2.3.5. Shared autonomy

Manual control or direct control of a robotic manipulator is highly demanding, tedious work for a human operator when it comes to manipulating over a large distance. These issues can occur because of the sheer number of DOF, or because large delays can build up a culture of move and wait. Shared autonomy solves some of these issues by delegating some control/autonomy to the robot. Which can reduce the workload done by the human operator. Some examples of where shared autonomy are in space exploration, where NASA´s space rovers land on Mars with delays of up to several minutes. Thereby making the mission impossible without delegating control/autonomy to the remote device.

As mentioned above shared autonomy is the deligation of some local autonomy or intelligence to the robotic system. The main categories in this are the shared control and the supervised control. Which are mostly separated in the way that the robotic system improves, and the control delegated to the system.

Shared control

Shared autonomy is a control architecture that tries to implement both the basic stability and sense of presence achievable by direct control with the smarts and possible safety guaranties provided by the autonomous control. This can occur in various forms, one example that helps in this regard is that the slave robot may need to correct motion commands, regulate subset/subtasks of joints or overlay additional commands.

Some advantages that comes with this control architecture is that when it comes to large time delays, a human operator may only be able to specify gross path commands, which the slave must fine-tune with local sensory information. We may also want the slave to assume control of sub-tasks, such as maintaining a grasp over long periods of time. A special application of shared control is the use of virtual fixtures. Virtual elements, such as virtual surfaces, virtual velocity field, guide tube, or other appropriate objects, are superimposed into the visual and/or haptic scene for the user. These fixtures can help the operator perform tasks by limiting movement into restricted regions, and/or influencing movement along desired paths. Control is thus shared at the master site, taking advantage for preknowledge of the system or task to modify the user´s commands and/or to combine them with autonomously general signals.

The Springer et. al [12] tells of a good way of explaining shared autonomy. "A straight line can be drawn by a human in freehand, but with the help of a ruler, the line will be drawn straighter and faster." Similarly the robot can apply forces and positions to help the human operator draw a straight line. Based on the nature

of the master robot and its controller, the virtual fixtures may apply corrective forces or constrain positions. In both cases, and in contrast to physical fixtures, the level and type of assistance can be programmed and varied.

An example of shared control is in surgical applications where the shared control can compensate for the movements of a beating heart. This sensed heart motion can be overlaid to the user command and help surgeons operate on a virtually stabilized patient.

Supervised control

Supervisory control derived from the analogy of supervising a human subordinate staff member. For the telerobotic system this translates to high-level directives given by the operator "master" side to the remote "slave" device. The "slave" device in turn returns a summary of the information. Thus making it possible for the "master" to build and improve the remote device algorithm/code improving the robots´s autonomous properties by each iteration of the work task. [12], [22]

Mixed control

From [6] defines another control architecture of mixed-initiative shared control approach. Where they combine the (position/forces/torques) for the robot to complete a work task autonomously. The biggest difference between the mixed-initiative, and the other architectures is that the remote device will hand over control of the remote device when the confidence of task completion is low.

2.4. Tools programming and simulation (ROS)

2.4.1. ROS General

ROS(Robot Operating System) is an open-source software development kit for robotic applications. From ROS tells that it is the "de facto" platform for the development of robot applications used in industries, research, and prototyping through to deployment and production. ROS can be separated into ROS1 and ROS2, where ROS1 was used for this thesis, because of the ease of debugging, and the amount of help already present and well documented online.[19]

2.4.2. ROS nodes

According to ROS wiki [13], ROS nodes is a process that performs computations. Nodes are combined into a graph and communicate with each other using ROS

topics, services, actions, etc. Nodes are a way to structure and help build subprograms inside of the ROS workspace. Some benefits that come from introducing ROS nodes are:

1. **Reduced code complexity**: It simplifies the process of upscaling the applications, because of the separation between the nodes and packages. It simplifies the re-usability of the code.
2. **Better fault tolerance**: All the nodes communicate with each other through ROS, the nodes are not directly linked. Thereby making it possible to discover crashes and failures for each node. And a failure/crash in one will not affect the other nodes.
3. **language agnostic**: ROS is language agnostics which in essence means that a program written in python will work with a program written in C++. Python and C++ are the two most common programming languages used in ROS.

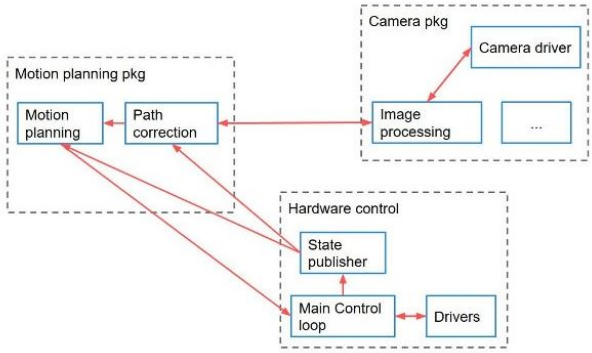


Figure 2.11.: ROS nodes example
[24]

2.4.3. ROS concepts

ROS concept goes through the diffent applications used for build up the workspace, and used in this thesis. The ROS control is taken directly for the specialization project [22]

catkin

Catkin is a ROS library mostly used as a folder builder. It is used to generate the necessary files, and folder setup for ROS to understand, and navigate through. Catkin generates folders such as build and devel, and the files being generated are

path files for your project, and dependencies for other libraries used. The src files are where you´re ROS project is stored. A typical catkin workspace will have a build similar to the Figure 2.12.

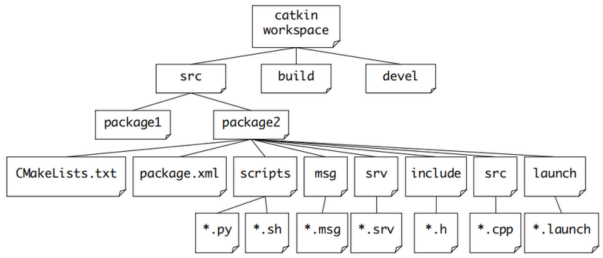


Figure 2.12.: Catkin workspace example [1]

xacro

Xacro is an XMl macro language used to construct shorter and more readable XML files by using macros that expand to large XML expressions. Xacro is used to build robot models, support for parameters, simple math functions, etc.

URDF

Unified Robot Description Format (URDF) is an XML format file that describes the robot´s physical description (links, joints). Links are the arms of the robot and are described in the URDF file by their position, inertia, and a 3D model with mesh and collision for the link. While the joint is the connection between the links and is described by the joint type (revolute, prismatic, etc.) and their connected links. URDF files are used to generate a virtual version of the robot model, and information such as end-effector, and other devices needed can be placed in the URDF file. A typical urdf-file is shown in Figure 2.13.

```
<xml version="1.0"?>
<robot name="robot_name">
  <link name="name_of_the_link">
    <visual>
      <geometry>
        <mesh filename="package://path/name_of_file"/>
      </geometry>
    </visual>
  </link>
</robot>
```

Figure 2.13.: Example of an URDF

ROS topics

ROS topics are explained as named buses where the nodes will exchange messages over. The topics have anonymous publish/subscribe, which decouples the production of information for its consumption. Nodes generally do not have the awareness of who they are communicating with, and the nodes that are interested in data will subscribe to the relevant topics. [26], [23]

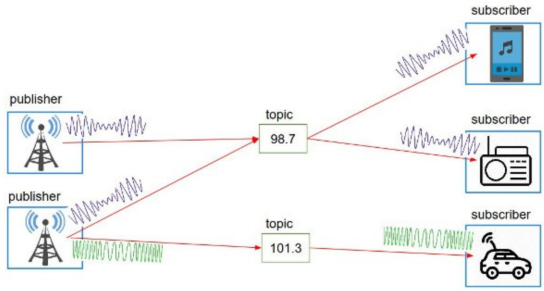


Figure 2.14.: Shows how topics work in ros
[26]

Topics are often intended for unidirectional and streaming communication.

ROS messages

ROS uses descriptive language to describe the data values that the nodes in ROS publish. This describing the data values is called messages and makes it easier for

ROS tools to automatically generate source code for the message type in several target languages. [11].

ROS service and clients

A ROS service is a client/server system, some characteristics of a ROS service are: [25]

1. It synchronous, which means it will send both requests, and blocks until it receives a response.
2. ROS service should only be used for computations and quick actions.
3. Service is often is defined by a name, and a pair of messages. One may be a request while the other is a response.
4. A service server can only exist once, but can have many client, and basically, the service will be create the server.

rqt graph

rqt graph is a plugin for ROS that uses ROS graph to construct a map of the connections between the nodes, topics, and messages sent to each other. It is helpful for the organization of the ROS nodes in the application.

2.4.4. Simulation programs

Simulation programs are often used as a virtual testing ground to find out the movement of the robot, and behavior while executing commands. Some of the most used simulation programs for this are Gazebo and RViz.

1. **Gazebo** is an open-source software library designed to build a virtual environment so that the operator can experience the movement and motion of the robot in a controlled environment. This robotic model only substitutes being the physical robot. Some use cases for gazebo include prototyping of new robots, development of new algorithms and behavior, testing and educational purposes, etc. [14][27]
2. **Rviz**(short for "ROS visualization") is used as a 3D visualization tool for robots, sensors, and algorithms. It enables the operator to visualize the robot´s perception of itself, and its environment. [27]

The most common comparison between RViz, and Gazebo, that is made is that "RViz shows you what the robot thinks is happening, while Gazebo shows you what is really happening.". [27]

2.4.5. ROS packages

The ROS packages encapsulate many of the large libraries that are either preinstalled or added to the workspace. There are many more packages for ROS, but these were used in this thesis.

Moveit

MoveIt is open-source software that utilizes libraries to work as a motion planner for ROS. According to MoveIt [4] the motion planner incorporates manipulation, 3D perception, kinematics (both FK and IK), control, and navigation. This is mostly done through MoveIt´s setup assistant which has a modular design, and several libraries for planning, kinematics, and collision checking. MoveIt is used by companies such as Google, Microsoft, NASA, etc. [10]. Figure 2.15, shows the system architecture for the primary node used by MoveIt. This node server works as an integrator. This integrator will work as a link that connects the individual components providing the user with a set of ROS actions and services.

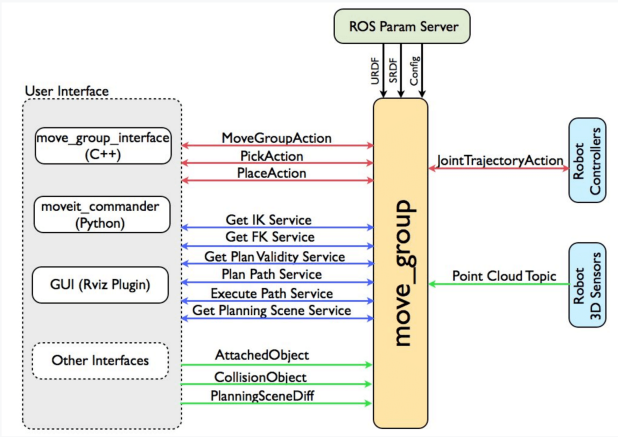


Figure 2.15.: Shows the information move group processes [4]

For the Figure 2.15 the move-group is connected to the user interface and ROS Param Server. The user interface encapsulates the different communication channels that the user goes through the user and the robot in input/output relations. Some of the well-known for this are C++, python, and Rviz which work as a GUI. [4] The ROS param server contains the URDF, SRDF, and config files. The URDF as explained in section 2.4.3, and the SRDF and the config files are autogenerated by the moveit-setup-assistant.

MoveIt uses a built-in plugin that calculates the forward, and inverse kinematics through jacobian, and integrates it within the RobotState class itself. This plugin will automatically be configurated by the Moveit-setup-assistant.

ROS-control

ROS control is a set of packages that includes transmissions, hardware interface, controller manager, controller interface, etc. All of these packages together will allow the user to interact and control the joint of the robot. [18], [22]

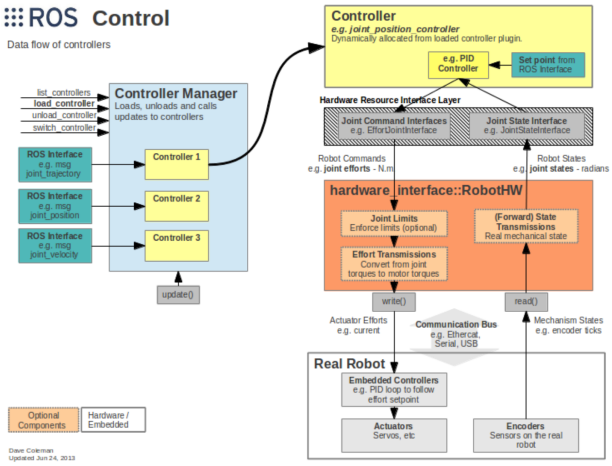


Figure 2.16.: ROS control
[3]

ROS control will take the joint state data, and inputs made by the user or 3rd party applications, and provide them to the robot as shown in Figure 2.16. The controllers(base controller/arm controller) are responsible for computing the output command required to achieve the set input set point. The hardware interface joints move, and will get feedback from the joint sensor. [3], [18], [22]

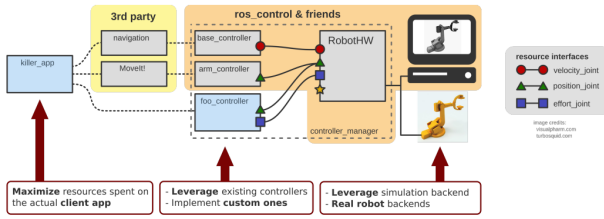


Figure 2.17.: ROS control affecting hardware [18]

Some types of controllers that are present in the ROS control package include joint-state-controller, effort controller, joint trajectory controller, etc. [18], [22]

Chapter 3.

