TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

MUHAMMAD QASIM IMRAN 182462IVSM

DUAL-ARM MANIPULATION IN ROBOTIC WAITER USE CASE

Master's Thesis

Supervisor: Gert Kanter
PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

MUHAMMAD QASIM IMRAN 182462IVSM

KAHE ROBOTKÄE KASUTAMINE ROBOT-ETTEKANDJA NÄITEL

Magistritöö

Juhendaja: Gert Kanter
Doktorikraad

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Muhammad Qasim Imran

05.01.2021

Abstract

The aim of the thesis is to develop a dual-arm manipulation capable mobile robot that can be used as a robotic waiter. In the robotic waiter use case, dual-arm manipulation task is to lift a single object, a tray, using two manipulators simultaneously while maintaining the object's orientation. The robotic waiter then has to move to a serving table to deliver the object.

The task has been divided into four subtasks of perception, manipulation, navigation and the integration of these three tasks to make a mobile robotic waiter. Robot Operating System (ROS) framework has been used for this work. For perception, 3D point cloud has been processed using PCL (Point Cloud Library) library to compute grasps (pick points on the object). MoveIt has been used for motion planning of the manipulation subtask. The goal of simultaneous dual-arm manipulation has been achieved using trajectory mirror method. ROS navigation stack has been used for indoor navigation of the robotic waiter. For the integration task, BehaviorTree.CPP library has been used.

The work also describes the main steps required to simulate Phoebe robot in Gazebo. Testing of the developed solution has been carried out on robot in Gazebo simulator.

This thesis is written in English and is 67 pages long, including 7 chapters, 26 figures and 9 tables.

Annotatsioon

Kahe robotkäe kasutamine robot-ettekandja näitel

Töö eesmärgiks on kahe robotkäega robotile tarkvaralahenduse loomine, mis võimaldab robotil täita robot-ettekandja funktsiooni. Robot-ettekandja ülesandeks on laualt tõsta üles kandik, kasutades selleks mõlemat robotkätt. Seejärel, peab robot-ettekandja viima kandiku serveerimislauale.

Ülesanne on jagatud neljaks osaks: taju, objekti haaramine ja asetamine, navigatsioon ning integratsioon. Integratsiooni alamülesande eesmärgiks on eelnevalt mainitud kolme sammu üheks tervikuks lõimimine.

Ülesande lahendamisel on kasutatud robotite operatsioonisüsteemi ROS (ingl Robot Operating System). Taju arenduses on kasutatud 3D punktipilvede töötlusteeki PCL (ingl Point Cloud Library), et arvutada objekti haardeid. Liigutamisplaanimine teostatakse MoveIt teegi abiga. Kahe robotkäe paralleelne juhtimine on saavutatud trajektoori peegeldamisega. Siseruumides navigeerimiseks on kasutatud ROS navigatsiooniteekide toel. Eelnevalt loetletud komponentide lõimimiseks on kasutatud BehaviorTree.CPP teeki.

Töös on muuhulgas ka kirjeldatud peamised sammud Phoebe roboti simuleerimiseks Gazebo simulaatoriga. Tulemuste valideerimine on teostatud simuleeritud robotiga.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 67 leheküljel, 7 peatükki, 26 joonist, 9 tabelit.

List of abbreviations and terms

API	Application Peripheral Interface
AMCL	Adaptive Monte Carlo Localization
BT	Behavior Tree
COLLADA	COLLAborative Design Activity
DAE	Digital Asset Exchange
FPS	Frames Per Second
GLUT	OpenGL Utility Toolkit
GPD	Grasp Pose Detection
HFSM	Hierarchical Finite State Machines
ODE	Open Dynamics Engine
OpenCV	Open Source Computer Vision
OpenGL	Open Graphics Library
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
SDF	Simulation Description Format
SVM	Support Vector Machine
URDF	Unified Robot Description Format
UVG	Universal Vacuum Gripper
UGV	Unmanned Ground Vehicle

Table of contents

List of figures

List of tables

1 Introduction

In this era of digitization, robots are performing daily tasks to ease human life. The robotic manipulation is used to perform numerous tasks such as dishwashing, pick and place, welding, lifting, etc. The importance of dual-arm robotic manipulation can be easily understood when it is compared to single-arm manipulation. The dual-arm manipulation allows a robot to perform its task more efficiently, lift heavier objects, carry out more complex tasks, or handle delicate things.

A mobile humanoid robotic waiter is required to perform a variety of operations. Some of these tasks require dual-arm manipulation e.g. carrying an object on a tray from the kitchen to the customer. Dual-arm manipulation of an object is more complex than its single-arm counterpart as it requires precise synchronization among the arms' movement.

Modelling and simulation tools are frequently used in the field of robotics. These tools can be used for testing and verification of new technologies and algorithms. Since the real robots are generally expensive and are not readily available for everyone to use, especially in larger groups like students in a class. Furthermore, the testing of experimental algorithms on real robots can be costly and time-consuming. Therefore, the simulated robots are the clear choice in such scenarios.

1.1 Research Goals

The main objective of this work is to develop software for the robot to identify the object in a 3D environment, pick the object using both the arms, take the object from current location to a target location while avoiding collisions with the obstacles in the environment and then place the object at the target location.

The secondary objective is to simulate a specific robot, a customized PeopleBot named Phoebe, in Gazebo. We will try to explain each step of the process so that this work can be used as a guide to carry out a similar task on other robots.

1.2 Research Questions

This work is carried out to answer the following questions:

1. How to lift the object using two manipulators of Phoebe robot?
2. How to identify the object in a 3D environment for the robotic waiter use case?
3. How to move the Phoebe robot while carrying an object with the manipulators from one point to another in a controlled environment?
4. How to simulate the Phoebe robot in Gazebo?

1.3 Organization of the work

This work is divided into following parts:

- **Background, Related works and Tools:** Since this thesis is the integration of robotic perception, manipulation and indoor-navigation, therefore, in this chapter the current research in the respective fields will be briefly described. The chapter also provides an overview of the tools used in this work.
- **Robot Configuration for Simulation:** This chapter will give an overview of the robot's hardware and sensors; it will further explain how to configure the robot for simulation in Gazebo.
- **Implementation of Robotic Waiter:** This section will cover object detection from the 3D point cloud and dual-arm manipulation. It further explains how to detect and avoid obstacles while moving around in a controlled environment.
- **Experiments and Conclusion:** The results of different experiments conducted will be discussed and analyzed in this section.

2 Background and Related Works

2.1 Theoretical Background and Tools

2.1.1 Perception

Perception is defined as a way of conceiving something. In the robotics perspective, perception is the system that enables a robot to understand its environment or be aware of the surroundings. The system consists of the sensors that scan the environment and provide data to the robot which is processed based on the functionality of the robot e.g. object detection and recognition, obstacle detection and avoidance, scene recognition, gesture understanding. Perception is necessary for a robot to plan, make decisions and carry out actions. A basic perception pipeline is shown Figure 1.

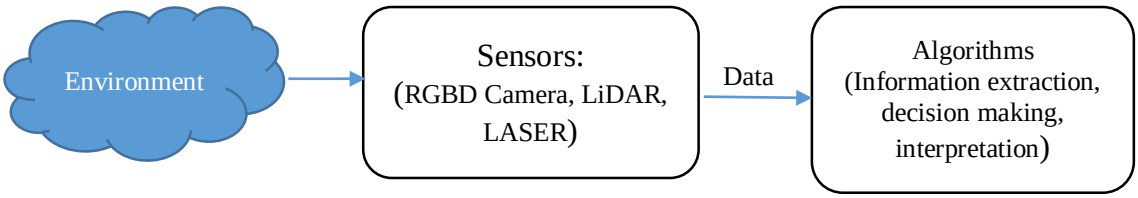


Figure 1. A basic Perception pipeline

2.1.2 Manipulation

The process of handling physical objects by a robot is called robotic manipulation e.g. Pick and Place an object, opening of the door, folding the laundry. Robotic manipulation is not a trivial computation task as it involves motion planning for robot joints while avoiding collision with the surrounding object based on forward and inverse kinematics analysis.

