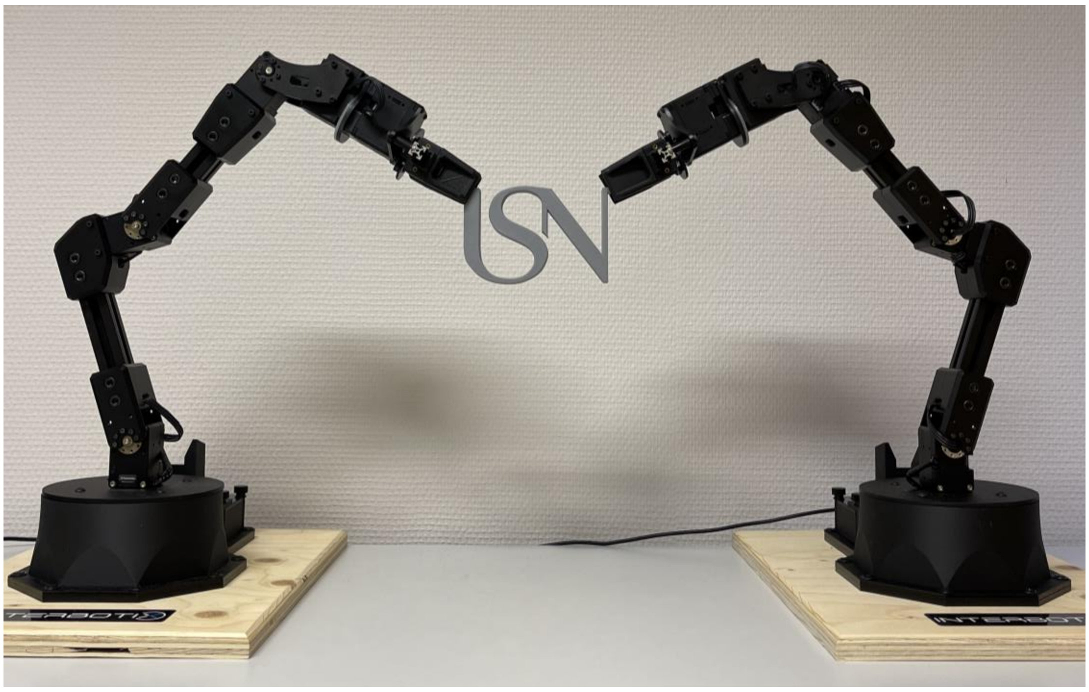
FMH606 Master's Thesis 2023

Master of Science – Industrial IT and Automation

Development of a simulation platform and testing of a 5 degrees of freedom ReactorX-150 robotic arm manipulator.



Christian Lauritzen

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2023

Title: Development of a simulation platform and testing of a 5 degrees of freedom ReactorX-150 robotic arm manipulator.

Number of pages: 182

Keywords: Robot, robotic arm, ReactorX-150, simulation, testing,

Student: Christian Lauritzen

Supervisor: Associate Professor Roshan Sharma (USN)

External partner:

Summary:

The University of South-Eastern Norway recently purchased several units of the Trossen Robotics ReactorX-150 educational robots, enabling campus students' access to some great robots for education and research purposes. The same possibility could be offered to online students with the access to a simulation platform for the ReactorX-150.

The primary objective of this thesis is to create a simulation platform for the ReactorX-150, integrating it with ROS on a Linux platform. This platform aims to support essential tasks such as position/trajectory control and the application of forward and inverse kinematics. Key technologies for this task include ROS in combination with Gazebo, RViz, and MoveIt.

With comprehensive testing and documentation of the ReactorX-150 and its control packages, the thesis has resulted in numerous guides for setting up the simulation platform for the ReactorX-150, providing students the ability to implement the simulation platform to their computer from any corner of the world.

Preface

This master thesis was written by one student in his final semester in the course “Industrial IT and Automation” at the University of South-Eastern Norway, campus Porsgrunn.

The project was both initiated by and intended for the University of South-Eastern Norway.

Working with the thesis has been challenging and time consuming, but also interesting and rewarding. Knowledge and skills obtained throughout the master course was utilized in this thesis, and new knowledge and skills was acquired while working with the thesis. I am satisfied with the outcome of the work presented here and with the journey this thesis has provided.

I would like to express my appreciation to the staff at the University of South-Eastern Norway, campus Porsgrunn throughout the total of five years I have been here, since this was also the place, I also did my bachelor’s degree in the course “Computer science and industrial automation”.

A special thanks to my supervisor, associate professor Roshan Sharma, for the guidance, feedback, and support throughout this entire process.

Finally, I would like to thank family and friends for their patience, understanding and support.

Recommended prior knowledge before reading this thesis is a bachelor’s degree in computer science, automation, cybernetics or similar.

Porsgrunn, 15.05.2023

Christian Lauritzen

Contents

Preface **3**
Contents **4**
Nomenclature **7**
1 Introduction **8**
1.1 Background **8**
1.2 Objectives **9**
1.3 Scope **9**
1.4 Report structure **9**
2 Specifications **10**
2.1 ReactorX-150 ..... **10**
2.1.1 Arm Reach and Joint Names ***11***
2.1.2 Workspace and Working Payload ***12***
2.1.3 Linkage Dimensions and Gripper Limitations ***13***
2.1.4 Default Joint Limits and Default Servo Configurations ***15***
2.1.5 Product of Exponentials Kinematic Properties ***16***
2.1.6 Technical Drawing and CAD files ***17***
2.1.7 X-Series code names ***17***
2.2 Hardware **19**
2.2.1 ReactorX-150 Actuators ***19***
2.2.2 ReactorX-150 Controller ***20***
2.2.3 ReactorX-150 Power hub ***21***
2.2.4 ReactorX-150 Power supply ***22***
2.2.5 3D Printed Custom End Effector ***22***
2.2.6 AprilTag ***23***
2.2.7 Raspberry Pi ***23***
2.2.8 Raspberry Pi Camera Module 2 ..... ***25***
2.2.9 Lumens DC125 ..... ***25***
2.2.10 Storage devices ***25***
2.2.11 Other ***25***
2.3 Software **26**
2.3.1 Ubuntu ***26***
2.3.2 ROS ***26***
2.3.3 IRROS ***27***
2.3.4 MoveIt ***28***
2.3.5 Gazebo ***28***
2.3.6 RViz ***29***
2.3.7 DYNAMIXEL Wizard 2.0 ..... ***29***
2.3.8 BalenaEtcher ***30***
2.3.9 Visual Studio Code ***30***
2.3.10 Main Programming Languages ***30***
2.3.11 Sharpr3D ***31***
2.3.12 FlashPrint 5 ..... ***31***
2.3.13 Windows and Microsoft Office ***31***
3 Denavit–Hartenberg Model **33**
3.1 Kinematic Diagram **33**
3.2 Denavit-Hartenberg parameters **34**

4 ReactorX-150 Implementation....................37
4.1 Arm Descriptions....................37
4.1.1 Structure....................37
4.1.2 Arguments....................38
4.2 Arm Control - Python and MATLAB....................39
4.2.1 Structure....................39
4.2.2 Kinematics without MoveIt....................40
4.2.3 Arguments....................40
4.3 MoveIt Configuration, Interface, and API....................41
4.3.1 Structure....................41
4.3.2 Kinematics with MoveIt....................43
4.3.3 Arguments....................43
4.4 Gazebo Configuration....................44
4.4.1 Structure....................44
4.4.2 Arguments....................45
4.5 ROS Controllers Configuration....................46
4.5.1 Structure....................46
4.5.2 Arguments....................47
4.6 Perception Configuration....................48
4.6.1 Structure....................48
4.6.2 Arguments....................50
4.7 Joystick Control....................51
4.7.1 Structure....................51
4.7.2 Arguments....................52
4.8 Record and Playback....................53
4.8.1 Structure....................53
4.8.2 Arguments....................54
4.9 Arm Diagnostic Tool....................56
4.9.1 Structure....................56
4.9.2 Arguments....................57
4.10 Arm Puppeteering....................59
4.10.1 Physical setup of two ReactorX-150 arms....................59
4.10.2 Structure....................60
4.10.3 Arguments....................61
4.11 Dual Arm Control....................62
4.11.1 Structure....................62
4.11.2 Arguments....................63
4.12 Dual Arm Joystick Control....................64
4.12.1 Structure....................64
4.12.2 Arguments....................65

5 Simulation of ReactorX-150....................67
5.1 Arm Descriptions....................67
5.2 Arm Control....................69
5.2.1 Interbotix Control Panel....................69
5.2.2 Python Scripts....................70
5.2.3 Terminal window commands....................71
5.3 MoveIt....................73
5.4 Joystick Control....................74
5.5 Dual Arm Control....................75
5.5.1 Terminal window commands....................75
5.5.2 Python Scripts....................76
5.6 Dual Arm Joystick Control....................77

6 Physical Testing of ReactorX-150....................78

6.1 Arm Control 78
6.1.1 Interbotix Control Panel 78
6.1.2 Python Scripts 80
6.1.3 Terminal window commands 81
6.2 MoveIt 82
6.3 Joystick Control 84
6.4 Record and Playback 85
6.5 Arm Diagnostic Tool 86
6.6 Arm Puppeteering 87
6.7 Dual Arm Control 88
6.7.1 Terminal window commands 88
6.7.2 Python Scripts 89
6.8 Dual Arm Joystick Control 90

7 Discussion 92

7.1 Motor ID correction 92
7.2 Homing Offset 92
7.3 MATLAB-ROS Interface 95
7.4 Virtual Machines 95
7.5 Network Setup 95
7.6 3D printing 96
7.7 Perception Configuration 98
7.8 Future Work 98
7.8.1 Perception package 98
7.8.2 AprilTag 98
7.8.3 Raspberry Pi 99

8 Conclusion 100

References 101

9 Appendices 107

9.1 Appendix A – Master thesis task description 108
9.2 Appendix B – Guide for Dual Booting Windows and Ubuntu 110
9.3 Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics 112
9.4 Appendix D – Quickstart Guide for the X-Series Arms from Trossen Robotics 114
9.5 Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms116
9.6 Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms149
9.7 Appendix G – Joystick controls 170
9.8 Appendix H – Joystick controller pairing 172
9.9 Appendix I – Raspberry Pi Ubuntu and ROS Setup Guide 174
9.10 Appendix J – bartender.py 178
9.11 Appendix K – “rostopic” guide 179
9.12 Appendix L – xsarm_dual_usn_lift.py 181
9.13 Appendix M – xsarm_dual_usn_down.py 182

Nomenclature

<List symbols alphabetically, with explanations and units>

API – Application Programming Interface

CAD – Computer-Aided Design

DH – Denavit-Hartenberg

Mbps – Megabits per second

N/A – Not Applicable or Not Available

POE – Product of Exponentials

ROS – Robot Operating System

RX-150 – ReactorX-150

STEP – Standard for the Exchange of Product model data [1]

STL – Stereolithography

TTL – Transistor-Transistor Logic

UART – Universal Asynchronous Receiver-Transmitter

VS Code – Visual Studio Code

Xacro – XML Macros

YAML – Yet Another Markup Language/YAML Ain’t Markup Language

1 Introduction

This chapter serves as an introduction to the thesis, which centers around the development of a simulation platform for the ReactorX-150 robotic arm. The introduction includes several key components. First, it provides the background of the thesis, setting the context and relevance of the study. Next, it outlines the objectives, which highlight the primary focus and goals of the thesis. The scope of the thesis, framing its extent. Finally, the report structure is presented to give the reader an overview of the organization and flow of the thesis.

1.1 Background

The field of robotics is rapidly evolving with constant advancement in technology. Robotic arm manipulators have emerged as versatile tools with a wide range of applications. Luckily for humans, robots such as robotic arm manipulators are experts at performing repetitive, tedious, and dangerous jobs. Their ability to perform precise, repetitive tasks has led to increased efficiency and productivity in various settings, such as assembly lines. Robots are an integral part of modern industrial processes and have significantly improved efficiency, productivity, and quality while reducing human error and workplace accidents.

The automotive industry heavily utilizes robotic arm manipulators for tasks such as lifting, panting and quality control. Lifting a car body to a separate location is tiering, risky and takes multiple men, while a Robotic arm manipulator can do that all day. Notably robots and robotic arm manipulators are used in space exploration missions, where they can endure harsh environments that would be lethal to humans.

Robotic arm manipulators are heavily used in the production of parts for aerospace applications. The accuracy, quality, and consistency a robot can deliver is unmatched in the aerospace department.

In electronics/electrical industries, robots are used for assembling tiny and delicate components, especially in the production of items like circuit boards, mobile phones, and computers.

Shipping and trade industry have implemented the use autonomous vehicles for a while, resulting in increased efficiency, decreased delivery times, and reduced labor costs.

The application of robots and robotic arm manipulators is extensive and growing. However, machines do not make themselves quite yet. There are a lot of work behind the creation of robots and the control of robots, so people need to learn about robotics.

The University of South-Eastern Norway recently purchased several units of the Trossen Robotics ReactorX-150 educational robots. The robots are great for research purposes with extensive capabilities. The physical ReactorX-150 units are located at the University of South-Eastern Norway, campus Porsgrunn, which is convenient for campus students. However, that potentially leaves the online and industry master student without the ability to interact with the robotic arms. That where this thesis will attempt to provide a solution.

1.2 Objectives

The primary objective of this thesis is to create a simulation platform for the ReactorX-150, integrating it with ROS on a Linux platform. This platform aims to support essential tasks such as position/trajectory control and the application of forward and inverse kinematics. Key technologies for this task include ROS in combination with Gazebo, RViz, and MoveIt.

Comprehensive testing and documentation are required for both simulated and physical testing of the ReactorX-150. This will ensure the platform's accessibility and usefulness to both online and campus students.

The simulation platform will be constructed in a way that enables students to convert their own computer into their own simulation platform, providing the ability to avoid some challenges regarding geographic location.

1.3 Scope

The project has a set timeframe set between 1.1.2023 and 15.5.2023.

Although the ReactorX-150 robotic arm possesses extensive capabilities, this thesis will not cover all of them. Instead, it will establish a solid foundation for both the simulation and physical testing of the ReactorX-150 robotic arms. However, to fully harness and explore the comprehensive capabilities of the arms, additional work beyond the scope of this thesis will be necessary."

1.4 Report structure

Chapter 2 describes the specifications regarding the ReactorX-150, and the software and hardware utilized in the thesis.

Chapter 3 describes the Denavit–Hartenberg model of the ReactorX-150.

Chapter 4 describes the implementation of the control packages for the Trossen Robotics X-Series arms, specifically the ReactorX-150.

Chapter 5 describes the testing of the control packages with the simulated ReactorX-150.

Chapter 6 describes the testing of the control packages using the physical ReactorX-150

Chapter 7 describes the discussion around the work with the ReactorX-150.

Chapter 8 describes the conclusion of the thesis.

2 Specifications

The specifications chapter provides in-depth information about the specifications related to the thesis. The chapter provides detailed specifications of ReactorX-150, as well as the hardware components and software used throughout the thesis.

2.1 ReactorX-150

The ReactorX-150 from Trossen Robotics is a robotic arm that is part of the X-Series family. The X-series utilizes the DYNAMIXEL X-Series Smart Servo Motors. These actuators are designed to deliver high performance, including high torque and efficient heat dissipation, all in a compact form factor that surpasses the previous DYNAMIXEL servo models. The ReactorX-150 offers five degrees of freedom and a full 360-degree range of rotation, providing exceptional versatility and capabilities for various applications.

The ReactorX-150 is equipped with two highly sophisticated servos, the DYNAMIXEL XM430-W350 and DYNAMIXEL XL430-W250, which offer a resolution of 4096 positions and the ability for users to define their own PID parameters. Additionally, the servos are equipped with temperature monitoring, positional feedback, and the capability to monitor and adjust voltage levels, load, and compliance settings.

At the heart of the ReactorX-150 lies the Robotis DYNAMIXEL U2D2, which serves as an interface between the servos and the DYNAMIXEL Wizard software and ROS, allowing for easy access and integration into robotic systems.

A quick overview of the capabilities and accuracy of the ReactorX-150 can be seen in Table 2-1.

Table 2-1: ReactorX-150 capabilities quick overview [2]

ReactorX-150	
Degrees of Freedom	5
Reach	450 mm
Total span	900 mm
Repeatability	2.5 mm
Accuracy	5 - 8 mm
Working payload	100 g*
Total servos	6
Rotating wrist	yes

Weight of the arm	4 lb. (approx. 1.8 kg)

2.1.1 Arm Reach and Joint Names

The reach of the ReactorX-150 is a critical factor in its design and capabilities. The reach of a robotic arm is an essential specification to consider when selecting or designing a robot for specific tasks. A longer reach may be required for tasks that require the arm to reach further distances or for larger workspaces, while a shorter reach may be more appropriate for tasks that require more precision or in a smaller workspace.

The reach of the ReactorX-150 can be divided into sections, which can be seen in Figure 2.1.There are two categories of data that can be relevant from Figure 2.1. One being the total length from point A to the remaining points, and the distance between each pair of points.

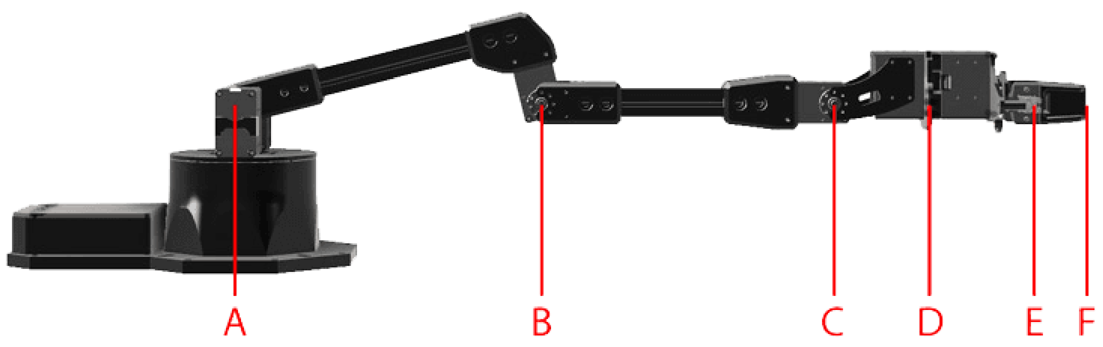


Figure 2.1: Arm reach with alphabetic indicators [3]

The joints of the ReactorX-150 has been given names for simpler explanation. The names of the joints can also be used when writing python scripts. The first joint is depicted in Figure 2.1, but not denoted by a letter. The first joint is named the “waist” and is located within the base of the ReactorX-150. The second joint is depicted as point A, which is named the “shoulder” joint. The third joint is the “elbow” joint, which is depicted as point B. The fourth joint is the “wrist tilt” joint, which is depicted as point C. The fourth joint is the “wrist rotate” joint which is depicted as point D. The final joint is the “gripper” joint, which can be depicted as point E.

The length from point A to each of the remaining points can be seen in Table 2-2.

Table 2-2: ReactorX-150 total reach points [3]

Robot Arm	B	C	D	E	F
	Elbow	Wrist tilt	Wrist rotate	Gripper rail	Fingertip
ReactorX-150	158 mm*	308 mm	373 mm	439 mm	482 mm

The length between each of the neighboring points can be seen in Table 2-3.

Table 2-3: ReactorX-150 total reach points [3]

Robot arm	A - B	B - C	C - D	D - E	E - F
	Upper arm	Forearm	Wrist tilt to wrist rotate	Gripper (to rail)	Fingertip
ReactorX-150	158 mm	150 mm	65 mm	66 mm	43 mm

2.1.2 Workspace and Working Payload

The workspace of a robotic arm refers to the range of motion within which it can operate. The ReactorX-150, like other robotic arms, has a specific workspace that is determined by its design, reach, and other factors. The recommended working space of the ReactorX-150 is 70% of its reach, which provides ample room for the arm to move and perform various tasks.

The span and the recommended workspace of the ReactorX-150 can be seen in Table 2-4, and is visualized in Figure 2.2.

Table 2-4: ReactorX-150 recommended workspace and total span [3]

Robot Arm	Recommended workspace	Total span
ReactorX-150	630 mm	900 mm

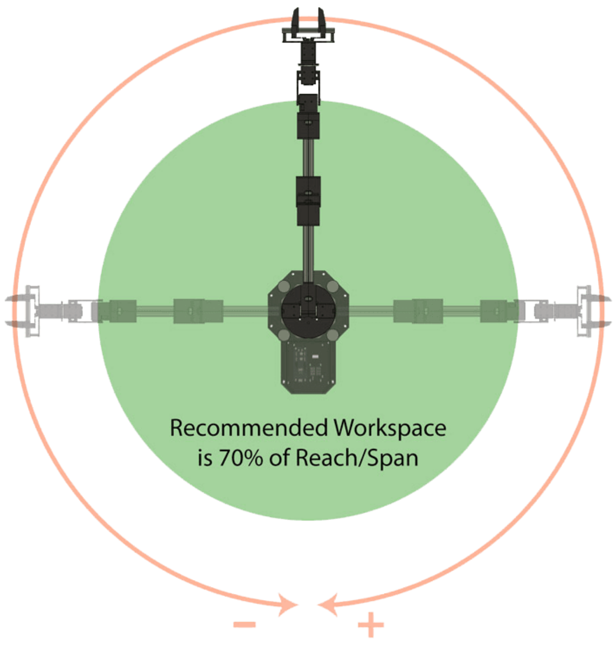


Figure 2.2: Visualization of ReactorX-150 span and recommended workspace [3]

The working payload of the ReactorX-150 refers to the recommended maximum weight that the arm can lift and manipulate during operation. The working payload is an important specification to consider when selecting or designing a robot for specific tasks since the weight of the payload can impact the accuracy, speed, and safety of the robot's movements. It is crucial to ensure that the arm's working payload is sufficient to handle the intended tasks to avoid overloading, overheating, or damaging the robot, and to maintain productivity and safety in the work environment.

The working payload for the ReactorX-150 can be seen in Table 2-5.

Table 2-5: ReactorX-150 working payload

Robot arm	Working payload
ReactorX-150	100 g

There is stated by the manufacturer that the work payload should not be exceeded during any operation. The working payload for the arm is the maximum recommended weight for periods of repeated movement inside the recommended workspace. If intending to manipulate a 100g object, the recommendation is to not extend the arm more than 50% of its reach. “Rest” poses should be incorporated when intending to operate over longer periods of time to prevent the servos from overheating. [2] [3]

2.1.3 Linkage Dimensions and Gripper Limitations

The linkage dimensions of the ReactorX-150 refer to the physical dimensions of the arm's links or segments, which determine the arm's range of motion and workspace. The ReactorX-150's given linkage dimensions consist of five links, including the upper arm, elbow offset, true upper arm length, forearm, and offset angle, denoted as letter A to E.

The linkage dimensions are specified in Table 2-6 and visualized in Figure 2.3.

Table 2-6: ReactorX-150 linkage dimensions

Robot arm	A	B	C	D	E
	Upper arm	Elbow offset	True upper arm length	Forearm	Offset angle
ReactorX-150	150 mm	50 mm	158 mm	150 mm	18.4°

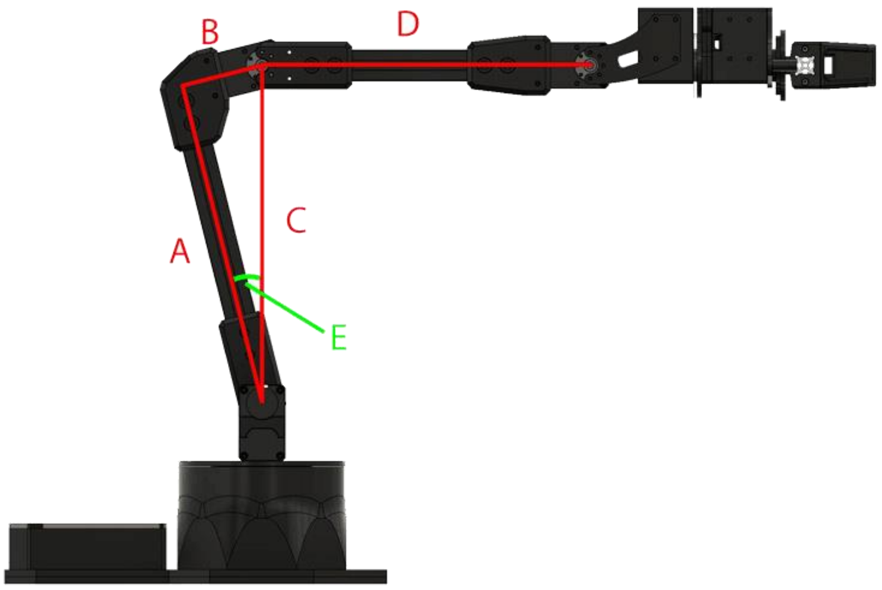


Figure 2.3: ReactorX-150 Linkage dimensions visualized [3]

The length of each link determines the range of motion of the arm, which affects its ability to reach specific points and perform certain tasks. The ReactorX-150's linkage dimensions provide a balance between flexibility and precision, enabling the arm to perform a wide range of tasks with accuracy and speed.

The gripper of the ReactorX-150 Robot Arm is an essential tool for manipulating objects during operation. The gripper's minimum and maximum opening dimensions determine the size and shape of objects that the arm can grasp and manipulate. The ReactorX-150 is compatible with various gripper options, which can have different opening dimensions. The minimum and maximum gripper dimensions of the ReactorX-150 will depend on the specific gripper option used with the arm.

The gripper minimum and maximums are specified in Table 2-7. Due to the almost unlimited gripper variations possible, the minimum and maximum gripper opening are given as distance from center of the gripper to the center of the gripper carriages from the manufacturer. There have also been taken measurements within the gripper configuration used in this thesis, with foam pads for additional grip. A visualization of the measurement points on the gripper and gripper carriages with their respective colors from Table 2-7 can be seen in Figure 2.4.

Table 2-7: ReactorX-150 minimum and maximum gripper widths [3]

ReactorX-150	Minimum	Maximum	Color
Gripper carriages	30 mm	74 mm	🟥
Grippers with foam pads	5 mm	49 mm	🟩

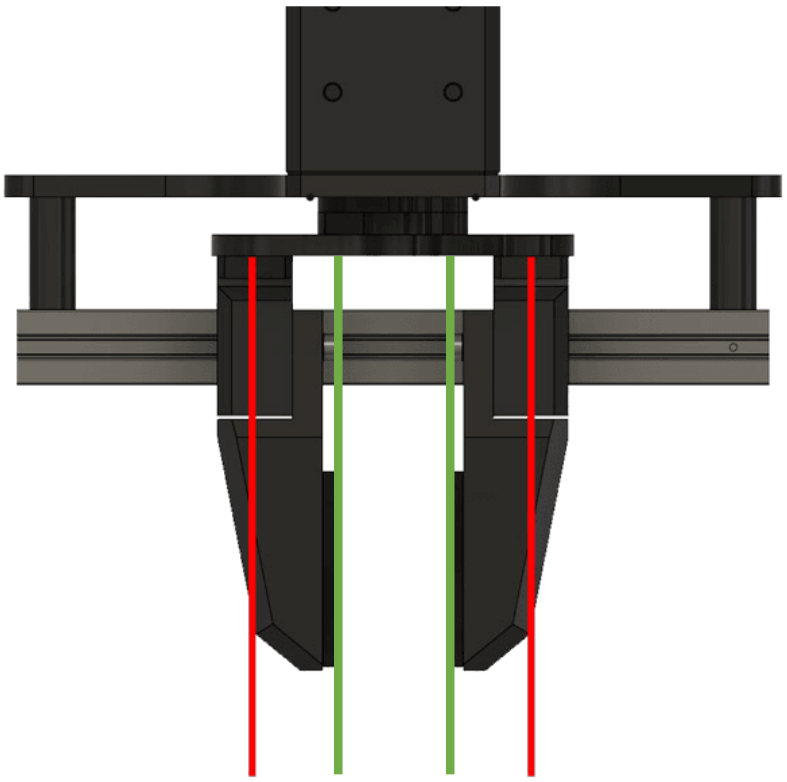


Figure 2.4: Visualization of ReactorX-150 gripper measurement points [3]

2.1.4 Default Joint Limits and Default Servo Configurations

The joint limits of the ReactorX-150 refer to the maximum and minimum angles that each joint is allowed to rotate. The default joint limits are preset in the arm's software (see chapter 2.3.7 DYNAMIXEL Wizard 2.0) and can be adjusted if needed. The ReactorX-150 has five joints, and each joint has different default limits depending on its design and position in the arm. The joint limits are crucial specifications to consider when programming the robot for specific tasks since exceeding the joint limits can cause damage to the arm or decrease its accuracy and speed.

The default joint limits of the ReactorX-150 can be seen in Table 2-8, and the limits are stated as degrees from servo center (zero degrees). The only exception being the “Gripper” joint, which is given as mm from gripper center to minimum and maximum of center gripper carriages.

Table 2-8: ReactorX-150 default joint limits

Joint	Minimum	Maximum	Servo ID
Waist	-180	180	1
Shoulder	-106	100	2
Elbow	-102	95	3

Wrist angle	-100	123	4
Wrist rotate	-180	180	5
Gripper	30 mm	74 mm	6

The default servo configurations of the ReactorX-150 Robot Arm refer to the preset settings for the servos that control the arm's movement. The servo configurations can be customized if needed to optimize the robot's performance for specific tasks, but it has been kept on the default setting for this thesis. The ReactorX-150 uses the DYNAMIXEL X-Series Smart Servo Motors, the two types of servos used in the RX150, the DYNAMIXEL XM430-W350, and the DYNAMIXEL XL430-W250.

The default servo configurations can be seen in Table 2-9. It shall be noted that the joint names used here are the names of the joints to be used when controlling single servos with commands or python scripts.

Table 2-9: ReactorX-150 default servo configurations

Joint name	Servo ID	Servo	Baud rate
waist	1	XM430-W350	1 Mbps
shoulder	2	XM430-W350	1 Mbps
elbow	3	XM430-W350	1 Mbps
wrist_angle	4	XL430-W250	1 Mbps
wrist_rotate	5	XL430-W250	1 Mbps
gripper	6	XL430-W250	1 Mbps

2.1.5 Product of Exponentials Kinematic Properties

The product of exponentials (POE) is an alternative to the DH parameterization. Although not utilized in this thesis, and will not be elaborated on, it is available from the manufacturer as seen below in formula 2.1 and 2.2.

$$M = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.258575 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.25457 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{2.1}$$

$$Slist = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & -0.10457 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & -0.25457 & 0.0 & 0.05 \\ 0.0 & 1.0 & 0.0 & -0.25457 & 0.0 & 0.2 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.25457 & 0.0 \end{bmatrix}^T \tag{2.2}$$

For more information about the method of POE, see [4].

2.1.6 Technical Drawing and CAD files

The technical drawing of the ReactorX-150 Robot Arm is a detailed and precise representation of the arm's physical dimensions, features, and components. The drawing includes information such as linkage dimensions, and the position of various components such as the servos and end effector. The technical drawing is a helpful reference tool for engineers, designers, and technicians working with the RX150 since it provides an overview of the arm's structure and specifications. The technical drawing of the ReactorX-150 can be seen in Figure 2.5.

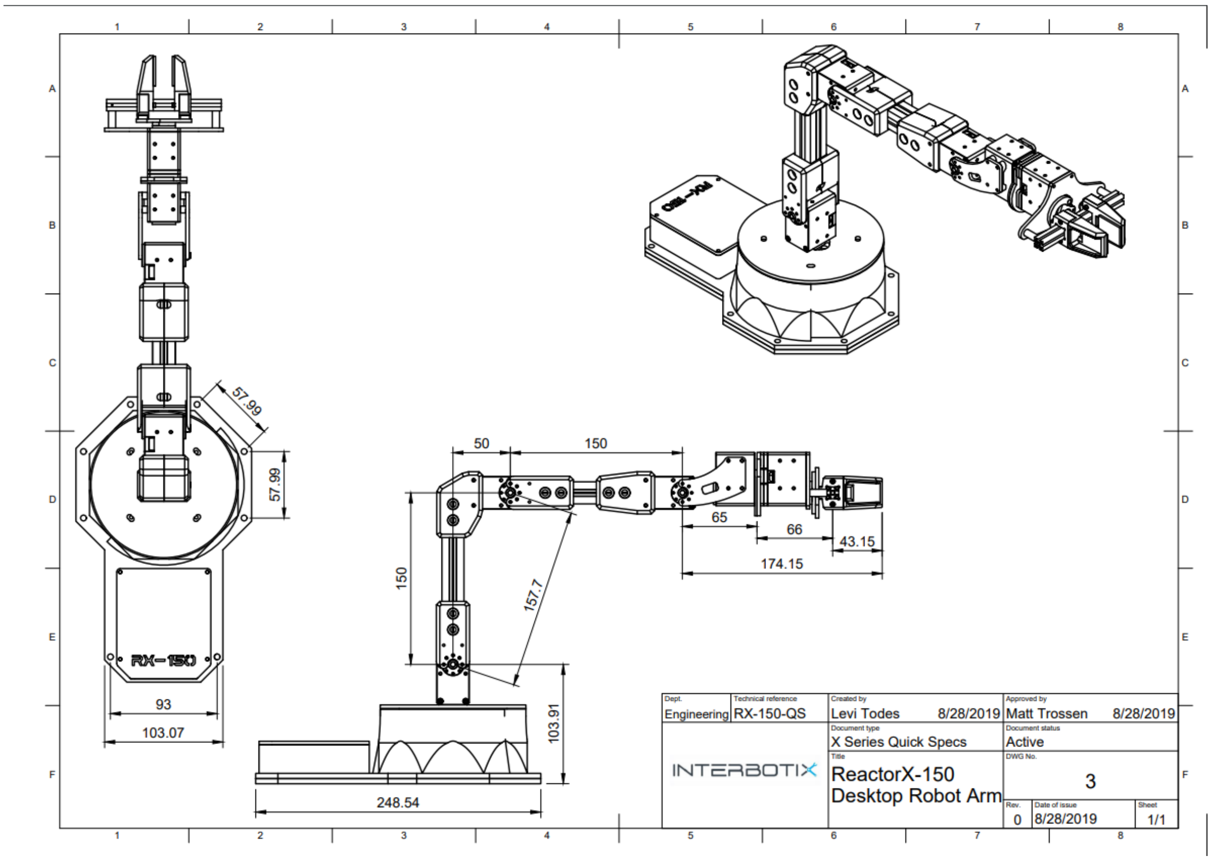


Figure 2.5: ReactorX-150 technical drawing [2]

The CAD files for the ReactorX-150 consists of:

- STEP files, available for download at [5].
- Mesh files in STL format, available at [6].

2.1.7 X-Series code names

When working with X-series arms from Trossen Robotics, users refer to the arms by their code names instead of their full names. The code names are shorter and more convenient to use, particularly when programming. Each X-series arm has a unique code name, the X-series arms have code names that are abbreviations of their full names. Even though this thesis revolves around the ReactorX-150, it can be useful for testing, research, and future development purposes to know the codenames of all the X-series arms. The model names and their corresponding codenames can be seen in Table 2-10.

Table 2-10: X-Series robotic arms model names and codenames [7]

Robot arm model name	Codename
PincherX-100 Robot Arm	px100
PincherX-150 Robot Arm	px150
ReactorX-150 Robot Arm	rx150
ReactorX-200 Robot Arm	rx200
WidowX-200 Robot Arm	wx200
WidowX-250 Robot Arm	wx250
WidowX-250 Robot 6DOF Arm	wx250s
ViperX-250 Robot Arm	vx250
ViperX-300 Robot Arm	vx300
ViperX-300 Robot 6DOF Arm	vx300s

2.2 Hardware

The hardware chapter of this thesis provides an overview of the ReactorX-150 robot arm's hardware components and external hardware components. The chapter discusses the specifications of each component, as well as their functionalities and compatibility with the ReactorX-150 system. This information is useful for understanding the capabilities of the ReactorX-150 and to operate it safely and effectively.

2.2.1 ReactorX-150 Actuators

The ReactorX-150 Robot Arm uses DYNAMIXEL X-Series Smart Servo Motors as actuators to control the arm's movement. These actuators offer high torque, efficient heat dissipation, and great durability, all at a smaller form factor than previous DYNAMIXEL servos. The DYNAMIXEL XM430-W350 and DYNAMIXEL XL430-W250 servos provide high resolution of 4096 positions, allowing for precise control of the arm's movement. Additionally, these servos offer user-definable PID parameters, temperature monitoring, positional feedback, as well as voltage levels, load, and compliance settings that are all user-accessible.

An overview/comparison of the two actuators can be seen in Table 2-11, and the location of the different actuators is covered in Table 2-9.

Table 2-11: ReactorX-150 actuators [8] [9]

ReactorX-150 actuators		
Specifications	DYNAMIXEL XL430-W250-T	DYNAMIXEL XM430-W350-T
Image		
Baud rate	9600 bps ~ 4.5 Mbps	9600 bps ~ 4.5 Mbps
Weight	57.2 g	82 g
Dimensions (W x H x D)	28.5 mm x 46.5 mm x 34 mm	28.5 mm x 46.5 mm x 34 mm
Resolution	4096 pulse/revolution	4096 pulse/revolution

Operating temperature	-5 ~ +72 °C	-5 ~ +80 °C
Gear ratio	258.5 : 1:00	353.5 : 1:00
Stall torque	1.0 Nm at 9.0 V, 1.0 A 1.4 Nm at 11.1 V, 1.3 A 1.5 Nm at 12.0 V, 1.4 A	3.8 Nm at 11.1 V, 2.1 A 4.1 Nm at 12.0 V, 2.3 A 4.8 Nm at 14.8 V, 2.7 A
No load speed	47 rev/min at 9.0 V 57 rev/min at 11.1 V 61 rev/min at 12.0 V	43 rev/min at 11.1 V 46 rev/min at 12.0 V 57 rev/min at 14.8 V
Input voltage	6.5 ~ 12.0 V Recommended: 11.1 V	10.0 ~ 14.8 V Recommended: 12.0 V
Feedback	Position, Velocity, Load, Realtime tick, Trajectory, Temperature, Input Voltage, etc.	Position, Velocity, Current, Realtime tick, Trajectory, Temperature, Input Voltage, etc.

2.2.2 ReactorX-150 Controller

The Robotis DYNAMIXEL U2D2 is a communication converter that is at the heart of the ReactorX-150. It enables easy access to DYNAMIXEL Wizard software as well as ROS, providing a comprehensive set of tools to control and program the robot arm. The U2D2 facilitates communication between the control computer and the DYNAMIXEL actuators. The layout of the U2D2 can be seen in Figure 2.6. and an overview of the technical specifications of the U2D2 can be seen in Table 2-12. The communication used by the ReactorX-150 is the 3 Pin TTL Level communication.

Table 2-12: Robotis DYNAMIXEL U2D2 specifications [10] [11]

Specifications	DYNAMIXEL U2D2
Dimensions (W x H x D)	48 mm x 18 mm x 14.6 mm
Weight	9 g
Available Ports	• 3 Pin TTL Level • 4 Pin RS-485 • 4 Pin UART

Baud rate	9600 bps ~ 6 Mbps

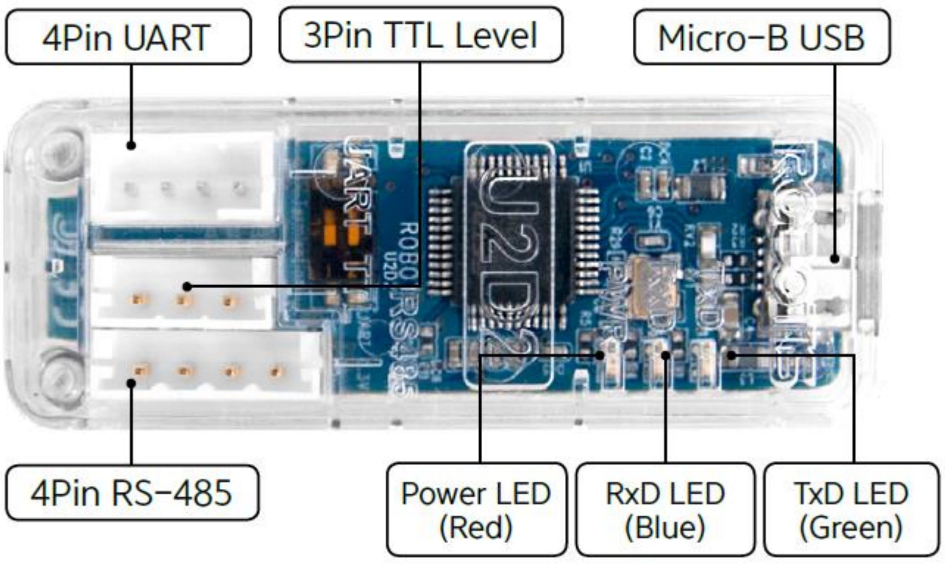


Figure 2.6: Robotis DYNAMIXEL U2D2 layout [10]

2.2.3 ReactorX-150 Power hub

The Robotis 6 Port XM/XL Power Hub is the power hub of the ReactorX-150. The power hub houses six ports for connecting 3 Pin DYNAMIXEL X-Series cables. In the case of the ReactorX-150, 3 Pin DYNAMIXEL X-Series cables connect from the U2D2 to the power hub to a DYNAMIXEL servo, which is then daisy chained to the remaining DYNAMIXEL servos. The power hub is compatible with all DYNAMIXEL XM and XL servos. A figure of the power hub can be seen in Figure 2.7.

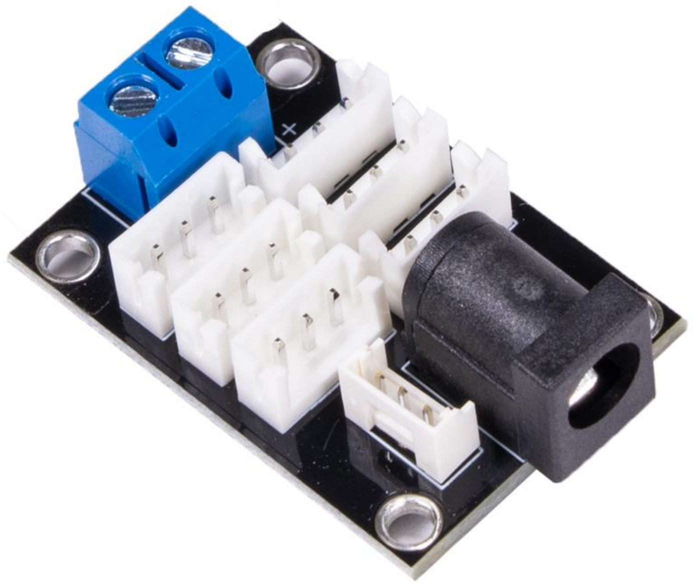


Figure 2.7: DYNAMIXEL 6 Port XM/XL Power Hub [12]

2.2.4 ReactorX-150 Power supply

The power supply for the ReactorX-150 is a 12 V DC, 5 Amp power supply. It is connected into the power hub with a 5.5x2.1 mm barrel jack connector with center positive polarity. [13]

2.2.5 3D Printed Custom End Effector

The customizability of the ReactorX-150 extends to its end effector (fingers) and gripper carriages. A picture highlighting the gripper carriages and fingers can be seen in Figure 2.8.

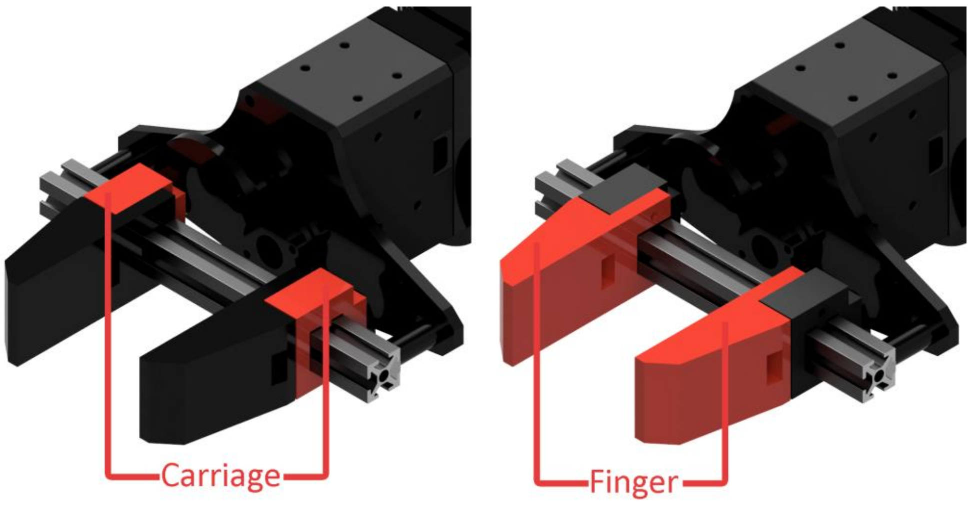


Figure 2.8: ReactorX-150 gripper carriages and fingers [14]

A technical drawing of the gripper carriages can be seen in Figure 2.9.

