



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MUNEEB-UL-HAQ
IMPLEMENTATION OF ROS-ENABLED INDUSTRIAL PRODUC-
TION ENVIRONMENT

Master of Science Thesis

Examiner: Associate Professor Minna Lanz, Dr. Niko Siltala
Examiner and topic approved on 31st May, 2017

ABSTRACT

MUNEEB-UL-HAQ: Implementation of ROS-enabled Industrial Production Environment

Master of Science Thesis, 61 pages, 02 Appendix pages, August 2017

Master's Degree Programme in Automation Engineering

Major: Factory Automation and Industrial Informatics

Examiner: Associate Professor Minna Lanz, Dr. Niko Siltala

Keywords: Industrial robot modelling, ROS, ROS-Industrial, MoveIt!, 3D perception, Environment adaptation & reaction

This thesis presents the procedure to implement the ROS-Industrial architecture into the industrial robotic environment. This is implemented by developing the packages related to modeling and configuring the industrial robot with ROS-I. Using the ROS capabilities, libraries and tools, this thesis develops an industrial robot application which enables the industrial robot to adapt and react to its workspace changes. This implies that it acknowledges the presence of obstacles in its path and re-plans an alternative path in order to avoid those. The obstacles may be the objects or the humans working near the robot. This system enables the industrial robot to exhibit the flexible movements in a fixed as well as dynamic environments. The fixed objects present in the robot workspace are created manually in the ROS environment while the dynamic objects are brought into the ROS environment by integrating the 3D sensor (camera) with the ROS.

Moreover, this thesis presents the mechanism for ROS integrated gripper control of the ABB IRB4600 robot. It provides the analysis on the motion planners for selecting the best motion planner for ABB IRB4600 robot. Furthermore, this research establishes the framework for letting the industrial robots out of their cages and enables the continuation of the robot operation while sharing the workspace with human operators. Safety of those operators is supported in this system with its collision avoidance feature. This feature provides an additional safety measure to human workers along with the already configured safety standards (ISO/TS 15066, ISO 10218-1 & 10218-2) of human-robot collaboration.

PREFACE

First of all, I am thankful to Almighty ALLAH for giving me the courage and determination for accomplishing this task. This thesis is completed at Department of Mechanical Engineering and Industrial Systems (MEI), Tampere University of Technology.

I owe my gratitude to Associate Prof. Minna Lanz for providing me the opportunity to work in her team. I thank to my supervisor Dr. Niko Siltala for his immense support and valuable inputs throughout this thesis. I am thankful to my advisors Alireza Changizi, Antti Hietanen, Jussi Halme and Jyrki Latokartano for their assistance at various stages. I am grateful to my friends Muhammad Hashim Abbas and Adnan Ali for motivating and helping me whenever I need.

Finally, this effort goes to my whole family especially my parents for their unconditional blessings and encouragement. I am in their debt for being where I am today.

Tampere, 02.8.2017

Muneeb-Ul-Haq

CONTENTS

1.	INTRODUCTION	1
1.1	Overview	1
1.2	Objective	2
1.3	Limitations	2
1.4	Research Methodology	3
1.5	Thesis Outline	4
2.	ROBOTS SHARING WORKSPACE WITH HUMANS IN INDUSTRY	5
2.1	Industrial Robots	5
2.2	Safety against Industrial Robot and Human Interventions	6
2.3	Safety Standards for Human and Robot Interaction	6
3.	ROBOT SOFTWARE FRAME WORK	8
3.1	ROS	8
3.1.1	Background	8
3.1.2	Why ROS?	8
3.1.3	Misconceptions about ROS	10
3.1.4	ROS Pre-requisites	10
3.1.5	Distributions of ROS	10
3.1.6	Levels of ROS	11
3.2	ROS-Industrial	14
3.2.1	Brief History about ROS-I	14
3.2.2	Intended Goals of ROS-I	15
3.2.3	ROS-I Block Diagram	15
3.2.4	Packages Available in ROS-I repository for ABB Robots	16
3.2.5	ROS-I drivers for ABB Robot Controller	17
3.3	Robot Modeling	18
3.3.1	URDF	18
3.3.2	Xacro	19
3.4	MoveIt!	20
3.4.1	MoveIt! High Level Architecture	20
3.4.2	Ways of interfacing with MoveIt! Central Node	21
3.4.3	MoveIt! Capabilities	21
4.	SYSTEM IMPLEMENTATION	23
4.1	Description of Environment	23
4.1.1	ABB IRB-4600 Robot Model	24
4.1.2	Pneumatic Magnetic Gripper	25
4.1.3	Kinect Camera	25
4.2	Approach of Implementation	26
4.3	ABB IRB4600 Robot: Support Package	28
4.3.1	Structure of ABB IRB4600 Support Package	28
4.3.2	Validation of Support Package Created for ABB IRB4600 Robot	35

4.4	MoveIt! Configuration Package.....	37
4.4.1	MoveIt! Motion Planning Library	39
4.4.2	Analysis on OMPL and STOMP Motion Planners.....	40
4.5	Mechanism for ROS Integrated Gripper Control of ABB IRB4600 Robot. 46	
4.5.1	Factors for Selecting the Gripper Control Method for ABB IRB4600 Robot 47	
4.5.2	Extension of ROS-I Driver for Receiving String Messages to Control the Gripper of ABB IRB4600 Robot	47
4.6	ROS Nodes for Robotic Task & Environment Adaptation for Safety.....	49
4.6.1	Pick-and-Place Task	49
4.6.2	Environment Adaptation for safety of Humans/Objects	50
4.7	ROS Architecture of overall System.....	58
5.	DISCUSSION.....	60
6.	CONCLUSION.....	63
	REFERENCES	64
	APPENDIX A: COMMUNICATION MECHANISM BETWEEN ROS-I AND CONTROLLER FOR ROBOT MOTION AND GRIPPER CONTROL.....	1
	APPENDIX B: ROS COMPUTATION GRAPH OF OVERALL SYSTEM.....	2

LIST OF FIGURES

<i>Figure 1: Demo industrial environment</i>	3
<i>Figure 2: Action Research Process [2]</i>	3
<i>Figure 3: Industrial Robots in the automobile industry [5]</i>	5
<i>Figure 4: a) Conventional fencing b) Human and robot sharing workspace [7]</i>	6
<i>Figure 5: ROS file system level [20]</i>	12
<i>Figure 6: Structure of ROS graph layer [20]</i>	13
<i>Figure 7: ROS-I Logo [26]</i>	14
<i>Figure 8: ROS-I High Level Architecture [26]</i>	15
<i>Figure 9. Basic robot structure [34]</i>	18
<i>Figure 10: MoveIt! High Level Architecture [42]</i>	20
<i>Figure 11: Overall system architecture including interfaces and messages</i>	24
<i>Figure 12: Pneumatic magnetic gripper, Kinetic camera and flange</i>	25
<i>Figure 13: Microsoft Kinect camera</i>	26
<i>Figure 14: Thesis tasks mapped into ‘Action Research Process’</i>	27
<i>Figure 15: URDF for link_1 and joint_1 of IRB4600 robot model</i>	29
<i>Figure 16: Frame assignment to each joint origin</i>	30
<i>Figure 17: (a) Visual model (b) Collision model of IRB4600</i>	31
<i>Figure 18: 3-D model of IRB4600 robot with magnetic gripper</i>	31
<i>Figure 19: (a) Visual (b) Collision meshes of the links of IRB4600 robot</i>	32
<i>Figure 20: Robot model in RViz with GUI slider</i>	33
<i>Figure 21: ABB Robot Studio integrated with ROS</i>	34
<i>Figure 22: (a) Poses of robot in RViz (b) ABB robot studio for five random positions</i>	37
<i>Figure 23: Generation of configuration files by MoveIt! Setup Assistant</i>	38
<i>Figure 24: Movement of robot from position 1 and 2 (a) In the presence of obstacle</i> <i>(2) In the absence of obstacle</i>	40
<i>Figure 25: Planning time of OMPL and STOMP motion planners (without obstacle)</i>	41
<i>Figure 26: Planning time of OMPL and STOMP motion planners (with obstacle)</i>	43
<i>Figure 27: Trajectories planned by OMPL and STOMP motion planners without an</i> <i>obstacle</i>	44
<i>Figure 28: Trajectories planned by “RRTkConfigDefault” motion planner for</i> <i>avoiding obstacle</i>	45
<i>Figure 29: Trajectories planned by “RRTConnectkConfigDefault” motion planner</i> <i>for avoiding obstacle</i>	45
<i>Figure 30: Trajectories produced by STOMP motion planner for avoiding obstacle</i>	46
<i>Figure 31 : Pick-and-Place task by ABB IRB4600 robot</i>	49
<i>Figure 32: Pick-and-Place task without obstacle in ROS (upper) and actual (lower)</i> <i>environments</i>	50
<i>Figure 33: Fix obstacle in (a) ROS environment (b) Actual environment</i>	51
<i>Figure 34 : Robot performing pick-and-place task with fixed obstacle in ROS (upper)</i> <i>and real environment (lower)</i>	52

<i>Figure 35 : Camera co-ordinate frame w.r.t robot base (reference frame)</i>	53
<i>Figure 36: Kinect camera images of robot environment (1) sd image (2) qhd image (3) hd image</i>	53
<i>Figure 37: Representation of dynamic robot environment with Octomap</i>	54
<i>Figure 38 : (a)Real environment (b) ROS environment</i>	55
<i>Figure 39: Final outcome of MoveIt! Octomap Updater sensor plugin</i>	56
<i>Figure 40: (a)Octomap of human (b) Robot re-planning path for avoiding collision with the human arm</i>	56
<i>Figure 41: (a) Fixed environment (b) Dynamic environment</i>	57
<i>Figure 42: Planning time of STOMP motion planner for fixed and dynamic environments</i>	58

LIST OF TABLES

<i>Table 1: List of ROS distributions [22]</i>	11
<i>Table 2: Layers of ROS-I High Level Architecture</i>	16
<i>Table 3: ABB Robot packages available in ROS-I meta-package [22]</i>	17
<i>Table 4 : Joint positions observed against the random poses separately from ROS and Teach Pendant</i>	35
<i>Table 5: OMPL planners valid for ABB IRB4600</i>	39
<i>Table 6: Planning time data of OMPL and STOMP motion planners (without obstacle)</i>	40
<i>Table 7 : Planning time data for OMPL and STOMP motion planners (with obstacle)</i>	42
<i>Table 8: Planning time data of STOMP motion planner for fixed and dynamic environments</i>	57

LIST OF ABBREVIATIONS

ANSI	American National Standards Institute
FCL	Flexible Collision Checking library
GUI	Graphic User Interface
I/O	Input/Output
ISO	International Standard Organization
IDE	Integrated Development Environment
MOOS	Mission Oriented Operating Suite
OMPL	Open Motion Planning Library
ROS	Robot Operating System
ROS-I	Robot Operating System-Industrial
RIA	Robot Industries Association
RRT	Rapidly exploring Random Trees
RViz	Robot Visualization (ROS tool)
SAIL	Stanford Artificial Intelligence Laboratory
STOMP	Stochastic Trajectory Optimization for Motion Planning
URDF	Unified Robot Description Format
XML	Extensible Markup Language

1. INTRODUCTION

1.1 Overview

The introduction of robots in the industry has revolutionized the production efficiency, accuracy and precision to the next level. Industrial robots have become the significant figure to share burden of work in the industry. A typical industry is equipped with industrial robots, installed for performing variety of tasks. The industrial robotic applications and technologies have undergone accelerating advancements. However, the potential of industrial robots in terms of effective human-robot interactions still remains limited [1]. Mostly industrial manipulators are caged and programmed in such a way to let them follow the pre-defined sequence of movements that limits the scope of alternative trajectories for them to accomplish their tasks. Currently, the field of industrial robotics is undergoing intensive research to bring the robots out of their cages and enable them to work in a shared workspace with human operators. Breakthrough in this research will help achieve the maximum production efficiency while maintaining the safety of humans who work in the vicinity of those robots.

A general industrial production environment comprises of a number of robots of different vendors, which are designed to accomplish the production tasks. These robots are controlled and monitored through different architectures and user-interfaces as designed by their respective manufactures. Each vendor has its own specific software to program and simulate the robots. These vendor-specific versions of software limit the functionalities and hardware-abstraction capabilities. Here the ROS (Robot Operating System) is brought to use. It offers a common platform along with a variety of functionalities, modularity and hardware-abstraction capabilities. ROS also provides a set of tools that help to visualize, control and add advance features for the robot applications.

This thesis presents the method to implement ROS-Industrial architecture into the industrial robot environment. This is done by developing the ROS packages. The implemented ROS-I architecture facilitates us to design our industrial robot application using the ROS capabilities, libraries and tools.

In this system, ROS nodes are developed to enable the industrial robot capable of adapting and reacting to the changes occurring in the environment. The changes in this case, are the appearance of obstacles (objects/humans) in the workspace of the robot. This system allows the robot to undergo flexible movements in order to avoid those appearing obstacles while providing the continuation of robot actions along with the safety of the workers/objects moving around it. This is done by integrating the camera with the ROS environment by using the ROS package. This camera is fixed in the robot workspace and provides the visual information of the robot environment into RViz (ROS tool). Based on the camera data, RViz updates the visualization in the planning scene

of robot model and provides the remaining available space to the motion planner by excluding the space occupied by robot model and the obstacles for planning a path. Consequently, it results the collision free trajectory of the robot. This system provides a continuation of robot actions along with the collision avoidance with the workers or the objects if they unintentionally or intendedly confront the robot's original path.

1.2 Objective

The goal of the thesis is to ensure the flexibility of the industrial robot for re-planning a path in the presence of obstacles (objects/humans), and hence provides the continuation of robot actions while avoiding collisions with humans or objects. For achieving this goal, it is required to implement the ROS-I architecture into the industrial robot environment (ABB IRB4600 industrial robot) that extends the ROS capabilities, tools and libraries available to use with this environment.

The final system gives the idea to implement the environment adaptation feature to the industrial robot. This feature enables the robot to keep its operation while changing its path, in order to avoid the collision with the dynamic objects (objects/humans) that come to its way. If the valid path is not possible to plan, the robot stops right away and waits until the path clears.

Achieving this goal leads to the following research questions:

- 1) How to make our industrial robot (ABB IRB4600) capable of adaptive and reactive to the changes (known and dynamic) occurring in its workspace for a general production task?
- 2) How could we utilize ROS libraries, capabilities and tools for our robot to adapt and react to these changes?
- 3) What is required to implement ROS-I architecture into our industrial robot environment?

1.3 Limitations

- 1) The thesis is implemented into the demo industrial environment of one of the TUT's laboratories as shown in Figure 1. This robot environment contains ABB industrial robot of model IRB4600 2.05/60.
- 2) Use ROS-I as robot software frame work to implement the objective.

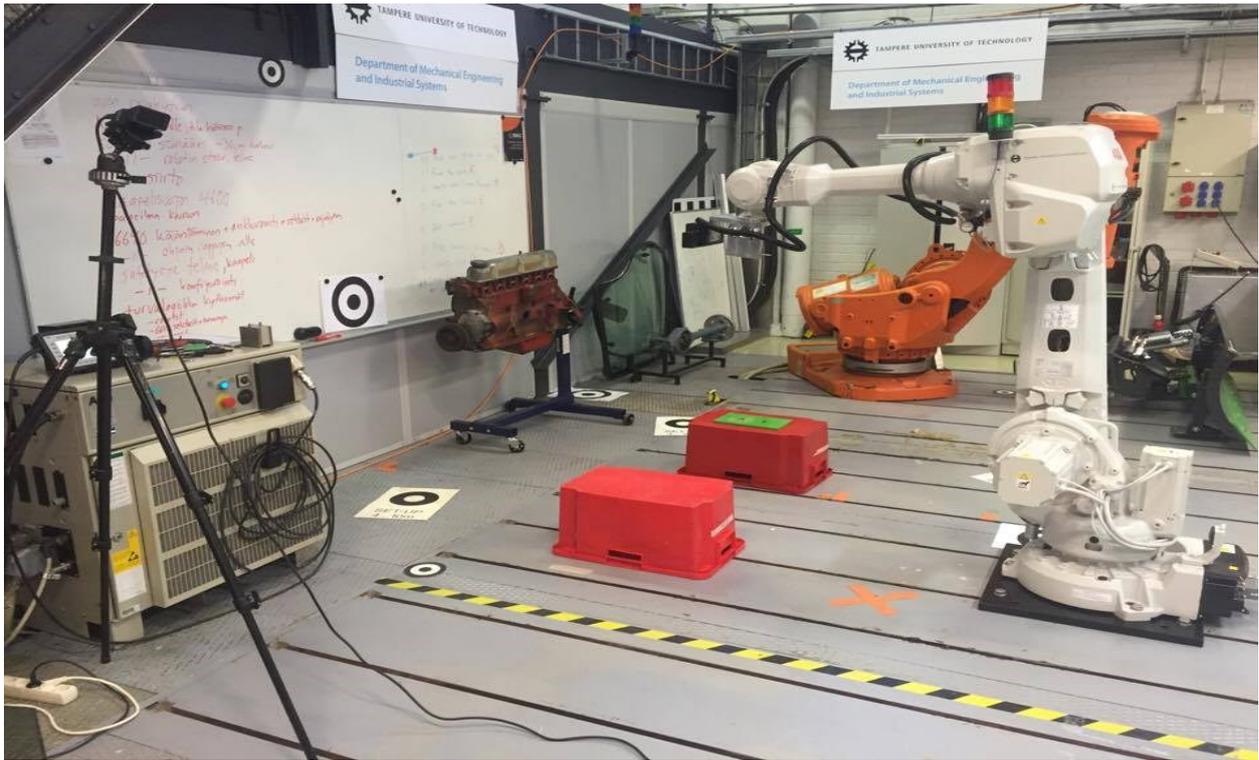


Figure 1: Demo industrial environment

1.4 Research Methodology

‘Action Based Research’ methodology is followed in this thesis. According to Power Richenda et al. [2], any research into practice that is carried by those participants who are involved in that practice with an aim to change and improve it. The action research process follows a cyclical process passing through the four inter-related stages such as *Plan*, *Act*, *Observe* and *Reflect* as shown in Figure 2.

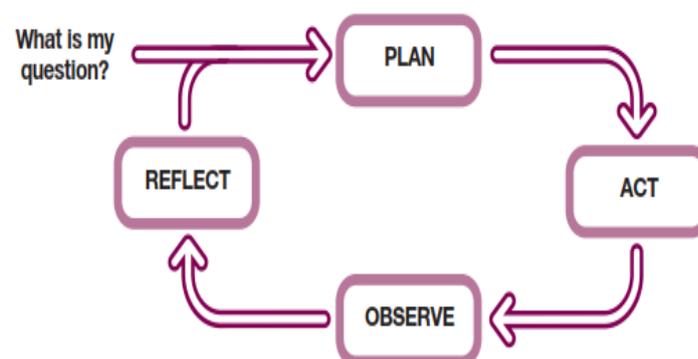


Figure 2: Action Research Process [2]

Plan stage takes the research questions, investigate them and plan the best practices to solve the questions.

Act stage initiates the enquiry to implement the practices planned.

Observe stage record the responses to the actions taken and raised further questions.

Reflect stage tackles the questions, further raised by Action & Observations and repeat the cycle.

The overall goal of the thesis is divided into set of tasks. These tasks are mapped into the action research process as shown in Figure 14

1.5 Thesis Outline

The thesis is divided into the following chapters.

The introduction gives the general overview and motivation of this thesis, objective of the thesis, research questions and research methodology followed for achieving the goal.

Chapter 2 gives the idea about the industrial robots, their importance in the industries, brief overview of conventional and advance methodologies to ensure safety in human- robot interactions and safety standards related to human-robot collaboration.

Chapter 3 illustrates the important concepts about ROS, ROS-I, MoveIt! and industrial robot modelling that are utilized in this thesis.

Chapter 4 describes the experimental setup, procedures, analysis and methods to perform the tasks that aim to achieve the goal

Chapter 5 discusses the overall system and its outcome.

Chapter 6 provides the conclusion.

2. ROBOTS SHARING WORKSPACE WITH HUMANS IN INDUSTRY

Industrial robots are hazardous machines. It is required to ensure safety against human and robot interactions. With the increase of the industrial robots world-wide, the efforts have been made to introduce advance safety technologies in the robot applications that can lessen the space between human and robot while prioritizing the safety prospect. The safety standards are defined to ensure safety of the human working near the industrial robots. Other than the safety standards of human-robot collaboration, one aspect of this thesis is to provide an additional safety feature on top of the safety standards, to the humans working in the robot environment. This is done by making the robot capable of acknowledges the humans in its path and re-plan an alternative path in order to avoid collision with them.

2.1 Industrial Robots

Industrial robots are programmable manipulators that can handle parts or tools through predefined sequence of motion. They are flexible to perform variety of tasks. One manipulator performing one specific task, can be made available to perform other task of its capacity by simply modifying the control settings without changing the hardware. They are designed to exhibit the functionality of both machine tools and the machine tool operators. [3]

Robots have been used in the industry for precise, fast and high-quality production [4]. The use of the industrial robots has been increased significantly over the last five decades. They are involved in variety of applications related to processing, assembling and handling of parts. The automobile manufacturers are the dominant consumers of industrial robots. There is a large number of manufacturers of industrial robots world-wide. However, major industrial robot manufacturers are ABB, Fanuc, Kuka, Kawasaki, Nachi, Yaskawa and other such companies.