2.1.3 Navigation.

The ability of the robot to move to a goal position is called robotic navigation. Navigation is an essential part of a mobile robotic waiter. The objective of the navigation system is to safely move the robot from one point to another. The task of navigation is broadly

divided into three subtasks: localization, path planning from source to destination and path execution.

2.1.4 ROS

ROS stands for Robot Operating System. It is an open-source meta-operating system [1]. It provides hardware abstraction, message-passing between processes, low-level control, and the implementation of commonly used functionality.

2.1.4.1 ROS Concepts:

ROS has three levels of concepts

I. **ROS FileSystem Level**: FileSystem level is the actual files and folders that are available in the disk regarding a project. It includes of the following:

   a. **Packages:** In ROS the software is mainly organized in units called Packages. A package can contain different runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most granular thing that can be built and released.

   b. **Metapackages:** A group of related packages is represented by a metapackage.

   c. **Package Manifest:** The metadata of a package: including its name, version, description, license information, dependencies, and other meta information like exported packages, is provided by Package Manifest. It is saved in package.xml file.

   d. **Repositories:** A collection of packages which share a common VCS system. Packages which share a VCS share the same version and can be released together.

   e. **Message Types (msg):** It defines the structure of the messages used by the package for sharing data among the processes.

   f. **Services Types (srv):** It defines the structure of request/response services provided by ROS packages.

II. **ROS Computation Graph Level:** The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

a. **Nodes:** The process that performs actual computation are called nodes. Since ROS is designed to be modular therefore different functionalities of a robot are implemented in different nodes. In addition to modularity, the use of several nodes also provides various other benefits to the system. The fault tolerance of the system increases as crashes are isolated to individual nodes. Code complexity is reduced in comparison to monolithic systems. Implementation details are also well hidden as the nodes expose a minimal API to the rest of the graph and alternate implementations, even in other programming languages, can easily be substituted.

b. **Parameter Server:** The Parameter Server is a central location that is used to store data. Generally, the system configurations are stored on the parameter server. Nodes can store and retrieve data on the server at runtime.

c. **Messages:** Messages are the main mechanism of communication between the nodes. Each message has a specific type. The default basic message types provided by ROS are boolean, float, integer or string. However, a user can create custom complex message types.

d. **Topics:** Each message shared between the nodes has a specific name called Topic. The simplest semantics of message sharing between the nodes is publish/subscribe. A node that produces data, for example, a sensor or a camera, sends out a message by publishing it to a given topic. And the node which needs this data subscribes to the topic. Each node can publish on and subscribe to multiple messages. Multiple nodes can concurrently publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics.

e. **Services:** Although the publish/subscribe model is a very flexible communication paradigm, it is not appropriate for distributed systems which need request/reply type of communication. In ROS a Service provides request / reply type of communication. Each service has two types of message structures: one for the request and one for the reply. A server node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. A service is a blocking remote procedure call: the client cannot do anything else until it has received the response from the server.

f. **Bags:** Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

g. **Actions:** An action is an asynchronous request/response type of communication with the option of cancellation and moderation during execution. They require an action server and an action client. Due to their asynchronous and non-blocking nature, actions are used for remote procedure calls. This allows the client to execute other commands while its request is being executed by the server. Action server and client use three types of messages to communicate: goal, feedback and result. The client sends the 'goal' to the server. The server may send feedback to the client about the current state of the execution and it sends a result message as upon the completion of the task. The client can cancel the goal if necessary, for example, if it takes too long or the feedback is negative.

The ROS Master is responsible for the peer-to-peer network of the nodes. It provides naming and registration services to the nodes. Each node which needs to publish/subscribe a topic registers with the master. Nodes connect to other nodes directly; the Master only provides lookup information. Nodes that subscribe to a topic will request connections from nodes that publish that topic and will establish that connection over an agreed upon connection protocol. The most common protocol used in a ROS is called TCPROS, which uses standard TCP/IP sockets.

III. **ROS Community Level**: provides the resources to different communities to exchange knowledge and software. Its main parts are:

a. **ROS Distributions:** The installable collections of ROS packages that are version controlled together e.g. ROS Kinetic and the latest, as of this writing, ROS Neotic

b. The online resources like ROS wiki and ROS answer provide documentation and support to the users of the ROS packages

2.1.5 Gazebo

The design and development of complex robotics systems can be eased by the use of simulation tools. These simulators allow developing robotic applications without an actual device/robot, hence saving cost and time. However, perfect simulation of real-world environments is impossible with the current technology level, therefore the simulators use abstraction and models to provide the almost-real environment. Gazebo is a multi-robot simulator for outdoor and indoor environments, created by Nathan Koenig and Andrew Howard at the University of Southern California [2]. Gazebo is capable of simulating robots, sensors and objects in a three-dimensional world. It provides a dynamic environment that a robot can encounter. The models simulated in Gazebo have mass, friction, velocity and other simulated physical attributes.

2.1.5.1 Architecture

A major feature of Gazebo is that it enables a user to easily create new robots, sensors, arbitrary objects and actuators. All these objects are called *models*. Gazebo maintains a simple API that enables the creation of the models.

A set of models and environmental factors such as gravity and lighting is called a *world*. A model consists of at least one body and a number of joints and sensors. A model shares data with a client through interfaces. A model can have multiple interfaces.

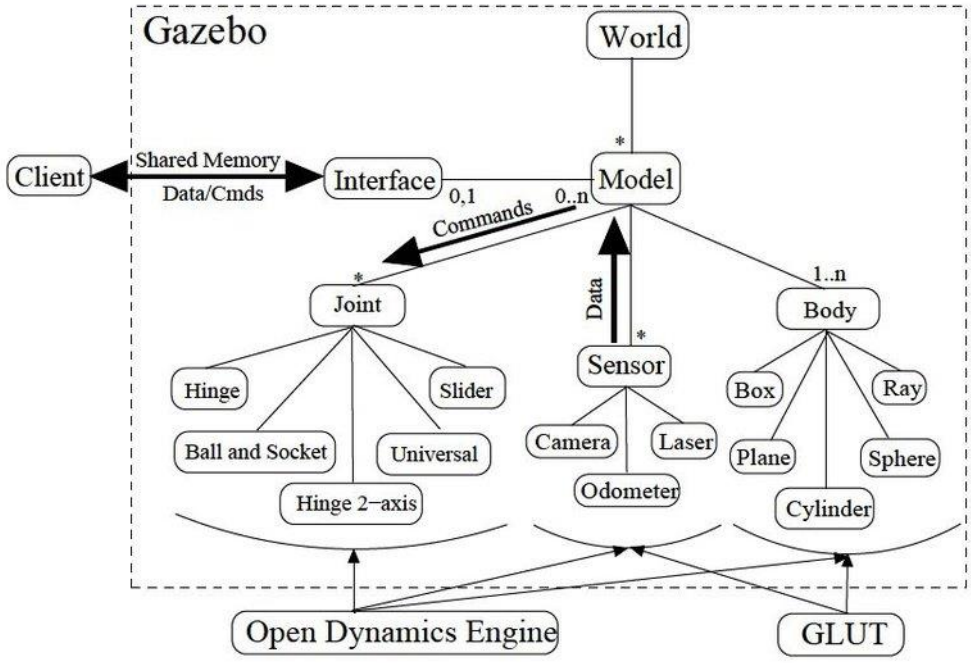


Figure 2. Architecture of Gazebo [2]

Gazebo uses ODE (Open Dynamics Engine) to simulate the dynamics and kinematics of articulated rigid bodies [2]. The physics engine is accessed through an abstraction layer provided by Gazebo.

Gazebo uses OpenGL and GLUT (OpenGL Utility Toolkit) for the user interface. Gazebo also uses the free open 3rd party library to import various well-known 3D model formats, so that the realistic reconstruction of real-world environments is possible.

Since Gazebo realistically simulates robots and environments, the robot software designed using Gazebo simulation can ideally be directly used on a physical robot. Unfortunately, this is rarely the case due to simulation inaccuracies.

