USING A GENERAL ROBOT PROGRAMMING SYSTEM TO CONTROL AN INDUSTRIAL ROBOT

Bachelor Degree Project in Automation Engineering
Bachelor Level 30 ECTS
Spring term 2018

Victor Igelmo

Supervisor: Patrik Gustavsson
Examiner: Richard Senington
Abstract

Industrial robot programs are usually created with the programming language that the manufacturer provides. These languages are often limited to cover the common usages within the industry. However, when a more advanced program is needed, then third-party programs are often used to, e.g., locating objects using vision systems, applying correct force with force torque sensors, etc.

Instead of using both the language of the robot and third-party programs to create more advanced programs, it is preferable to have one system that can fully control the robot. Such systems exist, e.g., Robot Operating System (ROS), Yet Another Robot Language (YARP), etc. These systems require more time to fully set up, but once they are set up supposedly they can be used for a lot of different applications and can be used on several industrial robots from different manufacturers.

Currently, University of Skövde have robots from Universal Robots (UR) with several peripheral equipment which has limited control because the built-in language does not support it. Therefore, they need help with both investigating which robot system could be used and implementing that robot system.

This thesis will prove the suitability of using ROS to control aforesaid hardware, fulfilling all the requirements. It will be also demonstrated the feasibility of ROS in the long-term, according to the future plans for this equipment in University of Skövde.
# Table of Contents

ABSTRACT ....................................................................................................................... 2

1 INTRODUCTION ........................................................................................................... 8

1.1 BACKGROUND ......................................................................................................... 8
1.2 PROBLEM DESCRIPTION ....................................................................................... 9
1.3 AIMS AND OBJECTIVES ....................................................................................... 10
1.4 EXTENT AND DELIMITATION ............................................................................... 10
1.5 SUSTAINABILITY .................................................................................................... 10
1.6 THESIS STRUCTURE ............................................................................................... 11

2 THEORETICAL FRAMEWORK ...................................................................................... 11

2.1 ROBOT MIDDLEWARE ............................................................................................ 11
2.2 ROBOTIC ARMS ..................................................................................................... 11
2.3 PERIPHERAL EQUIPMENT ...................................................................................... 12
2.4 HRC ......................................................................................................................... 12

3 LITERATURE REVIEW .................................................................................................. 12

3.1 OVERALL PERSPECTIVE OF MIDDLEWARES ...................................................... 12
3.2 SCOPE IN THE CHOOSING TASK ......................................................................... 13
3.2.1 CAPABILITY TO CONTROL EVERY DEVICE .................................................. 13
3.2.2 FREE-SOURCE .................................................................................................. 13
3.2.3 PERCEPTION ..................................................................................................... 14
3.2.4 ARCHITECTURE ............................................................................................... 14
3.2.5 FAULT DETECTION AND RECOVERY .............................................................. 14
3.3 RESEARCHED MIDDLEWARE ................................................................................ 14
3.3.1 MIDDLEWARE COMPARISON ....................................................................... 15
3.3.2 RESEARCH CONCLUSION ............................................................................... 16
3.4 WHY ROS? ............................................................................................................... 16
3.4.1 MAIN ADVANTAGES ....................................................................................... 16
3.4.2 ROS INDUSTRIAL ........................................................................................... 17
3.4.2.1 ROS Industrial libraries for Universal Robots ....................................................... 17
3.4.2.2 ROS Industrial libraries for Robotiq ........................................................................ 18
3.4.3 SIMULATION ............................................................................................................. 18
3.4.4 MOVEIT! AND ADVANTAGES OF ITS USE .............................................................. 19
3.4.5 PERCEPTION SYSTEMS ............................................................................................ 19
3.5 CONCLUSION ............................................................................................................. 19

4 METHOD .................................................................................................................... 20

5 UNDERSTANDING ROS ................................................................................................. 22

5.1 HOW ROS WORKS? .................................................................................................... 22
5.1.1 ROS FILE SYSTEM .................................................................................................. 22
5.1.2 ROS PACKAGES ..................................................................................................... 23
5.1.3 ROS MESSAGES ..................................................................................................... 24
5.1.4 ROS SERVICES ....................................................................................................... 24
5.1.5 ROS COMPUTATION GRAPH LEVEL ...................................................................... 25
5.2 ROS DISTRIBUTIONS ............................................................................................... 25

6 APPLICATION DEVELOPMENT ....................................................................................... 26

6.1 INTRODUCTION .......................................................................................................... 26
6.2 INSTALLING AND SETTING UBUNTU 16.04 LTS ....................................................... 28
6.3 FIRST STEP TO WORK WITH ROS ............................................................................. 28
6.3.1 INSTALLING ROS .................................................................................................... 28
6.3.2 SETTING IDE ......................................................................................................... 29
6.3.3 SETTING WORKSPACE ......................................................................................... 29
6.3.4 INSTALLING ROS-I UR, GAZEBO AND MOVEIT! LIBRARIES ................................ 30
6.4 CREATING PACKAGES ............................................................................................... 30
6.4.1 MYROBOT PACKAGE ................................................................................................. 30
6.4.1.1 myrobot_description package ............................................................................ 31
6.4.1.2 myrobot_gazebo package .................................................................................. 31
6.4.1.3 myrobot_moveit_config package ..................................................................... 33
6.4.1.4 myrobot_usage package .................................................................................... 37
6.4.2 UNIVERSAL_ROBOT PACKAGE ............................................................................. 37
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1</td>
<td>Scientific method (Wilson, L. and Shuttleworth, M. (2009))</td>
<td>20</td>
</tr>
<tr>
<td>Fig. 2</td>
<td>ROS File system level (Joseph, L. (2015))</td>
<td>22</td>
</tr>
<tr>
<td>Fig. 3</td>
<td>Structure of a typical ROS package. (Joseph, L. (2015))</td>
<td>23</td>
</tr>
<tr>
<td>Fig. 4</td>
<td>Structure of the ROS Graph layer. (Joseph, L. (2015))</td>
<td>25</td>
</tr>
<tr>
<td>Fig. 5</td>
<td>Robot application functional scheme.</td>
<td>26</td>
</tr>
<tr>
<td>Fig. 6</td>
<td>MoveIt! setup assistant before (left) and after (right) load robot description</td>
<td>39</td>
</tr>
<tr>
<td>Fig. 7</td>
<td>Self-collision checking MoveIt! tab.</td>
<td>39</td>
</tr>
<tr>
<td>Fig. 8</td>
<td>Virtual joints MoveIt! tab.</td>
<td>40</td>
</tr>
<tr>
<td>Fig. 9</td>
<td>Planning groups MoveIt! tab.</td>
<td>40</td>
</tr>
<tr>
<td>Fig. 10</td>
<td>Robot poses MoveIt! tab.</td>
<td>41</td>
</tr>
<tr>
<td>Fig. 11</td>
<td>End effector MoveIt! tab.</td>
<td>41</td>
</tr>
<tr>
<td>Fig. 12</td>
<td>Passive joints MoveIt! tab.</td>
<td>41</td>
</tr>
<tr>
<td>Fig. 13</td>
<td>Configuration files MoveIt! tab.</td>
<td>42</td>
</tr>
<tr>
<td>Fig. 14</td>
<td>MoveIt! running in RViz.</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 15</td>
<td>Example states running MoveIt! simulation in RViz.</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 16</td>
<td>Virtual objects in RViz</td>
<td>46</td>
</tr>
<tr>
<td>Fig. 17</td>
<td>Gazebo environment running myrobot.</td>
<td>46</td>
</tr>
<tr>
<td>Fig. 18</td>
<td>Example states running MoveIt! simulation in Gazebo.</td>
<td>47</td>
</tr>
<tr>
<td>Fig. 19</td>
<td>ROS Frames</td>
<td>85</td>
</tr>
<tr>
<td>Fig. 20</td>
<td>Robot description tree</td>
<td>86</td>
</tr>
<tr>
<td>Fig. 21</td>
<td>ROS state before run MoveIt! simulation.</td>
<td>87</td>
</tr>
<tr>
<td>Fig. 22</td>
<td>ROS state after run MoveIt! simulation.</td>
<td>88</td>
</tr>
<tr>
<td>Fig. 23</td>
<td>ROS state before run Gazebo simulation.</td>
<td>89</td>
</tr>
<tr>
<td>Fig. 24</td>
<td>ROS state after run Gazebo simulation.</td>
<td>90</td>
</tr>
<tr>
<td>Fig. 25</td>
<td>MoveIt! setup assistant (ABB robot)</td>
<td>92</td>
</tr>
<tr>
<td>Fig. 26</td>
<td>Self-collision checking MoveIt! tab (ABB robot)</td>
<td>92</td>
</tr>
<tr>
<td>Fig. 27</td>
<td>Virtual joints MoveIt! tab (ABB robot)</td>
<td>93</td>
</tr>
<tr>
<td>Fig. 28</td>
<td>Planning groups MoveIt! tab (ABB robot)</td>
<td>93</td>
</tr>
<tr>
<td>Fig. 29</td>
<td>Robot poses MoveIt! tab (ABB robot)</td>
<td>94</td>
</tr>
<tr>
<td>Fig. 30</td>
<td>End effector MoveIt! tab (ABB robot)</td>
<td>94</td>
</tr>
<tr>
<td>Fig. 31</td>
<td>Passive joints MoveIt! tab (ABB robot)</td>
<td>95</td>
</tr>
<tr>
<td>Fig. 32</td>
<td>Example states running MoveIt! simulation in Rviz (ABB robot)</td>
<td>96</td>
</tr>
</tbody>
</table>
# LIST OF ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interfaces</td>
</tr>
<tr>
<td>EOL</td>
<td>End-of-Life</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HRC</td>
<td>Human-Robot Collaboration</td>
</tr>
<tr>
<td>LTS</td>
<td>Long Term Support</td>
</tr>
<tr>
<td>OMPL</td>
<td>Open Motion Planning Library</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PE</td>
<td>Peripheral Equipment</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>ROS-I</td>
<td>Robot Operating System Industrial framework</td>
</tr>
<tr>
<td>SRDF</td>
<td>Semantic Robot Description Format</td>
</tr>
<tr>
<td>TCP</td>
<td>Tool Center Point</td>
</tr>
<tr>
<td>UR</td>
<td>Universal Robots</td>
</tr>
<tr>
<td>URDF</td>
<td>Unified Robot Description Format</td>
</tr>
<tr>
<td>VCS</td>
<td>Version Control System</td>
</tr>
<tr>
<td>YARP</td>
<td>Yet Another Robot Platform</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Years ago, industrial-purpose devices were simply mechanical, with simple monitoring or electronic controls. However, current devices, robots and other machines are controlled by complex systems, as Zhang, P., (2008) illustrates, which use to include software control. Software used for programing and managing complex machines, like robotic arms, is usually provided by the manufacturer. Common practice is to use devices from the same manufacturer when and if possible. However, certain needed features might not be available in devices of a single manufacturer. Products of manufacturers specialized in required devices may be more suitable for the specific purpose. Depending on process requirements, industrial designers should decide between the ease of connecting devices produced by the same manufacturer, or using specific devices produced by a different manufacturer/s. Developers are working on frameworks to control devices that come from different companies. Their main goal is to manage them using a single system. These devices can be classified in two types: sensors and actuator, depending on if they are providing the system with certain measurable data or if they are acting according to the same. Examples of measurable data are distance or forces measurements. Examples of actuator could be pneumatic actuators or electric engines.

According to Bakken, D. et al., (2001), general-purpose systems can contribute to a standardization in the communication between devices mentioned above. In order to improve, general-purpose systems developers are more interested in the industry improvement than in profits. For this reason, open-source philosophy is followed in general-purpose system development, where one of the most important benefits are developer communities. Since everyone in the community is interested in improving the system, each programmer tries to implement his own device. This means that after some time, a lot of devices/machines will be included in the database. Also, personal doubts can be solved through forums. There are more advantages and some disadvantages of using an open-source system which will be discussed in Chapter 3. Managing all the devices throughout one system could give some advantages. For instance, the programmer will be able to communicate with other machines through more than I/O signals or other communication protocols, which might be limited for a certain purpose. That limitation could affect all the devices involved and communication issues could occur. In this thesis, advantages and disadvantages of using one framework will be discussed. It is important to note the necessity of research of different systems, before deciding which one is going to be used, because some limitations could prevent the use of some features. In the same way, it could be very difficult to program some of the devices/machines in the selected system. System suitability depends on the objectives.
Nowadays, automation is one of the most powerful tools of industry. It allows carrying out a concrete task in a safe way, quickly and efficiently. Robotics is usually one the parts of an automated process (McKinsey & Company, 2018). Robotic arms are the most common robots in factories. There are several tasks which this kind of robot can carry out: assembling, blending, welding, transportation, etc. Implementation of new technologies in robotics gives an opportunity of adding new features or improving already existing ones. Such features can be achieved by supplying the robot with senses. Through the appropriate devices and framework to control them, robot can see or hear. The capability of vision provides the robot with the necessary tool to identify objects in its physical environment and select the best way to work with them. Other prominent feature of the incorporation of a vision system is the chance to control the quality of the workobject. Other features, like speech recognition, could be considered as an improvement in communication between humans and robots (Doostdar et al., 2008). Properly programed, speech recognition makes the control of the robot easier. Low-skilled workers can effortlessly manage the robot. This feature can be added readily and cheaply. Other interesting feature could be achieved by including force sensors. These robots would be able to control applied force in a grasping task, for instance. Also, a torque sensor could be useful in certain situations, for instance, screwing processes.

1.2 Problem description

The problem can be described as an excess of different systems when using devices from several manufacturers. Integration of technologies mentioned in previous section could be difficult if the system is not explicitly prepared to support them.

The main problem comes from the necessity to implement devices in a system which cannot take most of their features. Reasons to include devices which cannot be fully-implemented in the system provided by the manufacturer could be:

- To obtain new features
- To improve the system using devices with better characteristics in both hardware and software: response-time, accuracy, reliability, etc.

In this particular case, University of Skövde has hardware which cannot be fully controlled by the built-in software. These devices are a UR5 robotic arm from Universal Robots and 85 model gripper from Robotiq. They are also doing research in the Human Robot Collaboration (HRC) field. Therefore, some needed characteristics to implement HRC will be taken into account, like the possibility to add a vision system or speech recognition to the actual system.
To solve the described problem, designers can use general-purpose frameworks or middleware (defined in Chapter 2). When industrial programmers are trying to use a general-purpose software to include all their devices in the same system, some problems could appear:

- Programming their devices in the system could require a lot of resources.
- Devices are not supported by the system.
- Some of the system characteristics are not suitable for their purpose.

To avoid these problems, a previous research of different systems and framework was needed. Several systems and frameworks can be found, and it is important to know which one is the most suitable for a concrete purpose. Selected software has to be reliable, fast enough and must have necessary capabilities to manage the available hardware.

### 1.3 Aims and objectives

The aim of this thesis is to investigate the feasibility of the integration of UR5 robotic arm and 85 gripper from Robotiq. This aim will be fulfilled when everything is working in a general-purpose framework running in a specific operating system.

To reach aforesaid aim, the objectives of this thesis are:

- Selection of the most suitable framework/middleware and system to use selected devices.
- Develop and implementation of a certain task to check the selected software feasibility.
- Check how difficult is the implementation of all the devices in the framework.

### 1.4 Extent and delimitation

A robotic arm will be programmed to carry out a certain own-designed task. Mentioned task will be designed to include the robot and the gripper, which is one of the peripheral equipments. This application will be enough to determine the feasibility of the system and framework.

### 1.5 Sustainability

Like other researches of automated systems in the industry, it is necessary to be careful during the implementation. It is required to work with great attention when dealing with automated tasks which will replace workforce. Since this thesis is going to be used as a part of a research of University of Skövde, no jobs will be affected. However, author of this thesis wants to ask for responsibility in the implementation.

Always that is feasible, biodegradable materials will be used, in an attempt to generate as less residues as possible. Low-energy equipment will be used when available.
1.6 Thesis structure

This thesis is divided into 8 chapters. Theoretical framework to understand theory behind the project is provided in Chapter 2. Literature review and extracted conclusions are provided in Chapter 3. Followed method and task to do is provided in chapter 4. The basis to understand ROS are provided in Chapter 5. Development process is provided in Chapter 6. Discussion of the results of the project is provided in Chapter 7. Conclusions are provided in Chapter 8.

2 Theoretical framework

2.1 Robot middleware

A robot middleware is a software environment used to develop a robot application. It may include support programs, compilers, code, libraries, tool sets and application programming interfaces (APIs). Bakken et al. (2001) defined middleware as follows: “a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system.” Robot middlewares usually need a certain operating system to work. This is a virtual platform that manages robot frameworks and hardware connected to the machine in which the operating system is installed. Dominant operating systems in the market are Windows, macOS and Linux and all its varieties.

2.2 Robotic arms

Robotics arms are electro-mechanical devices which are made up of links and joints and they can be set to carry out an automated process. Widely implemented in factories and workshops, robotic arms are robust, fast, reliable and flexible. With the addition of a proper tool, they can manage several tasks usually performed by workers. There are too many types of robotic arms, depending on the task. In this case, the study will focus on articulated robots only with rotatory joints.