Methodology

In this chapter, the thesis will go through implementations planned, and completed in the laboratory. The parts that my lab-partner Mustafe Kahin has the biggest involved in is the setup of a working workspace for the laboratory, and the ROS build.

3.1. Method

As listed above in the problem description the main focus of this thesis will be a filter change through a telemanipulated system. This will be done through the 6 DOF robot manipulator situated in the NTNU robotics laboratory at Gløshaugen. Simplification was performed to tackle the task given. The proposed simplifications were to separate and segment the different actions. The actions that had to be performed were:

1. Unscrew the top lid of the filter container, and place the lid in some location in the laboratory
2. Pick up the old/used filter, and place it somewhere
3. Pick up the new filter and place it in the location of the old, and pick the lid screw it back on

These actions need to be performed by the KR16 robotic manipulator in the laboratory.

3.1.1. SW-implementation

The forward kinematics as defined above in Section 2.2 is the calculations to find the end-effector point when the position, and orientation of each of the joints in the

kinematic chain is known. While the inverse kinematics is finding the position, and orientation for each joint. When the information given is the end-effector point. In this thesis the kinematics calculations for both forward, and inverse is being done by MoveIts own kinematic solver mentioned in Section 2.4.5.

The Moveit Setup Assistant is a graphical user interface that helps the user configurate neccessary files for movements of robot joint through a URDF model. The main function of the setup assistant is to generate SRDF, and config files. This is done to help the move group node mentioned in section 2.4.5 to recognize, and perform movement/rotations of the joints for the robotic model.

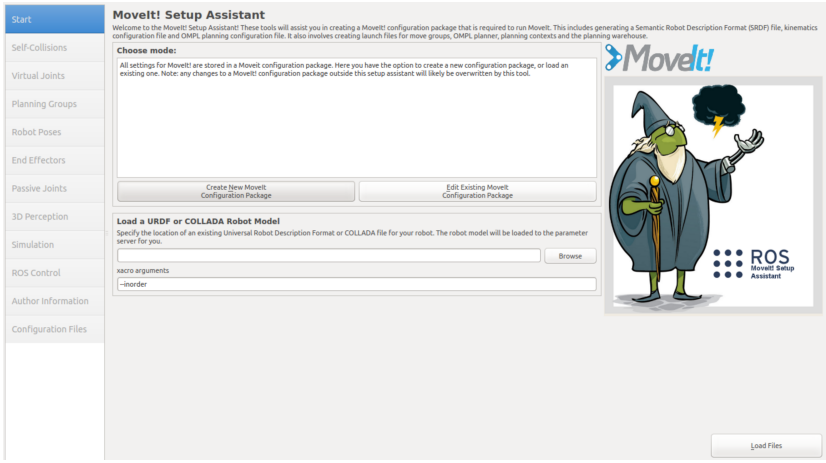


Figure 3.1.: Moveit-setup-assistant start page [9]

From Figure 3.1, we can see that the setup assistant can use different inputs from the user such as Start, Self-Collision, Virtual joints, Planning groups, Robot poses, End-effectors, Passive joints, 3D perception (Not relevant for this thesis), Simulation, ROS control and the other for a digital signature and export of the project file. The different inputs and our configurations for them are:

1. **Start:** Giving the MoveIt Setup Assistant your URDF file of the robot model. In our case the ROS industrial model for KR16
2. **Self-collision:** MoveIt Setup Assistant generate a collision matrix. This collision matrix will help MoveIt recognize the robot, and know what kind of movement will constitute a collision. Some of the reasons for optimizing the self-collision matrix is that the generator will search for pair of links that can be disabled from collision checking, which will help the motion planner decrease the time needed for processing the movements and orientation.

3. **Virtual Joints:** The Setup assistant uses virtual joints mostly to attach the robot to the world. Where you can pick the virtual joint name, Child link, Parent link, and the type of connection. For us, this is world, base link, world, and fixed, because the robot is fixed in place in the world.
4. **Planning group:** The setup assistant uses the planning groups to define what on the robot is the arm, and what is the end-effector. The setup assistant asks for the Group name, which kinematic solver you want, and what joints you want to add for each joint.
5. **Robot pose:** This is where the setup assistant lets you add a fixed pose for the robot. For example, if you want to add a position called home.
6. **End-effectors:** Defines the end-effector for the robot arm.
7. **Passive joints:**The passive joints are meant to allow users specification of any of the passive joints that may exist in a robot, which will tell the planner that they cannot kinematically plan for these joints. The KR16 model does not have any passive joint.
8. **3D perception:** This helps the setup assistant recognize configurations for sensor, camera, etc. This is not relevant for this thesis.
9. **Gazebo simulation:** Generates necessary additions to the URDF to translate the robot manipulator into Gazebo.
10. **ROS control:** Adds the necessary files to utilize the ROS control for your MoveIt package.

Now that a working virtual model of the robotic manipulator is pulled out of MoveIt. The robot model will have a working visualization in RViz with a motion planer. This motion planer can build a plan, and execute a trajectory specified by the user. The plan was to have an RViz simulation so that the operator can plan and execute a trajectory. This movement will then be translated to the Gazebo world. Since the Gazebo world is a realistic simulation of the robot's behavior this could work as a predictive display.

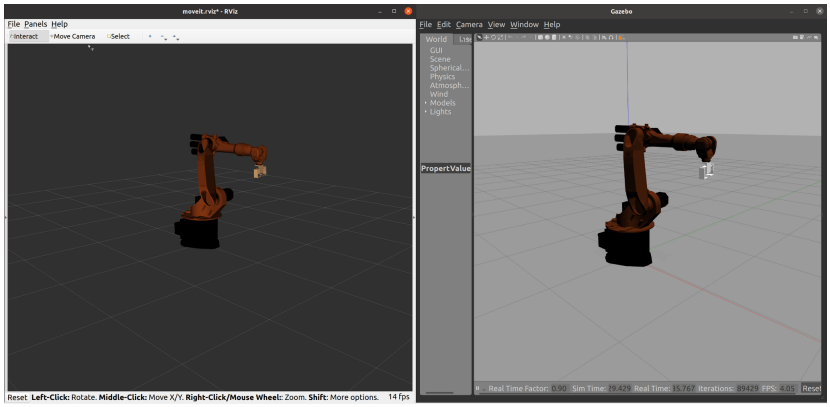


Figure 3.2.: KR16 in both Gazebo and RViz

The connection was established between the Gazebo and RViz as can be visualized in Figure 3.2, and the node connections in Figure 3.3

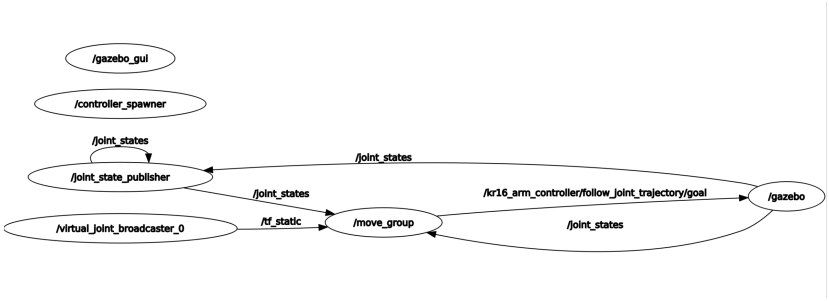


Figure 3.3.: ROS graph that shows nodes in the application

3.1.2. Robot

The KR16 is the robot manipulator that will be used to complete this set of actions mentioned above. Section 2.1.2 tells that the KR16 robot is categorized as an anthropomorphic robot. This robot category will have a similar 3D workspace as in Figure 2.3e. An example of the KR16 workspace will look similar to the Figure 2.4, but the laboratory will have dimensions that will limit the robot's workspace as shown in Figure 3.4.

(a) Robots distance right

(b) Robots distance left

Figure 3.4.: Robot arms testing limitations to movement

The physical robot can be controlled to a Ethernet cable, or through a control-pad connected to a control cabinet through cables. The KR16 robotic manipulator comes with an array of safety systems. Some of these features are depicted in Table 3.1.

Safety feature	T1	T2	AUT	AUT EXT
Emergency stop	STOP 0	STOP 0	STOP 1	STOP 1
Enabling switch	active	active	Not active	Not active
Operator safety	Not active	Not active	active	active
Max 250mm/s	Active	Not active	Not active	Not active
Jog mode	Active	Active	Not active	Not active

Table 3.1.: Safety features for the KR16

The control pad was used to derive positions by controlling the robot through its joints. These positions helped bridge the gap between virtual, and laboratory positions. The forward kinematics python code developed uses the joint's degrees to find the cartesian position for each joint. This was then tested by giving the robot random positions and checking through the TF node. Positions were taken in the T1 configuration. The code was executed in the T2 configuration for the following reasons safety, and the code not running in T1. The T2 configuration has an inbuilt "kill-switch" that is manually held done by the operator and can work as an end-all button if the robot performed undesirable motion. The AUT and the AUT EXT were not used, and therefore the system will not have any real-time capabilities. The different safety functions are depicted in Table 3.1

3.1.3. Theory

Since the use of forward, and inverse kinematics was done through the moveit-setup-assistant own kinematic solver. The construction of a forward kinematics

code was mainly used to compare the virtual simulation to the physical robot positioning. The DH convention was used to check the different joint positions, and orientations to find the end-effector position.

DH-convention

As mentioned above the position, and orientation coordinate was decided by the control pad, the KR16 will have a reference frame of Figure 3.5

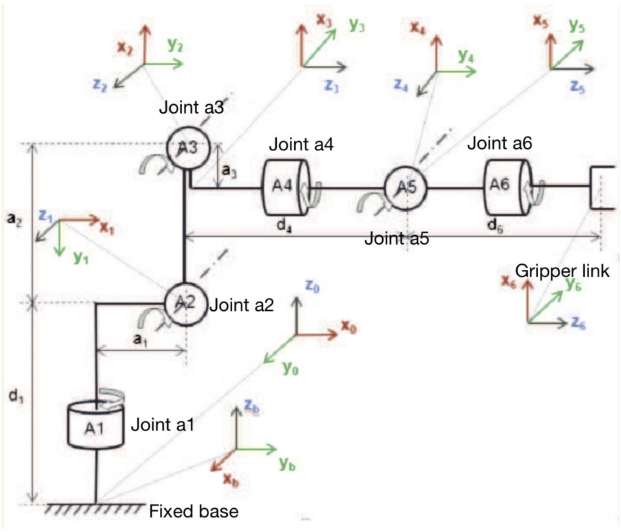


Figure 3.5.: DH-convention reference frame [16]

The table below shows the different link lengths and the connection between them from the KR16 URDF given to us by the ROS industrial Github repository shown in table 3.2

Joint name	Parent Link	Child Link	x(m)	y(m)	z(m)	roll	pitch	yaw
Joint a1	Base link	Link 1	0	0	0.675	0	0	0
Joint a2	Link 1	Link 2	0.26	0	0	0	0	0
Joint a3	Link 2	Link 3	0.68	0	0	0	0	0
Joint a4	Link 3	Link 4	0.67	0	−0.035	0	0	0
Joint a5	Link 4	Link 5	0	0	0	0	0	0
Joint a6	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 3.2.: Reference frame for kuka kr 16 urdf-file

This was then used to build a forward kinematics DH-python code with the formulas of rotation matrices for xyz-rotation shown in section 2.2.1, and the DH-transformation formulas shown Equation 2.7. These formulas where formulated into the different DH-table that was built as shown in Table 3.3

i	α	a	d	q
1	0	0	0.675	q_1
2	$-\pi/2$	0.26	0	q_2
3	0	0.68	0	$q_3 - \pi/2$
4	$-\pi/2$	-0.035	0.670	q_4
5	$\pi/2$	0	0	q_5
6	$-\pi/2$	0	0	q_6
7	0	0	0.115	0

Table 3.3.: DH-table from the python code

The Equation 3.1, shows the end-point for the joint 6 in the KR16 robot manipulator. This is shown when all the joints are at a zero-pose. The detail is in the positions from the transformation matrix. This correlates with the URDF description shown in Table 3.2.

$$T_{06} = \begin{bmatrix} 0 & 0 & 1 & 1.61 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.64 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} \tag{3.1}$$

Correlation between virtual, and physical space

The positions were mapped through the help of the python code, TF-node, and the joint angles. The TF-node will give information such as positions for each joint, and this can be checked with the python code when the joint angles were given. The joint angles are given in Table 3.4, and the comparison between the code, and the TF-node is given in Table 3.5.