Figure 3: Industrial Robots in the automobile industry [5]

2.2 Safety against Industrial Robot and Human Interventions

However, the industrial robots have acquired significant role in the industry, they are undoubtedly hazardous machines. The areas where the robot and humans share the workspace require proper shielding of the humans against the robot actions. The conventional fencing and safeguarding (as shown in Figure 4(a)) are usually applied to ensure safety that require large floor space. Apart from the conventional safeguarding, the recent years have seen the significant development of the complex safety functionalities. These functionalities are equipped with advanced sensing technologies. The use of these technologies enables the human operators to work safely in the robot workspace while maintaining the robot efficiency and flexibility of work. [6]

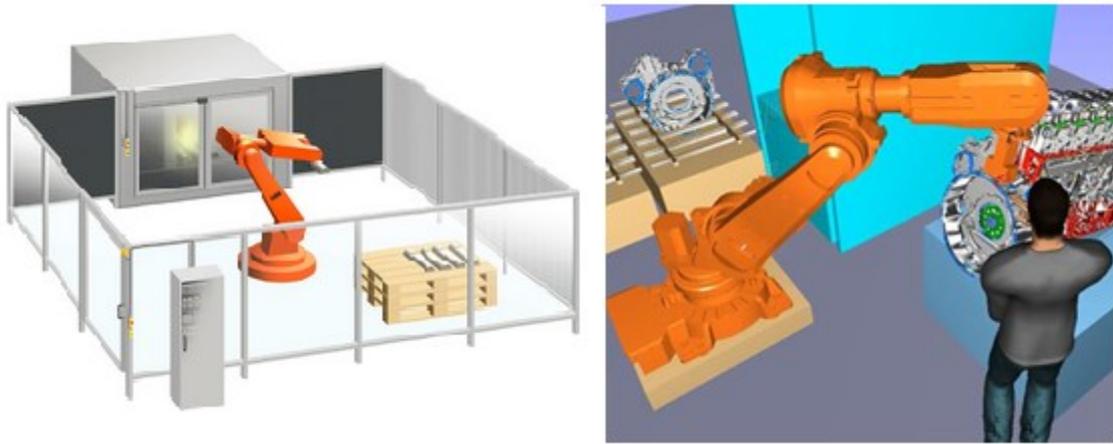


Figure 4: a) Conventional fencing b) Human and robot sharing workspace [7]

To ensure safety of the humans working around the industrial robots, it is required to ensure high levels of awareness and attention among the robot and the human. This insurance leads to put effort in the design of the robot regarding its sensing, controlling and reasoning capability along with modification in the working environment. [1]. Therefore, the designing and manufacturing of the industrial robots, and the integration, operation and planning of the robot work cells are the two independent engineering tasks. The safety standards are developed individually for both.

2.3 Safety Standards for Human and Robot Interaction

Safety of the workers performing tasks around the industrial robots, is the most significant factor that requires the foremost attention while manufacturing and designing the industrial robots and their applications. This is the reason that the work was started to make standards of safety requirements for the humans working around the industrial robots. R15.06 robot safety standard was developed in USA by Robot Industries Association (RIA) with the help of American National Standard Institute (ANSI). In 1992, International Standards Organization (ISO) in Europe delivered

forth the first addition of ISO 10218 related to the safety of humans working around the robots. ISO 10218 was finally followed by CEN as EN 775. [6]

In 2016, ISO/TS 15066 was developed to specify the requirements of safety for the robot workcell in which the robot and human are working together. These standards are the addition to ISO 10218-1 and 10218-2 standards, which specify the guidance and requirements for human and robot collaboration. However, these standards are not applicable to the applications of the collaborative robots that were in operation before the publication of these standards. These standards are particularly developed for industrial robots and their working environment. Anyhow, these standards can be beneficial for the other robots as well. [8]

In addition, it is a common practice to separate out the industrial robots by fencing around it, to avoid any injury because of its actions. However, after the publication of these standards (ISO / TS 15066 and ISO 10218-1 & 10218-2), there is a possibility to safely bring precision and power of industrial robots close together with the critical and problem-solving ability of the humans. The idea behind the development of these standards is to allow the humans to share the workspace with robots that could dramatically increase the production. [8]

In these standards, one of the ways to ensure safety between the robot and human working in the same workspace, is through the speed control of the robot and techniques to monitor the separation between them. The maximum allowed speed of the robot and the minimum allowed distance between the human and robot are mentioned in one of the specifications of these standards. According to the movement of the person in the concerned workspace, the robot moves away from the human to avoid collision. In addition, there are some protection devices integrated with the robot that sense the presence of a human in the vicinity of a robot so that the robot should stop or take an alternative path for avoiding collision with the human. [8]

Moreover, this thesis is intended to improve the safety of the humans working near the industrial robots. The implementation of this system is not to bypass any already configured safety standard. However, in case of obstacles (humans/or any other objects) in the robot path, it enables the robot to keep its operation while avoiding those obstacles by re-planning an alternative path. However, in case of in-valid path even after re-planning, the robot stops right away and waits until the clearance of the path.

3. ROBOT SOFTWARE FRAME WORK

Although there are other robot software frameworks also available like Microsoft Robotics Studio, Mission Oriented Operating Suite (MOOS), Player/Stage etc. but the Robot Operating System (ROS) is the leader among them because of its modular design, and extremely vibrant community world-wide. It has good hardware support. There are a variety of advance sensors and actuators available for robot applications that are easily configurable with ROS. The idea behind the ROS is to provide standard functionalities performing hardware abstraction, which could be shared and made available to use for all the robots. This reusability of the functionalities allows the robot developers to spent time in more advance and different functionalities rather than to start developing everything from scratch. These are the reasons that enable us to implement ROS-I architecture into ABB IRB4600 robot environment.

3.1 ROS

ROS is termed as Robot Operating System. Most common definition of ROS is that it is flexible robot software framework that contains a set of programs, tools and libraries for the development of robot applications [9]. It is an open source framework that helps to simplify the complex robot task by using its features such as hardware abstraction, visualizers, package management, device drivers and message-passing etc. [10].

Kerr and Nickels et al. [11] writes that Robot Operating System is similar to the computer operating system. The computer operating system provides the number of programs to offer control to the user. Likewise, Robot Operating System provides a set of programs and interfaces to the robot software developers for the control of the robots.

3.1.1 Background

According to Martinez and Fernandez et al. [12], ROS project started in 2007 in Stanford Artificial Intelligence Laboratory (SAIL) as a part of Stanford Artificial Intelligence Robot Project. Morgan Quigley initiated the development of ROS project with the name Switchyard [13]. ROS is completely an open source. It aims to provide common platform to the users to develop more advanced robot applications [14]. The development of ROS is not an individual effort rather it has many contributors. Many people of international community have contributed and many are involved for its development and maintenance. The list of the contributors is available on ROS.org website [9].

3.1.2 Why ROS?

ROS is the robot software framework similar to the other traditional frameworks such as Mission

Oriented Operating Suite (MOOS) [15], Player [16], Microsoft Robot Studio [17] etc. There are some issues, which ROS resolves successfully as compared with these robot software frameworks. These issues are discussed below:

3.1.2.1 Code Reusability

ROS consortium aims to have some already developed good and stable algorithms for common robot tasks like motion planning, navigation, collision detection and so on. ROS facilitates to reuse them rather than to waste resources of re-implementing them. ROS provides the standard packages that contains these kinds of algorithms. The algorithms used for robot hardware interface with ROS, are openly available in the form of ROS standard packages. ROS consortium has list of hundreds of publicly available ROS packages [18] dedicated to use for variety of robotic applications. The availability of the ROS packages facilitates the developers to spent time on new and more advanced ideas rather than to start implementation from the scratch every time. [19]

3.1.2.2 Distributed and standalone processing

O’Kane [19] reveals that the better way of controlling the complex robot tasks is to divide the robot-processing task among multiple computers or even in the same computer but with separate standalone programs. To accomplish the robot task, standalone programs in a single computer or separate computers need to communicate with each other. ROS provides simple and consistent ways of communication between these standalone programs using ‘Publisher-Subscribers’ and ‘Services’ methods. The idea about these methods of communication is illustrated in the section 3.1.6.

3.1.2.3 Efficient and easy testing

According to O’Kane [19], software development for the robots is more challenging than other types of software developments. He says that the reason behind this, is that testing of the robot software is often time consuming and error-prone. Algorithms need to be checked and tested on the robot simulations first, before testing them with real robot. When it comes to work with real robots, it sometimes makes the process slower and more demanding. However, ROS stands to resolve these issues by following ways:

- It separates out the low-level control from the high-level control. Low-level control relates to the hardware of the robot while the high-level control corresponds to the logic handler of the robot. This separation of the control provides us the edge to replace temporarily the low-level programs with the simulator (instead of real robot) to test the high-level algorithms.
- ROS provides the facility to record sensor data and other kinds of messages. It makes the testing easy because we can play back the stored data as many times as we want, in order to test other ways of processing as well. Thus, working in a ROS environment is almost similar in both scenarios i.e. robot simulation and working with a real robot.

3.1.2.4 ROS tools

ROS enables integration with many useful tools with the robot. Some of the strong open source tools are RViz, Gazebo and rqt_graph. These tools help in visualization, simulation, and debugging purposes. [20]

3.1.3 Misconceptions about ROS

Following are the common misconceptions about ROS mentioned by M. O’Kane [19]:

- ROS itself is not a programming language. However, its libraries contain a large number of programs, written mainly in C++. Some of its client libraries are also written in *Python*, *Java* and *Lisp*.
- ROS is not an IDE (Integrated Development Environment). However, we can use IDEs such as *Pycharm*, *NetBeans* and *Eclipse* with ROS.
- ROS does not only include the predefined libraries but it can also include the libraries from the user (because of plugin architecture). It has the build system, central sever, set of command lines and graphical tools.

3.1.4 ROS Pre-requisites

ROS only runs on UNIX based platforms. Tested operating systems for ROS are Ubuntu and Mac OS X. Different distributions of ROS are compatible with different Linux versions for example ROS Indigo Igloo is compatible to use with Ubuntu 14.04.2 LTS. [20] [21]

ROS is not yet available with Microsoft windows [21]. However, there is a way to run ROS through virtual machine on the host operating system (say Microsoft Windows) and use Linux as a guest operating system.

3.1.5 Distributions of ROS

ROS distributions are similar to the Linux distributions. According to Joseph [20], ROS distributions are the collection of ROS Meta packages. Each ROS distribution contains a set of ROS packages with unique version name. According to the ROS documentation [22], the purpose to release new ROS distribution every year or so, is to enable the developers to work against the stable and tested codebase until they are ready to push everything forward. Table 1 shows list of ROS distributions so far.

Table 1: List of ROS distributions [22]

Distro	Release date	EOL¹ date
ROS Lunar Logger-head	May,2017	May, 2019
ROS Kinetic Kame	May23rd, 2016	May, 2021
ROS Jade Turtle	May 23rd, 2015	May, 2017
ROS Indigo Igloo	July22nd, 2014	April, 2019
ROS Hydro Medusa	September 4th, 2013	May,2015
ROS Groovy Galapagos	December 31st,2012	July, 2014
ROS Fuerte Turtle	April 23rd, 2012	-
ROS Electric Emys	August 30th, 2011	-
ROS Diamondback	March 2nd, 2011	-
ROS C Turtle	August 2nd, 2010	-
ROS Box Turtle	March 2nd, 2010	-

3.1.6 Levels of ROS

There are three levels of ROS concepts as mentioned in the ROS documentation: [23]

- ROS file system level
- ROS computation graph level
- ROS community level

3.1.6.1 ROS File System Level

File system level of the ROS concerns that how ROS files are organized on the hard drive. The block diagram in Figure 5 shows the arrangement of ROS files on the hard disk.

¹ End-of-life (support is not available after specified date)

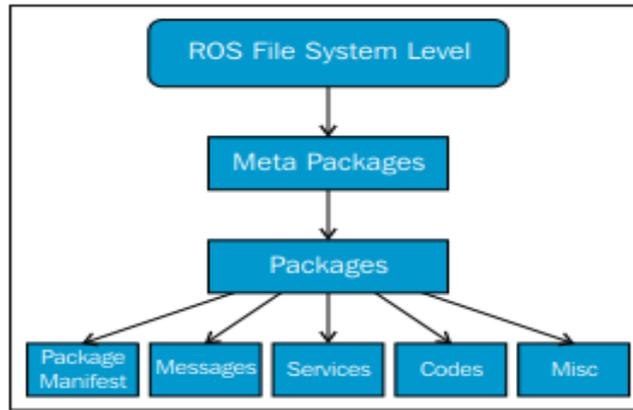


Figure 5: ROS file system level [20]

Below is the description of important blocks of the ROS file system:

- **Meta packages:** As its name suggests, meta package contains a set of packages. A group of packages collected as one entity for a particular purpose, is termed as meta-package. Similarly, ROS meta package is a collection of ROS packages. The meta package was termed as *stacks* in the earlier versions of ROS (ROS Feurte and ROS Electric). [20]
- **Packages:** According to O’Kane [19], ROS package is the group of both supporting and executable files, organized for a specific purpose. All ROS nodes, configuration files, ROS package dependencies, robot description files and all other supporting files are included in a ROS package.
- **Package manifest:** It is an XML file with the name *Package.xml*. It contains details of the package such as package developer, package maintainer, package name, package dependencies, version and license of the package. [24]
- **Message (*msg*):** Communication between the ROS nodes is carried out in the form of ROS messages. ROS message file defines the information type used for communication. One can define his own message type inside the ‘*msg*’ file in the ROS package. ‘*msg*’ is the extension of this file. [20]
- **Services (*src*):** This file has an extension ‘*src*’. Similar to ‘*msg*’ file, ‘*src*’ file defines the data structure of the messages of request and response for services between the ROS nodes. [23]

3.1.6.2 ROS Computation Graph Level

According to Mahtani et al. [13], all processes (ROS nodes) connect through the certain network created by ROS. Any node can transmit and receive data to other nodes through this computation network. This network is termed as *ROS computation graph*. It enables the communication between the ROS nodes and allows them to see the information flowing across it.

The main concepts included in the ROS computation graph level are ROS nodes, parameter server, ROS bags, messages, topics and services. There is one meta package/stack with the name 'ros_comm' available at [25] which contains ROS communication related packages, ROS C++ and Python core client libraries and the tools to introspect the ROS concepts. These all packages are collective termed as ROS graph layer. Figure 6 shows the ROS graph layer.

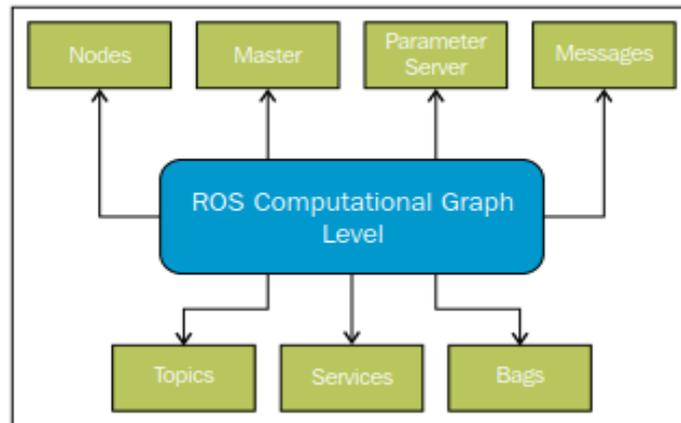


Figure 6: Structure of ROS graph layer [20]

Below is the brief description for the main concepts involved in the ROS computation graph:

Nodes: Nodes are the processes that executes the computation. Nodes are written using C++ and Python ROS client libraries. It is favorable to divide complex tasks among several ROS nodes rather than to have a few or one big ROS node. These nodes communicate with each other through the ROS network. Several nodes make the process easy to follow and debug. [13]

Master: According to ROS concepts [23], every node has to register with the master. The master provides registration information to the other nodes for sake of their connections with each other. The nodes without registering themselves to the master are not able to access the ROS network in order to perform computation.

Parameter server: It is the part of the ROS master. It allows the ROS nodes to store and retrieve the parameters in a central location at runtime.

Messages: ROS nodes sends messages to each other in the network for communication. These messages are simple data structures that contain fields related to standard data types (string, integer, float, Boolean etc.)

Topics: ROS topics are the busses which have the unique names. ROS node sends messages to another ROS node in the network through the defined topics. The sender ROS node is the publisher to this topic and the receiver ROS node is the subscriber of this topic. This communication between the ROS nodes continues as long as the message data type specified in publisher and subscriber nodes is the same.

Services: Publish/subscribe model (of ROS nodes communication) is not the effective in the case if the sender node also needs the response of sent message. The publish/subscribe model is the one-way of communication. In the robotic applications, we often need the two-way communication that undergoes request/response interaction. Here we use ROS services for this purpose. The definition of the service contains two parts, one is for requests and the other one is for responses.

Bags: ROS bags provide facility for saving ROS message data and playing back afterwards. For example, saving the sensor data is useful and can be retrieved for playing back later on, for testing and developing the robot algorithms.

3.1.6.3 ROS Community Level

There are some ROS resources, which contribute to share knowledge and software among the worldwide ROS community. The international community around the globe maintains these resources. These resources mainly include Distributions (ROS distributions), Repositories (Code repositories), The ROS Wiki (Main forum for documentation), and ROS Answers (Other ROS users respond to the questions here) etc. [20]

3.2 ROS-Industrial

ROS-Industrial is an extension of the ROS that provides the advanced capabilities of ROS software. It contains packages, tools and drivers, which supports integration of ROS with the Industrial Robots [20]. ROS-I is behaving as a bridge between the ROS environment and the industrial robots. Figure 7 is the logo of the ROS-I.



Figure 7: ROS-I Logo [26]

ROS-I packages are of two categories: General and Vendor specific packages. Vendors like ABB, Motoman, Fanuc, Robotiq and Universal Robot have their own specific packages based on their respective robot models. General packages contain the packages that helps for the general training to work with the industrial Robots in the ROS environment. [27]

3.2.1 Brief History about ROS-I

ROS-I was founded by Shaun Edwards in January 2012 [28]. It was started as open source project by the collaboration of Southwest Research Institute, Willow Garage and Yaskawa Motoman Robotics for the sake of extension of ROS to the industrial automation domain. ROS-I Consortium

was launched in 2013 in America and Germany directed by SwRI and Fraunhofer IPA respectively. [20]

Yaskawa's Motoman SIA 20D, with DX100 controller, was the first industrial manipulator handled as ROS-I robot client [29].

3.2.2 Intended Goals of ROS-I

The reasons, behind the development of the ROS-Industrial, are as follows: [20] [26]

- The idea of integrating ROS with the present on going industrial technologies is to look for more advanced ROS capabilities related to manufacturing field.
- Development of a reliable and robust software that contains tools, libraries and drivers for simulation, debugging and visualization of operations of industrial robots, and which can be utilized for variety of manufacturing applications.
- Facilitating research and development in industrial robotics applications.
- Providing a unique channel to look for the solutions of industrial robots in various manufacturing applications and getting support from the world-wide robotics professionals and researchers.

3.2.3 ROS-I Block Diagram

The ROS-I builds on top of the ROS. Architecture of the ROS-I is shown in Figure 8.

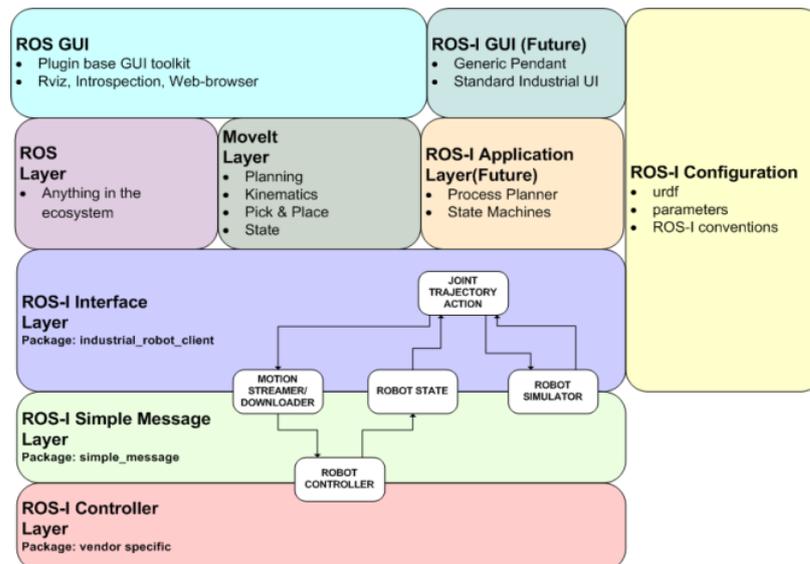


Figure 8: ROS-I High Level Architecture [26]

Each layer is briefly explained in Table 2: [20]

Table 2: Layers of ROS-I High Level Architecture

Layers of ROS-I High Level Architecture	Purpose
ROS GUI	Contains the ROS enabled GUI tools such as RViz and introspection etc.
ROS-I GUI	It is for the future use, it would carry the generic teach pendant that works as industrial robot user interface
ROS Layer	All communications are handled by this layer
MoveIt! Layer	It is organized to have solution to the industrial robots about its path planning, kinematics, pick & place and its state
ROS-I Application Layer	It aims for industrial process planning. It includes all planning that are involved in the manufacturing process
ROS-I Interface Layer	It establishes the interface between the industrial robot client (ROS node) with the industrial robot controller with the help of the message protocol
ROS-I Simple Message layer	It aims to establish two-way communication for sending data between industrial robot client and industrial robot controller using standard protocol
ROS Controller Layer	It contains the controllers as per specification of the vendors

3.2.4 Packages Available in ROS-I repository for ABB Robots

Some of the ABB robots have the packages available in the ROS-I ABB repository. These packages are developed to integrate ABB industrial robots with the ROS environment so that the ROS capabilities, tools and libraries can be used successfully to control the ABB robots. Table 3 shows the ABB robot models, which have these packages available in the ROS-I repository at the moment. [30]

Table 3: ABB Robot packages available in ROS-I meta-package [22]

Packages	ABB Robot Models				
	ABB IRB 2400	ABB IRB 5400	ABB IRB 6600	ABB IRB 6640	ABB IRB 4600
Support Package	✓	✓	✓	✓	× ²
Movelt! Configuration Package	✓	×	×	✓	×
Movelt! Plugins (Ikfast Kinematics)	✓	×	×	×	×

In the available packages for the ABB robots, it is clear that support package is the common available package for these robot models. This support package contains all the physical descriptions of the robot. The Unified Robot Description Format (URDF) is the main file of this package. It is the base package in order to integrate industrial robot with the ROS environment because the development of the other packages is dependent on this package. The support package for our industrial robot (ABB IRB4600) is not available in ROS-I official repository [30]. It is required to create these packages for integrating our robot with the ROS environment. Thus, we need to develop them. Although, `abb_irk4600_support` package is available now, but in the `abb_experimental` repository [31].

3.2.5 ROS-I drivers for ABB Robot Controller

There is a set of rapid files (ABB Robot controller programming language) readily available, which behave as the driver between the ROS environment and the robot controller (Implements ROS-I controller layer as shown in Figure 8). There are two drivers available for this purpose. These are ROS-I `abb_driver` and `Open-Abb_driver`. Both drivers work on those ABB robots having IRC5 controller installed with “PC Interfacing” and “Multitasking” features.