Robotic arms can be programmed in two ways: either online programming or offline programing. To program online, the robot programmer uses a teach pendant provided with the robot. In this procedure, the used language is always the one which is given by the manufacturer. Online programming is employed to make little changes in the code. Production stops are necessary. Offline programming is developed through computers and simulation environments. Programmers can choose the most suitable framework. Programming the robot this way can be done using robot script, which is a file that contains the information for making the robot work.
Working parameters can be set in the mentioned file. Targets (point in the space to reach) or robot poses can be also defined in the robot script.

2.3 Peripheral equipment

Every device connected to a concrete system can be considered as Peripheral Equipment (PE), which can exchange information with the system, providing it with new features. Focusing on robots, examples are pressure sensors, force sensors, torque sensors, vacuum systems, tool changers, cameras, etc. By adding more devices, robot flexibility can be increased. Using suitable sensors, several information of different kind can be provided to the robot to improve the process, including new features or senses to the robot. On the contrary, using suitable actuators, robot could perform different tasks, like welding, grasping, screwing, etc. Vision systems combine one or more cameras and a computer to manage compiled data. In robotics, vision systems are used to provide the robot with “eyes”, being able to accomplish quality control tasks or space/shape recognition tasks, for example.

2.4 HRC

HRC is a concept which encompasses in general tasks carry out by human workers with the help of robots, enabling direct interaction between robots and human operators. Usually, HRC tries to avoid harmful and repetitive tasks for workers, like carrying heavy loads or repetitive movements, enhancing their productivity at the same time their stress and fatigue are reduced. HRC provide with the opportunity to combine advantages of automation with the flexibility and cognitive capabilities of human workers. In this kind of processes, robot motion does not use to be strictly restricted, giving an opportunity to adapt its movements according to the environment. When HRC is going to be included, robots are specially design for this purpose so they do not need to be confined in a closed area. This kind of robots are called collaborative robots. Safety considerations have to be take in account, like the ability to respond fast to sudden motions of the workers without hurt them.

3 Literature review

3.1 Overall perspective of middlewares

Elkady, A. and Sobh, T., (2012). summarizes most of the middleware which can be used for the purpose of this thesis. In mentioned paper, characteristics of those middleware are perfectly described, providing reader with a good overall perspective. Yet Another Robot Platform (YARP) is not included in the previously mentioned paper, but it is also another middleware under study.

Appropriate literature review for this thesis are other projects in which authors research about different systems. Even so, most of these studies have a system already selected. Some
conclusions have been extracted depending on the system that other researchers chose, trying to figure it out the feasibility for this thesis.

One of the first noticed facts during literature review is that Lum, Joshua S., (2015), De Gier, M. R., (2015), Tsarouchi, P., et al. (2016) and De Figueiredo, M. M., (2015) they use ROS to program robot for general purposes. Other researched middleware’s are used to work with a specific purpose, like humanoid or autonomous robots.

3.2 Scope in the choosing task

After get a comprehensive overview of several available middleware, focus in particular characteristics is needed to select the most suitable. Main characteristics under review are detailed below.

3.2.1 Capability to control every device

According to the problem description, selected middleware has to have capability to control every device available in the system. That means, as starting point, middleware has to be able to control UR5 robotic arm from UR and 2-finger 85 gripper from Robotiq. However, even though they are not going to be used for this thesis, complete system has more devices. Therefore, middleware should be capable to manage:

- UR3, UR5 and UR10 from UR in case developer wants to switch robotic arm model.
- 3-finger adaptive gripper from Robotiq in case developer wants to switch gripper.
- FT300 force torque sensor from Robotiq.
- Wrist camera from Robotiq.
- Speech recognition from microphones.

Ability to add more devices to the system from different manufacturers is an incentive to choose a concrete middleware. Having the possibility to add more devices will provide the system with more flexibility.

Is important to mention that resources to control mentioned devices have to be already developed. Even developers working in this project could be capable to develop those resources (libraries, protocols...), is preferable using of tested resources. The more tested the resources, the more reliable the system will be.

3.2.2 Free-source

Hauge, Ø. (2010) define open source software (OSS) as a “multifaceted phenomenon consisting of a wide spectrum of software products provided by heterogeneous communities using a variety of software development and maintenance practices”.
For the final purpose of this thesis, work with and OSS middleware is the best choice. Biggest advantage to reach the already mentioned objectives is the existence of huge developer communities (Hauge, Ø., 2010 and Oruc’evic-Alagic’, A., 2016). Either to implement devices from different manufacturers or to develop libraries, OSS communities could determine success. Because of this reason, selected middleware will be OSS.

3.2.3 Perception

Selected robotic arm and torque sensor can be classify as an active perception process (Buttazzo, G., 1996), using dynamic sensing. "In active perception, the influence that the actuator movements have on the sensor responses forces the system to react in real-time, causing the control architecture to be hierarchically organized in a multilevel structure of feedback loops." (Buttazzo, G., 1996).

To manage an active perception process, a real-time system is required, which can provide with a real-time reactiveness and real-time capabilities for the components of the system.

3.2.4 Architecture

Depending on the system model and the control model, reaching goals could turn into an easy task. In this concern, the criteria for this thesis had been to choose an architecture which the developer can understand easily. This criterion depends on the previous knowledge of the developer.

3.2.5 Fault detection and recovery

According to Elkady, A. and Sobh, T., (2012) and aiming on manage a safety system, fault detection will avoid damages in the whole system. Recovery and fault mode runtimes could help the robot to finish its tasks in a safe way and it will provide the robot with the capability to start with secure motion.

This is not a real restriction to choose a certain middleware, but it is necessary if someone want to use the developed application for an industrial process. Therefore, fault detection and recovery will be an indispensable capability.

3.3 Researched middleware

Middleware under study had been Orocos, Pyro, Player, Orca, Miro, OpenRTMaist, ASEBA, MARIE, RSCA, MRDS, OPROS, CLARAty, ROS, YARP, SmartSoft, ERSP, Webots and RoboFrame.
To find substantial information of robotic middleware is difficult. The method to discard a specific system had been dismissing the middleware when one the characteristics was not appropriate for this thesis requirements. Considering distinctive features of this particular project, Elkady, A. and Sobh, T., (2012) provide with almost enough content to get the necessary information of each middleware. However, YARP is not included in that research. Therefore, author went through YARP, 2018, to obtain the needed data. The chance to use already developed libraries for the hardware and the possibility to run vision systems in each middleware is not specified in mentioned paper.

3.3.1 Middleware comparison

Table 1 shows scores of middleware under study in alphabetic order:

<table>
<thead>
<tr>
<th>Middleware</th>
<th>Devices libraries</th>
<th>Open source</th>
<th>Crashing procedures</th>
<th>Appropriate architecture</th>
<th>Perception systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASEBA</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CLARAty</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ERSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIE</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Miro</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OpenRTMaist</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OPROS</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Orca</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orocos</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Player</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Pyro</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RoboFrame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RSCA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SmartSoft</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Webots</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>YARP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

*Table 1: Middleware comparative*
3.3.2 Research conclusion

After comparing middleware attributes, focusing on characteristics described in Section 3.2, ROS was selected as most suitable middleware. ROS is the only middleware which fulfil all the requirements and is easy to implement (conclusion obtained once project is done, at the beginning was a supposition).

3.4 Why ROS?

3.4.1 Main advantages

ROS demonstrate ability to carry out real industrial processes when libraries are sufficiently developed. Readers can check in ROS-Industrial, (2018) some real applications of ROS in the industry, like palletizing or machining. Most part of the information can be found about other systems is focused in research or another kind of application, like mobile or autonomous robots. ROS has different mobile-robot based projects. However, several industrial-processes based projects can be also found. For this reason, can be conclude ROS is more reliable to work with this kind of processes.

Several high-end capabilities and tools are included in ROS to use directly in the robot software and they are highly configurable to adapt them properly to a specific purpose. Aforesaid capabilities will save a great deal of programming time. Tools work in a similar way, adding functionalities to the robot software, but they have to be launched. Tools are used for debugging, visualizing and simulations. Support packages to work with common sensors and actuators in robotics can be also interfaced in ROS.

ROS nodes, which are process that perform necessary computation to execute a certain programmed task, can be written in several languages: C/C++, Python, Java, etc. Is up to the developer to use the same language in all the nodes or use a different one, depending on the ease to program it or the desired performance of those nodes. This feature add flexibility to the system and make easier working with other developers. Is possible to control different devices using ROS facilities. Any number of ROS nodes can obtain information from the system and each node can perform different functionalities. If while a certain application is running a node crash, the system can still work. ROS provides robust methods to resume operation even if any sensors or motors are dead. Also, ROS architecture is ready for multitasking processes performing.

ROS works with Unified Robot Description Format (URDF) files. Through these files, developers are able to generate a customized robot description. Some commercial robot descriptions are already available to download and modify them if it is appropriate. This feature solves the initial problem of managing devices from different manufacturers in the same system.
Define them in a new URDF can be a hard task, but once done is easy to implement them. Developers can also include virtual objects to take into account collisions against them. This customize objects can be define in the virtual environment.

While it is true that the learning process is arduous and it will take time, whenever ROS is learned is easy to implement in the system. If necessary libraries are available and properly tested, it will save time. Every day, people around the world work with ROS. Either developing ROS features and its core or libraries to work with. If a concrete feature is not already implemented in ROS, is possible to create a request. Already developed libraries are in continuous development to make them more reliable. Other disadvantages are discussed in Section 7. Mentioned disadvantages did not appear until the middleware implementation stage.

3.4.2 ROS Industrial

"ROS-Industrial is an open-source project that extends the advanced capabilities of ROS to manufacturing automation and robotics. The ROS-Industrial repository includes interfaces for common industrial manipulators, grippers, sensors, and device networks. It also provides software libraries for automatic 2D/3D sensor calibration, process path/motion planning, applications, developer tools and training curriculum that is specific to the needs of manufacturers. ROS-I is supported by an international Consortium of industry and research members. ROS-Industrial:

- Provides a one-stop location for manufacturing-related ROS software.
- Striving towards software robustness and reliability that meets the needs of industrial applications.
- Combines the relative strengths of ROS and existing technology, combining ROS high-level functionality with the low-level reliability and safety of an industrial robot controller, as opposed to replacing any one technology entirely.
- Stimulates the development of hardware-agnostic software by standardizing interfaces.
- Provides an "easy" path to apply cutting-edge research to industrial applications by using a common ROS architecture.
- Provides simple, easy-to-use, well-documented application programming interfaces.”

This information has been extracted from ROS Industrial (2018).

3.4.2.1 ROS Industrial libraries for Universal Robots

One of the libraries included in the ROS Industrial repository is UR library. This library provides the programmer with all the necessary resources to performance motion planning, simulation and control of the physical robot. This library can be founded in ROS Industrial Github
The library status to the date of this thesis release is experimental, which means there are known issues and missing functionality. The use of this library in production systems is not recommended. However, this library is enough to control and simulate UR5 robot, while next version is released.

3.4.2.2 ROS Industrial libraries for Robotiq

One of the libraries included in the ROS Industrial repository is Robotiq library. This library provides the programmer with all the necessary resources to performance motion planning, simulation and control of the physical gripper. This library can be founded in Way Point Robotics Github repository (link in Section 8). The library status to the date of this thesis release is developmental, which means the library has some level of unit-testing. The use of this library will require some modifications and bug fixes. The actual ROS Industrial package for Robotiq is supported by indigo distribution (see Section 5.2). However, with modifications in the repository Way Point Robotics is possible to adapt it to kinetic distribution.

3.4.3 Simulation

Simulating a process is a useful tool for developers. Through simulation, programmers can test their scripts in a safety environment. A virtual environment can be as complex and accurate as developer decide. In simulation developers can set the environment either only for visualization of the scripts or the simulation with actual designed objects.

To plan a robot motion, RViz is the most suitable ROS tool, according to ROS developers (2018). RViz will create a virtual environment to simulate the robot. With RViz, developers will be able to control the robot through Motion Planning resource. RViz is a huge tool with different resources, but for this thesis RViz will provide us, with MoveIt! Running in the background, with:

- Control of different joints individually.
- Control and visualization of the planning path.
- Tool center point (TCP) control.
- Frames visualization.
- Objects (obstacles) visualization.

Gazebo is a virtual environment to simulate the robot. Gazebo includes a robust physics engine, high-quality graphics and graphical interfaces. Gazebo has been developed since 2012. Its actual version is Gazebo 9.0 with end-of-life (EOL) 2023-01-25. This tool is prepared to work perfectly with ROS and other tools, like MoveIt!. New versions of Gazebo are being developed.

With this environment, programmer will be able to:
- Simulate dynamics in an advanced 3D graphics simulation.
- Simulate sensors, optionally with noise from cameras, force-torque, contact sensors and more. A point cloud for *mapping* tasks can be carry out through Gazebo.
- Create custom plugins for robot, sensor and environmental control.
- Create robot models in case programmer need a custom one.
- Run simulations on remote server and interface Gazebo through TCP/IP.
- Run cloud simulations through browser.

### 3.4.4 Movelt! And advantages of its use

"Movelt! is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains" Joseph, L. (2015). For this thesis purpose, Movelt! will be use to define necessary control groups (robotic arm and gripper), carry out the motion planning and predefine certain poses. Movelt! will create the most suitable motion plan to reach a predefine position, taking into account the objects in the virtual environment and avoiding collisions against them. Motion plan creation by Movelt! is an advantage considering sometime developers only have to define poses if the motion path is irrelevant. Movelt! also generate necessary scripts to add useful features to the robot, like joystick control.

### 3.4.5 Perception systems

In this thesis, vision systems or control by speech recognition are not going to be implemented. However, since necessary peripheral equipment is available, selected system has to have capability to work with perception technologies. Generating, for instance, a point cloud in the system, is possible to add perception features, making the robot more flexible. ROS simulation tools like Gazebo, can manage this point could and work with it.

### 3.5 Conclusion

ROS demonstrates ability to adapt to a lot of situations, either finding the appropriate library or developing a new one to satisfy a specific necessity. It is also uses to work with vision systems. Therefore, is appropriate for a long-term. A big developer community supports ROS, making possible to find other people who already developed a fit for purpose library. Some of the studied thesis use their own libraries.

From this literature review, it can be concluded ROS is the most suitable middleware for this project because it uses an intuitive system model and it is already prepared to work with UR
and its peripheral equipment. ROS is not real-time middleware, but the time-response is enough for the purpose of this thesis. More information about time response is provided in Appendix II.

According to Joseph, L. (2015) and as has been explained previously, ROS has some disadvantages regarding learning stage. Having issues in that stage, could mean the failure of the system implementation. Apart on the mentioned community, is possible to find some ROS learning resources on the internet: books, online courses, external tutorials, etc.

4 Method

This thesis will follow the proposed method by Wilson, L. and Shuttleworth, M. (2009), which is a general and iterative scientific method. This basic method have appropriate steps to achieve a conclusion. Figure 1 shows the flow of selected method.

![Fig. 1: Scientific method (Wilson, L. and Shuttleworth, M. (2009))](image)

- Question: Research question are defined. Considerable time is spent in this step to clarify other questions which come from the main question. The problem is also defined in this step. In this case, questions are: Is a robot general-purpose system suitable to control the hardware? Which one is the most suitable? Does this system fulfil the requirements? Question step is detailed on Section 1.

- Collect data: Information is collected. The more information that is gathered, the better the result. For this thesis, useful information are comparatives of middleware and examples of implementation of different systems by other people. Data collection results is expounded on Section 3 (general data) and 5 (ROS information).

- Test hypothesis: Empirical part of the thesis. Practical objective is performance. Middleware is implemented and an analysis of the results is needed. Section 6 provide with information about the whole process.

- Conclusion: Conclusions contain all the results extracted from the project and are needed to have an overall perspective of the project.
Seven tasks and their subtasks have been scheduled to understand all the parts of the system and develop the application properly:

1- **Literature review:**
   a. Research of similar studies and the suggest software to work with.
   b. Research of different studies which have an already selected framework to work with.

2- **Choosing task:**
   a. Analysis of systems characteristics to determine the feasibility to this project.
   b. Deep research of characteristics of every available framework and the system in which they can run.
   c. Go through already developed libraries which can be used to program the actual physical system.
   d. Figure it out advantages and disadvantages of the use of the selected software.

3- **System basis:**
   a. Learn selected middleware basic concepts.
   b. Learn how it works and what is needed to run it.
   c. Install all the software to check its ease.
   d. Complete available tutorials and examples to learn the basis through real examples.

4- **UR robot basis:**
   a. Learn basic UR concepts working in selected middleware.
   b. Research of different UR projects managed by selected middleware.
   c. Search, download, try and understand UR libraries for selected middleware.
   d. Program simple examples and simulate them.

5- **UR robot programming:**
   a. Determine all the details of the proposed application.
   b. Draw flowcharts.
   c. Make an application draft.
   d. Program the application in selected middleware.
   e. Simulate application.

