TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

BURAK TEKE

REAL–TIME TARGET TRACKING AND FOLLOWING WITH UR5 COLLABORATIVE ROBOT ARM

Master of Science thesis

Examiners: Assoc. Prof. Minna Lanz,
Assoc. Prof. Joni–Kristian Kämäräinen
Examiners and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 2nd May 2018

ABSTRACT

BURAK TEKE: Real–time Target Tracking and Following
with UR5 Collaborative Robot Arm
Tampere University of Technology
Master of Science thesis, 70 pages, 6 Appendix pages
May 2018
Master's Degree Programme in Signal Processing
Major: Robotics
Examiners: Assoc. Prof. Minna Lanz, Assoc. Prof. Joni–Kristian Kämäräinen
Keywords: Universal Robot–5, Target tracking, Human–robot interaction, Collaborative robots, Trajectory planning, ROS, Microsoft Kinect

The rise of the camera usage and their availability give opportunities for developing robotics applications and computer vision applications. Especially, recent development in depth sensing (e.g., Microsoft Kinect) allows development of new methods for Human Robot Interaction (HRI) field. Moreover, Collaborative robots (co–bots) are adapted for the manufacturing industry.

This thesis focuses on HRI using the capabilities of Microsoft Kinect, Universal Robot–5 (UR5) and Robot Operating System (ROS). In this particular study, the movement of a fingertip is perceived and the same movement is repeated on the robot side. Seamless cooperation, accurate trajectory and safety during the collaboration are the most important parts of the HRI. The study aims to recognize and track the fingertip accurately and to transform it as the motion of UR5. It also aims to improve the motion performance of UR5 and interaction efficiency during collaboration.

In the experimental part, nearest–point approach is used via Kinect sensor's depth image (RGB–D). The approach is based on the Euclidean distance which has robust properties against different environments. Moreover, Point Cloud Library (PCL) and its built–in filters are used for processing the depth data. After the depth data provided via Microsoft Kinect have been processed, the difference of the nearest points is transmitted to the robot via ROS. On the robot side, MoveIt! motion planner is used for the smooth trajectory. Once the data has been processed successfully and the motion code has been implemented without bugs, 84.18% total accuracy was achieved. After the improvements in motion planning and data processing, the total accuracy was increased to 94.14%. Lastly, the latency was reduced from 3–4 seconds to 0.14 seconds.

PREFACE

The study presented in this thesis has been carried out with the UNITY team at the Mechanical Engineering and Industrial Systems (MEI) department of Tampere University of Technology (TUT), Finland from October 2017 to May 2018.

First and foremost, I would like to express my deepest gratitude to my supervisors Assoc. Prof. Minna Lanz and Assoc. Prof. Joni–Kristian Kämäräinen for their endless support and help in all kinds of issues I encountered throughout this project. Secondly, I would like to thank Asst. Prof. Roel Pieters, Antti Hietanen, Alireza Changizi, Jussi Halme and Jyrki Latokartano for their guidance during the study. I would also like to thank Jane & Aatos Erkko Foundation and Technology Industries of Finland Centennial Foundation for the support of UNITY (2016-2019) project.

Lastly, but definitely not least, I would like to express my deepest gratitude and love to my family, who taught me the value of lifelong learning and critical thinking, and to whom this work is dedicated.

Tampere, 5 April 2018

Burak Teke

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS

LIST OF ABBREVIATIONS AND SYMBOLS

AI	Artificial Intelligence
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
CPU	Central Processing Unit
C++	Programming language
D	Dimension
DH	Denavit–Hartenberg convention
DOF	Degrees of Freedom
FEMD	Finger–Earth Mover's Distance
GPU	Graphics Processing Unit
GST	Gazebo Simulation Tool
HRI	Human Robot Interaction
IK	Inverse Kinematics
IR	Infrared
ISO	International Organization for Standardization
k–NN	k–Nearest Neighbors
LTS	Long Term Support
MAPE	Mean Absolute Percentage Error
MDH	Modified Denavit–Hartenberg
OMPL	Open Motion Planning Library
P	Prismatic joint
PCL	Point Cloud Library
PERMIS	Performance Metrics for Intelligent Systems
R	Revolute joint
RGB–D	Red Green Blue and Depth data
ROR	Radius Outlier Removal
ROS	Robot Operating System
RPY	Roll–Pitch–Yaw angle
rviz	Robot Operating System visualization tool
R&D	Research and Development
SBPL	Search Based Planning Library
SCARA	Selective Compliant Articulated Robot for Assembly
SOR	Statistical Outlier Removal
SRDF	Semantic Robot Description Format
STOMP	Stochastic Trajectory Optimization for Motion Planning
TCP	Tool Center Point
TOF	Time of Flight

TS	Technical Specification
UAV	Unmanned Aerial Vehicles
URDF	Unified Robot Description Format
UR5	Universal Robot–5
USAR	Urban Search and Rescue
VG	Voxel Grid

z_i	Axis of rotation or axis of translation
θ	Joint variable for rotational joint
d	Joint variable for linear joint
R	Rotation matrix
$SO(n)$	Special Orthogonal group of order n
$o_i x_i y_i z_i$	Cartesian coordinate frame i, attached to link i
φ	First Euler angle
ϑ	Second Euler angle
ψ	Third Euler angle
ϕ	Orientation vector based on Euler angles
R_{ZYZ}	ZYZ Euler angle transformation
$o_0 x_0 y_0 z_0$	Base coordinate frame
r_{ij}	Elements of a rotation matrix
r	A unit vector that defines an axis for angle/axis representation
H	Homogeneous transformation matrix
n	Number of joints
j_i	Joint i
l_i	Link i
q	Joint position variables vector
0	Row vector (0,0,0)
A_i	Homogeneous transformation matrix of each link
T_j^i	Homogeneous transformation matrix that gives position and orientation of frame j with respect to frame i
o_j^i	Translation vector pointing from the origin of coordinate frame i to the origin of coordinate frame j
θ_i	MDH parameter, joint angle of link i
a_i	MDH parameter, link length of link i
d_i	MDH parameter, link offset of link i
α_i	MDH parameter, link twist of link i
$Rot_{\cdot,\cdot}$	Basic homogeneous transformation matrix describing a pure rotation about a given axis with a given angle

$Trans_{.,.}$	Basic homogeneous transformation matrix describing a pure translation in direction of a given axis with a given distance
h_{ij}	Elements of a desired homogeneous transformation matrix
J	Manipulator Jacobian
J_a	Analytical Jacobian
J_g	Geometric Jacobian
$\dot{q}$	Joint velocity variables vector
F	End–effector forces
τ	Joint torque vector
X	End–effector pose based on a minimal representation for the orientation of the end–effector frame with respect to the base frame
$\dot{X}$	End–effector velocity e based on a minimal representation for the orientation of the end–effector frame with respect to the base frame
$\ddot{X}$	End–effector acceleration e based on a minimal representation for the orientation of the end–effector frame with respect to the base frame
ω_j^i	Angular velocity vector of link j, expressed in coordinate frame i
v_j^i	Linear velocity vector of link j, expressed in coordinate frame i
S	Skew symmetric matrix
J_v	Linear part of geometric Jacobian
J_ω	Angular part of geometric Jacobian
ξ	Vector of body velocities (Not the derivation of a position variable)
β	Minimal representation of the the end effector frame's orientation with regard to the fixed frame.
B	Transformation matrix between the non–minimalistic angular velocity ω and the minimalistic angular velocity $\dot{\beta}$
$\mathcal{Q}$	Configuration space
q_0	Initial position for trajectory planning
q_f	Final position for trajectory planning
v_0	Initial velocity for trajectory planning
v_f	Final velocity for trajectory planning
α_0	Initial acceleration for trajectory planning
α_f	Final acceleration for trajectory planning
L^2	Euclidean norm
p	A point in Cartesian space
q	A point in Cartesian space
A_n	Current value of the reference trajectory
F_n	Current value of the test trajectory
j	Jump threshold factor parameter for MoveIt! motion planner

LIST OF PUBLICATIONS

[P1] B. Teke, M. Lanz, J. Kämäräinen, and A. Hietanen, "Real–time and Robust Collaborative Robot Motion Control with Microsoft Kinect v2," in 2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA), Oulu, Finland, Jul. 2018.

1. INTRODUCTION

In the new industrial revolution age, one of the most challenging fields of robotics is Human–Robot Interaction (HRI). Robots are designed to coexist and cooperate with humans in tasks like collaborative assembly, assisted industrial manipulation, hand guiding etc. [12, 14]. Clearly, in these collaborative tasks, the usage of camera is significant, especially the cameras that provide depth information. However, the classic 3D cameras like stereo cameras and Time–of–Flight (TOF) cameras are quite expensive [20]. At this point, Microsoft Kinect provides considerable advantages with its wide availability and lower cost than other traditional 3D cameras. Many researchers in computer science and robotics are leveraging the sensing technology to develop new ways in terms of interaction with machines and robots [24, 79]. Due to these reasons, Kinect is used in this work.

A trajectory is the path that a robot follows through as a function of time. Trajectory planning is sometimes referred to as motion planning and erroneously as path planning. Trajectory planning is distinct from path planning in that it is parametrized by time. Essentially, trajectory planning encompasses path planning in addition to planning how to move based on velocity, time and kinematics. In robotics there are inherent limitations to calculate the trajectory, especially for non–spherical wrist robots such as Universal Robot–5 (UR5). The computation time of a trajectory and data processing must be in milliseconds in tracking for seamless and safe HRI. On the other hand, human, robot and system categories should be considered first to understand HRI clearly. Taxonomy of HRI metrics were presented in [43] which identified 42 distinct metrics with 9 branches. Summarized metrics can be seen in Figure 1.1. From the consideration of these metrics, smooth robot movement is considered more trustworthy than jerky movements that is explained in Section 2.1.8. Besides that, if the movements are not recognized or planned movements, safety is endangered and comfort will be lower.

In this thesis, recognizing and tracking the target and following with UR5 and interaction efficiency are examined. In order to achieve these, the computation time of processing the depth data provided by Kinect is reduced via using downsampled filters in Point Cloud Library (PCL), after that outliers and noise in the data are

removed using statistical filters for accurate positioning. Next, the trajectory is calculated via MoveIt! motion planner. "Cartesian Path Planning" method is used for it and it has only the positions (waypoints) as inputs. After that,"Iterative Parabolic Time Parameterization" method is used. It allows us to add timestamps and velocity/acceleration values as inputs to MoveIt!. This method works better for creating smooth trajectories since we have a chance to control velocity and acceleration. For testing the methods, a trajectory is created that contained a change on the X, Y, and Z coordinate, and the data is recorded with the rosbag extension. As a result, robot follows the trajectory X,Y and Z with 85.79%, 87.14% and 79.63% by using "Cartesian Path Planning", respectively. For the "Iterative Parabolic Time Parameterization" method, robot follows the trajectory X,Y and Z with 97.32%, 93.34% and 91.77%, respectively. Percentage errors are calculated using Mean Absolute Percentage Error (MEPA) that is explained in Section 4.1.10.

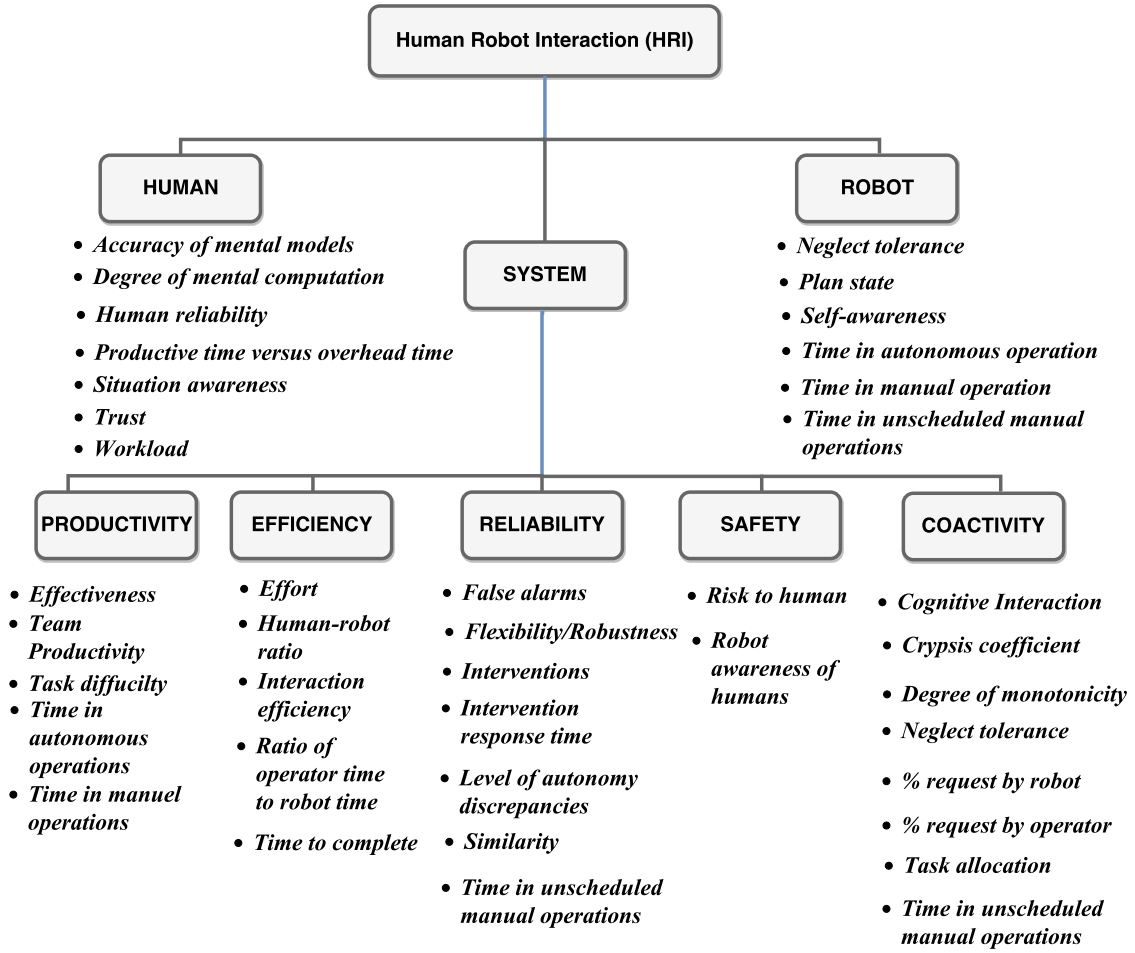


Figure 1.1 *Taxonomy of HRI metrics based on [43].*

1.1 Objectives and Outline of the Thesis

In this thesis, recognizing and tracking of the fingertip and transforming of this movement as UR5 motion are investigated. Thus, one can see the objectives of the thesis below:

- Recognizing and tracking fingertip based on robust approach by using Microsoft Kinect depth (RGB–D) data and transforming the movement of the fingertip to the Universal Robot–5.
- Develop and implement algorithms on real–time environment for smooth trajectory and select the proper trajectory planner according to accuracy.
- Test and improve different trajectory planner methods for collaborative robots and compare each other with regard to accuracy.

The thesis includes following chapters and sections. In Chapter 2, an overview of the robotics and HRI is explained. This background chapter includes principles of mathematical model of robots that are subjects of interest to this thesis in Section 2.1. HRI in Section 2.2 and related work in Section 2.3. Next, development environments and equipments as well as motion planners are mentioned in Chapter 3. In Chapter 4, data acquisition, Euclidean distance, k–Nearest Neighbors (k–NN) approach, filters in PCL environment are explained. One can see the visualization of trajectories and their accuracy in Chapter 5. Finally, problems and possible solutions are given during the implementation besides installing the libraries in Chapter 6.

1.2 Limitations

The limitations can be listed as follows:

- At the time of this study, ROS has been only running on Linux based operating systems. Therefore, it is necessary to use external libraries such as *libfreenect2* and *IAI Kinect2* for acquiring data from Kinect.
- Inverse kinematics calculation takes more time for non–spherical wrist robots such as UR5 and there are disconnections between trajectories. This leads to difficulties to plan smooth trajectories.
- MoveIt! motion planner has only two different types of trajectory calculation methods. And that's why there is not much choice.

- According to this project, Microsoft Kinect v2 can not detect the point differences below $1mm$ resolution.
- Point cloud environment has only support to C++.

1.3 Research Questions and Research Methodology

The research questions of the thesis can be listed as follows:

1. How is the fingertip location tracked in real–time environment using Kinect, point cloud, robot operating system and their tools?
2. How is Universal Robot–5 moved according to the followed fingertip in real–time environment?
3. How is the smooth robot trajectory for seamless collaboration planned by using ROS, MoveIt! and their tools?

By going out of the way of these research questions, various methods have been tested. If we elaborate on these;

- Firstly, it is focused on computation time from the first and second research questions, Voxel Grid (VG) filter is used for implementation and execution time is reduced in the real–time. And consequently, one of the sub–results, quick response to the target changes, is achieved.
- Secondly, from the second research question, nearest–point approach based on Euclidean distance is used for the aim of seamless interaction instead of using hand detection or gesture recognition. Because these detections would add extras to the calculation time. As a result of this, one of the sub–results, transforming the fingertip movement to UR5 as a motion, is achieved.
- Next, from the last research question, MoveIt! motion planner, Gazebo Simulation Tool (GST), rviz visualization tool and various libraries are used for implementation. Cartesian and parabolic time parameterization are compared to each other. It is shown that time parameterization method is working better for trajectory planning as well as for smooth path. Along with that, one of the sub–results, smooth robot trajectory for seamless collaboration using MoveIt! and ROS, is reached.

- During the application, testing and observation phase, it is determined that the nearest point is incorrectly calculated because of the noise in the depth data. In order to remove the noise, statistical and radius outlier filters are used. It is observed that statistical one is working better because it removes the outlier points with statistical information. In spite of the fact that it has considerable computation time, it is used together with VG filter. After this improvement, the nearest point is determined more accurately, and this provides us a more efficient interaction.

As a consequence of all these, with the following sub–results (quick response to the target changes, transforming the fingertip movement to UR5 as a motion and smooth robot trajectory for seamless collaboration using MoveIt! and ROS), main result (recognizing and tracking of the fingertip and transforming of this movement as UR5 motion) is achieved.

1.4 Publications and Author's Contribution

In [P1], the study related to this thesis and previous work are examined. Next, MoveIt! motion planner is used and tested in the real–time system. It is shown that the trajectory can be calculated via time parameterization method more accurately than the conventional Cartesian calculation method. In addition, the Microsoft Kinect sensor's 3D depth data can be processed in real–time by comparing the filters in the point cloud environment in terms of computation time and by observing the output in the way of using each other. As a result, while the most optimal tools and methods are used, it is shown how the trajectory is followed by UR5 [69].

2. BACKGROUND

In this chapter, what the robot means mathematically and the mathematical infrastructure needed to plan the trajectory will be given. Next, the practicalities and challenges of HRI will be addressed, and finally, the work related to this thesis will be explained.

2.1 Mathematical Overview of Robots

The robot can be considered as a mechanical system, therefore, by using mechanical inferences and conventions such as Denavit–Hartenberg (DH) convention [15], it will be addressed the principles of working of robots. Firstly, mathematical models that are used for developing and manipulating of robots are considered. It is necessary to understand how these mechanical systems work in order to plan smooth trajectory and control the position, speed and acceleration of the robot. The mathematical model is a key to achieving these kind of goals [66, 63].

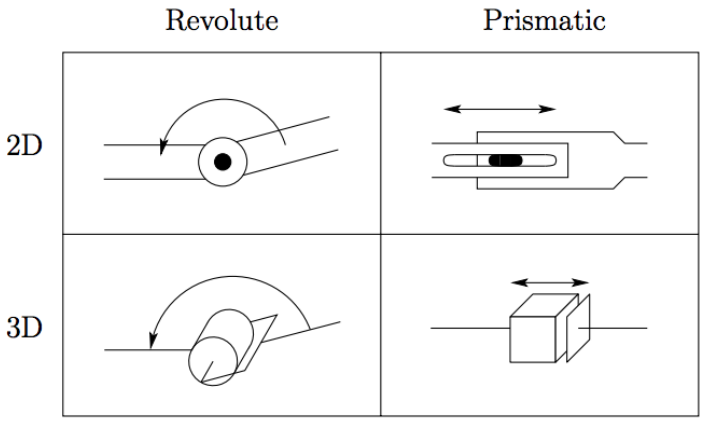


Figure 2.1 *Symbolic representations of robot joints adapted from [66].*

Robot manipulators consist of links and links are connected to a kinematic chain by joints. There are two types of joints such as revolute (rotary) joint and prismatic

(linear) joint. A revolute joint is able to rotate between two links, whereas a prismatic joint is able to move linearly between two links. The notation (R) represents revolute joints and the notation (P) represents prismatic joints that are shown in Figure 2.1 [18, 6, 48, 66].

Robotic joints are the connection between two links. If the connection of a joint is between the links i and $i+1$, then z_i is defined as the axis of rotation or axis of translation that depends on the joint types. The joint variables are θ and d for a rotational joint and for a linear joint, respectively [16, 66].

2.1.1 Robotic Systems

A robot manipulator is a series of mechanical links. A general robotic system's components are mechanical arm, external and internal sensors, power supply, computer interface, and controller. Components are shown in Figure 2.2.

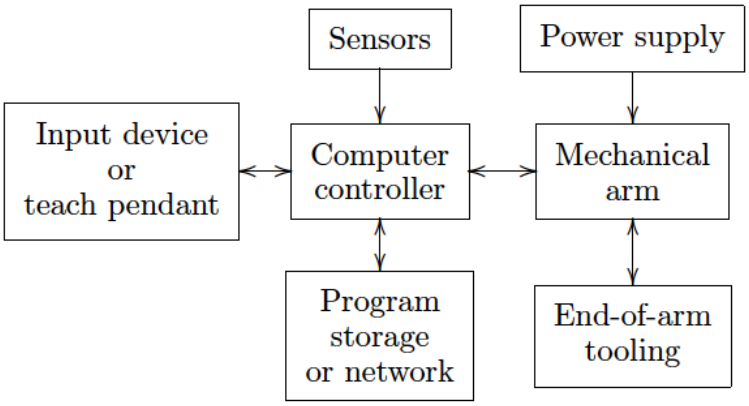


Figure 2.2 *Components of a robotic system adapted from [66].*

2.1.2 Kinematic Manipulators

One can create such a different kinematic chain by using prismatic and revolute joints, however, in practice only a few are commonly used. In this section, a few common manipulators will be shown, after that structure of the UR5 is considered.

- **Articulated manipulator (RRR):**
  All of the joints of articulated manipulator are revolute. It can also be called a revolute or an anthropomorphic manipulator [11, 29, 39, 66]. The structure and terminology are shown in Figure 2.3.

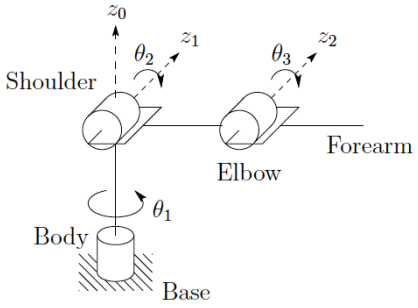


Figure 2.3 *Structure of the articulated manipulator (RRR) adapted from [66].*

- **Spherical Manipulator (RRP):**
  The first two joints of spherical manipulator are revolute and last one is prismatic [11, 29, 39, 66]. The structure and terminology are shown in Figure 2.4.