2.1.6 PCL

PCL is a collection of tools and algorithms designed for processing three-dimensional data. PCL was released in 2010 under the BSD License. PCL contains a number of algorithms that can be used in common perception problems like to filter outliers from noisy data, segmentation of the scene, joining 3D point clouds, extract keypoints and compute descriptors for object recognition, etc. [3] For simplification of development, PCL has been divided into different smaller code libraries that can be compiled separately. This also allows the use of PCL on platforms with reduced computational capabilities [4].

PCL is used in this thesis because PCL is free, open-source and it is also fully integrated with ROS [3]. For example, there are conversions from ROS messages to PCL messages and vice versa. The library version used in the thesis is 1.4.1.

2.1.7 MoveIt

MoveIt is an open-source free-space motion planning framework for ROS. It is an open-source mobile manipulation software for robots developed at Willow Garage by Ioan A. Sucan and Sachin Chitta [5]. MoveIt consists of ROS integrated software packages that provide the capabilities of motion planning for mobile manipulators while avoiding collision with the surrounding objects. MoveIt can be easily configured for any robot hence making it user-friendly.

2.1.7.1 MoveIt Architecture

The following figure shows the high-level system architecture of MoveIt [6]. Move Group is implemented in a ROS node called *move_group*. This node integrates the MoveIt core functionalities like *motion planning, planning scene monitoring and collision detection* with ROS through ROS actions.

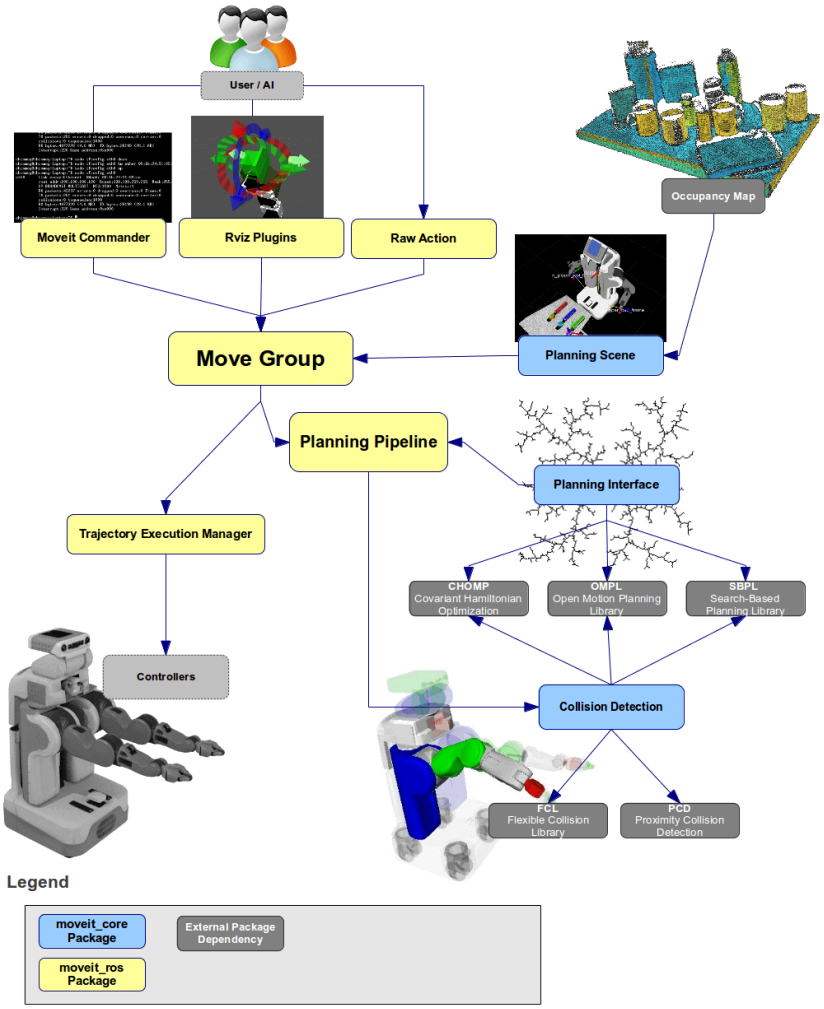


Figure 3. High-Level architecture of MoveIt [6]

2.1.7.2 MoveIt Configuration

MoveIt provides a GUI based configuration package called "*Setup Assistant*". The Setup Assistant can be used to configure MoveIt for any robot that has a robot model in URDF format. Using URDF, the Setup Assistant generated an SRDF (Semantic Robot Description Format) model of the robot. The SRDF file of the robot contains auto-generated information about the *Links* of the robot that can never be in collision with each other along with user-defined *Planning groups*. Users can select the Kinematics Solver, end-effector and various valid poses for each motion group. During the setup, the user

can also add 3D perception capability for motion planning, if the robot has an RGBD camera or other 3D point cloud sensor.

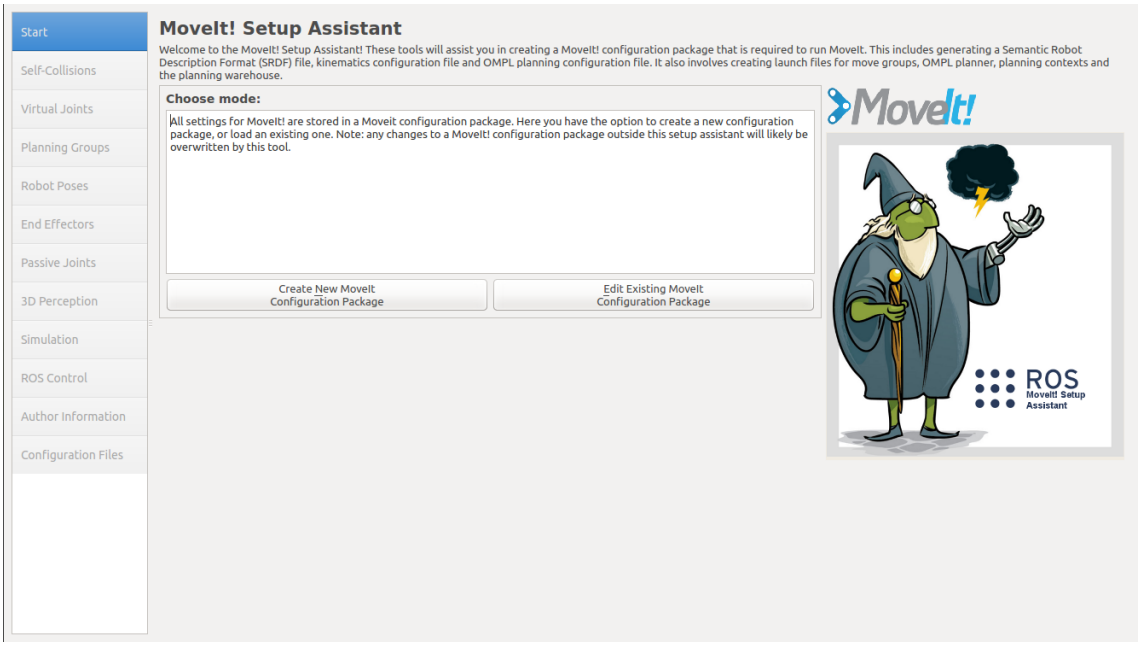


Figure 4: Picture of MoveIt Setup Assistant GUI [6]

2.1.7.3 Motion Planning and Control

MoveIt uses plugin infrastructure which provides the user with the flexibility of using different combinations of components for motion planning and kinematics solving. By default, MoveIt uses OMPL (Open Motion Planning Library) [7] for motion planning and KDL [8] kinematics solver. KDL is numerical inverse kinematics solver provided by Orocos KDL package. It currently only works for serial chains and follows joint limits specified in the URDF file of the robot. Due to the plugin style of MoveIt, the user can integrate different kinematics solvers e.g. IKFast kinematics solver, TRAC-IK [9] solver or LMA solver [6]. The OMPL has several algorithms for motion planning so the user can select any of them based on the application. By default, MoveIt uses RRT [7] (Rapidly-Exploring Random Trees) and RRT* [7] (optimized RRT) algorithms for manipulation.

