İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

Control and Design of Industrial Robotic Arm by Using ROS

Submitted to the Graduate School of Natural and Applied Sciences in partial fulfillment of the requirements for the degree of

Master of Science

in Mechanical Engineering

by

Adem Candemir

ORCID 0000-0002-0328-6636

July, 2022

Declaration of Authorship

I, **Adem Candemir**, declare that this thesis titled **Control and Design of Industrial Robotic Arm by Using ROS** and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for the Master's / Doctoral degree at this university.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. This thesis is entirely my own work, with the exception of such quotations.
- I have acknowledged all major sources of assistance.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: 06.07.2022

_____

Control and Design of Industrial Robotic Arm by Using ROS

Abstract

Industrial robots currently used in industry all have their own interfaces and unique languages. These commonly used interfaces and languages are often limited to industry needs. Although there are several robotic solutions, practically all of them can't rectify problems on their own. When a program is required for a more complex task, machine vision locates objects and puts them in desired locations and for situations such as applying the correct force for assembly operations by measuring with force-torque sensors, they often need third-party programs as existing systems cannot respond alone.

It is more preferable to have a system that can control the robot alone, rather than using both the robot's interface and language by integrating third-party programs to create advanced programs needed in the industry. Such middleware is available, one of which is the Robot Operating System (ROS).

This framework is easy to learn and install, and once the system is set up it can be utilized on a range of industrial robots from various manufacturers and can be used for a variety of purposes.

This thesis develops a method that is flexible enough for object recognition and pick&place applications in the industry. The main purpose of this study is to create a

serial physical scara manipulator with three degrees of freedom developed by the author for the real test and an Ur5 industrial robot for simulation application to move these manipulators from the positions of objects of predetermined color with the help of machine vision to the specified station by using the Ros framework. Our method consists of 3 steps:1-appropriate design in Solidworks,2-object recognition (open source OpenCV), and 3- control algorithm with Ros. The proposed method can be advantageous in the field of robotics and has the potential to be implemented in a variety of industries, such as the food industry for Packing and Palletizing, etc.

Keywords: Scara Manipulator, ur5 Universal Robot, Robot Operating System, Machine Vision, MoveIt

ROS Kullanarak Endüstriyel Robotik Kolun Kontrolü ve Tasarımı

Öz

Endüstride kullanılan endüstriyel robotların hepsinin kendine ait ara yüzler ve benzersiz farklılıkları vardır. Bu yaygın olarak kullanılan ara yüzler ve diller genellikle endüstri ihtiyaçları için sınırlıdır. Sahada birçok robotik çözüm bulunmasına rağmen, hemen hepsinin görevleri sırasında hataları otonom olarak düzeltme konusunda sınırlamaları vardır. Daha karmaşık bir görevler için bir program gerektiğinde, örneğin makine görmesiyle nesnelerin pozisyonlarını tespit etme ve istenen yerlere yerleştirme ve kuvvet-tork sensörleri ile ölçüm yaparak montaj işlemleri için doğru kuvveti uygulamak gibi durumlarda bu sistemler tek başına yanıt veremezler ve genellikle mevcut olarak üçüncü taraf programlarına ihtiyaç duyarlar.

Endüstride ihtiyaç duyulan gelişmiş programları oluşturmak için üçüncü parti programları entegre ederek hem robotun ara yüzünü hem de dilini kullanmak yerine robotu tek başına kontrol edebilen bir sisteme sahip olmak daha tercih edilir. Bu tür ara katman yazılımları mevcuttur ve bunlardan biri Robot İşletim Sistemi'dir.

Bu ara katman yazılımının öğrenilmesi ve kurulması kolaydır ve sistem kurulduktan sonra farklı üreticilerin endüstriyel robotlarında farklı amaçlar için kullanılabilir.

Bu tez, sektördeki nesne tanıma ve al ve yerleştir uygulamaları için yeterince esnek bir yöntem geliştirmektedir. Bu çalışmanın temel amacı, yazar tarafından gerçek test için

geliştirilmiş üç serbestlik dereceli seri bir fiziksel Scara manipülatörü ve simülasyon uygulaması için Ur5 endüstriyel robotlarını oluşturmak ve bu manipülatörleri makine görmesi yardımıyla önceden belirlenmiş renkteki nesnelerin konumlarını tespit ederek nesneleri bulunduğu noktadan hareket ettirerek belirtilen istasyona Ros ara katman çerçevesi yardımıyla bırakmaktır. Metodumuz 3 adımdan oluşur:1-Geliştirmiş Scara manipulatorü için Solidworks'te uygun tasarım veya Ur5 endüstri robotu için firma tarafından oluşturulmuş tasarım,2-nesne tanıma (açık kaynak OpenCV) ve 3-Ros ile kontrol algoritması. Önerilen yaklaşım robotik alanında değerli olabilir ve paketleme ve paletleme için gıda endüstrisi gibi birçok sektörde potansiyel olarak uygulanabilir.

Anahtar Kelimeler: Scara Manipulator, Ur5 Universal Robot, Robot İşletim Sistemi, Makine Görmesi, MoveIt.

To my mother, father, wife and siblings,

Acknowledgment

I would like to thank my esteemed advisor Assist. Prof. Dr. Fatih Cemal Can, who helped me at every stage of my thesis, who contributed to the realization of this study, and guided me with her ideas during my thesis follow-up. I would like to thank my dear wife Arife Nur Candemir, who has never spared her moral support, and my mother Zülfiye Candemir and my father Nevzat Candemir, who supported me with their wealth in my most difficult times and supported me throughout my life and brought me to where I am today.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

PLA	Polilaktik Asit
DH	Denavit-Hartenberg
Orocos	Open Robot Control Software
Pyro	Python Robotics
Orca	Optimal Reciprocal Collision Avoidance
RSCA	Robot Software Communications Architecture
MRDS	Microsoft Robotics Developer Studio
OPROS	Open Platform for Robotic Services
CLARAty	Coupled Layer Architecture for Robotic Autonomy
ROS	Robot Operating System
YARP	Yet Another Robot Platform
BSD	Berkeley Software Distribution
IMU	Inertial Measurement Unit
XML	Extensible Markup Language
URDF	Unified Robot Description Format
Xacro	XML Macro Language
OMPL	Open Motion Planning Library
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
STOMP	Stochastic Trajectory Optimization for Motion Planning
SBPL	Search-Based Planning Library
KDL	Kinematics and Dynamics Library
LMA	Levenberg-Marquardt

List of Symbols

P	Pressure [Pa]
ω	Angular velocity [deg/s]
V	Linear velocity [m/s]
α	Angular acceleration [deg/s^2]
a	Linear acceleration [m/s^2]
F	Force [N]
τ	Torque [N.m]
m	Mass [kg]

Chapter 1

Introduction

1.1 Purpose of the Thesis

In this thesis, it is aimed to create the basic data communication infrastructure necessary for the communication of the programs to be used in the operation of a manipulator or robot arm designed to work in industrial environments and it is aimed to create a software modular control infrastructure to control the actuators in these robots. The main aim of the study is to solve the synchronization problems that will occur in the message communication between the programs used for the manipulators to fulfill their tasks, with a specialized operating system for robotics, and to try to modularize the link between low-level tasks and high-level tasks. Thus, in any hardware change of the robot, the lower level will be adapted to the hardware change and the upper level will be prevented from being changed according to the hardware. Solving the communication and control infrastructure with an operating system will accelerate the development process of the robot. Due to the modular structure of the software, many different controls, image processing, localization, etc., methods can be tested on the same robot. Since such robotic structures are very complex, this infrastructure will allow developers from different disciplines to develop the system in parallel. In addition, since the software provides great convenience to users with its simulation environments, it will be possible for developers to contribute to the project globally, not only from the laboratory but from anywhere in the world

1.2 Problem Description

The interaction between humans and robots is increasing day by day, and as technology develops, robots have begun to enter people's living spaces more. The more

this interaction increases, the more important it becomes for a human to work in harmony with the robot. Proper human and robot communication is critical to achieving this harmony. In addition, since manipulators and robotic arms work together to assist people in working and industrial environments, the robot must have a stable operating system, latency-free message communication, and control infrastructure. Industrial robotics tasks include many infrastructures of the entire robotics field. When we evaluate the daily job descriptions of a person working in the industry, it is seen that all of these jobs are sub-topics researched in the manipulator and robotic arm. For this reason, more than one researcher should work on developing industrial robotic systems. Working with more than one researcher means tens, maybe hundreds of software operating on a robot at the same time. Gathering the different software and software libraries used under a common operating system and ensuring that they work in harmony on the robot is a very difficult problem and since many algorithms will be tried on the robot, this operating system must have a modular structure. In this study, the solution offered to the high-level software integration problem and control infrastructure problem of the Scara and Ur5 industrial robot project, which will be carried out under the supervision of Asst. Prof. Fatih Cemal Can, is ROS, which is called Robot Operating System and released as open software by Willow Garage. In addition, in this thesis, integration of ROS to Scara and Ur5 is designed.

1.3 Literature Search

The ideal features of the operating systems used by industrial robots are discussed in the study and the following features are presented: (i) easy to install, (ii) open to development, (iii) easy to use, and adaptable. Another criterion, which is taken into account in the study [1], is that students with limited programming knowledge are asked to write programs for these operating systems. Operating systems are rated out of a hundred. ROS was given a score of 71 and placed in fourth place. The reason why ROS is fourth in this ranking is that it is not easy to use and the user's adaptation period is long, but when developing an industrial, it should be considered that the developers must be at a higher level than limited programming knowledge and that more than one developer is working on this robot. It is emphasized in [2] that industrial robots are complex structures and should have an efficient communication structure and control

infrastructure. In addition, in [2], it was pointed out that ROS has grown with the collaboration of academia and industry, Bosch and Intel are among the prominent industry developers. Since industrial robotics has a complex structure, it is stated in [3] that if a standardized operating system structure is not established, the solved problems will have to be constantly reworked. This standardization will be a step in the rapid robot development step and will not waste developers' time with hardware and software incompatibilities. In addition, such a standardized structure will be a bridge between the developer in the academy and the professional developer, and the contributions of the industry, which progresses with science, to the science of robotics will accelerate. It is understood that ROS is a step in the standardization process, since DARPA, which organizes many robot competitions, chose Gazebo, which is a project of ROS and used as a simulator in this study, and invested six million dollars in simulation competitions [4]. Google, which has taken important steps in cloud robotics, has developed the ROS Java infrastructure so that ROS-based robots can work in harmony with Android-based embedded systems [5]. ROS software repositories of many important universities such as MIT, UC Berkeley, and Stanford are open to the public. The infrastructure of the Robonaut project, developed by NASA and General Motors, is being tried to be fully adapted to ROS [6].It has been emphasized that the benefits of ROS and OROCOS can be used in studies that require high performance [6]. OROCOS was used in this study not for its real-time operating system feature, but for the kinematic and dynamic libraries it provides. It is stated in [7] that the number of different types of robots working with ROS is 140, but 28 types of different robots officially use ROS, and the total number of ROS repositories passed 100 on 5 May. Manipulation and localization are the primary benefits of ROS. ROS is basically a middleware concentrated on 2 different robot sub-categories. These categories are robotic arms and autonomous mobile robots. Let's examine these two categories in order to better understand them respectively.

The strong high-level software infrastructure of ROS has begun to be configured to be used in industrial robots. Southwest Research Institute has started a program called ROS-Industrial. The purpose of this program is stated as filling the gap between commercial industrial automation and academia. The leading benefits of this program are stated as non-commercially available collision-free motion planning and autonomous pick-and-place applications [8] . Industrial manipulators for ABB, Adept,

Fanuc, Motoman, and Universal Robots are supported by ROS-Industrial packages [9]. The robot models of these famous brands supported by ros are as follows according to the brand. ABB brand IRB-2400, IRB-5400 models, Adept brand Viper 650, Fanuc brand LR Mate 200iC (all), LR Mate 200iD,5 M-10iA, M-16iB/20, M-20iA(/10L), M- 430iA/ (2F, 2P), M-900iA/260L 5, Motoman brand's SIA10D/F, SIA20D/F, MH5F, SDA10F, Universal Robot brand's UR 5, UR 10 models are supported. More information on ROS-Industrial can be found in [9] [10]. Unlike these, the following are robotic manipulators for training and testing purposes. myCobot, Net, MIPJunior, xArm, XSeries Robot Arm, Niryo one, Sciurus17, CRANE-X7, Doosan-Robot, gauss, r12, Open Manipulator, Revel, Elfin, Han's Cute, Schunk Lwa 4P, ROBOTIS Manipulator, MICO, JACO, SRA Service Robot Arm, Mover, AUBO I-Series, Cyton Gamma.

To better understand this, we can evaluate two robots commercially sold by Willow Garage. The first is the highly developed PR2 and the inexpensive Turtlebot. The PR2 robot is a service robot developed to perform humanoid tasks in human environments. Through the work of universities such as Freiburg, GATech, MIT, Stanford, TUM, and Berkeley, the PR2 has been taught to help people with household chores such as cleaning the table, setting the table, and unloading the dishwasher, washing the laundry, and in general. Bosch has granted remote access to PR2 for scientists working in academia [8]. These examples were made thanks to the ROS support and hardware support of the PR2 robot. The other robot is Turtlebot, which has a lower cost and open-source license. As a component, iRobot create is a robot equipped with Microsoft Kinect and a computer. Thanks to its low cost, the range of costs to access robots has been expanded from educators to hobbyists. Thanks to the open-source license, many derivatives have been produced and offered for sale [9]. Thanks to ROS support, Turtlebot has become a powerful robot where complex robotic algorithms can be tested. In the study [11], the ROS-based PIONEER-3DX robot was used for autonomous navigation, mapping, and localization purposes. The navigation applications provided by ROS have been tested in three different environments and successful results have been reported. An autonomous underwater robot is presented in the study [12]. ROS is used in the high-end software of the underwater robot and it has been shown that the purpose of using ROS is to provide communication and cooperation between autonomous underwater robots. The importance of ROS was

emphasized in the simulation, testing, and task realization stages and it was stated that the study had a successful application. In the study [13], the navigation package of ROS was combined with RFID, and object localization was applied, and according to the result of the study[13] , the unification revealed a system with a low margin of error, reliable, and power efficiency. The CAMBADA robot is a robot prepared by the University of Aveiro for the ROBOCUP@HOME competition. The team migrated the entire robot's high-end software to ROS in 2012. In the team introduction article, it was emphasized that ROS has become a standard in robotics and provides great convenience due to its modular structure [14]. Delft University of Technology's robot DRP1 uses the navigation package of ROS for localization and Microsoft Kinect for three-dimensional imaging structure [15]. The Donaxi@HOME robot is the robot of UPAEP University in Mexico and uses the KDL Library for manipulation and ROS for navigation [16]. The team of Koblenz and Landau University in Germany homer@UniKoblenz used ROS as the operating system in the ROBOCUP@HOME competition to take advantage of ROS's ever-growing algorithm repository and to have a modular structure [17]. The University of Bonn team, NimbRO, has solved the communication infrastructure problem and the navigation problem with ROS [18].

Chapter 2

Design and Analysis

In this chapter, design procedure of Scara manipulator is explained. Then, kinematics analysis of the manipulator is formulated. In addition, since only the simulation study of the Ur5 industrial robot will be carried out, information about the kinematic solutions using the mentioned resources is presented in section 2.3.

2.1 Three-Dimensional Mechanical Design of Scara Manipulator

Scara manipulator has 3 degrees of freedom. A total of 2 bus servo motors and 1 single-acting pneumatic cylinder are used, which gives the Scara manipulator 3 degrees of freedom. The reason for choosing bus servo motors as actuators is that the servo motors move according to the position references and remain fixed at these position references. The distributions of the actuators on the manipulator are as follows:

- ✓ 1 on the shoulder
- ✓ 1 on the elbow
- ✓ 1 on the end point

The manufacturer of servo motors is Hiwonder. Pneumatic cylinder manufacturer is Expflex. The electromagnet P20/15, whose brand is a Chinese company, was bought from the market. The reason for choosing Hiwonder motors is because the position reference exceeding problem found in most of the bus servo motors is minimized and the position, temperature, voltage feedback is received. In the creation of the Scara, 1 model of the Hiwonder servo motors and 1 model for the pneumatic cylinder were

used. The model of the motors on the shoulder and elbow is HTS-35H, the model of the pneumatic cylinder at the end is EXPFLEX MSA 16-50. The HTS-35H model is powered at 9.0 Volts and 12.6 Volts, and has an output of 35 kg-cm at 11.1 volts. EXPFLEX MSA 16-50 pneumatic cylinder works by being triggered by a relay with 12 Volt voltage. It can carry 3.0 Kg load applying a pressure P=4.5 Pa. The P20/15 electromagnet is activated with 12 Volts and has a holding power of 2.5 Kg. Since the maximum load that Scara is expected to lift is 500 gr, the torque outputs and load-lifting capacities of these motors, pneumatic cylinders and electromagnets have been predicted appropriately for the design. The calculation of the maximum load as 500 gr is due to the fact that Scara does not need to lift very heavy loads, as it will be used in training. These materials used in the construction of the Scara manipulator are shown in Figure 2.1, Figure 2.2, and Figure 2.3 respectively. A fruit or various toys can be given as an example of the loads that Scara will lift.

Figure 2.1: HTS-35H bus servo and its driver [19]

Figure 2.2: Single acting pneumatic cylinder [20]

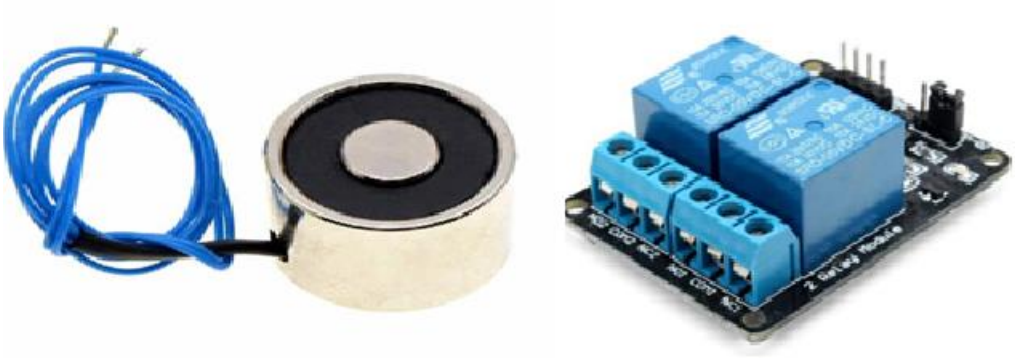


Figure 2.3: The P20/15 electromagnet [21]

Solid modeling of Scara manipulator was done in SolidWorks program. Materials moving together are combined as sub-assembly diagrams in SolidWorks, which provides both simplification and processing speed in the Gazebo simulation program. Figure 2.4 shows the shape of a manipulator part arranged as a sub-assembly.

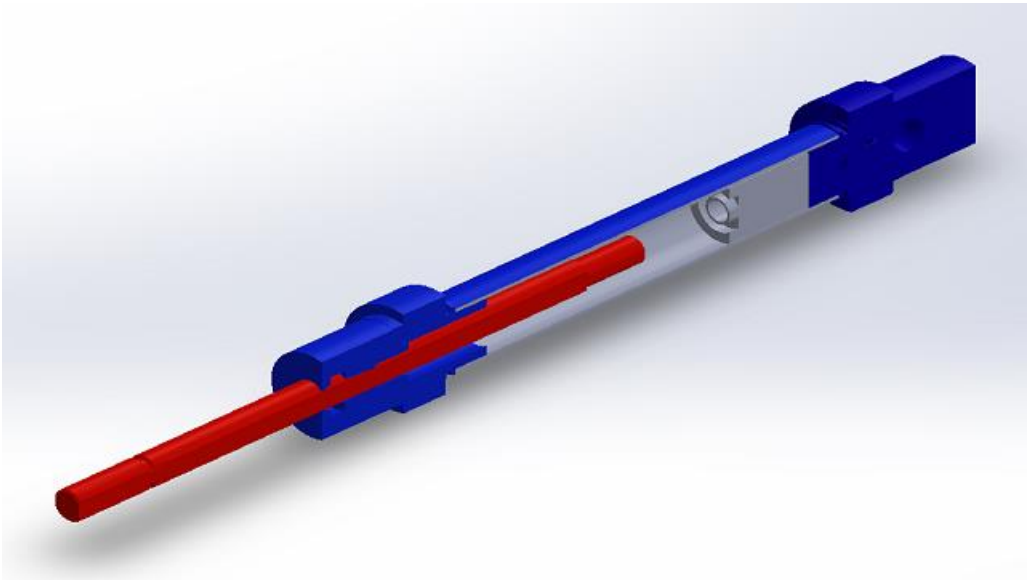


Figure 2.4: Sub-assembly of the end-effector

The solid model structure of Scara is shown in Figure 2.5 and Figure 2.6.

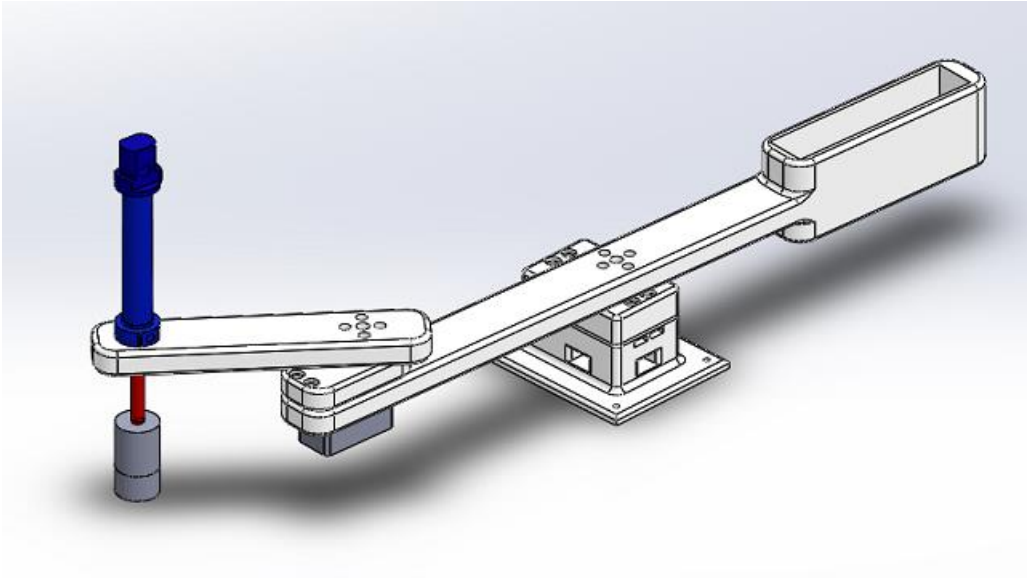


Figure 2.5: Assembled Scara manipulator isometric view

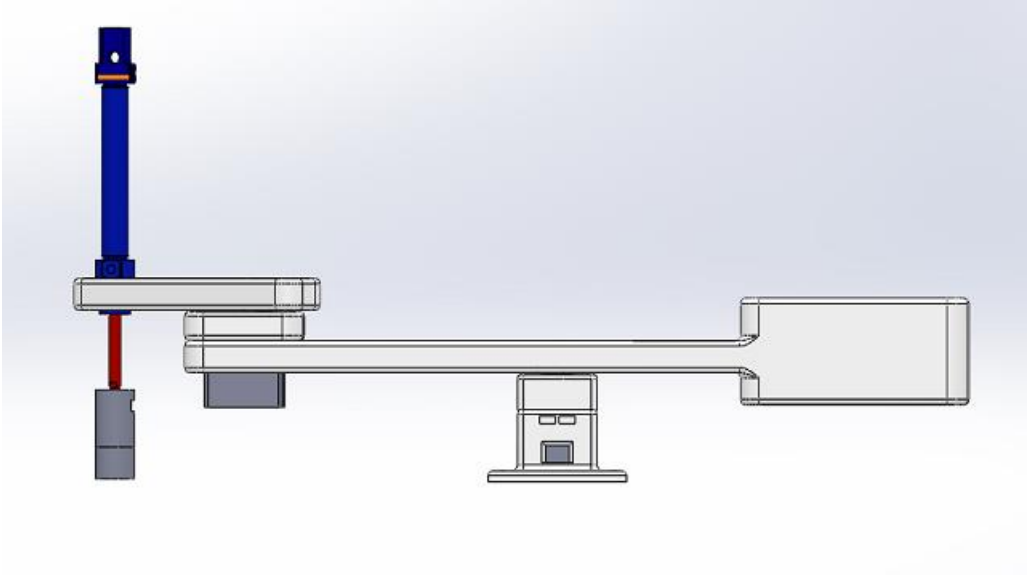


Figure 2.6: Side view of the assembled Scara manipulator

The limits of rotation of the joints are assigned from -120 degrees to +120 degrees, which is the limit of the motors. The first simulation tests of the arm were made in motion analysis, a plug-in of Solidworks. In this program, it has been observed that the torques on the joints match the forward dynamic torque values obtained in Theory, these data are given in detail in chapter 2, section 2.2.5.

All the links that make up the manipulator are 250 mm in size, as link1 is 150 mm and link2 is 100 mm. The materials of the links are Pla, the materials of the fasteners and bearings are chosen as aluminum. The PLA material was chosen because it allows a lightweight link design that can be lifted by the motors at the joints of the manipulator. The weight of Link1 is 60 gr. The weight of Link2 is 50 gr. The weight of the motors is the same, 90 gr. The total weight of the single-acting pneumatic cylinder and the electromagnet is 125 gr. The total weight of the Scara manipulator is approximately 325 gr. The design principle, which is prioritized in the entire design of the manipulator, is based on a modular structure of the manipulator as a whole. The convenience of this modular structure is that when any part of the manipulator is stretched or broken, it removes the broken area and replaces it with a new part, allowing the manipulator to perform its functions again quickly.

2.2 Kinematic and Dynamic Analysis

Manipulator kinematics recognizes the motion of the manipulator's joints and the motion of the solid body that makes up the entire manipulator. The joints of most robotic arms are driven by hydraulics, electric motors and pneumatic actuators. Manipulator dynamics defines how the manipulator will move with the effect of torques or forces applied to the joints [22] .

2.2.1 Forward Kinematic Analysis

Mathematically forward kinematics equations describe the position and orientation in cartesian coordinates and the joint position space [23]. The forward kinematics of a serial kinematic chain is written in its most general form as shown in Equation (2.1).

$$ {}^{0}_{n}A = \prod_{i=1}^{n} {}^{i-1}_{i}A(Q_i) \tag{2.1} $$

Here, Q_i is the joint angle parameter, and the ${}^{i-1}_{i}A$ parameter is the transformation matrix between the i. joint of the manipulator and the i-1. joint of the manipulator. In robotics, the transformation matrix expresses the transformations between the

coordinate axes that are added to the robot's joints virtually. The transformation matrix can be written in accordance with the DH (Denavit-Hartenberg) method. In order to start these analyses, we first define the Denavit Hartenberg parameters. This table is shown in Table 2.1.

Table 2.1:Denavit Hartenberg parameters

Axes	α_i	a_i	d_i	Q_i	**İnitial**
1	0	a_1	0	Q_1	0
2	0	a_2	0	Q_2	0

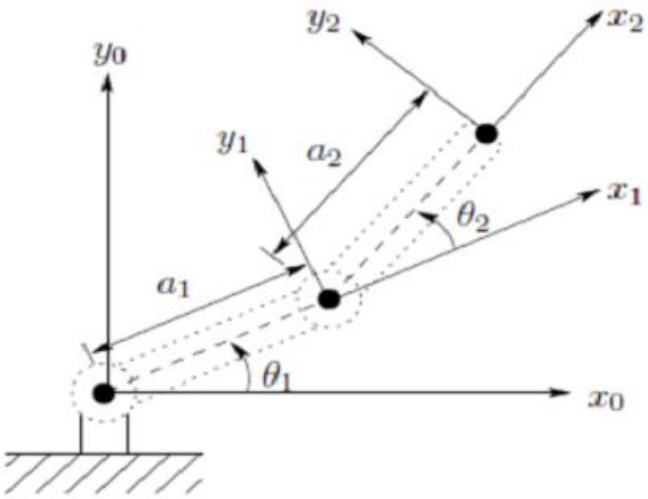


Figure 2.7: An example robot arm with two degrees of freedom

Denavit Hartenberg parameters are considered as twist angle(α_i), link length (a_i), offset (d_i), and joint angles (Q_i). Twist angles and offsets are zero for the Scara manipulator. Link lengths are constant (a1=150mm, a2=100mm). Joint angles are variables and they will be controlled using actuator and Ros control packages.

Transformation matrices should be calculated before forward and inverse tasks. The General Transformation formula and matrix are shown in Equations (2.2) and (2.3), respectively.

$$^{i-1}_{i}A = T_{(Z,d_i)}.T_{(Z,Q_i)}.\ T_{(X,a_i)}.\ T_{(X,\alpha_i)} \tag{2.2}$$

$$^{i-1}_{i}A = \begin{bmatrix} \cos(Q_i) & -\cos(\propto_i).\sin(Q_i) & \sin(\propto_i).\sin(Q_i) & a_i.\cos(Q_i) \\ \sin(Q_i) & \cos(\propto_i).\cos(Q_i) & -\sin(\propto_i).\cos(Q_i) & a_i.\sin(Q_i) \\ 0 & \sin(\propto_i) & \cos(\propto_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Transformation matrix of link1 wrt. base link is written as follows,

$$^{0}_{1}A = \begin{bmatrix} \cos(Q_1) & -\sin(Q_1) & 0 & a_1.\cos(Q_1) \\ \sin(Q_1) & \cos(Q_1) & 0 & a_1.\sin(Q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Transformation matrix of link2 wrt. link1 is written as follows,

$$^{1}_{2}A = \begin{bmatrix} \cos(Q_2) & -\sin(Q_2) & 0 & a_2.\cos(Q_2) \\ \sin(Q_2) & \cos(Q_2) & 0 & a_2.\sin(Q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Transformation matrix of link2 wrt. base link is calculated by multiplying matrixes which are given in Equations (2.4),and (2.5).Result matrix is found as follows,

$$^{0}_{2}A = \begin{bmatrix} \cos(Q_{12}) & -\sin(Q_{12}) & 0 & a_2.\cos(Q_{12}) + a_1.\cos(Q_1) \\ \sin(Q_{12}) & \cos(Q_{12}) & 0 & a_2.\sin(Q_{12}) + a_1.\sin(Q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Forward kinematics for a manipulator uses input joint angles to determine the end effector's Cartesian position and orientation. By looking at the matrix above, we see that it expresses end-effector position and orientation relative to the base link. From this knowledge, we can write the forward kinematic equations of the manipulator like below.

$$X = a_2.\cos(Q_{12}) + a_1.\cos(Q_1) \tag{2.7}$$

$$Y = a_2.\sin(Q_{12}) + a_1.\sin(Q_1) \tag{2.8}$$

2.2.2 Inverse Kinematic Analysis

Given the final position of the robot P= (Px, Py). Find Q_1, Q_2, for the scara robot. The final A matrix is given above:

To find Q_2, : if we square and sum Px and Py, we can get an expression in Q_2, :

$$P_x^2 + P_y^2 = (a_2.cos(Q_{12}) + a_1.cos(Q_1))^2 + (a_2.\sin(Q_{12}) + a_1.\sin(Q_1))^2$$

$$P_x^2 + P_y^2 = {a_1}^2 + {a_2}^2 + 2 * a_1 * a_2 * \cos(Q_2) \tag{2.9}$$

$$\cos(Q_2) = \frac{P_x^2 \mp P_{y-}^2 \; a_1\,2 - a_2\,2}{2 * \; a_1 * a_2}$$

$$\sin(Q_2) = \sqrt[2]{1 - \cos(Q_2)^2} \tag{2.10}$$

$$Q_2 = \mp cos^{-1}\left(\frac{P_x^2 + P_{y-}^2 \; a_1\,2 - a_2\,2}{2 * \; a_1 * a_2}\right) \tag{2.11}$$

This is really just the derivation of the law of cosines which we can also use to find Q_2

$${a_1}^2 + {a_2}^2 - 2 * a_1 * a_2 * \cos(180 - Q_2) = {P_x}^2 + {P_y}^2 \tag{2.12}$$

$$Cos(180 - Q_2) = \frac{P_x^2 + P_y^2 \; - \; a_1\,2 - a_2\,2}{-2 * \; a_1 * a_2}$$

$$-Cos(Q_2) = \frac{P_x^2+P_y^2 \ -a_1\,2-a_2\,2}{-2*a_1*a_2}$$

$$Cos(Q_2) = \frac{P_x^2+P_y^2 \ -a_1\,2-a_2\,2}{2*a_1*a_2} \tag{2.13}$$

To solve for Q_1 we solve for the following:

$$a_1.\cos(Q_1) + a_2.\cos(Q_{12}) = P_x \tag{2.14}$$

$$a_1.\sin(Q_1) + a_2.\sin(Q_{12}) = P_y \tag{2.15}$$

Two equations in two unknowns cos(Q_1),sin(Q_1),and Q_2 known from above.

$$\sin(Q_1) = \frac{a_2.\sin(Q_2) * P_x + (a_1 + a_2.\cos(Q_2)) * P_y}{(a_2.\sin(Q_2))^2 + (a_1 + a_2.\cos(Q_2))^2} \tag{2.16}$$

$$\cos(Q_1) = \frac{(a_1 + a_2.\cos(Q_2)) * P_{x-}a_2.\sin(Q_2) * P_y}{(a_2.\sin(Q_2))^2 + (a_1 + a_2.\cos(Q_2))^2} \tag{2.17}$$

$$(Q_1 = atan2((a_2.\sin(Q_2) * P_x + (a_1 + a_2.\cos(Q_2)) * P_y \ \ ,(a_1 + a_2.\cos(Q_2)) * P_{x-}a_2.\sin(Q_2) * P_y) \tag{2.18}$$

2.2.3 Compute System Jacobian and Singular Points

Jacobian is a robotics matrix that relates joint ($\dot{q}$) and end-effector ($\dot{X}$) velocities.

If the robot's joints move at specific velocity, we may wish to know the end effector's velocity. Jacobian helps here. Joint and end-effector velocities are related as follows:

$$\dot{X} = J \cdot \dot{q} \tag{2.19}$$

where $\dot{q}$ represents joint velocities and $\dot{X}$ represents end-effector velocity. Size of the $\dot{q}$matrix is nx1. 'n' is the number of active joints of the robot. Size of $\dot{X}$ matrix is mx1. 'm' is 3 for a planar manipulator. J is a pose-dependent Jacobian matrix. Size of jacobian matrix is mxn.

In general form jacobian and its elements for revolute joint can be written as follows,

$$J = [\ J_1 \quad J_2 \quad J_n\ ] \tag{2.20}$$

$$J_i = \begin{bmatrix} Z_{i-1} * {}^{i-1}P_n^{\,*} \\ Z_{i-1} \end{bmatrix} \tag{2.21}$$

Where,

Z_{i-1}=unit vector of i_{th} joint axis

${}^{i-1}_{n}P^*$=position vector from the origin of (i-1) th link to origin end-effector frame

$$Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\ ,\ Z_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

$${}^{1}_{2}P^{\,*} = \begin{bmatrix} a_2.\cos(Q_{12}) \\ a_2.\sin(Q_{12}) \\ 0 \end{bmatrix}\ ,\ {}^{0}_{2}P^{\,*} = \begin{bmatrix} a_1.\cos(Q_1) + a_2.\cos(Q_{12}) \\ a_1.\sin(Q_1) + a_2.\sin(Q_{12}) \\ 0 \end{bmatrix},$$

$$j = \begin{bmatrix} -(a_1.\sin(Q_1) + a_2.\sin(Q_{12})) & -a_2.\sin(Q_{12}) \\ a_1.\cos(Q_1) + a_2.\cos(Q_{12}) & a_2.\cos(Q_{12}) \end{bmatrix} \tag{2.22}$$

Let’s say jacobian matrix is simple form as follows,

$$j = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = J \cdot \begin{bmatrix} \dot{Q}_1 \\ \dot{Q}_2 \end{bmatrix}$$

To find the singularity points of the manipulator, the jacobian matrix is written. If the determinant of a matrix is equal to zero, that matrix cannot be inverted. The inverted Jacobian shown at Equation (2.23). A Jacobian matrix whose determinant is zero (ad − bc) in Equation (2.23) is said to be singular. The Jacobian matrix is singular at some values of the joint variables which shown in Figure 2.8. At the points where the Jacobian matrix is singular, the robot loses its degree of freedom and at these points where it loses, the matrix cannot be used in robot control. 2*2 jacobian matrix for Scara manipulator is written Equation (2.22). In below ad-bc is equal to a_1*a_2*sin(Q_2).

$$J(Q)^{-1} = \frac{1}{ad - bc} * \left(\begin{bmatrix} a2.\cos(Q_{12}) & a2.\sin(Q_{12}) \\ -(a_1.\cos(Q_1) + a_2.\cos(Q_{12})) & -(a_1.\sin(Q_1) + a_2.\sin(Q_{12})) \end{bmatrix} \right) \quad (2.23)$$

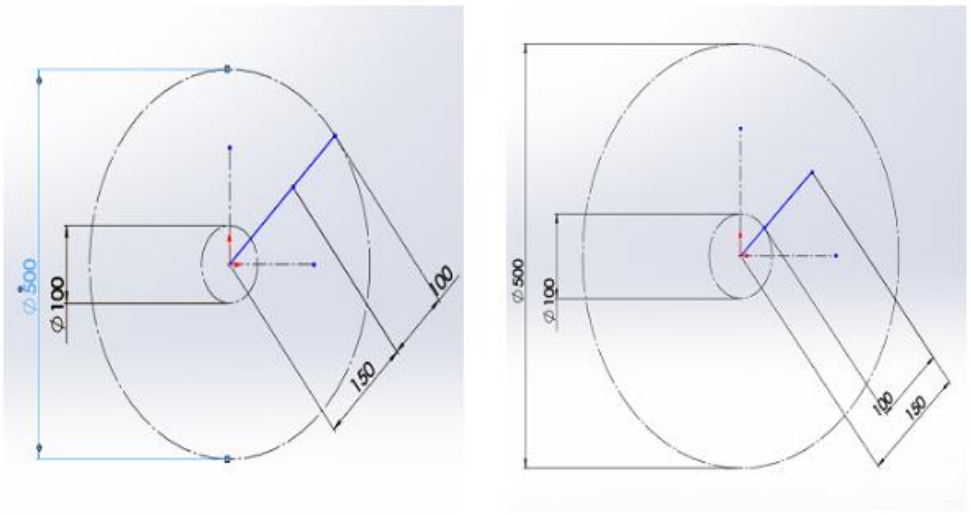


Figure 2.8: Two singularity point example

2.2.4 Workspace

Manipulator workspace is all reachable places. This depends on a number of factors and one of them link's length. In the workspace area analysis, the algorithm was developed in the python program in order to find the maximum workspace area excepting singular points. Workspace is found as shown in Figure 2.9 by help of Python code. Workspace is inside of the 250 mm diameter circle. The red dots in Figure 2.9 represent the singularity points.

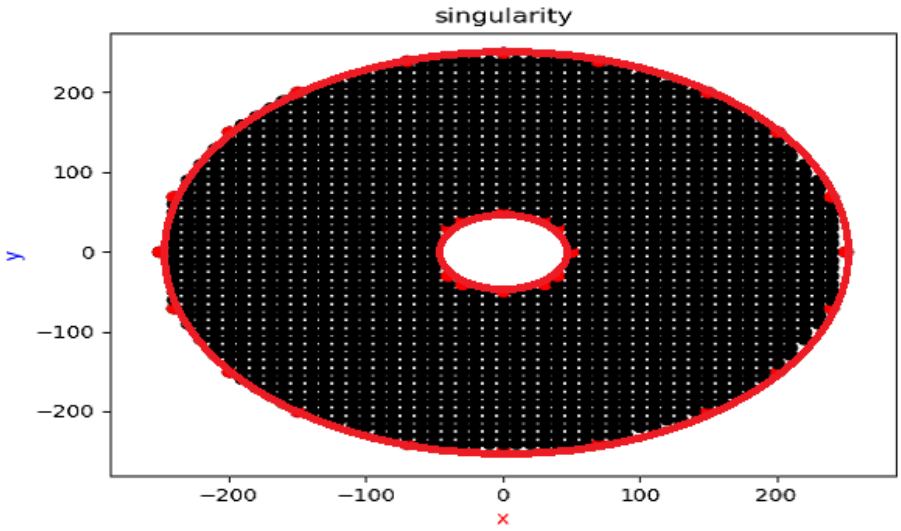


Figure 2.9: Scara workspace with singular points

However, this workspace analysis was performed as if the actuators were rotating 360 degrees before the actuators were taken. It was determined that there was a 240-degree limitation in the actuators when the actuators were taken. For this reason, 3/4 of the calculated workspace without renewing the workspace analysis was taken and shown in Figure 2.10.

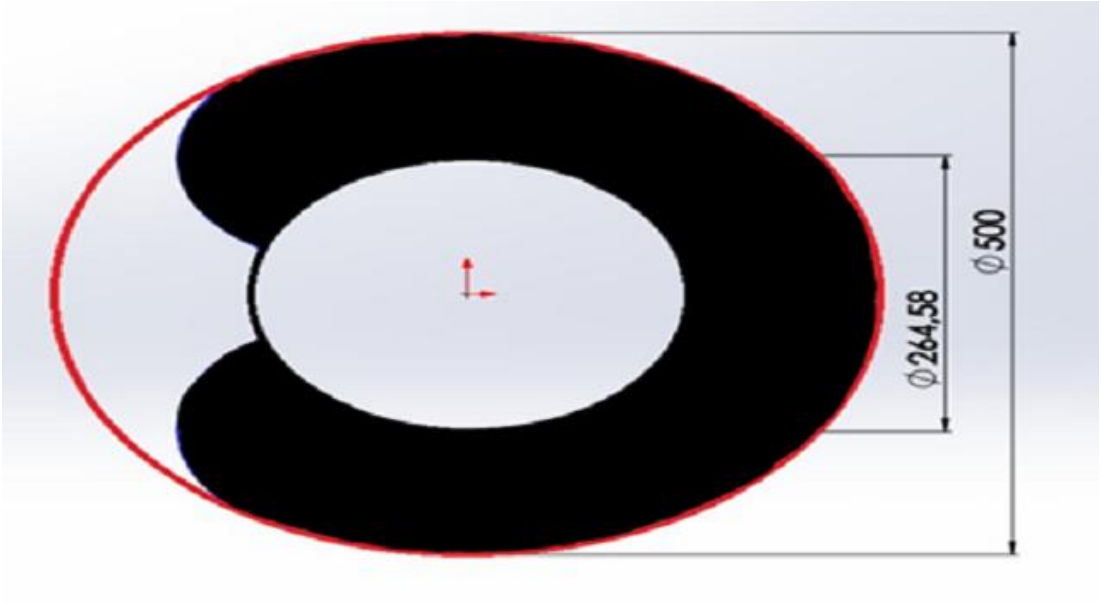


Figure 2.10: 3/4 Scara workspace

2.2.5 Dynamic Analysis

The development of a dynamical model is important in several ways. First, a dynamical model can be used for computer simulation of a robotic sysytem. Second, the dynamics analysis of a manipulator reveals all the joint reaction forces (and moments) needed for the design and sizing of links, bearings, and actuators. Third, it can be used for the development of suitable control strategies.

There are two types of dynamical problems: direct dynamics and inverse dynamics. Direct dynamic analysis is used for computer simulations and joint torque values are given as input and the motion of the end-effector is determined as output. In the inverse dynamic analysis, the desired trajectory is given as input to the end -effector and the torque values needed at the joints are found as output. There are two methods generally used for dynamic analysis in robotic systems. These are Lagrange's equations of motion and Newton-Euler formulations. In this thesis, calculations were made using the Recursive Newton-Euler method. Recursive Newton-Euler formulation consists of two steps as forward computation and backward computation. Forward calculation starting from link 1 to the end-effector, linear velocity, acceleration and angular velocity, accelerations of all links are calculated. In the backward computation, the forces and moments affecting the joints are calculated starting from the end-effector

up to the 1st link. The detailed explanation of all the notations used in dynamic analysis is included in the 7 subheadings (Recursive Newton-Euler Formulation) of the 9th chapter (Dynamics of Serial Manipulators) of the Lung-Wen Tsai Robot Analysis book [24].

$$ {}^{i}_{i}r = \begin{bmatrix} a_i \\ 0 \\ 0 \end{bmatrix} , \quad {}^{i}_{ci}r = \begin{bmatrix} \frac{-a_i}{2} \\ 0 \\ 0 \end{bmatrix} \quad \text{for i=1,2} \tag{2.24} $$

$$ {}^{1}_{1}r = \begin{bmatrix} a_1 \\ 0 \\ 0 \end{bmatrix} , \quad {}^{2}_{2}r = \begin{bmatrix} a_2 \\ 0 \\ 0 \end{bmatrix} , \quad {}^{1}_{c1}r = \begin{bmatrix} \frac{-a_1}{2} \\ 0 \\ 0 \end{bmatrix} , \quad {}^{2}_{c2}r = \begin{bmatrix} \frac{-a_2}{2} \\ 0 \\ 0 \end{bmatrix} \tag{2.25} $$

Links are homegenous with square cross-sectional area. Inertia matrix of two links about their center of mass.

$$ {}^{1}_{1}I = \frac{1}{12} * m1 * a_1 2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.26} $$

$$ {}^{2}_{2}I = \frac{1}{12} * m2 * a_2 2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27} $$

$$ {}^{0}g = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{2.28} $$

Acceleration of gravity is in $-z_0$ direction.

2.2.5.1 Forward Computation

i=1,2

For Link1 i=1

$$ {}^{\mathbf{0}}_{\mathbf{0}}\mathrm{w} = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad , \quad {}^{0}_{0}\dot{w} = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad , \quad {}^{\mathbf{0}}_{\mathbf{0}}\mathrm{V} = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad , \quad {}^{0}_{0}\dot{V} = \begin{bmatrix}0\\0\\0\end{bmatrix} \tag{2.29} $$

$$ {}^{\mathbf{1}}_{\mathbf{1}}\mathrm{w} = \begin{bmatrix}0\\0\\\dot{Q}_1\end{bmatrix} \quad , \quad {}^{1}_{1}\dot{w} = \begin{bmatrix}0\\0\\\ddot{Q}_1\end{bmatrix} \tag{2.30} $$

$$ {}^{\mathbf{1}}_{\mathbf{1}}\mathrm{V} = \begin{bmatrix}0\\a_1 * \dot{Q}_1\\0\end{bmatrix} \quad , \quad {}^{1}_{1}\dot{V} = \begin{bmatrix}-a_1 * \dot{Q}_1{}^2\\a_1 * \ddot{Q}_1\\0\end{bmatrix} \tag{2.31} $$

$$ {}^{1}_{c1}\dot{V} = \begin{bmatrix}\frac{-a_1}{2} * \dot{Q}_1{}^2\\\frac{a_1}{2} * \ddot{Q}_1\\0\end{bmatrix} \tag{2.32} $$

$$ {}^{1}g = \begin{bmatrix}0\\0\\-g\end{bmatrix} \tag{2.33} $$

For Link2 i=2

$$ {}^{\mathbf{2}}_{\mathbf{2}}\mathrm{w} = \begin{bmatrix}0\\0\\\dot{Q}_1 + \dot{Q}_2\end{bmatrix} \quad , \quad {}^{2}_{2}\dot{w} = \begin{bmatrix}0\\0\\\ddot{Q}_1 + \ddot{Q}_2\end{bmatrix} \tag{2.34} $$

$$ {}^{2}_{2}\dot{V} = \begin{bmatrix}a_1 * \left(\ddot{Q}_1 * \sin(Q_2) - \dot{Q}_1{}^2 * \cos(Q_2)\right) - a_2 * (\dot{Q}_1 + \dot{Q}_2)^2\\a_1 * \left(\ddot{Q}_1 * \cos(Q_2) + \dot{Q}_1{}^2 * \sin(Q_2)\right) + a_2 * (\ddot{Q}_1 + \ddot{Q}_2)\\0\end{bmatrix} $$

$$ {}^{2}_{2}\dot{V} = \begin{bmatrix}a_1 * \left(\ddot{Q}_1 * \sin(Q_2) - \dot{Q}_1{}^2 * \cos(Q_2)\right) - a_2 * (\dot{Q}_1 + \dot{Q}_2)^2\\a_1 * \left(\ddot{Q}_1 * \cos(Q_2) + \dot{Q}_1{}^2 * \sin(Q_2)\right) + a_2 * (\ddot{Q}_1 + \ddot{Q}_2)\\0\end{bmatrix} \tag{2.35} $$

$$ {}^{2}_{c2}\dot{V} = \begin{bmatrix} a_1 * \left(\ddot{Q}_1 * \sin(Q_2) - \dot{Q}_1{}^2 * \cos(Q_2)\right) - \frac{a_2}{2} * (\dot{Q}_1 + \dot{Q}_2)^2 \\ a_1 * \left(\ddot{Q}_1 * \cos(Q_2) + \dot{Q}_1{}^2 * \sin(Q_2)\right) + \frac{a_2}{2} * (\ddot{Q}_1 + \ddot{Q}_2) \\ 0 \end{bmatrix} \quad (2.36) $$

$$ {}^{2}g = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \quad (2.37) $$

2.2.5.2 Backward Computation

Start with i=2 end with Base frame

${}^{2}f_{2,3}$=0, ${}^{2}n_{2,3}$=0

Recursive form

$$ {}^{2}\mathrm{f}_{2,1} = \mathrm{m2} * \begin{bmatrix} a_1 * \left(\ddot{Q}_1 * \sin(Q_2) - \dot{Q}_1{}^2 * \cos(Q_2)\right) - \frac{a_2}{2} * (\dot{Q}_1 + \dot{Q}_2)^2 \\ a_1 * \left(\ddot{Q}_1 * \cos(Q_2) + \dot{Q}_1{}^2 * \sin(Q_2)\right) + \frac{a_2}{2} * (\ddot{Q}_1 + \ddot{Q}_2) \\ 0 \end{bmatrix} \quad (2.38) $$

$$ {}^{\mathbf{2}}\boldsymbol{n}_{\mathbf{2,1}} = \boldsymbol{m2} * \begin{bmatrix} 0 \\ 0 \\ \frac{1}{3} * a_2{}^2 * (\ddot{Q}_1 + \ddot{Q}_2) + \frac{1}{2} * a_1 * a_2 * (\ddot{Q}_1 + \cos(Q_2) + \dot{Q}_1{}^2 * \sin(Q_2)) \end{bmatrix} \quad (2.39) $$

$$ {}^{1}\mathrm{f}_{1,0} = \begin{bmatrix} m2 * [-a_1 * \dot{Q}_1{}^2 - \frac{1}{2} * a_2 * (\dot{Q}_1 + \dot{Q}_2)^2 * \cos(Q_2) - \frac{1}{2} * a_2 * (\ddot{Q}_1 + \ddot{Q}_2) * \sin(Q_2)] \\ +m1 * (-\frac{1}{2} * a_1 * \dot{Q}_1{}^2) \\ m2 * [a_1 * \ddot{Q}_1 - \frac{1}{2} * a_2 * (\dot{Q}_1 + \dot{Q}_2)^2 * \sin(Q_2) + \frac{1}{2} * a_2 * (\ddot{Q}_1 + \ddot{Q}_2) * \cos(Q_2)] \\ +m1 * (\frac{1}{2} * a_1 * \ddot{Q}_1) \\ 0 \end{bmatrix} \quad (2.40) $$

$$
{}^{1}\boldsymbol{n}_{1,0} = \begin{bmatrix} 0 \\ 0 \\ \left(\frac{1}{3} * m1 * {a_1}^2 + \frac{1}{3} * m2 * {a_2}^2 + m2 * {a_1}^2 + m2 * a_1 * a_2 * \cos(Q_2)\right) * \ddot{Q}_1 \\ \left(\frac{1}{3} * m2 * {a_2}^2 + \frac{1}{2} * m2 * a_1 * a_2 * \cos(Q_2)\right) * \ddot{Q}_2 - m2 * a_1 * a_2 * \sin(Q_2) * (\dot{Q}_1 * \dot{Q}_2 + \frac{1}{2} * {\dot{Q}_2}^2) \end{bmatrix} \quad (2.41)
$$

Joint torques computation:

$$
\begin{aligned}
\tau_1 = [\left(\frac{1}{3} * \mathrm{m1} + \mathrm{m2}\right) * {\mathrm{a_1}}^2 + \mathrm{m2} * \mathrm{a_1} * \mathrm{a_2} * \cos(\mathrm{Q_2}) + \frac{1}{3} * \mathrm{m2} * {\mathrm{a_2}}^2] \\
* \ddot{\mathrm{Q}}_1 + (\frac{1}{2} * \mathrm{m2} * \mathrm{a_1} * \mathrm{a_2} * \cos(\mathrm{Q_2}) + \frac{1}{3} * \mathrm{m2} * {\mathrm{a_2}}^2) \\
* \ddot{\mathrm{Q}}_2 - \mathrm{m2} * \mathrm{a_1} * \mathrm{a_2} * \sin(\mathrm{Q_2}) * \left(\dot{\mathrm{Q}}_1 * \dot{\mathrm{Q}}_2 + \frac{1}{2} * {\dot{\mathrm{Q}}_2}^2\right)
\end{aligned} \quad (2.42)
$$

$$
\begin{aligned}
\tau_2 = \left(\frac{1}{2} * m2 * a_1 * a_2 * \cos(Q_2) + \frac{1}{3} * m2 * {a_2}^2\right)\ddot{Q}_1 + \frac{1}{3} * m2 * {a_2}^2 \\
* \ddot{Q}_2 + \frac{1}{2} * m2 * a_1 * a_2 * \sin(Q_2) * {\dot{Q}_1}^2
\end{aligned} \quad (2.43)
$$

The torque of actuator1 and actuator2 is described in Equation (2.42) and Equation (2.43), respectively. The maximum actuator torques were found by using the trajectory analysis method in order to be able to select the actuator as a result of the theoretical torque calculation. The details of these theoretical calculations are examined in chapter 2, subtitle 2.4 (Trajectory Analysis of Scara). The masses of the link1 and link2 are measured using electronic scale. The result is m1= 150 gr and m2=175gr.Torque equations are solved by using 500 gr payload. Plots of the torque1 and torque2 are shown in Figure 2.11. Torque1 does not exceed the actuator torque that is given 35kg*cm (3.43 N*m) by motor manufacturer. In order to verify the torque values calculated according to the Newton-euler method, a torque simulation was performed from the Solidworks program. According to the simulation studies, it was seen that the theoretical calculations matched the graphics obtained from the simulation. These graphs are shown in Figure 2.11.

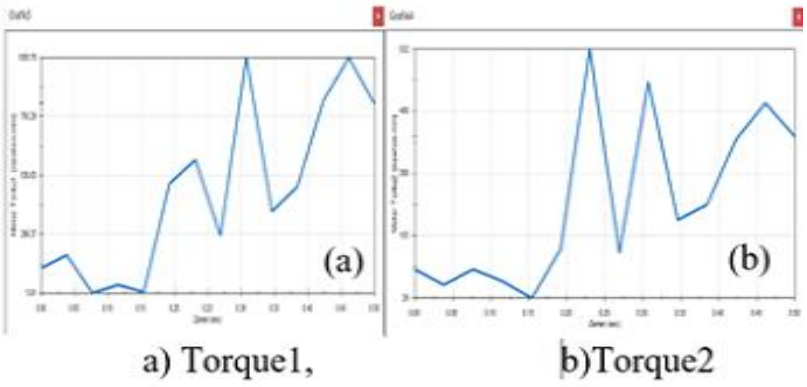


Figure 2.11: Plot of actuator torques

2.3 Kinematics Analysis of UR5

The business Universal Robots produced the UR5 robot. The Danish firm Universal Robots sells only four products. The UR3, UR5, UR10, and UR20. The sizes of the robots are increasing in order of name. Specifications provided by Universal Robots are shown in Figure 2.12.

6 axis robot arm with a working radius 850 mm	
Weight	18.4 Kg
Payload	5 Kg
Reach	850 mm
Joint ranges	$\pm 360^0$ on all joints
Speed	Joint max: 180^0/sec Tool: 1m/sec
Repeatability	± 0.1 mm / ± 0.0039 *in*
Footprint	Ø149 mm
Degrees of freedom	6 revolute joint
Control box size (W x H x D)	475 mm x 423 mm x 268 mm
Communication	TCP/IP, Ethernet socket & Modbus TCP
Programming	Polyscope gui on 12 inch touchscreen
Materials	Aluminium, ABS plastic
Power Supply	100-240 VAC, 50-6 Hz
Calculated operating life	35,000 Hours

Figure 2.12: Specifications of the UR5 [25]

"*In Figure 2.13 the UR5 is drawn with the dimensions of all the links. The robotic arm consists of six revolute joints. For the remainder of the report these joints will be referred to as Base, Shoulder, Elbow, Wrist1, Wrist2 and Wrist3. The names of these joints are also stated in Figure 2.13. The*

UR5 has a rather standard layout for these types of robotic arms. The Shoulder and Elbow joint are rotating perpendicular to the Base joint. These three joints are connected with long links. In this way the robot has a long reach. The Wrist joints are mainly to manoeuvre the Tool Center Point (TCP) in the right orientation. Other robotic arms with similar layouts often use a spherical joint instead of three revolute joints. Those spherical joints are less prone to self-collision as the solution of the UR5. [25] ”

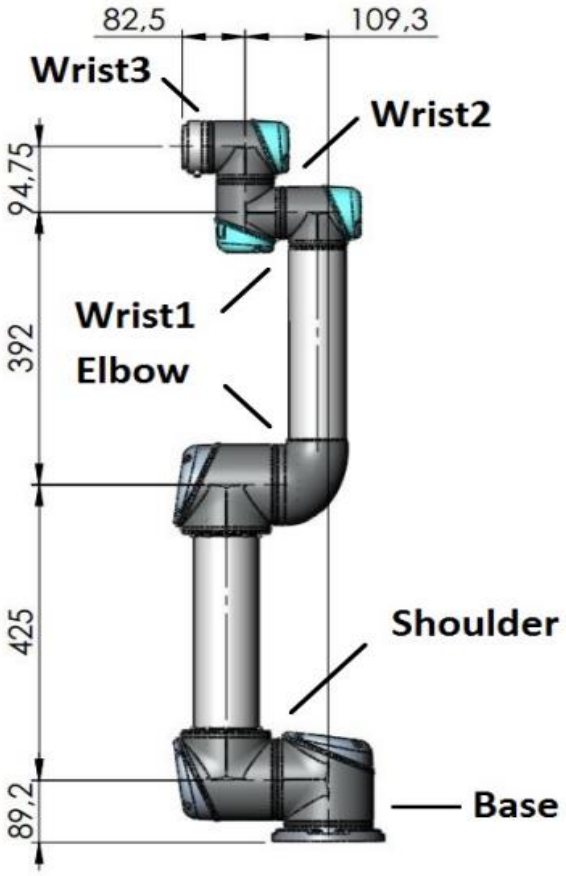


Figure 2.13: Dimensions of the UR5 [25]

This section refers to the Denavit-Hartenberg notation used by research [26] , sometimes known as modified DH-parameters. Moreover, the following brief notations are utilized:

$${}^{0}P_{6} = \begin{bmatrix} {}^{0}P_{6x} \\ {}^{0}P_{6y} \\ {}^{0}P_{6z} \end{bmatrix}$$ frame 6 to frame 0.

${}^0\hat{Y}_6 = \begin{bmatrix} {}^0\hat{Y}_{6x} \\ {}^0\hat{Y}_{6y} \\ {}^0\hat{Y}_{6z} \end{bmatrix}$ is a unit vector representing the direction of the y-axis in frame 6 as seen from? frame 0.

${}^0_6A$ *is a transformation from fame 6 to frame 0, meaning that* ${}^0P = {}^0_6A * {}^6P$

2.3.1 Forward Kinematics of the UR5

Based on the known joint angles θ_{1-6}, the forward kinematic (FK) equations compute the transformation matrix ${}^0_6A$. The transformation matrix is specified as follows:

$${}^0_6A(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = \begin{bmatrix} {}^0_6R & {}^0P_6 \\ 0 & 1 \end{bmatrix} \Rightarrow \quad {}^0_6A = \begin{bmatrix} {}^0\hat{X}_6 & {}^0\hat{Y}_6 & {}^0\hat{Z}_6 & {}^0P_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} {}^0\hat{X}_{6x} & {}^0\hat{Y}_{6x} & {}^0\hat{Y}_{6x} & {}^0P_{6x} \\ {}^0\hat{X}_{6y} & {}^0\hat{Y}_{6y} & {}^0\hat{Y}_{6y} & {}^0P_{6y} \\ {}^0\hat{X}_{6z} & {}^0\hat{Y}_{6z} & {}^0\hat{Y}_{6z} & {}^0P_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.44}$$

${}^0\hat{X}_6$, ${}^0\hat{Y}_6$, and ${}^0\hat{Z}_6$ are unit vectors that define the axes of frame 6 in reference to frame 0. The transformation matrix can be divided into a series of transformations, one for each joint:

$${}^0_6A(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = {}^0_1A(\theta_1) * {}^1_2A(\theta_2) * {}^2_3A(\theta_3) * {}^3_4A(\theta_4) * {}^4_5A(\theta_5) * {}^5_6A(\theta_6)$$

$$= \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.45}$$

Figure 2.14 shows the kinematic structure of the UR5 robot in the zero position:

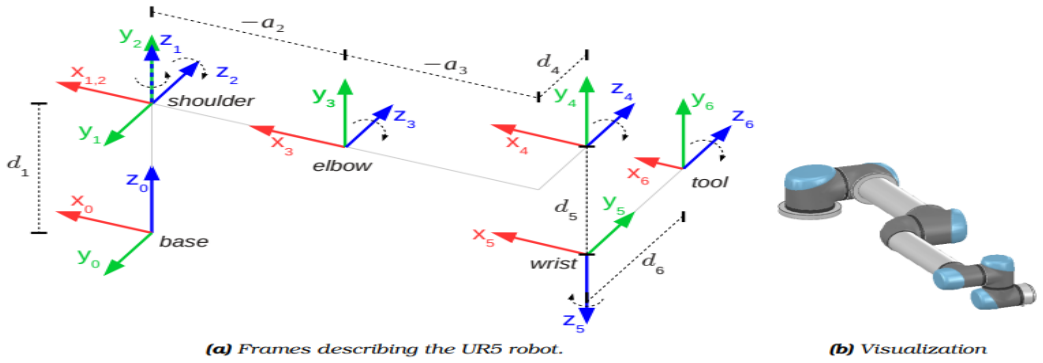


Figure 2.14: The UR5 robot in zero position [27]

The DH-parameters are according to [26] specified as:

- ✓ a_i = distance from Z_i to Z_{i+1}measured along X_i
- ✓ $\alpha_i = angle\ from\ Z_i\ to\ Z_{i+1}\ measured\ about\ X_i$
- ✓ $d_i = distance\ from\ X_{i-1}\ to\ X_i\ measured\ along\ Z_i$
- ✓ $\theta_i = angle\ from\ X_{i-1}\ to\ X_i\ measured\ about\ Z_i$

Table 2.2: Modified Denavit-Harteberg parameters (DH-parameters) of a UR5 robot, correspond ing to the frames in Figure 2.14. The θ_i parameters are variable, whereas the other parameters are constant.