3.2.5.1 `Open_abb_driver`

This driver works on ROS Fuerte distribution only. There are two parts of this driver. The first part includes the set of Rapid files. These files install on robot controller and allow the remote client to send the commands for robot actions. The second part contains the set of libraries in the remote computers for developing the control algorithms for the robot actions. One can use the ROS driver to develop control scheme using ROS publishers and subscribers. This driver also provides the `Python` or `C++` libraries for bypassing the ROS completely and facilitate the direct communication with the robot controller. [32]

3.2.5.2 `ROS-I_abb_driver`

Similar to `abb_open` driver, this driver also contains set of rapid files for interfacing ROS environment with the robot controller. This driver is included in the ROS-I meta-package. It provides

² However, support package for irb4600 is now available in ABB experimental repository of ROS-I

many features to work with ROS-I framework. It facilitates the use of ROS tools like MoveIt! and RViz to develop the control schemes for motion control of the robot. It supports ROS publishers/subscribers and services models of communications to send/receive robot motion commands. The *abb_driver* rapid files are available on ABB ROS-I repository [33] .

3.3 Robot Modeling

Simulation and visualization of any robot requires its 3D virtual model that should perfectly reflect the actual robot. Thus, ROS needs the accurate physical description of the robot in order to measure its kinematics and to track correctly the position and orientation of the robot with rest to its parts and the world model around it. URDF (Unified Robot Description Format) [33] is the way to represent robot model in ROS. [35]

3.3.1 URDF

In ROS, URDF describes the robot in terms of its physical parameters. URDF is an XML based representation of the robot. It contains information related to links, joints, dimensions and other robot's physical parameters. RViz, Gazebo and MoveIt! use the resulted virtual robot model for visualization, simulation and motion planning of the robot respectively. [12][36]

The basic structure of the robot, irrespective of the complexity of the robot, contains mainly the links and joints. Along the structure of the robot, one joint connects two links. Out of these two-connected links, one is the parent link and the other one is the child link as shown in Figure 9. [34]

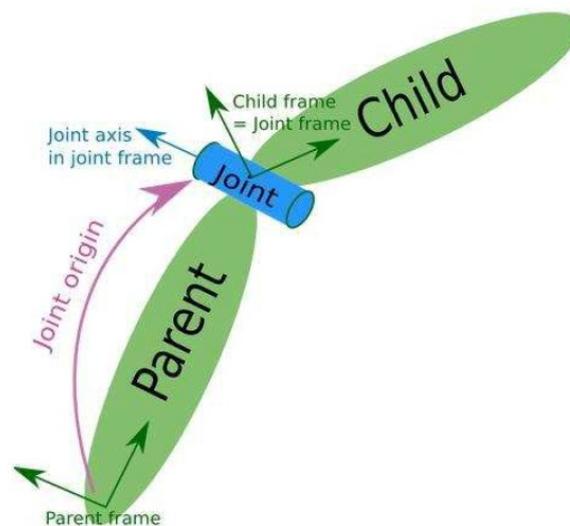


Figure 9. Basic robot structure [34]

URDF uses primarily two elements as main tags in XML file. These are links and joints. Link main tag contains further sub tags for specifying the description of rigid part of the robot like visual appearance, shape of the link, color, mass, origin, collision model etc. Whereas, the joint element contains information regarding the connection of links, type of joints (revolute, prismatic, continuous, fixed, planar) and the limits of the joints. [37]

A common robot description in URDF format looks as following: [24]

```
<robot name="Name of your robot">
<link1> ... </link1>
<link2> ... </link2>
.
.
<joint1> ...</joint1>
<joint2> ...</joint2>
.
.
</robot>
```

It is a difficult way to write the XML robot description file for each link of the robot. There is a more convenient way of generating them by developing the 3D model of the links of the robot using Computer Aided Design (CAD) software such as Solidworks and Pro-E. Export the files (.*stl* or .*dae*) generated by the software corresponding to each link of the robot. And then link these under the link tags of the URDF. [34]

3.3.2 Xacro

Xacro is an XML macro language, used to maintain the URDF file in more readable, understandable and shorter XML form. It is simple to say that the xacro is just another way of defining a URDF file. It is a scripting mechanism for defining URDF file that allows modularity and code re-usability. Xacro is useful when we have large robot description XML file. For example, in xacro macro language, one can generate a macro of the “wheel” and then initiate it four times in the code with different parameters to build four wheels on the robot. But in the case of the URDF file for building the same four wheels, one has to copy and paste the whole code four times manually.[38] [24]

Once we have the xacro (*model. xacro*) of the robot. We can generate URDF file (*model. urdf*) any time from it by the following command which runs the *xacro* ROS node (which converts the xacro file into urdf file) from the *xacro* ROS package. [39]

```
$ rosrn xacro xacro model. xacro > model. urdf
```

3.4 MoveIt!

Industrial robots are finding large range of applications where they have to perform actions in close interaction with the humans. The industrial robotic research is already in a way to find the possibilities to ensure the safe interaction between the human and the robot sharing the same workspace. The robots working in the industrial environment must have the ability to avoid collisions with the humans and other obstacles. MoveIt! is designed to have collision detection ability as one of its features. MoveIt! is a collection of the software packages and tools integrated with the ROS to provide capabilities such as motion planning, 3D perception, collision detection, kinematics solving, manipulation and control. [40] [41]

MoveIt! is the successor of Arm Navigation. MoveIt! has the same capabilities and features that Arm Navigation provides, but solves the problems which Arm Navigation had. In the Arm Navigation, each capability is acting as a separate ROS node. Integrating these capabilities requires large amount of sharing of data between the ROS nodes that results synchronization problems. The extension of the types of services offered by *move_arm* (Arm Navigation central node) requires altering the node structure itself because there is no plugin architecture in Arm Navigation. MoveIt! was developed to cater these problems. [41]

3.4.1 MoveIt! High Level Architecture

MoveIt! has the plugin based architecture, which allows the user to add their own capabilities without altering the architecture of *move_group* (MoveIt!'s central node). Figure 10 shows the architecture of the MoveIt!.

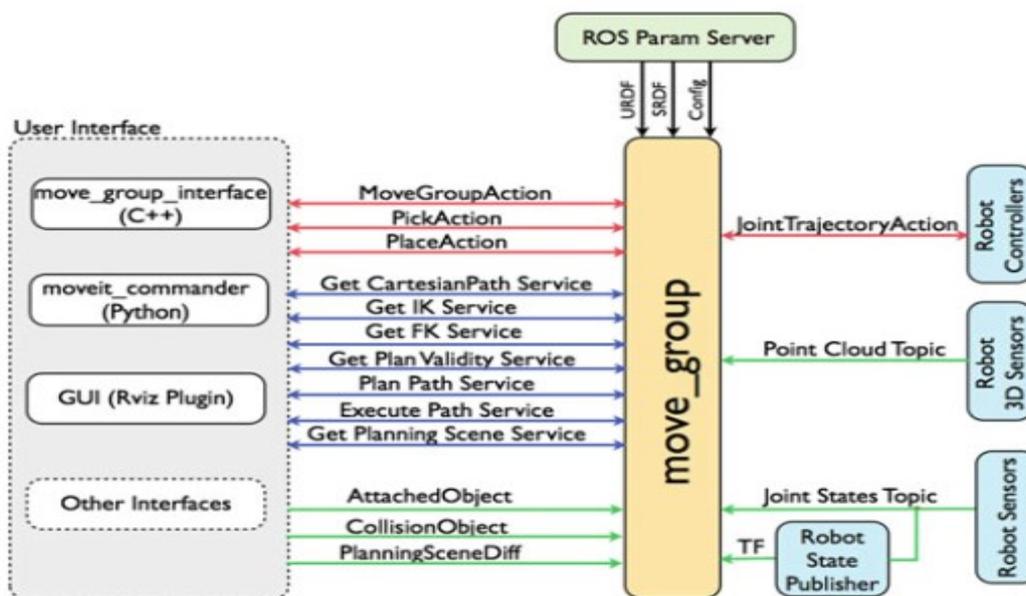


Figure 10: MoveIt! High Level Architecture [42]

The central node in the MoveIt! is called the *move_group*. This node integrates the capabilities of the MoveIt! and make them available for the users through ROS actions and services [42]. This node does not execute any motion planner algorithms, instead it enables the plugin-based architecture for all functionalities to integrate with it [20].

3.4.2 Ways of interfacing with *MoveIt! Central Node*

MoveIt! provides three ways to the users to interact with the *move_group* and use the capabilities of the MoveIt!. [42]

- **Through GUI-RViz Plugin**

It is a graphical interface used for motion planning from RViz itself.

- **Through C++ API**

Move_group_interface package provides access to move group by C++ API. [43]

- **Through Python API**

MoveIt_commander package gives access to MoveIt! capabilities through python API. [44]

3.4.3 MoveIt! Capabilities

MoveIt! has many capabilities. The main capabilities used in this implementation are discussed here.

3.4.3.1 Motion Planners

MoveIt! integrates motion planning algorithms through plugin interface. This capability of the MoveIt! facilitates us to plan a path for the robot by using motion planners from various motion planning libraries. The central node of the MoveIt! (*move_group*) offers interface to the motion planners through ROS service or action. OMPL (Open Motion Planning Library) is the default motion planning library that *MoveIt! Setup Assistant* integrates to use for the robot. OMPL includes a set of sampling based motion planners. As a matter of fact, OMPL is a general motion planner library and it does not have the concept of robotics. However, MoveIt! configures OMPL to work for the robot. [41] [45]

Apart from by-default motion planners, users can also integrate other (or their own) motion planners because of the plug-in architecture of the MoveIt!. Like STOMP (Stochastic Trajectory Optimization for Motion Planning) planner is integrated with this system for motion planning of our robot.

3.4.3.2 Solving Kinematics

Once the motion planner finds the path for the robot. The next step is to solve the kinematics of the robot, corresponding to the planned path, for moving it. MoveIt! uses plugin architecture for solving the inverse kinematics of the robot. By default, MoveIt! uses numerical solvers for the solving the kinematics. Anyhow, users can integrate their own solvers with the MoveIt!. Analytical solvers are faster than the numerical solvers. One of the famous analytical solver plugins is *IKFast* [46] that offers kinematic solutions for industrial manipulators.

3.4.3.3 Collision Detection

MoveIt! uses FCL (Flexible Collision Checking Library) as the default collision checking library. Same as with other capabilities, MoveIt! integrates collision checking capability as the plugin architecture, which offers the users to add their own algorithms for collision checking of the robot with itself and with the environment. FCL can perform continuous collision checking. Collision checking is, in fact, the most time-consuming task for generating a path. It takes around 80% of the total time that consumes for motion planning. In order to decrease the time of collision checking while motion planning, the user can specify Allowed Collision Matrix (ACM). The ACM allows the user to define those parts of the robot body, which can be ignored by the motion planner while collision checking (such as adjacent links of the robot i.e. link1-link2, link2-link3 etc.). This ACM is configured automatically by *MoveIt! Setup Assistant* and can be modified later.

3.4.3.4 3-D perception of robot environment

MoveIt! has the feature to build the 3D perception of the environment using the *Octomap* package. The user needs to input the point cloud image of the environment to the sensor plugin of the MoveIt!. As a result, MoveIt! creates the filtered cloud on *filtered_cloud_topic*. This filtered cloud includes the environment around the robot except the body parts of the robot. The filtered cloud excludes that body parts of the robot which are defined within the collision tags in the URDF file.

4. SYSTEM IMPLEMENTATION

As indicated in the introduction, the goal of this thesis is to develop the industrial robot flexible for planning a path in the presence of obstacles including human worker, and hence providing the continuous robot operation while acknowledging and taking care of the humans working around. This thesis is implemented into the specific robot environment as mentioned in the limitations section 1.3. The overall target is divided into a set of four main tasks. The first one focuses on Robot Modelling in ROS. The second task configures ABB IRB4600 robot to the ROS environment so that we can utilize ROS capabilities like motion planning, kinematics, collision checking and 3D perception for our industrial robot environment. The first two tasks are concerned with the development of the related ROS packages (*abb_irk4600_support* and *abb_irk4600_moveit_config* packages). The next task defines the mechanism adopted for ROS integrated gripper control. The final task is to utilize MoveIt! capabilities and ROS libraries to develop the ROS node. This ROS node enables the environment adaptation feature in our industrial robot so that it is able to acknowledge the obstacles (objects/humans) in its path and re-plans an alternative path to avoid them. The system is valid for both fixed and the dynamic environments.

4.1 Description of Environment

The main elements involved for this thesis implementation is shown in Figure 11. The robot environment mainly includes the ABB industrial robot (IRB4600-60/2.05), gripper and the Microsoft Kinect camera. The robot is installed with IRC5 controller and pneumatic actuated magnetic gripper. The gripper grasps the objects in the robot workspace whereas the camera provides the visual information of the robot's environment for one of the ROS tools i.e. RViz.

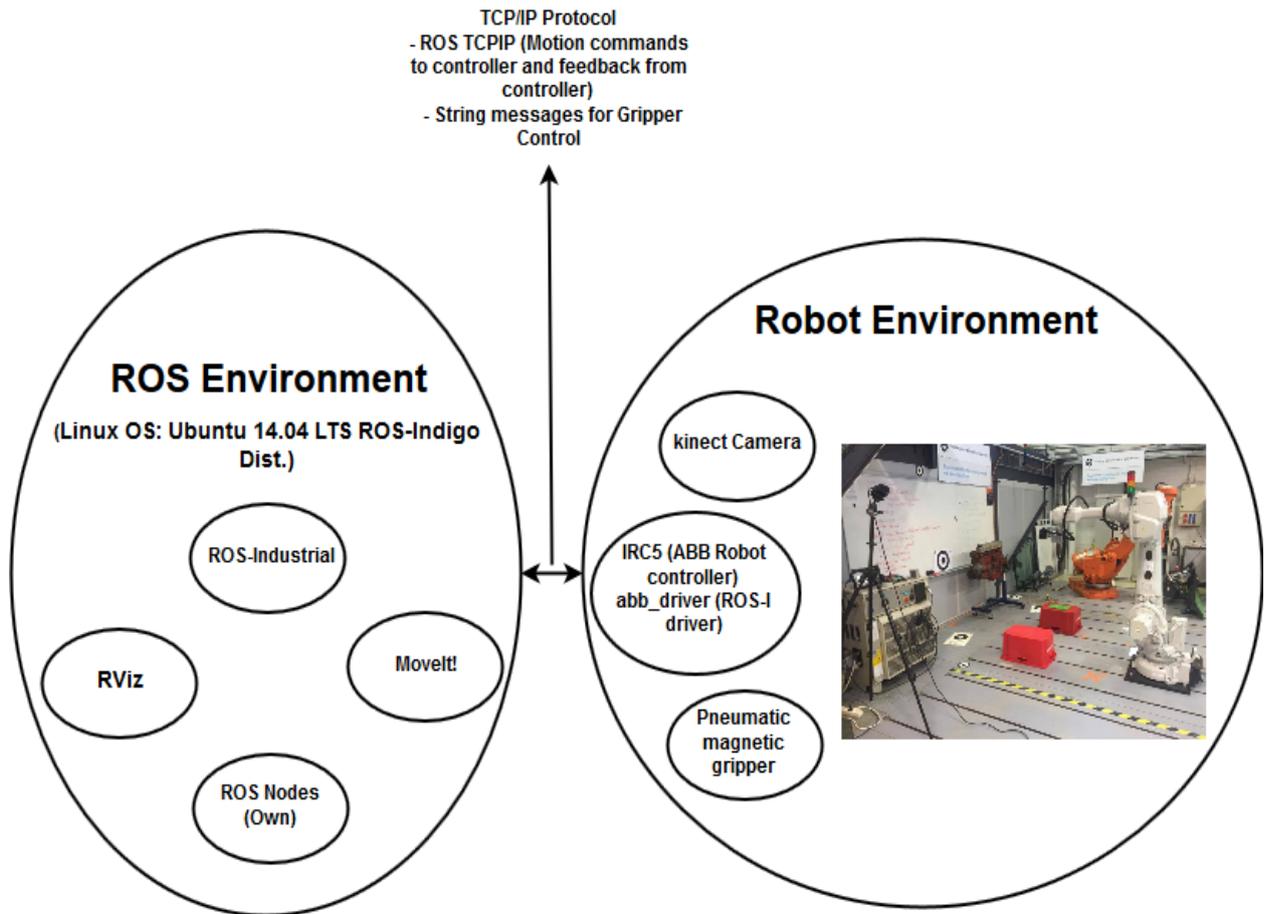


Figure 11: Overall system architecture including interfaces and messages

ROS environment is communicating with the robot environment by TCPROS, which uses standard TCP/IP sockets for sending/receiving ROS messages. ROS-indigo and Ubuntu 14.04 LTS distributions are used in this implementation. Since IRB4600 is the industrial robot, ROS-Industrial is behaving as the bridge between the ROS environment and IRC5 controller. Furthermore, ROS tools such as MoveIt! and RViz are used for motion planning, collision detection and visualization of the robot environment. ROS nodes are implemented for the robot actions for performing task and making the robot adaptive and reactive to the changes occurring in the fixed as well as in dynamic environments. Below is the brief description about the main figures of our robot environment.

4.1.1 ABB IRB-4600 Robot Model

The IRB 4600 is the leading series of ABB robotics with advanced and latest capabilities. This series of ABB robots are designed particularly for the most targeted applications such as material handling, welding, assembling, dispensing and cutting applications. Among the IRB 4600 series, IRB 4600-60/2.05 has the greater capacity for payload (60kg) and least reach (2 meters) in the workspace. These robot models are valid for robot safety standards. Since the IRC5 controller is

installed in this model, this controller gives the option of integrating it with the software support other than ABB's own software like ROS for performing advanced functions such as network communication, multitasking and sensor control. This design supports to work in the harsh environment where the robot exposes to lubricant sprays and metal splits. [47]

4.1.2 Pneumatic Magnetic Gripper

Figure 12 shows the 3-D model of assembly on which gripper and Kinect camera are mounted. However, we are not using this camera in our implementation. The gripper connected with this assembly is used for holding the metal plates while performing pick-and-place task. It is a pneumatic actuated gripper and hold weights up to 7 kg. The control of the gripper is simple on/off digital signal. The assembly has the flange so that it can be mounted easily on the ABB IRB4600 manipulator.

The working principle of these type of grippers is based on dual-action cylinder. This cylinder is attached with the permanent magnet, which moves away or towards the work piece while providing the holding force to grasp the work piece. They are designed particularly for gripping metal and ferromagnetic sheets. [48]

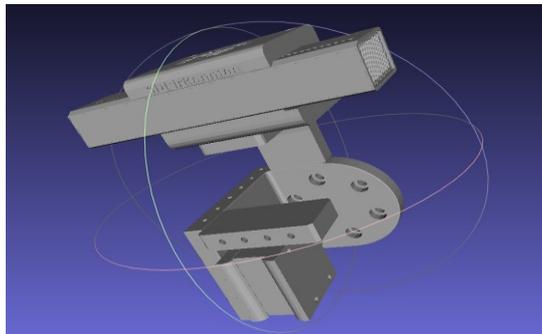


Figure 12: Pneumatic magnetic gripper, Kinetic camera and flange

4.1.3 Kinect Camera

It is a peripheral device manufactured by Microsoft for Xbox and Windows PC. It is analogous to webcam with additional features of providing the depth of the image and RGB image [49]. It also provides the point-cloud image. These additional features make it popular in the ROS projects for giving the visual information of the robot environment, object recognition and many other purposes. The Microsoft Kinect Xbox camera is shown in Figure 13.



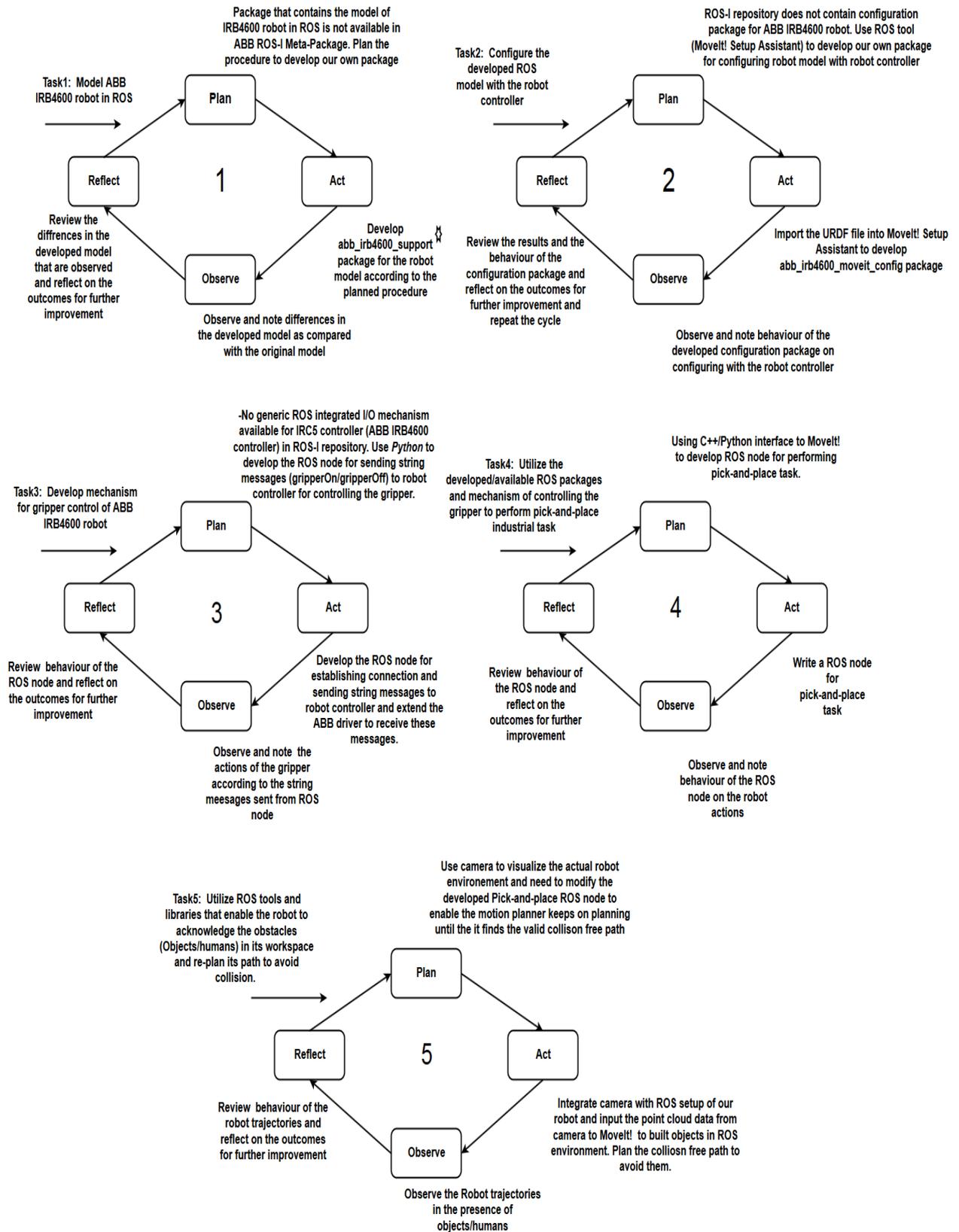
Figure 13: Microsoft Kinect camera

There are several drivers available to use the Kinect sensor with the ROS. These are *freenect_stack*, *Openni_kinect*, *Kinect* and *Kinect_aux*. Each driver has its own capabilities [50], and selection of the driver depends upon the application.