MoveIt uses *controller_manager* to control the motion of the joints. It provides two options for it: fake motion controller and real motion controller. The fake motion controller is used only to visualize a trajectory planned by the Planning pipe line while the real motion controller controls the actual joint of the robot may it be simulated in *Gazebo* simulator or a real hardware robot. Each *Planning Group* needs to have a

dedicated controller; however, a single joint can have multiple controllers since it can be part of more than one planning group.

ROS communication system is used for motion execution, monitoring and control. The computed execution trajectory is sent to the controller manager using the ROS action. The control manager uses *FollowJointTrajectoryAction* action, which is part of ROS *control_msgs* package, to request the execution of a required *action_goal* from MoveIt. The current status of the trajectory execution is provided through *action_feedback* message and the result through *action_result* message.

2.1.8 BehaviorTree.CPP

BehaviorTree.CPP [10] is a C++ framework developed by Davide Faconti. It is released under MIT license. It is used to design, execute, monitor and log robotic behaviors using Behavior Trees [11]. BehviorTree.CPP provides multiple tools to help the user design, compose and debug robot's behaviors. State transitions can be recorded on file or be published in real-time to allow tools such as *Groot* to visualize them in a human-friendly way.

A Behavior Tree is a tree of hierarchical nodes that controls the flow of decision and execution of tasks/actions. Behavior Trees are an alternative to hierarchical finite state machines (HFSM).

A Behavior Tree (BT) consists of *nodes.* The nodes which don't have any child are called *leaves.* Leaves are the actual commands sent by BT to the actual system (robot). The *nodes* of the tree which are not leaves control the flow of execution.

Nodes are of four types:

- ControlNodes have 1 to N children.
- DecorationNodes have only 1 child.
- ActionNodes are the LeafNode of the tree.
- ConditionNodes are special ActionNodes since they are always atomic and synchronous. These nodes should not alter the state of the system.

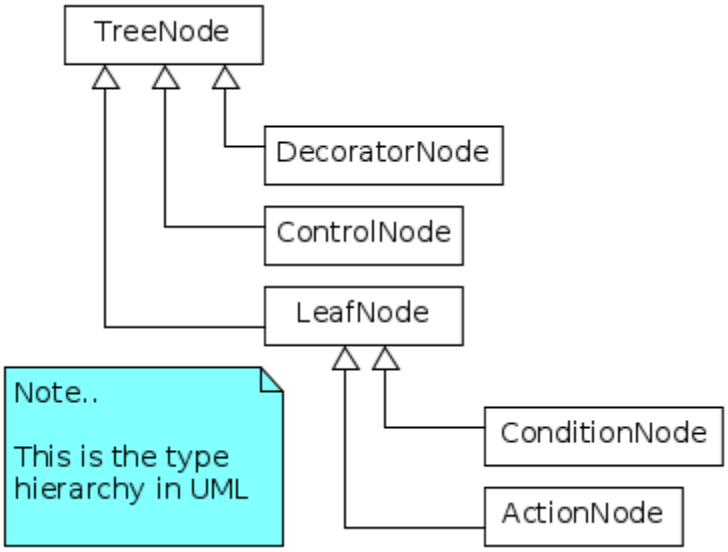


Figure 5. Types of Nodes of BT [10]

2.1.8.1 Advantages of Behavior Trees

- Hierarchical Nature of behavior tree (BT) allows the addition of subtrees to make complex behaviors. It also allows reusing previously created trees.
- The graphical representation of a BT makes it easier to understand the functionality of the system and the sequence of execution

2.2 Related Work

In this section, we will briefly mention some of the literature that has been studied for the implementation of this work. Due to the nature of the work, i.e. dual-arm manipulation, simulation of the robot and integration of different technologies, the exhaustive review of the state of the art in each field has not been carried out.

The authors in [12] have discussed the integration of perception and dual-arm manipulation for an aubergine harvesting robot. Support Vector Machine (SVM) has been used to detect and localize an aubergine in the picture, the location of the vegetable is passed to a controller which uses single or dual-arm to harvest it depending upon the situation of the scene. A novel algorithm has also been proposed to deal with the situations where the vegetable is partially hidden behind the leaves. In this case, dual-arms collaborate to harvest the vegetable. One arm removes the leaves from view and the other moves in to harvest the aubergine. Both arms can also work independently of each other to harvest two vegetables simultaneously. This work uses dual-arm manipulation, as is the objective of this thesis, however, the dual-arm collaborative manipulation is not synchronized.

The authors in [13] have developed a dual-arm mobile robot that can be used in a departmental store for stocking and disposing of items on the shelves. They have used a universal vacuum gripper (UVG). They have developed a selection algorithm that determines whether to use a single arm or both. Both arms are used when the place attitude of the item needs to be changed. Dual-arm manipulation is used to re-grasp the object whose orientation needs to be changed. The re-grasping idea of this work could have been used for this thesis but was later discarded since the orientation of the object was to be kept constant.

The work in [14] has programmed PR2 as a robotic waiter in the sense that it pushes/pulls a cart using a single arm as well as a dual-arm. They have implemented a compliance controller for co-manipulation of the cart with the human operator. They have also compared the performance of the compliance controller in the trajectory following task with the operations through joystick teleoperation. It was observed that using co-manipulation the PR2 was able to complete the task in less time while achieving a similar accuracy to that of teleoperation. This work was studied as it integrates dual-arm

manipulation and navigation however, it differs from the work of this thesis as it involves human operators.

The authors in [15] have used ROS navigation stack for the simulation of a mobile service platform. They have used an RGB-D camera, the Microsoft Kinect XBOX 360 with a turtlebot robot. model for visualization. They have used Gazebo for simulation and testing. The simulation results were visualized using RViz. Navigation based upon the RGBD camera proposed in this work has been used for this thesis in addition to LiDAR.

In [16], the authors have used ROS 2D navigation stack with minimal changes for the implementation of an autonomous mobile robot. They have used Pioneer 3-DX as a mobile base. They have carried out 2 different experiments with 2 different configurations of sensors and onboard computer. In the 1st test, they used Raspberry Pi 3 and 2D LiDAR and in the 2nd experiment, they used Intel NUC with 2D LiDAR and RGBD camera. They have concluded the robot can avoid objects in their path, or stop in case of unavoidable. Navigation stack configuration used in this thesis is similar to the work proposed in this reference.

The authors in [17] have shown the use of ROS and Gazebo for the simulation of a mobile robot. They have used the ROS navigation stack based on 2D laser finder, cameras and a SONAR. They show that the algorithms developed for the simulation of an accurate model of a robot developed in Gazebo can be directly used on the real robot. They have also discussed 3-D mapping for navigation.

The work in [18] has used ROS and Gazebo for modelling and simulation of an Unmanned Ground Vehicle (UGV). They used ROS navigation stack for the development of path planning algorithms in a known and unknown environment.

Study of the above two references has helped the author to better understand the modelling and simulation of a robot using ROS and Gazebo.

3 Robot Configuration for Simulation

3.1 Robot Modelling in ROS

In the absence of a real robot, an accurate and realistic model of a robot is required for the development and testing of algorithms/software. ROS provides different packages and tools that assist in modelling of robots such as URDF, RViz, *joint state publisher* and *robot state publisher* [1].

3.1.1 URDF

URDF stands for Unified Robot Description Format. It is an XML format that is used in ROS to represent a robot model. URDF specification only works for tree structured robots and cannot be used for parallel robots. Also it does not support the flexible elements [1]. URDF also supports Xacro. Xacro is an XML macro language. It allows repeating elements of a robot to be defined as a macro. This reduces the file size and brings modularity in the robot model. In URDF a robot is described by *link element* connected together by *joint elements*.

A link element describes the rigid body of the robot. It contains inertial, visual and collision properties of the robot. The visual properties include the position, shape, size, material, color, texture etc. of each link. Since URDF is an XML format each property has its own XML *tag*. 3D models developed in COLLADA [19] .dae and STL [20] formats can be directly used in visual tag. Therefore, the robot models developed in other CAD tools can be in ROS for better visualization. The inertial properties of a link element include mass, location of center of mass and rotational inertia matrix. The collision tag is necessary for simulation of the robot. It also contains the information about the shape of the robot but it needs not to be exactly like the actual robot but can be approximated by simpler geometric shapes to reduce processing time for simulations.