AXES	$\boldsymbol{\alpha_{i-1}}$	$\boldsymbol{a_{i-1}}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	0	0	d_1	θ_1
2	α_1=90°	0	0	θ_2
3	0	a_2	0	θ_3
4	0	a_3	d_4	θ_4
5	α_4=90°	0	d_5	θ_5
6	α_5=-90°	0	d_6	θ_6

Each link's transformations can be written utilizing the DH-parameters. The transformation between link i − 1 and i in general.

$$ {}^{i-1}_{i}A = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i * \cos(\alpha_{i-1}) & \cos\theta_i * cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -sin(\alpha_{i-1}) * d_i \\ sin\,\theta_i * sin(\alpha_{i-1}) & cos\,\theta_i * sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1}) * d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.46) $$

The joints are occasionally referred to informally as shoulder pan (θ_1), shoulder lift (θ_2), elbow (θ_3), wrist 1 (θ_4), wrist 2 (θ_5), and wrist 3 (θ_6). For the remainder of the document, we use the short-hand $c_i = cos\theta_i$, $s_i = sin\theta_i$, and for angle sums, $c_{ij} = \cos(\theta_i + \theta_j)$.

$n_x = c_6(s_1s_5 + ((c_1c_{234} - s_1 s_{234}) c_5)/2.0 + ((c_1 c_{234} + s_1 s_{234}) c_5)/2.0) - (s_6((s_1 c_{234} + c_1 s_{234}) - (s_1 c_{234} - c_1 s_{234})))/2.0$

$n_y = c_6 (((s_1 c_{234} + c_1 s_{234}) c_5)/2.0 - c_1 s_5 + ((s_1 c_{234} - c_1 s_{234}) c_5)/2.0) + s_6 ((c_1 c_{234} - s_1 s_{234})/2.0 - (c_1 c_{234} + s_1 s_{234})/2.0)$

$n_z = (s_{234} c_6 + c_{234} s_6)/2.0 + s_{234} c_5 c_6 - (s_{234} c_6 - c_{234} s_6)/2.0$

$o_x = -(c_6 ((s_1 c_{234} + c_1 s_{234}) - (s_1 c_{234} - c_1 s_{234})))/2.0 - s_6 (s_1 s_5 + ((c_1 c_{234} - s_1 s_{234}) c_5)/2.0 + ((c_1 c_{234} + s_1 s_{234}) c_5)/2.0)$

$o_y = c_6 ((c_1 c_{234} - s_1 s_{234})/2.0 - (c_1 c_{234} + s_1 s_{234})/2.0) - s_6 (((s_1 c_{234} + c_1 s_{234}) c_5)/2.0 - c_1 s_5 + ((s_1 c_{234} - c_1 s_{234}) c_5)/2.0)$

$o_z = (c_{234} c_6 + s_{234} s_6)/2.0 + (c_{234} c_6 - s_{234} s_6)/2.0 - s_{234} c_5 s_6$

$a_x = c_5 s_1 - ((c_1 c_{234} - s_1 s_{234}) s_5)/2.0 - ((c_1 c_{234} + s_1 s_{234}) s_5)/2.0$

$a_y = - c_1 c_5 - ((s_1 c_{234} + c_1 s_{234}) s_5)/2.0 + ((c_1 s_{234} - s_1 c_{234}) s_5)/2.0$

$a_z = (c_{234} c_5 - s_{234} s_5)/2.0 - (c_{234} c_5 + s_{234} s_5)/2.0$

$p_x =$

$-(d_5(s_1 c_{234} - c_1 s_{234}))/2.0+(d_5 (s_1 c_{234}+ c_1 s_{234}))/2.0+d_4 s_1-(d_6(c_1 c_{234} - s_1 s_{234}) s_5)/2.0-(d_6 (c_1 c_{234}+ s_1 s_{234}) s_5)/2.0 + a_2 c_1c_2 + d_6 c_5 s_1 + a_3 c_1 c_2 c_3 - a_3 c_1 s_2 s_3)$

$P_y = -(d_5 (c_1 c_{234} - s_1 s_{234}))/2.0+(d_5 (c_1 c_{234}+ s_1 s_{234}))/2.0-d_4 c_1-(d_6 (s_1 c_{234}+ c_1 s_{234}) s_5)/2.0-(d_6 (s_1 c_{234} - c_1 s_{234}) s_5)/2.0 - d_6 c_1 c_5 + a_2s_1 + a_3 c_2 c_3 s_1 - a_3 s_1 s_2 s_3)$

$P_z = d_1 + (d_6 (c_{234} c_5 - s_{234} s_5))/2.0 + a_3(s_2 c_3 + c_2 s_3) + a_2 s_2 - (d_6 (c_{234} c_5 + s_{234} s_5))/2.0 - d_5 c_{234}$

(2.47)

2.3.2 Inverse Kinematics of the UR5

- Finding θ_1

To find θ_1, we first determine the location of frame 5 (the wrist frame) in relation to the base frame; ${}^0P_5$. As shown in Figure 2.11, ${}^0P_5$ can be determined by translating from frame 6 to frame 5 along z_6 in backwards. Remember that ${}^0_6A$ and d_6 are both well-known.

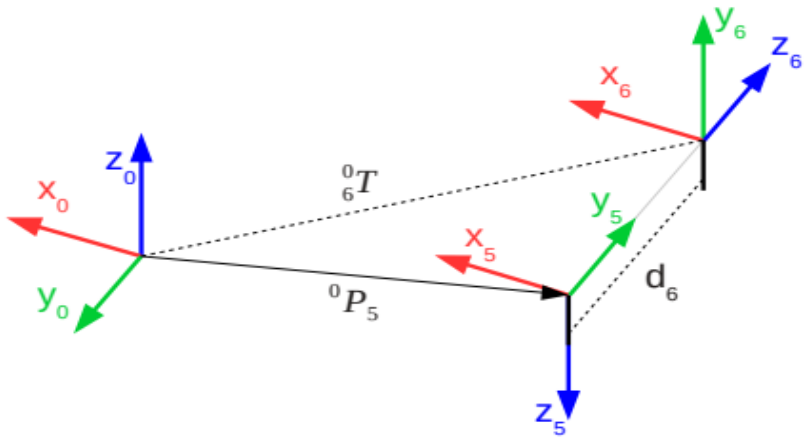


Figure 2.15: Finding the origin of frame5 [27]

The translation ${}^0P_5$ can be written as:

$$ {}^0P_5 = {}^0P_6 - d_6 * {}^0\hat{Z}_6 $$

$$ {}^0P_5 = {}^0_6A * \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \quad (2.48) $$

To determine θ_1, we inspect the robot indicated in Figure 2.16 from above (looking down into z_0):

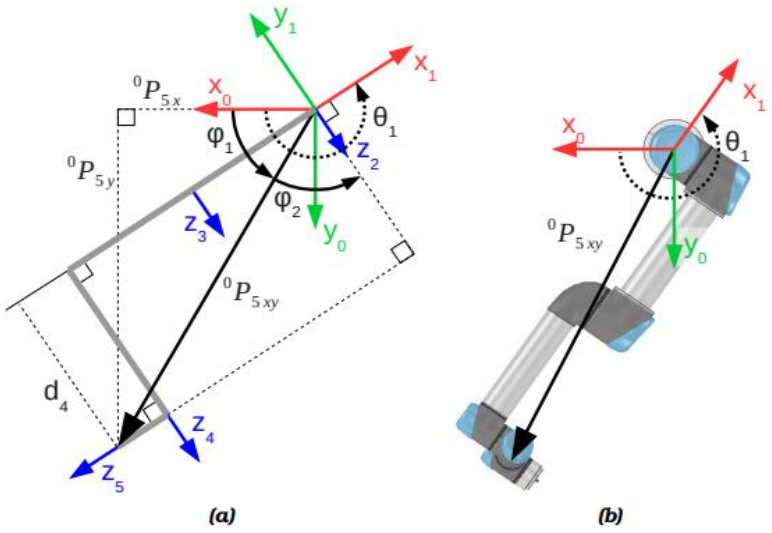


Figure 2.16: Robot (until frame5) seen from above. The robot is shown as gray lines. Note that z_5 should actually point into the page if all angles are 0. Here $\theta_4 \neq 0$ in order to better illustrate how θ_1 can be isolated[27].

Our method for determining θ_1 involves considering the wrist, P5, in frames 0 and 1, respectively. Intuitively, frame 0-to-1 rotation, θ_1, should equal the difference between frames 0-to-5 and 1-to-5 rotations. The results of formalizing and inserting symbols from Figure 2.16 are:

$$V_{0\to1}=V_{0\to5}-V_{1\to5}\leftrightarrow$$

$$V_{0\to1}=V_{0\to5}+V_{1\to5}\leftrightarrow$$

$$\theta_1 = \emptyset_1 + \left(\emptyset_{2+}\frac{\pi}{2}\right) \tag{2.49}$$

Examining the triangle with sides ${}^0P_{5x}$ and ${}^0P_{5y}$ allows us to determine the angle $\emptyset_1$:

$$\emptyset_1 = a\tan 2\left({}^0P_{5y}, {}^0P_{5x}\right) \tag{2.50}$$

Examining the rightmost triangle with $\emptyset_2$ as one of the angles reveals the angle $\emptyset_2$ .Two of the sides have lengths $|{}^0P_{5xy}|$ and d_4

$$\cos \emptyset_2 = \frac{d_4}{\left|{}^0P_{5xy}\right|}$$

$$\emptyset_2 = \pm a\cos\left(\frac{d_4}{\left|{}^0P_{5xy}\right|}\right) \leftrightarrow \emptyset_2 = \pm a\,cos\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) \qquad (2.51)$$

The desired angle θ_1 can now be found simply as:

$$\theta_1 = \emptyset_1 + \emptyset_1 + \frac{\pi}{2}$$

$$\theta_1 = a\,tan\,2\left({}^0P_{5y}, {}^0P_{5x}\right) \pm \text{acos}\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) + \frac{\pi}{2} \qquad (2.52)$$

The two options correlates to "left" or "right" shoulder placement.

- Finding θ_5

Figure 2.17 shows the robot from above again; this time with frame 6 included.

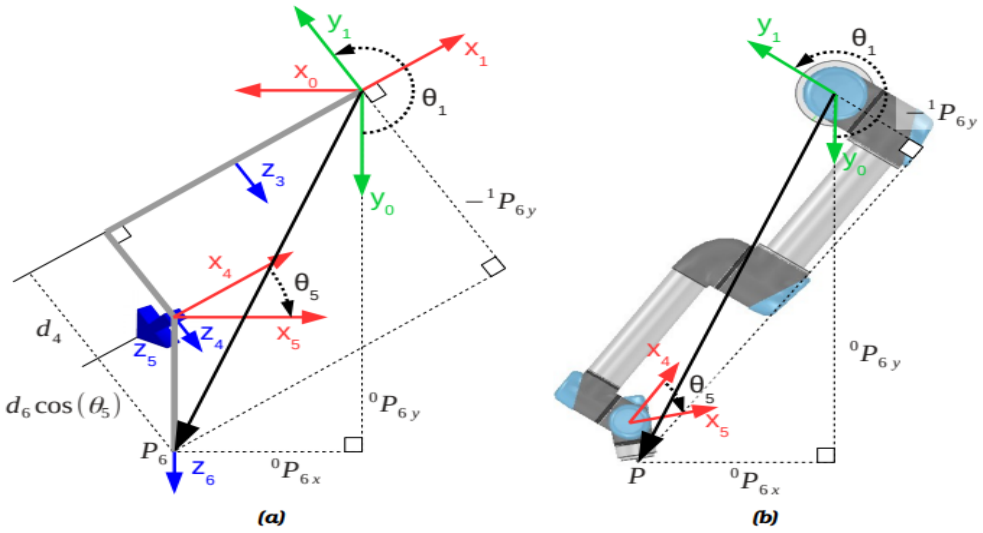


Figure 2.17: Robot (including frame 6) seen from above [27]

Our approach to finding θ_5 is to notice that ${}^1P_{6y}$ (the y-component of ${}^1P_6$) only depends on θ_5. In the figure, we can trace y_1 backwards to see that ${}^1P_{6y}$ is given by:

$$-{}^1P_{6y} = d_4 + d_6 \cos\theta_5 \tag{2.53}$$

The component ${}^1P_{6y}$ can also be expressed by looking at ${}^1P_6$ as a rotation of ${}^0P_6$ around z_1.

$${}^0P_6 = {}^0_1R * {}^1P_6$$

$${}^1P_6 = {}^0_1R^T * {}^0P_6$$

$$\begin{bmatrix} {}^1P_{6x} \\ {}^1P_{6y} \\ {}^1P_{6z} \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}^T * \begin{bmatrix} {}^0P_{6x} \\ {}^0P_{6y} \\ {}^0P_{6z} \end{bmatrix}$$

$$\begin{bmatrix} {}^1P_{6x} \\ {}^1P_{6y} \\ {}^1P_{6z} \end{bmatrix} = \begin{bmatrix} cos(\theta_1) & sin(\theta_1) & 0 \\ -sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} {}^0P_{6x} \\ {}^0P_{6y} \\ {}^0P_{6z} \end{bmatrix}$$

$${}^1P_{6y} = {}^0P_{6x} * (-sin(\theta_1)) + {}^0P_{6y} * cos(\theta_1) \tag{2.54}$$

By combining Equation (53) and (54), we can eliminate ${}^{1}P_{6y}$ and express θ_5 only using known values:

$$-d_4 - d_6 * cos\,\theta_5 = {}^{0}P_{6x} * (-sin(\theta_1)) + {}^{0}P_{6y} * cos(\theta_1)$$

$$\cos\theta_5 = \frac{{}^{0}P_{6x} * (sin\,\theta_1) - {}^{0}P_{6y} * cos(\theta_1) - d_4}{d_6}$$

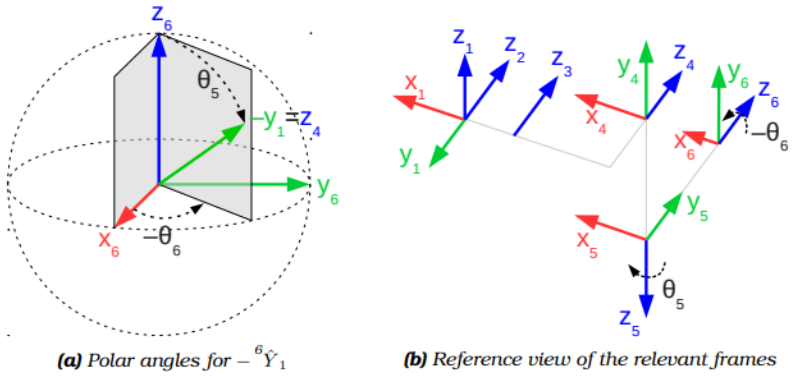
$$\theta_5 = \pm a\,cos\left(\frac{{}^{0}P_{6x} * (sin\,\theta_1) - {}^{0}P_{6y} * cos(\theta_1) - d_4}{d_6}\right) \qquad (2.55)$$

There are two solutions. These correlate, respectively, to the wrist being "up" or "down." This can be intuitively understood: The joint sum ($\theta_2 + \theta_3 + \theta_4$) can cause the end-effector to be located in the same position, but with the wrist flipped. The orientation can then be "corrected" by θ_6.

Also note that a solution is defined as long as the value inside acos has a magnitude not greater than 1; equivalent to $|\,{}^{1}P_{6y} - d_4| \leq |d_6|$

- Finding θ_6

To determine θ_6 we examine y1 seen from frame 6; ${}^{6}Y_1$. This axis will (ignoring translations) always be parallel to ${}^{6}Z_{2,3,4}$ as can be seen from Figure 2.18 (b). Therefore, it will only depend on θ_5 and θ_6. It turns out that $-\,{}^{6}Y_1$ can in fact be described using spherical coordinates, where azimuth is $-\theta_6$ and the polar angle is θ_5; see Figure 2.18(a).



(a) *Polar angles for* $-{}^{6}\hat{Y}_1$ **(b)** *Reference view of the relevant frames*

Figure 2.18: The axis -${}^{6}Y_1$ expressed in spherical coordinates with azimuth -θ_6 and polar angle θ_5. For simplicity ${}^{6}Y_1$ is denoted y_1 in the Figure [27].

Converting $-{}^{6}Y_1$ from spherical to Cartesian coordinates gives:

$$ {}^{6}\hat{Y}_1 = \begin{bmatrix} \sin\theta_5 * \cos(-\theta_6) \\ sin\,\theta_5 * \sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix} $$

$$ {}^{6}\hat{Y}_1 = \begin{bmatrix} -sin\,\theta_5 * cos(\theta_6) \\ sin\,\theta_5 * sin(\theta_6) \\ -\cos\theta_5 \end{bmatrix} \tag{2.56} $$

In Equation (56) we could isolate θ_6 and have an expression of θ_6 in relation to ${}^{6}_{1}A$. We want an expression of θ_6 all the way from ${}^{6}_{0}A$. To get this, we identify that ${}^{6}Y_1$ is given as a rotation of θ_1 in the x, y-plane of frame 0 (very similar to Equation (54)):

$$ {}^{6}\hat{Y}_1 = {}^{6}\hat{X}_0 * -sin(\theta_1) + {}^{6}\hat{Y}_0 * cos(\theta_1) $$

$$ {}^{6}\hat{Y}_1 = \begin{bmatrix} -{}^{6}\hat{X}_{0x} * sin(\theta_1) + {}^{6}\hat{Y}_{0x} * cos(\theta_1) \\ -{}^{6}\hat{X}_{0y} * sin(\theta_1) + {}^{6}\hat{Y}_{0y} * cos(\theta_1) \\ -{}^{6}\hat{X}_{0z} * sin(\theta_1) + {}^{6}\hat{Y}_{0z} * cos(\theta_1) \end{bmatrix} \tag{2.57} $$

Equating the first two entries of (56) and (57) give:

$$-sin\,\theta_5 * cos(\theta_6) = -{}^6\hat{X}_{0x} * sin(\theta_1) + {}^6\hat{Y}_{0x} * cos(\theta_1)$$

$$-sin\,\theta_5 * sin(\theta_6) = -{}^6\hat{X}_{0y} * sin(\theta_1) + {}^6\hat{Y}_{0y} * cos(\theta_1) \tag{2.58}$$

$$\begin{Bmatrix} cos(\theta_6) = \dfrac{{}^6\hat{X}_{0x} * sin(\theta_1) - {}^6\hat{Y}_{0x} * cos(\theta_1)}{\sin\theta_5} \\ sin(\theta_6) = \dfrac{-{}^6\hat{X}_{0y} * sin(\theta_1) + {}^6\hat{Y}_{0y} * cos(\theta_1)}{sin\,\theta_5} \end{Bmatrix}$$

$$\theta_6 = a\tan 2\left(\frac{-{}^6\hat{X}_{0y} * sin(\theta_1) + {}^6\hat{Y}_{0y} * cos(\theta_1)}{sin\,\theta_5}, \frac{{}^6\hat{X}_{0x} * sin(\theta_1) - {}^6\hat{Y}_{0x} * cos(\theta_1)}{sin\,\theta_5}\right) \tag{2.59}$$

This solution is undetermined if the denominator sin θ_5 = 0. As seen in Figure 2.18(b), the joint axes 2, 3, 4, and 6 are aligned in this instance. This is an excessive number of degrees of freedom. The axes 2, 3, and 4 can independently spin the end-effector (frame 6) around the z_6 axis without moving it, making the sixth joint superfluous. In this circumstance, θ_6 can be set to any value.

Similarly, if both numerators in Equation (2.59) are 0, the solution is uncertain. If this is the case, then sin θ_5 must likewise equal zero, and the situation remains unchanged. This is seen when considering both sides of Equation (2.58).

- Finding θ_3

The remaining three joints are examined (2, 3, and 4). Observe that all of their joint axes are parallel. As depicted in Figure 2.19, their combination constitutes a planar 3R-manipulator.

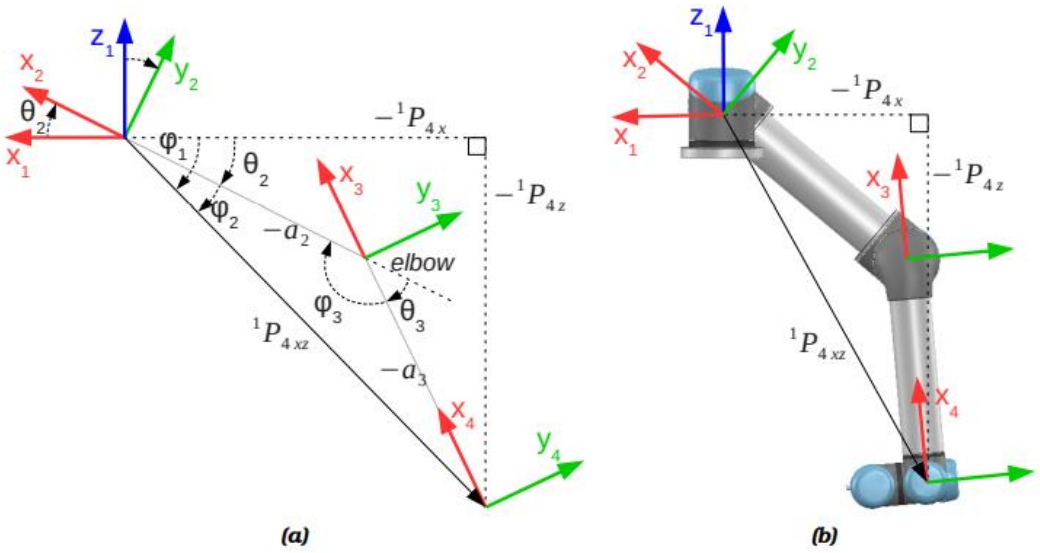


Figure 2.19: Joints 2, 3, and 4 comprise a 3R planar manipulator [27].

Because ${}^0_1A$, ${}^4_5A$, and ${}^5_6A$ are known at this time, we may limit our examination to ${}^1_4A$ (frame 4 relative to frame 1) This transformation is depicted in frame 1's x, z plane in Figure 2.19. (a). It is evident from the graph that the length of the translation $|{}^1P_{4xz}|$ is solely dictated by θ_3, or similarly $\emptyset_3$. The angle $\emptyset_3$ can be determined using the cosine law:

$$\cos\emptyset_3 = \frac{(-a_2)^2 + (-a_3)^2 - |{}^1P_{4xz}|^2}{2 * (-a_2) * (-a_3)} = \frac{{a_2}^2 + {a_3}^2 - |{}^1P_{4xz}|^2}{2 * a_2 * a_3} \qquad (2.60)$$

The relationship between $\cos\emptyset_3$ and $\cos\theta_3$ is:

$$\cos\theta_3 = \cos(\pi - \emptyset_3) = -\cos(\emptyset_3) \qquad (2.61)$$

Combining (Equation 2.60) and (Equation 2.61) give:

$$cos(\theta_3) = \frac{{a_2}^2 + {a_3}^2 - |{}^1P_{4xz}|^2}{2 * a_2 * a_3}$$

$$\theta_3 = \pm cos\left(\frac{|^1 P_{4xz}|^2 - {a_2}^2 - {a_3}^2}{2 * a_2 * a_3}\right) \tag{2.62}$$

Note that solutions exist for θ_3 if the argument of acos is within [−1; 1]. This may be demonstrated to be equivalent to$|\ ^1P_{4xz}| \in [|a_2 - a_3|; |a_2 + a_3|]$. In the majority of instances where solutions exist, two distinct solutions exist. These phrases are equivalent to "elbow up" and "elbow down."

- Finding θ_2

The angle θ_2 can be found as $\emptyset_1 - \emptyset_2$. Each of the following can be determined by examining Figure 2.19(a) and utilizing the atan2 and sine relations:

$$\emptyset_1 = a\ tan\ 2(-^1P_{4z}, -\ ^1P_{4x}) \tag{2.63}$$

$$\frac{\text{Sin}\ \emptyset_2}{-a_3} = \frac{\sin \emptyset_3}{|^1 P_{4xz}|}$$

$$\emptyset_2 = a\ \sin\left(\frac{-a_{3*sin\,\emptyset_3}}{|^1 P_{4xz}|}\right) \tag{2.64}$$

We can replace $\emptyset_3$ with θ_3 by noticing that sin $\emptyset_3$ = sin $(180^\circ - \theta_3)$ = sin θ_3. Combining the equations now give:

$$\theta_2 = \emptyset_1 - \emptyset_2 = a\tan 2(-^1P_{4z}, -^1P_{4x}) - a\sin\left(\frac{-a_3 * \sin\theta_3}{|^1 P_{4xz}|}\right) \tag{2.65}$$

- Finding θ_4

The last remaining angle θ_4 is defined as the angle from X_3 to X_4 measured about Z_4. It can thus easily be derived from the last remaining transformation matrix, ${}^{3}_{4}A$, using its first column ${}^{3}X_4$:

$$\theta_4 = a\tan 2\left({}^{3}\hat{X}_{4y}, {}^{3}\hat{X}_{4x}\right) \tag{2.66}$$

To sum up, a total of 8 solutions exists in general for the general inverse kinematic problem of the UR5: $2\theta_1 \times 2\theta_5 \times 1\theta_6 \times 2\theta_3 \times 1\theta_2 \times 1\theta_4$.

2.4 Trajectory Analysis of Scara

A Robot arm is asked to complete its movement from one point to another without any jerks, in such a way that all its joints complete its task at the same time and do not collide with the objects around it. For this purpose, trajectory planning is done for that robot arm. If trajectory planning is not done, we cannot know what kind of movement the robot will reach when we place the target position on the robot arm. Therefore, while performing this movement, the robot arm may hit the ground, an object around it, or itself. This will damage the robot arm. For this reason, orbital planning is done in cartesian space or joint space [28]. In the system developed in this study, trajectory planning in joint space is carried out using third-order polynomials with 2 segment method which shown Figure 2.20. (Equation 2.67).

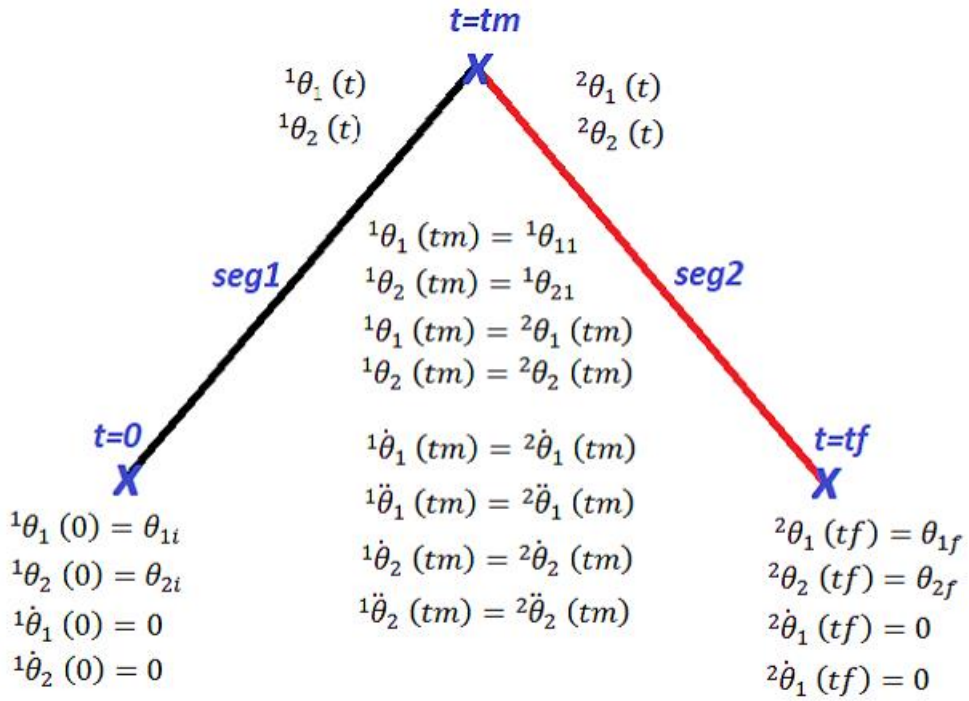


Figure 2.20: Two segment trajectory analysis

$$^1\theta_1(t) = a_1t^3 + b_1t^2 + c_1t + d_1$$

$$^1\theta_2(t) = a_2t^3 + b_2t^2 + c_2t + d_2$$

$$^2\theta_1(t) = a_3t^3 + b_3t^2 + c_3t + d_3$$

$$^2\theta_2(t) = a_4t^3 + b_4t^2 + c_4t + d_4$$

(2.67)

In order to plan trajectory in joint space, axis angle information must be found from inverse kinematics equations by using the given coordinate information. Trajectory planning can be performed using the determined inverse kinematic values. According to this;

The speed values at the start and end positions of the robot arm are zero (because it becomes stationary after a certain distance from the stationary state) and also, we use 2 segment method to solve trajectory. So, the middle point we will determined. In order

to plan the trajectory with a third-order polynomial, first the following six values (Equation 2.68-2.75) are determined.

$$^{1}\theta_1\,(0) = \theta_{1i} \tag{2.68}$$

$$^{2}\theta_1\,(tf) = \theta_{1f} \tag{2.69}$$

$$^{1}\theta_1\,(tm) = {}^{1}\theta_{11} \tag{2.70}$$

$$^{1}\theta_1\,(tm) = {}^{2}\theta_1\,(tm) \tag{2.71}$$

$$^{1}\dot{\theta}_1\,(tm) = {}^{2}\dot{\theta}_1\,(tm) \tag{2.72}$$

$$^{1}\ddot{\theta}_1\,(tm) = {}^{2}\ddot{\theta}_1\,(tm) \tag{2.73}$$

$$^{1}\dot{\theta}_1\,(0) = 0 \tag{2.74}$$

$$^{2}\dot{\theta}_1\,(tf) = 0 \tag{2.75}$$

Here θ_{1i} is the starting position, θ_{1f} is the target position, $^{1}\dot{\theta}_1\,(0)$ is the starting velocity, $^{2}\dot{\theta}_1\,(tf)$ is the ending velocity, $^{1}\theta_1\,(tm)$ middle point first segment first joint angle, $^{2}\theta_1\,(tm)$ middle point second segment first joint angle. Then, these values are written in the position, velocity, and acceleration equations in Equations (2.76-2.79) and position-time, velocity-time, acceleration-time values are obtained. The same procedure applies to θ_2.

$$^{1}\theta_1(t) = a_1t^3 + b_1t^2 + c_1t + d_1$$

$$^{1}\dot{\theta}_1(t) = 3 * a_1t^2 + 2 * b_1t + c_1$$

$$^{1}\ddot{\theta}_1(t) = 6 * a_1t + 2 * b_1 \quad (2.76)$$

$$^{1}\theta_2(t) = a_2t^3 + b_2t^2 + c_2t + d_2$$

$$^{1}\dot{\theta}_2(t) = 3 * a_2t^2 + 2 * b_2t + c_2$$

$$^{1}\ddot{\theta}_2(t) = 6 * a_2t + 2 * b_2 \quad (2.77)$$

$$^{2}\theta_1(t) = a_3t^3 + b_3t^2 + c_3t + d_3$$

$$^{2}\dot{\theta}_1(t) = 3 * a_3t^2 + 2 * b_3t + c_3$$

$$^{2}\ddot{\theta}_1(t) = 6 * a_3t + 2 * b_3 \quad (2.78)$$

$$^{2}\theta_2(t) = a_4t^3 + b_4t^2 + c_4t + d_4$$

$$^{2}\dot{\theta}_2(t) = 3 * a_4t^2 + 2 * b_4t + c_4$$

$$^{2}\ddot{\theta}_2(t) = 6 * a_4t + 2 * b_4 \quad (2.79)$$

Here are the coefficient values of the equation and since we take the start and end values of the robot motion as zero, the values of the coefficients in the equation are as follows. The same procedure applies to coefficients of θ_2.Finally we get result as in Figure 2.17.

$$d_1 = \theta_{1i}$$

$$c_1 = 0$$

$$b_1 = \frac{3 * (\theta_{1f} + \theta_{1i})}{tf^2}$$

$$a_1 = \frac{-2 * (\theta_{1f} + \theta_{1i})}{tf^3} \tag{2.80}$$

$$\begin{pmatrix} ti^3 & ti^2 & ti & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ti^3 & ti^2 & ti & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3\,ti^2 & 2\,ti & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3\,ti^2 & 2\,ti & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ tm^3 & tm^2 & tm & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & tm^3 & tm^2 & tm & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3\,tm^2 & 2\,tm & 1 & 0 & 0 & 0 & 0 & 0 & -3\,tm^2 & -2\,tm & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3\,tm^2 & 2\,tm & 1 & 0 & 0 & 0 & 0 & 0 & -3\,tm^2 & -2\,tm & -1 & 0 \\ 6\,tm & 2 & 0 & 0 & 0 & 0 & 0 & 0 & -6\,tm & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6\,tm & 2 & 0 & 0 & 0 & 0 & 0 & 0 & -6\,tm & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & tf^3 & tf^2 & tf & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3\,tf^2 & 2\,tf & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & tf^3 & tf^2 & tf & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3\,tf^2 & 2\,tf & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & tm^3 & tm^2 & tm & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & tm^3 & tm^2 & tm & 1 \end{pmatrix} \begin{pmatrix} a1 \\ b1 \\ c1 \\ d1 \\ a2 \\ b2 \\ c2 \\ d2 \\ a3 \\ b3 \\ c3 \\ d3 \\ a4 \\ b4 \\ c4 \\ d4 \end{pmatrix} = \begin{pmatrix} K \\ OO \\ 0 \\ 0 \\ M \\ P \\ 0 \\ 0 \\ 0 \\ 0 \\ C \\ 0 \\ E \\ 0 \\ M \\ P \end{pmatrix}$$

Figure 2.21: Two segment final equation

Where K, OO, M, P, C, E is boundry conditions. K first joint angle in first segment which is equal to θ_{1i} is shown Figure 2.20. OO second joint angle in first segment which is equal to to θ_{2i} is shown Figure 2.20.M first joint angle in the via point inverse kinematic solution which is θ_{11}. P second joint angle in the via point inverse kinematic solution which is θ_{21}.C is first joint angle in the target position. E is the second joint angle in the target position. The starting position of the robot arm end effector, which is designed as an example application, is X=75 mm, Y=80 mm, and X=100 mm, Y=125 mm for the target position. Figure 2.22 Start, Target, and Midpoint position in SolidWorks Figure 2.23 The trajectory drawn as a result of two segment analysis is shown.

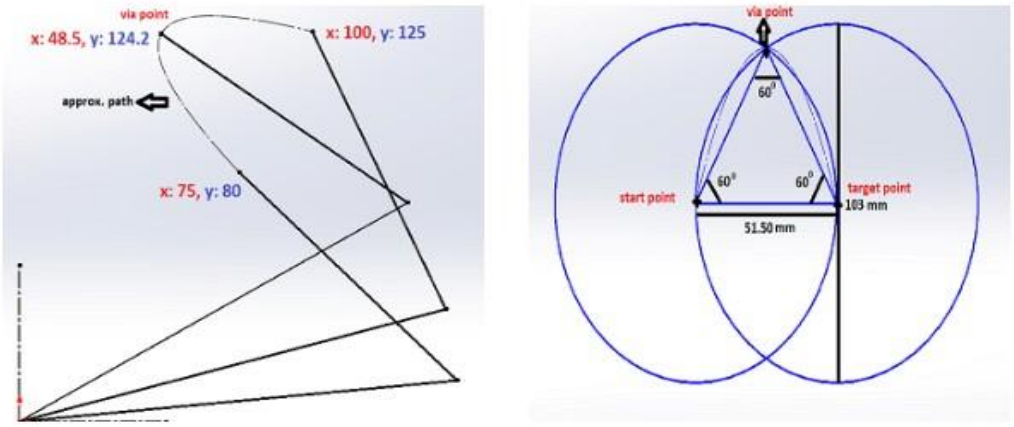


Figure 2.22: A example of 2 segment trajectory analysis in SolidWorks

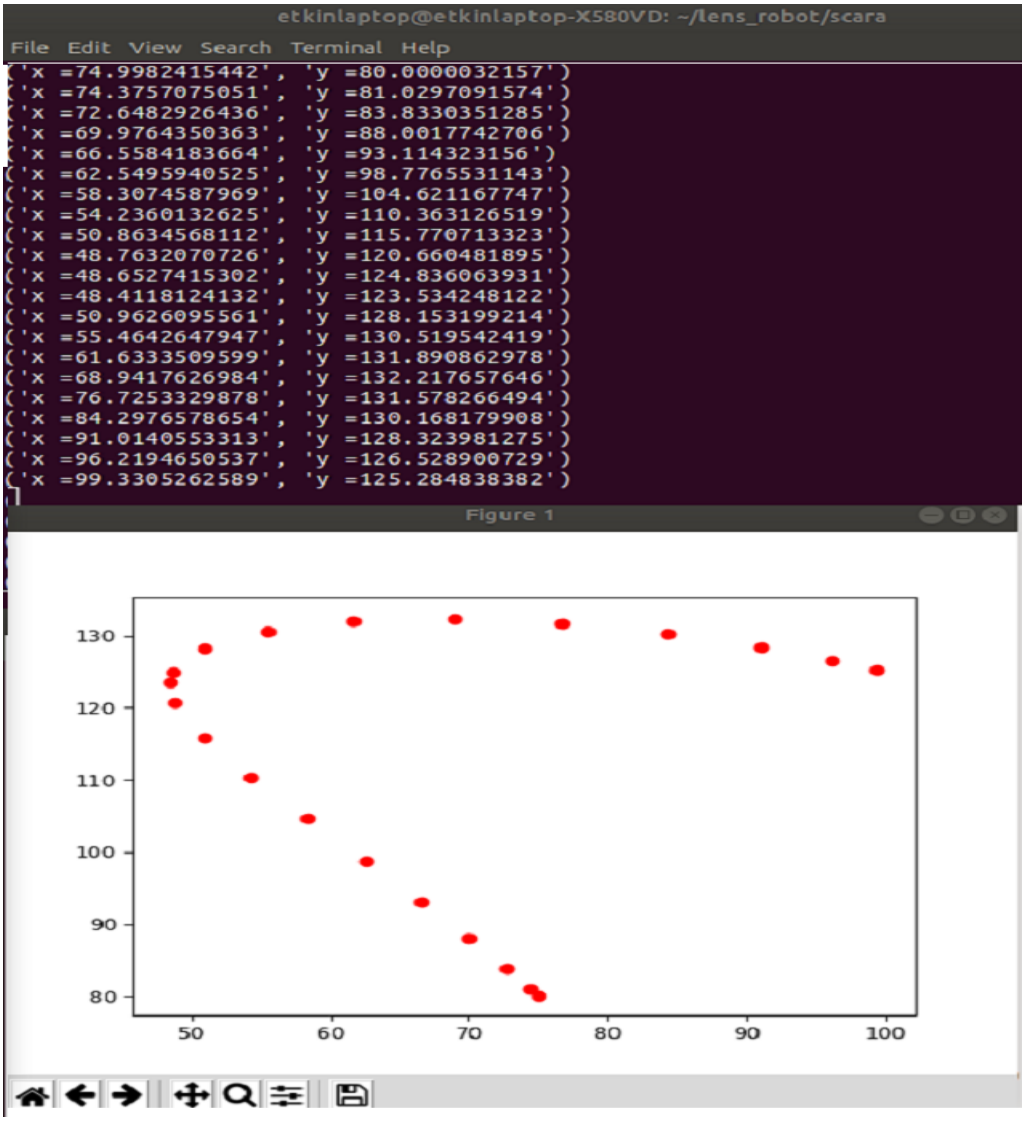


Figure 2.23: A example of 2 segment trajectory analysis in Python code

Angle values of a solution selected from inverse kinematic analysis results are as follows.

x: 75 mm, y: 80 mm => (ik solution for initial position) θ_1=5.05, θ_2=133.04

x: 48.5 mm, y: 124.2 mm => (ik solution for via point) θ_1=27.86, θ_2=119.39

x: 100 mm, y: 125 mm => (ik solution for target position) θ_1=13.89, θ_2=103.25

According to the results of the analysis, it was observed that all joints reached the starting position and ending position of the curves without any error, as seen in the Path-Time graph (Figure 2. 23). In the Speed-Time equations, the joint motors start at zero speed at t=0 because there is no motion at initial position of the manipulator. and reach the maximum speed in half of the targeted time with an increasing speed, and in the other half, the speed decreases at the same rate and slowly reaches zero because manipulator reach the target position. The acceleration is inversely proportional to the velocity. In this way, the motors in the joint complete their tasks at the same time, even if the joint angles are different, and the motors are given appropriate speed and acceleration values, providing a smooth movement.

In order to fully understand importance of the trajectory planning and to find the maximum torque values in the worst scenario for motor selection, 5 different points which are closest to the singular points were selected as a via point within the workspace of the robot arm. The x, y position values of 5 different points are shown in the Figure 2.24 below as mm. Analysis were made by determining the weight of the 1st link as m1=150 gr and the weight of the 2nd link as m2+mload=350 gr in total with the load on its end-effector. Since the same operations are performed for all joints, only the calculations for θ_1 are presented below. Because the torque value of the 1st actuator will be the maximum torque value and the selected bus servo actuators will be equal to each other, so it is known that the torque value obtained as a result of the calculations for the 1st actuator will be sufficient for the 2nd actuator. Path-time graph (degree-sec), velocity-time graph (mm-sec), acceleration-time graph (mm/sec squared) and torque-time graph (N.m-sec) are shown in the below graphs (Figure 2.25-2.29 to the via-points, respectively) which produced as a result of the analysis.

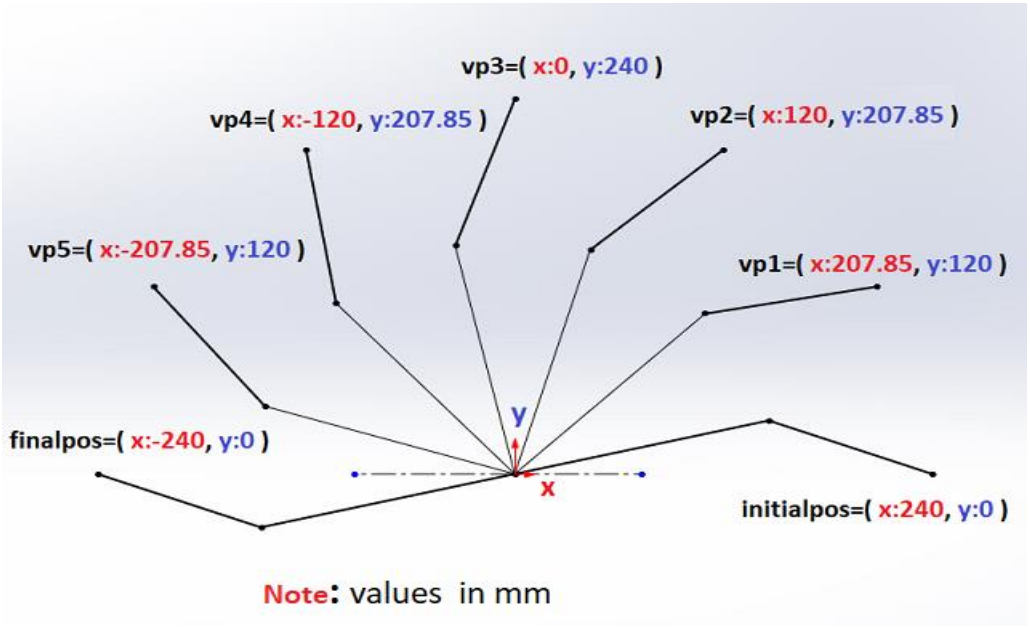
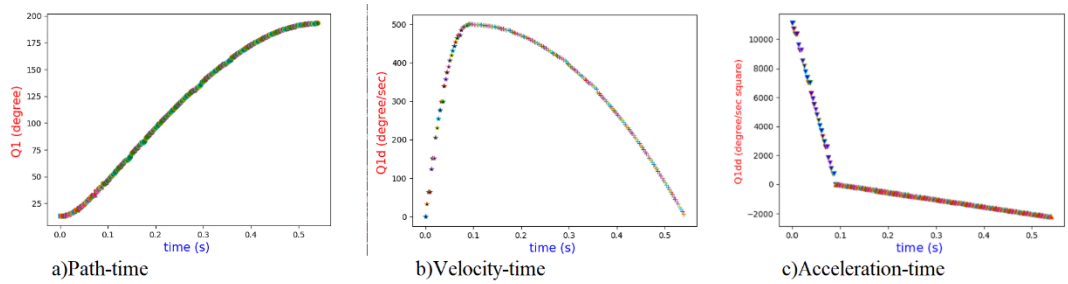
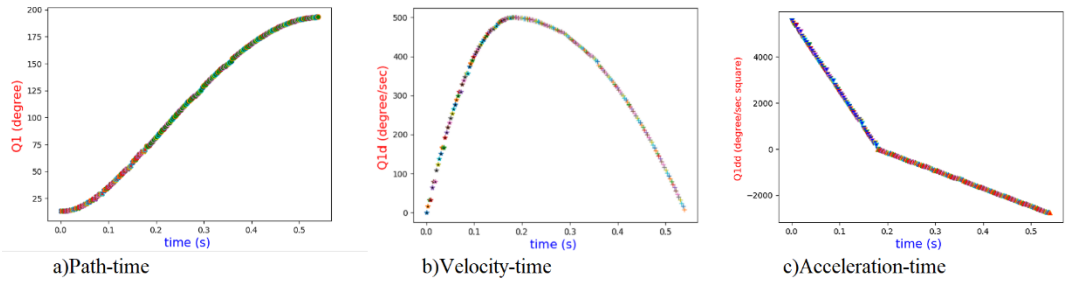


Figure 2.24: 5 different via points



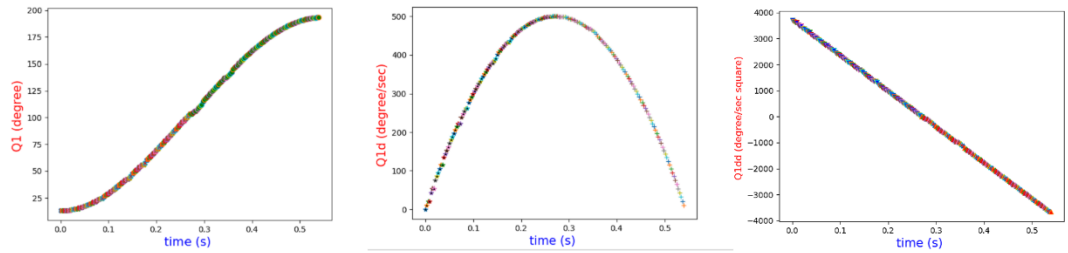
a)Path-time b)Velocity-time c)Acceleration-time

Figure 2. 25: For θ_1 vp1 a) path-time graph b) velocity-time graph c) acceleration-time graph



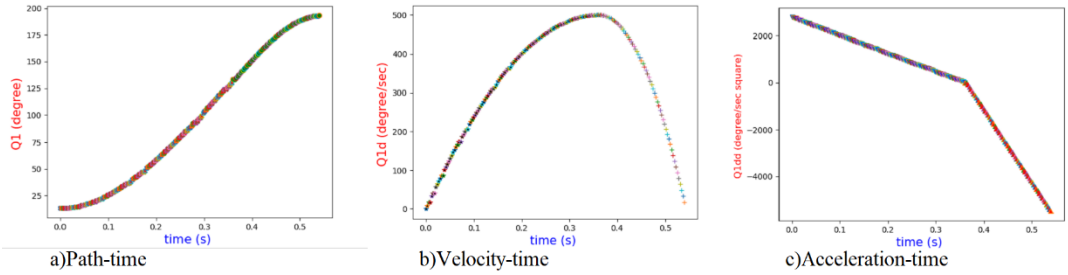
a)Path-time b)Velocity-time c)Acceleration-time

Figure 2.26: For θ_1 vp2 a) path-time graph b) velocity-time graph c) acceleration-time graph



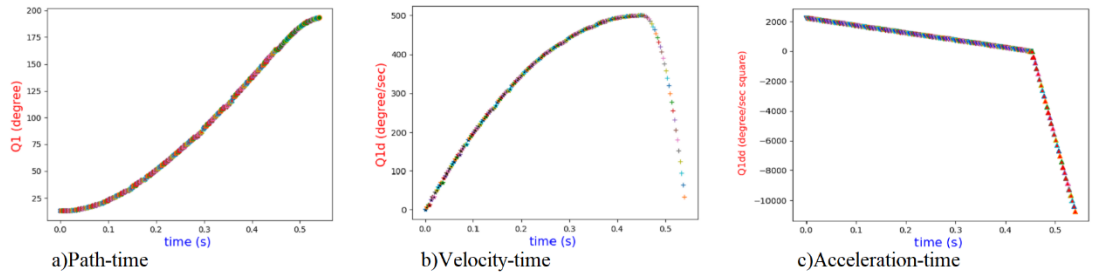
a)Path-time b)Velocity-time c)Acceleration-time

Figure 2.27: For θ_1 vp3 a) path-time graph b) velocity-time graph c) acceleration-time graph



a)Path-time b)Velocity-time c)Acceleration-time

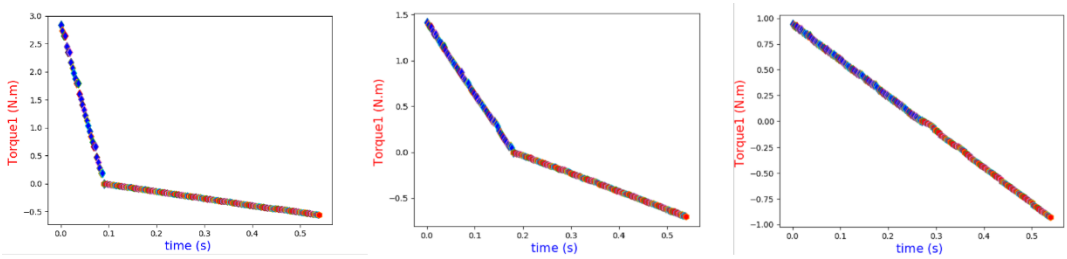
Figure 2.28: For θ_1 vp4 a) path-time graph b) velocity-time graph c) acceleration-time graph



a)Path-time b)Velocity-time c)Acceleration-time

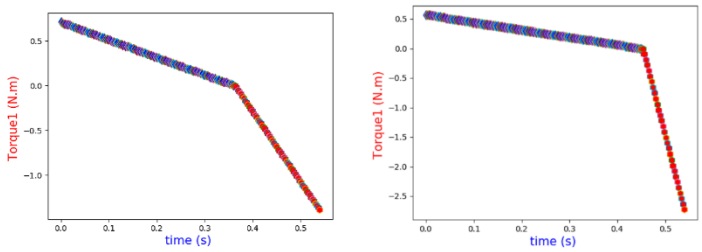
Figure 2.29: For θ_1 vp5 a) path-time graph b) velocity-time graph c) acceleration-time graph

Torque-time graphs showing torque values according to these 5 via-points are shown in Figure 2.30 and Figure 2.31, respectively.



a)vp1 torque-time b)vp2 torque-time c)vp3 torque-time

Figure 2.30: For θ_1 vp(1-3) a) torque -time graph b) torque -time graph c) torque -time graph



a)vp4 torque-time b)vp5 torque-time

Figure 2.31: For θ_1 vp(4-5) a) torque -time graph b) torque -time graph

To determine the motor torque, you can examine the maximum torque in detail in Table 2.3. As a result of the trajectory analysis, the maximum torque for the actuator was found to be 2.83 N.m, and since the selected actuator torque was 3.43 N.m, it was found suitable for the manipulator.

Table 2.3: Trajectory analysis for 5 different points include actuator torque calculation inside workspace.

	Initial Pos (x,y)	Final Pos (x,y)	Via Point (x,y)	Actuator Speed (sec/deg)	Total Travelled Angle (deg)	Total Travelled Time (sec)	Via Point Angle (deg)	Via Point Time (sec)	Time Increment Resolution (sec)	Total Number of Points	Via Point Number	Max Torque (N.m)
Anlys1	240, 0	-240, 0	207.85, 120	0.72/240	180	0.54	30	0.09	0.003	180	30	2.83
Anlys2	240, 0	-240, 0	120, 207.85	0.72/240	180	0.54	60	0.18	0.003	180	60	1.41
Anlys3	240, 0	-240, 0	0, 240	0.72/240	180	0.54	90	0.27	0.003	180	90	0.94
Anlys4	240, 0	-240, 0	-120, 207.85	0.72/240	180	0.54	120	0.36	0.003	180	120	1.41
Anlys5	240, 0	-240, 0	-207.85, 120	0.72/240	180	0.54	150	0.54	0.003	180	150	2.83

Chapter 3

Middleware

Almost all industrial robots have their software. Ur5 robot arm with 5 degrees of freedom also has its Polyscope control and URSim Offline Simulation software. There is additionally an HMI interface that uses pieces of drivers provided by the manufacturer with limited functionality. However, the industry needs a more robust system that gives exhaustive control over the robot and integrates it with the sensors for feedback. Therefore, the approach to using a middleware package was considered to get a working control system. Middleware is the software environment between applications that communicate with each other to perform repetitive tasks. They do not perform any action on their own. A robot middleware is a framework used to improve a robot application. Research has been done on middleware selection, and the decision to choose the appropriate middleware package has been explained by specifying the advantages and disadvantages. Robot middleware usually needs a specific operating system to run. It is a virtual platform that manages the robot frameworks and hardware connected to the computer on which the operating system is installed. The most used operating systems in the market are Windows, macOS, and Linux and all their variants.

3.1 Researched and comparison of middleware

There are many choices of robot middleware that can be used in robots as mentioned below. They are

Open Robot Control Software (Orocos)

Python Robotics (Pyro)

Player/Stage

Optimal Reciprocal Collision Avoidance (Orca)

OpenRTMaist

Robot Software Communications Architecture (RSCA)

Microsoft Robotics Developer Studio (MRDS)

Open Platform for Robotic Services (OPROS)

Coupled Layer Architecture for Robotic Autonomy (CLARAty)

Robot Operating System (ROS)

Microsoft Robotics Developer Studio (YARP)

Webots

To find significant information of robotic middlewareis difficult. The middleware that does not meet the requirements of this thesis was eliminated and it was desired to choose the appropriate one. Considering specific features of this special project, Robotic Systems: Applications, Control and Programming book which has written by Elkady, A, and Sobh, provides almost enough content to get the needed information of each middleware. The comparison of the above-mentioned robotic middleware is given in Table 3.1 in alphabetical order.

Table 3.1: Middleware comparasion

Middleware	Perception System	Appropriate architecture	Crashing procedures	Open source	Devices libraries	Community support
Orocos	×			×		
Pyro				×		
Player/Stage	×			×		
Orca	×			×		
OpenRTMaist	×	×	×	×		
RSCA	×			×		
MRDS		×				
OPROS		×	×	×		
CLARAty	×	×	×			
ROS	×	×	×	×	×	×
YARP	×	×	×	×		
Webots	×	×	×			

Why ROS and not other X software? This question has many answers. A short and clear answer would be as follows. The objective of the Ros framework is to establish a standard and develop a link between the robot and the programmer. In addition to the standard libraries and devices it supports, ROS increases its scope day by day and takes its standard position in the robot world, as it also supports libraries and drivers written by the developers. Another advantage offered by ROS is that the code written once can be run on all supported devices instead of writing the codes separately for each robot. A standard algorithm built for a typical robot will be operable at load-use speed when received in this fashion. With the help of ROS, the data from the robot's sensors will be transmitted to the computer, and the data processed by the computer will be able to be sent back to the robot as a task. Using the ROS interface's topics and messages, this communication system will be developed. ROS and the Gazebo simulation environment are fully compatible. Consequently, the Gazebo simulation environment may be utilized for rapid development to evaluate the established data structure, which was transformed into the ROS Package during the project's prototype phase.

Other advantages are listed below.

- ✓ Because the purpose of ROS is to support code, it increases reusability and doesn't urge you to invent the wheel from scratch on every project.
- ✓ It can run between two operating systems. For example; You can send code from MATLAB installed on Windows to the ROS Master that you created on the robot and return data from the robot. It's just that ROS doesn't force you to use only Linux, and you can take advantage of the conveniences offered by other operating systems.
- ✓ Language is independent. It supports many different programming languages such as C++, python, lisp, java, and Lua and does not force you to learn a particular programming language.

- ✓ It is open source. You can use and rearrange the powerful tools in the community repository it offers you as you wish. The most important of these are visualization, simulators, and debugging tools.
- ✓ The number of supported robots and sensors is increasing day by day.
- ✓ It can work cross-platform. We can write the nodes we create in different programming languages. One node can be C++ while the other node can be java or python.
- ✓ It is modular. If there is a problem with most of the other android applications, this may cause the main code to crash and stop the robot application. But this is not the case with ROS. Since there is one node for every transaction in ROS, when one node fails, the other nodes continue to work.
- ✓ It reduces complexity as a separate topic is published for each transaction. This makes debugging easier.
- ✓ It has an active community. You can find many resources, documents, books, articles, and papers about ROS.
- ✓ ROS has a lot of robot arm-based projects. However, several industrial-processes-based projects can be also found.

Considering the advantages from Table 1 and the above items, we conclude that ROS is more reliable for working with such processes.

3.2 Introduction to Ros

ROS first emerged in the mid-2000s at Stanford University to support the development of systems called STAIR (Standford AI Robot) and PR (Personal Robot Program).In 2007, Willow Garage, a company headquartered in Menlo Park, California, provided substantial resources to develop the system. This was the beginning of many studies

under the BSD license, providing significant resources and experts to the development of ROS. Over time, ROS has evolved into a widely utilized platform for robotics research. ROS development was handed to the Open Source Robotics Foundation (OSRF) in 2013, where it continues to this day. ROS is currently utilized by tens of thousands of users worldwide in experimental and industrial applications. The versions and dates of support for various operating systems are listed in Table 3.2.In this thesis, the version named Melodic Morenia [29] with the longest support period was preferred.

Table 3.2: ROS distribution

ROS Distribution	Relase Date	EOL Date
Box Turtle	March 2, 2010	
C Turtle	August 2, 2010	
Diamondback	March 2, 2011	
Electric Emys	August 30, 2011	
Fuerte Turtle	April 23, 2012	
Groovy Galapagos	December 31, 2012	Jul, 2014
Hydro Medusa	September 4, 2013	May, 2015
Indigo lgloo	July 22, 2014	April, 2021
Kinetic Kame	May 23, 2016	April, 2021
Melodic Morenia	May 23, 2018	May, 2023
Noetic Ninjemys	May 23, 2020	

Stacks and packages included in the ROS operating system are developed by universities, companies, and private individuals all over the world and are often licensed with open source code. A package and stack can be indexed on the main ROS website and made available to all communities [30]. This sharing network of ROS is of great importance in terms of the increasing dissemination of ROS software around the world.

ROS is designed as a specialized software and framework unit for robotic platforms. Basically, many robotic tools provide hardware simulation and message transfer systems between various software nodes. Nodes are stand-alone modules that can operate independently, communicating on said topics using a one-to-many subscriber model and TCP/IP protocol. This feature of ROS is of great importance when working with distributed systems. For example, the functionality required even in a single robot operation, namely the components for sensor and actuator controls, can increase

heavily. Motion planning, trajectory planning, collision detection, mapping, navigation, and obstacle avoidance are just a few of the applications that can be realized with this distributed sensor network. Each system has a name server, or ROS-Master, which informs nodes regarding which topic is published by which node. Each node must therefore save the issues it publishes to the ROS-Master. Another node that wishes to subscribe to a topic queries ROS-Master for the addresses of relevant publications and contacts them directly. After the initial phase of searching, the nodes link directly as in a peer-to-peer network.

ROS is written primarily in C++ and Python. ROS nodes utilize client libraries for C++, Python, Java, Matlab, and Lisp, among others. However, any programming language may be used for message delivery system and protocol communication. ROS-Master. In addition, the ROS operating system mainly runs under Ubuntu/Linux and is partially supported by other operating systems. However, the modular nature of ROS allows various configurations to be easily created for different settings. The codes used in the thesis study were tested on the simulation and real system. For these studies to be run on other ROS-enabled robots, the urdf file, image processing code, and Moveit module must be adapted according to the robot's mechanical, motion and sensor model.

It is possible to handle the ROS structure at 17 different levels;

1) Introduction to Ros (Ros File system, Ros Computational Graphics Levels)

2) Urdf and Xacro

3) Coordinate Frames and Transforms

4) Visualization

5) Ros Stacks and Packages

6) Rosserial Protocol

7) Basic Linux and Ros Commands

8) Ros Industrial

9) Community Level

10) Installing and settling Ubuntu 18.04 Melodic LTS

11) Start to work with Ros

12) First Step to working with Ros

13) How to Create Packages

14) Rviz

15) Gazebo

16) Moveit

17) Creating URDF packages

These levels and concepts are examined in detail and explained in related titles.

3.2.1 ROS File System

All concepts and applications developed on ROS need resources to process, move and store data. A unique file system has been created for this need. This file system ensures proper data management, applications, and packaging parts. Figure 3.1. shows the schematic structure of how the ROS files and folder are arranged in memory [31].

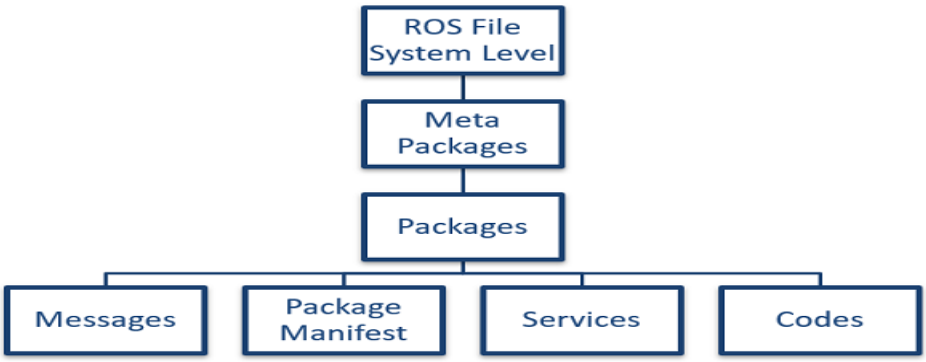


Figure 3.1: ROS file system structure

3.2.1.1 Packages

In an application developed in ROS, components are combined and organized into packages. This enables the software to be created and released over ROS. A package can contain nodes, ROS-dependent-independent libraries, configurations, datasets, and anything related to the application.

3.2.1.2 Metapackages

Metapackages are virtual packages specialized in ROS. They do not load files and do not contain tests, code, files, or other items usually found in packages. It only serves to refer to one or more packages.

3.2.1.3 Package Manifest

Package manifests are an XML file named package.xml that must be located in the root directory of any catkin-compatible package. This file defines package-related properties such as package name, version number, author, maintainer, and dependencies on other catkin packages. Also, the package creation system uses this information provided by the package manifests.

3.2.1.4 Message Types

The message type, in its basic form, is a file with the extension **".msg"** and a predefined text representation for the included data structures. ROS employs simplified message description language to define the data values that ROS nodes broadcast. This description makes it simple for ROS tools to produce source code automatically for the message type in several target languages. Message descriptions in a ROS package's msg/ subfolder msg files are kept. The two components of a.msg file are fields and constants. The fields contain the data transmitted within the message. Constants define interpretable values that can be applied to certain fields.

ROS gives numerous distinct messages. Even while we can design our own messages, it is recommended that we utilize ROS default messages wherever possible. Messages are defined in.msg files, which are stored in a package's msg directory. For example; std_msgs/Int32

[std_msgs/Int32]:

int32 data

In this instance, the Int32 message has only one variable of type int32 named data. This Int32 message originates from the std msgs package and can be found in its msg directory [32].

3.2.1.5 Services

The service type is similar to the message type in file format. It also has an extension **“. srv"** and its data structure representation contains two sections separated as request and response. It is used in the name service to provide service communication over the system. For example;

trajectory_by_name_srv/TrajByName

string traj_name

---

bool success

string status_message

Does it seem familiar? It should, because it's the same structure as the Topics messages, with some addons.