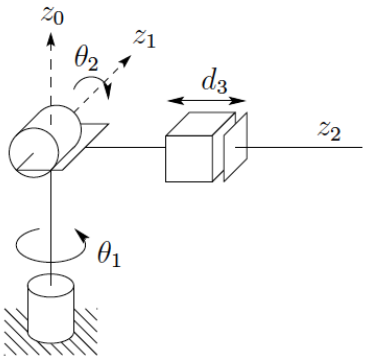


Figure 2.4 *Structure of the spherical manipulator (RRP) adapted from [66].*

- **SCARA Manipulator (RRP):**
  The SCARA arm has an RRP structure. The name of SCARA is abbreviation of Selective Compliant Articulated Robot for Assembly. In spite of the fact that its structure is an RRP, it is not similar to the spherical manipulator in terms of appearance and range [11, 29, 39, 66]. It is shown in Figure 2.5.

- **Cylindrical Manipulator (RPP):**
  The cylindrical manipulator has an RPP structure. Its axes' form is a cylindrical coordinate system [66, 63]. It is shown in Figure 2.6.

- **Cartesian manipulator (PPP):**
  All of the joints of cartesian manipulator are prismatic [11, 29, 39, 66]. It can be seen in Figure 2.7.

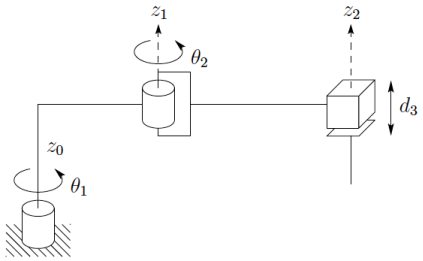


Figure 2.5 *Structure of the SCARA adapted from [66].*

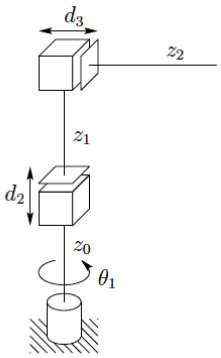


Figure 2.6 *Structure of the cylindrical manipulator adapted from [66].*

- **Universal Robot 5 (UR5):**
  All of the joints of UR5 are revolute; hence it is revolute manipulator. The structure and terminology are shown in Figure 2.8.

2.1.3 Modeling of Robots

The acquisition of direct kinematics, inverse kinematics and velocity kinematics have been clearly defined in literature for robots, especially for industrial robots [9, 63, 66, 30]. Their acquisition is based on the step–by–step procedure. Firstly, it is needed to assign the coordinate frames to the joints. To do this, it is used the method called the DH convention [15]. In this thesis, the Modified Denavit-Hartenberg (MDH) convention will be used to describe the basic kinematic calculations. MDH and DH differ from each other in terms of the coordinate system attachment to the links and the order of the performed transformations. To make it clear, one can see the details in Figure 2.9. The aim of the DH convention is to simplify and to standardize the assignment of the coordinate frames as well as to create homogeneous transformation matrices. The positions, orientations, rotations and translations between assigned coordinate frames are represented by rigid motions

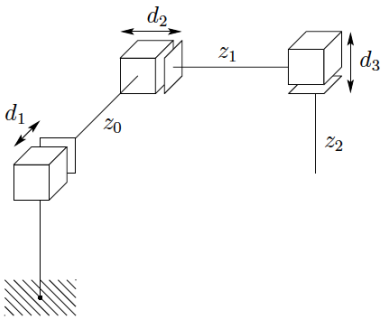


Figure 2.7 *Structure of the cartesian manipulator adapted from [66].*

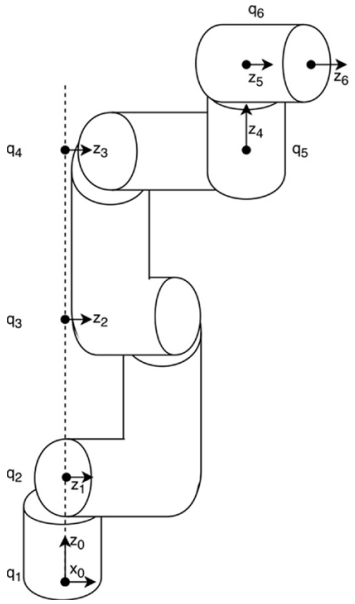


Figure 2.8 *Structure of the Universal Robot 5 adapted from [42].*

and homogeneous transformations [66, 30]. By using homogeneous transformation matrices, it is easy to obtain the direct kinematics. Moreover, the manipulator Jacobian is defined to derive the velocity kinematics.

When UR5 is considered with the all of these informations, the end–effector structure of it is different from other common industrial robots. Thus, this difference cause limitation for calculating the inverse kinematics in real–time. As a result of this limitation, it takes more time to create smooth trajectory planning. The details are explained in Section 2.1.6.

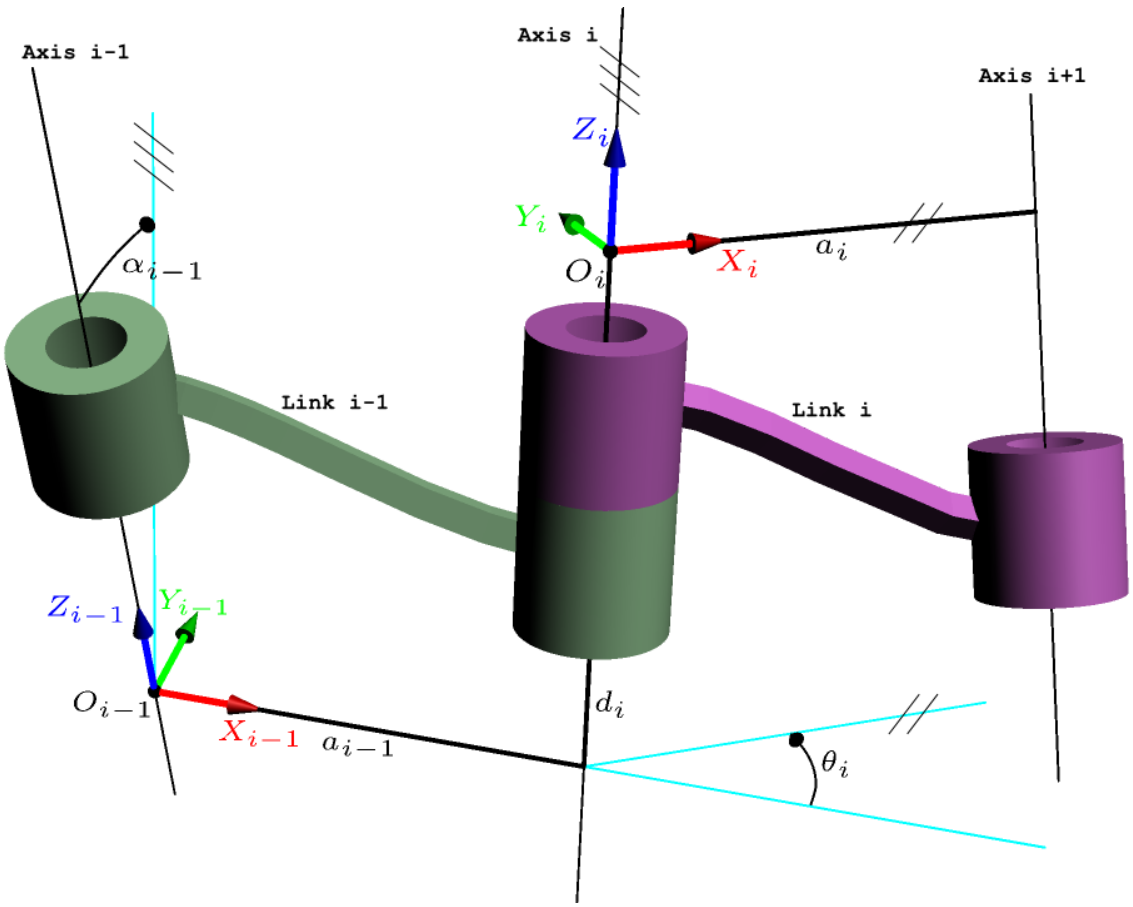


Figure 2.9 *Modified Denavit–Hartenberg convention parameters. Using this parameters, one can calculate the rotation and orientation of the robot with respect to the base frame. It helps us to derive velocity and acceleration of the robot for trajectory planning.*

2.1.4 Rigid Motions and Homogeneous Transformation

As mentioned in the previous section, robot has prismatic and/or revolute joints. These joints are connected to each other with rigid bodies (*links*). If one wanted to define the robot kinematics in a very simple way, the answer would be "relations between the sequential coordinate frames that are assigned to the joints of robot". The positions and orientations of rigid objects are calculated with the help of the rigid motions and homogeneous transformation approach. 3D space geometry and rigid motion geometry are the key in all kind of robotic manipulation [8, 63, 66, 59].

The first frame of the robot is called the base, the inertial or the fixed frame, and the last frame of the robot is called the end–effector. To manipulate the robot in 3D space, it is required to calculate the end–effector position and orientation with relative to the inertial frame. The motion relationship between these two frames is obtained by combining the homogeneous transformation matrices of all links according to the previous link. Rigid motions and homogeneous transformations are the

key to define the relationship. One can simplify the operations of translation and rotation of a manipulator into a single matrix multiplication by using homogeneous transformations [53, 65, 22, 34, 66].

$(\boldsymbol{d}, \boldsymbol{R})$ means that a rigid motion is an ordered pair such as $\boldsymbol{d} \in \mathbb{R}^3$ and $\boldsymbol{R} \in SO(3)$. A translation vector denoted as $\boldsymbol{d}$ and a rotation matrix denoted as $\boldsymbol{R}$ [66, 63]. $SO(3)$ is used to define the 3D rotation group and it is the abbreviation of the Special Orthogonal group of order three. All of the rotation group are under the operation of 3D Euclidean space ($\mathbb{R}^3$). For any $\boldsymbol{R} \in SO(n)$, the rotation matrix has the following properties [66, 34, 77, 19, 61].

- $\boldsymbol{R}^T = \boldsymbol{R}^{-1}$ means that rotation matrix is an orthogonal matrix,
- $\det \boldsymbol{R} = 1$ means a geometric interpretation would be that the area does not change.

The orientation of one coordinate frame according to the another and the coordinates transforming from one to the another are represented by rotation matrices. Sequential rotations like a rotational transformation of the frame $o_i x_i y_i z_i$ to the frame $o_j x_j y_j z_j$ and further to the frame $o_k x_k y_k z_k$ are obtained by Equation (2.1) [66, 63, 30].

$$\boldsymbol{R}^i_k = \boldsymbol{R}^i_j \boldsymbol{R}^j_k \tag{2.1}$$

Euler–angle representation, the roll–pitch–yaw representation, and the axis/angle representation are common rotation representations [66, 5]. They are explained in the following section.

Euler Angle

The orientation of a frame $o_j x_j y_j z_j$ according to a frame $o_i x_i y_i z_i$ is derived by using $\phi = [\varphi, \vartheta, \psi]^T$ that is called Euler angles, it is a vector of three angles [8, 63, 66]. After three successive rotations, a rotation matrix is obtained:

1. φ represents rotation of z-axis,
2. ϑ represents rotation of y'-axis,
3. ψ represents rotation of z''-axis.

Result of the rotation matrix is generated via multiplication of the matrices of elementary rotation[1]. The matrix $\boldsymbol{R_{ZYZ}}$ is called in literature as a ZYZ–Euler angle transformation. The following Equation (2.2) shows the connection between the Euler angles and the rotation matrix $\boldsymbol{R}$ [8, 63, 66].

$$\begin{aligned}\boldsymbol{R_{ZYZ}} = \boldsymbol{R(\phi)} =& \boldsymbol{R_z}(\varphi)\boldsymbol{R_{y'}}(\vartheta)\boldsymbol{R_{z''}}(\psi) \\ =& \begin{bmatrix} c_\varphi & -s_\varphi & 0 \\ s_\varphi & c_\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\vartheta & 0 & s_\vartheta \\ 0 & 1 & 0 \\ -s_\vartheta & 0 & c_\vartheta \end{bmatrix} \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ =& \begin{bmatrix} c_\varphi c_\vartheta c_\psi - s_\varphi s_\psi & -c_\varphi c_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta \\ s_\varphi c_\vartheta c_\psi + c_\varphi s_\psi & -s_\varphi c_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta \\ -s_\vartheta c_\psi & s_\vartheta s_\psi & c_\vartheta \end{bmatrix}\end{aligned} \tag{2.2}$$

Deriving the Euler angle $(\varphi, \vartheta, \psi)$ is crucial to solve the inverse kinematics that is shown in Equation (2.2). Next, the general rotation matrix can be seen in Equation (2.3) [8, 63, 66].

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.3}$$

In addition, the arctangent function is used to find the Euler angles. Atan2 is a function that consists of fragmented functions which are written by each quadrant using the arctangent. Notation of Atan2(y, x) means the arctangent function of two variables[2].

To summarize, the solution can be seen in Equation (2.4) [66, 63].

$$\begin{aligned}\varphi &= \text{Atan2}(r_{23}, r_{13}) \\ \vartheta &= \text{Atan2}\left(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right) \\ \psi &= \text{Atan2}(r_{32}, -r_{31})\end{aligned} \tag{2.4}$$

Equation (2.5) [66, 63] shows another solution that can be derived by choosing φ in

[1] In Equation (2.2); c_φ is the abbreviations for $\cos\varphi$ and s_φ is the abbreviations for $\sin\varphi$ *etc.*

[2] Atan2(y, x) computes the arctangent of y/x. It uses the sign of the variables to identify which region the output angle belongs to. Therefore, the correct determination of an angle is obtained in the interval $[0, 2\pi]$.

$[-\pi, 0]$.

$$\begin{aligned}\varphi &= \text{Atan2}(-r_{23}, -r_{13}) \\ \vartheta &= \text{Atan2}\left(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\right) \\ \psi &= \text{Atan2}(-r_{32}, r_{31})\end{aligned} \tag{2.5}$$

Roll–Pitch–Yaw (RPY) Angles

One of the another method to derive the rotation matrix as a product of consecutive rotations about the fixed frame $(o_0x_0y_0z_0)$ is roll, pitch and yaw angles [8, 63, 66].

1. ψ represents a yaw through x_0,
2. ϑ represents a pitch through y_0,
3. φ represents a roll through z_0.

The sequential rotations are respect to the principal axis. The result transformation matrix can be seen in Equation (2.6) [8, 63, 66].

$$\begin{aligned}\boldsymbol{R_{ZYX}} = \boldsymbol{R(\phi)} =& \boldsymbol{R_z}(\varphi)\boldsymbol{R_y}(\vartheta)\boldsymbol{R_x}(\psi) \\ =& \begin{bmatrix} c_\varphi & -s_\varphi & 0 \\ s_\varphi & c_\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\vartheta & 0 & s_\vartheta \\ 0 & 1 & 0 \\ -s_\vartheta & 0 & c_\vartheta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{bmatrix} \\ =& \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix}\end{aligned} \tag{2.6}$$

For the inverse extraction to obtain the φ, ϑ, ψ angles, the method in Section 2.1.4 can be used.

Angle and Axis

Not all of the rotation cases are related to the fixed frame. It is usually interested in a rotation according to an arbitrary axis in space. It has numerous advantageous especially in the problem of trajectory planning as well as the derivation for the end–effector orientation [66, 63].

A unit vector that defines an axis ($\boldsymbol{r} = [r_x\ r_y\ r_z]^T$) is expressed in the frame $o_i x_i y_i z_i$ with a rotation of an angle ϑ. The rotation matrix related to a given angle and axis $\boldsymbol{R}(\vartheta, \boldsymbol{r})$ is derived by Equation (2.7) [8, 63, 66].

$$\boldsymbol{R}(\vartheta, \boldsymbol{r}) = \begin{bmatrix} r_x^2(1-c_\vartheta)+c_\vartheta & r_x r_y(1-c_\vartheta)-r_z s_\vartheta & r_x r_z(1-c_\vartheta)+r_y s_\vartheta \\ r_x r_y(1-c_\vartheta)+r_z s_\vartheta & r_y^2(1-c_\vartheta)+c_\vartheta & r_y r_z(1-c_\vartheta)-r_x s_\vartheta \\ r_x r_z(1-c_\vartheta)-r_y s_\vartheta & r_y r_z(1-c_\vartheta)+r_x s_\vartheta & r_z^2(1-c_\vartheta)+c_\vartheta \end{bmatrix} \tag{2.7}$$

The inverse extraction to obtain the axis $\boldsymbol{r}$ and angle ϑ can be seen in Equations (2.8) and (2.9).

$$\vartheta = \arccos\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right) \tag{2.8}$$

$$\boldsymbol{r} = \frac{1}{2\sin\vartheta}\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix} \tag{2.9}$$

Homogeneous Transformation

It has been shown that how to denote positions and orientations of a manipulator in Section 2.1.4. To define homogeneous transformations, these are combined in this section. Homogeneous transformations reduce the composition of rigid motions to matrix multiplication. In Equation (2.10), $\boldsymbol{H} \in \mathbb{R}^{4\times4}$ is a homogeneous transformation matrix, $\boldsymbol{R}$ denotes the rotation matrix, a translation vector denoted as $\boldsymbol{d}$ and $\boldsymbol{0}$ denotes the row vector (0, 0, 0) [8, 63, 66].

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{d} \\ \boldsymbol{0} & 1 \end{bmatrix}, \quad \boldsymbol{R} \in SO(3), \quad \boldsymbol{d} \in \mathbb{R}^3 \tag{2.10}$$

Inverse of the homogeneous transformation matrix can be seen in Equation (2.11) [66, 63].

$$\boldsymbol{H}^{-1} = \begin{bmatrix} \boldsymbol{R}^T & -\boldsymbol{R}^T\boldsymbol{d} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.11}$$

In Equation (2.12) to calculate subsequent transformations, the homogeneous transformation matrices must be multiplied [66, 63].

$$\boldsymbol{H}_j^i = \boldsymbol{H}_{i+1}^i \cdots \boldsymbol{H}_j^{j-1} \tag{2.12}$$

Rotation matrix and translation vector are shown in Equations (2.13) and (2.14), respectively.

$$\boldsymbol{R}_j^i = \boldsymbol{R}_{i+1}^i \cdots \boldsymbol{R}_j^{j-1} \tag{2.13}$$

$$\boldsymbol{d}_j^i = \boldsymbol{d}_{j-1}^i + \boldsymbol{R}_{j-1}^i \boldsymbol{d}_j^{j-1} \tag{2.14}$$

2.1.5 Direct Kinematics

Before going through the direct kinematics, it is necessary to mention the meaning of the operational space (also called task space) and configuration space.

- *Task space* is the cartesian space where the operation of robot is required. It has X,Y,Z, ortho and normal axes and Roll, Pitch and Yaw (RPY) rotations about each axes. In other words, it is the space in which we live.
- *Configuration space* is the description of a particular configuration of robot or also called "posture". These postures are defined by individual and independent actuation of the joints. That means, it is those joints (revolute, prismatic, spherical, cylindrical etc.) which do not depend on any other joint. The other name of the **number** that signifies the independency which describes the posture of the robot in concrete terms is called Degrees of Freedom (DOF). Now, the configuration space is the $\boldsymbol{n^{th}}$ dimensional space where the robot is represented as a point[3]. This has enormous advantages, especially in trajectory planning problems.

In order to obtain the position and orientation of the end–effector, direct kinematics are used to derive in terms of the joint variables. Firstly, it is necessary to assign the coordinate frames to the joint. MDH convention [15] helps to assign the coordinate frames in a simplified way that is explained the section below.

Modified Denavit–Hartenberg Convention

If MDH is considered, n joints robot manipulator has $n+1$ links:

1. $j_i : i \in \{1, \cdots, n\}$ is for the joints,
2. $l_i : i \in \{0, \cdots, n\}$ is for the links.

[3] n denotes number of the Degree of Freedom.

As one can see from the Figure 2.9 that joint j_i connects link l_{i-1} to link l_i. It is considered to be fixed for the location of joint j_i with regard to link l_{i-1}. Moreover, a coordinate frame $o_i x_i y_i z_i$ is rigidly attached to link l_i. Link l_i and its attached frame $o_i x_i y_i z_i$ move, if joint j_i is actuated. A joint variable $q_i = \theta_i$ is denoted with the i^{th} joint. The frame $o_0 x_0 y_0 z_0$ attached to the robot base is called as the base frame, the inertial frame or the fixed frame.

As mentioned earlier, homogeneous transformations are significant to derive the direct kinematics. Homogeneous transformation matrix of each link is denoted as $\boldsymbol{A}_i$ and it gives the position and orientation of $o_i x_i y_i z_i$ with relative to $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$ and is shown in Equation (2.15) [66].

$$\boldsymbol{A}_i = \begin{bmatrix} \boldsymbol{R}_i^{i-1} & \boldsymbol{o}_i^{i-1} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.15}$$

$\boldsymbol{T}_j^i$ is a homogeneous transformation matrix that expresses the position and orientation of $o_j x_j y_j z_j$ according to $o_i x_i y_i z_i$. $\boldsymbol{T}_j^i$ is derived by multiplication of the transformation matrices $\boldsymbol{A}_i$, also can be seen in Equation (2.16) [8, 63, 66].

$$\boldsymbol{T}_j^i = \begin{cases} \boldsymbol{A}_{i+1}\boldsymbol{A}_{i+2}\cdots\boldsymbol{A}_{j-1}\boldsymbol{A}_j, & \text{if } i < j \\ (\boldsymbol{T}_j^i)^{-1}, & \text{if } i > j \\ \boldsymbol{I}, & \text{otherwise} \end{cases} \tag{2.16}$$

In Equation (2.17), the position and orientation of the end–effector according to the inertial frame are seen and $\boldsymbol{o}_n^0$ is a translation vector[4].

$$\boldsymbol{T}_n^0 = \boldsymbol{A}_1(q_1)\cdots\boldsymbol{A}_n(q_n) = \begin{bmatrix} \boldsymbol{R}_n^0 & \boldsymbol{o}_n^0 \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.17}$$

Commonly, six parameters are necessary to define a rigid motion; defining the rotation is related to the first three parameters and defining the translation is related to the last three parameters. On the other hand, the MDH decreases the number of parameters needed to define a homogeneous transformation. It reduces the parameters from six to four by using the common manipulator geometry that is defined in Section 2.1.2.

In the MDH, the homogeneous transformation of each link ($\boldsymbol{A}_i$) is a product of four basic transformations, is seen in Equation (2.18) [8, 63, 66]. It has four variables:

[4]If the transformation matrices affect coordinate systems rather then general objects, $\boldsymbol{o}_n^0$ is often used instead of $\boldsymbol{d}_n^0$.

- The joint angle is represented by θ_i,
- The link length is represented by a_i,
- The link offset is represented by d_i,
- The link twist of joint j_i and link l_i is represented by α_i.