A *joint element* of URDF describes the kinematics and dynamics of each joint of the robot. Each joint has two mandatory attributes i.e. name and type, and two mandatory elements i.e. parent link name and child link name. Joint limits for revolute and prismatic type joints are also mandatory. These limits include effort, lower and upper limits of motion, and velocity. For simulation some additional properties like friction, and damping are also required.

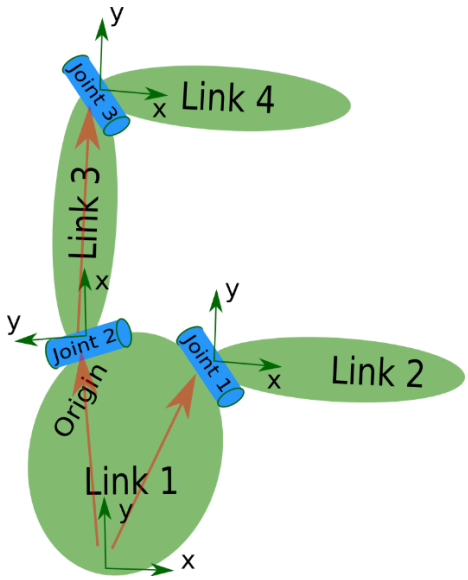


Figure 6. A generic tree structured robot model showing links and joints [21]

3.1.2 URDF configuration for simulation in Gazebo

Gazebo uses Simulation Description Format (SDF) for representing robot model, world and other objects in the simulation. SDF is also an XML format that has additional information to URDF. To use a URDF in Gazebo some additional tags must be included. To visualize the colors in Gazebo <*gazebo*> tag is required for each link element. This also coverts the .stl files to .dae file if the <*mesh*> tag has been used with .stl file. Similarly, the simulated sensors are also added to the links under <*gazebo*> tag. In addition to sensors and visuals, Gazebo requires a <*transmission*> tag for each moveable joint. The transmission tag is required to actuate the joints of the robot. This will be further explained in the controller configuration section. Gazebo uses different plugins for different type of activities and these plugins need to be added in the URDF file of the robot. For example, to simulate a robot's controller in Gazebo *gazebo_ros_control* plugin is required to be added in the URDF file.

3.2 Robotic Hardware to Simulate

For this work *Phoebe* has been used as a target mobile robot to simulate. The fully integrated *Phoebe* robot is shown in Figure 7.

Figure 7. Picture of Phoebe robot in the research laboratory

The constituent parts of Phoebe are described as follows.

3.2.1 PeopleBot

PeopleBot is a differential-drive mobile robotic platform. It is designed for service and human interface projects. Some key characteristics of the robot are presented in Table 1

S.No	Parameter	Value	Unit
1	Base Robot Weight	12 (changes with installed sensors)	Kg
2	Turn Radius	0	cm
3	Swing Radius	33	cm
4	Max. Forward/Backward Speed	0.8	m/s
5	Rotation Speed	150	deg/sec

Table 1. Phoebe Specifications [22]

3.2.1.1 Configuration for simulation

PeopleBot has a differential-drive wheel system. Two options are available to control the motion in simulation: *gazebo_ros_diff_drive* plugin for Gazebo and *differential_drive_controller* package. Both options have been explored and used in this work, however as the final choice the *differential_drive_controller* package has been selected for it can also be directly used with the actual hardware with minimum or no changes. The parameters of the controller have been configured as per the actual specifications of the robot. Further documentation of the controller and the parameters can be found at [23]. The transmission tags for each wheel have also been added in the URDF.

3.2.2 Cyton Gamma 1500

Phoebe robot has two Cyton Gamma 1500 arms attached to it. Cyton Gamma is a humanoid robot arm developed by *Robai Corporation.* It has 7 independent axes (motors) which provide it kinematic redundancy like that of a human arm.

Figure 8. Picture of Cyton Gamma 1500 robot arm [24]

Cyton Gamma 1500 mechanical specifications are given in Table 2.

S.No	Parameter	Value	Unit
1	Total weight	03	Kg
2	Payload at full reach	1200	g
3	Payload at mid reach	1500	g
4	Arm Length (base to tip)	76	cm
5	Arm Reach	68	cm
6	Gripper open range	3.5	cm
7	Max Linear arm speed	45	cm/sec

Table 2. Cyton Gamma 1500 mechanical specifications [24]

The joint specifications for Cyton Gamma 1500 are given in Table 3.

Joint	Rotation Range(◦)
Shoulder Roll	300
Shoulder Pitch	210
Shoulder Yaw	210
Elbow Pitch	210
Wrist Pitch	210
Wrist Yaw	210
Wrist Roll	300

Table 3. Cyton Gamma 1500 Joint Specifications [24]

3.2.2.1 Configuration for simulation

As mentioned in the URDF section above, each moveable joint of the robot needs to have a transmission tag in order to be controlled in Gazebo simulator. The transmission element is an extension to the robot URDF that describes the relationship between an actuator and a joint. Each transmission element has *name* attribute and multiple elements like <type>, <joint> and <actuator>. An example of a transmission for Cyton Gamma 1500 arm is given below:

```
<transmission name="right_arm_shoulder_roll_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="right_arm_shoulder_roll_joint">

<hardwareInterface>hardware_interface/PositionJointInterface</hardware
Interface>
    </joint>
    <actuator name="right_arm_shoulder_roll_motor">

<hardwareInterface>hardware_interface/PositionJointInterface</hardware
Interface>
          <mechanicalReduction>1.0</mechanicalReduction>
    </actuator>
  </transmission>
```

An Example of transmission element of URDF

3.2.3 Flea3 Camera

The Flea3 camera is mounted on Cyton Gamma 1500 manipulators so that the manipulator can be moved based on the image from the camera. The physical dimensions of the camera are 29 x 30 x 30 mm. The camera creates an image 1280 x 900 with a horizontal FOV of 25.9 degrees and a vertical FOV of 16 degrees.

Figure 9. Picture of Flea3 Camera [25]

3.2.3.1 Configuration for Simulation

Flea3 Camera is mounted on the manipulators through fixed joints therefore it doesn't require a transmission interface. For the simulation in Gazebo, *gazebo_ros_camera* plugin has been used. This plugin publishes camera info and the image captured by camera as *sensor_msgs*. The parameters of the plugin, e.g. FOV, resolution and focal length, have been configured as per the actual specifications of Flea3 camera.

3.2.4 Bumblebee2 Stereo Vision Camera

Phoebe robot has a bumblebee stereo vision camera. This camera can capture 648 x 488 video at 48 FPS.

Figure 10. Picture of Bumblebee 2 Camera [26]

3.2.4.1 Configuration for simulation

The Bumblebee stereo vision camera is mounted on PTU (Pan-Tilt-Unit) in the Phoebe robot so it is connected through two moveable joints. Two *transmission* tags have been added in the URDF for controlling of Pan joint and tilt joint of PTU. For the simulation in Gazebo, *gazebo_ros_multicamera* plugin has been used. This plugin synchronizes multiple camera's shuggers such that they publish their images together. This plugin is used for stereo cameras. It publishes the camera info and image as *sensor_msgs*. The parameters of the plugin, e.g. FOV, update rate, resolution and focal length, have been configured as per the actual specifications of Bumblebee2 stereo camera.

3.2.5 Xtion PRO LIVE

Xtion Pro Live is a 3D camera. The camera uses infrared sensors and adaptive depth detection technology. Xtion PRO LIVE provides color (RGB) image sensing therefore it can be used to capture color images. It is also capable to monitor audio in real time. Xtion Pro Live is installed above Bumblebee2 camera on Phoebe.

Figure 11. Picture of Xtion PRO LIVE [27]

3.2.5.1 Configuration for simulation

The Xtion Pro Live is attached to PTU of Phoebe robot through a fixed joint hence no transmission is required. Its orientation is controlled through the joints of PTU. For simulation in Gazebo, *gazebo_ros_openni_kinect* plugin has been configured and used. This plugin simulates Microsoft Kinect, ASUS Xtion Pro and Xtion Pro Live. The plugin publishes depth, RGB, and IR image streams.