6- **PE basis:**
   a. Learn basic Robotiq’s gripper basic concepts working in selected middleware.
   b. Research of different Robotiq’s gripper projects managed by selected middleware.
   c. Search, download, try and understand Robotiq’s gripper libraries for selected middleware.
   d. Program simple examples and simulate them.
7- **PE programing:**

a. Determine all the details of the purpose application.
b. Draw flowcharts.
c. Make an application draft.
d. Program the application in selected middleware.
e. Simulate application.

5 **Understanding ROS**

5.1 **How ROS works?**

ROS is a Linux-based, open-source, middleware framework for modular use in robot applications. It is a collection of libraries, tools and conventions to help in robot software development. ROS is not an operating system, but it is possible to use it in one of them, like Ubuntu. ROS uses nodes, which are processes that performs computations. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot system will usually comprise many nodes. Nodes can be written by other nodes, called publishers (nodes which provide the system with information), and read by other nodes, called subscribers (nodes which receive information from the system).

5.1.1 **ROS file system**

Figure 2 shows how ROS files are organized on the hard disk:

- **Meta-packages** are a single logical package which group several packages inside. In this thesis, universal_robot package from ROS Industrial repository is a *meta-package* with several packages inside (see Section 6.4.2). Meta-packages only contain a `package.xml` file.
- **Packages** contain nodes, libraries, configuration files and other files which are necessary to performance a concrete action.
- **Package manifest** and **meta-packages manifest** are files which contain information about the package (XML version, package version, name...), author, license, dependencies and other relevant information.

- ROS **messages** (.msg) are a type of information which different ROS processes exchange between them.

- ROS **services** (.srv) are a special interaction between ROS processes. To perform this interaction, ROS processes have to generate **requests** to obtain a certain **reply**.

- **Repositories** are collections of packages which share a common Version Control System (VCS). In this thesis, most of the utilized packages are in Github repository.

### 5.1.2 ROS packages

Figure 3 shows how is the usual ROS package structure:

![Structure of a typical ROS package](image)

*Fig. 3: Structure of a typical ROS package. (Joseph, L. (2015))*

Customized packages do not contain folders when they are created. It is up to the developer to follow this structure, even though it is strongly recommended to follow it.

- **config** folder contains all the configuration files used in the package.
- **include** folder contains headers and libraries used in the package.
- **scripts** folder contains executable Python scripts. `talker.py` and `listener.py` are examples.
- **src** folder contains C++ source codes. `talker.cpp` and `listener.cpp` are examples.
- **launch** folder contains launch files used to run several nodes from the package.
- **msg** folder contains custom message definitions.
- **action** folder contains service definitions.
- **action** folder contains the action definition.
- *package.xml* is the **package manifest** file of this package.
- **CMakeLists.txt** is the **CMake build** file of this package.
5.1.3 ROS messages

ROS messages are a special type of data which are described using a simplified message description language. The data type description of ROS messages are stored in .msg files. A recommended habit in developers is to create a msg subdirectory in the ROS package. Some of the built-in field types available to use in ROS messages are shown in table 1:

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>Serialization</th>
<th>C++</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>unsigned 8-bit int</td>
<td>uint8_t</td>
<td>bool</td>
</tr>
<tr>
<td>int8</td>
<td>signed 8-bit int</td>
<td>int8_t</td>
<td>int</td>
</tr>
<tr>
<td>uint8</td>
<td>unsigned 8-bit int</td>
<td>uint8_t</td>
<td>int</td>
</tr>
<tr>
<td>int16</td>
<td>signed 16-bit int</td>
<td>int16_t</td>
<td>int</td>
</tr>
<tr>
<td>uint16</td>
<td>unsigned 16-bit int</td>
<td>uint16_t</td>
<td>int</td>
</tr>
<tr>
<td>int32</td>
<td>signed 32-bit int</td>
<td>int32_t</td>
<td>int</td>
</tr>
<tr>
<td>uint32</td>
<td>unsigned 32-bit int</td>
<td>uint32_t</td>
<td>int</td>
</tr>
<tr>
<td>int64</td>
<td>signed 64-bit int</td>
<td>int64_t</td>
<td>long</td>
</tr>
<tr>
<td>uint64</td>
<td>unsigned 64-bit int</td>
<td>uint64_t</td>
<td>long</td>
</tr>
<tr>
<td>float32</td>
<td>32-bit IEEE float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>float64</td>
<td>64-bit IEEE float</td>
<td>double</td>
<td>float</td>
</tr>
<tr>
<td>string</td>
<td>ascii string</td>
<td>std::string</td>
<td>string</td>
</tr>
<tr>
<td>time</td>
<td>secs/nsecs unsigned 32-bits ints</td>
<td>ros::Time</td>
<td>rosypy.Time</td>
</tr>
<tr>
<td>duration</td>
<td>secs/nsecs unsigned 32-bits ints</td>
<td>ros::Duration</td>
<td>rosypy.Duration</td>
</tr>
</tbody>
</table>

Table 2: Built-in field types ROS messages (Joseph, L. (2015))

A special type of ROS message is called message headers. Headers can carry information such as time, frame of reference or frame_id, and sequence number. Using headers, developers can get numbered messages and more clarity in who is sending the current message. The header information is mainly used to send data such as robot joint transforms (TF).

5.1.4 ROS services

The ROS services are a request/response type communication between ROS nodes. One node will send a request and wait until it gets a response from the other.

Service definition are stored in a file called .srv in the srv subdirectory of the package. A service description language is used to define the ROS service types.

First section is the message type of request that is separated by --- and in the next section is the message type of response.
5.1.5 ROS computation graph level

The computation in ROS will be performed through nodes and the communication network between them, called the computation graph, shown in Figure 3.

![Diagram of ROS computational graph level](image)

*Fig. 4: Structure of the ROS Graph layer. (Joseph, L. (2015))*

To obtain more information about ROS master, nodes, topics, services, and parameter, the author recommends the reader to check Joseph, L. (2015).

5.2 ROS Distributions

A ROS distribution is a versioned set of ROS packages. These are akin to Linux distributions (e.g. Ubuntu). The purpose of the ROS distributions is to let developers work against a relatively stable codebase until they are ready to roll everything forward. Therefore, once a distribution is released, ROS try to limit changes to bug fixes and non-breaking improvements for the core packages. (Robot Operating System, 2018)

Release rules:

- There is a ROS release every year in May.
- Releases on even numbered years will be a Long-Term Support (LTS) release, supported for five years.
- Releases on odd numbered years are normal ROS releases, supported for two years.
- ROS releases will drop support for EOL Ubuntu distributions, even if the ROS release is still supported.

Side effects of the release policy:

- Every ROS release will be supported on exactly one Ubuntu LTS.
- Releases on odd numbered years will share a common Ubuntu release with the LTS ROS release of the previous year.
- LTS releases will not share a common Ubuntu release with any previous releases.
- ROS releases will not add support for new Ubuntu distributions after their release date.
These simplified rules and side effects are subject to change with changes to the underlying Ubuntu release policy.

ROS recommend *Kinetic* distribution if a major update frequency is not preferred and to work with newer Gazebo. *Kinetic* is the latest LTS and it works in one of the LTS distributions of Ubuntu (16.04 LTS). To benefit from new capabilities, developers must switch to the latest LTS every two years.

For this thesis, *Indigo* distribution working in Ubuntu 14.04 LTS was the first choice to work, since Robotiq ROS-Industrial package has its last release for mentioned distribution. Nevertheless, some issues were found running simulations. A patch to work in *Kinetic* distribution was found trying to solve those simulations problems. Followed procedure was switch to *Kinetic* distribution in Ubuntu 16.04 LTS.

If a new distribution is released, is up to the developer to switch to it. Depending on the included changes in this distribution, aforesaid task could be easy to performance. Anyway, this is not mandatory until EOL date, due to system will work in a stable version.

6 Application development

6.1 Introduction

To reach set objectives, a simple application is going to be developed. Target will be a UR5 robot working with a Robotiq 85 gripper. Function scheme for designed application is detailed in Figure 5.

![Fig. 5: Robot application functional scheme](image)
Application will consist in a *pick & place* task, and it will perform the following tasks:

- In the *init step* all the necessary resources (libraries, tools, etc.) will be included in the system. Also, ROS elements (nodes, topics, messages, etc.) will be generate. ROS state diagram is detailed in Appendix II.
- Robot will move to *home* position at the beginning, where it should be.
- In order to control the robot speed to try the application in the real system, a scaling factor (number between 0 and 1) has to be set in the system. Once the application is checked, developers could set different scaling factors depending on the step or device.
- To simulate ”next step” order usually managed by the worker, system will wait for the user pressing ENTER.
- Robot will move few centimeters above the *pick* position and gripper will be opened. Opening task is not included in Figure 5 because gripper should be already opened. However, should be included in the robot script.
- Gripper will be finely adjusted to reach a perfect parallelism with the supporting surface.
- Robot will move to the *pick* position and gripper will be closed, grasping the item.
- Robot will move again to the *pick approach* position, few centimeters above the *pick* position.
- Robot will move few centimeters above the *place* position.
- Robot will move to the *place* position and gripper will be opened, releasing the item.
- Robot will move again to the *place approach* position, few centimeters above the *place* position.
- Robot will move again to the *pick approach* position, few centimeters above the *pick* position, to follow only a path which was used previously.
- Application will come back to *home* position.
- Application will stop and will wait to be run again.

All the motion is executed after performing automatically the most suitable motion plan by MoveIt! (Section 6.7.1). If any object is added to the virtual environment (Section 6.7.2), robot will avoid them. *Home* position is defined in the Semantic Robot Description Format (SRDF), which is explained in Section 6.4.1.3. *Pick approach* position is defined by a vector with the values which the robot has to reach in its joints. Other positions are reached using incremental motion.
6.2 Installing and setting Ubuntu 16.04 LTS

To work with ROS, first needed resource is a computer running Ubuntu. This operating system can be downloaded for free from the official Ubuntu website (https://www.ubuntu.com). Several Ubuntu distributions are available in this website. Selected one will depend on chosen ROS distribution. To run ROS Kinetic, required Ubuntu distribution is Ubuntu 16.04 LTS. EOL of mentioned distribution is expected on April 2021. Once Ubuntu is installed, is recommended to update the included software to the latest version.

6.3 First step to work with ROS

6.3.1 Installing ROS

To install ROS, developers must follow the instructions provided by ROS in the official website (http://wiki.ros.org). Following Ubuntu terminal actions will install ROS and some of the most used tools:

1. Configure Ubuntu repositories to allow "restricted," "universe," and "multiverse."
2. Setup computer to accept software from packages.ros.org.

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

3. Set up the keys

```bash
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

4. Installation

```bash
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
```

5. Initialize rosdep

```bash
$ sudo rosdep init
$ rosdep update
```

6. Environment setup

```bash
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source /opt/ros/kinetic/setup.bash
```

7. Dependencies for building packages

```bash
$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```
Instructions above are a guide. Author strongly recommend to install ROS following website instructions, avoiding future issues with installation on account of possible updates.

6.3.2 Setting IDE

A comfortable way to work with ROS package content is using IDEs. ROS offers the opportunity to work with some of them, like Anaconda, Eclipse or NetBeans. For this thesis, Eclipse will be used because author has previous work experience with this IDE. It is important to note the creation of different packages will be performance through Ubuntu terminal (instructions with $ behind).

Recommended Eclipse version is Eclipse Oxygen (latest version) for C/C++ developers. Additionally, Java OpenJDK Java 8 Runtime is needed. This software can be founded in Ubuntu Software Centre. Optionally, developers can add different plugins to work in a better way with this IDE. Used plugins in this thesis had been: Darkest Dark, CodeMix Plug-in and PyDev.

To add a workspace, follow steps in next section and then File --> Import --> Existing projects into workspace, press next, select root directory of desired package. Do NOT select Copy projects into workspace. Press finish.

6.3.3 Setting workspace

To create the ROS package for this thesis, catkin workspaces will be used. With catkin developers are able to build multiple, interdependent packages together all at once. Catkin is included in the ROS installation. All the necessary files to work with UR robot will be located in this workspace.

1. Create a catkin Workspace

   $ source /opt/ros/kinetic/setup.bash
   $ mkdir -p ~/my_ur_180423_ws/src
   $ cd ~/catkin_ws/
   $ catkin_make

2. Make workspace suitable to work with Eclipse

   $ catkin_make --force-cmake -G"Eclipse CDT4 - Unix Makefiles"
   $ awk -f $(rospack find mk)/eclipse.awk build/.project >
   build/.project_with_env && mv build/.project_with_env build/.project
6.3.4 Installing ROS-I UR, Gazebo and MoveIt! libraries

To use some of the features which will be included in the packages, ROS-Industrial universal robots, Gazebo and MoveIt! ROS packages have to be installed in ROS system. Gazebo and MoveIt! should be already installed if user used full desktop installation of ROS. However, to avoid future issues and to update those tools if it is necessary, user must open terminal and write:

```
$ sudo apt-get install ros-kinetic-moveit
$ sudo apt-get install ros-kinetic-gazebo-ros
$ sudo apt-get install ros-kinetic-universal-robot
```

6.4 Creating packages

Exportable packages will contribute to provide the robot application with modularity and portability. Since libraries from other developers are going to be used, creation of one custom package is inappropriate. Complete code and structure of all the packages are provided in Appendix I.

It is important to notice that executable files to use with `rosrun` and `roslaunch` actions should have permissions to be executed. To check these permissions, open a terminal, go to file directory and execute:

```
$ ls -la
```

A file list will be shown. Executable files should have the `x` bit to be executed. In the permissions column, `-rwxrwxr-x` is required. If permissions do not contain the `x` bit, addition of mentioned bit is needed:

```
$ chmod +x [file_name]
```

Every time a package is created and modified, developers can check everything is going well consolidating the packages through:

```
$ cd [source_directory]
$ catkin_make
```

6.4.1 `myrobot` package

`myrobot` folder will contain all the necessary packages to run the robot application. Inside `myrobot` folder, four packages will describe the customized robot (a compendium of already developed packages, see Section 6.4.2 and 6.4.3), the MoveIt! and Gazebo files and the files to control and run the other packages. All the packages inside `myrobot` folder are customized packages created by the author of this thesis.
myrobot is a simple folder, located in the src folder of the source. To create this folder, open a terminal and write the following instructions:

```
$ mkdir -p [source_directory]/src/myrobot
```

To create the packages explained in Sections 6.4.1.1 to 6.4.1.4, open a terminal and write the following instructions:

```
$ cd [source_directory]/src/myrobot
$ catkin_create_pkg [package_name] [dependencies]
```

In these packages, dependencies are not needed. To add more reliability, rospy dependencies can be included.

### 6.4.1.1 myrobot_description package

Basically, myrobot_description package will contain URDF for the customized robot. URDF file format will be xacro. Xacro files are XML files with some simplifications comparing to usual URDF files. Most important information in this file is:

- Setting of parameter limited to true. This will solve some known issues during Gazebo simulation.
- Creation of world link.
- Inclusion of common stuff from universal_robots package to simulate the robot in Gazebo.
- Inclusion of UR5 description from universal_robots package.
- Inclusion of Robotiq 85 gripper from Beta Robots github repository. Attachment to UR5 tool.

This basic information will allow developers to use MoveIt! setup assistant to configure the robot (see Section 6.5).

### 6.4.1.2 myrobot_gazebo package

This package will contain YAML configuration files and launch files to run Gazebo simulation. myrobot.launch will be the file which is necessary to run Gazebo with the necessary parameters to display the robot in this environment.

The config folder will contain 3 configuration files: arm_controller_ur5.yaml, gripper_controller_robotiq.yaml and joint_state_controller.yaml.

Information provided by arm_controller_ur5.yaml and gripper_controller_robotiq.yaml will be:
- Type of controller.
- Name of different joints.
- Constraints for the arm and gripper: goal time, stopped velocity tolerance, trajectory and goal tolerances.
- Stop trajectory duration.
- Robot state publishing frequency/rate. Note that is different to joint state publishing.
- Action monitor rate.

If developer wants to switch between MoveIt! and the real robot, those configurations files should change to fit in with robot information, like joint names.

Information provided by joint_state_controller.yaml will be joint state publishing frequency/rate.

The launch folder will contain 2 configuration files: controller_utils.launch and myrobot.launch.

Fulfilled task by controller_utils.launch will be:
- Robot state publisher node creation with publish_frequency and tf_prefix parameters.
- Fake calibration node creation.
- Joint state controller parameter creation.
- Joint state spawn node creation.