Position name	Angles for the joint(deg)						
	q_1	q_2	q_3	q_4	q_5	q_6	q_7
Home	-4	-72	62	180	-99	-67	0
P_1	-4	-51	95	180	-45	-67	0
P_2	-4	-72	62	180	-99	-67	0
P_3	-48	-75	65	180	-98	-67	0
P_4	-48	-51	97	180	-44	-67	0
P_5	-42	-48	21	180	-117	-61	0
P_6	-42	-35	72	180	-53	-61	0
P_7	-42	-48	21	180	-117	-61	0
P_8	-4	-72	62	180	-99	-67	0
P_9	-4	-51	95	180	-45	-67	0

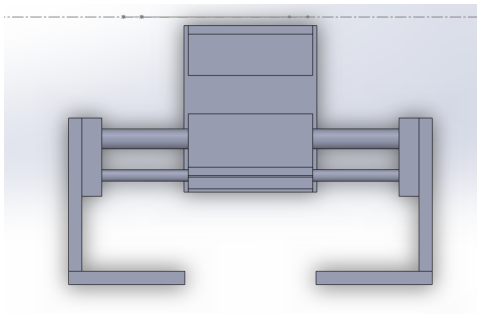
Table 3.4.: Angles of the joints for each point, from RViz

Position name	Position from RViz			Position from code		
	x	y	z	x	y	z
Home	1.13	0.08	1.40	1.13	-0.08	1.40
P_1(filter1)	1.14	0.08	0.71	1.14	-0.08	0.71
P_2(filter1up)	1.13	0.08	1.40	1.13	-0.08	1.40
P_3(filter1predrop)	0.74	0.82	1.41	0.74	-0.82	1.41
P_4(filter1drop)	0.75	0.84	0.70	0.75	-0.84	0.70
P_5(filter2pre)	0.99	0.89	1.45	0.99	-0.89	1.45
P_6(filter2)	0.99	0.89	0.63	0.99	-0.89	0.63
P_7(filter2up)	0.99	0.89	1.45	0.99	-0.89	1.45
P_8(filter2predrop)	1.13	0.08	1.40	1.13	-0.08	1.40
P_9(filter2drop)	1.14	0.08	0.71	1.14	-0.08	0.71

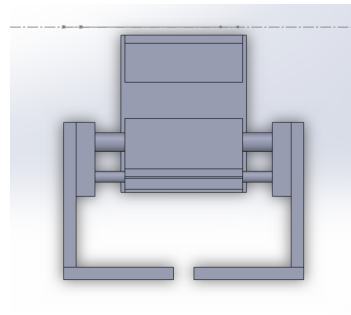
Table 3.5.: Positions for joint 6 in both the TF-node and the python code(rounded up to closest second decimal)

3.1.4. Hardware

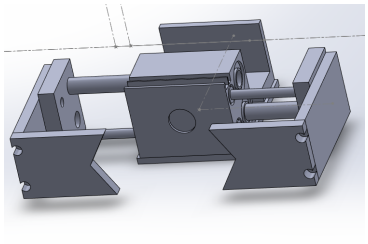
A specialized Gipper was not designed to perform the filter change. The gripper used in this experiment was found in the laboratory, and some modifications were performed to give the gripper a better grip on the filter. The gripper was then designed in SolidWorks and thereafter exported into a URDF format. The gripper is visualized in both open and closed states in Figure 3.6. Mechanical drawings for the gripper were constructed, and can be found in Appendix A.1, and Appendix A.2.



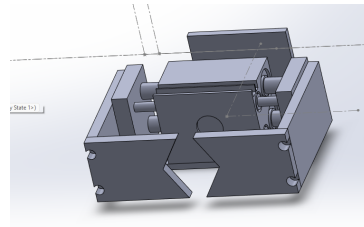
(a) Gripper front open solidworks



(b) Gripper front closed solidworks



(c) Gripper open



(d) Gripper closed

Figure 3.6.: Gripper both open, and closed form

The URDF tree for both the robot model, and the gripper can be seen in Appendix A.3.

Chapter 4.

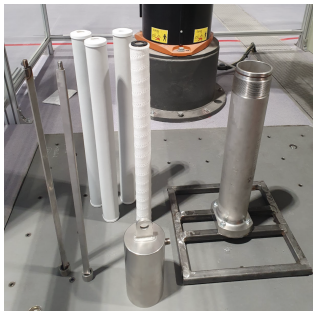
Results

In this chapter, the result will be presented as a visual representation of reality, and the simulation versions. There is a digital attachment in the form of a video showing the whole process. This will be more of a rundown of the improvement, and what is happening. There will be figures that show the manipulator in the real, and simulated world. The manipulator's trajectory, and a plot of movement for each joint. There will be a representation of the point without the manipulator and equipment.

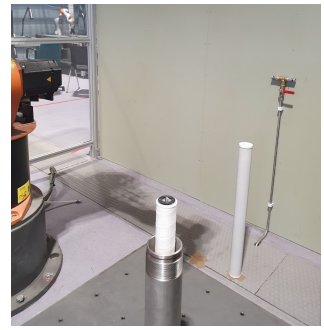
4.1. Experiment

4.1.1. Laberatory setup

The filter assembly consists of the cylinder container, cylinder lid, metal rod, and four filters as shown in Figure 4.1a. In these experiments, only two of the filters will be used. The cylinder container and the metal rods will not be a part of the experiments. The used/old filter is located inside the filter container, and the position of the new filter is depicted in Figure 4.1b. This figure is the items that will be used in the experiment.



(a) All the parts for the experiment



(b) Position in the laberatory

Figure 4.1.: Showing the laberatory setup without the manipulator

4.1.2. Pick and Place

Section 3 mentions the action that needs to be performed for a filter change. Some simplifications have been made to complete the filter change. The first simplification is that the top lid of the cylinder has been taken out of the experiment, because of the end-effector dimensions. The second is that the metal rod has been taken out, and the reasoning will be explained later. These simplifications turns this into a pick and place. Where the old filter must be picked up, and replaced with the new. For this, we constructed commands that will correspond with the different positions. These positions could be typed in, and MoveIt would calculate the trajectory thereafter plan and execute it.

The commands, and there respective positions are:

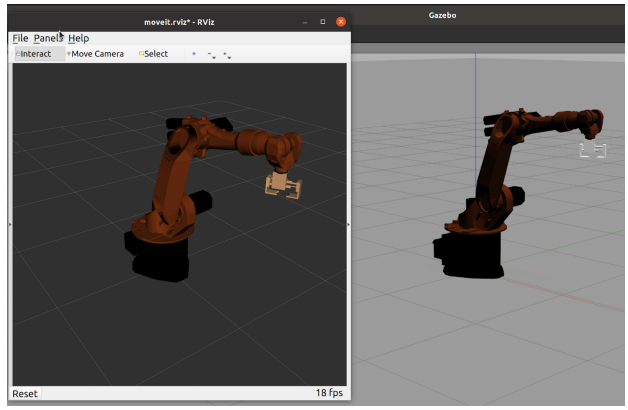
1. P0 - Home position
2. P1 - At the old filter (filter1)
3. P2 - Lift the old filter (filter1up/home)
4. P3 - Above the old filter drop position (filter1predrop)
5. P4 - Drops the old filter (filter1drop)
6. P5 - Moves to above new filter position (filter2pre)
7. P6 - At the new filter position (filter2)
8. P7 - Lift the new filter to (above filter position) (filter2up)
9. P8 - Move to above cylinder position (home)

10. P9 - Place filter in cylinder (center + filter1)
11. Back to the home position (home)

P0 - Home position

The home position often refers to the resting position of the robot manipulator. In our case this position will be above the cylinder container and the last position that the manipulator will be in before moving to a new position. This position is visualized in reality in Figure 4.2a, and the simulation environment in Figure 4.2b. Figure 4.3 shows the position without the manipulator or the objects used.

(a) Start "Home" position real



(b) Start "Home" position in both RViz and Gazebo

Figure 4.2.: Robot comparison between real and simulation position "Home"

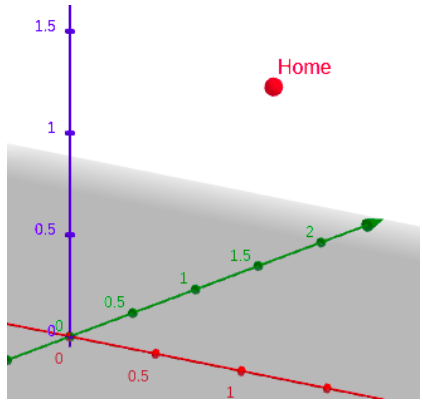
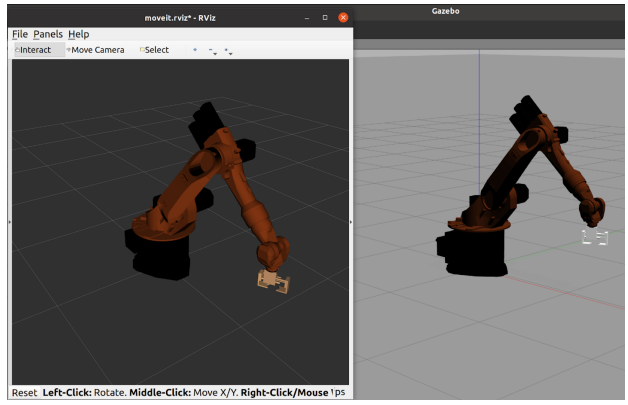


Figure 4.3.: "Home" position

P1 - At the old filter (filter1)

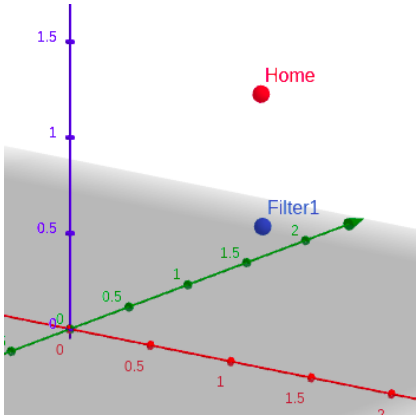
This position will refer to the manipulator picking up the old filter. The video in the digital attachment will show a segmented movement. This movement was changed after filming, and a more direct movement was chosen for practical/logical reasons. The real version in Figure 4.4a, the simulated position in Figure 4.4b. The trajectory between "home" and "filter1direct" in Figure 4.5b, and the position without the manipulator or the objects are seen in Figure 4.5.

(a) Start "filter1" position real

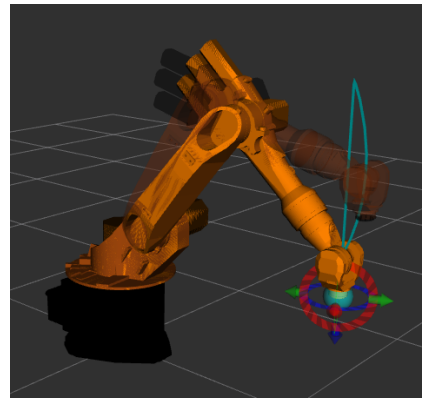


(b) Start "filter1" position in both RViz and Gazebo

Figure 4.4.: Robot comparison between real and simulation position "filter1"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter1direkte"

Figure 4.5.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

The plot shown in Figure 4.6 depicts the movement that each joint will have from the "home" position to the "filter1direct" position. The graph on the top is the actual movement for the joint, and the graph on the bottom shows the desired movements.

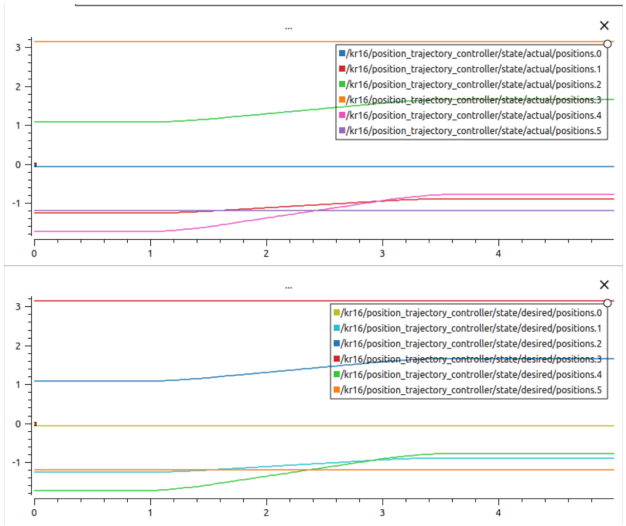


Figure 4.6.: Plot of the desired, and actual movements between "home", and "filter1direct"

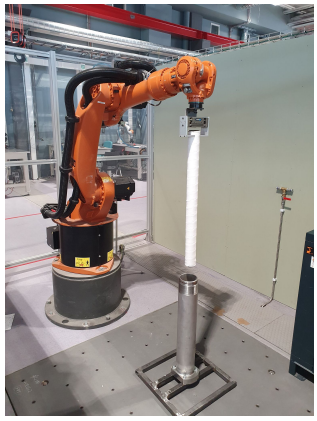
P2 - Lift the old filter (filter1up)

Under these movements, the main issue was the manipulator trajectory, since the manipulator has now picked up the filter, and moved it out of the filter container. The path it takes is important. If the command was directly up to "home" the trajectory of the manipulator would look something like in Figure 4.5b. This trajectory would damage the filter by scraping against the wall of the cylinder container. Therefore the way that this was circumvented was by incrementally adding points upward. The movement needs to only increase on the z-axis. Since the inverse kinematics problem was not solved. The movement of each joint was too tedious and imperfect to use. Therefore RVizs motion planner was used to slowly increment the movement upwards with the z-axis. Then slowly moved the point upward to create a chain shown in Table 4.1.