In this implementation, Kinect sensor is used to detect and visualize the dynamic objects in the robot workspace for sake of collision avoidance with that objects. Humans are also appearing as the obstacles in this case. Therefore, it is one of the safety concerns to avoid collision with the objects.

4.2 Approach of Implementation

The overall goal of the thesis is organized into set of tasks. ‘Action Based Research’ (discussed in chapter 1) is followed for the implementation of these tasks. These tasks are mapped into this research process as shown in Figure 14.



However, there is now support package for ABB IRB4600 robot is available but in ROS-I experimental repository

Figure 14: Thesis tasks mapped into ‘Action Research Process’

4.3 ABB IRB4600 Robot: Support Package

Some of the ABB robot models have the packages readily available in the ROS-I repository [28]. These robot models are IRB2400, IRB4400, IRB5400, IRB6600 and IRB6640. These packages contain a common set of files which exhibit a standard way of naming. The use of standard structure and naming in these packages provides the convenient way to work in these packages and make it helpful to create a similar package for the other industrial robot models.

ABB support package is the primary package to get start-integrating robot with the ROS environment. It includes the description of the robot model and robot environment in the form of URDF (Description of the robot model in XML format). Therefore, *abb_irb4600_support_package* is the base package for implementing ROS-I architecture into our robot environment. This package is not available in ABB ROS-I meta-package. This is the reason that it is first task of this implementation to develop this support package for our robot. However, this support package is now available in ABB ROS-I experimental repository.

4.3.1 Structure of ABB IRB4600 Support Package

There is a standardize naming and structure in all the official support packages. The same structure is followed for the development of support package for IRB4600 robot. This standard supports the common directory layout and common files in all the packages [51]. The structure of support package for our robot contains the following directories:

```
abb_irb4600_support_package
├── urdf
├── config
├── launch
├── meshes
└── test
```

4.3.1.1 URDF

URDF (Unified Robot Description Format) is the main file of this package. It contains all the physical description of the robot in the XML format. It includes a set of the standard Xacro, Xacro macro and the URDF file. The same standard way (as followed in other ROS packages of ABB robots) of naming the robot joints and links is followed in these files.

4.3.1.1.1 Structure of URDF

The structure of URDF directory is:

```

abb_irb4600_support_package
├──..
├──urdf
│   ├──irb4600_macro.xacro
│   ├──irb4600.xacro
│   └──irb4600.urdf

```

- ***Irb4600_macro.xacro*** file defines the main xacro macro developed for ABB IRB4600 robot model. It is used to maintain the URDF in more readable and understandable form. It is written in macro XML format. It gets initiated in another file (*irb4600.xacro*) to create the instance of this macro. In this case, *irb4600.urdf* is the instance created from *irb4600_macro.xacro*.
- ***Irb4600.xacro*** is the top-level xacro file that creates the instance (*irb4600.urdf*) according to the instructions defined in the *irb4600_macro.xacro* file. This file is invoked in *load_irb4600.launch* file (the XML file used for passing parameters and initiating ROS nodes) for loading URDF file (that contains the 3D model of ABB IRB4600 robot) into the *robot_description* variable (standard name of the variable that contains current loaded robot model in ROS) of the parameter server.
- ***Irb4600.urdf*** is the robot description file in XML format. This file corresponds to the real model of our robot. The main tags of this file are `<link>` and `<joint>` tags. The meshes of both visual and collision links are linked in their visual and collision tags respectively as shown in Figure 15. A small portion of the URDF for link_1 and joint_1 of IRB4600 is shown below:

```

<?xml version="1.0" ?>
<link name="link_1">
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://abb_irb2400_support/meshes/irb2400/visual/link_1.stl"/>
    </geometry>
    <material name="abb_orange"/>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://abb_irb2400_support/meshes/irb2400/collision/link_1.stl"/>
    </geometry>
    <material name="yellow"/>
  </collision>
</link>
<joint name="joint_1" type="revolute">
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <parent link="base_link"/>
  <child link="link_1"/>
  <axis xyz="0 0 1"/>
  <limit effort="0" lower="-3.1416" upper="3.1416" velocity="2.618"/>
</joint>

```

Figure 15: URDF for link_1 and joint_1 of IRB4600 robot model

4.3.1.1.2 Development of URDF for IRB4600 robot

The URDF for IRB4600 is generated in accordance with the structure and standard as followed in already developed URDF files for other models of ABB robots that are present in the abb ROS-I repository. The parameters like joint limit, joint velocity and link lengths etc. defined in this URDF file, are taken from the ABB IRB4600 manual [47]. However, the origin and direction of rotation of each joint are specified with respect to the frame assigned to each joint.

4.3.1.1.3 Method of assigning frame to each joint

The reference frame is assigned to each joint by following the below mentioned conventions. These conventions are typically used to specify the frames to the joints for finding the Denavit-Hartenberg (DH) parameters. These parameters are specifically used for solving forward kinematics of the manipulator [52]. According to these conventions:

- Z-axis is along the axes of rotation of the joint (blue color axes as shown in Figure 16)
- X-axis is directed parallel to the common normal drawn between the z-axes of the previous joint reference frame and the current joint reference frame. In the case of no unique common normal i.e. if those mentioned z-axes are coincident/parallel or intersecting to each other, the x-axes of the current joint reference frame will be perpendicular to both of those z-axes. (red color axes in Figure 16)
- Y-axis is simply drawn by following the right-hand rule. (green color axes as shown in Figure 16)

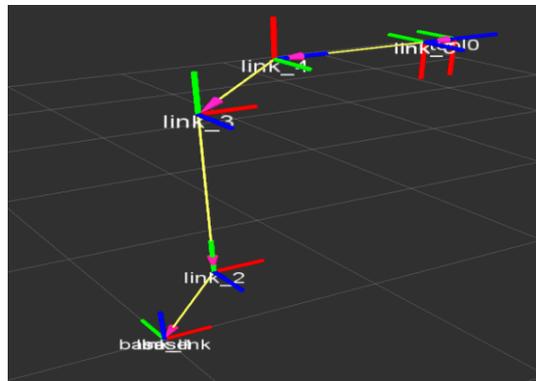


Figure 16: Frame assignment to each joint origin

Figure 16 shows the frame assignment to each joint origin of IRB4600 robot. As this robot is 6-DOF robot so there are six frames with one extra base frame. The distance between these frames is the length of the respective links. The value of the links length of our robot is taken from the ABB official manual for IRB4600 robot model. Figure 17 shows the final 3-D robot visual model and collision model that are defined in the created *irb4600.urdf* file.

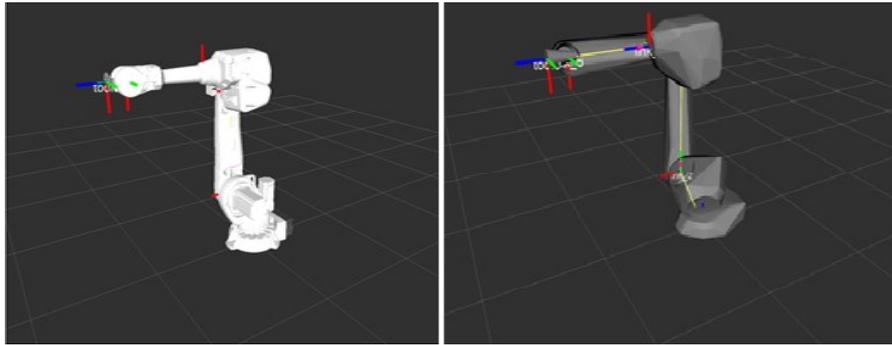


Figure 17: (a) Visual model (b) Collision model of IRB4600

The magnetic gripper as shown in Figure 12, is the tool attached to this manipulator. The 3D model of this tool is linked in the *irb4600.urdf* by adding an additional link tag (`<link name="tool0">...</link>`) at the end of all the links of the robot. This *tool0* link tag contains the visual and collision geometry of the whole assembly of the tool. Figure 18 shows the final model of the robot with tool attached to its last link.

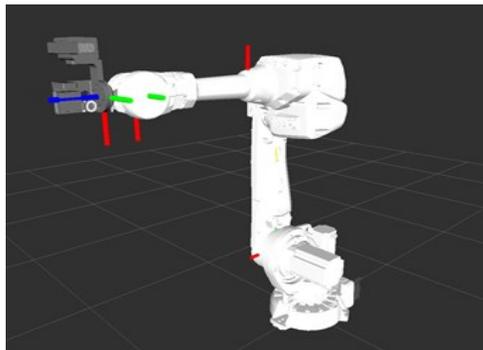


Figure 18: 3-D model of IRB4600 robot with magnetic gripper

4.3.1.2 Meshes

Meshes includes the 3-D CAD models of all the links of our robot. It contains two types of the meshes. One is the visual meshes (actual 3D models of links) and the other one is the collision meshes (meshes used for checking between links). The structure of the meshes folder in our *abb_irb4600_support_package* is:

```
Meshes
├── irb4600
│   ├── visual
│   │   ├── base_link.dae
│   │   ├── link_1.dae
│   │   ├── link_2.dae
│   │   ├── link_3.dae
│   │   ├── link_4.dae
│   │   ├── link_5.dae
│   │   └── link_6.dae
```

```

├── ABB_magnet_tool_kinect.dae
├── collision
│   ├── base_link.dae
│   ├── link_1.dae
│   ├── link_2.dae
│   ├── link_3.dae
│   ├── link_4.dae
│   ├── link_5.dae
│   └── link_6.dae
└── ABB_magnet_tool_kinect.dae

```

- **Visual** meshes includes the 3-D models of all the links of our robot that are exported from ABB Robot Studio. The reason of adding these 3-D models in the visual folder is to link these meshes under the `<visual>` tag of the URDF file. These models show the actual links of the IRB4600 robot. These meshes are shown in Figure 19(a).
- **Collision** meshes includes the collision CAD models of all the links of IRB4600 robot. The area of collision CAD model is bigger than its corresponding visual CAD model. These collision models are involved to check the collisions of the links between each other and with the environment. The collision CAD model of each link of IRB4600 robot is created using *Convex Hull* feature in MeshLab software (*Filters > Remeshing, Simplification and Reconstruction > convex hull*). The collision CAD models of each link of IRB4600 is shown in the Figure 19(b).

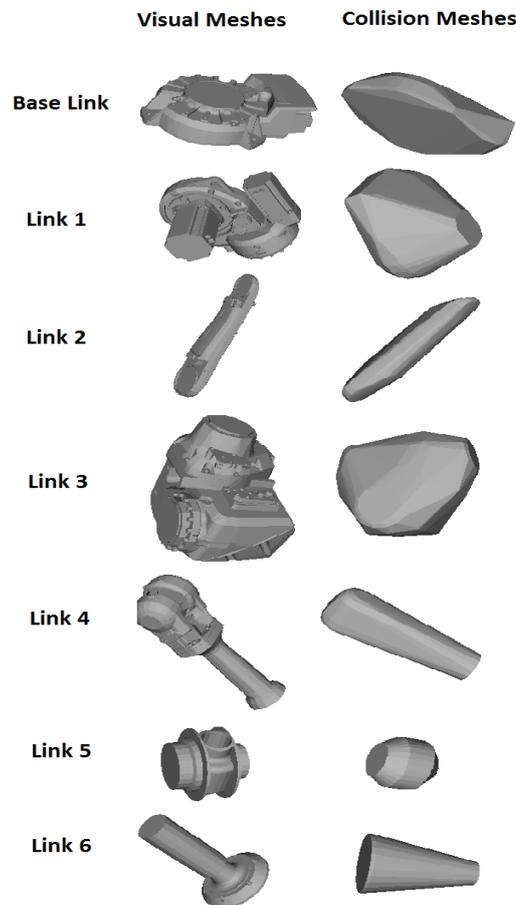


Figure 19: (a) Visual (b) Collision meshes of the links of IRB4600 robot

4.3.1.3 Config

Config contains the configuration file (*joint_names_irb4600.yaml*). This file contains the joint names of the IRB4600 robot defined in its URDF file. The joints (unique names of joints) of the robot are accessed in the other packages with the help of this configuration file.

4.3.1.4 Launch

Launch directory of this support package of our robot contains the launch files. These files are common to all support packages that are available for other ABB robot models. The structure of these launch files for *abb_irb4600_support_package* is:

```
abb_irb4600_support_package
├── ..
├── launch
│   ├── ..
│   ├── load_irb4600.launch
│   ├── robot_interface_download_irb4600.launch
│   ├── robot_state_visualize_irb4600.launch
│   └── test_irb4600.launch
└── ..
```

- **Load_irb4600.launch** file simply loads the URDF (Unified Robot Description Format) file of the IRB4600 robot model to the parameter server on the *robot_description* (Standard variable to store the description of the current robot) variable. When ROS tool (such as MoveIt!) runs, it consults the parameter server and fetches the description of the current robot model from this variable.
- **Test_irb4600.launch** file is not only loading the URDF by calling *load_irb4600.launch* file but also starts the *robot_state_publisher* and *joint_state_publisher* nodes for the user to inspect the movements of the joints of the robot through GUI sliders associated with each joint. Figure 20 shows the outcome of this launch file.

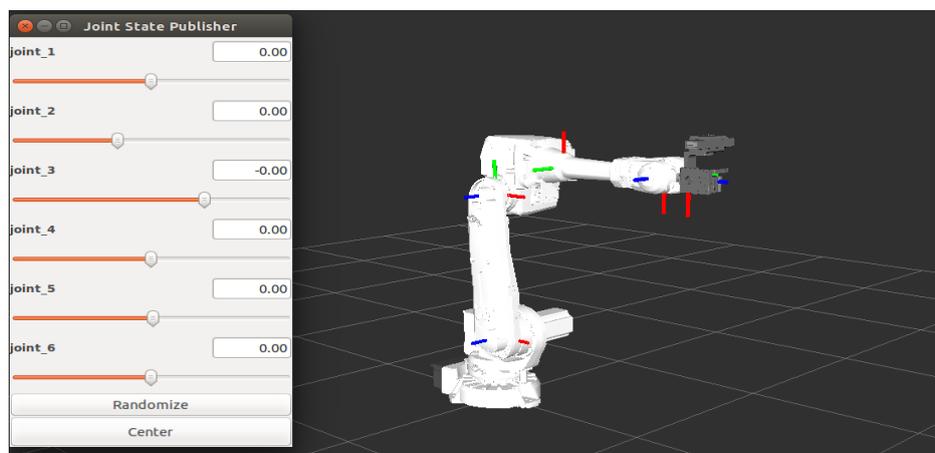


Figure 20: Robot model in RViz with GUI slider

- ***Robot_interface_download_irk4600.launch*** file initiates all the necessary ROS nodes for complete bi-directional communication between the IRC5 controller (IRB4600 controller) and ROS environment. It calls the *robot_interface.launch* file in the *abb_driver* package that initiates the ROS nodes such as *robot_state* (get feedback from the robot controller about the joint states), *motion_download_interface* (send motion commands to the robot controller) and *joint_trajectory_action* (for higher-level robot motion control) and pass the required parameters such as *robot_ip* and *j23_coupler*. The *robot_ip* parameter defines the IP of the robot controller while the *j23_coupler* parameter uses to identify the ROS nodes if the feedback from the robot controller provides the 3rd joint position which is the merger value of current joint2 and joint3 positions (This is basically the case with IRC5 controller when it sends the joint3 position to the ROS, it adds the joint2 angle value in it as well). These ROS nodes establishes the socket-based connection between the robot controller and the ROS environment using standard ROS-I *simple_message* protocol.
- ***Robot_state_visualize_irk4600.launch*** file is not only establishing the connection as in the case of *robot_interface_download_irk4600.launch* file but also enables the user to visualize the current state of the real or the stimulated robot in RViz. This is followed by loading the URDF file of the robot on the parameter server and initiate the appropriate ROS nodes: *robot_state_publisher* and *rviz*. Figure 21 shows the outcome of this launch file in which the 3-D model of IRB4600 robot in RViz keeps on tracking the state of the real or the stimulated robot (ABB Robot Studio).

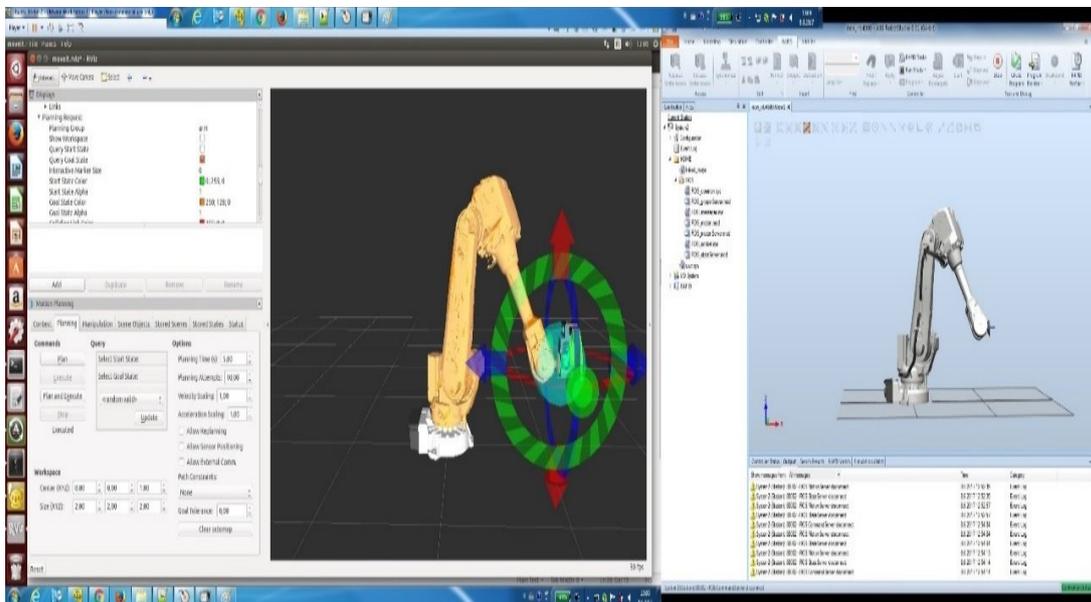


Figure 21: ABB Robot Studio integrated with ROS

4.3.1.5 Test

Test directory includes the `roslaunch_test.xml` file that contains a set of standard launch files. It initiates all the launch files from the launch directory of `abb_irb4600_support` package for the purpose of testing them against errors.

4.3.2 Validation of Support Package Created for ABB IRB4600 Robot

To validate the support package created for IRB4600 robot, the simulated ABB IRB4600 robot (analogous to real robot) in ABB Robot Studio is moved to five different random poses with the help of virtual teach pendant. The robot model in the ROS is also moved to the same poses. The joint angles corresponding to these positions and orientations, are observed separately from ROS (through the `Joint_States` rostopic) and virtual teach pendant of the ABB Robot Studio (through jogging interface). These joint angles and their difference between each other are listed in Table 4.

Table 4 : Joint positions observed against the random poses separately from ROS and Teach Pendant

Poses commanded from ROS to Simulated robot and corresponding joint angles are observed in ROS (rostopic) and Robot Studio (teach pendant)		
Joint angles in ROS (Observed in rad. from ROS Topic)	Joint angles in ABB Robot Studio (Observed in rad. from Teach Pendant)	Difference
Pose1		
Position: x=0.85428 y=0.08989 z=2.22860 Orientation: x=-0.00019 y=0.70706 z=0.00012 w=0.70715		
0.12427745754091832	0.12427746504545212	-0.00000007504
0.0492369521938677	0.049236953258514404	-0.00000001064
-0.8951209633395059	-0.8951209187507629	-0.00000004458
6.118493236814293	6.118493556976318	0.0000003201
0.8526273785315708	0.8526273965835571	-0.00000001805
0.10876592965945900	0.10876595973968506	-0.00000003008
Pose 2		
Position: x=0.33941 y= -0.05050 z= 1.13170 Orientation: x= -0.00026 y= 0.70705 z= 0.00021 w= 0.70716		
-0.24264939408602368	-0.24264942109584808	0.00000002700
-1.2824702800091008	-1.2824702262878418	-0.00000005372
1.0620109821001034	1.0620110034942627	-0.00000002139
-5.434866064367458	-5.434865951538086	-0.0000001128
0.3267387554699913	0.3267386853694916	0.00000007010
-0.8214345357125746	-0.8214345574378967	0.00000002172

Pose 3		
Position: x= -0.07833 y=0.091300 z=1.60476 Orientation: x=-0.00013 y=0.70710 z=0.00014 w=0.70711		
-0.404264921402269	-0.4042649567127228	0.00000003531
-1.2915094189781489	-1.2915093898773193,	-0.00000002910
0.3895814627997032	0.38958147168159485	-0.000000008881
6.782803322413843	6.782803535461426	-0.0000002130
0.9640703548703059	0.9640703201293945	0.00000003474
-6.584852886843878	-6.58485221862793	-0.0000006682
Pose 4		
Position: x= 1.20480 y=-0.37843 z=1.06333 Orientation: x=-0.000009 y=0.70703 z=0.000006 w=0.70718		
-0.339996958867162	-0.33999699354171753	0.00000003467
0.04641141248875822	0.046411413699388504	-0.000000001210
0.4798206326003651	0.4798206388950348	-0.000000006294
-0.6138056825737717	-0.6138058304786682	0.0000001479
-0.6179934268294863	-0.6179933547973633	-0.00000007203
0.5213052393614964	0.52130526304245	-0.00000002368
Pose 5		
Position: x= 1.14443 y= -0.04791 z= 0.46990 Orientation: x=-0.00013 y=0.70702 z=0.000004 w=0.70718		
-0.04739318744362264	-0.04739319533109665	0.000000007887
0.4090161232197769	0.40901613235473633	-0.0000000009134
0.8306709373800536	0.830670952796936	-0.00000001541
-0.05037629650902677	-0.05037630349397659	0.000000006984
-1.240252904622503	-1.2402528524398804	-0.00000005218
0.016314488009325803	0.01631448045372963	0.000000007555

Figure 22 shows the visualization of the robot for these five random poses in Rviz and ABB Robot Studio (Robot simulation: analogous to actual robot).