1- Service message files have the extension .srv. Remember that Topic message files have the extension .msg

2- Service message files are defined inside an srv directory. Remember that Topic message files are defined inside a msg directory.

3- Service messages have TWO parts:

REQUEST

RESPONSE

REQUEST contains a string called traj name, and RESPONSE consists of a boolean named success and a string called status message.

Depending on the requirements of the service, the amount of components on each component of the service message can vary. Even if it is unnecessary to our service, we can enter "none." Because they identify the file as a Service Message, the three dashes are the most crucial component of the message [33].

Summarizing:

The REQUEST portion of the service message specifies HOW a service call will be executed. This indicates which variables must be passed to the Service Server for it to accomplish its duty.

The RESPONSE portion of the service message specifies HOW our service will respond once its functionality has been completed. If, for example, it returns a string

with a specific message indicating that everything went fine, or if it returns nothing, etc...

3.2.2 ROS computation graph level

One of the most exciting aspects of ROS is a distributed system that is not controlled by just a master data controller, extending the system through large networks and making applications scalable. Communication from all over the network is between colleagues and peers. This peer-to-peer network spawns the "Computation Graph", a chain of networks to process data together. Computing in ROS is done using a transaction network called ROS nodes. This computational mesh can also be called a computational graph. The main concepts in the computational graph are the concepts of ROS nodes, ROS Master, Parameter Server, Messages, Topics, Services, and Ros bags where ROS messages are stored. (Figure 3.2) [34].

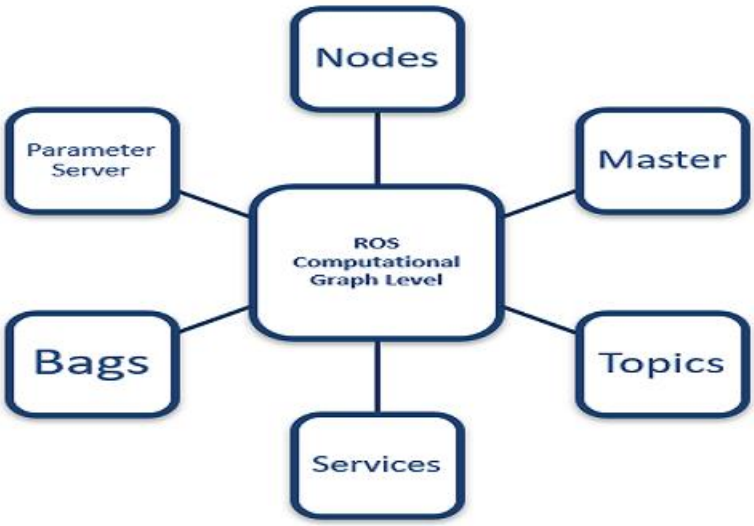


Figure 3.2: ROS computation graph level

3.2.2.1 Nodes

Node is the most basic unit that does the computation in ROS. Each node in ROS is involved in a transaction. Considering an example robot arm application, a node can control transformations between axes. One controls the motion commands and the last

node controls the graphical display of the system. Node units play an important role in the distributed and scalable nature of ROS [35].

3.2.2.2 Master

When comparing the main control system, the "Master" in ROS does not do any calculations. Instead, it provides name registration and searches for the rest of the calculation graph. Controls all node communication, message exchange, and service calls. From another perspective, it is similar to the famous DNS system of the internet [36].

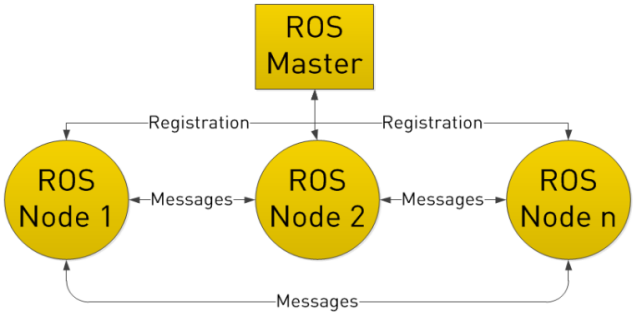


Figure 3.3: Role of ROS master

3.2.2.3 Parameter Server

Parameter Server ensures that data is stored by the switch in a central location. A parameter server is a shared, multivariate dictionary with network API accessibility. This server is utilized by nodes to store and retrieve parameters at runtime. It is best suited for static, non-binary data such as configuration parameters because it is not optimized for fast performance.

3.2.2.4 Messages

Communication between nodes is provided via messages. There is more than one message standard available in ROS, in the same way, new ones can be created specially and added here.

ROS has its default messages but we can even create our messages when it’s necessary. Messages are defined in .msg files as mentioned above, msg is located inside a msg

directory of a package. For example; std_msgs/Int32 [geometry_msgs/Pose]: Point position, Quaternion orientation

In this case, the Pose message has two variables named position and orientation of type Point and Quaternion. This Pose message originates from the geometry msgs package and can be found in its msg directory.

3.2.2.5 Topics

The subject is a data source system in the ROS network. One can subscribe or publish data on the subject and exchange the predefined message over the network. Multiple nodes can publish or subscribe to the same topic at the same time. For example, let's consider an inertial measurement unit called IMU (Inertial Measurement Unit), which includes angular positions in the x, y, and z axes. A node collecting data from the hardware can broadcast the received data to the subject. After that many other nodes can subscribe and receive IMU data about it. It is also the same in the opposite direction. Different nodes can feed IMU topic broadcast data. A node can subscribe to this topic [37].

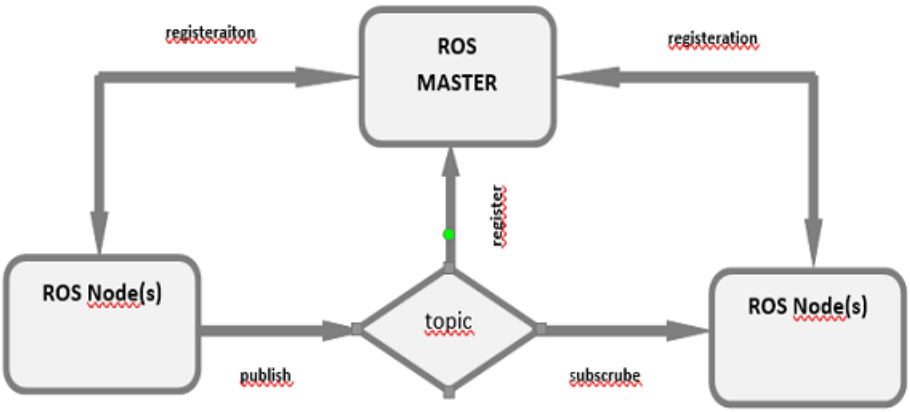


Figure 3.4: ROS topic visual explanation

3.2.2.6 Services

Numerous ROS packages simply use topics and their work is flawless. Why then does Ros provide services? In some instances, subjects are either insufficient or too complex to employ. If we use services it makes life easier.

To comprehend what services are and when they should be utilized, it is necessary to compare them to themes and actions. Imagine that a mobile robot exists. It is equipped with a laser sensor, face recognition technology, and a navigation system. The laser will publish all laser readings at 20 Hz via a Topic. Because other ROS systems, like as the navigation system, require constant access to this information, we utilize a topic to store it.

The facial recognition technology will render a service. Your ROS application will call that service and wait until it returns the name of the individual in front of the mobile robot.

Utilize services when the program cannot continue until the service's result is received [38].

3.2.2.7 Actions

> "*ROS Actions have a client-to-server communication relationship with a specified protocol. The actions use ROS topics to send goal messages from a client to the server. You can cancel goals using the action client. After receiving a goal, the server processes it and can give information back to the client.*[39]"

To comprehend what actions are and when to employ them, it is necessary to compare them to services. Action is provided by the navigation system. A ROS software will invoke the action to move the robot, and while it performs that duty, it will conduct other duties, such as grumbling about how exhausted C-3PO is. In addition, this operation will provide feedback (such as the remaining distance to the specified coordinates) throughout the movement process. Consequently, what is the distinction between a Service and an Action? Services are real-time. When a ROS program requests a service, the program cannot proceed until the service returns a result. An

action is asynchronous. Similar to starting a new thread. When the ROS program issues a call to action, the program is able to conduct other tasks while the action is completed in a separate thread.

3.2.2.8 Ros Bags

A bag is a file format used in ROS to store message data. Important data shared between nodes can be stored in bags as a message. Bags are useful for storing messages as real actions and then using them in simulations and reviews. For example, an application using a laser range finder could collect data in a real environment and put it in a bag. It can be reused later in the algorithm development phases.

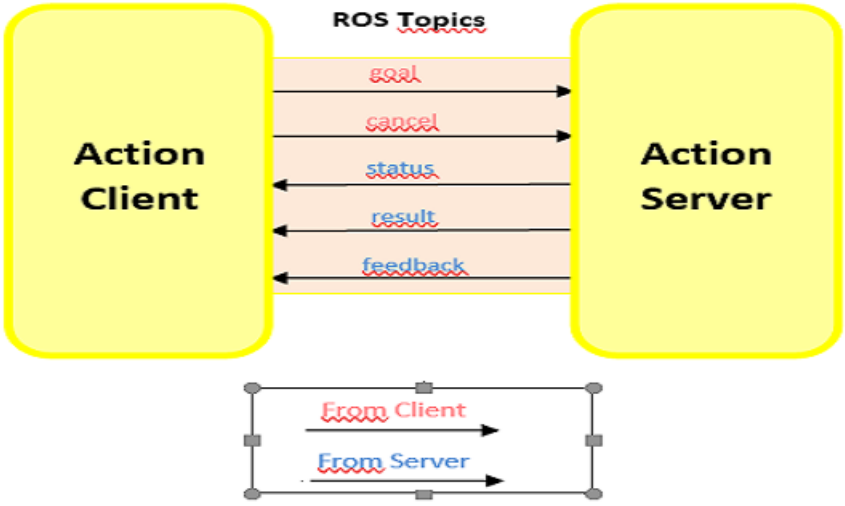


Figure 3.5: ROS action structure

3.2.2.9 Name Ordering

For the ROS computation graph to work properly, each resource in the graph must have an appropriate name. "Graph Resource Names" provides this hierarchical naming structure. Every node, parameter, topic, or service must be named under this hierarchy. These naming conventions are similar to namespace conventions in object-oriented programming languages. As an example, the global namespace can be defined as follows:

- ✓ / Any resource name in this global namespace could be the following example:
- ✓ /sample parameter Under this parameter we can add other resources. In this situation:
- ✓ /sample parameter/other parameters can be defined. In this example, "other parameters" is defined under the "example parameter" namespace.

An important thing to remember is the naming conventions for these resources. "other parameters" can be a node name or a parameter name that holds the value.

With namespace logic, resources create their resources under namespaces. Communication between different namespaces is possible, but usually above both namespaces and according to the appropriate semantics. Otherwise, the encapsulation payoff of the namespace logic is meaningless.

Resources are not aware of their namespaces. So they can be written as if they were all working at the same level. If they are used under another chart, it is sufficient to put them in a suitable subgraph. For example, suppose an application named LaserLibraries has a node named Scanner. If someone else wants to use it in their application, then simply add it to the subgraph called LaserLibraries. If the top-level application types a Scanner node name, it will not conflict with the other Scanner node. Because naming

- ✓ /Scanner
- ✓ /Laser Libraries/Scanner.

With this nomenclature, for code reuse, a researcher can use the laser libraries package in their application without affecting their application and save time by reusing the code.

3.3 URDF (Unified Robot Description Format) and Xacro (XML Macros)

URDF (universal robot description file) package is a parser developed for URDF robot description file structure in XML file structure. The robot definition file has a tree

structure. In the system named after the tree structure, there can be only one main link to which a link is connected, and thus a chain is formed. It does not support robots with parallel structures. In the robot definition file structure, there are definitions of all infrastructures that make up the robot. These definitions include the dimensions of the links, the image files and position orientations of the links, the joints, the rotation and translation limits of the joints, the rotation axes of the joints, the positions and orientations of the sensors, and the positions and orientations of the actuators. In addition to these, physical properties such as mass, inertia, and collision properties can be assigned to the links and joints that make up the robot. Links are introduced as a fixed link in the robot description file if they are stationary, and as joints, if they are actuators. The extension of the robot definition file is urdf and the robot definition file can be uploaded to the variable server. The robot definition uploaded to the variable server becomes available to nodes, motion planners, transformation package, 3D viewer rviz, and Gazebo. Almost every robot using ROS has a robot definition file. Although these files are not mandatory, they are very useful in the design phase. Detailed information about URDF is given on the roswiki community webpage [40]. The URDF file contains the robot-specific information required for nodes to execute their procedures. Each link of the robot is the child of a parent link, with joints linking each link, and joints are described with their offset from the reference frame of the parent link and axis of rotation [41]. This creates a comprehensive kinematic model of the robot.

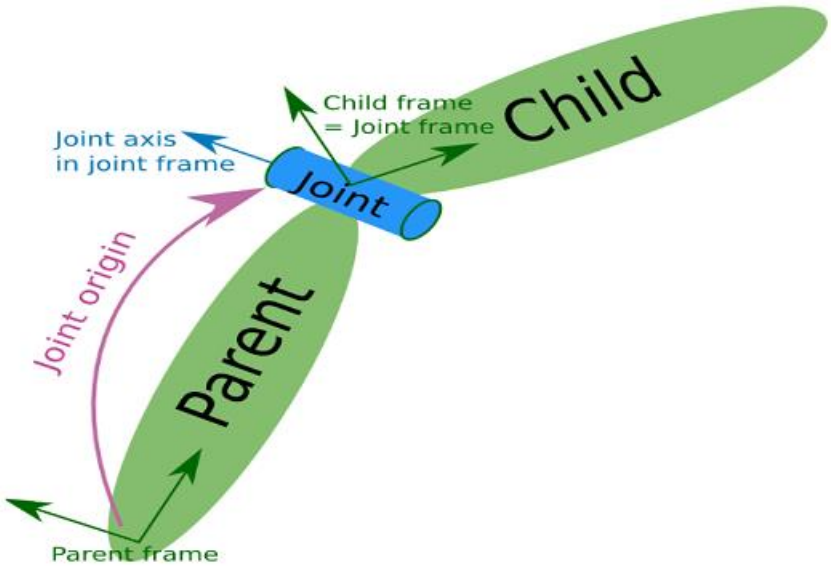


Figure 3.6: Urdf elements [42]

Very simple robots can have really big and complex URDF files. For this reason and to be able to make URDF files more flexible for change, XACROS are used in Robot Model definitions even more than URDF files. XACROS also gives some tools for making simple Math operations inside the URDF files, as well as having Constants and MACROS. Xacro is based on a macro language. All Xacro files end up being a single URDF file, but this way, we will have much cleaner and easier-to-maintain robot model definitions.

3.4 Coordinate Frames and Transforms

There is more than one coordinate system in a robot. Transformations between these coordinate systems have an important place in robotics. If no trace of where one coordinate system is located relative to another is kept, many difficulties may arise in planning motion, avoiding obstacles, and performing tasks. ROS has a package called 'tf' [43] that solves this problem. This package is very modular and can be easily applied to any robot. Each robot link and/or sensor is assigned a position, orientation information, and a set of axes, which is relative to the reference coordinate system. When the robot moves, it can be easily followed where this sensor and/or link is located according to the main coordinate system, at what time, and where. This package is perhaps the most powerful package ROS has to offer. Almost every robot using ROS benefits from this package.

Since these data are kept on the main server of ROS, it is possible to track where a coordinate system was compared to the main coordinate system in the past. Since ROS can work in a distributed network structure, if we choose our main coordinate system as a room or a factory hangar, many robots can be tracked by each other in the building or room. Assigned axes can be broadcast either programmatically under a hood or, if the axis is fixed, statically in ROS. Extensive information on publishing and signing up for these published titles are covered in detail in [44]. In Figure 3.7, it is shown how the assigned to coordinate systems of the robot named PR2 produced by Willow Garage company appear in tf.

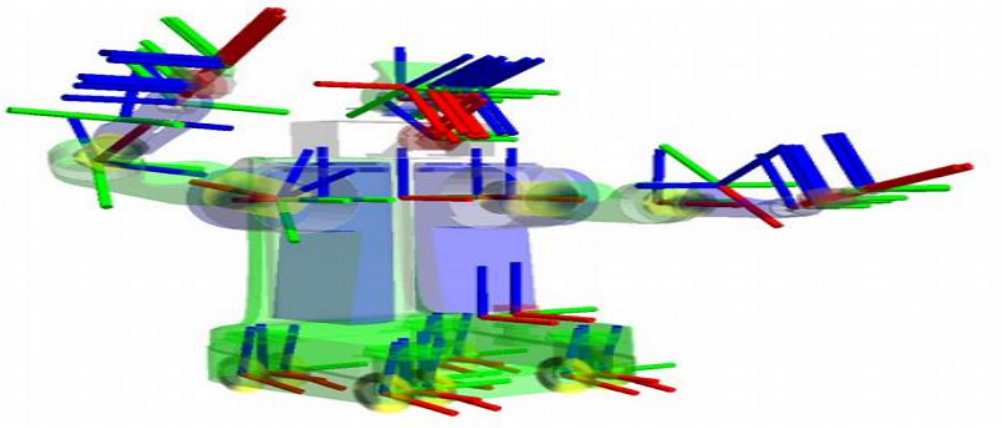


Figure 3.7: PR2 coordinate frame

3.5 Visualization

The Rviz package, developed by Willow Garage, comes standard with ROS. All titles published in ROS and the packages that make up these titles are visualized with a 3D visualization tool called rviz. All standardized packages can be viewed in ROS, but if there is a special message structure created by the developer, the developer must program a software plug-in for rviz to display it in rviz. A few examples of the message types that ROS can display are as follows.

- ✓ Sensor messages
- ✓ 2D and 3D Map messages
- ✓ point clouds
- ✓ Position, orientation messages
- ✓ Coordinate systems messages
- ✓ Camera messages

Since ROS has a dispersed structure, messages published on a robot can be viewed in rviz on a computer used by another operator. As a 3D imaging package, rviz can run on any operating system platform where ROS can run. Detailed information about rviz

is given in [45]. As an example, the display of different messages of rviz is given in Figures 3.8.

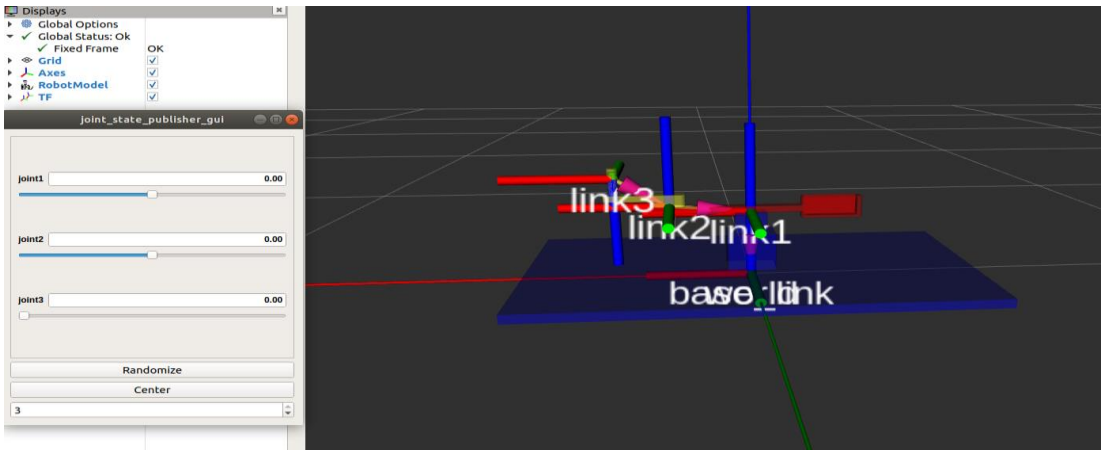


Figure 3.8: Image of Scara in rviz

3.6 Ros Stacks and Packages

The ROS code is grouped into two different levels;

- ✓ Packages (rplidar package, navi_goals package etc.)
- ✓ Stacks (Navigation Stack)

Typically, a repository is composed of stacks, which are collections of packages. Typically, a repository stores all stacks provided by a single enterprise. Stacks contain packages with identical subject matter [46] . For instance, the navigation stack includes many planner packages, a high-level ROS node, a localization package, and obstacle data structures.

3.7 Rosserial Protocol

For a new sensor or new actuator or any kind of device that want to add to the ROS ecosystem two situations whether this hardware already has a driver or not. If the sensor has a driver this means that it is already supported and the information which is coming from the sensor is directly processable by Ros. However, there is some other

hardware that does not support a robot operating system and this is where the problems come out [47]. In such a case two different situations;

Develop drivers (this is a time-consuming and very tedious operation)

Rosserial protocol (Mcu, serial port, or network socket.)

The rosserial protocol is aimed at point-to-point ROS communications over a serial transmission line.

3.8 Basic Linux and Ros Command

Tables 3.3 and Table 3.4 list several fundamental Ubuntu and Ros commands used throughout this thesis. Table3.3 displays the use of 30 fundamental commands that facilitate the execution of numerous Ubuntu operations using the CLI.

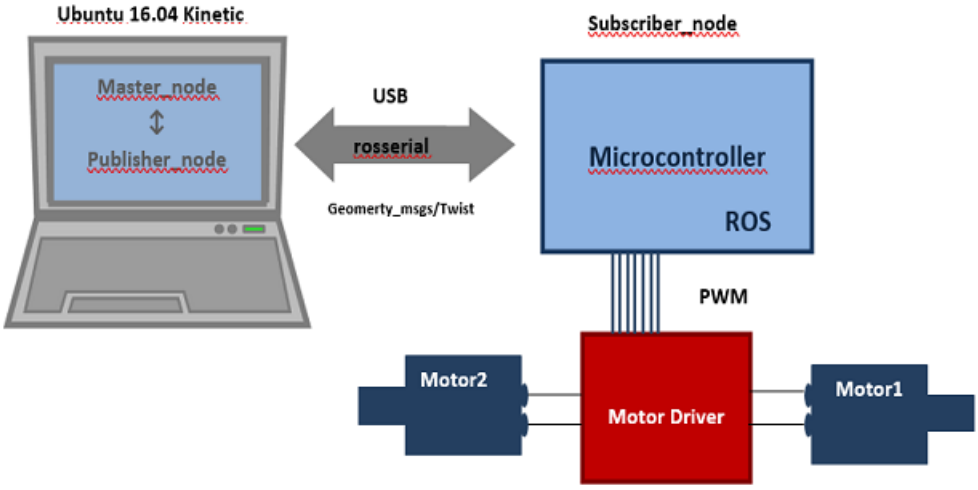


Figure 3.9: Rosserial communication

Table 3.3: Basic Linux comcommands

File Commands	
Is	Directory Listing
Is -al	Formatted Listing with Hidden Files
cd *dir*	Change Directory to *dir*
cd	Change Directory to home
pwd	Show Current Directory
mkdir *dir*	Make New Diretory *dir*
rm *file*	Remove File *file*
rm –r *dir*	Remove Directory *dir*
rm –f *file*	Force Remove File *file*
rm –rf *dir*	Force Remove Directory *dir*
cp *file1 file2*	Copy *file1* to *file2*
mv *file1 file2*	Rename/Move *file1* to *file2*
In –s *file link*	Creates a Symbolic *link* to *file*
touch *file*	Create or Update *file*
cat > *file*	Place Standart Input into *file*
more *file*	Output Contents of *file*
less *file*	Output Contents of *file*
head *file*	Output First 10 Lines of *file*
tail *file*	Output Last 10 Lines of *file*
tail –f *file*	Output Contents of *file* as it Grows

Package Management	
apt-get update	Refresh Available Updates
apt-get upgrade	Update All Packages
apt-get dist-upgrade	Upgrade Ubuntu Version
apt-get install *pkg*	Install *pkg*
apt-get purge *pkg*	Uninstall *pkg*
apt-get –f install	Try to Fix Broken Packages

Network	
ifconfig	Display Network Information
iwconfig	Display Wireless Information
ifup *interface*	Enable *interface*
ifdown *interface*	Disable *interface*

Table 3.4: Basic ROS commands

Roscore	Starts ROS Master.
Rosrun <pkg_name> <node_name>	Starts executable node.
Roslaunch < pkg_name> <launch_file>	Starts launch file.
Rostopic list	Lists all active topics.
Rostopic info </ topic_name>	Provides data on topic such as type, subscribers and publishers.
Rostopic echo </ topic_name>	Prints topic messages to screen.
Rostopic hz </ topic_name>	Prints publishing rate to screen.
Resnode list	Lists all nodes running.
Resnode info <node_name>	Provides data on node such as publications, subscriptions, services, and Pid.
Rosmsg show –r <msg_type>	Prints raw message text.
Rospack find package_name>	Prints file path to package.
Rosrun rqt_graph rqt-graph	Tool to visualize graphical representation of active packages, nodes, and topics.
Rosrun rviz rviz	Starts ROS visualization tool.
Rosbag record -0 <filename> </topic>	Starts rosbag tool to record data from a desired topic.

3.9 Ros Industrial

"*ROS-Industrial is a BSD (legacy) / Apache 2.0 (preferred) licensed program that contains libraries, tools, and drivers for industrial hardware. It is supported and guided by the ROS-Industrial Consortium. The goals of ROS-Industrial are to:*

- ✓ *Create a community supported by industrial robotics researchers and professionals*
- ✓ *Provide a one-stop location for industry-related ROS applications*
- ✓ *Develop robust and reliable software that meets the needs of industrial applications*
- ✓ *Combine the relative strengths of ROS with existing industrial technologies (i.e. combining ROS's high-level functionality with the low-level reliability and safety of industrial robot controllers).*
- ✓ *Create standard interfaces to stimulate "hardware-agnostic" software development (using standardized ROS messages)*
- ✓ *Provide an easy path to apply cutting-edge research in industrial applications, using a common ROS architecture*
- ✓ *Provide simple, easy-to-use, well-documented APIs.*"

This valuable information has been extracted from ros industrial web page [48]**.**

3.9.1 Current Members of ROS Industrial

Figure 3.10: Current members of ROS Industrial[49]

In Fıgure 3.10, some known companies are current members of Ros Industrial for 2022 year.

3.9.2 Supported Hardware

Some famous robotic companies such as ABB, Adept, Fanuc, Motoman, and Universal Robot support their products with Ros (see Figure 3.11).

Vendor	Controller(s)	Position Streaming	Trajectory Downloading	Trajectory Streaming	Torque Control	IO Control	Manipulator	MoveIt Pkg
ABB	IRC5	NO	YES	NO	NO	NO	IRB-2400	YES
							IRB-5400	NO
Adept	CX, CS	YES [3]	NO	NO	NO	NO	Viper 650	NO
Fanuc	R-30iA / R-30iB	YES[4]	NO	NO	NO	NO	LR Mate 200iC (all)	YES
							LR Mate 200iD	YES [5]
							M-10iA	YES
							M-16iB/20	YES
							M-20iA(/10L)	YES
							M-430iA/(2F, 2P)	YES
							M-900iA/260L [5]	NO
							Other	NO [7]
Motoman	DX100	NO	NO	YES	NO	YES[11]	SIA10D/F	NO
	FS100	NO	NO	YES	NO	YES[11]	SIA20D/F	YES
	DX200	NO	NO	YES	NO	YES[11]	MH5F	YES
	YRC1000	NO	NO	YES	NO	YES[11]	SDA10F	YES
							Other	NO [7]
Universal Robot	CB2/CB3 [10]	YES [8]	NO	NO	NO	YES [12]	UR 5	YES
							UR 10	YES

Figure 3.11: Supported hardware[50]

3.10 Ros Community Level

ROS Community Level are ROS resources that allow researchers, and hobbyists to exchange software and information about Ros. Some of these resources are ROS Wiki, ROS Answers, Blogs, ROS distributions, and repositories.

Researchers and developers can post documentation, links to their repositories, and tutorials for any open-source software they have created on the ROS Wiki. The ROS Answers community is a website dedicated to answering questions about ROS.

3.11 Installing and Setting Ubuntu Melodic 18.04 LTS

In order to install and use Ros, first of all, a computer with the appropriate Ubuntu operating system is required. For Ros Melodic to work properly, we need to have Ubuntu 18.04 LTS version. This version will be supported until 30.05.2023. All desired versions of Ubuntu can be installed from the official site of Ubuntu [51].

3.12 Start to work with Ros

3.12.1 Installing Ros

To install Ros, the commands on the relevant version installation web page of Ros can be followed step by step [52].

1. Configure Ubuntu repositories to allow "restricted," "universe," and "multiverse."

2. Setup computer to accept software from packages.ros.org.

- $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

3. Set up the keys

- $ sudo apt install curl # if you haven't already installed curl
- curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

4. Installation

- $ sudo apt-get update
- $ sudo apt-get install ros-melodic-desktop-full

5.Install and Initialize rosdep

- $ sudo apt install python-rosdep
- $ sudo rosdep init
- $ rosdep update

6.Environment setup

- $ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
- $ source /opt/ros/melodic/setup.bash

7. Dependencies for building packages

- $ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

3.12.2 Setting IDE

A easy way to work with ROS package content is using IDEs. For this thesis, Visual Studio Code will be used because author has previous work experience with this IDE. Is important to note the creation of different packages will be performance through Ubuntu terminal.

To add a workspace to Visual Studio Code, Follow steps File—>Import —> Existing projects into the workspace, press next, and pick the package's root directory. Then press finish.

3.12.3 Setting Workspace

To study with Ros need to create workspace inside home directory. This workspace is the file system where all the studied and ready packages and source codes are located.Follow the steps below to create a Workspace [53].

1. Create a catkin Workspace

- $ source /opt/ros/melodic/setup.bash
- $ mkdir -p ~/catkin_ws/src
- $ cd ~/catkin_ws/
- $ catkin_make

3.12.4 Installing Gazebo, MoveIt, ROS-I UR, and Ros Control Libraries

If the desktop version of Ros is installed, Gazebo and Moveit should already has. So, just need to be install ROS-Industrial universal robots and ros-control.

- $ sudo apt-get install ros-melodic-universal-robot
- $ sudo apt-get install ros-melodic-ros-controllers ros-melodic-ros-control

3.13 Creating Package

This is the syntax for creating a ROS package [54].

- $ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]

package_name is the name given to the package that wanted to be create.depend1, depend2, depend3 specifies the dependencies on which the package to build will depend. Follow the steps to create a package.

1. Go to src folder of catkin_ws

- $ cd ~/catkin_ws/src

2. Create package

- $ catkin_create_pkg beginner_tutorials std_msgs tf urdf xacro rviz gazebo_ros ros_controllers rospy roscpp

In the packages created based on dependencies, only the files originating from the dependencies exist. Some required files are shown in Figure 3.12 below and their usage purposes are explained.

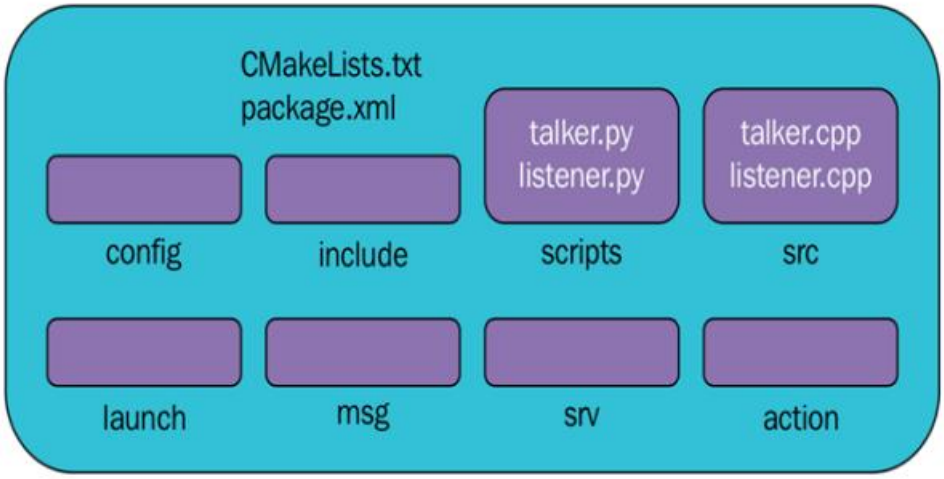


Figure 3.12: ROS package structure

- ✓ The include folder contains headers and libraries used in the package.
- ✓ The scripts folder contains executable Python scripts. talker.py and listener.py are examples.
- ✓ The src folder contains C++ source codes. talker.cpp and listener.cpp are examples.
- ✓ The launch folder contains launch files used to run several nodes from the package.
- ✓ The meshes folder contains stl file of robot parts
- ✓ The urdf folder include urdf description file
- ✓ The config folder include controller.yaml file
- ✓ The msg folder provides definitions for custom messages.
- ✓ The srv directory holds service definitions.
- ✓ The action definition is located within the action folder..
- ✓ package.xml is the *package manifest* file of this package.
- ✓ CMakeLists.txt is the *CMake build* file of this package.

3.14 Rviz

All titles published in ROS and the packages that make up these titles are visualized with a 3D imaging package called rviz. All standardized packages can be viewed in ROS, but if there is a special message structure created by the developer, the developer must program a software plug-in for rviz in order to display it in rviz. A few examples of the message types that ROS can display are as follows:

- ✓ Sensor messages
- ✓ 2D and 3D Map messages
- ✓ Position, orientation messages
- ✓ Coordinate systems messages
- ✓ Camera messages

- ✓ PointClouds
- ✓ Lasers
- ✓ Kinematic Transformations
- ✓ RobotModels

The list is endless. Even defining your own markers is possible. It is one of the reasons why ROS was so well received. Prior to Rviz, it was really challenging to comprehend what the Robot was sensing. And it is the key idea. Rviz does not simulate reality. Rviz is a visualization of what the simulation or actual robot is publishing in the subjects.

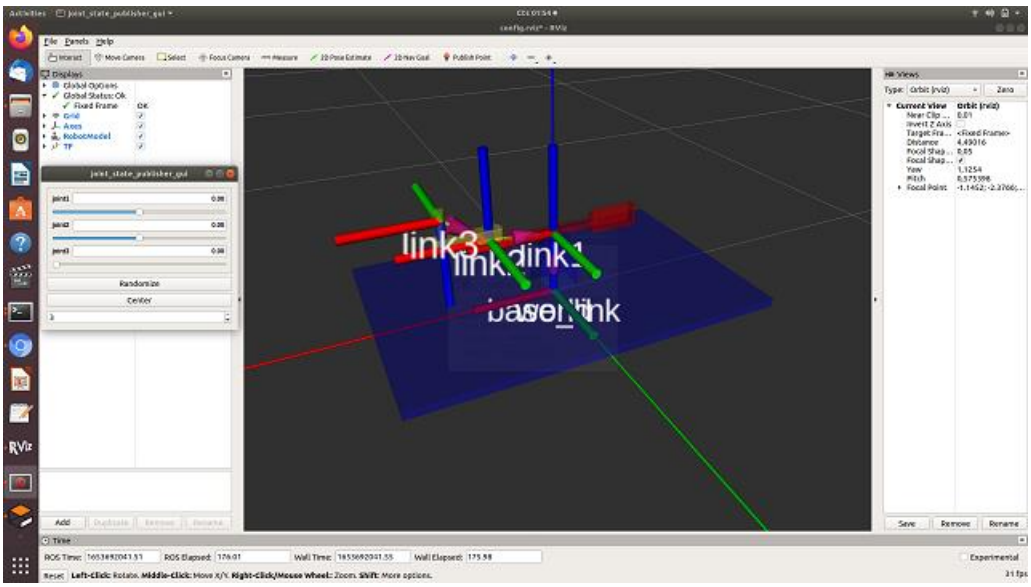


Figure 3.13: Scara in Rviz

3.15 Gazebo

Gazebo is an open source software project that began development at the University of Southern California in 2002. Today, the development of the project has been financially undertaken by Willow Garage. Gazebo is a three-dimensional simulation program for multi-robots working outdoors and indoors. It has the ability to simulate many objects, sensors, and the interaction between the object and the robot [55]. Gazebo also includes a physics engine that simulates solid-body dynamics. Gazebo can be used alone or combined with ROS. Some of the tools Gazebo offers are as follows:

- ✓ Imu sensor
- ✓ Force-torque sensor
- ✓ Laser sensor
- ✓ Sonar sensor
- ✓ Kinect sensor
- ✓ Camera sensor
- ✓ Dc motor
- ✓ Real-time controller infrastructure

URDF files are very useful, for having a virtual representation of the different links and joints. Yet, it is not simulating its weight, its inertia, what sensors it has, how it collides with other objects, the friction with the floor, or how the servo position control will react to the robot. These are just some of the topics that gazebo cover. To make a model appear in Gazebo you need to add the following:

- ✓ Inertias: The inertias and mass of the model is important if you want it to have physics applied
- ✓ Gazebo Physical Properties: Here you specify the frictions and material properties like colour or softness.
- ✓ Collisions: Without collisions the Robot models would just go through the floor.
- ✓ Visual Properties: We add colour to the visual, which only will affect the URDF in RVIZ, the colour in the simulation is defined in the Gazebo Physical Properties.
- ✓ Gazebo Sensors: Here is when you add cameras, motor controllers and all the robots systems.

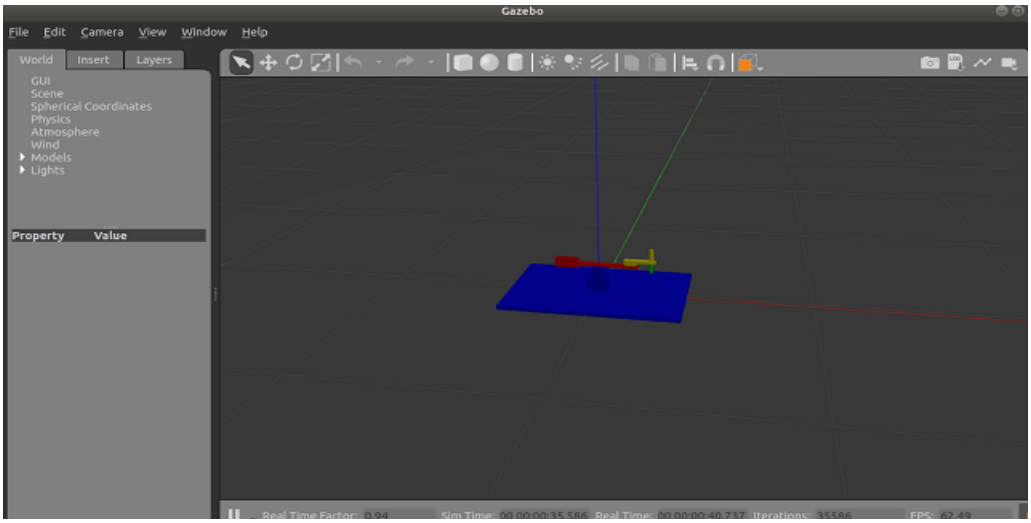


Figure 3.14: Scara in Gazebo

3.16 Moveit

The MoveIt Setup Assistant provides a graphical interface for configuring MoveIt for a new robot. It starts with the robot's URDF model and builds an SRDF (Semantic Robot Description Format) model on top of it. The SRDF model includes data on pairs of links that will never collide, such as any nearby linkages. The MoveIt Setup Assistant generates this type of information, but the user controls everything else. One or more motion groups can be defined by the user. The user can set up an end-effector, a kinematic solver, and numerous postures for each group [56].

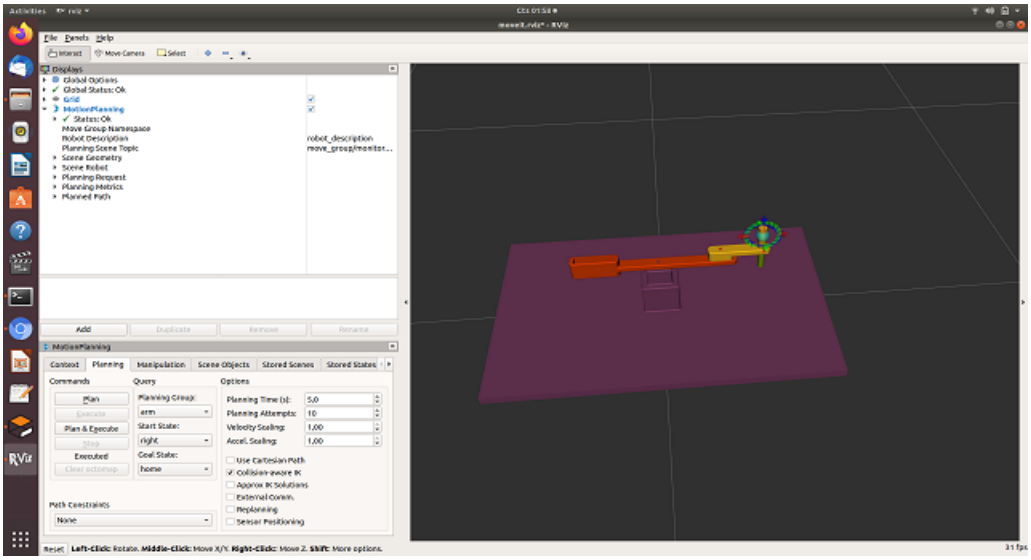


Figure 3.15: Scara in Moveit

3.16.1 Motion Planning

Move It is equipped with a variety of motion planners, they are:

- ✓ OMPL (Open Motion Planning Library)
- ✓ CHOMP (Covariant Hamiltonian Optimization for Motion Planning)
- ✓ STOMP (Stochastic Trajectory Optimization for Motion Planning)
- ✓ SBPL (Search-Based Planning Library) (full integration work in progress)

The Open Motion Planning Library (OMPL) [57] is used by Moveitas a default configuration, and it contains a range of sampling-based planning techniques.The developer has the option of using one of the OMPL algorithms or creating a custom algorithm that can be readily integrated into the app. The KDL, a numerical Jacobian-based kinematics solver, is Moveit's kinematics solver. However, the user can create and integrate his own kinematics solver into the software. The IKFAST plug-in from OpenRave can also be used to create one. For a kinematic chain with 6 or less DOFs, KDL and IKFAST provide better inverse kinematics results. When dealing with kinematic chains with more than six degrees of freedom, it's best to employ a specialized kinematics solver.

There are three solver plugins currently available:

- ✓ KDL (Kinematics and Dynamics Library) Kinematics Plugin
- ✓ LMA (Levenberg-Marquardt) Kinematics Plugin
- ✓ Trac-IK Kinematics Plugin

3.16.2 Motion Control

Move It comes with two versions of a motion control manager for the user to choose from. The motion control manager can start, stop, and monitor motions. The robot's motors are controlled by a motion control manager, and the fake motion control manager simulates the motion of the robot's motors. The fake motion control manager isn't suitable with Gazebo (simulator of ROS). It can, however, be used if simply the

robot's movements, not the environment or sensor output, is to be simulated. The non-fake manager and Gazebo should be used together when the user wants to replicate robot mobility in a simulated environment. Each MoveIt motion group has its own dedicated motion controller, however joints may have numerous controllers if they are part of multiple motion groups.

3.17 Creating URDF packages

While creating our own Ros package, first of all, it will be explained how to create urdf and mesh files from the 3D design we created in Solidworks and transfer them to the Ubuntu environment.

3.17.1 Solidworks to URDF

To configure your URDF Export, the exporter first shows up a property manager page. We willl have to manually configure each link and construct the tree. The tool will keep our configuration with the assembly itself after we have configured it for the first time. After that, you should be able to skip this step.

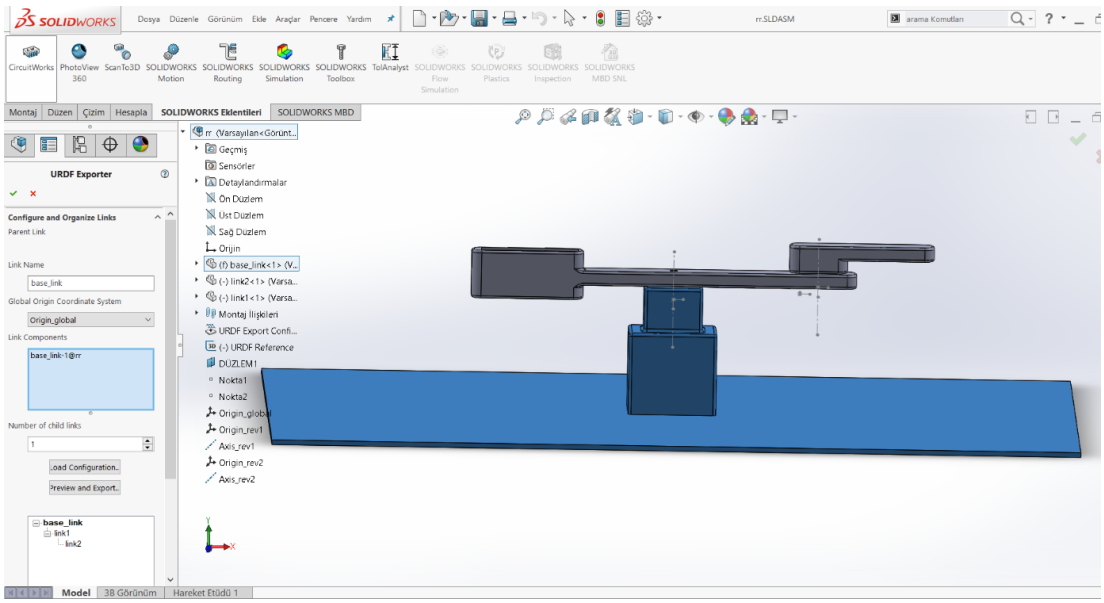


Figure 3.16: Solidworks to Urdf exporter

We must give each connection a unique name, a unique joint name, pick the SolidWorks components (assemblies or parts) that will be connected with that link,

and add the required number of children. You can choose from the list if you have any reference coordinate systems or axes that you'd like to use with the tool. You can also explicitly designate which joint type each joint belongs to. There is only one base link in each URDF. Because there is no joint to a higher level link, the joint configurations are disabled for this one link. All you have to do now is give the link a name (if you don't want it to be called base link), choose its components, and construct its child links.

3.17.1.1 Joint Properties

If you realize that these joint origins are not always formed in the optimal position, you can modify the reference coordinate systems and axes outside of the exporter to meet your requirements. On the other side, the model will perform flawlessly as-is. The 3D Sketch is only for temporary building, although you may find it useful for modifying the positioning of the reference geometry. On the Joint Properties page, you can save the names of the coordinate systems and axes to your configuration by pressing cancel. You can then modify the coordinate systems and axes. To restart the export process, select "File”> "Export to URDF" from the menu bar. Ensure that the right coordinate system and axis names are maintained for every connection. Instead of creating them, the tool will resort to the existing reference geometry.

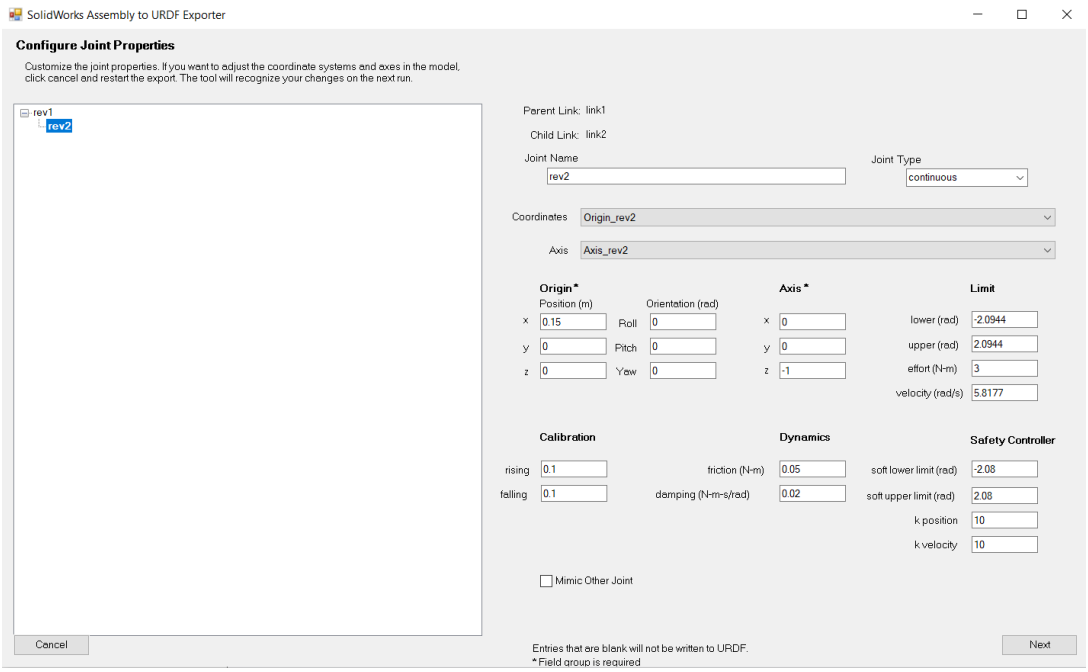


Figure 3.17: Urdf joint properties

The joint page features a non-configurable joint tree, with each joint's characteristics displayed on the right-hand side when we click it. Before exporting, you can change the attributes of any joint. These settings, however, will not be preserved with the setup. Each time, they'll have to be re-entered. Click next to go to the Link Properties page.

3.17.1.2 Link Properties

This page looks identical to the Part Exporter we looked at earlier. Any connection in your tree can have its link characteristics changed. You can alter the texture, the color, the origins of different parts, the moment of inertia tensor, the mass, and so on. These modifications will not be preserved with your settings as well.

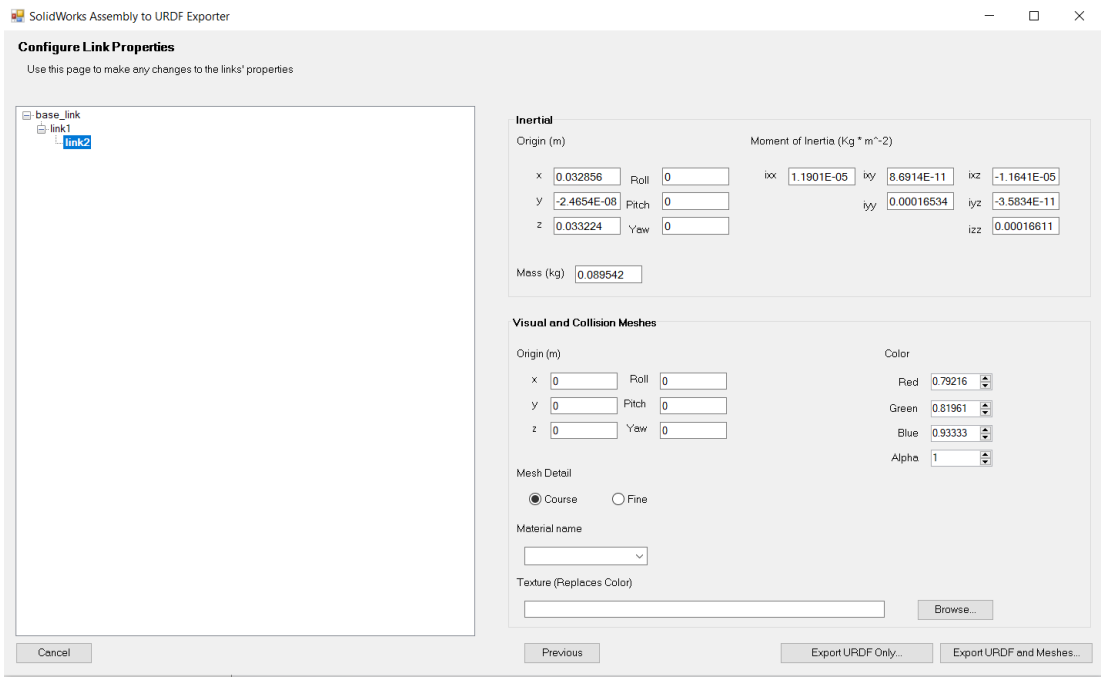


Figure 3.18: Urdf link properties

3.17.1.3 The built package

After all the adjustments are made, the urdf and mesh files are saved by clicking the export urdf and meshes button. Then, these files in the windows or mac environment are moved to the Ubuntu environment with the help of a memory stick.

3.17.2 Changes to be Made in Urdf

Urdf files taken from Solidworks do not contain colors for the rviz environment. For Rviz, only the colors are assigned to the links.

For the Gazebo simulation environment, physical properties must be added, these physical properties are listed below.

Inertias: The inertias and mass of the model is important if you want it to have physics applied

Gazebo Physical Properties: Here you specify the frictions and material properties like colour or softness.

Collisions: Without collisions the Robot models would just go through the floor.

Visual Properties: We add colour to the visual, which only will affect the URDF in RVIZ, the colour in the simulation is defined in the Gazebo Physical Properties.

Gazebo Sensors: Here is when you add cameras, motor controllers and all the robots systems.

In addition, motor control is provided by adding the following add-ons.

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/scara</robotNamespace>
  </plugin>
</gazebo>
```

This activates the gazebo control plugin for the scara namespace. This namespace will be set in the main spawn launch file.