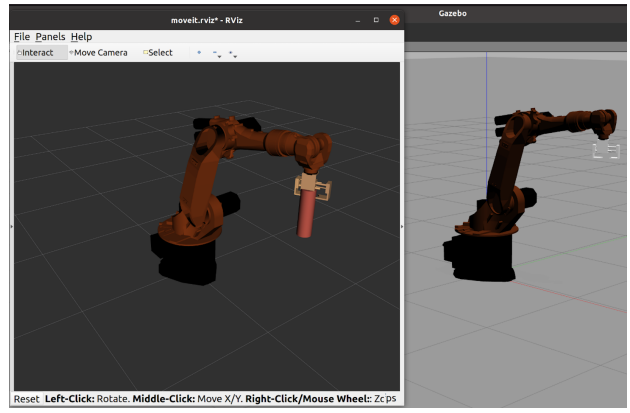
Position name	Angles for the joint(deg)							Position from RViz		
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	x	y	z
$P_{2.1}$(filterpåveined4)	-4	-54	95	180	-49	-68	0	1.14	0.08	0.76
$P_{2.2}$(filterpåveined3)	-4	-65	90	180	-65	-68	0	1.14	0.08	0.98
$P_{2.3}$(filterpåveined2)	-4	-71	82	180	-78	-68	0	1.13	0.08	1.16
$P_{2.4}$(Home)	-4	-72	62	180	-99	-67	0	1.13	0.08	1.40

Table 4.1.: Points that for a chain in point P_3

When the points where added to the code, and chained into a motion that made the KR16 performed a more linear trajectory as shown in Figure 4.8b. The points shown in Figure 4.8a.

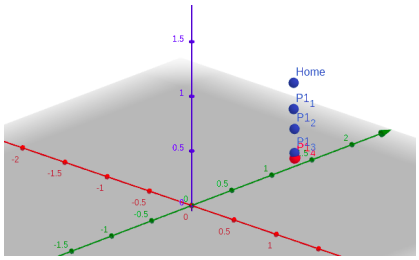


(a) Start "filter1up" position real in end-position

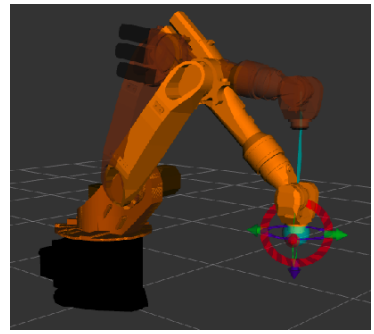


(b) Start "filter1up" position in both RViz and Gazebo in end-position

Figure 4.7.: Robot comparison between real and simulation position "filter1up"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter1up"

Figure 4.8.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

Since the manipulator will have many points to cross to complete this motion. The graph will show a start stop pattern as can be seen in Figure 4.9

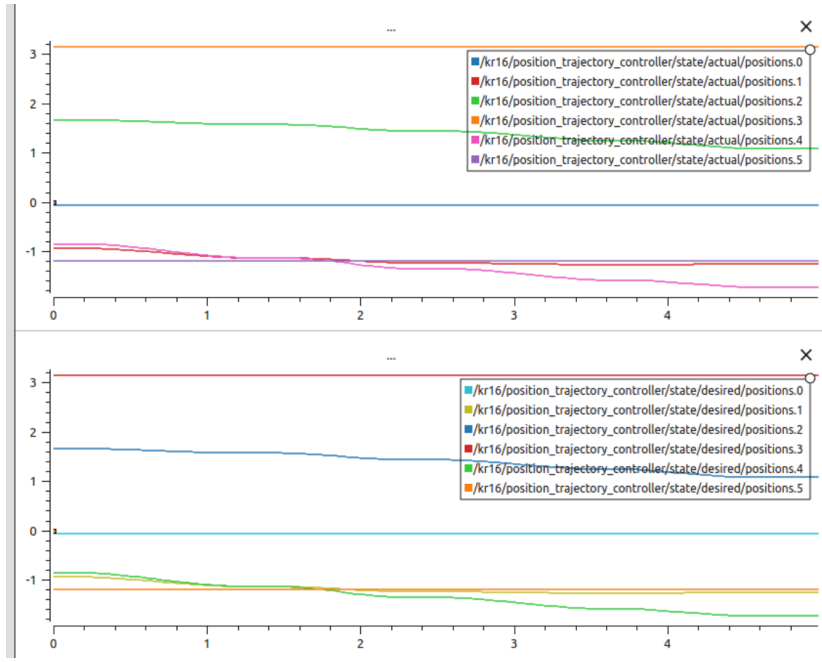
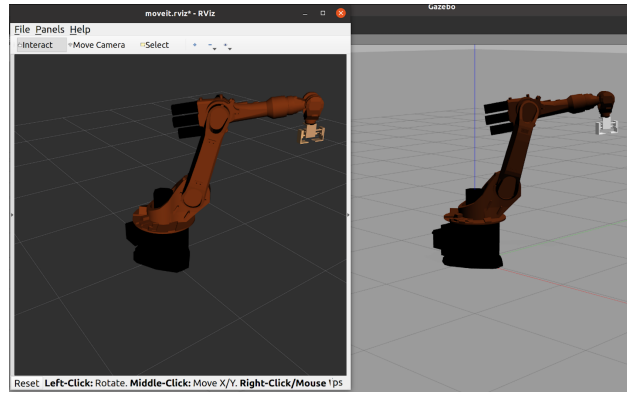


Figure 4.9.: Plot of movement between "filter1", and "filter1up"

P3 - Above the old filter drop position (filter1predrop)

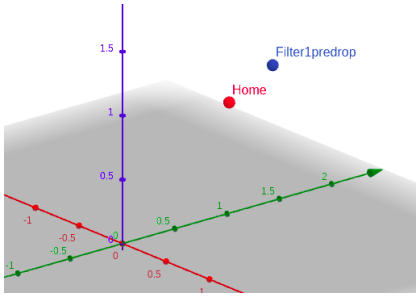
This position is the above the drop location for the old filter. This position was chosen mostly out of fear of the robot potentially hitting the wall, and therefore we wanted some seconds extra to react if the manipulator performed an undesired movement. The Figure 4.10a shows the real-life end-point for the robot, the simulation equivalent in Figure 4.10b. Trajectory in Figure 4.11b, and the position in Figure 4.11a

(a) Start "filter1predrop" position real

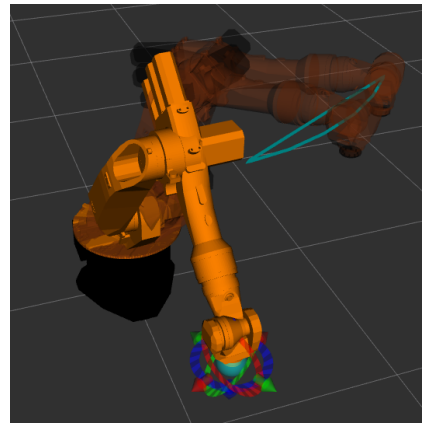


(b) Start "filter1predrop" position in both RViz and Gazebo

Figure 4.10.: Robot comparison between real and simulation position "filter1predrop"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter1predrop"

Figure 4.11.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

The plot shows the large movements between the "home" position, and "filter1predrop" positions in Figure 4.12

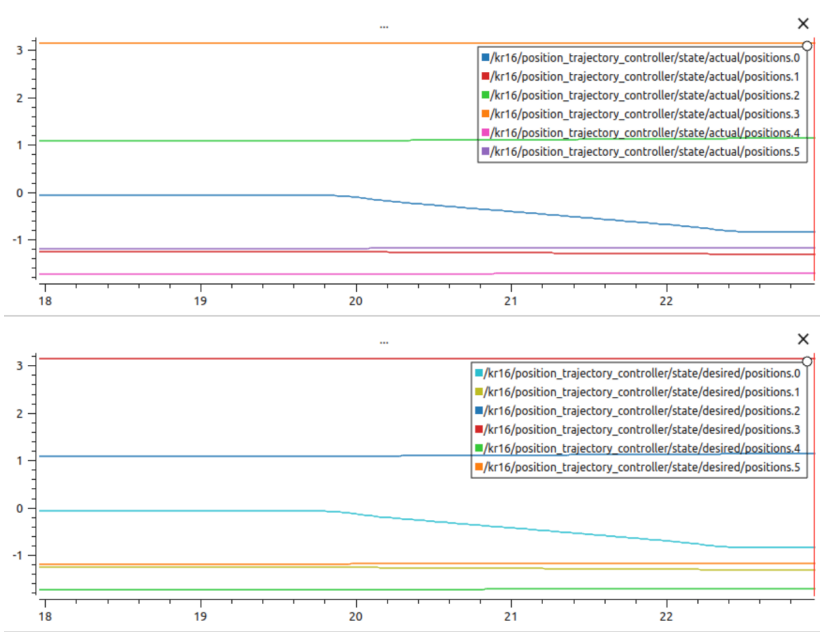
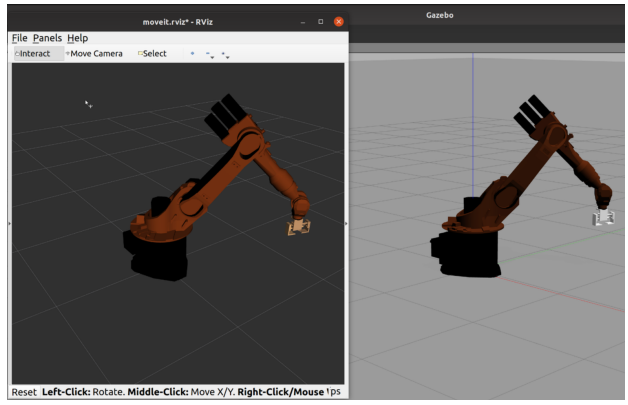


Figure 4.12.: Plot of movement between "filter1up", and "filter1predrop"

P4 - Drops the old filter (filter1drop)

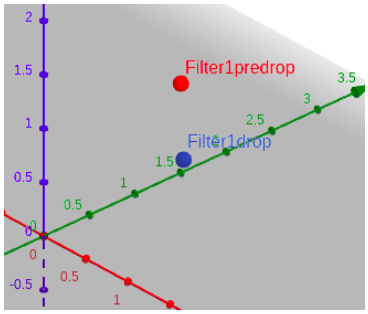
This position is below the "filter1predrop" position and is a drop position. The Figure 4.13a shows the real-life end-point for the robot, the simulation equivalent in Figure 4.13b. Trajectory in Figure 4.14b, and the position in Figure 4.14a

(a) Start "filter1drop" position real

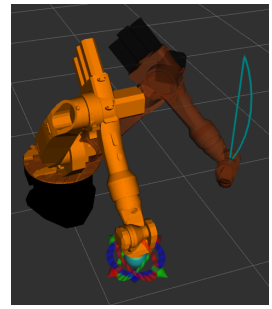


(b) Start "filter1drop" position in both RViz and Gazebo

Figure 4.13.: Robot comparison between real and simulation position "filter1drop"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter1predrop"

Figure 4.14.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

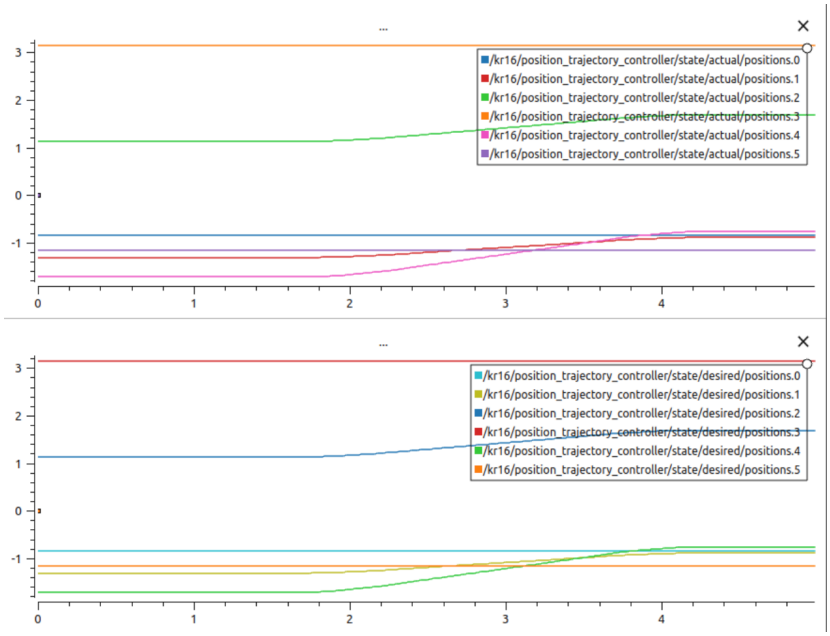
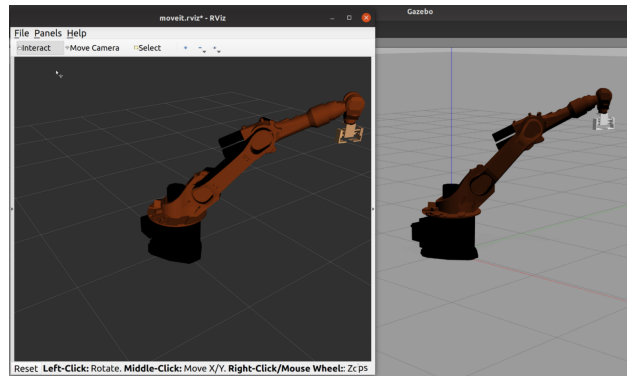


Figure 4.15.: Plot of movement between "filter1predrop", and "filter1drop"

P5 - Moves to above new filter position (filter2pre)

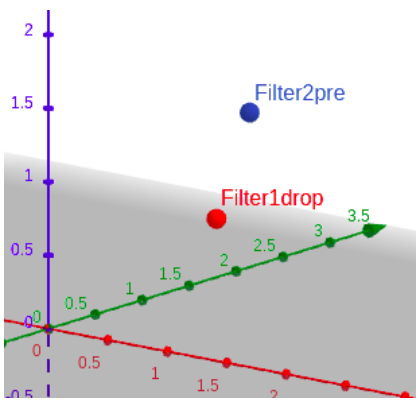
This movement can be exchanged for a direct movement to the new filter, but as this was tested the filter would get knocked over. Therefore this position was chosen for consistency.