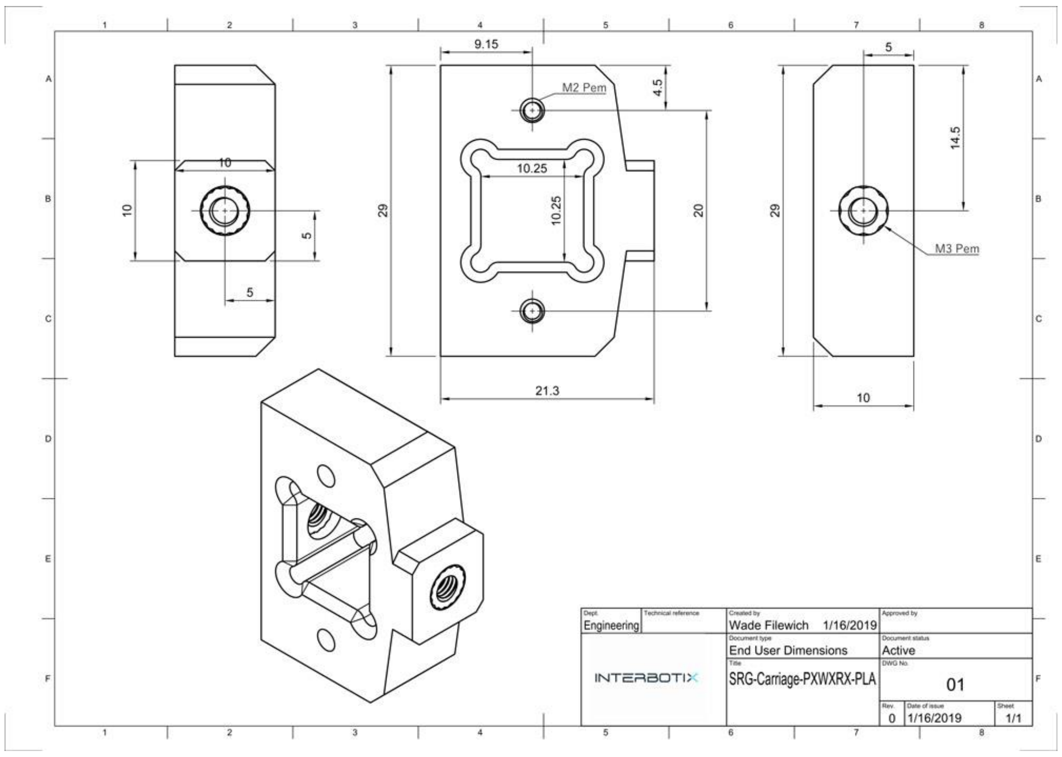


Figure 2.9: ReactorX-150 gripper carriages technical drawing [14]

The screws used to mount the fingers to the grippers are M2 bolts, and the bolts used to attach the gripper carriages to the motor arms are M3 bolts. The listed specifications about the gripper carriages allows users to tailor the end effector and gripper carriages to suit their specific use case.

2.2.6 AprilTag

AprilTag markers are a type of visual marker that are widely used in robotics for object detection and pose estimation. These tags consist ideally of a black and white pattern that is designed to be easily recognizable by computer vision systems. The AprilTag marker is supposed to be used to locate the ReactorX-150 in its environment. This can be done manually, but it is time consuming and prone to error. [15] [16]

The AprilTag marker was supposed to be utilized in the perception part of the ReactorX-150 capabilities. However due to the lack of a depth camera, the marker was not utilized. The camera intended for the perception part was the Intel RealSense Depth Camera D415. [17]

The default AprilTag family utilized by the Perception package, covered in chapter 4.6, is the “tagStandard41h12” family.

2.2.7 Raspberry Pi

A Raspberry Pi 3 Model B and a Raspberry Pi 4 Model B were attempted to use for the perception part for the ReactorX-150. The Raspberry Pi 3 Model B and the Raspberry Pi 4 Model B are credit card-sized single-board computers developed by the Raspberry Pi Foundation.

The specifications of the Raspberry Pi 3 Model B and the Raspberry Pi 4 can be seen in Table 2-13, and a figure of the Raspberry Pi 3 Model B and the Raspberry Pi 4 Model B can be seen in Figure 2.10.

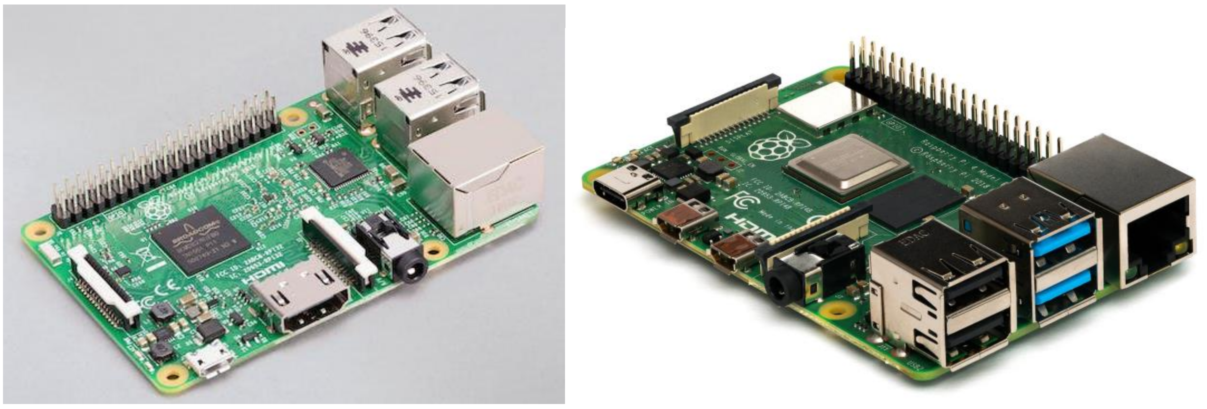


Figure 2.10: Raspberry Pi 3 Model B and Raspberry Pi 4 Model B [18] [19]

Table 2-13: Raspberry Pi 3 Model B and Raspberry Pi 4 Model B specifications [18]

Raspberry Pi specifications	
Raspberry Pi 3 Model B	Raspberry Pi 4 Model B

Quad Core 1.2GHz Broadcom BCM2837 64bit CPU	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz Broadcom BCM2711
1GB RAM	4GB RAM
BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
100 Base Ethernet	Gigabit Ethernet
40-pin extended GPIO	Raspberry Pi standard 40 pin GPIO header
4 USB 2 ports	2 USB 3.0 ports and 2 USB 2.0 ports
4 Pole stereo output and composite video port	4-pole stereo audio and composite video port
Full size HDMI	2 × micro-HDMI ports (up to 4kp60 supported)
CSI camera port for connecting a Raspberry Pi camera	2-lane MIPI CSI camera port
DSI display port for connecting a Raspberry Pi touchscreen display	2-lane MIPI DSI display port
Micro SD port for loading the operating system and storing data	Micro-SD card slot for loading operating system and data storage
Upgraded switched Micro USB power source up to 2.5A	5V DC via USB-C connector 5V DC via GPIO header
	Power over Ethernet (PoE) enabled (requires separate PoE HAT)
	H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode) OpenGL ES 3.1, Vulkan 1.0

2.2.8 Raspberry Pi Camera Module 2

The Raspberry Pi Camera Module 2 connects to a Raspberry pi using a 15 cm ribbon cable connected to the CSI port on the Raspberry Pi. The Raspberry Pi Camera Module 2 has a Sony IMX219 8-megapixel sensor, it supports 1080p30, 720p60 and VGA90 video modes and still capture. The camera is compatible with all models of Raspberry Pi 1, 2, 3 and 4. [20]

2.2.9 Lumens DC125

The Lumens DC125 is a high-definition document camera, known for its portability and excellent image capabilities. The camera connects to a computer with a USB 2.0 cable. The Lumens DC125 has a 1/3" 3M CMOS Color Image Sensor, some of the output resolutions it supports are supports 720p30, 1080p30 and QXGA(2048x1536) for both video and still capture. The gooseneck design allows for 360 degrees of rotation. [21]

2.2.10Storage devices

A USB flash drive was utilized for the setup of the Ubuntu operating system. The Ubuntu operating system is covered in chapter 2.3.1 and the actual setup of the Ubuntu operating system is covered in [Appendix B – Guide for Dual Booting Windows and Ubuntu]. The USB flash drive utilized for the setup was:

- SanDisk Ultra 128GB Dual Drive Go

A micro-SD card with a SD card adapter was used for the use and setup of Raspberry Pi. The Raspberry Pi is covered in chapter 2.2.7 and the Raspberry Pi setup is covered in [Appendix I – Raspberry Pi Ubuntu and ROS Setup]. The micro-SD card utilized for the setup was:

- PNY Elite Micro SDXC 64 GB

2.2.11Other

Other hardware used for this thesis, but where specifications are less important:

- PC: Acer Predator Helios 300
- Monitor with HDMI capability: For use and setup of the Raspberry Pi
- USB keyboard: For use and setup of the Raspberry Pi
- USB mouse: For use and setup of the Raspberry Pi
- HDMI cable: For use and setup of the Raspberry Pi
- Ethernet cable: For use and setup of the Raspberry Pi
- 3D printer: FlashForge Adventurer 3 [22]
- Camera for filming: iPhone 12 Pro Max

2.3 Software

The software chapter of this thesis provides an overview of the software used, and how or where some the software was used. This chapter also covers the decisions made regarding software used in the thesis.

2.3.1 Ubuntu

Ubuntu is a Linux distribution based on the Debian architecture. Debian is designed to be reliable and secure, while Ubuntu is designed to be user friendly. The combination results in a flexible, reliable, secure, and user-friendly Linux distribution. The Ubuntu logo can be seen in Figure 2.11.

Figure 2.11: Ubuntu logo

The supported Ubuntu versions for the X-series arms were Ubuntu 18.04, Ubuntu 20.04, and Ubuntu 22.04. The Ubuntu version utilized in this thesis was the Ubuntu 20.04. The reason for utilizing Ubuntu 20.04 is covered in chapter 2.3.2.

The installation guide for dual booting with Ubuntu can be seen in [Appendix B – Guide for Dual Booting Windows and Ubuntu].

The installation guide for Ubuntu with MATE desktop on a Raspberry Pi can be seen in [Appendix I – Raspberry Pi Ubuntu and ROS Setup Guide].

2.3.2 ROS

ROS (Robot Operating System) is an open-source framework for building robot software. It provides a set of libraries and tools to help developers create complex robot applications, including drivers, algorithms, and communication protocols. ROS was developed by Willow Garage, a robotics research lab, and is now maintained by the Open Robotics organization. The ROS logo can be seen in Figure 2.12.

Figure 2.12: ROS logo [23]

Packages for ROS and the newer version ROS2 are supported for the ReactorX-150, as well as for the whole X-Series lineup of robotic arms.

The available supported ROS distributions on different Ubuntu versions for the X-Series arm are listed below:

- ROS 1 Melodic, Ubuntu Linux 18.04

- ROS 1 Noetic, Ubuntu Linux 20.04
- ROS 2 Galactic, Ubuntu Linux 20.04
- ROS 2 Humble and Rolling on Ubuntu Linux 22.04

ROS 1 was the preferred ROS version requested by the supervisor. This narrowed down the choice of ROS versions to ROS 1 Melodic and ROS 1 Noetic. The choice was made to utilize the latest supported ROS 1 distribution. The ROS distribution used in this thesis is ROS 1 Noetic, which is compatible with Ubuntu 20.04. The logo for ROS 1 Noetic can be seen in Figure 2.13.

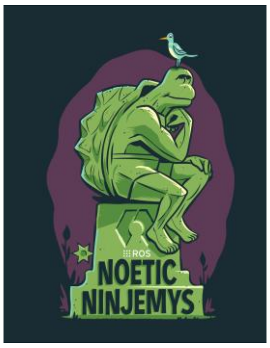


Figure 2.13: ROS 1 Noetic logo [24]

The installation guide to ROS 1 Noetic can be seen in [Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics]. To test if ROS has been correctly installed and is working with the robotic arm, see the quickstart guide in [Appendix D – Quickstart Guide for the X-Series Arms from Trossen Robotics].The Appendix includes some packages such as the Description and Control packages which are covered in chapter 4.1 and 4.2.

2.3.3 IRROS

IRROS (Interbotix Research Robotics Open Standard) is a framework developed by Trossen Robotics to provide a common hardware and software platform for their line of research-grade robots, such as the X-Series arms. The framework is designed to simplify the development and integration of custom hardware and software components for Interbotix robots. IRROS is based on ROS and utilizes many of its features and tools, such as the ROS middleware, message passing system, and visualization tools. The overview of IRROS can be seen in Figure 2.14. The repository utilized by the robotic arms, and in this thesis is the “interbotix_ros_manipulators”.

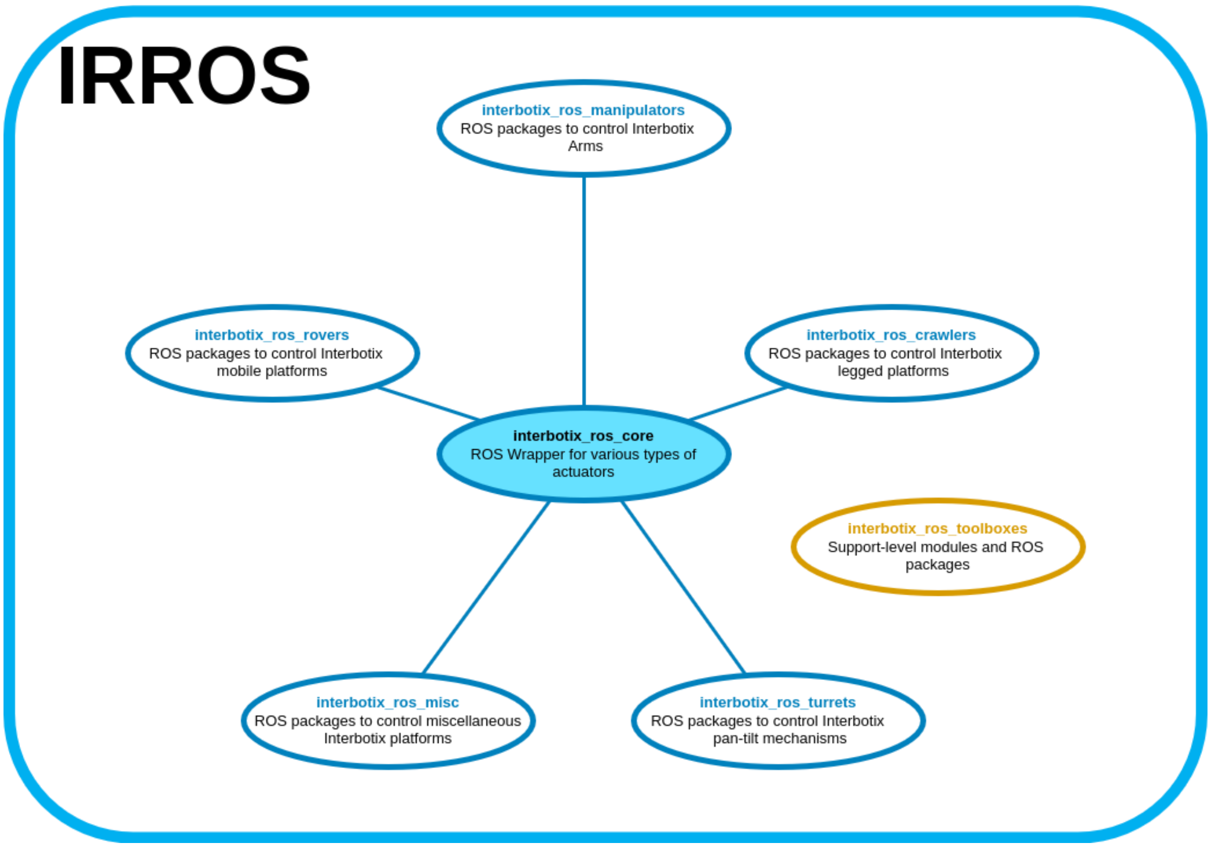

IRROS
interbotix_ros_manipulators
ROS packages to control Interbotix Arms
interbotix_ros_rovers
ROS packages to control Interbotix mobile platforms
interbotix_ros_crawlers
ROS packages to control Interbotix legged platforms
interbotix_ros_core
ROS Wrapper for various types of actuators
interbotix_ros_toolboxes
Support-level modules and ROS packages
interbotix_ros_misc
ROS packages to control miscellaneous Interbotix platforms
interbotix_ros_turrets
ROS packages to control Interbotix pan-tilt mechanisms

Figure 2.14: IRROS overview [25]

2.3.4 MoveIt

MoveIt is an open-source software framework for motion planning and manipulation in robotics. It provides a set of tools, libraries, and APIs for creating and executing motion plans for robotic systems, such as the Trossen Robotics X-Series arms. The MoveIt logo can be seen in Figure 2.15.

Figure 2.15: MoveIt logo [23]

In the context of the X-Series arms, MoveIt can be used to plan and execute complex motion trajectories for the robot arms, including path planning, obstacle avoidance, collision checking and 3D perception.

2.3.5 Gazebo

Gazebo is an open-source, 3D robotics simulator that allows users to simulate and test robotic systems in a virtual environment. It provides a physics engine that can accurately model the dynamics and behavior of a wide range of robotic systems, including robots, drones, and vehicles. The Gazebo logo can be seen in Figure 2.16.

Figure 2.16: Gazebo logo [23]

Gazebo was used in this thesis for testing of the robotic arms, specifically the ReactorX-150. Gazebo is a powerful tool for testing with the ReactorX-150 and the Trossen Robotics X-Series arms, increasing safety, flexibility, reproducibility, and cost-effectiveness.

2.3.6 RViz

ROS Visualization (RViz) is a 3D visualization tool that is used in the field of robotics to provide a graphical representation of data from robots and robotic systems. It is a powerful tool that enables users to visualize and debug robots and robotic systems in a 3D environment, facilitating the analysis of robot perception systems, motion planning, and control algorithms. The RViz logo can be seen in Figure 2.17.

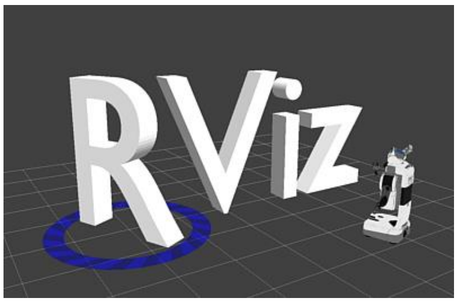


Figure 2.17: RViz logo

RViz was used in thesis as a tool for displaying and/or controlling a virtual model of the simulated robot arm(s) in Gazebo or for visualizing the real robot arm(s) in a virtual environment.

2.3.7 DYNAMIXEL Wizard 2.0

DYNAMIXEL Wizard 2.0 is a software tool developed by Robotis for configuring and managing DYNAMIXEL servos. The software tool supports a variety of communication protocols, including USB, RS-232, and TTL. This allows connection and configuration of DYNAMIXEL servos using different communication methods. It also provides a real-time display of servo status and feedback, enabling users to monitor and debug servo systems in real-time. The DYNAMIXEL Wizard 2.0 logo can be seen in Figure 2.18.

Figure 2.18: DYNAMIXEL Wizard 2.0 logo [26]

The DYNAMIXEL Wizard 2.0 was primarily used in this thesis to deal with troubleshooting of the DYNAMIXEL servos and correcting any offset in the servos. The full guide for installation, uninstallation, basic features, and advanced features can be found at [27].

2.3.8 BalenaEtcher

BalenaEtcher is a cross-platform application that simplifies the process of creating bootable USB drives or SD cards from ISO and IMG files. It is compatible with Windows, macOS, and Linux operating systems. The BalenaEtcher logo can be seen in Figure 2.19.

Figure 2.19: BalenaEtcher logo [28]

BalenaEtcher was used in this thesis to create a bootable USB drive for the installation of Ubuntu and for creating a bootable SD card for the Raspberry Pi.

2.3.9 Visual Studio Code

Visual Studio Code (VS Code) is a free open-source code editor developed by Microsoft. VS Code includes a modular and extensible architecture, with the ability to add custom functionality and integrate with other tools and services. The Visual Studio Code logo can be seen in Figure 2.20.

Figure 2.20: Visual Studio Code logo [29]

VS Code was primarily used in this thesis for creating and editing executable scripts for the ReactorX-150 arms and editing configuration files.

2.3.10 Main Programming Languages

The main programming languages used for everything from config and launch files, to executable scripts are listed below:

- Python
- C++
- MATLAB

2.3.11 Sharpr3D

Sharpr3D is a 3D modelling software tool designed with intuition at its core. Sharpr3D is designed to be compatible with Windows PCs and tablets, macOS and iPadOS, making Sharpr3D versatile and accessible. The Sharpr3D logo can be seen in Figure 2.21.

Shapr3D

Figure 2.21: Sharpr3D logo [30]

Sharpr3D was used in this thesis for modelling objects to be manipulated by the ReactorX-150 arms, see chapter 7.6. Although Sharpr3D was also intended to model a stand for a camera to be used in the perception part of the ReactorX-150, it was not utilized for this purpose.

2.3.12 FlashPrint 5

FlashPrint 5 is a slicing and printing software used for 3D printing with FlashForge printers, in this case the FlashForge Adventurer 3. [22] FlashPrint 5 enables preparation and optimization of 3D models for printing by slicing the models into layers and generating machine-readable code that controls the printer's movements. The logo for FlashPrint can be seen in Figure 2.22.

Figure 2.22: FlashPrint logo [31]

FlashPrint 5 was used in this thesis as a slicing and printing software for the models created in Sharpr3D, see chapter 7.6.

2.3.13 Windows and Microsoft Office

Ubuntu was utilized for the work with the robotic arms, but Windows operating system was also utilized due to the lack of compatibility between Ubuntu and Microsoft Office. Microsoft Office applications can be used on Linux systems via a web browser, a virtual machine running Windows or installed via third-party installers. However, none of these solutions were found optimal for this thesis and therefore was not utilized. Since the computer used was dual booted with Windows and Ubuntu, the Microsoft Office applications on Windows was utilized. The Windows version used was Windows 10 Home.

2 Specifications

Microsoft Office is a collection of applications developed by the Microsoft Corporation. Microsoft office covers an array of applications, but the applications used in this thesis are listed below:

- Microsoft Word: Used for report writing.
- Microsoft Project: Creation of gantt chart
- Microsoft Visio: Creation of visualizations and small-scale photo editing
- Microsoft PowerPoint: Creation of videos and video editing

3 Denavit–Hartenberg Model

The Denavit-Hartenberg (DH) representation is a way of representing almost all robotic arm manipulators. The DH representation is widely used in the industry to configure industrial robotic arms and makes it possible to calculate forward kinematics for complex robot arms with the help two displacement parameters for displacement in the direction of the x- and z-axis, and two rotation parameters. For this application the rotation is defined such that clockwise rotation is negative.

3.1 Kinematic Diagram

A DH representation in the form of a kinematic diagram can be seen in Figure 3.1. The kinematics was made following "The three rules for coordinate frames" and "The steps for DH representation" seen below the kinematics.

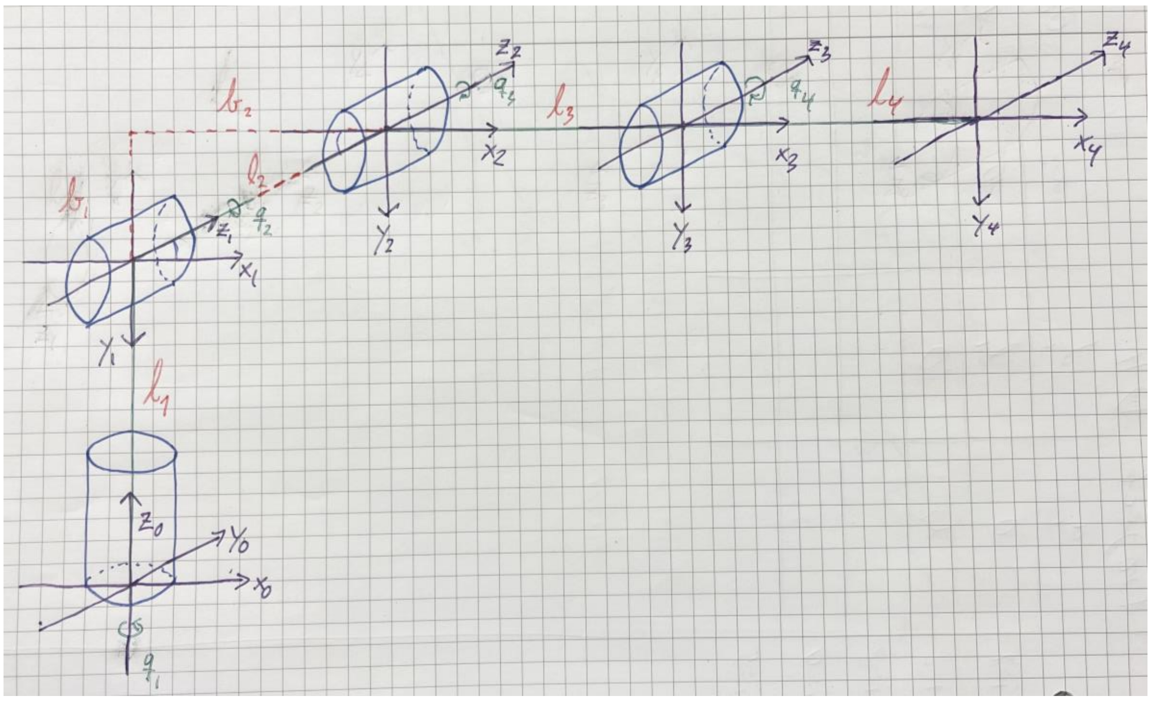


Figure 3.1: DH representation with kinematic diagram of the ReactorX-150 [32]

The three rules for coordinate frames: [33] [32]

1. The z-axis was chosen in the direction of the joint axis.
2. The y-axis followed the right-hand rule, where the thumb is in the direction of the z-axis, the index finger in the direction of the x-axis and the middle finger in the direction of the y-axis.
3. The x_i axis must interest the z_{i-1} axis.

The "Steps for DH representation" step 1 to 5 from [33] was followed.

3.2 Denavit-Hartenberg parameters

Utilizing the specifications of the arm from chapter 2 and the kinematic diagram in Figure 3.1, a DH table can be made using the previously mentioned displacement parameters and rotation parameters: [32]

- a_i is the distance in the x-axis between the joints.
- α_i is the rotation around the x_i axis to get z_{i-1} axis to match z_i.
- θ_i is the rotation around the z_{i-1} axis.
- d_i is the distance in the z-axis.

All the z axis in Figure 3.1 points the same direction, except z_0, so it is rotated -90 degrees revolving around x_1 to match up with z_1 and θ_i is then equal to the q_1 variable since the rotation is around the z_{i-1} axis.

There is a displacement between coordinate frame one and two in the y-axis, which means it does not fit directly into the DH table. z_2 and z_3 are then rotated to compensate for the displacement. The angle seen in (Eq. 1) was then found by utilizing Pythagoras.

$$\sin^{-1}\left(\frac{150}{157.7}\right) = 72.02° \qquad (Eq. 1)$$

Filling in the known parameters in the DH table, seen in Table 3-1.

Table 3-1: DH table

i	a_i	d_i	α_i	θ_i
1	0	103.91	-90°	q_1
2	157.7	0	0°	q_2-72.02°
3	150	0	0°	q_3+72.02°
4	174.15	0	0°	q_4

Next up is to insert the parameters from Table 3-1 into the homogeneous transformation matrix in (Eq. 2).

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (\text{Eq. } 2)$$

The result is four matrices seen in (Eq. 3 – 6).

$$A_1 = \begin{bmatrix} \cos(q_1) & 0 & -\sin(q_1) & 0 \\ \sin(q_1) & 0 & \cos(q1) & 0 \\ 0 & -1 & 0 & 103.91 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (\text{Eq. } 3)$$

$$A_2 = \begin{bmatrix} \cos(q_2 - 72.02) & -\sin(q_2 - 72.02) & 0 & 157.7\cos(q_2 - 72.02) \\ \sin(q_2 - 72.02) & \cos(q_2 - 72.02) & 0 & 157.7\sin(q_2 - 72.02) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 4)}$$

$$A_3 = \begin{bmatrix} \cos(q_3 + 72.02) & -\sin(q_3 + 72.02) & 0 & 150\cos(q_3 + 72.02) \\ \sin(q_3 + 72.02) & \cos(q_3 + 72.02) & 0 & 150\sin(q_3 + 72.02) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 5)}$$

$$A_4 = \begin{bmatrix} \cos(q_4) & -\sin(q_4) & 0 & 174.15\cos(q_4) \\ \sin(q_4) & \cos(q_4) & 0 & 174.15\sin(q_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 6)}$$

Multiplying the A_{1-4} matrices as seen in the H-matrix in (Eq. 8 – 8.4)

$$\text{H} = A_1A_2A_3A_4 \quad (Eq.7)$$

$$\text{H} = \begin{bmatrix} H_{11} & H_{12} & -\sin(q1) & x_e \\ H_{21} & H_{22} & \cos(q1) & y_e \\ -\sin(q2 + q3 + q4) & -\cos(\text{q2 + q3 + q4}) & 0 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (Eq.8)$$

$$H_{11} = 0.5\cos(q2 - q1 + q3 + q4) + 0.5\cos(q1 + q2 + q3 + q4) \quad (Eq.8.1)$$

$$H_{12} = -0.5 * \sin(q2 - q1 + q3 + q4) - 0.5 * \sin(q1 + q2 + q3 + q4) \quad (Eq.8.2)$$

$$H_{21} = 0.5 * sin(q1 + q2 + q3 + q4) - 0.5 * sin(q2 - q1 + q3 + q4) \quad (Eq.8.3)$$

$$H_{22} = 0.5 * \cos(q1 + q2 + q3 + q4) - 0.5 * \cos(q2 - q1 + q3 + q4) \quad (Eq.8.4)$$

The x_e, y_e and z_e define the x, y, and z equations of the end effector.

Table 3-2: Denavit-Hartenberg end-effector position equations [32]

End-effector position equations	
$\boldsymbol{x_e}$	$75.0 * \cos(q1 + q2 + q3) + 87.075 * \cos(q2 - 1.0 * q1 + q3 + q4) + 87.075 * \cos(q1 + q2 + q3 + q4) + 78.85 * \cos(q1 - 1.0 * q2 + 1.257) + 75.0 * \cos(q2 - 1.0 * q1 + q3) + 78.85 * \cos(q1 + q2 - 1.257)$

$\boldsymbol{y_e}$	$78.85 * \sin(q1 + q2 - 1.257) + 75.0 * \sin(q1 + q2 + q3) - 87.075 * \sin(q2 - 1.0 * q1 + q3 + q4) + 87.075 * \sin(q1 + q2 + q3 + q4) + 78.85 * \sin(q1 - 1.0 * q2 + 1.257) - 75.0 * \sin(q2 - 1.0 * q1 + q3)$
$\boldsymbol{z_e}$	$103.91 - 157.7 * \sin(q2 - 1.257) - 150.0 * \sin(q2 + q3) - 174.15 * \sin(q2 + q3 + q4)$

4 ReactorX-150 Implementation

The implementation chapter provides an explanation of the building blocks and implementation of the different software packages designed to operate and manipulate the Trossen Robotics X-Series robot arms. The primary focus is on the various software packages developed with the ROS framework. The specific robot arm in focus is the ReactorX-150, but the packages covered in this chapter are compatible with any X-Series arm.

The chapter begins with an introduction to what can be considered the main packages, the Arm Descriptions and Arm Control packages. These packages provide the base structure and functionalities upon which the other packages are built.

The subsequent sections describe the additional packages developed to extend the functionality of the robot arms. Each of these sections provides a detailed account of the purpose of the package, its structural breakdown, and the key arguments required for their operation. The structure of each package is visualized for better understanding of the relationships and interactions with the ROS nodes and other components.

The user manual for the X-series arms with the main arguments for the packages can be seen in [Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms]. The more extensive user manual containing the full list of arguments for the packages can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

4.1 Arm Descriptions

The Arm Descriptions package (“interbotix_xsarm_descriptions”) is responsible for accurately representing the X-Series arms’ appearance, structure, and kinematics in a simulation or visualization environment. The package contains the URDFs and meshes for all the X-Series arms. The appearance and textures of the arms come from the “interbotix_black.png” picture located in the “meshes” directory. The URDF’s are stored in xacro format, which enables customizability regarding which parts of the URDF’s shall be utilized. All packages for visualizing and/or controlling the X-Series arms reference this package.

The Arm Description package launched as a standalone package enables users to manipulate the individual joints for performing forward kinematics.

The simulated testing of the Arm Description package is covered in chapter 5.1.

4.1.1 Structure

The structure of the Arm Descriptions package can be seen visually represented in Figure 4.1.

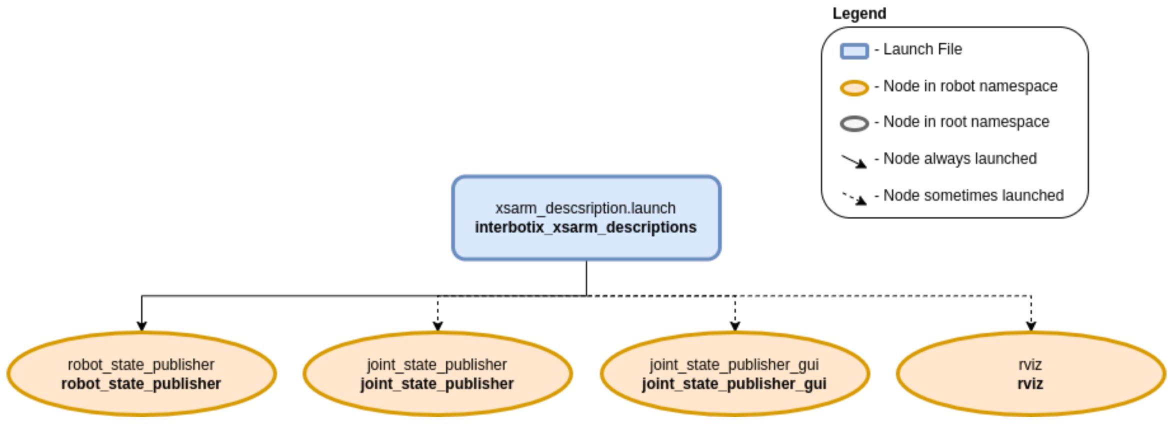


Figure 4.1: Arm descriptions package structure [34]

The launch file for the Arm Descriptions package is the “xsarm_description.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.1.2.

The Arm Descriptions package launches up to four nodes:

- **robot_state_publisher:** Responsible for calculating the forward kinematics of the robot. The node utilizes the joint positions from the “joint_states” topic and the URDF specified by the “robot_description” parameter for the calculations. The results are published via the “tf” topic.
- **joint_state_publisher:** Parses the “robot_description” parameter to identify all non-fixed joints and subsequently publishes a JointState message containing the definitions of these joints.
- **joint_state_publisher_gui:** Launches a GUI for manipulation of the joints of the robot.
- **rviz:** Utilizes the “tf” topic transforms to display a virtual model of the robot.

4.1.2 Arguments

The arguments seen in Table 4-1 are the utilized and/or most important arguments in the Arm Descriptions package regarding the simulation of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-1: Arm Descriptions package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_joint_pub_gui	false	Launches a user-friendly GUI for controlling joint angles called the joint_state_publisher GUI

4.2 Arm Control - Python and MATLAB

The Arm Control package (“interbotix_xsarm_control”) is responsible for controlling the physical robot. The package contains YAML files for all the X-Series arms. The YAML files specify initial register values, and names for the actuators for each robot arm, as well as publishing frequency, joint-group names, etc. The “modes.yaml” file is a common file for all the X-Series arms. The file defines operating mode parameters for the group of joints and the single joint. All packages that control the physical X-Series arm(s) reference this package.

The Arm Control package has the ability to run both Python and MATLAB scripts. It is made possible by utilizing the Python and MATLAB API of the “Interbotix_xs_modules” node seen in Figure 4.2. The simulated and physical testing of the Arm Control package with “Interbotix Control Panel” in RViz, Python scripts and “rostopic” commands from the terminal window is covered chapter 5.2 and 6.1. Controlling the physical or simulated robot arm with MATLAB scripts is not covered in this thesis. How to code Python and the Python scripts is not covered in this thesis, but to understand the commands in the Python scripts, see [35].

There are a few requirements that must be fulfilled in order to utilize the Python and MATLAB API:

- Arm joints must be set to “position” control.
- Gripper set to “PWM” control.
- Drive Mode registers of the arm-joint motors set to Time-Based-Profile.

The requirements above are conveniently enough the default values when running the Arm Control package.

4.2.1 Structure

The structure of the Arm Control package can be seen visually represented in Figure 4.2.

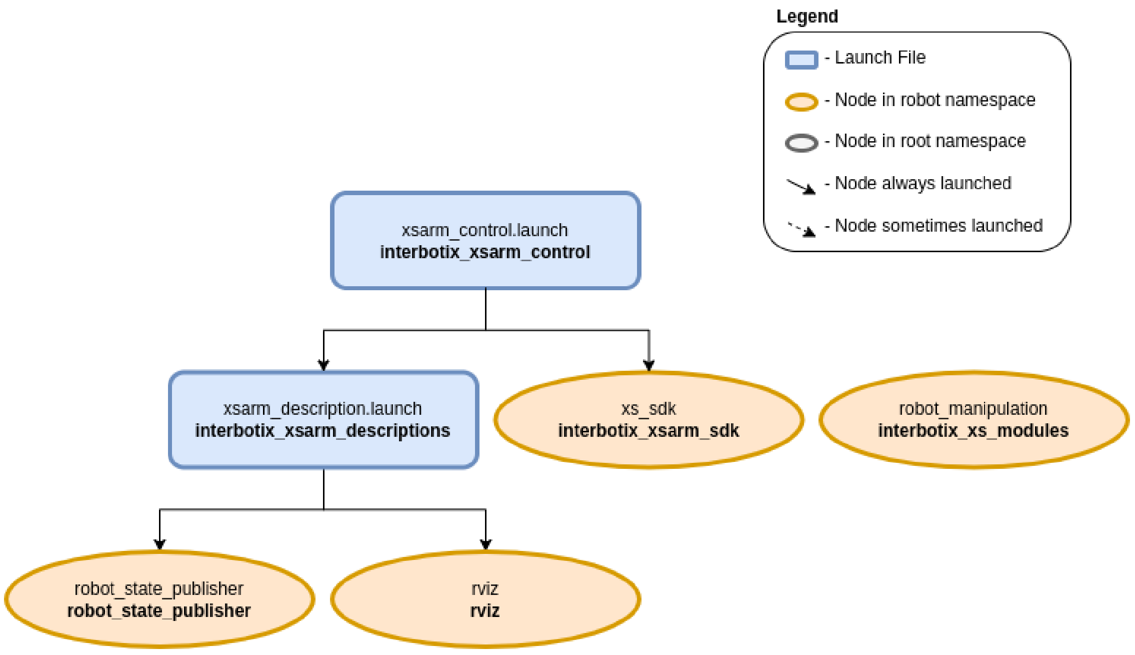


Figure 4.2: Arm Control package structure for Python and MATLAB [36] [37]

The launch file for the Arm Control package is the “xsarm_control.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.2.3. The Arm Control package is built on top of the “interbotix_xsarm_descriptions” package covered in chapter 4.1.

The Arm Control package launches up to two nodes and a separate package:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_descriptions” package, see chapter 4.1.
- **xs_sdk:** Controls the DYNAMIXEL motors of the robotic arm and loads the URDF to the “robot_description” parameter.
- **robot_manipulation:** The node launches when a Python or MATLAB script is run, publishes data to the necessary ROS topics and upon completion of the script the node is killed. This node cannot be launched from a launch file or the terminal window, it can only be launched by executing a Python or MATLAB script.

4.2.2 Kinematics without MoveIt

Forward/direct and inverse kinematics are handled within the “interbotix_xs_sdk” package. The package utilizes custom-written kinematic solvers from the “xs_sdk” node for the X-Series arms to handle direct and inverse kinematics. The package is designed to work with various Interbotix robot arm models, and the custom kinematic solvers account for the differences in robot configurations and joint constraints. This approach allows the “interbotix_xs_sdk” package to provide control of the X-Series arms without relying on a separate kinematics library.

4.2.3 Arguments

The arguments seen in Table 4-2 are the utilized and/or most important arguments in the Arm Control package regarding the simulation and physical testing in chapter 5.2 and 6.1. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-2: Arm Control package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true

4.3 MoveIt Configuration, Interface, and API

This subchapter covers two packages regarding MoveIt. The first package is the MoveIt Configuration package and the second one is the MoveIt Interface and API package. The MoveIt packages utilize several sub packages described in other chapters. The MoveIt packages enables users to perform both forwards and inverse kinematics.

The MoveIt Configuration package ("interbotix_xsarm_moveit") is the basis package containing the configuration files necessary to run MoveIt with the X-Series arms, either if it's with the physical arm, a Gazebo simulated arm or just a RViz simulated arm.

The MoveIt Interface and API package ("interbotix_xsarm_moveit_interface") is built on top of the basis "interbotix_xsarm_moveit" package. The MoveIt Interface and API package contains a small API and GUI, enabling the user to command the arm and end-effector to desired poses and positions. The package also contains a short Python script guide, which is a modified version of the Move Group Python Interface Tutorial script. [38]

The simulated and physical testing of the MoveIt Configuration and the MoveIt Interface and API packages are covered in chapter 5.3 and 6.2.

4.3.1 Structure

The structure of the MoveIt Configuration package can be seen visually represented in Figure 4.3 from the "xsarm_moveit.launch" file and down, the entire structure is the structure of the MoveIt Interface and API package.

The launch file for the MoveIt Configuration package is the "xsarm_moveit.launch" file, and the launch file for the MoveIt Interface and API package is the "xsarm_moveit_interface.launch" file. The launch files are XML files that specify the ROS nodes, parameters, and other settings required to launch the packages. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.3.2.

The MoveIt Configuration package contains one node which is always launched, one node available for launch and three more packages available for launch:

- **move_group:** The node is always launched and is responsible for trajectory planning for the arm and gripper of the X-Series arm.

Optional to launch:

- **rviz:** Responsible for visualizing the X-Series arm with the MoveIt MotionPlanning plugin in RViz.

One of the following three packages must be launched:

- **xsarm_gazebo.launch:** Launches the Gazebo package for simulation of the arm in Gazebo, see chapter 4.4.
- **xsarm_ros_control.launch:** Launches the Arm Control package to control the physical X-Series arm, see chapter 4.2.
- **xsarm_xsarm_description.launch:** Launches the Arm Descriptions package for simulation and visualization of the X-Series arm in RViz, see chapter 4.1.

The MoveIt Interface and API package is built on top of the basis “interbotix_xsarm_moveit” package (covered above), but has three additional nodes available for launch:

- **moveit_interface:** A small C++ API utilizing MoveIt’s planner, designed to simplify the process of commanding custom poses to the end-effector of an X-Series arm.
- **moveit_interface_gui:** A GUI with text boxes and sliders for custom end-effector poses. Utilizes the “moveit_interface” API for planning and executing trajectories.
- **moveit_python_interface:** A short Python script guide, press “enter” in the terminal window to start the tutorial.

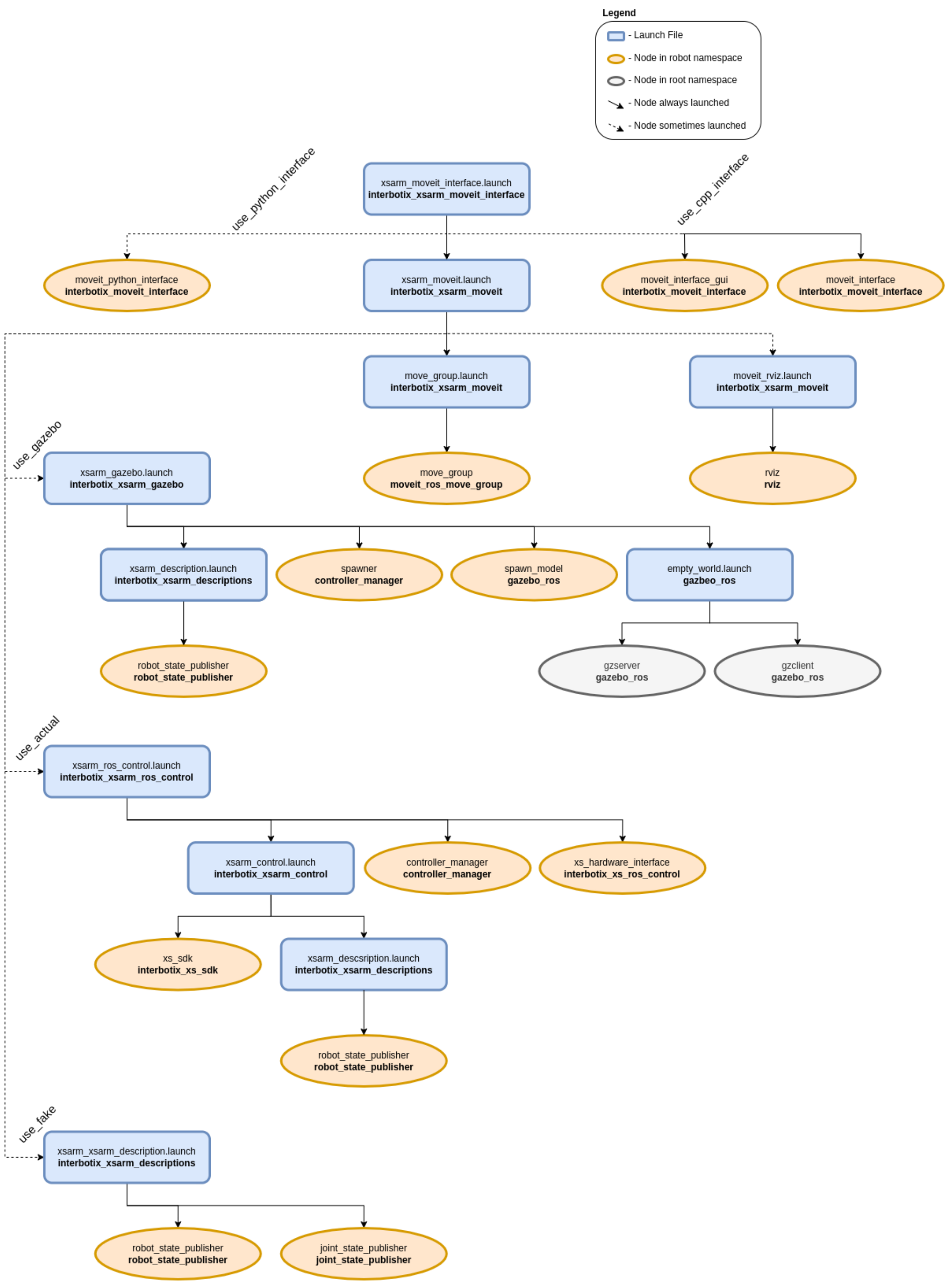

Legend
- Launch File
- Node in robot namespace
- Node in root namespace
- Node always launched
- Node sometimes launched
use_python_interface
xsarm_moveit_interface.launch interbotix_xsarm_moveit_interface
use_cpp_interface
moveit_python_interface interbotix_moveit_interface
xsarm_moveit.launch interbotix_xsarm_moveit
moveit_interface_gui interbotix_moveit_interface
moveit_interface interbotix_moveit_interface
move_group.launch interbotix_xsarm_moveit
moveit_rviz.launch interbotix_xsarm_moveit
use_gazebo
xsarm_gazebo.launch interbotix_xsarm_gazebo
move_group moveit_ros_move_group
rviz rviz
xsarm_description.launch interbotix_xsarm_descriptions
spawner controller_manager
spawn_model gazebo_ros
empty_world.launch gazbeo_ros
robot_state_publisher robot_state_publisher
gzserver gazebo_ros
gzclient gazebo_ros
use_actual
xsarm_ros_control.launch interbotix_xsarm_ros_control
xsarm_control.launch interbotix_xsarm_control
controller_manager controller_manager
xs_hardware_interface interbotix_xs_ros_control
xs_sdk interbotix_xs_sdk
xsarm_descscription.launch interbotix_xsarm_descriptions
robot_state_publisher robot_state_publisher
use_fake
xsarm_xsarm_description.launch interbotix_xsarm_descriptions
robot_state_publisher robot_state_publisher
joint_state_publisher joint_state_publisher

Figure 4.3: MoveIt Interface and API package structure [39]

4.3.2 Kinematics with MoveIt

MoveIt does not use the default kinematic solver for the X-Series robotic arms, it utilizes the LMA (Levenberg-Marquardt Algorithm). [40] [41] The default kinematic solver used by MoveIt is the KDL (Kinematics and Dynamics Library) from Orocos. [42] [43]

The LMA is a kinematic solver is used for solving generic curve-fitting problems. The LMA is also known as damped least-squares method. LMA utilizes interpolation between the Gauss-Newton algorithm (GNA) and the method of gradient decent. [44]

4.3.3 Arguments

The arguments seen in Table 4-3 are the utilized and/or most important arguments in the MoveIt Interface and API package regarding the simulation and physical testing covered in chapter 5.3 and 6.2. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Note that the “moveit_interface_gui” and “use_python_interface” arguments are only available for the “interbotix_xsarm_moveit_interface” package, and not the basic “interbotix_xsarm_moveit” package.