Fulfilled task by myrobot.launch will be:
- Inclusion of desire world. Empty predefined world from Gazebo ROS package will be used for this simulation.
- Conversion of xacro file into a readable URDF robot description under robot_description parameter.
- Model spawn node creation.
- Arm spawn node creation.
- Gripper spawn node creation.
- Arm controller parameter creation.
- Gripper controller parameter creation.
- Call controller_utils.launch

Executable file will be myrobot.launch. Executing this file Gazebo will be launched, spawning the UR5 robotic arm with Robotiq 85 gripper properly attached. All the necessary information to run a simulation through MoveIt! (Section 6.9) will be upload to ROS system. To
execute this file, open a new terminal, set the source and write (roscore should be already initialized):

```bash
$ roslaunch myrobot_gazebo myrobot.launch
```

6.4.1.3 myrobot_moveit_config package

This package will contain YAML configuration files and launch files to run MoveIt!. Through the different launch files, developers will be able to run MoveIt! tools and to plan robot motion. Details about files are explained below. This package has been generated at most by MoveIt! setup assistant. Aforesaid process is explained in Section 6.5. Most part of the generated files are not going to be modified.

The config folder will contain 6 configuration files:

- controllers.yaml
- fake_controllers.yaml
- joint_state_limits.yaml
- kinematics.yaml
- myrobot.srdf
- ompl_planning.yaml

Information provided by controllers.yaml and fake_controllers.yaml will be:

- Name of controller.
- Action.
- Type.
- Name of the joints.

If developer wants to switch between Gazebo simulation and the real robot, those configurations files should change to fit in with robot information, like joint names.

joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed. Specific joint properties can be changed (max_position, max_velocity and max_acceleration). Joint limits can be turned off (has_velocity_limits and has_velocity_limitis).

kinematics.yaml set the kinematic solver to performance the motion plan.

SRDF files are extensions of URDF files. SRDF is a format for representing semantic information about the robot structure. Information provided by myrobot.srdf will be:
- Robot name. Groups (set of joint and links). Names and joints included.
- Chains. Names and joints included.
- Group states (predefined poses). Name and joint values.
- Information about end effector.
- Definition of necessary virtual joints to attach the robot to the real world.
- Passive joints (necessary for simulation or planning but not actuated)
- Collisions disabled. Either for being adjacent to each other or for constitute an impossible situation.

ompl_planning.yaml set the planner configuration.

The launch folder will contain 18 files:

- default_warehouse_db.launch
- demo.launch
- fake_moveit_controller_manager.launch.xml
- joystick_control.launch
- move_group.launch
- moveit_rviz.launch
- moveit.rviz
- myrobot_moveit_controller_manager.launch.xml
- myrobot_moveit_sensor_manager.launch.xml
- ompl_planning_pipeline.launch.xml
- planning_context.launch
- planning_pipeline.launch.xml
- run_benchmark_ompl.launch
- sensor_manager.launch.xml
- setup_assistant.launch
- trajectory_execution.launch.xml
- warehouse_settings.launch.xml
- warehouse.launch

Fulfilled tasks by default_warehouse_db.launch, warehouse_settings.launch.xml and warehouse.launch will be set and run a database where planning scenes (and associated planning queries) can be stored. Database can be located either in the same computer in which MoveIt! is running or in another one, being possible
communication through TCP/IP. Since warehouses is not an included feature in this project, more details are not needed.

Information provided by fake_moveit_controller_manager.launch.xml and myrobot_moveit_controller_manager.launch.xml will be:

- Set the parameter that trajectory_execution_manager needs to find the controller plugin.
- Specify the file in which the rest of the parameters are defined.

Fulfilled task by joystick_control.launch will be start all the necessary nodes and parameters to connect a joystick device to the system. Since joysticks are not an included devices in this project, more details are not needed.

moveit.rviz will configure all the adjustable parameters in RViz.

Fulfilled tasks by moveit_rviz.launch will be:

- Manage debug option, if has been activated in the main launch file.
- Manage config option, if has been activated in the main launch file.
- Start RViz calling moveit.rviz to set all the parameters.
- Set kinematic configuration calling kinematics.yaml

"Open Motion Planning Library (OMPL) consists of many state-of-the-art sampling-based motion planning algorithms. OMPL itself does not contain any code related to, e.g., collision checking or visualization." (The Open Motion Planning Library, OMPL, 2018. ompl_planning_pipeline.launch.xml has necessary information to set the plugin in case developers want to use it through ompl_planning.yaml.

planning_pipeline.launch.xml will set the desire plugin for all the planning pipelines. OMPL is the default plugin.

run_benchmark_ompl.launch will run a default benchmark included in OMPL. Since benchmark is not an included feature in this project, more details are not needed.

Fulfilled task by planning_context.launch will be:

- Overwrite the URDF it is needed.
- Set the name of the parameter under which the URDF is loaded
- Load URDF, adapting the xacro file.
- Load SRDF file.
- Load updated joint limits from joint_limits.yaml
- Load default settings for kinematics from `kinematics.yaml`

Fulfilled task by `trajectory_execution.launch` will be:

- Load/unload or switch controllers if it is needed.
- Determine the maximum error between the estimated duration of the trajectory and the duration of the actual trajectory.
- Determine the maximum joint-value tolerance for validation with the actual robot state.
- Load the robot specific controller manager and set corresponding ROS parameter.

`sensor_manager.launch.xml` and `myrobot_moveit_sensor_manager.launch.xml` will provide with all the information to run and manage different external sensor. Since external sensors are not an included devices in this project, more details are not needed.

Fulfilled tasks by `move_group.launch` will be:

- Manage debug option, if has been activated in the main launch file.
- Include planning functionalities calling `planning_pipeline.launch.xml`
- Include trajectory execution functionalities calling `trajectory_execution.launch`.
- Include sensors functionalities calling `sensor_manager.launch.xml`
- Start the actual `move_group node/action server`.
- Load non-default capabilities.
- Publish the planning scene of the physical robot to work with RViz.

`demo.launch` will run a demo to try MoveIt! and RViz after package creation. Fulfilled task by `demo.launch` will be:

- Start (or not) and locate a database.
- Enable/disable debug mode.
- Hide/show `joint_state_publisher`'s graphical user interface (GUI).
- Load URDF, SFDR and other .yaml configuration files on the param server.
- Publish a fake joint states node.
- Publish `tf` robot transform for the robot links through a robot state publisher node.
- Run RViz and load default config to see the state of the `move_group node`.

`setup_assistant.launch` will run MoveIt! setup assistant loading customized package. Is an easy way to reconfigure generated MoveIt! package if it is needed. Developers can use this tool to reconfigure the robot through a GUI.
6.4.1.4 myrobot_usage package

This package contains the python scripts which the robot is going to use and the launch files to run the simulation either in MoveIt! and RViz or MoveIt! and Gazebo. Since this package will work with python scripts, setup.py is added and the following line has to be added to CMakeLists.txt:

catkin_python_setup()

The scripts folder will contain 2 configuration files: pandp.py and get_data.py

- pandp.py which contains the customized functions and main program to perform a pick and place task. A complete explanation is provided in Section 6.6.
- get_data.py which contains the instructions to show in the screen information about the robot, its joints, groups, pose, etc. This script will not be launched unless developer wants to implement it.

The launch folder will contain 2 configuration files: moveit_pandp.launch and pandp.launch. Those files fulfill the following tasks:

- Start (or not) and locate a database.
- Enable/disable debug mode.
- Hide/show joint_state_publisher’s graphical user interface (GUI).
- Load URDF, SFDR and other .yaml configuration files on the param server.
- Publish a fake joint states node (only moveit_pandp.launch)
- Publish tf robot transform for the robot links through a robot state publisher node.
- Run RViz and load default config to see the state of the move_group node (only moveit_pandp.launch)
- Start the node which will call a specific python script.

pandp.launch will read the joint state from Gazebo simulation. Therefore, myrobot.launch from myrobot_gazebo package has to be initialized before, otherwise, an error will occur. Since simulation is going to be performance by Gazebo, RViz is not indispensable.

6.4.2 universal_robot package

ROS Industrial universal_robot package can be founded in the ROS Industrial github repository. Universal robot descriptions, drivers, gazebo resources, kinematic, ROS messages and MoveIt! packages are included. This package include UR3, UR5 and UR10, and support kinetic distribution. To work with the real robot, ur_modern_driver is recommended, instead the included driver. To install this package, developers have to execute the following instructions in terminal:
$ cd [source_directory]/src
$ git clone https://github.com/ros-industrial/universal_robot.git

Git tool has to be installed in the system to use git commands.

6.4.3 Robotiq package

ROS Industrial robotiq package can be founded in the ROS Industrial github repository. Robotiq descriptions, drivers, gazebo resources, kinematic, ROS messages and MoveIt! packages are included. This package include c-model and s-models grippers and force torque sensor. However, only support indigo distribution. Therefore, the used package will be robotiq_85_gripper provided by Stanley Innovation, which is an adapted version to kinetic distribution. To install this package, developers have to execute the following instructions in terminal:

$ cd [source_directory]/src
$ git clone https://github.com/waypointrobotics/robotiq_85_gripper.git

Git tool has to be installed in the system to use git commands.

6.5 Robot definition using MoveIt!

MoveIt! can create all the required files to performance motion tasks in the robot which has been described in the URDF. Instead of write all those files, developer can use setup assistant tool to generate the MoveIt! configuration package. Only requirement to execute this tool is the URDF/xacro file which define the robot, see Section 6.4.1.1 to know more details about xacro file in this project.

To configure the robot, open a new terminal and write (roscore has to be launched):

$ roslaunch moveit_setup_assistant

MoveIt! setup assistant GUI should appear in the screen, as it is shown in Figure 6 left. Pressing creating a new package, developer will be able to choose and load the robot description which, in this case, is locate in the urdf folder of myrobot_description package. Pressing Load Files button, rest of tabs will be active, as it is shown in Figure 6 right.
In the Self Collisions tab, certain collisions can be disabled through an automated checking. The interest in disabling certain known collisions is to save time during motion planning. Collisions can be disabled either because two objects are adjacent or because the collision cannot occur. Pressing Generate Collision Matrix and setting the sampling density parameter to 10000 random poses to check, collision matrix will be generated properly. This information will be included in the SRDF file in the built MoveIt! package. A layout example is shown in Figure 7.

In the Virtual Joints tab, joints to attach the real robot to the world are going to be defined. Since in this case there is only one robotic arm, one virtual joint is enough to do it. Pressing Add Virtual Joint Button configuration window will appear. Virtual joint name will be fixed_base, linked to simple_arm_base_link child, parent to world frame and being a fixed joint type. A layout example is shown in Figure 8.
In the Planning Groups tab, different parts of the robot are going to be defined. In this case: gripper, manipulator and complete. Complete is a compendium of the other groups to define poses for the whole robot in one action. Used configuration for those groups are:

- Kinematic solver: *kdl_kinematics_plugin/KDLKinematicsPlugin*.
- Kinematic Search Resolution: 0.005
- Kinematic Search Timeout: 0.005 sec
- Kinematic Solver Attempts: 3

Joints have to be added to the planning group as is shown in Figure 9. Some chains have to be also created. This information will be added to the SRDF file.

In the Robot Poses tab, some known poses are going to be defined. Pressing Add pose button, joint values of different groups can be modified. If developers do not know the desired poses, notice that is possible to change them in the SRDF file in the future. Knowing poses in this project are *up, home* (manipulator group), *open* and *close* (gripper group). A layout example is shown in Figure 10.
In the *End Effector* tab, the desired grippers and end effector can be set. In this case, a gripper is the only end effector. End effector name and group name will be *gripper*, linked to *simple_arm_tool_0* link and parent to *manipulator*. A layout example is shown in Figure 11.

In the *Passive Joints* tab, non-actuated joints can be specified. Joint state node will not publish this joints, saving memory and computing time. On account of the special code of the gripper package, only *simple_gripper_gripper_finger1_joint* will be actuated. Other joints will move according mentioned joint using a special script to define their motion. A layout example is shown in Figure 12.
**Author information** tab provides with some information about the maintainer of the project. This information is necessary to build the package. A null value in these fields will produce an error during the building process.

In **Configuration files** tab, developers are able to generate the MoveIt! package files, choosing the most suitable for them. A layout example is shown in Figure 13.

![Configuration files MoveIt! tab](image)

**Fig. 13: Configuration files MoveIt! tab**

### 6.6 Robot script

The robot script `pandp.py` will be written using Python and will be located in the `scripts` folder of `myrobot_usage` package. The robot script will be called from `pandp.launch`, located in the `launch` folder of `myrobot_usage` package, using a customized node called `manual_PandP_general_motion`.

Robot script will perform a pick and place task, controlling both robot and gripper. Full code is provided in Appendix I.

#### 6.6.1 Included libraries

In order to use certain ROS features, some libraries and ROS messages must be load. The call for these resources will be included at the beginning of the script:

- `sys` library.
- `copy` library.
- `rospy` library.
- `moveit_commander` library.
- `moveit_msgs.msg` message.
- `geometry_msgs.msg` message.
- `string` definition from `std_msgs.msg` message.
6.6.2 Customized functions

In order to reuse recurring piece of code, some functions will be defined:

- adjustGripperOrientation(): this function will performance and execute a motion plan which will orientate the gripper perfectly parallel to the X-Y plane.
- goPick_approach(): this function will performance and execute a motion plan which will move the robot to the approach pick spot, few centimeters above the pick position. This function will determine a concrete pose for the joints, avoiding random poses to reach the same point.
- moveIncremental(direction, num): this function will performance and execute a motion plan which will move the robot from its actual position to num units in the specified direction. This motion will be performance increasing/decreasing the x, y or z value of group_pose_values.pose.position.
- openGripper(): this function will performance and execute a motion plan which will open the gripper. This function will determine a concrete pose for the joints, searching in the SRDF a specific name in group_state.
- closeGripper(): this function will performance and execute a motion plan which will close the gripper. This function will determine a concrete pose for the joints.
- goHome(): this function will performance and execute a motion plan which will move the robot to the home position, completely horizontal. This function will determine a concrete pose for the joints, searching in the SRDF a specific name in group_state.
- obtainPoseValues(): this function will obtain the actual robot pose, showing it in the terminal.

6.6.3 main function

Main functions will performance the following tasks:

- Initialize moveit_commander and rospy.
- Assign objects for moveit_commander classes:
  o RobotCommander()
  o PlanningSceneInterface()
  o MoveGroupCommander(“manipulator”)
  o MoveGroupCommander(“gripper”)
- Create a publisher to visualize motion plans in RViz.
- Wait for RViz during 7 sec.
- Assign an object which will read a number between 0 and 1 from keyboard. This object will be a velocity scaling factor.
- Performance the pick and place motion using appropriate customized functions.
- Go home.
- Kill the process once is finish.

6.6.4 Optional customized functions

Example function to add virtual objects to the environment (a table in this case):

```python
def add_table(name):
    p = PoseStamped()
    p.header.frame_id = robot.get_planning_frame()
    p.header.stamp = rospy.Time.now()
    p.pose.position.x = 0.45
    p.pose.position.y = 0.0
    p.pose.position.z = 0.22
    q = quaternion_from_euler(0.0, 0.0, numpy.deg2rad(90.0))
    p.pose.orientation = Quaternion(*q)
    _scene.add_box(name, p, (0.45, 0.0, 0.22))
    return p.pose
```

Notice defined function will require:

- Import PoseStamped, PoseArray and Quaternion from geometry_msgs.msg
- Import quaternion_from_euler from tf.transformations
- Import numpy
- Adapt objects names

Special pick and place already developed functions in MoveIt! can be used. However, is not necessary for this project. Use of this function is strongly recommended when the system has perception.

6.7 Testing in MoveIt!

6.7.1 Testing without objects

Once all the packages has been created, developers are able to test the robot in MoveIt!. This step will save time and computing resources launching RViz instead Gazebo. The only considerations is fake joint values have to be published. To launch the application check roscore is already running, open a new terminal, set the source and write:

```bash
$ roslaunch myrobot_usage moveit_pandp.launch
```

RViz should appear automatically as it is shown in Figure 14. In the terminal developers will be able to check ROS information. The actual environment should look as it is shown below.
While terminal is waiting for the user to introduce a valid velocity scale factor, user will be able to configure and use MoveIt! in RViz. Most used features in this project had been:

- Manual move of the robot through markers.
- Plan, visualize and execute motion plans (planning tab).
- Manage virtual objects (scene objects tab)

As soon as user introduces the velocity scaling factor, MoveIt! will performance the most suitable motion plan for each movement. The followed process will be plan the motion, show all the plan information in terminal, performance a visualization in RViz and performance the motion. To have a clear visualization of the motion in RViz, developers can disable Show Robot Visual parameter in Displays - MotionPlanning - Planned Path. Figure 15 shown some of the system states during the simulation.

![Image of system states during simulation](image-url)
6.7.2 Testing with objects

In order to try MoveIt! capability to avoid virtual obstacles, it is possible to add those objects through certain functions in the script. Aforesaid objects could either exist in the real system (wall, fence, glass...) or virtual walls. It is recommended to add floor to the environment. Otherwise, robot will be “floating” in the space, being able to perform the motion piercing the floor. An example of MoveIt! working with virtual objects is shown in figure 16.

Fig. 16: Virtual objects in RViz

In this case, MoveIt! will perform the motion plan taking into account the virtual objects, avoiding collisions.