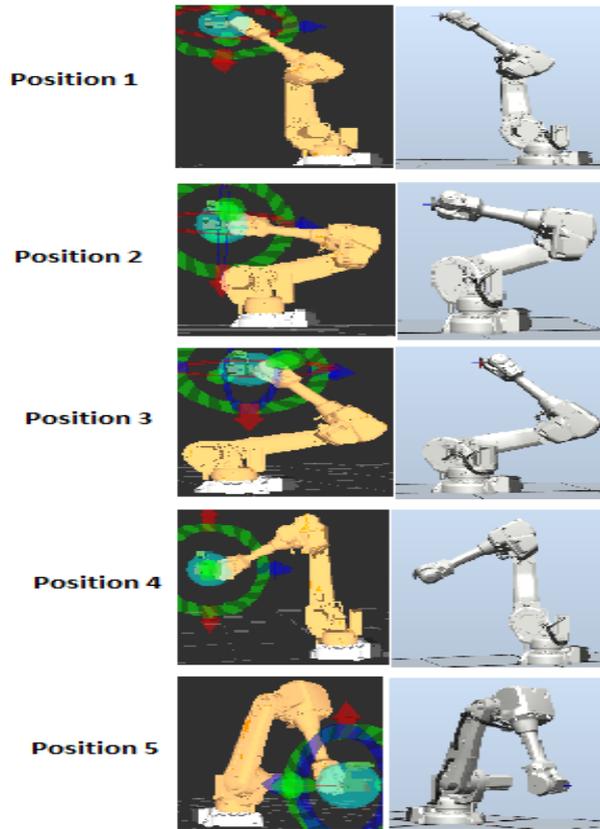


Figure 22: (a) Poses of robot in RViz (b) ABB robot studio for five random positions

The difference between these positions of the joint angles corresponding to their poses as listed in Table 4 is almost zero. Moreover, it can be observed from Figure 22 that visualization of the position and orientation of the robot in both environments, corresponding to each pose is the same. Thus, it shows that the XML description of the robot (URDF) in *abb_irb4600_support* package is analogous to the IRB4600 actual robot. This analysis of the joint positions against the different poses validates our developed *abb_irb4600_support* package.

4.4 MoveIt! Configuration Package

In order to configure the industrial robot and 3D sensor with the ROS, we need the configuration package that contains the configuration files related to the robot model, sensor and other ROS tools. *Abb_irb4600_moveit_config* package contains the set of configuration files for this purpose. These files are utilized in this implementation mainly for motion planning, collision checking, Re-planning of the path in case of obstacle, 3-D perception of the robot environment and visualization of the robot in ROS tools. The structure of this package is shown below:

abb_irb4600_moveit_config

```

├── config
│   ├── abbirb4600.srdf
│   ├── sensors_kinect.yaml
│   ├── controllers.yaml
│   ├── fake_controllers.yaml
│   ├── joint_limits.yaml
│   ├── kinematics.yaml
│   ├── stomp_config.yaml
│   └── ompl_planning.yaml
├── launch
│   ├── abb_irb4600_moveit_controller_manager.launch.xml
│   ├── abb_irb4600_moveit_sensor_manager.launch.xml
│   ├── default_warehouse_db.launch
│   ├── demo.launch
│   ├── fake_moveit_controller_manager.launch.xml
│   ├── joystick_control.launch
│   ├── move_group.launch
│   ├── moveit.rviz
│   ├── moveit_planning_execution.launch
│   ├── moveit_rviz.launch
│   ├── ompl_planning_pipeline.launch.xml
│   ├── planning_context.launch
│   ├── planning_pipeline.launch.xml
│   ├── run_benchmark_ompl.launch
│   ├── sensor_manager.launch.xml
│   ├── setup_assistant.launch
│   ├── stomp_planning_pipeline.launch.xml
│   ├── trajectory_execution.launch.xml
│   ├── warehouse.launch
│   └── warehouse_settings.launch.xml

```

These files are generated using *MoveIt! Setup Assistant*, which is a graphical user interface. Figure 23 shows the way to generate these files.

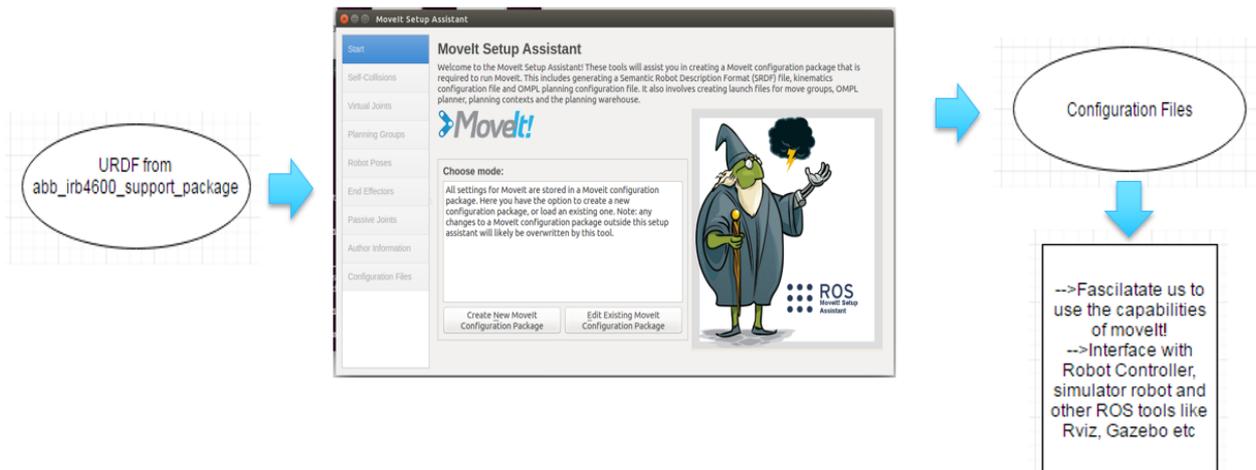


Figure 23: Generation of configuration files by MoveIt! Setup Assistant

The URDF file created in the *abb_irb4600_support* package is used as input and loaded into the *MoveIt! Setup Assistant* and complete the required steps [52]. Consequently, it generates the set of

necessary configuration files. Once the configuration files are generated, it is ready to integrate MoveIt! (ROS-I tool) with robot controller for setting up the capabilities of MoveIt! for our robot.

Some of the configuration files are not generated automatically by *MoveIt! Setup Assistant*. These files are *controllers.yaml*, *Stomp.yaml*, *sensors_kinect.yaml* and *moveit_planning_execution.launch*. *Controller.yaml*, *Stomp.yaml* and *sensors_kinect.yaml* files contains the parameters for configuring ROS with robot controller, STOMP motion planner and Kinect camera respectively. *Moveit_planning_execution.launch* is the main launch file in which all ROS nodes are initiated for this system.

4.4.1 MoveIt! Motion Planning Library

MoveIt! uses OMPL as its default motion planning library. There are eleven motion planning algorithms available in this library. All these planners are tested and integrated with the ABB IRB4600 robot. Some planners are quite slow, while some of the others are unable to provide a solution for our robot. All of these motion planners are tested to find the solution for a simple case in which we try to move the robot between two fixed positions without any obstacle in the path. Table 5 is the result of our observation.

Table 5: OMPL planners valid for ABB IRB4600

OMPL Motion Planners	Solution for Motion Planning of ABB IRB4600?
BKPIECEkConfigdefault	FAILED
ESTkConfigDefault	FAILED
KPIECEkConfigDefault	FAILED
LBKPIECEkConfigdefault	FAILED
PRMkConfigDefault	YES
PRMstarkConfigDefault	YES
RRTConnectkConfigDefault	YES
RRTkConfigDefault	YES
RRTstarkConfigDefault	YES
SBLkConfigdefault	FAILED
TRRTkconfigDefault	FAILED

According to Table 5, there are five motion planners in OMPL, which provides a path planning solution for our robot. However, all these five motion planners consume different times for finding a solution. Other than OMPL planners, STOMP planner is tested with our robot as well. Analysis of these motion planners and factors for choice are described in the next section.

4.4.2 Analysis on OMPL and STOMP Motion Planners

The goal of this thesis suggests that the motion planner for our robot must provide the shortest trajectories in minimum planning time for avoiding the obstacles. Two factors are discussed here that evaluates the planner for our robot. One factor is the planning time and the other factor is the trajectory provided by the planner. Since collision avoidance is one of the major tasks in this thesis, so smooth and shortest trajectories followed by the robot for avoiding the obstacle is very significant factor for choosing the planner in our system.

In this test setup, two fix positions are defined in the planning scene of the robot, and give a try to OMPL and STOMP motion planners to plan a path between them. The setup is tested with two scenarios. The first scenario does not contain any obstacle while the second case includes an obstacle in the robot path as shown in Figure 24.

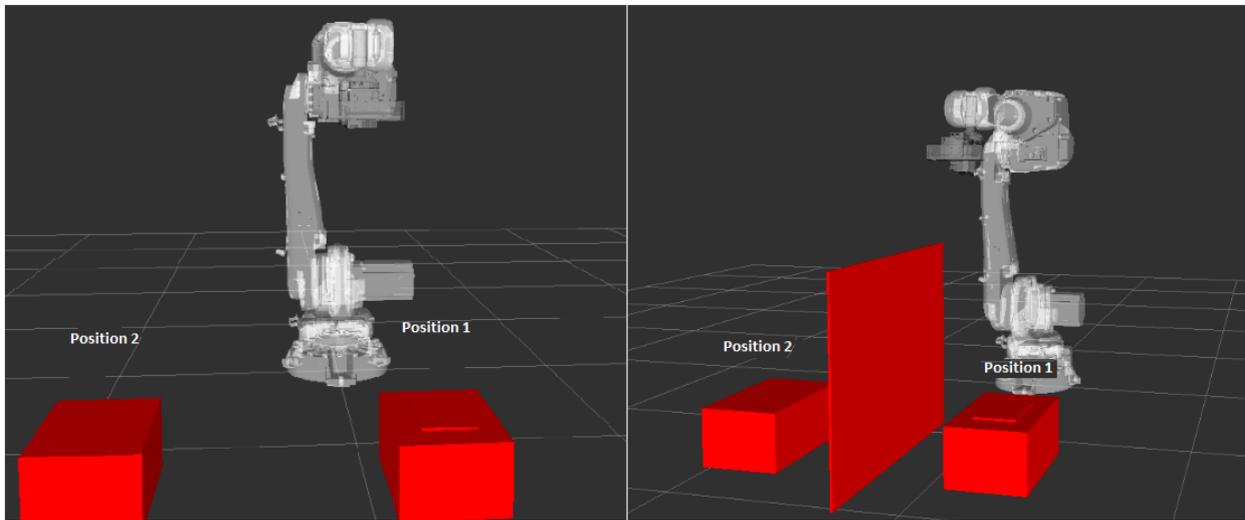


Figure 24: Movement of robot from position 1 and 2 (a)In the presence of obstacle (2) In the absence of obstacle

Table 6 shows actual, average, standard deviation, upper bound, lower bound, maximum and minimum values of the time that OMPL and STOMP planners take to plan a path between two fix positions without obstacles in five different tries.

Table 6: Planning time data of OMPL and STOMP motion planners (without obstacle)

Sr No.	PRMkCon- figDefault	PRMstark- ConfigDe- fault	RRTCon- nectkCon- figDefault	RRTkCon- figDefault	RRTstark- ConfigDe- fault	STOMP
1	5.004	5.009	0.035	0.069	5.002	0.328
2	5.005	5.011	0.037	0.056	5.001	0.309
3	5.007	5.008	0.026	0.05	5.003	0.338
4	5.005	5.009	0.037	0.041	5.002	0.323

5	5.004	5.008	0.032	0.078	5.002	0.357
Average	5.005	5.009	0.0334	0.0588	5.002	0.331
Std dev	0.00109544 5	0.00109544 5	0.00412795 3	0.01322724 5	0.00063245 6	0.01601249 5
Upper bound	5.00609544 5	5.01009544 5	0.03752795 3	0.07202724 5	5.00263245 6	0.34701249 5
Lower bound	5.00390455 5	5.00790455 5	0.02927204 7	0.04557275 5	5.00136754 4	0.31498750 5
Max	5.007	5.011	0.037	0.078	5.003	0.357
Min	5.004	5.008	0.026	0.041	5.001	0.309

The graph is plotted for the values of the planning times of OMPL and STOMP planners as shown in Figure 25.

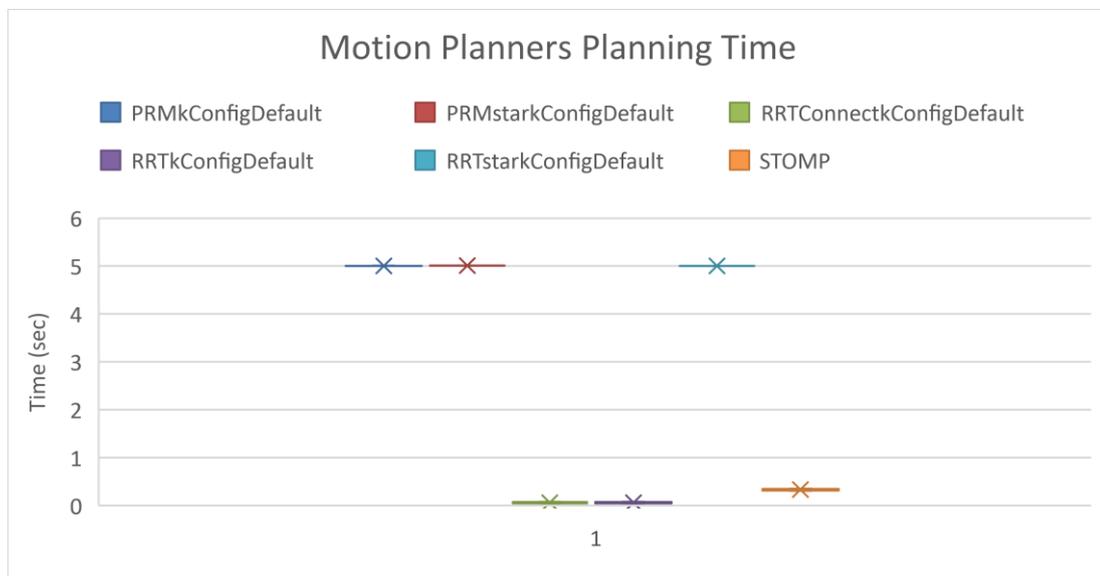


Figure 25: Planning time of OMPL and STOMP motion planners (without obstacle)

The graph in Figure 25 compares the time consumed by each OMPL and STOMP motion planners for planning a path between these two fixed positions. Thin lines of the graph show that standard deviation in the values of the planning times is very small. The cross in this graph represents the average values of planning time for each planner. This is the case when there is no obstacle in the path of the robot.

Next, an obstacle is added at the fix position as shown in Figure 24(b). The robot is commanded to move between the same two fixed positions using the same planners, but in the presence of obstacle. The test is repeated again five times for each planner. In the presence of an obstacle, we have observed that three of the OMPL motion planners *PRMkConfigDefault*, *PRMstarkConfigDefault* and *RRTstarkConfigDefault* are failed to provide solution on every try. However, they provide

solution once in 5 or 6 tries even after taking too long (more than 5 seconds). The rest of the three motion planners provide solution on every try. The time data related to their planning times is shown in Table 7.

Table 7 : Planning time data for OMPL and STOMP motion planners (with obstacle)

Sr No.	RRTConnectkConfigDefault	RRTkConfigDefault	STOMP
1	0.051	0.302	0.454
2	0.056	0.297	0.417
3	0.063	0.534	0.558
4	0.047	0.3	0.554
5	0.046	0.218	0.781
Average	0.0526	0.3302	0.5528
Std dev	0.006280127	0.106707825	0.126733421
Upper bound	0.058880127	0.436907825	0.679533421
Lower bound	0.046319873	0.223492175	0.426066579
Max	0.063	0.534	0.781
Min	0.046	0.218	0.417

The graphs in Figure 26 is plotted for the values of planning times for the current setup i.e. in the presence of obstacle.

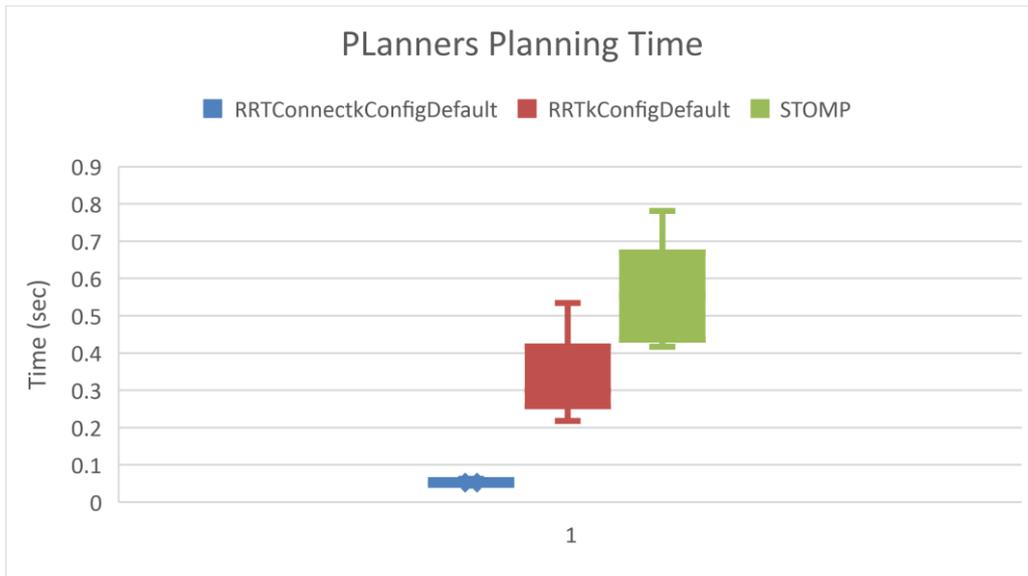


Figure 26: Planning time of OMPL and STOMP motion planners (with obstacle)

The above graph shows that in case of obstacle, the STOMP motion planner shows comparatively more deviation and takes more average times for planning a path while *RRTConnectkConfigDefault* is the efficient planner and shows the least deviation in the planning time.

Thus, *RRTConnectkConfigDefault*, *RRTkConfigdefault* and STOMP are the planners that are finally sorted out from the rest of the planners as they are most efficient and provide a solution at each run for our robot in the absence as well as in the presence of obstacle.

In fact, planning time is not the only factor that decides planner for our robot application. The trajectory of the robot also plays an important role for selecting the planner.

Now the same setup is created as discussed above. In this setup, we have observed the trajectories provided by each of the three selected planners. In the absence of an obstacle, we let the robot to move between the two fixed positions four times, using each planner separately. Almost linear trajectories are followed by the robot on every case. All the trajectories were almost the same and follow the expected trajectory (red line). Figure 27 shows one such trajectory from each planner.

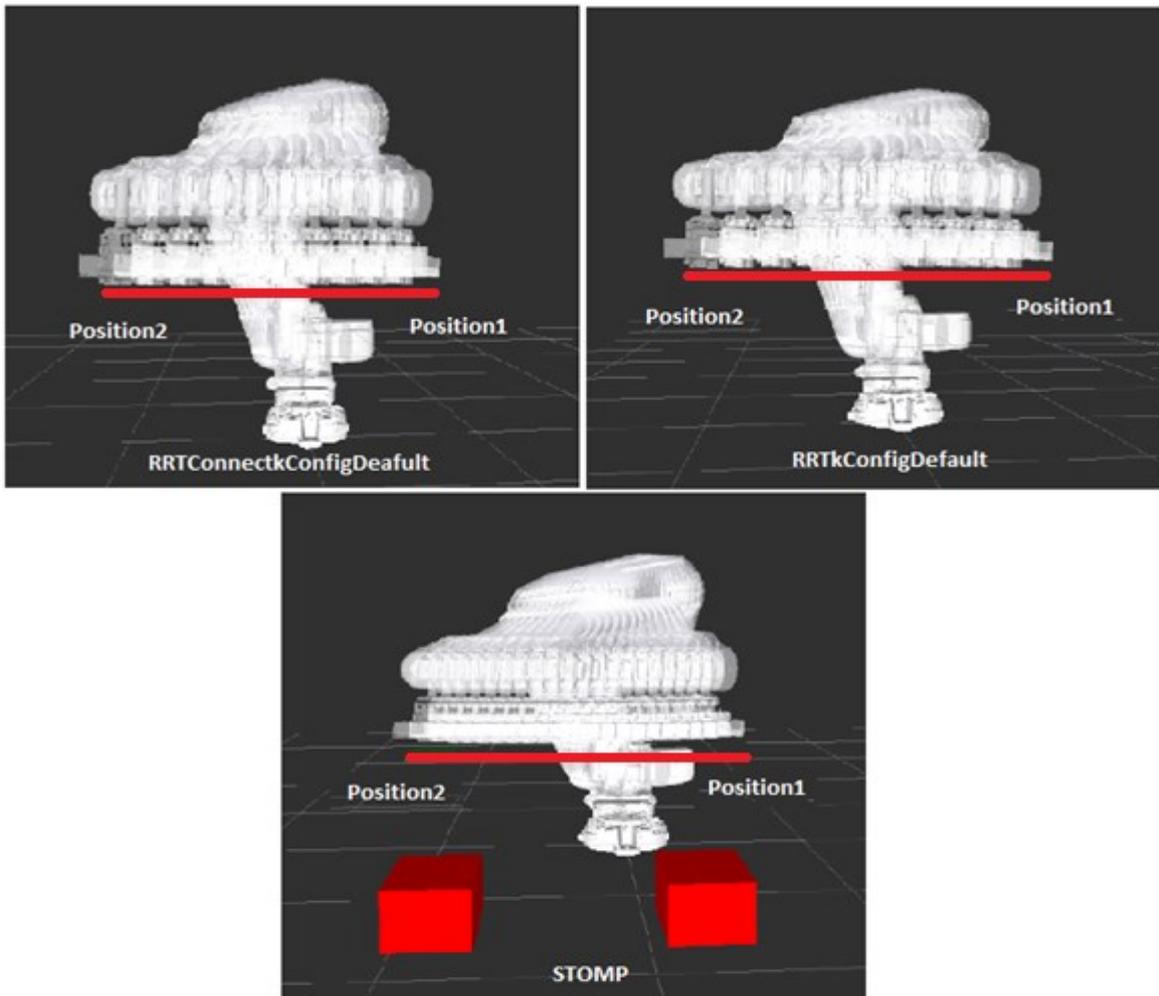


Figure 27: Trajectories planned by OMPL and STOMP motion planners without an obstacle

In the presence of an obstacle in the robot path, the trajectories provided by these three planners in order to avoid the obstacle are shown in Figure 28, Figure 29 and Figure 30. Four consecutive runs are illustrated.