```
<transmission name="tran1">
```

Here you define the name of the transmission, which will have to be unique through the URDF file.

<type>transmission_interface/SimpleTransmission</type>

The type of transmission that, for the moment, is the only one implemented: "transmission_interface/SimpleTransmission"

<joint name="roll_joint">
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>

</joint>

Here you link the joint to the transmission and select the hardware interface, which also only has an effort interface working.

<actuator name="motor1">
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>

<mechanicalReduction>1</mechanicalReduction>

</actuator>

Set a unique name for the actuator and again select the hardware interface. As for the reduction, it's self-explanatory.

3.17.3 Scara package

All of the necessary folders for running the robot application will be found in the scara package. Eight folder will be found in the scara package: config, launch, meshes, src, script, urdf All of the folder in the scara package were created specifically for this thesis by the author. Scara is a simple package found in the source's src directory. Open a terminal and type the following commands to make this package:

- $ cd catkin_ws/src/
- $ catkin_create_pkg scara rospy roscpp urdf xacro rviz gazebo_ros std_msgs

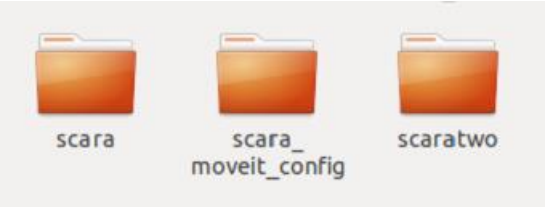


Figure 3.19: Scara ROS package from Ubuntu

The config folder will contain 1 configuration files:

- ✓ ros_controllers.yaml

Information provided by ros_controllers.yaml will be:

- ✓ Name of controller (joint_state_controller, arm_controller)
- ✓ Type of controller
- ✓ (joint_state_controller/ JointStateController, position_controllers/JointTrajectoryController)
- ✓ Name of the joints (joint1, joint2, joint3)
- ✓ Pid values of each joint

The launch folder will contain 8 files:

- ✓ rviz_urdf.launch
- ✓ gazebo_ urdf.launch
- ✓ check_motors.launch
- ✓ check_motors_gazebo.launch
- ✓ rviz_all.launch
- ✓ joint_state_all.launch
- ✓ gazebo_command_all. launch
- ✓ urdf_config.rviz

The rviz urdf.launch file is launched and transferred to the rviz visualization environment of the robot model. Here it can be checked whether the joints and links are correct.

The gazebo urdf.launch file is launched and transferred to the gazebo simulation environment of the robot model. Here, it can be checked whether the physical properties work correctly.

With the check_motors.launch file, robot movements in rviz are provided by using rviz gui sliders.

The check_motors_gazebo.launch file is launched and transferred to the gazebo simulation environment of the robot model by adding a controller. Here, it can be checked whether the controller is working correctly or not by punlishing the data suitable for the relevant topic from the shell.

The rviz_all.launch file is launched and transferred to the rviz visualization environment of the robot model, by adding a controller to the gazebo simulation environment. Also, rosserial is started for the communication between Arduino and Ros. Here, the joint angles found from the trajectory algorithm written by the thesis author are pressed on the joints, respectively, and the robot in the view, the robot in the gazebo simulation environment and the physical robot are shown to follow the simultaneous trajectory.

The joint_state_all.launch file is launched and transferred to the rviz visualization environment of the robot model, by adding a controller to the gazebo simulation environment. The joint_state_publisher gui is also added to rviz. Also, rosserial is started for the communication between Arduino and Ros. Here, it is shown that the joint angles found from the trajectory algorithm written by the thesis author are pressed on the joints, and the simultaneous trajectory of the robot in the view, the robot in the gazebo simulation environment and the physical robot are followed.

The gazebo_command_all.launch file is launched and transferred to the rviz visualization environment of the robot model, by adding a controller to the gazebo simulation environment. Also, rosserial is started for the communication of Arduino and Ros. Here, it is shown that the joint angles found from the trajectory algorithm

written by the thesis author are pressed respectively on the joints of the robot in the gazebo, and the simultaneous trajectory of the robot in the view, the robot in the gazebo simulation environment and the physical robot are followed.

The urdf_config.rviz file ensures that the plugins to be added in rviz are pre-registered and added to each launch file in a ready-made form, so that the additions are ready in rviz.

The meshes folder contains STL files of the robot extracted from Solidworks. These STL files are used inside urdf file to display the simulation robot looks like real robot.

The script folder contains python files. These are:

- ✓ Joint_states_to_gazebo.py
- ✓ path_generator_v5_1.py
- ✓ path_generator_v5_2.py
- ✓ pub_joint1.py
- ✓ pub_joint2.py

Joint_states_to_gazebo.py python code is used to transfer joint state values from rviz or arduino to the move the robot in the gazebo.

path_generator_v5_1.py and pub_joint1.py python codes performs trajectory analysis for join_state_publisher and finds the necessary joint angles for trajectory tracking and transfers them to rviz, gazebo and physical robot with joint_state_publisher as in rviz.

path_generator_v5_2.py and pub_joint2.py python codes performs trajectory analysis for command topic and finds the necessary joint angles for trajectory tracking and transfers them to gazebo, rviz, and physical robot with command topic as in shell topic command.

The src folder normally contains c++ files. In this scara package c++ language is not used.

The urdf folder contains the urdf file which are the descriptions of the robot.

- ✓ Scara.urdf
- ✓ Scara_gazebo.urdf
- ✓ Scara_joint_state.urdf

3.17.4 Scara Robot MoveIt Config Package and Motion

3.17.4.1 Building a MoveIt package

MoveIt! can generate all of the necessary files for the robot to complete the motion tasks indicated in the URDF. Developers may create the MoveIt! configuration package using the Moveit setup assistance tool. The URDF file that defines the robot is the only required to run this tool. Roscore has to be launched,before run the setup assistant tool [58]. To do that open a new terminal and write roscore:

- ➢ $roscore

Afterthat, run the setup assistance by typing the following in another terminal.

- ➢ $ rosl aunch moveit_setup_assistant setup_assistant.launch
- ✓ Loading your robot's URDF file

As seen in Figure 6 left, the MoveIt! setup aid GUI should display on the screen. By using the create a new package button, the developer will be able to select and load the robot description, which is located in the urdf folder of the scaratwo package in this example. When you press the Load Files button, the other of the tabs become active, as illustrated in Figure 3.20.

Figure 3.20: MoveIt setup assistant befor and after load robot Urdf

✓ Define the self-collision matrix

When you press the Regenerate Default Collision Matrix button, a collision-free matrix will be created. This information will be included in the MoveIt! package's SRDF file. Figure 3.21 shows an example of a layout

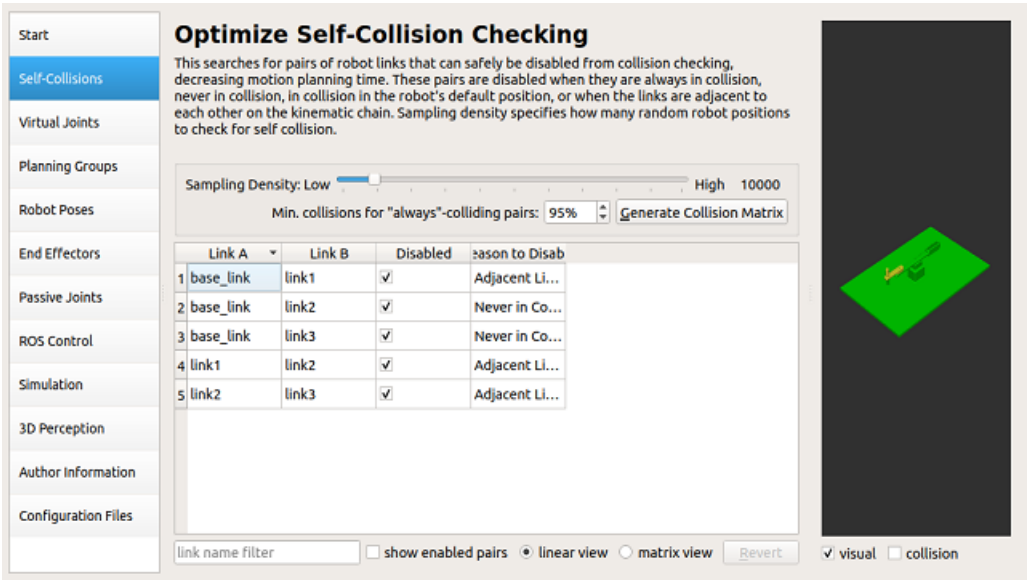


Figure 3.21: Self-collision checking tab

✓ Define virtual joints

Joints to connect the real robot to the world will be defined under the Virtual Joints tab. Because there is just one manipulator in this scenario, a virtual joint will sufficient. When the Add Virtual Joint Button button pressed the configuration window will pop-

up. Fixed base will be the virtual joint name, between base_link and world frame. Base_link is the child link while world frame is parent_link, and the joint type in here is fixed. Figure 3.22 shows an example of a layout.

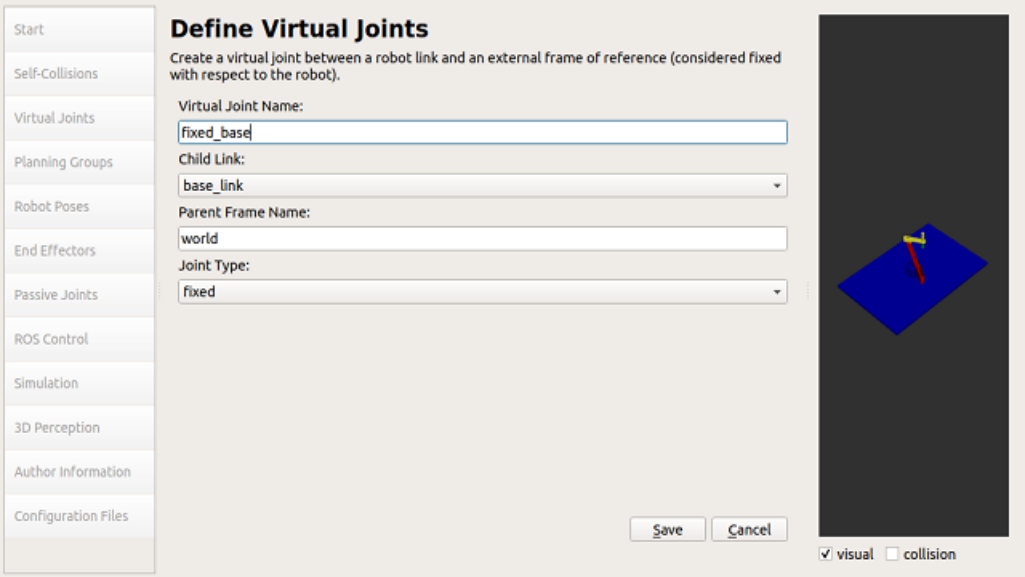


Figure 3.22: Virtual joint tab

- ✓ Define Planning Groups

Click the "Add Group" button on the "Planning Groups" tab.

Now make a new group named arm that uses the KDLKinematicsPlugin.

. Figure 3.23 shows an example of a layout.

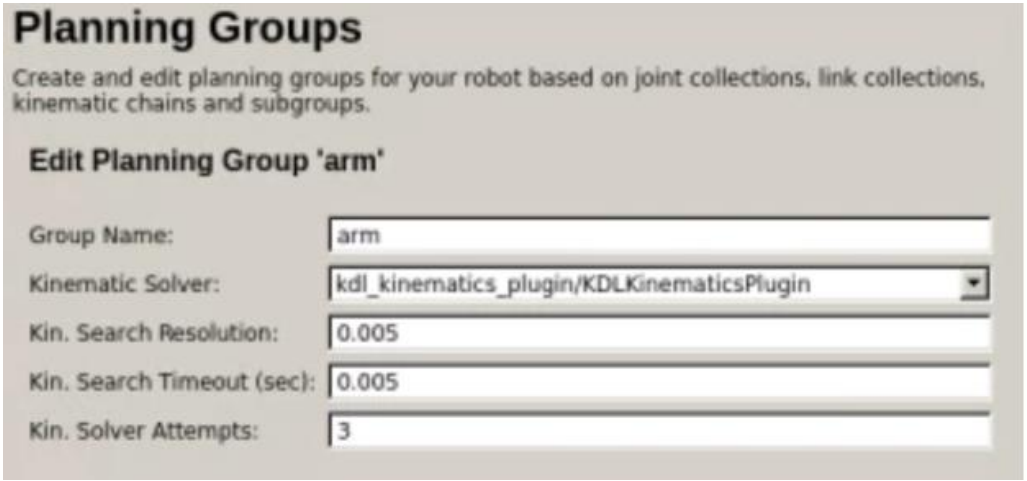


Figure 3.23: Planning group tab

Then, using the "Add Joints" button, pick (from the Available Joints box) all of the jo ints that make up the robot's arm, excluding the gripper.

Figure 3.24 shows an example of a layout.

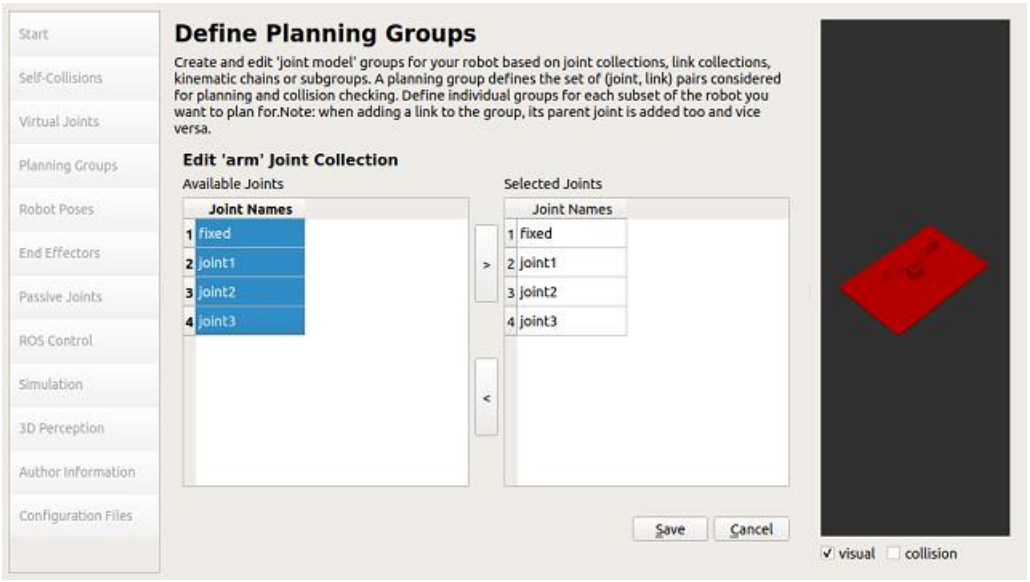


Figure 3.24: Add joints tab

Whenever all the joints selected, just click on the big "*>" button in order to move them to the* Selected Joints box. Figure 3.24 shows an example of a layout.

Finally, click the "Save" button to generate the following result:

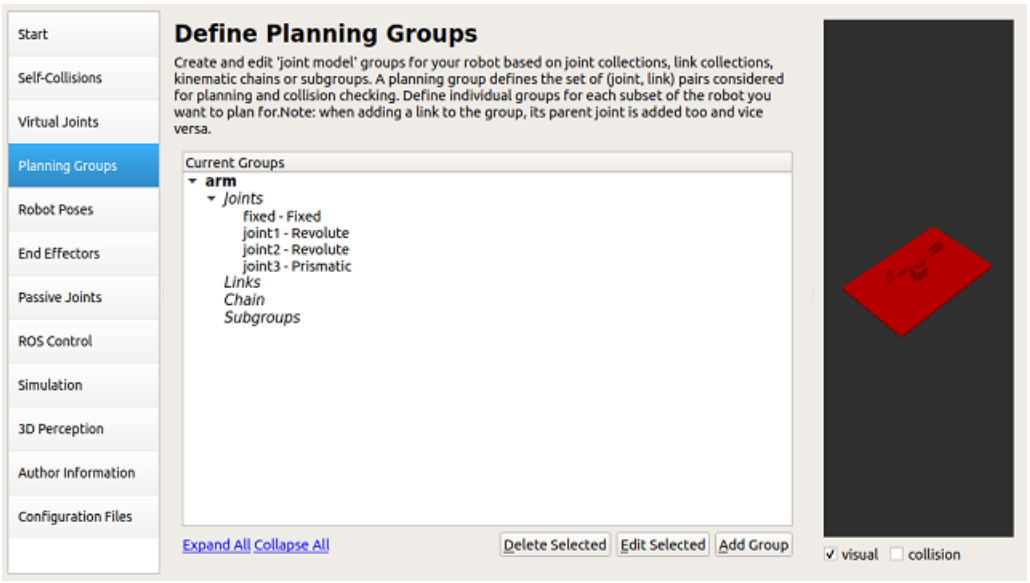


Figure 3.25: Planning group final appearance

We have now established a set of joints for Motion Planning, as well as the plugin that will be used to compute those plans.

- ✓ Define Robot Poses

Next g oto Robot Poses tab. Now, we will establish a few predetermined positions for our manipulator. And why are these poses predefined? MoveIt will save these poses (particularly the position and orientation of the manipulator joints) so that we may set them as targets with a few of clicks at any time. This is useful, for example, when our robot must frequently revisit specific stances. To generate the these poses. We will have to define the positions of the joints that will be related to each pose, as well as the planning group they will be related to.

The first pose will be named home, and its joint positions will be the following:

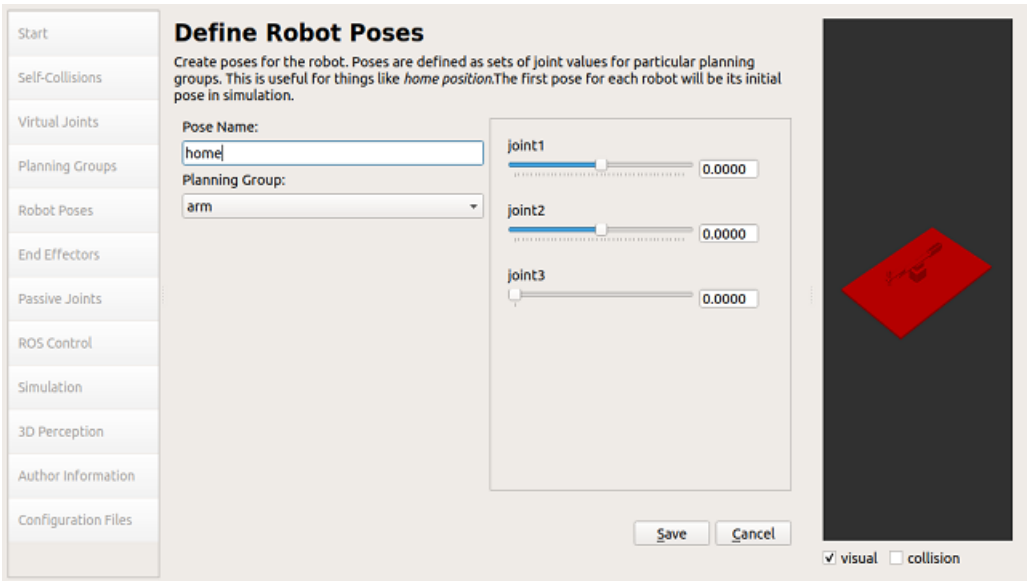


Figure 3.26: Robot poses home pose

The second pose will be named left, and its joint positions will be the following:

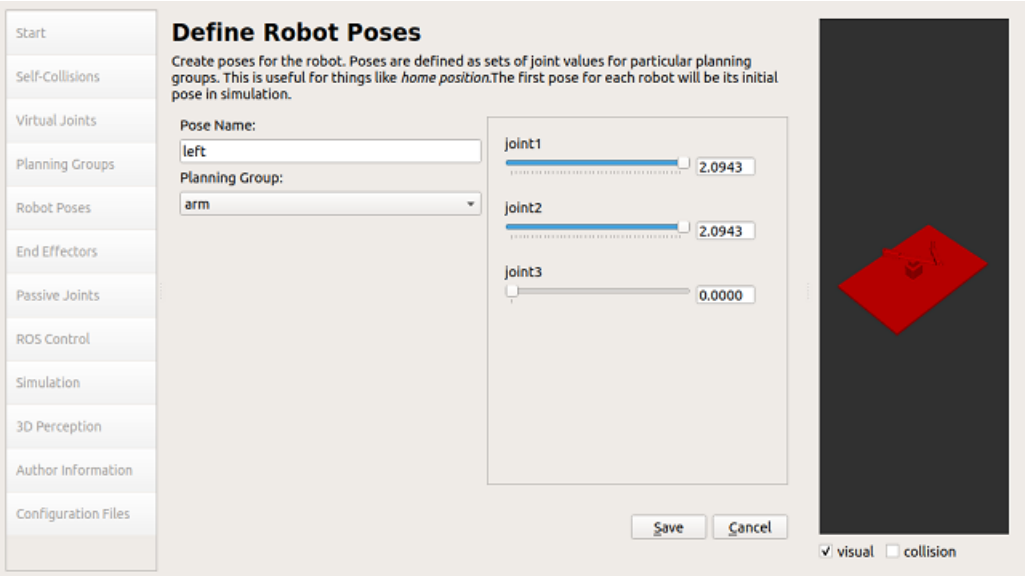


Figure 3.27: Robot poses left pose

The second pose will be named right, and its joint positions will be the following:

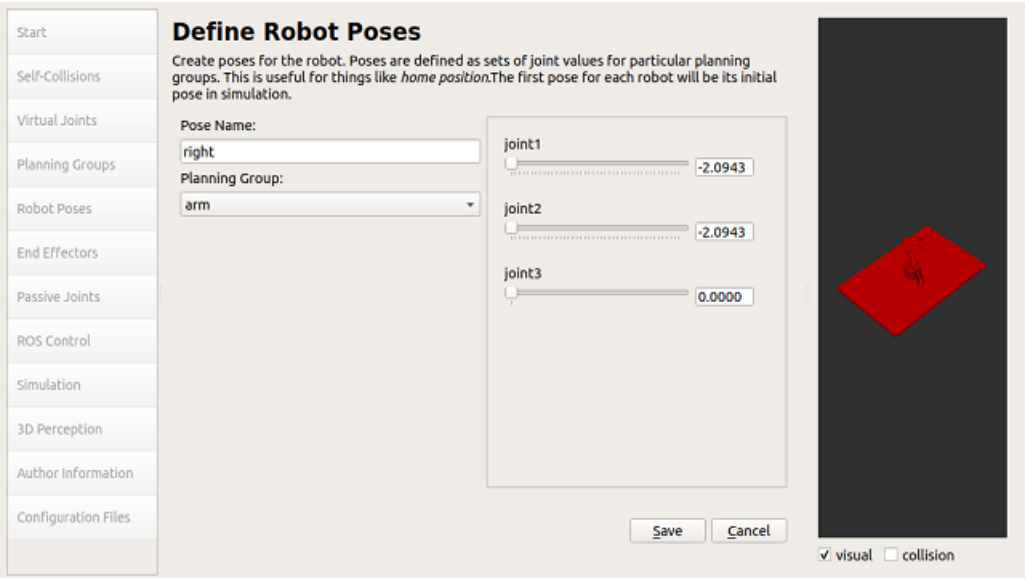


Figure 3.28: Robot poses left pose

Eventually it will be as it appears in Figure 3.29.

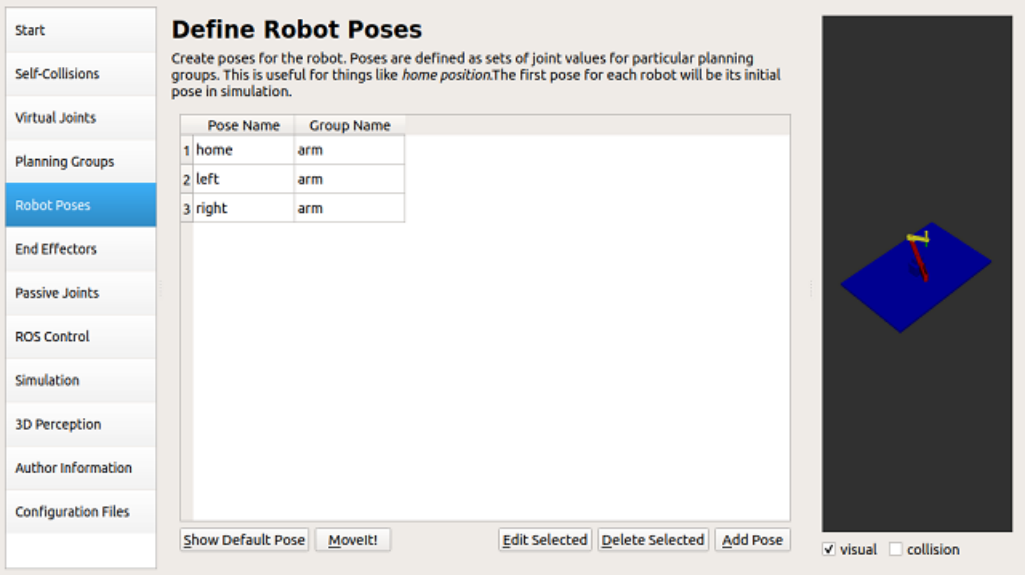


Figure 3.29: Robot poses final appearance

- ✓ Define End Effector

The next step is to configure the robot's End-Effector. Simply go to the End Effectors tab and click the "Add End Effector" button to accomplish this. We'll give our End Effector gripper a name.

The End Effector Group is the Planning group that will control it, which in this case was previously defined as the gripper group.

The Parent Link is the arm lnik to which the end effector is connected.

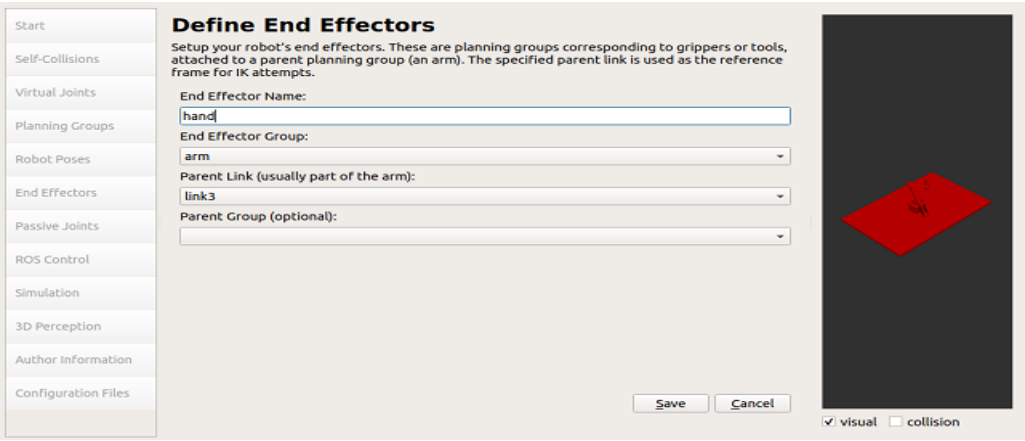


Figure 3.30: End effector tab

3.17.4.2 Setup ROS Controllers

Next, we will define the ROS Controllers that will enable interaction (and motion execution) within the Gazebo simulation.

Initially, we will need to collect data for this purpose.

We need to know the name of the ROS Controllers that will enable us to move the arm in the simulation.

Execute the following command for this purpose:

- $ rostopic list | grep arm

With this filter, we can quickly identify the following group of topics:

```
/rbkairos/arm_controller/command
/rbkairos/arm_controller/follow_joint_trajectory/cancel
/rbkairos/arm_controller/follow_joint_trajectory/feedback
/rbkairos/arm_controller/follow_joint_trajectory/goal
/rbkairos/arm_controller/follow_joint_trajectory/result
/rbkairos/arm_controller/follow_joint_trajectory/status
/rbkairos/arm_controller/state
```

Figure 3.31: ROS controller type

However, with this data, we are already prepared to configure the ROS Controllers for our MoveIt package. Basically, we now know that the name of the control is arm_controller, and that its type is FollowJointTrajectory. Also, we can see that it has a namespace with the name rbkairos, but we will deal with this later.

Then, click on the Add Controller button and fill in the data as follows:

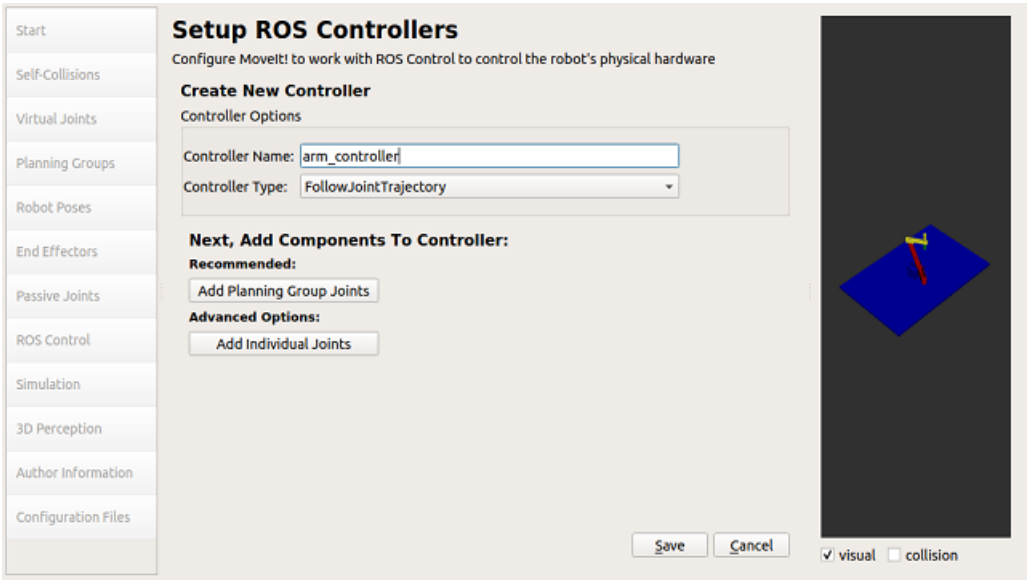


Figure 3.32: Setup ROS controller

The controller's name is arm controller and its type is FollowJointTrajectory, as displayed.

To associate the controller with a planning group, click the Add Planning Group Joints button. Here, just select the arm group which was created before.

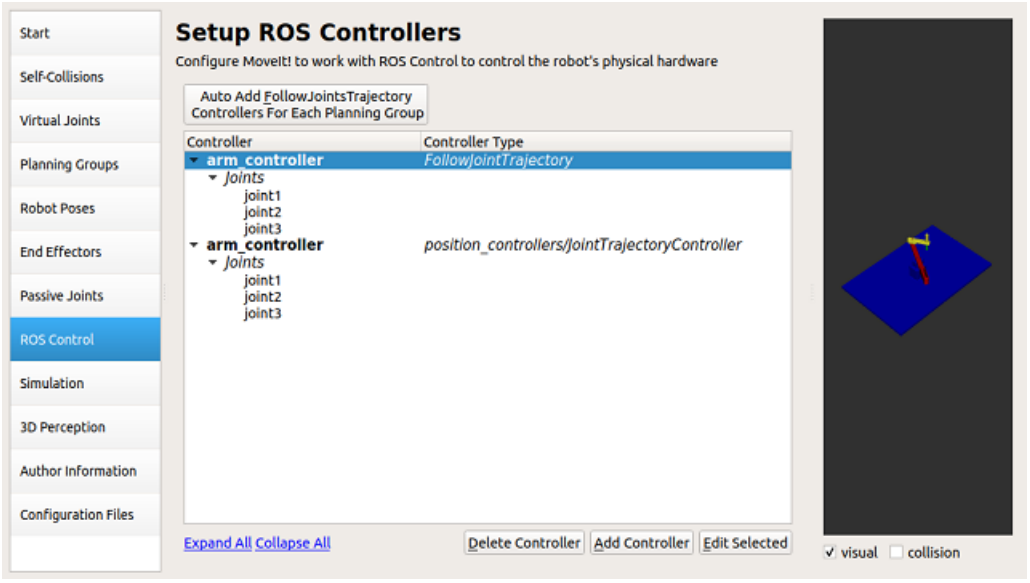


Figure 3.33: ROS controller final apperance

Click the Save button. We have just defined the ROS Controllers that will allow the MoveIt package to plan and execute the motions on the simulated robot.

✓ Generate the MoveIt package

There is one last step we will need to complete before generating the MoveIt package. We need to fill the Author Information. We complete the fields with my name and e-mail, like in the image below:

Figure 3.34: Author Information

Lastly, navigate to the "Configuration Files" page and select "Browse." Create a new directory in the catkin ws/src directory and name it scara moveit config. Select the directory that was just created.

Now, click the "Package Generation" button. If all goes well, the following will occur:

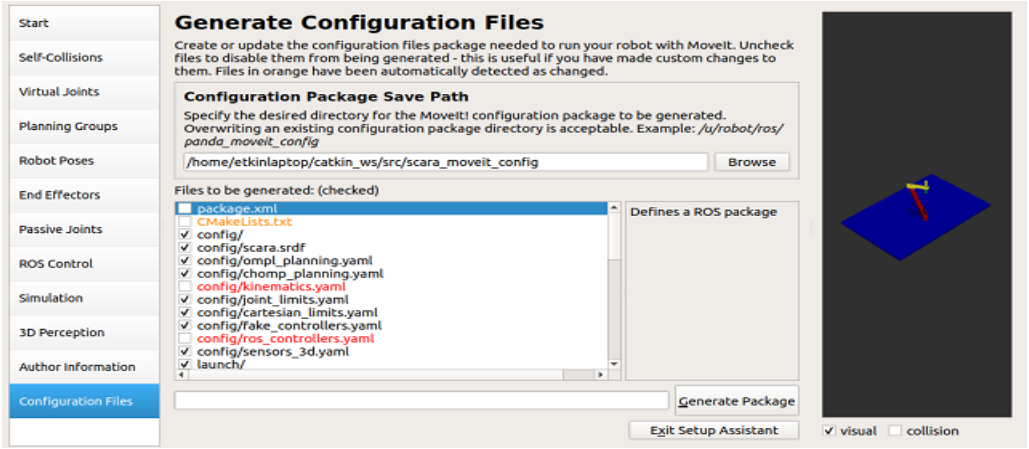


Figure 3.35: Generate configuration files

And that concludes it! For our articulated robot, we designed a MoveIt package.

3.17.4.3 Basic Motion Planning

Execute the following command in order to start the MoveIt Rviz demo environment.

- $ roslaunch my_moveit_config demo.launch

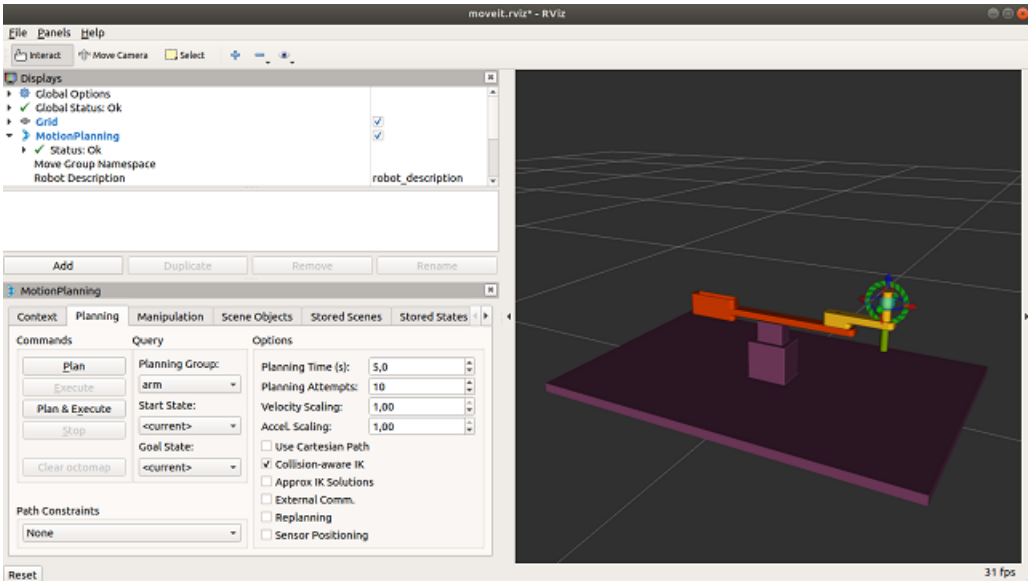


Figure 3.36: Scara in Moveit for basic motion planning

At the query section, in the Goal State, you can choose the random_valid option in order to set a random valid goal for the manipulator robot.

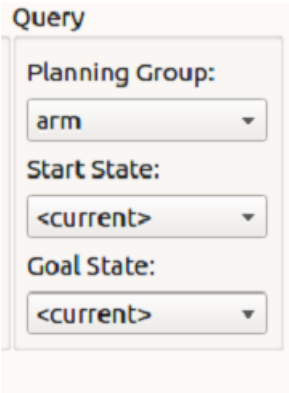


Figure 3.37: Selection of start and goal state

Alternately, we can select any of the Robot Poses we defined within the Moveit package. After modifying the Goal State, the robot scene will be updated to fit the changing position.

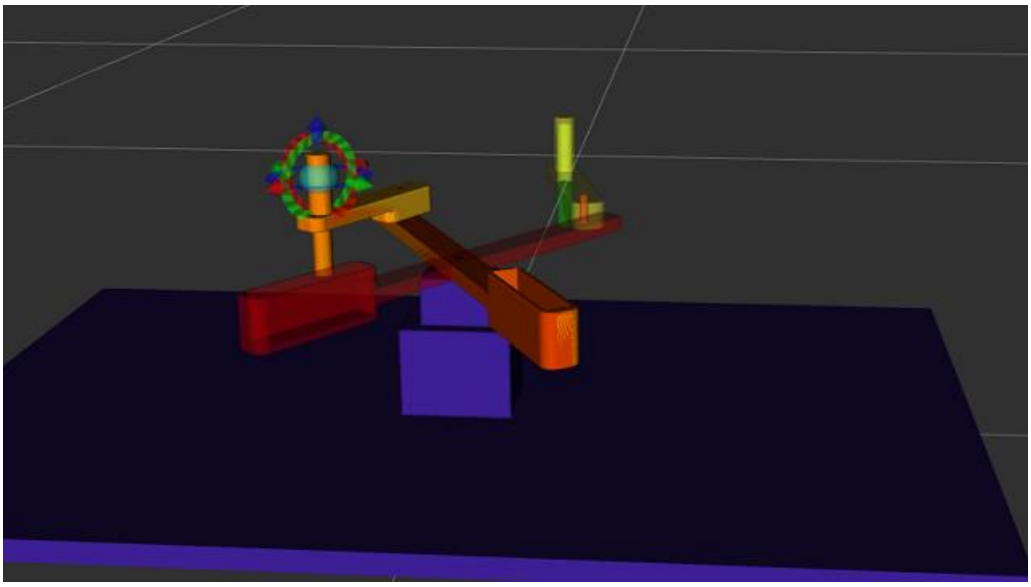


Figure 3.38: Moveit motion planning

In the "Commands" section, we can now click the "Plan" button. The robot will begin to plan a trajectory to reach that point.

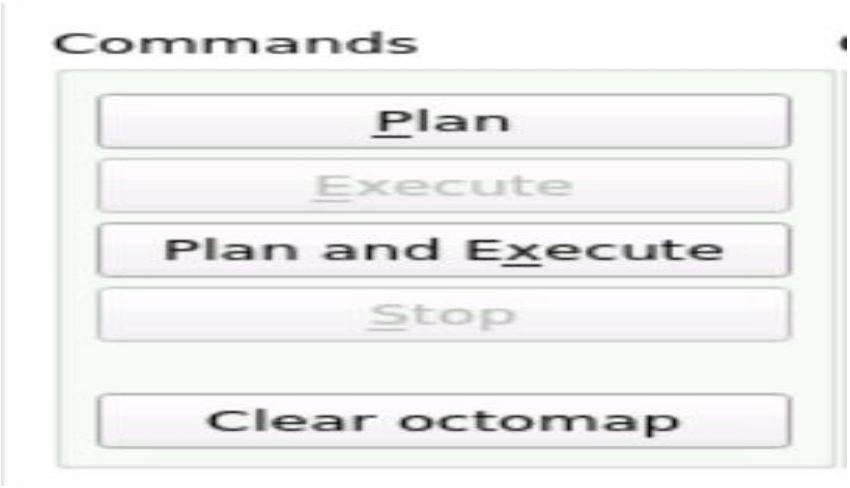


Figure 3.39: Moveit plan and execute buttons

Finally, click on the "Execute" button, the robot will execute that trajectory.

We can repeat this process with other goal states. Additionally, select and deselecting the various display settings in the "Displays" section.

- ✓ Moving the real robot

In the previous basic motion section we have moved the robot inside the MoveIt Rviz application. This is very useful because we can do many tests without worrying about any damage. However, the final goal will always be to move the real robot, right?

The MoveIt package we have created is able to provide the necessary ROS services and actions in order to plan and execute trajectories, but it is not able to pass these trajectories to the real robot. All the kinematics we have been performing were executed in an internal simulator that MoveIt provides. In order to communicate with the real robot, it will be necessary to do a couple of modifications to the MoveIt package you created in the previous exercise.

In order to see what we need to change in your MoveIt package, just follow the following information.

Firstly go to scara_moveit_config package. Enter the config folder and see the ros_controllers.yaml file that has been created by the MoveIt Setup Assistant.

Let's have a look at the bottom part of the file, where we define the controller_list. Change name of controller, action_ns, and type which shown as below.

```
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    default: True
    type: FollowJointTrajectory
    joints:
      - joint1
      - joint2
      - joint3
```

```
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    default: True
    type: FollowJointTrajectory
    joints:
      - joint1
      - joint2
      - joint3
arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
    - joint1
    - joint2
    - joint3
  gains:
    joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint2:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint3:
      p: 100
      d: 1
      i: 1
```

Figure 3.40: Scara Moveit controller list

Now we have solved the issue related to the configuration of the Action Server, but there's still one last thing we need to do.

Basically, we are going to create a new launch file that will start all the required elements that MoveIt needs in order to be able to control our simulated and physical robot. You can name this new file *my_planning_execution.launch*, and it should look like this:

```
<launch>

 <include file="$(find scara_moveit_config)/launch/planning_context.launch" >

  <arg name="load_robot_description" value="true" />

 </include>
```

```
 <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
   <param name="/use_gui" value="false"/>
   <rosparam param="/source_list">[/joint_states]</rosparam>
 </node>
 <include file="$(find scara_moveit_config)/launch/move_group.launch">
   <arg name="publish_monitored_planning_scene" value="true" />
 </include>
 <include file="$(find scara_moveit_config)/launch/moveit_rviz.launch">
   <arg name="rviz_config" value="$(find scara_moveit_config)/launch/moveit.rviz"/>
 </include>
</launch>
```

```
<launch>

  <include file="$(find  scara_moveit_config)/launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
  type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam param="/source_list">[/joint_states]</rosparam>
  </node>

  <include file="$(find  scara_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <include file="$(find  scara_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>

</launch>
```

Figure 3.41: Scara new planning execution launch file

Here we are starting some launch files we need in order to set up the MoveIt environment. Focus a moment on the joint_state_publisher node that is being launched.

If you do a rostopic list again, there is a topic called /joint_states. The states of the joints of the simulated and physical robot are published in this topic. So, we need to

put this topic into the /source_list parameter, so that MoveIt can know where the robot is at each moment. Let's execute this newly-created launch file, my planning execution.launch, using the following command:

- $ roslaunch scara_moveit_config my_planning_execution.launch

Once MoveIt loads, we will see that the robot is initialized in a different position than in the previous demo. This is because MoveIt is actually getting the data from the simulated and physical robot, which is in this initial position.

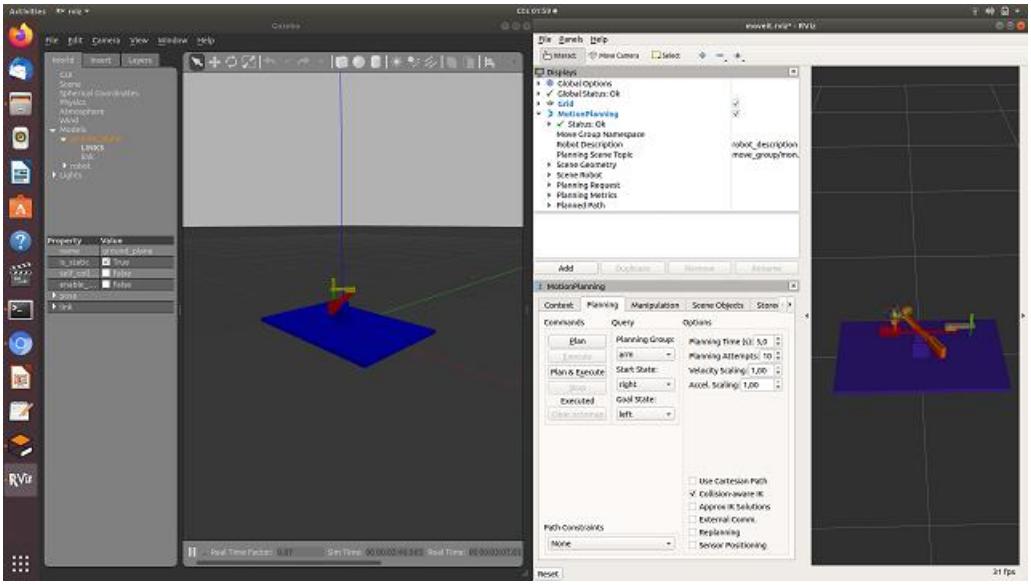


Figure 3.42: Scara Gazebo Moveit interaction

Now, plan a path to the home position. Once the trajectory has been planned, click the Execute button to execute it with the simulated and physical robot. We should observe our robot performing similar actions:

3.17.5 Ur5 package

The urdf and xacro files of the supported hardware industrial robots mentioned in chapter 3, section 9.2, Figure 3.11 can be found on the github web page of ROS-Industrial. The urdf.xacro file of the ur5 industrial robot is located in the ros-industrial repositories universal_robot folder [59]. The ur5_package is created by repeating the same steps in the scara_package mentioned above, using the

universal_robot.urdf.xacro file and stl files that are ready-made from here. The steps to create this package are briefly as follows:

Open a terminal and type the following commands to make this package:

- $ cd catkin_ws/src/
- $ catkin_create_pkg ur5_package rospy roscpp urdf xacro rviz gazebo_ros std_msgs

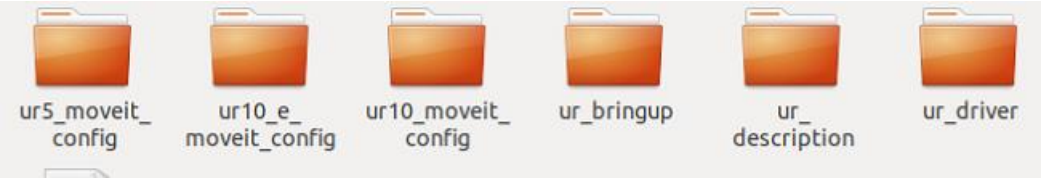


Figure 3.43: Ur5 Moveit package

The config folder will contain 1 configuration files:

- ✓ ros_controllers.yaml

Information provided by ros_controllers.yaml will be:

- ✓ Name of controller(joint_state_controller, arm_controller)
- ✓ Type of controller
- ✓ (joint_state_controller/ JointStateController,position_controllers/JointTrajectoryController)
- ✓ Name of the joints (joint1, joint2, joint3)
- ✓ Pid values of each joint

The launch folder will contain 8 files:

- ✓ rviz_urdf.launch
- ✓ gazebo_urdf.launch
- ✓ rviz_all.launch

- ✓ urdf_config.rviz

The rviz urdf.launch file is launched and transferred to the rviz visualization environment of the robot model. Here it can be checked whether the joints and links are correct.

The gazebo urdf.launch file is launched and transferred to the gazebo simulation environment of the robot model. Here, it can be checked whether the physical properties work correctly.

The rviz_all.launch file is launched and transferred to the rviz visualization environment of the robot model, by adding a controller to the gazebo simulation environment. Here, the joint angles found from the trajectory algorithm written by the thesis author are pressed on the joints, respectively, and the robot in the view, the robot in the gazebo simulation environment is shown to follow the simultaneous trajectory.

The urdf_config.rviz file ensures that the plugins to be added in rviz are pre-registered and added to each launch file in a ready-made form, so that the additions are ready in rviz.

The meshes folder contains STL files of the robot taken from ROS-Industrial web page. These STL files are used inside urdf file to display the simulation robot looks like real robot.

The script folder contains python files. These are:

- ✓ Joint_states_to_gazebo.py
- ✓ Solidworks_to_excel_to_path.py
- ✓ pub_joint_values.py

Joint_states_to_gazebo.py python code is used to transfer joint state values from rviz to the move the robot in the gazebo.

Solidworks_to_excel_to_path.py and pub_joint_values.py python codes give the necessary joint angles for trajectory tracking and transfers them to rviz, gazebo simulation robot with joint_state_publisher as in rviz.

The urdf folder contains the urdf file which are the descriptions of the robot.

- ✓ ur5_robot.urdf.xacro

3.17.6 Ur5 Robot MoveIt Config Package and Motion

3.17.6.1 Building a MoveIt package

The steps for scara_moveit_config in chapter3, section 19.4.1 are done exactly in the Ur5 robot. Also, the perception tab in moveit_setup_asssistant for the Ur5 robot is handled differently than the scara_moeveit_config for the image processing technique. Roscore has to be launched, before run the setup assistant tool. To do that open a new terminal and write roscore:

- ➢ $roscore

Afterthat, run the setup assistance by typing the following in another terminal.

- ➢ $ roslaunch moveit_setup_assistant setup_assistant.launch

Figure 3.44: Moveit setup assistant for Ur5

✓ Loading Ur5 Robot URDF file

Select the xarco file.

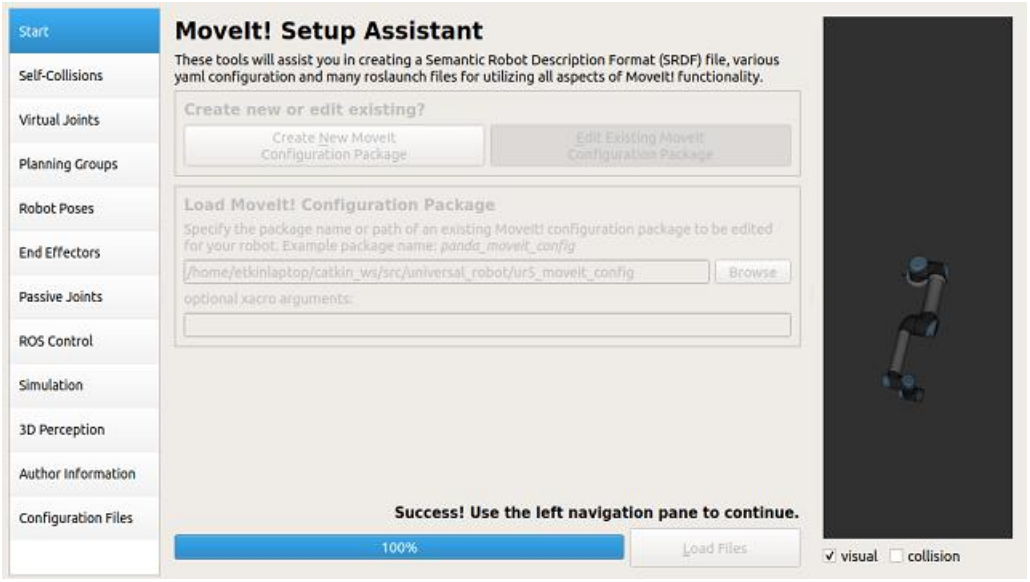


Figure 3.45: Loading Ur5 Urdf

✓ Define the Self-collision Matrix

Click generate collision matrix.

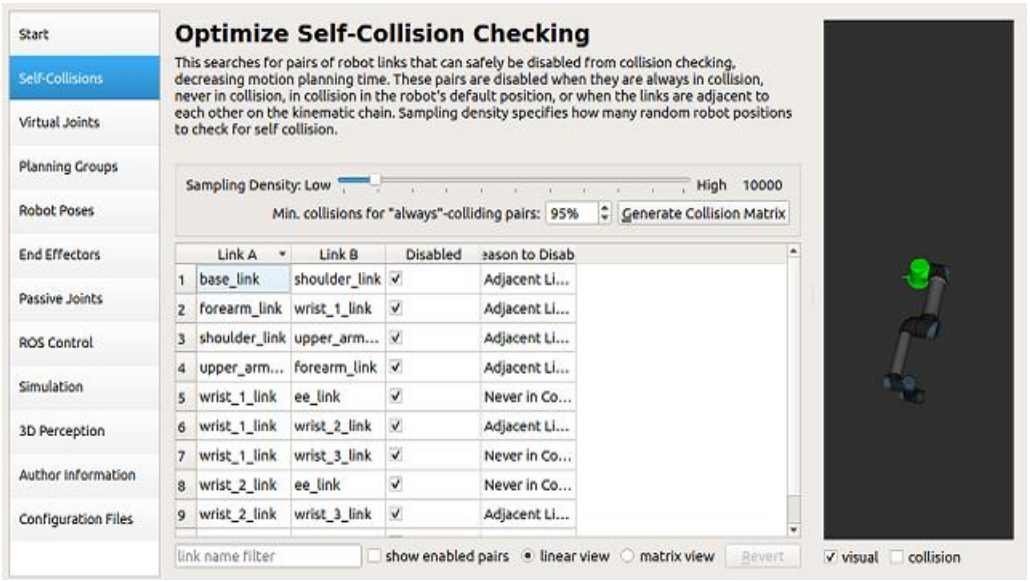


Figure 3.46: Self collision checking

✓ Define virtual joints

Click add virtual joint then edit it as the following image

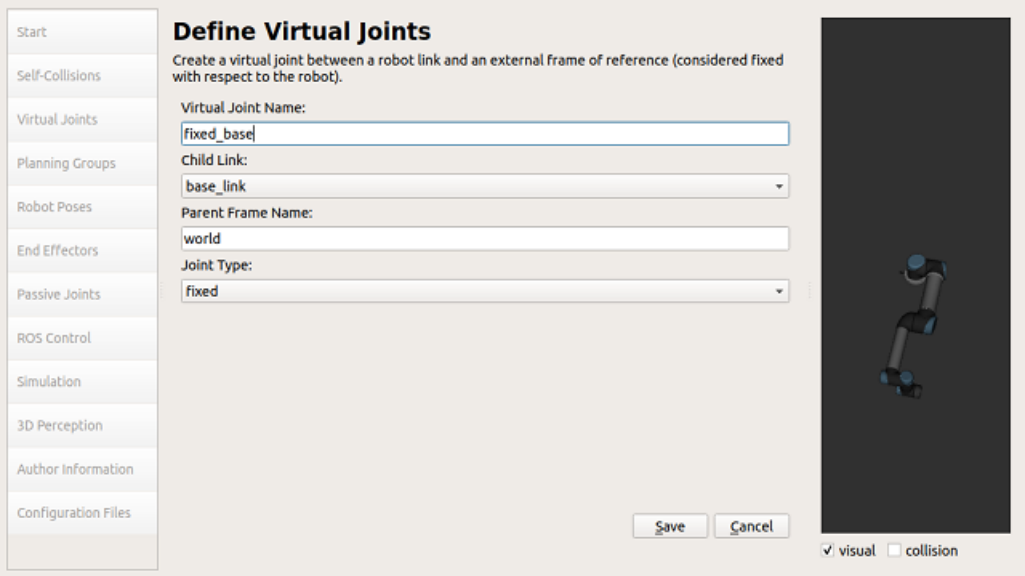


Figure 3.47: Virtual joint tab

✓ Define Planning Groups

Click the "Add Group" button on the "Planning Groups" tab.

Now make a new group named arm that uses the KDLKinematicsPlugin.

Figure 3.48 shows an example of a layout.

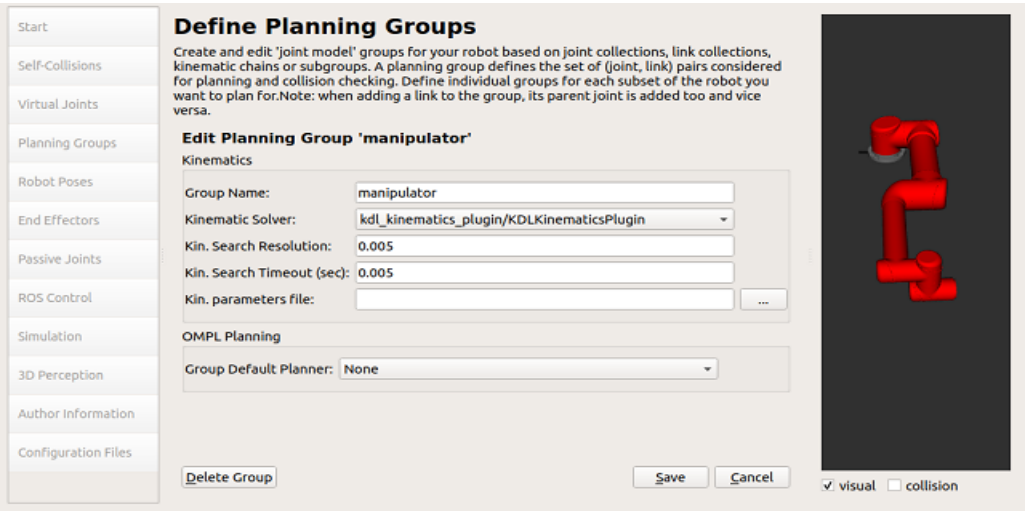


Figure 3.48: Kinematics solver selection

Then, using the "Add Joints" button, pick (from the Available Joints box) all of the jo ints that make up the robot's arm, excluding the gripper.

Figure 3.49 shows an example of a layout.

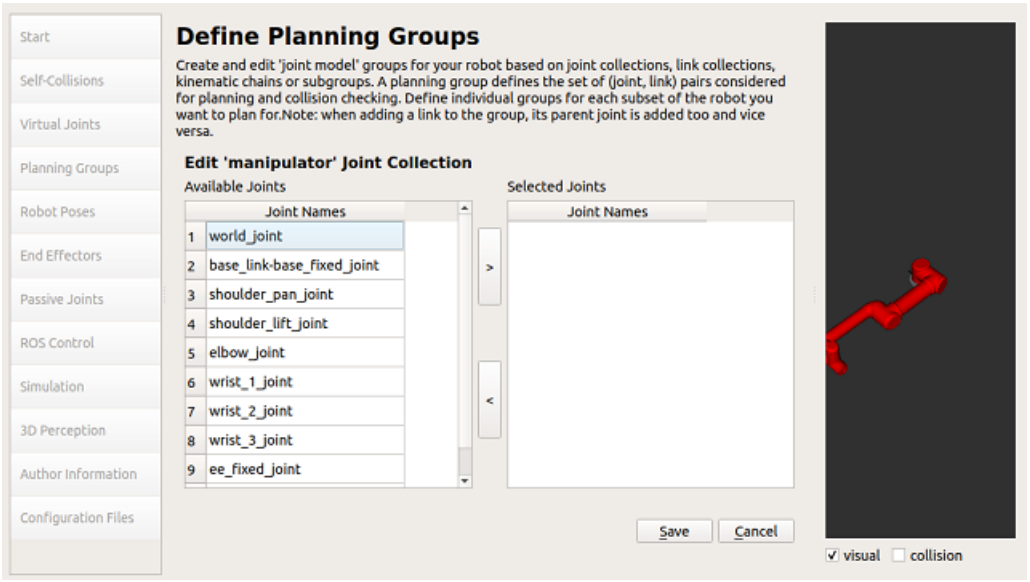


Figure 3.49: Planning group

Whenever all the joints selected, just click on the big "*>" button in order to move them to the* Selected Joints box. Figure 3.50 shows an example of a layout.

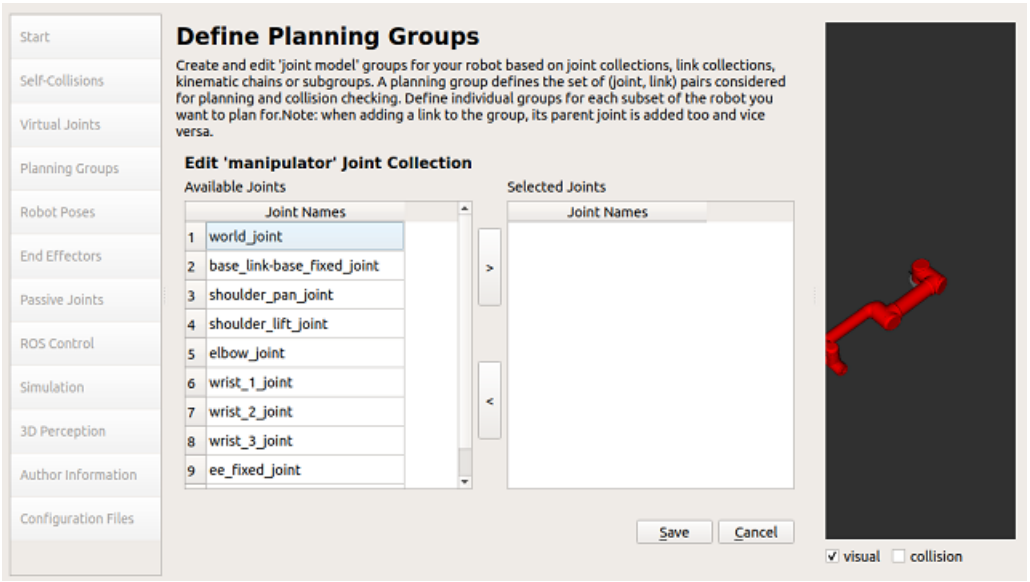


Figure 3.50: Planning group joints selected

Finally, click the "Save" button to generate the following result:

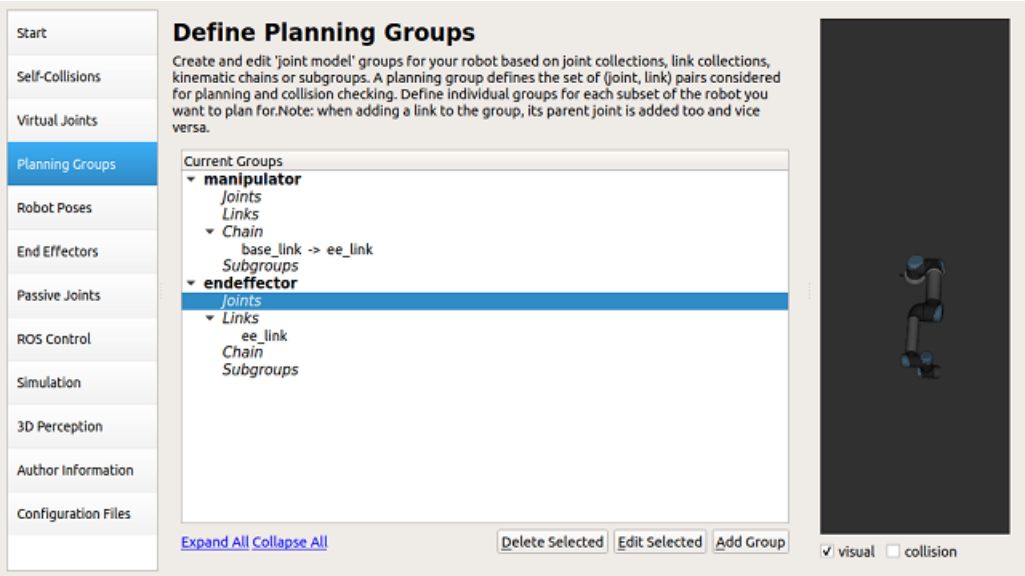


Figure 3.51: Planning group final apperance

We have now established a set of joints for Motion Planning, as well as the plugin that will be used to compute those plans.

✓ Define Robot Poses

Add the following poses. The first pose will be named home, and its joint positions will be the following:

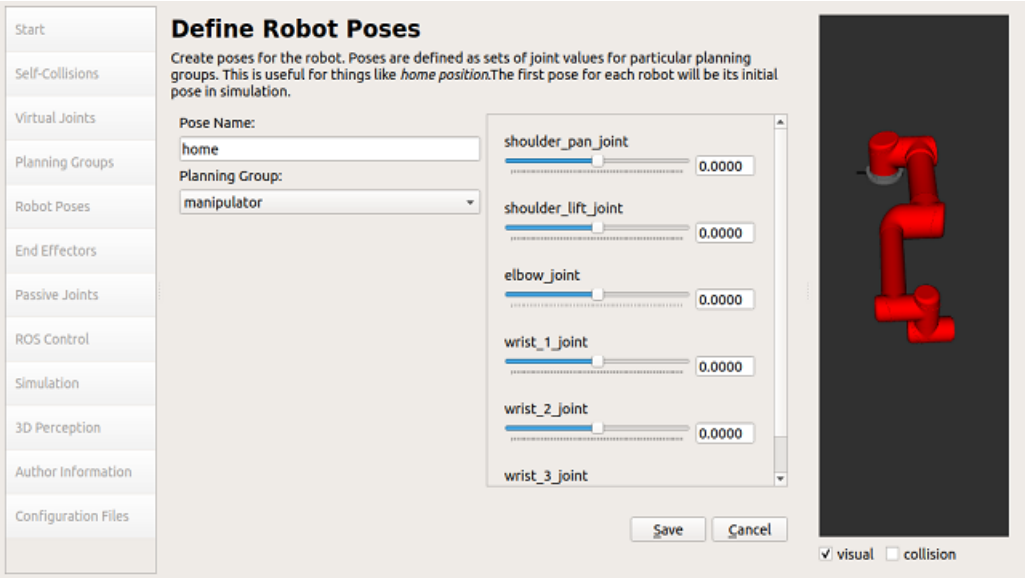


Figure 3.52: Ur5 home pose

The second pose will be named start, and its joint positions will be the following:

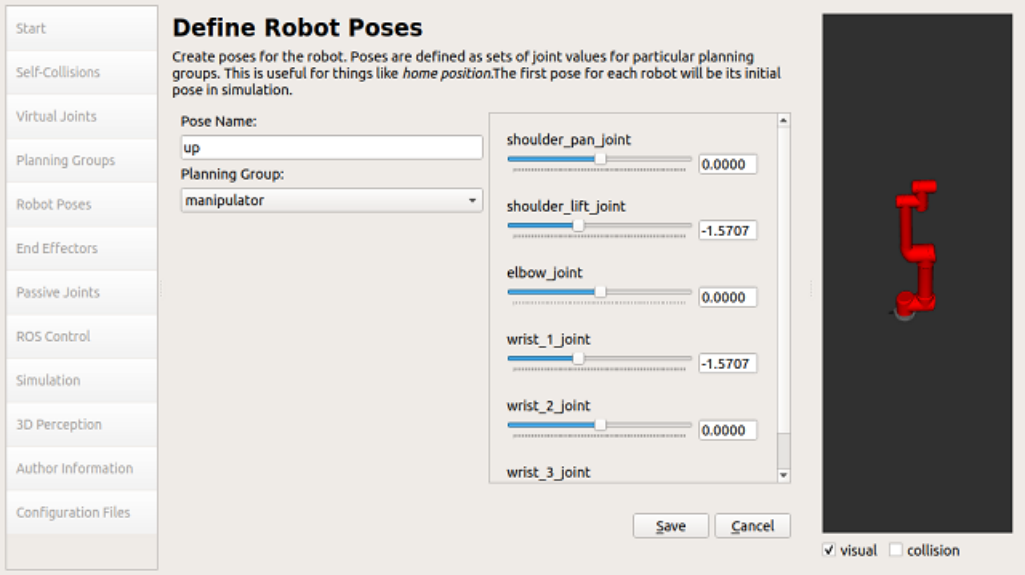


Figure 3.53: Ur5 start pose

Eventually it will be as it appears in Figure 3.54.

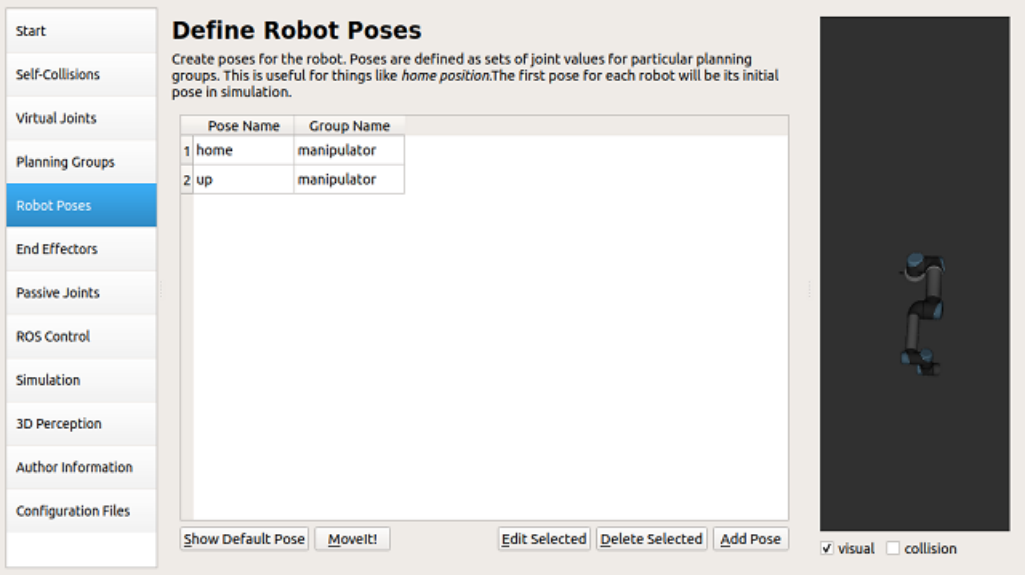


Figure 3.54: Ur5 Poses

✓ Define End Effector

The next step is to configure the robot's End-Effector. Simply navigate to the End Effectors tab and click the "Add End Effector" button to accomplish this. We'll give our End Effector gripper a name.

The End Effector Group is the Planning group that will control it, which in this case was previously defined as the gripper group.

The Parent Link is the arm link to which the end effector is connected.

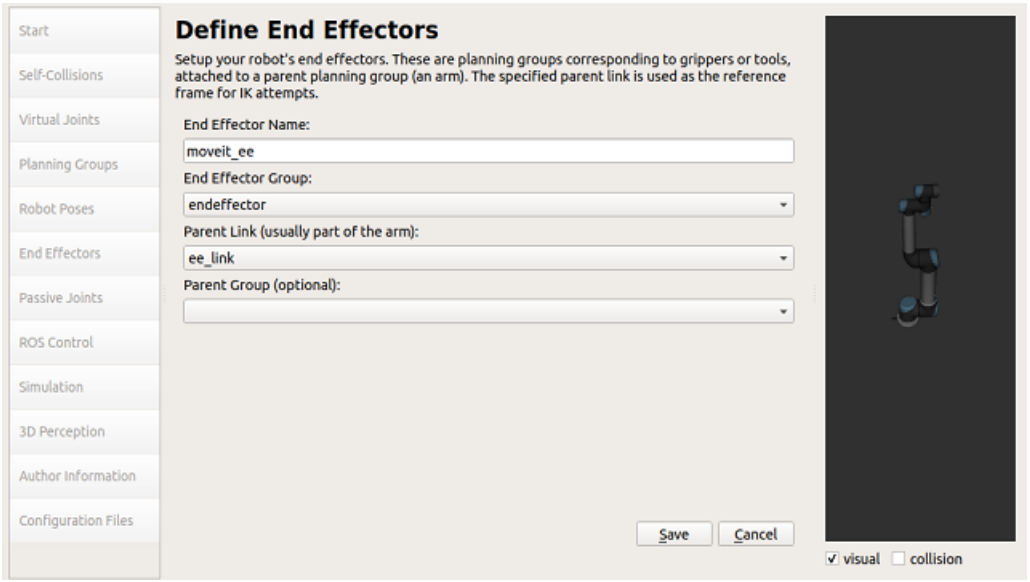


Figure 3.55: Ur5 end effector

✓ Setup ROS Controllers

Next, we will define the ROS Controllers that will enable interaction (and motion ex ecution) within the Gazebo simulation.

Then, click on the Add Controller button and fill in the data as follows:

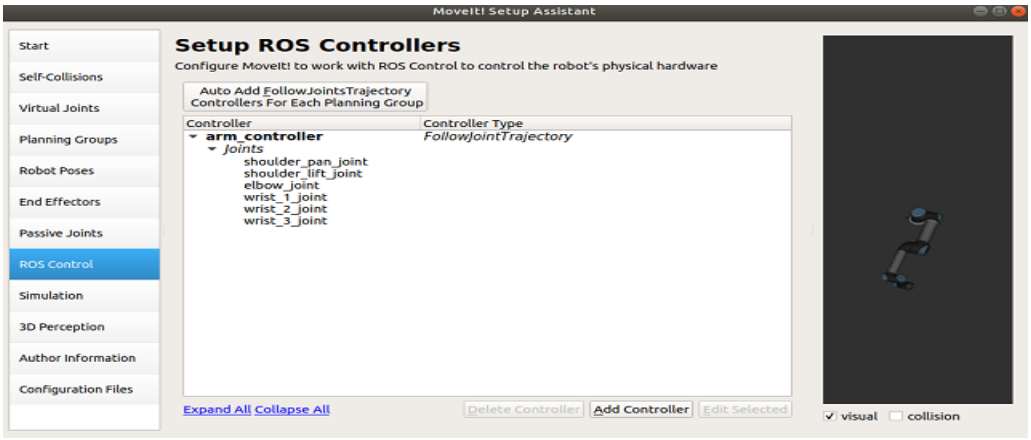


Figure 3.56: Setup ROS controller

The controller's name is arm controller and its type is FollowJointTrajectory, as displayed.

To associate the controller with a planning group, click the Add Planning Group Joints button. Here, just select the arm group which was created before.

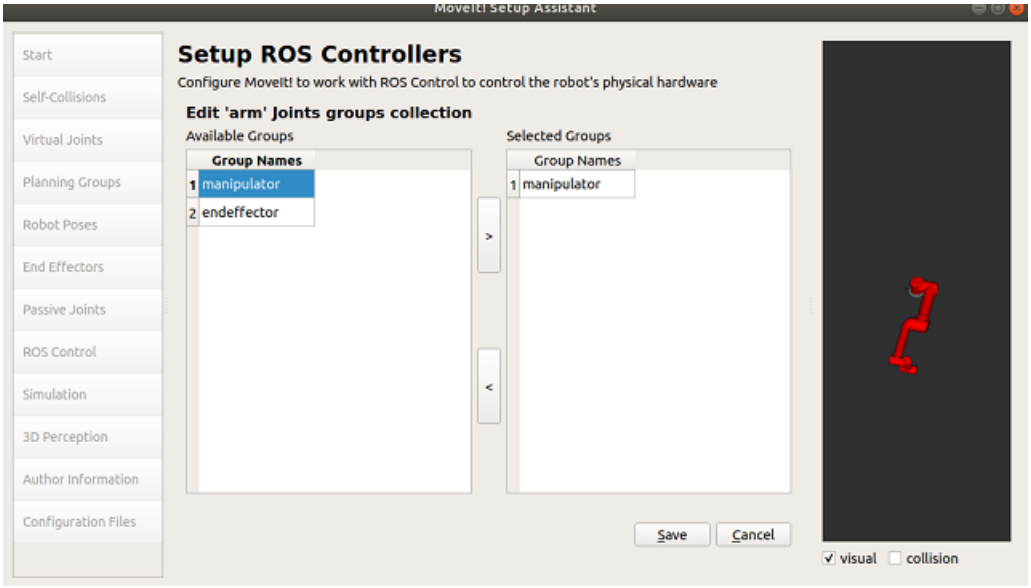


Figure 3.57: ROS controller creation

Click the Save button. We have just defined the ROS Controllers that will allow the MoveIt package to plan and execute the motions on the simulated robot.

- ✓ Generate the MoveIt package

Go to author information tab. Write name mail.

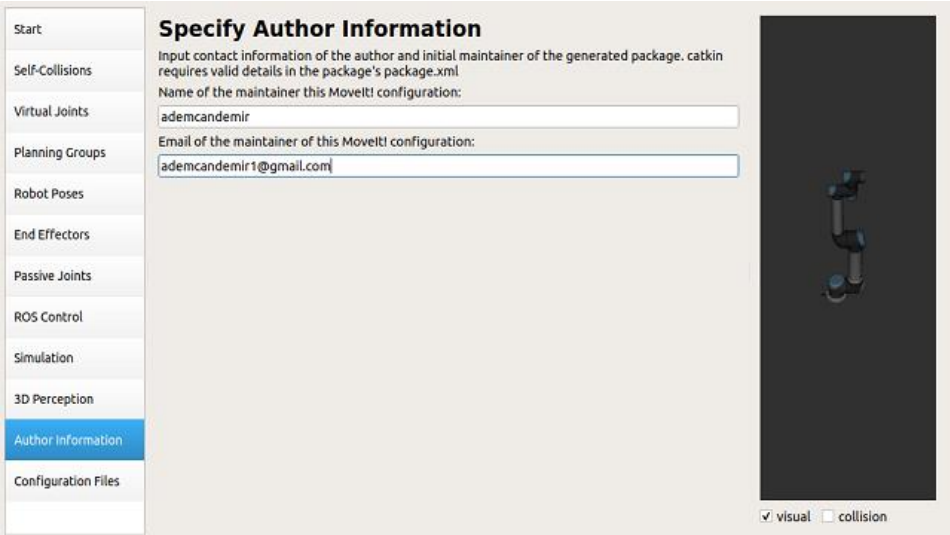


Figure 3.58: Author information

Lastly, navigate to the "Configuration Files" page and select "Browse." Create a new directory in the catkin ws/src directory and name it ur5 moveit config. Select the directory that was just created.

Now, click the "Package Generation" button. If all goes well, the following will occur:

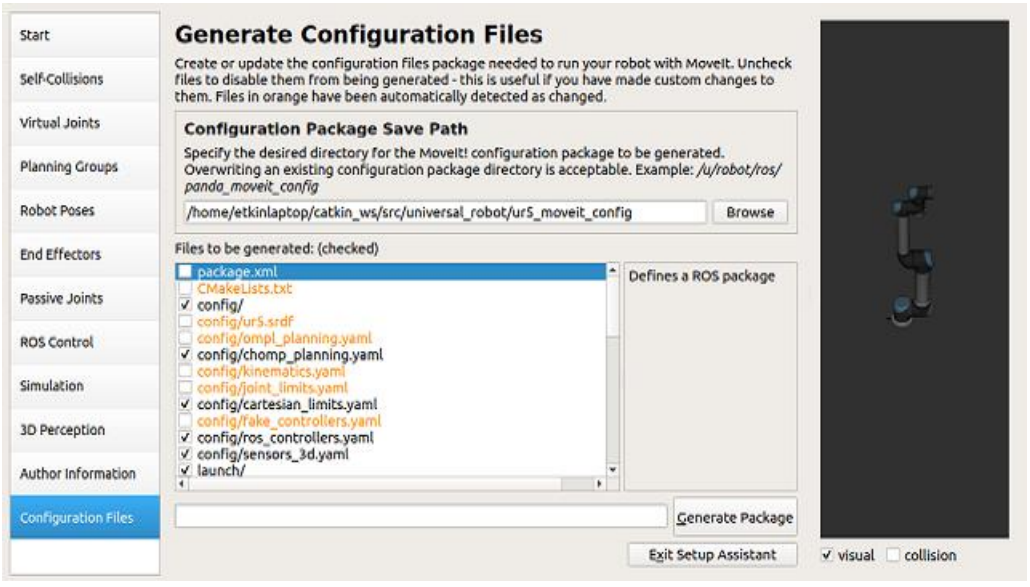


Figure 3.59: Ur5 configuation files tab

And that's it! We created a MoveIt package for our articulated robot.

3.17.6.2 Basic Motion Planning

Execute the following command in order to start the MoveIt RViz demo environment.

- $ roslaunch my_moveit_config demo.launch

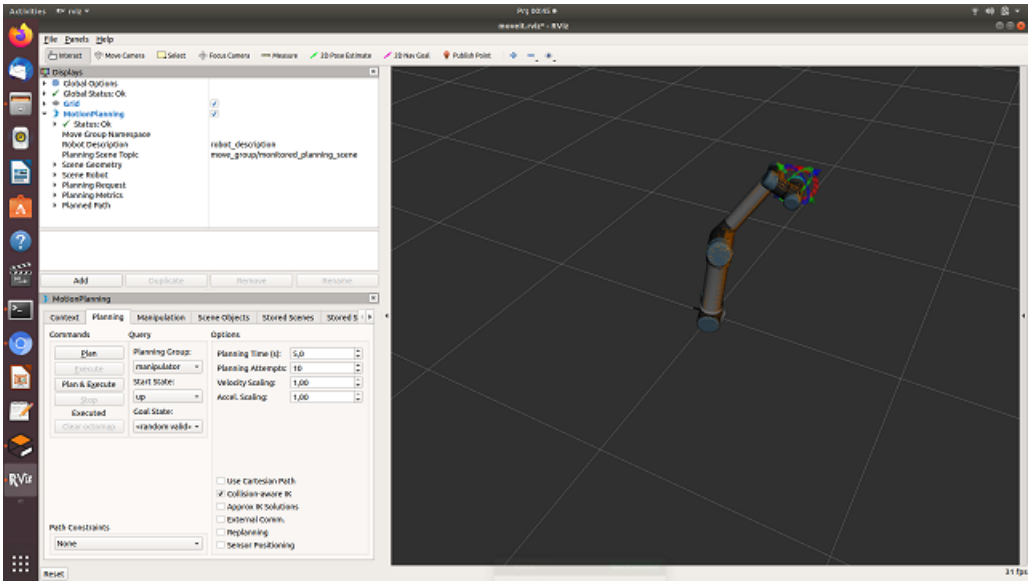


Figure 3.60: Ur5 in Moveit for basic motion planning

At the query section, in the Goal State, you can choose the *random_valid* option in order to set a random valid goal for the manipulator robot.

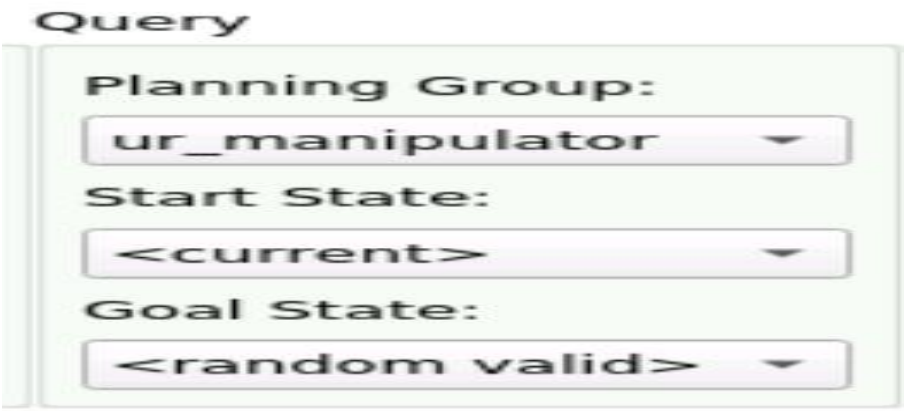


Figure 3.61: Ur5 Selection of start and goal state

Alternately, we can select any of the Robot Poses we defined within the Moveit package. After modifying the Goal State, the robot scene will be updated to fit the changing position.

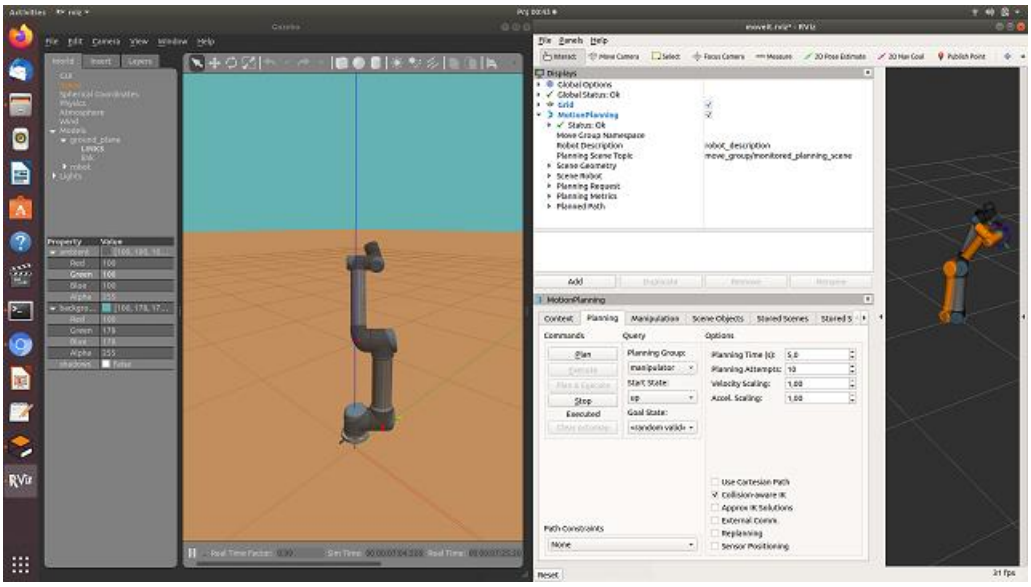


Figure 3.62: Ur5 Moveit motion planning

In the "Commands" section, we can now click the "Plan" button. The robot will begin to plan a trajectory to reach that point.

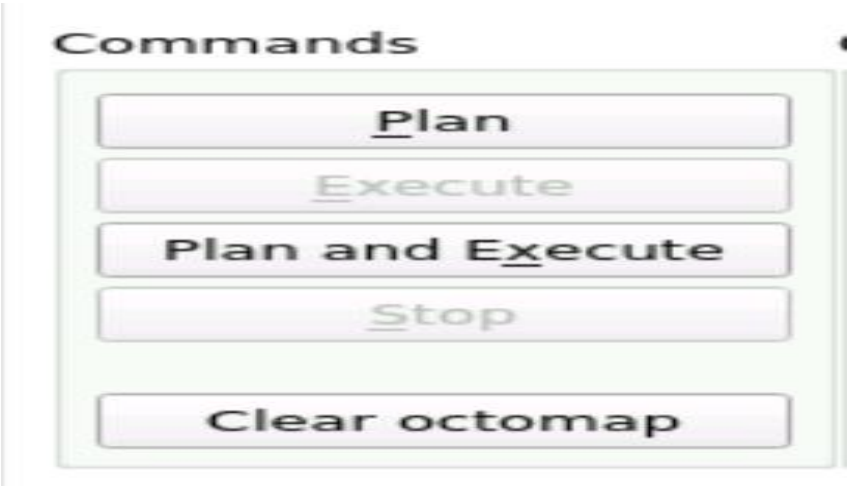


Figure 3.63: Ur5 Moveit plan and execute buttons

Finally, click on the "Execute" button, the robot will execute that trajectory.

We can repeat this process with other goal states. Additionally, select and deselecting the various display settings in the "Displays" section.

- ✓ Moving the Simulation Robot

In the last basic motion part we have moved the robot inside the MoveIt Rviz application. This is incredibly handy because we can run multiple experiments without worrying about any damage. However, the final goal will always be to move the simulation robot.

MoveIt provides the essential ROS services and actions to plan and execute trajectories, but it cannot transfer them to the simulation robot. We performed all kinematics in MoveIt's internal simulator. To communicate with the real robot, modify the MoveIt package from the previous exercise.

First, go to ur5_moveit_config directory. MoveIt Setup Assistant created ros controllers.yaml file in the config folder.

We define the controller list at the bottom of the file. Change controller, action ns, and type as given.