6.8 Simulation in Gazebo

Once all the packages have been created and tested in MoveIt!, developers are able to simulate the robot in Gazebo. To launch the Gazebo environment check roscore is already running, open a new terminal, set the source and write:

$ roslaunch myrobot_gazebo myrobot.launch

Gazebo should appear automatically, as it is shown in Figure 17. In the terminal developers will be able to check ROS information. The actual environment should look as it is shown below.

Fig. 17: Gazebo environment running myrobot
To launch the robot application, open a new terminal, set the source and write:

```
$ roslaunch myrobot_usage pandp.launch
```

As soon as user introduces the velocity scaling factor, MoveIt! will performance the most suitable motion plan for each movement. The followed process will be plan the motion, show all the plan information in terminal, performance the motion and show it in Gazebo. Figure 18 shown some of the system states during the simulation.

![Simulation Images](image1.png)

*Fig. 18: Example states running MoveIt! simulation in Gazebo*
7 Discussion

7.1 ROS performance

Even though programming in ROS may require more time, especially at the beginning, it seems like ROS system can perform all the tasks which can be carried out through the usual software included with the devices. Controlling the equipment through ROS entail the following advantages, compared to using software provided by manufacturer system:

- Chance to simulate the robot in an accurate virtual environment: in Section 6.7 the robot is visualized in Rviz virtual environment, with the inclusion of virtual objects, like walls or floor (Section 6.7.2). In Section 6.8, simulation in Gazebo is put into practice. This simulation tool can simulate physics in the robot and its environment. With data included, like inertial properties, real forces in the system or accurate time of execution, they can be displayed and studied. Author’s previous work with robots included performing robot applications in frameworks, provided by a certain manufacturer, which could have either a not good enough simulation tool or no simulation tool.

- Easy hardware implementation and switching: once the developer knows how ROS works, if devices libraries are available, the implementation process is short. During this thesis, author switched to Ubuntu 16.04 from 14.04, even though robot was working (with some errors) in Ubuntu 14.04. This meant rebuilding of the entire code. However, reassembling customized ROS packages second time took way less time, compared to the spent time at the beginning. In the same way, in the hardware switching task (Appendix III), robot programming, debugging and simulation performing took a couple of hours. Only few changes in the robot description (Section 6.4.1.1) and MoveIt! packages generation (6.4.1.3) were needed. It is true that the more difficult the process, the more time is needed; but will not take as long as starting from the beginning. Reusing code rate in ROS is vast. Author used to work with Stäubli robots, programming them using VAL 3 language. At the time to work with ABB robots, author had to learn RAPID language and the differences with VAL 3. ROS does not need mentioned task, saving time.

- Multitasking control and crashing procedures: due to ROS architecture: ROS developers have designed an architecture which supports multitasking control (ROS, (2018)). The communication method between different nodes make that possible. Also, if one of the nodes crash, it will affect the branches which are dependent on crashed node, providing the system with the chance of continuation of work. Author could check aforesaid feature when Gazebo nodes crashed by running one the simulations. Reader can check
the network in Appendix II. Motion plan was still working, being capable of moving the robot, for instance, in Rviz.

- Chance to choose between different programming languages (C++, Python, Java...): although Robotiq and Universal Robots ROS Industrial libraries have nodes programmed using C++, the robot script (Section 6.4.1.4) has been written using Python. Author used Python for the robot script because he found it easier, but other developers could prefer working with C++ language. This feature provides the system with some flexibility. Also, some procedures could be easier to implement using one or another. However, there was no chance to check it.

- Automated generation of the motion plans: with author’s previously programmed robot scripts, a defined motion plan had to be made. Author found it to be more productive to have the chance to use automatically generated motion plans because he could use the most efficient one. In his previous work, he had to check the time difference in the performance of different motion plans and then choose the fastest. Besides that, generation of motion plans allows developers to check and visualize the motion before executing it. Nevertheless, author found some random behavior when not enough motion constrains were defined, like addition of the floor. If no floor is added to the virtual environment, robot is floating, enabling motion which could collide with the robot support.

- Ability to avoid collisions during the motion adding virtual objects to the simulation: tested at the end but not included in this report, avoiding collisions is a really interesting feature. During the testing, robot had to reach a point in the space above a table (virtual object) to carry out a grasping task. Motion plan was different to the generated without virtual objects. This feature is useful, for instance, when more than one robot are working in the same item, situation in which robots will try to avoid each other.

There are other features which are not going to be commented on because the built-in system can already perform them.

Nevertheless, using ROS imply some disadvantages:

- Some libraries are yet not suitable to work in real processes: in order to have as many developers as possible working in ROS libraries, these are released in an early-developing stage. That means libraries could have some, either known or unknown, errors. It is indispensable to check ROS library status before decide to implement ROS to work with a particular hardware. An option to consider is to use these libraries
anyway and fix them to gain suitability. That option could need great programming and ROS knowledge. Viability of UR and Robotiq ROS libraries is commented in Section 8.

- Some features are difficult to implement: careful study about equipment which is going to be implemented is required before choosing to work with ROS. For this project, author started to implement adaptive feature of Robotiq gripper. However, that turned out difficult and, due to a lack of time, implementation of mentioned feature could not be possible.

- Difficult to learn about the system, which means way more time to start programming and the necessity to have skilled programmers: comparing to spent time in learning software which was previously used by the author like RobotStudio or Stäubli Robotics Suite, ROS needs way more time in order to understand it. Also, more debugging time is required because the framework is not design to work with a particular hardware, so random behavior or process crashing could occur.

Due to a lack of time, UR software could not be properly compared with ROS system. However, taking a look to advantages and disadvantages and knowing ROS can performance desired task, developers can choose between them. Because of mentioned lack of time, it had been impossible to work with the real robot. Even so, adaptation to a real robot is simple since developers only have to change certain configuration files.

7.2 Improvements

First thing that should be considered as future work is the improvement of certain facets of the work done in this thesis. Debugging process is necessary to make sure robot script runs correctly in every single run. Also, motion constrains have to be studied and applied; that will help to always have an appropriate motion plan.

Regarding Robotiq library, it should be improved. Since ROS Industrial Robotiq library is not available to work in kinetic ROS distribution, a user customized library is used. This library provides with all the files needed to work with the gripper but is not as reliable as ROS Industrial libraries. What is more, adaptive control can be implemented. This feature can grasp object with a fitted force, avoiding deforming the item.

7.3 Future work

University of Skövde is researching about HRC. To implement HRC in this system, 3 features could be useful: haptic control, speech recognition and vision systems. ROS has been selected, among other characteristics, because it is suitable to work with mentioned features. It is known that haptic control can be carried out through the implementation of the FT300 torque sensor
from Robotiq. Pertinent library is in ROS Industrial repository. Touching vision system, there are libraries and tools in ROS to perform it.

8 Conclusions

In this thesis, capability to work with a general robot programming system to control an industrial robot is demonstrated. ROS was used because it is the most suitable middleware according to the ingoing demands for this project suggested by University of Skövde. Aforesaid system, composed of UR5 robotic arm from Universal Robots and model 85 gripper from Robotiq, can be perfectly simulated and controlled with ROS system.

It is important to notice that the UR ROS library is actually in a developing stage. This means, that it is not recommended to use the mentioned library in a real process. In spite of this weakness, ROS strength lies in the developer’s community. Once UR ROS library is in experimental stage, it could be implemented in the real process. Another possibility is to develop this library and submit it to the ROS Industrial repository branch, where they will check it and implement it in the repository. On the other hand, Robotiq library is ready to work in industrial processes after the debugging process.
9 Bibliography


De Gier, M. R., (2015). *Control of a robotic arm: Application to on-surface 3D-printing.* [online] Delft, Netherlands. Available at: [https://repository.tudelft.nl/islandora/object/uuid%3Aa674a3fa-2534-44c4-b251-1e49a5194079](https://repository.tudelft.nl/islandora/object/uuid%3Aa674a3fa-2534-44c4-b251-1e49a5194079) [Accessed 26 Feb. 2018].


Appendix I: myrobot package code (change code, functions)

myrobot_description/urdf/myrobot_macro.xacro

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro"
   name="myrobot">

   <xacro:arg name="limited" default="true"/>
   <link name="world"/>

   <!-- common stuff -->
   <xacro:include filename="$(find ur_description)/urdf/common.gazebo.xacro"/>

   <!-- ur5 -->
   <xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro"/>
   <joint name="world_joint" type="fixed">
     <parent link="world"/>
     <child link="simple_arm_base_link"/>
     <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
   </joint>
   <xacro:ur5_robot prefix="simple_arm_" joint_limited="true"/>

   <!-- Robotiq from Beta Robots fork-->
   <xacro:include filename="$(find robotiq_85_description)/urdf/robotiq_85_gripper.urdf.xacro"/>
   <xacro:robotiq_85_gripper prefix="simple_gripper_" parent="simple_arm_tool0">
     <origin xyz="0 0 0" rpy="$\pi/2 -$\pi/2 0"/>
   </xacro:robotiq_85_gripper>

   <!-- arm -->
   <!-- <xacro:ur5_robot prefix="simple_arm_" joint_limited="false"/> -->
   <link name="world"/>