Table 4-3: MoveIt Interface and API package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_moveit_rviz	true	Launches RViz with the MoveIt plugin
use_gazebo	false	Simulate the robot with Gazebo
use_actual	false	Use the physical robot
use_fake	false	MoveIt generates a simulated robot to be controlled in RViz
dof	5	Defines the degrees of freedom of the robot arm
moveit_interface_gui	true	Launches a GUI customized to interface with the moveit_interface node
use_python_interface	false	Launches a Python Interface Tutorial node. Press “enter” in the terminal window to step through the tutorial

4.4 Gazebo Configuration

The Gazebo Configuration package ("interbotix_xsarm_gazebo") is responsible for simulating the X-Series arms in Gazebo. The package contains configuration files required to simulate the X-Series arms, such as YAML files with tuned PID gains and the "interbotix_texture.gazebo" file. The YAML files provide "ros_control" the necessary parameters for controlling the arms and gripper effectively.

There are two ways to utilize this package:

- Launching the package as a standalone package and controlling via the JointPositionController interface.
- Launch the package together with MoveIt via the FollowJointTrajectory interface.

In this thesis the Gazebo Configuration package is utilized together with MoveIt. The simulation with Gazebo and MoveIt is covered in chapter 5.3.

4.4.1 Structure

The structure of the Gazebo Configuration package can be seen visually represented in Figure 4.4.

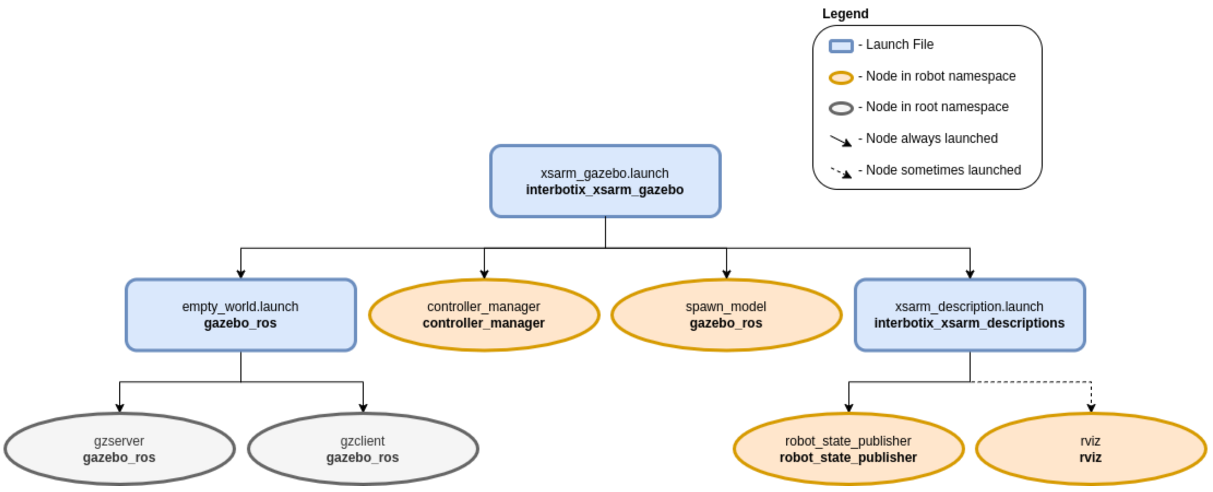


Figure 4.4: Gazebo package structure [45]

The launch file for the Gazebo Configuration package is the "xsarm_gazebo.launch" file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.4.2. The Arm Control package is built on top of the "interbotix_xsarm_descriptions" package covered in chapter 4.1 and the "gazebo_ros" package covered below.

The Gazebo Configuration package launches two packages and two nodes:

- **xsarm_descriptions.launch:** Launches the "interbotix_xsarm_descriptions" package, see chapter 4.1.
- **controller_manager:** Loads and starts a set of controllers when launching the Gazebo Configuration package and stops and unloads the controllers on exit.

- **spawn_model:** Responsible for adding the robot model into the Gazebo world. The robot to add is defined by the "robot_description" parameter.
- **gazebo_ros** nodes:
  - **gzserver:** In charge of executing the physics update loop and generating sensor data in Gazebo.
  - **gzclient:** Responsible for the GUI for visualizing the simulation of the robot in Gazebo.

4.4.2 Arguments

The arguments seen in Table 4-4 are the utilized and/or most important arguments in the Gazebo package regarding the simulation of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-4:Gazebo package main arguments

Argument	Default Value	Description
robot_model	""	Requires the codename for the specific robot arm. "rx150" in the case of the ReactorX-150
use_rviz	true	Launches RViz
dof	5	Defines the degrees of freedom of the robot arm
world_name	see the Gazebo launch package link below	File path to the world file to be loaded by Gazebo
gui	true	Launches the GUI of Gazebo
paused	true	Launches Gazebo in a paused state
use_position_controllers	false	Enables commanding of arbitrary arm joint positions in Gazebo
use_trajectory_controllers	false	Enables commanding of arbitrary arm joint trajectories in Gazebo

4.5 ROS Controllers Configuration

The ROS Controllers Configuration package (“Interbotix_xsarm_ros_control”) is responsible for the ROS controllers providing MoveIt with the ability to control the X-Series arms. The ROS Controllers Configuration package is not meant to be run as a standalone package. This package is meant to be used via MoveIt. The ROS Controllers Configuration package works by receiving commands from the “FollowJointTrajectoryAction” interface from MoveIt and publishing them to the topics the “xs_sdk” node subscribes to.

In this thesis the ROS Controllers Configuration package is utilized together with MoveIt. The physical testing with the ROS Controllers Configuration package and MoveIt is covered in chapter 6.2.

4.5.1 Structure

The structure of the ROS Controllers Configuration package can be seen visually represented in Figure 4.5.

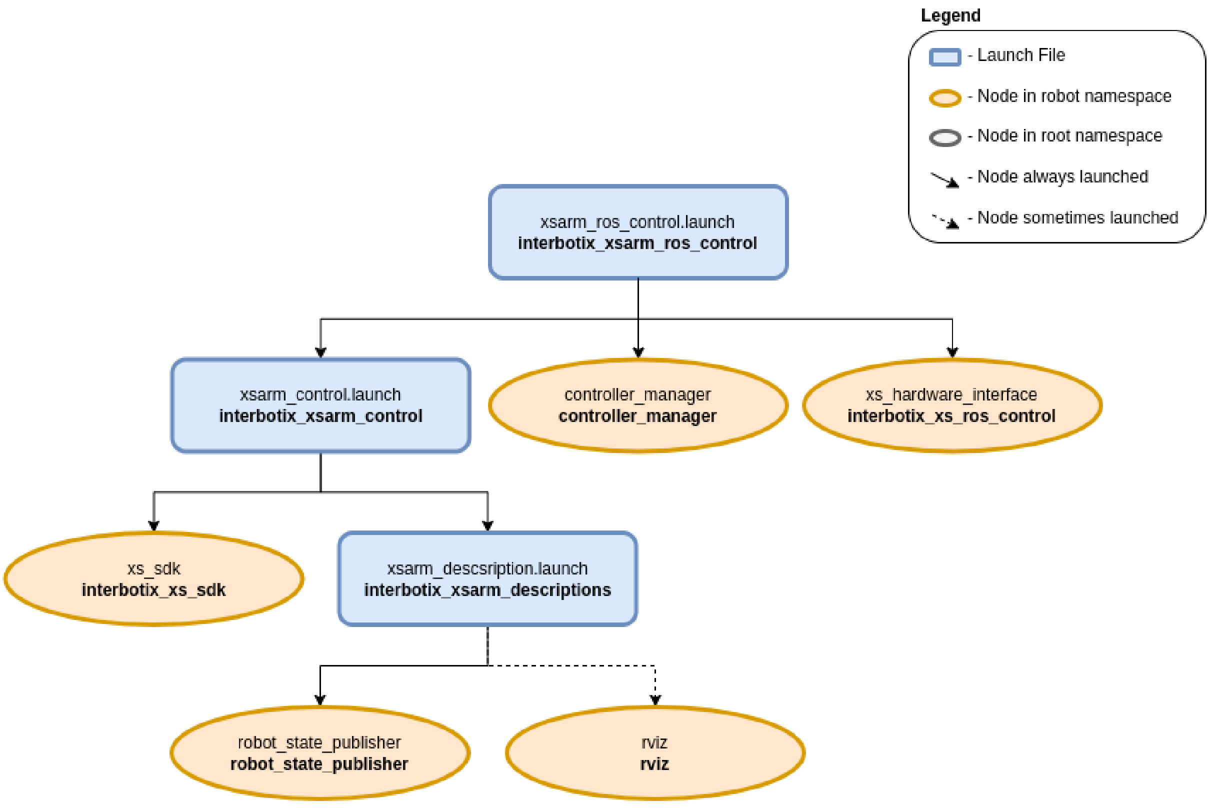


Figure 4.5: ROS Controllers package structure [46]

The launch file for the ROS Controllers Configuration package is the “xsarm_ros_control.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.5.2. The ROS Controller Configuration package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The ROS Controller Configuration package launches one package and two nodes:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_control” package, see chapter 4.2.
- **controller_manager:** Loads and starts a set of controllers when launching the Gazebo Configuration package and stops and unloads the controllers on exit.
- **xs_hardware_interface:** Responsible for publishing the commands received from the ROS controllers and publish them to the topics the “xs_sdk” node subscribes to.

4.5.2 Arguments

The arguments seen in Table 4-5 are the utilized and/or most important arguments in the ROS Controllers package regarding the physical testing of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-5: ROS Controllers main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	false	Launches RViz
dof	5	Defines the degrees of freedom of the robot arm

4.6 Perception Configuration

The Perception package ("interbotix_xsarm_perception") enables the X-Series robotic arms to recognize and manipulate small, non-reflective objects positioned on a non-reflective and non-transparent surface similar to a tabletop. The package is designed to work with any Intel RealSense color/depth camera, but only tested by Trossen Robotics with the Intel RealSense Depth Camera SR305 and D415 cameras. [47] [48]. The package utilizes the perception pipeline to get the point cloud data and GUI for object detection.

An Intel RealSense camera was not available for this thesis, so there was an attempt to launch it with a Lumens DC 125 camera and with a Raspberry Pi Camera Module 2, which is covered in chapter 7.7. The setup of the Raspberry Pi is covered in [Appendix I – Raspberry Pi Ubuntu and ROS Setup Guide].

Note that in order to run the Perception package optimally an AprilTag marker should be utilized. Otherwise, the position of the arm relative to the camera must be input manually. The manual input of the position of the arm relative to the camera is prone to error and will not be covered in this thesis.

The Perception package is not available for simulation and was not successfully tested with the physical robot, it is therefore covered in chapter 7.7.

4.6.1 Structure

The structure of the Perception package can be seen visually represented in Figure 4.6.

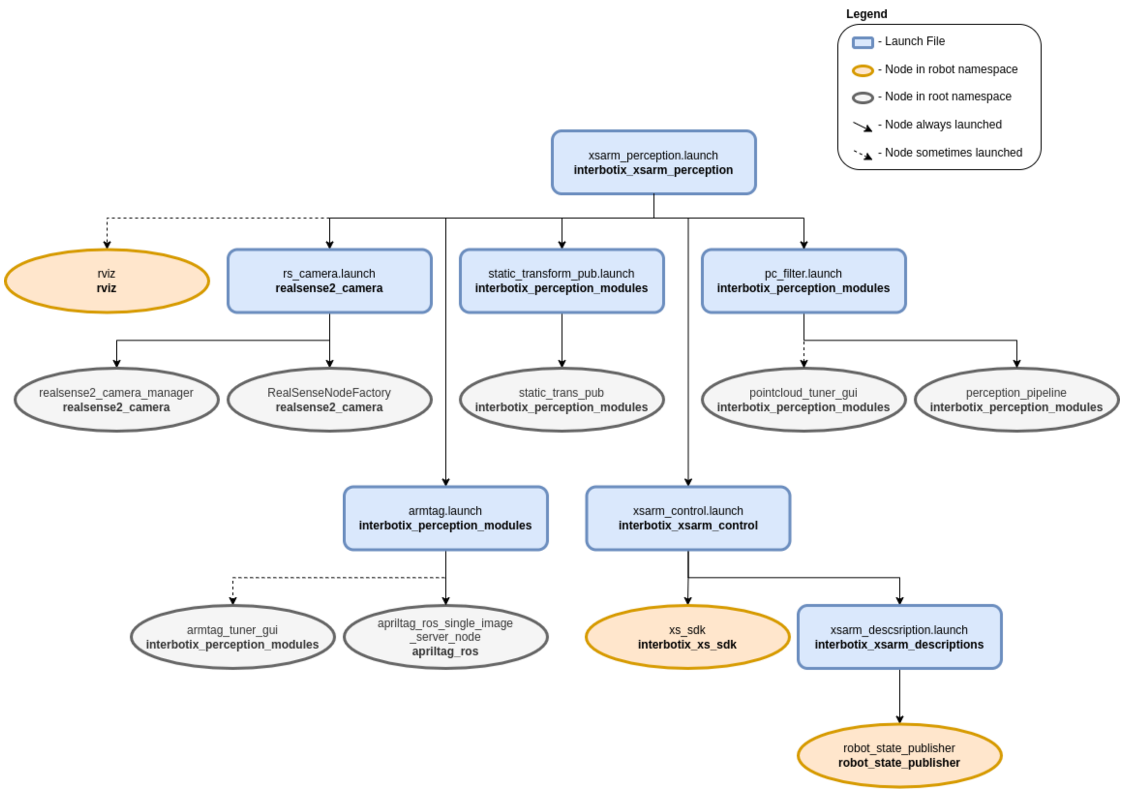

Legend
- Launch File
- Node in robot namespace
- Node in root namespace
- Node always launched
- Node sometimes launched
xsarm_perception.launch
interbotix_xsarm_perception
rviz
rviz
rs_camera.launch
realsense2_camera
static_transform_pub.launch
interbotix_perception_modules
pc_filter.launch
interbotix_perception_modules
realsense2_camera_manager
realsense2_camera
RealSenseNodeFactory
realsense2_camera
static_trans_pub
interbotix_perception_modules
pointcloud_tuner_gui
interbotix_perception_modules
perception_pipeline
interbotix_perception_modules
armtag.launch
interbotix_perception_modules
xsarm_control.launch
interbotix_xsarm_control
armtag_tuner_gui
interbotix_perception_modules
apriltag_ros_single_image_server_node
apriltag_ros
xs_sdk
interbotix_xs_sdk
xsarm_descsription.launch
interbotix_xsarm_descriptions
robot_state_publisher
robot_state_publisher

Figure 4.6: Perception package structure [49]

The launch file for the Perception package is the “xsarm_perception.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 0. The Perception package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The Perception package launches four packages, with sub nodes, and one optional node:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_control” package, see chapter 4.2.
- **rs_camera.launch:** Launches the nodes responsible for being able to use the Intel RealSense cameras with ROS.
    - **RealSenseNodeFactory:** A nodelet responsible for the creation of ROS interface for the RealSense camera
- **armtag.launch:** Launches the nodes responsible for establishing the necessary transform for accurate control of the arm’s end-effector in reference to the camera, using the AprilTag markers.
    - **armtag_tuner_gui:** A node responsible for generating a GUI for obtaining the previously mentioned transform.
    - **apriltag_ros_single_image_server_node:** The node responsible for obtaining the transform of the end-effector in reference to the camera.
- **static_transform_pub.launch:** Responsible for loading, saving and publishing the static transform to the correct ROS topic
- **pc_filter.launch:** Launches the nodes responsible for obtaining the point cloud and tuning the point cloud parameters for obtaining the objects from the generated clusters.
    - **pointcloud_tuner_gui:** The node responsible for launching a GUI with the ability to tune filter parameters used in the perception pipeline.
    - **perception_pipeline:** The node responsible for the implementation of the Perception Pipeline using the PointCloud Library. [50]

Optional to launch:

- **rviz:** Responsible for visualizing the X-Series arm with the MoveIt MotionPlanning plugin in RViz.

For clarification, a nodelet in ROS is a tool that allows multiple algorithms to run within the same process for efficient message passing, and the nodelet manager is responsible for loading and unloading nodelets as requested.

4.6.2 Arguments

The arguments seen in Table 4-6 are the utilized and/or most important arguments in the Perception package regarding the physical testing of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-6: Perception package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
load_configs	true	Set to true if the motor configurations shall be written to the motors. Is only necessary to do it the first time the node starts up, reduces startup time if set to “false”
use_pointcloud_tuner_gui	false	Displays a GUI for tuning filter parameters
use_armtag_tuner_gui	false	Enables the user to publish the “ref_frame” to “arm_base_frame” transform via a GUI

4.7 Joystick Control

The Joystick Control package (“interbotix_xsarm_joy”) provides the functionality to be able to control the X-Series arms with either a PlayStation or Xbox controller wireless controller via Bluetooth. The package is set up to be compatible with PlayStation 3, PlayStation 4, and Xbox 360 wireless controllers.

There are two preliminary requirements to running the Joystick Control package:

- The “arm” joint’s operating mode must be set to “position”.
- The “gripper” joint’s operating mode must be set to “pwm”.
- The wireless joystick controller must be connected to the computer, see [Appendix H – Joystick controller pairing] to connect the joystick controller.

The requirements above are conveniently enough the default values when running the Joystick Control package.

The simulated and physical testing of the Joystick Control package is covered in chapter 5.4 and 6.3.

4.7.1 Structure

The structure of the Joystick Control package can be seen visually represented in Figure 4.7.

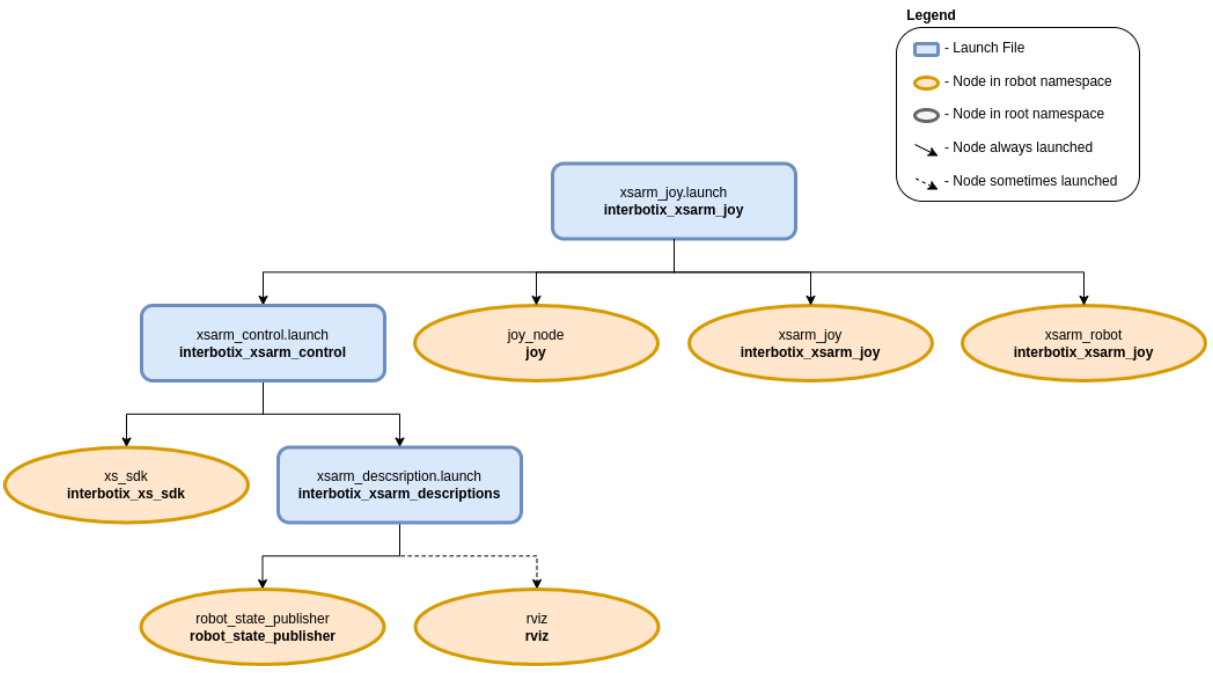


Figure 4.7: Joystick control package structure [51]

The launch file for the Joystick Control package is the “xsarm_joy.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.7.2. The Joystick Control package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The Joystick Controller package launches one package and three nodes:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_control” package, see chapter 4.2.
- **joy:** A ROS driver that interfaces with the previously mentioned joysticks, capturing input data and publishing it as “sensor_msgs/Joy” messages. The messages are published to the “commands/joy_raw” topic.
- **xsarm_joy:** Responsible for converting the “sensor_msgs/Joy” messages to “ArmJoy” messages. [52] The result is a more readable code, and it enables the user to remap buttons more conveniently.
- **xsarm_robot:** Reads the ArmJoy messages and sends commands for the joints and gripper to the “xs_sdk” node.

4.7.2 Arguments

The arguments seen in Table 4-7 are the utilized and/or most important arguments in the Joystick control package regarding the simulation and physical testing in chapter 5.4 and 6.3. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-7: Joystick control package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
controller	ps4	Define the type of controller to be used, either “ps3”, “ps4” or “xbox360”

4.8 Record and Playback

The Record and Playback package is located within the “interbotix_xsarm_puppet” package covered in 4.10. The Record and Playback package only utilizes some of the features of the “interbotix_xsarm_puppet” package and launches with its own launch file. The Record and Playback package enables the user to record manual manipulation of an X-Series arm, store the recording and playback the recorded motions with the same robot. The recording is stored in a ROS bag file, and it is possible to store multiple recordings by simply changing the name of the bag file. The ROS bag file is a binary file format and does not have a human-readable format.

The physical testing of the Record and Playback package is covered in chapter 6.4.

The Record and Playback package has the option of being simulated, but no user-friendly GUI for controlling the simulated model. It is possible to control the simulated arm with “rostopic” commands from the terminal window, but that will not be included in the thesis. The procedure and commands for controlling the simulated arm are the same as for the simulated Arm Control package covered in chapter 5.2.3. The only difference being the name of the topic the command is published to.

4.8.1 Structure

The structure of the Record and Playback package can be seen visually represented in Figure 4.8.

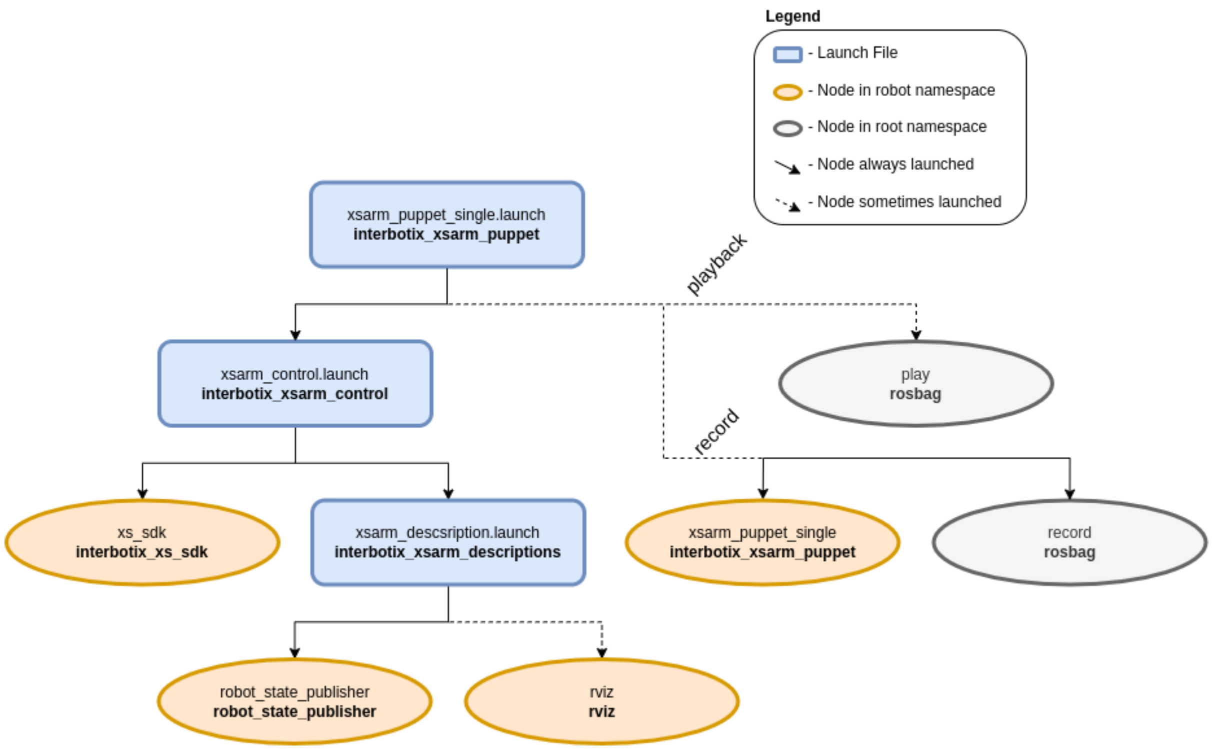


Figure 4.8: Record and playback package structure [53]

The launch file for the Record and Playback package is the “xsarm_puppet_single.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.8.2. The Record and Playback package is built on top of the "interbotix_xsarm_control" package and the "interbotix_xsarm_descriptions" package covered in chapter 4.2 and 4.1, respectively.

The Record and Playback package launches one package and three nodes available for launch:

- **xsarm_control.launch:** Launches the "interbotix_xsarm_control" package, see chapter 4.2.
- **xsarm_puppet_single:** Reads the X-Series arm's joint states, converts them to position commands and publishes them to the "/<robot__code_name>/commands/joint_group" and "/<robot_code_name>/commands/joint_single" topics. Node is launched together with the "record" node.
- **record:** Record the "/<robot__code_name>/commands/joint_group" and "/<robot_code_name>/commands/joint_single" topics.
- **play:** Responsible for interoperating the recorded ROS bag file and play back the file as commands to the arm. The playback has a three second start delay ensuring the "xs_sdk" node is able to load.

4.8.2 Arguments

The arguments seen in Table 4-8 are the utilized and/or most important arguments in the Record and playback package regarding the simulation and physical testing of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-8: Record and playback package main arguments

Argument	Default Value	Description
robot_model	""	Requires the codename for the specific robot arm. "rx150" in the case of the ReactorX-150
use_rviz	true	Launches RViz
record	false	Record the physical manipulation of the robot to a bag file
playback	false	Playback the recorded manipulation of the arm
bag_name	$(arg robot_name)_commands	Change this to set an arbitrary file name to the ROS bag file

use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true

4.9 Arm Diagnostic Tool

The Arm Diagnostic Tool package ("interbotix_xsarm_diagnostic_tool") is designed as a joint data analyzer. One specified joint is given commands while the rest are torqued in a stationary position. The data of one arbitrary joint is then plotted and recorded. The package enables the user to observe live joint data over a specified period of time, which can be useful if there are concerns about the strain on a specific joint when performing a task. The data is saved to a ROS bag file and can be converted to a CSV file for improved readability and potential further use.

The live joint data is shown in three plots (y-axis vs. x-axis):

- Position [rad] and velocity [rad/s] vs. time [s]
- Effort [mA] vs time [s]
- Temperature [°C] vs. time [s]

The Arm Diagnostic Tool package is reserved for the physical testing of the robot arms only. The physical testing of the Arm Diagnostic Tool package is covered in chapter 6.5.

The commanded joint follows a symmetrical sinusoidal trajectory around 0 radians. The minimum absolute value of the upper and lower joint limits defines the upper bound of the sinusoidal trajectory.

4.9.1 Structure

The structure of the Arm Diagnostic Tool package can be seen visually represented in Figure 4.9.

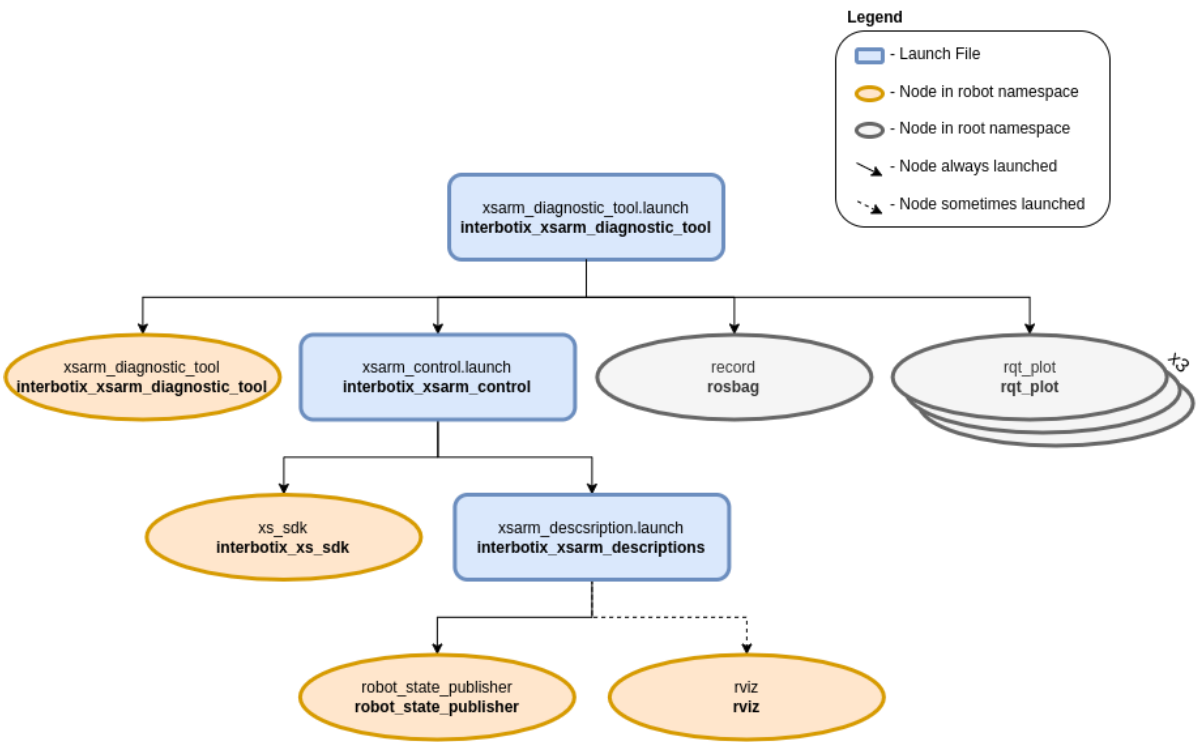


Figure 4.9: Arm Diagnostic Tool package structure [54]

The launch file for the Arm Diagnostic Tool package is the “xsarm_diagnostic_tool.lauch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.9.2. The Arm Diagnostic Tool package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The Arm Diagnostic Tool package launches a single package and four distinct nodes, with one of these nodes being launched in three separate instances:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_control” package, see chapter 4.2.
- **xsarm_diagnostic_tool:** Responsible for sending the joint position commands to the specified joint, and for publishing the temperatures of all the joints except the gripper to the “/<robot_name>/temperatures/joint_group” topic.
- **record:** Saves the recording of the “/<robot_name>/commands/joint_single”, “/<robot_name>/joint_states”, and “/<robot_name>/temperatures/joint_group” topics to a ROS bag file.
- **rqt_plot:** Launches three instances of this node to plot the three plots specified in the introduction of chapter 4.9.

4.9.2 Arguments

The arguments seen in Table 4-9 are the utilized and/or most important arguments in the Arm Diagnostic Tool package regarding the physical testing of the arm covered in chapter 6.5. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-9: Arm Diagnostic Tool package main arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
cmd_joint	waist	The joint name of the joint to be rotated
observe_joint	waist	The joint name of the joint to be observed
test_duration	600	Defines the duration of the test in seconds

bag_name	“$(arg observe_joint)_diagnostics”	Change this to set an arbitrary file name to the ROS bag file
use_rqt	true	Launches the rqt plots. The rqt plots are set up with preloaded topics

4.10 Arm Puppeteering

The Arm Puppeteering package ("interbotix_xsarm_puppet") provides the functionality to be able to manually manipulate one X-Series arm and one or more X-series arm(s) replicate the movements in real time.

The Arm Puppeteering package has the option of being simulated, but no user-friendly GUI for controlling the simulated model. It is possible to control the simulated "master" arm with "rostopic" commands from the terminal window, but that will not be included in the thesis. The procedure and commands for controlling the simulated "master" arm and subsequently the simulated "puppet" arm are the same as for the simulated Arm Control package covered in chapter 5.2.3. The only difference being the name of the topic the command is published to.

The current subchapter and the two following subchapters include working with two ReactorX-150 arms, so the physical setup of the two ReactorX-150 arms are covered in chapter 4.10.1.

The physical testing of the Arm Puppeteering package can be seen in chapter 6.6.

4.10.1Physical setup of two ReactorX-150 arms

The physical setup of the two ReactorX-150 arms consists of the two ReactorX-150 arms lined up parallel to each other with 60 cm distance from the center of "waist" joints. When manipulating one of the 3D printed USN logos, the logo is placed directly at the center between the two robotics arms. The setup with a measuring stick for reference can be seen in Figure 4.10.

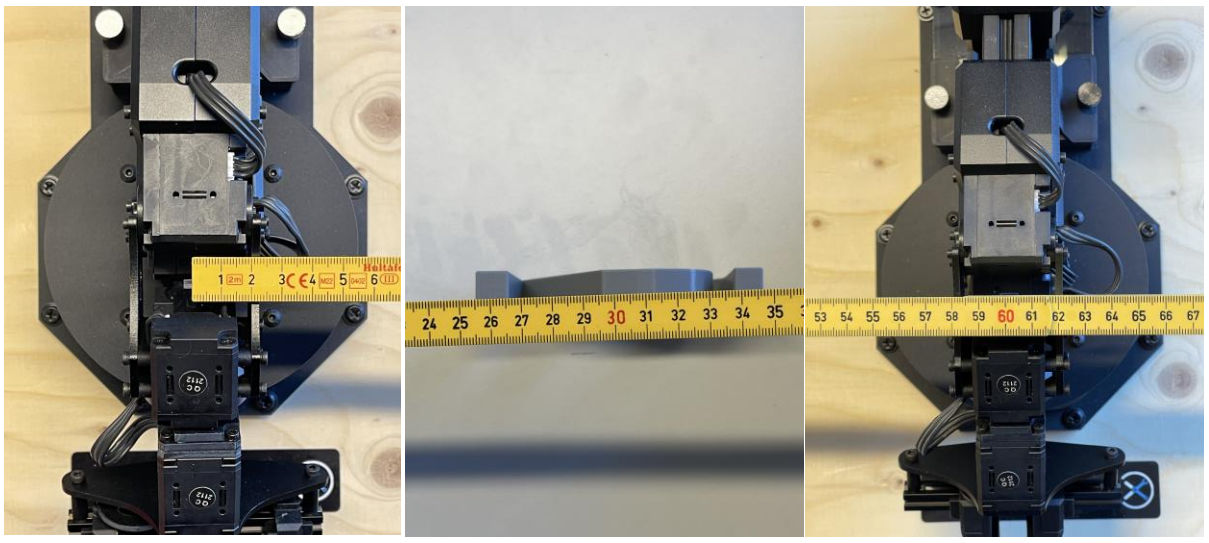


Figure 4.10: Individual arms and the USN logo with measurements

The full setup with and without the measuring stick can be seen in Figure 4.11.

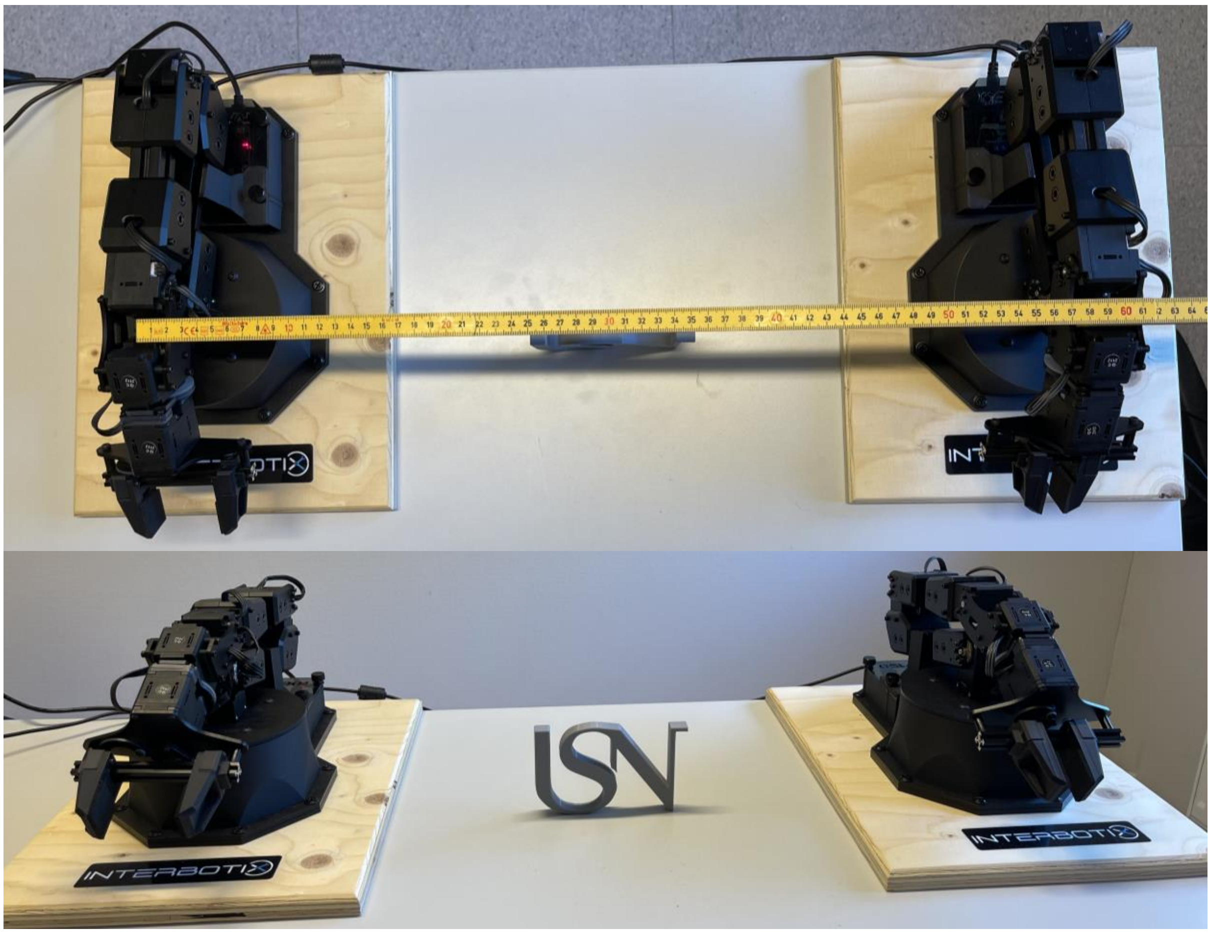


Figure 4.11: Full ReactorX-150 setup with and without measurements

4.10.2Structure

The structure of the Arm Puppeteering package can be seen visually represented in Figure 4.12.

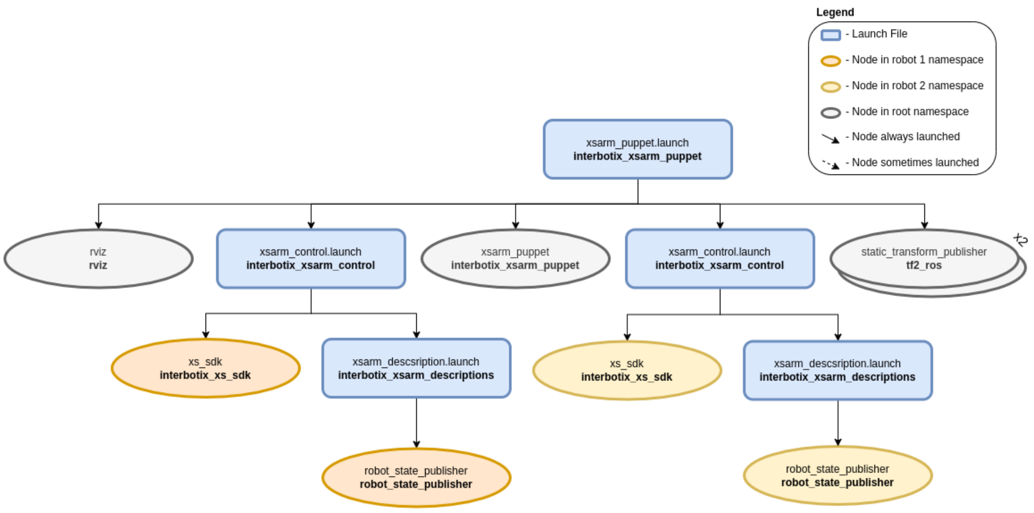


Figure 4.12: Arm Puppeteering package structure [55]

The launch file for the Arm Puppeteering package is the “xsarm_puppet.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.10.3.

The Arm Diagnostic Tool package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The Arm Puppeteering package launches three nodes and one package, where one of the nodes, as well as the package, is launched in a number of instances equal to the number of arms:

- **xsarm_control.launch:** Launches the “interbotix_xsarm_control” package, see chapter 4.2. This package is launched a number of instances equal to the number of arms.
- **xsarm_puppet:** Reads the joint states from the “master” arm and publishes them as position commands to the “puppet” arm(s).
- **static_transform_publisher:** This node is responsible for specifying the position of the X-Series arms relative to the “world” frame in RViz. This node is launched a number of instances equal to the number of arms.
- **rviz:** Launches one instance of RViz with a robot model for each arm. In the case of this thesis, RViz displays two arms.

4.10.3Arguments

The arguments seen in Table 4-10 are the utilized and/or most important arguments in the Arm Puppeteering package regarding the physical of the arm. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-10: Arm Puppeteering package main arguments

Argument	Default Value	Description
robot_model_master	“”	Requires the codename for the specific robot arm to be the master. “rx150” in the case of the ReactorX-150
robot_model_puppet	“”	Requires the codename for the specific robot arm to be the puppet. “rx150” in the case of the ReactorX-150
use_puppet_rviz	true	launches RViz with visualization of both arms
use_rviz	true	Launches RViz

4.11 Dual Arm Control

The Dual Arm Control package (“interbotix_xsarm_dual”) provides the ability to get multiple X-Series arms working simultaneously. This thesis will cover two X-Series arms working simultaneously, but in theory the only limitation to the number of arms working simultaneously is the number of USB ports available.

The simulated and physical testing of the Dual Arm Control package is covered in chapter 5.5 and 6.7. The physical setup of the two ReactorX-150 arms can be seen in chapter 4.10.1.

4.11.1Structure

The structure of the Dual Arm Control package can be seen visually represented in Figure 4.13.