(a) Start "filter2pre" position real

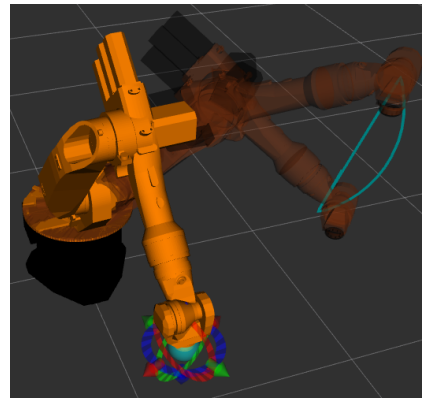


(b) Start "filter1pre" position in both RViz and Gazebo

Figure 4.16.: Robot comparison between real and simulation position "filter1pre"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter1predrop"

Figure 4.17.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

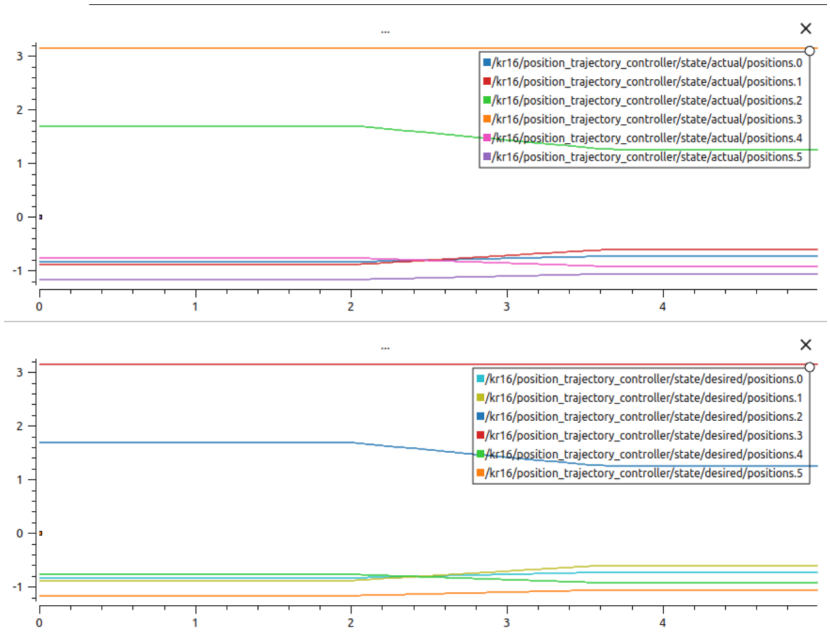
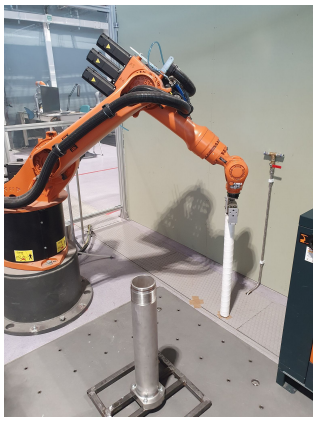


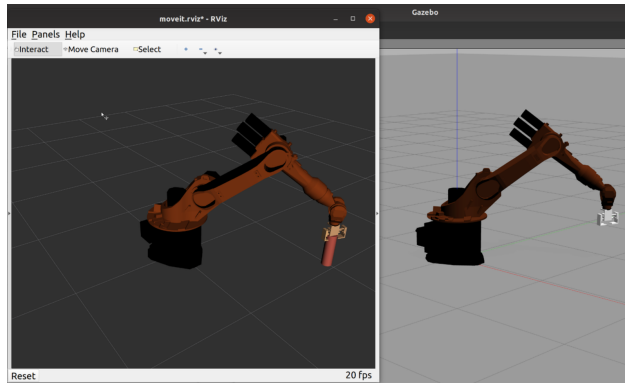
Figure 4.18.: Plot of movement between "filter1drop", and "filter1pre"

P6-P7 - Pick up new filter

This motion is about picking up the new filter, the manipulator will return to point "filter2pre". The reason for returning to the point is because it is directly above, and will minimize the possibility of swinging motion of the filter when raised. If this swinging motion was in place, it is harder for us to minimize it without interfering or waiting for the swinging motion to end. This can also change the angle of the filter, and compremise the precision needed to place the filter in the cylinder container later. Figure 4.19a, and 4.19b are the real and simulation version of the manipulator at this point.

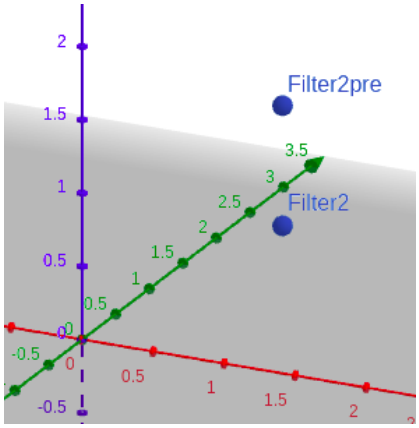


(a) Start "filter2pre" position real

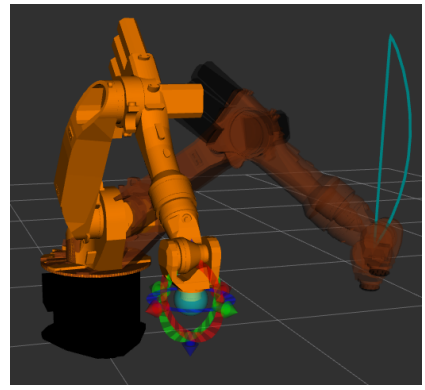


(b) Start "filter1drop" position in both RViz and Gazebo

Figure 4.19.: Robot comparison between real and simulation position "filter2"+"filter2up"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter2"+"filter2up"

Figure 4.20.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

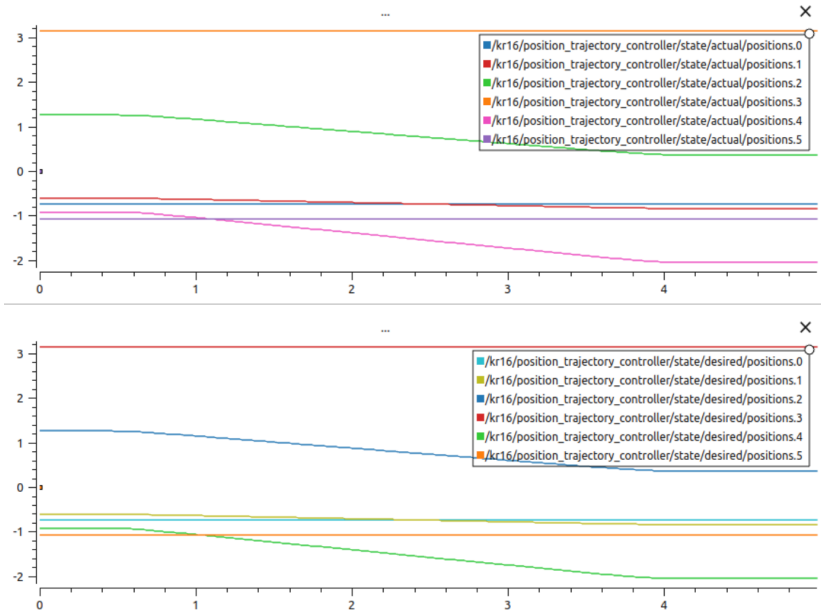


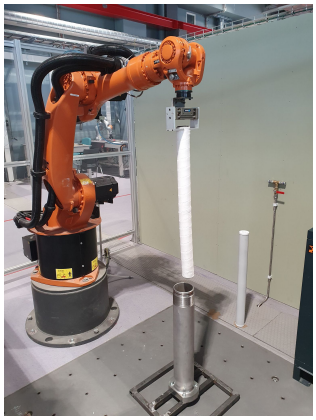
Figure 4.21.: Plot of movement between "filter2pre", and "filter2"

Since the motion from "filter2pre" to "filter2", and "filter2" to "filter2up" is the same motion only reversed. The plots, trajectory, and points are the same. These parts were combined, and Figure 4.16a is the correct depiction of the manipulator

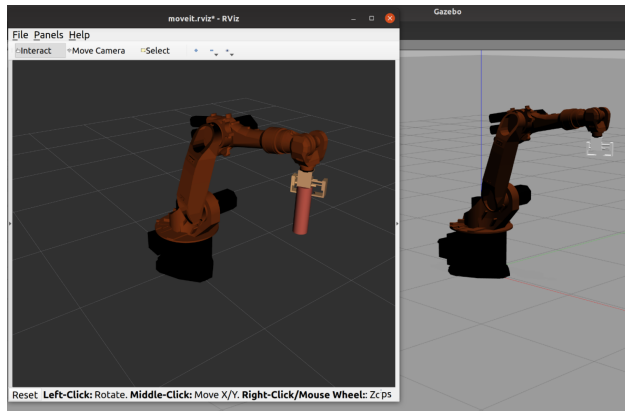
in this instance.

P8 - Move to above cylinder position (home)

As mentioned above the swinging motion of the filter is what is detrimental at this point. The gripper is great for attaching the filter, but large movements in the horizontal direction will end up causing, an angle to be formed. If this angle is too large the manipulator will in the next step smash the filter into the cylinder container, and bend it. This is shown in Figure 5.1 Therefore this point is to minimize the swinging motion and prevent massive failure to acquire.

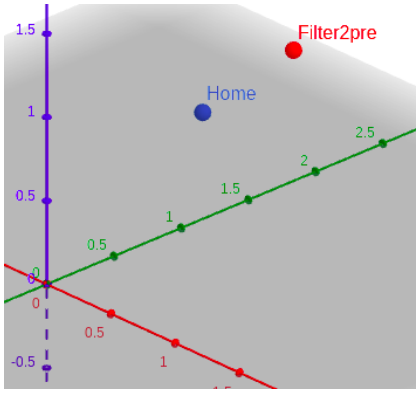


(a) Start "filter2predrop" position real

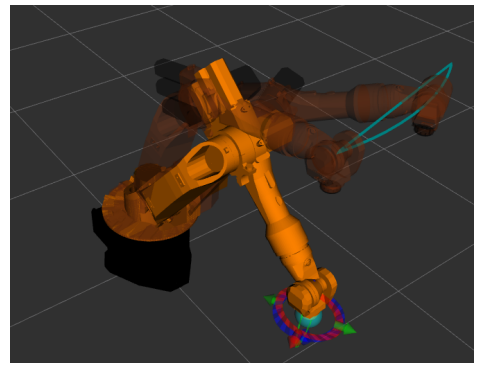


(b) Start "filter2predrop" position in both RViz and Gazebo

Figure 4.22.: Robot comparison between real and simulation position "filter2predrop"



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)



(b) Path between two points "filter2predrop"

Figure 4.23.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

The plot in Figure 4.24 depicts large distance is crossed.

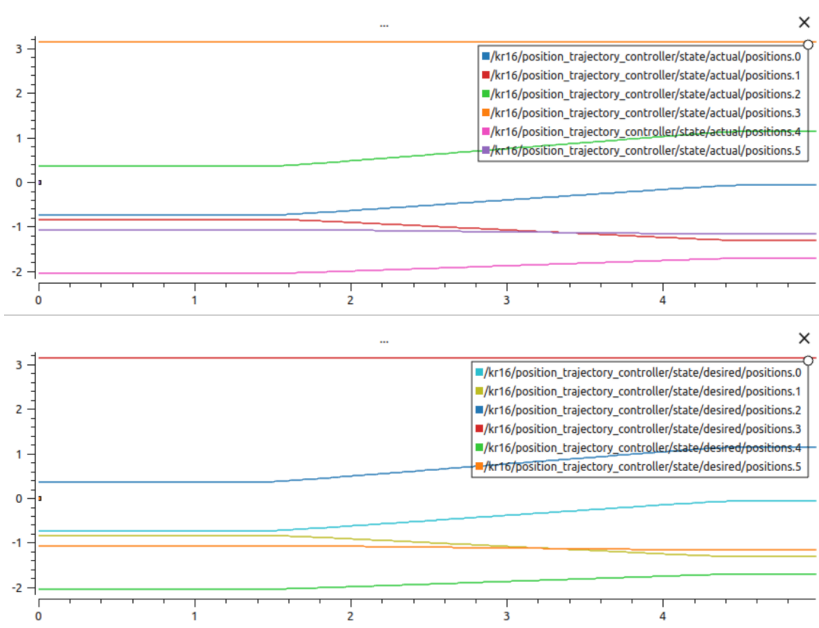


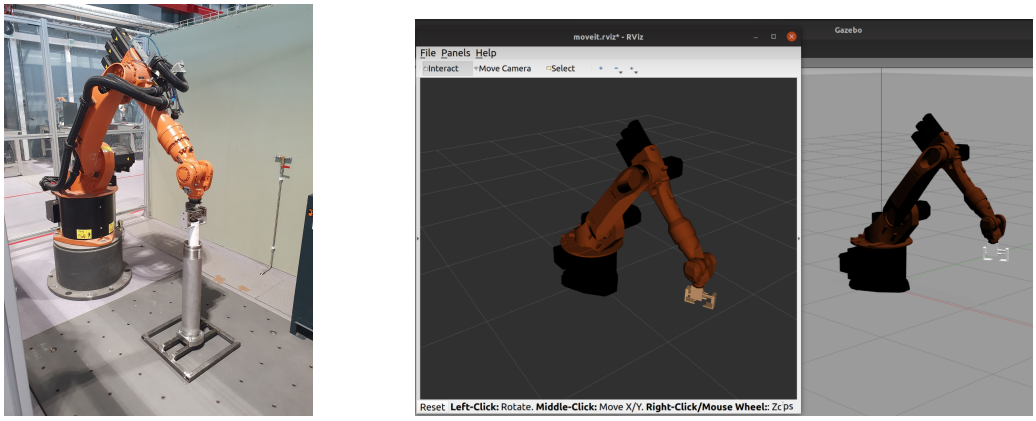
Figure 4.24.: Plot of movement between "filter2up and "filter2predrop"

P9 - Place filter in cylinder (center + filter1)

This motion is separated into two. The first is the command "center", which is used to check if the filter will be able to enter the cylinder container. This is to minimize the damage, and potentially try for another attempt. If the filter enters the cylinder container, then we can run the "filter1" command that chains different points into a more linear trajectory that the manipulator can follow. The points as shown from Table 4.2, and is reverse of the trajectory shown in P2 from the Table 4.1. The "center" was added after the Figure 5.1.