Moreover, $\boldsymbol{A}_i$ is a function of θ_i, the others are constant.

$$\begin{aligned}\boldsymbol{A}_i =& \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} \\ =& \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ =& \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{2.18}$$

The following properties must be considered while assigning the coordinate frames $o_i x_i y_i z_i$ to the joint j_i and link l_i according to the MDH convention:

- The axis of rotation of joint j_{i+1} is represented via z_i,
- The axis x_i is perpendicular to the axis z_{i-1} and z_i,
- The axis x_i intersects the axis z_{i-1},
- All frames are assigned with right–hand rule.

After the coordinate frames have been setting up according to the MDH convention, the following definitions are considered to establish the parameters:

- a_i = distance along x_i from the intersection of the x_i and z_{i-1} axes to o_i,
- d_i = distance along z_{i-1} from o_{i-1} to the intersection of the x_i and z_{i-1} axes,
- α_i = the angle from z_{i-1} to z_i measured along x_i,
- θ_i = the angle from x_{i-1} to x_i measured along z_{i-1}.

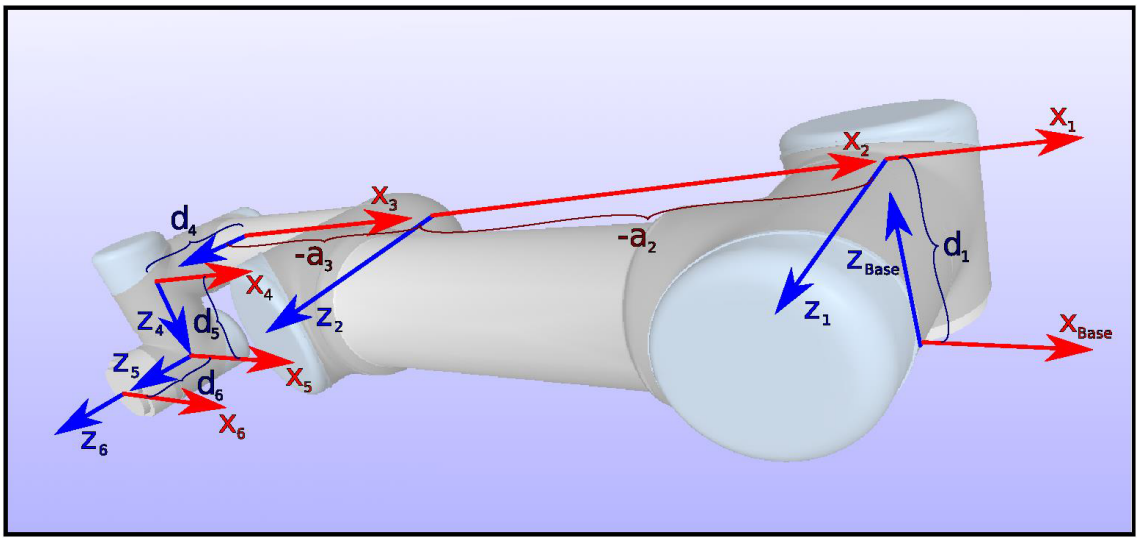


Figure 2.10 *UR5 coordinate frames with respect to the MDH convention.*

Without the MDH reduction, more than four parameters would have been required, if the link reference frames had been assigned in arbitrarily. The MDH gives us an opportunity to assign the frames by the carefully specified rules. As a consequence, the final transformation matrix describes the transformation from the fixed frame to the end–effector of the robot as a joint variable function (q_i). As all the joints in the UR5 used in this study are revolute, θ_i is a variable. The coordinate frames of UR5 that are assigned to the joints with respect to the MDH convention are seen in Figure 2.10. Firstly, the homogeneous transformation matrix $\boldsymbol{A}_i$ is calculated using the MDH parameters in Equation (2.18), and then the transformation matrix in Equation (2.17) is derived.

The following sections will focus on inverse kinematics.

2.1.6 Inverse Kinematics

Direct kinematic is about how to identify the end–effector's position and orientation whereas inverse kinematic is about finding such joint variables in terms of the desired end–effector position and orientation. Finding the joint variables for given end–effector positions is crucial for smooth trajectory planning. Inverse kinematics problem is more difficult than the direct kinematics problem [8, 63, 66]. In this section, a briefly explanation of the inverse kinematics problem is considered and then how the kinematic decoupling simplifies the inverse kinematic problem is mentioned. Using kinematic decoupling, it can be addressed the position and orientation problem separately with the help of Euler angle that is explained in Section 2.1.4.

Lastly, it is mentioned that why kinematic decoupling could not be applied to the UR5 robot and it is one of the limitations of this project.

Given a 4×4 homogeneous transformation, $\boldsymbol{H}$ is defined for the desired position and orientation of the end–effector that is shown in Equation (2.19).

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{d} \\ \boldsymbol{0} & 1 \end{bmatrix}, \quad \boldsymbol{R} \in SO(3), \quad \boldsymbol{d} \in \mathbb{R}^3 \tag{2.19}$$

The purpose of the inverse kinematics is to find the values with the help of closed form solutions for the joint variables $(q_1, \cdots, q_n)$.

$$\boldsymbol{T}_n^0 = \boldsymbol{A}_1(q_1) \cdots \boldsymbol{A}_n(q_n) \tag{2.20}$$

$$\boldsymbol{T}_n^0 = \boldsymbol{H} \tag{2.21}$$

Equations (2.20) and (2.21) result in twelve nonlinear equations with n unknown variables are shown in Equation (2.22). $\boldsymbol{T}_{ij}$, $\boldsymbol{h}_{ij}$ represent the twelve significant entries of $\boldsymbol{T}_n^0$ and $\boldsymbol{H}$, respectively [66, 63].

$$\boldsymbol{T}_{ij}(q_1, \cdots, q_n) = \boldsymbol{h}_{ij} \tag{2.22}$$

In real–time application, a closed form solution is preferred over a numerical solution for solving the inverse kinematics problem. The advantages of closed form solutions can be examined in two different perspectives. Firstly, when this project is addressed, it is about tracking a fingertip that its location is provided by Microsoft Kinect, therefore, the inverse kinematic equations of robot must be solved at a rapid rate. It is a practical requirement to have a closed form solution rather than an iterative search. Secondly, there can be more than one solution for the kinematic equations. If we have closed form solutions, it allows us to develop rules for selecting a particular solution between them [8, 63, 66].

On the other hand, the inverse kinematic problem can be solved faster by limiting the joint constraint. Depending on the application, the motion of the joints can be limited for example less than 360 degrees. Thus, it is not necessary to consider all mathematical solutions of the kinematic equations. In other respects, solutions to the kinematic equations are required to check that whether or not they satisfy all constraints on the ranges of possible joint motions. One of the advantages of the given homogeneous matrix ($\boldsymbol{H}$) is that it ensures the obtained mathematical solutions have achievable configurations features [66, 63].

Kinematic Decoupling

The aim of the kinematic decoupling is considering the manipulator joints separately. For example, the first three joints and the last three joints can be considered independently for six–DOF robots. For common six–DOF manipulator, it is probable to derive the inverse kinematics problem into two simpler problems such as *inverse position kinematics* and *inverse orientation kinematics* [8, 63, 66]. In other words, if it is desired to divide into two simpler problem, the six joints robot must have a spherical wrist structure. As far as the calculation part is concerned, it is being found the position of the intersection of the wrist axes and it is called the wrist center. Next, the orientation of the wrist is being found [66].

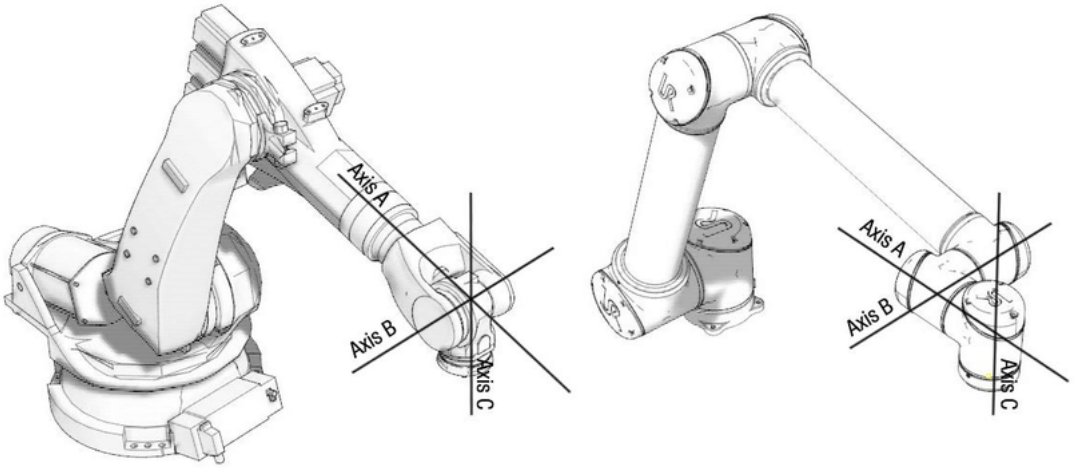


Figure 2.11 *The robot to the left of the figure has spherical wrist and right one (Universal Robot 5) has non-spherical wrist adapted from [17].*

UR5 which is used for this study has an uncommon servo specification called **non-spherical wrist** configuration. The common spherical axis configuration for robot manipulators have the three wrist axes of rotations intersect at one point, on the other hand, the non-spherical configuration have a shifted wrist axis that can be shown in Figure 2.11. The main distinction between the two configurations is about calculating the Inverse Kinematics (IK). Whereas the robots with spherical wrist have a simpler IK solution, non-spherical wrist one has complex IK solution since it is not able to divide into two simpler problems. Moreover, at any point in the most of operational space of UR5 there are nine solutions that can be achieved with nine different configurations [17], shown in Figure 2.12. It has more solutions than common manipulators, hence, solutions must be further checked that whether or not they satisfy all constraints for possible joint motions. Because of this, more

calculation time is needed for checking process. As a result, it is one of the limitations for real–time application with using UR5.

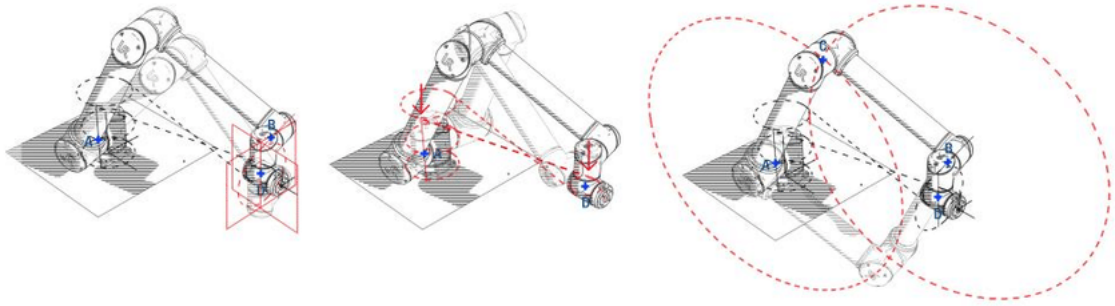


Figure 2.12 *Inverse kinematics solutions for UR5 adapted from [17].*

2.1.7 Velocity Kinematics

Velocity kinematics are related to the end–effector's linear and angular velocities with regard to the joint velocities. As mentioned in the previous section, direct kinematic is about how to identify the end–effector position and orientation. After the derivation of the function, the velocity relationships derived via Jacobian. The manipulator Jacobian ($\boldsymbol{J} \in \mathbb{R}^{6\times n}$) is either used for the derivation of the velocity kinematics, and also in many tasks in robotic manipulation (e.g., planning and execution of smooth trajectories, determination of singular configurations, derivation of the dynamic equations of motion) [8, 63, 66, 59, 30].

The transformation between the joint velocities ($\dot{\boldsymbol{q}} \in \mathbb{R}^n$) and the end–effector's angular and linear velocities is given by the manipulator Jacobian. There are two types of Jacobian such as the geometric Jacobian $\boldsymbol{J}_g \in \mathbb{R}^{6\times n}$ and analytical Jacobian $\boldsymbol{J}_a \in \mathbb{R}^{6\times n}$. The geometric Jacobian $\boldsymbol{J}_g$ and the analytical Jacobian $\boldsymbol{J}_a$, are computed separately and are used for particular objectives [8, 63, 66, 59].

The geometric technique is used for the derivation of the geometric Jacobian ($\boldsymbol{J}_g$). It maps the effect of each joint velocity to the linear and angular velocity of the end–effector. In addition, if the end–effector pose can be specified as the Euler angle, the axis/angle or roll–pitch–yaw angle representation in the task space, a direct calculation of the Jacobian by differentiating the direct kinematics function is preferred. One can see the explanation of it in the analytical Jacobian ($\boldsymbol{J}_a$) section. As far as the calculation of them is concerned, geometric Jacobian ($\boldsymbol{J}_g$) is used to derive the manipulator dynamics and to relate end–effector forces ($\boldsymbol{F} \in \mathbb{R}^6$) with

joint torques ($\boldsymbol{\tau} \in \mathbb{R}^6$) whereas the analytical Jacobian ($\boldsymbol{J}_a$) is often used to describe end–effector velocities ($\dot{\boldsymbol{X}}$) and accelerations ($\ddot{\boldsymbol{X}}$) [8, 63, 66, 59].

Manipulator Jacobian

In short, if one wants to explain the meaning of Jacobian in a simple way, it is the matrix of all first–order partial derivatives of a vector–valued function according to vector calculus. As far as the matrix is a square matrix, the matrix and its determinant are defined as the Jacobian in literature [21, 64, 76].

The transformation from the inertial frame to the end–effector frame for an n–link manipulator can be seen in Equation (2.23) [66, 63]. The joint variables vector can be shown as $\boldsymbol{q} = [q_1, \cdots, q_n]^T$.

$$\boldsymbol{T}_n^0(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{R}_n^0(\boldsymbol{q}) & \boldsymbol{o}_n^0(\boldsymbol{q}) \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.23}$$

The concept of skew symmetric matrices simplifies many of the computations involved [66]. Its definition and properties are described in Appendix A. $\boldsymbol{\omega}_n^0$ denotes the angular velocity of the end–effector, $\boldsymbol{v}_n^0$ denotes the linear velocity of the end–effector and $\boldsymbol{S}$ denotes the skew symmetric matrices, that are seen in Equations (2.24) and (2.25).

$$\boldsymbol{S}(\boldsymbol{\omega}_n^0) = \dot{\boldsymbol{R}}_n^0(\boldsymbol{R}_n^0)^T \tag{2.24}$$

$$\boldsymbol{v}_n^0 = \dot{\boldsymbol{o}}_n^0 \tag{2.25}$$

The Jacobian ($\boldsymbol{J} \in \mathbb{R}^{6\times n}$) includes both the angular and linear velocity of the end–effector that are shown in Equations (2.26) and (2.27)

$$\boldsymbol{v}_n^0 = \boldsymbol{J}_{\boldsymbol{v}}\dot{\boldsymbol{q}} \tag{2.26}$$

$$\boldsymbol{\omega}_n^0 = \boldsymbol{J}_{\boldsymbol{\omega}}\dot{\boldsymbol{q}} \tag{2.27}$$

The Jocabian consists of $\boldsymbol{J}_{\boldsymbol{\omega}} \in \mathbb{R}^{3\times n}$ and $\boldsymbol{J}_{\boldsymbol{v}} \in \mathbb{R}^{3\times n}$. $\boldsymbol{\xi}$ is vector of body velocities that is seen in Equation (2.28).

$$\boldsymbol{\xi} = \boldsymbol{J}\dot{\boldsymbol{q}} \quad \text{where} \quad \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{v}_n^0 \\ \boldsymbol{\omega}_n^0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_{\boldsymbol{v}} \\ \boldsymbol{J}_{\boldsymbol{\omega}} \end{bmatrix} \tag{2.28}$$

One of the important things to note is that the velocity vector ($\boldsymbol{\xi}$) does not mean the derivation of a position variable, because the angular velocity vector ($\boldsymbol{\omega}_n^0$) does not

mean the derivation of any particular time varying quantity [8, 63, 66, 59, 30]. The following sections will explain the calculation of geometric and analytical Jacobian.

Geometric Jacobian

The geometric Jacobian ($\boldsymbol{J}_g(\boldsymbol{q})$) is derived via the homogeneous transformation that has already been mentioned in Section 2.1.4. In order to calculate Jacobian, direct kinematics must first be calculated with regard to the MDH convention. For n–link manipulator, the calculation of angular part of the Jacobian ($\boldsymbol{J}_{g,\boldsymbol{\omega}}$) is shown in Equations (2.29) and (2.30) [66, 63, 50, 30].

$$\boldsymbol{J}_{g,\boldsymbol{\omega}} = \left[\boldsymbol{z}_0^0, \cdots, \boldsymbol{z}_{n-1}^0\right] \tag{2.29}$$

$$\boldsymbol{z}_{i-1}^0 = \boldsymbol{R}_{i-1}^0 \boldsymbol{k}, \quad \boldsymbol{k} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \tag{2.30}$$

In Equation (2.31) the calculation of linear part of the Jacobian ($\boldsymbol{J}_{g,\boldsymbol{v}}$) is shown[5].

$$\begin{aligned} \boldsymbol{J}_{g,\boldsymbol{v_i}} &= \frac{\partial \boldsymbol{o}_n^0}{\partial \boldsymbol{q}_i} \\ &= \boldsymbol{z}_{i-1}^0 \times \left(\boldsymbol{o}_n^0 - \boldsymbol{o}_{i-1}^0\right) \\ \boldsymbol{J}_{g,\boldsymbol{v}} &= \left[\boldsymbol{J}_{g,\boldsymbol{v_1}}, \cdots, \boldsymbol{J}_{g,\boldsymbol{v_n}}\right] \end{aligned} \tag{2.31}$$

Therefore, geometric Jocabian is acquired by using Equation (2.28).

Analytical Jacobian

The analytical Jacobian ($\boldsymbol{J}_a(\boldsymbol{q})$) is depend on a minimal representation for the orientation of the end–effector frame such as Euler angle, axis/angle or roll–pitch-yaw angle representation. Using minimal representations are the common way for description of the end–effector's position and orientation. It is necessary to derive it for especially smooth trajectory planning that is one of the goals of the thesis. The analytical Jacobian is used for deriving the end–effector's velocity and acceleration. The calculation of the end–effector pose is shown in Equation (2.32) [8, 63, 66, 59].

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{o}_n^0(\boldsymbol{q}) \\ \boldsymbol{\beta}_n^0(\boldsymbol{q}) \end{bmatrix} \tag{2.32}$$

[5]The meaning of the sign "×" in Equation (2.31) is cross product. For further reading, see [76].

$\boldsymbol{o}_n^0(\boldsymbol{q})$ is the fixed frame's origin relative to the end–effector frame's origin and $\boldsymbol{\beta}_n^0(\boldsymbol{q})$ is a minimal representation of the the end–effector frame's orientation with regard to the fixed frame. According to the Euler angles, one can define the orientation vector as $\beta = \phi = [\varphi, \vartheta, \psi]^T$. The analytical Jacobian is shown in Equation (2.33) [66, 63].

$$\begin{aligned} \dot{\boldsymbol{X}} &= \begin{bmatrix} \boldsymbol{v}_n^0 \\ \dot{\boldsymbol{\beta}}_n^0 \end{bmatrix} \\ &= \boldsymbol{J}_a(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned} \tag{2.33}$$

Considering the $\boldsymbol{R} = \boldsymbol{R_{ZYZ}}$ Euler angle transformation and the skew symmetric matrix in Equation (2.34) [66, 63],

$$\dot{\boldsymbol{R}}_{ZYZ} = \boldsymbol{S}(\boldsymbol{\omega})\boldsymbol{R}_{ZYZ} \tag{2.34}$$

the angular velocity ($\boldsymbol{\omega}$) is given by Equation (2.35) [66].

$$\begin{aligned} \boldsymbol{\omega} &= \begin{bmatrix} c_\psi s_\vartheta \dot{\varphi} - s_\psi \dot{\vartheta} \\ s_\psi s_\vartheta \dot{\varphi} + c_\psi \dot{\vartheta} \\ \dot{\psi} + c_\vartheta \dot{\varphi} \end{bmatrix} \\ &= \begin{bmatrix} c_\psi s_\vartheta & -s_\psi & 0 \\ s_\psi s_\vartheta & c_\psi & 0 \\ c_\vartheta & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\vartheta} \\ \dot{\psi} \end{bmatrix} = \boldsymbol{B}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}} \end{aligned} \tag{2.35}$$

Hereby, the analytical Jacobian ($\boldsymbol{J}_a(\boldsymbol{q})$) can be calculated from the geometric Jacobian ($\boldsymbol{J}_g(\boldsymbol{q})$) that is shown in Equation (2.36) under the condition of $\det \boldsymbol{B}(\boldsymbol{\phi}) \neq 0$.

$$\boldsymbol{J}_a(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}^{-1}(\boldsymbol{\phi}) \end{bmatrix} \boldsymbol{J}_g(\boldsymbol{q}) \tag{2.36}$$

The acceleration of the end–effector according to the fixed frame is shown in Equation (2.37) [66, 63].

$$\ddot{\boldsymbol{X}} = \boldsymbol{J}_a(\boldsymbol{q})\ddot{\boldsymbol{q}} + \Big(\frac{d}{dt}\boldsymbol{J}_a(\boldsymbol{q})\Big)\dot{\boldsymbol{q}} \tag{2.37}$$

The inverse velocity and acceleration of 6–DOF manipulators are obtained in Equations (2.38) and (2.39) under the condition of $\det \boldsymbol{J}_a(\boldsymbol{q}) \neq 0$ [66].

$$\dot{\boldsymbol{q}} = \big[\boldsymbol{J}_a(\boldsymbol{q})\big]^{-1}\dot{\boldsymbol{X}} \tag{2.38}$$

$$\ddot{\boldsymbol{q}} = \big[\boldsymbol{J}_a(\boldsymbol{q})\big]^{-1}\left[\ddot{\boldsymbol{X}} - \Big(\frac{d}{dt}\boldsymbol{J}_a(\boldsymbol{q})\Big)\dot{\boldsymbol{q}}\right] \tag{2.39}$$

In this section, Jacobian is computed via differentiation of the direct kinematics function relative to the joint variables. Jacobian is important for smooth trajectory planning because the speed and acceleration parameters are obtained through it. By controlling the acceleration and velocity parameters, a much smoother and safer robot trajectory can be derived.

2.1.8 Trajectory Planning

How the trajectory planning problem relates to path planning are explained in this section, after that the basic case of planning a trajectory between two configurations are mentioned.

The path planning problem can be described as finding a path from a starting configuration $\boldsymbol{q}_{\text{init}}$ to an ending configuration $\boldsymbol{q}_{\text{final}}$. A path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q}_{\text{final}}$ is a continuous map, $\tau : [0, 1] \to \mathcal{Q}$, where $\tau(0) = \boldsymbol{q}_{\text{init}}$ and $\tau(1) = \boldsymbol{q}_{\text{final}}$. A trajectory is a function of time $\boldsymbol{q}(t)$ where $\boldsymbol{q}(t_0) = \boldsymbol{q}_{\text{init}}$ and $\boldsymbol{q}(t_f) = \boldsymbol{q}_{\text{final}}$ [63, 66].