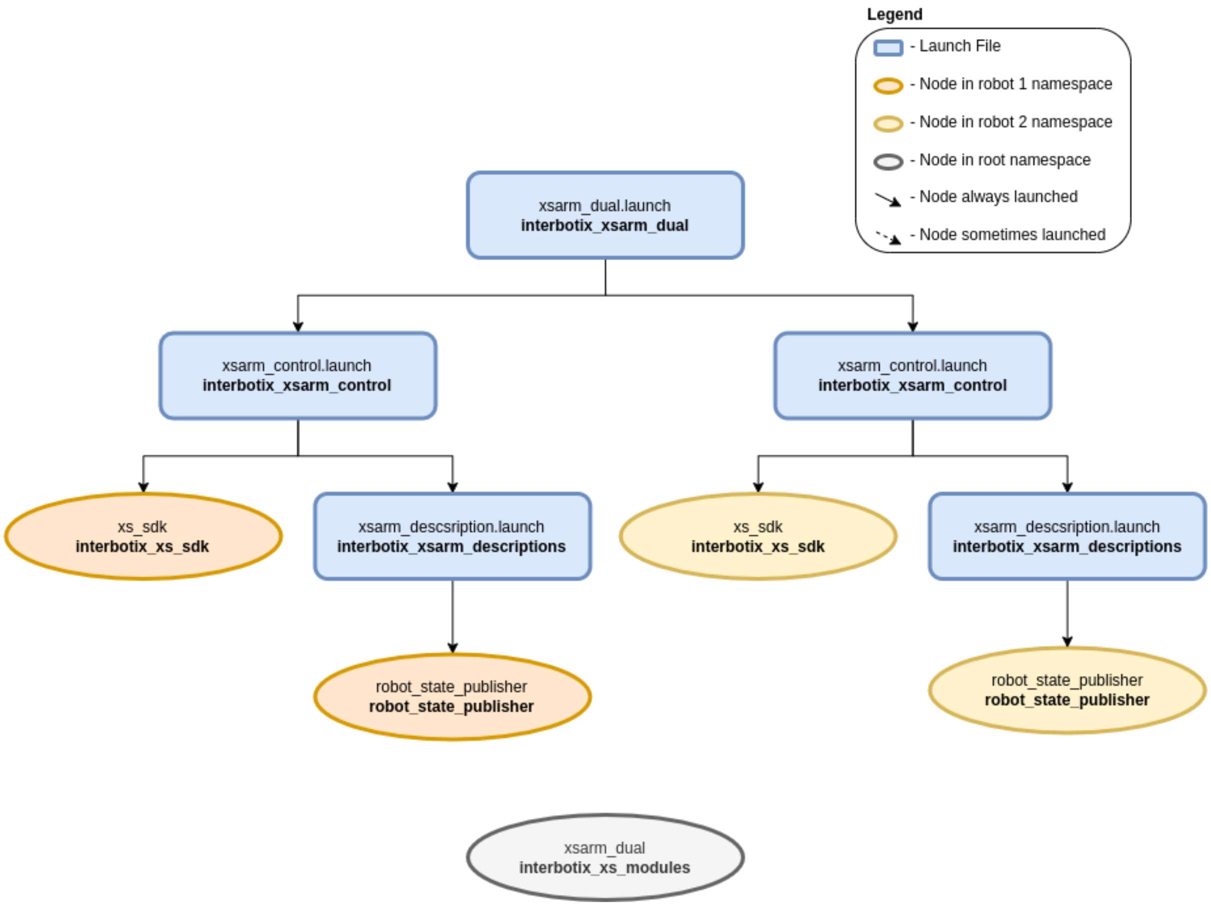


Figure 4.13: Dual Arm Control package structure [56]

The launch file for the Dual Arm Control package is the “xsarm_dual.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.11.2.

The Arm Diagnostic Tool package is built on top of the “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.2 and 4.1, respectively.

The Dual Arm Control package launches one package and one node, where the package is launched in a number of instance equal to the number of arms, which in this case is two:

- **xsarm_control.launch:** Launches the "interbotix_xsarm_control" package, see chapter 4.2. This package is launched a number of instances equal to the number of arms.
- **xsarm_dual:** Responsible for the X-Series arms to work with Python and MATLAB scripts.

4.11.2Arguments

The arguments seen in Table 4-11 are the utilized and/or most important arguments in the Dual Arm Control package regarding the simulation and physical testing in chapter 5.5 and 6.7. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-11: Dual Arm Control package main arguments

Argument	Default Value	Description
robot_model_1	""	Requires the codename for the specific robot arm. "rx150" in the case of the ReactorX-150
robot_model_2	""	Requires the codename for the specific robot arm. "rx150" in the case of the ReactorX-150
use_sim	false	Set to "true" if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_dual_rviz	false	launches RViz with visualization of both arms

4.12 Dual Arm Joystick Control

The Dual Arm Joystick package (“interbotix_dual_arm_joy”) provides the ability to operate multiple X-series arms simultaneously with either a PlayStation or Xbox controller wireless via Bluetooth. The package is set up to be compatible with PlayStation 3, PlayStation 4, and Xbox 360 wireless controllers.

It is recommended to be familiar with both the Joystick Control package and the Dual arm control package covered in chapter 4.7 and 4.11, respectively.

The simulated and physical testing of the Dual Arm Joystick Control package is covered in chapter 5.6 and 6.8. The physical setup of the two ReactorX-150 arms can be seen in chapter 4.10.1.

The wireless joystick controller of choice must be connected when running this package, see [Appendix H – Joystick controller pairing] to connect the wireless joystick controller.

4.12.1Structure

The structure of the Dual Arm Joystick Control package can be seen visually represented in Figure 4.14.

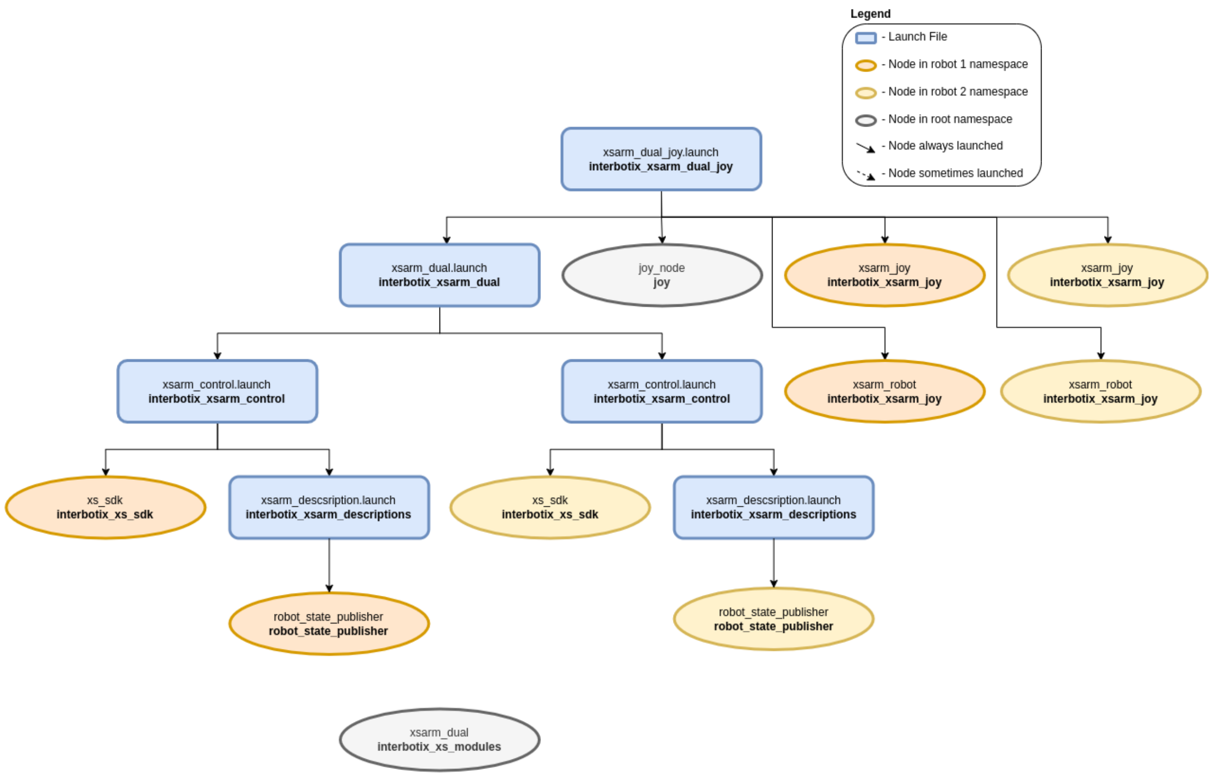

Legend
- Launch File
- Node in robot 1 namespace
- Node in robot 2 namespace
- Node in root namespace
- Node always launched
- Node sometimes launched
xsarm_dual_joy.launch interbotix_xsarm_dual_joy
xsarm_dual.launch interbotix_xsarm_dual
joy_node joy
xsarm_joy interbotix_xsarm_joy
xsarm_joy interbotix_xsarm_joy
xsarm_robot interbotix_xsarm_joy
xsarm_robot interbotix_xsarm_joy
xsarm_control.launch interbotix_xsarm_control
xsarm_control.launch interbotix_xsarm_control
xs_sdk interbotix_xs_sdk
xsarm_descsription.launch interbotix_xsarm_descriptions
xs_sdk interbotix_xs_sdk
xsarm_descsription.launch interbotix_xsarm_descriptions
robot_state_publisher robot_state_publisher
robot_state_publisher robot_state_publisher
xsarm_dual interbotix_xs_modules

Figure 4.14: Dual Arm Joystick Control package structure [57]

The launch file for the Dual Arm Joystick package is the “xsarm_dual_joy.launch” file. The launch file is an XML file that specifies the ROS nodes, parameters, and other settings required to launch the package. The nodes, parameters and other setting can be customized with the package arguments covered in chapter 4.12.2.

The Arm Diagnostic Tool package is built on top of the “interbotix_xsarm_joy” package, “interbotix_xsarm_dual” package, “interbotix_xsarm_control” package and the “interbotix_xsarm_descriptions” package covered in chapter 4.7, 4.11, 4.2 and 4.1, respectively.

The Dual Arm Joystick package launches one package and up to four nodes, where two of the nodes are launched in a number of instance equal to the number of arms, which in this case is two:

- **xsarm_dual.launch:** launches the “Interbotix_xsarm_dual” package, see chapter 4.11.
- **joy:** A ROS driver that interfaces with the previously mentioned joysticks, capturing input data and publishing it as “sensor_msgs/Joy” messages. The messages are published to the “commands/joy_raw” topic.
- **xsarm_joy:** Responsible for converting the “sensor_msgs/Joy” messages to ArmJoy messages. [52] The result is a more readable code, and it enables the user to remap buttons more conveniently. This package is launched a number of instances equal to the number of arms.
- **xsarm_robot:** Reads the ArmJoy messages and sends commands for the joints and gripper to the “xs_sdk” node. This package is launched a number of instances equal to the number of arms.
- **xsarm_dual:** The node launches when a Python or MATLAB script is run, publishes data to the necessary ROS topics of the two arms, and upon completion of the script the node is killed. This node cannot be launched from a launch file or the terminal window, it can only be launched by executing a Python or MATLAB script.

4.12.2Arguments

The arguments seen in Table 4-12 are the utilized and/or most important arguments in the Dual Arm Joystick Control package regarding the simulation and physical testing in chapter 5.6 and 6.8. The full list of arguments together with a simple user manual can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Table 4-12: Dual Arm Joystick Control package main arguments

Argument	Default Value	Description
robot_model_1	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_model_2	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150

use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_dual_rviz	false	Launches RViz with visualization of both arms
threshold	0.75	Specifies the sensitivity of the controller from 0 to 1, 1 being the highest sensitivity
controller	ps4	Define the type of controller to be used, either “ps3”, “ps4” or “xbox360”

5 Simulation of ReactorX-150

This chapter presents the simulated testing of the ReactorX-150 robotic arm. Simulation of the ReactorX-150 accurately models the physical attributes of the robot and its environment. The simulation tools utilized are RViz and Gazebo, both powerful visualization/simulation tools.

The user manual for the X-series arms with the main arguments for the packages can be seen in [Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms]. The more extensive user manual containing the full list of arguments for the packages can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

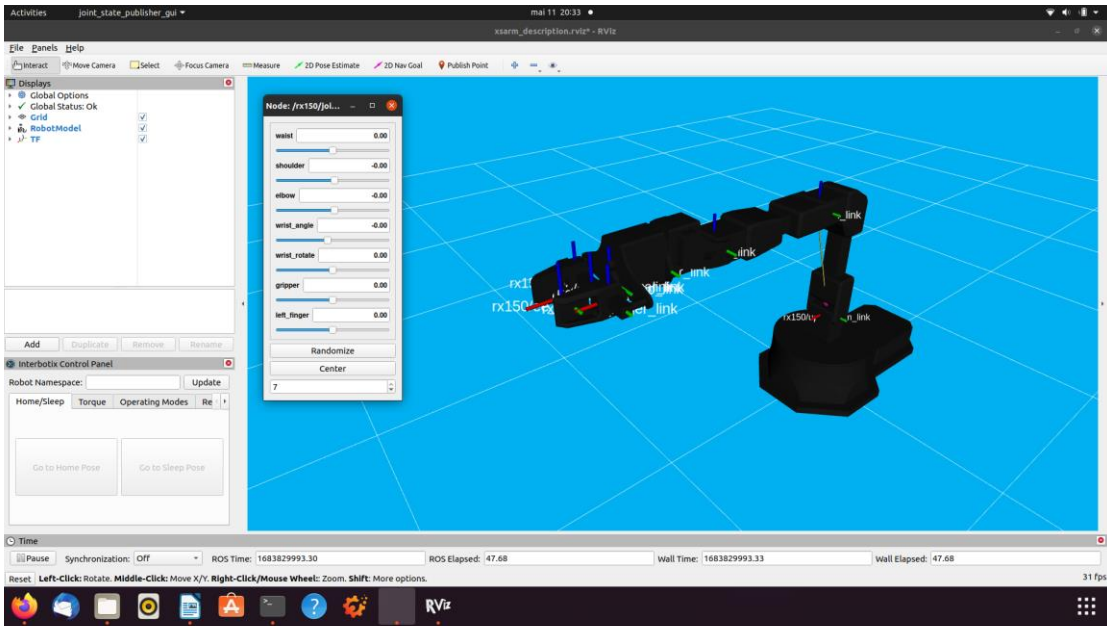
All commands throughout this chapter are meant to be run in an Ubuntu terminal window.

5.1 Arm Descriptions

The following command launches the Arm Descriptions package for the ReactorX-150 with RViz for visualization/simulation, and a GUI for performing forward kinematics:

```
$ roslaunch interbotix_xsarm_descriptions xsarm_description.launch robot_model:=rx150
use_joint_pub_gui:=true
```

The resulting RViz window with the model of the ReactorX-150 and the GUI for performing forward kinematics ("joint_state_publisher_gui") can be seen in Figure 5.1. The "Interbotix Control Panel" appears in RViz but is not possible to utilize. The "Interbotix Control Panel" is only available for use by the Arm Control package simulated in chapter 5.2.



waist 0.00
shoulder -0.00
elbow -0.00
wrist_angle -0.00
wrist_rotate 0.00
gripper 0.00
left_finger 0.00

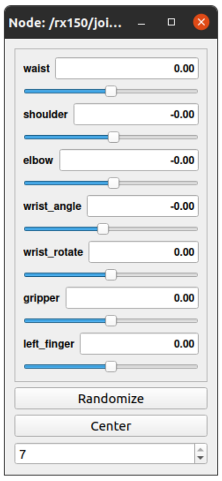
Synchronization: Off
ROS Time: 1683829993.30
ROS Elapsed: 47.68
Wall Time: 1683829993.33
Wall Elapsed: 47.68

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

31 fps

Figure 5.1: Arm Descriptions startup

The GUI for performing forward kinematics can be seen in Figure 5.1 appears in a separate window from RViz. A better view of the GUI can be seen in Figure 5.2. The GUI enables the control of all the joint individually, set random positions to all the joints with the "Randomize" button and to set the arm to "Home" position with the "Center" button.

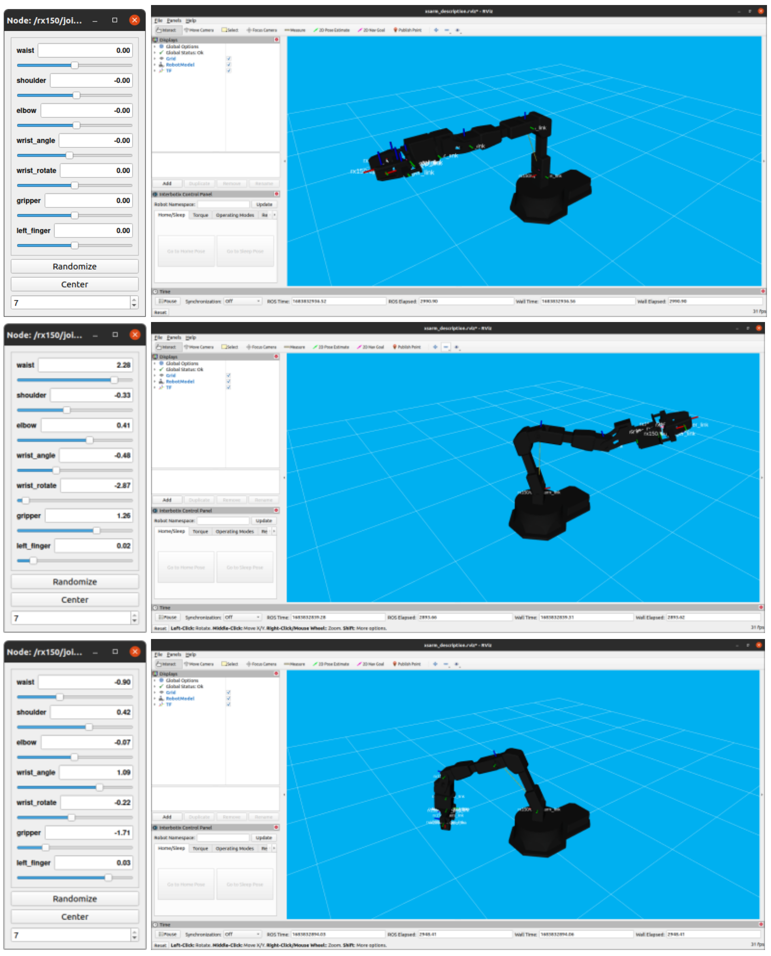


Node: /rx150/joi...

waist	0.00
shoulder	-0.00
elbow	-0.00
wrist_angle	-0.00
wrist_rotate	0.00
gripper	0.00
left_finger	0.00

Figure 5.2: Forward kinematics GUI provided by the "joint_state_publisher_gui" node

A demonstration of three positions of the simulated ReactorX-150 using the forward kinematics GUI can be seen in Figure 5.3. The first position being the “Home” position where all motor positions are set to 0.0. The two remaining positions are two arbitrary positions.



	Position 1	Position 2	Position 3
waist	0.00	2.28	-0.90
shoulder	-0.00	-0.33	0.42
elbow	-0.00	0.41	-0.07
wrist_angle	-0.00	-0.48	1.09
wrist_rotate	0.00	-2.87	-0.22
gripper	0.00	1.26	-1.71
left_finger	0.00	0.02	0.03

Figure 5.3: Arm Descriptions, ReactorX-150 in three different positions

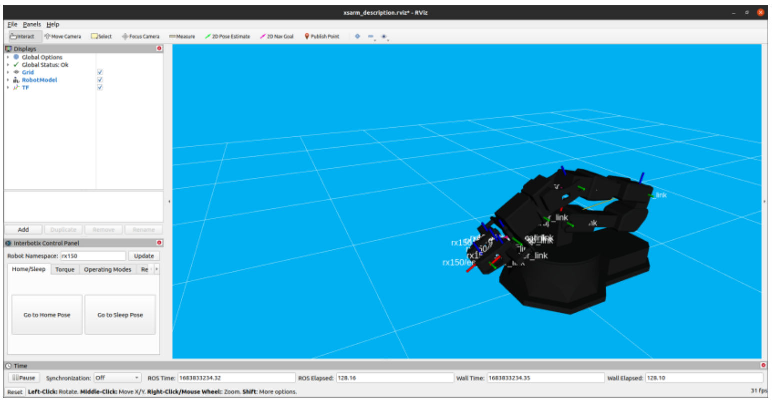
5.2 Arm Control

The Arm Control package has the ability to control the simulated ReactorX-150 arm in three ways. The first method of controlling the arm is through the “Interbotix Control Panel” in RViz, covered in chapter 5.2.1. The second method of controlling the arm is with the utilization of Python scripts, covered in chapter 5.2.2. The third method of controlling the robotic arm is with commands passed from the Ubuntu terminal window, covered in chapter 5.2.3.

The one thing the three control methods have in common is the startup of the Arm Control package. The following command launches the Arm Control package for the ReactorX-150 with RViz for visualization/simulation:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
use_sim:=true
```

The resulting RViz window with the model of the ReactorX-150 and the “Interbotix Control Panel” available can be seen in Figure 5.4.


xsarm_description.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Global Options
Global Status: Ok
Grid
RobotModel
TF
Add Duplicate Remove Rename
Interbotix Control Panel
Update
Home/Sleep Torque Operating Modes Re
Go to Home Pose
Go to Sleep Pose
Time
Pause
Reset
31 fps

Robot Namespace: rx150

Synchronization: Off ROS Time: 1683833234.32 ROS Elapsed: 128.16 Wall Time: 1683833234.35 Wall Elapsed: 128.10

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.

Figure 5.4: Arm Control RViz window

5.2.1 Interbotix Control Panel

The utilization of the “Interbotix Control Panel” requires the input of the codename of the arm in question. In this case “rx150” was written to the input field. The “Interbotix Control Panel” in RViz offers very limited control over the arm in terms of position of the arm, but it has a total of six tabs that offer control of a selection of other parameters. In terms of position control, the “Interbotix Control Panel” has the ability to set the arm in “Sleep” and “Home” position as seen in Figure 5.5.

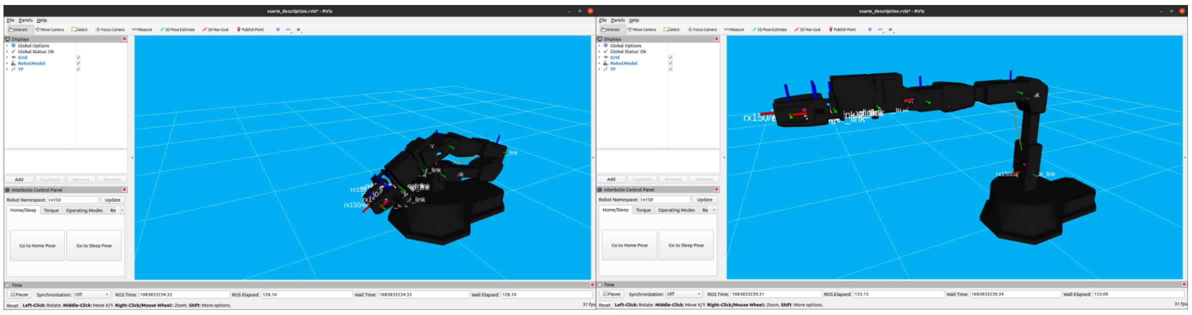


Figure 5.5: ReactorX-150 "Sleep" and "Home" pose from "Interbotix Control Panel"

There are six tabs in the “Interbotix Control Panel”. The five tabs except the “Home/Sleep” tab serve no practical use for the simulated model and will therefore be further covered in chapter 6.1.1.

5.2.2 Python Scripts

Python scripts provide a powerful method for controlling the simulated ReactorX-150. Utilizing the flexibility of Python, users can create complex control sequences, enabling precise manipulation of the robot's movements and interactions. Both forwards and inverse kinematics are available when controlled with python scripts.
The “bartender.py” Python script is located in the “Python demos” directory and can be seen in [Appendix J – bartender.py]. The series of figures seen in Figure 5.6 visualize the series of movements of the simulated ReactorX-150 executed when running the “bartender.py” script.
To run the “bartender.py” Python script, navigate to the “Python demos” directory and execute the following command:

```
$ python3 bartender.py
```

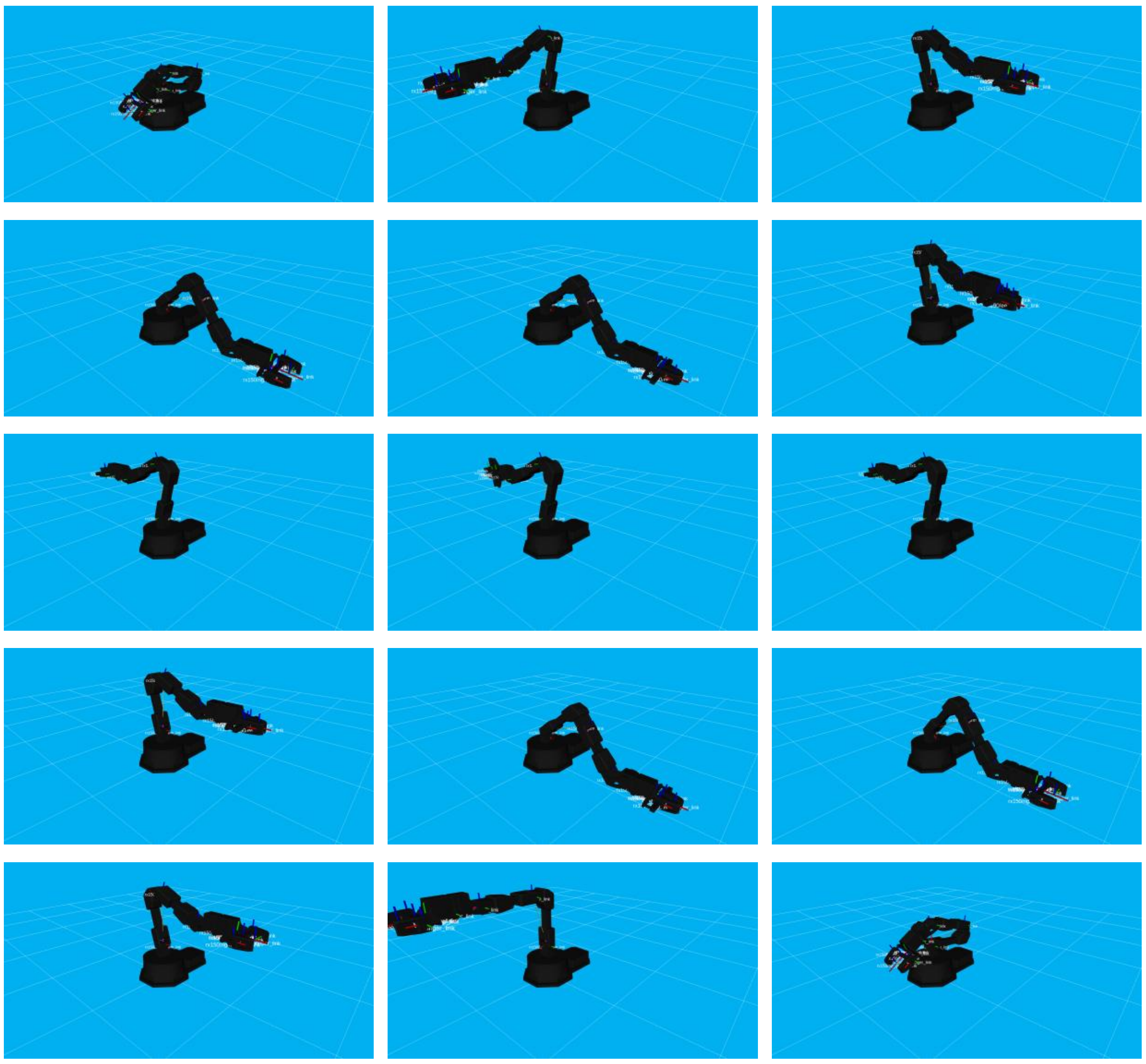
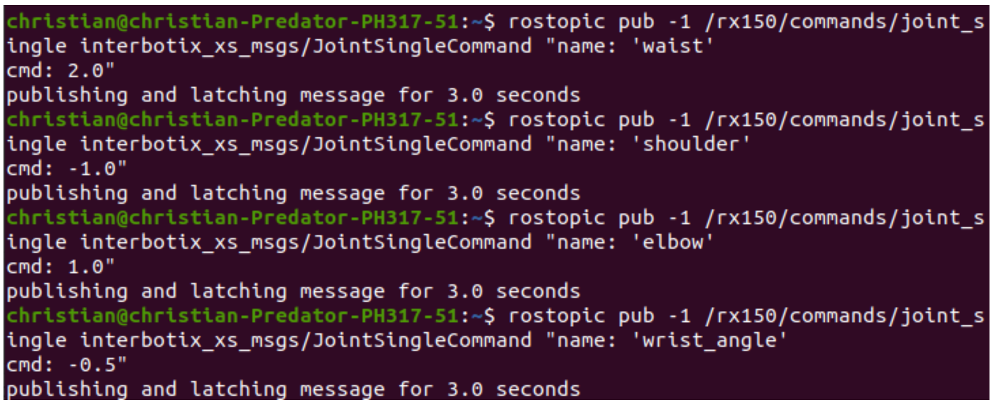


Figure 5.6: "bartender.py" script movements on simulated ReactorX-150

5.2.3 Terminal window commands

The Arm Control package enables control of the simulated arm through the Ubuntu terminal window with the help of "rostopics". The ReactorX-150 can be controlled by sending commands to "rostopic" subscribers. For a better understanding of "rostopic" commands see [Appendix K – "rostopic" guide].
Figure 5.7 shows four Ubuntu terminal commands controlling the simulated ReactorX-150.



```
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'waist'
cmd: 2.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'shoulder'
cmd: -1.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'elbow'
cmd: 1.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'wrist_angle'
cmd: -0.5"
publishing and latching message for 3.0 seconds
```

Figure 5.7: Four Ubuntu terminal commands for the simulated ReactorX-150

The visualization of the simulated ReactorX-150 executing the Ubuntu terminal commands can be seen in Figure 5.8, where the simulated robot arm starts in the “Home” position.

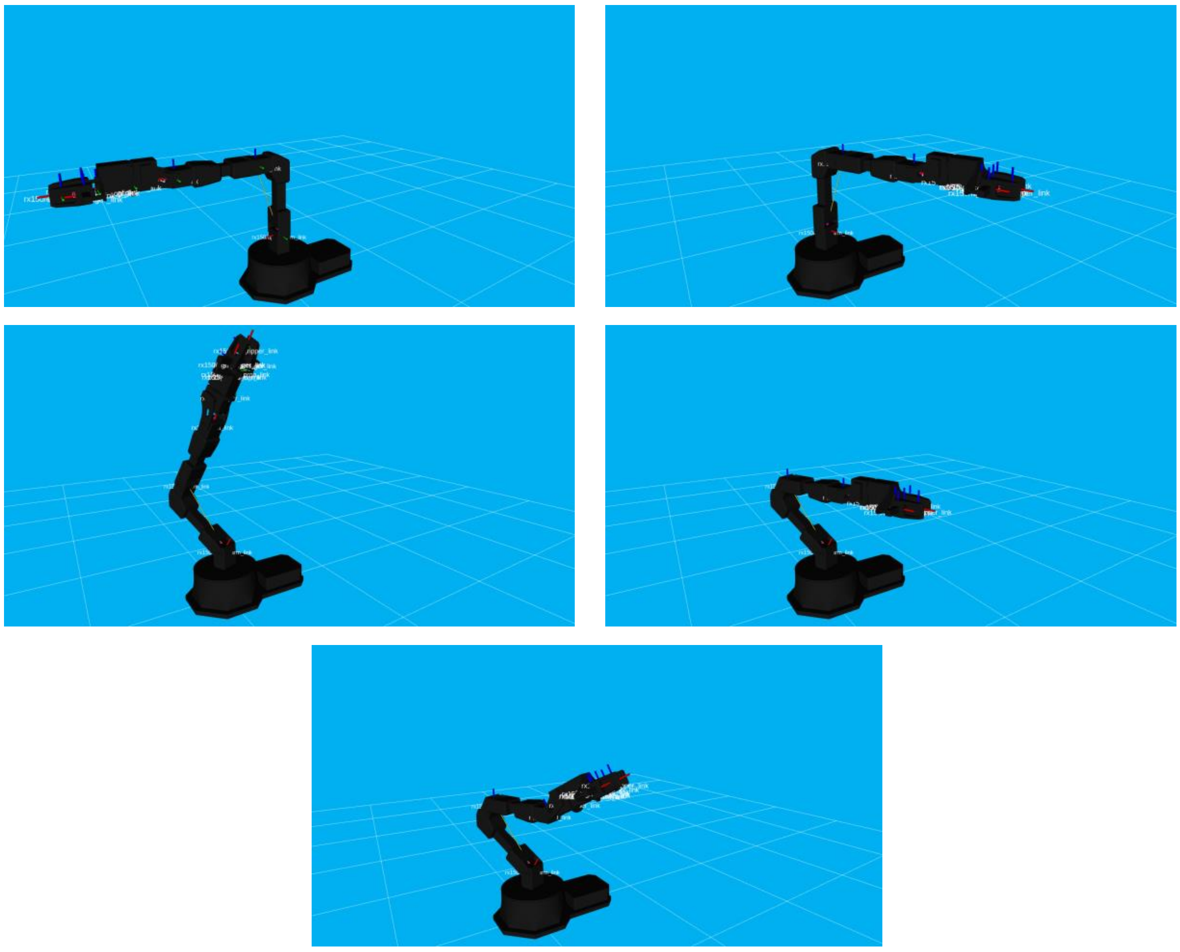


Figure 5.8: Visualization of Ubuntu terminal command to the simulated ReactorX-150

5.3 MoveIt

MoveIt enables the user to control the arm using both forwards and inverse kinematics. The following command launches the MoveIt package with the simulated ReactorX-150 model and RViz:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_fake:=true
```

The resulting RViz window with the MoveIt plugin and the model of the ReactorX-150 can be seen in Figure 5.9. “Query Goal State” shows the next goal position of the arm and end-effector, and “Trail Step Size” shows some of the positions of the arm on the way to the goal position. The “Query Goal State” is not checked by default but should always be checked like shown on both the figures for both forwards and inverse kinematics. The figure on the right has “Show Trail” checked and “Trail Step Size” set to seven.

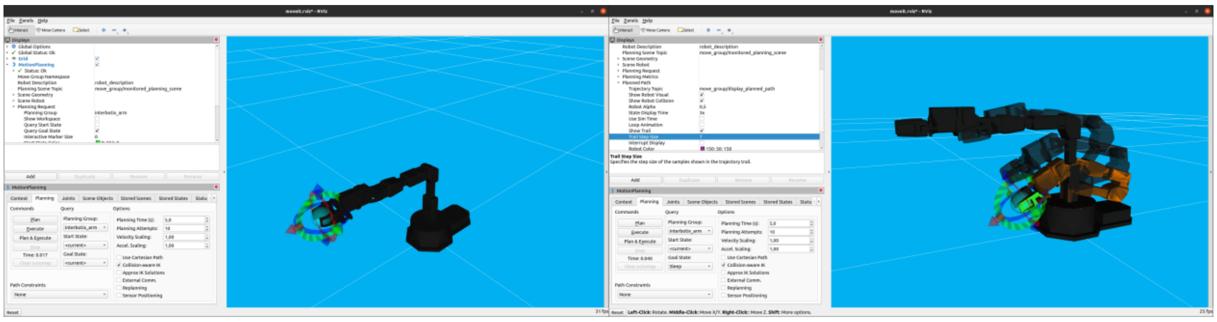


Figure 5.9: MoveIt RViz with "Query Goal State" and "Show Trail" checked

The following command launches the MoveIt Configuration package with the ReactorX-150 model, RViz and Gazebo:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_gazebo:=true
```

The resulting RViz window with the MoveIt plugin and Gazebo window, both with the model of the ReactorX-150 can be seen in Figure 5.10. As seen in Figure 5.9, The “Query Goal State” is checked.

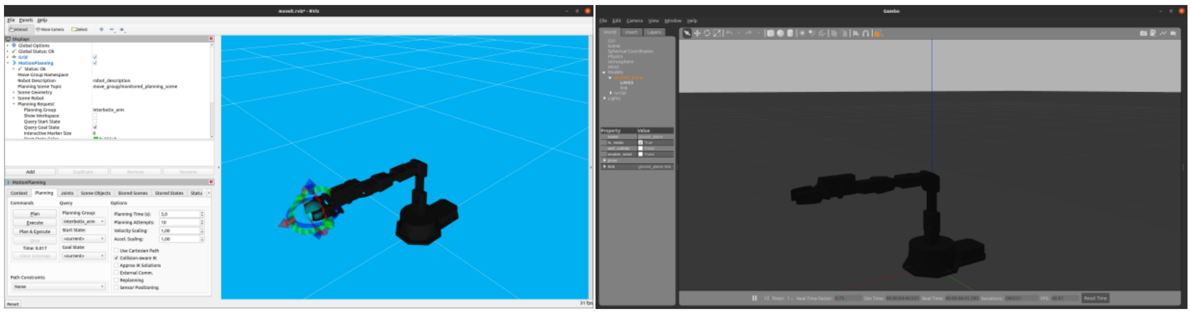


Figure 5.10: MoveIt RViz with "Query Goal State" and Gazebo

Figure 5.11 shows a few different positions of the simulated ReactorX-150 in RViz and Gazebo achieved with inverse kinematics.

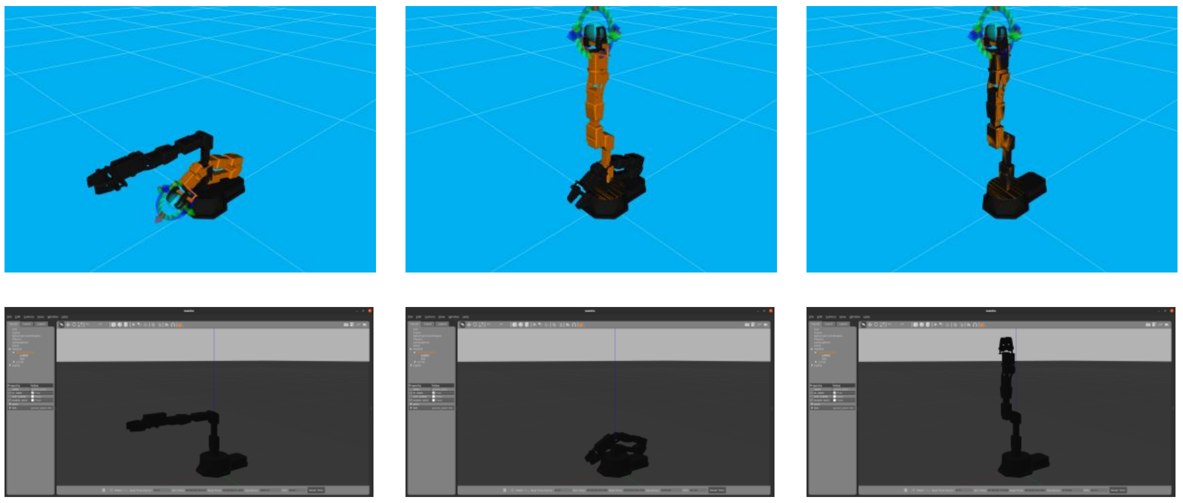


Figure 5.11: RViz and Gazebo ReactorX-150 positions with MoveIt

MoveIt can also be used for forward kinematics for the simulated robot arm by utilizing the “Joints” tab, as seen in Figure 5.12. Note that the position still has to be planned and executed from the “Planning” tab.

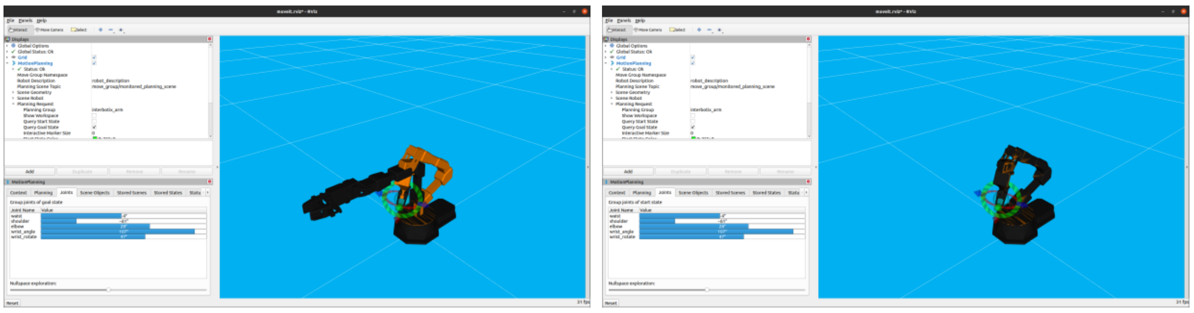


Figure 5.12: MoveIt forwards kinematics

5.4 Joystick Control

The Joystick Control package enables the control of the simulated ReactorX-150 to be done via a joystick controller. The following command launches the Joystick Control package with RViz and a simulated model of the ReactorX-150:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150 use_sim:=true
```

The Joystick Control package is not particularly suitable for demonstration with pictures, so a video made for this thesis demonstrating the capabilities of the simulated Joystick Control package can be viewed at [58]. The joystick controls can be seen in [Appendix G – Joystick controls].The video shows the control of the simulated ReactorX-150 arm with the PlayStation 4 joystick controller. The video demonstrates the robot arm capabilities such as:

- Commanding the robot arm to “Home” position and “Sleep” position using the “OPTIONS” and “SHARE” button, respectively.

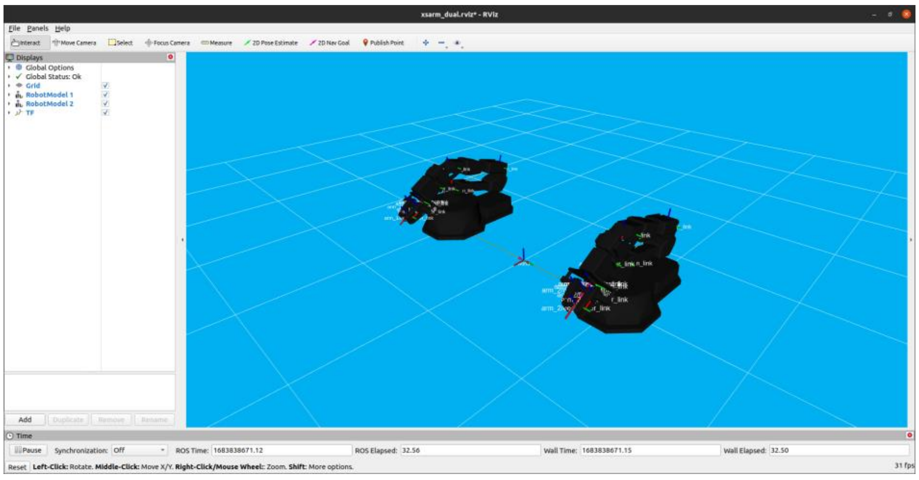
- Controlling the “waist” joint left and right using the “L2” and “R2” buttons, respectively.
- Commanding the opening and closing of the gripper with the circle and square buttons, respectively.
- Controlling the roll and tilt of the wrist of the arm by manipulating the right joystick left/right and up/down, respectively.
- Controlling the position of the end effector up/down and inwards/outwards by manipulating the left joystick up/down and left/right, respectively.

5.5 Dual Arm Control

The Dual Arm Control package enables the control of two ReactorX-150 arms simultaneously. The following command launches the Dual Arm Control package with two simulated ReactorX-150 arms in RViz:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_sim:=true use_dual_rviz:=true
```

The resulting window with the two simulated ReactorX-150 models can be seen in Figure 5.13. Unlike the Arm Control package, the Dual Arm Control package does not come with the “Interbotix Control Panel”. The arms can be controlled with Python scripts and terminal window commands covered in chapter 5.5.2 and 5.5.1, respectively.



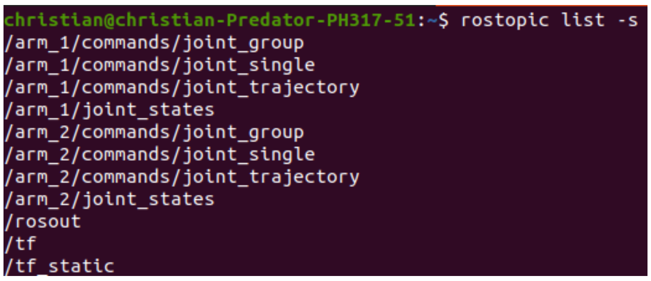
Synchronization: Off ROS Time: 1683838671.12 ROS Elapsed: 32.56 Wall Time: 1683838671.15 Wall Elapsed: 32.50

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Figure 5.13: Dual Arm Control RViz window

5.5.1 Terminal window commands

The terminal commands work the exact same way as seen in chapter 5.2.3. The only difference being the names of the topics, which can be seen in Figure 5.14. The arms are to be controlled independently, hence the separate “arm_1” and “arm_2” topics. For a better understanding of “rostopic” commands see [Appendix K – “rostopic” guide].