The depth and image streams provided by the plugin were rotated around x and z-axis therefore a virtual link "*xtion_optical_fram*" has been added in the URDF to transform the streams back to accurate rotations.

3.2.6 Hokuyo LiDAR Scanner

Phoebe robot has a LiDAR range finder. This sensor has also been simulated in Gazebo. Specifications of the sensor are given in Table 4

S.No	Parameter	Value	Unit
1	Total weight	130	g
2	Dimensions	50x50x70	mm
3	Light Source	Semiconductor Laser λ=905nm	nm
4	Detection Range	30	m
5	Accuracy	±40	mm
6	Scan Angle	270	deg
7	Scan speed	25	ms
8	Angular Resolution	0.25	deg

Table 4. Hokuyo Specifications [28]

Figure 12. Picture of Hokuyo LiDAR [28]

3.2.6.1 Configuration for simulation

In Phoebe robot the Hokuyo LiDAR is connected to the body of PeopleBot through a fixed joint. For simulation in Gazebo, *gazebo_ros_laser* plugin has been configured and used.

3.3 ROS Control and controller configuration

In order to move each joint in Gazebo, a ROS controller is required. This controller needs to be compatible with the hardware interface type of each joint. This interface type is mentioned in the <transmission> tag of each joint. A ROS controller takes the current state and the required state of the joint as input and calculates the output using a generic loop feedback mechanism like a PID controller. The output depends upon the hardware_interface type: effort (force/torque), position, velocity.

3.3.1 ros_control framework

The ros_control framework is a set of ROS packages that include controller interfaces, controller managers, transmissions and hardware_interfaces. It provides the capability to implement and manage robot controllers that can be shared within the robotic community. The main feature of the framework is the Hardware Abstraction Layer, which serves as a bridge to different simulated and real robots [29]. It also allows for integrating heterogeneous hardware components transparently. The *hardware_interface* provides this abstraction. The controller_manager is responsible for starting, stopping, loading and unloading of the controllers. The framework also provides several ready-made controllers for manipulation and mobile robots. The *joint_trajectory_controller* provided by the framework is extensively used by position-controlled robots to interface with MoveIt!. The Gazebo simulator communicates with ROS through ros_control framework.

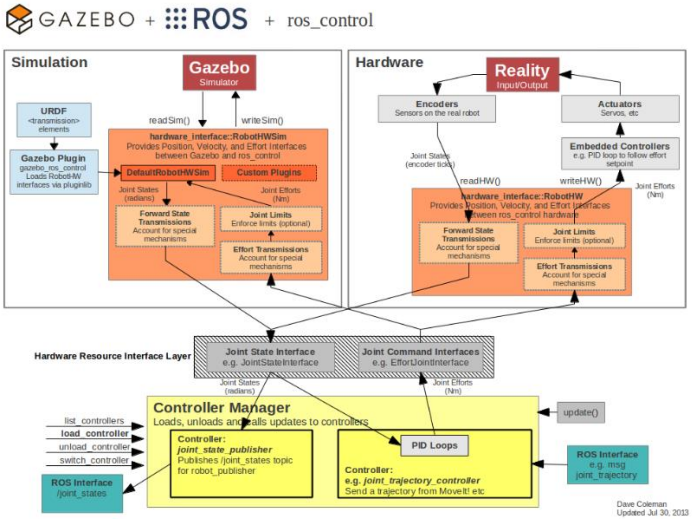


Figure 13. Overview of relationship between Gazebo, ROS and ros_control [30]

3.3.1.1 Controllers in ros_control

The ros_controllers package of the ros_control framework provides following built-in ready to use controllers:

- effort_controllers: Used to control the joint through effort (force)
- position_controllers: Control the joint position
- velocity_controllers: control the velocity of the joint
- joint_trajectory_controllers: It provides the control for executing joint-space trajectories on a group of joints

3.3.2 Controller configuration for manipulation using MoveIt

As mentioned above, MoveIt uses *joint_trajectory_controller* for controlling the position of group of joints therefore, the *position_controller/joint_trajectory_controller* has been used for this work. The following steps summarize the process of configuration of controller:

- In the <transmission> tag of the URDF file use following hardware interface for all the moveable joints of the robotic arm to be controlled through MoveIt

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

- To save the controller's settings create a .yaml file. This file is generally saved in *config* folder of the package. These settings are loaded to the parameter server through *roslaunch.* Each manipulation group needs to have a separate controller. Following is the example of *Phoebe's* Right arm controller:

```
right_arm_controller: #Name of the controller
  type: "position_controllers/JointTrajectoryController"
  joints: #Joints of the right_arm
    - right_arm_shoulder_roll_joint
    - right_arm_shoulder_pitch_joint
    - right_arm_shoulder_yaw_joint
    - right_arm_elbow_pitch_joint
    - right_arm_elbow_yaw_joint
    - right_arm_wrist_pitch_joint
    - right_arm_wrist_roll_joint
```

Example of Joint Trajectory Controller

- Create a launch file to load the configuration file created in the previous step to the parameter server. Run the *spawner* node from *controller_manager* package and pass all the controllers of the configuration file as arguments. Following code listing shows an example:

```
<node name="controller_spawner" pkg="controller_manager"
type="spawner" respawn="false" output="screen"
args="  joint_state_controller
        left_arm_controller
        right_arm_controller
        left_gripper_controller
        right_gripper_controller
        ptu_controller
        wheels_controller
                              "/>
```

Example of loading Controllers using controller_spawner node

- Now create *controllers.yaml* file in the *robot_moveit_config/config* folder. This file contains the controller configuration of the move groups to be controlled through MoveIt. Following code listing shows an example from *phoebe_moveit_config* package of this work:

```
controller_list:


  - name: left_arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - left_arm_shoulder_roll_joint
      - left_arm_shoulder_pitch_joint
      - left_arm_shoulder_yaw_joint
      - left_arm_elbow_pitch_joint
      - left_arm_elbow_yaw_joint
      - left_arm_wrist_pitch_joint
      - left_arm_wrist_roll_joint


  - name: right_arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    joints:
      - right_arm_shoulder_roll_joint
      - right_arm_shoulder_pitch_joint
      - right_arm_shoulder_yaw_joint
      - right_arm_elbow_pitch_joint
      - right_arm_elbow_yaw_joint
      - right_arm_wrist_pitch_joint
      - right_arm_wrist_roll_joint
```

Example of controller list for MoveIt

- In *phoebe_moveit_config/launch* folder create a *phoebe_moveit_controller_manager.launch.xml* file and load the *controllers.yaml* to the parameter server. Following is the example:

```
<launch>

  <!-- loads moveit_controller_manager on the parameter server
which is taken as argument if no argument is passed,
moveit_simple_controller_manager will be set -->
  <arg name="moveit_controller_manager"
default="moveit_simple_controller_manager/MoveItSimpleController
Manager" />
  <param name="moveit_controller_manager" value="$(arg
moveit_controller_manager)"/>

  <!-- loads ros_controllers to the param server -->
  <rosparam file="$(find
phoebe_moveit_config)/config/controllers.yaml"/>
</launch>
```

Example of Launch file for loading controllers for MoveIt

4 Implementation of Robotic waiter

This chapter describes different approaches explored for the execution of the dual-arm manipulation task using MoveIt and perception pipeline. Configuration of navigation stack has been elaborated along with the integration of perception manipulation and navigation pipeline for the robotic waiter use case.

4.1 Explored methods for single arm Pick and Place

There are different ways to pick an object using a robotic manipulator. The simplest is to use a predefined set of joint values of the robotic arm, calculate the forward kinematics based on these values and then place the object on that location. This approach is not useful for any practical purposes, especially for an autonomous robot. The opposite and more practical approach is to get the position of the object and use inverse kinematics to calculate the joint values to reach that position. Given the end position and orientation, also known as pose, of the end-effector of the robotic manipulator the inverse kinematics(IK) solvers can give different values. So a valid combination of these values has to be selected to perform a collision free pick and place.