Here on, it is considered to plan the trajectory for a single joint, since the trajectories for the remaining joints can be created independently and in exactly the same way. Before moving on to the *Cubic Polynomial Trajectories* and the *Quintic Polynomial Trajectories*, it is needed to determine the variables in Equations (2.40) to (2.45) [66, 63].

At time t_0 joint variables are,

$$\boldsymbol{q}(t_0) = \boldsymbol{q}_0 \tag{2.40}$$

$$\dot{\boldsymbol{q}}(t_0) = \boldsymbol{v}_0 \tag{2.41}$$

Variables required to reach at t_f are,

$$\boldsymbol{q}(t_f) = \boldsymbol{q}_f \tag{2.42}$$

$$\dot{\boldsymbol{q}}(t_f) = \boldsymbol{v}_f \tag{2.43}$$

In addition, determinations of the constraints on initial and final accelerations are,

$$\ddot{\boldsymbol{q}}(t_0) = \boldsymbol{\alpha}_0 \tag{2.44}$$

$$\ddot{\boldsymbol{q}}(t_f) = \boldsymbol{\alpha}_f \tag{2.45}$$

Cubic Polynomial Trajectories

In order to to generate smooth trajectory between two configurations, it is necessary to specify the start and end velocities of it. One can derive a smooth curve via a polynomial function of t. Since we have four constraints $(\boldsymbol{q}_0, \boldsymbol{v}_0, \boldsymbol{q}_f, \boldsymbol{v}_f)$ to satisfy in Equations (2.40) to (2.43), it is required four independent coefficients that can be chosen to satisfy these constraints. Thus, in Equation (2.46) it is considered as a cubic trajectory of the form [8, 63, 66, 59].

$$\boldsymbol{q}(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{2.46}$$

Equation (2.47) shows the desired velocity.

$$\dot{\boldsymbol{q}}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{2.47}$$

Merging Equations (2.46) and (2.47) with the four constraints leads to four equations with four unknowns that are shown in following Equations (2.48) to (2.51).

$$\boldsymbol{q}_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \tag{2.48}$$

$$\boldsymbol{v}_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 \tag{2.49}$$

$$\boldsymbol{q}_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \tag{2.50}$$

$$\boldsymbol{v}_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 \tag{2.51}$$

Next, one can write these four equations as a single matrix equation.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_0 \\ \boldsymbol{v}_0 \\ \boldsymbol{q}_f \\ \boldsymbol{v}_f \end{bmatrix} \tag{2.52}$$

$(t_f - t_0)^4$ is the determinant of the coefficient matrix in Equation (2.52) and if a nonzero time interval is considered, it always has a exclusive solution. This solution is allowed for executing the trajectory. A sequence of moves can be planned using Equation (2.52) by setting the end conditions $\boldsymbol{q}_f$, $\boldsymbol{v}_f$ of the $(i)^{th}$ move as initial conditions for the $(i+1)^{th}$ move.

Quintic Polynomial Trajectories

Using multiple cubic trajectories for planning trajectories gives us an opportunity to create continuous positions and velocities at the blend times, whereas discontinuities in the acceleration. For overcoming this discontinuities, the trajectories can be created as a quintic polynomial trajectory. In addition to this, *Jerk* is the rate of change of acceleration; that is, the derivative of acceleration with respect to time. A discontinuity in acceleration leads to an impulsive *Jerk*, which is one of the reasons of vibrational modes in the manipulator and reduce tracking accuracy and safety. These problems reduces the safety and efficiency in HRI. One can specify constraints on the acceleration as well as on the position and velocity with quintic polynomial. Apart from the cubic polynomial trajectories, quintic has six constraints (e.g., initial and final of the configurations,velocities and accelerations). Therefore, it is required a fifth order polynomial which is shown in Equation (2.53) [8, 63, 66, 59].

$$\boldsymbol{q}(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{2.53}$$

The six constraints yield six equations in six unknowns that are shown in following Equations (2.54) to (2.59).

$$\boldsymbol{q}_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 \tag{2.54}$$

$$\boldsymbol{v}_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 \tag{2.55}$$

$$\boldsymbol{\alpha}_0 = 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 \tag{2.56}$$

$$\boldsymbol{q}_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \tag{2.57}$$

$$\boldsymbol{v}_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \tag{2.58}$$

$$\boldsymbol{\alpha}_f = 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 \tag{2.59}$$

Next, one can combine these six equations into a single matrix equation.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_0 \\ \boldsymbol{v}_0 \\ \boldsymbol{\alpha}_0 \\ \boldsymbol{q}_f \\ \boldsymbol{v}_f \\ \boldsymbol{\alpha}_f \end{bmatrix} \tag{2.60}$$

In the same way as in the cubic polynomial, the determinant of the coefficient matrix in Equation (2.60) is not equal to zero in a nonzero time interval. This means that

it always has a unique solution for the execution of the trajectory.

The following section will focus on HRI.

2.2 Human Robot Interaction

In our new era, modern industries are expanding day by day and running with continuous production cycles. In addition to the expansion, robots have begun to appear in almost every task. An inevitable consequence of the frequent use of robots is that there will be HRI in future. On the other hand, in near future intelligent machines and robots are awaited to bring business–level services by collaborating independently in teams with humans as a crucial part of a multi–company business process [58, 73].

Collaborative robots need to meet different requirements from traditional robots used in the industry. Although there is a standard for traditional robots that is published in [26], there is only technical specification for collaborative robots can be seen in [27]. If a HRI study is to be applied, this technical specification must be considered. Performing a complicated task with human interaction and coordination is defined as a collaboration [12]. It can be specified that there are two different types of collaboration, one is physical collaboration and another one is contactless collaboration. Physical collaboration means that there is an explicit and intentional contact between human and robot and the robot can predict human motion intention and react accordingly by measuring or estimating the exchange of force [28, 37] whereas contactless collaboration means there is no physical interaction. It is being done with gestures and/or voice commands [55] as well as indirect communication, by recognizing intentions [45] or attention [36].

Apart from these definitions, metrics of HRI must be considered, concordantly, taxonomy of HRI metrics were presented in [43] can be seen in Figure 1.1. When we consider these metrics, If collaboration with the robot happens properly and instantly, efficiency in interaction will also be high. On the other hand, if the robot's trajectory is not planned, the robot may go undesirable position and it may also make vibrational movements. This leads to risk to human. As a result, the trajectory must be planned in advance.

Practices and challenges in HRI will be examined in the following sections.

2.2.1 Practices in HRI

Although there are plenty of acknowledged practices that are appearing in HRI, it is possible to separate them under five categories. First of all, including professionals from multiple disciplines on research efforts is a key practice. Robotics, cognitive science, human–computer interaction, electrical and mechanical engineering, computer science can be given examples of these disciplines [12, 14]. Creating real systems (robot autonomy, interaction modes, and etc.) and then evaluate these systems using experiments with human subjects are the second emerging practice. It is more useful to elaborate the psychological principles that are underlying expressive interaction phenomena eye toward harnessing. Therefore, modeling, evaluation and engineering can be given as key aspects of HRI [12, 14].

Running experiments that also include a detailed results from physical and simulated robot is the third significant practice. It is often strenuous to carry out cautiously controlled experiments with physical robots due to cost and reliability issues [12, 14]. Moreover, it is not always possible to expect same results with simulation robots because the physical environment has own unique challenges and details that are not existed in numerous simulations. Establishing standards and accepted metrics can be counted as a fourth emerging area. One of the most detailed surveys of metrics can be seen [67], other metrics exists in the literature and one can also see in the proceedings of the annual Performance Metrics for Intelligent Systems (PERMIS) workshops. The fields of Urban Search and Rescue (USAR) domain [44, 72], space applications and Unmanned Aerial Vehicles (UAVs) [70, 71] have the strongest standardization efforts.

Long term studies are the fifth emerging practice. It has been made such studies possible with the availability of reliable service robots and personal home robots in public areas [25]. Long–term studies shift research methodologies from small–scale experiments that are carefully controlled and questioned to other methodologies such as ethnography.

2.2.2 Challenges in HRI

In this section, it is specified a collection of problems that are likely to shape HRI in the near future. For each problem, it is discussed those aspects of the problem that make it particularly challenging and useful. One of the high profile challenge problem in HRI is USAR. The highly unstructured nature of USAR environments makes it a challenge problem. This forces strict challenges on map–building, situation

awareness, robot mobility and communications. Another high profile challenge area in HRI is developing military and combat robots. Its environments tend to be unstructured, on the other hand, most importantly operators are required to operate under extreme stress.

Space robotics is another area and it has unstructured environment as well as it is often challenging due to the presence of dust, the vacuum of space, temperature and radiation. Another noteworthy characteristics of space robotics is that operators can be highly trained for observation. Assistive robotics is a challenge area, the proximity and vulnerability of the human in the interaction are the challenges rather than the unstructured environment. Humanoid robotics is a challenging field both in terms of engineering of human movements and expressions and the difficulties that arise when a robot receives human form. With such a form, the social and emotional aspects of interaction become more important than anything else.

2.3 Related Work

Due to the lack of highly skilled robotics programmers, programming should be as simple as telling a colleague to perform a specific task. For this reason, future robot instruction schemes require human communication channels, multimodal interfaces and the use of intuition together with these developments, tasks can be done by using speech and gestures along with human and accompanied by safe robots manipulation in the same work area without any fences. Identification and localization of workpieces are also required for autonomous robots in order to minimize the effort of automatic production or adaptation to programming, program parameters and process parameters. In a possible robotic collaboration with human, the use of sensors and 3D cameras is indispensable. After the Kinect was launched in 2010, a great deal of research has been done. Significant number of researches investigate gesture recognition and hand tracking. However, there is a lack of research about the use of 3D cameras together with collaborative robots on HRI directly.

First of all, methods will be explained briefly; see [68] for more complete review. As one can see in [51], it proposes a method for tracking fingertips and palm center using depth data from Kinect. A big circle filter is applied for detecting palm center and following that fingertip detection is achieved. On the other hand, in [54] a different method is presented called Finger–Earth Mover's Distance (FEMD). It uses both color and depth data provided by Kinect to enhance robustness and efficiency. In addition to these, [35] uses K–means clustering for hand detection, eight point neighborhood for finger identification to recognize the gestures. A novel

technique proposed by [32] which based on action graphs and its steps are segmentation of Kinect depth data, tracking, filtering, normalization and feature extraction to achieve the recognized gestures.

Apart from these methods, there are considerable amount of works using both Kinect and robots. Real–time hand guiding of UR5 worked in [41]. Kinect data were used to generate hand position and a smartphone was used for hand orientation. Both data are being sent to UR5 via a client. Robot navigation with the capabilities of the ROS and Kinect that relies on the fuzzy logic approach was proposed in [60]. Moreover, visual guidance systems are examined in [31], [62], [52]. Collision avoidance can be shown as a reflection of industry application in [40] and [74]. Both of them are using predictive methods to estimate collision–free UR5 robot trajectory.

In this particular HRI study, it is shown an efficient HRI with the planned trajectory based on nearest–point approach by using UR5, Microsoft Kinect and the power of ROS and its tools.

2.4 Chapter Summary

In this chapter, theoretical background of robot is explained by giving the fundamental robot kinematics that are necessary to calculate trajectory. In addition to this, the trajectory planning methods used during the implementation of the thesis such as cubic and quintic polynomial trajectories are specified. Moreover, HRI, its challenges and practices are emphasized. Finally, one can see the related works about the thesis, too.

3. SYSTEM OVERVIEW

In this chapter, development environments and equipments are explained and UR5 specifications with regard to MDH, Microsoft Kinect and motion planners are discussed. System overview can be seen in Figure 3.1.

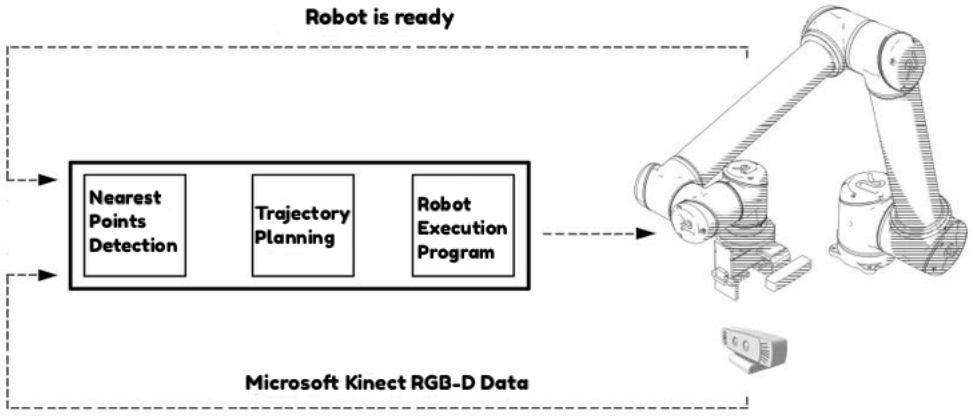


Figure 3.1 *An overview of the operation of the system.*

3.1 Development Environments and Equipments

One can see that there are plenty of tools and environments for robotics as well as computer vision systems. In this thesis, ROS, PCL, MoveIt! motion planner, the Open Motion Planning Library (OMPL) and libraries for connecting and communicating between computer vision equipment and ROS are used for implementation (e.g., *libfreenect2, IAI Kinect2*). On the other hand, Microsoft Kinect v2 and Universal Robot 5 are used as equipments for real–time experiment. In this particular research case one can see libraries and tools in the following sections.

3.1.1 Robot Operating System (ROS)

ROS is a set of software libraries and tools that help someone build robot applications. From drivers to state–of–the–art algorithms, ROS has capabilities for different

tasks with the power of peer–to–peer communication. ROS is an open source package [1, 49]. In this project, Kinetic Kame version of ROS was used with Ubuntu 16.04 Long Term Support (LTS) installed computer.

3.1.2 Point Cloud Library (PCL)

A point cloud is a data structure, which is used to represent a collection of multi–dimensional points. It is usually used to represent 3D data. In a 3D point cloud, the points ordinarily represent the X, Y, and Z geometric coordinates. When color information is present, the point cloud becomes 4D [56]. Point clouds can be acquired from hardware sensors such as stereo cameras, 3D scanners, or TOF cameras [2]. PCL also supports the 3D interfaces, therefore, it can acquire and process data from devices such as the PrimeSensor 3D cameras, the Microsoft Kinect, or the Asus XTionPRO. The environment of the PCL as well as the state before and after the filter is applied can be seen in Figure 3.2.

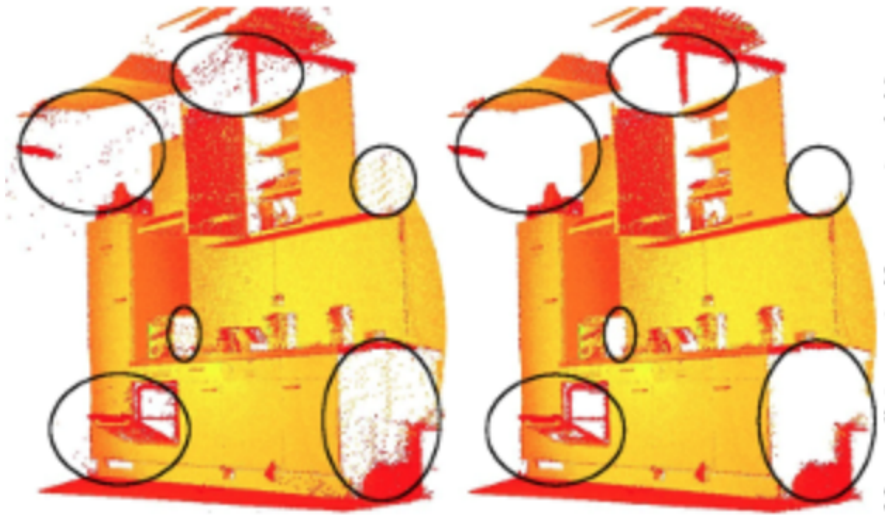


Figure 3.2 *Point cloud environment, left one is the state before applying a filter and it has outliers, right one is after applying the filter, adapted from [56].*

3.1.3 Gazebo Simulation Tool (GST) and ROS visualization (rviz)

GST is a well-designed simulator that makes it possible to test algorithms rapidly, design robots, perform regression testing, and train Artificial Intelligence (AI) system using realistic scenarios [46]. It also offers to simulate populations of robots in indoor and outdoor environments accurately and efficiently. It has a robust engine,

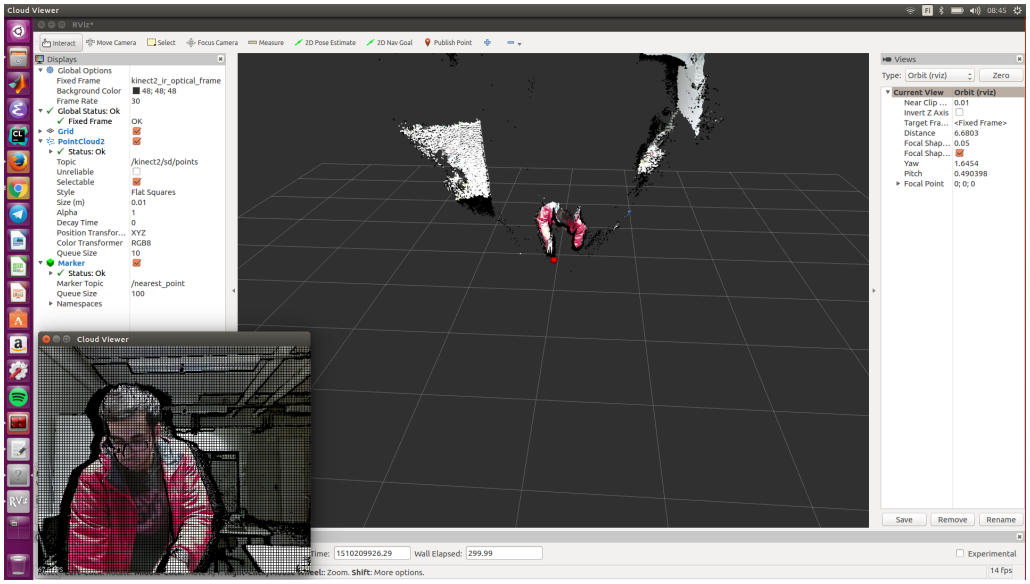


Figure 3.3 *Visualization of the nearest point in the PCL using rviz.*

high-quality graphics and easy to use graphical interfaces. On the other hand, **rviz** is a 3D visualizer for displaying sensor data and state information from ROS. Using rviz, one can visualize the current configuration on a virtual model of the robot. It is also possible to display live representations of sensor values coming over ROS topics including camera data, infrared distance measurements, sonar data, and more. It is visualized that whether or not the position difference is being sent to the simulation environment. Test and visualization environment can be seen in Figure 3.3. For

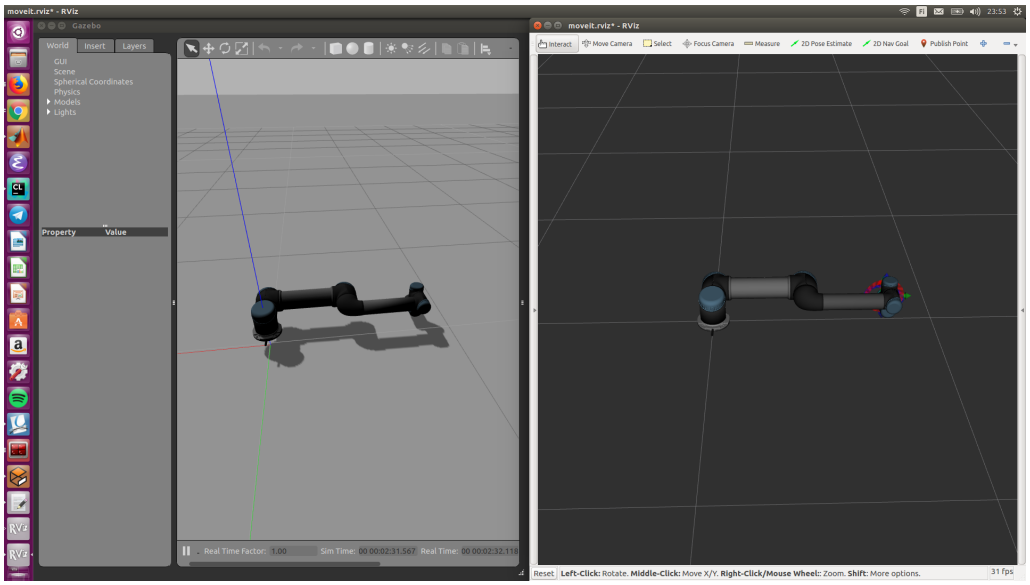


Figure 3.4 *Start state of Gazebo Simulation Tool (GST) and ROS visualization (rviz).*

initial test, only Z–coordinate difference is created after that it is being sent to the GST via ROS topics. Initial and final state of the robot can be seen in Figure 3.4

and Figure 3.5, respectively.

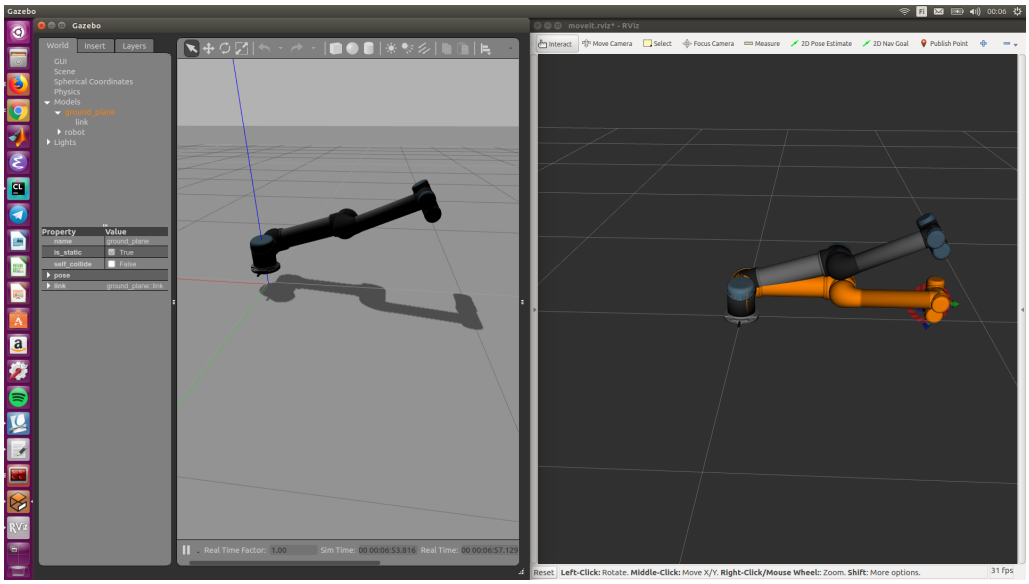


Figure 3.5 *Final state of Gazebo Simulation Tool (GST) and ROS visualization (rviz).*

3.1.4 Universal Robot–5 (UR5)

It is important that the robotic system should be reliable and safe. There is a ISO specification that is named ISO/TS 15066 [27] and it is supplement to ISO 10218 [26] "Safety Requirements for Industrial Robots" standards. ISO/TS 15066 describes the different collaborative concepts and details the requirements to achieve. It also presents a research study on pain thresholds, robot speed, pressure and impact for specific body parts. The Universal Robots has eight adjustable safety functions such as joint positions and speeds, Tool Center Point (TCP) positions, orientation, speed and force, momentum and power of the robot [38]. For these reasons, the six–DOF UR5 from Universal Robots has been chosen to use in this study. The sketch of the coordinate frames according to the MDH convention is shown in Figure 3.6. Kinematic equations of UR5 with respect to the MDH can be seen in Appendix B. The robot is lightweight (18kg), its reach is $850mm$ and its payload is $5kg$.