Position name	Angles for the joint(deg)							Position from RViz		
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	x	y	z
P_0(center)	-4	-72	62	180	-99	-68	0	1.13	0.08	1.4
$P_{1.1}$(filterpåveined2)	-4	-71	82	180	-78	-68	0	1.13	0.08	1.16
$P_{1.2}$(filterpåveined3)	-4	-65	90	180	-65	-68	0	1.14	0.08	0.98
$P_{1.3}$(filterpåveined4)	-4	-54	95	180	-49	-68	0	1.14	0.08	0.76
$P_{1.4}$(filterbottom)	-4	-51	95	180	-45	-68	0	1.14	0.08	0.71

Table 4.2.: Points that for a chain in point P_2

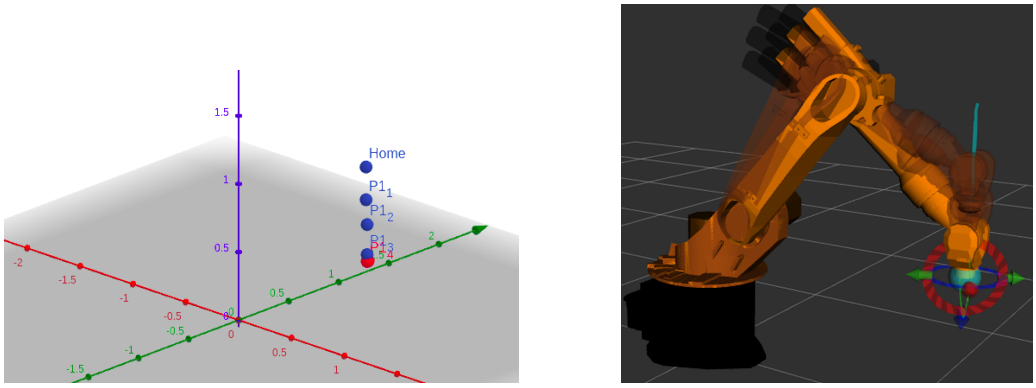


(a) Start "filter2drop" position real

(b) Start "filter2drop" position in both RViz and Gazebo

Figure 4.25.: Robot comparison between real and simulation position "filter2drop"

Figure 4.27 depicts the filter moving the short distance from "home" to "center". While Figure 4.28 depicts the filter in the chain of short distance from "center" to "filter2drop".



(a) Points KR16 will want to reach Red(startpoint), blue (endpoint)

(b) Path between two points "filter2drop"

Figure 4.26.: Showing the points the start, and end-point for the KR16 Robot, and its trajectory between those points

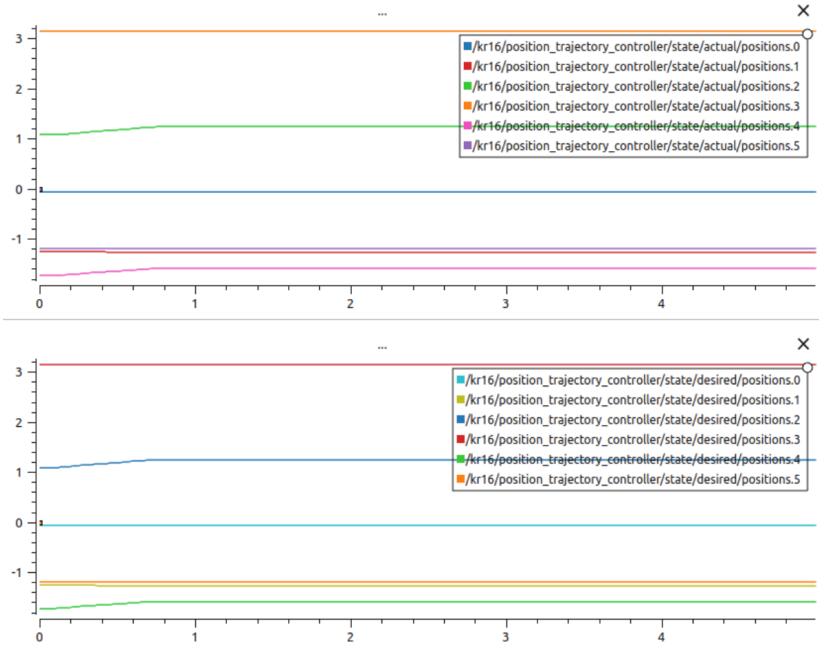


Figure 4.27.: Plot of movement from "home", and "center"

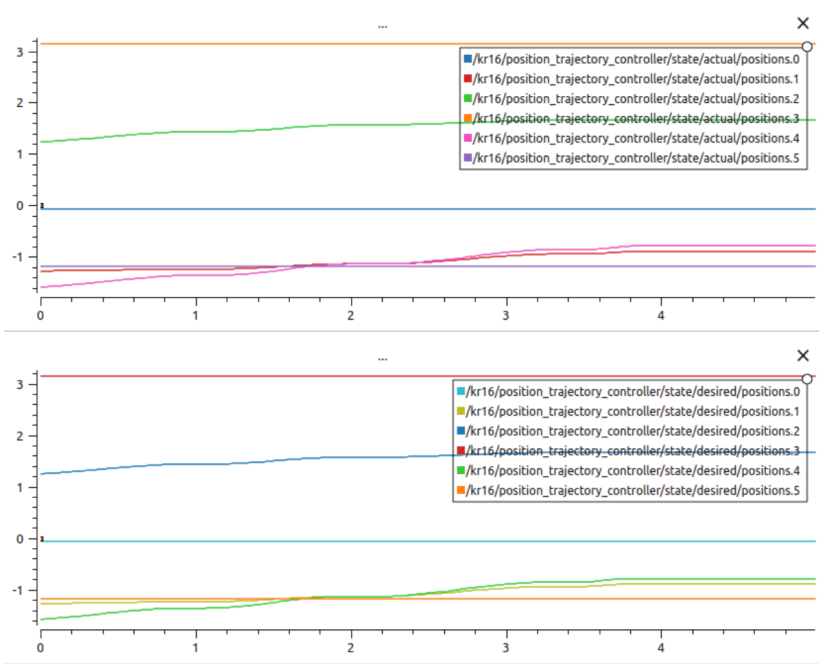


Figure 4.28.: Plots the movements from "Center" to "filter2drop"

Chapter 5.

Discussion

This thesis was constructed on the basis of a filter change through teleoperation where factors such as time delay, and precision were taken into account. This was simplified was done through encountering challenges along the way:

1. The metal rod in the middle of the cylinder container was taken out. This was done to minimize the potential for failure, because of the rod not being connected to the bottom of the cylinder. The rod ended up moving around inside the cylinder container when the slightest unprecise placement was made.

2. A gripper was not constructed in this thesis and found one already available in the laboratory. The modification made to the gripper did not stabilize the movement this can be experienced viewed in the digital attachment. This unstable movement occurs when the filter is moved over large distances or at high speed. Some of the ways that this was circumvented were by adding points and lowering the speed.

3. The filter ended up hitting the side of the cylinder container, and bending as depicted in Figure 5.1. This was caused by either the filter getting an unwanted angle, or the position of the filter container moving. The way that this was fixed was by adding another point called "center", which checks if the filter is inside the cylinder. If the filter was inside then we could proceed with the placement. If the filter was not inside we could go back, and try again with minimal damage.

4. The joystick/controller was not attempted in a laboratory setting, because of the connection issues between the joystick and the hardware. The delay between the input and the movement of the manipulator was tested in the simulation. The results can be seen in the video with the digital attachments. The movements performed were done on a DualShock controller,

but the delay from the inputs to the movement of the robot was too large. This would be more tedious, and less precise to move the manipulator this way in a laboratory setting. Therefore we settled with movement performed by a mouse.

5. The top lid for the cylinder was also removed from the experiment because the gripper dimensions did not make this task possible.

The control architecture envisioned was a system that could perform high precision for inserting the filter, and withstand the potential of time delay. Since the laboratory experiment was conducted through a direct connection between KR16 and the computer. The natural time delay that would be experienced by distance, material disturbances, etc. Does not occur. Therefore we needed to envision a system that could theoretically withstand a random amount of delay, and be able to complete this pick and place.

Some improvement mentioned above is that the point "center" was added to check if the filter would be able to enter at this angle. Thereafter we added multiple points between the "home" and "filter2drop" to minimize the elliptic trajectory. This turned the trajectory more linear. The points "center" was mostly added for damage control, and to prevent filter bending. This can be visualized in Figure 5.1, and in the second video in digital attachment.

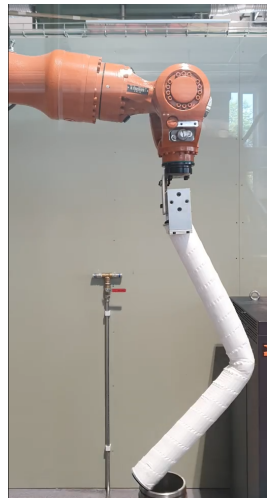


Figure 5.1.: Failure when inserting the filter in the cylinder container

The KR16 could be controlled through a predictive display where the RViz will work as a planer, and executer while Gazebo will show the robot in its environment. This was not attempted in the laboratory experiments, because of hardware

limitations. Since the connection between Gazebo, and RViz is already made this should be possible. RViz already recognizes the positions in the code, and this is translated to Gazebo.

In the RViz simulation environment that was developed, there should be some sort of shared control. Since we did not solve the inverse kinematics and used the end-effector node to drag it on the z-axis to achieve a vertical point for the linear trajector. In both "filter1up", and for "filter2drop".

Chapter 6.

Conclusions & Future work

In conclusion, the task of completing a filter exchange with the help of a KR16 manipulator has been completed. This has been done by giving general commands which indicate positions in the laboratory space. These movements were completed in RViz and then translated to Gazebo. These commands were then improved for smoother motion and improved precision when inserting the filter into the cylinder container.

Visual, and haptic augmentation: For the visual aspect in the laboratory. We as the operators had a full visual view of the KR16 action when a command was executed. For the visual aspect in the laboratory, we as the operators have the full view of the manipulator's actions and have a built-in "kill-switch" through the KR16 control pad. Which gave us the possibility of ending the movement if the manipulator performed unwanted movement. This can occur for human error in executing commands (wrong commands), and or filter not aligning properly with the cylinder container. A way that this can be implemented is by stationary cameras and a Gazebo world environment. This makes the operator able to visually see the movements in the real world and the simulated world. The operator can then compare the virtual world to the real world example, and find deviations. The second way is to give the robot a "brain", and let it compute the motions performed, and adjust accordingly by itself. This is hard to complete, and a simple solution with the camera working together with a Gazebo world should be adequate to perform the pick and place.

For the haptic augmentation, the joystick can give resistance or rumble when it detects large deviations from the path.

Gripper: The design of a gripper was not performed in this thesis, because of time constraints. The gripper did not have any function apart from holding the filter. A gripper needs to be designed where the operator can stabilize the swinging motion, or designed in a way that the motion does not occur. The

gripper needs an open, and close mechanism that can be controlled by inputs by the operator. This need to be performed to minimize the potential of failure which can be visualized in Figure 5.1. In this experiment imperfections in the filter path caused the filter to collide with the cylinder container. The motion performed by the manipulator caused the filter to bend.

Time-delay: Since we were directly connected to the robot manipulator system we did not experience any delay in the system. As we ran the program and performed the action in T2 we did not have any real-time action performed by the system. There should be improvements to the code, and with the implementation of a digital "kill-switch", the system should be run in a real-time environment. The system should have some sort of artificial delay that is in the random interval to test the system. If it can perform its action in a predictive manner. The code was designed for the possibility of delay, and operator control. That is why the commands can be executed one by one.

References

[1] Arsalan Anwar. *Part 2: 7 Simple Steps to Create and Build Yout first ROS Package*. 2021. URL: https://medium.com/swlh/7-simple-steps-to-create-and-build-our-first-ros-package-7e3080d36faa.