MoveIt provides a pick and place pipeline for single manipulators. This pipeline uses *moveit_msgs::Grasp* message for defining various poses and postures involved in the grasping operation. The most important and complex among these poses include the *grasp_posture*. The *grasp_posture* is the position and orientation of the manipulator's end-effector.

Author decided to use MoveIt pick and place pipeline for this work. MoveIt pick and place pipeline uses the *Grasp* message which uses *grasp_pose* as an input. There are different available open source ROS packages that provide the *Grasp Pose.* The following sections briefly describe which packages have been explored during the implementation of this work and the results obtained during the experiments.

4.1.1 Grasp pose detection using *Simple Grasping* package

Simple Grasping [31] is an open source package developed by Michael Ferguson. This package generates end effector pose for grasping. It takes the point cloud data, and the gripper's geometry as input and generates a pose for grasping the detected objects. This

package has a *basic_grasping_perception* node that processes the point cloud data, detects the distinct objects in the point cloud and then generates grasp poses for the end effector. These end effector poses can then be passed to the MoveIt pick and place pipeline.

4.1.1.1 Experiments with Simple Grasping and results

For using the package, we created a configuration file for our gripper and remapped our point cloud topic to *basic_grasping_perception* node's input topic. This node provides an action server for detecting "Graspable Objects" *FindGraspableObjectsAction*. During the experiments, it was observed that this action server couldn't properly classify our tray as a graspable object. The next test was carried out using PR2 playground world that was used in the tutorial of the package. The package succeeded in identifying the objects but could not produce any grasp poses for our gripper. The reason assumed was that all the items were large in size to fit in the gripper. In order to verify this assumption another test was carried out. In this test a thin rod was used as an object to be grasped, this rod was detected but no pose was generated for it as well. After these unsuccessful tests, it was decided to not use this package any further and search for an alternate solution.

4.1.2 Grasp Pose Detection(GPD) package

The GPD [32] is an open source package to detect 6-DOF grasp poses for parallel jaw grippers in 3D point cloud. This package can be used for new objects without needing their CAD. It has on average 93% end-to-end grasp success rate for novel objects in dense clutter. GPD operates in two main steps: first, a large number of grasp candidates are generated from the point cloud and then each grasp is classified as a viable grasp or not. This package uses CNN for the generation and classification of the grasps. GPD provides a ROS wrapper, gpd_ros, that can be used directly in ROS.

4.1.2.1 Experiments with gpd_ros

Since this package utilizes CNN, the developers have provided the configuration weights for the model. These weight configuration files need to be included in the user's package. The gripper configuration file has to be updated according to the used gripper. The package also has two configurable parameters, *workspace* and *num_samples,* that can be used to improve the number of grasps found [33]. The smaller *workspace* and larger *num_samples* improves the number of grasps detected. The package also allows 3D point cloud processing, like filtration and voxel grid, through rosparam configuration.

Following experiments were conducted using this package:

- In the first test unfiltered point cloud was used, containing both the table and tray. In this test the grasps were generated for both the tray and table.

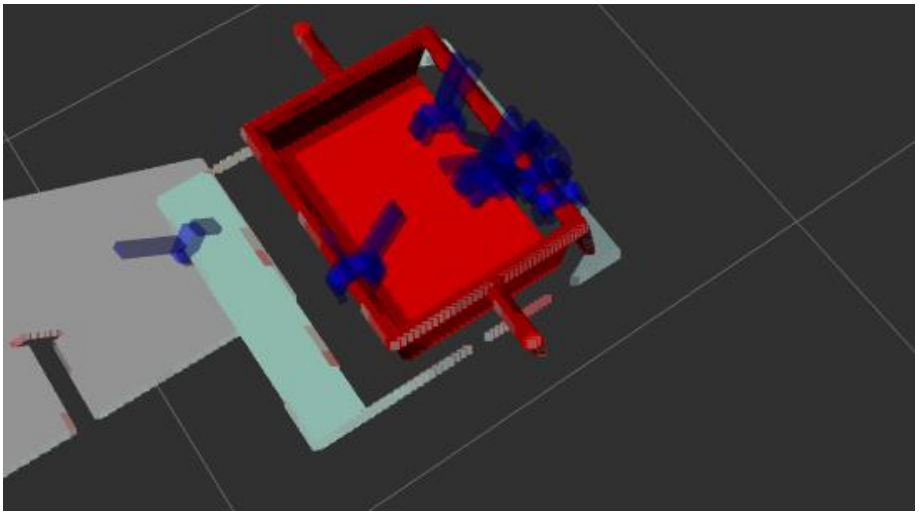


Figure 14: GPD output for unfiltered point cloud

- In the 2nd test table was filtered out using a pass-through filter of PCL and provided only tray to the package. In this test, the grasp for all sides of the tray were generated. The most suitable and desired grasp location for a tray is its handles, however, the package generated minimum grasps for handles and majority of the generated grasp were top down which is not suitable orientation to lift using parallel jaw gripper.

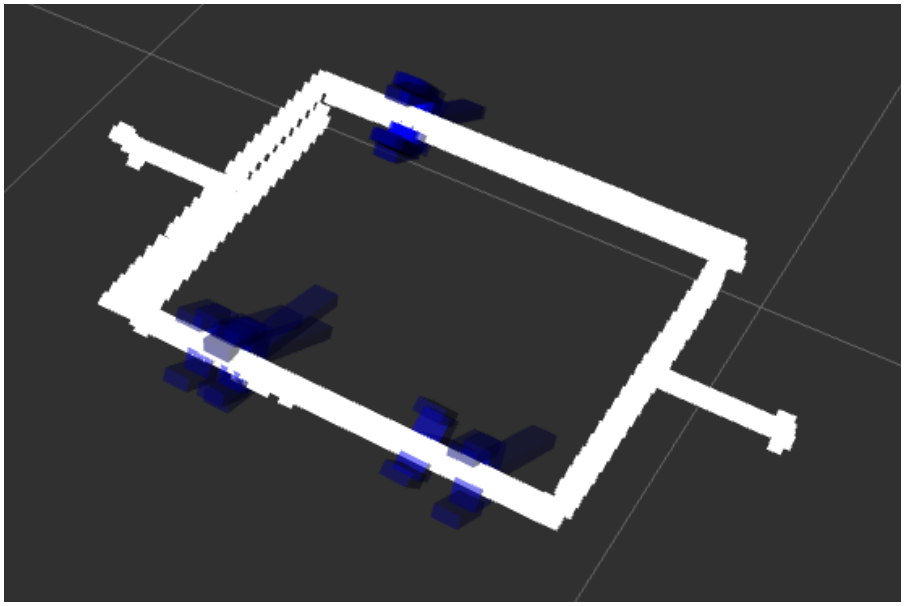


Figure 15: GPD output for filtered cloud

- In the 3rd test the body of the tray was filtered out and only handles were used to generate the grasps. In this case few grasps were generated. But since most of the tray body was filtered out therefore some of the generated poses were through the walls of the tray.

The grasps from the 3rd test were passed on to MoveIt pick pipeline and it always failed to produce a solution. Therefore, this approach was also discarded.

4.2 Selected Method for perception and manipulation

Based upon experiments conducted with the above packages, it was observed that MoveIt! Pick and Place pipeline is not suitable for our manipulator and gripper. Therefore, it was decided to implement our own perception pipeline, using PCL, and manipulation pipeline using MoveIt!.

4.2.1 Implemented Perception Pipeline

The objective of the perception pipeline is to locate the left and right handles of the tray in 3D point cloud generated by XtionPro Live camera. Following are the main steps of the perception pipeline

4.2.1.1 Transformation of 3D point cloud

The XtionPro Live camera publishes point cloud data as ROS sensor_msgs in its own reference frame. This data is received by the perception node using a subscriber. In order to use this point cloud data in PCL and subsequently for manipulation two operations are required to be performed upon it.

- **Reference frame transformation:** Since the published data is in camera frame, it is useful to transform it to robot base frame or world frame so that it can be easily visualized and used in manipulation. In this work the point cloud is transformed from *xtion_optical_frame* to *base_footprint.*

- **Conversion from ROS to PCL:** Point cloud from ROS *sensor_msgs* needs to be converted to *pcl::PointCloud* to be used in PCL library.