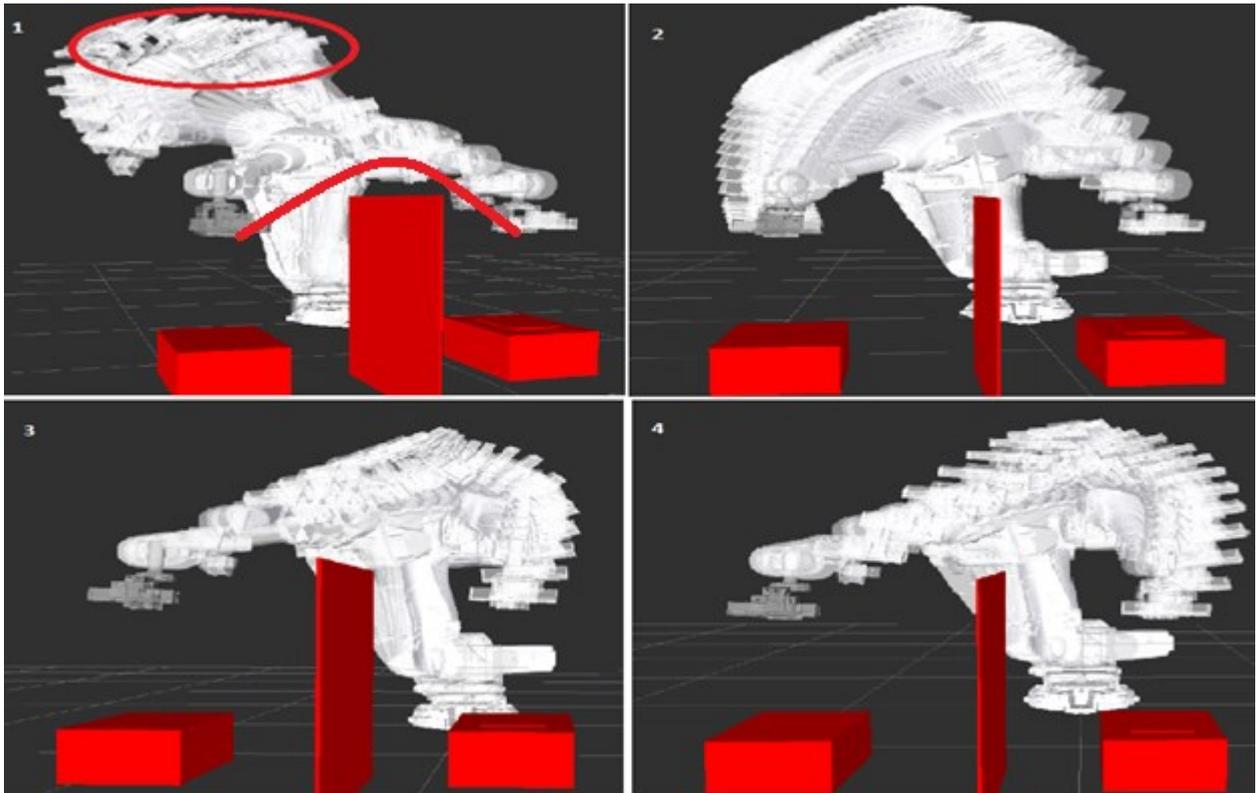


Figure 28: Trajectories planned by “RRTkConfigDefault” motion planner for avoiding obstacle

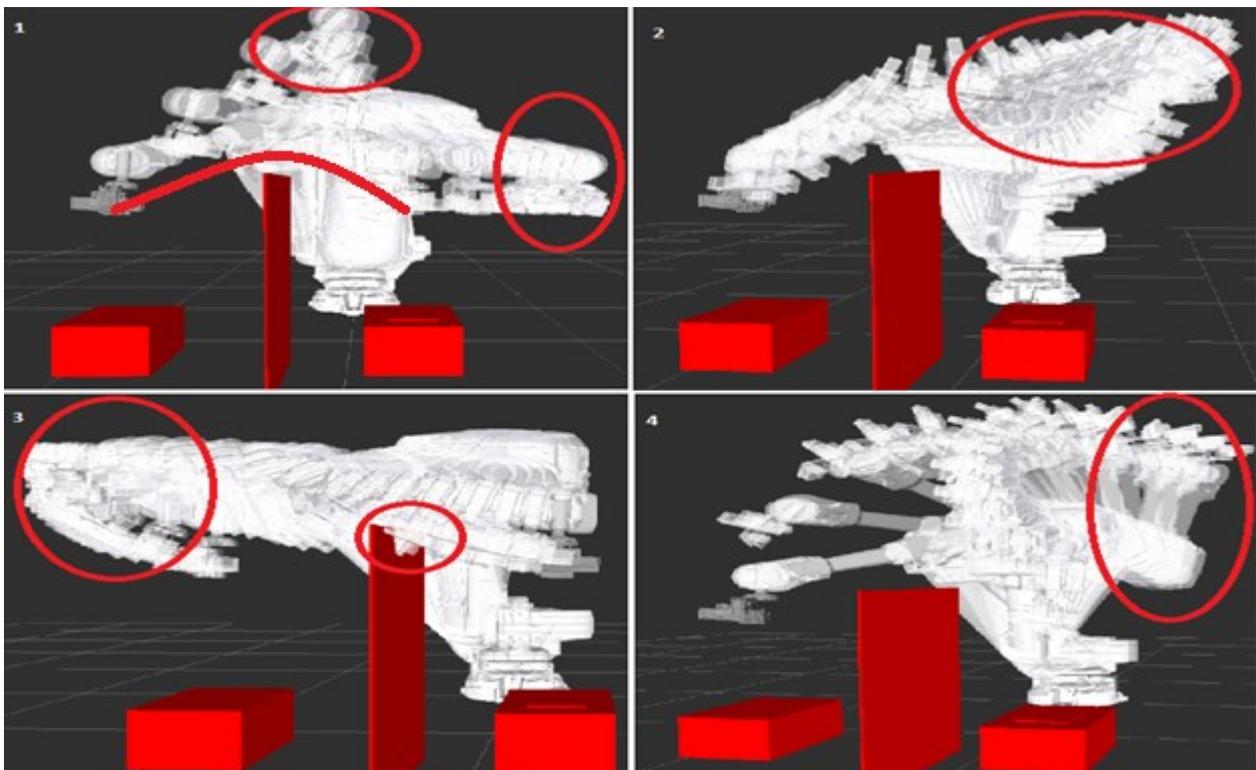


Figure 29: Trajectories planned by “RRTConnectkConfigDefault” motion planner for avoiding obstacle

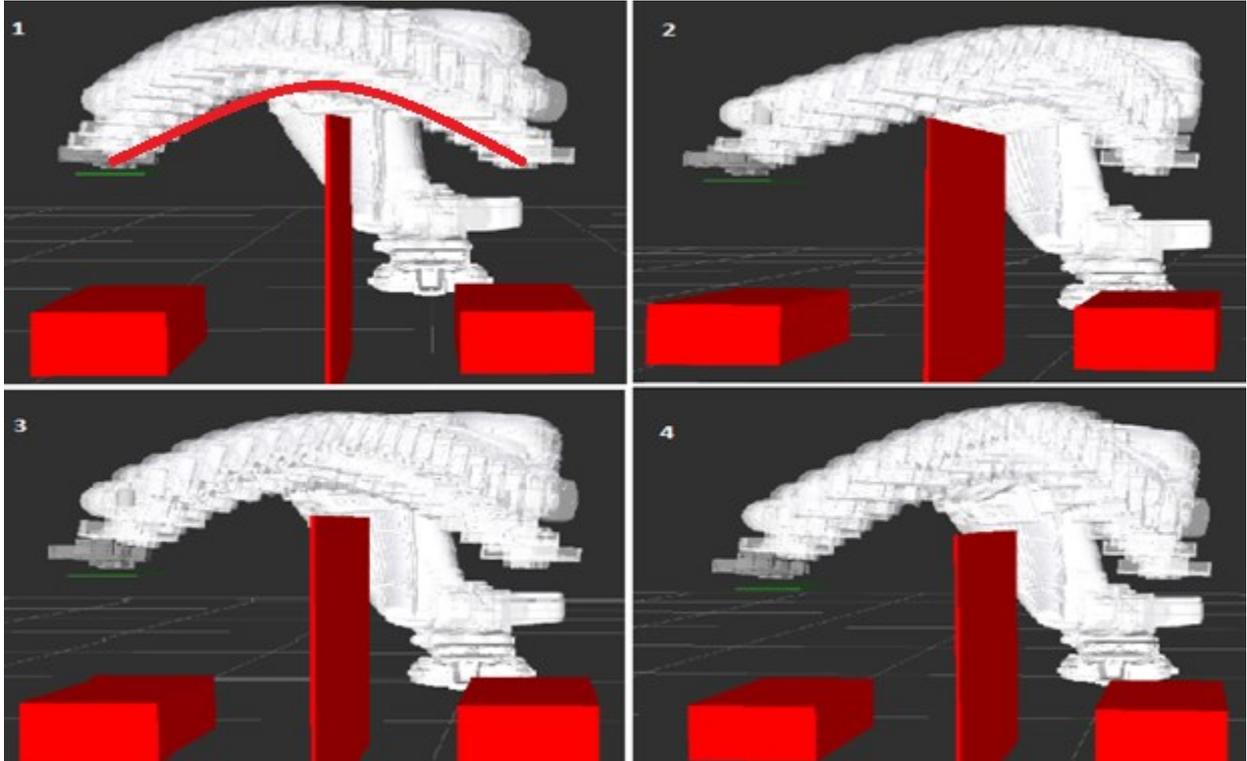


Figure 30: Trajectories produced by STOMP motion planner for avoiding obstacle

Based on the trajectories provided to the robot for avoiding obstacle, it is clear from above observations that STOMP is the best planner than the other two planners. As it provides the trajectories which are smooth and closer to the expected trajectory (indicated by red line). It provides most deterministic response for the path planning.

Since the obstacle avoidance is one of the key tasks that the robot is supposed to perform in this thesis. Therefore, it is concluded from the above analysis that although the *RRTConnectkConfigDefault* is the most efficient planner and provides linear trajectories in the absence of obstacle, but most of the trajectories it provides for avoiding the obstacles are weird and un-natural. It even hits the obstacle on one run. Similar observations can be made on *RRTkConfigDefault* planner. On the other hand, STOMP provides smooth and closer to the expected trajectories in the presence of an obstacle.

Hence, on the basis of the objective of this system that aims to avoid collisions with the obstacles by re-planning a path, STOMP is the best planner to use with ABB IRB4600 robot.

4.5 Mechanism for ROS Integrated Gripper Control of ABB IRB4600 Robot

In order to control the magnetic gripper attached to the IRB4600 robot, there is a need to have the mechanism for I/O control of IRC5 controller (ABB IRB4600 controller) in ROS. Unfortunately,

there is no generic ROS integrated I/O control available for IRC5 controller. However, Fanuc and Universal Robots have generic I/O control available in ROS [54]. There are two possible ways to control the I/O of IRC5 controller through the ROS framework.

- 1) One method is to use field bus (say Ethernet or Profibus) that IRB4600 controller supports. In this case, there is a need to install the compatible I/O interface card in the PC and a separate ROS node needs to be developed for communicating with this interface card and exposing those I/O's.
- 2) Another alternative is to extend the ROS-I *abb_driver* so that it receives the I/O related messages. Develop the ROS nodes for establishing the socket connection with IRC5 controller and sending those specific string type messages from ROS node to controller for handling I/O's.

4.5.1 Factors for Selecting the Gripper Control Method for ABB IRB4600 Robot

The field bus approach mentioned above may demand additional hardware to install. However, it requires least development at the robot side as the *abb_driver* requires no extension in this case. The weak point in this mechanism is that it may create synchronization issues between the ROS network and field-bus network.

On the contrary, the second approach requires no hardware changes. It mainly involves the software changes. It uses the same ROS network. Through this network, we can send defined string header messages corresponding to that specific I/O we need to access.

Based on these factors and the available resources, the second approach is selected for the gripper control of our robot. The mechanism of this approach is described below.

4.5.2 Extension of ROS-I Driver for Receiving String Messages to Control the Gripper of ABB IRB4600 Robot

Appendix A shows the communication mechanism for the motion and gripper control of ABB IRB4600 robot between ROS environment and the IRB4600 robot controller.

Originally, there are two servers in the available *abb_driver* (the driver that is installed in controller of our robot for interfacing with ROS-I). One is the *ROS_StateServer* and the other one is the *ROS_MotionServer*. *ROS_StateServer* is dedicated for sending the current position of each joint of the robot to the ROS (*move_group*: MoveIt! central node) while the *ROS_MotionServer* receives the motion commands from the ROS environment (*move_group*). Both servers listening at different ports (11000 for *ROS_MotionServer* and 11002 for *ROS_StateServer*) over the same TCP/IP network as shown in Appendix A. These servers send and receive the ROS messages to and from the ROS environment through the *move_group* (developed in MoveIt! configuration

package). *Move_group* is the middle ware in this case which integrates the ROS nodes, robot controller, sensors etc. Figure 10 provides the overview of the *move_group* actions and services. These two servers are only dedicated to move the robot to the desired positions and orientations. Therefore, these servers provide no information regarding the I/O's of the robot.

In order to control the gripper, the *abb_driver* is extended with one more server with the name *ROS_GripperServer*. This server is listening at port 11004 of the same network as shown in Appendix A. This server only receives the string messages related to gripper on/off information. Based on these string messages, the digital output dedicated to the gripper is set or reset. The piece of the rapid code (ABB programming language) in the robot controller is shown below. It is used for handling the gripper according to the message received from ROS to controller.

```

WHILE (TRUE) DO
    ROS_receive_msg1 client_socket1, message1;
    IF message1 = "gripperON" THEN
        TPWrite "GripperStatus:"+ message1;
            Set do4MagnetILock; // Activating the magnetic gripper
    ELSEIF message1 = "gripperOFF" THEN
        TPWrite "GripperStatus: gripperOFF";
            Reset do4MagnetILock; // Deactivating the magnetic gripper
    ENDIF
    WaitTime update_rate;
ENDWHILE

```

According to this algorithm, the *message1* is the variable that receives the message (*gripperOn/gripperOFF*) from the ROS node. Based on this message, the digital output associated with the gripper (*do4MagnetILock*) is turned on and off.

On the ROS side, the separate ROS node (*gripperState*) is developed that establishes the connection with the *ROS_GripperServer* (runs at robot controller) and sending the specific string messages (i.e. *gripperON/gripperOFF*) in that connection. This ROS node(*gripperState*) receives the gripper on/off messages from the other active ROS nodes (*cpp_node*: performing pick-and-place task) through the ROS topic (*RobotGripperStatus*) as shown in Appendix B and sends these messages to the *ROS_GripperServer*. This server receives these messages and handles it according to the rapid code mentioned above.

The key point in this communication is that the sending of the string messages has nothing to do with the MoveIt! central node (*move_group*). This is because these string messages flow in a separate connection over the same network, which completely bypasses the *move_group* as shown in Appendix A. The use of one network for sending ROS and gripper control messages avoids the fear of synchronization issue as it might be the problem of using a separate network for sending these messages.

In the same way, this approach can be utilized for handling the other I/Os of IRC5 controller. There is a need to define specific string messages corresponding to each I/O of the IRC5 controller in

some standard table. Extend the *ROS_GripperServer* to set/reset the I/Os of the IRC5 controller against those defined messages (declared in the standard table). Thus, the user may send the defined messages from the ROS node (*gripperState*) to IRC5 controller (*ROS_GripperServer*) for controlling the corresponding I/O.

4.6 ROS Nodes for Robotic Task & Environment Adaptation for Safety

A ROS node is created using C++ programming language to perform the pick-and-place task that also enables the motion planner to keep on planning the path until it finds the collision free path for our robot in presence of obstacles.

4.6.1 Pick-and-Place Task

Pick-and-place operation is generally a common task that robots are designed to perform in the industry. In this implementation, the robot is required to perform the same task. It picks the metallic plate (colored in green in Figure 31) from right pedestal with the help of magnetic gripper and places it to the left pedestal. There is no obstacle in the path of the robot. Thus, the motion planner plans the shortest linear path between the picking and placing target positions via the approach points as shown in Figure 32.

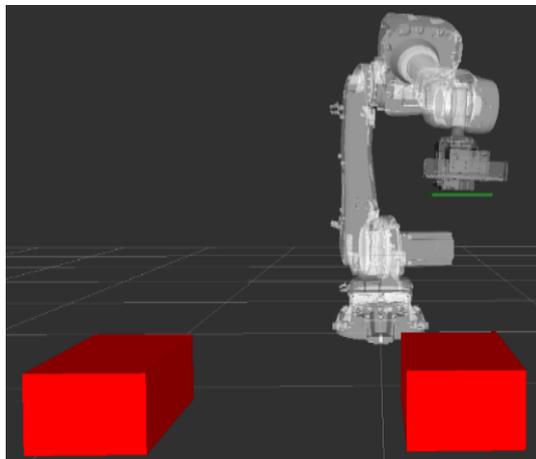


Figure 31 : Pick-and-Place task by ABB IRB4600 robot

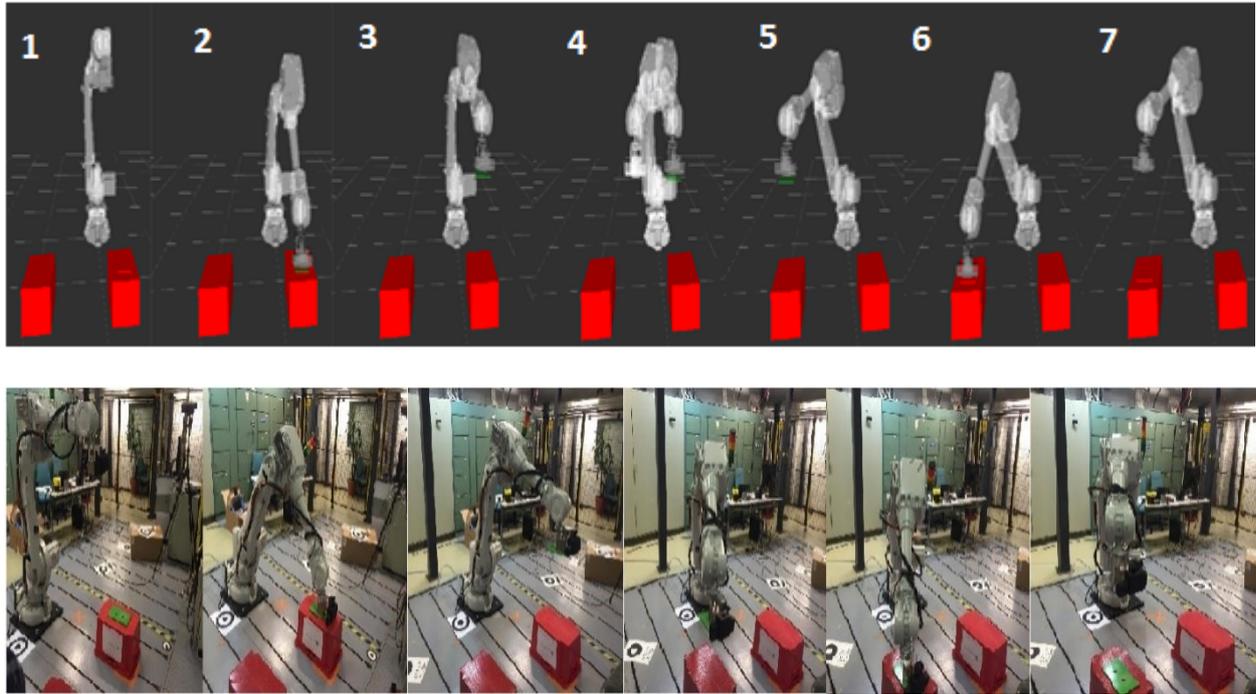


Figure 32: Pick-and-Place task without obstacle in ROS (upper) and actual (lower) environments

All the logic for performing pick-and-place task is developed and control in the ROS environment. ROS is sending commands to the robot controller for performing actions.

A ROS node is developed in C++ using *move_group_interface* package [43] to control the logic for picking and placing the metal plate. Four positions are defined in this ROS node. These positions are retreat (above pick position), pick, preplace and place positions. The ROS node sends these positions to the *move_group* (MoveIt! central node). The *move_group* feeds these positions to the motion planner to plan a collision free path. Once a path is planned, the *move_group* let the kinematic solver to find the joint angles of the robot corresponding to the planned path. The *move_group* finally send these joint angles to the *ROS_MotionServer* running at the robot controller, for moving the robot to these positions. In between these position commands, the gripper on/off string messages are deliver to the controller by the ROS node (*gripperState*). The *ROS_Gripper-Server* running at the robot controller, act to these messages and accordingly activate or deactivate the magnetic gripper for grasping or releasing the metallic plate respectively.

4.6.2 Environment Adaptation for safety of Humans/Objects

In this system, the environment adaptation means the changes occurring in the robot workspace and the robot adapts itself according to these changes. The changes are the appearance of obstacles

including human operators. In this case, the ROS environment integrated with our robot acknowledges those obstacles and decide whether it must stop the robot or plans an alternative path for avoiding them. The collision detection feature of the MoveIt! is utilized for detecting the obstacles in the planning scene of the robot. This thesis refers the environment adaptation and robot reaction feature to the safety of the human operator, if they un-intentionally or intentionally confront the robot path.

The system is tested for the two cases. In the first case, the environment is pre-defined and fixed, which is mapped in the planning scene of the ROS environment of our robot as shown in Figure 33. In this case, the MoveIt! let the motion planner to plan the collision free path for our robot by considering only those objects that are defined in ROS environment. In the second case, the environment is dynamic. Objects (including human operators) may come dynamically to the planning scene. The system must be capable to acknowledge these dynamic objects (including humans) during run-time and considers them as obstacles for planning a collision free path for the robot.