[2] Marian Badica, Florian Schramm, Philippe Gravez, Peter Weiss, and J Catret. "An architecture for supervising the telemanipulation of an IAUV-mounted robotic arm". In: *proceedings of the International Carpathian Control Conference*. Citeseer. 2004.

[3] 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.

[4] *Concepts*. URL: https://moveit.ros.org/documentation/concepts/ (visited on 06/03/2022).

[5] Ivar Eriksen. "Setup and interfacing of a kuka robotics lab". MA thesis. NTNU, 2017.

[6] Thomas Hulin, Michael Panzirsch, Harsimran Singh, Andre Coelho, Ribin Balachandran, Aaron Pereira, Bernhard M Weber, Nicolai Bechtel, Cornelia Riecke, Bernhard Brunner, et al. "Model-augmented haptic telemanipulation: Concept, retrospective overview, and current use cases". In: *Frontiers in Robotics and AI* 8 (2021), p. 611251.

[7] Serdar KuCuk and Zafer Bingul. "The inverse kinematics solutions of industrial robot manipulators". In: *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM'04.* IEEE. 2004, pp. 274–279.

[8] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. ISBN: 9781107156302.

[9] *MoveIt Setup Assistant*. URL: http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html (visited on 06/23/2022).

[10] *Moving robots into the future*. URL: https://moveit.ros.org/ (visited on 06/03/2022).

[11] *msg*. URL: http://wiki.ros.org/msg (visited on 06/03/2022).

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

[13] *Nodes*. URL: http://wiki.ros.org/Nodes (visited on 05/31/2022).

[14] *Open Source Robotics: Getting Started with Gazebo and ROS 2*. URL: https://www.infoq.com/articles/ros-2-gazebo-tutorial/ (visited on 05/31/2022).

[15] Luis Penin. "Teleoperation with time delay. A survey and its application to space robots". In: Dec. 2000.

[16] Norbert Piotrowski and Adam Barylski. "Modelling a 6-DOF manipulator using Matlab software". In: 2014.

[17] Rigzone. *How do ROVs work?* URL: https://www.rigzone.com/training/insight.asp?insight_id=343&c_id= (visited on 12/17/2021).

[18] *ROS Control*. URL: https://www.rosroboticslearning.com/ros-control (visited on 06/03/2022).

[19] *ROS-Robot Operating System*. URL: https://www.ros.org/ (visited on 06/03/2022).

[20] L Sciavicco and L Villani. *Robotics: modelling, planning and control. 2009*. 2009.

[21] Irfan Suvalija. *KR16-submission folder*. URL: https://github.com/irfan1108/KR16-submission-folder (visited on 06/23/2022).

[22] Irfan Suvalija. "Robot Telemanipulation for Remote Maintenance". In: (Dec. 2021), pp. 1–27.

[23] *topics*. URL: http://wiki.ros.org/Topics (visited on 06/03/2022).

[24] *What is a ROS node?* URL: https://roboticsbackend.com/what-is-a-ros-node/ (visited on 05/31/2022).

[25] *What is a ROS Service?* URL: https://roboticsbackend.com/what-is-a-ros-service/ (visited on 06/03/2022).

[26] *What is a ROS Topic?* URL: https://roboticsbackend.com/what-is-a-ros-topic/ (visited on 06/03/2022).

[27] *What is the Difference Between rviz and Gazebo?* URL: https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/ (visited on 05/31/2022).

Appendix A.

Appendix

A.1. Miscellaneous

Gripper Open mechanical drawing

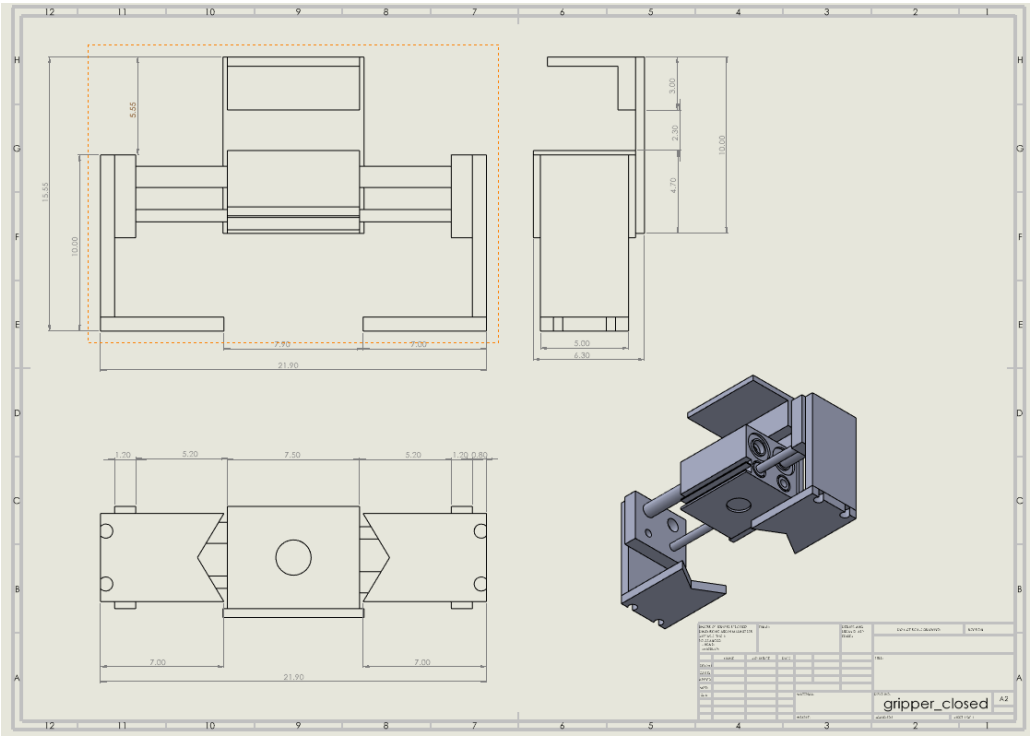


Figure A.1.: Gripper open mechanical drawings

Gripper closed mechanical drawing

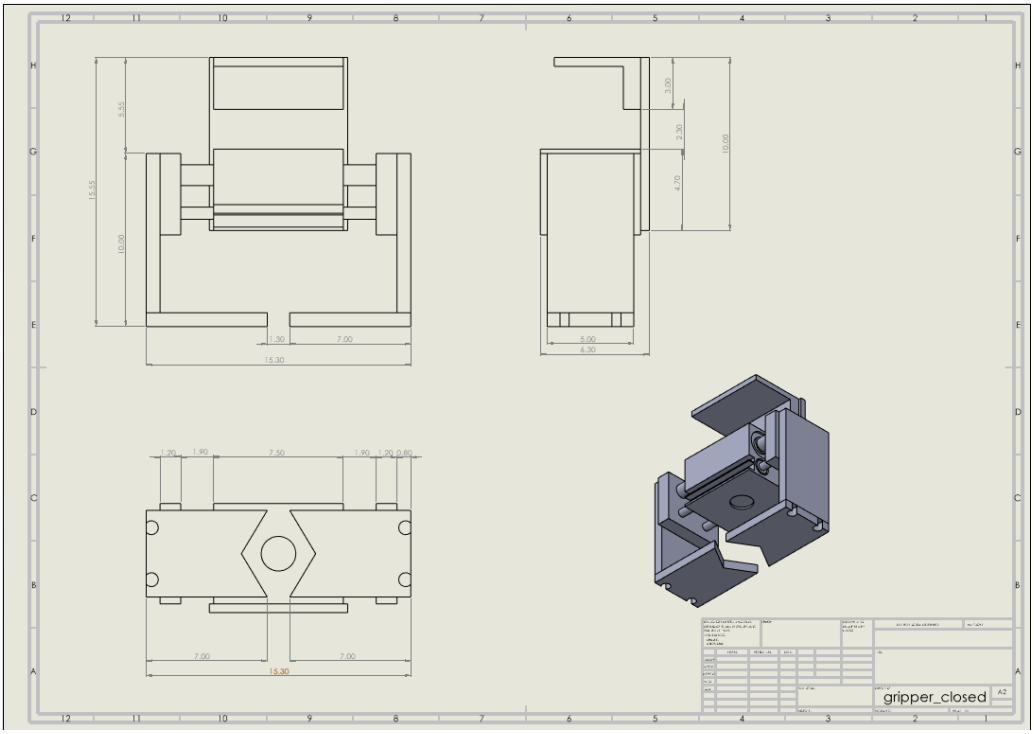


Figure A.2.: Gripper closed mechanical drawings

A.2. Setup of workspace

The full workspace is added in as digital attachment called kuka-experimental. The way to build, and use the workspace is added in the Github repository in a readme file in [21]. So for the digital attachment this is the step needed to be taken:

Then go into Ubuntu terminal, and path to the workspace and type -catkin make, and after that source the through source/devel/setup.bash.

To run in the laberatory you need to start ROS core in one terminal window with -roscore

In another you need to run start robot.launch with -roslaunch kuka kr16 support start robot.launch sim:=false (if sim:=true then you are not connected to the robot is sim:=false then you are connected)

To run the code open up another terminal window and run the python code

teleop.py

A.3. Videos

There are three video provided as digital attachments:

1. **Filter failure:** This video shows the filter bending after the filter angle collides with the cylinder container edges.
2. **Final movement:** This video show the full filter movements, and performs the filter change
3. **Teleoperation with joystick:** This video demonstrates the delay for the system. The ubuntu terminal shows the inputs from the DualShock controller.

A.4. Setup of the Kuka kr16 robot

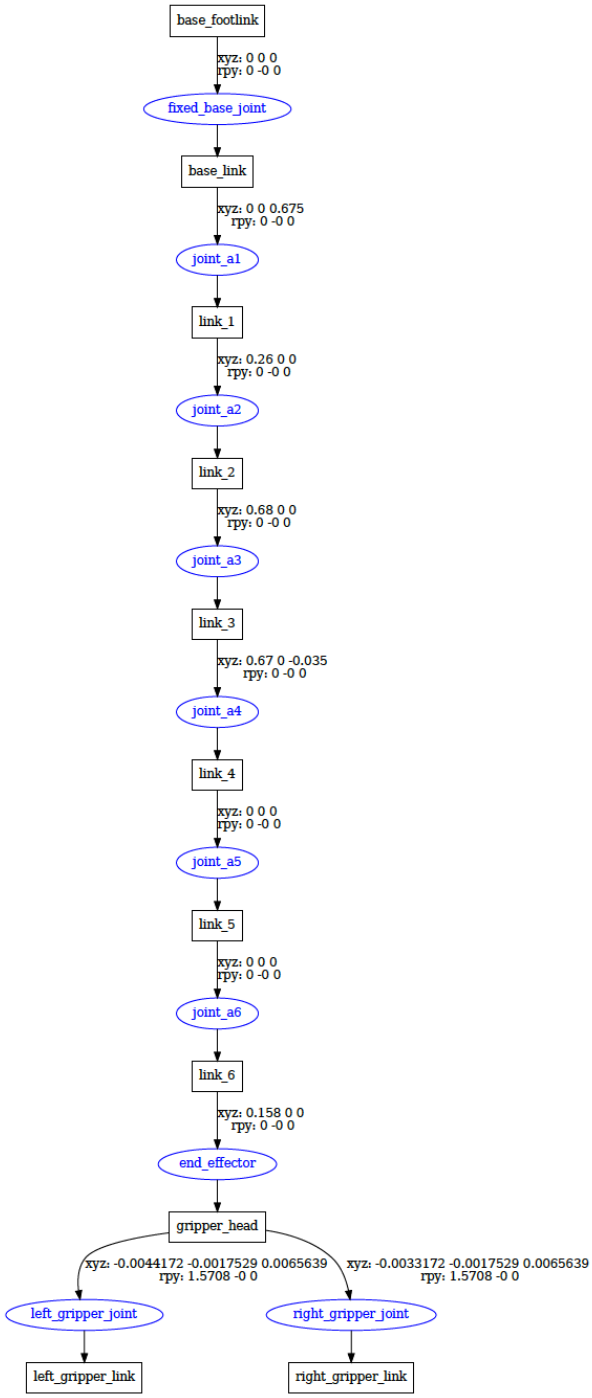


Figure A.3.: kuka kr16 with gripper

A.5. Python code

python code

[1]:
```
import numpy as np
from sympy.matrices import Matrix
from sympy import symbols, atan2, sqrt, pi, cos, sin, atan2, simplify, pprint
```

[2]:
```
# Denavit-Hartenberg needed functions

# Transformations, and important values
rads = 180/pi; deg = pi/180
x = np.array([1, 0, 0]); y = np.array([0, 1, 0]); z = np.array([0, 0, 1])

# Defines the rotation in xyz-axis
def skewm(r):
    return np.array([[0,-r[2],r[1]], [r[2],0,-r[0]], [-r[1],r[0],0]])

def Rot_x(q):
    return Matrix([[1,      0,       0],
                   [0, cos(q), -sin(q)],
                   [0, sin(q),  cos(q)]])

def Rot_y(q):
    return Matrix([[ cos(q), 0,  sin(q)],
                   [      0, 1,       0],
                   [-sin(q), 0,  cos(q)]])

def Rot_z(q):
    return Matrix([[cos(q), -sin(q), 0],
                   [sin(q),  cos(q), 0],
                   [     0,       0, 1]])

# Define the transformation matrix
def trans(Rot, p):
    return Matrix([[Rot, p],[0, 0, 0, 1]])

# Define the Denavit-Hartenberg transformation matrix
def DH_trans(alpha, q, a, d):
    return Matrix([[       cos(q),           -sin(q),              0,             a],
                   [sin(q)*cos(alpha), cos(q)*cos(alpha),-sin(alpha), -sin(alpha)*d],
                   [sin(q)*sin(alpha), cos(q)*sin(alpha), cos(alpha),  cos(alpha)*d],
                   [                0,                0,          0,             1]])
```