```
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    default: True
    type: FollowJointTrajectory
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint
```

Figure 3.64: Ur5 Moveit controller list

Now we have solved the issue related to the configuration of the Action Server, but there's still one last thing we need to do.

Basically, we are going to create a new launch file that will start all the required elements that MoveIt needs in order to be able to control our simulated and physical robot. You can name this new file my_planning_execution.launch, and it should look like this:

```
<launch>

  <include file="$(find myur5_moveit_config)/launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>

  <node name="joint_state_publisher" pkg="joint_state_publisher"␣
↪type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam param="/source_list">[/joint_states]</rosparam>
  </node>

  <include file="$(find myur5_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <include file="$(find myur5_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>

</launch>
```

Figure 3.65: Ur5 Moveit controller list

Here we are starting some launch files we need in order to set up the MoveIt environment. Focus a moment on the joint_state_publisher node that is being launched.

If you do a rostopic list again, there is a topic called /joint_states. The states of the joints of the simulated and physical robot are published in this topic. So, we need to put this topic into the /source_list parameter, so that MoveIt can know where the robot is at each moment. Let's execute this newly-created launch file, my planning execution.launch, using the following command:

- $ roslaunch scara_moveit_config my_planning_execution.launch

Once MoveIt loads, we will see that the robot is initialized in a different position than in the previous demo. This is because MoveIt is actually getting the data from the simulated which is in this initial position.

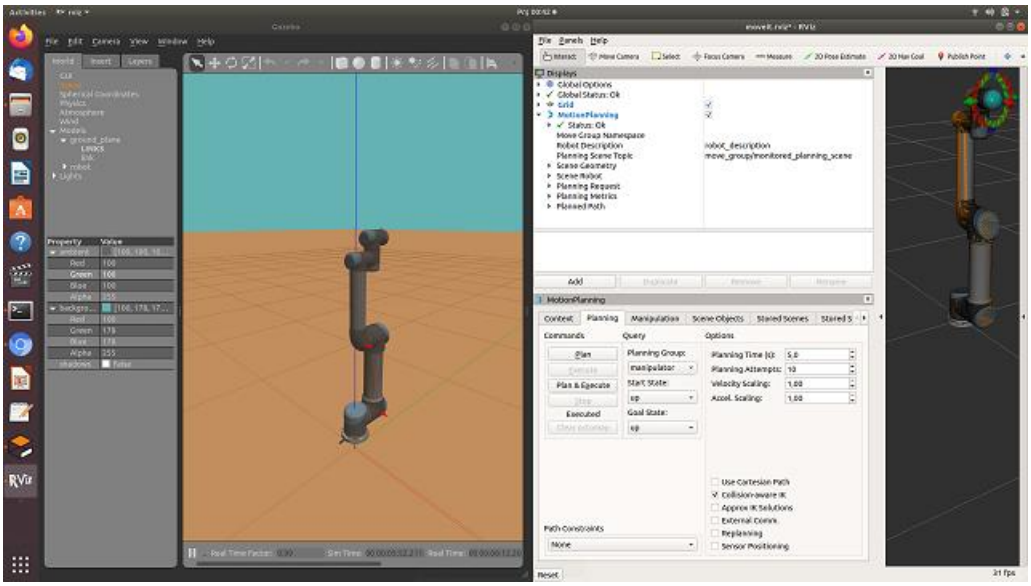


Figure 3.66: Ur5 Gazebo Moveit interaction

Now, plan a trajectory to the home position. Once the trajectory has been planned, press the Execute button to execute it using the. We should observe our robot performing similar actions:

3.18 Vision-Based object Recognition and Precise Positioning

Image Processing done by using "opencv" library in Pyhton. This library allows users to do many things with the camera, such as recognizing the shape, color and etc. Here, an image processing technique called Color Detection and Segmentation is used. Color detection is a technique of detecting any color in a given range of HSV (hue saturation value) color space. Figure 3.67 shows the HSV values of colors [60]. Image segmentation is the process of partitioning digital image and labeling every pixel, where each pixel having the same label shares certain characteristics. Steps are shown to detect objects of determined color using OpenCV (see Figure 3.68). In addition, the total process is explained visually in Figure 3.69 in detail.

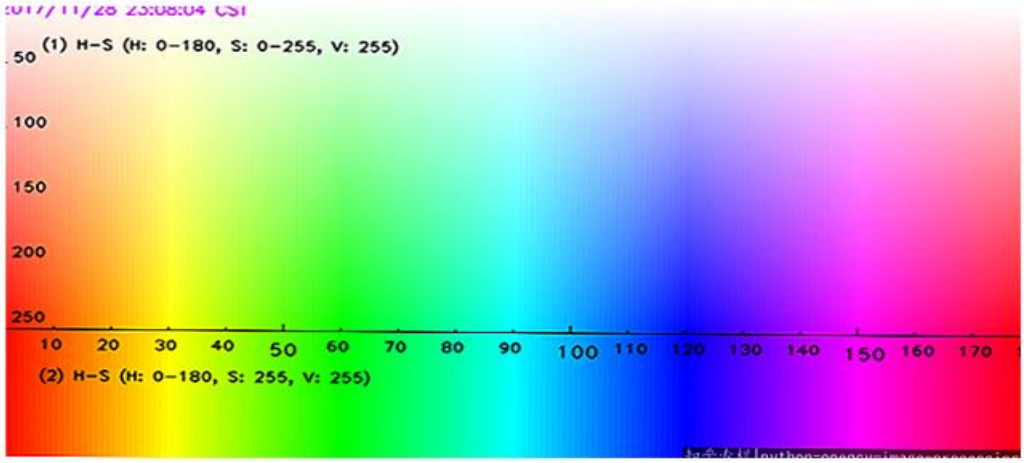


Figure 3.67: HSV values of colors [60].

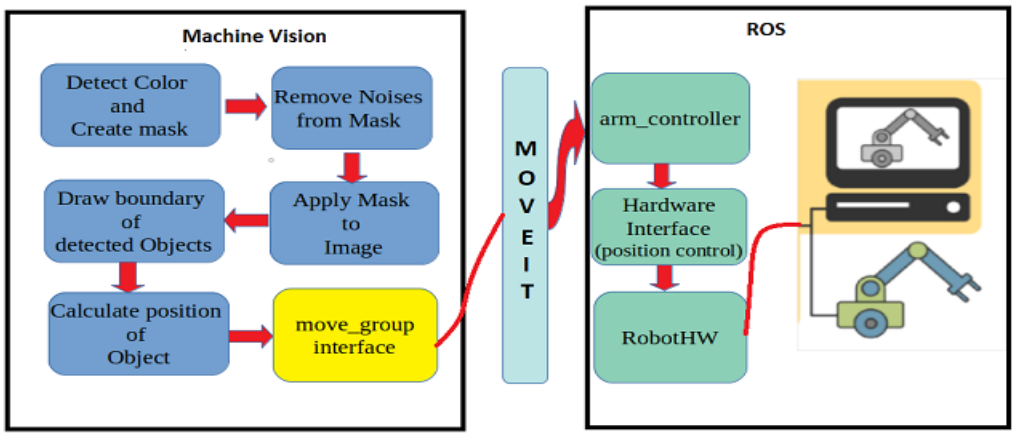


Figure 3.68: Schematic of machine vision and entire process

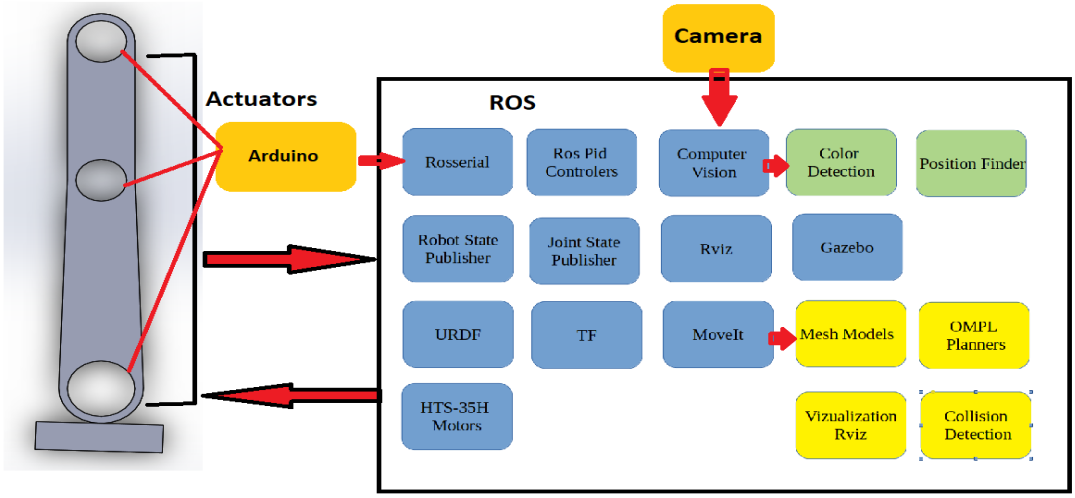


Figure 3.69: Overall process

In this study, the blue object was chosen as the predetermined color in the experimental studies. For machine vision HSV (hue, saturation, value) model is used which is an alternative representation of the RGB color model. We write a Python script that gives a chance to users to use sliders to adjust the HSV min and max values (see Figure 3.70). Adjust the min and max slide bars in the track bars window until you get the desired object alone and the rest is black which shown below at Figure 3.71. We take the corresponding HSV values and used these in our main algorithm.

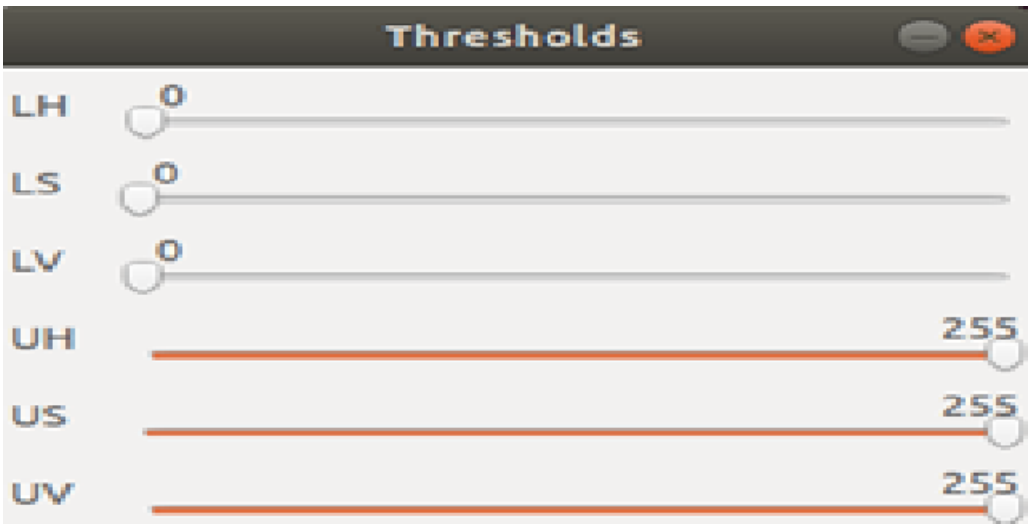


Figure 3.70: HSV trackbars window

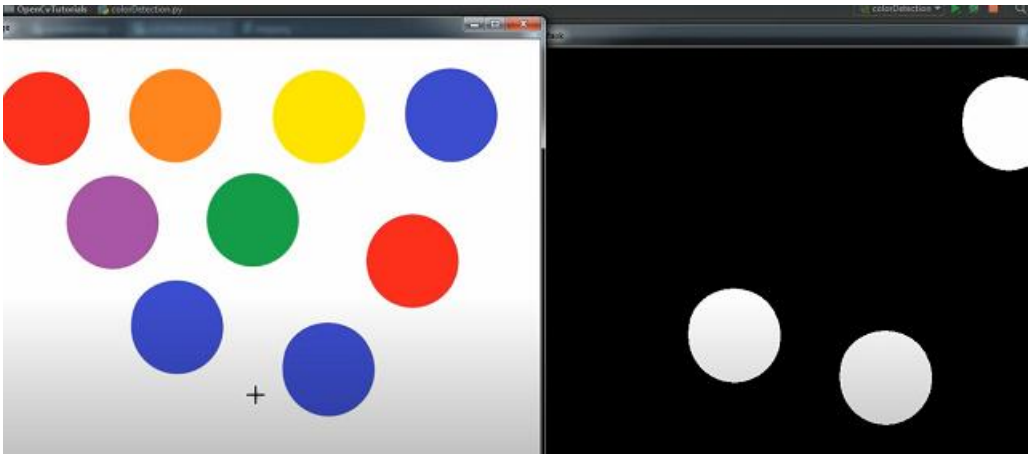


Figure 3.71: Blue object detection

Chapter 4

Experiment

4.1 Tests for Scara Manipulator

A total of 12 tests were carried out for the designed and assembled Scara manipulator, and these tests were presented to the thesis defense jury in order from simple to complex. These tests are all designed for different purposes. First of all, the first 11 test objectives and how they were applied were explained. The 12th test, which is the most complex and will cover almost all tests, is also explained in detail below.

- Task1

This test was carried out by running the roslaunch scaratwo rviz_urdf.launch file, the content of which is included in the appendix tasks launch files section. The purpose of task 1 is just to check if the urdf is written correctly or not, the scara manipulator is provided with the joint_state_publiher_gui to be displayed in the Rviz environment. The joint_state_publisher_gui slider values were changed to check whether the joints rotate and translate in the Rviz environment. In addition, the correctness of the transformation of all joints and links was confirmed by looking at the tf topic.

- Task2

The purpose of task 2 is to have the robot be spawned to the gazebo simulation environment without a controller to check if the real-world physical conditions are set correctly in the urdf. These physical properties are collision, inertia, gravity, friction-damping coefficients, and solid body colors.

- Task3

This test was carried out by running the roslaunch scaratwo check_motors.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 3 was to activate the RobotModel in the Rviz display environment and to check whether both the real robot and the robot in Rviz move equivalently with the help of the Joint_state_publisher_gui slider. It was observed that by changing the values of the joint_state_publisher_gui slider, the real robot and Rviz are synchronized.

- Task4

This test was carried out by running the roslaunch scaratwo check_motors_gazebo.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 4 was spawned to the gazebo simulation environment with the Robot Model and its position controller and joint_state_controller. Real robot control was achieved by controlling the gazebo simulation robot.

- Task5

This test was carried out by running the roslaunch scaratwo rviz_all.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 5 is to publish the RobotModel to the Rviz display environment with joint_state_publisher_gui, to spawn it to the gazebo simulation environment with the pid controller, and also to communicate with the physical robot with rosserial. With the Joint_state_publisher_gui sliders, both the gazebo simulation Robot Model and the real robot move simultaneously to the joint values in Rviz.

- Task6

This test was carried out by running the roslaunch scaratwo gazebo_command_all.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 6 is to load the RobotModel to the Rviz display environment, to spawn it to the gazebo simulation environment with the pid controller, and also to communicate with the physical robot with rosserial. This time, by published the joint values on the gazebo arm_controller/command topic, both the

Robot model in Rviz and the real robot moved simultaneously. The joint values of this specified arm_controller/command topic can be published from the terminal as in the example below, or can be printed with a special script file.

rostopic pub /arm_controller/command trajectory_msgs/JointTrajectory '{joint_names: ["joint1", "joint2", "joint3"], points: [{positions: [0.1, 0.57, 0.57], time_from_start: [1.0, 0.0]}]}' -1

- Task7

This test was carried out by running the roslaunch scaratwo joint_state_all.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 7 is to load the RobotModel to the Rviz display environment, to spawn it to the gazebo simulation environment with the pid controller, and also to communicate with the physical robot with rosserial. In this task, 2 different terminals are opened. In the first of the new terminals, rosrun scartwo path_generator_v5_2.py is run. Here, the user is asked to enter the starting and ending values of the robot's end-effector. The via-point, which is 3.rd point, is calculated as the midpoint of a circle that will accept the distance between the start and end points as a diameter.In this task, 2 different terminals are opened. In the first of the new terminals, rosrun scartwo path_generator_v5_2.py is run. Here, the user is asked to enter the starting and ending values of the robot's end-effector. The via-point, which is 3.rd point, is calculated as the midpoint of a circle that will accept the distance between the start and end points as a diameter, and the robot must pass with a trajectory algorithm suitable for a function of 3 degrees so that the robot's end point passes here, and the angle values of the orbit falling on each axis are written in the path.txt file. Then run rosrun scaratwo pub_joint2_deneme2.py in terminal 2. Then, rosrun scaratwo pub_joint2_deneme2.py is run on the 2nd terminal. Thus, the trajectory angle values written in the path.txt file are published simultaneously to the RobotModel in Rviz, the gazebo simulation robot and the real robot.

- Task8

This test was carried out by running the roslaunch scaratwo joint_state_all.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 8 is to load the RobotModel to the Rviz display environment, to spawn it to

the gazebo simulation environment with the pid controller, and also to communicate with the physical robot with rosserial. In this task, first of all, a desired trajectory is drawn in Solidworks program, followed by the end-effector. Then, a mate relationship is established between this drawn trajectory and the end effector. With the Solidworks motion analysis module, the speed to be followed by the end effector is determined and the angle values are written to the path22.txt file for the joints to follow the trajectory. New two terminal are opened and by running the following commands in this terminal (rosrun rosserial_python serial_node.py _port: =/dev/ttyACM0 _baud: =115200, rosrun scaratwo pub_joint2_solid.py), the trajectory angle values obtained from solidworks are published on the joints, and the RobotModel in Rviz, the robot in the gazebo simulation environment and the real robot follow this trajectory.

- Task9

This test was carried out by running the roslaunch scara_moveit_config demo.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 9 is to control the real robot with moveit. The commands below are run on different terminals and the specified angle values are sent to the joints via moveit_commander (rosrun rosserial_python serial_node.py _port: =/dev/ttyACM0 _baud: =115200, rosrun scaratwo scara_jointspace_trajectory.py). Moveit KDL solver solve forward kinematics of Scara robot arm and provide the movement both moveit and the real robot.

- Task10

This test was carried out by running the roslaunch scara_moveit_config demo.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 10 is to control the real robot with moveit. The commands below are run on different terminals and the specified position values are sent to the manipulatör end-effector via moveit_commander (rosrun rosserial_python serial_node.py _port: =/dev/ttyACM0 _baud: =115200, rosrun scaratwo scara_trajectory.py). Moveit KDL solver solve inverse kinematics of Scara robot arm and provide the movement both moveit and the real robot.

- Task11

This test was carried out by running the roslaunch scara_moveit_config demo.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 11 is to control the real robot with moveit. In this task, 2 positions are determined. 1st position pick 2nd position place. These positions are written into the array in the scara_trajectory_new_topic.py script file. The commands below are run on different terminals and the specified 2 determined position values are sent to the manipulatör end-effector via moveit_commander (rosrun rosserial_python serial_node.py _port: =/dev/ttyACM0 _baud: =115200, rosrun scaratwo scara_trajectory_new_topic.py). Kinematic solver written by the author solve inverse kinematics of Scara robot arm and provide the movement both moveit and the real robot.

- Task12

For the physical manipulator and the robot in the simulation environment to behave equally (position and orientation) as a result of kinematic solutions, the transformation tree has been adjusted to be equal.

In the Figure 4.1 and Figure 4.2 the environment in which the manipulator is located is shown. Since the machine vision algorithm works during the task, the object to be grasped can be left in a random position.

The position of the object is known with the help of machine vision. In Figure 4.1 and Figure 4.2 the environment in which the manipulator is located as shown. For this work, lots of planning algorithms from OMPL are tried.

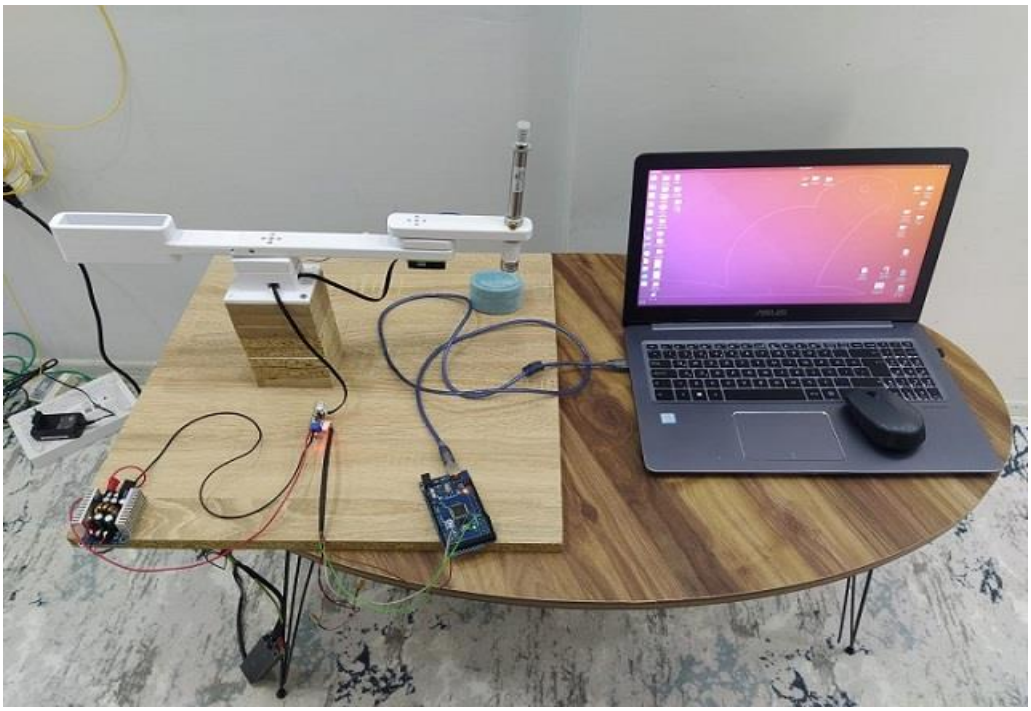


Figure 4.1: Experimental Setup 1

Figure 4.2: Experimental setup 2

While determining the object's position by machine vision and placing it in the target position, it is shown as the physical manipulator Figure 4.3(1-12) A in the manipulation experiment, while the manipulator (1-12) B in the simulation.

Figure 4.3 (1-6) A shows the real-world experimental environment of the physical manipulator while Figure 4.3 (1-6) B shows the simulation environment which is Rviz. The position of the manipulator in the initial state is shown in the physical manipulator Figure 4.3 (1A) and the manipulator in the simulation is shown in Figure 4.3 (1B). The position of the object with the predetermined color is determined by machine vision and this information is sent to the motion planner as the target position, and the planner moves the manipulator to the target position. The execution of the trajectory can be seen in Figure 4.3 (1-6) A for the physical manipulator and in Figure 4.3 (1-6) B for the visualized manipulator. When the manipulator comes to the Figure 4.3 (6A) position, the air is pressed into the pneumatic cylinder and the electromagnet becomes active at the same time. This part is the pre-pick phase. The cylinder in the end effector goes up and down very quickly, picking up the object by magnetizing it. After taking the object, the cylinder is deactivated and the object is taken up with the help of the spring in the single-acting cylinder. The electromagnet is active until the manipulator end-effector reaches the target position which is shown Figure 4.3 (11A). When the end effector reaches the position shown in Figure 4.3 (12A), the object is brought closer to the ground by first giving air to the pneumatic cylinder, and then the electromagnet is deactivated and the object is released.

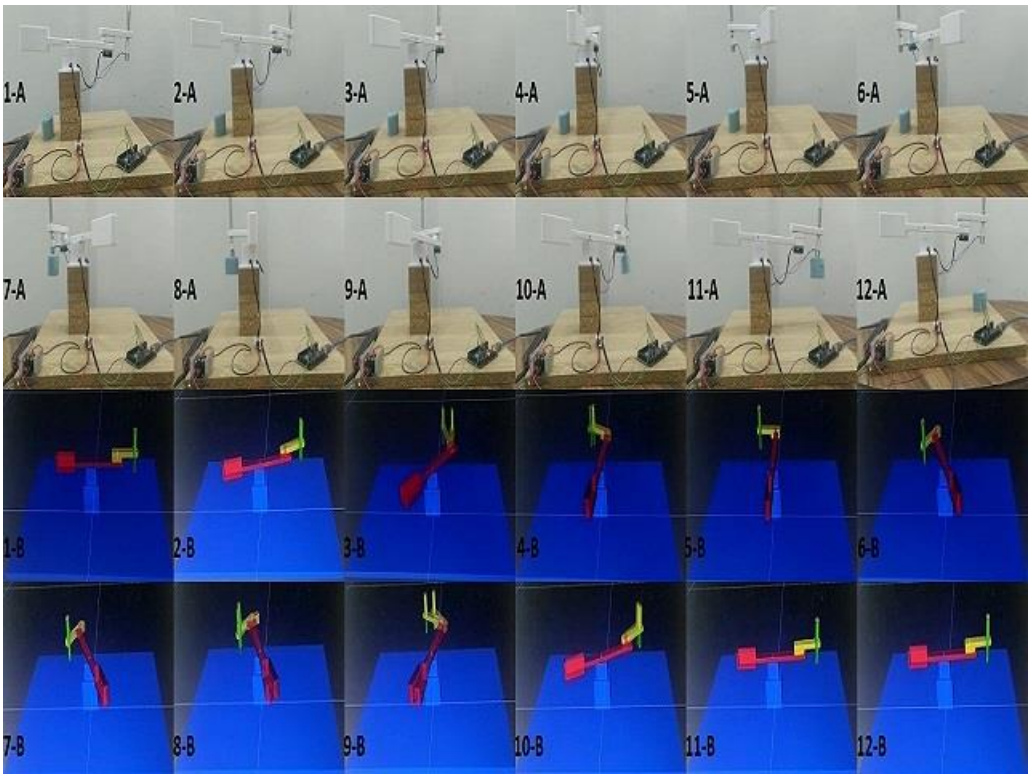


Figure 4.3: Entire Process of this study which mentioned chapter 4 section 4.1

4.2 Tests for UR5 Robot

A total of 6 tests were carried out for the industrial Ur5 robot arm, and these tests were presented to the thesis defense jury in order from simple to complex. These tests are all designed for different purposes. First of all, the first 5 test objectives and how they were applied were explained. The 6th test, which is the most complex and will cover almost all tests, is also explained in detail below.

- Test1

This test was carried out by running the roslaunch arm_description view_ur5.launch file, the content of which is included in the appendix tasks launch files section. The purpose of task 1 is just to check if the urdf is written correctly or not, the Ur5 manipulator is provided with the joint_state_publiher_gui to be displayed in the Rviz environment. The joint_state_publisher_gui slider values were changed to check whether the joints rotate and translate in the Rviz environment. In addition, the correctness of the transformation of all joints and links was confirmed by looking at the tf topic.

- Test2

This test was carried out by running the roslaunch arm_gazebo ur5.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 2 is to control the gazebo simulation robot with moveit. The following commands are run on different terminals(roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true, roslaunch ur5_moveit_config moveit_rviz.launch config:=true) and with the end effector interaction tool provided by Moveit it, the end effector of the robot arm is pulled to a desired position in Rviz and by pressing the plan&execute button, Moveit KDL solver solve appropriate kinematics of Ur5 robot arm.So, Moveit provide the movement both moveit and the gazebo simulation robot.

- Test3

This test was carried out by running the roslaunch arm_gazebo ur5.launch file, the content of which is included in the appendix tasks launch files section. The purpose of Task 3 is to control the gazebo simulation robot with moveit. The commands below are run on different terminals and the specified position values are sent to the manipulator end-effector via moveit_commander (roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true, roslaunch ur5_moveit_config moveit_rviz.launch config:=true,python ur5_tajectory.py). Moveit KDL solver solve inverse kinematics of Ur5 robot arm and provide the movement both moveit and the real robot.

- Test4

This test was carried out by running the (roslaunch arm_gazebo ur5.launch, roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true, roslaunch ur5_moveit_config moveit_rviz.launch config:=true) files, the content of which is included in the appendix tasks launch files section. The purpose of Task 4 is to control the real robot with moveit. The commands below are run on different terminals and the specified angle values are sent to the joints via moveit_commander (python ur5_jointspace_trajectory.py). Moveit KDL solver solve forward kinematics of Ur5 robot arm and provide the movement both moveit and the gazebo simulation robot.

- Test5

This test was carried out by running the (roslaunch arm_gazebo ur5.launch, roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true, roslaunch ur5_moveit_config moveit_rviz.launch config:=true) files, the content of which is included in the appendix tasks launch files section. In this task, first of all, a desired trajectory is drawn in Solidworks program, followed by the end-effector. Then, a mate relationship is established between this drawn trajectory and the end effector. With the Solidworks motion analysis module, the speed to be followed by the end effector is determined and the angle values are written to the path22.txt file for the joints to follow the trajectory. A new terminal is opened and by running the following commands in this terminal (python ur5_jointspace_trajectory_solid.py), the trajectory angle values obtained from solidworks are published on the joints, and the RobotModel in Rviz, the robot in the gazebo simulation environment follow this trajectory.

- Test6

For the moveit manipulator and the robot in the simulation environment(gazebo) to behave equally (position and orientation) as a result of kinematic solutions, the transformation tree has been adjusted to be equal.

In the Figure 4.4 the simulation environment in which the manipulator is located is shown. Since the pick-and place algorithm works during the task, the object to be grasped can be left in a known position and orientation.

The position and orientation of the object is defined by the user. For this work, lots of planning algorithms from OMPL are tried.

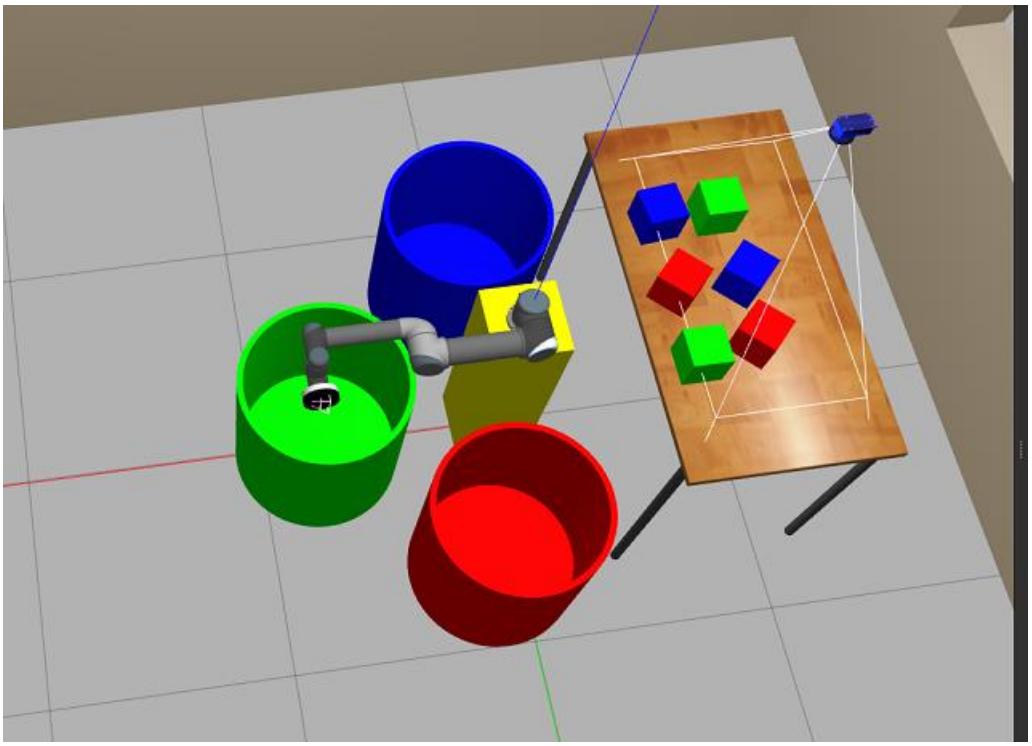


Figure 4.4: UR5 experimental setup

There are 3 different colored object groups. These color groups are red green and blue. In order to fulfill the condition of throwing these objects into the same-colored box, it is determined with the help of the camera in which box the received object will be placed. Figure 4.5 (1-8) A shows the simulation experimental environment of the simualtion manipulator. The position of the manipulator in the initial state is shown in the simualtion manipulator Figure 4.4. The position of the object with the predetermined color is determined by user and this information is sent to the motion planner as the target position, and the planner moves the manipulator to the target position. The execution of the trajectory can be seen in Figure 4.5 (1-8) A for the visualized manipulator. When the manipulator comes to the Figure 4.5 (1A) position, the vacuum actuator started to work and the suction becomes active at the same time. This part is the pre-pick phase. The vacuum actuator in the end effector very strong, picking up the object by sucking it. After taking the object, the vacuum actuator is active until the manipulator end-effector reaches the target position which is shown Figure 4.5 (8A). When the end effector reaches the position shown in Figure 4.5 (8A), the object is brought closer to the ground, and then the vacuum gripper is deactivated and the object is released.

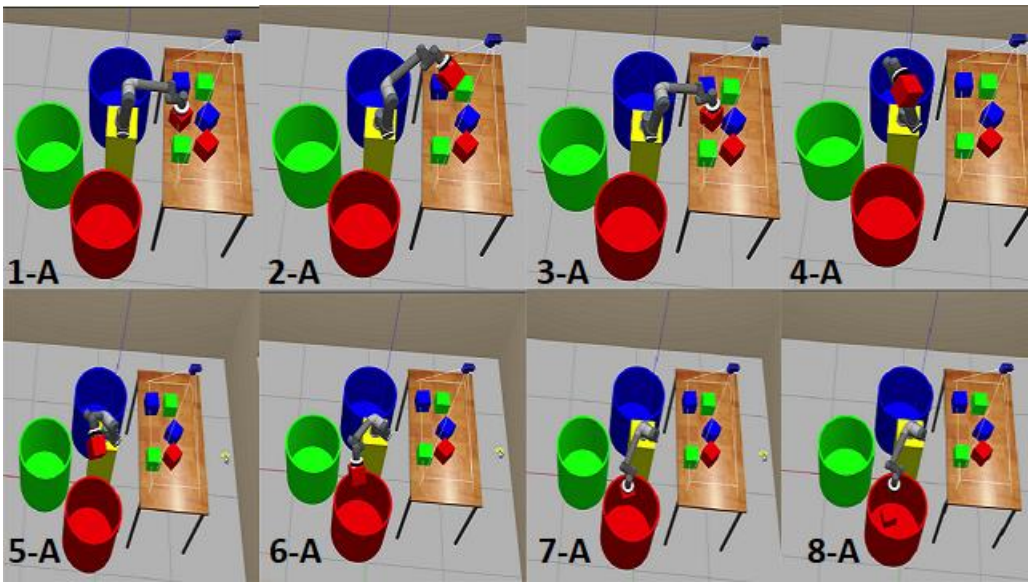


Figure 4.5: The whole process of the task of picking&placing red objects

The same tasks are performed by the robot in the green and blue objects. In Figure 4.6 and Figure 4.7, the task steps of green and blue colors are shown, respectively.

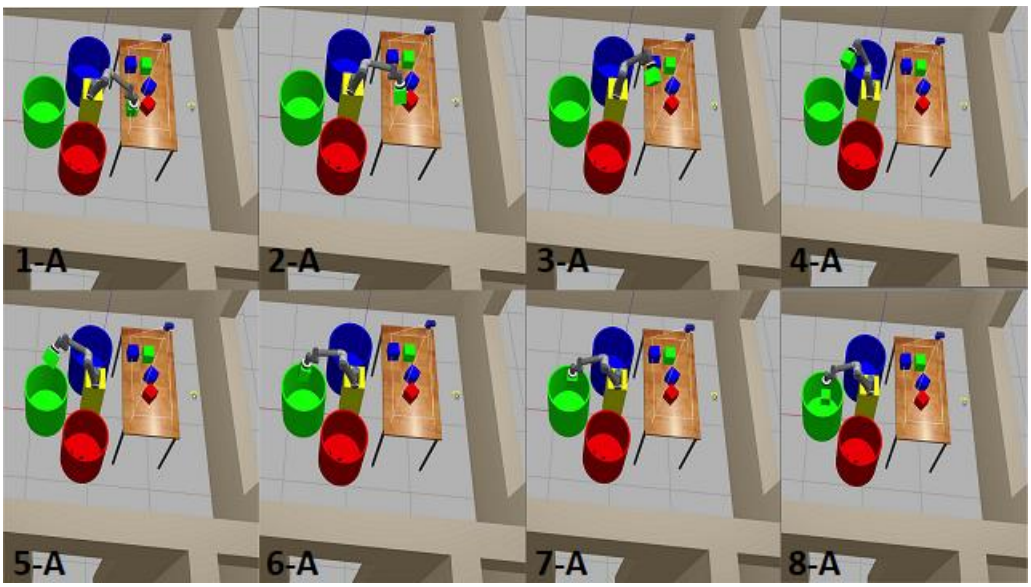


Figure 4.6: The whole process of the task of picking&placing green objects

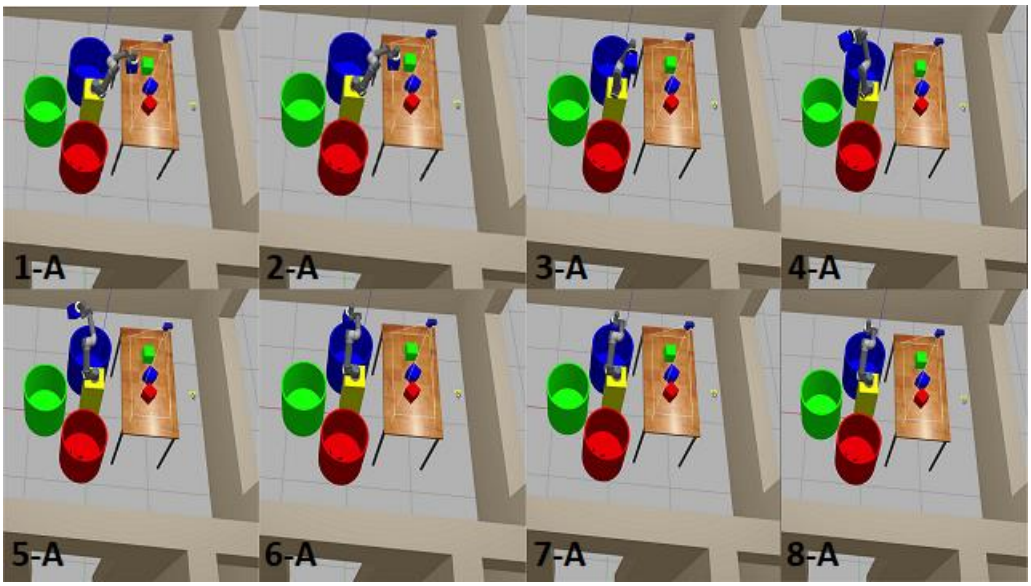


Figure 4.7: The whole process of the task of picking&placing blue objects

Chapter 5

Conclusion

In this study, pick&place task implementation of a Scara Manipulator and a UR5 via ROS and machine vision were presented. A physical manipulator of Scara manipulator, machine vision, control algorithms, and simulation studies have been combined with the Ros framework and its packages. With the help of this integration, it was possible to visualize the state of the physical manipulator in Rviz during the pick&place task. 3rd party software MoveIt was used to calculate the trajectory and move the object from the position determined by image processing to the predetermined station position. Trajectory tries and pick&place studies performed in the simulation fully matched the joint angles and end-effector position values calculated in the kinematic analysis. The method that we developed in this thesis is easy and flexible enough for object recognition and pick&place applications for industry. In future work, this study's outputs can be implemented for real-world problems in the industrial task needs. Industrial tasks can be summarized as follows,

- ✓ In the food industry for Packing and Palletizing,
- ✓ In agriculture industry for harvesting products on fields,
- ✓ In machine industry for placing smd electronic components on PCBs,
- ✓ In 3D printer industry for advanced manufacturing of parts,
- ✓ In automative industry for welding parts and assembling components,

This thesis will help reader to understand all detail and implementation of ROS on physical robot as well as simulation of an industrial robot manipulator.

References

[1] J. Kerr and K. Nickels, “Robot operating systems: Bridging the gap between human and robot,” *Proc. Annu. Southeast. Symp. Syst. Theory*, pp. 99–104, 2012, doi: 10.1109/SSST.2012.6195127.

[2] S. Cousins, “Welcome to ROS topics,” *IEEE Robot. Autom. Mag.*, vol. 17, no. 1, pp. 13–14, 2010, doi: 10.1109/MRA.2010.935808.

[3] S. Cousins, “Is ROS good for robotics?,” *IEEE Robot. Autom. Mag.*, vol. 19, no. 2, pp. 13–14, 2012, doi: 10.1109/MRA.2012.2193935.

[4] Open robotics. ROS in Science Robotics [internet]. 2019 [accessed 15.10.2019]. https://www.openrobotics.org/blog.

[5] Open robotics. Sig/Ros/java/Android Studio [internet]. 2019 [accessed 10.11.2021]. http://wiki.ros.org/sig/Rosjava/Android Studio.

[6] J. M. Badger, J. D. Yamokoski, and B. J. Wightman, “Towards autonomous operation of Robonaut 2,” *AIAA Infotech Aerosp. Conf. Exhib. 2012*, no. June, pp. 1–8, 2012, doi: 10.2514/6.2012-2441.

[7] Wikipedia. Robot Operating System [internet]. 2022 [accessed 17.05.2022]. https://en.wikipedia.org/wiki/Robot_Operating_System.

[8] R. A. Brooks, “Planning Collision Free Motions for Pick and Place Operations.,” pp. 5–37, 1984.

[9] Open robotics GvdHoorn. Industrial [internet]. 2020 [accessed 24.06.2020]. http://wiki.ros.org/Industrial#:~:text=Industrial manipulators for ABB%2C Adept,supported by ROS-Industrial packages.

[10] S. M. Edwars, “Leveraging the Open Source Robot Operating System (ROS) for New Industrial Applications,” 2011.

[11] S. Zaman, W. Slany, and G. Steinbauer, “ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues,” *Saudi Int. Electron. Commun. Photonics Conf. 2011, SIECPC 2011*, pp. 0–4, 2011, doi: 10.1109/SIECPC.2011.5876943.

[12] K. Demarco, M. E. West, and T. R. Collins, “An implementation of ROS on the Yellowfin autonomous underwater vehicle (AUV),” *Ocean. - MTS/IEEE Kona, Progr. B.*, 2011, doi: 10.23919/oceans.2011.6107001.

[13] S. Gong, H. Liu, Y. Hu, and J. Zhang, “ROS-based object localization using RFID and laser scan,” *2012 IEEE Int. Conf. Inf. Autom. ICIA 2012*, no. June, pp. 406–411, 2012, doi: 10.1109/ICInfA.2012.6246839.

[14] J. Cunha *et al.*, “CAMBADA @ Home ’ 2012 : Team Description Paper,” 2012.

[15] Bharatheesha, M., Rudinac, M., Chandarr, A., Gaisser, F., Rueda, S.P. and J. P. Küpers, B., Driessen, S., Bruinnik, M., Wisse, M., “Delft Robotics Robocup,” *Robocup@Home*, 2012.

[16] E. Vargas, H.S., Olmedo, E., Martínez, A.D., López, M.L.M., Medina and J. L. Perez, “Donaxi,” *Donaxi@HomeProject*, 2012, [Online]. Available: http://wla.orgfree.com/prueba/english/pagina1.php

[17] L. Luing, V., Sarnecki, L., Grün, T., Knauf, Barthen, A., Syre, “Robocup 2012,” *omer@UniKoblenz*, 2012, [Online]. Available: http://robots.uni-koblenz.de

[18] K. Stückler, J., Droeschel, D., Grave, “Team Description,” *Nimbro@Home*, 2012, [Online]. Available: http://www.nimbro.net

[19] Hiwonder. Hiwonder HTS-35H High Voltage Bus Servo 35KG Torque with Data Feedback [internet]. 2022 [accessed 15.12.2021]. https://www.hiwonder.hk/products/hiwonder-hts-35h-high-voltage-bus-servo-35kg-torque-with-data-feedback.

[20] EstoMarket. EXPFLEX MSA 16-50 TEK ETKILI KALEM SILINDIR PISTON [internet]. 2022 [accessed 11.01.2022]. https://www.estomarket.com/expflex-msa-16-50-tek-etkili-kalem-silindir-

piston-22111.html.

[21] Robotistan. P20/15 Elektromıknatıs 2.5 kg Tutma Gücü [internet]. 2022 [accessed 12.01.2022]. https://www.robotistan.com/p2015-elektro-miknatis-25kg-tutma-gucu.

[22] R. Murray, Z. Li, and S. Sastry, *A mathematical introduction to robotic manipulation Cited by me analytic_grasp_synt... grasp_quality_metrics*, vol. 29. 1994. [Online]. Available: /citations?view_op=view_citation&continue=/scholar%3Fhl%3Den%26start%3D140%26as_sdt%3D0,5%26scilib%3D1&citilm=1&citation_for_view=OZNzz9gAAAAJ:35N4QoGY0k4C&hl=en&oi=p

[23] M. Spong, M.W., Hutchinson, S., Vidyasagar, *Robot Modeling and Control*, First. John Wiley & Sons, 2005.

[24] Lung-Wen Tsai, *Robot Analysis*, 1st editio. USA: Wiley-Interscience, 1999.

[25] M. R. de Gier, "Control of a robotic arm: Application to on-surface 3D-printing," Delft University of Technology, 2015.

[26] John J. Craig, *Introduction to Robotics*, Third edit. USA: Pearson Education International, 2005.

[27] R. S. Andersen, "Kinematics of a UR5," *Aalborg Univ. May 2018*, pp. 1–12, 2018, [Online]. Available: http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf

[28] S. Uzuner, N. Akkuş, and M. Toz, "5 Eksenli Manipülatörün Eklem Uzayında Yörünge Planlaması," *J. Polytech.*, vol. 20, no. 1, pp. 151–157, 2017.

[29] Open robotics TullyFoote. Ubuntu install of ROS Melodic [internet]. 2020 [accessed 24.09.2021]. http://wiki.ros.org/melodic/Installation/Ubuntu.

[30] ROS.org Marguedas. ROS Melodic [internet]. 2018 [accessed 24.09.2021]. http://wiki.ros.org/melodic.

[31] J. C. Joseph, Lentin, *Mastering ROS for Robotics Programming*, Second. Packt

Publishing Ltd, 2018.

[32] ROS.org AustinHendrix. msg [internet]. 2019 [accessed 14.05.2020]. http://wiki.ros.org/msg.

[33] TheConstruct Ruben Alves. What is ROS Service [internet]. 2018 [accessed 05.01.2020]. https://www.theconstructsim.com/ros-5-mins-027-ros-service/.

[34] ROS.org AaronMR. ROS Concepts [internet]. 2014 [accessed 05.01.2020]. http://wiki.ros.org/ROS/Concepts#:~:text=services in ROS.-,ROS Computation Graph Level,the Graph in different ways.

[35] ROS.org Dulani Woods. Understanding ROS Nodes [internet]. 2016 [accessed 06.01.2020]. http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[36] Sriram Murali, "An analysis on controlling humanoid robot arm using Robot Operating System (ROS)," Middlesex University, 2017.

[37] ROS.org TullyFoote. Topics [internet]. 2019 [accessed 07.01.2020]. http://wiki.ros.org/Topics.

[38] ROS.org AnisKoubaa. Services [internet]. 2019 [accessed 07.01.2020]. http://wiki.ros.org/Services.

[39] MathWorks. ROS Actions Overview [internet]. 2020 [accessed 08.01.2020]. https://www.mathworks.com/help/ros/ug/ros-actions.html.

[40] ROS.org Davetcoleman. Understanding PR2 URDF Advanced [internet]. 2013 [accessed 06.032021]. http://wiki.ros.org/urdf/Tutorials/UnderstandingPR2URDF .

[41] ROS.org AdamAllevato. Create Your Own URDF File [internet]. 2017 [accessed 21.03.2021]. http://wiki.ros.org/urdf/Tutorials/Create your own urdf file.

[42] ROS.org IlyaPankov. urdf/XML/joint [internet]. 2016 [accessed 26.03.2021]. http://wiki.ros.org/urdf/XML/joint.

[43] ROS.org Jarvisschultz. Tf [internet]. 2017 [accessed 09.01.2020].

https://wiki.ros.org/tf.

[44] ROS.org TullyFoote. Tf Tutorials [internet]. 2020 [accessed 25.01.2020]. http://ros.org/wiki/tf/Tutorials.

[45] ROS.org WilliamWoodall. Rviz [internet]. 2018 [accessed 27.01.2020]. http://wiki.ros.org/rviz.

[46] ROS.org Dornhege. What-is-a-package-stack-repository [internet]. 2014 [accessed 26.01.2020]. https://answers.ros.org/question/10254/what-is-a-package-stack-repository/.

[47] I. Zubrycki and G. Granosik, “Abstract :,” vol. 8, 2014, doi: 10.1431/JAMRIS.

[48] ROS.org GvdHoorn. Industrial [internet]. 2020 [accessed 15.10.2021]. http://wiki.ros.org/Industrial.

[49] ROS.org. ROS-INDUSTRIAL [internet]. 2022 [accessed 20.11.2020]. https://rosindustrial.org/ric/current-members.

[50] ROS.org GvdHoorn. Industrial supported_hardware [internet]. 2020 [accessed 15.11.2020]. http://wiki.ros.org/Industrial/supported_hardware.

[51] Ubuntu. Ubuntu Melodic 18.04 LTS [internet]. 2018 [accessed 20.01.2020]. https://ubuntu.com/.

[52] ROS.org TullyFoote. ROS Melodic Installation [internet]. 2020 [accessed 23.03.2021]. http://wiki.ros.org/melodic/Installation/Ubuntu.

[53] ROS.org KatherineScott. Installing and Configuring Your ROS Environment [internet]. 2020 [accessed 27.03.2020]. http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment.

[54] ROS.org Muhammad Luqman.Creating a ROS Package [internet]. 2021 [accessed 27.03.2021]. http://wiki.ros.org/ROS/Tutorials/CreatingPackage.

[55] Gazebo. Gazebo simulation [internet]. 2016 [accessed 30.03.2021]. https://gazebosim.org/home.

[56] TheConstruct Ricardo Tellez. What is moveit_ros? All about MoveIt! ROS [internet]. 2018 [accessed 05.04.2021]. https://www.theconstructsim.com/ros-moveit/.

[57] O. A. M. A. H Kara, “The open motion planning library,” *Pap. Knowl. . Towar. a Media Hist. Doc.*, vol. 7, no. 2, pp. 107–15, 2014.

[58] ROS.org Jschleicher. Create a MoveIt Package for an Industrial Robot [internet]. 2018 [accessed 28.07.2021]. http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot.

[59] Universal Robot Github. Universal Robot [internet]. 2019 [accessed 20.04.2021]. https://github.com/ros-industrial/universal_robot.

[60] S. Li and G. Guo, “The application of improved HSV color space model in image processing,” *Proc. 2010 2nd Int. Conf. Futur. Comput. Commun. ICFCC 2010*, vol. 2, pp. 10–13, 2010, doi: 10.1109/ICFCC.2010.5497299.

Appendices

Appendix A

Codes for Scara Manipulator

- ✓ workspace_analysis.py