```
christian@christian-Predator-PH317-51:~$ rostopic list -s
/arm_1/commands/joint_group
/arm_1/commands/joint_single
/arm_1/commands/joint_trajectory
/arm_1/joint_states
/arm_2/commands/joint_group
/arm_2/commands/joint_single
/arm_2/commands/joint_trajectory
/arm_2/joint_states
/rosout
/tf
/tf_static
```

Figure 5.14: Dual Arm Control package topics

5.5.2 Python Scripts

The following two commands executes two scripts intended for picking up a 3D printed USN logo that can be seen in chapter 7.6. The two custom scripts below can be seen in [Appendix L – xsarm_dual_usn_lift.py] and [Appendix M – xsarm_dual_usn_down.py]. The first script lifts the object up, the second script lowers the object back down:

```
$ python3 xsarm_dual_usn_lift.py
$ python3 xsarm_dual_usn_down.py
```

The two scripts above are visualized in a sequence in Figure 5.15, where the first script is represented by the five first frames and the second script is represented by the four remaining frames.

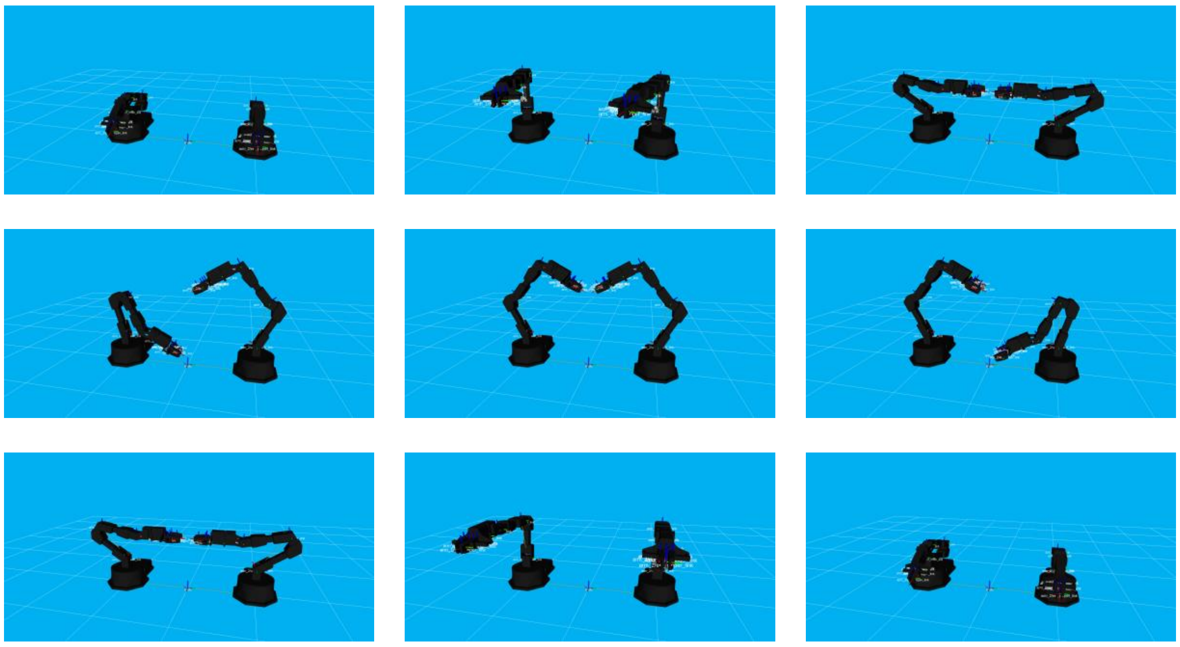


Figure 5.15: "python3 xsarm_dual_usn_lift.py" and "python3 xsarm_dual_usn_lift.py" visualized

5.6 Dual Arm Joystick Control

The Dual Arm Joystick Control package enables the control of two ReactorX-150’s to be done via one joystick controller. The following command launches the Dual Arm Joystick control package with RViz and the simulated models of the ReactorX-150:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150 use_sim:=true
```

The resulting window is a regular RViz window with two simulated models of the ReactorX-150, exactly like the RViz window from the launch of chapter 5.5

The Joystick Control package is not particularly suitable for demonstration with pictures, so a video made for this thesis demonstrating the capabilities of the simulated Joystick Control package can be viewed at [59]. The joystick controls can be seen in [Appendix G – Joystick controls].The video shows the control of the two simulated ReactorX-150 arms controlled simlotaniously with the PlayStation 4 joystick controller. The video demonstrates the robot arms capabilities such as:

- Commanding the robot arms to “Home” position and “Sleep” position using the “OPTIONS” and “SHARE” button, respectively.
- Controlling the “waist” joints left and right using the “L2” and “R2” buttons, respectively.
- Commanding the opening and closing of the grippers with the circle and square buttons, respectively.
- Controlling the roll and tilt of the arms wrists by manipulating the right joystick left/right and up/down, respectively.
- Controlling the position of the end effectors up/down and inwards/outwards by manipulating the left joystick up/down and left/right, respectively.

6 Physical Testing of ReactorX-150

This chapter presents the physical testing of the ReactorX-150 robotic arm. This chapter looks into the application of the various packages that were used to control and manipulate the ReactorX-150, including their respective setup and configuration.

The user manual for the X-series arms with the main arguments for the packages can be seen in [Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms]. The more extensive user manual containing the full list of arguments for the packages can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

All commands throughout this chapter are meant to be run in an Ubuntu terminal window.

6.1 Arm Control

The Arm Control package has the ability to control the physical ReactorX-150 arm in three ways. The first method of controlling the arm is through the “Interbotix Control Panel” in RViz, covered in chapter 6.1.1. The second method of controlling the arm is with the utilization of Python scripts, covered in chapter 6.1.2. The third method of controlling the robotic arm is with commands passed from the Ubuntu terminal window, covered in chapter 6.1.3.

The one thing the three control methods have in common is the startup of the Arm Control package. The following command launches the Arm Control package for the physical ReactorX-150 with RViz:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
```

The resulting RViz window with the model of the ReactorX-150 with the “Interbotix Control Panel” available and the physical ReactorX-150 can be seen in Figure 6.1.

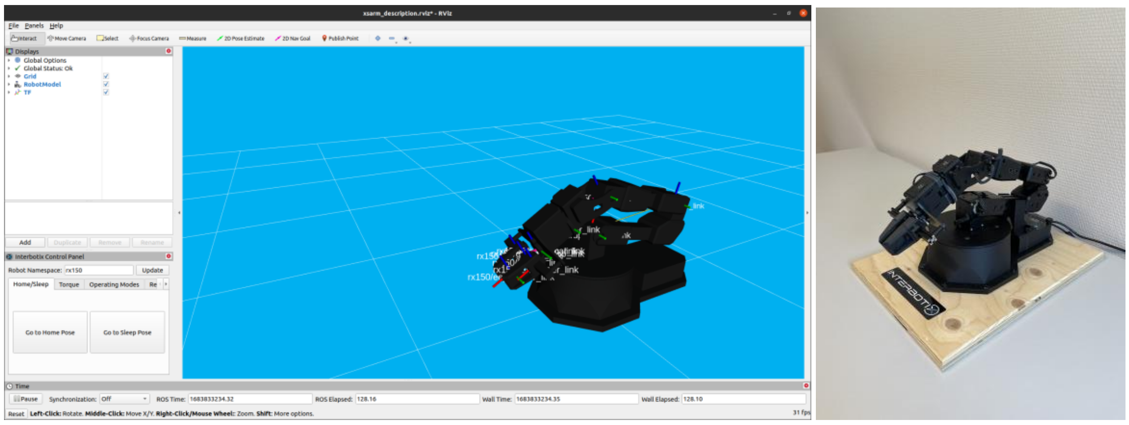


Figure 6.1: Arm Control RViz window with physical ReactorX-150 in “Sleep” position

6.1.1 Interbotix Control Panel

The utilization of the “Interbotix Control Panel” requires the input of the codename of the arm in question. In this case “rx150” was written to the input field. The “Interbotix Control

Panel” in RViz offers very limited control over the arm in terms of position of the arm, but it has a total of six tabs that offer control of a selection of other parameters. In terms of position control, the “Interbotix Control Panel” has the ability to set the arm in “Sleep” and “Home” position as seen in Figure 6.1and Figure 6.2, respectively.

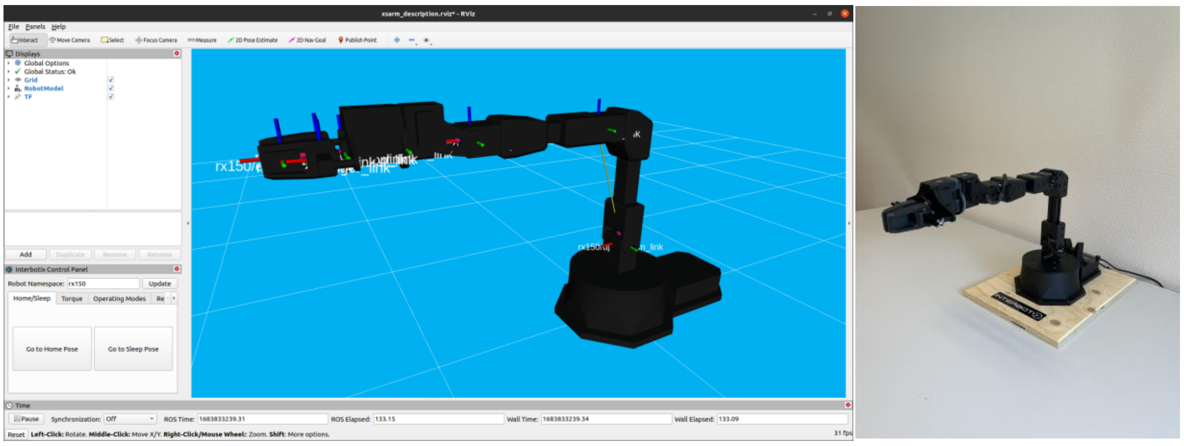
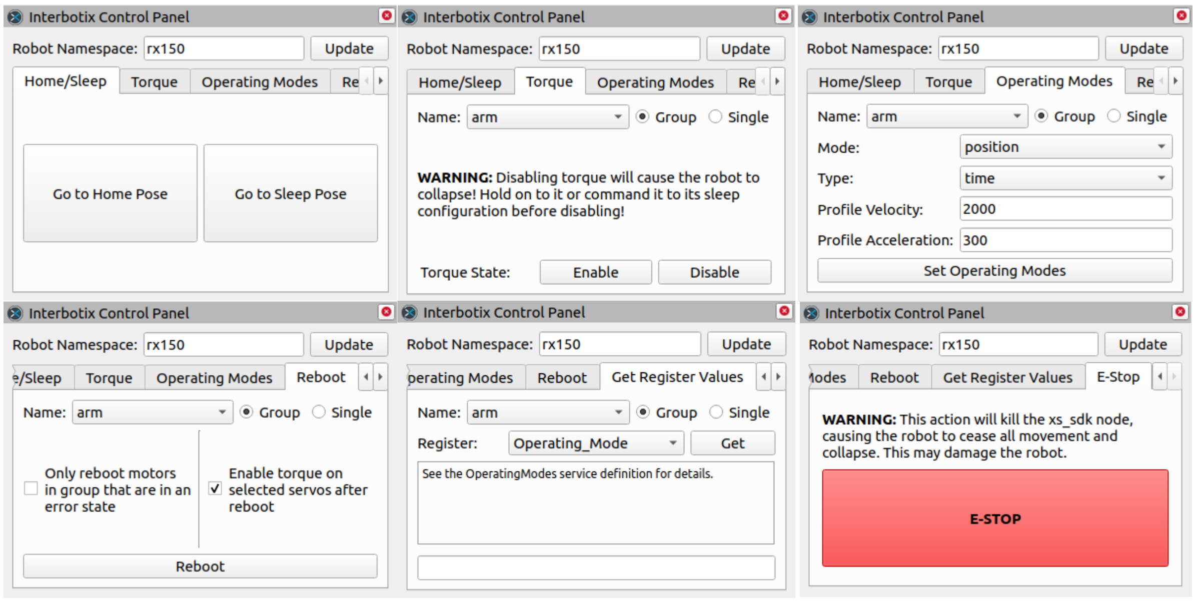


Figure 6.2: Arm Control RViz window with physical ReactorX-150 in "Home" position

The six tabs available in the “Interbotix Control Panel” can be seen in Figure 6.3.The main abilities of the different tabs are:

- **“Home/Sleep”:** Has the ability to command the arm to “Home” or “Sleep” position.
- **“Torque”:** Has the ability to torque the motor groups or the individual motors of the robotic arm on and off.
- **“Operating Modes”:** Has the ability to configure the operating modes of the motor groups or the individual motors of the robotic arm.
- **“Reboot”:** Has the ability to reboot the motor groups or the individual motors of the robotic arm.
- **“Get Register Values”:** Has the ability to display the register values of the motor groups or the individual motors of the robotic arm.
- **“E-Stop”:** Has the ability to kill the “xs_sdk” node, causing an immediate torque-off in all the motors. Acts as an emergency stop button, and damage of the robot arm may ensue.



Robot Namespace: rx150

Name: arm ● Group ○ Single

WARNING: Disabling torque will cause the robot to collapse! Hold on to it or command it to its sleep configuration before disabling!

Torque State:

Mode: position

Type: time

Profile Velocity: 2000

Profile Acceleration: 300

☐ Only reboot motors in group that are in an error state

✓ Enable torque on selected servos after reboot

Register: Operating_Mode

See the OperatingModes service definition for details.

WARNING: This action will kill the xs_sdk node, causing the robot to cease all movement and collapse. This may damage the robot.

Figure 6.3: The six tabs of the "Interbotix Control Panel"

6.1.2 Python Scripts

Python scripts provide a powerful method for controlling the physical ReactorX-150. Utilizing the flexibility of Python, users can create complex control sequences, enabling precise manipulation of the robot's movements and interactions. Both forwards and inverse kinematics are available when controlled with python scripts.
The “bartender.py” Python script is located in the “Python demos” directory and can be seen in [Appendix J – bartender.py]. The series of figures seen in Figure 6.4 visualize the series of movements of the physical ReactorX-150 executed when running the “bartender.py” script.
To run the “bartender.py” Python script, navigate to the “Python demos” directory and execute the following command:

```
$ python3 bartender.py
```

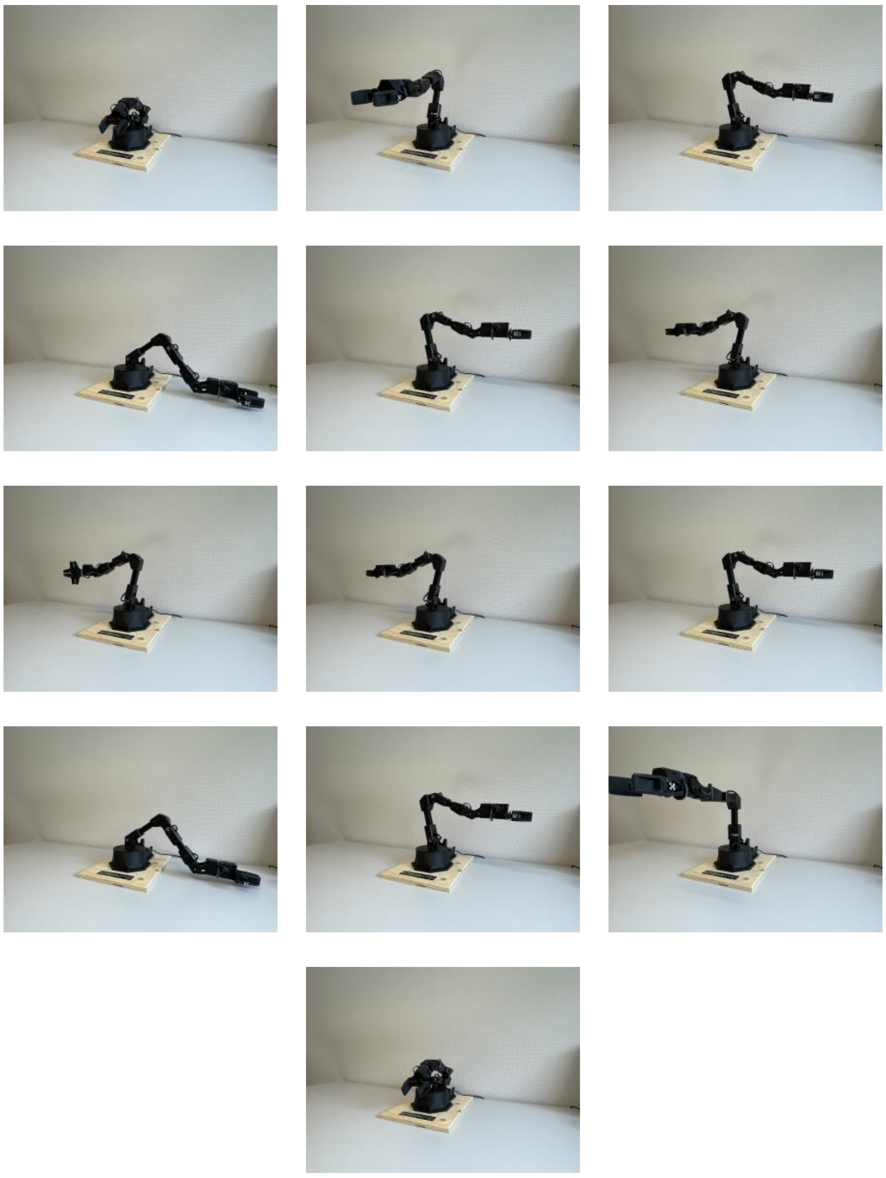
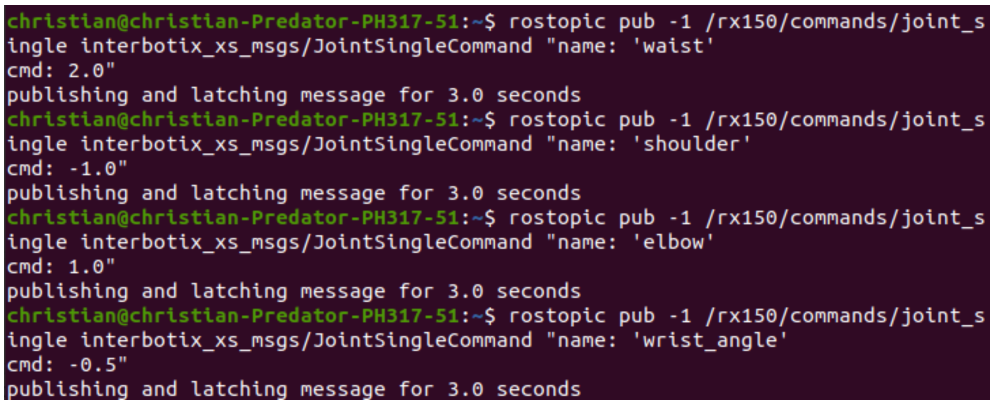


Figure 6.4: "bartender.py" script movements on physical ReactorX-150

6.1.3 Terminal window commands

The Arm Control package enables control of the arm through the Ubuntu terminal window with the help of “rostopics”. The ReactorX-150 can be controlled by sending commands to “rostopic” subscribers. For a better understanding of “rostopic” commands see [Appendix K – “rostopic” guide].

Figure 6.5 shows four Ubuntu terminal commands controlling the physical ReactorX-150.



```
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'waist'
cmd: 2.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'shoulder'
cmd: -1.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'elbow'
cmd: 1.0"
publishing and latching message for 3.0 seconds
christian@christian-Predator-PH317-51:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: 'wrist_angle'
cmd: -0.5"
publishing and latching message for 3.0 seconds
```

Figure 6.5: Four Ubuntu terminal commands for the physical ReactorX-150

The execution of the Ubuntu terminal commands by the physical ReactorX-150 can be seen in Figure 6.6, where the physical robot arm starts in the “Home” position.

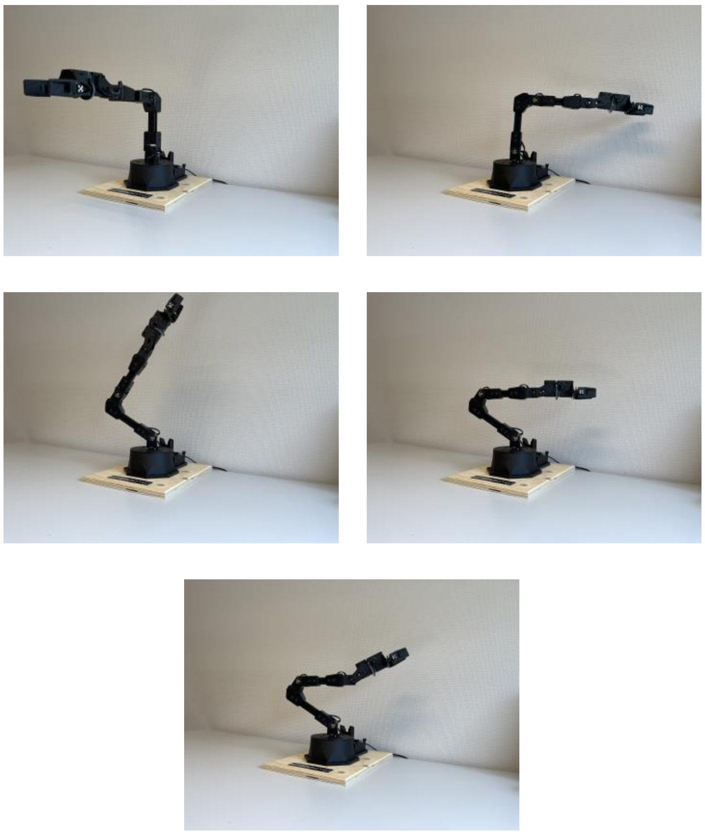


Figure 6.6: The physical ReactorX-150 executing Ubuntu terminal commands

6.2 MoveIt

MoveIt enables the user to control the arm using both forwards and inverse kinematics. The following command launches the MoveIt package with the simulated ReactorX-150 model and RViz:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_actual:=true
```

The resulting RViz window with the MoveIt plugin and the model of the ReactorX-150 together with the physical ReactorX-150 can be seen in Figure 6.7, going from “Sleep” positions to two other positions. “Query Goal State” shows the next goal position of the arm and end-effector, and “Trail Step Size” shows some of the positions of the arm on the way to the goal position. The “Query Goal State” and “Show Trail” are not checked by default but should always be checked for both forwards and inverse kinematics. “Trail Step Size” were in this case set to seven.

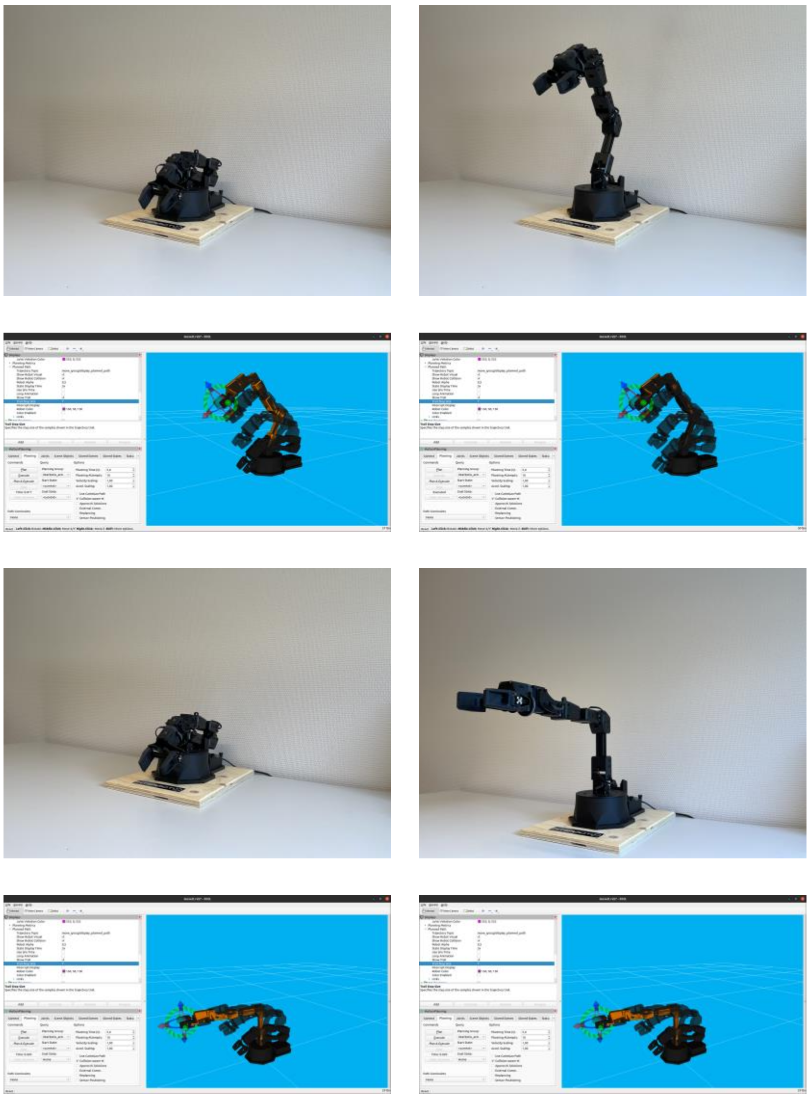


Figure 6.7: Physical ReactorX-150 visualized with RViz, moving from "Sleep" position

MoveIt can also be used for forward kinematics for the physical robot arm by utilizing the “Joints” tab as seen in Figure 6.8. Note that the position still must be planned and executed from the “Planning” tab.

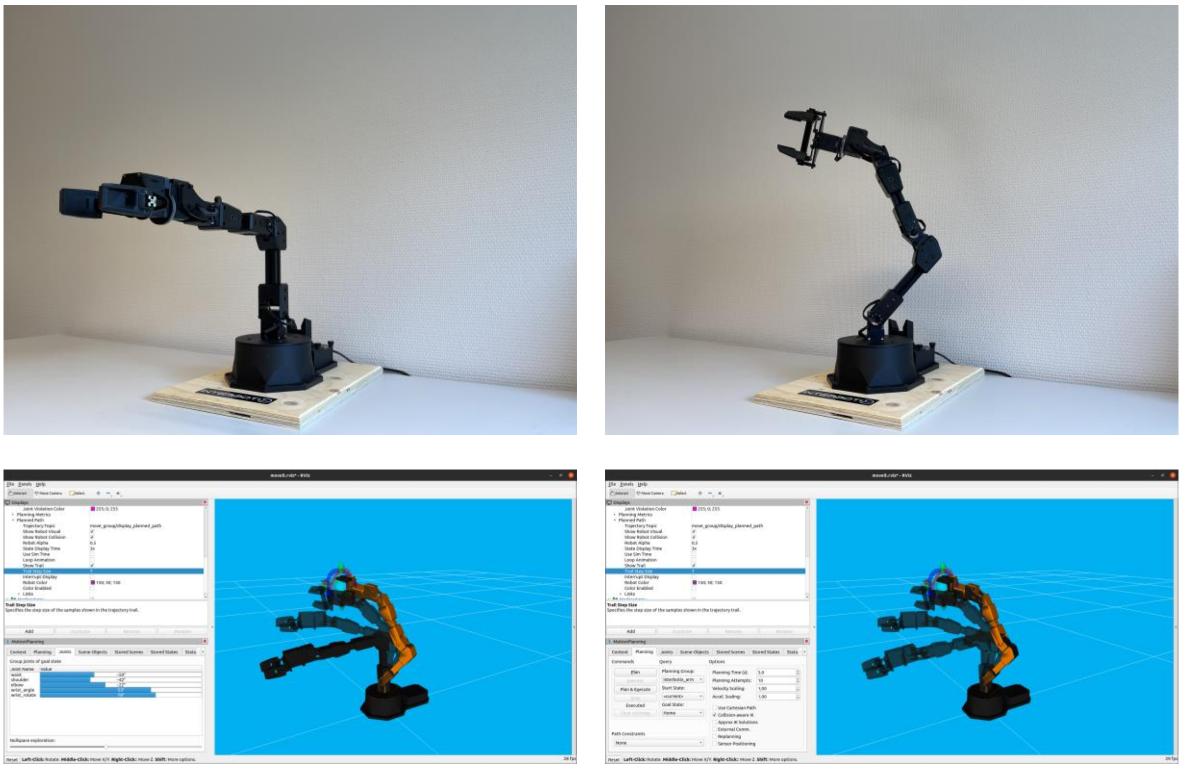


Figure 6.8: Forward kinematics performed on the physical ReactorX-150

6.3 Joystick Control

The Joystick Control package enables the control of the physical ReactorX-150 to be done via a joystick controller. The following command launches the Joystick Control package with RViz and enables control of the physical ReactorX-150:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150 use_sim:=true
```

The Joystick Control package is not particularly suitable for demonstration with pictures, so a video made for this thesis demonstrating the capabilities of the physical Joystick Control package can be viewed at [60]. The joystick controls can be seen in [Appendix G – Joystick controls].The video shows the control of the physical ReactorX-150 arm with the PlayStation 4 joystick controller. The video demonstrates the robot arm capabilities such as:

- Commanding the robot arm to “Home” position and “Sleep” position using the “OPTIONS” and “SHARE” button, respectively.
- Controlling the “waist” joint left and right using the “L2” and “R2” buttons, respectively.
- Commanding the opening and closing of the gripper with the circle and square buttons, respectively.
- Controlling the roll and tilt of the wrist of the arm by manipulating the right joystick left/right and up/down, respectively.
- Controlling the position of the end effector up/down and inwards/outwards by manipulating the left joystick up/down and left/right, respectively.

6.4 Record and Playback

The Record and Playback package enables the user to record the manual manipulation of the physical ReactorX-150 robot arm and play it back at any time. The following command launches the Record and Playback package with RViz and records the manual manipulation of the physical ReactorX-150:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
record:=true
```

The recording of the manual manipulation of the physical ReactorX-150 can be seen in Figure 6.9.

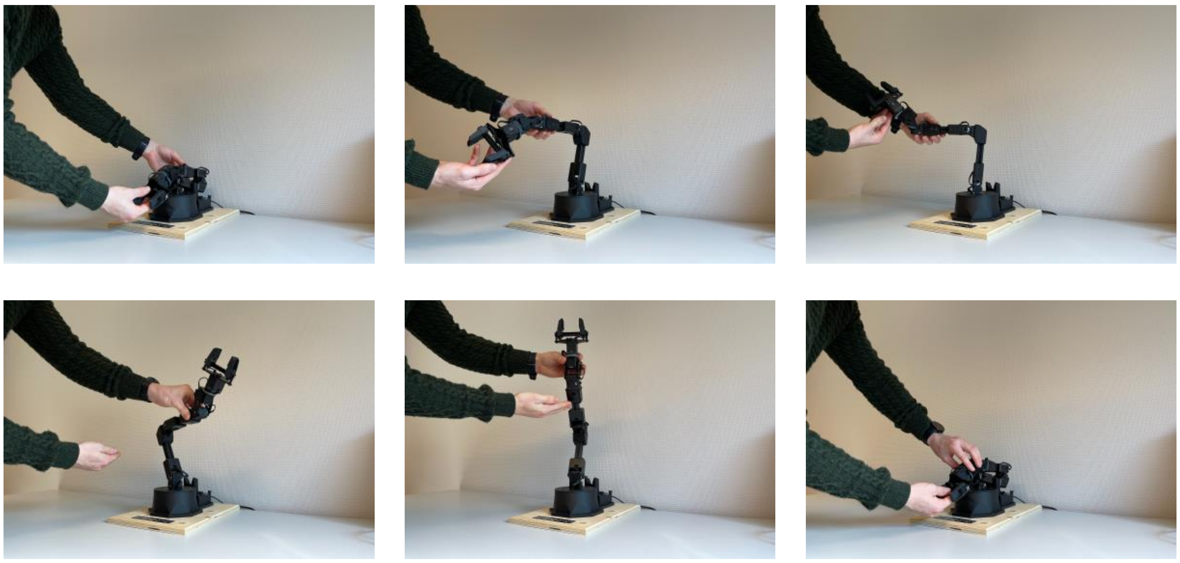


Figure 6.9: Recording of the manual manipulation of the ReactorX-150

The following command launches the Record and Playback package with RViz and playback the manual manipulation of the physical ReactorX-150:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
playback:=true
```

The recording of the manual manipulation of the physical ReactorX-150 can be seen in Figure 6.10.

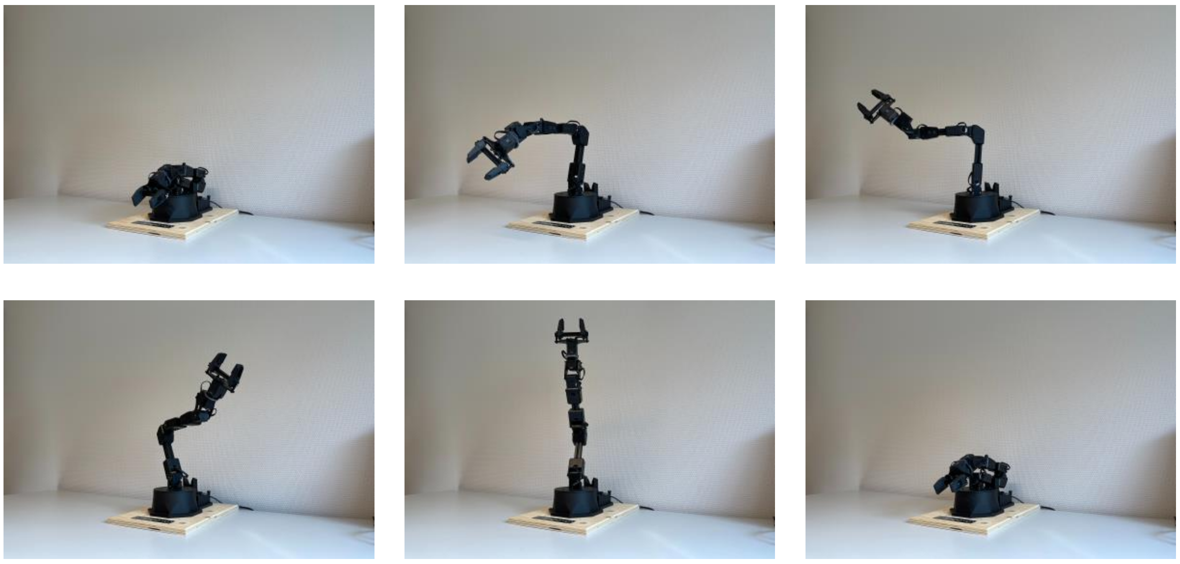


Figure 6.10: Playback of the manual manipulation of the ReactorX-150

6.5 Arm Diagnostic Tool

The Arm Diagnostic Tool package enables the activation and observation of the same or different joints. The procedure for the Arm Diagnostic Tool can be seen in Figure 6.11. The first frame displays the manual manipulation of the robot to a desired position for activation and observation of the “waist” joint. The next for frames show a front and top view of how far each direction the “waist” joint is tested. After the given test duration, the arm return to the “Sleep” position as shown in the final frame.

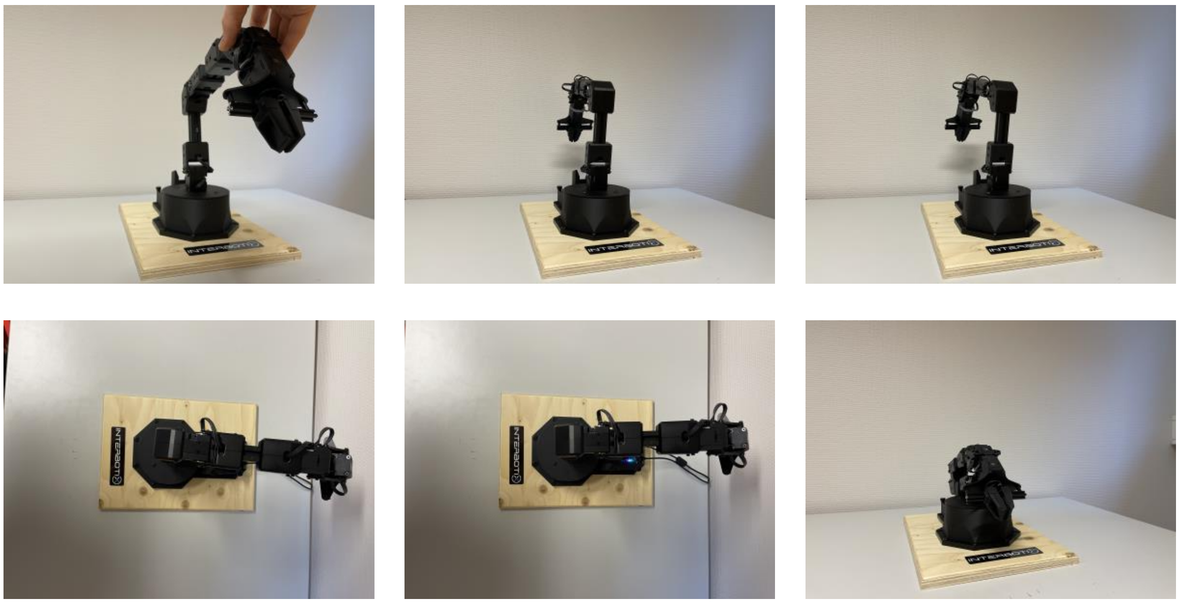


Figure 6.11: Arm Diagnostic tool procedure

The following command launches the Arm Diagnostic Tool package for activation and observation of the “waist” joint for 60 seconds:

```
$ roslaunch interbotix_xsarm_diagnostic_tool xsarm_diagnostic_tool.launch
robot_model:=rx150 cmd_joint:=waist observe_joint:=waist test_duration:=60
```

The results of the test can be seen Figure 6.12 as three rqt plots. The information displayed in the three plots (y-axis vs. x-axis):

- Position [rad] and velocity [rad/s] vs. time [s]
- Temperature [°C] vs. time [s]
- Effort [mA] vs time [s]

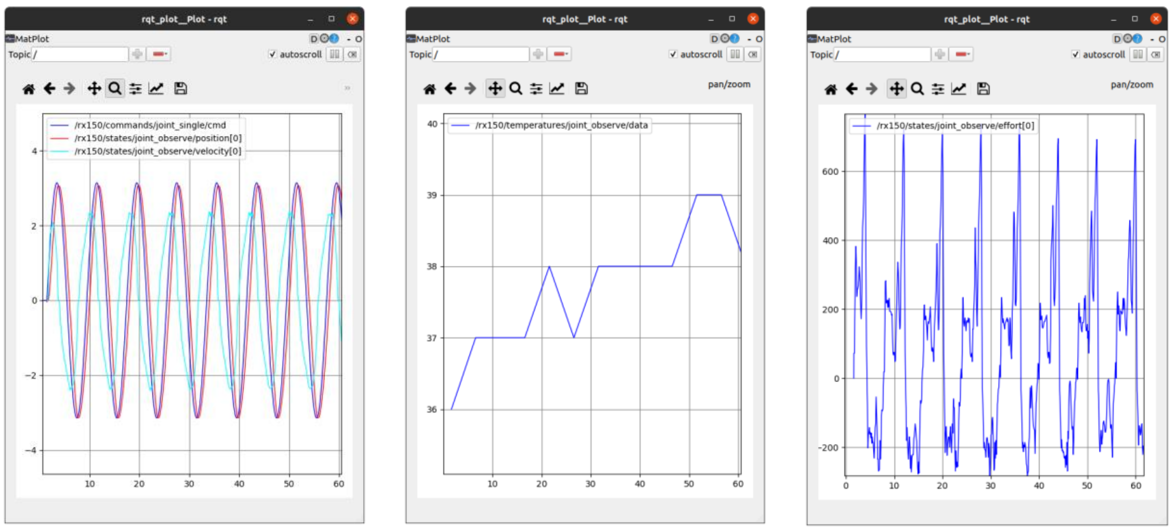

rqt_plot__Plot - rqt
MatPlot
Topic /
autoscroll
/rx150/commands/joint_single/cmd
/rx150/states/joint_observe/position[0]
/rx150/states/joint_observe/velocity[0]
pan/zoom
/rx150/temperatures/joint_observe/data
/rx150/states/joint_observe/effort[0]

Figure 6.12: Arm Diagnostic tool results

6.6 Arm Puppeteering

The Arm Puppeteering package enables the manipulation of one ReactorX-150 to be repeated in real time by a second X-Series arm, in this case a second ReactorX-150. The following command launches the Arm Puppeteering package with two ReactorX-150s and RViz displaying them both:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet.launch robot_model_master:=rx150
robot_model_puppet:=rx150
```

The designation of the "master" and "puppet" arm is determined by the sequence of connection to the computer. Specifically, the arm that is connected first assumes the role of the "master" arm.

The puppeteering of the two ReactorX-150 arms can be seen in Figure 6.13.

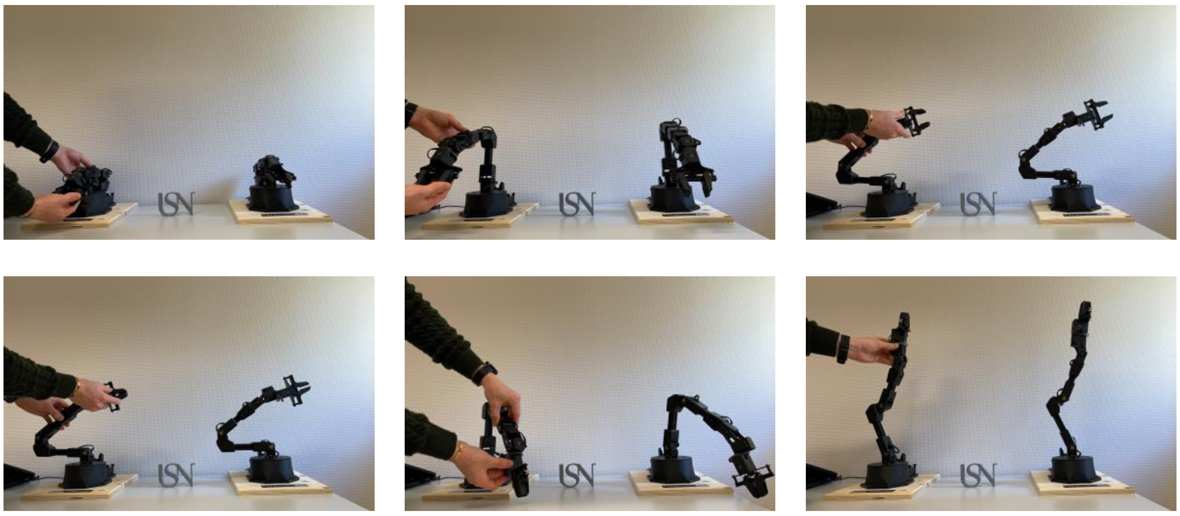


Figure 6.13: Puppeteering with two ReactorX-150s

6.7 Dual Arm Control

The Dual Arm Control package enables the control of two ReactorX-150 arms simultaneously. The following command launches the Dual Arm Control package with two physical ReactorX-150 arms together with RViz:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_dual_rviz:=true
```

The resulting window with the two ReactorX-150 models in RViz and the two physical ReactorX-150 arms can be seen in Figure 6.14. Unlike the Arm Control package, the Dual Arm Control package does not come with the “Interbotix Control Panel”. The arms can be controlled with Python scripts and terminal window commands covered in chapter 6.7.2 and 6.7.1, respectively.

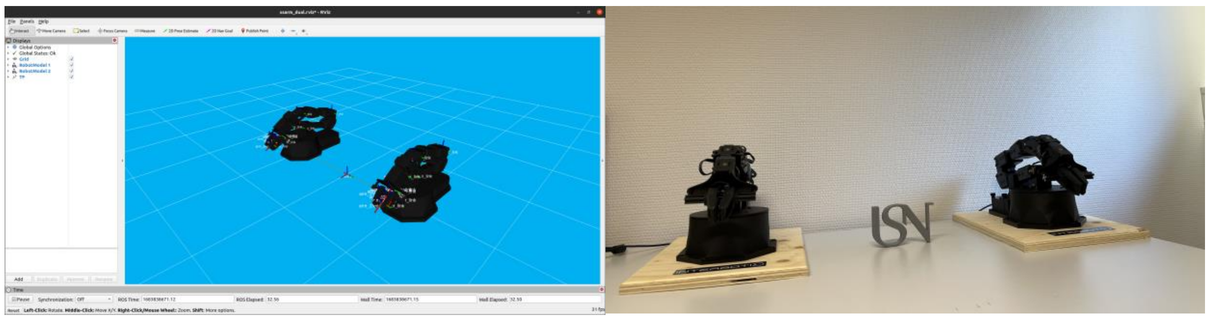
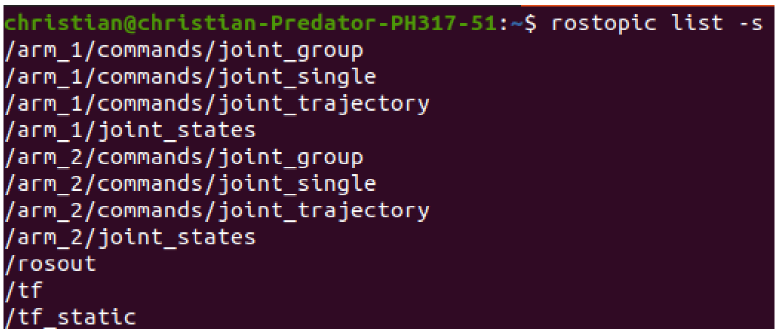


Figure 6.14: Dual Arm Control RViz window and physical ReactorX-150 arms

6.7.1 Terminal window commands

The terminal commands for the Dual Arm Control package work the exact same way as seen in chapter 6.1.3. The only difference being the names of the topics, which can be seen in Figure

6.15. The arms are to be controlled independently, hence the separate “arm_1” and “arm_2” topics. For a better understanding of “rostopic” commands see [Appendix K – “rostopic” guide].