[3]:
```
# DH convention symbols, and print

# Creating symbols for variables
q1, q2, q3, q4, q5, q6, q7 = symbols("q1:8") # Theta_i
d1, d2, d3, d4, d5, d6, d7 = symbols("d1:8") # d_i
a0, a1, a2, a3, a4, a5, a6 = symbols("a0:7") # a_i
alpha0, alpha1, alpha2, alpha3, alpha4, alpha5, alpha6 = symbols("alpha0:7")

# Create Modified DH parameters (change with DH-values from rapport)
DH_table = {
    alpha0:     0, a0:      0, d1: 0.675, q1:       q1,         # i = 1
    alpha1: -pi/2, a1:   0.26, d2:     0, q2:       q2,         # i = 2
    alpha2:     0, a2:   0.68, d3:     0, q3:  q3-pi/2,         # i = 3
    alpha3: -pi/2, a3: -0.035, d4: 0.670, q4:       q4,         # i = 4
    alpha4:  pi/2, a4:      0, d5:     0, q5:       q5,         # i = 5
    alpha5: -pi/2, a5:      0, d6:     0, q6:       q6,         # i = 6
    alpha6:     0, a6:      0, d7: 0.115, q7:       0           # i = 7
}

# Define Modified DH transformation matrix
T_01 = DH_trans(alpha0, q1, a0, d1)
T_12 = DH_trans(alpha1, q2, a1, d2)
T_23 = DH_trans(alpha2, q3, a2, d3)
T_34 = DH_trans(alpha3, q4, a3, d4)
T_45 = DH_trans(alpha4, q5, a4, d5)
T_56 = DH_trans(alpha5, q6, a5, d6)
T_6E = DH_trans(alpha6, q7, a6, d7)


# Perform symbolic substitution with the defined symbol values
T_01 = T_01.subs(DH_table)
T_12 = T_12.subs(DH_table)
T_23 = T_23.subs(DH_table)
T_34 = T_34.subs(DH_table)
T_45 = T_45.subs(DH_table)
T_56 = T_56.subs(DH_table)
T_6E = T_6E.subs(DH_table)

# Create individual transformation matrices
T_02 = simplify(T_01@T_12)
T_03 = simplify(T_02@T_23)
T_04 = simplify(T_03@T_34)
T_05 = simplify(T_04@T_45)
T_06 = simplify(T_05@T_56)
T_0E = simplify(T_06@T_6E)
```

[4]:
```
# Before simplify
T_0E.simplify()

# Extract rotation matrices from the rotation matrices
R_01 = T_01[:3, :3]
R_02 = T_02[:3, :3]
R_03 = T_03[:3, :3]
R_04 = T_04[:3, :3]
R_05 = T_05[:3, :3]
R_06 = T_06[:3, :3]
R_0E = T_0E[:3, :3]

# Extract the position of the joints
p_01 = Matrix([T_01[0, 3], T_01[1, 3], T_01[2, 3]])
p_02 = Matrix([T_02[0, 3], T_02[1, 3], T_02[2, 3]])
p_03 = Matrix([T_03[0, 3], T_03[1, 3], T_03[2, 3]])
p_04 = Matrix([T_04[0, 3], T_04[1, 3], T_04[2, 3]])
p_05 = Matrix([T_05[0, 3], T_05[1, 3], T_05[2, 3]])
p_06 = Matrix([T_06[0, 3], T_06[1, 3], T_06[2, 3]])
p_0E = Matrix([T_0E[0, 3], T_0E[1, 3], T_0E[2, 3]])
```

python code answers

[5]:
```
# Test if right
substitute_values = {q1: 0, q2: 0, q3: 0, q4: 0, q5: 0, q6:0, q7: 0}

pprint(T_06.evalf(subs=substitute_values))
```

```
⎡ 0    0    1.0  1.61⎤
⎢                    ⎥
⎢ 0   -1.0   0    0   ⎥
⎢                    ⎥
⎢1.0   0     0   0.64⎥
⎢                    ⎥
⎣ 0    0     0   1.0 ⎦
```

[6]:
```
# P1 - Holde filter 1st time

substitute_values = {q1: -4*deg, q2: -51*deg, q3: 95*deg, q4: 180*deg, q5: -45*deg, q6:-67*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.14279190780787  ⎤
⎢                   ⎥
⎢-0.0799117948278461⎥
⎢                   ⎥
⎣ 0.712861252571359 ⎦
```

[7]:
```
# P2 - Home / filter1up

substitute_values = {q1: -4*deg, q2: -72*deg, q3: 62*deg, q4: 180*deg, q5: -99*deg, q6:-67*deg, q7: 0}

pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.13326312386386  ⎤
⎢                   ⎥
⎢-0.0792454773449433⎥
⎢                   ⎥
⎣ 1.40359443876212  ⎦
```

[8]:
```
# P3- Filter_1_preddrop

substitute_values = {q1: -48*deg, q2: -75*deg, q3: 65*deg, q4: 180*deg, q5: -98*deg, q6:-67*deg, q7: 0}

pprint(p_06.evalf(subs=substitute_values))
```

```
⎡0.737312225813153 ⎤
⎢                  ⎥
⎢-0.818868185324656⎥
⎢                  ⎥
⎣ 1.41370556955798 ⎦
```

[9]:
```
# P4 - filter 1 drop

substitute_values = {q1: -48*deg, q2: -51*deg, q3: 97*deg, q4: 180*deg, q5: -44*deg, q6:-67*deg, q7: 0}

pprint(p_06.evalf(subs=substitute_values))
```

```
⎡0.754901160016042 ⎤
⎢                  ⎥
⎢-0.838402675772891⎥
⎢                  ⎥
⎣0.697188544597779 ⎦
```

[10]:
```
# P5 - filter 2 hente

substitute_values = {q1: -42*deg, q2: -35*deg, q3: 72*deg, q4: 180*deg, q5: -53*deg, q6:-61*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡0.989159691758713⎤
⎢                 ⎥
⎢-0.89064338691595⎥
⎢                 ⎥
⎣0.633863668355184⎦
```

[11]:
```
# P6 - filter 2 up

substitute_values = {q1: -42*deg, q2: -48*deg, q3: 21*deg, q4: 180*deg, q5: -117*deg, q6:-61*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡0.986801838128256 ⎤
⎢                  ⎥
⎢-0.888520365971224⎥
⎢                  ⎥
⎣ 1.45332688780353 ⎦
```

[12]:
```
# P7 - home

substitute_values = {q1: -4*deg, q2: -72*deg, q3: 62*deg, q4: 180*deg, q5: -99*deg, q6:-67*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.13326312386386  ⎤
⎢                   ⎥
⎢-0.0792454773449433⎥
⎢                   ⎥
⎣ 1.40359443876212  ⎦
```

[13]:
```
# P8 - Tilbake til filter drop new

substitute_values = {q1: -4*deg, q2: -51*deg, q3: 95*deg, q4: 180*deg, q5: -45*deg, q6:-67*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.14279190780787  ⎤
⎢                   ⎥
⎢-0.0799117948278461⎥
⎢                   ⎥
⎣ 0.712861252571359 ⎦
```

[14]:
```
# Home-filter gripper chain

# P1.1
substitute_values = {q1: -4*deg, q2: -71*deg, q3: 82*deg, q4: 180*deg, q5: -78*deg, q6:-68*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.12963977789656  ⎤
⎢                   ⎥
⎢-0.0789921083128817⎥
⎢                   ⎥
⎣ 1.15575365308458  ⎦
```

[15]:
```
# Home-filter gripper chain

#P1.2
substitute_values = {q1: -4*deg, q2: -65*deg, q3: 90*deg, q4: 180*deg, q5: -65*deg, q6:-68*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.13703846409362  ⎤
⎢                   ⎥
⎢-0.0795094748512123⎥
⎢                   ⎥
⎣ 0.976414287272371 ⎦
```

[16]:
```
# Home-filter gripper chain

# P1.3
substitute_values = {q1: -4*deg, q2: -54*deg, q3: 95*deg, q4: 180*deg, q5: -49*deg, q6:-68*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

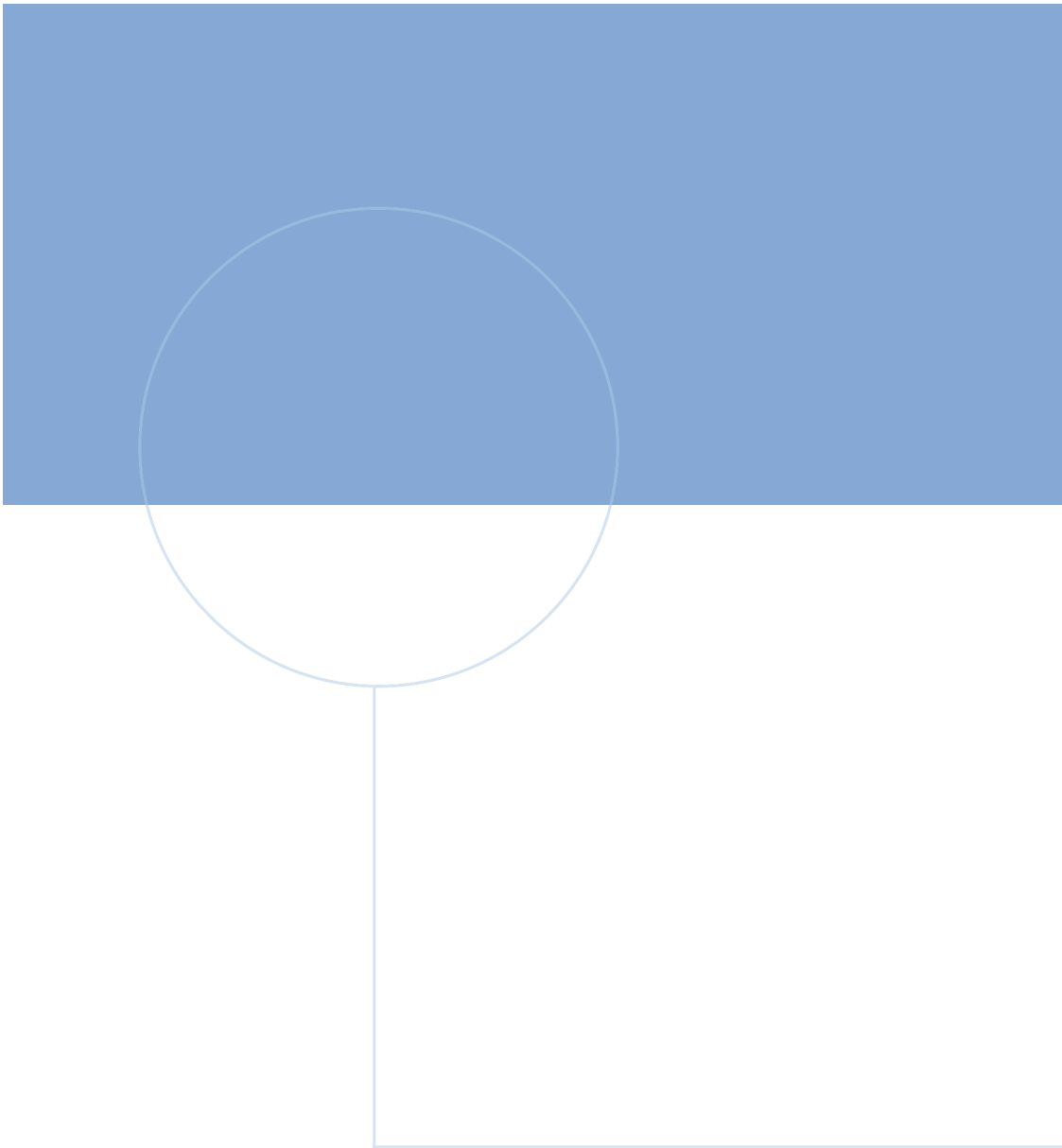
```
⎡ 1.13960452618768  ⎤
⎢                   ⎥
⎢-0.0796889113926991⎥
⎢                   ⎥
⎣ 0.759157171443527 ⎦
```

[17]:
```
# Home-filter gripper chain

# P1.4
substitute_values = {q1: -4*deg, q2: -51*deg, q3: 95*deg, q4: 180*deg, q5: -45*deg, q6:-68*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.14279190780787  ⎤
⎢                   ⎥
⎢-0.0799117948278461⎥
⎢                   ⎥
⎣ 0.712861252571359 ⎦
```

[18]:
```
# Center

substitute_values = {q1: -4*deg, q2: -72*deg, q3: 62*deg, q4: 180*deg, q5: -99*deg, q6:-68*deg, q7: 0}
pprint(p_06.evalf(subs=substitute_values))
```

```
⎡ 1.13326312386386  ⎤
⎢                   ⎥
⎢-0.0792454773449433⎥
⎢                   ⎥
⎣ 1.40359443876212  ⎦
```



NTNU

Norwegian University of
Science and Technology