3.1.5 Microsoft Kinect v2

Kinect is an RGB–D sensor that provides synchronized color and depth images. With its wide availability and lower cost than other traditional 3D cameras such as stereo cameras and TOF cameras [20], many researchers in computer science and robotics are leveraging the sensing technology to develop new ways in terms

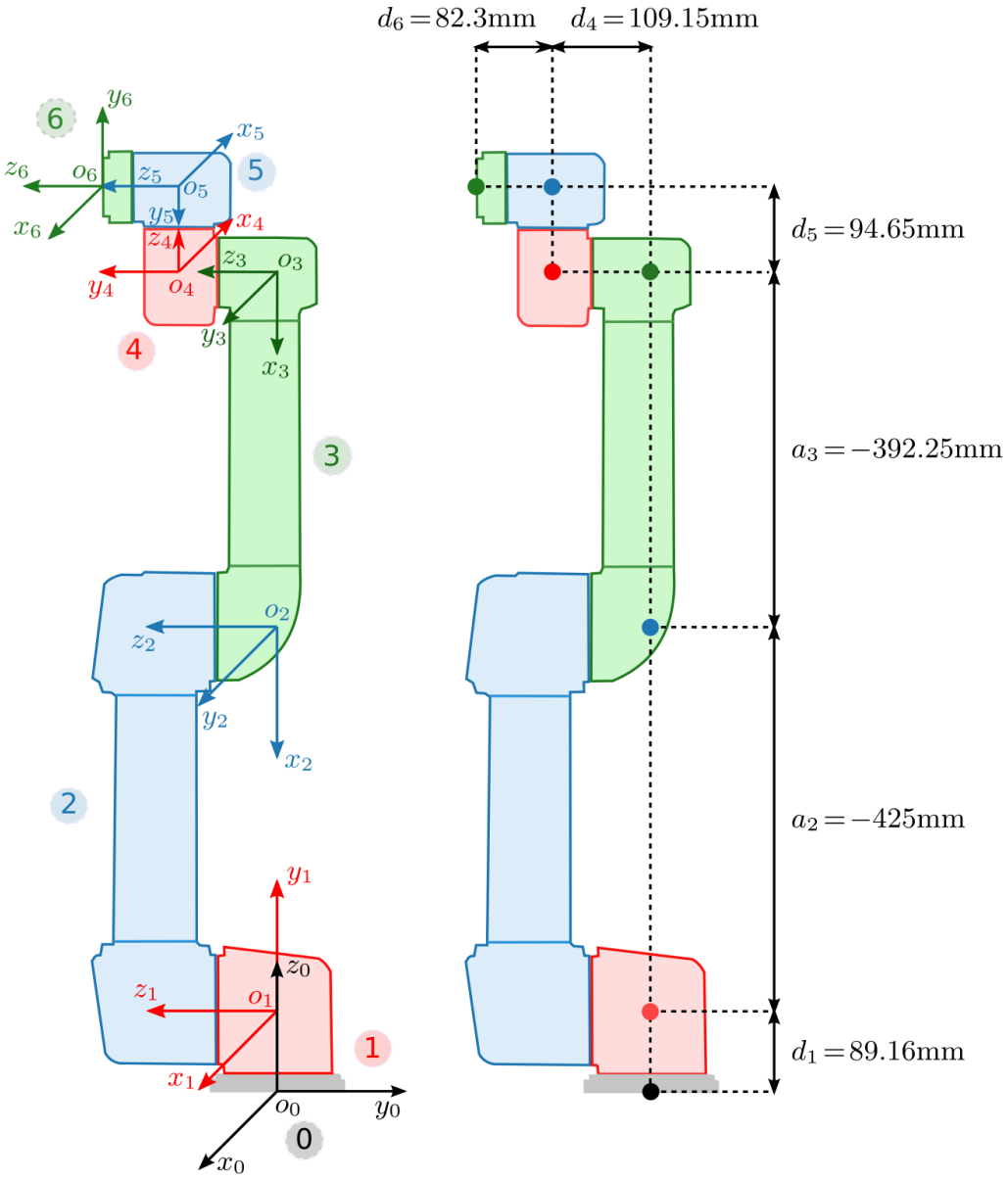


Figure 3.6 *Left of the figure shows sketch of the UR5 manipulator's coordinate frames according to the DH convention and right shows the MDH parameters adapted from [30].*

of interaction with machines [24, 79]. Kinect v2 has an RGB camera, infrared (IR) camera, IR emitter and multi-array microphone. In addition, right–handed coordinate system is also shown in Figure 3.7.

3.1.6 libfreenect2 and IAI Kinect2

libfreenect2 is an open source driver for Kinect v2 devices created by the OpenKinect community. Its features are color image processing, IR image and depth image processing, registration of color and depth images [33, 78]. *IAI Kinect2* is a collection of tools and libraries for a ROS Interface to the Kinect v2 [75]. For accurate detection

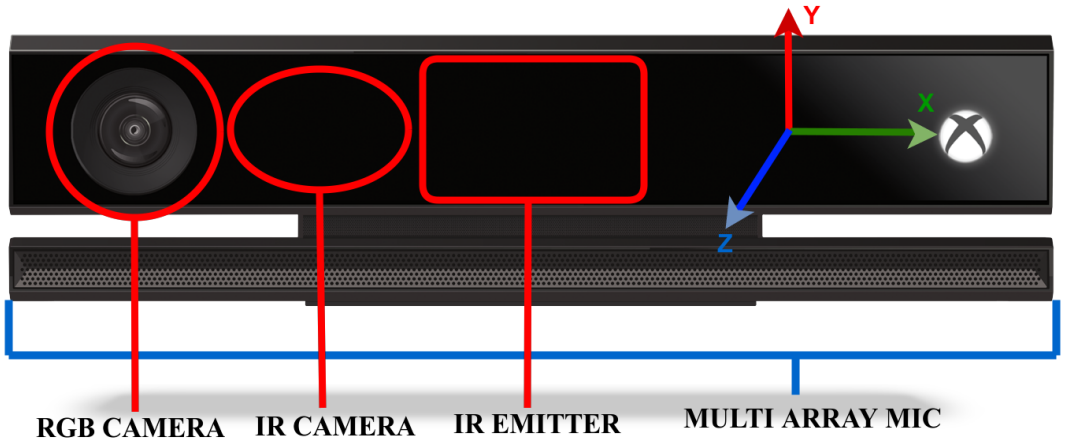


Figure 3.7 *Microsoft Kinect v2 right–handed coordinate system and its components.*

of the points, it is better to calibrate the Kinect. Figure 3.8 shows an example setup for calibration[1].

Figure 3.8 *An example setup for Microsoft Kinect v2 calibration [75].*

3.2 Motion Planners

Robots are increasingly in co–operation with humans and industrial robotic applications are starting to examine the possibility of robots and humans as coworkers,

[1]For further details about calibration and drivers can be found at https://github.com/code-iai/iai_kinect2 and https://github.com/OpenKinect/libfreenect2.

sharing tasks and workspace. As a consequence of these, the planning of the trajectory in the task of the robot is significant for safety. As a use case scenario, when one would like to motion a robot from one point to another, the robot can do this with multiple solutions. However, some of these solutions are not feasible due to singularities and obstacles. The singularity and obstacle will lead to a security weakness without the planned trajectory. If the trajectory is computed and planned, the security weakness will not occur and it will be a safer interaction.

In this study, it is expected that the movement of the detected nearest points will also be observed on the robot, so this real–time changes must be planned on the robot side. For this, the motion planners are used. These motion planners take the position, velocity and acceleration variables as inputs, depending on their solution methods, and give the feasible trajectory as output. OMPL and MoveIt! motion planners have been used in this project. OMPL consists of state–of–the–art sampling–based motion planning algorithms running under MoveIt!. The OMPL itself does not contain any code related to collision checking or visualization. MoveIt! consists collision checking, visualization, obstacle avoidance features based on ROS. Another reason for its use is that it supports UR5. The following sections will explain you what is needed to use its structure and planner. In short, it is necessary to create a configuration file before using the planner. This configuration file is a file that contains the relevant robot parameters and it is used by MoveIt! during the calculation and planning the feasible trajectory.

3.2.1 Moveit! Motion Planner

MoveIt! Motion Planner is a set of software packages integrated with the ROS and designed specifically to provide such capabilities, especially for the manipulators. MoveIt! will allow robots to build up a representation of their environment using data fused from 3D and other sensors, generate motion plans that effectively and safely move the robot around in the environment, and execute the motion plans while constantly monitoring the environment for changes. It is state–of–the–art software for mobile manipulation and it provides the latest advances in motion planning, manipulation, 3D perception, robot kinematics and control. There are several planners such as Stochastic Trajectory Optimization for Motion Planning (STOMP), Search–Based Planning Library (SBPL) and Covariant Hamiltonian Optimization for Motion Planning (CHOMP). More details can be found in [47]. It also provides an easy–to–use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains [7]. Due to its strength and simplicity, the MoveIt!

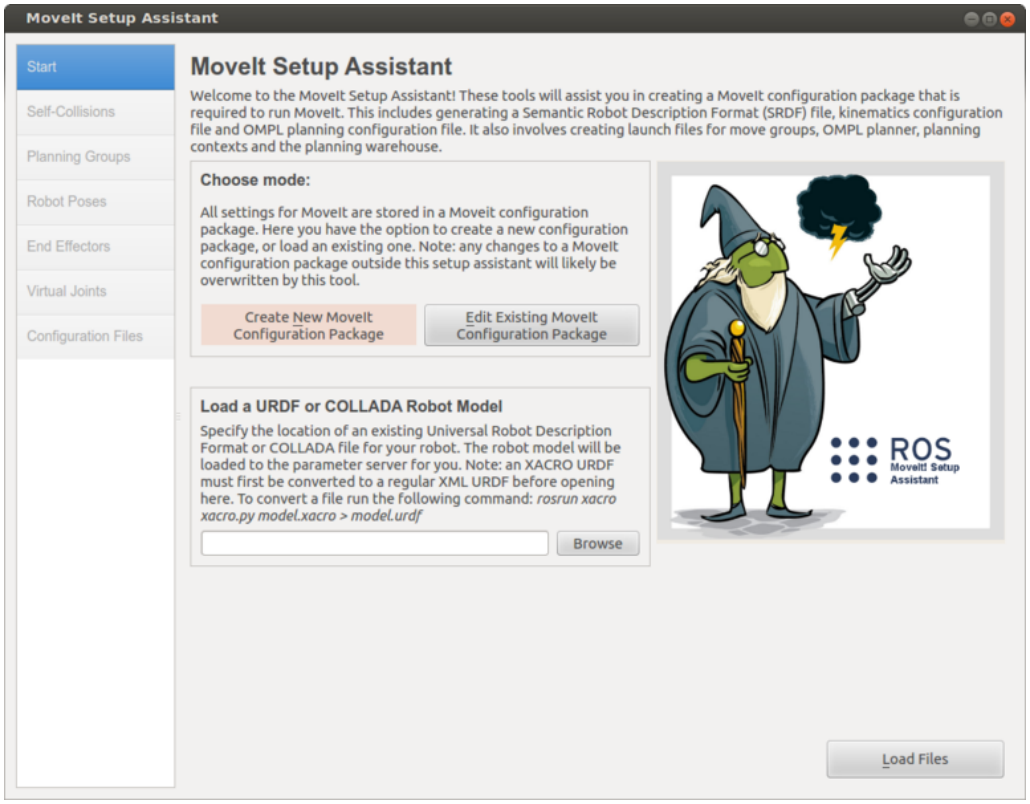


Figure 3.9 *MoveIt! Setup Assistant [47].*

motion planner was used in this project. On the other hand, there is a user interface for creating configuration files for robots that can be seen in Figure 3.9. For UR5, it is not necessary for using the interface, there is a ready–made driver[2] published in [3].

ROS users can easily use MoveIt! motion planner. It contains ROS messages and topics, therefore, one can subscribe a topic for a specific purpose (eg, saving the data of current joints values). For advanced users, without being dependent on a large part of ROS, they can make changes and developments at the core level. New features can be built in MoveIt! using the ROS message generation infrastructure. In this direction, MoveIt! provides numerous flexibilities. General structure of the Moveit! can be seen in Figure 3.10.

In MoveIt!, there are configuration files to extract all the needed information for calculating trajectories, detecting collision etc. The Unified Robot Description Format (URDF) is existing configuration file and newly defined is Semantic Robot Description Format (SRDF). These files are an advantage for getting rid of being dependent on specific robots. Robot joints, links information and their relationship

[2]The details of driver can be found at https://github.com/ThomasTimm/ur_modern_driver.

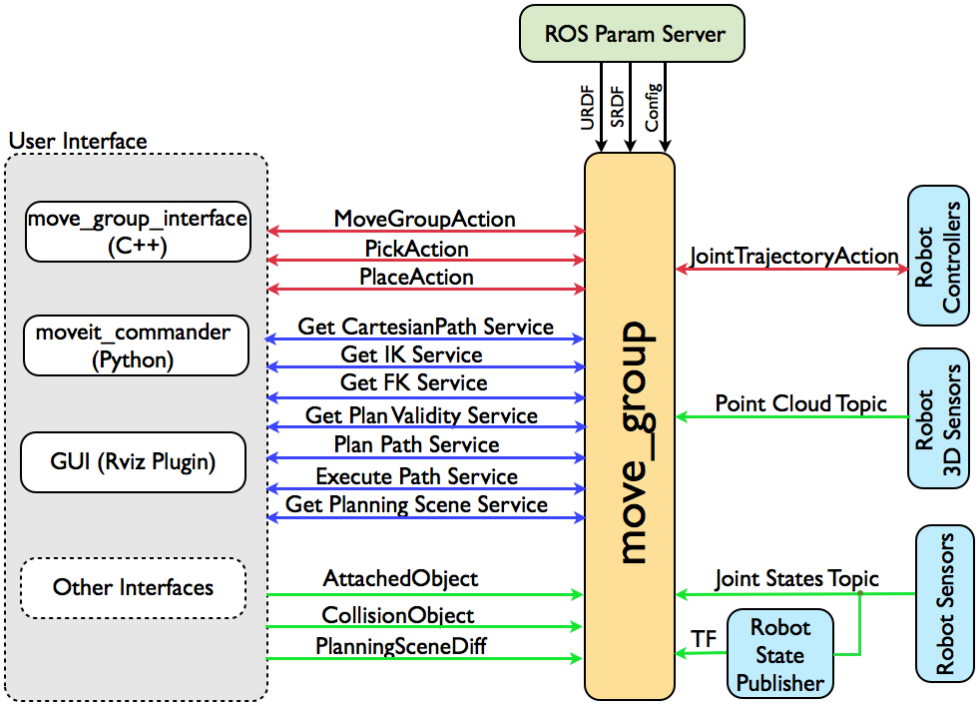


Figure 3.10 *Moveit! structure [47].*

to each other are stored via these files. By specifying these information, any robot can be controlled via MoveIt!.

```
1 double moveit::planning_interface::MoveGroup::computeCartesianPath
2 (
3 const std::vector<geometry_msgs::Pose>&
4 waypoints, // Robot paths
5 double  eef_step, // set to 0.01 (1cm steps)
6 double  jump_threshold, //parameter "j"
7 moveit_msgs::RobotTrajectory&  trajectory, //Trajectory request
8 bool  avoid_collisions = false // There is no need for collision check
    for this project. It is set to "false" for faster calculation.
9 )
```

Program 3.1 *MoveIt! Cartesian path planning example code*

The parameter "*j*" shown in Program 3.1 is "jump threshold factor parameter". It is determined in order to limit the joint change momentarily while the robot is being motioned. If it is set to zero, there is no threshold and it leads to undesired and unsafe motion while executing the trajectories. The value of zero can be used in simulation environment not for real task. Planning request adapters that are seen in Figure 3.11 include the complete motion planning pipeline chains together a motion planner with other components such as pre–processing motion plan requests and

post–processing motion plan responses. Pre–processing is useful in several situations (e.g., when a start state for the robot is slightly outside the specified joint limits for the robot.); post–processing is needed for several other operations (e.g., to convert paths generated for a robot into time–parameterized trajectories.). MoveIt! provides a set of default motion planning adapters that each perform a very specific function.

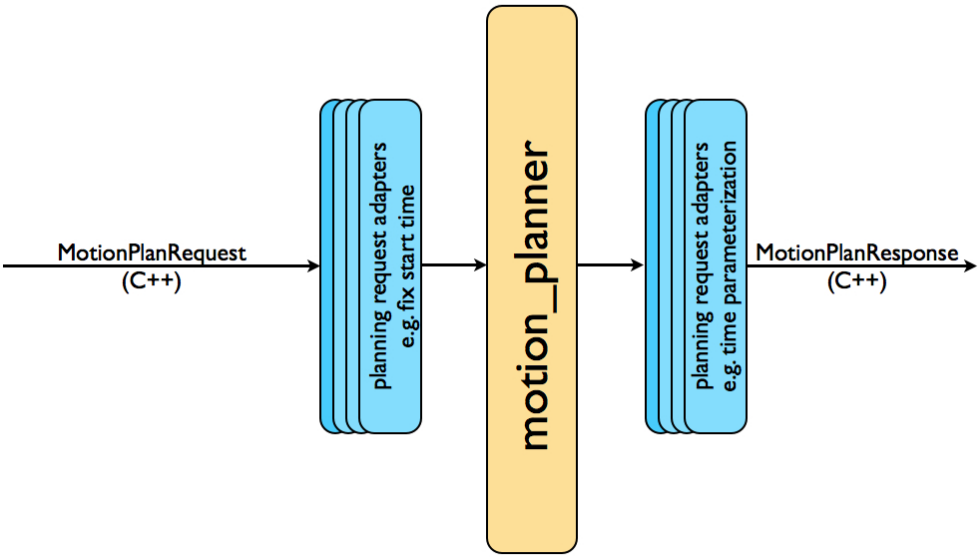


Figure 3.11 *Motion planner request [47].*

`move_group` is a ROS node. It uses the ROS parameter server to get three kinds of information:

1. URDF: `move_group` looks for the `robot_description` parameter on the ROS parameter server to get the URDF for the robot.
2. SRDF: `move_group` looks for the `robot_description_semantic` parameter on the ROS parameter server to get the SRDF for the robot. The SRDF is typically created once by a user using the MoveIt! Setup Assistant.
3. MoveIt! configuration: `move_group` will look on the ROS parameter server for other configuration specific to MoveIt! including joint limits, kinematics, motion planning, perception and other information. Config files for these components are automatically generated by the MoveIt! setup assistant and

stored in the config directory of the corresponding MoveIt! config package for the robot.

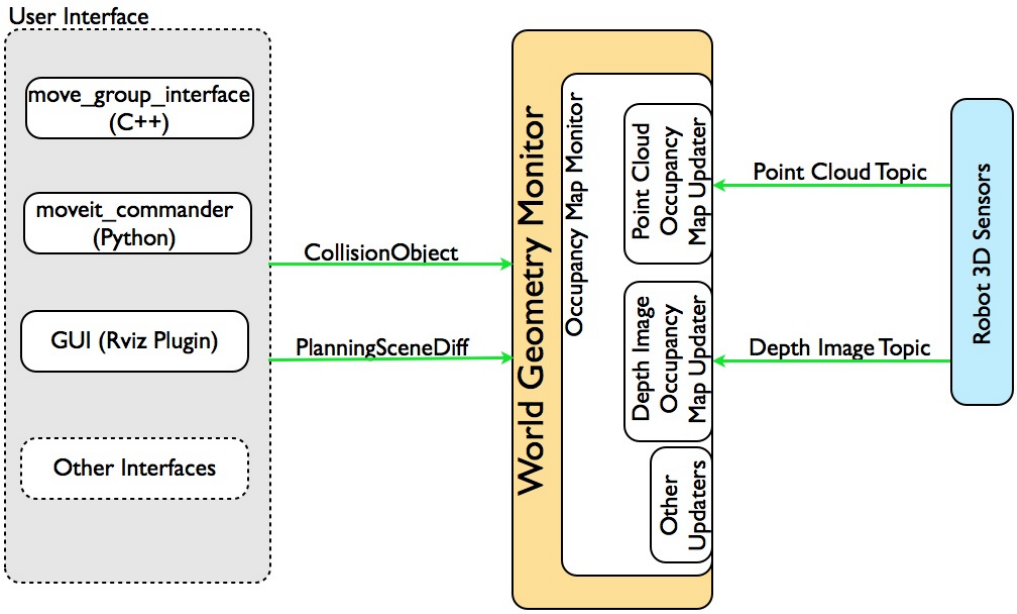


Figure 3.12 *World Geometry Monitor [47].*

An interactive graphical user interface allows users to specify motion–plan requests for their new robots, with a minimal amount of interface implementation required from the user on the robot side. This allows non–motion planning experts to easily configure the motion planning and associated components in ROS for their own robots. World geometry monitor of the Moveit! is shown in Figure 3.12. The world geometry monitor builds world geometry using information from the sensors on the robot and from user input. It uses the occupancy map monitor to build a 3D representation of the environment around the robot. By using the data on the `planning_scene` topic, object information can be added according it.

3.2.2 The Open Motion Planning Library (OMPL)

OMPL is an open–source motion planning library that primarily implements motion planners. MoveIt! integrates directly with OMPL and uses the motion planners from that library as its primary or default set of planners. The planners in OMPL are abstract; i.e. OMPL has no concept of a robot. Instead, MoveIt! configures OMPL and provides the back–end for OMPL to work with problems in robotics. The structure of OMPL can be seen in Figure 3.13.