```
#! /usr/bin/env python
import math
import matplotlib.pyplot as plt
import numpy as np
l1=150
l2=100
c=l1+l2
d=l1-l2
def w():
        for x in np.arange(-250.0, 250.1, 10.0):
                for y in np.arange(-250.0,250.1,1.0):
                        if ((x**2+y**2<=c**2) and (x**2+y**2>=d**2)):
                                plt.scatter(x,y,color='black')
                                #print("x="+str(x),"y="+str(y))
        plt.show()
w()
```

- ✓ singularity_analysis.py

```
#! /usr/bin/env python
import math
import matplotlib.pyplot as plt
import numpy as np
from time import *
import time
import sys
from sympy import *
from sympy import symbols, Eq, solve
q1 = symbols('q1')
q2 = symbols('q2')
```

```
l1=150.0
l2=100.0
c=l1+l2
d=l1-l2
def w():
        plt.title('singularity')
        plt.xlabel('x',color='red')
        plt.ylabel('y',color='blue')
        plt.xlim([-1000, 1000])
        plt.ylim([-1000, 1000])
        for x in np.arange(-250.0,250.2, 1.0,dtype=float):
                x=round(x,2)
                for y in np.arange(-250.0,250.2,1.0,dtype=float):
                        y=round(y,2)
                        if ((x**2+y**2<=c**2) and (x**2+y**2>=d**2)):
                                #plt.scatter(x,y,color='black')
                                print(x,y)
                                j(x,y)
        plt.show()
def j(x,y):
        A=2.0*x*l1
        B=2.0*y*l1
        C=x**2+y**2+l1**2-l2**2
        if ((C+A!=0)):
                q11=2.0*math.degrees(math.atan((B+math.sqrt(A**2+B**2-
C**2))/(C+A)))
                q12=2.0*math.degrees(math.atan((B-math.sqrt(A**2+B**2-
C**2))/(C+A)))
                q11=round(q11,2)
                q12=round(q12,2)
                q31=math.degrees(math.atan2((y-l1*math.sin(math.radians(q11))),(x-
l1*math.cos(math.radians(q11)))))
```

```
            q32=math.degrees(math.atan2((y-l1*math.sin(math.radians(q12))),(x-
l1*math.cos(math.radians(q12)))))
            q31=round(q31,2)
            q32=round(q32,2)
            q21=q31-q11
            q22=q32-q12
            #print("C+A!=0 bolgesi")
        #print("q11="+str(q11),"q12="+str(q12),"q21="+str(q21),"q22="+str(q22),"q3
1="+str(q31),"q32="+str(q32))
            if((q31-q11==0) or (abs(q31-q11)==180) or (q32-q12==0) or (abs(q32-
q12)==180)):
                plt.scatter(x,y,color='red')
                #plt.show()
                #print("gidilmeyen nokta C+A!=0")
        elif(C+A==0):
            if (A**2+B**2!=0):
                e1=(B*C-A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e11=(B*C+A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e2=(A*C+B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e22=(A*C-B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                q11=math.degrees(math.atan2(e1,e2))
                q12=math.degrees(math.atan2(e11,e22))
                q11=round(q11,2)
                q12=round(q12,2)
                #print("C+A=0 bolgesi")
                #print("q11="+str(q11),"q12="+str(q12))
                q31=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q11))),(x-l1*math.cos(math.radians(q11)))))
                q32=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q12))),(x-l1*math.cos(math.radians(q12)))))
                q31=round(q31,2)
                q32=round(q32,2)
                q21=q31-q11
```

```
                    q22=q32-q12
                    print("q31="+str(q31),"q32="+str(q32))
                    if((abs(q31)-abs(q11)==0)  or  (abs(q31)-abs(q11)==180)  or
(abs(q32)-abs(q12)==0) or (abs(q32)-abs(q12)==180)):
                        plt.scatter(x,y,color='red')
                        #plt.show()
                        print("gidilmeyen nokta C+A=0")
                else:
                    plt.scatter(x,y,color='red')
                    #plt.show()
                    print("gidilmeyen nokta else konumu")
w()
```

- ✓ path_generator_with_torque_analysis.py

```
#! /usr/bin/env python
import math
import matplotlib.pyplot as plt
import numpy as np
#import smbus
import sys
from sympy import *
from sympy import symbols, Eq, solve
# import sympy, Point and Circle
from sympy import Point, Circle
import random
class Path():
   def __init__(self):
      #robotun constant parametreleri
      self.m1 = 150.0
      self.m2 = 350.0
      self.g = 0
      self.l1 = 150.0
      self.l2 = 100.0
      #forward kin. parametreleri
```

```
self.q1 = 0
self.q2 = 0
#inv kin. parametreleri
self.x = 0.000
self.y = 0.000
self.q12 = None
self.q32 = None
self.q22 = None
#robotun sabit parametreleri
self.c = self.l1 + self.l2
self.d = 132.29
self.a1 = None
self.b1 = None
self.c1 = None
self.d1 = None
self.a2 = None
self.b2 = None
self.c2 = None
self.d2 = None
self.a3 = None
self.b3 = None
self.c3 = None
self.d3 = None
self.a4 = None
self.b4 = None
self.c4 = None
self.d4 = None
self.i=1
self.x1=None
self.y1 = None
self.x2 = None
self.y2 = None
 self.WW2= None
```

```
        self.in_min= 0.0
        self.in_max= 240.0
        self.out_min= 0.0
        self.out_max= 0.72
        self.yaricap=None
    def path_matrix(self):
        tt1=[];tt2=[]
       tt1d=[];tt2d=[]
       tt1dd=[];tt2dd=[]
       tork1=[];tork2=[]
       ti=0
       tm=self.sonuc_sure
        tf=0.54
       A_matrix=np.array([[ti**3 ,ti**2 ,ti, 1, 0 ,0, 0, 0,0 ,0, 0, 0,0 ,0, 0, 0],[0 ,0, 0,
0,ti**3,ti**2,ti,1,0 ,0, 0, 0,0 ,0, 0, 0],
                          [3*ti**2,2*ti,1,0,0 ,0, 0, 0,0 ,0, 0, 0,0 ,0, 0, 0],[0 ,0, 0,
0,3*ti**2,2*ti,1,0,0 ,0, 0, 0,0 ,0, 0, 0],
                          [tm**3,tm**2,tm,1,0 ,0, 0, 0,0,0,0,0,0 ,0, 0, 0],[0 ,0, 0,
0,tm**3,tm**2,tm,1,0 ,0, 0, 0,0,0,0,0,0],
                             [3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0, 0, 0,
0, 0], [ 0, 0, 0, 0,3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0],
                          [6*tm,2,0 ,0, 0, 0,0,0,-6*tm,-2,0,0,0 ,0, 0, 0],[0 ,0, 0, 0,6*tm,2,0 ,0, 0,
0,0,0,-6*tm,-2,0,0],
                          [0 ,0, 0, 0,0 ,0, 0, 0,tf**3,tf**2,tf,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,3*tf**2,2*tf,1,0,0 ,0, 0, 0],
                          [0 ,0, 0, 0,0,0 ,0, 0, 0,0 ,0, 0,tf**3,tf**2,tf,1], [0 ,0, 0, 0,0,0 ,0, 0, 0,0
,0, 0,3*tf**2,2*tf,1,0],
                          [0,0,0,0,0,0,0,0,0,tm**3,tm**2,tm,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,0,0,0,0,0,tm**3,tm**2,tm,1]])
B_matrix=np.array([[self.a1],[self.b1],[self.c1],[self.d1],[self.a2],[self.b2],[self.c2],[s
elf.d2],
[self.a3],[self.b3],[self.c3],[self.d3],[self.a4],[self.b4],[self.c4],[self.d4]])
```

```
C_matrix=([[self.K],[self.OO],[0],[0],[self.M],[self.P],[0],[0],[0],[0],[self.C],[0],[self
.E],[0],[self.M],[self.P]])
        A_matrix_inv=np.linalg.inv(A_matrix)
        R_matrix=np.matmul(A_matrix_inv,C_matrix)
         # Convert 2D Numpy array to a flat list
         num_list = R_matrix.flatten().tolist()
         print(num_list)
         self.a1=round(num_list[0],2)
         self.b1=round(num_list[1],2)
         self.c1=round(num_list[2],2)
         self.d1=round(num_list[3],2)
         self.a2=round(num_list[4],2)
         self.b2=round(num_list[5],2)
         self.c2=round(num_list[6],2)
         self.d2=round(num_list[7],2)
         self.a3=round(num_list[8],2)
         self.b3=round(num_list[9],2)
         self.c3=round(num_list[10],2)
        self.d3=round(num_list[11],2)
         self.a4=round(num_list[12],2)
         self.b4=round(num_list[13],2)
         self.c4=round(num_list[14],2)
         self.d4=round(num_list[15],2)
         print(self.a1,self.b1,self.c1,self.d1)
        print(self.a2,self.b2,self.c2,self.d2)
         print(self.a3,self.b3,self.c3,self.d3)
         print(self.a4,self.b4,self.c4,self.d4)
         for ti in np.arange(0.0,(self.sonuc_sure)+0.001,0.003,dtype=float):
                ti=round(ti,3)
                print("ti =",ti)
              print("ti/0.003",int(ti/0.003))
                Q11=self.a1*ti**3+self.b1*ti**2+self.c1*ti+self.d1
              Q11d=3*self.a1*ti**2+2*self.b1*ti+self.c1
```

```
            Q11dd=6*self.a1*ti+2*self.b1
            Q21=self.a2*ti**3+self.b2*ti**2+self.c2*ti+self.d2
            Q21d=3*self.a2*ti**2+2*self.b2*ti+self.c2
            Q21dd=6*self.a2*ti**2+2*self.b2

t1=((self.m1/3+self.m2)*self.l1**2+self.m2*self.l1*self.l2*math.cos(Q21)+self.m2/
3*self.l2**2)*Q11dd+(self.m2/2*self.l1*self.l2*math.cos(Q21)+self.m2/3*self.l2**
2)*Q21dd-
self.m2*self.l1*self.l2*math.sin(Q21)*(Q11d*Q21d+Q21d**2/2)+self.g*((self.m1/2
+self.m2)*self.l1*math.cos(Q11)+self.m2/2*self.l2*math.cos(Q11+Q21))

              tt1.append(round(Q11,2))
              tt2.append(round(Q21,2))
              tt1d.append(round(Q11d,2))
              tt2d.append(round(Q21d,2))
              tt1dd.append(round(Q11dd,2))
              tt2dd.append(round(Q21dd,2))
            tork1.append(round((t1*0.000000001),2))
            plt.xlabel("time (s) ",color='blue',size = 15)
            plt.ylabel("Torque1 (N.m) ",color='red',size =15)
            #plt.plot(int(ti/0.003),tt1[int(ti/0.003)],marker = 'X',markerfacecolor =
'r',ms=6)
            #plt.plot(int(ti/0.003),tt1d[int(ti/0.003)],marker = '*',markerfacecolor =
'k',ms=6)
            #plt.plot(int(ti/0.003),tt1dd[int(ti/0.003)],marker = 'v',markerfacecolor =
'b',ms=6)
            #plt.plot(int(ti/0.003),tork1[int(ti/0.003)],marker = 'd',markerfacecolor =
'b',ms=8)
            plt.plot(ti,tork1[int(ti/0.003)],marker = 'd',markerfacecolor = 'b',ms=8)
        print("q1 seg1 pozisyon",tt1)
        print(" "*50)
        print("q2 seg1  pozisyon",tt2)
        print("q1d seg1  velocity",tt1d)
```

```
        print("q2d seg1  velocity",tt2d)
        print("q1dd seg1 acceleration",tt1dd)
        print("q2dd seg1  acceleration",tt2dd)
        print("seg1 tork1",tork1)
         for tf in np.arange(self.sonuc_sure,tf+0.001, 0.003,dtype=float):
                tf=round(tf,3)
                print("tf =",tf)
                Q12=self.a3*tf**3+self.b3*tf**2+self.c3*tf+self.d3
              Q12d=3*self.a3*tf**2+2*self.b3*tf+self.c3
              Q12dd=6*self.a3*tf+2*self.b3

                Q22=self.a4*tf**3+self.b4*tf**2+self.c4*tf+self.d4
              Q22d=3*self.a4*tf**2+2*self.b4*tf+self.c4
              Q22dd=6*self.a4*tf+2*self.b4
t1=((self.m1/3+self.m2)*self.l1**2+self.m2*self.l1*self.l2*math.cos(Q22)+self.m2/
3*self.l2**2)*Q12dd+(self.m2/2*self.l1*self.l2*math.cos(Q22)+self.m2/3*self.l2**
2)*Q22dd-
self.m2*self.l1*self.l2*math.sin(Q22)*(Q12d*Q22d+Q22d**2/2)+self.g*((self.m1/2
+self.m2)*self.l1*math.cos(Q12)+self.m2/2*self.l2*math.cos(Q12+Q22))
                tt1.append(round(Q12,2))
                tt2.append(round(Q22,2))
                tt1d.append(round(Q12d,2))
                tt2d.append(round(Q22d,2))
                tt1dd.append(round(Q12dd,2))
                tt2dd.append(round(Q22dd,2))
              tork1.append(round((t1*0.000000001),2))
              #plt.plot(int(tf/0.003),tt1[int(tf/0.003)],marker = 'o',markerfacecolor = 'g')
              #plt.plot(int(tf/0.003),tt1d[int(tf/0.003)],marker = '+',markerfacecolor = 'y')
              #plt.plot(int(tf/0.003),tt1dd[int(tf/0.003)],marker = '^',markerfacecolor = 'r')
              #plt.plot(int(tf/0.003),tork1[int(tf/0.003)],marker  =  'h',markerfacecolor  =
'r',ms=8)
              plt.plot(tf,tork1[int(tf/0.003)],marker = 'h',markerfacecolor = 'r',ms=8)
        print("q1 seg2 pozisyon",tt1)
```

```
        print("q2 seg2 pozisyon",tt2)
        print("q1d seg2 velocity",tt1d)
        print("q2d seg2 velocity",tt2d)
        print("q1dd  seg2 acceleration",tt1dd)
        print("q2dd seg2  acceleration",tt2dd)
        print("seg2 tork1",tork1)
        #plt.plot(tt1d)
        #plt.plot(tt1dd)
        plt.show()
         for s in np.arange(0,((tf/0.003)+1), 1,dtype=int):
                self.fk(tt1[s],tt2[s])
                #print(tt1[s],tt2[s])
    def ik(self):
        ik_ok_durum=(self.x) ** 2 + (self.y) ** 2 <=self.c ** 2 and (self.x) ** 2 + (self.y)
** 2 >= self.d ** 2
        if ( ik_ok_durum):
            self.q22=round(math.degrees(math.acos((self.x**2+self.y**2-self.l1**2-
self.l2**2)/(2.0*self.l1*self.l2))),3)
            self.q21=round(-1*math.degrees(math.acos((self.x**2+self.y**2-self.l1**2-
self.l2**2)/(2.0*self.l1*self.l2))),3)
self.q12=math.degrees(math.atan2(self.l2*math.sin(self.q21*(math.pi/180))*self.x+(
self.l1+self.l2*math.cos(self.q21*(math.pi/180)))*self.y,(self.l1+self.l2*math.cos(sel
f.q21*(math.pi/180)))*self.x-self.l2*math.sin(self.q21*(math.pi/180))*self.y))
self.q11=math.degrees(math.atan2(self.l2*math.sin(self.q22*(math.pi/180))*self.x+(
self.l1+self.l2*math.cos(self.q22*(math.pi/180)))*self.y,(self.l1+self.l2*math.cos(sel
f.q22*(math.pi/180)))*self.x-self.l2*math.sin(self.q22*(math.pi/180))*self.y))
            print("elbow-down
configuration","q11="+str(self.q11),"q21="+str(self.q21))
            print("elbow-up   configuration","q12="+str(self.q12),"q22="+str(self.q22))
           if (self.q22==0 or self.q22==180 or self.q22==360 or self.q21==0 or
self.q21==180 or self.q21==360):
              plt.scatter(self.x, self.y, color='red')
              # plt.show()
```

```
            print("Singularity q2==0 or q2=180 or q2= 360")
        else:
            plt.scatter(self.x, self.y, color='red')
            # plt.show()
            print("Workspace disinda bir nokta girdiniz Gidilemeyen Nokta")
        if self.ii == 1:
            self.K = round(self.q11*(math.pi/180), 3)
            self.OO = round(self.q21*(math.pi/180), 3)
            print("K =",self.K )
            print("OO =", self.OO)
        elif self.ii == 2:
            self.C = round((self.q11+360.000)*(math.pi/180), 3)
            self.E = round(self.q21*(math.pi/180), 3)
             print("C =",self.C)
             print("E =",self.E)
        elif self.ii == 3:
            self.M = round(self.q11*(math.pi/180), 3)
            self.P = round(self.q21*(math.pi/180), 3)
            print("M =",self.M )
            print("P =",self.P )


    def via_point_finder(self):
        #print("please enter x value betweeen -240 to 240 mm with x point xnum and y
value between -150 to 150 mm with y point ynum ")
         xsaved_array= []
         xstart=240
         xend=-240
         ystart=0
         yend=0
         for self.i in np.arange(0,1,1):
                self.x1=xstart
                self.y1=ystart
                self.x=self.x1
```

```
                self.y=self.y1
                self.ii=1
                self.ik()
                self.x2=xend
                self.y2=yend
                self.x=self.x2
                self.y=self.y2
                self.ii=2
                self.ik()
                print("x1,y1 value",self.x1,self.y1,"and","x2,y2 value",self.x2,self.y2)
                self.WW2 = math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
                print("iki nokta arasi uzaklik",self.WW2)

            xres=[207.85,120,0,-120,-207.85]
            yres=[120,207.85,240,207.85,120]
            taranan_aci=[30,60,90,120,150]
              self.ii = 3
              if self.ii==3:
              for i in np.arange(0,len(xres)):
                  print("via_point x =", xres[i])
                    print("via_point y =", yres[i])
                    print("taranan aci =", taranan_aci[i])
                  self.map(taranan_aci[i])
                     self.x=xres[i]
                     self.y=yres[i]
                     self.ik()
                    self.path_matrix()

              ######via point has finish###
#forward kinematics
    def fk(self,q1,q2):
        x=self.l1*math.cos(math.radians(q1))+self.l2*math.cos(math.radians(q1+q2))
        y=self.l1*math.sin(math.radians(q1))+self.l2*math.sin(math.radians(q1+q2))
```

```
        print("x ="+str(x),"y ="+str(y))
        #plt.scatter(x,y,color='red')
#Prominent Arduino map function :)
   def map(self,x):
        self.sonuc_sure=round(float((x - self.in_min) * (self.out_max - self.out_min) /
(self.in_max - self.in_min) + self.out_min),3)
        print("sonuc_sure =",self.sonuc_sure)
nesne=Path()
nesne.via_point_finder()
#nesne.path_matrix()
plt.show()
```

Tasks Launch Files

- ✓ #Scara Tasks

```
#Task1 just show robot modelin rviz and control joint with joint_state_publisher_gui
roslaunch scaratwo rviz_urdf.launch
#Task2 spawn robot in to gazebo world without controller
roslaunch scaratwo gazebo_urdf.launch
#Task3 robot show in rviz and control real robot joint with joint_state_publisher_gui
roslaunch scaratwo check_motors.launch
#Task4
roslaunch scaratwo check_motors_gazebo.launch
#Task5
roslaunch scaratwo rviz_all.launch
#Task6
roslaunch scaratwo gazebo_command_all.launch
rostopic pub /arm_controller/command trajectory_msgs/JointTrajectory
'{joint_names: ["joint1", "joint2", "joint3"], points: [{positions: [0.1, 0.57, 0.57],
time_from_start: [1.0, 0.0]}]}' -1
#Task7
roslaunch scaratwo joint_state_all.launch
cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts
python path_generator_v5_2.py
python pub_joint2_deneme2.py
```

#Task8

roslaunch scaratwo joint_state_all.launch

rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200

cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts

python pub_joint2_deneme2_solid.py

#Task9

roslaunch scara_moveit_config demo.launch

rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200

cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts

python scara_jointspace_trajectory.py

#Task10

roslaunch scara_moveit_config demo.launch

rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200

cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts

python scara_trajectory.py

#Task11

roslaunch scara_moveit_config demo.launch

rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200

cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts

python scara_trajectory_new_topic.py

#Task12

cd opencv_kodlari

python lesson11.py

cd /home/etkinlaptop/catkin_ws/src/scara_moveit_config/scripts

python color_thresholding.py

roslaunch scara_moveit_config demo.launch

rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200

cd /home/etkinlaptop/catkin_ws/src/scara_moveit_config/scripts

python get_pose_openCV.py

cd /home/etkinlaptop/catkin_ws/src/scaratwo/scripts

python test.py

- ✓ #Ur5 Tasks

#Task1

roslaunch arm_description view_ur5.launch

#Task2

roslaunch arm_gazebo ur5.launch

roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true

roslaunch ur5_moveit_config moveit_rviz.launch config:=true

#Task3

roslaunch arm_gazebo ur5.launch

roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true

roslaunch ur5_moveit_config moveit_rviz.launch config:=true

cd /home/etkinlaptop/catkin_ws/src

python ur5_trajectory.py

#Task4

roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true

roslaunch ur5_moveit_config moveit_rviz.launch config:=true

cd /home/etkinlaptop/catkin_ws/src

python ur5_jointspace_trajectory.py

#Task5

roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true

roslaunch ur5_moveit_config moveit_rviz.launch config:=true

cd /home/etkinlaptop/catkin_ws/src

python ur5_jointspace_trajectory_solid.py

#Task6

roslaunch ur5_control task.launch

ScaraTwo

Urdf

- ✓ scara.urdf

<?xml version="1.0" encoding="utf-8"?>

<!-- This URDF was automatically created by SolidWorks to URDF Exporter! Originally created by Stephen Brawner (brawner@gmail.com)

Commit Version: 1.5.1-0-g916b5db Build Version: 1.5.7152.31018

For more information, please see http://wiki.ros.org/sw_urdf_exporter -->

```
<robot
  name="scara">
  <material name="blue">
     <color rgba="0 0 0.8 1"/>
  </material>
  <material name="red">
     <color rgba="0.8 0 0 1"/>
  </material>
  <material name="green">
     <color rgba="0 0.8 0 1"/>
  </material>
  <material name="yellow">
     <color rgba="1 1 0.2 1"/>
  </material>
  <gazebo>
     <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
        <robotNamespace>/</robotNamespace>
     </plugin>
  </gazebo>
  <!--
  <gazebo>
    <plugin                                                    name="joint_state_publisher"
filename="libgazebo_ros_joint_state_publisher.so">
     <jointName>joint1, joint2, joint3</jointName>
    </plugin>
  </gazebo> -->

  <!-- * * * Link Definitions * * * -->
  <link
    name="world"/>
  <link
    name="base_link">
    <inertial>
```

```
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <mass
      value="9.9007" />
    <inertia
      ixx="0.23671"
      ixy="0"
      ixz="0"
      iyy="0.54727"
      iyz="0"
      izz="0.77689" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scara/meshes/base_link.STL" />
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scara/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

```
<gazebo reference="base_link">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Blue</material>
</gazebo>
<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>
<link
  name="link1">
  <inertial>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <mass
      value="0.20189" />
    <inertia
      ixx="0.000005625"
      ixy="0"
      ixz="0"
      iyy="0.0008011"
      iyz="0"
      izz="0.0008045" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <geometry>
      <mesh
```

```
      filename="package://scaratwo/meshes/link1.STL" />
    </geometry>
    <material name="red"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link1.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="link1">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Red</material>
</gazebo>
<joint
  name="joint1"
  type="revolute">
  <origin
    xyz="0 0 0.1504"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="link1" />
  <axis
    xyz="0 0 1" />
```

```
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="40"
    velocity="5.8177" />
</joint>
<link
  name="link2">
  <inertial>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <mass
      value="0.093136" />
    <inertia
      ixx="0.0000046875"
      ixy="0"
      ixz="0"
      iyy="0.00007135"
      iyz="0"
      izz="0.00007417" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link2.STL" />
    </geometry>
    <material name="yellow"/>
  </visual>
  <collision>
```

```
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link2.STL" />
    </geometry>
  </collision>
</link>

<gazebo reference="link2">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Yellow</material>
</gazebo>
<joint
  name="joint2"
  type="revolute">
  <origin
    xyz="0.15 0 0.015"
    rpy="0 0 0" />
  <parent
    link="link1" />
  <child
    link="link2" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="30"
```

```
    velocity="5.8177" />
</joint>
<link
  name="link3">
  <inertial>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <mass
      value="0.023122" />
    <inertia
      ixx="0.0001677"
      ixy="0"
      ixz="0"
      iyy="0.0001677"
      iyz="0"
      izz="0.0000048" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link3.STL" />
    </geometry>
    <material name="green"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <geometry>
```

```
      <mesh
        filename="package://scaratwo/meshes/link3.STL" />
    </geometry>
  </collision>
</link>

<gazebo reference="link3">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Green</material>
</gazebo>

<joint
  name="joint3"
  type="prismatic">
  <origin
    xyz="0.1 0 0.065"
    rpy="-3.141592 0 0" />
  <parent
    link="link2" />
  <child
    link="link3" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="0"
    upper="0.05"
    effort="30"
    velocity="1.0" />
</joint>
```

```
  <transmission name="tran1">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="tran2">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <transmission name="tran3">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor3">
```

```
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

</robot>
```

- ✓ scara_gazebo_command.urdf

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This URDF was automatically created by SolidWorks to URDF Exporter! Originally created by Stephen Brawner (brawner@gmail.com)
    Commit Version: 1.5.1-0-g916b5db  Build Version: 1.5.7152.31018
    For more information, please see http://wiki.ros.org/sw_urdf_exporter -->
<robot
  name="scara">
  <material name="blue">
    <color rgba="0 0 0.8 1"/>
  </material>
  <material name="red">
    <color rgba="0.8 0 0 1"/>
  </material>
  <material name="green">
    <color rgba="0 0.8 0 1"/>
  </material>
  <material name="yellow">
    <color rgba="1 1 0.2 1"/>
  </material>

  <gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
      <robotNamespace>/</robotNamespace>
    </plugin>
  </gazebo>
```

```
  <gazebo>
    <plugin                                                    name="joint_state_publisher"
filename="libgazebo_ros_joint_state_publisher.so">
      <jointName>joint1, joint2, joint3</jointName>
    </plugin>
  </gazebo>
  <!-- * * * Link Definitions * * * -->
  <link
    name="world"/>
  <link
    name="base_link">
    <inertial>
      <origin
        xyz="0 0 0.0"
        rpy="0 0 0" />
      <mass
        value="9.9007" />
      <inertia
        ixx="0.23671"
        ixy="0"
        ixz="0"
        iyy="0.54727"
        iyz="0"
        izz="0.77689" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scara/meshes/base_link.STL" />
      </geometry>
```

```
    <material name="blue"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scara/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="base_link">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Blue</material>
</gazebo>
<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>
<link
  name="link1">
  <inertial>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <mass
      value="0.20189" />
    <inertia
      ixx="0.000005625"
```

```
      ixy="0"
      ixz="0"
      iyy="0.0008011"
      iyz="0"
      izz="0.0008045" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0.0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link1.STL" />
      </geometry>
      <material name="red"/>
    </visual>
    <collision>
      <origin
        xyz="0 0 0.0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link1.STL" />
      </geometry>
    </collision>
  </link>
  <gazebo reference="link1">
      <kp>1000.0</kp>
      <kd>10.0</kd>
      <mu1>10.0</mu1>
      <mu2>10.0</mu2>
      <material>Gazebo/Red</material>
  </gazebo>
```

```
<joint
  name="joint1"
  type="revolute">
  <origin
    xyz="0 0 0.1504"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="link1" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="40"
    velocity="5.8177" />
</joint>
<link
  name="link2">
  <inertial>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <mass
      value="0.093136" />
    <inertia
      ixx="0.0000046875"
      ixy="0"
      ixz="0"
      iyy="0.00007135"
      iyz="0"
      izz="0.00007417" />
```

```
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link2.STL" />
    </geometry>
    <material name="yellow"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link2.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="link2">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Yellow</material>
</gazebo>
<joint
  name="joint2"
  type="revolute">
  <origin
    xyz="0.15 0 0.015"
```

```
    rpy="0 0 0" />
  <parent
    link="link1" />
  <child
    link="link2" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="30"
    velocity="5.8177" />
</joint>
<link
  name="link3">
  <inertial>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <mass
      value="0.023122" />
    <inertia
      ixx="0.0001677"
      ixy="0"
      ixz="0"
      iyy="0.0001677"
      iyz="0"
      izz="0.0000048" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
```

```
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link3.STL" />
    </geometry>
    <material name="green"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link3.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="link3">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Green</material>
</gazebo>
<joint
  name="joint3"
  type="prismatic">
  <origin
    xyz="0.1 0 0.065"
    rpy="-3.141592 0 0" />
  <parent
    link="link2" />
  <child
    link="link3" />
```

```
    <axis
      xyz="0 0 1" />
    <limit
      lower="0"
      upper="0.05"
      effort="30"
      velocity="1.0" />
  </joint>

  <transmission name="tran1">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="joint1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      </joint>
      <actuator name="motor1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
          <mechanicalReduction>1</mechanicalReduction>
      </actuator>
  </transmission>
  <transmission name="tran2">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="joint2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      </joint>
      <actuator name="motor2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
          <mechanicalReduction>1</mechanicalReduction>
      </actuator>
  </transmission>
```

```
  <transmission name="tran3">
     <type>transmission_interface/SimpleTransmission</type>
     <joint name="joint3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
     </joint>
     <actuator name="motor3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
     </actuator>
  </transmission>
</robot>
```

- ✓ scara_joint_state.urdf

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This URDF was automatically created by SolidWorks to URDF Exporter! Originally created by Stephen Brawner (brawner@gmail.com)
     Commit Version: 1.5.1-0-g916b5db  Build Version: 1.5.7152.31018
     For more information, please see http://wiki.ros.org/sw_urdf_exporter -->
<robot
  name="scara">
  <material name="blue">
     <color rgba="0 0 0.8 1"/>
  </material>
  <material name="red">
     <color rgba="0.8 0 0 1"/>
  </material>
  <material name="green">
     <color rgba="0 0.8 0 1"/>
  </material>
  <material name="yellow">
     <color rgba="1 1 0.2 1"/>
  </material>
```

```
  <gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
      <robotNamespace>/</robotNamespace>
    </plugin>
  </gazebo>
  <!--
  <gazebo>
   <plugin                                                    name="joint_state_publisher"
filename="libgazebo_ros_joint_state_publisher.so">
    <jointName>joint1, joint2, joint3</jointName>
   </plugin>
  </gazebo> -->
  <!-- * * * Link Definitions * * * -->
  <link
   name="world"/>
  <link
   name="base_link">
   <inertial>
    <origin
     xyz="0 0 0.0"
     rpy="0 0 0" />
    <mass
     value="9.9007" />
    <inertia
     ixx="0.23671"
     ixy="0"
     ixz="0"
     iyy="0.54727"
     iyz="0"
     izz="0.77689" />
   </inertial>
   <visual>
    <origin
```

```
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scara/meshes/base_link.STL" />
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scara/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="base_link">
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Blue</material>
</gazebo>
<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>
<link
  name="link1">
  <inertial>
    <origin
```

```
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <mass
      value="0.20189" />
    <inertia
      ixx="0.000005625"
      ixy="0"
      ixz="0"
      iyy="0.0008011"
      iyz="0"
      izz="0.0008045" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link1.STL" />
    </geometry>
    <material name="red"/>
  </visual>
  <collision>
    <origin
      xyz="0 0 0.0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://scaratwo/meshes/link1.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="link1">
```

```
    <kp>1000.0</kp>
    <kd>10.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Red</material>
</gazebo>
<joint
  name="joint1"
  type="revolute">
  <origin
    xyz="0 0 0.1504"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="link1" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="40"
    velocity="5.8177" />
</joint>
<link
  name="link2">
  <inertial>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <mass
      value="0.093136" />
    <inertia
```

```
      ixx="0.0000046875"
      ixy="0"
      ixz="0"
      iyy="0.00007135"
      iyz="0"
      izz="0.00007417" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link2.STL" />
      </geometry>
      <material name="yellow"/>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link2.STL" />
      </geometry>
    </collision>
  </link>
  <gazebo reference="link2">
      <kp>1000.0</kp>
      <kd>10.0</kd>
      <mu1>10.0</mu1>
      <mu2>10.0</mu2>
      <material>Gazebo/Yellow</material>
```

```
</gazebo>

<joint
  name="joint2"
  type="revolute">
  <origin
    xyz="0.15 0 0.015"
    rpy="0 0 0" />
  <parent
    link="link1" />
  <child
    link="link2" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-2.0943"
    upper="2.0943"
    effort="30"
    velocity="5.8177" />
</joint>

<link
  name="link3">
  <inertial>
    <origin
      xyz="0 0 -0.065"
      rpy="0 0 0" />
    <mass
      value="0.023122" />
    <inertia
      ixx="0.0001677"
      ixy="0"
      ixz="0"
```

```
      iyy="0.0001677"
      iyz="0"
      izz="0.0000048" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 -0.065"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link3.STL" />
      </geometry>
      <material name="green"/>
    </visual>
    <collision>
      <origin
        xyz="0 0 -0.065"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://scaratwo/meshes/link3.STL" />
      </geometry>
    </collision>
  </link>
  <gazebo reference="link3">
      <kp>1000.0</kp>
      <kd>10.0</kd>
      <mu1>10.0</mu1>
      <mu2>10.0</mu2>
      <material>Gazebo/Green</material>
  </gazebo>
  <joint
    name="joint3"
```

```
    type="prismatic">
    <origin
      xyz="0.1 0 0.065"
      rpy="-3.141592 0 0" />
    <parent
      link="link2" />
    <child
      link="link3" />
    <axis
      xyz="0 0 1" />
    <limit
      lower="0"
      upper="0.05"
      effort="30"
      velocity="1.0" />
  </joint>
  <transmission name="tran1">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="joint1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      </joint>
      <actuator name="motor1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
          <mechanicalReduction>1</mechanicalReduction>
      </actuator>
  </transmission>
  <transmission name="tran2">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="joint2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
```

```
    </joint>
    <actuator name="motor2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>
  <transmission name="tran3">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="motor3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>
</robot>
```

Config

- ✓ joint_names_scara.yaml

```
controller_joint_names: ['', 'joint1', 'joint2', 'joint3', ]
```

- ✓ ros_controllers.yaml

```
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
    - joint1
    - joint2
    - joint3
```

```
  gains:
    joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint2:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint3:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
```

Launch

- ✓ check_motors.launch

```
<?xml version="1.0"?>
<launch>
  <!-- upload urdf -->
  <param name="robot_description" textfile="$(find scaratwo)/urdf/scara.urdf" />
  <!-- Combine joint values -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
  <!-- Show in Rviz   -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find scaratwo)/launch/config.rviz" />
  <!-- send joint values -->
  <arg name="use_gui" default="true" doc="Should the joint_state_publisher use a GUI for controlling joint states" />
```

```
 <node pkg="joint_state_publisher" type="joint_state_publisher" name="joint_state_publisher" output="screen" unless="$(arg use_gui)" />
 <node pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" name="joint_state_publisher_gui" output="screen" if="$(arg use_gui)" />
 <!--Run serial_node.py for activate arduino-Ros communaction-->
 <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
      <param name="port" value="/dev/ttyACM0"/>
      <param name="baud" value="115200"/>
 </node>
</launch>
```

- ✓ check_motors_gazebo.launch

```
<?xml version="1.0"?>
<launch>
 <arg name="paused" default="false"/>
 <arg name="gazebo_gui" default="true"/>
 <arg name="urdf_path" default="$(find scaratwo)/urdf/scara.urdf"/>
 <!-- startup simulated world -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" default="worlds/empty.world"/>
  <arg name="paused" value="$(arg paused)"/>
  <arg name="gui" value="$(arg gazebo_gui)"/>
 </include>
 <!-- send robot urdf to param server -->
 <param name="robot_description" textfile="$(arg urdf_path)" />
 <!-- push robot_description to factory and spawn robot in gazebo at the origin, change x,y,z arguments to spawn in a different position -->
 <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
  respawn="false" output="screen" />
 <!-- Load joint controller configurations from YAML file to parameter server -->
 <rosparam file="$(find scaratwo)/config/ros_controllers.yaml" command="load"/>
 <!-- Load the controllers -->
```

```
 <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
   output="screen" args="arm_controller "/>
</launch>
```

- ✓ check_motors_gazebo_command_control.launch

```
<?xml version="1.0"?>
<launch>
 <!-- upload urdf -->
 <param name="robot_description" textfile="$(find scaratwo)/urdf/scara_gazebo_command.urdf" />
 <!-- Combine joint values -->
 <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
 <!-- Show in Rviz   -->
 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find scaratwo)/launch/config.rviz" />
</launch>
```

- ✓ check_motors_gazebo_command_control_gazebo.launch

```
<?xml version="1.0"?>
<launch>
 <arg name="paused" default="false"/>
 <arg name="gazebo_gui" default="true"/>
 <arg name="urdf_path" default="$(find scaratwo)/urdf/scara_gazebo_command.urdf"/>
 <!-- startup simulated world -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
   <arg name="world_name" default="worlds/empty.world"/>
   <arg name="paused" value="$(arg paused)"/>
   <arg name="gui" value="$(arg gazebo_gui)"/>
 </include>
 <!-- send robot urdf to param server -->
 <param name="robot_description" textfile="$(arg urdf_path)" />
```

```
  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find scaratwo)/config/ros_controllers.yaml" command="load"/>
  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" args="arm_controller "/>
</launch>
```

- ✓ check_motors_joint_state_control.launch

```
<?xml version="1.0"?>
<launch>
  <!-- upload urdf -->
  <param name="robot_description" textfile="$(find scaratwo)/urdf/scara_joint_state.urdf" />
  <!-- Combine joint values -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
  <!-- Show in Rviz   -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find scaratwo)/launch/config.rviz" />
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
</launch>
```

- ✓ check_motors_joint_state_control_gazebo.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
```

```
  <arg name="urdf_path" default="$(find scaratwo)/urdf/scara_joint_state.urdf"/>
  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" default="worlds/empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>
  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />
  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change
x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find scaratwo)/config/ros_controllers.yaml" command="load"/>
  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false"
    output="screen" args="arm_controller "/>
</launch>
```

- ✓ gazebo_command_all.launch

```
<?xml version="1.0"?>
<launch>
  <!--- Run configured Rviz with gui,robot_state_pub. and joint_state_publisher -->
  <include                                                    file="$(find
scaratwo)/launch/check_motors_gazebo_command_control.launch" />
  <!--- Run gazebo with custom controller -->
  <include                                                    file="$(find
scaratwo)/launch/check_motors_gazebo_command_control_gazebo.launch" />
  <!--Run serial_node.py for activate arduino-Ros communaction-->
  <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
        <param name="port" value="/dev/ttyACM0"/>
```

```
        <param name="baud" value="115200"/>
  </node>
</launch>
```

- ✓ gazebo_moveit.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
  <arg name="urdf_path" default="$(find scaratwo)/urdf/scara.urdf"/>

  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" default="worlds/empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>
  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />
  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change
x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find scaratwo)/config/ros_controllers.yaml" command="load"/>
  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false"
    output="screen" args="arm_controller joint_state_controller"/>
</launch>
```

- ✓ gazebo_urdf.launch

```
<?xml version="1.0"?>
<launch>
```

```
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
  <arg name="urdf_path" default="$(find scaratwo)/urdf/scara.urdf"/>
  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" default="worlds/empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>
  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />
  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change
x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
</launch>
```

- ✓ joint_state_all.launch

```
<?xml version="1.0"?>
<launch>
  <!--Firstly we have to run a script that give us path point as a list or from Solidworks
     and then we must run pub_joint2.py script to execute path
  -->
  <!--- Run configured Rviz with gui,robot_state_pub. and joint_state_publisher -->
  <include file="$(find scaratwo)/launch/check_motors_joint_state_control.launch" />
  <!--- Run gazebo with custom controller -->
  <include file="$(find
scaratwo)/launch/check_motors_joint_state_control_gazebo.launch" />
  <!--Run joint_states_to_gazebo.py for send joint value to change gazebo-->
  <node pkg="scaratwo" type="joint_states_to_gazebo.py" name="states"
output="screen"/>
  <!--Run serial_node.py for activate arduino-Ros communaction-->
  <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
```

```
        <param name="port" value="/dev/ttyACM0"/>
        <param name="baud" value="115200"/>
  </node>
</launch>
```

- ✓ rviz_all.launch

```
<?xml version="1.0"?>
<launch>
  <!--- Run configured Rviz with gui,robot_state_pub. and joint_state_publisher -->
  <include file="$(find scaratwo)/launch/check_motors.launch" />
  <!--- Run gazebo with custom controller -->
  <include file="$(find scaratwo)/launch/check_motors_gazebo.launch" />
  <!--Run joint_states_to_gazebo.py for send joint value to change gazebo-->
  <node pkg="scaratwo" type="joint_states_to_gazebo.py" name="states"
output="screen"/>
  <!--Run serial_node.py for activate arduino-Ros communaction-->
  <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
        <param name="port" value="/dev/ttyACM0"/>
        <param name="baud" value="115200"/>
  </node>
</launch>
```

- ✓ rviz_urdf.launch

```
<?xml version="1.0"?>
<launch>
  <!-- upload urdf -->
  <param name="robot_description" textfile="$(find scaratwo)/urdf/scara.urdf" />
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" >
    <param name="use_gui" value="true"/>
  </node>
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />
  <!-- Show in Rviz   -->
```

```
 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find scaratwo)/launch/urdf_config.rviz" />
</launch>
```

Scripts

- ✓ joint_states_to_gazebo.py

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.position)
    pub = rospy.Publisher('arm_controller/command', JointTrajectory, queue_size=10)
    joints_str = JointTrajectory()
    joints_str.header = Header()
    joints_str.header.stamp = rospy.Time.now()
    joints_str.joint_names = ['joint1', 'joint2','joint3']
    point=JointTrajectoryPoint()
    point.positions = [data.position[0], data.position[1],data.position[2]]
    point.time_from_start = rospy.Duration(2)
    joints_str.points.append(point)
    pub.publish(joints_str)
    rospy.loginfo("position updated")
def listener():
    rospy.init_node('states', anonymous=True)
    rospy.Subscriber("joint_states", JointState, callback)
    rospy.spin()
if __name__ == '__main__':
    try:
       listener()
    except rospy.ROSInterruptException:
```

pass

- ✓ path_generator_v5_1.py

```
#! /usr/bin/env python
import math
import matplotlib.pyplot as plt
import numpy as np
#import smbus
import sys
from sympy import *
from sympy import symbols, Eq, solve
# import sympy, Point and Circle
from sympy import Point, Circle
class Path():
    def __init__(self):
        self.m1 = 0.1 #birinci kol agirligi
        self.m2 = 0.2 #ikinci kol agirligi
        self.g = 0
        self.l1 = 150.0  #birinci kol uzunlugu
        self.l2 = 100.0  #ikinci kol uzunlugu
        self.q1 = 0
        self.q2 = 0
        self.x = 0.000
        self.y = 0.000
        self.q12 = None #ilk kol acisi
        self.q32 = None #ikinci kol acisi total
        self.q22 = None #ikinci kol acisi real
        self.c = self.l1 + self.l2  #kol uzunluklari toplami
        self.d = self.l1 - self.l2  #kol uzunluklari farki
        #polinom denklem katsayilari
        self.a1 = None
        self.b1 = None
        self.c1 = None
        self.d1 = None
```

```
        self.a2 = None
        self.b2 = None
        self.c2 = None
        self.d2 = None
        self.a3 = None
        self.b3 = None
        self.c3 = None
        self.d3 = None
        self.a4 = None
        self.b4 = None
        self.c4 = None
        self.d4 = None
        self.i=1
        self.x1=None
        self.y1 = None
        self.x2 = None
        self.y2 = None
         self.WW2= None
         self.in_min= 0.0
         self.in_max= 240.0
         self.out_min= 0.0
         self.out_max= 0.72
         self.yaricap=None
        open('path1.txt', 'w').close()  #clear txt file each code runing
    def path_matrix(self):
         tt1=[];tt2=[]
        ti=0
        tm=self.sonuc_sure/2
         tf=self.sonuc_sure
        A_matrix=np.array([[ti**3 ,ti**2 ,ti, 1, 0 ,0, 0, 0,0 ,0, 0, 0,0 ,0, 0, 0],[0 ,0, 0,
0,ti**3,ti**2,ti,1,0 ,0, 0, 0,0 ,0, 0, 0],
                           [3*ti**2,2*ti,1,0,0 ,0, 0, 0,0 ,0, 0, 0,0 ,0, 0, 0],[0 ,0, 0,
0,3*ti**2,2*ti,1,0,0 ,0, 0, 0,0 ,0, 0, 0],
```

```
                [tm**3,tm**2,tm,1,0 ,0, 0, 0,0,0,0,0,0 ,0, 0, 0],[0 ,0, 0,
0,tm**3,tm**2,tm,1,0 ,0, 0, 0,0,0,0,0],
                    [3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0, 0, 0,
0, 0], [ 0, 0, 0, 0,3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0],
                [6*tm,2,0 ,0, 0, 0,0,0,-6*tm,-2,0,0,0 ,0, 0, 0],[0 ,0, 0, 0,6*tm,2,0 ,0, 0,
0,0,0,-6*tm,-2,0,0],
                [0 ,0, 0, 0,0 ,0, 0, 0,tf**3,tf**2,tf,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,3*tf**2,2*tf,1,0,0 ,0, 0, 0],
                [0 ,0, 0, 0,0,0 ,0, 0, 0,0 ,0, 0,tf**3,tf**2,tf,1], [0 ,0, 0, 0,0,0 ,0, 0, 0,0
,0, 0,3*tf**2,2*tf,1,0],
                [0,0,0,0,0,0,0,0,0,tm**3,tm**2,tm,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,0,0,0,0,0,tm**3,tm**2,tm,1]])

B_matrix=np.array([[self.a1],[self.b1],[self.c1],[self.d1],[self.a2],[self.b2],[self.c2],[s
elf.d2],

[self.a3],[self.b3],[self.c3],[self.d3],[self.a4],[self.b4],[self.c4],[self.d4]])

C_matrix=([[self.K],[self.OO],[0],[0],[self.M],[self.P],[0],[0],[0],[0],[self.C],[0],[self
.E],[0],[self.M],[self.P]])
        A_matrix_inv=np.linalg.inv(A_matrix)
        R_matrix=np.matmul(A_matrix_inv,C_matrix)
         # Convert 2D Numpy array to a flat list
         num_list = R_matrix.flatten().tolist()
         #print(num_list)
         self.a1=round(num_list[0],2)
         self.b1=round(num_list[1],2)
         self.c1=round(num_list[2],2)
         self.d1=round(num_list[3],2)
         self.a2=round(num_list[4],2)
         self.b2=round(num_list[5],2)
         self.c2=round(num_list[6],2)
         self.d2=round(num_list[7],2)
```

```
        self.a3=round(num_list[8],2)
        self.b3=round(num_list[9],2)
        self.c3=round(num_list[10],2)
       self.d3=round(num_list[11],2)
        self.a4=round(num_list[12],2)
        self.b4=round(num_list[13],2)
        self.c4=round(num_list[14],2)
        self.d4=round(num_list[15],2)
        #print(self.a1,self.b1,self.c1,self.d1)
        #print(self.a3,self.b3,self.c3,self.d3)
        for ti in np.arange(0.0,(self.sonuc_sure/2)+0.001,0.003,dtype=float):
                ti=round(ti,3)
                #print("ti =",ti)
                Q11=self.a1*ti**3+self.b1*ti**2+self.c1*ti+self.d1
                Q21=self.a2*ti**3+self.b2*ti**2+self.c2*ti+self.d2
                tt1.append(round(Q11,2))
                tt2.append(round(Q21,2))
        for tf in np.arange(self.sonuc_sure/2,self.sonuc_sure+0.001, 0.003,dtype=float):
                tf=round(tf,3)
                #print("tf =",tf)
                Q12=self.a3*tf**3+self.b3*tf**2+self.c3*tf+self.d3
                Q22=self.a4*tf**3+self.b4*tf**2+self.c4*tf+self.d4
                tt1.append(round(Q12,2))
                tt2.append(round(Q22,2))
        for s in np.arange(0,((self.sonuc_sure/0.003)+1), 1,dtype=int):
                self.fk(tt1[s],tt2[s])
                #print(tt1[s],tt2[s])
            f = open("path1.txt", "a")
            f.write("%.3f %s %.3f %s "
%((math.radians(tt1[s])),(","),(math.radians(tt2[s])),(",")) )
            f.close()
    def ik(self):
        A = 2.0 * self.x * self.l1
```

```
        B = 2.0 * self.y * self.l1
        C = self.x ** 2 + self.y ** 2 + self.l1 ** 2 - self.l2 ** 2
        ik_ok_durum=(self.x) ** 2 + (self.y) ** 2 <=self.c ** 2 and (self.x) ** 2 + (self.y) ** 2 >= self.d ** 2
        if ((C + A != 0) and ik_ok_durum):
            # elbow-up configuration
            self.q12 = 2.0 * math.degrees(math.atan((B - math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (C + A)))
            # elbow-up configuration
            self.q32 = math.degrees(math.atan2((self.y - self.l1 * math.sin(math.radians(self.q12))), (self.x - self.l1 * math.cos(math.radians(self.q12)))))
            # elbow-up configuration
            self.q22 = self.q32 - self.q12
            #print("elbow-up   configuration","q12="+str(self.q12),"q22="+str(self.q22))
            if ((self.q32 - self.q12 == 0) or (abs(self.q32 - self.q12) == 180)):
                plt.scatter(self.x, self.y, color='red')
                # plt.show()
                print("Singularity C+A!=0")
        elif (C + A == 0):
            if (A ** 2 + B ** 2 != 0):
                e11 = (B * C + A * math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (A ** 2 + B ** 2)
                e22 = (A * C - B * math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (A ** 2 + B ** 2)
                self.q12 = math.degrees(math.atan2(e11, e22))
                self.q12 = round(self.q12, 2)
                #print("q12="+str(self.q12))
                self.q32 = math.degrees(math.atan2((self.y - self.l1 * math.sin(math.radians(self.q12))), (self.x - self.l1 * math.cos(math.radians(self.q12)))))
                self.q32 = round(self.q32, 2)
                self.q22 = self.q32 - self.q12
                #print("q22="+str(self.q22))
```

```
            if ((abs(self.q32) - abs(self.q12) == 0) or (abs(self.q32) - abs(self.q12) ==
180)):
                plt.scatter(self.x, self.y, color='red')
                # plt.show()
                print("Singularity C+A=0")
        else:
            plt.scatter(self.x, self.y, color='red')
            # plt.show()
            print("Gidilemeyen Nokta")
        if self.ii == 1:
            self.K = round(self.q12, 3)
            self.OO = round(self.q22, 3)
            #print("K =",self.K )
            #print("OO =", self.OO)
        elif self.ii == 2:
            self.C = round(self.q12, 3)
            self.E = round(self.q22, 3)
             #print("C =",self.C)
             #print("E =",self.E)
        elif self.ii == 3:
            self.M = round(self.q12, 3)
            self.P = round(self.q22, 3)
            #print("M =",self.M )
             #print("P =",self.P )
    def via_point_finder(self):
        saved_array= []
        prompt = "-> "
        line = raw_input(prompt)
        while line:
                saved_array.append(int(line))
                line = raw_input(prompt)
                #print("saved_array =",len(saved_array))
        for self.i in np.arange(0,len(saved_array)-2, 2,dtype=int):
```

```
                self.x1=saved_array[self.i]
                self.y1=saved_array[self.i+1]
                self.x=self.x1
                self.y=self.y1
                self.ii=1
                self.ik()
            self.x2=saved_array[self.i+2]
                self.y2=saved_array[self.i+3]
                self.x=self.x2
                self.y=self.y2
                self.ii=2
                self.ik()
                #print(self.x1,self.y1,self.x2,self.y2)
                self.WW2 = math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
                #print("WW2",self.WW2)
                self.yaricap=150.0
                taranan_aci=2*math.degrees(math.asin((self.WW2/2)/150.0))
                taranan_cevre=round((taranan_aci/360.0)*(2*math.pi*self.yaricap),3)
                #print("taranan_cevre =",taranan_cevre)
                self.map(taranan_aci)
                via_point_angle = 60  ##bending angle (it can be change)
                L2 = (self.WW2 * math.cos(math.radians(via_point_angle)))  ##midpoint
of distance
                L2 = round(L2, 2)
                CC21 = L2 * 2  ##radius of 1st circle
                CC22 = L2 * 2  ##radius for 2nd circle
                # using Circle()
                c1 = Circle(Point(self.x1, self.y1), CC21)
                c2 = c1.equation()  ##first circle equ
                c3 = Circle(Point(self.x2, self.y2), CC22)
                c4 = c3.equation()  ##second circle equ

                solution2 = solve([c2, c4])  ##system equations solution
```

```
                S21 = solution2[0]
                S22 = solution2[1]

                VI2 = [list(S21.values()), list(S22.values())]  ##values for both solution
                ##first solution
                self.y22 = round(VI2[0][0],1)
                self.x22 = round(VI2[0][1],1)
                #print("via_point x =", self.x22)
                #print("via_point y =", self.y22)
                self.ii = 3
                if self.ii==3:
                        self.x=self.x22
                        self.y=self.y22
                        self.ik()
                self.path_matrix()
                ######via point has finish###
#forward kinematics
    def fk(self,q1,q2):
        x=self.l1*math.cos(math.radians(q1))+self.l2*math.cos(math.radians(q1+q2))
        y=self.l1*math.sin(math.radians(q1))+self.l2*math.sin(math.radians(q1+q2))
        #print("x ="+str(x),"y ="+str(y))
        plt.scatter(x,y,color='red')
#Prominent Arduino map function :)
    def map(self,x):
         self.sonuc_sure=round(float((x - self.in_min) * (self.out_max - self.out_min) /
(self.in_max - self.in_min) + self.out_min),3)
         #print("sonuc =",self.sonuc_sure)
nesne=Path()
nesne.via_point_finder()
#nesne.path_matrix()
plt.show()
```

- ✓ path_generator_v5_2.py

```
#! /usr/bin/env python
```

```
import math
import matplotlib.pyplot as plt
import numpy as np
#import smbus
import sys
from sympy import *
from sympy import symbols, Eq, solve
# import sympy, Point and Circle
from sympy import Point, Circle
class Path():
    def __init__(self):
        self.m1 = 0.1 #birinci kol agirligi
        self.m2 = 0.2 #ikinci kol agirligi
        self.g = 0
        self.l1 = 150.0  #birinci kol uzunlugu
        self.l2 = 100.0  #ikinci kol uzunlugu
        self.q1 = 0
        self.q2 = 0
        self.x = 0.000
        self.y = 0.000
        self.q12 = None #ilk kol acisi
        self.q32 = None #ikinci kol acisi total
        self.q22 = None #ikinci kol acisi real
        self.c = self.l1 + self.l2  #kol uzunluklari toplami
        self.d = self.l1 - self.l2  #kol uzunluklari farki
        #polinom denklem katsayilari
        self.a1 = None
        self.b1 = None
        self.c1 = None
        self.d1 = None

        self.a2 = None
        self.b2 = None
```

```
        self.c2 = None
        self.d2 = None
        self.a3 = None
        self.b3 = None
        self.c3 = None
        self.d3 = None
        self.a4 = None
        self.b4 = None
        self.c4 = None
        self.d4 = None
        self.i=1
        self.x1=None
        self.y1 = None
        self.x2 = None
        self.y2 = None
         self.WW2= None
         self.in_min= 0.0
         self.in_max= 240.0
         self.out_min= 0.0
         self.out_max= 0.72
         self.yaricap=None
        open('path2.txt', 'w').close()  #clear txt file each code runing
    def path_matrix(self):
         tt1=[];tt2=[]
        ti=0
        tm=self.sonuc_sure/2
         tf=self.sonuc_sure
        A_matrix=np.array([[ti**3 ,ti**2 ,ti, 1, 0 ,0, 0, 0,0 ,0, 0, 0,0 ,0, 0, 0],[0 ,0, 0,
0,ti**3,ti**2,ti,1,0 ,0, 0, 0,0 ,0, 0, 0],
                           [3*ti**2,2*ti,1,0,0   ,0,   0,   0,0   ,0,   0,   0,0   ,0,   0,   0],[0   ,0,   0,
0,3*ti**2,2*ti,1,0,0 ,0, 0, 0,0 ,0, 0, 0],
                           [tm**3,tm**2,tm,1,0   ,0,   0,   0,0,0,0,0,0,0   ,0,   0,   0],[0   ,0,   0,
0,tm**3,tm**2,tm,1,0 ,0, 0, 0,0,0,0,0,0],
```

```
                [3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0, 0, 0,
0, 0], [ 0, 0, 0, 0,3 * tm ** 2, 2 * tm, 1, 0, 0, 0, 0, 0,-3 * tm ** 2, -2 *tm,-1, 0],
                [6*tm,2,0 ,0, 0, 0,0,0,-6*tm,-2,0,0,0 ,0, 0, 0],[0 ,0, 0, 0,6*tm,2,0 ,0, 0,
0,0,0,-6*tm,-2,0,0],
                [0 ,0, 0, 0,0 ,0, 0, 0,tf**3,tf**2,tf,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,3*tf**2,2*tf,1,0,0 ,0, 0, 0],
                [0 ,0, 0, 0,0,0 ,0, 0, 0,0 ,0, 0,tf**3,tf**2,tf,1], [0 ,0, 0, 0,0,0 ,0, 0, 0,0
,0, 0,3*tf**2,2*tf,1,0],
                [0,0,0,0,0,0,0,0,0,tm**3,tm**2,tm,1,0 ,0, 0, 0],[0 ,0, 0, 0,0 ,0, 0,
0,0,0,0,0,tm**3,tm**2,tm,1]])

B_matrix=np.array([[self.a1],[self.b1],[self.c1],[self.d1],[self.a2],[self.b2],[self.c2],[s
elf.d2],

[self.a3],[self.b3],[self.c3],[self.d3],[self.a4],[self.b4],[self.c4],[self.d4]])

C_matrix=([[self.K],[self.OO],[0],[0],[self.M],[self.P],[0],[0],[0],[0],[self.C],[0],[self
.E],[0],[self.M],[self.P]])
        A_matrix_inv=np.linalg.inv(A_matrix)
        R_matrix=np.matmul(A_matrix_inv,C_matrix)
         # Convert 2D Numpy array to a flat list
         num_list = R_matrix.flatten().tolist()
         #print(num_list)
         self.a1=round(num_list[0],2)
         self.b1=round(num_list[1],2)
         self.c1=round(num_list[2],2)
         self.d1=round(num_list[3],2)
         self.a2=round(num_list[4],2)
         self.b2=round(num_list[5],2)
         self.c2=round(num_list[6],2)
         self.d2=round(num_list[7],2)
         self.a3=round(num_list[8],2)
         self.b3=round(num_list[9],2)
```

```
        self.c3=round(num_list[10],2)
       self.d3=round(num_list[11],2)
        self.a4=round(num_list[12],2)
        self.b4=round(num_list[13],2)
        self.c4=round(num_list[14],2)
        self.d4=round(num_list[15],2)
        #print(self.a1,self.b1,self.c1,self.d1)
        #print(self.a3,self.b3,self.c3,self.d3)
        for ti in np.arange(0.0,(self.sonuc_sure/2)+0.001,0.003,dtype=float):
                ti=round(ti,3)
                #print("ti =",ti)
                Q11=self.a1*ti**3+self.b1*ti**2+self.c1*ti+self.d1
                Q21=self.a2*ti**3+self.b2*ti**2+self.c2*ti+self.d2
                tt1.append(round(Q11,2))
                tt2.append(round(Q21,2))
        for tf in np.arange(self.sonuc_sure/2,self.sonuc_sure+0.001, 0.003,dtype=float):
                tf=round(tf,3)
                #print("tf =",tf)
                Q12=self.a3*tf**3+self.b3*tf**2+self.c3*tf+self.d3
                Q22=self.a4*tf**3+self.b4*tf**2+self.c4*tf+self.d4
                tt1.append(round(Q12,2))
                tt2.append(round(Q22,2))
        for s in np.arange(0,((self.sonuc_sure/0.003)+1), 1,dtype=int):
                self.fk(tt1[s],tt2[s])
                #print(tt1[s],tt2[s])
            f = open("path2.txt", "a")
                f.write("%.3f\n"%(math.radians(tt1[s])))
                f.write("%.3f\n"%(math.radians(tt2[s])))
            f.close()
    def ik(self):
        A = 2.0 * self.x * self.l1
        B = 2.0 * self.y * self.l1
        C = self.x ** 2 + self.y ** 2 + self.l1 ** 2 - self.l2 ** 2
```

```
        ik_ok_durum=(self.x) ** 2 + (self.y) ** 2 <=self.c ** 2 and (self.x) ** 2 + (self.y) ** 2 >= self.d ** 2
        if ((C + A != 0) and ik_ok_durum):
            # elbow-up configuration
            self.q12 = 2.0 * math.degrees(math.atan((B - math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (C + A)))
            # elbow-up configuration
            self.q32 = math.degrees(math.atan2((self.y - self.l1 * math.sin(math.radians(self.q12))), (self.x - self.l1 * math.cos(math.radians(self.q12)))))
            # elbow-up configuration
            self.q22 = self.q32 - self.q12
            #print("elbow-up  configuration","q12="+str(self.q12),"q22="+str(self.q22))
            if ((self.q32 - self.q12 == 0) or (abs(self.q32 - self.q12) == 180)):
                plt.scatter(self.x, self.y, color='red')
                # plt.show()
                print("Singularity C+A!=0")
        elif (C + A == 0):
            if (A ** 2 + B ** 2 != 0):
                e11 = (B * C + A * math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (A ** 2 + B ** 2)
                e22 = (A * C - B * math.sqrt(A ** 2 + B ** 2 - C ** 2)) / (A ** 2 + B ** 2)
                self.q12 = math.degrees(math.atan2(e11, e22))
                self.q12 = round(self.q12, 2)
                #print("q12="+str(self.q12))
                self.q32 = math.degrees(math.atan2((self.y - self.l1 * math.sin(math.radians(self.q12))), (self.x - self.l1 * math.cos(math.radians(self.q12)))))
                self.q32 = round(self.q32, 2)
                self.q22 = self.q32 - self.q12
                #print("q22="+str(self.q22))
                if ((abs(self.q32) - abs(self.q12) == 0) or (abs(self.q32) - abs(self.q12) == 180)):
                    plt.scatter(self.x, self.y, color='red')
```