   <joint name="world_joint" type="fixed">
     <parent link="world"/>
     <child link="simple_arm_base_link"/>
     <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
   </joint>
</robot>
```
myrobot_gazebo/config/arm_controller_ur5.yaml

# arm_controller:
#   type: position_controllers/JointTrajectoryController
#   joints:
#     - shoulder_pan_joint
#     - shoulder_lift_joint
#     - elbow_joint
#     - wrist_1_joint
#     - wrist_2_joint
#     - wrist_3_joint
#   constraints:
#     - goal_time: 0.6
#       stopped_velocity_tolerance: 0.05
#       shoulder_pan_joint: {trajectory: 0.1, goal: 0.1}
#       shoulder_lift_joint: {trajectory: 0.1, goal: 0.1}
#       elbow_joint: {trajectory: 0.1, goal: 0.1}
#       wrist_1_joint: {trajectory: 0.1, goal: 0.1}
#       wrist_2_joint: {trajectory: 0.1, goal: 0.1}
#       wrist_3_joint: {trajectory: 0.1, goal: 0.1}
#     - stop_trajectory_duration: 0.5
#     - state_publish_rate: 25
#     - action_monitor_rate: 10

arm_controller:
  type: position_controllers/JointTrajectoryController
  joints:
    - simple_arm_shoulder_pan_joint
    - simple_arm_shoulder_lift_joint
    - simple_arm_elbow_joint
    - simple_arm_wrist_1_joint
    - simple_arm_wrist_2_joint
    - simple_arm_wrist_3_joint
  constraints:
    - goal_time: 0.6
      stopped_velocity_tolerance: 0.05
    - simple_arm_shoulder_pan_joint: {trajectory: 0.1, goal: 0.1}
    - simple_arm_shoulder_lift_joint: {trajectory: 0.1, goal: 0.1}
    - simple_arm_elbow_joint: {trajectory: 0.1, goal: 0.1}
    - simple_arm_wrist_1_joint: {trajectory: 0.1, goal: 0.1}
    - simple_arm_wrist_2_joint: {trajectory: 0.1, goal: 0.1}
    - simple_arm_wrist_3_joint: {trajectory: 0.1, goal: 0.1}
  stop_trajectory_duration: 0.5
  state_publish_rate: 25
  action_monitor_rate: 10

myrobot_gazebo/config/gripper_controller_robotiq.yaml

gripper:
  type: position_controllers/JointTrajectoryController
  joints:
    - simple_gripper_gripper_finger1_joint
  constraints:
    - goal_time: 0.6
      stopped_velocity_tolerance: 0.05
    - simple_gripper_gripper_finger1_joint: {trajectory: 0.1, goal: 0.1}
  stop_trajectory_duration: 0.5
  state Publish_rate: 25
  action_monitor_rate: 10

myrobot_gazebo/config/joint_state_controller.yaml

joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

myrobot_gazebo/launch/controller_utils.launch

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

<!-- Robot state publisher -->

</launch>
<node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
  <param name="publish_frequency" type="double" value="50.0" />
  <param name="tf_prefix" type="string" value="" />
</node>

<!-- Fake Calibration -->
<node pkg="rostopic" type="rostopic" name="fake_joint_calibration"
  args="/calibrated std_msgs/Bool true" />

<!-- joint_state_controller -->
<rosparam file="$(find myrobot_gazebo)/config/joint_state_controller.yaml" command="load"/>
<node name="joint_state_controller_spawner" pkg="controller_manager"
  type="controller_manager" args="spawn joint_state_controller" respawn="false"
  output="screen"/>
</launch>

myrobot_gazebo/launch/myrobot.launch

<?xml version="1.0"?>
<launch>
  <arg name="limited" default="false"/>
  <arg name="paused" default="false"/>
  <arg name="gui" default="true"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" default="worlds/empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gui)"/>
  </include>

  <!-- send robot urdf to param server -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find myrobot_description)/urdf/myrobot_macro.xacro'" />

  <!-- push robot_description to factory and spawn robot in gazebo -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
    args="-urdf -param robot_description -model myrobot -z 0.1" respawn="false" output="screen"/>

  <include file="$(find myrobot_gazebo)/launch/controller_utils.launch">
    <rosparam file="$(find myrobot_gazebo)/config/gripper_controller_robotiq.yaml" command="load"/>
    <node name="gripper_controller_spawner" pkg="controller_manager" type="spawner"
      args="gripper --shutdown-timeout 0.5" respawn="false" output="screen"/>
  </include>

  <rosparam file="$(find myrobot_gazebo)/config/arm_controller_ur5.yaml" command="load"/>
  <node name="arm_controller_spawner" pkg="controller_manager" type="controller_manager"
    args="spawn arm_controller" respawn="false" output="screen"/>
</launch>

myrobot_moveit_try2/config/controllers.yaml

controller_list:
  #---------------------for gazebo UR5-----------------------------
  - name: "arm_controller"
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - simple_arm_shoulder_pan_joint
      - simple_arm_shoulder_lift_joint
      - simple_arm_elbow_joint
      - simple_arm_wrist_1_joint
      - simple_arm_wrist_2_joint
      - simple_arm_wrist_3_joint
  #---------------------for gazebo gripper-----------------------
  - name: "gripper"
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
joints:
  - simple_gripper_gripper_finger1_joint

# #------------------for real gripper------------------#
# - name: /simple_gripper/joint_position_controller
#     action_ns: gripper_cmd #follow_joint_trajectory #Robotiq2FCommand #gripper_action
#     type: GripperCommand #FollowJointTrajectory #Robotiq2FActionController #GripperCommand
#     default: true
#     joints:
# #------------------for real UR5------------------#
# - name: vel_based_pos_traj_controller #or pos_based_pos_traj_controller
#     action_ns: follow_joint_trajectory
#     type: FollowJointTrajectory
#     joints:
#     - simple_arm_shoulder_pan_joint
#     - simple_arm_shoulder_lift_joint
#     - simple_arm_elbow_joint
#     - simple_arm_wrist_1_joint
#     - simple_arm_wrist_2_joint
#     - simple_arm_wrist_3_joint

myrobot_moveit_try2/config/fake_controllers.yaml

controller_list:
  - name: fake_gripper_controller
    joints:
      - simple_gripper_gripper_finger1_joint
  - name: fake_manipulator_controller
    joints:
      - simple_arm_shoulder_pan_joint
      - simple_arm_shoulder_lift_joint
      - simple_arm_elbow_joint
      - simple_arm_wrist_1_joint
      - simple_arm_wrist_2_joint
      - simple_arm_wrist_3_joint

myrobot_moveit_try2/config/kinematics.yaml

# gripper:
#   kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
#   kinematics_solver_search_resolution: 0.005
#   kinematics_solver_timeout: 0.005
#   kinematics_solver_attempts: 3
# manipulator:
#   kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
#   kinematics_solver_search_resolution: 0.005
#   kinematics_solver_timeout: 0.005
#   kinematics_solver_attempts: 3
myrobot_moveit_try2/config/joint_limits.yaml

# joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed
# Specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [has_velocity_limits, has_acceleration_limits]
joint_limits:
  simple_arm_elbow_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    max_acceleration: 0
  simple_arm_shoulder_lift_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    max_acceleration: 0
  simple_arm_shoulder_pan_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    max_acceleration: 0
  simple_arm_wrist_1_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    max_acceleration: 0
  simple_arm_wrist_2_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    max_acceleration: 0
  simple_arm_wrist_3_joint:
    has_velocity_limits: true
    max_velocity: 3.2
    max_acceleration: 0
  simple_gripper_gripper_finger1_finger_tip_joint:
    has_velocity_limits: true
    max_velocity: 100
    max_acceleration: 0
  simple_gripper_gripper_finger1_inner_knuckle_joint:
    has_velocity_limits: true
    max_velocity: 100
    max_acceleration: 0
  simple_gripper_gripper_finger1_joint:
    has_velocity_limits: true
    max_velocity: 2
    max_acceleration: 0
  simple_gripper_gripper_finger2_finger_tip_joint:
    has_velocity_limits: true
    max_velocity: 100
    max_acceleration: 0
  simple_gripper_gripper_finger2_inner_knuckle_joint:
    has_velocity_limits: true
    max_velocity: 100
    max_acceleration: 0
  simple_gripper_gripper_finger2_joint:
    has_velocity_limits: true
    max_velocity: 100
    max_acceleration: 0
myrobot_moveit_try2/config/ompl_planning

planner_configs:
  SBLkConfigDefault:
    type: geometric::SBL
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
  ESTkConfigDefault:
    type: geometric::EST
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0 setup()
goal_bias: 0.05  # When close to goal select goal, with this probability. default: 0.05
  LBKPIECEkConfigDefault:
    type: geometric::LBKPIECE
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    border_fraction: 0.9  # Fraction of time focused on boarder default: 0.9
    min_valid_path_fraction: 0.5  # Accept partially valid moves above fraction. default: 0.5
  BKPIECEkConfigDefault:
    type: geometric::BKPIECE
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    border_fraction: 0.9  # Fraction of time focused on boarder default: 0.9
    failed_expansion_score_factor: 0.5  # When extending motion fails, scale score by factor. default: 0.5
    min_valid_path_fraction: 0.5  # Accept partially valid moves above fraction. default: 0.5
  RRTkConfigDefault:
    type: geometric::RRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
  RRTConnectkConfigDefault:
    type: geometric::RRTConnect
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found.
default: 1
  TRRTkConfigDefault:
    type: geometric::TRRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found.
default: 1
  PRMkConfigDefault:
    type: geometric::PRM
    max_nearest_neighbors: 10  # use K nearest neighbors. default: 10
  PRMstarkConfigDefault:
    type: geometric::PRMstar
    num_samples: 1000  # number of states that the planner should sample. default: 1000
    radius_multiplier: 1.1  # multiplier used for the nearest neighbors search radius.
default: 1
  FMTkConfigDefault:
    type: geometric::FMT
    num_samples: 1000  # number of states that the planner should sample. default: 1000
    radius_multiplier: 1.1  # multiplier used for the nearest neighbors search radius.
default: 1
  k_constant: 0.0  # value used to normalize expression. default: 0.0 set in setup()
  heuristics: 0  # activate cost to go heuristics. default: 0
extended_fmt: 1  # activate the extended FMT*: adding new samples if planner does not finish successfully. default: 1
BFMTkConfigDefault:
  type: geometric::BFMT
  num_samples: 1000  # number of states that the planner should sample. default: 1000
  radius_multiplier: 1.0  # multiplier used for the nearest neighbors search radius. default: 1.0
  nearest_k: 1  # use the Knearest strategy. default: 1
  balanced: 0  # exploration strategy: balanced true expands one tree every iteration. False will select the tree with lowest maximum cost to go. default: 1
  optimality: 1  # termination strategy: optimality true finishes when the best possible path is found. Otherwise, the algorithm will finish when the first feasible path is found. default: 1
  heuristics: 1  # activates cost to go heuristics. default: 1
  cache_cc: 1  # use the collision checking cache. default: 1
  extended_fmt: 1  # Activates the extended FMT*: adding new samples if planner does not finish successfully. default: 1
PDSTkConfigDefault:
  type: geometric::PDST
STRIDEkConfigDefault:
  type: geometric::STRIDE
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
  goal_bias: 0.05  # When close to goal select goal, with this probability. default: 0.05
  use_projected_distance: 0  # whether nearest neighbors are computed based on distances in a projection of the state rather distances in the state space itself. default: 0
  degree: 16  # desired degree of a node in the Geometric Near-neighbor Access Tree (GNAT).
  max_degree: 18  # max degree of a node in the GNAT. default: 12
  min_degree: 12  # min degree of a node in the GNAT. default: 12
  max_pts_per_leaf: 6 # max points per leaf in the GNAT. default: 6
  estimated_dimension: 0.0  # estimated dimension of the free space. default: 0.0
  min_valid_path_fraction: 0.2  # Accept partially valid moves above fraction. default: 0.2
BiTRRTkConfigDefault:
  type: geometric::BiTRRT
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
  temp_change_factor: 0.1  # how much to increase or decrease temp. default: 0.1
  init_temperature: 100  # initial temperature. default: 100
  frontier_threshold: 0.0  # dist new state to nearest neighbor to disqualify as frontier. default: 0.0
  frountier_node_ratio: 0.1  # 1/10, or 1 nonfrontier for every 10 frontier. default: 0.1
  cost_threshold: 1e300  # the cost threshold. Any motion cost that is not better will not be expanded. default: inf
LBTRRTkConfigDefault:
  type: geometric::LBTRRT
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
  epsilon: 0.4  # optimality approximation factor. default: 0.4
BiESTkConfigDefault:
  type: geometric::BiEST
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
  epsilon: 0.4  # optimality approximation factor. default: 0.4
ProjESTkConfigDefault:
  type: geometric::ProjEST
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
LazyPRMkConfigDefault:
  type: geometric::LazyPRM
  range: 0.0  # Max motion added to tree. ==> maxDistance_default: 0.0, if 0.0, set on setup()
LazyPRMstarkConfigDefault:
  type: geometric::LazyPRMstar
SPARSkConfigDefault:
  type: geometric::SPARS
  stretch_factor: 3.0  # roadmap spanner stretch factor. multiplicative upper bound on path quality. It does not make sense to make this parameter more than 3. default: 3.0
  sparse_delta_fraction: 0.25  # delta fraction for connection distance. This value represents the visibility range of sparse samples. default: 0.25
  dense_delta_fraction: 0.001  # delta fraction for interface detection. default: 0.001
  max_failures: 1000  # maximum consecutive failure limit. default: 1000
SPARStwokConfigDefault:
  type: geometric::SPARStwo
  stretch_factor: 3.0  # roadmap spanner stretch factor. multiplicative upper bound on path quality. It does not make sense to make this parameter more than 3. default: 3.0
sparse_delta_fraction: 0.25  # delta fraction for connection distance. This value represents the visibility range of sparse samples. default: 0.25
dense_delta_fraction: 0.001  # delta fraction for interface detection. default: 0.001
max_failures: 5000  # maximum consecutive failure limit. default: 5000
gripper:
    planner_configs:
    - SBLkConfigDefault
    - ESTkConfigDefault
    - LBKPIECEkConfigDefault
    - BKPIECEkConfigDefault
    - KPIECEkConfigDefault
    - RRTkConfigDefault
    - RRTConnectkConfigDefault
    - RRTstarkConfigDefault
    - TRRTkConfigDefault
    - PRMkConfigDefault
    - PRMstarkConfigDefault
    - FMTkConfigDefault
    - BMkConfigDefault
    - PSTkConfigDefault
    - STRIDEkConfigDefault
    - BiTRRTkConfigDefault
    - LBTRRTkConfigDefault
    - BiESTkConfigDefault
    - ProjectESTkConfigDefault
    - LazyPRMkConfigDefault
    - LazyPRMstarkConfigDefault
    - SPARSkConfigDefault
    - SPARStwoConfigDefault
manipulator:
    planner_configs:
    - SBLkConfigDefault
    - ESTkConfigDefault
    - LBKPIECEkConfigDefault
    - BKPIECEkConfigDefault
    - KPIECEkConfigDefault
    - RRTkConfigDefault
    - RRTConnectkConfigDefault
    - RRTstarkConfigDefault
    - TRRTkConfigDefault
    - PRMkConfigDefault
    - PRMstarkConfigDefault
    - FMTkConfigDefault
    - BMkConfigDefault
    - PSTkConfigDefault
    - STRIDEkConfigDefault
    - BiTRRTkConfigDefault
    - LBTRRTkConfigDefault
    - BiESTkConfigDefault
    - ProjectESTkConfigDefault
    - LazyPRMkConfigDefault
    - LazyPRMstarkConfigDefault
    - SPARSkConfigDefault
    - SPARStwoConfigDefault
projection_evaluator: joints(simple_arm_shoulder_pan_joint, simple_arm_shoulder_lift_joint)
longest_valid_segment_fraction: 0.002
<?xml version="1.0" ?>
<!--This does not replace URDF, and is not an extension of URDF. This is a format for representing semantic information about the robot structure. A URDF file must exist for this robot as well, where the joints and the links that are referenced are defined -->
<robot name="myrobot">
<!--GROUPS: Representation of a set of joints and links. This can be useful for specifying DOF to plan for, defining arms, end effectors, etc-->
<!--LINKS: When a link is specified, the parent joint of that link (if it exists) is automatically included-->
<!--JOINTS: When a joint is specified, the child link of that joint (which will always exist) is automatically included-->
<!--CHAINS: When a chain is specified, all the links along the chain (including endpoints) are included in the group. Additionally, all the joints that are parents to included links are also included. This means that joints along the chain and the parent joint of the base link are included in the group-->
<!--SUBGROUPS: Groups can also be formed by referencing to already defined group names-->
<group name="gripper">
  <joint name="simple_gripper_gripper_base_joint" />
  <joint name="simple_gripper_gripper_finger1_inner_knuckle_joint" />
  <joint name="simple_gripper_gripper_finger1_finger_tip_joint" />
  <joint name="simple_gripper_gripper_finger1_finger_joint" />
  <joint name="simple_gripper_gripper_finger2_inner_knuckle_joint" />
  <joint name="simple_gripper_gripper_finger2_finger_tip_joint" />
  <joint name="simple_gripper_gripper_finger2_finger_joint" />
  <!--chain base_link="simple_gripper_gripper_base_link" tip_link="simple_gripper_gripper_finger1_finger_tip_link"-->
</group>
<group name="manipulator">
  <!--joint name="simple_arm_shoulder_pan_joint" -->
  <joint name="simple_arm_shoulder_lift_joint" />
  <joint name="simple_arm_elbow_joint" />
  <joint name="simple_arm_wrist_1_joint" />
  <joint name="simple_arm_wrist_2_joint" />
  <joint name="simple_arm_wrist_3_joint" />
  <joint name="simple_arm_ee_fixed_joint" />
  <joint name="simple_arm_wrist_3_link-tool0_fixed_joint" />
  <!--chain base_link="simple_arm_base_link" tip_link="simple_arm_tool0"-->
</group>
<group name="complete">
  <joint name="simple_gripper_gripper_base_joint" />
  <joint name="simple_gripper_gripper_finger1_inner_knuckle_joint" />
  <joint name="simple_gripper_gripper_finger1_finger_tip_joint" />
  <joint name="simple_gripper_gripper_finger1_finger_joint" />
  <joint name="simple_gripper_gripper_finger2_inner_knuckle_joint" />
  <joint name="simple_gripper_gripper_finger2_finger_tip_joint" />
  <joint name="simple_gripper_gripper_finger2_finger_joint" />
  <!--chain base_link="simple_gripper_gripper_base_link" tip_link="simple_gripper_gripper_finger1_finger_tip_link"-->
</group>
<group name="up" group="manipulator">
  <joint name="simple_arm_elbow_joint" value="0" />
  <joint name="simple_arm_shoulder_lift_joint" value="-1.5708" />
  <joint name="simple_arm_shoulder_pan_joint" value="0" />
  <joint name="simple_arm_wrist_1_joint" value="-1.5708" />
  <joint name="simple_arm_wrist_2_joint" value="0" />
  <joint name="simple_arm_wrist_3_joint" value="0" />
</group>
<group name="home" group="manipulator">
  <joint name="simple_arm_elbow_joint" value="0" />
</group>
</robot>
<!--GROUP STATES: Purpose: Define a named state for a particular group, in terms of joint values. This is useful to define states like 'folded arms'-->
<group_state name="up" group="manipulator"/>
<group_state name="home" group="manipulator"/>

<joint name="simple_arm_shoulder_lift_joint" value="0" />
<joint name="simple_arm_shoulder_pan_joint" value="0" />
<joint name="simple_arm_wrist_1_joint" value="0" />
<joint name="simple_arm_wrist_2_joint" value="0" />
<joint name="simple_arm_wrist_3_joint" value="0" />
</group_state>
<group_state name="open" group="gripper">
<joint name="simple_gripper_gripper_finger1_finger_tip_joint" value="0" />
<joint name="simple_gripper_gripper_finger1_inner_knuckle_joint" value="0" />
<joint name="simple_gripper_gripper_finger1_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_finger_tip_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_inner_knuckle_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_joint" value="0" />
</group_state>
<group_state name="close" group="gripper">
<joint name="simple_gripper_gripper_finger1_finger_tip_joint" value="0" />
<joint name="simple_gripper_gripper_finger1_inner_knuckle_joint" value="0.804" />
<joint name="simple_gripper_gripper_finger1_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_finger_tip_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_inner_knuckle_joint" value="0" />
<joint name="simple_gripper_gripper_finger2_joint" value="0" />
</group_state>
</end_effector>
<virtual_joint name="fixed_base" type="fixed" parent_frame="world" child_link="simple_arm_base_link" />
<passive_joint name="simple_gripper_gripper_finger1_inner_knuckle_joint" />
<passive_joint name="simple_gripper_gripper_finger1_finger_tip_joint" />
<passive_joint name="simple_gripper_gripper_finger2_inner_knuckle_joint" />
<passive_joint name="simple_gripper_gripper_finger2_finger_tip_joint" />
<disable_collisions link1="simple_arm_base_link" link2="simple_arm_shoulder_link" reason="Adjacent" />
<disable_collisions link1="simple_arm_base_link" link2="simple_arm_shoulder_pan_joint" reason="Never" />
<disable_collisions link1="simple_arm_base_link" link2="simple_arm_shoulder_lift_joint" reason="Never" />
<disable_collisions link1="simple_arm_base_link" link2="simple_arm_tool0" reason="Manipulator" />
<disable_collisions link1="simple_arm_ee_link" link2="simple_arm_wrist_3_link" reason="Never" />
<disable_collisions link1="simple_arm_ee_link" link2="simple_gripper_gripper_base_link" reason="Never" />
<disable_collisions link1="simple_arm_forearm_link" link2="simple_arm_upper_arm_link" reason="Adjacent" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_arm_wrist_2_link" reason="Adjacent" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_arm_wrist_3_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_base_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_upper_arm_link" link2="simple_arm_tool0" reason="Manipulator" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />

<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_1_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_arm_wrist_3_link" reason="Adjacent" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_base_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_2_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_base_link" reason="Adjacent" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_arm_wrist_3_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_finger_link" reason="Default" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger1_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_base_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />

<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_tip_link" link2="simple_gripper_gripper_finger1_inner_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_tip_link" link2="simple_gripper_gripper_finger2_finger_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_tip_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_tip_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger1_finger_tip_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Default" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_knuckle_link" reason="Adjacent" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_finger_tip_link" reason="Never" />
<disable_collisions link1="simple_gripper_gripper_finger2_finger_link" link2="simple_gripper_gripper_finger2_inner_knuckle_link" reason="Never" />
</robot>
myrobot_moveit_try2/launch/default_warehouse_db.launch

<launch>
  <arg name="reset" default="false" />
  <!-- If not specified, we'll use a default database location -->
  <arg name="moveit_warehouse_database_path" default="$(find myrobot_moveit_try2)/default_warehouse_mongo_db" />
  <!-- Launch the warehouse with the configured database location -->
  <include file="$(find myrobot_moveit_try2)/launch/warehouse.launch">
    <arg name="moveit_warehouse_database_path" value="$(arg moveit_warehouse_database_path)" />
  </include>
  <!-- If we want to reset the database, run this node -->
  <node if="$(arg reset)" name="$(anon moveit_default_db_reset)" type="moveit_init_demo_warehouse" pkg="moveit_ros_warehouse" respawn="false" output="screen" />
</launch>

myrobot_moveit_try2/launch/demo.launch

<launch>
  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
  <arg name="db_path" default="$(find myrobot_moveit_try2)/default_warehouse_mongo_db" />
  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />
  <!-- By default, hide joint_state_publisher's GUI
  MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of the simulated robot.
  It also provides a GUI to move the simulated robot around "manually". This corresponds to moving around the real robot without the use of MoveIt.
  -->
  <arg name="use_gui" default="false" />
  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server -->
  <include file="$(find myrobot_moveit_try2)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true" />
  </include>
  <!-- If needed, broadcast static tf for robot root -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="/use_gui" value="$(arg use_gui)" />
    <rosparam param="/source_list">[/move_group/fake_controller_joint_states]</rosparam>
  </node>
  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="true" output="screen" />
  <!-- Run the main MoveIt executable without trajectory execution (we do not have controllers configured by default) -->
  <include file="$(find myrobot_moveit_try2)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true" />
    <arg name="fake_execution" value="true" />
    <arg name="info" value="true" />
    <arg name="debug" value="$(arg debug)" />
  </include>
  <!-- Run Rviz and load the default config to see the state of the move_group node -->
  <include file="$(find myrobot_moveit_try2)/launch/moveit_rviz.launch"/>
If database loading was enabled, start mongodb as well -->
<include file="${find myrobot_moveit_try2)/launch/default_warehouse_db.launch" if="${arg db}">
  <arg name="moveit_warehouse_database_path" value="${arg db_path}"/>
</include>
</launch>

myrobot_moveit_try2/launch/joystick_control.launch

<!-- See moveit_ros/visualization/doc/joystick.rst for documentation -->
<arg name="dev" default="/dev/input/js0" />
<!-- Launch joy node -->
<node pkg="joy" type="joy_node" name="joy">
  <param name="dev" value="${arg dev}" />
  <param name="deadzone" value="0.2" />
  <param name="autorepeat_rate" value="40" />
  <param name="coalesce_interval" value="0.025" />
</node>
<!-- Launch python interface -->
<node pkg="moveit_ros_visualization" type="moveit_joy.py" output="screen" name="moveit_joy"/>
</launch>

myrobot_moveit_try2/launch-move_group.launch

<!-- GDB Debug Option -->
<arg name="debug" default="false" />
<arg unless="${arg debug}" name="launch_prefix" value="" />
<arg if="${arg debug}" name="launch_prefix" value="gdb -x ${find myrobot_moveit_try2)/launch/gdb_settings.gdb --ex run -- args" />
<!-- Verbose Mode Option -->
<arg name="info" default="${arg debug}" />
<arg unless="${arg info}" name="command_args" value="" />
<arg if="${arg info}" name="command_args" value="--debug" />
<!-- move_group settings -->
<arg name="allow_trajectory_execution" default="true" />
<arg name="fake_execution" default="false" />
<arg name="max_safe_path_cost" default="1" />
<arg name="jiggle_fraction" default="0.05" />
<arg name="publish_monitored_planning_scene" default="true" />
<!-- Planning Functionality -->
<include ns="move_group" file="${find myrobot_moveit_try2)/launch/planning_pipeline.launch.xml">
  <arg name="pipeline" value="ompl" />
</include>
<!-- Trajectory Execution Functionality -->
<include ns="move_group" file="${find myrobot_moveit_try2)/launch/trajectory_execution.launch.xml" if="${arg allow_trajectory_execution}">
  <arg name="moveit_manage_controllers" value="true" />
  <arg name="moveit_controller_manager" value="myrobot" unless="${arg fake_execution}" />
  <arg name="moveit_controller_manager" value="fake" if="${arg fake_execution}" />
</include>
<!-- Sensors Functionality -->
<include ns="move_group" file="${find myrobot_moveit_try2}/launch/sensor_manager.launch.xml" if="${arg allow_trajectory_execution}"

  <arg name="moveit_sensor_manager" value="myrobot" />
</include>

<!-- Start the actual move_group node/action server -->