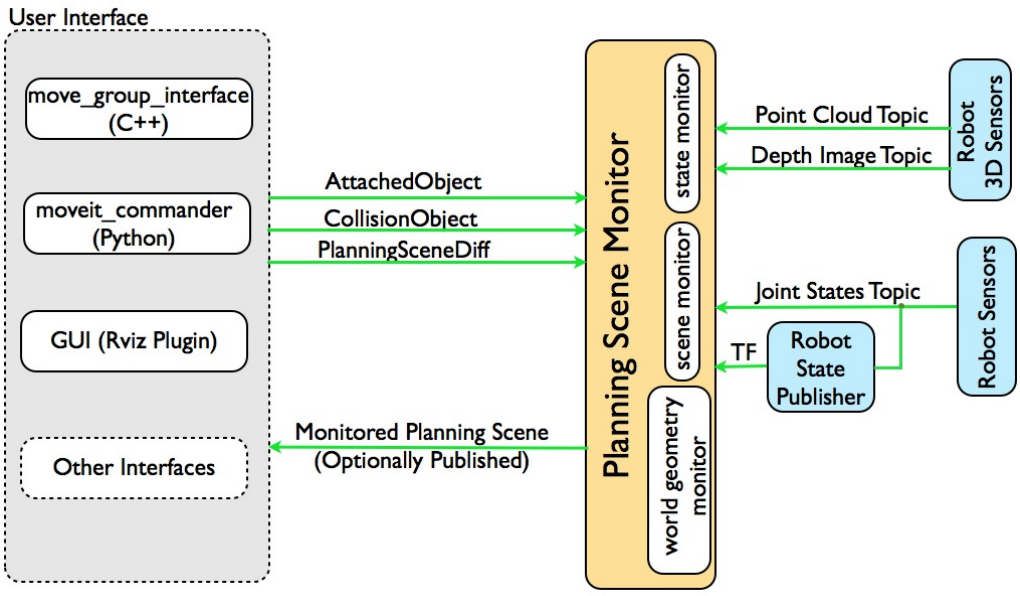


Figure 3.13 *The Open Motion Planning Library (OMPL) structure [47].*

3.3 Chapter Summary

In this chapter, detailed description of system such as ROS, PCL, GST, rviz and motion planners such as MoveIt! and OMPL are examined.

4. IMPLEMENTATION

In this chapter, a data acquisition method, filters, their features and their implementation on C++ are explained.

4.1 Execution and Filtering Process

A point cloud without any downsampling can be seen in Figure 4.1 whereas after the

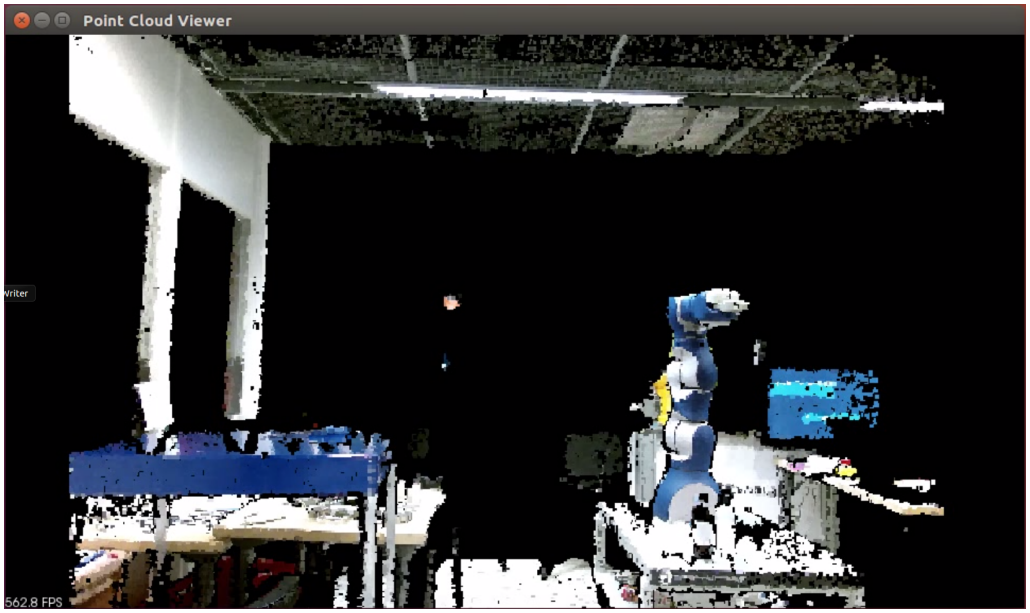


Figure 4.1 *Point cloud environment before downsampling.*

downsampling is shown in Figure 4.2. As one can see that after the downsampling, the resolution of the point cloud environment is reduced. Although reducing the number of points means gain in terms of speed, it reduces accuracy in finding the nearest point.

4.1.1 Mapping

Cartesian coordinates need to be mapped as shown in Figure 4.3. According to the setting of our system, no change is made on the X–axis. The Kinect's Y–axis is

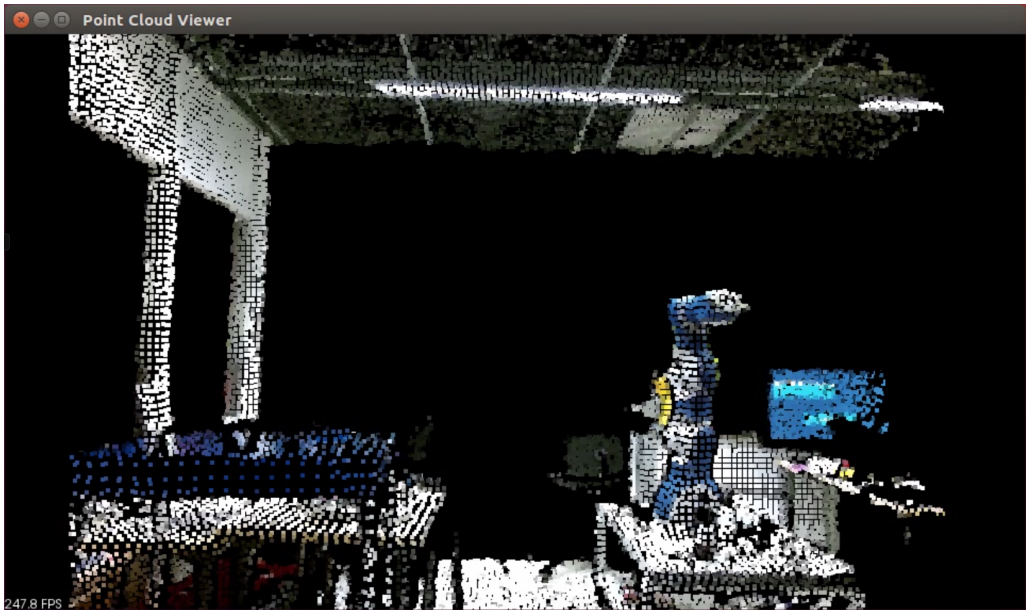


Figure 4.2 *Point cloud environment after downsampling.*

mapped to the Z–axis of UR5 and finally the Z–axis of Kinect is mapped inversely to the Y–axis of UR5.

4.1.2 Pre–Processing of Point Cloud

The point cloud is processed using VG, Radius Outlier Removal (ROR) and Statistical Outlier Removal (SOR) filters. Performance of filters based on processing time can be shown in Figure 4.4. According to the results, although VG has best execution time, it is not enough to use only it due to the outliers. ROR or SOR filters have to be used to estimate the nearest point accurately. After the implementation, it is figured out that SOR filter worked better because it is using statistical information from points. Our system has got 200 thousand points approximately. Figure 4.4 also gives information about the latency. It is observed from visualization tool that nearest point is calculated incorrectly. Therefore, pre–processing was done to recognize the fingertip accurately.

4.1.3 Data Acquisition

Data are acquired from Kinect using PCL, *libfreenect2* and *IAI Kinect2* libraries. Data are the point representation of the environment in which it is located. For testing the system, raw depth data are saved as a rosbag extension using ROS. It means that it is possible to play data via the rosplay function. It allows us to send

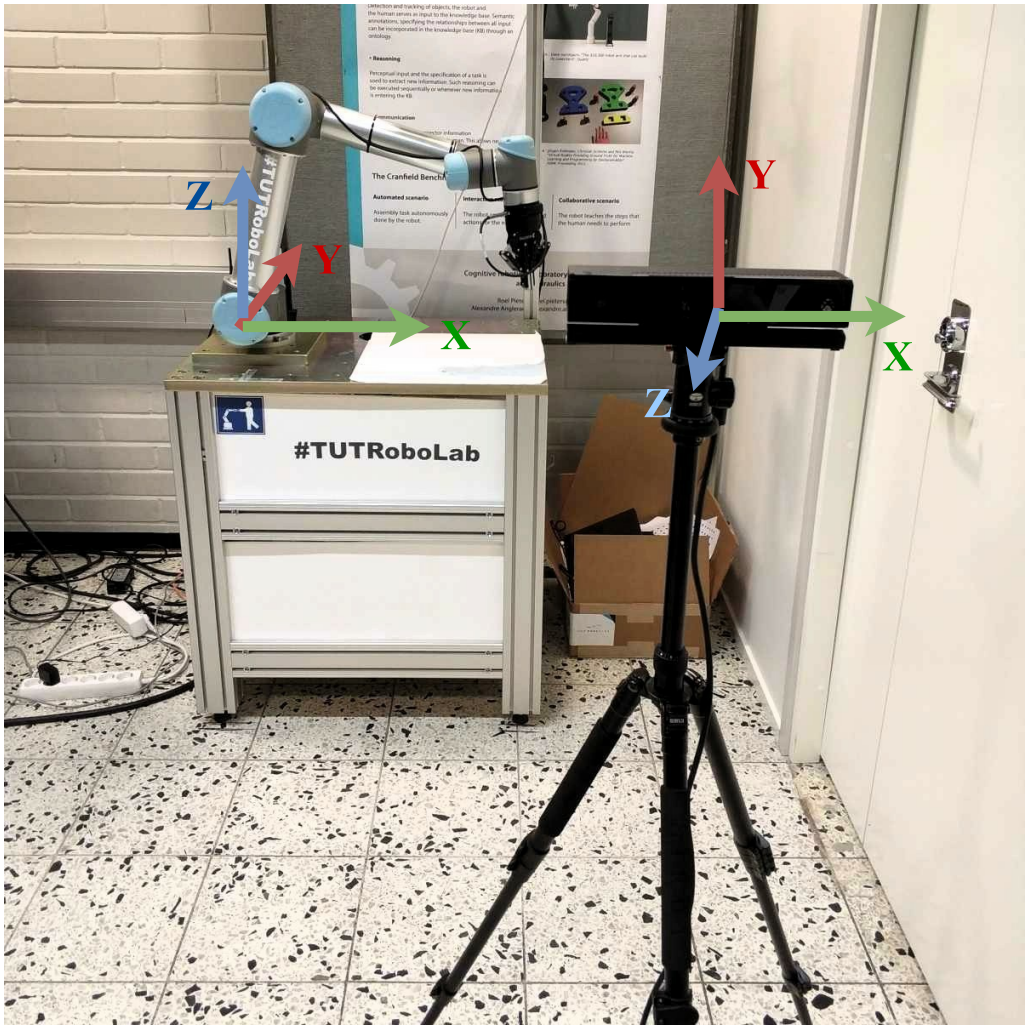


Figure 4.3 *Environment of system and Cartesian coordinate system mapping between UR5 and Kinect.*

the saved 3D depth data to the system like the data came from Kinect. On the other hand, this point representation enables efficient algorithms and filters to be applied to our system. Output of data and results can be seen in Chapter 5.

4.1.4 Euclidean Distance

The Euclidean distance or Euclidean metric is the "ordinary" straight–line distance between two points in Cartesian space. With this distance, Euclidean space becomes a metric space. The associated norm is called the Euclidean norm. A generalized term for the Euclidean norm is the L^2 norm or L^2 distance. In general, for an n–dimensional space, the distance can be generalized in Equation (4.1). p and q represent two points in Cartesian space.

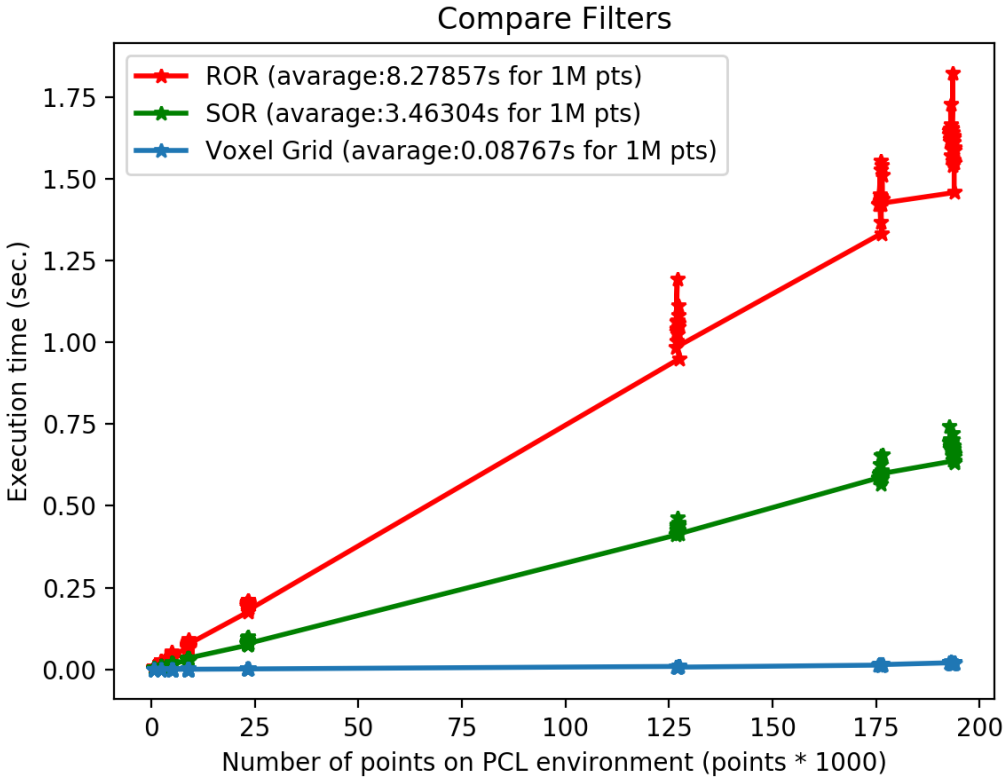


Figure 4.4 *Compare of three of filters such as Voxel Grid, Radius Outlier Removal and Statistical Outlier Removal based on execution time in second. A test program was run and it was measured by setting max_depth and min_depth parameters of Kinect in PCL.*

$$d(p,q) = \sqrt{(p_1 - q_2)^2 + (p_1 - q_2)^2 + \ldots + (p_i - q_i)^2 + \ldots + (p_n - q_n)^2} \qquad (4.1)$$

Euclidean distance was used to calculate the difference between nearest points. Points has X,Y and Z dimensions. In order to calculate the Euclidean distance, Eigen tool was used for matrix calculation [23]. Moreover, this distance is used for filtering process in PCL, too.

4.1.5 k–Nearest Neighbors (k–NN)

In pattern recognition, the k–Nearest Neighbors (k–NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the "k" closest training examples in the feature space. The output depends on whether k–NN is used for classification or regression [10]. It is used in the SOR filter, "k" can be specified by using *setMeanK* function. For the implementation of filters in PCL, k–NN is used to determine how many neighbor points will participate for calculation during the removal of noise and outlier points.

4.1.6 Point Cloud Environment without Filter

Before the explanation of the filters, in Figures 4.5 to 4.7 green square represents the true nearest point and red circle represents the calculated nearest point. One

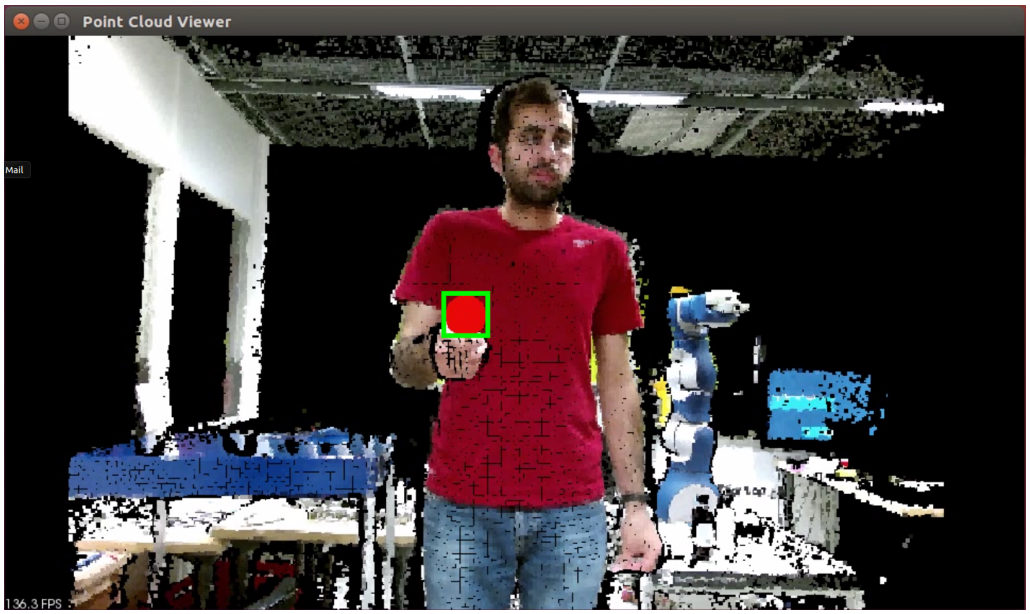


Figure 4.5 *The first frame without any filters applied. At the first instantaneous moment, it can be seen that the nearest point is calculated correctly. But it does not mean that the filter is worked properly.*

can see the initial state of the point cloud in Figure 4.5. The situation without any hand movements in the PCL can be seen in Figure 4.6. Finally, one more wrong calculation of nearest point is shown in Figure 4.7. The nearest point is miscalculated because of the noise in the point cloud.

4.1.7 Voxel Grid (VG) Filter

A voxel grid is a set of tiny 3D boxes in space. VG filter aims to downsample the point cloud. In each voxel (3D box), all the points downsampled with their centroid [56]. Nearest point calculation at the starting time after applying VG filter is shown in Figure 4.8. Nearest point calculation at the final time after applying VG filter is shown in Figure 4.9. One can see that when the fingertip is moved, the nearest point calculation is done incorrectly. This means that the VG filter is not enough and additional filtering is necessary. C++ implementation of it can be seen in Algorithm 1.

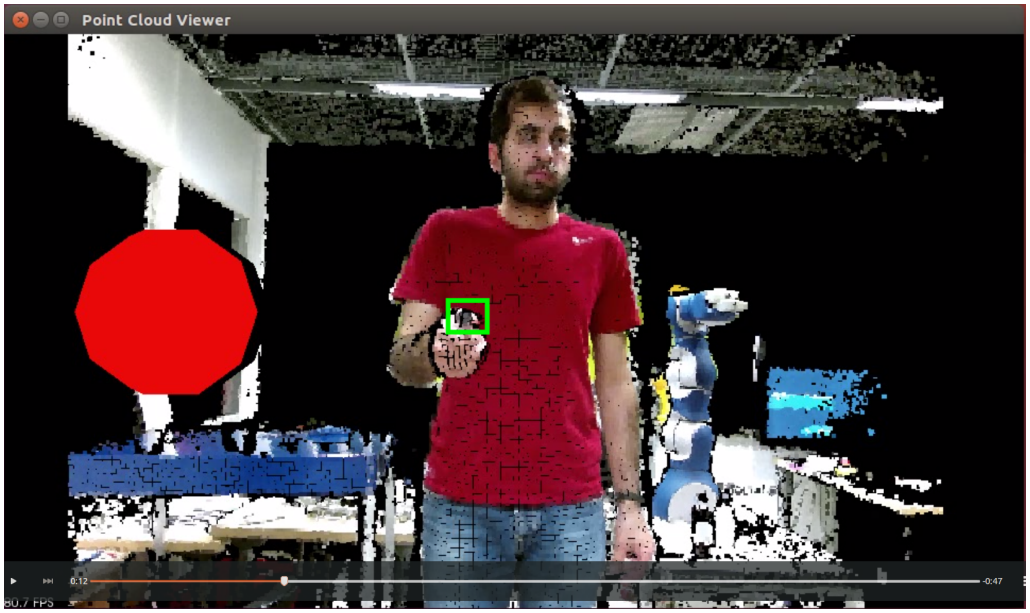


Figure 4.6 *The second frame without any filters applied. Although the fingertip is not moved, it can be seen that the nearest point is miscalculated.*

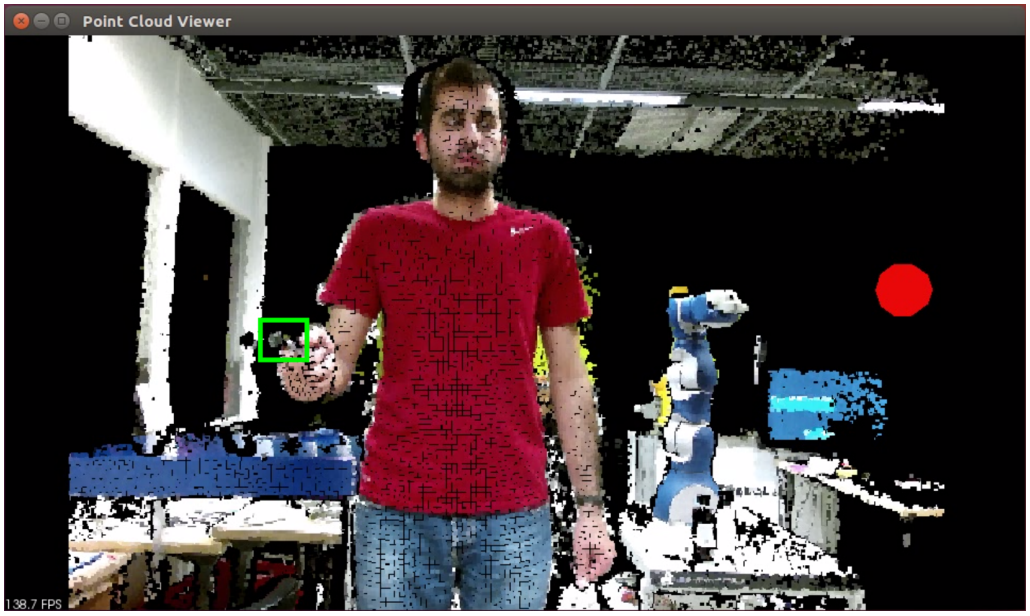


Figure 4.7 *The third frame without any filters applied. When the fingertip is moved at the first time, the location of the nearest point changes. But it is still miscalculated.*

4.1.8 Radius Outlier Removal (ROR) Filter

ROR filter removes all indices in its input cloud that do not have at least some number of neighbors within a certain range [56]. The user specifies a number of neighbors which every indice must have within a specified radius to remain in the PCL. This kind of approach can be useful because when the data are acquired as points representation of the environment, there are outlier points on the corners and