4.6.2.1 Fixed Environment (Fixed Position Obstacles)

Initially, the system is tested with the fixed obstacle. The obstacle of same size and shape present in the real environment is triggered manually to the path of the robot in the ROS environment as shown in Figure 33. The robot plans the path followed by the execution. The algorithm is developed such that the motion planner keeps on planning until it gets a collision free path for the robot. Figure 34 shows the similar case for which the motion planner at first tries to make a straight-line path from position 1 to position 2 but now there is an obstacle created manually in its path. Now the system lets the motion planner to keep on planning until it finds a valid path that avoids the obstacle. If the valid path is not found or it is not possible, the robot will stop at its position and wait until the path clears.

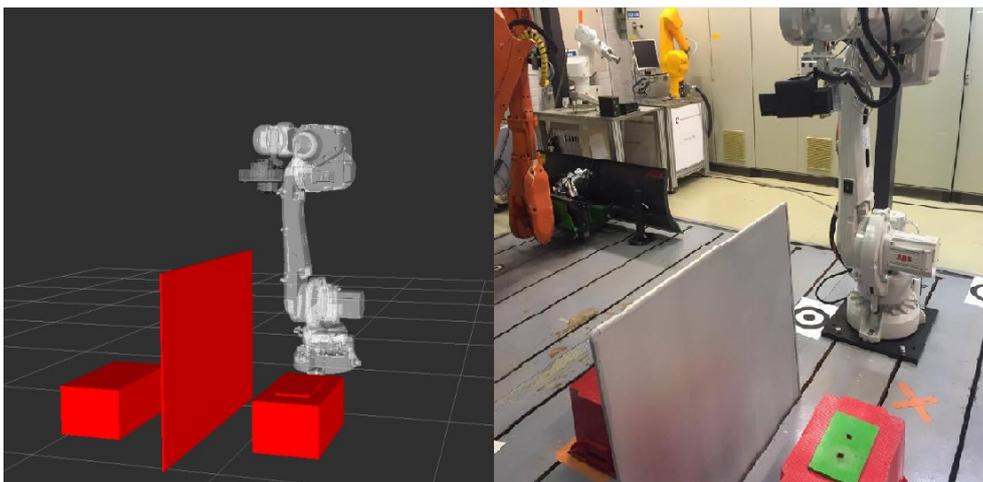


Figure 33: Fix obstacle in (a)ROS environment (b)Actual environment

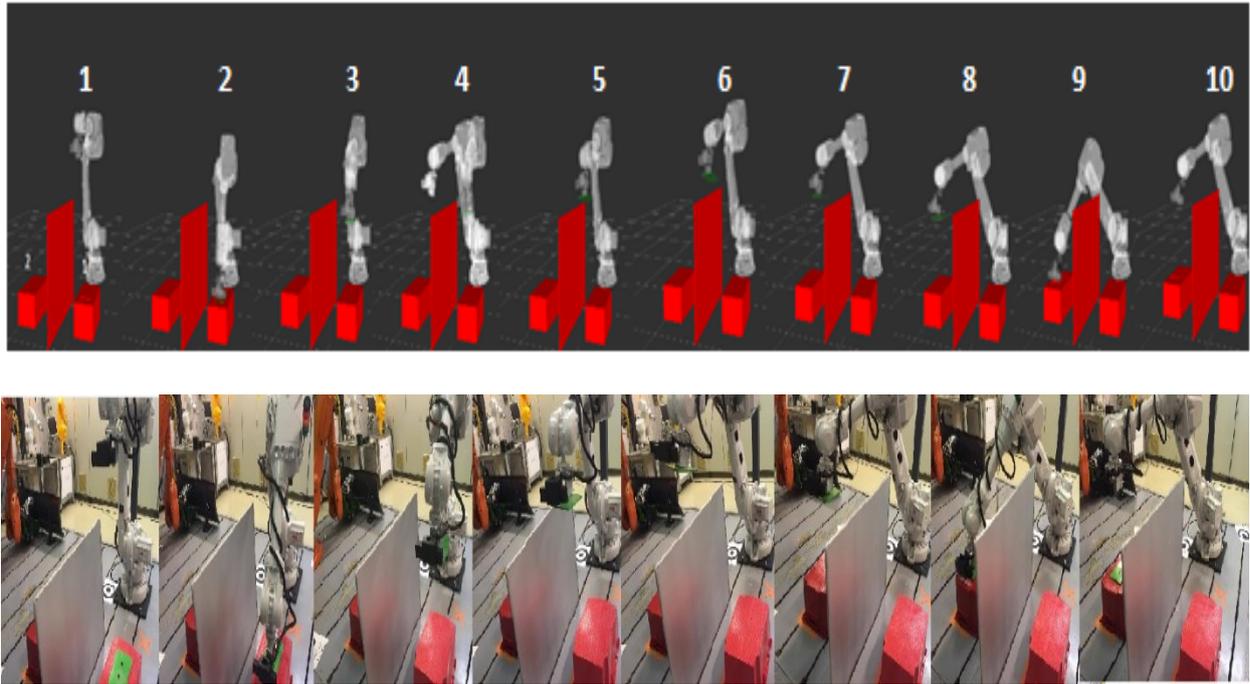


Figure 34 : Robot performing pick-and-place task with fixed obstacle in ROS (upper) and real environment (lower)

4.6.2.2 Dynamic Environment (Dynamic Obstacles)

The above case is for the obstacle at fixed pre-defined position. In the second case, positions of the obstacles are not fixed. They are coming dynamically in the planning scene of the robot. In order to get the real-time objects into the planning scene, there is a need to integrate the 3D sensor with the MoveIt! that provides the visual information of the real environment into the ROS planning scene of the robot.

4.6.2.2.1 Kinect Camera Integration with MoveIt! (ROS)

In this system, the Microsoft Kinect camera is used to observe the real-time changes from the robot environment, and pass that information into the ROS environment. There are some ROS drivers available to integrate this camera with ROS i.e. *freenect_stack*, *openni_kinect*, *kinect*, *kinect_aux* and *iai_kinect2* [50]. All of these drivers provide point clouds. However, this system uses *iai_kinect2* driver [55] developed by Tiemo and Alexis, because it is comparatively easy to setup the camera with ROS using this package. The working of this ROS driver depends on *libfreenect2* [56] package which must be installed to get it functional.

The camera is fixed in the robot workspace as shown in the Figure 1. It is placed in such a way to cover the maximum moving area of the robot. Once the installation of the packages and integration of camera have been completed. The next step is to calibrate the camera with the robot environ-

ment so that visual information provided by the camera gets synchronized with the real environment of the robot. The ROS camera package (*iai_kinect2*) sets the camera co-ordinate frame with respect to the base reference frame of the robot as shown in Figure 35.

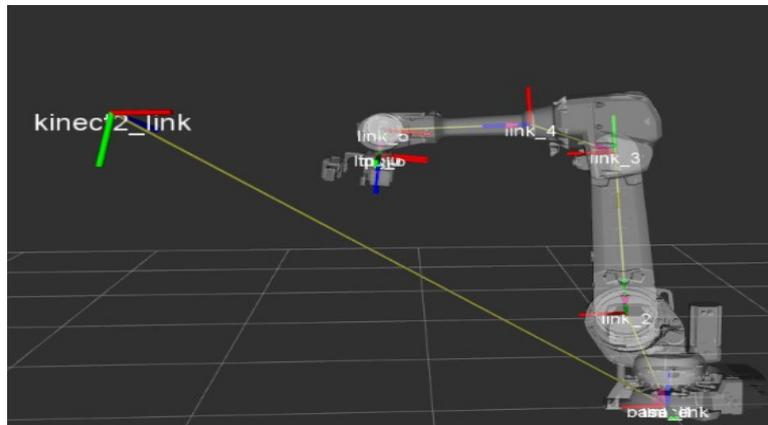


Figure 35 : Camera co-ordinate frame w.r.t robot base (reference frame)

There are three qualities of the image denoted as sd, qhd and hd that Kinect camera provides as shown in Figure 36.

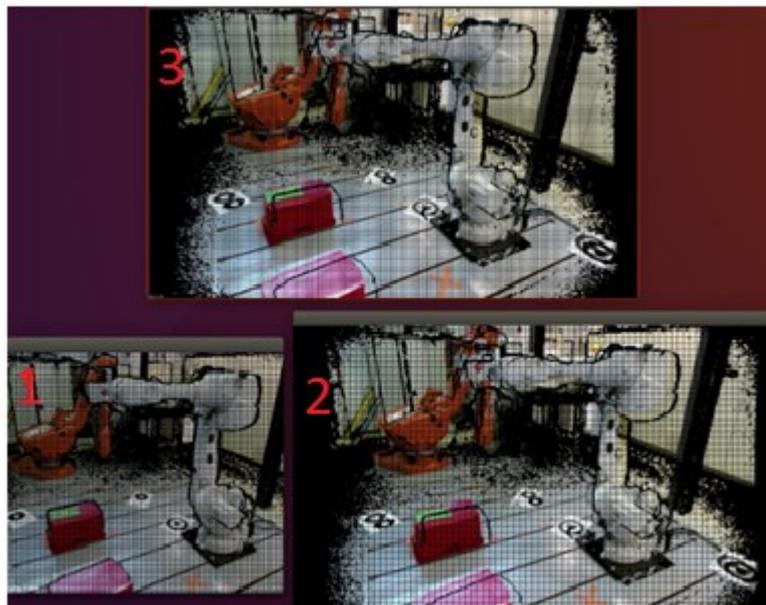


Figure 36: Kinect camera images of robot environment (1) sd image (2) qhd image (3) hd image

Figure 36 is visualization of the environment. MoveIt! can't consider the appearing objects in the images as obstacles. There is a need to create the 3D occupancy map of the robot workspace into the planning scene so that the MoveIt! considers the objects as obstacles. This 3-D occupancy map is known as Octomap.

4.6.2.2.2 OctoMap in MoveIt!

The Octomap is the 3-D occupancy map that provides the three-Dimensional models in the form of volumetric representation of space. It is commonly used for the variety of robotic applications. [57]

MoveIt! has the built-in Octomap updater package that makes Octomap of the actual robot environment into ROS environment (RViz) and let the motion planner plans the collision free path by taking into consideration the objects appeared in the Octomap. This can be implemented by configuring the Kinect camera with the MoveIt!. The configuration file (*sensors_kinect.yaml*) is created in the *config* directory of *abb_irb4600_moveit_config* package that inputs the point cloud data from the Kinect camera through the topic (*kinect2/sd/points*) to the default MoveIt! sensor plugin (*occupancy_map_monitor/PointCloudOctomapUpdater*). This configuration file (*sensors_kinect.yaml*) contains the parameters related to the camera. The range of the camera is adjusted in this file in order to capture only the consult area of the robot environment to avoid un-necessary point cloud data. As a result, Octomap updater package of MoveIt! generates the Octomap that can be shown in RViz by subscribing to the topic (*filtered_output*) as shown green in Figure 37. This Octomap updates dynamically with the change in the point cloud data of the robot environment.

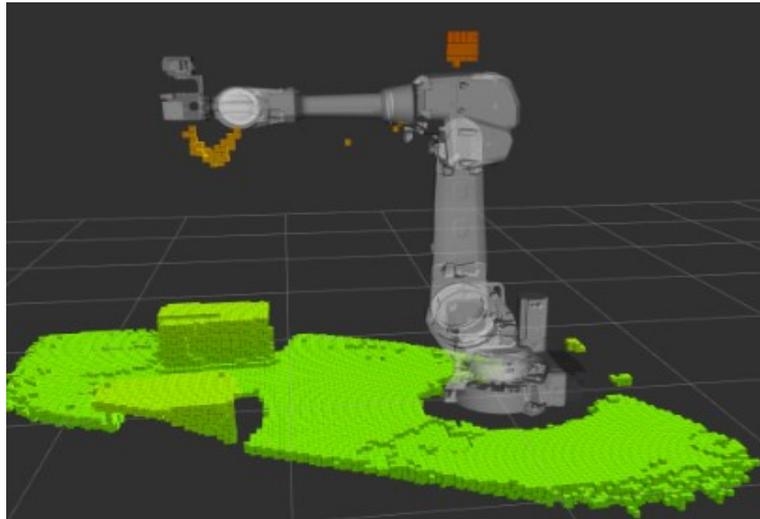


Figure 37: Representation of dynamic robot environment with Octomap

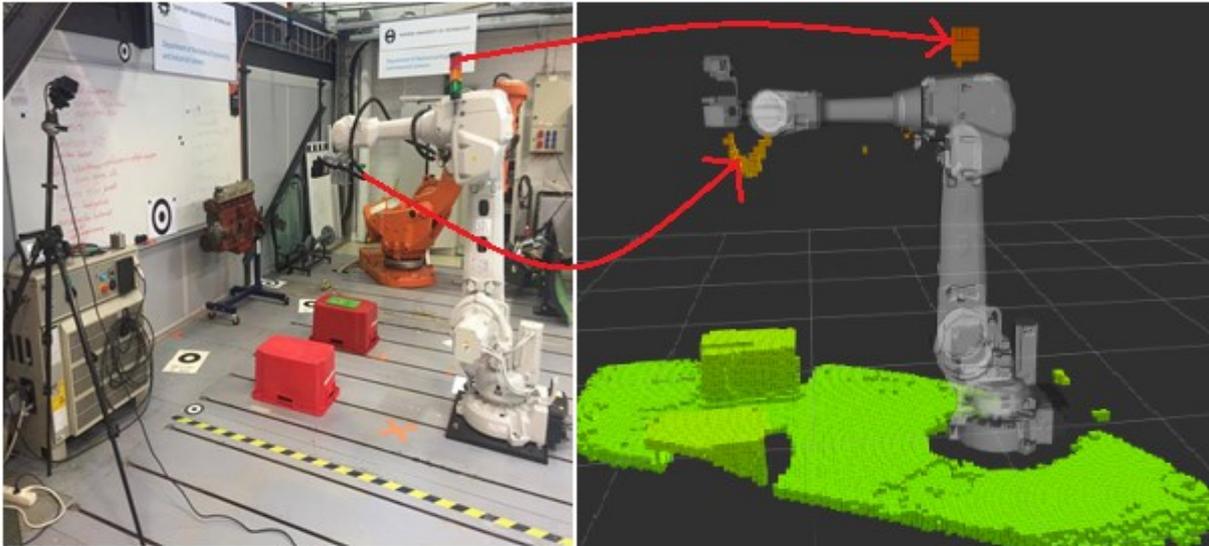


Figure 38 : (a)Real environment (b) ROS environment

Figure 38(b) shows the Octomap representation of the environment that is dynamically updated according to the point cloud data from the camera. It can be observed that the robot itself is not built in the Octomap and surely, it must not be part of the Octomap. Otherwise the model of the robot itself would become the colliding object. While generating the Octomap, the MoveIt! excludes those models from the planning scene which are defined in the URDF file. For this reason, the robot is excluded in the Octomap.

Figure 38 shows some of the Octomap points near the robot model. Basically, these points represent the wires/air pipes wrapping around the robot and the beacon post light as indicated by arrows in Figure 38. Although these axillaries are the part of the robot, but appear in the Octomap as obstacles for the robot, because these are not defined in the URDF. There is a need to define those in the URDF. As an alternative, one may remove the extra components from the robot if they are not in use. A rather quick solution to ignore those from the Octomap is to increase the padding parameter in the configuration file of camera (*sensors_kinect.yaml*). The value can be increase such that it covers all this kind of axillaries of the robot so that these may not appear as obstacle for the robot. This is usually not recommended as any other object or part of the object, if comes to this padding area, would also get excluded from Octomap.

In this implementation, the wires are removed as camera mounted on the robot is not being used in this system. Air pipes are carefully adjusted so that these may not hang around to appear as obstacle. The range of the camera is adjusted so that it may just give the image of the required area of the robot environment. The outcome of the dynamic environment corresponding to our ABB IRB4600 robot environment is finally shown in Figure 39.

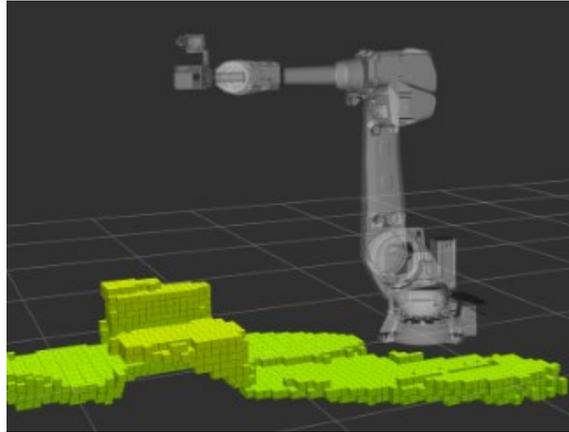


Figure 39: Final outcome of MoveIt! Octomap Updater sensor plugin

This robot application manages to acknowledge the objects dynamically in the robot workspace as shown in Figure 40(a). The industrial robot during performing pick-and-place task is able to change its path if external objects or humans confront its path. Figure 40(b) shows the behavior of the robot to work in a dynamic environment and changing of the path in case of human arm is in its path.

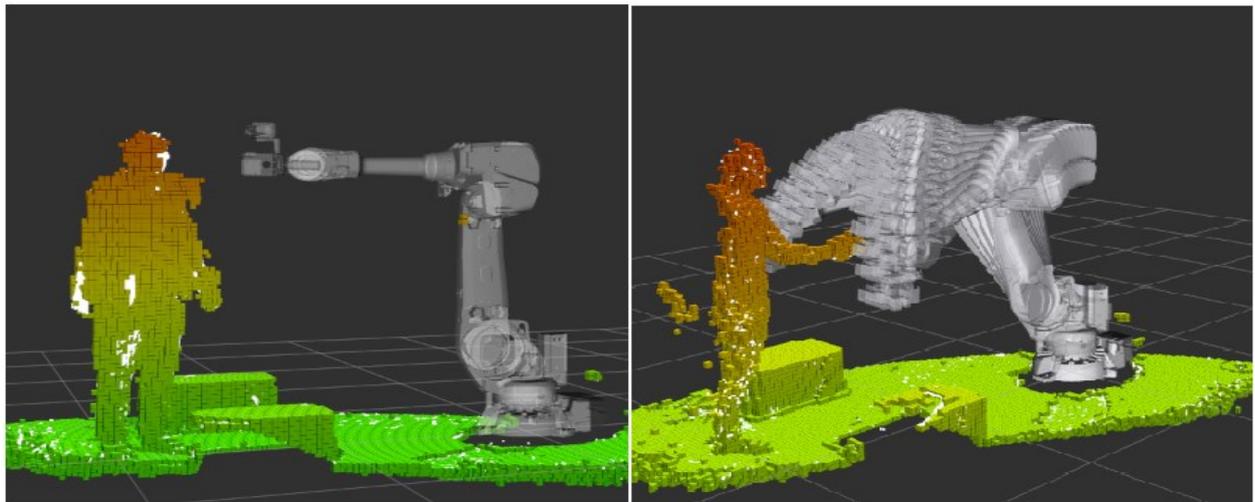


Figure 40: (a) Octomap of human (b) Robot re-planning path for avoiding collision with the human arm

Analysis for choosing the planner is performed in chapter 4. In that case, the system is tested with and without fix obstacle. Here the system is analyzed again with two scenarios. Both scenarios have the obstacle in the robot path. In the first case, the obstacle position is defined beforehand in the planning scene of the robot while in the other case, the obstacles are not defined beforehand. The environment is dynamic and objects may come to the path of the robot dynamically. Both system setups are shown in Figure 41. STOMP planner is used to plan trajectories for avoiding the obstacle.

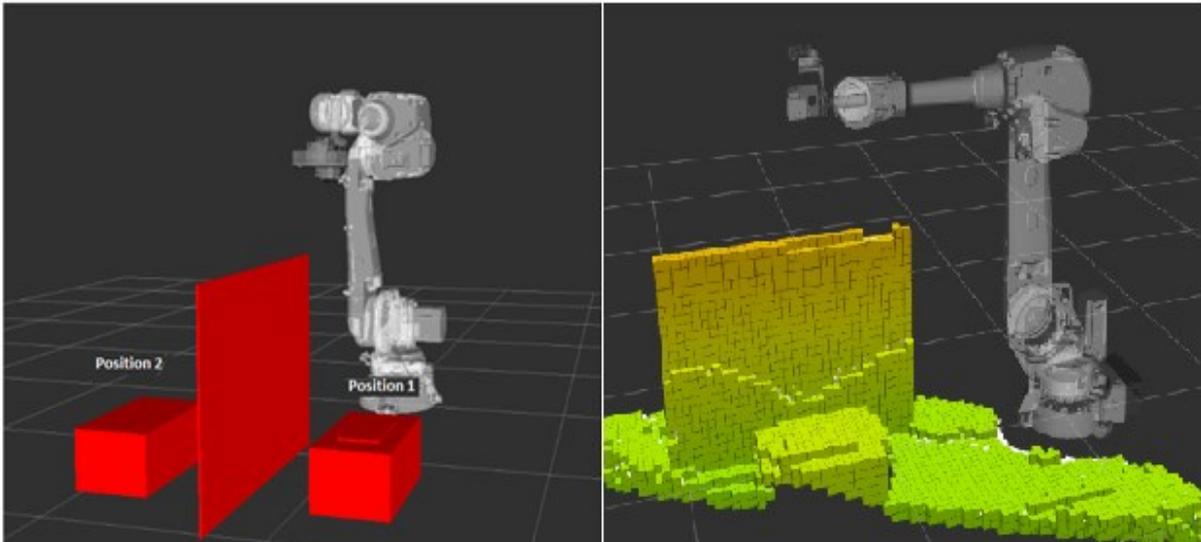


Figure 41: (a) Fixed environment (b) Dynamic environment

The system takes on average 0.5528 sec in case of the fixed obstacle as observed in chapter 4. Now the setup is created to let the motion planner to plan a path for moving the robot between the same two fixed positions (the positions that are used in chapter 4 for the analysis of the planner) in the dynamic environment. The test is performed five times. Table 8 is created that shows the planning time data of the STOMP motion planner for avoiding fixed and dynamic obstacles. Figure 42 shows the box and whisker chart which compares the planning time data for the current setup.

However, the obstacle position is same in both cases. In the first case, the position is already defined in the planning scene while in the second case, the obstacle is built first in the Octomap by the point cloud data from the camera.