```
                # plt.show()
                print("Singularity C+A=0")
    else:
        plt.scatter(self.x, self.y, color='red')
        # plt.show()
        print("Gidilemeyen Nokta")
    if self.ii == 1:
        self.K = round(self.q12, 3)
        self.OO = round(self.q22, 3)
        #print("K =",self.K )
        #print("OO =", self.OO)
    elif self.ii == 2:
        self.C = round(self.q12, 3)
        self.E = round(self.q22, 3)
         #print("C =",self.C)
         #print("E =",self.E)
    elif self.ii == 3:
        self.M = round(self.q12, 3)
        self.P = round(self.q22, 3)
        #print("M =",self.M )
         #print("P =",self.P )
def via_point_finder(self):
     saved_array= []
     prompt = "-> "
     line = raw_input(prompt)
     while line:
                saved_array.append(int(line))
                line = raw_input(prompt)
                #print("saved_array =",len(saved_array))
     for self.i in np.arange(0,len(saved_array)-2, 2,dtype=int):
                self.x1=saved_array[self.i]
                self.y1=saved_array[self.i+1]
                self.x=self.x1
```

```
                self.y=self.y1
                self.ii=1
                self.ik()
            self.x2=saved_array[self.i+2]
                self.y2=saved_array[self.i+3]
                self.x=self.x2
                self.y=self.y2
                self.ii=2
                self.ik()
                #print(self.x1,self.y1,self.x2,self.y2)
                self.WW2 = math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
                #print("WW2",self.WW2)
                self.yaricap=150.0
                taranan_aci=2*math.degrees(math.asin((self.WW2/2)/150.0))
                taranan_cevre=round((taranan_aci/360.0)*(2*math.pi*self.yaricap),3)
                #print("taranan_cevre =",taranan_cevre)
                self.map(taranan_aci)
                via_point_angle = 60  ##bending angle (it can be change)
                L2 = (self.WW2 * math.cos(math.radians(via_point_angle)))  ##midpoint
of distance
                L2 = round(L2, 2)
                CC21 = L2 * 2  ##radius of 1st circle
                CC22 = L2 * 2  ##radius for 2nd circle
                # using Circle()
                c1 = Circle(Point(self.x1, self.y1), CC21)
                c2 = c1.equation()  ##first circle equ
                c3 = Circle(Point(self.x2, self.y2), CC22)
                c4 = c3.equation()  ##second circle equ
                solution2 = solve([c2, c4])  ##system equations solution
                S21 = solution2[0]
                S22 = solution2[1]
                VI2 = [list(S21.values()), list(S22.values())]  ##values for both solution
                ##first solution
```

```
                self.y22 = round(VI2[0][0],1)
                self.x22 = round(VI2[0][1],1)
                #print("via_point x =", self.x22)
                #print("via_point y =", self.y22)
                self.ii = 3
                if self.ii==3:
                        self.x=self.x22
                        self.y=self.y22
                        self.ik()
                self.path_matrix()
                ######via point has finish###
#forward kinematics
    def fk(self,q1,q2):
        x=self.l1*math.cos(math.radians(q1))+self.l2*math.cos(math.radians(q1+q2))
        y=self.l1*math.sin(math.radians(q1))+self.l2*math.sin(math.radians(q1+q2))
        #print("x ="+str(x),"y ="+str(y))
        plt.scatter(x,y,color='red')
#Prominent Arduino map function :)
    def map(self,x):
         self.sonuc_sure=round(float((x - self.in_min) * (self.out_max - self.out_min) /
(self.in_max - self.in_min) + self.out_min),3)
         #print("sonuc =",self.sonuc_sure)
nesne=Path()
nesne.via_point_finder()
#nesne.path_matrix()
plt.show()
```

- ✓ pub_joint.py

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
def talker():
    pub = rospy.Publisher('joint_states', JointState, queue_size=10)
```

```
    rospy.init_node('joint_state_publisher')
    rate = rospy.Rate(10) # 10hz
    hello_str = JointState()
    hello_str.header = Header()
    hello_str.name = ['joint1', 'joint2']
    hello_str.position = [0.00, 0.00,0.00]
    hello_str.velocity = []
    hello_str.effort = []
    while not rospy.is_shutdown():
      hello_str.header.stamp = rospy.Time.now()
      pub.publish(hello_str)
      rate.sleep()
if __name__ == '__main__':
    try:
       talker()
    except rospy.ROSInterruptException:
       pass
```

✓ pub_joint2.py

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
def talker(x,y):
    pub = rospy.Publisher('joint_states', JointState, queue_size=10)
    rospy.init_node('joint_state_publisher')
    rate = rospy.Rate(0.1) # 10hz
    hello_str = JointState()
    hello_str.header = Header()
    hello_str.name = ['joint1', 'joint2']
    hello_str.position = [x, y,0.00]
    hello_str.velocity = []
    hello_str.effort = []
    hello_str.header.stamp = rospy.Time.now()
```

```
    pub.publish(hello_str)
    rate.sleep()
if __name__ == '__main__':
    try:
        # Python3 code to iterate over a list
        list = [0,0,0.1,0.1,0.2,0.2,0.3,0.3,0.4,0.4,0.5,
0.5,0.6,0.6,0.7,0.7,0.8,0.8,0.9,0.9,1.0,1.0,1.1,1.1,1.2,1.2,1.3,1.3,1.4,1.4,1.5,1.5,1.6,1.6
,1.7,1.7,1.8,1.8,1.9,1.9,2,2,2.1,2.1,2.1,2.1]
       #list = [0,0]
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,2):
          x =list[i]
           y=list[i+1]
           print(x,y)
          talker(x,y)
    except rospy.ROSInterruptException:
       pass
```

- ✓ pub_joint2_deneme1.py

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
def talker(x,y):
    pub = rospy.Publisher('joint_states', JointState, queue_size=10)
    rospy.init_node('joint_state_publisher')
    rate = rospy.Rate(15) # 10hz
    hello_str = JointState()
    hello_str.header = Header()
    hello_str.name = ['joint1', 'joint2']
    hello_str.position = [x, y,0.0]
    hello_str.velocity = []
```

```
    hello_str.effort = []
    hello_str.header.stamp = rospy.Time.now()
    pub.publish(hello_str)
    rate.sleep()
if __name__ == '__main__':
    try:
        # opening the file in read mode
        my_file = open("path1.txt", "r")
        # reading the file
        data = my_file.read()
        print(data)
        # replacing end splitting the text
        # when newline ('\n' or ',') is seen.
        data_into_list = data.split(',')
       remove_item = data_into_list.pop(len(data_into_list)-1)
        print(data_into_list)
        list = []
        for element in data_into_list:
            list.append(float(element))
        print(list)
        my_file.close()
        # Python3 code to iterate over a list
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,2):
          x =list[i]
           y=list[i+1]
           print(x,y)
          talker(x,y)
    except rospy.ROSInterruptException:
        pass
```

- ✓ pub_joint2_deneme2.py

```
#!/usr/bin/env python
import math
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
def talker(x,y):
    pub = rospy.Publisher('joint_states', JointState, queue_size=10)
    rospy.init_node('joint_state_publisher')
    rate = rospy.Rate(15) # 10hz
    hello_str = JointState()
    hello_str.header = Header()
    hello_str.name = ['joint1', 'joint2']
    hello_str.position = [x, y,0.00]
    hello_str.velocity = []
    hello_str.effort = []
    hello_str.header.stamp = rospy.Time.now()
    pub.publish(hello_str)
    rate.sleep()
if __name__ == '__main__':
    try:
        # opening the file in read mode
        my_file = open("path2.txt", "r")
        # reading the file
        data = my_file.read()
        print(data)
        # replacing end splitting the text
        # when newline ('\n' or ',') is seen.
        data_into_list = data.split('\n')
        del data_into_list[len(data_into_list)-1]
        #print(data_into_list)
        list = []
        for element in data_into_list:
```

```
            list.append(float(element))
        #print(list)
        my_file.close()
        # Python3 code to iterate over a list
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,2):
          x =(list[i])*(math.pi/180.0)
           y=list[i+1]*(math.pi/180.0)
           #print(x,y)
          talker(x,y)
          if i==length-2:
               rate = rospy.Rate(1) # 10hz
                rate.sleep()
              talker(list[length-2]*(math.pi/180.0),list[length-1]*(math.pi/180.0))
    except rospy.ROSInterruptException:
        pass
```

✓ pub_joint2_deneme2_solid.py

```
#!/usr/bin/env python
import math
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
def talker(x,y):
    pub = rospy.Publisher('joint_states', JointState, queue_size=10)
    rospy.init_node('joint_state_publisher')
    rate = rospy.Rate(15) # 10hz
    hello_str = JointState()
    hello_str.header = Header()
    hello_str.name = ['joint1', 'joint2']
    hello_str.position = [x, y,0.00]
    hello_str.velocity = []
```

```
    hello_str.effort = []
    hello_str.header.stamp = rospy.Time.now()
    pub.publish(hello_str)
    rate.sleep()
if __name__ == '__main__':
    try:
        # opening the file in read mode
        my_file = open("path.txt", "r")
        # reading the file
        data = my_file.read()
        print(data)
        # replacing end splitting the text
        # when newline ('\n' or ',') is seen.
        data_into_list = data.split('\n')
        del data_into_list[len(data_into_list)-1]
        #print(data_into_list)
        list = []
        for element in data_into_list:
            list.append(float(element))
        #print(list)
        my_file.close()
        # Python3 code to iterate over a list
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,2):
          x =(list[i])*(math.pi/180.0)
           y=list[i+1]*(math.pi/180.0)
           #print(x,y)
          talker(x,y)
          if i==length-2:
                rate = rospy.Rate(1) # 10hz
                 rate.sleep()
```

```
            talker(list[length-2]*(math.pi/180.0),list[length-1]*(math.pi/180.0))
    except rospy.ROSInterruptException:
        pass
```

- ✓ scara_jointspace_trajectory.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("arm")
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory,queue_size=20)
# We can get the joint values from the group and adjust some of the values:
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = -pi/3
joint_goal[1] = -pi/3
# The go command can be called with joint values, poses, or without any
# parameters if you have already set the pose or joint target for the group
move_group.go(joint_goal, wait=True)
# Calling ``stop()`` ensures that there is no residual movement
move_group.stop()
```

- ✓ scara_opencv_oop2.py

```
#!/usr/bin/env python
import os
```

```
import rospy
import copy
import sys
import cv2
from std_msgs.msg import String
from geometry_msgs.msg import Pose
import numpy as np
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
import math
from std_msgs.msg import String
from std_msgs.msg import Int32
from moveit_commander.conversions import pose_to_list
from std_srvs.srv import SetBool
class NumberCounter(object):
    def __init__(self):
        self.counter = 0
        self.pub = rospy.Publisher("/gripper_state", Int32, queue_size=10)
        self.number_subscriber = rospy.Subscriber("/direction", Pose,
self.callback_number)
        self.reset_service = rospy.Service("/reset_counter", SetBool,
self.callback_reset_counter)

        super(NumberCounter, self).__init__()
        moveit_commander.roscpp_initialize(sys.argv)
        robot = moveit_commander.RobotCommander()
        scene = moveit_commander.PlanningSceneInterface()
        group_name = "arm"
        move_group = moveit_commander.MoveGroupCommander(group_name)
        display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
                                       moveit_msgs.msg.DisplayTrajectory,
```

```
                                    queue_size=20)

    self.robot = robot
    self.scene = scene
    self.move_group = move_group
    self.display_trajectory_publisher = display_trajectory_publisher
def callback_number(self, msg):
    move_group = self.move_group
    self.counterx = msg.position.x*1000
    self.countery = msg.position.y*1000
    list= [self.counterx,self.countery,0,250,0,1]
    length=len(list)
    new_msg = Int32()
    new_msg.data = self.counterx
    #rospy.init_node('aaa')
    rate1 = rospy.Rate(0.1) # 10hz

    for i in range(0,length,3):
        x=list[i]
        y=list[i+1]
        z=list[i+2]
        print(x,y,z)
        new_msg.data = z
        l1=150.0
        l2=100.0
        c=l1+l2
        d=l1-l2
         A=2.0*x*l1
         B=2.0*y*l1
         C=x**2+y**2+l1**2-l2**2
         if ((C+A!=0)):
                #elbow-down configuration
```

```
                q11=2.0*math.degrees(math.atan((B+math.sqrt(A**2+B**2-
C**2))/(C+A)))
                #elbow-up configuration
                q12=2.0*math.degrees(math.atan((B-math.sqrt(A**2+B**2-
C**2))/(C+A)))

                #elbow-down configuration
                q31=math.degrees(math.atan2((y-l1*math.sin(math.radians(q11))),(x-
l1*math.cos(math.radians(q11)))))
                #elbow-up configuration
                q32=math.degrees(math.atan2((y-l1*math.sin(math.radians(q12))),(x-
l1*math.cos(math.radians(q12)))))
                #elbow-down configuration
                q21=q31-q11
                #elbow-up configuration
                q22=q32-q12
                print("elbow-down configuration","q11="+str(q11),"q21="+str(q21))
                print("elbow-up   configuration","q12="+str(q12),"q22="+str(q22))
                if((q31-q11==0) or (abs(q31-q11)==180) or (q32-q12==0) or (abs(q32-
q12)==180)):
                        #plt.scatter(x,y,color='red')
                        #plt.show()
                        print("gidilmeyen nokta C+A!=0")
        elif(C+A==0):
                if (A**2+B**2!=0):
                        e1=(B*C-A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                        e11=(B*C+A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                        e2=(A*C+B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                        e22=(A*C-B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                        q11=math.degrees(math.atan2(e1,e2))
                        q12=math.degrees(math.atan2(e11,e22))
                        q11=round(q11,2)
                        q12=round(q12,2)
```

```
                    print("q11="+str(q11),"q12="+str(q12))

                    q31=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q11))),(x-l1*math.cos(math.radians(q11)))))
                    q32=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q12))),(x-l1*math.cos(math.radians(q12)))))
                    q31=round(q31,2)
                    q32=round(q32,2)
                    q21=q31-q11
                    q22=q32-q12
                    print("q31="+str(q31),"q32="+str(q32))
                    if((abs(q31)-abs(q11)==0) or (abs(q31)-abs(q11)==180) or
(abs(q32)-abs(q12)==0) or (abs(q32)-abs(q12)==180)):
                        #plt.scatter(x,y,color='red')
                        #plt.show()
                        print("gidilmeyen nokta C+A=0")
            else:
                #plt.scatter(x,y,color='red')
                #plt.show()
                print("gidilemeyen nokta")
            # We can get the joint values from the group and adjust some of the values:
            joint_goal = move_group.get_current_joint_values()
            joint_goal[0] = q12*(math.pi/180)
            joint_goal[1] = q22*(math.pi/180)
            # The go command can be called with joint values, poses, or without any
            # parameters if you have already set the pose or joint target for the group
            move_group.go(joint_goal, wait=True)
            # Calling ``stop()`` ensures that there is no residual movement
            move_group.stop()
            self.pub.publish(new_msg)
            rate1.sleep()
            #rate.sleep()
        #rate.sleep()
```

```
        sys.exit("finished")
    def callback_reset_counter(self, req):
        if req.data:
            self.counter = 0
            return True, "Counter has been successfully reset"
        return False, "Counter has not been reset"
if __name__ == '__main__':
    rospy.init_node('number_counter')
    NumberCounter()
    rospy.spin()
```

- ✓ test.py

```
import os
import sys
import time
while(1):
  #time.sleep(5)
  os.system('python ../scripts/scara_opencv_oop2.py')
  answer = input("ENTER something to quit: ")
  if(answer):
     break
```

- ✓ scara_trajectory.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
import math
from std_msgs.msg import String
from std_msgs.msg import Int32
from moveit_commander.conversions import pose_to_list
l1=150.0
```

```
l2=100.0
c=l1+l2
d=l1-l2
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("arm")
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory,queue_size=20)
rate = rospy.Rate(0.1) # 10hz
#inverse kinematics
def ik(x,y,z):
    pub = rospy.Publisher('gripper_state', Int32, queue_size=10)
    #rospy.init_node('aaa')
    rate1 = rospy.Rate(10) # 10hz
    rospy.loginfo("Publisher has been started.")
    msg = Int32()
    msg.data = z
     A=2.0*x*l1
     B=2.0*y*l1
     C=x**2+y**2+l1**2-l2**2
     if ((C+A!=0)):
            #elbow-down configuration
            q11=2.0*math.degrees(math.atan((B+math.sqrt(A**2+B**2-
C**2))/(C+A)))
            #elbow-up configuration
            q12=2.0*math.degrees(math.atan((B-math.sqrt(A**2+B**2-
C**2))/(C+A)))
            #elbow-down configuration
            q31=math.degrees(math.atan2((y-l1*math.sin(math.radians(q11))),(x-
l1*math.cos(math.radians(q11)))))
            #elbow-up configuration
```

```
            q32=math.degrees(math.atan2((y-l1*math.sin(math.radians(q12))),(x-
l1*math.cos(math.radians(q12)))))
            #elbow-down configuration
            q21=q31-q11
            #elbow-up configuration
            q22=q32-q12
            print("elbow-down configuration","q11="+str(q11),"q21="+str(q21))
            print("elbow-up   configuration","q12="+str(q12),"q22="+str(q22))
            if((q31-q11==0) or (abs(q31-q11)==180) or (q32-q12==0) or (abs(q32-
q12)==180)):
                  #plt.scatter(x,y,color='red')
                  #plt.show()
                  print("gidilmeyen nokta C+A!=0")
      elif(C+A==0):
            if (A**2+B**2!=0):
                  e1=(B*C-A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e11=(B*C+A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e2=(A*C+B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e22=(A*C-B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  q11=math.degrees(math.atan2(e1,e2))
                  q12=math.degrees(math.atan2(e11,e22))
                  q11=round(q11,2)
                  q12=round(q12,2)
                  print("q11="+str(q11),"q12="+str(q12))

                  q31=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q11))),(x-l1*math.cos(math.radians(q11)))))
                  q32=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q12))),(x-l1*math.cos(math.radians(q12)))))
                  q31=round(q31,2)
                  q32=round(q32,2)
                  q21=q31-q11
                  q22=q32-q12
```

```
                print("q31="+str(q31),"q32="+str(q32))
                if((abs(q31)-abs(q11)==0)  or  (abs(q31)-abs(q11)==180)  or
(abs(q32)-abs(q12)==0) or (abs(q32)-abs(q12)==180)):
                    #plt.scatter(x,y,color='red')
                    #plt.show()
                    print("gidilmeyen nokta C+A=0")
        else:
            #plt.scatter(x,y,color='red')
            #plt.show()
            print("gidilemeyen nokta")

        # We can get the joint values from the group and adjust some of the values:
        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = q12*(math.pi/180)
        joint_goal[1] = q22*(math.pi/180)

        # The go command can be called with joint values, poses, or without any
        # parameters if you have already set the pose or joint target for the group
        move_group.go(joint_goal, wait=True)

        # Calling ``stop()`` ensures that there is no residual movement
        move_group.stop()
        pub.publish(msg)
        rate1.sleep()
if __name__ == '__main__':
    try:
        # Python3 code to iterate over a list
        list = [-100,150,0,-150,100,1]
        #list = [0,0]
        # Getting length of list
        length = len(list)
        # Iterating using while loop
```

```
        for i in range(0,length,3):
          x =list[i]
           y=list[i+1]
          z=list[i+2]
           print(x,y,z)
          ik(x,y,z)
          rate.sleep()
    except rospy.ROSInterruptException:
       pass
```

✓ scara_trajectory_new_topic.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
import math
from std_msgs.msg import String
from std_msgs.msg import Int32
from moveit_commander.conversions import pose_to_list
l1=150.0
l2=100.0
c=l1+l2
d=l1-l2
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("arm")
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory,queue_size=20)
rate = rospy.Rate(0.1) # 10hz
```

```
#inverse kinematics
def ik(x,y,z):
	pub = rospy.Publisher('gripper_state', Int32, queue_size=10)
	#rospy.init_node('aaa')
	rate1 = rospy.Rate(10) # 10hz
	rospy.loginfo("Publisher has been started.")
	msg = Int32()
	msg.data = z

	 A=2.0*x*l1
	 B=2.0*y*l1
	 C=x**2+y**2+l1**2-l2**2
	 if ((C+A!=0)):
		#elbow-down configuration
		q11=2.0*math.degrees(math.atan((B+math.sqrt(A**2+B**2-
C**2))/(C+A)))
		#elbow-up configuration
		q12=2.0*math.degrees(math.atan((B-math.sqrt(A**2+B**2-
C**2))/(C+A)))
		#elbow-down configuration
		q31=math.degrees(math.atan2((y-l1*math.sin(math.radians(q11))),(x-
l1*math.cos(math.radians(q11)))))
		#elbow-up configuration
		q32=math.degrees(math.atan2((y-l1*math.sin(math.radians(q12))),(x-
l1*math.cos(math.radians(q12)))))
		#elbow-down configuration
		q21=q31-q11
		#elbow-up configuration
		q22=q32-q12
		print("elbow-down configuration","q11="+str(q11),"q21="+str(q21))
		print("elbow-up   configuration","q12="+str(q12),"q22="+str(q22))
		if((q31-q11==0) or (abs(q31-q11)==180) or (q32-q12==0) or (abs(q32-
q12)==180)):
```

```
                #plt.scatter(x,y,color='red')
                #plt.show()
                print("gidilmeyen nokta C+A!=0")
        elif(C+A==0):
            if (A**2+B**2!=0):
                e1=(B*C-A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e11=(B*C+A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e2=(A*C+B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                e22=(A*C-B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                q11=math.degrees(math.atan2(e1,e2))
                q12=math.degrees(math.atan2(e11,e22))
                q11=round(q11,2)
                q12=round(q12,2)
                print("q11="+str(q11),"q12="+str(q12))

                q31=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q11))),(x-l1*math.cos(math.radians(q11)))))
                q32=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q12))),(x-l1*math.cos(math.radians(q12)))))
                q31=round(q31,2)
                q32=round(q32,2)
                q21=q31-q11
                q22=q32-q12

                print("q31="+str(q31),"q32="+str(q32))
                if((abs(q31)-abs(q11)==0) or (abs(q31)-abs(q11)==180) or
(abs(q32)-abs(q12)==0) or (abs(q32)-abs(q12)==180)):
                    #plt.scatter(x,y,color='red')
                    #plt.show()
                    print("gidilmeyen nokta C+A=0")

        else:
            #plt.scatter(x,y,color='red')
```

```
                #plt.show()
                print("gidilemeyen nokta")
        # We can get the joint values from the group and adjust some of the values:
        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = q12*(math.pi/180)-math.pi/2
        joint_goal[1] = q22*(math.pi/180)
        # The go command can be called with joint values, poses, or without any
        # parameters if you have already set the pose or joint target for the group
        move_group.go(joint_goal, wait=True)
        # Calling ``stop()`` ensures that there is no residual movement
        move_group.stop()
        pub.publish(msg)
        rate1.sleep()
if __name__ == '__main__':
    try:
        # Python3 code to iterate over a list
        list = [-100,150,0,-150,100,1]
        #list = [0,0]
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,3):
            x =list[i]
             y=list[i+1]
            z=list[i+2]
             print(x,y,z)
            ik(x,y,z)
            rate.sleep()
    except rospy.ROSInterruptException:
        pass
```

✓ path.txt

```
0.00
0.00
```

-4.51

10.39 …

- ✓ path1.txt

-0.187 , 1.262 , -0.190 , 1.269 ,

- ✓ path2.txt

0.000

0.000

-0.003

0.003

Scara_moveit_config package

Config

- ✓ cartesian_limits.yaml

cartesian_limits:

  max_trans_vel: 1

  max_trans_acc: 2.25

  max_trans_dec: -5

  max_rot_vel: 1.57

- ✓ chomp_planning.yaml

planning_time_limit: 10.0

max_iterations: 200

max_iterations_after_collision_free: 5

smoothness_cost_weight: 0.1

obstacle_cost_weight: 1.0

learning_rate: 0.01

smoothness_cost_velocity: 0.0

smoothness_cost_acceleration: 1.0

smoothness_cost_jerk: 0.0

ridge_factor: 0.01

use_pseudo_inverse: false

pseudo_inverse_ridge_factor: 1e-4

joint_update_limit: 0.1

collision_clearence: 0.2

collision_threshold: 0.07

use_stochastic_descent: true
enable_failure_recovery: true
max_recovery_attempts: 5

- ✓ fake_controllers.yaml

```
controller_list:
  - name: fake_arm_controller
    type: $(arg fake_execution_type)
    joints:
      - joint1
      - joint2
      - joint3
initial:  # Define initial robot poses.
  - group: arm
    pose: home
```

- ✓ joint_limits.yaml

```
# joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed
# Specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [has_velocity_limits, has_acceleration_limits]
joint_limits:
  joint1:
    has_velocity_limits: true
    max_velocity: 5.8177
    has_acceleration_limits: false
    max_acceleration: 0
  joint2:
    has_velocity_limits: true
    max_velocity: 5.8177
    has_acceleration_limits: false
    max_acceleration: 0
  joint3:
    has_velocity_limits: false
```

```
    max_velocity: 0
    has_acceleration_limits: false
    max_acceleration: 0
      ✓ kinematics.yaml
arm:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
      ✓ ompl_planning.yaml
planner_configs:
  SBL:
    type: geometric::SBL
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
  EST:
    type: geometric::EST
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0 setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability. default: 0.05
  LBKPIECE:
    type: geometric::LBKPIECE
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    border_fraction: 0.9  # Fraction of time focused on boarder default: 0.9
    min_valid_path_fraction: 0.5  # Accept partially valid moves above fraction. default: 0.5
  BKPIECE:
    type: geometric::BKPIECE
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    border_fraction: 0.9  # Fraction of time focused on boarder default: 0.9
    failed_expansion_score_factor: 0.5  # When extending motion fails, scale score by factor. default: 0.5
```

```
    min_valid_path_fraction: 0.5  # Accept partially valid moves above fraction. default: 0.5
  KPIECE:
    type: geometric::KPIECE
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability. default: 0.05
    border_fraction: 0.9  # Fraction of time focused on boarder default: 0.9 (0.0,1.]
    failed_expansion_score_factor: 0.5  # When extending motion fails, scale score by factor. default: 0.5
    min_valid_path_fraction: 0.5  # Accept partially valid moves above fraction. default: 0.5
  RRT:
    type: geometric::RRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
  RRTConnect:
    type: geometric::RRTConnect
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
  RRTstar:
    type: geometric::RRTstar
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
    delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found. default 1
  TRRT:
    type: geometric::TRRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
```

```
    max_states_failed: 10  # when to start increasing temp. default: 10
    temp_change_factor: 2.0  # how much to increase or decrease temp. default: 2.0
    min_temperature: 10e-10  # lower limit of temp change. default: 10e-10
    init_temperature: 10e-6  # initial temperature. default: 10e-6
    frountier_threshold: 0.0  # dist new state to nearest neighbor to disqualify as frontier. default: 0.0 set in setup()
    frountierNodeRatio: 0.1  # 1/10, or 1 nonfrontier for every 10 frontier. default: 0.1
    k_constant: 0.0  # value used to normalize expresssion. default: 0.0 set in setup()
  PRM:
    type: geometric::PRM
    max_nearest_neighbors: 10  # use k nearest neighbors. default: 10
  PRMstar:
    type: geometric::PRMstar
  FMT:
    type: geometric::FMT
    num_samples: 1000  # number of states that the planner should sample. default: 1000
    radius_multiplier: 1.1  # multiplier used for the nearest neighbors search radius. default: 1.1
    nearest_k: 1  # use Knearest strategy. default: 1
    cache_cc: 1  # use collision checking cache. default: 1
    heuristics: 0  # activate cost to go heuristics. default: 0
    extended_fmt: 1  # activate the extended FMT*: adding new samples if planner does not finish successfully. default: 1
  BFMT:
    type: geometric::BFMT
    num_samples: 1000  # number of states that the planner should sample. default: 1000
    radius_multiplier: 1.0  # multiplier used for the nearest neighbors search radius. default: 1.0
    nearest_k: 1  # use the Knearest strategy. default: 1
    balanced: 0  # exploration strategy: balanced true expands one tree every iteration. False will select the tree with lowest maximum cost to go. default: 1
```

optimality: 1 # termination strategy: optimality true finishes when the best possible path is found. Otherwise, the algorithm will finish when the first feasible path is found. default: 1

heuristics: 1 # activates cost to go heuristics. default: 1

cache_cc: 1 # use the collision checking cache. default: 1

extended_fmt: 1 # Activates the extended FMT*: adding new samples if planner does not finish successfully. default: 1

PDST:

type: geometric::PDST

STRIDE:

type: geometric::STRIDE

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

goal_bias: 0.05 # When close to goal select goal, with this probability. default: 0.05

use_projected_distance: 0 # whether nearest neighbors are computed based on distances in a projection of the state rather distances in the state space itself. default: 0

degree: 16 # desired degree of a node in the Geometric Near-neightbor Access Tree (GNAT). default: 16

max_degree: 18 # max degree of a node in the GNAT. default: 12

min_degree: 12 # min degree of a node in the GNAT. default: 12

max_pts_per_leaf: 6 # max points per leaf in the GNAT. default: 6

estimated_dimension: 0.0 # estimated dimension of the free space. default: 0.0

min_valid_path_fraction: 0.2 # Accept partially valid moves above fraction. default: 0.2

BiTRRT:

type: geometric::BiTRRT

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

temp_change_factor: 0.1 # how much to increase or decrease temp. default: 0.1

init_temperature: 100 # initial temperature. default: 100

frountier_threshold: 0.0 # dist new state to nearest neighbor to disqualify as frontier. default: 0.0 set in setup()

frountier_node_ratio: 0.1 # 1/10, or 1 nonfrontier for every 10 frontier. default: 0.1

cost_threshold: 1e300 # the cost threshold. Any motion cost that is not better will not be expanded. default: inf

LBTRRT:

type: geometric::LBTRRT

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

goal_bias: 0.05 # When close to goal select goal, with this probability. default: 0.05

epsilon: 0.4 # optimality approximation factor. default: 0.4

BiEST:

type: geometric::BiEST

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

ProjEST:

type: geometric::ProjEST

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

goal_bias: 0.05 # When close to goal select goal, with this probability. default: 0.05

LazyPRM:

type: geometric::LazyPRM

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

LazyPRMstar:

type: geometric::LazyPRMstar

SPARS:

type: geometric::SPARS

stretch_factor: 3.0 # roadmap spanner stretch factor. multiplicative upper bound on path quality. It does not make sense to make this parameter more than 3. default: 3.0

sparse_delta_fraction: 0.25 # delta fraction for connection distance. This value represents the visibility range of sparse samples. default: 0.25

dense_delta_fraction: 0.001 # delta fraction for interface detection. default: 0.001

max_failures: 1000 # maximum consecutive failure limit. default: 1000

SPARStwo:

type: geometric::SPARStwo

```
    stretch_factor: 3.0  # roadmap spanner stretch factor. multiplicative upper bound on path quality. It does not make sense to make this parameter more than 3. default: 3.0
    sparse_delta_fraction: 0.25  # delta fraction for connection distance. This value represents the visibility range of sparse samples. default: 0.25
    dense_delta_fraction: 0.001  # delta fraction for interface detection. default: 0.001
    max_failures: 5000  # maximum consecutive failure limit. default: 5000
arm:
  planner_configs:
    - SBL
    - EST
    - LBKPIECE
    - BKPIECE
    - KPIECE
    - RRT
    - RRTConnect
    - RRTstar
    - TRRT
    - PRM
    - PRMstar
    - FMT
    - BFMT
    - PDST
    - STRIDE
    - BiTRRT
    - LBTRRT
    - BiEST
    - ProjEST
    - LazyPRM
    - LazyPRMstar
    - SPARS
    - SPARStwo
```

- ✓ ros_controllers.yaml

```
# Simulation settings for using moveit_sim_controllers
moveit_sim_hw_interface:
  joint_model_group: arm
  joint_model_group_pose: home
# Settings for ros_control_boilerplate control loop
generic_hw_control_loop:
  loop_hz: 300
  cycle_time_error_threshold: 0.01
# Settings for ros_control hardware interface
hardware_interface:
  joints:
    - joint1
    - joint2
    - joint3
  sim_control_mode: 1  # 0: position, 1: velocity
# Publish all joint states
# Creates the /joint_states topic necessary in ROS
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50
controller_list:
  - name: arm_controller
    action_ns: follow_joint_trajectory
    default: True
    type: FollowJointTrajectory
    joints:
      - joint1
      - joint2
      - joint3
arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
```

```
    - joint1
    - joint2
    - joint3
  gains:
    joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint2:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    joint3:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
```

✓ sensors_3d.yaml

```
# The name of this file shouldn't be changed, or else the Setup Assistant won't detect
it
sensors:
  - filtered_cloud_topic: filtered_cloud
    max_range: 5.0
    max_update_rate: 1.0
    padding_offset: 0.1
    padding_scale: 1.0
    point_cloud_topic: /head_mount_kinect/depth_registered/points
    point_subsample: 1
    sensor_plugin: occupancy_map_monitor/PointCloudOctomapUpdater
  - far_clipping_plane_distance: 5.0
    filtered_cloud_topic: filtered_cloud
```

image_topic: /head_mount_kinect/depth_registered/image_raw
max_range: 5.0
max_update_rate: 1.0
near_clipping_plane_distance: 0.3
padding_offset: 0.03
padding_scale: 4.0
point_cloud_topic: /head_mount_kinect/depth_registered/points
point_subsample: 1
queue_size: 5
sensor_plugin: occupancy_map_monitor/DepthImageOctomapUpdater
shadow_threshold: 0.2

Launch

✓ chomp_planning_pipeline.launch.xml

```
<launch>
  <!-- CHOMP Plugin for MoveIt! -->
  <arg name="planning_plugin" value="chomp_interface/CHOMPPlanner" />
  <!-- define capabilites that are loaded on start (space seperated) -->
  <arg name="capabilities" default=""/>
  <!-- inhibit capabilites (space seperated) -->
  <arg name="disable_capabilities" default=""/>
  <arg name="start_state_max_bounds_error" value="0.1" />
  <!-- The request adapters (plugins) used when planning.
       ORDER MATTERS -->
  <arg name="planning_adapters"
       value="default_planner_request_adapters/AddTimeParameterization
              default_planner_request_adapters/ResolveConstraintFrames
              default_planner_request_adapters/FixWorkspaceBounds
              default_planner_request_adapters/FixStartStateBounds
              default_planner_request_adapters/FixStartStateCollision
              default_planner_request_adapters/FixStartStatePathConstraints"
              />
  <param name="planning_plugin" value="$(arg planning_plugin)" />
  <param name="request_adapters" value="$(arg planning_adapters)" />
```

```
 <param name="start_state_max_bounds_error" value="$(arg start_state_max_bounds_error)" />
 <param name="capabilities" value="$(arg capabilities)" />
 <param name="disable_capabilities" value="$(arg disable_capabilities)" />
 <rosparam command="load" file="$(find scara_moveit_config)/config/chomp_planning.yaml" />
</launch>
```

- ✓ default_warehouse_db.launch

```
<launch>
 <arg name="reset" default="false"/>
 <!-- If not specified, we'll use a default database location -->
 <arg name="moveit_warehouse_database_path" default="$(find scara_moveit_config)/default_warehouse_mongo_db" />
 <!-- Launch the warehouse with the configured database location -->
 <include file="$(find scara_moveit_config)/launch/warehouse.launch">
  <arg name="moveit_warehouse_database_path" value="$(arg moveit_warehouse_database_path)" />
 </include>
 <!-- If we want to reset the database, run this node -->
 <node if="$(arg reset)" name="$(anon moveit_default_db_reset)" type="moveit_init_demo_warehouse" pkg="moveit_ros_warehouse" respawn="false" output="screen" />
</launch>
```

- ✓ demo.launch

```
<launch>
 <!-- specify the planning pipeline -->
 <arg name="pipeline" default="ompl" />
 <!-- By default, we do not start a database (it can be large) -->
 <arg name="db" default="false" />
 <!-- Allow user to specify database location -->
 <arg name="db_path" default="$(find scara_moveit_config)/default_warehouse_mongo_db" />
 <!-- By default, we are not in debug mode -->
```

```
  <arg name="debug" default="false" />
  <!-- By default, we will load or override the robot_description -->
  <arg name="load_robot_description" default="true"/>
  <!-- Set execution mode for fake execution controllers -->
  <arg name="fake_execution_type" default="interpolate" />
  <!--
  By default, hide joint_state_publisher's GUI
  MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of the simulated robot.
  It also provides a GUI to move the simulated robot around "manually".
  This corresponds to moving around the real robot without the use of MoveIt.
  -->
  <arg name="use_gui" default="false" />
  <arg name="use_rviz" default="true" />
  <!-- If needed, broadcast static tf for robot root -->
  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server -->
  <include file="$(find scara_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>
  <!-- We do not have a robot connected, so publish fake joint states -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" unless="$(arg use_gui)">
    <rosparam param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>
  <node name="joint_state_publisher" pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" if="$(arg use_gui)">
    <rosparam param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>
  <!-- Given the published joint states, publish tf for the robot links -->
```

```
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="true" output="screen" />
  <!-- Run the main MoveIt! executable without trajectory execution (we do not have controllers configured by default) -->
  <include file="$(find scara_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
    <arg name="fake_execution_type" value="$(arg fake_execution_type)" />
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="pipeline" value="$(arg pipeline)"/>
    <arg name="load_robot_description" value="$(arg load_robot_description)"/>
  </include>
  <!-- Run Rviz and load the default config to see the state of the move_group node -->
  <include file="$(find scara_moveit_config)/launch/moveit_rviz.launch" if="$(arg use_rviz)">
    <arg name="rviz_config" value="$(find scara_moveit_config)/launch/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <!-- If database loading was enabled, start mongodb as well -->
  <include file="$(find scara_moveit_config)/launch/default_warehouse_db.launch" if="$(arg db)">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
  </include>
</launch>
```

- ✓ demo_gazebo.launch

```
<launch>
  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
```

```
  <arg name="db_path" default="$(find scara_moveit_config)/default_warehouse_mongo_db" />
  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />
  <!-- By default, we won't load or override the robot_description -->
  <arg name="load_robot_description" default="false"/>
  <!--
  By default, hide joint_state_publisher's GUI
  MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of the simulated robot.
  It also provides a GUI to move the simulated robot around "manually".
  This corresponds to moving around the real robot without the use of MoveIt.
  -->
  <arg name="use_gui" default="false" />
  <!-- Gazebo specific options -->
  <arg name="gazebo_gui" default="true"/>
  <arg name="paused" default="false"/>
  <!-- By default, use the urdf location provided from the package -->
  <arg name="urdf_path" default="$(find scaratwo)/urdf/scara.urdf"/>
  <!-- launch the gazebo simulator and spawn the robot -->
  <include file="$(find scara_moveit_config)/launch/gazebo.launch" >
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gazebo_gui" value="$(arg gazebo_gui)"/>
    <arg name="urdf_path" value="$(arg urdf_path)"/>
  </include>
  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server -->
  <include file="$(find scara_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="false"/>
  </include>
  <!-- If needed, broadcast static tf for robot root -->
  <!-- We do not have a robot connected, so publish fake joint states -->
```

```
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" unless="$(arg use_gui)">
    <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
    <rosparam param="source_list">[/joint_states]</rosparam>
  </node>
  <node name="joint_state_publisher" pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui" if="$(arg use_gui)">
    <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
    <rosparam param="source_list">[/joint_states]</rosparam>
  </node>
  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="true" output="screen" />
  <!-- Run the main MoveIt! executable without trajectory execution (we do not have
controllers configured by default) -->
  <include file="$(find scara_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="false"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="load_robot_description" value="$(arg load_robot_description)"/>
  </include>

  <!-- Run Rviz and load the default config to see the state of the move_group node --
>
  <include file="$(find scara_moveit_config)/launch/moveit_rviz.launch">
    <arg name="rviz_config" value="$(find
scara_moveit_config)/launch/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <!-- If database loading was enabled, start mongodb as well -->
```

```
 <include file="$(find scara_moveit_config)/launch/default_warehouse_db.launch" if="$(arg db)">
   <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
 </include>
</launch>
```

- ✓ fake_moveit_controller_manager.launch.xml

```
<launch>
 <!-- execute the trajectory in 'interpolate' mode or jump to goal position in 'last point' mode -->
 <arg name="fake_execution_type" default="interpolate" />
 <!-- Set the param that trajectory_execution_manager needs to find the controller plugin -->
 <param name="moveit_controller_manager" value="moveit_fake_controller_manager/MoveItFakeControllerManager"/>

 <!-- The rest of the params are specific to this plugin -->
 <rosparam subst_value="true" file="$(find scara_moveit_config)/config/fake_controllers.yaml"/>
</launch>
```

- ✓ gazebo.launch

```
<?xml version="1.0"?>
<launch>
 <arg name="paused" default="false"/>
 <arg name="gazebo_gui" default="true"/>
 <arg name="urdf_path" default="$(find scaratwo)/urdf/scara.urdf"/>
 <!-- startup simulated world -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
   <arg name="world_name" default="worlds/empty.world"/>
   <arg name="paused" value="$(arg paused)"/>
   <arg name="gui" value="$(arg gazebo_gui)"/>
 </include>
 <!-- send robot urdf to param server -->
 <param name="robot_description" textfile="$(arg urdf_path)" />
```

```
  <!-- push robot_description to factory and spawn robot in gazebo at the origin, change x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
  <include file="$(find scara_moveit_config)/launch/ros_controllers.launch"/>
</launch>
```

- ✓ joystick_control.launch

```
<launch>
  <!-- See moveit_ros/visualization/doc/joystick.rst for documentation -->
  <arg name="dev" default="/dev/input/js0" />
  <!-- Launch joy node -->
  <node pkg="joy" type="joy_node" name="joy">
    <param name="dev" value="$(arg dev)" /> <!-- Customize this to match the location your joystick is plugged in on-->
    <param name="deadzone" value="0.2" />
    <param name="autorepeat_rate" value="40" />
    <param name="coalesce_interval" value="0.025" />
  </node>
  <!-- Launch python interface -->
  <node pkg="moveit_ros_visualization" type="moveit_joy.py" output="screen" name="moveit_joy"/>
</launch>
```

- ✓ move_group.launch

```
<launch>
  <!-- GDB Debug Option -->
  <arg name="debug" default="false" />
  <arg unless="$(arg debug)" name="launch_prefix" value="" />
  <arg     if="$(arg debug)" name="launch_prefix"
         value="gdb -x $(find scara_moveit_config)/launch/gdb_settings.gdb --ex run --args" />
```

```
<!-- Verbose Mode Option -->
<arg name="info" default="$(arg debug)" />
<arg unless="$(arg info)" name="command_args" value="" />
<arg     if="$(arg info)" name="command_args" value="--debug" />
<!-- move_group settings -->
<arg name="pipeline" default="ompl" />
<arg name="allow_trajectory_execution" default="true"/>
<arg name="fake_execution" default="false"/>
<arg name="fake_execution_type" default="interpolate" />
<arg name="max_safe_path_cost" default="1"/>
<arg name="jiggle_fraction" default="0.05" />
<arg name="publish_monitored_planning_scene" default="true"/>
<arg name="capabilities" default=""/>
<arg name="disable_capabilities" default=""/>
<!-- load these non-default MoveGroup capabilities (space seperated) -->
<!--
<arg name="capabilities" value="
            a_package/AwsomeMotionPlanningCapability
            another_package/GraspPlanningPipeline
            " />
-->
<!-- inhibit these default MoveGroup capabilities (space seperated) -->
<!--
<arg name="disable_capabilities" value="
            move_group/MoveGroupKinematicsService
            move_group/ClearOctomapService
            " />
-->
<arg name="load_robot_description" default="true" />
<!-- load URDF, SRDF and joint_limits configuration -->
<include file="$(find scara_moveit_config)/launch/planning_context.launch">
  <arg name="load_robot_description" value="$(arg load_robot_description)" />
</include>-->
```

```
  <!-- Planning Functionality -->
  <include ns="move_group" file="$(find scara_moveit_config)/launch/planning_pipeline.launch.xml">
    <arg name="pipeline" value="$(arg pipeline)" />
    <arg name="capabilities" value="$(arg capabilities)"/>
    <arg name="disable_capabilities" value="$(arg disable_capabilities)"/>
  </include>
  <!-- Trajectory Execution Functionality -->
  <include ns="move_group" file="$(find scara_moveit_config)/launch/trajectory_execution.launch.xml" if="$(arg allow_trajectory_execution)">
    <arg name="moveit_manage_controllers" value="true" />
    <arg name="moveit_controller_manager" value="scara" unless="$(arg fake_execution)"/>
    <arg name="moveit_controller_manager" value="fake" if="$(arg fake_execution)"/>
    <arg name="fake_execution_type" value="$(arg fake_execution_type)" />
  </include>
  <!-- Sensors Functionality -->
  <include ns="move_group" file="$(find scara_moveit_config)/launch/sensor_manager.launch.xml" if="$(arg allow_trajectory_execution)">
    <arg name="moveit_sensor_manager" value="scara" />
  </include>
  <!-- Start the actual move_group node/action server -->
  <node name="move_group" launch-prefix="$(arg launch_prefix)" pkg="moveit_ros_move_group" type="move_group" respawn="false" output="screen" args="$(arg command_args)">
    <!-- Set the display variable, in case OpenGL code is used internally -->
    <env name="DISPLAY" value="$(optenv DISPLAY :0)" />
    <param name="allow_trajectory_execution" value="$(arg allow_trajectory_execution)"/>
    <param name="max_safe_path_cost" value="$(arg max_safe_path_cost)"/>
```

```
    <param name="jiggle_fraction" value="$(arg jiggle_fraction)" />

    <!-- Publish the planning scene of the physical robot so that rviz plugin can know actual robot -->
    <param name="planning_scene_monitor/publish_planning_scene" value="$(arg publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_geometry_updates" value="$(arg publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_state_updates" value="$(arg publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_transforms_updates" value="$(arg publish_monitored_planning_scene)" />
  </node>
</launch>
```

- ✓ moveit_rviz.launch

```
<launch>
  <arg name="debug" default="false" />
  <arg unless="$(arg debug)" name="launch_prefix" value="" />
  <arg     if="$(arg debug)" name="launch_prefix" value="gdb --ex run --args" />
  <arg name="rviz_config" default="" />
  <arg     if="$(eval rviz_config==")" name="command_args" value="" />
  <arg unless="$(eval rviz_config==")" name="command_args" value="-d $(arg rviz_config)" />
  <node name="$(anon rviz)" launch-prefix="$(arg launch_prefix)" pkg="rviz" type="rviz" respawn="false"
        args="$(arg command_args)" output="screen">
    <rosparam command="load" file="$(find scara_moveit_config)/config/kinematics.yaml"/>
  </node>
</launch>
```

- ✓ ompl_planning_pipeline.launch.xml

```
<launch>
  <!-- OMPL Plugin for MoveIt! -->
  <arg name="planning_plugin" value="ompl_interface/OMPLPlanner" />
  <!-- define capabilites that are loaded on start (space seperated) -->
  <arg name="capabilities" default=""/>

  <!-- inhibit capabilites (space seperated) -->
  <arg name="disable_capabilities" default=""/>
  <!-- The request adapters (plugins) used when planning with OMPL.
       ORDER MATTERS -->
  <arg name="planning_adapters"
value="default_planner_request_adapters/AddTimeParameterization
                                default_planner_request_adapters/FixWorkspaceBounds
                                default_planner_request_adapters/FixStartStateBounds
                                default_planner_request_adapters/FixStartStateCollision

default_planner_request_adapters/FixStartStatePathConstraints" />
  <arg name="start_state_max_bounds_error" value="0.1" />
  <param name="planning_plugin" value="$(arg planning_plugin)" />
  <param name="request_adapters" value="$(arg planning_adapters)" />
  <param name="start_state_max_bounds_error" value="$(arg
start_state_max_bounds_error)" />
  <param name="capabilities" value="$(arg capabilities)" />
  <param name="disable_capabilities" value="$(arg disable_capabilities)" />
  <rosparam command="load" file="$(find
scara_moveit_config)/config/ompl_planning.yaml"/>
</launch>
```

- ✓ pilz_industrial_motion_planner_planning_pipeline.launch.xml

```
<launch>
  <!-- Pilz Command Planner Plugin for MoveIt -->
```

```
  <arg name="planning_plugin" value="pilz_industrial_motion_planner::CommandPlanner" />
  <!-- The request adapters (plugins) used when planning.
       ORDER MATTERS -->
  <arg name="planning_adapters" value="" />
  <!-- define capabilites that are loaded on start (space seperated) -->
  <arg name="capabilities" default=""/>
  <!-- inhibit capabilites (space seperated) -->
  <arg name="disable_capabilities" default=""/>
  <arg name="start_state_max_bounds_error" value="0.1" />
  <param name="planning_plugin" value="$(arg planning_plugin)" />
  <param name="request_adapters" value="$(arg planning_adapters)" />
  <param name="start_state_max_bounds_error" value="$(arg start_state_max_bounds_error)" />
  <!-- MoveGroup capabilities to load, append sequence capability -->
  <param name="capabilities" value="$(arg capabilities)
                                    pilz_industrial_motion_planner/MoveGroupSequenceAction
                                    pilz_industrial_motion_planner/MoveGroupSequenceService
                                   " />
  <param name="disable_capabilities" value="$(arg disable_capabilities)" />
</launch>
```

- ✓ planning_context.launch

```
<launch>
  <!-- By default we do not overwrite the URDF. Change the following to true to change the default behavior -->
  <arg name="load_robot_description" default="false"/>
  <!-- The name of the parameter under which the URDF is loaded -->
  <arg name="robot_description" default="robot_description"/>
  <!-- Load universal robot description format (URDF) -->
  <param if="$(arg load_robot_description)" name="$(arg robot_description)" textfile="$(find scaratwo)/urdf/scara.urdf"/>
  <!-- The semantic description that corresponds to the URDF -->
```

```
  <param name="$(arg robot_description)_semantic" textfile="$(find scara_moveit_config)/config/scara.srdf" />
  <!-- Load updated joint limits (override information from URDF) -->
  <group ns="$(arg robot_description)_planning">
    <rosparam command="load" file="$(find scara_moveit_config)/config/joint_limits.yaml"/>
    <rosparam command="load" file="$(find scara_moveit_config)/config/cartesian_limits.yaml"/>
  </group>

  <!-- Load default settings for kinematics; these settings are overridden by settings in a node's namespace -->
  <group ns="$(arg robot_description)_kinematics">
    <rosparam command="load" file="$(find scara_moveit_config)/config/kinematics.yaml"/>
  </group>
</launch>
```

- ✓ planning_pipeline.launch.xml

```
<launch>
  <!-- This file makes it easy to include different planning pipelines;
       It is assumed that all planning pipelines are named XXX_planning_pipeline.launch  -->
  <arg name="pipeline" default="ompl" />
  <!-- define capabilites that are loaded on start (space seperated) -->
  <arg name="capabilities" default=""/>
  <!-- inhibit capabilites (space seperated) -->
  <arg name="disable_capabilities" default=""/>
  <include file="$(find scara_moveit_config)/launch/$(arg pipeline)_planning_pipeline.launch.xml">
    <arg name="capabilities" value="$(arg capabilities)"/>
    <arg name="disable_capabilities" value="$(arg disable_capabilities)"/>
  </include>
</launch>
```

- ✓ ros_controllers.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find scara_moveit_config)/config/ros_controllers.yaml" command="load"/>
  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" args="arm_controller"/>
</launch>
```

- ✓ run_benchmark_ompl.launch

```
<launch>
  <!-- This argument must specify the list of .cfg files to process for benchmarking -->
  <arg name="cfg" />
  <!-- Load URDF -->
  <include file="$(find scara_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>
  <!-- Start the database -->
  <include file="$(find scara_moveit_config)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path" value="moveit_ompl_benchmark_warehouse"/>
  </include>
  <!-- Start Benchmark Executable -->
  <node name="$(anon moveit_benchmark)" pkg="moveit_ros_benchmarks" type="moveit_run_benchmark" args="$(arg cfg) --benchmark-planners" respawn="false" output="screen">
    <rosparam command="load" file="$(find scara_moveit_config)/config/ompl_planning.yaml"/>
  </node>
</launch>
```

- ✓ scara_moveit_controller_manager.launch.xml

```
<launch>
  <!-- Define the controller manager plugin to use for trajectory execution -->
  <param name="moveit_controller_manager" value="moveit_simple_controller_manager/MoveItSimpleControllerManager" />

  <!-- loads controller list to the param server -->
  <rosparam file="$(find scara_moveit_config)/config/ros_controllers.yaml"/>
</launch>
```

- ✓ sensor_manager.launch.xml

```
<launch>

  <!-- This file makes it easy to include the settings for sensor managers -->
  <!-- Params for 3D sensors config -->
  <rosparam command="load" file="$(find scara_moveit_config)/config/sensors_3d.yaml" />
  <!-- Params for the octomap monitor -->
  <!--  <param name="octomap_frame" type="string" value="some frame in which the robot moves" /> -->
  <param name="octomap_resolution" type="double" value="0.025" />
  <param name="max_range" type="double" value="5.0" />
  <!-- Load the robot specific sensor manager; this sets the moveit_sensor_manager ROS parameter -->
  <arg name="moveit_sensor_manager" default="scara" />
  <include file="$(find scara_moveit_config)/launch/$(arg moveit_sensor_manager)_moveit_sensor_manager.launch.xml" />
</launch>
```

- ✓ trajectory_execution.launch.xml

```
<launch>
  <!-- This file makes it easy to include the settings for trajectory execution  -->
  <arg name="fake_execution_type" default="interpolate" />
  <!-- Flag indicating whether MoveIt! is allowed to load/unload  or switch controllers -->
```

```
  <arg name="moveit_manage_controllers" default="true"/>
  <param name="moveit_manage_controllers" value="$(arg moveit_manage_controllers)"/>

  <!-- When determining the expected duration of a trajectory, this multiplicative factor is applied to get the allowed duration of execution -->
  <param name="trajectory_execution/allowed_execution_duration_scaling" value="1.2"/> <!-- default 1.2 -->
  <!-- Allow more than the expected execution time before triggering a trajectory cancel (applied after scaling) -->
  <param name="trajectory_execution/allowed_goal_duration_margin" value="0.5"/> <!-- default 0.5 -->
  <!-- Allowed joint-value tolerance for validation that trajectory's first point matches current robot state -->
  <param name="trajectory_execution/allowed_start_tolerance" value="0.01"/> <!-- default 0.01 -->
  <!-- Load the robot specific controller manager; this sets the moveit_controller_manager ROS parameter -->
  <arg name="moveit_controller_manager" default="scara" />
  <!-- We use pass_all_args=true here to pass execution_type, which is required by fake controllers, but not by real-robot controllers.
       As real-robot controller_manager.launch files shouldn't be required to define this argument, we use the trick of passing all args. -->
  <include file="$(find scara_moveit_config)/launch/$(arg moveit_controller_manager)_moveit_controller_manager.launch.xml"
pass_all_args="true" />
</launch>
```

- ✓ warehouse.launch

```
<launch>
  <!-- The path to the database must be specified -->
  <arg name="moveit_warehouse_database_path" />
  <!-- Load warehouse parameters -->
```

```
 <include file="$(find scara_moveit_config)/launch/warehouse_settings.launch.xml"
/>
 <!-- Run the DB server -->
 <node name="$(anon mongo_wrapper_ros)" cwd="ROS_HOME"
type="mongo_wrapper_ros.py" pkg="warehouse_ros_mongo">
   <param name="overwrite" value="false"/>
   <param name="database_path" value="$(arg moveit_warehouse_database_path)"
/>
 </node>
</launch>
```

- ✓ warehouse_settings.launch.xml

```
<launch>
 <!-- Set the parameters for the warehouse and run the mongodb server. -->

 <!-- The default DB port for moveit (not default MongoDB port to avoid potential
conflicts) -->
 <arg name="moveit_warehouse_port" default="33829" />

 <!-- The default DB host for moveit -->
 <arg name="moveit_warehouse_host" default="localhost" />
 <!-- Set parameters for the warehouse -->
 <param name="warehouse_port" value="$(arg moveit_warehouse_port)"/>
 <param name="warehouse_host" value="$(arg moveit_warehouse_host)"/>
 <param name="warehouse_exec" value="mongod" />
 <param name="warehouse_plugin"
value="warehouse_ros_mongo::MongoDatabaseConnection" />
</launch>
```

Scripts

- ✓ color_thresholding.py

```
#!/usr/bin/env python
import cv2
import numpy as np
def nothing(pos):
```

```
    pass
cap=cv2.VideoCapture(2)
cv2.namedWindow('Thresholds')
cv2.createTrackbar('LH','Thresholds',0,255, nothing)
cv2.createTrackbar('LS','Thresholds',0,255, nothing)
cv2.createTrackbar('LV','Thresholds',0,255, nothing)
cv2.createTrackbar('UH','Thresholds',255,255, nothing)
cv2.createTrackbar('US','Thresholds',255,255, nothing)
cv2.createTrackbar('UV','Thresholds',255,255, nothing)
while(1):
    _, img = cap.read()
    #converting frame(img i.e BGR) to HSV (hue-saturation-value)
    hsv=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
    lh=cv2.getTrackbarPos('LH','Thresholds')
    ls=cv2.getTrackbarPos('LS','Thresholds')
    lv=cv2.getTrackbarPos('LV','Thresholds')
    uh=cv2.getTrackbarPos('UH','Thresholds')
    us=cv2.getTrackbarPos('US','Thresholds')
    uv=cv2.getTrackbarPos('UV','Thresholds')
    #defining the Range of color
    color_lower=np.array([lh,ls,lv],np.uint8)
    color_upper=np.array([uh,us,uv],np.uint8)
    #finding the range of color in the image
    color=cv2.inRange(hsv,color_lower,color_upper)
    #Morphological transformation, Dilation
    kernal = np.ones((5 ,5), "uint8")
    color=cv2.dilate(color,kernal)

    cv2.imshow("Color",color)
    cv2.imshow("Original Image",img)
    if cv2.waitKey(1)== ord('q'):
        break
cap.release()
```

```
cv2.destroyAllWindows()
```

- ✓ get_pose_openCV.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Pose
import sys
import cv2
import numpy as np
cap=cv2.VideoCapture(2)
pub = rospy.Publisher('direction', Pose, queue_size=10)
rospy.init_node('vision', anonymous=True)
rate = rospy.Rate(10) # 10hz
rospy.loginfo("Direction is published ..... # see /direction topic")

positionX = 0.25
positionY = 0.0
positionZ = 0.2304
orientationX = -1.0
orientationY = -0.0
orientationZ = -0.0
orientationW = 3.26794896538e-07
while(1):
        _, img = cap.read()
        #converting frame(img i.e BGR) to HSV (hue-saturation-value)
        hsv=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
        blue_lower=np.array([91,142,151],np.uint8)
        blue_upper=np.array([255,255,255],np.uint8)
        blue=cv2.inRange(hsv,blue_lower,blue_upper)
        #Morphological transformation, Dilation
        kernal = np.ones((5 ,5), "uint8")
        blue=cv2.dilate(blue,kernal)
```

```
(_,contours,hierarchy)=cv2.findContours(blue,cv2.RETR_TREE,cv2.CHAIN_
APPROX_SIMPLE)
if len(contours)>0:
        contour= max(contours,key=cv2.contourArea)
        area = cv2.contourArea(contour)
        if area>20:
            x,y,w,h = cv2.boundingRect(contour)
          x=x+int(w/2)
          y=y+int(h/2)
            ss = 'x:' + str(x) + ', y:' + str(y)
            #cv2.putText(img,ss,(x-20,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),1,cv2.LINE_AA)
            OldRangeX = (640 - 0)
            NewRangeX = (0.2 - (-0.44))
            positionX = (((x - 0) * NewRangeX) / OldRangeX) + (-0.18)

            OldRangeY = (480 - 0)
            NewRangeY = (0.2 - (-0.28))
            positionY = -1*(((y - 0) * NewRangeY) / OldRangeY) + (+0.26)
            positionZ = 0.0087
            orientationX = 0.0
            orientationY = 0.0
            orientationZ = 0.0
            orientationW = 1.0
          cv2.putText(img,str(positionX),(x-20,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),1,cv2.LINE_AA)
           cv2.putText(img,str(positionY),(x-20,y-
20),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,0,0),1,cv2.LINE_AA)
    cv2.imshow("Mask",blue)
    cv2.imshow("Color Tracking",img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
```

```
        if not rospy.is_shutdown():
                p = Pose()
                p.position.x = positionX
                p.position.y = positionY
                p.position.z = positionZ
                p.orientation.x = orientationX
                p.orientation.y = orientationY
                p.orientation.z = orientationZ
                p.orientation.w = orientationW

                pub.publish(p)
                rate.sleep()
cap.release()
cv2.destroyAllWindows()
```

- ✓ get_pose_openCV_move.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Pose
import sys
import cv2
import numpy as np
import math
import sys
import geometry_msgs.msg
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
from std_msgs.msg import Int32
cap=cv2.VideoCapture(2)
rospy.init_node('vision', anonymous=True)
rate = rospy.Rate(10) # 10hz
rospy.loginfo("Direction is published ..... # see /direction topic")
```

```
positionX = 0.25
positionY = 0.0
positionZ = 0.2304
orientationX = -1.0
orientationY = -0.0
orientationZ = -0.0
orientationW = 3.26794896538e-07
l1=150.0
l2=100.0
c=l1+l2
d=l1-l2
def ik(x,y,z):
        pub = rospy.Publisher('gripper_state', Int32, queue_size=50)
        rate2 = rospy.Rate(10) # 10hz
        rospy.loginfo("Publisher has been started.")
        msg = Int32()
        msg.data = z
        pub1 = rospy.Publisher('joint_states', JointState, queue_size=50)
        hello_str = JointState()
        hello_str.header = Header()



         A=2.0*x*l1
         B=2.0*y*l1
         C=x**2+y**2+l1**2-l2**2
         if ((C+A!=0)):
                #elbow-down configuration
                q11=2.0*math.degrees(math.atan((B+math.sqrt(A**2+B**2-
C**2))/(C+A)))
```

```
            #elbow-up configuration
            q12=2.0*math.degrees(math.atan((B-math.sqrt(A**2+B**2-
C**2))/(C+A)))

            #elbow-down configuration
            q31=math.degrees(math.atan2((y-l1*math.sin(math.radians(q11))),(x-
l1*math.cos(math.radians(q11)))))
            #elbow-up configuration
            q32=math.degrees(math.atan2((y-l1*math.sin(math.radians(q12))),(x-
l1*math.cos(math.radians(q12)))))
            #elbow-down configuration
            q21=q31-q11
            #elbow-up configuration
            q22=q32-q12
            print("elbow-down configuration","q11="+str(q11),"q21="+str(q21))
            print("elbow-up   configuration","q12="+str(q12),"q22="+str(q22))
            if((q31-q11==0) or (abs(q31-q11)==180) or (q32-q12==0) or (abs(q32-
q12)==180)):
                  #plt.scatter(x,y,color='red')
                  #plt.show()
                  print("gidilmeyen nokta C+A!=0")
      elif(C+A==0):
            if (A**2+B**2!=0):
                  e1=(B*C-A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e11=(B*C+A*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e2=(A*C+B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  e22=(A*C-B*math.sqrt(A**2+B**2-C**2))/(A**2+B**2)
                  q11=math.degrees(math.atan2(e1,e2))
                  q12=math.degrees(math.atan2(e11,e22))
                  q11=round(q11,2)
                  q12=round(q12,2)
                  print("q11="+str(q11),"q12="+str(q12))
```

```
                    q31=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q11))),(x-l1*math.cos(math.radians(q11)))))
                    q32=math.degrees(math.atan2((y-
l1*math.sin(math.radians(q12))),(x-l1*math.cos(math.radians(q12)))))
                    q31=round(q31,2)
                    q32=round(q32,2)
                    q21=q31-q11
                    q22=q32-q12

                    print("q31="+str(q31),"q32="+str(q32))
                    if((abs(q31)-abs(q11)==0)  or  (abs(q31)-abs(q11)==180)  or
(abs(q32)-abs(q12)==0) or (abs(q32)-abs(q12)==180)):
                        #plt.scatter(x,y,color='red')
                        #plt.show()
                        print("gidilmeyen nokta C+A=0")
        else:
            #plt.scatter(x,y,color='red')
            #plt.show()
            print("gidilemeyen nokta")

        # We can get the joint values from the group and adjust some of the values:
        joint_goal0 = q12*(math.pi/180)
        joint_goal1 = q22*(math.pi/180)
        hello_str.name = ['joint1', 'joint2']
        hello_str.position = [joint_goal0, joint_goal1,0.00]
        hello_str.velocity = []
        hello_str.effort = []
        hello_str.header.stamp = rospy.Time.now()
        pub1.publish(hello_str)
        pub.publish(msg)
        rate2.sleep()
while(1):
        _, img = cap.read()
```

```
        #converting frame(img i.e BGR) to HSV (hue-saturation-value)
        hsv=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
        blue_lower=np.array([87,145,171],np.uint8)
        blue_upper=np.array([255,255,255],np.uint8)
        blue=cv2.inRange(hsv,blue_lower,blue_upper)
        #Morphological transformation, Dilation
        kernal = np.ones((5 ,5), "uint8")
        blue=cv2.dilate(blue,kernal)
        (_,contours,hierarchy)=cv2.findContours(blue,cv2.RETR_TREE,cv2.CHAIN_
APPROX_SIMPLE)
        if len(contours)>0:
                contour= max(contours,key=cv2.contourArea)
                area = cv2.contourArea(contour)
                if area>20:
                        x,y,w,h = cv2.boundingRect(contour)
                        ss = 'x:' + str(x) + ', y:' + str(y)
                        #cv2.putText(img,ss,(x-20,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),1,cv2.LINE_AA)
                        OldRangeX = (640 - 0)
                        NewRangeX = (0.2 - (-0.44))
                        positionX = (((x - 0) * NewRangeX) / OldRangeX) + (-0.18)

                        OldRangeY = (480 - 0)
                        NewRangeY = (0.2 - (-0.28))
                        positionY = -1*(((y - 0) * NewRangeY) / OldRangeY) + (+0.26)
                        positionZ = 0.0087
                        orientationX = 0.0
                        orientationY = 0.0
                        orientationZ = 0.0
                        orientationW = 1.0
                    cv2.putText(img,str(positionX),(x-20,y-
5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),1,cv2.LINE_AA)
```

```
                    cv2.putText(img,str(positionY),(x-20,y-
20),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,0,0),1,cv2.LINE_AA)
        cv2.imshow("Mask",blue)
        cv2.imshow("Color Tracking",img)
        k = cv2.waitKey(25) & 0xff # Press 'ESC' for exiting video
        if k == 27:
              break
        if not rospy.is_shutdown():
          list = [positionX*1000,positionY*1000,0,250,0,1]
          # Getting length of list
          length = len(list)
          # Iterating using while loop
         #ik(positionX*1000,positionY*1000,1)
         for i in range(0,length,3):
            x =list[i]
             y=list[i+1]
            z=list[i+2]
             print(x,y,z)
            ik(x,y,z)
            rate.sleep()
cap.release()
cv2.destroyAllWindows()
```