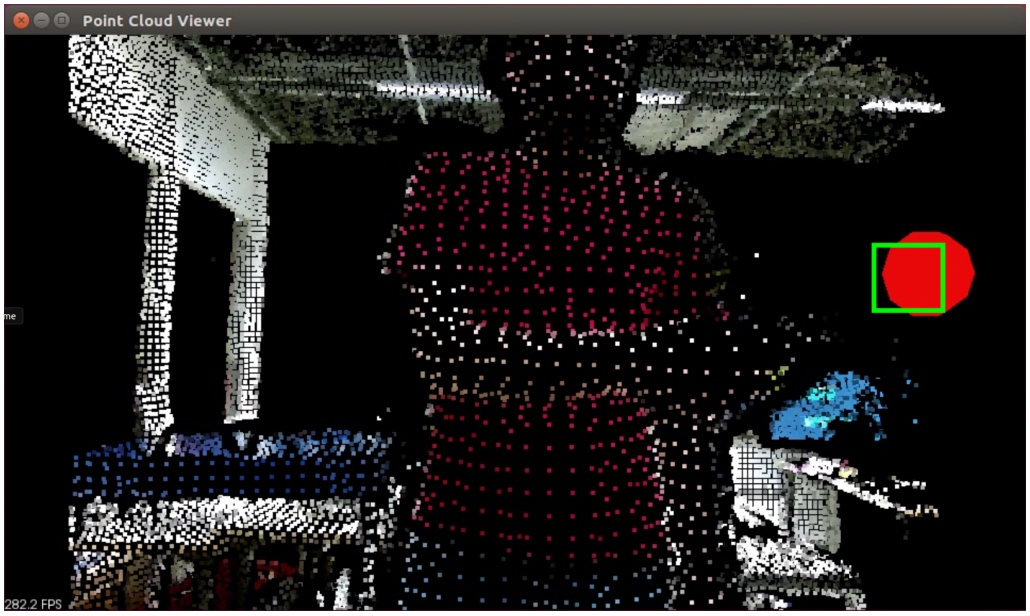


Figure 4.8 *Nearest point calculation at the starting time after applying VG filter. At the first instantaneous moment, it can be seen that the nearest point is calculated almost correctly with VG filter.*

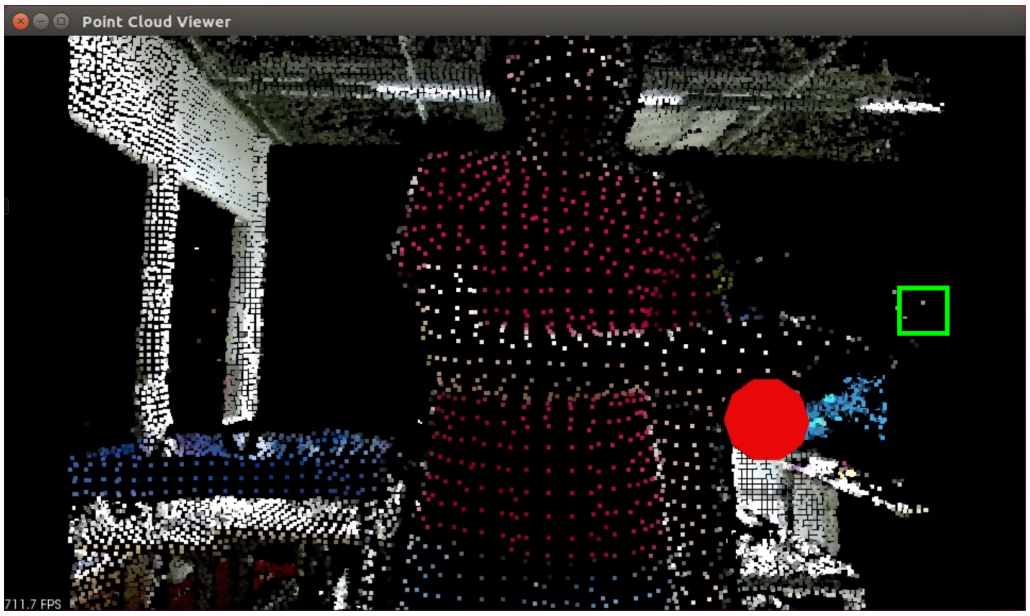


Figure 4.9 *Nearest point calculation at the final time after applying VG filter. When the fingertip is moved at the first time, the location of the nearest point changes. But the error is lower than the situation of no filters is applied.*

in the space of environment. The nearest point is calculated incorrectly due to these points. ROR filter is tested in this study because it allows us to set a specific radius and also to set how many points will be in the radius. Figure 4.10 helps to visualize what the ROR filter object does. For example if 1 neighbor is specified, only the yellow point will be removed from the PCL. If 2 neighbors are specified then both

Algorithm 1 Voxel Grid Filter

```
1: pcl::VoxelGrid <pcl::PointXYZRGB>vg;
2: vg.setInputCloud (input_cloud);
3: vg.setLeafSize(0.01f, 0.01f, 0.01f);
4: vg.filter (filtered_cloud);
```

the yellow and green points will be removed from the PCL. C++ implementation can be seen in Algorithm 2.

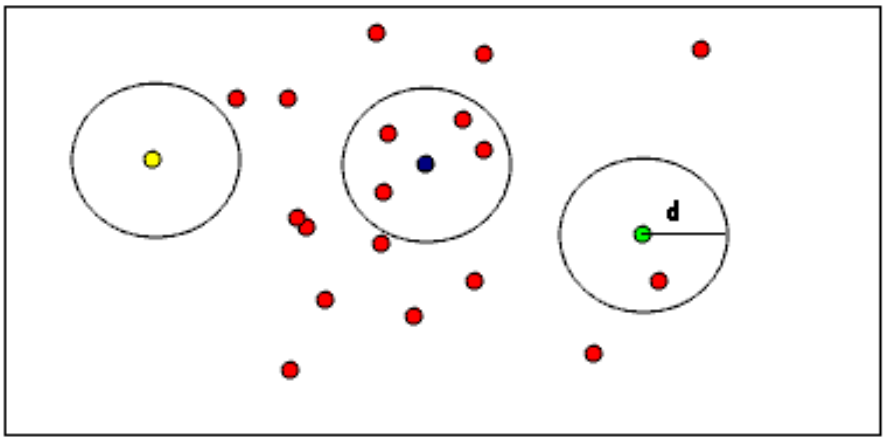


Figure 4.10 *Radius Outlier Removal (ROR) filter.*

Algorithm 2 Radius Outlier Removal Filter

```
1: pcl::RadiusOutlierRemoval <pcl::PointXYZRGB>ror;
2: ror.setInputCloud (input_cloud);
3: ror.setRadiusSearch (0.5);
4: ror.setMinNeighborsInRadius (80);
5: ror.filter (filtered_cloud);
```

4.1.9 Statistical Outlier Removal (SOR) Filter

SOR filter is based on the computation of the distribution of distances in points in comparison with its neighbors in the input cloud. For each point, the mean distance from all its neighbors is being computed. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset [56]. Similar to ROR filter, this kind of approach can be useful because when the data are acquired as points representation of the environment, there are outlier points on

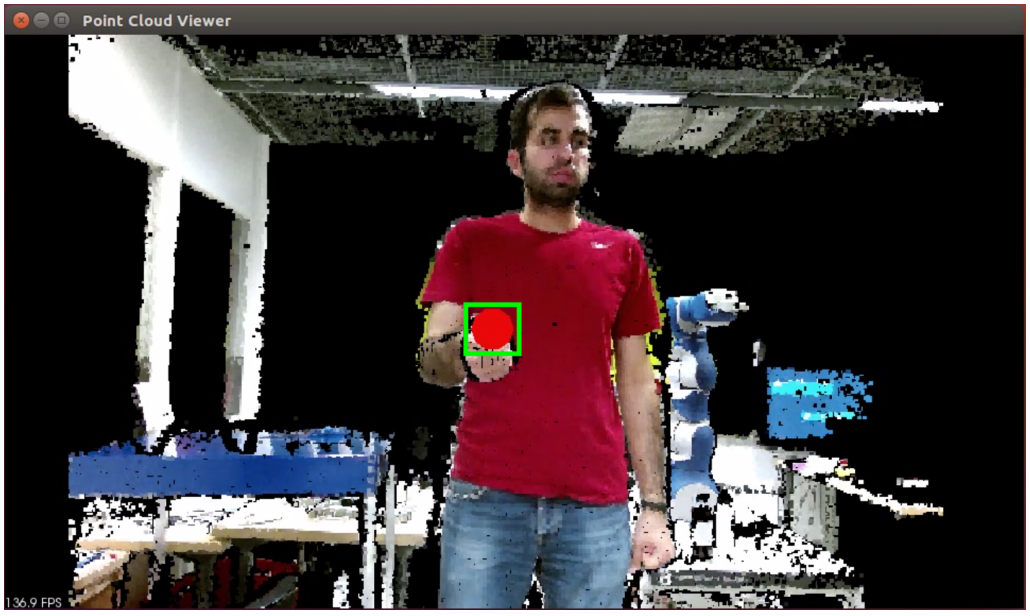


Figure 4.11 *The initial state of point cloud environment after applying SOR and VG filter.*

the corners and in the space of environment. During the outlier and noise removal process, SOR filter is approaching to the points statistically and it removes them by using the correlation with each other. Assuming that these points are distributed in Gauss, it does not remove the important points for calculation. In order not to increase the calculation time, VG downsampling filter was applied before applying this filter. Figure 4.11 shows the initial frame of point cloud environment after applying SOR and VG filter together and Figure 4.12 shows the final frame of point cloud environment after applying SOR and VG filter together. SOR uses point neighborhood statistics to filter outlier data. The algorithm iterates through the entire input twice. During the first iteration it will compute the average distance that each point has to its nearest k neighbors. The value of k can be set using `setMeanK()`. Next, the mean and standard deviation of all these distances are computed in order to determine a distance threshold. The distance threshold will be equal to: `mean + stddev_mult * stddev`. The multiplier for the standard deviation can be

Algorithm 3 Statistical Outlier Removal

```
1: pcl::StatisticalOutlierRemoval <pcl::PointXYZRGB>sor;
2: sor.setInputCloud (input_cloud);
3: sor.setMeanK (25);
4: sor.setStddevMulThresh (1.0);
5: sor.filter (filtered_cloud);
```

set using `setStddevMulThresh()`. During the next iteration the points will be classified as inlier or outlier if their average neighbor distance is below or above this

threshold, respectively. The neighbors found for each query point will be found

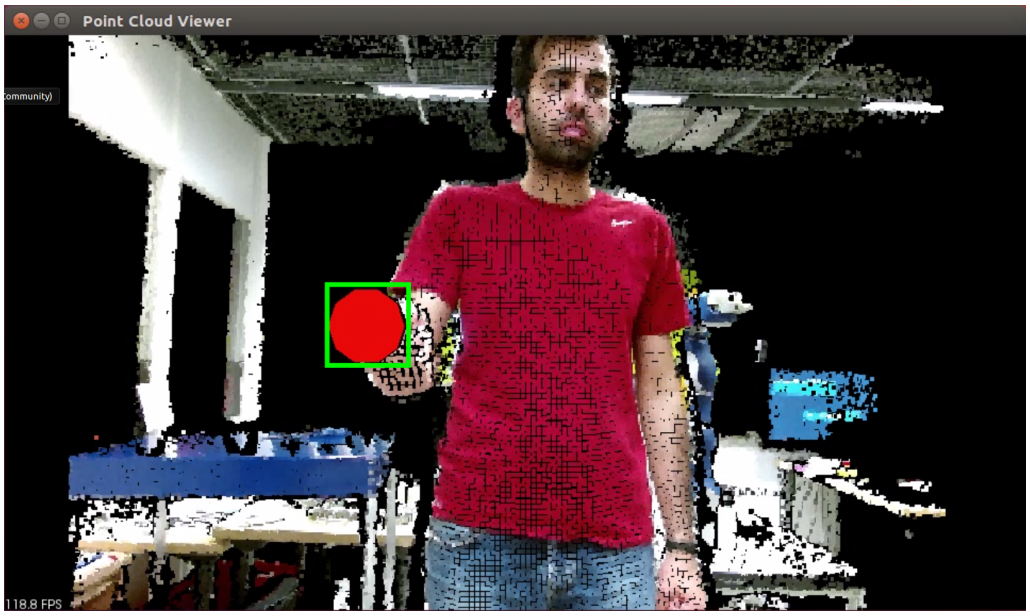


Figure 4.12 *The final state of point cloud environment after applying SOR and VG filter.*

amongst all points of `setInputCloud()`, not just those indexed by `setIndices()`. The `setIndices()` method only indexes the points that will be iterated through as search query points [57]. Algorithm 3 represents C++ implementation of the filter.

4.1.10 Performance Measures

Mean Absolute Percentage Error (MAPE) is used to measure the performance of the methods [13, 4]. MAPE equation is

$$\text{MAPE} = \frac{100\%}{N} \sum_{n=1}^{N} \left\{ \left| \frac{A_n - F_n}{A_n} \right| \right\} \tag{4.2}$$

where N is the number of the sample points with regard to trajectories, A_n is the current value of the reference trajectory and F_n is the current value of the test trajectory. It is a measure of accuracy of a method for constructing fitted time series values in statistics. This measure is easy to understand because it provides the error in terms of percentages.

4.2 Chapter Summary

In this chapter, the coordinate systems mapping between UR5 and Kinect are given. Following that, the comparison of filters such as SOR, ROR and VG with regard to

execution time as well as in terms of the effect on point cloud are explained. Lastly, the performance measurement criterion is mentioned.

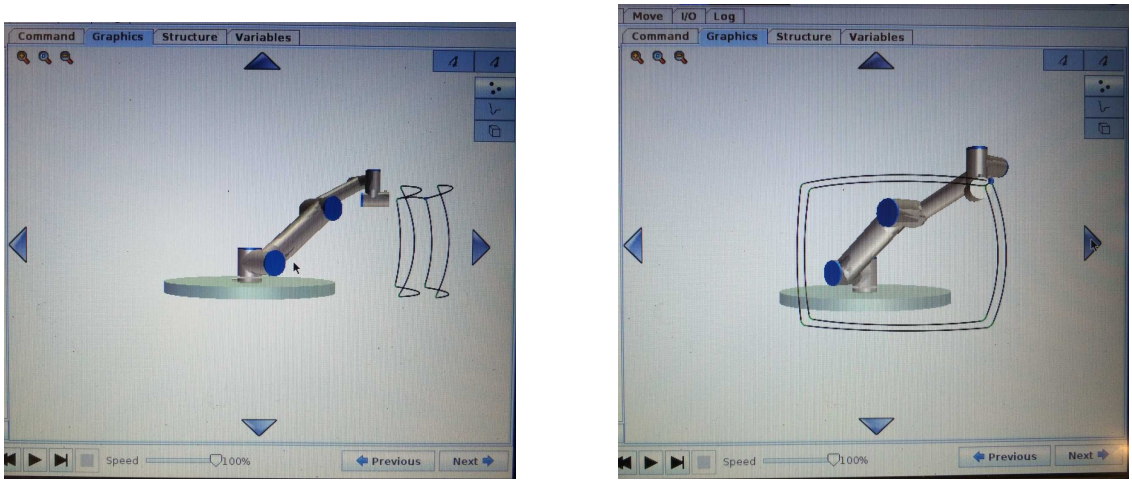
5. RESULTS AND DISCUSSION

In this chapter, the raw RGB–D data are sent to the system for testing the trajectory and results are obtained. The accuracy of trajectories can be seen in the following sections.

5.1 Results

In order to test the system under difficult conditions, a test trajectory is created with the help of teach pendant of UR5 that is shown in Figure 5.1. Kinect turned to the robot and the test trajectory is run. Following that the RGB–D data are saved as "rosbag" extension using `rosbag record` function during the first cycle of the trajectory and the robot is brought to the starting position. This time, the system is run with the recorded data using `rosbag play` function without using Kinect and the motion of the robot is observed. The main idea of the test is that how the algorithm follows the fingertip accurately. First test, "Cartesian path planning method" with VG, ROR and SOR filters is used and X with 85.79%, Y with 87.14% and Z with 79.63% are obtained. One can see the results in Table 5.1. Second test, "Time parameterization method" with VG and SOR filters is used and X with 97.32%, Y with 93.34% and Z with 91.77% are obtained. One can see the results in Table 5.2.

Secondly, since the jump threshold factor (**j**) mentioned in Section 3.2.1 is chosen as non–optimal, the feasible trajectory calculation time is between 3 and 4 seconds. Even, the total latency of existing system is more than 5 seconds. In order to reduce the latency and pre–processing execution time, Using the tests and observations, the **j** parameter is set to the optimal value ($j = 1.7$). Next, the number of points is reduced either using VG filter and by setting `max_depth` and `min_depth` in `kinect_bridge`. At the start time, the system has 200 thousand points and SOR and ROR filters are applied to it. Pre–processing time is 1.9 seconds now. As you can see in Figure 4.4, VG filter is quite fast and it has had no effect on time. The points are downsampled to 55–60 thousand with VG filter and the maximum and minimum depth parameters on the Kinect side are set to their optimal values.



(a) View of the test script which used for implementation

(b) View of the test script from a different angle

Figure 5.1 *The test trajectory of UR5*

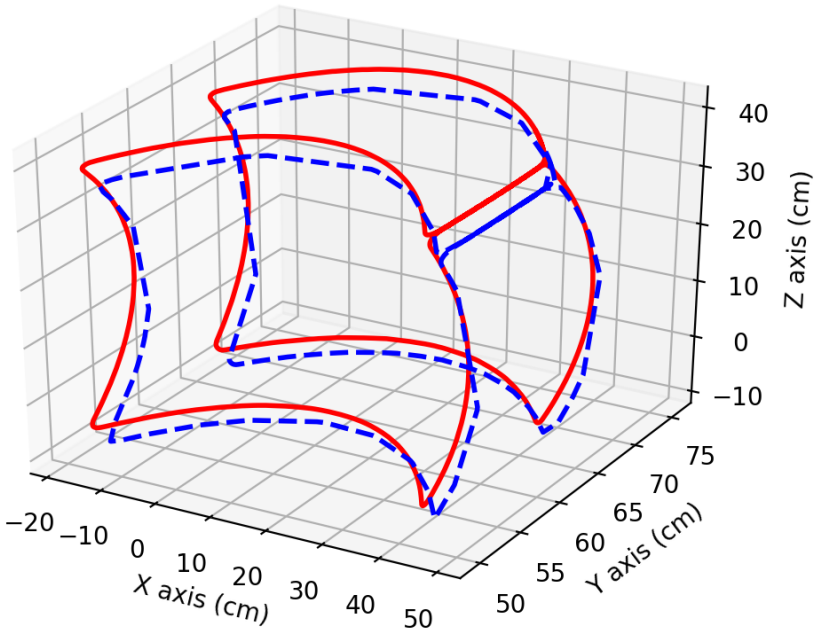


Figure 5.2 *VG, ROR and SOR filters are applied. Cartesian path planning method is used. Red straight line represents the trajectory which robot should follow and blue dash line represents the trajectory which robot actually follows.*

After that, ROR and SOR filters are applied. The result trajectories can be seen

Table 5.1 *ACCURACY OF THE TRAJECTORY USING CARTESIAN PATH PLANNING METHOD WITH VG, ROR AND SOR FILTERS*

Coordinates	Accuracy (Cartesian method)
X	85.79%
Y	87.14%
Z	79.63%

in Figure 5.2. The pre–processing time is reduced to 0.55 seconds. It is observed that it is not adequate for efficient interaction. Following that, the ROR filter is not used in the last stage because it has too much execution time and the capability of removing outlier points is worse than the SOR filter. Finally, VG filter, SOR filter and optimal **j** parameter are used, the total latency is reduced to 0.14 seconds. This is an acceptable level for efficient collaboration. The result trajectories can be seen in Figure 5.3.

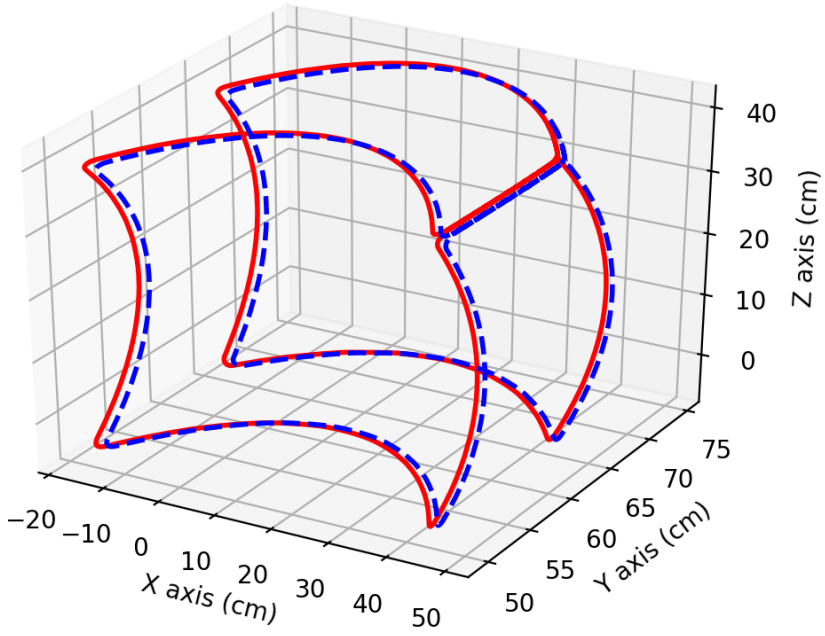


Figure 5.3 *VG and SOR filters are applied. Time parameterization method is used. Red straight line represents the trajectory which robot should follow and blue dash line represents the trajectory which robot actually follows.*

Table 5.2 *ACCURACY OF THE TRAJECTORY USING TIME PARAMETERIZATION METHOD WITH VG AND SOR FILTERS*

Coordinates	Accuracy (Iterative Time Parameterization method)
X	97.32%
Y	93.34%
Z	91.77%

5.2 Discussion

In this study, fingertip is recognized and tracked, same motion is observed on UR5. A real–time algorithm is developed and implemented for smooth trajectory. Moreover, different trajectory planners are tested and compared each other with respect to accuracy. Using the methods, faster response is received. Researchers who want to increase the HRI and trajectory's efficiency can develop their systems by using the PCL library and various filters as it is done in this thesis.

There are limitations during the implementation part. If the points in the PCL is downsampled under 50K points, the resolution of nearest points will be lost. To solve this, Kinect's area is restricted via `min_depth` and `max_depth`. Next, If someone wants to make a computer vision application, the robot manufacturers' software can not be used for acquiring data. For this reason, using open source libraries and softwares becomes a necessity. Several libraries with ROS are used in this project. On the other hand, Microsoft Kinect v2 can not detect the point differences below $1mm$. If less than $1mm$ resolution is required for the project, Intel RealSense[1] 3D camera can be used. At this time, MoveIt! has only search based trajectory planning methods. In order to reach in miliseconds order to plan trajectory, low–level programming (e.g, `movej`, `servoj`) can be used via the `ur_modern_driver` and additional drivers[2]. Lastly, it is difficult to test the real–time application such as recognizing and tracking fingertip and transforming same motion on the robot. Because one can not move his/her hand in the same way accurately. In order to overcome this problem, data can be saved by using "rosbag" file format and then this data can be used for all tests as it is done in this study.

The ROS and its tools have a wide range of capabilities. In the near future, these tools will be used much more. Moreover, robotic companies tend to use open source trajectory planners to reduce the outsourcing costs of trajectory planning. At this

[1]See Intel RealSense.
[2]See hrl_kdl and pykdl_utils.

point, MoveIt! motion planner is a very useful tool for this purpose. By using this work and all these tools, this study pioneers the future works.