```
christian@christian-Predator-PH317-51:~$ rostopic list -s
/arm_1/commands/joint_group
/arm_1/commands/joint_single
/arm_1/commands/joint_trajectory
/arm_1/joint_states
/arm_2/commands/joint_group
/arm_2/commands/joint_single
/arm_2/commands/joint_trajectory
/arm_2/joint_states
/rosout
/tf
/tf_static
```

Figure 6.15: Dual Arm Control package topics

6.7.2 Python Scripts

The following two commands executes two scripts intended for picking up a 3D printed USN logo that can be seen in chapter 7.6. The two custom scripts below can be seen in [Appendix L – xsarm_dual_usn_lift.py] and [Appendix M – xsarm_dual_usn_down.py]. It is mentioned in the scripts, but the values for running the physical ReactorX-150 with 60 cm from center to center are utilized. The first script lifts the object up, the second script lowers the object back down:

```
$ python3 xsarm_dual_usn_lift.py
$ python3 xsarm_dual_usn_down.py
```

The execution of two scripts above can be seen in Figure 6.16, where the first script is represented by the five first frames and the second script is represented by the four remaining frames.

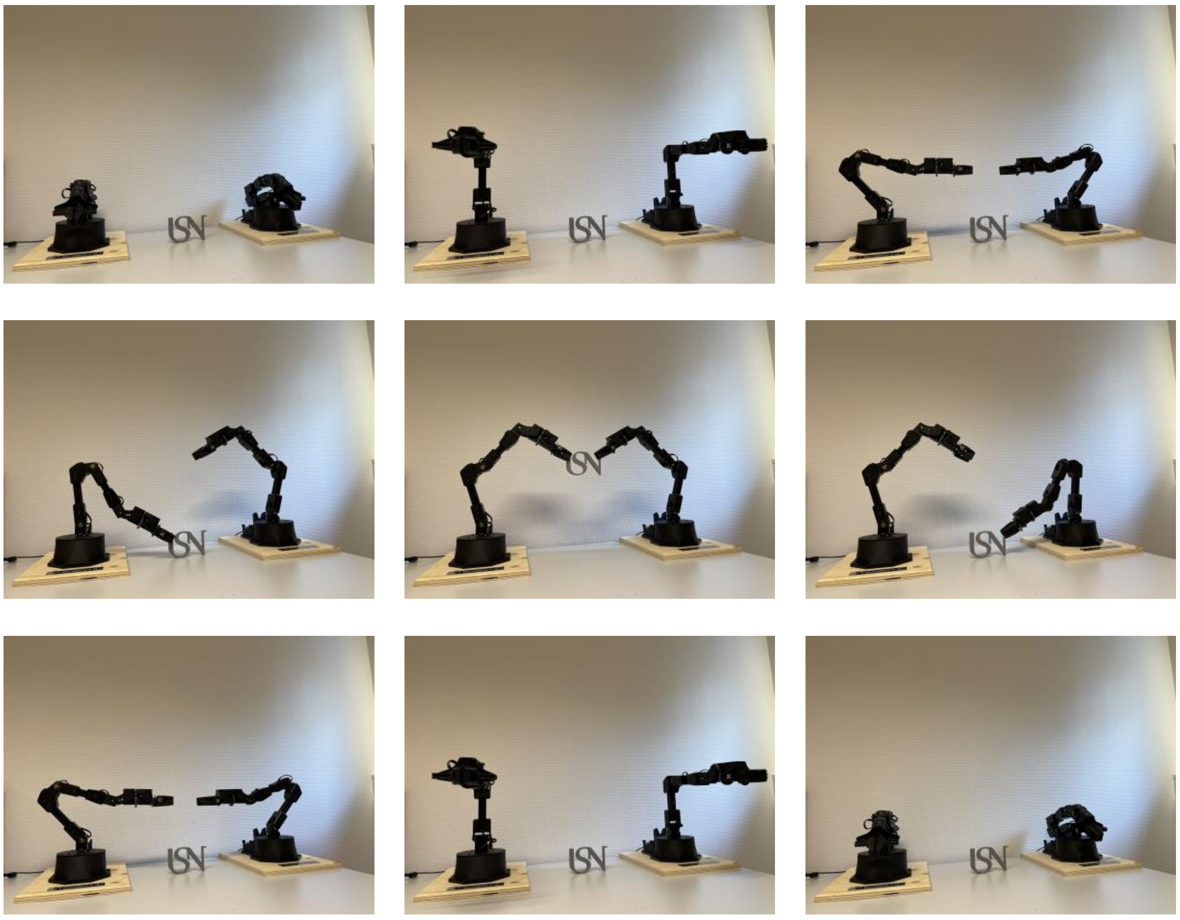


Figure 6.16: "python3 xsarm_dual_usn_lift.py" and "python3 xsarm_dual_usn_lift.py" executed by the physical ReactorX-150 arms with the USN logo

6.8 Dual Arm Joystick Control

The Dual Arm Joystick Control package enables the control of two ReactorX-150’s to be done via one joystick controller. The following command launches the Dual Arm Joystick control package with RViz and physical ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150
```

The resulting window is a regular RViz window with two simulated models of the ReactorX-150, exactly like the RViz window from the launch of chapter 6.7.

The Joystick Control package is not particularly suitable for demonstration with pictures, so a video made for this thesis demonstrating the capabilities of the simulated Joystick Control package can be viewed at [61]. The joystick controls can be seen in [Appendix G – Joystick controls].The video shows the control of the two physical ReactorX-150 arms controlled simlotaniously with the PlayStation 4 joystick controller. The video demonstrates the robot arms capabilities such as:

- Commanding the robot arms to “Home” position and “Sleep” position using the “OPTIONS” and “SHARE” button, respectively.

- Controlling the “waist” joints left and right using the “L2” and “R2” buttons, respectively.
- Commanding the opening and closing of the grippers with the circle and square buttons, respectively.
- Controlling the roll and tilt of the arms wrists by manipulating the right joystick left/right and up/down, respectively.
- Controlling the position of the end effectors up/down and inwards/outwards by manipulating the left joystick up/down and left/right, respectively.

7 Discussion

The discussion chapter of this thesis presents an analysis of some parts of the development and testing of the simulation platform for the ReactorX-150 robotic arm manipulator. This chapter aims to provide insight into aspects of the thesis not included in the specifications or simulation and testing of the ReactorX-150. The discussion will also identify potential limitations and challenges encountered during the study and propose future work.

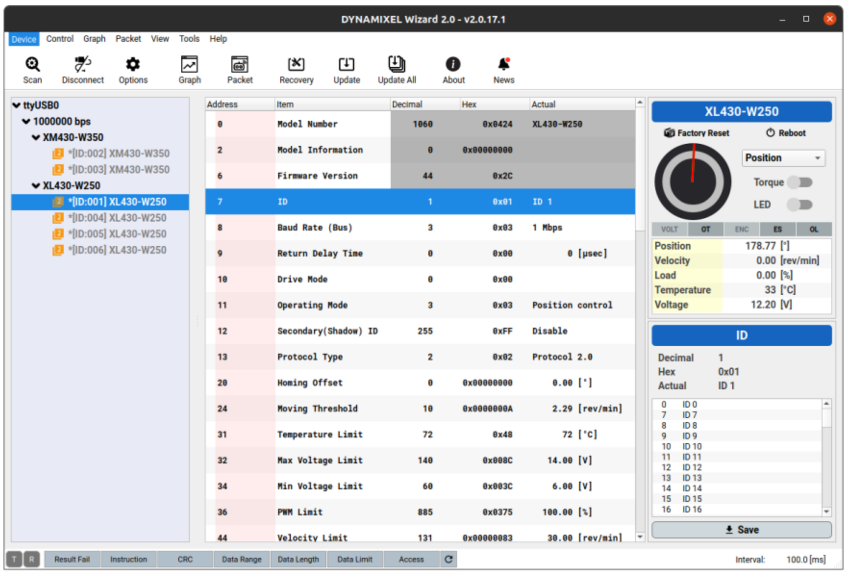
7.1 Motor ID correction

When testing the two provided ReactorX-150 arms a number of error message occurred when testing one of the arms. The error messages appeared for each of the joints (ID 1-6), it was roughly as follows:

"[xs_sdk] Can't find DYNAMIXEL ID '1', Joint Name: 'waist'"

The error message indicated that the system could not locate the motor with the ID '1', which was crucial for the proper functioning of the 'waist' joint.

It turned out that the preset motor IDs were configured from 7 to 12, but the packages were designed to recognize and interact with motor IDs ranging from 1 to 6. To resolve this issue and ensure compatibility between the motor IDs and the software packages, a modification to the motor IDs was implemented using the DYNAMIXEL Wizard 2.0. Figure 7.1 shows the DYNAMIXEL Wizard 2.0 window with the motor ID selected. The motor ID was changed by selecting the available IDs in the bottom right corner and saving it to the motor.



Address	Item	Decimal	Hex	Actual
0	Model Number	1060	0x0424	XL430-W250
2	Model Information	0	0x00000000	
6	Firmware Version	44	0x2C	
7	ID	1	0x01	ID 1
8	Baud Rate (Bus)	3	0x03	1 Mbps
9	Return Delay Time	0	0x00	0 [μsec]
10	Drive Mode	0	0x00	
11	Operating Mode	3	0x03	Position control
12	Secondary(Shadow) ID	255	0xFF	Disable
13	Protocol Type	2	0x02	Protocol 2.0
20	Homing Offset	0	0x00000000	0.00 [°]
24	Moving Threshold	10	0x0000000A	2.29 [rev/min]
31	Temperature Limit	72	0x48	72 [°C]
32	Max Voltage Limit	140	0x008C	14.00 [V]
34	Min Voltage Limit	60	0x003C	6.00 [V]
36	PWM Limit	885	0x0375	100.00 [%]
44	Velocity Limit	131	0x00000083	30.00 [rev/min]

Position	178.77 [°]
Velocity	0.00 [rev/min]
Load	0.00 [%]
Temperature	33 [°C]
Voltage	12.20 [V]

ID

Decimal	1
Hex	0x01
Actual	ID 1

0	ID 0
7	ID 7
8	ID 8
9	ID 9
10	ID 10
11	ID 11
12	ID 12
13	ID 13
14	ID 14
15	ID 15
16	ID 16

Interval: 100.0 [ms]

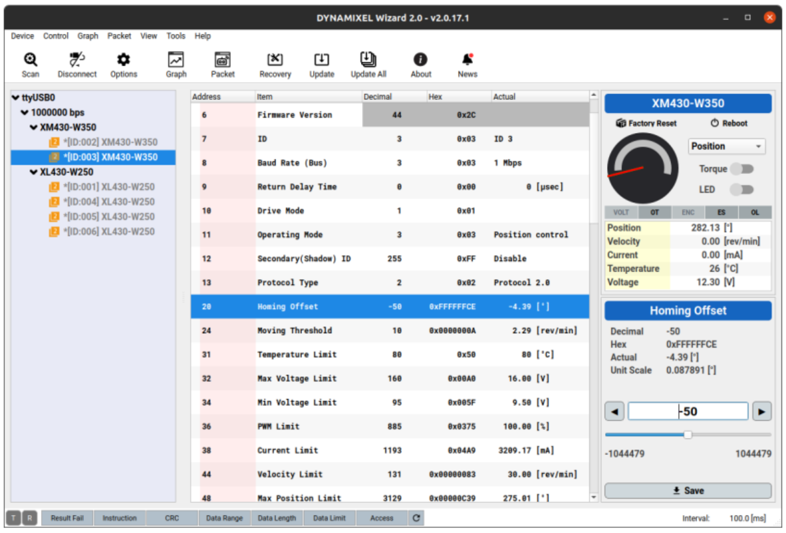
Figure 7.1: Motor ID correction with the DYNAMIXEL Wizard 2.0

7.2 Homing Offset

While working with the ReactorX-150 arms, it was observed that the physical arms showed a tendency to sag or deflect under gravity, compared to the simulated arms. To resolve this issue, the DYNAMIXEL Wizard 2.0 was utilized to adjust the “Homing Offset” of the associated DYNAMIXEL motors.

By default, all Homing Offsets are set to zero. The DYNAMIXEL motors offer 4096 positions for a full 360 degrees of rotation, equaling a unit scaling of the Homing offset of approximately 0.087891 degrees. Labeling the two arms as arm 1 and arm 2, the adjusted “Homing Offset" for the two ReactorX-150 were as follows:

- Arm 1, “elbow” joint (ID: 3) adjusted: -50 units, equaling -4.39 degrees, see Figure 7.2
- Arm 2, “elbow” joint (ID: 3) adjusted: -40 units, equaling -3.52 degrees, see Figure 7.3
- Arm 2, “wrist_angle” joint (ID: 4) adjusted: -50 units, equaling -4.39 degrees, see Figure 7.4



Address	Item	Decimal	Hex	Actual
6	Firmware Version	44	0x2C	
7	ID	3	0x03	ID 3
8	Baud Rate (Bus)	3	0x03	1 Mbps
9	Return Delay Time	0	0x00	0 [μsec]
10	Drive Mode	1	0x01	
11	Operating Mode	3	0x03	Position control
12	Secondary(Shadow) ID	255	0xFF	Disable
13	Protocol Type	2	0x02	Protocol 2.0
20	Homing Offset	-50	0xFFFFFFCE	-4.39 [°]
24	Moving Threshold	10	0x0000000A	2.29 [rev/min]
31	Temperature Limit	80	0x50	80 [°C]
32	Max Voltage Limit	160	0x00A0	16.00 [V]
34	Min Voltage Limit	95	0x005F	9.50 [V]
36	PWM Limit	885	0x0375	100.00 [%]
38	Current Limit	1193	0x04A9	3209.17 [mA]
44	Velocity Limit	131	0x00000083	30.00 [rev/min]
48	Max Position Limit	3129	0x00000C39	275.01 [°]

Position	282.13 [°]
Velocity	0.00 [rev/min]
Current	0.00 [mA]
Temperature	26 [°C]
Voltage	12.30 [V]

Homing Offset

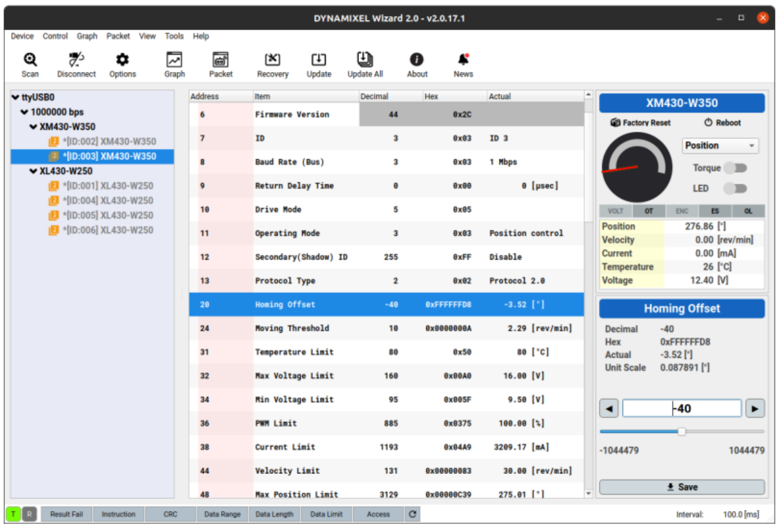
Decimal	-50
Hex	0xFFFFFFCE
Actual	-4.39 [°]
Unit Scale	0.087891 [°]

-50

-1044479 1044479

Interval: 100.0 [ms]

Figure 7.2: "Homing Offset" of the "elbow" joint (ID: 3) of arm 1



Address	Item	Decimal	Hex	Actual
6	Firmware Version	44	0x2C	
7	ID	3	0x03	ID 3
8	Baud Rate (Bus)	3	0x03	1 Mbps
9	Return Delay Time	0	0x00	0 [μsec]
10	Drive Mode	5	0x05	
11	Operating Mode	3	0x03	Position control
12	Secondary(Shadow) ID	255	0xFF	Disable
13	Protocol Type	2	0x02	Protocol 2.0
20	Homing Offset	-40	0xFFFFFFD8	-3.52 [°]
24	Moving Threshold	10	0x0000000A	2.29 [rev/min]
31	Temperature Limit	80	0x50	80 [°C]
32	Max Voltage Limit	160	0x00A0	16.00 [V]
34	Min Voltage Limit	95	0x005F	9.50 [V]
36	PWM Limit	885	0x0375	100.00 [%]
38	Current Limit	1193	0x04A9	3209.17 [mA]
44	Velocity Limit	131	0x00000083	30.00 [rev/min]
48	Max Position Limit	3129	0x00000C39	275.01 [°]

Position	276.86 [°]
Velocity	0.00 [rev/min]
Current	0.00 [mA]
Temperature	26 [°C]
Voltage	12.40 [V]

Homing Offset

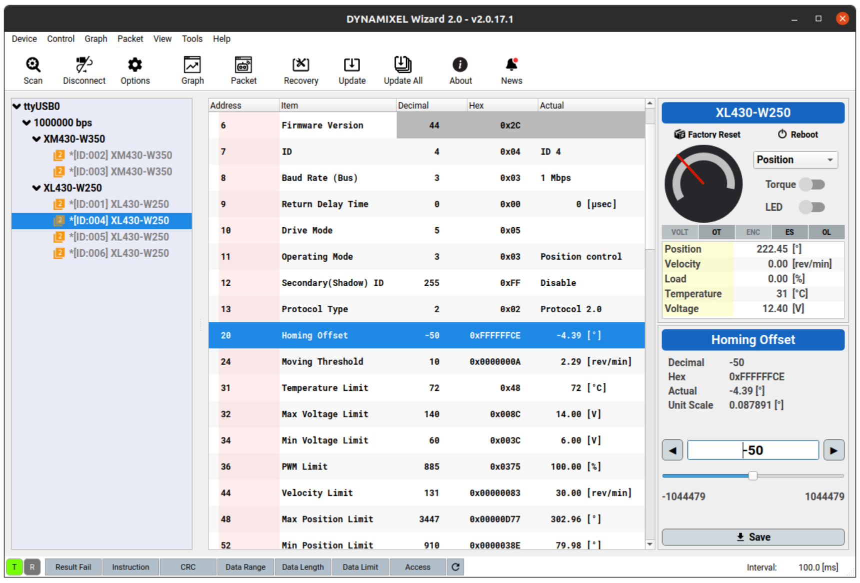
Decimal	-40
Hex	0xFFFFFFD8
Actual	-3.52 [°]
Unit Scale	0.087891 [°]

-40

-1044479 1044479

Interval: 100.0 [ms]

Figure 7.3: "Homing Offset" of the "elbow" joint (ID: 3) of arm 2



Address	Item	Decimal	Hex	Actual
6	Firmware Version	44	0x2C	
7	ID	4	0x04	ID 4
8	Baud Rate (Bus)	3	0x03	1 Mbps
9	Return Delay Time	0	0x00	0 [μsec]
10	Drive Mode	5	0x05	
11	Operating Mode	3	0x03	Position control
12	Secondary(Shadow) ID	255	0xFF	Disable
13	Protocol Type	2	0x02	Protocol 2.0
20	Homing Offset	-50	0xFFFFFFCE	-4.39 [°]
24	Moving Threshold	10	0x0000000A	2.29 [rev/min]
31	Temperature Limit	72	0x48	72 [°C]
32	Max Voltage Limit	140	0x008C	14.00 [V]
34	Min Voltage Limit	60	0x003C	6.00 [V]
36	PWM Limit	885	0x0375	100.00 [%]
44	Velocity Limit	131	0x00000083	30.00 [rev/min]
48	Max Position Limit	3447	0x00000D77	302.96 [°]
52	Min Position Limit	910	0x0000038E	79.98 [°]

Position	222.45 [°]
Velocity	0.00 [rev/min]
Load	0.00 [%]
Temperature	31 [°C]
Voltage	12.40 [V]

Homing Offset

Decimal	-50
Hex	0xFFFFFFCE
Actual	-4.39 [°]
Unit Scale	0.087891 [°]

-50

-1044479 1044479

Interval: 100.0 [ms]

Figure 7.4: "Homing Offset" of the “wrist_angle" joint (ID: 4) of arm 2

The results of the “Homing Offset” adjustment can be seen in

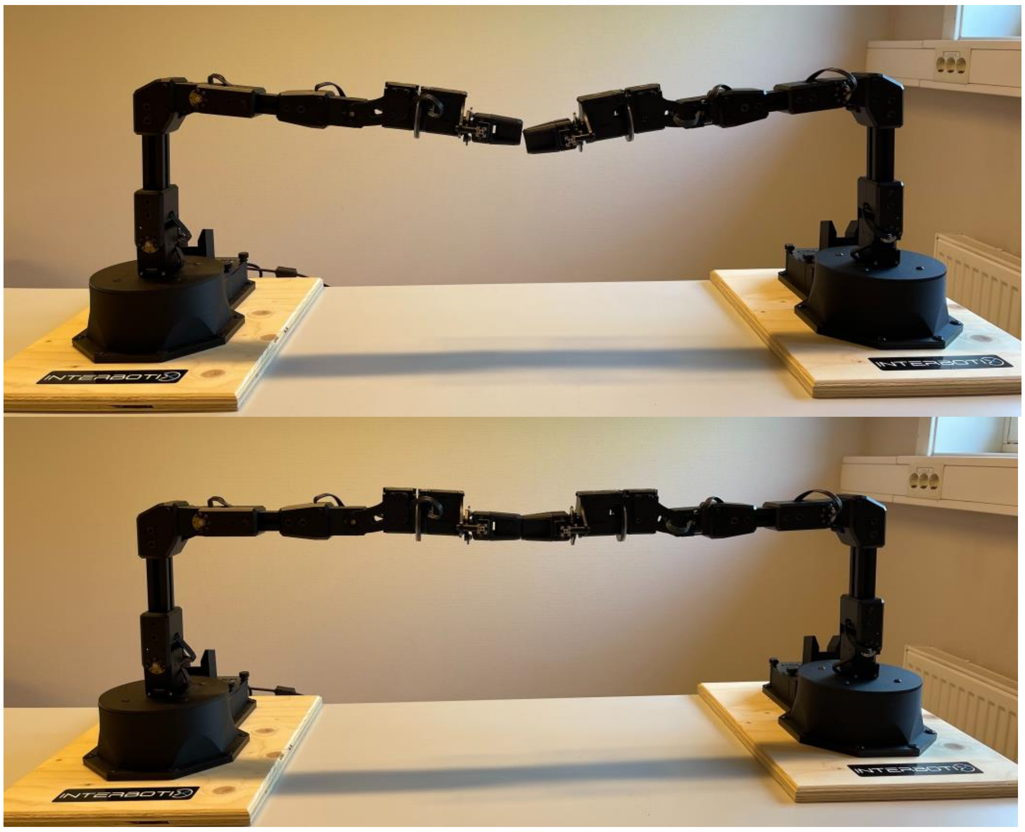


Figure 7.5: Before and after the "Homing Offset" adjustment

It is important to keep in mind that the adjustment of the “Homing Offset” for this application was relatively small and have not been tested with larger adjustments. Adjusting the “Homing Offset” with larger offset or when pushing the operational boundaries may cause unwanted results. However, the adjustment of the “Homing Offset” was purely beneficial for this application.

7.3 MATLAB-ROS Interface

The implementation of MATLAB and its capabilities were not covered in this thesis, it is noteworthy to mention the potential integration with MATLAB. MATLAB is a powerful tool in combination with ROS, which further extends the capabilities of the ReactorX-150 and the X-Series arms. Although the integration with MATLAB was beyond the scope of this thesis, a separate study at USN including the integrations with MATLAB has been conducted. Interested readers can refer to the work seen in [62].

7.4 Virtual Machines

An alternative method to dual booting, which was considered for running Ubuntu, involved the use of a virtual machine. This approach offers several advantages for online students and industry-based master's students. Utilizing a virtual machine would eliminate the risks associated with dual booting such as data loss or system crashes and protect the host system from potential threats.

An attempt was made to set up a virtual machine using Oracle VM VirtualBox. However, the attempt was unsuccessful as the virtual machine resulted in a "kernel panic" - an error from which the operating system could not safely recover.

It is important to note that it is stated from Trossen Robotics that virtual machines are not tested, and therefore not supported. [63]

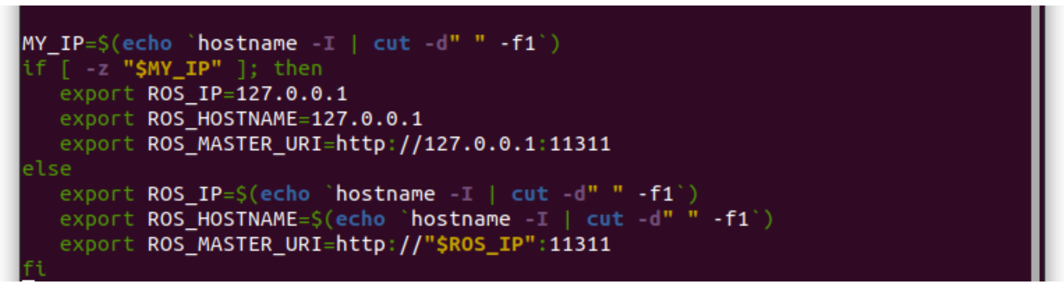
Being able to utilize virtual machines for working with the X-Series arms would be a great addition and should not be dismissed. The ability to utilize virtual machines together with the X-Series robotic arms, both in their physical and simulated states, would improve the flexibility and accessibility of working with the robot arms.

7.5 Network Setup

During the setup of ROS, an error message appeared preventing the launch of ROS packages with the ReactorX-150. The error message was roughly as follows:

“… RLException: Unable to contact my own server at [http:<ip-adress>] …”

The issue was resolved by adding the lines shown in Figure 7.6 to the “~/.bashrc” file.



```
MY_IP=$(echo `hostname -I | cut -d" " -f1`)
if [ -z "$MY_IP" ]; then
   export ROS_IP=127.0.0.1
   export ROS_HOSTNAME=127.0.0.1
   export ROS_MASTER_URI=http://127.0.0.1:11311
else
   export ROS_IP=$(echo `hostname -I | cut -d" " -f1`)
   export ROS_HOSTNAME=$(echo `hostname -I | cut -d" " -f1`)
   export ROS_MASTER_URI=http://"$ROS_IP":11311
fi
```

Figure 7.6: Network setup in the "~/.bashrc" file [64]

7.6 3D printing

The application of 3D printing was briefly explored in this thesis. Two USN logos were designed, and 3D printed for manipulation by the ReactorX-150, and for the presentation at the USN Expo. The 3D models of the logos were designed using Sharpr3D. Sharpr3D offered an intuitive way of designing the 3D models. It enabled the creation of the simple, yet intricate USN logos, even for someone without previous experience with 3D modelling.

The reason behind the creation and printing of two USN logos was that the first USN logo had no frame, which meant that the object had no uniform shape. Without the uniform shape, precautions had to be taken when manipulating the logo with the arm. To eliminate the need for precautions, the framed USN logo was created.

The framed and unframed USN logos with measurements in Sharpr3D can be seen in Figure 7.7 and Figure 7.8, respectively.

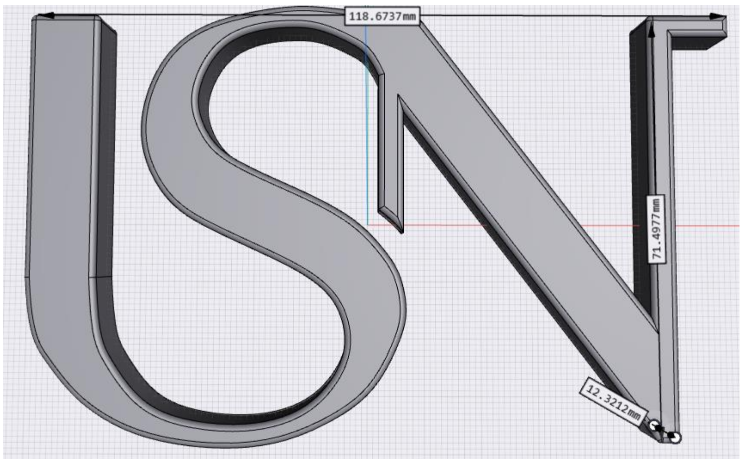


Figure 7.7: Sharpr3D USN logo with measurements

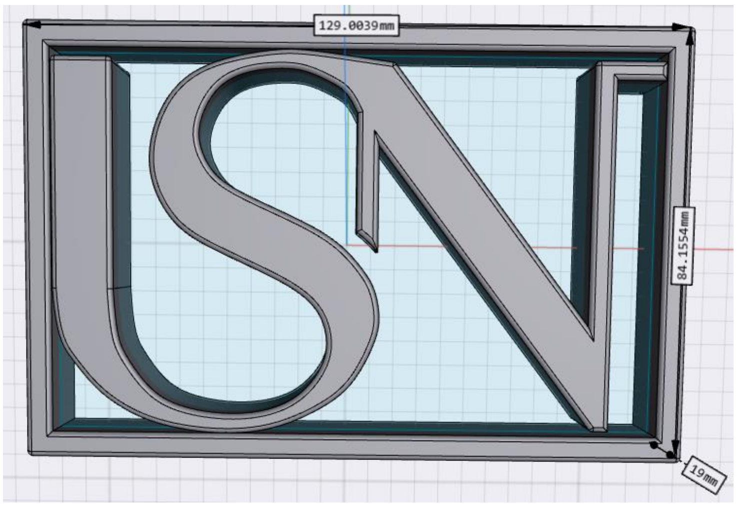
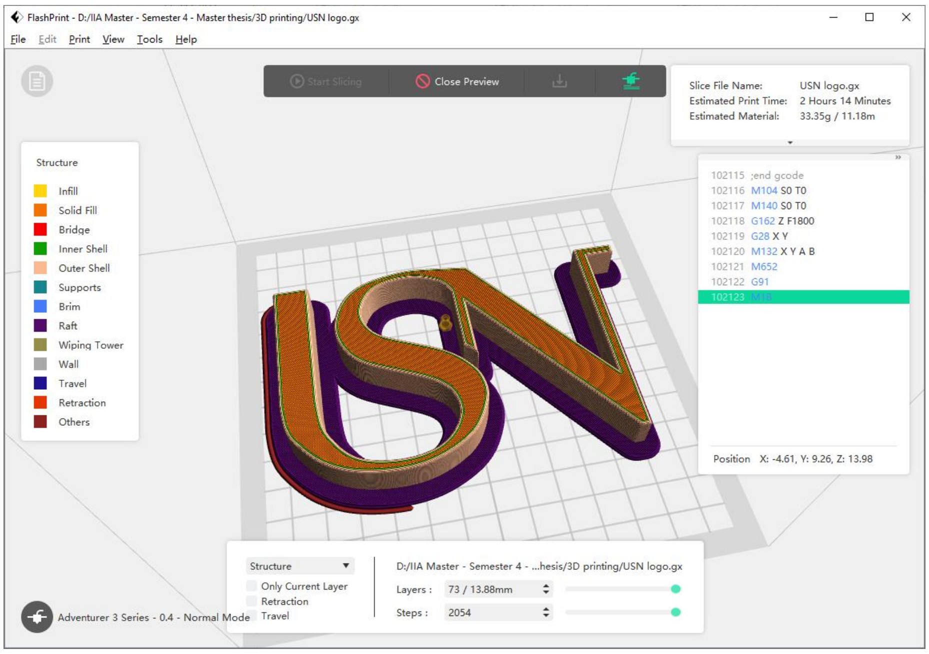


Figure 7.8: Sharpr3D framed USN logo with measurements

Once the 3D models of the logos were completed, the models were prepared for 3D printing using FlashPrint 5, a slicing and printing software specifically designed for FlashForge printers.

The framed and unframed USN logos in the FlashForge software can be seen in Figure 7.9 and Figure 7.10, respectively.



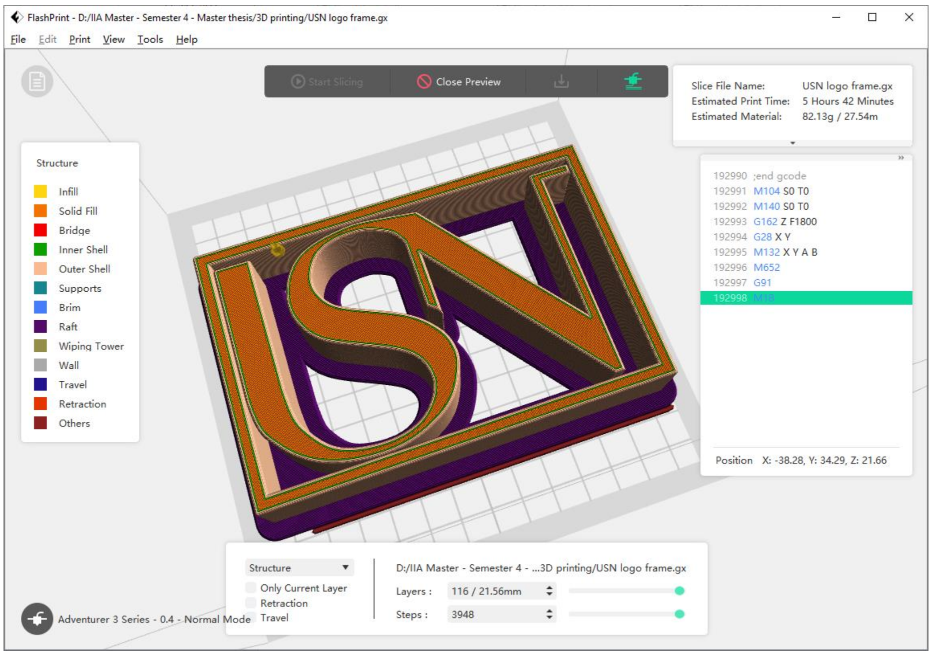
Slice File Name: USN logo.gx
Estimated Print Time: 2 Hours 14 Minutes
Estimated Material: 33.35g / 11.18m

```
102115 ;end gcode
102116 M104 S0 T0
102117 M140 S0 T0
102118 G162 Z F1800
102119 G28 X Y
102120 M132 X Y A B
102121 M652
102122 G91
102123
```

Position X: -4.61, Y: 9.26, Z: 13.98

D:/IIA Master - Semester 4 - ...hesis/3D printing/USN logo.gx
Layers : 73 / 13.88mm
Steps : 2054

Figure 7.9: FlashForge USN logo



Slice File Name: USN logo frame.gx
Estimated Print Time: 5 Hours 42 Minutes
Estimated Material: 82.13g / 27.54m

```
192990 ;end gcode
192991 M104 S0 T0
192992 M140 S0 T0
192993 G162 Z F1800
192994 G28 X Y
192995 M132 X Y A B
192996 M652
192997 G91
192998
```

Position X: -38.28, Y: 34.29, Z: 21.66

D:/IIA Master - Semester 4 - ...3D printing/USN logo frame.gx
Layers : 116 / 21.56mm
Steps : 3948

Figure 7.10: FlashForge framed USN logo

The two logos were then saved from the computer as ".gx" files to a flash drive. The files were then uploaded from the flash drive to the FlashForge Adventurer 3 3D printer.

7.7 Perception Configuration

The Perception package with the ReactorX-150 had the potential to be an exciting aspect of this thesis, but due to hardware constraints it could not be successfully tested. The main hardware constraint was the unavailability of an Intel RealSense color/depth camera. The specific camera used for testing the Perception package by Trossen Robotics was the Intel RealSense Depth Camera D415.

Attempts were made to utilize readily accessible devices such as the Raspberry Pi and its camera module, as well as the Lumens DC125 camera. Unfortunately, these efforts were unsuccessful.

The Raspberry Pi's and the Raspberry Pi Camera Module v2, widely utilized in various projects for its affordability and accessibility, failed to provide the required compatibility with the perception package. Similarly, the Lumens DC125 did not meet the specific hardware requirements of the perception package for the ReactorX-150. The primary hardware limitation of the cameras was the absence of depth perception capabilities, which are essential for generating the point cloud data required by the Perception package.

The obstacle regarding the lack of depth perception for the cameras underscores the importance of hardware compatibility. For future work involving the ReactorX-150, consideration should be given to the compatibility of camera hardware with the Perception package.

7.8 Future Work

The capabilities of the Reactor-X 150 reach beyond what this thesis has been able to cover, and there are some key areas with the possibility for future work.

7.8.1 Perception package

The perception package is an extensive package with unexplored capabilities. Future work could focus on identifying and testing cameras for compatibility or explore the development of custom camera hardware solutions. There exists software that can extract point cloud data from cameras without the capabilities of depth perception, an intriguing subject regarding use of readily accessible devices.

However, the most promising direction for future work with the Perception package would be acquiring an Intel RealSense color/depth camera. Trossen Robotics do supply a kit containing the Intel RealSense Depth Camera D415, a camera stand, colored blocks for manipulation, AprilTag marker and remaining accessories needed for the Perception package at [65].

7.8.2 AprilTag

The AprilTag marker is necessary hardware for the Perception package. Although not required, it is recommended to utilize the AprilTag markers. The alternative to the AprilTag markers is to manually input the position of the arm/end-effector relative to the camera, and that method is prone to error. For future work a physical AprilTag marker should be created, especially if not intending to buy the kit seen in [65]. The AprilTag families with a small user manual on how to rescale the AprilTag markers can be seen at [66]

7.8.3 Raspberry Pi

The Raspberry Pi is a powerful computer for its size. The Raspberry Pi 3 Model B and the Raspberry Pi 4 Model B utilized in this thesis have both been fully set up to be able to run the packages and control the ReactorX-150 in the attempt to get the Perception package up and running. The utilization of The Raspberry Pi as a standalone computer for controlling the ReactorX-150 arms has the potential for future work.

The utilization of the Raspberry Pi could eliminate the need for dual booting especially for campus students with Raspberry Pi's available at campus, or even online students with a Raspberry Pi available.

8 Conclusion

This thesis has successfully implemented the majority of the packages discussed, with comprehensive testing and documentation that can serve as a valuable resource for future students working with the ReactorX-150 arms. It has created a simulation platform that students can implement on their own computers, enabling access from any corner of the world, thus overcoming geographical limitations.

The full potential of the ReactorX-150, however, remains to be explored. Particularly, the perception package offers exciting opportunities for future research and development. This thesis has laid a foundation for further exploration, testing, and enhancement.

References

[1] ISO, «ISO 10303-1:2021(en),» [Internett]. Available: https://www.iso.org/obp/ui/#iso:std:iso:10303:-1:ed-2:v1:en. [Funnet 16 March 2023].

[2] Trossen Robotics, «ReactorX-150,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications/rx150.html. [Funnet 13 March 2023].

[3] Trossen Robotics, «Specifications,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications.html#specifications. [Funnet 16 February 2023].

[4] Wikipedia, «Product of exponentials formula,» 21 January 2023. [Internett]. Available: https://en.wikipedia.org/wiki/Product_of_exponentials_formula. [Funnet 16 March 2023].

[5] Trossen Robotics, [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/_downloads/21a942b8312b7865edc6de72eb57fb50/3_RXA-150-M.zip. [Funnet 16 March 2023].

[6] Trossen Robotics, «interbotix_ros_manipulators,» 22 October 2020. [Internett]. Available: https://github.com/Interbotix/interbotix_ros_manipulators/tree/main/interbotix_ros_xsarms/interbotix_xsarm_descriptions/meshes/rx150_meshes. [Funnet 16 March 2023].

[7] Trossen Robotics, «Interbotix X-Series Arms,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/. [Funnet 20 April 2023].

[8] Trossen robotics, «DYNAMIXEL XL430-W250-T Robot Actuator,» [Internett]. Available: https://www.trossenrobotics.com/dynamixel-xl430-w250-t.aspx. [Funnet 17 March 2023].

[9] Trossen Robotics, «DYNAMIXEL XM430-W350-T Robot Actuator,» [Internett]. Available: https://www.trossenrobotics.com/dynamixel-xm430-w350-t.aspx. [Funnet 17 March 2023].

[10] Robotis, «U2D2,» [Internett]. Available: https://emanual.robotis.com/docs/en/parts/interface/u2d2/. [Funnet 20 March 2023].

[11] Trossen Robotics, «DYNAMIXEL U2D2,» [Internett]. Available: https://www.trossenrobotics.com/dynamixel-u2d2.aspx. [Funnet 20 March 2023].

[12] Trossen Robotics, «6 Port XM/XL Power Hub (3pin),» [Internett]. Available: https://www.trossenrobotics.com/3-pin-x-series-power-hub.aspx. [Funnet 20 March 2023].

[13] Trossen Robotics, «12v5a Power Supply,» [Internett]. Available: https://www.trossenrobotics.com/12v5a-power-supply.aspx. [Funnet 20 March 2023].

[14] Trossen Robotics, «Customized Grippers,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/getting_started/customized_grippers.html. [Funnet 30 March 2023].

[15] W. M. Danylo Malyuta, «apriltag_ros,» [Internett]. Available: https://github.com/AprilRobotics/apriltag_ros. [Funnet 30 March 2023].

[16] Trossen Robotics, «Perception Configuration,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/perception_pipeline_configuration.html. [Funnet 30 March 2023].

[17] Intel Corporation, «Intel® RealSense™ Depth Camera D415,» [Internett]. Available: https://www.intelrealsense.com/depth-camera-d415/. [Funnet 17 April 2023].

[18] Raspberry Pi Foundation, «Raspberry Pi 3 Model B,» [Internett]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/. [Funnet 30 March 2023].

[19] Wikipedia, «Raspberry Pi 4 Model B from the side.,» [Internett]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi#/media/File:Raspberry_Pi_4_Model_B_-_Side.jpg. [Funnet 10 May 2023].

[20] Raspberry Pi Foundation, «Raspberry Pi Camera Module 2,» [Internett]. Available: https://www.raspberrypi.com/products/camera-module-v2/. [Funnet 30 March 2023].

[21] Lumens, «DC125,» [Internett]. Available: https://www.mylumens.com/en/Products_detail/10/DC125-Document-Camera. [Funnet 13 May 2023].

[22] Flashforge, «Adventurer 3,» [Internett]. Available: https://www.flashforge.com/product-detail/flashforge-adventurer-3-3d-printer. [Funnet 30 March 2023].

[23] Trossen Robotics, «ReactorX 150 Robot Arm,» [Internett]. Available: https://www.trossenrobotics.com/reactorx-150-robot-arm.aspx. [Funnet 31 March 2023].

[24] Trossen Robotics, «ROS Interface,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface.html. [Funnet 31 March 2023].

[25] Trossen Robotics, «IRROS,» [Internett]. Available: https://github.com/Interbotix/interbotix_ros_core. [Funnet 31 March 2023].

[26] Trossen Robotics, «Software,» [Internett]. Available: https://emanual.robotis.com/docs/en/software/. [Funnet 3 April 2023].

[27] Trossen Robotics, «DYNAMIXEL Wizard 2.0,» [Internett]. Available: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/. [Funnet 20 April 2023].

[28] Wikipedia, «Etcher-icon.png,» [Internett]. Available: https://upload.wikimedia.org/wikipedia/commons/2/2d/Etcher-icon.png. [Funnet 18 April 2023].

[29] Microsoft, «Icons and names usage guidelines,» [Internett]. Available: https://code.visualstudio.com/brand. [Funnet 3 April 2023].

[30] Sharpr3D, «Sharpr3D,» [Internett]. Available: https://www.shapr3d.com. [Funnet 7 April 2023].

[31] Zhejiang Flashforge 3D Technology Co., Ltd, «FlashPrint,» [Internett]. Available: https://www.flashforge.com/download-center/63. [Funnet 12 April 2023].

[32] C. L. Sindre Haugseter, «ReactorX 150 robotic arm manipulator,» Porsgrunn, 2022.

[33] R. Sharma, «Lecture Notes for IIA xxxx: Control for Robotics,» [Internett]. Available: https://web01.usn.no/~roshans/cfr/downloads/Control-for-Robotics-Lecture-notes.pdf. [Funnet 15 May 2023].

[34] Trossen Robotics, «Arm Descriptions,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/arm_descriptions.html. [Funnet 19 April 2023].

[35] Trossen Robotics, «arm.py,» [Internett]. Available: https://github.com/Interbotix/interbotix_ros_toolboxes/blob/main/interbotix_xs_toolbox/interbotix_xs_modules/src/interbotix_xs_modules/arm.py. [Funnet 12 May 2023].

[36] Trossen Robotics, «Python Demos,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/python_demos.html. [Funnet 20 April 2023].

[37] Trossen Robotics, «MATLAB Demos,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/matlab_demos.html. [Funnet 20 April 2023].

[38] SRI International, «move_group_python_interface_tutorial.py,» [Internett]. Available: https://github.com/ros-planning/moveit_tutorials/blob/482dc9db944c785870274c35223b4d06f2f0bc90/doc/move_group_python_interface/scripts/move_group_python_interface_tutorial.py. [Funnet 4 May 2023].

[39] Trossen Robotics, «MoveIt Interface and API,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/moveit_interface_and_api.html. [Funnet 20 April 2023].

[40] Trossen Robotics, «kinematics.yaml,» [Internett]. Available: https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_moveit/config/kinematics.yaml. [Funnet 9 May 2023].

[41] Wikipedia, «Levenberg–Marquardt algorithm,» [Internett]. Available: https://en.wikipedia.org/wiki/Levenberg–Marquardt_algorithm. [Funnet 9 May 2023].

[42] Orocos Kinematics and Dynamics, «KDL wiki,» [Internett]. Available: https://www.orocos.org/kdl.html. [Funnet 9 May 2023].

[43] MoveIt, «Kinematics Configuration Tutorial,» [Internett]. Available: https://docs.ros.org/en/indigo/api/moveit_tutorials/html/doc/pr2_tutorials/kinematics/src/doc/kinematics_configuration.html. [Funnet 9 May 2023].

[44] Wikipedia, «Levenberg–Marquardt algorithm,» [Internett]. Available: https://en.wikipedia.org/wiki/Levenberg–Marquardt_algorithm. [Funnet 15 May 2023].

[45] Trossen Robotics, «Gazebo Configuration,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/gazebo_simulation_configuration.html. [Funnet 2 May 2023].

[46] Trossen Robotics, «ROS Controllers Configuration,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/ros_control.html. [Funnet 2 May 2023].

[47] Intel Corporation, «Intel® RealSense™ Depth Camera SR305,» [Internett]. Available: https://www.intelrealsense.com/depth-camera-sr305/. [Funnet 9 May 2023].

[48] Intel Corporation, «Intel® RealSense™ Depth Camera D415,» [Internett]. Available: https://www.intelrealsense.com/depth-camera-d415/. [Funnet 9 May 2023].

[49] Trossen Robotics, «Perception Configuration,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/perception_pipeline_configuration.html. [Funnet 2 May 2023].