- ✓ gecikmesiz.ino

```
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
#else
  #include <WProgram.h>
#endif
#include <ros.h>
#include <std_msgs/UInt16.h>
#include <std_msgs/Int32.h>
#include <sensor_msgs/JointState.h>
```

```
//*****************************************Lx-16a servo tanimlamalari*****************************************//
#define GET_LOW_BYTE(A) (uint8_t)((A))
//Macro function  get lower 8 bits of A
#define GET_HIGH_BYTE(A) (uint8_t)((A) >> 8)
//Macro function  get higher 8 bits of A
#define BYTE_TO_HW(A, B) ((((uint16_t)(A)) << 8) | (uint8_t)(B))
//put A as higher 8 bits   B as lower 8 bits   which amalgamated into 16 bits integer
#define LOBOT_SERVO_FRAME_HEADER         0x55
#define LOBOT_SERVO_MOVE_TIME_WRITE      1
#define LOBOT_SERVO_MOVE_TIME_READ       2
#define LOBOT_SERVO_MOVE_TIME_WAIT_WRITE 7
#define LOBOT_SERVO_MOVE_TIME_WAIT_READ  8
#define LOBOT_SERVO_MOVE_START           11
#define LOBOT_SERVO_MOVE_STOP            12
#define LOBOT_SERVO_ID_WRITE             13
#define LOBOT_SERVO_ID_READ              14
#define LOBOT_SERVO_ANGLE_OFFSET_ADJUST  17
#define LOBOT_SERVO_ANGLE_OFFSET_WRITE   18
#define LOBOT_SERVO_ANGLE_OFFSET_READ    19
#define LOBOT_SERVO_ANGLE_LIMIT_WRITE    20
#define LOBOT_SERVO_ANGLE_LIMIT_READ     21
#define LOBOT_SERVO_VIN_LIMIT_WRITE      22
#define LOBOT_SERVO_VIN_LIMIT_READ       23
#define LOBOT_SERVO_TEMP_MAX_LIMIT_WRITE 24
#define LOBOT_SERVO_TEMP_MAX_LIMIT_READ  25
#define LOBOT_SERVO_TEMP_READ            26
#define LOBOT_SERVO_VIN_READ             27
#define LOBOT_SERVO_POS_READ             28
#define LOBOT_SERVO_OR_MOTOR_MODE_WRITE  29
#define LOBOT_SERVO_OR_MOTOR_MODE_READ   30
#define LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE 31
#define LOBOT_SERVO_LOAD_OR_UNLOAD_READ  32
```

```
#define LOBOT_SERVO_LED_CTRL_WRITE       33
#define LOBOT_SERVO_LED_CTRL_READ        34
#define LOBOT_SERVO_LED_ERROR_WRITE      35
#define LOBOT_SERVO_LED_ERROR_READ       36
//#define LOBOT_DEBUG 1  /*Debug  : print debug value*/
byte LobotCheckSum(byte buf[])
{
  byte i;
  uint16_t temp = 0;
  for (i = 2; i < buf[3] + 2; i++) {
    temp += buf[i];
  }
  temp = ~temp;
  i = (byte)temp;
  return i;
}

void LobotSerialServoMove(HardwareSerial &SerialX, uint8_t id, int16_t position,
uint16_t time)
{
  byte buf[10];
  if(position < 0)
    position = 0;
  if(position > 1000)
    position = 1000;
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 7;
  buf[4] = LOBOT_SERVO_MOVE_TIME_WRITE;
  buf[5] = GET_LOW_BYTE(position);
  buf[6] = GET_HIGH_BYTE(position);
  buf[7] = GET_LOW_BYTE(time);
  buf[8] = GET_HIGH_BYTE(time);
```

```
  buf[9] = LobotCheckSum(buf);
  SerialX.write(buf, 10);
}
void LobotSerialServoStopMove(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[6];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_MOVE_STOP;
  buf[5] = LobotCheckSum(buf);
  SerialX.write(buf, 6);
}
void LobotSerialServoSetID(HardwareSerial &SerialX, uint8_t oldID, uint8_t
newID)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = oldID;
  buf[3] = 4;
  buf[4] = LOBOT_SERVO_ID_WRITE;
  buf[5] = newID;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO ID WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
```

```
#endif
}
void LobotSerialServoSetMode(HardwareSerial &SerialX, uint8_t id, uint8_t Mode,
int16_t Speed)
{
  byte buf[10];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 7;
  buf[4] = LOBOT_SERVO_OR_MOTOR_MODE_WRITE;
  buf[5] = Mode;
  buf[6] = 0;
  buf[7] = GET_LOW_BYTE((uint16_t)Speed);
  buf[8] = GET_HIGH_BYTE((uint16_t)Speed);
  buf[9] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO Set Mode");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
  SerialX.write(buf, 10);
}
void LobotSerialServoLoad(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 4;
```

```
  buf[4] = LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE;
  buf[5] = 1;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO LOAD WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
}
void LobotSerialServoUnload(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 4;
  buf[4] = LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE;
  buf[5] = 0;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO LOAD WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
```

```
  Serial.println(" ");
#endif
}
int LobotSerialServoReceiveHandle(HardwareSerial &SerialX, byte *ret)
{
  bool frameStarted = false;
  bool receiveFinished = false;
  byte frameCount = 0;
  byte dataCount = 0;
  byte dataLength = 2;
  byte rxBuf;
  byte recvBuf[32];
  byte i;
  while (SerialX.available()) {
    rxBuf = SerialX.read();
    delayMicroseconds(100);
    if (!frameStarted) {
      if (rxBuf == LOBOT_SERVO_FRAME_HEADER) {
        frameCount++;
        if (frameCount == 2) {
          frameCount = 0;
          frameStarted = true;
          dataCount = 1;
        }
      }
      else {
        frameStarted = false;
        dataCount = 0;
        frameCount = 0;
      }
    }
    if (frameStarted) {
      recvBuf[dataCount] = (uint8_t)rxBuf;
```

```
      if (dataCount == 3) {
        dataLength = recvBuf[dataCount];
        if (dataLength < 3 || dataCount > 7) {
          dataLength = 2;
          frameStarted = false;
        }
      }
      dataCount++;
      if (dataCount == dataLength + 3) {

#ifdef LOBOT_DEBUG
        Serial.print("RECEIVE DATA:");
        for (i = 0; i < dataCount; i++) {
          Serial.print(recvBuf[i], HEX);
          Serial.print(":");
        }
        Serial.println(" ");
#endif
        if (LobotCheckSum(recvBuf) == recvBuf[dataCount - 1]) {
#ifdef LOBOT_DEBUG
          Serial.println("Check SUM OK!!");
          Serial.println("");
#endif
          frameStarted = false;
          memcpy(ret, recvBuf + 4, dataLength);
          return 1;
        }
        return -1;
      }
    }
  }
}
int LobotSerialServoReadPosition(HardwareSerial &SerialX, uint8_t id)
```

```
{
  int count = 10000;
  int ret;
  byte buf[6];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_POS_READ;
  buf[5] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO Pos READ");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
  while (SerialX.available())
    SerialX.read();
  SerialX.write(buf, 6);
  while (!SerialX.available()) {
    count -= 1;
    if (count < 0)
      return -2048;
  }
  if (LobotSerialServoReceiveHandle(SerialX, buf) > 0)
    ret = (int16_t)BYTE_TO_HW(buf[2], buf[1]);
  else
    ret = -2048;
#ifdef LOBOT_DEBUG
  Serial.println(ret);
```

```
#endif
  return ret;
}
int LobotSerialServoReadVin(HardwareSerial &SerialX, uint8_t id)
{
  int count = 10000;
  int ret;
  byte buf[6];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_VIN_READ;
  buf[5] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO VIN READ");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif

  while (SerialX.available())
    SerialX.read();
  SerialX.write(buf, 6);
  while (!SerialX.available()) {
    count -= 1;
    if (count < 0)
      return -2048;
  }
  if (LobotSerialServoReceiveHandle(SerialX, buf) > 0)
```

```
    ret = (int16_t)BYTE_TO_HW(buf[2], buf[1]);
  else
    ret = -2049;
#ifdef LOBOT_DEBUG
  Serial.println(ret);
#endif
  return ret;
}
int base_angle_pos;
int shoulder_angle_pos;
double base_angle_offset=120;
double shoulder_angle_offset=120;
//**********************************************Lx-16a servo tanimlamalari bitti**********************************************//
//*****************ros moveitten gelen acilar cin yazilan kisim*******************//
ros::NodeHandle  nh;
#define ID1   1  //servoların bagli oldugu pinler atanır
#define ID2   2
int hava_role_pin=7;
int miknatis_role_pin=8;
//----------------------Setup Function Variable-------------------------------------//
int baud_rate=9600;
//----------------------timers and delays----------------------------------------//
int role_delay_sure=0;
double  base_angle=base_angle_offset;          //servoların baslangic acilari home pozisyonuna gore belirlenmistir.
double shoulder_angle=shoulder_angle_offset;
double prev_base = 0;
double prev_shoulder = 0;
int gripperState;    //burada pistonun durumu en basta 0 olarak belirlenmistir.bu state moveit scriptten gelecek.
int positionChanged = 0;
```

```
void gripper_cb(const std_msgs::Int32& msg)
{
  gripperState=msg.data;
    if (gripperState==0)      //gripper stateler moveit script dosyasindan gonderilecek.
     {
    gripper_hold();
     }
  if (gripperState==1)
    {
      gripper_release();
    }
}
void servo_cb(const sensor_msgs::JointState& cmd_msg){
  base_angle=radiansToDegrees(cmd_msg.position[0]);     //ros tarafindan gonderilen joint radian degerleri dereceye cevirilir.
  shoulder_angle=radiansToDegrees(cmd_msg.position[1]);
  int rq11=(base_angle_offset+(base_angle));
  int rq22=(shoulder_angle_offset+(shoulder_angle));
  int q1=map(rq11,0,240,0,1000);
  int q2=map(rq22,0,240,0,1000);
  // put your main code here, to run repeatedly:
  LobotSerialServoMove(Serial1, ID1, q1, 100);      //dereceye donen aci degerleri motorlara basilmistir.
  LobotSerialServoMove(Serial1, ID2, q2, 100);
  base_angle_pos=LobotSerialServoReadPosition(Serial1, ID1);
  shoulder_angle_pos=LobotSerialServoReadPosition(Serial1, ID2);
  //Serial.println(base_angle_pos);
  //Serial.println(shoulder_angle_pos);
  delay(20);
  if (prev_base==base_angle && prev_shoulder==shoulder_angle && positionChanged==0)
  {                          //eger tum eklemler istenilen aci degerine ulasirsa daha sonra gripper sabit alma veya bırakma hareketini yapacaktır.
```

```
   if (gripperState==0)      //gripper stateler moveit script dosyasindan gonderilecek.
   {
     //gripper_hold();
   }
   if (gripperState==1)
   {
     //gripper_release();
   }
   positionChanged = 1;
  }
  else if ((prev_base!=base_angle || prev_shoulder!=shoulder_angle) &&
positionChanged==1)
  {
    positionChanged = 0;
  }
  prev_base = base_angle;
  prev_shoulder = shoulder_angle;
}
ros::Subscriber<sensor_msgs::JointState> sub("joint_states", servo_cb);
ros::Subscriber<std_msgs::Int32> sub1("gripper_state", &gripper_cb);
void gripper_hold()
{
 digitalWrite(hava_role_pin,HIGH);            // Röleyi Kapali konuma getir
 digitalWrite(miknatis_role_pin,HIGH);        // Röleyi Kapali konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,LOW);          // Röleyi açik konuma getir
 delay(role_delay_sure);
 digitalWrite(miknatis_role_pin,LOW);     // Röleyi açik konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,HIGH);         // Röleyi Kapali konuma getir
}
void gripper_release()
{
```

```
 digitalWrite(hava_role_pin,LOW);          // Röleyi açik konuma getir
 delay(2*role_delay_sure);
 digitalWrite(miknatis_role_pin,HIGH);     // Röleyi Kapali konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,HIGH);         // Röleyi Kapali konuma getir
}
void relay_locking()
{
  digitalWrite(hava_role_pin,HIGH);
  digitalWrite(miknatis_role_pin,HIGH);
}
void setup(){
  Serial1.begin(115200);  //servo motorlara angle degerini girmek icin
  nh.getHardware()->setBaud(115200);   //haberlesme hizi belirlendi.
  nh.initNode();
  nh.subscribe(sub);
  nh.subscribe(sub1);
  //----------Relay & mosfet Locking ----------------------//
  // Rölenin bağlı olduğu role_pin numaralı pini çıkış pini olarak ayarla
  pinMode(hava_role_pin,OUTPUT);
  pinMode(miknatis_role_pin,OUTPUT);
  relay_locking();
  delay(1);
  int rq11=(base_angle_offset);    //home position robot gonderilir.
  int rq22=(shoulder_angle_offset);
  float q1=map(rq11,0,240,0,1000);
  float q2=map(rq22,0,240,0,1000);
  LobotSerialServoMove(Serial1, ID1, q1,3000);  //720
  LobotSerialServoMove(Serial1, ID2, q2, 3000); //720
  base_angle_pos=LobotSerialServoReadPosition(Serial1, ID1);
  shoulder_angle_pos=LobotSerialServoReadPosition(Serial1, ID2);
  //Serial.println(base_angle_pos);
  //Serial.println(shoulder_angle_pos);
```

```
  delay(20);
  //gripper.write(0);
}
void loop(){
  nh.spinOnce();
}
double radiansToDegrees(float position_radians)
{
  float position_degree=position_radians * 57.2958;
  return position_degree;

}
```

- ✓ arduino_code_birlestirilmis_deneme.ino

```
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
#else
  #include <WProgram.h>
#endif
#include <ros.h>
#include <std_msgs/UInt16.h>
#include <std_msgs/Int32.h>
#include <sensor_msgs/JointState.h>
//*****************************************Lx-16a servo tanimlamalari*****************************************//
#define GET_LOW_BYTE(A) (uint8_t)((A))
//Macro function  get lower 8 bits of A
#define GET_HIGH_BYTE(A) (uint8_t)((A) >> 8)
//Macro function  get higher 8 bits of A
#define BYTE_TO_HW(A, B) ((((uint16_t)(A)) << 8) | (uint8_t)(B))
//put A as higher 8 bits   B as lower 8 bits   which amalgamated into 16 bits integer
#define LOBOT_SERVO_FRAME_HEADER         0x55
#define LOBOT_SERVO_MOVE_TIME_WRITE      1
#define LOBOT_SERVO_MOVE_TIME_READ       2
```

```
#define LOBOT_SERVO_MOVE_TIME_WAIT_WRITE 7
#define LOBOT_SERVO_MOVE_TIME_WAIT_READ  8
#define LOBOT_SERVO_MOVE_START          11
#define LOBOT_SERVO_MOVE_STOP           12
#define LOBOT_SERVO_ID_WRITE            13
#define LOBOT_SERVO_ID_READ             14
#define LOBOT_SERVO_ANGLE_OFFSET_ADJUST  17
#define LOBOT_SERVO_ANGLE_OFFSET_WRITE   18
#define LOBOT_SERVO_ANGLE_OFFSET_READ    19
#define LOBOT_SERVO_ANGLE_LIMIT_WRITE    20
#define LOBOT_SERVO_ANGLE_LIMIT_READ     21
#define LOBOT_SERVO_VIN_LIMIT_WRITE      22
#define LOBOT_SERVO_VIN_LIMIT_READ       23
#define LOBOT_SERVO_TEMP_MAX_LIMIT_WRITE 24
#define LOBOT_SERVO_TEMP_MAX_LIMIT_READ  25
#define LOBOT_SERVO_TEMP_READ           26
#define LOBOT_SERVO_VIN_READ            27
#define LOBOT_SERVO_POS_READ            28
#define LOBOT_SERVO_OR_MOTOR_MODE_WRITE  29
#define LOBOT_SERVO_OR_MOTOR_MODE_READ   30
#define LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE 31
#define LOBOT_SERVO_LOAD_OR_UNLOAD_READ  32
#define LOBOT_SERVO_LED_CTRL_WRITE       33
#define LOBOT_SERVO_LED_CTRL_READ        34
#define LOBOT_SERVO_LED_ERROR_WRITE      35
#define LOBOT_SERVO_LED_ERROR_READ       36
//#define LOBOT_DEBUG 1  /*Debug  : print debug value*/
byte LobotCheckSum(byte buf[])
{
  byte i;
  uint16_t temp = 0;
  for (i = 2; i < buf[3] + 2; i++) {
    temp += buf[i];
```

```
  }
  temp = ~temp;
  i = (byte)temp;
  return i;
}
void LobotSerialServoMove(HardwareSerial &SerialX, uint8_t id, int16_t position,
uint16_t time)
{
  byte buf[10];
  if(position < 0)
    position = 0;
  if(position > 1000)
    position = 1000;
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 7;
  buf[4] = LOBOT_SERVO_MOVE_TIME_WRITE;
  buf[5] = GET_LOW_BYTE(position);
  buf[6] = GET_HIGH_BYTE(position);
  buf[7] = GET_LOW_BYTE(time);
  buf[8] = GET_HIGH_BYTE(time);
  buf[9] = LobotCheckSum(buf);
  SerialX.write(buf, 10);
}
void LobotSerialServoStopMove(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[6];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_MOVE_STOP;
  buf[5] = LobotCheckSum(buf);
  SerialX.write(buf, 6);
```

```
}
void LobotSerialServoSetID(HardwareSerial &SerialX, uint8_t oldID, uint8_t newID)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = oldID;
  buf[3] = 4;
  buf[4] = LOBOT_SERVO_ID_WRITE;
  buf[5] = newID;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO ID WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
}
void LobotSerialServoSetMode(HardwareSerial &SerialX, uint8_t id, uint8_t Mode, int16_t Speed)
{
  byte buf[10];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 7;
  buf[4] = LOBOT_SERVO_OR_MOTOR_MODE_WRITE;
  buf[5] = Mode;
  buf[6] = 0;
```

```
  buf[7] = GET_LOW_BYTE((uint16_t)Speed);
  buf[8] = GET_HIGH_BYTE((uint16_t)Speed);
  buf[9] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO Set Mode");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
  SerialX.write(buf, 10);
}
void LobotSerialServoLoad(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 4;
  buf[4] = LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE;
  buf[5] = 1;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO LOAD WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
```

```
  Serial.println(" ");
#endif
}
void LobotSerialServoUnload(HardwareSerial &SerialX, uint8_t id)
{
  byte buf[7];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 4;
  buf[4] = LOBOT_SERVO_LOAD_OR_UNLOAD_WRITE;
  buf[5] = 0;
  buf[6] = LobotCheckSum(buf);
  SerialX.write(buf, 7);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO LOAD WRITE");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
}
int LobotSerialServoReceiveHandle(HardwareSerial &SerialX, byte *ret)
{
  bool frameStarted = false;
  bool receiveFinished = false;
  byte frameCount = 0;
  byte dataCount = 0;
  byte dataLength = 2;
  byte rxBuf;
  byte recvBuf[32];
```

```
  byte i;
  while (SerialX.available()) {
    rxBuf = SerialX.read();
    delayMicroseconds(100);
    if (!frameStarted) {
      if (rxBuf == LOBOT_SERVO_FRAME_HEADER) {
        frameCount++;
        if (frameCount == 2) {
          frameCount = 0;
          frameStarted = true;
          dataCount = 1;
        }
      }
      else {
        frameStarted = false;
        dataCount = 0;
        frameCount = 0;
      }
    }
    if (frameStarted) {
      recvBuf[dataCount] = (uint8_t)rxBuf;
      if (dataCount == 3) {
        dataLength = recvBuf[dataCount];
        if (dataLength < 3 || dataCount > 7) {
          dataLength = 2;
          frameStarted = false;
        }
      }
      dataCount++;
      if (dataCount == dataLength + 3) {
#ifdef LOBOT_DEBUG
        Serial.print("RECEIVE DATA:");
        for (i = 0; i < dataCount; i++) {
```

```
          Serial.print(recvBuf[i], HEX);
          Serial.print(":");
        }
        Serial.println(" ");
#endif
        if (LobotCheckSum(recvBuf) == recvBuf[dataCount - 1]) {
#ifdef LOBOT_DEBUG
          Serial.println("Check SUM OK!!");
          Serial.println("");
#endif
          frameStarted = false;
          memcpy(ret, recvBuf + 4, dataLength);
          return 1;
        }
        return -1;
      }
    }
  }
}
int LobotSerialServoReadPosition(HardwareSerial &SerialX, uint8_t id)
{
  int count = 10000;
  int ret;
  byte buf[6];

  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_POS_READ;
  buf[5] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO Pos READ");
  int debug_value_i = 0;
```

```
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
  while (SerialX.available())
    SerialX.read();
  SerialX.write(buf, 6);
  while (!SerialX.available()) {
    count -= 1;
    if (count < 0)
      return -2048;
  }
  if (LobotSerialServoReceiveHandle(SerialX, buf) > 0)
    ret = (int16_t)BYTE_TO_HW(buf[2], buf[1]);
  else
    ret = -2048;
#ifdef LOBOT_DEBUG
  Serial.println(ret);
#endif
  return ret;
}
int LobotSerialServoReadVin(HardwareSerial &SerialX, uint8_t id)
{
  int count = 10000;
  int ret;
  byte buf[6];
  buf[0] = buf[1] = LOBOT_SERVO_FRAME_HEADER;
  buf[2] = id;
  buf[3] = 3;
  buf[4] = LOBOT_SERVO_VIN_READ;
```

```
  buf[5] = LobotCheckSum(buf);
#ifdef LOBOT_DEBUG
  Serial.println("LOBOT SERVO VIN READ");
  int debug_value_i = 0;
  for (debug_value_i = 0; debug_value_i < buf[3] + 3; debug_value_i++)
  {
    Serial.print(buf[debug_value_i], HEX);
    Serial.print(":");
  }
  Serial.println(" ");
#endif
  while (SerialX.available())
    SerialX.read();
  SerialX.write(buf, 6);
  while (!SerialX.available()) {
    count -= 1;
    if (count < 0)
      return -2048;
  }
  if (LobotSerialServoReceiveHandle(SerialX, buf) > 0)
    ret = (int16_t)BYTE_TO_HW(buf[2], buf[1]);
  else
    ret = -2049;
#ifdef LOBOT_DEBUG
  Serial.println(ret);
#endif
  return ret;
}
int base_angle_pos;
int shoulder_angle_pos;
double base_angle_offset=120;
double shoulder_angle_offset=120;
```

```
//*******************************************Lx-16a servo tanimlamalari bitti*******************************************//
//****************ros moveitten gelen acilar cin yazilan kisim******************//
ros::NodeHandle  nh;
#define ID1   1  //servoların bagli oldugu pinler atanır
#define ID2   2
int hava_role_pin=7;
int miknatis_role_pin=8;
//----------------------Setup Function Variable-------------------------------------//
int baud_rate=9600;
//----------------------timers and delays----------------------------------------//
int role_delay_sure=1500;
double  base_angle=base_angle_offset;          //servoların baslangic acilari home pozisyonuna gore belirlenmistir.
double shoulder_angle=shoulder_angle_offset;
double prev_base = 0;
double prev_shoulder = 0;
int gripperState;    //burada pistonun durumu en basta 0 olarak belirlenmistir.bu state moveit scriptten gelecek.
int positionChanged = 0;
void gripper_cb(const std_msgs::Int32& msg)
{
  gripperState=msg.data;
    if (gripperState==0)      //gripper stateler moveit script dosyasindan gonderilecek.
     {
    gripper_hold();
     }
  if (gripperState==1)
    {
      gripper_release();
    }
}
```

```
void servo_cb(const sensor_msgs::JointState& cmd_msg){
  base_angle=radiansToDegrees(cmd_msg.position[0]);     //ros tarafindan gonderilen joint radian degerleri dereceye cevirilir.
  shoulder_angle=radiansToDegrees(cmd_msg.position[1]);
  int rq11=(base_angle_offset+(base_angle));
  int rq22=(shoulder_angle_offset+(shoulder_angle));
  int q1=map(rq11,0,240,0,1000);
  int q2=map(rq22,0,240,0,1000);
  // put your main code here, to run repeatedly:
  LobotSerialServoMove(Serial1, ID1, q1, 50);       //dereceye donen aci degerleri motorlara basilmistir.
  LobotSerialServoMove(Serial1, ID2, q2, 50);
  base_angle_pos=LobotSerialServoReadPosition(Serial1, ID1);
  shoulder_angle_pos=LobotSerialServoReadPosition(Serial1, ID2);
  //Serial.println(base_angle_pos);
  //Serial.println(shoulder_angle_pos);
  delay(20);
  if (prev_base==base_angle && prev_shoulder==shoulder_angle && positionChanged==0)
  {                          //eger tum eklemler istenilen aci degerine ulasirsa daha sonra gripper sabit alma veya bırakma hareketini yapacaktır.
    if (gripperState==0)      //gripper stateler moveit script dosyasindan gonderilecek.
    {
      //gripper_hold();
    }
    if (gripperState==1)
    {
      //gripper_release();
    }
    positionChanged = 1;
  }
  else if ((prev_base!=base_angle || prev_shoulder!=shoulder_angle) && positionChanged==1)
```

```
  {
    positionChanged = 0;
  }
  prev_base = base_angle;
  prev_shoulder = shoulder_angle;
}
ros::Subscriber<sensor_msgs::JointState> sub("joint_states", servo_cb);
ros::Subscriber<std_msgs::Int32> sub1("gripper_state", &gripper_cb);
void gripper_hold()
{
 digitalWrite(hava_role_pin,HIGH);            // Röleyi Kapali konuma getir
 digitalWrite(miknatis_role_pin,HIGH);       // Röleyi Kapali konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,LOW);          // Röleyi açik konuma getir
 delay(role_delay_sure);
 digitalWrite(miknatis_role_pin,LOW);     // Röleyi açik konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,HIGH);         // Röleyi Kapali konuma getir
}
void gripper_release()
{
 digitalWrite(hava_role_pin,LOW);          // Röleyi açik konuma getir
 delay(2*role_delay_sure);
 digitalWrite(miknatis_role_pin,HIGH);     // Röleyi Kapali konuma getir
 delay(role_delay_sure);
 digitalWrite(hava_role_pin,HIGH);         // Röleyi Kapali konuma getir
}
void relay_locking()
{
  digitalWrite(hava_role_pin,HIGH);
  digitalWrite(miknatis_role_pin,HIGH);
}
void setup(){
```

```
  Serial1.begin(115200);  //servo motorlara angle degerini girmek icin
  nh.getHardware()->setBaud(115200);   //haberlesme hizi belirlendi.
  nh.initNode();
  nh.subscribe(sub);
  nh.subscribe(sub1);
  //----------Relay & mosfet Locking ----------------------//
  // Rölenin bağlı olduğu role_pin numaralı pini çıkış pini olarak ayarla
  pinMode(hava_role_pin,OUTPUT);
  pinMode(miknatis_role_pin,OUTPUT);
  relay_locking();
  delay(1);
  int rq11=(base_angle_offset);    //home position robot gonderilir.
  int rq22=(shoulder_angle_offset);
  float q1=map(rq11,0,240,0,1000);
  float q2=map(rq22,0,240,0,1000);
  LobotSerialServoMove(Serial1, ID1, q1,3000);  //720
  LobotSerialServoMove(Serial1, ID2, q2, 3000); //720
  base_angle_pos=LobotSerialServoReadPosition(Serial1, ID1);
  shoulder_angle_pos=LobotSerialServoReadPosition(Serial1, ID2);
  //Serial.println(base_angle_pos);
  //Serial.println(shoulder_angle_pos);
  delay(20);
  //gripper.write(0);
}
void loop(){
  nh.spinOnce();
}
double radiansToDegrees(float position_radians)
{
  float position_degree=position_radians * 57.2958;
  return position_degree;}
```

Appendix B

Codes for Ur5 Robot Arm

Ur5 codes

arm_description package

Launch

- ✓ ur5_upload.launch

```
<?xml version="1.0"?>
<launch>
 <arg name="limited" default="false" doc="If true, limits joint range [-PI, PI] on all joints." />
 <arg name="transmission_hw_interface" default="hardware_interface/PositionJointInterface" />
 <param unless="$(arg limited)" name="robot_description" command="$(find xacro)/xacro --inorder '$(find arm_description)/urdf/arm.urdf.xacro' transmission_hw_interface:=$(arg transmission_hw_interface)" />
 <param if="$(arg limited)" name="robot_description" command="$(find xacro)/xacro --inorder '$(find arm_description)/urdf/arm_joint_limited.urdf.xacro' transmission_hw_interface:=$(arg transmission_hw_interface)" />
</launch>
```

- ✓ view_ur5.launch

```
<?xml version="1.0"?>
<launch>
 <include file="$(find arm_description)/launch/ur5_upload.launch"/>
 <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" >
   <param name="use_gui" value="true"/>
 </node>
 <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
```

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find arm_description)/config/view_robot.rviz" required="true" />

</launch>

Scripts

- ✓ joint_states_to_gazebo.py

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Header
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.position)
    pub = rospy.Publisher('arm_controller/command', JointTrajectory, queue_size=20)
    joints_str = JointTrajectory()
    joints_str.header = Header()
    joints_str.header.stamp = rospy.Time.now()
    joints_str.joint_names = ['elbow_joint', 'shoulder_lift_joint','shoulder_pan_joint','wrist1_joint','wrist2_joint','wrist3_joint']
    point=JointTrajectoryPoint()
    point.positions = [data.position[0], data.position[1],data.position[2],data.position[3],data.position[4],data.position[5]]
    point.time_from_start = rospy.Duration(2)
    joints_str.points.append(point)
    pub.publish(joints_str)
    rospy.loginfo("position updated")
def listener():
    rospy.init_node('states', anonymous=True)
    rospy.Subscriber("joint_states", JointState, callback)
    rospy.spin()
if __name__ == '__main__':
    try:
        listener()
```

```
    except rospy.ROSInterruptException:
        pass
```

Urdf

- ✓ arm.urdf.xacro

```
<?xml version="1.0"?>
<robot name="arm" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <link name="world"/>
  <link name="pedestal">
    <inertial>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
      <mass value="20" />
      <inertia ixx="200" ixy="200" ixz="200" iyy="200" iyz="200" izz="200" />
    </inertial>
    <visual>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
      <geometry>
        <cylinder radius="0.1" length="1"/>
      </geometry>
      <material name="Red">
        <color rgba="0.8 0.0 0.0 1"/>
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
      <geometry>
        <cylinder radius="0.1" length="1"/>
      </geometry>
    </collision>
  </link>
  <gazebo reference="pedestal">
      <mu1>0.2</mu1>
      <mu2>0.2</mu2>
      <material>Gazebo/Orange</material>
```

```
  </gazebo>
  <joint name="world_joint" type="fixed">
    <parent link="world"/>
    <child link="pedestal"/>
    <origin xyz="0 0 0" rpy="0.0 0.0 0.0"/>
  </joint>
  <xacro:arg name="transmission_hw_interface"
default="hardware_interface/PositionJointInterface"/>
  <!-- common stuff -->
  <xacro:include filename="$(find ur_description)/urdf/common.gazebo.xacro" />
  <!-- ur5 -->
  <xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro" />
  <!-- arm -->
  <xacro:ur5_robot prefix="" joint_limited="false" transmission_hw_interface="$(arg
transmission_hw_interface)" />
  <joint name="base_joint" type="fixed">
    <parent link= "pedestal" />
    <child link = "base_link" />
    <origin xyz="0.0 0.0 1.0" rpy="0.0 0.0 0.0" />
  </joint>
</robot>
```

- ✓ arm_joint_limited.urdf.xacro

```
<?xml version="1.0"?>
<robot name="arm" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <link name="world"/>
  <link name="pedestal">
    <inertial>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
      <mass value="20" />
      <inertia ixx="200" ixy="200" ixz="200" iyy="200" iyz="200" izz="200" />
    </inertial>
    <visual>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
```

```
      <geometry>
        <cylinder radius="0.1" length="1"/>
      </geometry>
      <material name="Red">
        <color rgba="0.8 0.0 0.0 1"/>
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0.5" rpy="0 0 0" />
      <geometry>
        <cylinder radius="0.1" length="1"/>
      </geometry>
    </collision>
  </link>
  <gazebo reference="pedestal">
      <mu1>0.2</mu1>
      <mu2>0.2</mu2>
      <material>Gazebo/Orange</material>
  </gazebo>
  <joint name="world_joint" type="fixed">
    <parent link="world"/>
    <child link="pedestal"/>
    <origin xyz="0 0 0" rpy="0.0 0.0 0.0"/>
  </joint>
  <xacro:arg                                    name="transmission_hw_interface"
default="hardware_interface/PositionJointInterface"/>
  <!-- common stuff -->
  <xacro:include filename="$(find ur_description)/urdf/common.gazebo.xacro" />
  <!-- ur5 -->
  <xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro" />
  <!-- arm -->
  <xacro:ur5_robot prefix="" joint_limited="true"
    shoulder_pan_lower_limit="${-pi}" shoulder_pan_upper_limit="${pi}"
```

```
    shoulder_lift_lower_limit="${-pi}" shoulder_lift_upper_limit="${pi}"
    elbow_joint_lower_limit="${-pi}" elbow_joint_upper_limit="${pi}"
    wrist_1_lower_limit="${-pi}" wrist_1_upper_limit="${pi}"
    wrist_2_lower_limit="${-pi}" wrist_2_upper_limit="${pi}"
    wrist_3_lower_limit="${-pi}" wrist_3_upper_limit="${pi}"
    transmission_hw_interface="$(arg transmission_hw_interface)"
  />
  <joint name="base_joint" type="fixed">
    <parent link= "pedestal" />
    <child link = "base_link" />
    <origin xyz="0.0 0.0 1.0" rpy="0.0 0.0 0.0" />
  </joint>
</robot>
```

arm_gazebo package

Controller

- ✓ arm_controller_ur5.yaml

```
arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
     - shoulder_pan_joint
     - shoulder_lift_joint
     - elbow_joint
     - wrist_1_joint
     - wrist_2_joint
     - wrist_3_joint
  constraints:
      goal_time: 0.6
      stopped_velocity_tolerance: 0.05
      shoulder_pan_joint: {trajectory: 0.1, goal: 0.1}
      shoulder_lift_joint: {trajectory: 0.1, goal: 0.1}
      elbow_joint: {trajectory: 0.1, goal: 0.1}
      wrist_1_joint: {trajectory: 0.1, goal: 0.1}
      wrist_2_joint: {trajectory: 0.1, goal: 0.1}
```

```
    wrist_3_joint: {trajectory: 0.1, goal: 0.1}
  stop_trajectory_duration: 0.5
  state_publish_rate:  25
  action_monitor_rate: 10
joint_group_position_controller:
  type: position_controllers/JointGroupPositionController
  joints:
    - shoulder_pan_joint
    - shoulder_lift_joint
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
```

✓ joint_state_controller.yaml

```
joint_state_controller:
   type: joint_state_controller/JointStateController
   publish_rate: 50
```

✓ controller_utils.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Robot state publisher -->
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" type="string" value="" />
  </node>
  <!-- Fake Calibration -->
  <node pkg="rostopic" type="rostopic" name="fake_joint_calibration"
      args="pub /calibrated std_msgs/Bool true" />
  <!-- joint_state_controller -->
  <rosparam file="$(find ur_gazebo)/controller/joint_state_controller.yaml"
command="load"/>
```

```
 <node name="joint_state_controller_spawner" pkg="controller_manager" type="controller_manager" args="spawn joint_state_controller" respawn="false" output="screen"/>
</launch>
```

- ✓ ur5.launch

```
<?xml version="1.0"?>
<launch>
 <arg name="limited" default="false"  doc="If true, limits joint range [-PI, PI] on all joints." />
 <arg name="paused" default="false" doc="Starts gazebo in paused mode" />
 <arg name="gui" default="true" doc="Starts gazebo gui" />
 <!-- startup simulated world -->
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" default="worlds/empty.world"/>
  <arg name="paused" value="$(arg paused)"/>
  <arg name="gui" value="$(arg gui)"/>
 </include>
 <!-- send robot urdf to param server -->
 <include file="$(find arm_description)/launch/ur5_upload.launch">
  <arg name="limited" value="$(arg limited)"/>
 </include>
 <!-- push robot_description to factory and spawn robot in gazebo -->
 <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -z 0.1" respawn="false" output="screen" />
 <include file="$(find arm_gazebo)/launch/controller_utils.launch"/>
 <!-- start this controller -->
 <rosparam file="$(find arm_gazebo)/controller/arm_controller_ur5.yaml" command="load"/>
 <node name="arm_controller_spawner" pkg="controller_manager" type="controller_manager" args="spawn arm_controller" respawn="false" output="screen"/>
 <!-- load other controllers -->
```

<node name="ros_control_controller_manager" pkg="controller_manager" type="controller_manager" respawn="false" output="screen" args="load joint_group_position_controller" />

</launch>

- ✓ ur5_joint_limited.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="gui" default="true" doc="Starts gazebo gui" />
  <include file="$(find ur_gazebo)/launch/ur5.launch">
    <arg name="limited" value="true"/>
    <arg name="gui" value="$(arg gui)"/>
  </include>
</launch>
```

ur5_moveit_config package

Config

- ✓ controllers.yaml

```
controller_list:
  - name: ""
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
      - elbow_joint
      - wrist_1_joint
      - wrist_2_joint
      - wrist_3_joint
```

- ✓ fake_controllers.yaml

```
controller_list:
  - name: fake_manipulator_controller
    joints:
      - shoulder_pan_joint
      - shoulder_lift_joint
```

```
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
  - name: fake_endeffector_controller
    joints:
      []
    ✓ joint_limits.yaml
# joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed
# Specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [has_velocity_limits, has_acceleration_limits]
joint_limits:
  shoulder_pan_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    has_acceleration_limits: true
    max_acceleration: 3.15
  shoulder_lift_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    has_acceleration_limits: true
    max_acceleration: 3.15
  elbow_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    has_acceleration_limits: true
    max_acceleration: 3.15
  wrist_1_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    has_acceleration_limits: true
```

```
    max_acceleration: 3.2
  wrist_2_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    has_acceleration_limits: true
    max_acceleration: 3.2
  wrist_3_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    has_acceleration_limits: true
    max_acceleration: 3.2
```

✓ kinematics.yaml

```
#manipulator:
#  kinematics_solver: ur_kinematics/UR5KinematicsPlugin
#  kinematics_solver_search_resolution: 0.005
#  kinematics_solver_timeout: 0.005
#  kinematics_solver_attempts: 3
manipulator:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
  kinematics_solver_attempts: 3
```

✓ ompl_planning.yaml

```
planner_configs:
  SBLkConfigDefault:
    type: geometric::SBL
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set
on setup()
  ESTkConfigDefault:
    type: geometric::EST
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0 setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability. default: 0.05
  LBKPIECEkConfigDefault:
```

type: geometric::LBKPIECE

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

border_fraction: 0.9 # Fraction of time focused on boarder default: 0.9

min_valid_path_fraction: 0.5 # Accept partially valid moves above fraction. default: 0.5

BKPIECEkConfigDefault:

type: geometric::BKPIECE

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

border_fraction: 0.9 # Fraction of time focused on boarder default: 0.9

failed_expansion_score_factor: 0.5 # When extending motion fails, scale score by factor. default: 0.5

min_valid_path_fraction: 0.5 # Accept partially valid moves above fraction. default: 0.5

KPIECEkConfigDefault:

type: geometric::KPIECE

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

goal_bias: 0.05 # When close to goal select goal, with this probability. default: 0.05

border_fraction: 0.9 # Fraction of time focused on boarder default: 0.9 (0.0,1.]

failed_expansion_score_factor: 0.5 # When extending motion fails, scale score by factor. default: 0.5

min_valid_path_fraction: 0.5 # Accept partially valid moves above fraction. default: 0.5

RRTkConfigDefault:

type: geometric::RRT

range: 0.0 # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()

goal_bias: 0.05 # When close to goal select goal, with this probability? default: 0.05

RRTConnectkConfigDefault:

type: geometric::RRTConnect

```
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
  RRTstarkConfigDefault:
    type: geometric::RRTstar
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
    delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found. default 1
  TRRTkConfigDefault:
    type: geometric::TRRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
    max_states_failed: 10  # when to start increasing temp. default: 10
    temp_change_factor: 2.0  # how much to increase or decrease temp. default: 2.0
    min_temperature: 10e-10  # lower limit of temp change. default: 10e-10
    init_temperature: 10e-6  # initial temperature. default: 10e-6
    frountier_threshold: 0.0  # dist new state to nearest neighbor to disqualify as frontier. default: 0.0 set in setup()
    frountierNodeRatio: 0.1  # 1/10, or 1 nonfrontier for every 10 frontier. default: 0.1
    k_constant: 0.0  # value used to normalize expresssion. default: 0.0 set in setup()
  PRMkConfigDefault:
    type: geometric::PRM
    max_nearest_neighbors: 10  # use k nearest neighbors. default: 10
  PRMstarkConfigDefault:
    type: geometric::PRMstar
manipulator:
  planner_configs:
    - SBLkConfigDefault
    - ESTkConfigDefault
    - LBKPIECEkConfigDefault
    - BKPIECEkConfigDefault
```

```
    - KPIECEkConfigDefault
    - RRTkConfigDefault
    - RRTConnectkConfigDefault
    - RRTstarkConfigDefault
    - TRRTkConfigDefault
    - PRMkConfigDefault
    - PRMstarkConfigDefault
  ##Note: commenting the following line lets moveit chose RRTConnect as default planner rather than LBKPIECE
  #projection_evaluator: joints(shoulder_pan_joint,shoulder_lift_joint)
  longest_valid_segment_fraction: 0.01
endeffector:
  planner_configs:
    - SBLkConfigDefault
    - ESTkConfigDefault
    - LBKPIECEkConfigDefault
    - BKPIECEkConfigDefault
    - KPIECEkConfigDefault
    - RRTkConfigDefault
    - RRTConnectkConfigDefault
    - RRTstarkConfigDefault
    - TRRTkConfigDefault
    - PRMkConfigDefault
    - PRMstarkConfigDefault
```

Launch

- ✓ default_warehouse_db.launch

```
<launch>
  <arg name="reset" default="false"/>
  <!-- Launch the warehouse with a default database location -->
  <include file="$(find ur5_moveit_config)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path" value="$(find ur5_moveit_config)/default_warehouse_mongo_db" />
```

```
  </include>
  <!-- If we want to reset the database, run this node -->
  <node if="$(arg reset)" name="$(anon moveit_default_db_reset)"
type="moveit_init_demo_warehouse" pkg="moveit_ros_warehouse"
respawn="false" output="screen" />
</launch>
```

- ✓ demo.launch

```
<launch>
  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />

  <arg name="limited" default="false"/>

  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server
-->
  <include file="$(find ur5_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
    <arg name="limited" value="$(arg limited)"/>
  </include>
  <!-- If needed, broadcast static tf for robot root -->
  <!-- We do not have a robot connected, so publish fake joint states -->
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam
param="/source_list">[/move_group/fake_controller_joint_states]</rosparam>
  </node>
  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="true" output="screen" />
```

```
  <!-- Run the main MoveIt executable without trajectory execution (we do not have controllers configured by default) -->
  <include file="$(find ur5_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <!-- Run Rviz and load the default config to see the state of the move_group node -->
  <include file="$(find ur5_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <!-- If database loading was enabled, start mongodb as well -->
  <include file="$(find ur5_moveit_config)/launch/default_warehouse_db.launch" if="$(arg db)"/>
</launch>
```

- ✓ fake_moveit_controller_manager.launch

```
<launch>
  <!-- Set the param that trajectory_execution_manager needs to find the controller plugin -->
  <param name="moveit_controller_manager" value="moveit_fake_controller_manager/MoveItFakeControllerManager"/>
  <!-- The rest of the params are specific to this plugin -->
  <rosparam file="$(find ur5_moveit_config)/config/fake_controllers.yaml"/>
</launch>
```

- ✓ move_group.launch

```
<launch>
  <arg name="limited" default="false"/>
  <include file="$(find ur5_moveit_config)/launch/planning_context.launch">
    <arg name="limited" value="$(arg limited)" />
  </include>
```

```
  <!-- GDB Debug Option -->
  <arg name="debug" default="false" />
  <arg unless="$(arg debug)" name="launch_prefix" value="" />
  <arg     if="$(arg debug)" name="launch_prefix" value="gdb --ex run --args" />
  <!-- Verbose Mode Option -->
  <arg name="info" default="$(arg debug)" />
  <arg unless="$(arg info)" name="command_args" value="" />
  <arg     if="$(arg info)" name="command_args" value="--debug" />
  <!-- move_group settings -->
  <arg name="allow_trajectory_execution" default="true"/>
  <arg name="fake_execution" default="false"/>
  <arg name="max_safe_path_cost" default="1"/>
  <arg name="jiggle_fraction" default="0.05" />
  <arg name="publish_monitored_planning_scene" default="true"/>
  <!-- Planning Functionality -->
  <include                    ns="move_group"                    file="$(find
ur5_moveit_config)/launch/planning_pipeline.launch.xml">
    <arg name="pipeline" value="ompl" />
  </include>
  <!-- Trajectory Execution Functionality -->
  <include                    ns="move_group"                    file="$(find
ur5_moveit_config)/launch/trajectory_execution.launch.xml"          if="$(arg
allow_trajectory_execution)">
    <arg name="moveit_manage_controllers" value="true" />
    <arg      name="moveit_controller_manager"      value="ur5"      unless="$(arg
fake_execution)"/>
    <arg      name="moveit_controller_manager"      value="fake"      if="$(arg
fake_execution)"/>
  </include>
  <!-- Sensors Functionality -->
  <include                    ns="move_group"                    file="$(find
ur5_moveit_config)/launch/sensor_manager.launch.xml"          if="$(arg
allow_trajectory_execution)">
```

```
    <arg name="moveit_sensor_manager" value="ur5" />
  </include>
  <!-- Start the actual move_group node/action server -->
  <node name="move_group" launch-prefix="$(arg launch_prefix)"
pkg="moveit_ros_move_group" type="move_group" respawn="false"
output="screen" args="$(arg command_args)">
    <!-- Set the display variable, in case OpenGL code is used internally -->
    <env name="DISPLAY" value="$(optenv DISPLAY :0)" />
    <param name="allow_trajectory_execution" value="$(arg
allow_trajectory_execution)"/>
    <param name="max_safe_path_cost" value="$(arg max_safe_path_cost)"/>
    <param name="jiggle_fraction" value="$(arg jiggle_fraction)" />
    <!-- MoveGroup capabilities to load -->
    <param name="capabilities"
value="move_group/MoveGroupCartesianPathService
                move_group/MoveGroupExecuteTrajectoryAction
                move_group/MoveGroupKinematicsService
                move_group/MoveGroupMoveAction
                move_group/MoveGroupPickPlaceAction
                move_group/MoveGroupPlanService
                move_group/MoveGroupQueryPlannersService
                move_group/MoveGroupStateValidationService
                move_group/MoveGroupGetPlanningSceneService
                move_group/ClearOctomapService
                " />

    <!-- Publish the planning scene of the physical robot so that rviz plugin can know
actual robot -->
    <param name="planning_scene_monitor/publish_planning_scene" value="$(arg
publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/publish_geometry_updates" value="$(arg
publish_monitored_planning_scene)" />
```

```
  <param name="planning_scene_monitor/publish_state_updates" value="$(arg publish_monitored_planning_scene)" />
  <param name="planning_scene_monitor/publish_transforms_updates" value="$(arg publish_monitored_planning_scene)" />
 </node>
</launch>
```

- ✓ moveit_rviz.launch

```
<launch>
 <arg name="debug" default="false" />
 <arg unless="$(arg debug)" name="launch_prefix" value="" />
 <arg     if="$(arg debug)" name="launch_prefix" value="gdb --ex run --args" />

 <arg name="config" default="false" />
 <arg unless="$(arg config)" name="command_args" value="" />
 <arg if="$(arg config)" name="command_args" value="-d $(find ur5_moveit_config)/launch/moveit.rviz" />

 <node name="$(anon rviz)" launch-prefix="$(arg launch_prefix)" pkg="rviz" type="rviz" respawn="false"
       args="$(arg command_args)" output="screen">
  <rosparam command="load" file="$(find ur5_moveit_config)/config/kinematics.yaml"/>
 </node>
</launch>
```

- ✓ ompl_planning_pipeline.launch.xml

```
<launch>
 <!-- OMPL Plugin for MoveIt! -->
 <arg name="planning_plugin" value="ompl_interface/OMPLPlanner" />
 <!-- The request adapters (plugins) used when planning with OMPL.
     ORDER MATTERS -->
 <arg name="planning_adapters" value="default_planner_request_adapters/AddTimeParameterization
                                      default_planner_request_adapters/FixWorkspaceBounds
```

default_planner_request_adapters/FixStartStateBounds
default_planner_request_adapters/FixStartStateCollision

default_planner_request_adapters/FixStartStatePathConstraints" />
<arg name="start_state_max_bounds_error" value="0.1" />
<param name="planning_plugin" value="$(arg planning_plugin)" />
<param name="request_adapters" value="$(arg planning_adapters)" />
<param name="start_state_max_bounds_error" value="$(arg start_state_max_bounds_error)" />
<rosparam command="load" file="$(find ur5_moveit_config)/config/ompl_planning.yaml"/>
</launch>

- ✓ planning_context.launch

<launch>
<!-- By default we do not overwrite the URDF. Change the following to true to change the default behavior -->
<arg name="load_robot_description" default="false"/>
<arg name="limited" default="false"/>
<!-- The name of the parameter under which the URDF is loaded -->
<arg name="robot_description" default="robot_description"/>
<!-- Load universal robot description format (URDF) -->
<group if="$(arg load_robot_description)">
<param unless="$(arg limited)" name="$(arg robot_description)" command="$(find xacro)/xacro --inorder '$(find ur_description)/urdf/ur5_robot.urdf.xacro'" />
<param if="$(arg limited)" name="$(arg robot_description)" command="$(find xacro)/xacro --inorder '$(find ur_description)/urdf/ur5_joint_limited_robot.urdf.xacro'" />
</group>
<!-- The semantic description that corresponds to the URDF -->
<param name="$(arg robot_description)_semantic" textfile="$(find ur5_moveit_config)/config/ur5.srdf" />
<!-- Load updated joint limits (override information from URDF) -->

```
  <group ns="$(arg robot_description)_planning">
    <rosparam command="load" file="$(find ur5_moveit_config)/config/joint_limits.yaml"/>
  </group>
  <!-- Load default settings for kinematics; these settings are overridden by settings in a node's namespace -->
  <group ns="$(arg robot_description)_kinematics">
    <rosparam command="load" file="$(find ur5_moveit_config)/config/kinematics.yaml"/>
  </group>
</launch>
```

- ✓ planning_pipeline.launch.xml

```
<launch>
  <!-- This file makes it easy to include different planning pipelines;
       It is assumed that all planning pipelines are named XXX_planning_pipeline.launch  -->
  <arg name="pipeline" default="ompl" />

  <include file="$(find ur5_moveit_config)/launch/$(arg pipeline)_planning_pipeline.launch.xml" />
</launch>
```

- ✓ run_benchmark_ompl.launch

```
<launch>
  <!-- This argument must specify the list of .cfg files to process for benchmarking -->
  <arg name="cfg" />
  <!-- Load URDF -->
  <include file="$(find ur5_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>
  <!-- Start the database -->
  <include file="$(find ur5_moveit_config)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path" value="moveit_ompl_benchmark_warehouse"/>
```

```
  </include>
  <!-- Start Benchmark Executable -->
  <node name="$(anon moveit_benchmark)" pkg="moveit_ros_benchmarks" type="moveit_run_benchmark" args="$(arg cfg) --benchmark-planners" respawn="false" output="screen">
    <rosparam command="load" file="$(find ur5_moveit_config)/config/kinematics.yaml"/>
    <rosparam command="load" file="$(find ur5_moveit_config)/config/ompl_planning.yaml"/>
  </node>
</launch>
```

- ✓ sensor_manager.launch.xml

```
<launch>
  <!-- This file makes it easy to include the settings for sensor managers -->
  <!-- Params for the octomap monitor -->
  <!--  <param name="octomap_frame" type="string" value="some frame in which the robot moves" /> -->
  <param name="octomap_resolution" type="double" value="0.025" />
  <param name="max_range" type="double" value="5.0" />
  <!-- Load the robot specific sensor manager; this sets the moveit_sensor_manager ROS parameter -->
  <arg name="moveit_sensor_manager" default="ur5" />
  <include file="$(find ur5_moveit_config)/launch/$(arg moveit_sensor_manager)_moveit_sensor_manager.launch.xml" />
</launch>
```

- ✓ trajectory_execution.launch.xml

```
<launch>
  <!-- This file makes it easy to include the settings for trajectory execution  -->
  <!-- Flag indicating whether MoveIt! is allowed to load/unload  or switch controllers -->
  <arg name="moveit_manage_controllers" default="true"/>
  <param name="moveit_manage_controllers" value="$(arg moveit_manage_controllers)"/>
```

<!-- When determining the expected duration of a trajectory, this multiplicative factor is applied to get the allowed duration of execution -->
<param name="trajectory_execution/allowed_execution_duration_scaling" value="1.2"/> <!-- default 1.2 -->
<!-- Allow more than the expected execution time before triggering a trajectory cancel (applied after scaling) -->
<param name="trajectory_execution/allowed_goal_duration_margin" value="0.5"/> <!-- default 0.5 -->
<!-- Load the robot specific controller manager; this sets the moveit_controller_manager ROS parameter -->
<arg name="moveit_controller_manager" default="ur5" />
<include file="$(find ur5_moveit_config)/launch/$(arg moveit_controller_manager)_moveit_controller_manager.launch.xml" />
</launch>

- ✓ ur5_moveit_controller_manager.launch.xml

<launch>
<rosparam file="$(find ur5_moveit_config)/config/controllers.yaml"/>
<param name="use_controller_manager" value="false"/>
<param name="trajectory_execution/execution_duration_monitoring" value="false"/>
<param name="moveit_controller_manager" value="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>
</launch>

- ✓ ur5_moveit_planning_execution.launch

<launch>
<arg name="sim" default="false" />
<arg name="limited" default="false"/>
<arg name="debug" default="false" />
<!-- Remap follow_joint_trajectory -->
<remap if="$(arg sim)" from="/follow_joint_trajectory" to="/arm_controller/follow_joint_trajectory"/>
<!-- Launch moveit -->
<include file="$(find ur5_moveit_config)/launch/move_group.launch">

```
    <arg name="limited" default="$(arg limited)"/>
    <arg name="debug" default="$(arg debug)" />
  </include>
</launch>
```

- ✓ warehouse.launch

```
<launch>
  <!-- The path to the database must be specified -->
  <arg name="moveit_warehouse_database_path" />
  <!-- Load warehouse parameters -->
  <include file="$(find ur5_moveit_config)/launch/warehouse_settings.launch.xml"
/>
  <!-- Run the DB server -->
  <node name="$(anon mongo_wrapper_ros)" cwd="ROS_HOME"
type="mongo_wrapper_ros.py" pkg="warehouse_ros">
    <param name="overwrite" value="false"/>
    <param name="database_path" value="$(arg moveit_warehouse_database_path)"
/>
  </node>
</launch>
```

- ✓ warehouse_settings.launch.xml

```
<launch>
  <!-- Set the parameters for the warehouse and run the mongodb server. -->
  <!-- The default DB port for moveit (not default MongoDB port to avoid potential
conflicts) -->
  <arg name="moveit_warehouse_port" default="33829" />
  <!-- The default DB host for moveit -->
  <arg name="moveit_warehouse_host" default="localhost" />
  <!-- Set parameters for the warehouse -->
  <param name="warehouse_port" value="$(arg moveit_warehouse_port)"/>
  <param name="warehouse_host" value="$(arg moveit_warehouse_host)"/>
  <param name="warehouse_exec" value="mongod" />
</launch>
```

Tests

- ✓ moveit_planning_execution.xml

```
<?xml version="1.0"?>
<launch>
  <arg name="robot_ip" value="127.0.0.1" />
  <group>
    <include file="$(find ur5_moveit_config)/launch/ur5_moveit_planning_execution.launch">
      <arg name="sim" value="false" />
    </include>
  </group>
  <group ns="sim">
    <include file="$(find ur5_moveit_config)/launch/ur5_moveit_planning_execution.launch">
      <arg name="sim" value="true" />
    </include>
  </group>
</launch>
```

- ✓ ur5_trajectory.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
group_name = "manipulator"
```

```
move_group = moveit_commander.MoveGroupCommander(group_name)
display_trajectory_publisher =
rospy.Publisher("/move_group/display_planned_path",moveit_msgs.msg.DisplayTra
jectory,queue_size=20)
pose_goal = geometry_msgs.msg.Pose()
pose_goal.orientation.w = 1.0
pose_goal.position.x = 0.1
pose_goal.position.y = 0.2
pose_goal.position.z = 0.5
move_group.set_pose_target(pose_goal)
plan = move_group.go(wait=True)
move_group.stop()
move_group.clear_pose_targets()
#rospy.spin()
#roscpp_shutdown()
```

- ✓ ur5_jointspace_trajectory.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list

moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("manipulator")
```

```
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory,queue_size=20)
# We can get the joint values from the group and adjust some of the values:
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0
joint_goal[1] = -pi/3
joint_goal[2] = 0
joint_goal[3] = -pi/4
joint_goal[4] = 0
joint_goal[5] = pi/2

# The go command can be called with joint values, poses, or without any
# parameters if you have already set the pose or joint target for the group
move_group.go(joint_goal, wait=True)
# Calling ``stop()`` ensures that there is no residual movement
move_group.stop()
```

- ✓ ur5_jointspace_trajectory_solid.py

```
#! /usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
import math
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface_tutorial', anonymous=True)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("manipulator")
```

```
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory,queue_size=20)
def talker(x,y,z,xx,yy,zz):
    rate = rospy.Rate(100) # 10hz
    # We can get the joint values from the group and adjust some of the values:
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = x
    joint_goal[1] = y
    joint_goal[2] = z
    joint_goal[3] = xx
    joint_goal[4] = yy
    joint_goal[5] = zz
    # The go command can be called with joint values, poses, or without any
    # parameters if you have already set the pose or joint target for the group
    move_group.go(joint_goal, wait=True)
    # Calling ``stop()`` ensures that there is no residual movement
    move_group.stop()
    rate.sleep()
if __name__ == '__main__':
    try:
        # opening the file in read mode
        my_file = open("full.txt", "r")
        # reading the file
        data = my_file.read()
        print(data)
        # replacing end splitting the text
        # when newline ('\n' or ',') is seen.
        data_into_list = data.split('\n')
        del data_into_list[len(data_into_list)-1]
        #print(data_into_list)
        list = []
        for element in data_into_list:
            list.append(float(element))
```

```
        #print(list)
        my_file.close()
        # Python3 code to iterate over a list
        # Getting length of list
        length = len(list)
        # Iterating using while loop
        for i in range(0,length,6):
          x =(list[i])*(math.pi/180.0)
           y=list[i+1]*(math.pi/180.0)
          z=list[i+2]*(math.pi/180.0)
          xx=list[i+3]*(math.pi/180.0)
          yy=list[i+4]*(math.pi/180.0)
          zz=list[i+5]*(math.pi/180.0)
           #print(x,y)
          talker(x,y,z,xx,yy,zz)
          if i==length-6:
                rate = rospy.Rate(1) # 10hz
                 rate.sleep()
              talker(list[length-6]*(math.pi/180.0),list[length-
5]*(math.pi/180.0),list[length-4]*(math.pi/180.0),list[length-
3]*(math.pi/180.0),list[length-2]*(math.pi/180.0),list[length-1]*(math.pi/180.0))
    except rospy.ROSInterruptException:
       pass
```

- ✓ full.txt

include joints angle values

Curriculum Vitae

Name Surname : Adem Candemir

Education:

2010–2014	Samandıra Highschool, Natural and Applied Sciences
2018–2019	İzmir Kâtip Çelebi University, Dept. of Mechatronic Eng.
2019–2022	İzmir Kâtip Çelebi University, Natural and Applied Sciences, Dept. of Mechanical Eng.

Work Experience:

2019 – 2020	Massive Process A.Ş
2020 – 2022	Etkin Tıbbi Cihazlar A.Ş

Publications (if any):

1. Pick&Place Task Implementation of a Scara Manipulator via Robot Operating System and Machine Vision, International Conference on Intelligence and Safety for Robotics, published in International Journal of Mechanical Engineering