5.3 Chapter Summary

In this chapter, the results obtained related to this thesis are shown with figures and tables. According to the results, fingertip was recognized and tracked, same movement of it was observed on UR5. Moreover, it is shown that the parabolic solution is working better than the classic Cartesian solution for smooth trajectory in this particular study.

6. TROUBLESHOOTING

In this chapter, it is explained the problems encountered and its possible solutions during the thesis study.

6.1 Problems and Solutions

All dependencies must be installed to avoid errors. The dependencies can be checked via *libfreenect2* and *IAI Kinect2*. ROS must be installed with full package and all dependencies must be checked beforehand. In this thesis Kinetic Kame version of ROS is used. Microsoft Kinect v2 is working only with Universal Serial Bus 3.0 (USB 3.0). Robot IP must be defined for the real application, following that, the connection of the UR5 can be established by using `ur_modern_driver`.

- If you get permission error for UR driver, invoke Program 6.1.
- If you get the error related to `iai_kinect2/kinect2_registration/CMakeLists.txt`, open this file as a superuser and add this line `add_definitions( -fexceptions )` to it.
- Error related to beignet. If you are using `beignet-1.1.1` version, it must be updated to newer version[1].

```
1 chmod +x /home/<username>/catkin_ws/src/universal_robot/ur_driver/cfg/URDriver.cfg
```

Program 6.1 *UR driver permission error*

[1] See https://launchpad.net/ubuntu/+source/beignet/1.3.1-1.

7. CONCLUSIONS

Firstly, during this work, Microsoft Kinect v2 and Universal Robot 5 were used. As the software and library, ROS, PCL, *libfreenect2*, *IAI Kinect2*, OpenCV, Eigen, `ur_modern_driver`, MoveIt!, OMPL and rviz were used. The data generated by Microsoft Kinect v2 that contain RGB and depth information were received via *IAI Kinect2* (bridge between ROS and *libfreenect2*) and *libfreenect2* (driver required to receive data from Kinect). Next, This data were represented as points through the PCL. VG, ROR and SOR filters were applied to remove the outliers and noises in these points. The nearest point (fingertip) was calculated in real–time using the data purified from outliers and noise with the help of Eigen matrix calculation tool. On every $1cm$ change the calculation was restarted and the nearest point updated. The updated nearest point was published continuously on a `ros_topic`. Following that, it was visualized with PCL and rviz visualization tools.

On the robot side, with the help of `ur_modern_driver`, UR5 was controlled via ROS. In this work, Kinetic Kame version of ROS was run on a computer with Ubuntu 16.04 LTS version installed. The nearest point was subscribed by the relevant `ros_topic`. Then these fingertip differences were sent as a request for trajectory planning to MoveIt!. Next, UR5 was motioned if there was a reasonable solution in these calculations. One thing needs to be mentioned about MoveIt! is the parameter "j" which is used to calculate the feasible trajectory. It can be defined as "joint jump threshold factor" and when the value is set to a value other than zero, it is possible to control how much instantaneous change will occur during motion of the robot joints. It should not be set to zero in real–time test because this causes the robot to motion to an undesirable target. It took 1.9 seconds to execute 1 cycle without setting the parameter "j" and without applying any filter to the system. After that, VG, SOR and ROR filters were applied to the system to recognize the nearest point accurately. In the last case, VG and SOR filters were used. Despite being filtered and performing more calculations, the latency was reduced to 0.14 seconds. Setting the optimal value of the "j" parameter as the result of experiment and observation is a significant step for latency and safety.

Secondly, the limitations can be listed as follows. ROS only works on Linux based

operating systems, so the built–in functions of Kinect can not be used and the external library must be used to get the data (e.g., *libfreenect2* and *IAI Kinect2* for this project). Next, when the downsampling is done with the VG filter, the resolution is lost in point cloud and the point is calculated incorrectly if the points go below 50K. On the other hand, in order to use built–in filters in PCL, only C++ development and programming can be used at this time. Moreover, in MoveIt!, since the search–based calculation method is used, milliseconds can not be reached for the trajectory planning.

As a consequence, the lower the delay is, the better the interaction will be. For future work, one can focus on closed form trajectory planning method to reduce the trajectory calculation time in milliseconds order. In order to achieve this, low–level programming (e.g, `movej`, `servoj`) can be used via the `ur_modern_driver` and related drivers. For the pre–processing delay, Graphics Processing Unit (GPU) can be used instead of using Central Processing Unit (CPU). Besides, the efficiency can be enhanced by using different kind of filters in the PCL.

BIBLIOGRAPHY

[1] "Robot Operating System ROS," accessed: 29-01-2018. [Online]. Available: http://www.ros.org/

[2] A. Aldoma, Z. C. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. B. Rusu, S. Gedikli, and M. Vincze, "Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 80–91, 2012.

[3] T. Andersen, *Optimizing the Universal Robots ROS driver.* Technical University of Denmark, Department of Electrical Engineering, 2015.

[4] J. S. Armstrong and F. Collopy, "Error measures for generalizing about forecasting methods: Empirical comparisons," *International journal of forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[5] H. Asada and J.-J. E. Slotine, "Robot analysis and control," *Automatica*, vol. 24, no. 2, p. 289, 1988. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0005109888900428

[6] M. Brady, *Robot motion: Planning and control.* MIT press, 1982.

[7] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ROS topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[8] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

[9] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised.* Springer, 2017, vol. 118.

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[11] A. J. Critchlow, *Introduction to robotics.* Macmillan New York, 1985.

[12] A. De Luca and F. Flacco, "Integrated control for phri: Collision avoidance, detection, reaction and collaboration," in *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on.* IEEE, 2012, pp. 288–295.

[13] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.

[14] A. De Santis, B. Siciliano, A. De Luca, and A. Bicchi, "An atlas of physical human–robot interaction," *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253–270, 2008.

[15] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices." *Trans. of the ASME. Journal of Applied Mechanics*, vol. 22, pp. 215–221, 1955. [Online]. Available: http://ci.nii.ac.jp/naid/10008019314/en/

[16] R. C. Dorf, *Robotics and automated manufacturing.* Reston Publishing Company, 1983.

[17] K. Elashry and R. Glynn, "An approach to automated construction using adaptive programing," in *Robotic Fabrication in Architecture, Art and Design 2014*. Springer, 2014, pp. 51–66.

[18] J. F. Engelberger, "Robotics in Practice'Kogan Page," 1980.

[19] K. S. Fu, R. C. Gonzales, and C. Lee, "Robotics: Control, Sensing, Vision, and Intelligence. McGrawHill," *Inc., Singapore*, 1987.

[20] S. B. Gokturk, H. Yalcin, and C. Bamji, "A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions," in *2004 Conference on Computer Vision and Pattern Recognition Workshop*, 2004, p. 35.

[21] I. Gradshteyn and I. Ryzhik, "Jacobian determinant," *Proceeding of Tables of Integrals, Series, and Products, 6th ed. San Diego*, pp. 1068–1069, 2000.

[22] M. P. Groover, M. Weiss, R. N. Nagel, and N. G. Odrey, *Industrial robotics: Technology, programming, and applications.* McGraw-Hill New York, 1986.

[23] G. Guennebaud, B. Jacob, and Others, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[24] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced Computer Vision With Microsoft Kinect Sensor: A Review," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[25] T. Hoeniger, "Dynamically shared control in human-robot teams through physical interactions," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 2. IEEE, 1998, pp. 744–749.

[26] *Robots and robotic devices – Safety requirements for industrial robots*, ISO Std. 10 218, 2011.

[27] *Robots and robotic devices – Collaborative robots*, ISO/TS Std. 15 006, 2016.

[28] O. Khatib, K. Yokoi, O. Brock, K. Chang, and A. Casal, "Robots in human environments: Basic autonomous capabilities," *The International Journal of Robotics Research*, vol. 18, no. 7, pp. 684–696, 1999.

[29] Y. Koren and Y. Koren, *Robotics for engineers*. McGraw-Hill New York et al, 1985, vol. 168.

[30] K. Kufieta, "Force Estimation in Robotic Manipulators: Modeling, Simulation and Experiments," M. Eng. thesis, Trondheim, Norway, 2014.

[31] N. B. Kumbla, J. A. Marvel, and S. K. Gupta, "Using Sensor Feedback to Accurately Estimate Part Pose in a Gripper."

[32] A. Kurakin, Z. Zhang, and Z. Liu, "A real time system for dynamic hand gesture recognition with a depth sensor," in *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*. IEEE, 2012, pp. 1975–1979.

[33] F. J. Lawin, P.-E. Forssén, and H. Ovrén, "Efficient multi-frequency phase unwrapping using kernel density estimation," in *European Conference on Computer Vision*. Springer, 2016, pp. 170–185.

[34] C. S. G. Lee, R. C. Gonzalez, and K. S. Fu, "Tutorial on robotics," 1986.

[35] Y. Li, "Hand gesture recognition using Kinect," in *2012 IEEE International Conference on Computer Science and Automation Engineering*, 2012, pp. 196–199.

[36] J. Mainprice, E. A. Sisbot, T. Siméon, and R. Alami, "Planning safe and legible hand-over motions for human-robot interaction," in *IARP workshop on technical challenges for dependable robots in human environments*, vol. 2, no. 6, 2010, p. 7.

[37] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.

[38] K. Mathiassen, J. E. Fjellin, K. Glette, P. K. Hol, and O. J. Elle, "An Ultrasound Robotic System Using the Commercial Robot UR5," *Frontiers in Robotics and AI*, vol. 3, p. 1, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2016.00001

[39] M. L. Minsky, *Robotics*. Doubleday, 1985.

[40] J. Mišeikis, K. Glette, O. J. Elle, and J. Torresen, "Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8.

[41] S. Moe and I. Schjølberg, "Real-time hand guiding of industrial manipulator in 5 DOF using Microsoft Kinect and accelerometer," in *2013 IEEE RO-MAN*, 2013, pp. 644–649.

[42] S. Moe, G. Antonelli, A. R. Teel, K. Y. Pettersen, and J. Schrimpf, "Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results," *Frontiers in Robotics and AI*, vol. 3, p. 16, 2016.

[43] R. R. Murphy and D. Schreckenghost, "Survey of metrics for human-robot interaction," *ACM/IEEE International Conference on Human-Robot Interaction*, pp. 197–198, 2013.

[44] Y. Nakauchi and R. Simmons, "A social robot that stands in line," *Autonomous Robots*, vol. 12, no. 3, pp. 313–324, 2002.

[45] C. L. Nehaniv, K. Dautenhahn, J. Kubacki, M. Haegele, C. Parlitz, and R. Alami, "A methodological approach relating the classification of gesture to identification of human intent in the context of human-robot interaction," in *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on*. IEEE, 2005, pp. 371–377.

[46] "Gazebo Robot Simulation Tool," Open Source Robotics Foundation OSRF, accessed: 29-01-2018. [Online]. Available: http://gazebosim.org/

[47] "MoveIt! Motion Planning Framework," Open Source Robotics Foundation (OSRF), accessed: 29-01-2018. [Online]. Available: http://moveit.ros.org/

[48] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, 1st ed. Cambridge, MA, USA: MIT Press, 1982.

[49] M. Quigley, J. Faust, T. Foote, and J. Leibs, "ROS: an open-source Robot Operating System."

[50] M. R. P. Ragazzon, "TTK4550 Specialization Project Robot Manipulator Collision Handling in Unknown Environment without using External Sensors," p. 90, 2012.

[51] J. L. Raheja, A. Chaudhary, and K. Singal, "Tracking of fingertips and centers of palm using kinect," in *Computational intelligence, modelling and simulation (CIMSiM), 2011 third international conference on*. IEEE, 2011, pp. 248–252.

[52] R. Raja and S. Kumar, "A Hybrid Image Based Visual Servoing for a Manipulator using Kinect," in *Proceedings of the Advances in Robotics*. ACM, 2017, p. 52.

[53] J. A. Rehg, *Introduction to robotics : a systems approach*. Englewood Cliffs, N.J. : Prentice-Hall, 1985.

[54] Z. Ren, J. Meng, and J. Yuan, "Depth camera based hand gesture recognition and its applications in Human-Computer-Interaction," in *2011 8th International Conference on Information, Communications Signal Processing*, 2011, pp. 1–5.

[55] O. Rogalla, M. Ehrenmann, R. Zollner, R. Becher, and R. Dillmann, "Using gesture and speech control for commanding a robot assistant," in *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*. IEEE, 2002, pp. 454–459.

[56] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, may 2011.

[57] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[58] J. Saarinen, J. Suomela, and A. Halme, "The Concept of Future Worksite–Towards Teamwork-centered Field Robotic Systems," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 14 952–14 957, 2011.

[59] P. Sanz, *Robotics: Modeling, Planning, and Control*, 2009, vol. 16, no. 4.

[60] D. G. Schneider, L. L. d. Silva, P. Diehl, A. H. R. Leite, and G. S. Bastos, "Robot Navigation by Gesture Recognition with ROS and Kinect," in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, 2015, pp. 145–150.

[61] M. Shahinpoor, *A robot engineering textbook*. Harper & Row Publishers, Inc., 1987.

[62] D. Shin, S. Shin, D. Kim, S. Kim, J. Hwang, and Y. Kim, "Visual guidance system for remote-operation," in *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2016, pp. 657–661.

[63] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics–Modelling, Planning and Control. Advanced Textbooks in Control and Signal Processing Series.* London, UK: Springer-Verlag, 2009.

[64] C. P. Simon and L. E. Blume, *Mathematics for Economists. 500 Fifth Avenue, New York, NY 10110.* WW Norton & Company, Inc, 1994.

[65] W. E. Snyder, *Industrial robots: computer interfacing and control.* Prentice Hall PTR, 1985.

[66] M. W. Spong, S. Hutchinson, M. Vidyasagar *et al.*, *Robot modeling and control.* Wiley New York, 2006, vol. 3.

[67] K. Stubbs, P. Hinds, and D. Wettergreen, "Challenges to grounding in human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction.* ACM, 2006, pp. 357–358.

[68] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," in *Ro-man, 2012 IEEE.* IEEE, 2012, pp. 411–417.

[69] B. Teke, M. Lanz, J. Kämäräinen, and A. Hietanen, "Real-time robust collaborative robot motion control with microsoft kinect," in *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA) (MESA2018)*, Oulu, Finland, Jul. 2018.

[70] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[71] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro *et al.*, "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 79–91, 2004.

[72] J. G. Trafton, A. C. Schultz, D. Perznowski, M. D. Bugajska, W. Adams, N. L. Cassimatis, and D. P. Brock, "Children and robots learning to play hide and seek," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction.* ACM, 2006, pp. 242–249.

[73] M. R. Walter, M. Antone, E. Chuangsuwanich, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, Y. Friedman, J. Glass, J. P. How, and Others, "A Situationally Aware Voice-commandable Robotic Forklift Working Alongside People in Unstructured Outdoor Environments," *Journal of Field Robotics*, vol. 32, no. 4, pp. 590–628, 2015.

[74] Y. Wang, Y. Sheng, J. Wang, and W. Zhang, "Optimal Collision-Free Robot Trajectory Generation Based on Time Series Prediction of Human Motion," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 226–233, 2018.

[75] T. Wiedemeyer, "IAI Kinect2," https://github.com/code-iai/iai_kinect2, Institute for Artificial Intelligence, University Bremen, 2014 – 2015, accessed June 12, 2015.

[76] K. Wilfred, *Advanced calculus.* Addison-Wesley Longman, Boston, 2002.

[77] W. A. Wolovich, *Robotics: basic analysis and design.* Holt, Rinehart & Winston, 1986.

[78] L. Xiang, F. Echtler, C. Kerl, T. Wiedemeyer, Lars, Hanyazou, R. Gordon, F. Facioni, Laborer2008, R. Wareham, M. Goldhoorn, Alberth, Gaborpapp, S. Fuchs, Jmtatsch, J. Blake, Federico, H. Jungkurth, Y. Mingze, Vinouz, D. Coleman, B. Burns, R. Rawat, S. Mokhov, P. Reynolds, P. E. Viau, M. Fraissinet-Tachet, Ludique, J. Billingham, and Alistair, "libfreenect2: Release 0.2," 2016. [Online]. Available: https://doi.org/10.5281/zenodo.50641

[79] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," *IEEE MultiMedia*, vol. 19, no. 2, pp. 4–10, 2012.

Appendices

A. SKEW SYMMETRIC MATRIX

Skew symmetric matrices simplify the calculation of the derivative of a rotation matrix. A matrix is called skew symmetric if the transpose is equal to its negative. It can be seen in Equation (A.1).

$$\boldsymbol{S}^T + \boldsymbol{S} = 0 \tag{A.1}$$

The set of 3×3 skew symmetric matrices can be denoted by $SO(3)$. s_{ij} is the elements of $\boldsymbol{S}$ that can be seen in Equation (A.2).

$$s_{ij} + s_{ji} = 0 \quad i, j = 1, 2, 3 \tag{A.2}$$

Equation (A.3) shows that $\boldsymbol{S}$ contains only three independent entries.

$$\boldsymbol{S} = \begin{bmatrix} 0 & -s_3 & s_2 \\ s_3 & 0 & -s_1 \\ -s_2 & s_1 & 0 \end{bmatrix} \tag{A.3}$$

$\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ can be denoted as the three unit basis coordinate vectors. Definitions are shown in Equation (A.4).

$$\boldsymbol{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{A.4}$$

The skew symmetric matrices $\boldsymbol{S}(\boldsymbol{i})$, $\boldsymbol{S}(\boldsymbol{j})$ and $\boldsymbol{S}(\boldsymbol{k})$ are given by Equation (A.5).

$$\boldsymbol{S}(\boldsymbol{i}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{S}(\boldsymbol{j}) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{S}(\boldsymbol{k}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{A.5}$$

Following that, the derivative of the basic rotation matrices can be seen in Equation (A.6).

$$\begin{aligned}\frac{d}{d\alpha}\boldsymbol{R}_{x,\alpha} &= \boldsymbol{S}(\boldsymbol{i})\boldsymbol{R}_{x,\alpha}\\ \frac{d}{d\beta}\boldsymbol{R}_{y,\beta} &= \boldsymbol{S}(\boldsymbol{j})\boldsymbol{R}_{y,\beta}\\ \frac{d}{d\gamma}\boldsymbol{R}_{z,\gamma} &= \boldsymbol{S}(\boldsymbol{k})\boldsymbol{R}_{z,\gamma}\end{aligned} \tag{A.6}$$

Time varying rotation matrix denoted as $\boldsymbol{R} = \boldsymbol{R}(t)$.

$$\boldsymbol{S}(\omega(t)) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{A.7}$$

$\omega(t)$ is the angular velocity of the rotating frame with relative to the base frame at time t. Therefore, $\boldsymbol{S}(t)$ can be represented as $\boldsymbol{S}(\omega(t))$. It is shown in Equation (A.7). The time derivative $\dot{\boldsymbol{R}}(t)$ is given by Equation (A.8).

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{S}(\omega(t))\boldsymbol{R}(t) \tag{A.8}$$

B. UNIVERSAL ROBOT–5 SPECIFICATIONS

In this appendix, UR5 specifications and its MDH parameters are given.

B.1 Universal Robot–5 Technical Specifications

Weight:	18.4kg
Payload:	5kg
Reach:	$850mm$
Joint ranges:	$\pm 360°$
Speed:	Joint: max. $180°$/sec. ; Tool: $\approx 1m$/sec.
Repeatability:	$\pm 0.1mm$
Degrees of freedom:	6 revolute joints
Controller size:	$475 \times 423 \times 268mm$
I/O ports:	10 Digital Input, 10 Digital Output;
	4 Analog Input, 2 Analog Output
I/O power supply:	24V 1.2A in controller, 12V/24V 0.6A in tool
Communication:	TCP/IP–Ethernet, Modbus TCP
Programming:	URScript
Noise:	Comparative noiseless
IP classification:	IP54
Power consumption:	≈ 200 watts
Collaboration operation:	EN ISO $13849 : 2008$; EN ISO $10218 - 1 : 2011$
Materials:	Aluminum, PP plastic
Temperature:	$0 - 50°$C
Power supply:	$100 - 240$ VAC, $50 - 60$ Hz

B.2 Universal Robot–5 MDH Parameters

MDH parameters of UR5 can be seen in Table B.1. On the other hand, the homogeneous transformation of each link can be seen in Equations (B.1) to (B.6).

Table B.1 *UR5 MDH PARAMETERS*

(a) MDH parameters symbols

Link i	θ_i	d_i	a_i	α_i
1	θ_1	d_1	0	α_1
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0
4	θ_4	d_4	0	α_4
5	θ_5	d_5	0	α_5
6	θ_6	d_6	0	0

(b) MDH parameters values

Link i	$d_i[mm]$	$a_i[mm]$	$\alpha_i[degree]$
1	89.16	0	90°
2	0	−425	0
3	0	−392.25	0
4	109.15	0	90°
5	94.65	0	−90°
6	82.3	0	0

$$\boldsymbol{A}_1 = \begin{bmatrix} \boldsymbol{R}_1^0 & \boldsymbol{o}_1^0 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_1} & 0 & s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & -c_{\theta_1} & 0 \\ 0 & 1 & 0 & 89.16 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

$$\boldsymbol{A}_2 = \begin{bmatrix} \boldsymbol{R}_2^1 & \boldsymbol{o}_2^1 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_2} & -s_{\theta_2} & 0 & -425c_{\theta_2} \\ s_{\theta_2} & c_{\theta_2} & 0 & -425s_{\theta_2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.2}$$

$$\boldsymbol{A}_3 = \begin{bmatrix} \boldsymbol{R}_3^2 & \boldsymbol{o}_3^2 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_3} & -s_{\theta_3} & 0 & -392.25c_{\theta_3} \\ s_{\theta_3} & c_{\theta_3} & 0 & -392.25s_{\theta_3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.3}$$

$$\boldsymbol{A}_4 = \begin{bmatrix} \boldsymbol{R}_4^3 & \boldsymbol{o}_4^3 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_4} & 0 & s_{\theta_4} & 0 \\ s_{\theta_4} & 0 & -c_{\theta_4} & 0 \\ 0 & 1 & 0 & 109.15 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.4}$$

$$\boldsymbol{A}_5 = \begin{bmatrix} \boldsymbol{R}_5^4 & \boldsymbol{o}_5^4 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_5} & 0 & -s_{\theta_5} & 0 \\ s_{\theta_5} & 0 & c_{\theta_5} & 0 \\ 0 & -1 & 0 & 94.65 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.5}$$

$$\boldsymbol{A}_6 = \begin{bmatrix} \boldsymbol{R}_6^5 & \boldsymbol{o}_6^5 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_6} & -s_{\theta_6} & 0 & 0 \\ s_{\theta_6} & c_{\theta_6} & 0 & 0 \\ 0 & 0 & 1 & 82.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.6}$$

Finally, the transformation matrix $\boldsymbol{T}_6^0$ that describes the end–effector position and orientation with respect to the base frame can be derived by using Equation (B.7).

$$\boldsymbol{T}_6^0 = \boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_4\boldsymbol{A}_5\boldsymbol{A}_6 = \begin{bmatrix} \boldsymbol{R}_6^0 & \boldsymbol{o}_6^0 \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.7}$$