Table 8: Planning time data of STOMP motion planner for fixed and dynamic environments

Sr No.	Fixed Environment	Dynamic Environment
1	0.454	0.749
2	0.417	0.7
3	0.558	0.793
4	0.554	1.297
5	0.781	0.878
Average	0.5528	0.8834
Standard deviation	0.126733421	0.214925662
Upper bound	0.679533421	1.098325662
Lower bound	0.426066579	0.668474338
Max Value	0.781	1.297
Min Value	0.417	0.7

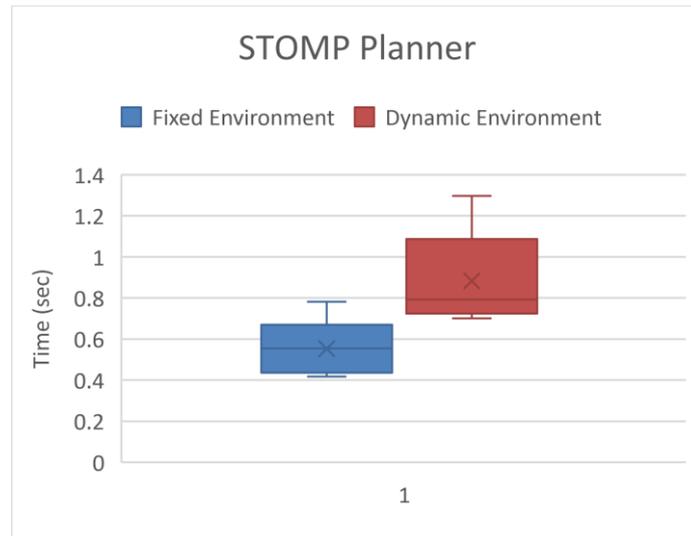


Figure 42: Planning time of STOMP motion planner for fixed and dynamic environments

The above graph shows that the response of the motion planner in case of the dynamic environment is slower than that of the fixed environment. The average time the planner takes to plan a path, while avoiding obstacle, between two positions is 0.8834 sec. This difference of the time is not so big as compared with time of the fixed environment.

On reality, the response of the system in case of dynamic environment is much slower than this time. After completion of each movement, the *move_group* sends request to the motion planner to plan a path for the next target position. In case of the dynamic environment after execution of one movement of the robot, the *move_group* takes around 60 seconds to send next request to motion planner. The reason is that the load on the CPU gets saturated because of large amount of processing and transferring of point cloud data from the camera. The processing of point cloud data and updating of Octomap heavily slows down the overall system. It is observed that the load on the CPU by *move_group* without camera is just 2.9% whereas it is increased to 94% with the camera. It is required to decrease the load on the CPU in order to improve the response of this system in the dynamic environment.

4.7 ROS Architecture of overall System

The overall ROS computation graph of this system, built by using *rqt_graph* GUI plugin of *rqt* ROS package [58], is shown in Appendix B. It is a dynamic graph, which automatically updates on refreshing it and shows all the ROS nodes and topics involved in the system.

The *move_group* is the central node that integrates the ROS nodes, which performs actions through ROS topics and provides ROS services. All commands to the robot controller is sent through this central node. The *Kinect2* is the camera node as shown in Appendix B, which sends the different qualities of images of the environment through different topics. One can get the required image

type into the RViz by subscribing to concerned topic. In this system, the MoveIt! gets point cloud data of the environment from the camera through the topic */kinect2/qhd/points*. All the other ROS nodes and topics are by-default activated when we run MoveIt!. *Cpp_node* and *gripperState* are the ROS nodes which are developed during this thesis work. *CPP_node* performs pick-and-place task and it enables the motion planner to keep on planning the path for our robot until it finds valid collision free path. *GripperState* is only node in this network that sends data (string type messages) directly to the controller while bypassing the *move_group*.

5. DISCUSSION

The limitations of this thesis mentioned in chapter 1, offer a setup that creates the ROS-enabled demo industrial robot environment. Based on these limitations and objective of the thesis, three research questions arise to achieve the goal.

One of the research questions is to implement ROS-I architecture into our industrial robot environment (ABB IRB4600 robot). This requires the ROS packages related to the robot modelling and configuring our robot with the ROS-I framework. *Abb_irk4600_support* and *abb_irk4600_moveit_config* are the packages which integrate our robot with the ROS-I architecture. These packages are developed by following the same standard structure that is followed in other packages of ABB robot models, present in ROS-I repository. The URDF is the main file of our created support package that represents our robot model in XML format. This file makes the 3D model of our robot in ROS environment. The same file is loaded into the *MoveIt! Setup Assistant* for generating the configuration files related to our robot. These configuration files enable our robot to use MoveIt! capabilities such as motion planning, collision detection, 3D perception of robot environment and solving kinematics.

The other research question is the use of the ROS libraries, capabilities and tools to enable our robot capable of adapting and reacting to the workspace changes. These changes refer to the appearance of obstacles in the robot workspace. This system acknowledges and reacts to these changes, by re-planning an alternative path for avoiding them. Collision detection feature of MoveIt! is used to detect the obstacles (objects/humans) appearing in the planning scene of the robot. A ROS node is developed which enables the motion planner to keep planning the path for our robot until it finds a collision free path.

The last research question is to make this system workable for the fixed as well as for the dynamic environments for a general production task. A ROS node is developed which enables our robot to perform pick-and-place task. In the case of the fixed environment, the obstacle is created manually while in case of the dynamic environment, the camera is integrated with ROS to get the visual information of the robot workspace into the ROS environment of our robot. In both the cases, the robot keeps its operation by re-planning a path to avoid obstacles.

Moreover, the analysis for selecting the motion planner is performed in this thesis. Two of the motion planning libraries are analyzed according to their planning times and shape of the provided trajectories. These are OMPL and STOMP motion planners. OMPL is the motion planning library which MoveIt! uses by-default. There are nine motion planning algorithms in this library. All algorithms of OMPL and STOMP planners are tested with our robot in the presence and absence of obstacle. Most of the planners does not provide solution for our robot on both of these cases. Only two OMPL (RRTConnectkConfigDefault, RRTkConfigDefault) and STOMP motion planners provide solution on every try for our robot with each case (with/without obstacle).

The OMPL motion planners are quite efficient than STOMP motion planner. But most of the trajectories they provide in case of obstacle, are quite weird and un-natural. On the other hand, STOMP motion planner provides smooth and closer to expected trajectories. Based on the smooth and shorter trajectories, STOMP is selected to be the best motion planner for our robot in the current setup.

The mechanism of ROS integrated gripper control for ABB IRB4600 robot is presented in this thesis. However, in this system, the gripper control mechanism is used to control the gripper of our robot. But this mechanism can be used as generic I/O control mechanism for IRC5 controller. This can be implemented by extending the *ROS_GripperServer* in such a way to add all the I/O related string messages that should be defined in some standard table. The user will send the defined message to the *ROS_GripperServer* from the ROS node (gripperState) to set/reset the desired I/O.

The implementation of ROS-I framework into our industrial robot environment makes it possible to integrate the variety of advance devices with our robot. However, this thesis presents the implementation of the ROS-I architecture into the specific industrial robot environment. But the ROS packages developed in this implementation can be utilize to any other industrial robot environment that contains ABB IRB4600 robot.

The collision avoidance attribute of this system in the dynamic robot environment takes care of the safety of the humans working in the robot environment. It enables the robot to detect their presence and avoid collision with them by re-planning an alternative path, if they intentionally or un-intentionally confronts the robot's original path. Moreover, this feature is an additional measure of safety for humans sharing the robot workspace other than safety standards (ISO/TS 15066, ISO 10218-1 & 10218-2) of human robot collaboration.

The response of this system is very slow in the dynamic environment. The large amount of processing and transferring of point cloud data from the camera are handled by the CPU. This processing and transferring of point cloud data saturate the load on the CPU, which slow down the overall system. There is a need to separate out the processes of handling the point cloud data and motion control of robot in two different systems so that they work independently. That may result to increase the overall response of the system.

This thesis implements the system with one camera in the robot workspace. In the future, it is recommended to use multiple cameras to capture the maximum area around the robot. However, the system needs to be refined in terms of efficiency and reliability in order to implement it commercially.

It is recommended to add the teaching feature to this system. At present, this system plans a new trajectory on every cycle, to avoid obstacles. However, in case of fixed or static obstacles even in the dynamic environment, the robot should follow the same collision free trajectory, which the motion planner planned for the first time. This implies that the system should compare the current environment with the changing environment. If both the environments are same, the previous

planned path should be resumed, instead of triggering the motion planner again to plan the collision free path. This teaching of system will make the system efficient and save the time, which the planner takes to plan a new trajectory on every cycle.

6. CONCLUSION

For achieving the goal, the overall objective is organized into the set of tasks. The accomplishment of the tasks leads to the implementation of the ROS-I architecture into our industrial robot environment (ABB IRB4600 2.05/60). Consequently, the goal to make the robot adaptive and reactive to its environmental changes is achieved by using ROS capabilities, libraries and tools. This thesis presents the system which is implemented in specific industrial-robot environment as shown in Figure 38. However, this system exhibits the general idea to take industrial robots out of the cages and enable them to work in a dynamic environment. The environment adaptation and reaction feature of this system takes into consideration the collision avoidance with the human operators working in the robot workspace. This results an additional safety measure for the workers sharing the industrial robot workspace, along with already configured safety standards (ISO/TS 15066 and 10218-1 & 10218-2) of human-robot collaboration.

The integration of ROS with the industrial-robot environment provides the benefits of using advance and growing collection of devices by using the packages and drivers developed by the worldwide ROS community. The opportunity of using them with the industrial-robots allows the addition of variety of features. Like, in this thesis we have utilized the ROS packages and tools available in the ROS consortium for the integration of the 3D sensor (camera) that enables the robot to acknowledge the objects in the robot environment and use that information accordingly.

Hence, this thesis develops the demo industrial robot environment using ROS capabilities and tools in which the system allows the industrial robot to work in the fixed as well as in dynamic environments and re-plans its path in case of obstacles (humans/objects). However, the response of the system in case of the dynamic environment is quite slow.

REFERENCES

- [1] Frank JA, Moorhead M, Kapila V. Mobile Mixed-Reality Interfaces that Enhance Human-Robot Interaction in Shared Spaces. *Frontiers in Robotics and AI*. 2017.
- [2] Power R, Naysmith J. *Action research: A guide for associate lecturers*.
- [3] Ayres R, Miller S. The impacts of industrial robots. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST; 1981 Nov.
- [4] Cubero S. *Industrial robotics: Theory, modelling and control*. Pro Literatur Verlag; 2006.
- [5] [Online]. Available: <https://princesss63.wordpress.com/2013/04/24/the-advantages-and-disadvantages-of-industrial-robots/>. [Accessed 19 7 2017].
- [6] Fryman J, Matthias B. Safety of industrial robots: From conventional to collaborative applications. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on 2012 May 21* (pp. 1-5). VDE.
- [7] [Online]. Available: <http://weldingdesign.com/operations/modular-work-cell-safety-fencing>. [Accessed 20 7 2017].
- [8] International Organization for Standardization, [Online]. Available: <https://www.iso.org/standard/62996.html>. [Accessed 10 7 2017].
- [9] ROS.org, [Online]. Available: <http://www.ros.org/>. [Accessed 10 3 2017].
- [10] ROS Wiki, [Online]. Available: <http://wiki.ros.org/>. [Accessed 7 3 2017].
- [11] Kerr J, Nickels K. Robot operating systems: Bridging the gap between human and robot. In *System Theory (SSST), 2012 44th Southeastern Symposium on 2012 Mar 11* (pp. 99-104). IEEE.
- [12] Martinez A, Fernández E. *Learning ROS for robotics programming*. Packt Publishing Ltd; 2013 Sep 25.
- [13] 3rd ed., Mahtani A, Sanchez L, Fernandez E. *Effective Robotics Programming with ROS*. Birmingham: Packt Publishing Ltd; 2016.

- [14] WillowGarage, [Online]. Available: <http://www.willowgarage.com/pages/software/ros-platform>. [Accessed 7 3 2017].
- [15] MOOS, [Online]. Available: <http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php>. [Accessed 10 3 2017].
- [16] The Player Project, [Online]. Available: <http://playerstage.sourceforge.net/>. [Accessed 10 3 2017].
- [17] Microsoft Robotics Studio, [Online]. Available: <http://msdn.microsoft.com/en-us/robotics>. [Accessed 3 7 2017].
- [18] ROS Packages, [Online]. Available: <http://www.ros.org/browse>. [Accessed 8 3 2017].
- [19] O'Kane JM. A gentle introduction to ROS.
- [20] Joseph L. Mastering ROS for robotics programming. Packt Publishing Ltd; 2015 Dec 21.
- [21] ROS Introduction, [Online]. Available: <http://wiki.ros.org/ROS/Introduction>. [Accessed 10 3 2017].
- [22] ROS Distributions, [Online]. Available: <http://wiki.ros.org/Distributions>. [Accessed 11 3 2017].
- [23] ROS Concepts, [Online]. Available: <http://wiki.ros.org/ROS/Concepts>. [Accessed 10 3 2017].
- [24] Joseph L. Learning Robotics Using Python. Packt Publishing Ltd; 2015 May 27.
- [25] ros_comm, [Online]. Available: http://wiki.ros.org/ros_comm. [Accessed 18 3 2017].
- [26] ROS-Industrial, [Online]. Available: <http://wiki.ros.org/Industrial>. [Accessed 27 6 2017].
- [27] ROS-I Tutorials, [Online]. Available: <http://wiki.ros.org/Industrial/Tutorials>. [Accessed 20 5 2017].
- [28] ROS ABB, [Online]. Available: <http://wiki.ros.org/abb>. [Accessed 10 5 2017].
- [29] ROBOTIQ, [Online]. Available: <http://blog.robotiq.com/bid/70845/What-is-ROS-and-ROS-Industrial-for-Robots>. [Accessed 12 4 2017].

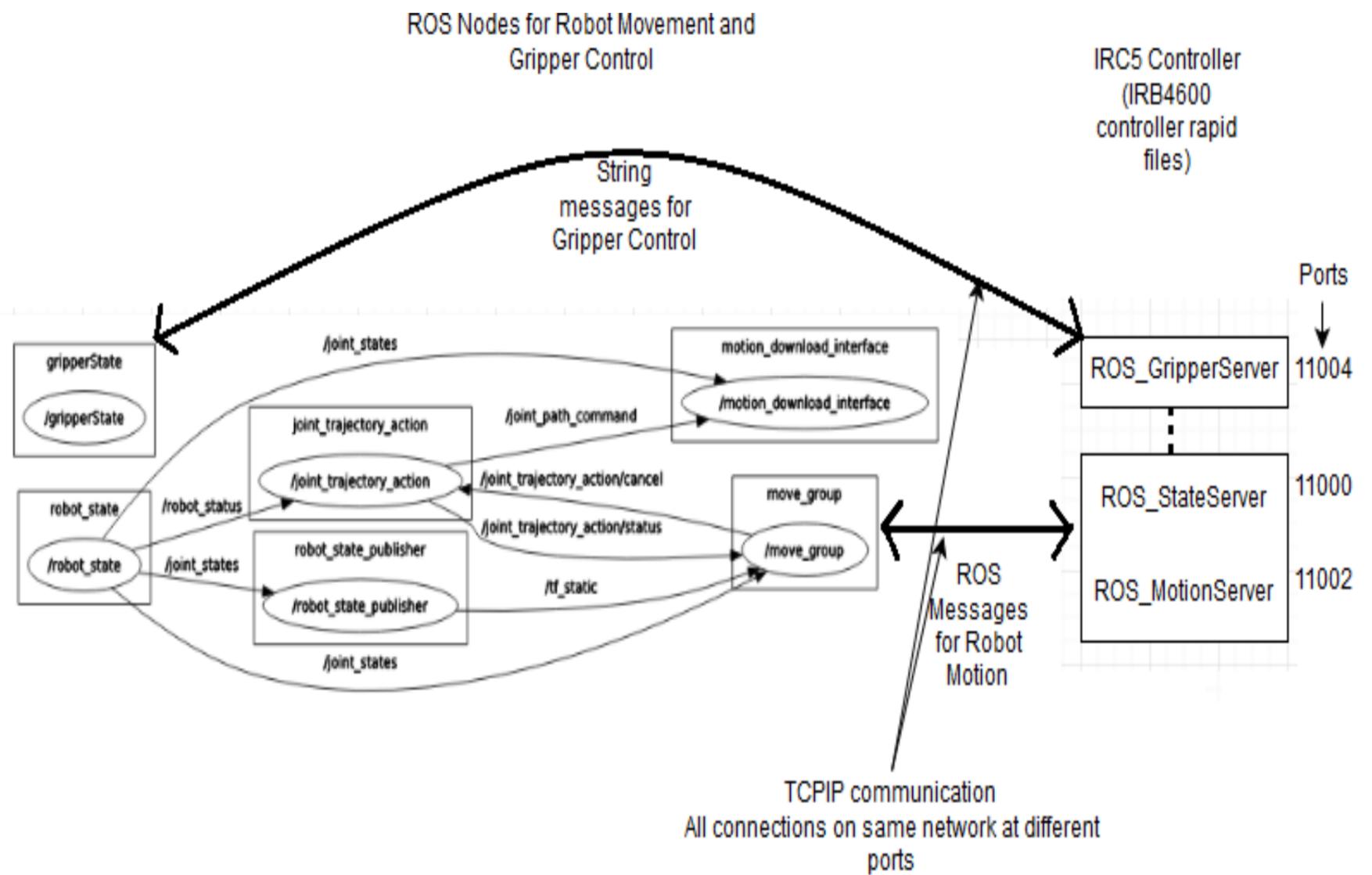
- [30] ABB /ROS Meta-package, [Online]. Available: <https://github.com/ros-industrial/abb>. [Accessed 16 5 2017].
- [31] ABB ROS-I Experimental Repository, [Online]. Available: (https://github.com/ros-industrial/abb_experimental). [Accessed 26 7 2017].
- [32] Open ABB ROS Driver Package, [Online]. Available: https://github.com/robotics/open_abb. [Accessed 16 5 2017].
- [33] ABB Robots ROS-I Driver, [Online]. Available: https://github.com/ros-industrial/abb/tree/indigo/abb_driver. [Accessed 22 5 2017].
- [34] Qian W, Xia Z, Xiong J, Gan Y, Guo Y, Weng S, Deng H, Hu Y, Zhang J. Manipulation task simulation using ros and gazebo. InRobotics and Biomimetics (ROBIO), 2014 IEEE International Conference on 2014 Dec 5 (pp. 2594-2598). IEEE.
- [35] Goebel RP. ROS by example. Lulu. com; 2015.
- [36] Quigley M, Gerkey B, Smart WD. Programming Robots with ROS: a practical introduction to the Robot Operating System. " O'Reilly Media, Inc."; 2015 Nov 16.
- [37] Nüchter A, Hertzberg J. Towards semantic maps for mobile robots. Robotics and Autonomous Systems. 2008 Nov 30;56(11):915-26.
- [38] Xacro, [Online]. Available: <http://wiki.ros.org/xacro>. [Accessed 10 4 2017].
- [39] XacroToCleanUpURDF, [Online]. Available: <http://wiki.ros.org/urdf/Tutorials/UsingXacroCleanUpURDFFile>. [Accessed 10 4 2017].
- [40] Chitta S, Sucas I, Cousins S. Moveit![ROS topics]. IEEE Robotics & Automation Magazine. 2012 Mar;19(1):18-9.
- [41] Koubaa A. Robot operating system (ROS). Springer Verlag; 2017.
- [42] MoveIt!, [Online]. Available: <http://moveit.ros.org/documentation/concepts/>. [Accessed 30 5 2017].
- [43] MoveIt! Cpp Interface, [Online]. Available: http://docs.ros.org/jade/api/moveit_ros_planning_interface/html/index.html. [Accessed 23 7 2017].

- [44] MoveIt! Python Interface, [Online]. Available: http://docs.ros.org/jade/api/moveit_commander/html/index.html. [Accessed 23 7 2017].
- [45] Ioan A. Şucan, Mark Moll, Lydia E. Kavraki, The Open Motion Planning Library, IEEE Robotics & Automation Magazine, 19(4):72–82, December 2012.
- [46] MoveIt! IK Fast Documentation, [Online]. Available: http://docs.ros.org/hydro/api/moveit_ikfast/html/doc/ikfast_tutorial.html. [Accessed 20 5 2017].
- [47] ABB Manual, [Online]. Available: <https://library.e.abb.com/public/c42d0ea4327d473ea9b00f50351ed047/3HAC032885-en.pdf>. [Accessed 6 6 2017].
- [48] FIPA, [Online]. Available: http://www.fipa.com/en_GB/newsItem/1213111-Magnetic-grippers. [Accessed 6 6 2017].
- [49] Microcoft Kinect Sensor, [Online]. Available: <http://shiffman.net/p5/kinect/>. [Accessed 7 6 2017].
- [50] Kinect, [Online]. Available: <http://wiki.ros.org/kinect>. [Accessed 7 6 2017].
- [51] WorkingWithRosIndustrialRobotSupportPackages, [Online]. Available: <http://wiki.ros.org/Industrial/Tutorials/WorkingWithRosIndustrialRobotSupportPackages>. [Accessed 13 6 2017].
- [52] Spong MW, Hutchinson S, Vidyasagar M. Robot modeling and control. New York: Wiley; 2006 Dec, vol. 3.
- [53] MoveIt! Setup Assistant Tutorial, [Online]. Available: http://docs.ros.org/hydro/api/moveit_setup_assistant/html/doc/tutorial.html. [Accessed 24 7 2017].
- [54] ROS-I Supported Hardware, [Online]. Available: http://wiki.ros.org/Industrial/supported_hardware. [Accessed 21 6 2017].
- [55] Microsoft Kinect V2 Driver, [Online]. Available: <http://www.ros.org/news/2014/09/microsoft-kinect-v2-driver-released.html>. [Accessed 16 7 2017].
- [56] Libfreenect, [Online]. Available: <https://zenodo.org/record/50641#.WWuJ3GeW79s>. [Accessed 24 7 2017].

[57] Hornung A, Wurm KM, Bennewitz M, Stachniss C, Burgard W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*. 2013 Apr 1;34(3):189-206.

[58] rqt ROS package, [Online]. Available: http://wiki.ros.org/rqt_graph. [Accessed 21 7 2017].

APPENDIX A: COMMUNICATION MECHANISM BETWEEN ROS-I AND CONTROLLER FOR ROBOT MOTION AND GRIPPER CONTROL



APPENDIX B: ROS COMPUTATION GRAPH OF OVERALL SYSTEM