<node name="move_group" launch-prefix="${arg launch_prefix}" pkg="moveit_ros_move_group" type="move_group" respawn="false" output="screen" args="${arg command_args}"

  <!-- Set the display variable, in case OpenGL code is used internally -->
  <env name="DISPLAY" value="${optenv DISPLAY :0}" />

  <param name="allow_trajectory_execution" value="${arg allow_trajectory_execution}" />
  <param name="max_safe_path_cost" value="${arg max_safe_path_cost}" />
  <param name="jiggle_fraction" value="${arg jiggle_fraction}" />

  <!-- load these non-default MoveGroup capabilities -->
  <!--
    <param name="capabilities" value="
      a_package/AwesomeMotionPlanningCapability
      another_package/GraspPlanningPipeline
    " />

  -->

  <!-- inhibit these default MoveGroup capabilities -->
  <!--
    <param name="disable_capabilities" value="
      move_group/MoveGroupKinematicsService
      move_group/ClearOctomapService
    " />

  -->

  <!-- Publish the planning scene of the physical robot so that rviz plugin can know actual robot -->
  <param name="planning_scene_monitor/publish_planning_scene" value="${arg publish_monitored_planning_scene}" />
  <param name="planning_scene_monitor/publish_geometry_updates" value="${arg publish_monitored_planning_scene}" />
  <param name="planning_scene_monitor/publish_state_updates" value="${arg publish_monitored_planning_scene}" />
  <param name="planning_scene_monitor/publish_transforms_updates" value="${arg publish_monitored_planning_scene}" />

</node>
</launch>

myrobot_moveit_try2/launch/moveit_rviz.launch

<launch>

  <arg name="debug" default="false" />
  <arg unless="${arg debug}" name="launch_prefix" value="" />
  <arg if="${arg debug}" name="launch_prefix" value="gdb --ex run --args" />

  <arg name="config" default="false" />
  <arg unless="${arg config}" name="command_args" value="" />
  <arg if="${arg config}" name="command_args" value="-d ${find myrobot_moveit_try2)/launch/moveit.rviz}" />

  <node name="${anon rviz}" launch-prefix="${arg launch_prefix}" pkg="rviz" type="rviz" respawn="false"
    args="${arg command_args}" output="screen">
    <rosparam command="load" file="${find myrobot_moveit_try2)/config/kinematics.yaml}" />
</node>

</launch>
myrobot_moveit_try2/launch/planning_context.launch

<launch>
<!-- By default we do not overwrite the URDF. Change the following to true to change the default behavior -->
<arg name="load_robot_description" default="false"/>

<!-- The name of the parameter under which the URDF is loaded -->
<arg name="robot_description" default="robot_description"/>

<!-- Load robot description format (URDF) -->
<param if="$(arg load_robot_description)" name="$(arg robot_description)" command="$(find xacro)/xacro --inorder '$(find myrobot_description)/urdf/myrobot_macro.xacro')"/>

<!-- The semantic description that corresponds to the URDF -->
<param name="$(arg robot_description)_semantic" textfile="$(find myrobot_moveit_try2)/config/myrobot.srdf" />

<!-- Load updated joint limits (override information from URDF) -->
<group ns="$(arg robot_description)_planning">
  <rosparam command="load" file="$(find myrobot_moveit_try2)/config/joint_limits.yaml"/>
</group>

<!-- Load default settings for kinematics; these settings are overridden by settings in a node's namespace -->
<group ns="$(arg robot_description)_kinematics">
  <rosparam command="load" file="$(find myrobot_moveit_try2)/config/kinematics.yaml"/>
</group>
</launch>

myrobot_moveit_try2/launch/run_benchmark_ompl.launch

<launch>
<!-- This argument must specify the list of .cfg files to process for benchmarking -->
<arg name="cfg"/>

<!-- Load URDF -->
<include file="$(find myrobot_moveit_try2)/launch/planning_context.launch">
  <arg name="load_robot_description" value="true"/>
</include>

<!-- Start the database -->
<include file="$(find myrobot_moveit_try2)/launch/warehouse.launch">
  <arg name="moveit_warehouse_database_path" value="moveit_ompl_benchmark_warehouse"/>
</include>

<!-- Start Benchmark Executable -->
<node pkg="moveit_ros_benchmarks" type="moveit_run_benchmark" name="moveit_run_benchmark" args="--config_pkg=myrobot_moveit_try2" launch-prefix="$(arg launch_prefix)" output="screen">
  <rosparam command="load" file="$(find myrobot_moveit_try2)/config/kinematics.yaml"/>
  <rosparam command="load" file="$(find myrobot_moveit_try2)/config/ompl_planning.yaml"/>
</node>
</launch>

myrobot_moveit_try2/launch/setup_assistant.launch

<!-- Re-launch the MoveIt Setup Assistant with this configuration package already loaded -->
<launch>

<!-- Debug Info -->
<arg name="debug" default="false"/>
<arg unless="$(arg debug)" name="launch_prefix" value=""/>
<arg if="$(arg debug)" name="launch_prefix" value="gdb --ex run --args"/>

<!-- Run -->
<node pkg="moveit_setup_assistant" type="moveit_setup_assistant" name="moveit_setup_assistant" args="--config_pkg=myrobot_moveit_try2" launch-prefix="$(arg launch_prefix)" output="screen"/>
</launch>
myrobot_moveit_try2/launch/warehouse.launch

<launch>
  <!-- The path to the database must be specified -->
  <arg name="moveit_warehouse_database_path" />

  <!-- Load warehouse parameters -->
  <include file="$(find myrobot_moveit_try2)/launch/warehouse_settings.launch.xml" />

  <!-- Run the DB server -->
  <node name="$(anon mongo_wrapper_ros)" cwd="ROS_HOME" type="mongo_wrapper_ros.py"
    pkg="warehouse_ros_mongo">
    <param name="overwrite" value="false"/>
    <param name="database_path" value="$(arg moveit_warehouse_database_path)" />
  </node>
</launch>

myrobot_moveit_try2/launch/moveit.rviz

Panels:
  - Class: rviz/Displays
    Help Height: 84
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /MotionPlanning1
          Splitter Ratio: 0.74256
          Tree Height: 664
      - Class: rviz/Help
        Name: Help
        Expanded:
          - /Current View1
            Name: Views
            Splitter Ratio: 0.5

Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.5
      Cell Size: 1
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.03
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
        X: 0
        Y: 0
        Z: 0
      Plane: XY
      Plane Cell Count: 10
      Reference Frame: <Fixed Frame>
      Value: true
    - Class: moveit_rviz_plugin/MotionPlanning
      Enabled: true
      MoveIt_Goal_Tolerance: 0
      MoveIt_Planning_Time: 5
      MoveIt_Use_Constraint_Aware_IK: true
      MoveIt_Warehouse_Host: 127.0.0.1
      MoveIt_Warehouse_Port: 33829
      Name: MotionPlanning
      Planned Path:
        Links:
          base_bellow_link:
            Alpha: 1
            Show Axes: false
            Show Trail: false
            Value: true
          base_footprint:
head_mount_kinect_rgb_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_prosilica_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_plate_frame:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_tilt_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_elbow_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_forearm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_l_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_l_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_motor_accelerometer_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_palm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_r_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_r_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_shoulder_lift_link:
Alpha: 1
Show Axes: false
Show Trail: false
Value: true

l_shoulder_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

l_upper_arm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

l_upper_arm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

l_wrist_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

l_wrist_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

laser_tilt_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_elbow_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_forearm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_forearm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_l_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_l_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_motor_accelerometer_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_palm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_r_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true

r_gripper_r_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
Show Trail: false
Value: true
r_shoulder_lift_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_shoulder_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_upper_arm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_upper_arm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_wrist_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_wrist_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
sensor_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
torso_lift_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
Loop Animation: false
Robot Alpha: 0.5
Show Robot Collision: false
Show Robot Visual: true
Show Trail: false
State Display Time: 0.05 s
Trajectory Topic: move_group/display_planned_path
Planning Metrics:
  Payload: 1
  Show Joint Torques: false
  Show Manipulability: false
  Show Manipulability Index: false
  Show Weight Limit: false
Planning Request:
  Colliding Link Color: 255; 0; 0
  Goal State Alpha: 1
  Goal State Color: 250; 128; 0
  Interactive Marker Size: 0
  Joint Violation Color: 255; 0; 255
  Planning Group: left_arm
  Query Goal State: true
  Query Start State: false
  Show Workspace: false
  Start State Alpha: 1
  Start State Color: 0; 255; 0
Planning Scene Topic: move_group/monitored_planning_scene
Robot Description: robot_description
Scene Geometry:
  Scene Alpha: 1
  Scene Color: 50; 230; 50
  Scene Display Time: 0.2
  Show Scene Geometry: true
  Voxel Coloring: Z-Axis
  Voxel Rendering: Occupied Voxels
Scene Robot:
Attached Body Color: 150; 50; 150

Links:
  base_bellow_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  base_footprint:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  base_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  bl_caster_l_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  bl_caster_r_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  bl_caster_rotation_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  br_caster_l_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  br_caster_r_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  br_caster_rotation_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  double_stereo_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fl_caster_l_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fl_caster_r_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fl_caster_rotation_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fr_caster_l_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fr_caster_r_wheel_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
  fr_caster_rotation_link:
    Alpha: 1
    Show Axes: false
    Show Trail: false
    Value: true
fr_caster_rotation_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_kinect_ir_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_kinect_rgb_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_mount_prosilica_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_plate_frame:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
head_tilt_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_elbow_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_forearm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_forearm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_l_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_l_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_motor_accelerometer_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_palm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_gripper_r_finger_link:
  Alpha: 1
Show Axes: false
Show Trail: false
Value: true
l_gripper_r_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_shoulder_lift_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_shoulder_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_upper_arm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_upper_arm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_wrist_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
l_wrist_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
laser_tilt_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_elbow_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_forearm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_forearm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_gripper_l_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_gripper_l_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_gripper_motor_accelerometer_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_gripper_palm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
Value: true
r_gripper_r_finger_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_gripper_r_finger_tip_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_shoulder_lift_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_shoulder_pan_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_upper_arm_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_upper_arm_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_wrist_flex_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
r_wrist_roll_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
sensor_mount_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
torso_lift_link:
  Alpha: 1
  Show Axes: false
  Show Trail: false
  Value: true
Robot Alpha: 0.5
Show Scene Robot: true
Value: true
Enabled: true
Global Options:
  Background Color: 48; 48; 48
  Fixed Frame: /world
Name: root
Tools:
  - Class: rviz/Interact
    Hide Inactive Objects: true
  - Class: rviz/MoveCamera
  - Class: rviz/Select
Value: true
Views:
  Current:
    Class: rviz/XYOrbit
    Distance: 2.0965
    Focal Point:
      X: 0.113567
      Y: 0.10592
      Z: 2.23518e-07
    Name: Current View
    Near Clip Distance: 0.01
    Pitch: 0.509797
    Target Frame: /world
myrobot_moveit_try2/launch/fake_moveit_controller_manager.launch.xml

<launch>
  <!-- Set the param that trajectory_execution_manager needs to find the controller plugin -->
  <param name="moveit_controller_manager" value="moveit_fake_controller_manager/MoveItFakeControllerManager"/>
  <!-- The rest of the params are specific to this plugin -->
  <rosparam file="$(find myrobot_moveit_try2)/config/fake_controllers.yaml"/>
</launch>

myrobot_moveit_try2/launch/myrobot(moveit_controller_manager).launch.xml

<launch>
  <rosparam file="$(find myrobot_moveit_try2)/config/controllers.yaml"/>
  <param name="use_controller_manager" value="false"/>
  <param name="trajectory_execution/execution_duration_monitoring" value="false"/>
  <param name="moveit_controller_manager" value="moveit_simple_controller_manager/MoveItSimpleControllerManager"/>
</launch>

myrobot_moveit_try2/launch/myrobot(moveit_sensor_manager).launch.xml

<launch/>

myrobot_moveit_try2/launch/ompl_planning_pipeline.launch.xml

<launch>
  <!-- OMPL Plugin for MoveIt! -->
  <arg name="planning_plugin" value="ompl_interface/OMPLPlanner"/>
  <!-- The request adapters (plugins) used when planning with OMPL. ORDER MATTERS -->
  <arg name="planning_adapters" value="default_planner_request_adapters/AddTimeParameterization default_planner_request_adapters/FixWorkspaceBounds default_planner_request_adapters/FixStartStateBounds default_planner_request_adapters/FixStartStatePathConstraints"/>
</launch>
<arg name="start_state_max_bounds_error" value="0.1" />
<param name="planning_plugin" value="$(arg planning_plugin)" />
<param name="request_adapters" value="$(arg planning_adapters)" />
<param name="start_state_max_bounds_error" value="$(arg start_state_max_bounds_error)" />
<rosparam command="load" file="$(find myrobot_moveit_try2)/config/ompl_planning.yaml"/>
</launch>

myrobot_moveit_try2/launch/planning_pipeline.launch.xml

<launch>

<!-- This file makes it easy to include different planning pipelines;
It is assumed that all planning pipelines are named XXX_planning_pipeline.launch -->

<arg name="pipeline" default="ompl" />

<include file="$(find myrobot_moveit_try2)/launch/$(arg pipeline)_planning_pipeline.launch.xml"/>
</launch>

myrobot_moveit_try2/launch/sensor_manager.launch.xml

<launch>

<!-- This file makes it easy to include the settings for sensor managers -->

<!-- Params for the octomap monitor -->
<!-- <param name="octomap_frame" type="string" value="some frame in which the robot moves" /> -->
<param name="octomap_resolution" type="double" value="0.025" />
<param name="max_range" type="double" value="5.0" />

<!-- Load the robot specific sensor manager; this sets the moveit_sensor_manager ROS parameter -->
<arg name="moveit_sensor_manager" default="myrobot" />
<include file="$(find myrobot_moveit_try2)/launch/$(arg moveit_sensor_manager)_moveit_sensor_manager.launch.xml"/>
</launch>

myrobot_moveit_try2/launch/trajectory_execution.launch.xml

<launch>

<!-- This file makes it easy to include the settings for trajectory execution -->
<!-- Flag indicating whether MoveIt! is allowed to load/unload or switch controllers -->
<arg name="moveit_manage_controllers" default="true"/>
<param name="moveit_manage_controllers" value="$(arg moveit_manage_controllers)"/>

<!-- When determining the expected duration of a trajectory, this multiplicative factor is
applied to get the allowed duration of execution -->
<param name="trajectory_execution/allowed_execution_duration_scaling" value="1.2"/>

<!-- Allow more than the expected execution time before triggering a trajectory cancel
(applied after scaling) -->
<param name="trajectory_execution/allowed_goal_duration_margin" value="0.5"/>

<!-- Allowed joint-value tolerance for validation that trajectory's first point matches
current robot state -->
<param name="trajectory_execution/allowed_start_tolerance" value="0.01"/>

<!-- Load the robot specific controller manager; this sets the moveit_controller_manager ROS parameter -->
<arg name="moveit_controller_manager" default="myrobot" />
<include file="$(find myrobot_moveit_try2)/launch/$(arg moveit_controller_manager)_moveit_controller_manager.launch.xml"/>
</launch>
myrobot_moveit_try2/launch/warehouse_settings.launch.xml

<launch>
  <!-- Set the parameters for the warehouse and run the mongodb server. -->
  <!-- The default DB port for moveit (not default MongoDB port to avoid potential conflicts) -->
  <arg name="moveit_warehouse_port" default="33829" />
  <!-- The default DB host for moveit -->
  <arg name="moveit_warehouse_host" default="localhost" />
  <!-- Set parameters for the warehouse -->
  <param name="warehouse_port" value="$(arg moveit_warehouse_port)" />
  <param name="warehouse_host" value="$(arg moveit_warehouse_host)" />
  <param name="warehouse_exec" value="mongod" />
  <param name="warehouse_plugin" value="warehouse_ros_mongo::MongoDatabaseConnection" />
</launch>

myrobot_usage/launch/pandpTry.launch

<launch>
  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
  <arg name="db_path" default="$(find myrobot_moveit_try2)/default_warehouse_mongo_db" />
  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />
  <!-- By default, hide joint_state_publisher's GUI -->
  MoveIt!’s “demo” mode replaces the real robot driver with the joint_state_publisher. The latter one maintains and publishes the current joint configuration of the simulated robot.
  It also provides a GUI to move the simulated robot around "manually". This corresponds to moving around the real robot without the use of MoveIt. -->
  <arg name="use_gui" default="false" />
  <!-- Load the URDF, SRDF and other .yaml configuration files on the param server -->
  <include file="$(find myrobot_moveit_try2)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true" />
  </include>
  <!-- Run the main MoveIt executable without trajectory execution (we do not have controllers configured by default) -->
  <include file="$(find myrobot_moveit_try2)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true" />
    <arg name="false_execution" value="false" />
    <arg name="info" value="true" />
    <arg name="debug" value="$(arg debug)" />
  </include>
  <!-- Run Rviz and load the default config to see the state of the move_group node -->
  <include file="$(find myrobot_moveit_try2)/launch/moveit_rviz.launch">
    <arg name="debug" value="$(arg debug)" />
  </include>
  <!-- If database loading was enabled, start mongodb as well -->
  <include file="$(find myrobot_moveit_try2)/launch/default_warehouse_db.launch" if="$(arg db)">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)" />
  </include>
  <!-- Starting python scripts to work with -->
  <node name="manual_PandP_general_motion" pkg="myrobot_usage" type="pandp.py" respawn="false" output="screen" />
</launch>
MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher. The latter one maintains and publishes the current joint configuration of the simulated robot. It also provides a GUI to move the simulated robot around "manually". This corresponds to moving around the real robot without the use of MoveIt.

By default, hide joint_state_publisher's GUI.

If database loading was enabled, start mongodb as well -->
<include file="$(find myrobot_moveit_try2)/launch/default_warehouse_db.launch" if="$(arg db)"/>
  <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
</include>

<include file="$(find myrobot_moveit_try2)/launch/planning_context.launch">
  <arg name="load_robot_description" value="true"/>
</include>

<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="true" output="screen"/>

<include file="$(find myrobot_moveit_try2)/launch/move_group.launch">
  <arg name="allow_trajectory_execution" value="true"/>
  <arg name="fake_execution" value="true"/>
  <arg name="info" value="true"/>
  <arg name="debug" value="$(arg debug)"/>
</include>

<include file="$(find myrobot_moveit_try2)/launch/moveit_rviz.launch">
  <arg name="config" value="true"/>
  <arg name="debug" value="$(arg debug)"/>
</include>

<node name="manual_PandP_general_motion" pkg="myrobot_usage" type="pandp.py" respawn="false" output="screen"/>

<launch>

<!-- By default, we do not start a database (it can be large) -->
<arg name="db" default="false"/>

<!-- Allow user to specify database location -->
<arg name="db_path" default="${find myrobot_moveit_try2}/default_warehouse_mongo_db"/>

<!-- By default, we are not in debug mode -->
<arg name="debug" default="false"/>

By default, hide joint_state_publisher's GUI.

By default, hide joint_state_publisher's GUI.

MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher. The latter one maintains and publishes the current joint configuration of the simulated robot. It also provides a GUI to move the simulated robot around "manually". This corresponds to moving around the real robot without the use of MoveIt.

<!-- By default, hide joint_state_publisher's GUI -->
<arg name="use_gui" default="false"/>

<!-- Load the URDF, SRDF and other .yaml configuration files on the param server -->
<include file="$(find myrobot_moveit_try2)/launch/planning_context.launch">
  <arg name="load_robot_description" value="true"/>
</include>

<!-- If needed, broadcast static tf for robot root -->
</launch>
#!/usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String

def adjustGripperOrientation():
    group_pose_values = group.get_current_pose()
    group_pose_values.pose.orientation.y = 0.7071
    group_pose_values.pose.orientation.z = 0.7071
    group.set_pose_target(group_pose_values)
    plan_adjustGripperOrientation = group.plan()
    group.execute(plan_adjustGripperOrientation, wait=True)

def goPick_approach():
    group.clear_pose_targets()
    group_variable_values = [0.86, -1.33, 1.31, 0.01, 0.86, 0.00]
    group.set_joint_value_target(group_variable_values)
    plan_goPick = group.plan()
    group.execute(plan_goPick, wait=True)

def moveIncremental(direction, num):
    group.clear_pose_targets()
    group_pose_values = group.get_current_pose()
    if direction is 'x':
        group_pose_values.pose.position.x = group_pose_values.pose.position.x + num
    elif direction is 'y':
        group_pose_values.pose.position.y = group_pose_values.pose.position.y + num
    elif direction is 'z':
        group_pose_values.pose.position.z = group_pose_values.pose.position.z + num
    group.set_pose_target(group_pose_values)
    plan_moveIncremental = group.plan()
    group.execute(plan_moveIncremental, wait=True)

def openGripper():
    gripper.clear_pose_targets()
    gripper.set_named_target("open")
    plan_open = gripper.plan()
    gripper.execute(plan_open, wait=True)

def closeGripper():
    gripper.clear_pose_targets()
    gripper_values = gripper.get_current_joint_values()
    gripper_values = [0.80392, -0.80392, 0.80392, 0.80392, -0.80392, 0.80392]
    gripper.set_joint_value_target(gripper_values)
    plan_close = gripper.plan()
    gripper.execute(plan_close)

def goHome():
    group.clear_pose_targets()
    group.set_named_target("home")
    planX = group.plan()
    group.execute(planX, wait=True)

def obtainPoseValues():
    pose = group.get_current_pose()
    print "============ Pose: ", pose

if __name__=='__main__':
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('manual_PandP_general_motion', anonymous=True)
    robot = moveit_commander.RobotCommander()
    scene = moveit_commander.PlanningSceneInterface()
    group = moveit_commander.MoveGroupCommander("manipulator")
    gripper = moveit_commander.MoveGroupCommander("gripper")
display_trajectory_publisher = rospy.Publisher('/move_group/display_planned_path', moveit_msgs.msg.DisplayTrajectory, queue_size=20)
rospy.sleep(7)
vel = float(raw_input('Max vel parameter (0 - 1): '))
group.set_max_velocity_scaling_factor(vel)

print "============ Running pick and place"
goPick_approach()
adjustGripperOrientation()
moveIncremental('z',-0.2)
openGripper()
moveIncremental('z',0.2)
moveIncremental('x',-0.6)
adjustGripperOrientation()
moveIncremental('z',-0.2)
moveIncremental('x',0.2)
moveIncremental('z',0.2)
moveIncremental('x',0.6)

print "============ Running go Home"
goHome()

moveit_commander.roscpp_shutdown()

print "============ STOPPING"

myrobot_usage/scripts/get_data.py

#!/usr/bin/env python
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg

moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('get_data', anonymous=True)
print "Script running"

group = moveit_commander.MoveGroupCommander("manipulator")
robot = moveit_commander.RobotCommander()

print "Reference frame: %s" % group.get_planning_frame()
print "End effector: %s" % group.get_end_effector_link()
print "Robot Groups:"
print robot.get_group_names()
print "Current Joint Values:"
print robot.get_current_joint_values()
print "Current Pose:"
print robot.get_current_pose()
print "Robot State:"
print robot.get_current_state()
Appendix II: ROS State. Diagrams.

Fig. 19: ROS Frames
Fig. 20: Robot description tree
Fig. 21: ROS state before run MoveIt! simulation
Fig. 22: ROS state after run Movelt simulation
Fig. 23: ROS state before run Gazebo simulation
Fig. 24: ROS state after run Gazebo simulation
12 Appendix III: Implementation using different hardware.

In order to demonstrate the capability of ROS to switch hardware in an easy way and reusing code, UR robot will be switch. Chosen robotic arm will be ABB IRB2400. The reason to choose this robot is because University of Skövde works with ABB robots for academic reasons. Other reason which motivate the implementation of this library is because is already in developmental stage; which means it is possible to implement this hardware in a real process.

The way to implement this new hardware is downloading the necessary packages and changing the robot description. Once that is done, a testing in MoveIt! and simulation in Gazebo will be performance.

In the following sections changes are going to be described. Even so, in the practice, a new package will be created to keep the previous package and to avoid running issues.

**Changes in myrobot_description package**

To work with ABB robots, ROS Industrial library have to be included in the workspace:

```
$ cd [source_directory]/src
$ git clone https://github.com/ros-industrial/abb.git
```

To program the new robot, new URDF file must be created. In this case robotic arm have to be switched in *myrobot_macro.xacro* in *myrobot_description* package. Instead including UR5 description, xacro file have to find robot description in the ABB ROS Industrial package. So locating the following lines:

```
<!-- common stuff -->
<xacro:include filename="$(find ur_description)/urdf/common.gazebo.xacro" />

<!-- ur5 -->
<xacro:include filename="$(find ur_description)/urdf/ur5.urdf.xacro" />
<xjoint name="world_joint" type="fixed">
  <parent link="world" />
  <child link = "simple_arm_base_link" />
  <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
</xjoint>
<xacro:ur5_robot prefix="simple_arm_" joint_limited="true"/>
```

Change for these lines:

```
<!-- abb_irb2400 -->
<xacro:include filename="$(find abb_irb2400_support)/urdf/irb2400_macro.xacro" />
<xjoint name="world_joint" type="fixed">
  <parent link="world" />
  <child link = "simple_arm_base_link" />
  <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
</xjoint>
<xacro:abb_irb2400 prefix="simple_arm_"/>
```
Creating `myrobot_moveit_config` package

Once the URDF file is created, developers are able to use MoveIt! setup assistant. The following figures detail the same process followed to program UR robot.

To configure the robot, open a new terminal and write (roscore has to be launched):

```bash
$ roslaunch moveit_setup_assistant
```

MoveIt! setup assistant GUI should appear in the screen. Pressing creating a new package, developer will be able to choose and load the robot description which, in this case, is locate in the `urdf` folder of `myrobot_description` package. Pressing Load Files button, rest of tabs will be active, as it is shown in Figure 25.

![Fig. 25: MoveIt! setup assistant (ABB robot)](image)

In the Self Collisions tab, certain collisions can be disable through an automated checking. The interest in disable certain known collision is to save time during motion planning. Collisions can be disable either because two objects are adjacent or because the collision cannot occur. Pressing Generate Collision Matrix and setting the sampling density parameter to 10000 random poses to check, collision matrix will be generated properly. This information will be included in the SRDF file in the built MoveIt! package. A layout example is shown in Figure 26.

![Fig. 26: Self-collision checking MoveIt! tab (ABB robot)](image)
In the Virtual Joints tab, joints to attach the real robot to the world are going to be defined. Since in this case there is only one robotic arm, one virtual joint is enough to do it. Pressing Add Virtual Joint Button configuration window will appear. Virtual joint name will be **fixed_base**, linked to **simple_arm_base_link** child, parent to **world** frame and being a **fixed** joint type. A layout example is shown in Figure 27.

![Virtual joints MoveIt! tab (ABB robot)](image)

**Fig. 27: Virtual joints MoveIt! tab (ABB robot)**

In the Planning Groups tab, different parts of the robot are going to be defined. In this case: **gripper**, **manipulator** and **complete**. Complete is a compendium of the other groups to define poses for the whole robot in one action. Used configuration for those groups are:

- Kinematic solver (gripper): *abb_irb2400_manipulator_kinematics/IKFastKinematics-Plugin*
- Kinematic solver (manipulator): *kdl_kinematics_plugin/KDLKinematicsPlugin.*
- Kinematic Search Resolution: 0.005
- Kinematic Search Timeout: 0.005 sec
- Kinematic Solver Attempts: 3

Joints have to be added to the planning group as is shown in Figure 28. Some chains have to be also created. This information will be added to the SRDF file.

![Planning groups MoveIt! tab (ABB robot)](image)

**Fig. 28: Planning groups MoveIt! tab (ABB robot)**
In the **Robot Poses** tab, some known poses are going to be defined. Pressing Add pose button, joint values of different groups can be modified. If developers do not know the desired poses, notice that it is possible to change them in the SRDF file in the future. Knowing poses in this project are **home**, **pick_appro** (manipulator group), **open** and **close** (gripper group). A layout example is shown in Figure 29.

![Fig. 29: Robot poses MoveIt! tab (ABB robot)](image)

In the **End Effector** tab, the desired grippers and end effector can be set. In this case, a gripper is the only end effector. End effector name and group name will be **gripper**, linked to **simple_arm_tool_0** link and parent to **manipulator**. A layout example is shown in Figure 30.

![Fig. 30: End effector MoveIt! tab (ABB robot)](image)

In the **Passive Joints** tab, non-actuated joints can be specified. Joint state node will not publish this joints, saving memory and computing time. On account of the special code of the gripper package, only **simple_gripper_gripper_finger1_joint** will be actuated. Other joints will move according mentioned joint using a special script to define their motion. A layout example is shown in Figure 31.
Author information tab provides with some information about the maintainer of the project. This information is necessary to build the package. A null value in these fields will produce an error during the building process.

In Configuration files tab, developers are able to generate the MoveIt! package files, choosing the most suitable for them.

Creating myrobot_usage package

The robot script will be almost the same. Developers have to adapt goPick_approach() function, changing from joint values to pose named pick_appro in the SRDF. Also, this robot will move in y direction instead z, when is looking for the place approach spot.

Launch file is the same, changing the route of the moveit_config package.

Testing in MoveIt!

Once all the packages has been created, developers are able to test the robot in MoveIt!. To launch the application check roscore is already running, open a new terminal, set the source and write:

$ rosrun [robot_package]_usage moveit_pandp.launch

RViz should appear automatically as it is shown in Figure 14. In the terminal developers will be able to check ROS information. The actual environment should look as it is shown below.

While terminal is waiting for the user to introduce a valid velocity scale factor, user will be able to configure and use MoveIt! in RViz. Most used features in this project had been:

- Manual move of the robot through markers.
- Plan, visualize and execute motion plans (planning tab).
- Manage virtual objects (scene objects tab)
As soon as user introduces the velocity scaling factor, MoveIt! will performance the most suitable motion plan for each movement. The followed process will be plan the motion, show all the plan information in terminal, performance a visualization in RViz and performance the motion. To have a clear visualization of the motion in RViz, developers can disable *Show Robot Visual* parameter in *Displays - MotionPlanning - Planned Path*. Figure 32 shown some of the system states during the simulation.

![Fig. 32: Example states running MoveIt! simulation in Rviz (ABB robot)](image)
13 Appendix IV: Containers

Linux containers can keep applications isolated from the host system in which they are running. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and wrap everything as one package. Through container is easier move code from development environments into production in a fast and replicable way.

“Unlike a virtual machine, rather than creating a whole virtual operating system, containers do not need to replicate an entire operating system, only the individual components they need in order to operate. This gives a significant performance boost and reduces the size of the application. They also operate much faster, as unlike traditional virtualization the process is essentially running natively on its host, just with an additional layer of protection around it.

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code. And importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren't available out of the box. Docker brings security to applications running in a shared environment, but containers by themselves are not an alternative to taking proper security measures.” ([https://opensource.com](https://opensource.com)).

ROS is prepared to work with Docker, Docker Network, Docker Compose, Docker Machine and Docker Swarm. "Network is a tool for defining networks to use when managing services and containers running complex applications with Docker. With Network, you define a multi-host networked application, then attach your services so that containers across your cluster running your swarm in communicate with each other. Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.” ([http://wiki.ros.org/docker](http://wiki.ros.org/docker)).

---