[50] Open Perception, «perception_pcl,» [Internett]. Available: http://wiki.ros.org/perception_pcl. [Funnet 9 May 2023].

[51] Trossen Robotics, «Joystick Control,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/joystick_control.html. [Funnet 20 April 2023].

[52] Trossen Robotics, «ArmJoy.msg,» [Internett]. Available: https://github.com/Interbotix/interbotix_ros_core/blob/main/interbotix_ros_xseries/interbotix_xs_msgs/msg/ArmJoy.msg. [Funnet 5 May 2023].

[53] Trossen Robotics, «Record And Playback,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/record_and_playback.html. [Funnet 2 May 2023].

[54] Trossen Robotics, «Arm Diagnostic Tool,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/arm_diagnostic_tool.html. [Funnet 2 May 2023].

[55] Trossen Robotics, «Arm Puppeteering,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/arm_puppeteering.html. [Funnet 2 May 2023].

[56] Trossen Robotics, «Dual Arm Control,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/dual_arm_control.html. [Funnet 2 May 2023].

[57] Trossen Robotics, «Dual Arm Joystick Control,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/dual_arm_joystick_control.html. [Funnet 2 May 2023].

[58] C. Lauritzen, «Joystick Control - Simulated,» [Internett]. Available: https://usn.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=01300696-9759-4011-a652-b00300539579. [Funnet 13 May 2023].

[59] C. Lauritzen, «Dual Arm Joystick Control - Simulated,» [Internett]. Available: https://usn.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=100f6079-7403-4110-b742-b0030053f237. [Funnet 13 May 2023].

[60] C. Lauritzen, «Joystick Control,» [Internett]. Available: https://usn.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=6294d56b-e7e2-4011-b7b6-b003002669b5. [Funnet 15 May 2023].

[61] C. Lauritzen, «Dual Arm Joystick Control,» [Internett]. Available: https://usn.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=b1d667e4-9b89-4b31-81e9-b00300444e8f. [Funnet 15 May 2023].

[62] M. S. Chowdhury, «Real time control of robotic arm manipulators,» [Internett]. Available: https://openarchive.usn.no/usn-xmlui/bitstream/handle/11250/3000634/no.usn%3Awiseflow%3A6583421%3A50226134.pdf?sequence=1&isAllowed=y. [Funnet 13 May 2023].

[63] Trossen Robotics, «ROS Standard Software Setup,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface/ros1/software_setup.html. [Funnet 13 May 2023].

[64] Trossen Robotics, «ROS 1 Configuration,» [Internett]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface/ros1/config.html. [Funnet 14 May 2023].

[65] Trossen Robotics, «Interbotix ROS Arm Vision Kit,» [Internett]. Available: https://www.trossenrobotics.com/interbotix-arm-vision-kit.aspx. [Funnet 14 May 2023].

[66] «AprilTag-imgs,» [Internett]. Available: https://github.com/AprilRobotics/apriltag-imgs. [Funnet 14 May 2023].

9 Appendices

9.1 Appendix A – Master thesis task description

9.2 Appendix B – Guide for Dual Booting Windows and Ubuntu

9.3 Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics

9.4 Appendix D – Quickstart Guide for the X-Series Arms from Trossen Robotics

9.5 Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms

9.6 Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms

9.7 Appendix G – Joystick controls

9.8 Appendix H – Joystick controller pairing

9.9 Appendix I – Raspberry Pi Ubuntu and ROS Setup Guide

9.10 Appendix J – bartender.py

9.11 Appendix K – "rostopic" guide

9.12 Appendix L – xsarm_dual_usn_lift.py

9.13 Appendix M – xsarm_dual_usn_down.py

9.1 Appendix A – Master thesis task description

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Development of a simulation platform and testing of a 5 degrees of freedom ReactorX 150 robotic arm manipulator.

USN supervisor: Associate Professor Roshan Sharma (USN)

Task background:

Robotic arm manipulators are widely used in industries for various applications. They are used in automotive, aerospace, electronic/electrical industries, shipping and trade etc. (just to name a few), for example, for performing repetitive tasks like those involved in an assembly line. USN has recently purchased several units of a ReactorX 150 robotic arm manipulators from Trossen Robotics. These robotic arms are planned to be used in teaching and research activities here at USN. The ReactorX 150 offers 5 degrees of freedom and a full 360 degree of rotation. At the heart of the ReactorX150 is the Robotis DYNAMIXEL X-Series smart servo motors and DYNAMIXEL U2D2 which enables easy access to Dynamixel software development kit. Figure 1 shows the Reactorx150 robotic arm manipulator.

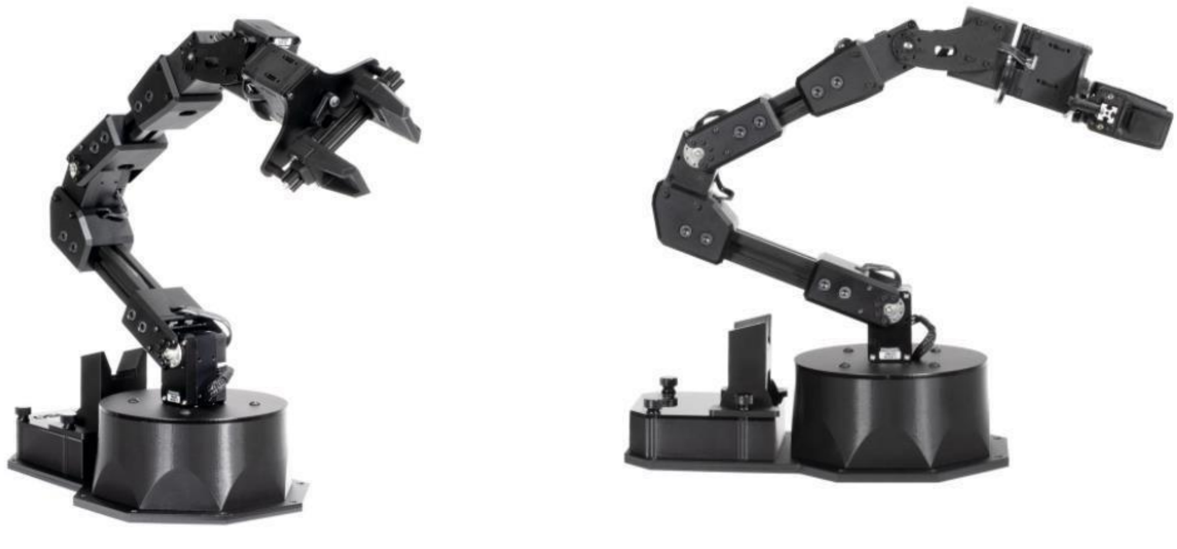


Figure 1: ReactorX 150 robotic arm manipulator

Aim:

It is of interest to make a simulation platform for this robotic arm in ROS (Robot Operating System) and possibly in Linux platform. Creating such a simulation platform would allow online students (or industry master students) to test, implement and simulate various functionalities of this robot arm by simply sitting at any corner of world and not actually having to be physically present at campus to use the lab equipment. Various tasks like position/trajectory control, direct/inverse kinematics etc. can be performed using the designed simulation platform. On the other hand, to facilitate the campus students with lab equipment, it is also of interest to test these tasks on a real ReactorX 150 robotic arm lab unit.

Task description:

The following are the main tasks:

a) Install Linux and ROS. In addition install various ROS packages for ReactorX 150 robot arm. Learn about ROS.
b) Learn how to use Gazebo or Rviz and/or MoveIt as simulation tools to interact with virtual ReactorX150 robot arm.
c) Simulate the robotic arm for direct and inverse kinematics using Gazebo or Rviz and/or MoveIt and ROS
d) Test the direct and inverse kinematics on a physical RX 150 robot arm using ROS.
e) If time allows, develop and simulate your own inverse kinematic solver using simpler kinematic model and some sort of optimization.
f) Document the work in a report. Presentation of the work.

Student category: Reserved

This thesis is reserved for campus IIA student Christian Lauritzen.

Practical arrangements:

The student will be provided with a physical ReactorX 150 robotic arm. The place for using the physical lab unit is campus Porsgrunn.

Signatures:

Supervisor (date and signature): 01.02.2023 [illegible]

Student (write clearly in all capitalized letters): CHRISTIAN LAURITZEN

Students (date and signature): 01.02.2023 [illegible]

9.2 Appendix B – Guide for Dual Booting Windows and Ubuntu

Dual booting allows users to install multiple operating systems on a single machine, providing the flexibility to switch between the operating systems as needed. This guide aims to walk the user through the steps to dual boot Windows 10 and Ubuntu 20.04, using the Disk Management tool in Windows for disk partitioning and BalenaEtcher for burning the ISO image.

Note that this guide has used an Acer PC and some steps may be differ for computers from other manufacturers.

Note that this guide may differ if other versions of Windows or Ubuntu is utilized.

Requirements

- A PC with Windows 10 installed
- Minimum 30 GB of available space on PC
- Wi-Fi internet access
- The Ubuntu 20.04 desktop ISO file
  - Download from the Ubuntu web page at: https://releases.ubuntu.com/focal/
- A USB flash drive (recommended minimum 8 GB)
- The BalenaEtcher software
  - Download from the Balena web page at: https://www.balena.io/etcher#download-etcher
  - An alternative to BalenaEtcher: Rufus (will not be covered in this guide)

Procedure

Follow the subsequent procedure in the following order.

1) Partition the hard drive using Disk Management in Windows

a) Open Disk management by one of the two methods:
   i) Press Windows key + X and select “Disk Management” from the menu.
   ii) Press Windows key, search for “Disk Management” and click “Create and format hard disk partitions”.
b) Locate the primary partition (usually C: drive), or the partition of choice and right-click on it.
c) Select “Shrink Volume” and enter the desired amount of space to allocate for Ubuntu. Minimum 30GB is recommended, 100GB was used for the thesis.
d) Click 'Shrink' to create unallocated space for the Ubuntu installation.

2) Create a bootable USB drive with Ubuntu 20.04 ISO

a) Insert the USB flash drive into the computer.
b) Open BalenaEtcher and click “Select Image” to browse and select the Ubuntu 20.04 ISO file.
c) Ensure the correct USB drive is selected under “Select target”.
d) Click “Flash!” to begin the process. Once completed, safely eject the USB drive.

3) **Boot from the USB drive**
   a) Restart the computer and press the appropriate key (F2, F10, F12, or Delete) to enter the BIOS/UEFI settings.
   b) Navigate to the boot options and set the USB drive as the first boot device.
      i) Note that in some cases "Secure Boot" also must be disabled in the BIOS/UEFI settings.
   c) Save changes and exit BIOS/UEFI settings. The computer should now boot from the USB drive.

4) **Install Ubuntu 20.04 alongside Windows 10**
   a) Select desired language and click "Install Ubuntu".
   b) Select desired keyboard layout.
   c) Connect to a Wi-Fi network (if applicable).
   d) Choose "Normal installation" and check the boxes under "Other options".
   e) Select "Install Ubuntu alongside Windows Boot Manager" and click "Continue".
   f) Adjust the slider to allocate space for Ubuntu and Windows partitions as desired, then click "Install Now".
   g) Confirm the partition changes and proceed with the installation.
   h) Choose the region and keyboard layout, then create a user account.
   i) Wait for the installation to complete, then click "Restart Now" to finish.

5) **Dual booting Windows 10 and Ubuntu 20.04**
   a) After restarting, the computer will now display the GRUB bootloader menu, allowing to choose between Windows 10 and Ubuntu 20.04.
   b) Use the arrow keys to navigate between the options, and press "enter" to boot into the desired operating system.

The setup is now complete, and the computer can now be dual booted with both Windows and Ubuntu

Additional

1) **Change back to normal Windows boot**
   a) Restart the computer and press the appropriate key (F2, F10, F12, or Delete) to enter the BIOS/UEFI settings.
   b) Navigate to the boot options and set the Windows partition as the first boot device.
      i) If "Secure Boot" was disabled, it can now be enabled again in the BIOS/UEFI settings.
   c) Save changes and exit BIOS/UEFI settings. The computer should now boot from the USB drive**.**
2) **Change default operating system with GRUB**
   a) To change the default operating system or adjust the bootloader timeout, edit the GRUB configuration file (usually located at /etc/default/grub) within Ubuntu.

9.3 Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics

The X-Series robotic arms from Trossen Robotics are versatile, high-performance robotic arms designed for education and research. This guide aims to provide a step-by-step process for installing ROS1 (Robot Operating System) for the X-Series robotic arms from Trossen Robotics.

Requirements

- A PC running Ubuntu Linux 18.04, 20.04, or 22.04. In this case the computer should be running Ubuntu Linux 20.04.
- Internet access.

Note that virtual machines running Ubuntu have not been tested, hence it is not supported.

Procedure

Follow the subsequent procedure in the following order. All commands throughout this guide are meant to be run in a Ubuntu terminal window.

1) Install ROS1 Noetic

a) Install ROS1 Noetic with the following four commands:

```
$ sudo apt install curl
$ curl
'https://raw.githubusercontent.com/Interbotix/interbotix_ros_manipulators/main/interboti
x_ros_xsarms/install/amd64/xsarm_amd64_install.sh'
$ chmod +x xsarm_amd64_install.sh
$ ./xsarm_amd64_install.sh -d noetic
```

2) Installation check (with the physical robot arm)

a) Connect the robot arm to a power source.
b) Connect the USB cable to the robot arm and the PC.
c) Run the following command to verify that the PC is successful in finding the U2D2:

```
$ ls /dev | grep ttyDXL
```

If successful, the expected output is:

```
ttyDXL
```

ROS 1 Noetic should now be installed and ready for use. If there is a problem with ROS or ROS did not install correctly, see the procedure on the next page.

1) Alternative installation of ROS 1 Noetic

If the installation fails or the program fails, this alternative installation of ROS1 Noetic might be the solution.

a) Remove ROS from the PC with the following two commands:

```
$ sudo apt-get purge ros-*
$ sudo apt-get autoremove
```

b) Install ROS1 Noetic with the following commands:

```
$ sudo apt update
$ sudo apt upgrade
$ wget https://raw.githubusercontent.com/ROBOTIS-
GIT/robotis_tools/master/install_ros_noetic.sh
$ chmod 755 ./install_ros_noetic.sh
$ bash ./install_ros_noetic.sh
$ sudo apt install ros-noetic-dynamixel-sdk
```

The commands can be found here:
https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

c) Repeat the steps in the previous "Install ROS 1 Noetic" step.

If the alternative installation also fails, see the official ROS1 Noetic installation guide at:
https://wiki.ros.org/noetic/Installation/Ubuntu

9.4 Appendix D – Quickstart Guide for the X-Series Arms from Trossen Robotics

This guide aims to provide knowledge of interfaces and basic functions of both the simulated and physical X-Series arms with ROS 1.

Requirements

- A PC running Ubuntu Linux 18.04, 20.04, or 22.04. In this case the computer should be running Ubuntu Linux 20.04.
- A PC with ROS 1 Noetic and the necessary packages for the Trossen Robotics X-Series arms installed.
- An X-Series robot arm from Trossen Robotics, in this case a ReactorX-150, codename “rx150”.

Procedure

Follow the subsequent procedure in the following order. All commands throughout this guide are meant to be run in an Ubuntu terminal window.

1) Simulate the robot arm for testing

a) Run the following command to simulate the ReactorX-150 with a GUI for testing different angles:

```
$ roslaunch interbotix_xsarm_descriptions xsarm_description.launch
robot_model:=rx150 use_joint_pub_gui:=true
```

b) Control the sliders or press “Randomize” to observe the arm in various positions and press “Center” to set the arm back in “home position”.

c) Go to the terminal window and press Ctrl + C to exit the session, closing the windows are not sufficient.

2) Testing the physical robot arm

a) Connect the robot arm to a power source.

b) Connect the USB cable to the robot arm and the PC.

c) Run the following command for testing basic functionality of the physical ReactorX-150:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
```

d) The following command will cause the physical arm to collapse, so the arm **must be manually secured before executing!** Run the following command to torque off the motors:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'all', enable:
false}"
```

e) Manipulate the arm to a desired position and run the following command to torque the motors on again:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'all', enable:
true}"
```

f) Insert “rx150” into the “Robot Namespace” textbox in the “Interbotix Control Panel” window on the lower left-hand side of the RViz window.
g) In the “Home/Sleep” tab in the “Interbotix Control Panel” press “Home” to bring the arm to the “home position”.
h) In the “Home/Sleep” tab in the “Interbotix Control Panel” press “Sleep” to bring the arm to the “sleep position”.
i) The following command will cause the physical arm to collapse. Except for when the arm is in “sleep position”, the arm **must be manually secured before executing!**

Go to the terminal window and press Ctrl + C to exit the session, closing the windows is not sufficient.

The quickstart guide is now completed and should have given the user an introduction into controlling the simulated and physical ReactorX-150.

9.5 Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms

This user manual aims to serve as a guide for understanding and utilization of the Trossen Robotics X-Series robotic arms, both for simulation and physical usage. It is designed to provide users with an overview of the various command line arguments and package options available for the X-Series arms. The in-depth information about the packages is covered in the thesis.

A user manual with only the main arguments for each package can be seen in [Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms].

Requirements

- A PC running Ubuntu Linux 18.04, 20.04, or 22.04. In this case the computer should be running Ubuntu Linux 20.04.
    - Installation guide: [Appendix B – Guide for Dual Booting Windows and Ubuntu]
- A PC with ROS 1 Noetic and the necessary packages for the Trossen Robotics X-Series arms installed.
    - Installation guide: [Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics]
- At least one but recommending two X-Series robot arms from Trossen Robotics. In this case two ReactorX-150's, codename "rx150".

Packages overview

1. **Arms Descriptions**
2. **Arm Control – Python and MATLAB**
3. **MoveIt Configuration, Interface, and API**
4. **Gazebo Configuration**
5. **ROS Controllers Configuration**
6. **Perception Configuration**
7. **Joystick Control**
8. **Record and Playback**
9. **Arm Diagnostic Tool**
10. **Arm Puppeteering**
11. **Dual Arm Control**
12. **Dual Arm Joystick Control**

Packages

The packages are sorted in no particular order but note that some packages are built upon other packages and some packages are not meant to be run as standalone packages. All commands throughout this guide are meant to be run in an Ubuntu terminal window.

Be sure to manually secure the arm(s) before pressing “Ctrl + C” when exiting any package.

1. Arms Descriptions

This chapter contains the commands for getting the Arms Descriptions package up and running and the arguments available in package.

Note that the Arms Descriptions package is a visualization/simulation package only.

1.1. Commands

Launch with GUI for controlling the joints:

```
$ roslaunch interbotix_xsarm_descriptions xsarm_description.launch
robot_model:=rx150 use_joint_pub_gui:=true
```

1.2. Arguments

The arguments for the Arms Description package can be seen in Table 9-1.

Table 9-1: Arms Descriptions arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_ar_tag	false	Set to “true” if intending to use AR tag such as the AprilTag
show_gripper_bar	true	If using a custom gripper attachment, set this to “false”

show_gripper_fingers	true	If using custom gripper fingers, set this to “false”
use_rviz	true	Launches RViz
use_joint_pub_gui	false	Launches a user-friendly GUI for controlling joint angles called the joint_state_publisher GUI
use_joint_pub	false	Launches the joint_state_publisher node
use_world_frame	true	Set this to false when working with multiple robots or if attaching the “base_link” frame to another frame.
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
load_gazebo_configs	false	Set to true if Gazebo shall be used, includes necessary
rvizconfig	See the Arm Descriptions launch package link below	File path of RViz config file
model	See the Arm Descriptions launch package link below	File path of URDF and argument to be passed in for the specific robot

Arm descriptions launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_descriptions/launch/xsarm_description.launch

2. Arm Control – Python and MATLAB

This chapter contains simple commands for getting the Arm Control package up and running with both Python and MATLAB scripts, and the arguments available in package.

2.1. Commands

Launch with physical robot:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
```

Lauch with simulated robot:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
use_sim:=true
```

Navigate to the python demos script directory. Launch Python script for ROS Noetic:

```
$ python3 bartender.py
```

Navigate to the MATLAB demo script directory. Launch MATLAB script:

```
bartender
```

The bartender scripts for both Python and MATLAB are included upon download of the necessary packages for the X-Series arms.

2.2. Arguments

The arguments for the Arm Control package can be seen in Table 9-2.

Table 9-2: Arm Control arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_ar_tag	false	Set to “true” if intending to use AR tag such as the AprilTag
show_gripper_bar	true	If using a custom gripper attachment, set this to “false”

show_gripper_fingers	true	If using custom gripper fingers, set this to “false”
use_world_frame	true	Set this to false when working with multiple robots or if attaching the “base_link” frame to another frame.
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
use_rviz	true	Launches RViz
motor_configs	See the Arm Control launch package link below	File path of the YAML file containing the motor configurations
mode_configs	See the Arm Control launch package link below	File path of the YAML file containing the mode configurations
load_configs	true	Set to true if the motor configurations shall be written to the motors. Is only necessary to do it the first time the node starts up, reduces startup time if set to “false”
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true

Arm Control launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_control/launch/xsarm_control.launch

3. MoveIt Configuration, Interface, and API

This chapter contains simple commands for getting the MoveIt Configuration and MoveIt Interface and API package up and running, and the arguments available in package.

3.1. Commands

Lauch with physical robot and the MoveIt Interface GUI:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_actual:=true
```

Lauch with simulated robot in Gazebo and the MoveIt Interface GUI, unpause the Gazebo physics for RViz to load:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_gazebo:=true
```

Lauch with simulated robot in RViz and the MoveIt Interface GUI:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_fake:=true
```

There is also an option to run MoveIt without the interface/GUI. The plugin for MoveIt in RViz can be used instead. Note that the MoveIt plugin in RViz is available for use when launching both the “interbotix_xsarm_moveit_interface” and “interbotix_xsarm_moveit” package. Enter the robot code name in the lower left corner of RViz and press the “update” button.

Launch MoveIt without the interface/GUI with physical robot:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_actual:=true
```

Launch MoveIt without the interface/GUI with simulated robot in Gazebo, unpause the Gazebo physics for RViz to load:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_gazebo:=true
```

Launch MoveIt without the interface/GUI with simulated robot in RViz:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_fake:=true
```

3.2. Arguments

The arguments for the MoveIt Configuration, Interface and API package can be seen in Table 9-3.

Note that the “use_cpp_interface”, “moveit_interface_gui” and “use_python_interface” arguments only apply to the “interbotix_xsarm_moveit_interface” package, and not the “interbotix_xsarm_moveit” package.

Table 9-3: MoveIt configuration, interface, and API arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_ar_tag	false	Set to “true” if intending to use AR tag such as the AprilTag
use_world_frame	true	Set this to false when working with multiple robots or if attaching the “base_link” frame to another frame.
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
external_srdf_loc	“”	If intending to include a custom srdf.xacro file, include the file path here
mode_configs	See the MoveIt launch package link below	File path of the YAML file containing the mode configurations
use_moveit_rviz	true	Launches RViz with the MoveIt plugin
rviz_frame	world	The value set as the fixed frame parameter in RViz, change to an existing frame if “use_world_frame” is set to false

use_gazebo	false	Simulate the robot with Gazebo
use_actual	false	Use the physical robot
use_fake	false	MoveIt generates a simulated robot to be controlled in RViz
dof	5	Defines the degrees of freedom of the robot arm
world_name	See the MoveIt launch package link below	File path to the world file to be loaded by Gazebo
use_cpp_interface	false	Launches the custom moveit_interface C++ API node
moveit_interface_gui	true	Launches a GUI customized to interface with the moveit_interface node
use_python_interface	false	Launches a Python Interface Tutorial node. Press "enter" in the terminal window to step through the tutorial

MoveIt launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_moveit_interface/launch/xsarm_moveit_interface.launch

4. Gazebo Configuration

This chapter contains simple commands for getting the Gazebo package up and running, and the arguments available in package.

Note that this package is meant to be run in conjunction with MoveIt or by itself via the JointPositionController interface. The JointPositionController interface is utilized by sending joint position commands to the arm using ROS subscriber topics. Gazebo is a simulation tool, so this package will not interact with the physical robot.

4.1. Commands

Launch the simulated robot with the ability to command arbitrary positions to the arm joints:

```
$ roslaunch interbotix_xsarm_gazebo xsarm_gazebo.launch robot_model:=rx150
use_position_controllers:=true
```

Unpause the physics engine by pressing the “play”-button in the left-hand corner of gazebo, or enter the following command:

```
$ rosservice call /gazebo/unpause_physics
```

Controlling the waist of the ReactorX-150 in radians:

```
$ rostopic pub -1 /rx150/waist_controller/command std_msg/Float64 “data: 1.0”
```

List of topics for controlling other joints (subscribers):

```
$ rostopic list -s
```

4.2. Arguments

The arguments for the Gazebo Configuration package can be seen in Table 9-4.

Table 9-4: Gazebo Configurations arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_ar_tag	false	Set to “true” if intending to use AR tag such as the AprilTag
show_gripper_bar	true	If using a custom gripper attachment, set this to “false”
show_gripper_fingers	true	If using custom gripper fingers, set this to “false”
use_rviz	true	Launches RViz
use_world_frame	true	Set this to false when working with multiple robots or if attaching the “base_link” frame to another frame.
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
dof	5	Defines the degrees of freedom of the robot arm
world_name	see the Gazebo launch package link below	File path to the world file to be loaded by Gazebo

gui	true	Launches the GUI of Gazebo
debug	false	Launches Gazebo in debug mode using GNU Debugger (gdb)
paused	true	Lauches Gazebo in a paused state
recording	false	Enables recording of Gazebo state log
use_sim_time	true	When set to true ROS nodes will get the Gazebo simulation time from the ROS topic /clock
use_position_controllers	false	Enables commanding of arbitrary arm joint positions in Gazebo
use_trajectory_controllers	false	Enables commanding of arbitrary arm joint trajectories in Gazebo

Gazebo launch package:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_gazebo/launch/xsarm_gazebo.launch

5. ROS Controllers Configuration

This chapter contains simple commands for getting the ROS controllers package up and running, and the arguments available in package.

Note that this package is only meant to be used in conjunction with MoveIt but can be used with other nodes that are able to communicate with the joint_trajectory_controller package.

5.1. Commands

Launch the package:

```
$ roslaunch interbotix_xsarm_ros_control xsarm_ros_control.launch robot_model:=rx150
```

5.2. Arguments

The arguments for the ROS Controllers Configuration package can be seen in Table 9-5.

Table 9-5: ROS Controllers Configurations arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_ar_tag	false	Set to “true” if intending to use AR tag such as the AprilTag
use_world_frame	true	Set this to false when working with multiple robots or if attaching the “base_link” frame to another frame.
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
use_rviz	false	Launches RViz
mode_configs	See the ROS Controllers launch package link below	File path of the YAML file containing the mode configurations

dof	5	Defines the degrees of freedom of the robot arm

ROS Controllers launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_ros_control/launch/xsarm_ros_control.launch

6. Perception Configuration

This chapter contains simple commands for getting the Perception package up and running and the arguments available in package.

Note that this package is meant to be run with a physical X-Series arm with an AprilTag (AR tag).

6.1. Setup commands

Start by finding the transform of the AprilTag relative to the camera's color optical frame with the following series of commands:

```
$ roslaunch interbotix_xsarm_perception xsarm_perception.launch robot_model:=rx150
use_pointcloud_tuner_gui:=true use_armtag_tuner_gui:=true
```

Manually secure the arm! In a new terminal window, torque the arm off:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
false}"
```

Move the arm so it is visible for the camera and execute the following command:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
true}"
```

Manually secure the arm! Torque the arm off and place it in the "sleeping position":

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
false}"
```

6.2. Utilization commands

Launch the package:

```
$ roslaunch interbotix_xsarm_perception xsarm_perception.launch robot_model:=rx200
use rviz:=false
```

Navigate to the "scripts" directory. Locate the "pick_place.py" script and comment out lines 25-28, since the transform of the arm relative to the camera is already found by running the "Setup commands".

The following script command will make the arm pick up the objects in the field of view of the camera and drop them in a certain location.

```
$ python3 pick_place.py
```

6.3. Arguments

The arguments for the Perception package can be seen in Table 9-6.

Table 9-6: Perception arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
show_gripper_bar	true	If using a custom gripper attachment, set this to “false”
show_gripper_fingers	true	If using custom gripper fingers, set this to “false”
use_rviz	true	Launches RViz
external_urdf_loc	“”	If intending to include a custom urdf.xacro file, include the file path here
load_configs	true	Set to true if the motor configurations shall be written to the motors. Is only necessary to do it the first time the node starts up, reduces startup time if set to “false”
filters	pointcloud	Specifies the type of filters to use with the RealSense camera
color_fps	30	Specifies the capture frame rate for color images on the RealSense camera

depth_fps	30	Specifies the capture frame rate for depth images on the RealSense camera
color_width	640	Specifies the horizontal resolution of the color images captured on the RealSense camera
color_height	480	Specifies the vertical resolution of the color images captured on the RealSense camera
filter_ns	pc_filter	Namespace pointing to the location of the pointcloud related nodes and parameters
filter_params	See the Perception launch package link below	File path to the tuning parameters for the perception pipeline filters
use_pointcloud_tuner_gui	false	Displays a GUI for tuning filter parameters
enable_pipeline	$(arg use_pointcloud_tuner_gui)	Runs the perception pipeline filters continuously. Unless actively tuning the filter parameters this should be set to false, which saves processing power
cloud_topic	/camera/depth/color/points	The path and name of the ROS topic used to subscribe to the raw pointcloud data
tag_family	tagStandard41h12	Specifies which family the utilized AprilTag belongs to
standalone_tags	See the Perception Modules link below	Specifies the individual AprilTags for the algorithm to look for

camera_frame	camera_color_optical_frame	Specifies in which camera frame the AprilTag shall be detected
apriltag_ns	apriltag	Namespace pointing to the location of the ApilTag related nodes and parameters
camera_color_topic	camera/color/image_raw	The path and name of the ROS topic used to subscribe to the color images
camera_info_topic	camera/color/camera_info	The path and name of the ROS topic used to subscribe to the camera color info
armtag_ns	armtag	Namespace pointing to the location of the Armtag related nodes and parameters
ref_frame	$(arg camera_frame)	Specifies the reference frame to be used by the armtag node when publishing a static transform of the arms position relative to the camera
arm_base_frame	$(arg robot_name)/$(arg base_link_frame)	Specifies the child frame to be used by the armtag node when publishing a static transform of the arms position relative to the camera
arm_tag_frame	$(arg robot_name)/ar_tag_link	Specifies the name of the frame where the AprilTag is located, which is usually defined in the URDF for the arm
use_armtag_tuner_gui	false	Enables the user to publish the “ref_frame” to

		"arm_base_frame" transform via a GUI
position_only	false	Specifies when calculating the "ref_frame" to "arm_base_frame" transform, whether to only use the position component of the detected AprilTag pose. If a tf chain connecting the camera and arm base_link frame is already defined in the URDF, and the use of the AprilTag is only to refine the pose further, this can be set to true
load_transforms	true	Specifies if the static_trans_pub node should publish any poses stored in the static_transforms.yaml file at startup. If a" tf" chain connecting the camera and arm base_link frame is already defined in the URDF and intending to use that "tf" chain instead of the one specified in the static_transforms.yaml file, this can be set to false
transform_filepath	See the Perception launch package link below	Specifies the file path to the to the static_transforms.yaml file utilized by the static_trans_pub node.
rviz_frame	$(arg robot_name)/$(arg base_link_frame)	The value set as the fixed frame parameter in RViz

rvizconfig	See the Perception launch package link below	File path of RViz config file

Perception launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_perception/launch/xsarm_perception.launch

Perception Modules link:

https://github.com/Interbotix/interbotix_ros_toolboxes/tree/main/interbotix_perception_toolbox/interbotix_perception_modules

7. Joystick Control

This chapter contains simple commands for getting the Joystick Control package up and running, and the arguments available in package.

Note that this package is recommended to be used with a PS4 controller by default but will also work with a ps3 controller or an Xbox 360 controller. If the controller is not connected to the PC, follow one of these two guides:

[Appendix H – Joystick controller pairing]

https://docs.trossenrobotics.com/interbotix_xsarms_docs/getting_started/pairing_controller.html

7.1. Commands

Launch the package with the physical arm and start controlling the arm with the joystick controller:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150
```

Launch the package with the simulated arm and start controlling the arm with the joystick controller:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150 use_sim:=true
```

7.2. Arguments

The arguments for the Joystick Control package can be seen in Table 9-7.

Table 9-7: Joystick Control arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
use_rviz	true	Launches RViz
mode_configs	See the Joystick Control launch package link below	File path of the YAML file containing the mode configurations

use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
controller	ps4	Defines the type of controller to be used, either “ps3”, “ps4” or “xbox360”
threshold	0.75	Specifies the sensitivity of the controller from 0 to 1, 1 being the highest sensitivity
launch_driver	true	Launches the “xsarm_control.launch” file, if intending to use a custom launch file set this to false

Joystick Control launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_joy/launch/xsarm_joy.launch

7.3. Controls

The controls for the robot can be seen in Table 9-8.

Table 9-8: Joystick controls

Button	Action
START/OPTIONS	Move robot arm to its “Home” pose
SELECT/SHARE	Move robot arm to its “Sleep” pose
R2	Rotate the “waist” joint clockwise
L2	Rotate the “waist” joint counterclockwise
Triangle	Increase gripper pressure in 0.125 step increments (max is 1)
X	Decrease gripper pressure in 0.125 step increments (min is 0)
O	Open gripper
Square	Close gripper
D-pad Up	Increase the control loop rate in 1 Hz step increments (max of 40)
D-pad Down	Decrease the control loop rate in 1 Hz step increments (min of 10)

D-pad Left	Coarse control - sets the control loop rate to a user-preset “fast” rate
D-pad Right	Fine control - sets the control loop rate to a user-preset “slow” rate
Right stick Up/Down	Increase/decrease pitch of the end-effector
Right stick Left/Right	Increase/decrease roll of the end-effector
R3	Reverses the Right stick Left/Right control
Left stick Up/Down	Move the end-effector (defined at ‘ee_gripper_link’) vertically in Cartesian space
Left stick Left/Right	Move the end-effector (defined at ‘ee_gripper_link’) horizontally in Cartesian space
L3	Reverses the Left stick Left/Right control
R1	If the arm has 6dof, this moves the end-effector in a negative direction along its own ‘y’ axis
L1	If the arm has 6dof, this moves the end-effector in a positive direction along its own ‘y’ axis
PS	If torqued on, holding for 3 seconds will torque off the robot; if torqued off, tapping the button will torque on the robot

The controls table can be seen at:
https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/joystick_control.html

8. Record and Playback

This chapter contains simple commands for getting the Record and Playback package up and running, and the arguments available in package.

8.1. Commands

Commands for physical and simulated record and playback

8.1.1. Physical arm

Record the position of the joints while manipulating the physical arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
record:=true
```

Return the arm to its initial position and press “Ctrl + C” to stop the recording.

Playback the recording of the arm on the physical arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
playback:=true
```

8.1.2. Simulated arm

Record the position of the joints while manipulating the simulated arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
use_sim:=true record:=true
```

Return the arm to its initial position and press “Ctrl + C” to stop the recording.

Playback the recording of the arm on the simulated arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
use_sim:=true playback:=true
```

8.2. Arguments

The arguments for the Record and Playback package can be seen in Table 9-9.

Table 9-9: Record and Playback arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary

base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
use_rviz	true	Launches RViz
record	false	Record the physical manipulation of the robot to a bag file
playback	false	Playback the recorded manipulation of the arm
bag_name	$(arg robot_name)_commands	Change this to set an arbitrary file name to the ROS bag file
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
launch_driver	true	Launches the “xsarm_control.launch” file, if intending to use a custom launch file set this to false

9. Arm Diagnostic Tool

This chapter contains simple commands for getting the Arm Diagnostic Tool package up and running, and the arguments available in package.

Note that the diagnostic tool is only meant for the physical arms and that if the “test_duration” argument is not changed, the test will run for 600 seconds.

9.1. Commands

Manipulate the arm to the desired starting position.

Launch the package for commanding and observing the “waist” joint:

```
$ roslaunch interbotix_xsarm_diagnostic_tool xsarm_diagnostic_tool.launch
robot_model:=rx150 cmd_joint:=waist observe_joint:=waist bag_name:=rx150_diagnostics
```

To convert the rosbag data to CSV, navigate to the scripts directory and execute the following commands:

```
$ chmod a+x bag2csv.py
$ python bag2csv.py rx150 waist rx150_diagnostics.bag rx150_diagnostics.csv
```

9.2. Arguments

The arguments for the Arm Diagnostic Tool package can be seen in Table 9-10.

Table 9-10: Arm Diagnostic Tool arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name	“$(arg robot_model)”	Typically the same as “robot_model”, but is arbitrary
base_link_frame	“base_link”	The name of the “root” link on the arm. If attaching the arm to another base, change the name.
use_rviz	true	Launches RViz
mode_configs	See the Arm Diagnostic Tool launch package link below	File path of the YAML file containing the mode configurations
cmd_joint	waist	The joint name of the joint to be rotated

observe_joint	waist	The joint name of the joint to be observed
test_duration	600	Defines the duration of the test in seconds
bag_name	“$(arg observe_joint)_diagnostics”	Change this to set an arbitrary file name to the ROS bag file
use_rqt	true	Launches the rqt plots. The rqt plots are set up with preloaded topics
launch_driver	true	Launches the “xsarm_control.launch” file, if intending to use a custom launch file set this to false

Arm Diagnostic Tool launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_diagnostic_tool/launch/xsarm_diagnostic_tool.launch

10. Arm Puppeteering

This chapter contains simple commands for getting the Arm Puppeteering package up and running, and the arguments available in package.

Connect the two X-Series arms to USB ports on the desired PC. The first robot arm to be connected will be the master.

10.1. Commands

Launch the package with two physical ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet.launch robot_model_master:=rx150
robot_model_puppet:=rx150
```

Start manipulating the “master” arm.

Launch the package with two simulated ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet.launch robot_model_master:=rx150
robot_model_puppet:=rx150 use_sim:=true
```

Send commands to the ROS topic subscribers of the master arm to operate the arms.

10.2. Arguments

The arguments for the Arm Puppeteering package can be seen in Table 9-11.

Table 9-11: Arm Puppeteering arguments

Argument	Default Value	Description
robot_model_master	“”	Requires the codename for the specific robot arm to be the master. “rx150” in the case of the ReactorX-150
robot_model_puppet	“”	Requires the codename for the specific robot arm to be the puppet. “rx150” in the case of the ReactorX-150
base_link_master	“base_link”	The name of the “root” link on the master arm. If attaching the arm to another base, change the name.
base_link_puppet	“base_link”	The name of the “root” link of the puppet arm. If attaching the arm to another base, change the name.
master_modes	See the Arm Puppeteering launch package link below	File path of the YAML file containing the mode configurations for the master arm

puppet_modes	See the Arm Puppeteering launch package link below	File path of the YAML file containing the mode configurations for the puppet arm
use_puppet_rviz	true	launches RViz with visualization of both arms
rvizconfig	See the Arm Puppeteering launch package link below	File path of RViz config file
use_sim	false	Set to "true" if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
launch_driver	true	Launches the "xsarm_control.launch" file, if intending to use a custom launch file set this to false

Arm Puppeteering launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_puppet/launch/xsarm_puppet.launch

11. Dual Arm Control

This chapter contains simple commands for getting the Dual Arm Control package up and running, and the arguments available in package.

Configuring udev rules are not a requirement but can be helpful if there is an error message regarding the connection to the arms.

11.1. Configure udev rules

This is done with one arm connected at a time but shall be done for both arms.

Find the serial of the U2D2 of each of the arms:

```
$ udevadm info -a -n /dev/ttyUSB0 | grep {serial}
```

Copy the serial id's of both of the arms. Navigate to "/etc/udev/rules.d" and edit the "99-interbotix-udev.rules" file with the following lines, inserting the serial id's inside the empty quotation marks:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6014",
ATTRS{serial}=="", ENV{ID_MM_DEVICE_IGNORE}="1", ATTR{device/latency_timer}="1",
SYMLINK+="ttyRBT1"
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6014",
ATTRS{serial}=="", ENV{ID_MM_DEVICE_IGNORE}="1", ATTR{device/latency_timer}="1",
SYMLINK+="ttyRBT2"
```

In the /Interbotix_xsarm_dual/config/" directory, edit the "port" in the "modes_1.yaml" and "modes_2.yaml" files with the new symlinks.

11.2. Commands

Launch the package with the physical arms and RViz:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_dual_rviz:=true
```

Launch the package with the simulated arms:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_dual_rviz:=true use_sim:=true
```

Navigate to the scripts directory in a new terminal window and execute the example script:

```
$ python3 xsarm_dual.py
```

11.3. Arguments

The arguments for the Dual Arm Control package can be seen in Table 9-12.

Table 9-12: Dual Arm Control arguments

Argument	Default Value	Description
robot_model_1	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_model_2	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name_1	“arm_1”	Typically the same as “robot_model_1”, but is arbitrary
robot_name_2	“arm_2”	Typically the same as “robot_model_2”, but is arbitrary
base_link_1	“base_link”	The name of the “root” link on the first arm. If attaching the arm to another base, change the name.
base_link_2	“base_link”	The name of the “root” link on the first arm. If attaching the arm to another base, change the name.
modes_1	See the Dual Arm Control launch package link below	File path of the YAML file containing the mode configurations for the first arm
modes_2	See the Dual Arm Control launch package link below	File path of the YAML file containing the mode configurations for the second arm
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_dual_rviz	false	launches RViz with visualization of both arms

rvizconfig	See the Dual Arm Control launch package link below	File path of RViz config file

Dual Arm Control launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_dual/launch/xsarm_dual.launch

12. Dual Arm Joystick Control

This chapter contains simple commands for getting the Dual Arm Joystick Control package up and running, and the arguments available in package.

Note it is recommended that the udev rules are configured as seen in chapter 11.1. This package is recommended to be used with a PS4 controller. If the controller is not connected to the PC, follow this guide:

https://docs.trossenrobotics.com/interbotix_xsarms_docs/getting_started/pairing_controller.html

12.1. Commands

Launch the package with two physical ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_dual_joy xsarm_dual_joy.launch robot_model_1:=rx150
robot_model_2:=rx150
```

Launch the package with two simulated ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_dual_joy xsarm_dual_joy.launch robot_model_1:=rx150
robot_model_2:=rx150 use_sim:=true
```

12.2. Arguments

The arguments for the Dual Arm Joystick Control package can be seen in Table 9-13.

Table 9-13: Dual Arm Joystick Control arguments

Argument	Default Value	Description
robot_model_1	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_model_2	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_name_1	“arm_1”	Typically the same as “robot_model_1”, but is arbitrary
robot_name_2	“arm_2”	Typically the same as “robot_model_2”, but is arbitrary

base_link_1	“base_link”	The name of the “root” link on the first arm. If attaching the arm to another base, change the name.
base_link_2	“base_link”	The name of the “root” link on the first arm. If attaching the arm to another base, change the name.
modes_1	See the Dual Arm Joystick Control launch package link below	File path of the YAML file containing the mode configurations for the first arm
modes_2	See the Dual Arm Joystick Control launch package link below	File path of the YAML file containing the mode configurations for the second arm
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_rviz	false	Launches RViz with visualization of both arms
rvizconfig	See Dual Arm Joystick Control launch package link below	File path of RViz config file
threshold	0.75	Specifies the sensitivity of the controller from 0 to 1, 1 being the highest sensitivity
controller	ps4	Define the type of controller to be used, either “ps3”, “ps4” or “xbox360”
topic_joy_raw	“/commands/joy_raw”	The path to the topic the xs_arm_joy nodes should subscribe to. The default path provides raw joystick commands

Dual Arm Joystick Control launch package link:
https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/examples/interbotix_xsarm_dual_joy/launch/xsarm_dual_joy.launch

9.6 Appendix F – User Manual with Main Arguments for Trossen Robotics X-Series arms

This user manual aims to serve as a guide for understanding and utilization of the Trossen Robotics X-Series robotic arms, both for simulation and physical usage. It is designed to provide users with an overview of the main command line arguments and package options available for the X-Series arms. The in-depth information about the packages is covered in the thesis.

A user manual including the full list of arguments for each package can be seen in [Appendix E – User Manual with Full Arguments for Trossen Robotics X-Series arms].

Requirements

- A PC running Ubuntu Linux 18.04, 20.04, or 22.04. In this case the computer should be running Ubuntu Linux 20.04.
    - Installation guide: [Appendix B – Guide for Dual Booting Windows and Ubuntu]
- A PC with ROS 1 Noetic and the necessary packages for the Trossen Robotics X-Series arms installed.
    - Installation guide: [Appendix C – ROS Installation Guide for the X-Series Arms from Trossen Robotics]
- At least one but recommending two X-Series robot arms from Trossen Robotics. In this case two ReactorX-150's, codename "rx150".

Packages overview

1. **Arms Descriptions**
2. **Arm Control – Python and MATLAB**
3. **MoveIt Configuration, Interface, and API**
4. **Gazebo Configuration**
5. **ROS Controllers Configuration**
6. **Perception Configuration**
7. **Joystick Control**
8. **Record and Playback**
9. **Arm Diagnostic Tool**
10. **Arm Puppeteering**
11. **Dual Arm Control**
12. **Dual Arm Joystick Control**

Packages

The packages are sorted in no particular order but note that some packages build upon other packages and some packages are not meant to be run as standalone packages. All commands throughout this guide are meant to be run in an Ubuntu terminal window.

Be sure to manually secure the arm(s) before pressing “Ctrl + C” when exiting any package.

1. Arms Descriptions

This chapter contains the commands for getting the Arms Descriptions package up and running and the arguments available in package.

Note that the Arms Descriptions package is a visualization/simulation package only.

1.1. Commands

Launch with GUI for controlling the joints:

```
$ roslaunch interbotix_xsarm_descriptions xsarm_description.launch
robot_model:=rx150 use_joint_pub_gui:=true
```

1.2. Arguments

The arguments for the Arms Description package can be seen in Table 9-14.

Table 9-14: Arms Descriptions arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_joint_pub_gui	false	Launches a user-friendly GUI for controlling joint angles called the joint_state_publisher GUI

2. Arm Control – Python and MATLAB

This chapter contains simple commands for getting the Arm Control package up and running with both Python and MATLAB scripts, and the arguments available in package.

2.1. Commands

Launch with physical robot:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
```

Lauch with simulated robot:

```
$ roslaunch interbotix_xsarm_control xsarm_control.launch robot_model:=rx150
use_sim:=true
```

Navigate to the Python demos script directory. Launch Python script for ROS Noetic:

```
$ python3 bartender.py
```

Navigate to the MATLAB demo script directory. Launch MATLAB script:

```
bartender
```

The bartender scripts for both Python and MATLAB are included upon download of the necessary packages for the X-Series arms.

2.2. Arguments

The arguments for the Arm Control package can be seen in Table 9-15.

Table 9-15: Arm Control arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true

3. MoveIt Configuration, Interface, and API

This chapter contains simple commands for getting the MoveIt Configuration and MoveIt Interface and API package up and running, and the arguments available in package.

3.1. Commands

Lauch with physical robot and the MoveIt Interface GUI:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_actual:=true
```

Lauch with simulated robot in Gazebo and the MoveIt Interface GUI, unpause the Gazebo physics for RViz to load:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_gazebo:=true
```

Lauch with simulated robot in RViz and the MoveIt Interface GUI:

```
$ roslaunch interbotix_xsarm_moveit_interface xsarm_moveit_interface.launch
robot_model:=rx150 use_cpp_interface:=true use_fake:=true
```

There is also an option to run MoveIt without the interface/GUI. The plugin for MoveIt in RViz can be used instead. Note that the MoveIt plugin in RViz is available for use when launching both the “interbotix_xsarm_moveit_interface” and “interbotix_xsarm_moveit” package. Enter the robot code name in the lower left corner of RViz and press the “update” button.

Launch MoveIt without the interface/GUI with physical robot:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_actual:=true
```

Launch MoveIt without the interface/GUI with simulated robot in Gazebo, unpause the Gazebo physics for RViz to load:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_gazebo:=true
```

Launch MoveIt without the interface/GUI with simulated robot in RViz:

```
$ roslaunch interbotix_xsarm_moveit xsarm_moveit.launch robot_model:=rx150
use_fake:=true
```

3.2. Arguments

The arguments for the MoveIt Configuration, Interface and API package can be seen in Table 9-16.

Note that the “use_cpp_interface”, “moveit_interface_gui” and “use_python_interface” arguments only apply to the “interbotix_xsarm_moveit_interface” package, and not the “interbotix_xsarm_moveit” package.

Table 9-16: MoveIt configuration, interface, and API arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_moveit_rviz	true	Launches RViz with the MoveIt plugin
use_gazebo	false	Simulate the robot with Gazebo
use_actual	false	Use the physical robot
use_fake	false	MoveIt generates a simulated robot to be controlled in RViz
dof	5	Defines the degrees of freedom of the robot arm
moveit_interface_gui	true	Launches a GUI customized to interface with the moveit_interface node
use_python_interface	false	Launches a Python Interface Tutorial node. Press “enter” in the terminal window to step through the tutorial

4. Gazebo Configuration

This chapter contains simple commands for getting the Gazebo package up and running, and the arguments available in package.

Note that this package is meant to be run in conjunction with MoveIt or by itself via the JointPositionController interface. The JointPositionController interface is utilized by sending joint position commands to the arm using ROS subscriber topics. Gazebo is a simulation tool, so this package will not interact with the physical robot.

4.1. Commands

Launch the simulated robot with the ability to command arbitrary positions to the arm joints:

```
$ roslaunch interbotix_xsarm_gazebo xsarm_gazebo.launch robot_model:=rx150
use_position_controllers:=true
```

Unpause the physics engine by pressing the “play”-button in the left hand corner of gazebo, or enter the following command:

```
$ rosservice call /gazebo/unpause_physics
```

Controlling the waist of the ReactorX-150 in radians:

```
$ rostopic pub -1 /rx150/waist_controller/command std_msg/Float64 “data: 1.0”
```

List of topics for controlling other joints (subscribers):

```
$ rostopic list -s
```

4.2. Arguments

The arguments for the Gazebo Configuration package can be seen in Table 9-17.

Table 9-17: Gazebo Configurations arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
dof	5	Defines the degrees of freedom of the robot arm
world_name	see the Gazebo launch package link below	File path to the world file to be loaded by Gazebo
gui	true	Launches the GUI of Gazebo
paused	true	Launches Gazebo in a paused state
use_position_controllers	false	Enables commanding of arbitrary arm joint positions in Gazebo
use_trajectory_controllers	false	Enables commanding of arbitrary arm joint trajectories in Gazebo

5. ROS Controllers Configuration

This chapter contains simple commands for getting the ROS controllers package up and running, and the arguments available in package.

Note that this package is only meant to be used in conjunction with MoveIt but can be used with other nodes that are able to communicate with the joint_trajectory_controller package.

5.1. Commands

Launch the package:

```
$ roslaunch interbotix_xsarm_ros_control xsarm_ros_control.launch robot_model:=rx150
```

5.2. Arguments

The arguments for the ROS Controllers Configuration package can be seen in Table 9-18.

Table 9-18: ROS Controllers Configurations arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	false	Launches RViz
dof	5	Defines the degrees of freedom of the robot arm

6. Perception Configuration

This chapter contains simple commands for getting the Perception package up and running and the arguments available in package.

Note that this package is meant to be run with a physical X-Series arm with an AprilTag (AR tag).

6.1. Setup commands

Start by finding the transform of the AprilTag relative to the camera's color optical frame with the following series of commands:

```
$ roslaunch interbotix_xsarm_perception xsarm_perception.launch robot_model:=rx150
use_pointcloud_tuner_gui:=true use_armtag_tuner_gui:=true
```

Manually secure the arm! In a new terminal window, torque the arm off:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
false}"
```

Move the arm so it is visible for the camera and execute the following command:

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
true}"
```

Manually secure the arm! Torque the arm off and place it in the "sleeping position":

```
$ rosservice call /rx150/torque_enable "{cmd_type: 'group', name: 'arm', enable:
false}"
```

6.2. Utilization commands

Launch the package:

```
$ roslaunch interbotix_xsarm_perception xsarm_perception.launch robot_model:=rx200
use rviz:=false
```

Navigate to the scripts directory. Locate the "pick_place.py" script and comment out lines 25-28, since the transform of the arm relative to the camera is already found by running the "Setup commands".

The following script command will make the arm pick up the objects in the field of view of the camera and drop them in a certain location.

```
$ python3 pick_place.py
```

6.3. Arguments

The arguments for the Perception package can be seen in Table 9-19.

Table 9-19: Perception arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
load_configs	true	Set to true if the motor configurations shall be written to the motors. Is only necessary to do it the first time the node starts up, reduces startup time if set to “false”
use_pointcloud_tuner_gui	false	Displays a GUI for tuning filter parameters
use_armtag_tuner_gui	false	Enables the user to publish the “ref_frame” to “arm_base_frame” transform via a GUI

7. Joystick Control

This chapter contains simple commands for getting the Joystick Control package up and running, and the arguments available in package.

Note that this package is recommended to be used with a PS4 controller by default but will also work with a ps3 controller or an Xbox 360 controller. If the controller is not connected to the PC, follow one of these two guides:

- [Appendix H – Joystick controller pairing]
- https://docs.trossenrobotics.com/interbotix_xsarms_docs/getting_started/pairing_controller.html

7.1. Commands

Launch the package with the physical arm and start controlling the arm with the joystick controller:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150
```

Launch the package with the simulated arm and start controlling the arm with the joystick controller:

```
$ roslaunch interbotix_xsarm_joy xsarm_joy.launch robot_model:=rx150 use_sim:=true
```

7.2. Arguments

The arguments for the Joystick Control package can be seen in Table 9-20.

Table 9-20: Joystick Control arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
controller	ps4	Define the type of controller to be used, either “ps3”, “ps4” or “xbox360”
threshold	0.75	Specifies the sensitivity of the controller from 0 to 1, 1 being the highest sensitivity

7.3. Controls

The controls for the robot can be seen in Table 9-21.

Table 9-21: Joystick controls

Button	Action
START/OPTIONS	Move robot arm to its “Home” pose
SELECT/SHARE	Move robot arm to its “Sleep” pose
R2	Rotate the “waist” joint clockwise
L2	Rotate the “waist” joint counterclockwise
Triangle	Increase gripper pressure in 0.125 step increments (max is 1)
X	Decrease gripper pressure in 0.125 step increments (min is 0)
O	Open gripper
Square	Close gripper
D-pad Up	Increase the control loop rate in 1 Hz step increments (max of 40)
D-pad Down	Decrease the control loop rate in 1 Hz step increments (min of 10)
D-pad Left	Coarse control - sets the control loop rate to a user-preset “fast” rate
D-pad Right	Fine control - sets the control loop rate to a user-preset “slow” rate
Right stick Up/Down	Increase/decrease pitch of the end-effector
Right stick Left/Right	Increase/decrease roll of the end-effector
R3	Reverses the Right stick Left/Right control
Left stick Up/Down	Move the end-effector (defined at ‘ee_gripper_link’) vertically in Cartesian space
Left stick Left/Right	Move the end-effector (defined at ‘ee_gripper_link’) horizontally in Cartesian space
L3	Reverses the Left stick Left/Right control

R1	If the arm has 6dof, this moves the end-effector in a negative direction along its own ‘y’ axis
L1	If the arm has 6dof, this moves the end-effector in a positive direction along its own ‘y’ axis
PS	If torqued on, holding for 3 seconds will torque off the robot; if torqued off, tapping the button will torque on the robot

The controls table can be seen at:
https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros1_packages/joystick_control.html

8. Record and Playback

This chapter contains simple commands for getting the Record and Playback package up and running, and the arguments available in package.

8.1. Commands

Commands for physical and simulated record and playback

8.1.1. Physical arm

Record the position of the joints while manipulating the physical arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
record:=true
```

Return the arm to its initial position and press “Ctrl + C” to stop the recording.

Playback the recording of the arm on the physical arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
playback:=true
```

8.1.2. Simulated arm

Record the position of the joints while manipulating the simulated arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
use_sim:=true record:=true
```

Return the arm to its initial position and press “Ctrl + C” to stop the recording.

Playback the recording of the arm on the simulated arm:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet_single.launch robot_model:=rx150
use_sim:=true playback:=true
```

8.2. Arguments

The arguments for the Record and Playback package can be seen in Table 9-22.

Table 9-22: Record and Playback arguments

Argument	Default Value	Description
robot_model	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_rviz	true	Launches RViz
record	false	Record the physical manipulation of the robot to a bag file
playback	false	Playback the recorded manipulation of the arm
bag_name	$(arg robot_name)_commands	Change this to set an arbitrary file name to the ROS bag file
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true

9. Arm Diagnostic Tool

This chapter contains simple commands for getting the Arm Diagnostic Tool package up and running, and the arguments available in package.

Note that the diagnostic tool is only meant for the physical arms and that if the "test_duration" argument is not changed, the test will run for 600 seconds.

9.1. Commands

Manipulate the arm to the desired starting position.

Launch the package for commanding and observing the "waist" joint:

```
$ roslaunch interbotix_xsarm_diagnostic_tool xsarm_diagnostic_tool.launch
robot_model:=rx150 cmd_joint:=waist observe_joint:=waist bag_name:=rx150_diagnostics
```

To convert the rosbag data to CSV, navigate to the scripts directory and execute the following commands:

```
$ chmod a+x bag2csv.py
$ python bag2csv.py rx150 waist rx150_diagnostics.bag rx150_diagnostics.csv
```

9.2. Arguments

The arguments for the Arm Diagnostic Tool package can be seen in Table 9-23.

Table 9-23: Arm Diagnostic Tool arguments

Argument	Default Value	Description
robot_model	""	Requires the codename for the specific robot arm. "rx150" in the case of the ReactorX-150
use_rviz	true	Launches RViz
cmd_joint	waist	The joint name of the joint to be rotated
observe_joint	waist	The joint name of the joint to be observed
test_duration	600	Defines the duration of the test in seconds
bag_name	"$(arg observe_joint)_diagnostics"	Change this to set an arbitrary file name to the ROS bag file
use_rqt	true	Launches the rqt plots. The rqt plots are set up with preloaded topics

10. Arm Puppeteering

This chapter contains simple commands for getting the Arm Puppeteering package up and running, and the arguments available in package.

Connect the two X-Series arms to USB ports on the desired PC. The first robot arm to be connected will be the master.

10.1. Commands

Launch the package with two physical ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet.launch robot_model_master:=rx150
robot_model_puppet:=rx150
```

Start manipulating the “master” arm.

Launch the package with two simulated ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_puppet xsarm_puppet.launch robot_model_master:=rx150
robot_model_puppet:=rx150 use_sim:=true
```

Send commands to the ROS topic subscribers of the master arm to operate the arms.

10.2. Arguments

The arguments for the Arm Puppeteering package can be seen in Table 9-24.

Table 9-24: Arm Puppeteering arguments

Argument	Default Value	Description
robot_model_master	“”	Requires the codename for the specific robot arm to be the master. “rx150” in the case of the ReactorX-150
robot_model_puppet	“”	Requires the codename for the specific robot arm to be the puppet. “rx150” in the case of the ReactorX-150
use_puppet_rviz	true	launches RViz with visualization of both arms
use_rviz	true	Launches RViz

11. Dual Arm Control

This chapter contains simple commands for getting the Dual Arm Control package up and running, and the arguments available in package.

Configuring udev rules are not a requirement but can be helpful if there is an error message regarding the connection to the arms.

11.1. Configure udev rules

This is done with one arm connected at a time but shall be done for both arms.

Find the serial of the U2D2 of each of the arms:

```
$ udevadm info -a -n /dev/ttyUSB0 | grep {serial}
```

Copy the serial id's of both of the arms. Navigate to "/etc/udev/rules.d" and edit the "99-interbotix-udev.rules" file with the following lines, inserting the serial id's inside the empty quotation marks:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6014",
ATTRS{serial}=="", ENV{ID_MM_DEVICE_IGNORE}="1", ATTR{device/latency_timer}="1",
SYMLINK+="ttyRBT1"
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6014",
ATTRS{serial}=="", ENV{ID_MM_DEVICE_IGNORE}="1", ATTR{device/latency_timer}="1",
SYMLINK+="ttyRBT2"
```

In the /Interbotix_xsarm_dual/config/" directory, edit the "port" in the "modes_1.yaml" and "modes_2.yaml" files with the new symlinks.

11.2. Commands

Launch the package with the physical arms and RViz:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_dual_rviz:=true
```

Launch the package with the simulated arms:

```
$ roslaunch interbotix_xsarm_dual xsarm_dual.launch robot_model_1:=rx150
robot_model_2:=rx150 use_dual_rviz:=true use_sim:=true
```

Navigate to the scripts directory in a new terminal window and execute the example script:

```
$ python3 xsarm_dual.py
```

11.3. Arguments

The arguments for the Dual Arm Control package can be seen in Table 9-25.

Table 9-25: Dual Arm Control arguments

Argument	Default Value	Description
robot_model_1	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_model_2	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_dual_rviz	false	launches RViz with visualization of both arms

12. Dual Arm Joystick Control

This chapter contains simple commands for getting the Dual Arm Joystick Control package up and running, and the arguments available in package.

Note it is recommended that the udev rules are configured as seen in chapter 11.1. This package is recommended to be used with a PS4 controller. If the controller is not connected to the PC, follow one of these two guides:

- [Appendix H – Joystick controller pairing]
- https://docs.trossenrobotics.com/interbotix_xsarms_docs/getting_started/pairing_controller.html

12.1. Commands

Launch the package with two physical ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_dual_joy xsarm_dual_joy.launch robot_model_1:=rx150
robot_model_2:=rx150
```

Launch the package with two simulated ReactorX-150 arms:

```
$ roslaunch interbotix_xsarm_dual_joy xsarm_dual_joy.launch robot_model_1:=rx150
robot_model_2:=rx150 use_sim:=true
```

12.2. Arguments

The arguments for the Dual Arm Joystick Control package can be seen in Table 9-26.

Table 9-26: Dual Arm Joystick Control arguments

Argument	Default Value	Description
robot_model_1	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
robot_model_2	“”	Requires the codename for the specific robot arm. “rx150” in the case of the ReactorX-150
use_sim	false	Set to “true” if intending to simulate the robot arm. Runs the DYNAMIXEL simulator node if set to true
use_dual_rviz	false	Launches RViz with visualization of both arms

threshold	0.75	Specifies the sensitivity of the controller from 0 to 1, 1 being the highest sensitivity
controller	ps4	Define the type of controller to be used, either “ps3”, “ps4” or “xbox360”

9.7 Appendix G – Joystick controls

The X-series robotic arms from Trossen Robotics can be controlled with joystick controllers when utilizing the Joystick Control package and the Dual Arm Joystick Control package. The joystick controllers suitable for controlling the arms are the PlayStation 4, PlayStation 3 and Xbox360 controllers. How the controllers interact with the arms can be seen in Table 9-27.

Table 9-27: Joystick controls

Button	Action
START/OPTIONS	Move robot arm to its “Home” pose
SELECT/SHARE	Move robot arm to its “Sleep” pose
R2	Rotate the “waist” joint clockwise
L2	Rotate the “waist” joint counterclockwise
Triangle	Increase gripper pressure in 0.125 step increments (max is 1)
X	Decrease gripper pressure in 0.125 step increments (min is 0)
O	Open gripper
Square	Close gripper
D-pad Up	Increase the control loop rate in 1 Hz step increments (max of 40)
D-pad Down	Decrease the control loop rate in 1 Hz step increments (min of 10)
D-pad Left	Coarse control - sets the control loop rate to a user-preset “fast” rate
D-pad Right	Fine control - sets the control loop rate to a user-preset “slow” rate
Right stick Up/Down	Increase/decrease pitch of the end-effector
Right stick Left/Right	Increase/decrease roll of the end-effector
R3	Reverses the Right stick Left/Right control
Left stick Up/Down	Move the end-effector (defined at ‘ee_gripper_link’) vertically in Cartesian space
Left stick Left/Right	Move the end-effector (defined at ‘ee_gripper_link’) horizontally in Cartesian space

L3	Reverses the Left stick Left/Right control
R1	If the arm has 6dof, this moves the end-effector in a negative direction along its own ‘y’ axis
L1	If the arm has 6dof, this moves the end-effector in a positive direction along its own ‘y’ axis
PS	If torqued on, holding for 3 seconds will torque off the robot; if torqued off, tapping the button will torque on the robot

9.8 Appendix H – Joystick controller pairing

The X-Series robotic arms from Trossen Robotics have the ability to be controlled with a Bluetooth joystick controller. This guide covers the setup of the SONY PlayStation 3 and 4 wireless controllers to an Ubuntu Linux computer. It is stated from Trossen Robotics that controlling the arms with a Microsoft Xbox 360 could work, but it is neither tested nor documented.

Requirements

- A computer with Ubuntu Linux installed and Bluetooth capability.
- A SONY PlayStation 3 or 4 wireless controller.
- A USB to Mini USB cable if intending to use a SONY PlayStation 3 wireless controller.

Procedure

Follow the subsequent procedure intended for the SONY PlayStation 3 or 4 wireless controller. All commands throughout this guide are meant to be run in a Ubuntu terminal window.
Note that the setup for the controller should only be required to be done once unless connection to another device is made between each time.

1) SONY PlayStation 4 wireless controller setup

a) Enter the Bluetooth setting on the computer by clicking in the top right corner of the screen, click the Bluetooth icon and select "Bluetooth Settings".
b) Press and hold the "Share" button while simultaneously pressing and holding the PlayStation button.
c) Release when the triangular shaped LED on the frontside of the controller starts rapidly flashing white.
d) A device named "Wireless Controller" should pop up shortly.
e) Click on the "Wireless Controller" and wait for it to say "connected". The triangular shaped LED on the frontside of the controller should now turn blue.
f) The PlayStation 4 controller is now connected.

2) SONY PlayStation 3 wireless controller setup

a) Open a terminal Linux window by pressing "Ctrl + Alt + T" and enter:

```
$ sudo bluetoothctl
[bluetooth]# power on
[bluetooth]# agent on
[bluetooth]# scan on
```

a) Connect the PlayStation 3 controller to the computer with the USB cable and something similar to the following line should appear (with its own MAC address):

```
[NEW] Device 42:06:9Q:T3:66:S5 PLAYSTATION(R)3 Controller
```

b) Then type in the same terminal:

```
[bluetooth]# trust <MAC-address>
```

c) Disconnect the USB cable and press the PlayStation button. The four LEDs at the front side of the controller should flash red a couple of times. When one red LED remains lit, the controller is paired successfully.

Troubleshooting

1) **Previously connected SONY PlayStation 4 wireless controller not connecting**
   a) Press and hold the “Share” button while simultaneously pressing and holding the PlayStation button.
   b) Release when the triangular shaped LED on the frontside of the controller starts rapidly flashing white.
   c) Select the device and click the “Connect” toggle button.
      i) If the controller won’t connect, remove the device, and retry from point 1).

2) **Cursor goes crazy when connecting a PlayStation 3 controller**
   a) Open the “~/.bashrc” file, using for example nano, with the following command:
   ```
   $ sudo nano ~/.bashrc
   ```
   b) Add the following line to the “~/.bashrc” file, save it and close the file:
   ```
   alias joy_stop='xinput set-prop "PLAYSTATION(R)3 Controller" "Device Enabled" 0'
   ```
   c) From now on if the cursor goes crazy, type the following command in the terminal:
   ```
   $ joy_stop
   ```

3) **Controller is not turning on or LEDs flashes a few times before turning off**
   a) Try charging the controller for a while, or connect it to the computer with a cable.

9.9 Appendix I – Raspberry Pi Ubuntu and ROS Setup Guide

The X-Series arms from Trossen Robotics can also be controlled from a Raspberry Pi. This guide provides instructions to setup the Raspberry Pi with Ubuntu Linux with MATE desktop and subsequently setting up ROS (Robot Operating System).

Note that this guide is designed for a Raspberry Pi 4 Model B. The guide has been tested with a Raspberry Pi 3 Model B, but without any promising results.

Requirements

- Raspberry Pi 4 Model B
- USB-C power cable
- Mini-to-full-size HDMI cable
- Monitor with HDMI capability
- USB keyboard
- USB mouse
- Ethernet cable
- Computer with Wi-Fi connection, BalenaEtcher installed and an SD card port
- Full size HDMI cable (Raspberry Pi 3)
- Mini-USB power cable (Raspberry Pi 3)

Procedure

Follow the subsequent procedure in the following order. All commands throughout this guide are meant to be run in a Ubuntu terminal window.

1) Installing Ubuntu

a) Download the Ubuntu 20.04.2 64-bit Server Image from: https://ubuntu.com/download/raspberry-pi/thank-you?version=20.04.2&architecture=server-arm64+raspi
b) Insert the SD card adapter with the micro-SD card and open BalenaEtcher
c) In BalenaEtcher click “Select Image” to browse and select the Ubuntu 20.04.2 64-bit Server Image.
d) Ensure the correct SD card is selected under “Select target”.
e) Click “Flash!” to begin the process. Once completed, safely eject the USB drive.
f) **Before connecting power to the Raspberry Pi!** Connect the mouse, keyboard, HDMI to the monitor and the Ethernet from to the computer.
g) Connect the power cable to the Raspberry Pi and wait for the option to log in.
h) The default computer and username is “ubuntu”, so type in “ubuntu” and create a password.
i) Once logged in check the internet connection by pinging a website such as “www.vg.no” with the following command:

```
$ ping www.vg.no
```

j) If internet connection has been established, move on to step number 3, if not follow step number 2.

2) **Allow sharing internet connection over ethernet**
   a) On Windows 10:
      i) Press the "Windows" button, type in "View network connections" and select it.
      ii) Right click the active internet connection and select "Properties".
      iii) Select the "Sharing" tab in the window that just popped up.
      iv) Toggle the "Allow other network users to connect through this computer's Internet connection" checkbox to "checked".
      v) In the dropdown menu select the correct ethernet port and press "OK".
      vi) The Raspberry Pi should now have internet connection.
   b) On Ubuntu Linux 20.04:
      i) Go to the "Setting" and the "Network" tab.
      ii) Under the "Wired" section click the "+" sign.
      iii) Give the new setting profile a new name like "Shared connection".
      iv) Navigate to the "IPv4" tab and select "Shared to other computers" in the dropdown menu.
      v) Click "Apply" and the Raspberry Pi should now have internet connection.
   c) Check the internet connection on the raspberry pi by retrying to ping "www.vg.no":

```
$ ping www.vg.no
```

3) **Modifying username (optional)**
   Change the username of the Raspberry Pi to something other than "ubuntu", for example "interbotix".
   a) Add a temporary user to execute modifications:

```
$ sudo adduser tempusr
$ sudo adduser tempusr sudo
$ exit
```

   b) Log in to the new "tempusr" user to make the following modifications:

```
$ sudo usermod -l interbotix ubuntu
$ sudo groupmod -n interbotix ubuntu
$ sudo usermod -d /home/interbotix -m interbotix
$ sudo usermod -c "interbotix" interbotix
$ exit
```

   c) Log into the new "interbotix" user and delete the "tempusr" as it is no longer needed:

```
$ sudo deluser tempusr
$ sudo rm -r /home/tempusr
```

   d) Change the "hostname" if intending to have multiple Raspberry Pi's on the same network with the next steps. Change the "hostname" in the following file:

```
$ sudo nano /etc/hostname
```

   e) Check that Linux is not updating by typing the following command:

```
$ ps aux | grep -i apt
```

   f) If "apt.systemd.daily" is using the "apt" process, run the previous command every few minutes until it doesn't.
   g) Then finally reboot:

```
$ sudo reboot
```

4) Install MATE desktop

a) Login and start by updating and upgrading with the following commands:

```
$ sudo apt update && sudo apt upgrade
$ sudo reboot
```

b) Login again and install MATE desktop with the following commands:

```
$ sudo apt install ubuntu-mate-desktop
$ sudo reboot
```

The installation may take around 20 minutes, and at boot there should now be a login screen.

c) Before logging in after reboot, click the Ubuntu sign next to the username text box. Select "MATE" as desktop environment and then login.

5) Enable automatic login (optional)

a) Click in the top right corner of the screen and select "System Settings".
b) Click "Login Window" and navigate to the "Users" tab.
c) In the Automatic login text box enter the username ("interbotix")
d) Exit and done.

6) (Do NOT do this step unless the Raspberry Pi has an onboard heat sink and an active cooling mechanism) Overclocking Raspberry Pi CPU

Increases the CPU clock frequency from 1.5 GHz to 2.0 GHz.

a) Modify the boot config file with the following commands:

```
$ cd /boot/firmware/
$ sudo nano usercfg.txt
```

b) Add the following lines to the boot config file:

```
over_voltage=6
arm_freq=2000
```

c) Save the changes to the file ("Ctrl + S") and exit nano ("Ctrl + X")
d) Reboot:

```
$ sudo reboot
```

7) Disable password for sudo privileges (optional)

For convenience.

a) The following commands opens a file:

```
$ sudo visudo
```

b) Add the following line to disable password for sudo privileges for the "interbotix" user:

```
interbotix ALL=(ALL) NOPASSWD:ALL
```

8) Enable Bluetooth

The Bluetooth module is disabled by default for some reason.

a) Enable the Bluetooth module on Ubuntu 20.04 with the following command:

```
$ sudo apt install pi-bluetooth
```

9) Install ROS

a) Install ROS with robot-specific packages, and pre-configured drivers and environment variables from Interbotix:

```
$ sudo apt install curl
$ curl
'https://raw.githubusercontent.com/Interbotix/interbotix_ros_manipulators/main/inter
botix_ros_xsarms/install/rpi4/xsarm_rpi4_install.sh' > xsarm_rpi4_install.sh
$ chmod +x xsarm_rpi4_install.sh
$ ./xsarm_rpi4_install.sh
```

The Raspberry Pi should now be ready to use with the X-Series arms.

9.10 Appendix J – bartender.py

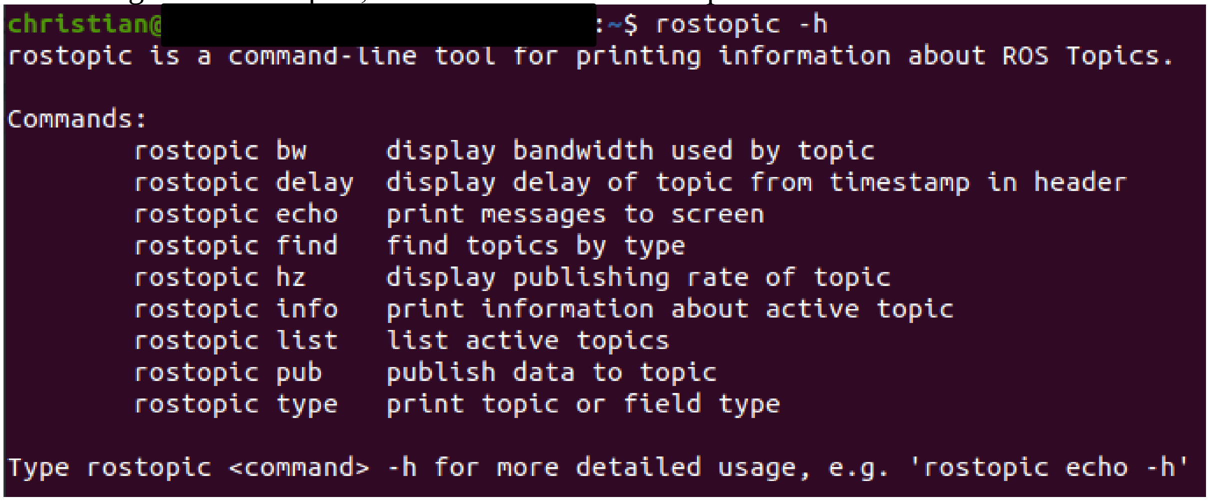
```
from interbotix_xs_modules.arm import InterbotixManipulatorXS
import numpy as np

# This script makes the end-effector perform pick, pour, and place tasks
#
# To get started, open a terminal and type 'roslaunch interbotix_xsarm_control
xsarm_control.launch robot_model:=wx250'
# Then change to this directory and type 'python bartender.py  # python3
bartender.py if using ROS Noetic'

def main():
    bot = InterbotixManipulatorXS("wx250", "arm", "gripper")
    bot.arm.set_ee_pose_components(x=0.3, z=0.2)
    bot.arm.set_single_joint_position("waist", np.pi/2.0)
    bot.gripper.open()
    bot.arm.set_ee_cartesian_trajectory(x=0.1, z=-0.16)
    bot.gripper.close()
    bot.arm.set_ee_cartesian_trajectory(x=-0.1, z=0.16)
    bot.arm.set_single_joint_position("waist", -np.pi/2.0)
    bot.arm.set_ee_cartesian_trajectory(pitch=1.5)
    bot.arm.set_ee_cartesian_trajectory(pitch=-1.5)
    bot.arm.set_single_joint_position("waist", np.pi/2.0)
    bot.arm.set_ee_cartesian_trajectory(x=0.1, z=-0.16)
    bot.gripper.open()
    bot.arm.set_ee_cartesian_trajectory(x=-0.1, z=0.16)
    bot.arm.go_to_home_pose()
    bot.arm.go_to_sleep_pose()

if __name__=='__main__':
    main()
```

9.11 Appendix K – “rostopic” guide

This guide provides a basic guide on how to control an X-Series arm with “rostopic” commands from the Ubuntu terminal. The X-Series robotic arms can be controlled by “rostopic” commands when launching most packages utilizing the “interbotix_xsarm_control” package and/or the “interbotix_xsarm_gazebo” package.

1) Find the rostopic command for listing active topics

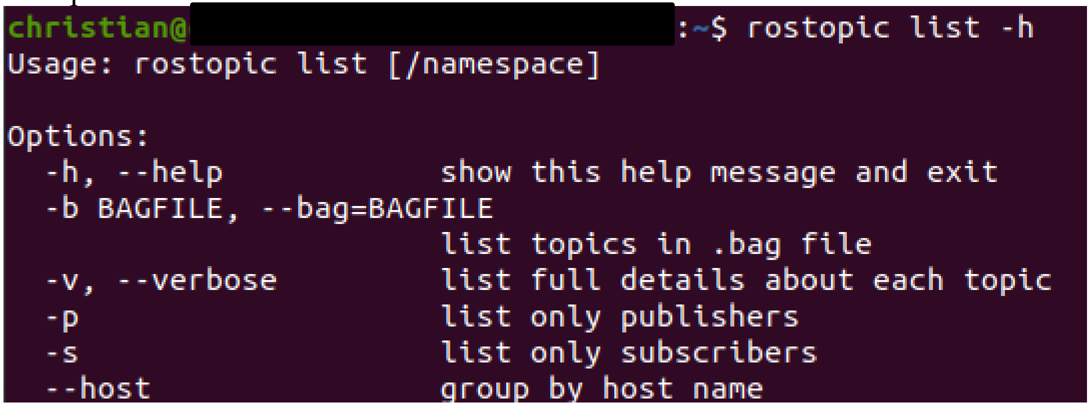
As seen in the figure below, utilize the “rostopic -h” command to locate the command for finding the active topics, in this case it is the “rostopic list” command.



```
christian@[redacted]:~$ rostopic -h
rostopic is a command-line tool for printing information about ROS Topics.

Commands:
        rostopic bw     display bandwidth used by topic
        rostopic delay  display delay of topic from timestamp in header
        rostopic echo   print messages to screen
        rostopic find   find topics by type
        rostopic hz     display publishing rate of topic
        rostopic info   print information about active topic
        rostopic list   list active topics
        rostopic pub    publish data to topic
        rostopic type   print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

2) Filter the “rostopic list” command

As seen in the figure below, utilize the "rostopic list -h” command to see the available options. The rostopics receiving commands are subscribers, so filter the list with the “rostopic list -s” command.

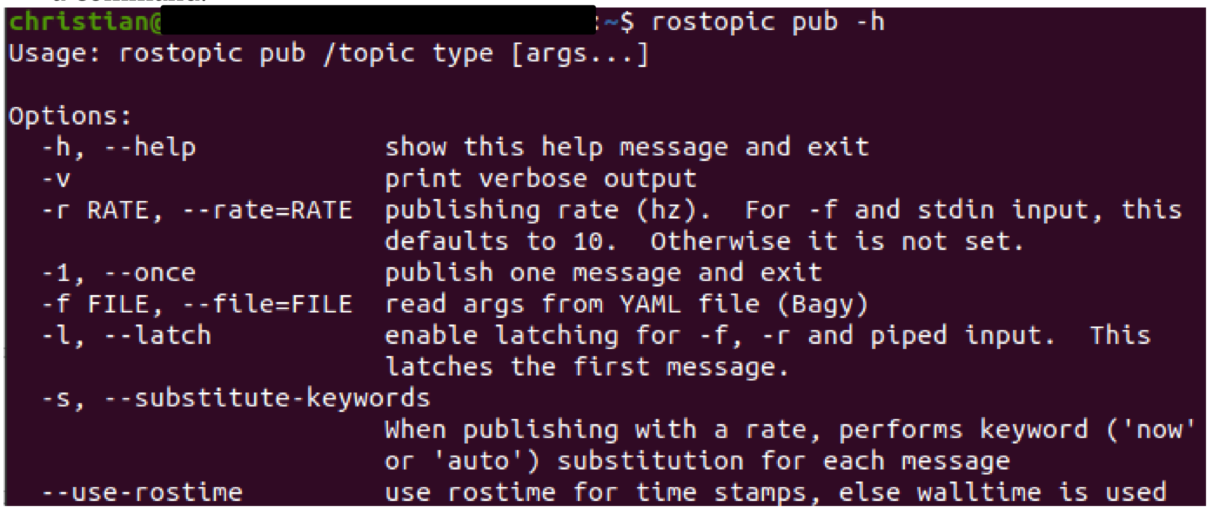


```
christian@[redacted]:~$ rostopic list -h
Usage: rostopic list [/namespace]

Options:
  -h, --help            show this help message and exit
  -b BAGFILE, --bag=BAGFILE
                        list topics in .bag file
  -v, --verbose         list full details about each topic
  -p                    list only publishers
  -s                    list only subscribers
  --host                group by host name
```

Locate the desired subscriber topic before moving to the next step.

3) How to publish the command

As seen in the figure below, utilize the “rostopic pub -h” to view the options for publishing a command.



```
christian@[redacted]:~$ rostopic pub -h
Usage: rostopic pub /topic type [args...]

Options:
  -h, --help            show this help message and exit
  -v                    print verbose output
  -r RATE, --rate=RATE  publishing rate (hz).  For -f and stdin input, this
                        defaults to 10.  Otherwise it is not set.
  -1, --once            publish one message and exit
  -f FILE, --file=FILE  read args from YAML file (Bagy)
  -l, --latch           enable latching for -f, -r and piped input.  This
                        latches the first message.
  -s, --substitute-keywords
                        When publishing with a rate, performs keyword ('now'
                        or 'auto') substitution for each message
  --use-rostime         use rostime for time stamps, else walltime is used
```

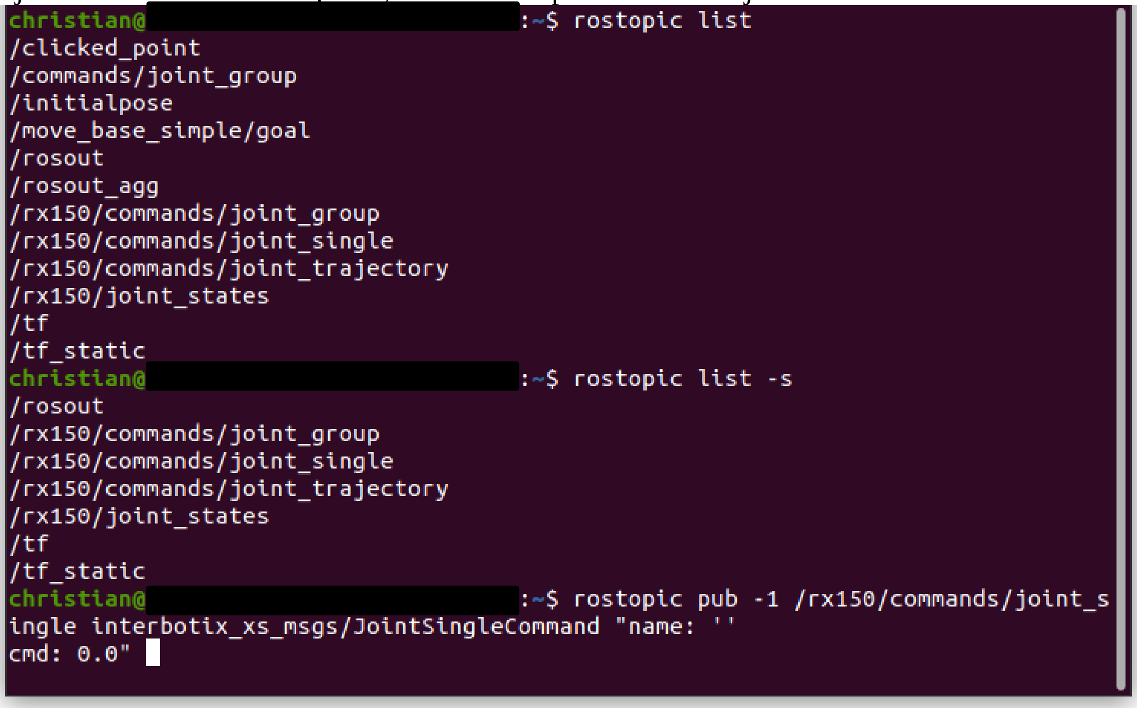
In this case the command should be published only once, so the “rostopic pub -1” shal be used.

4) Publishing the command

The figure below shows the unfiltered “rostopic” list first, then filters it on subscribers. The “/rx150/commands/joint_single” topic is chosen.

After typing in “rostopic pub -1 /<the selected topic>” press “Tab” twice and it will autocomplete the command. Then fill in the necessary information.

For the command in the figure below, the only thing left to do is to fill in the “name” of the joint to command and specify the desired position of the joint in radians.



```
christian@[redacted]:~$ rostopic list
/clicked_point
/commands/joint_group
/initialpose
/move_base_simple/goal
/rosout
/rosout_agg
/rx150/commands/joint_group
/rx150/commands/joint_single
/rx150/commands/joint_trajectory
/rx150/joint_states
/tf
/tf_static
christian@[redacted]:~$ rostopic list -s
/rosout
/rx150/commands/joint_group
/rx150/commands/joint_single
/rx150/commands/joint_trajectory
/rx150/joint_states
/tf
/tf_static
christian@[redacted]:~$ rostopic pub -1 /rx150/commands/joint_s
ingle interbotix_xs_msgs/JointSingleCommand "name: ''
cmd: 0.0"
```

9.12 Appendix L – xsarm_dual_usn_lift.py

```
import math
import time
import rospy
from threading import Thread
from interbotix_xs_modules.arm import InterbotixManipulatorXS

# This script is used to make two ReactorX150 lift and hold the 3D-printed USN logo
#
# To get started, open a terminal and type 'roslaunch interbotix_xsarm_dual
xsarm_dual.launch'
# Note that the 'robot_name' argument used when instantiating an
InterbotixManipulatorXS instance
# is the same name as the 'robot_name_X' launch file argument

speed = 1 #s
sleep_time = 0.1
pickup_y_pose = 0.21 #0.235 when running the physical ReactorX-150 with 60cm from
center to center
pickup_z_pose = 0.05
pickup_pitch = 0.5

def robot_1_lift():
    global wait_for_robot_1

    robot_1 = InterbotixManipulatorXS(robot_model="rx150", robot_name="arm_1",
moving_time=speed, gripper_pressure=1.0, init_node=False)

    wait_for_robot_1 = True
    robot_1.arm.go_to_home_pose()
    robot_1.arm.set_ee_pose_components(y=pickup_y_pose, z=0.2)
    robot_1.gripper.open()
    robot_1.arm.set_ee_pose_components(y=pickup_y_pose, z=pickup_z_pose,
pitch=pickup_pitch)
    robot_1.gripper.close()
    robot_1.arm.set_ee_pose_components(y=pickup_y_pose, z=0.3, pitch=pickup_pitch)
    wait_for_robot_1 = False

def robot_2_lift():
    global wait_for_robot_2

    wait_for_robot_2 = True
    robot_2 = InterbotixManipulatorXS(robot_model="rx150", robot_name="arm_2",
moving_time=speed, gripper_pressure=1.0, init_node=False)

    robot_2.arm.go_to_home_pose()
    robot_2.arm.set_ee_pose_components(y=-pickup_y_pose, z=0.2)
    robot_2.gripper.open()
    robot_2.arm.set_ee_pose_components(y=-pickup_y_pose, z=0.3, pitch=pickup_pitch)
    while wait_for_robot_1:
        time.sleep(sleep_time)
    robot_2.gripper.close()

def main():
    rospy.init_node("xsarm_dual")
    Thread(target=robot_1_lift).start()
    Thread(target=robot_2_lift).start()

if __name__=='__main__':
    main()
```

9.13 Appendix M – xsarm_dual_usn_down.py

```python
import math
import time
import rospy
from threading import Thread
from interbotix_xs_modules.arm import InterbotixManipulatorXS

# This script is used to make two ReactorX150 put down the 3D-printed USN logo
#
# To get started, open a terminal and type 'roslaunch interbotix_xsarm_dual xsarm_dual.launch'
# Note that the 'robot_name' argument used when instantiating an InterbotixManipulatorXS instance
# is the same name as the 'robot_name_X' launch file argument

speed = 1 #[s]
sleep_time = 0.1 #[s]
pickup_y_pose = 0.21 #0.235 when running the physical ReactorX-150 with 60cm from center to center
pickup_z_pose = 0.05
pickup_pitch = 0.5

def robot_1_down():
    global wait_for_robot_1
    global robot_1

    wait_for_robot_1 = True
    robot_1 = InterbotixManipulatorXS(robot_model="rx150", robot_name="arm_1", moving_time=speed,
gripper_pressure=1.0, init_node=False)

    robot_1.gripper.open()
    wait_for_robot_1 = False
    while wait_for_robot_2:
        time.sleep(sleep_time)
    robot_1.arm.set_ee_pose_components(y=pickup_y_pose, z=0.2, moving_time=speed)
    robot_1.gripper.close()
    robot_1.arm.go_to_home_pose()
    robot_1.arm.go_to_sleep_pose()

def robot_2_down():
    global wait_for_robot_2
    global robot_2

    wait_for_robot_2 = True
    robot_2 = InterbotixManipulatorXS(robot_model="rx150", robot_name="arm_2", moving_time=speed,
gripper_pressure=1.0, init_node=False)

    while wait_for_robot_1:
        time.sleep(sleep_time)
    robot_2.arm.set_ee_pose_components(y=-pickup_y_pose, z=pickup_z_pose, pitch=pickup_pitch,
moving_time=speed*2)
    robot_2.gripper.open()
    wait_for_robot_2 = False
    robot_2.arm.set_ee_pose_components(y=-pickup_y_pose, z=0.2, moving_time=speed)
    robot_2.gripper.close()
    robot_2.arm.go_to_home_pose()
    robot_2.arm.go_to_sleep_pose()

def main():
    rospy.init_node("xsarm_dual")
    Thread(target=robot_1_down).start()
    Thread(target=robot_2_down).start()

if __name__=='__main__':
    main()
```