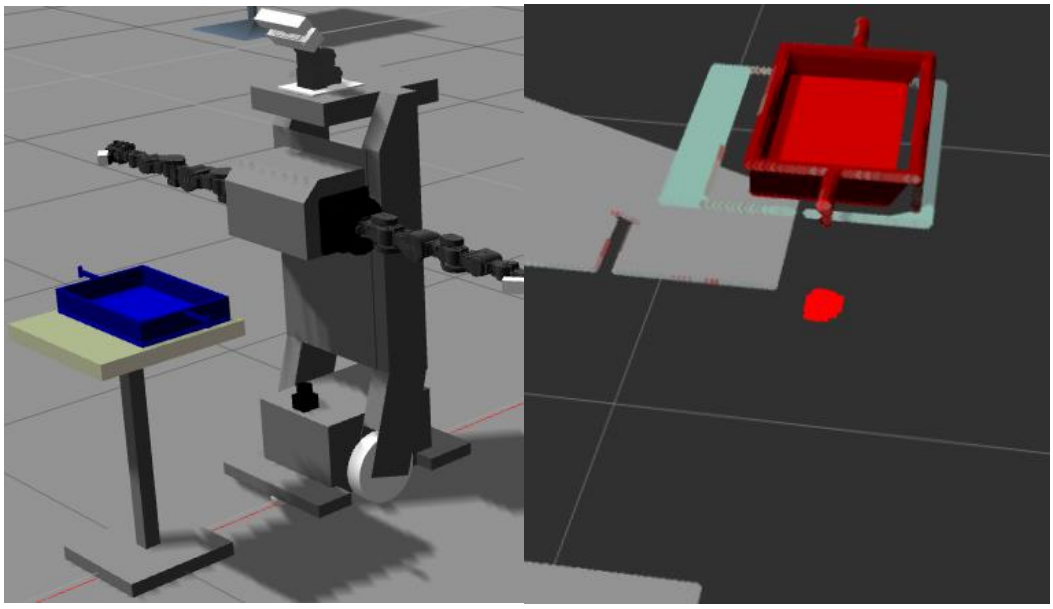


Figure 16: Original Scene(Left) and Unprocessed point cloud (right)

4.2.1.2 Down-Sampling and Filtering

The point cloud from the sensor is very dense and heavy for computation therefore it needs to be down-sampled. Voxel grid filter has been used to down-sample the data. This filter also removes any noise in the data in the form of NaN.

The down-sampled data is then used to filter out the unnecessary data related to, for example, ground, etc. and extract the region of interest. Three series filters, one for each axis, have been used in this work to crop out the table-top and tray.

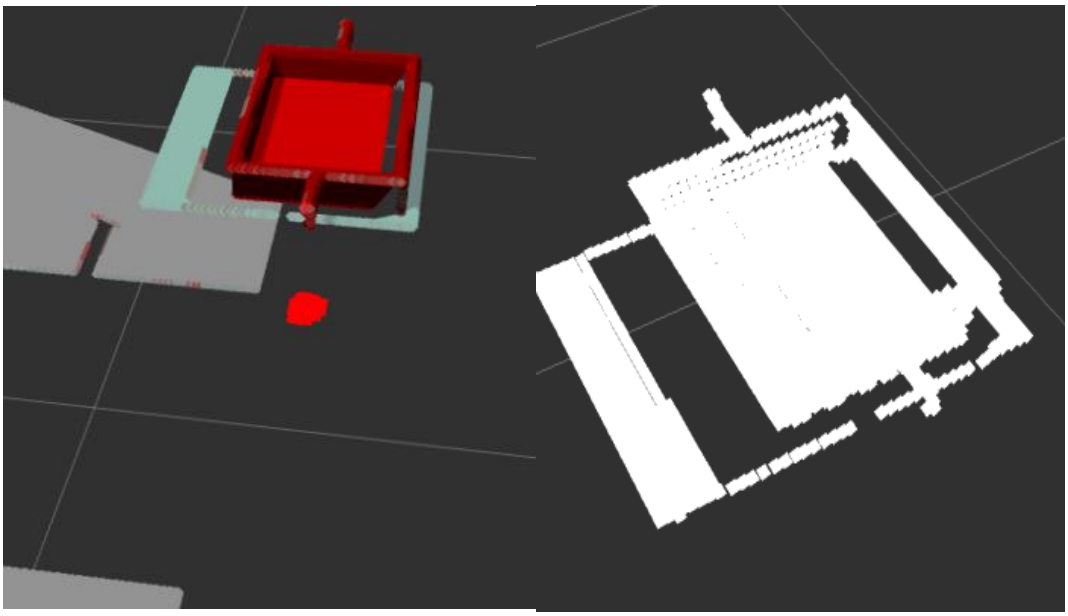


Figure 17: Down-sampled and filtered point cloud (right)

4.2.1.3 Plane Segmentation and Euclidean Cluster Extraction

In order to isolate the tray from the table we performed plane segmentation using RANSAC [34] algorithm. Using the plane segmentation, the table surface was detected and removed to get the tray was isolated.

The isolated tray cloud is then fed to the Euclidean cluster extraction algorithm. This process is performed to detect multiple objects from a point cloud. In this work it has been used to detect the handles of the tray.

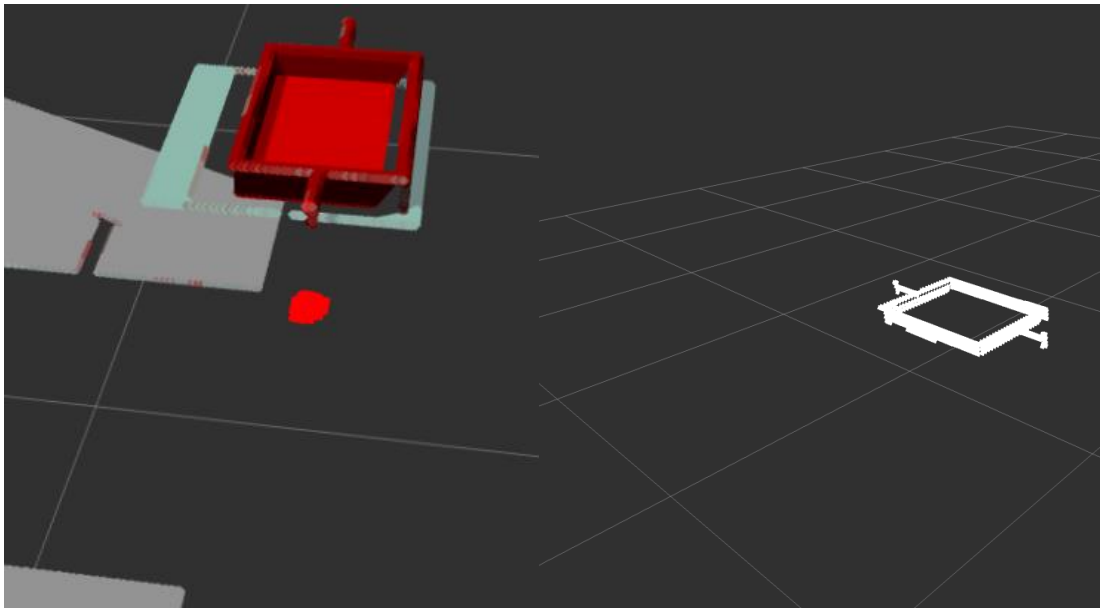


Figure 18: Point Cloud after Plane segmentation and Euclidean Extraction

4.2.1.4 Pose detection of handles

To calculate the location of handles two approaches have been explored. In the first approach the Euclidean cluster extraction output has been used. The extracted clusters provided by the algorithm are arranged on the base of the size of each cluster, however the generated clusters are not the same every time so detecting a cluster as a handle was not straightforward therefore this approach has not been used in the final solution.

The other approach used the geometry of the tray. Since the handles of the tray are the outermost parts on either side therefore this information has been used to calculate the position.

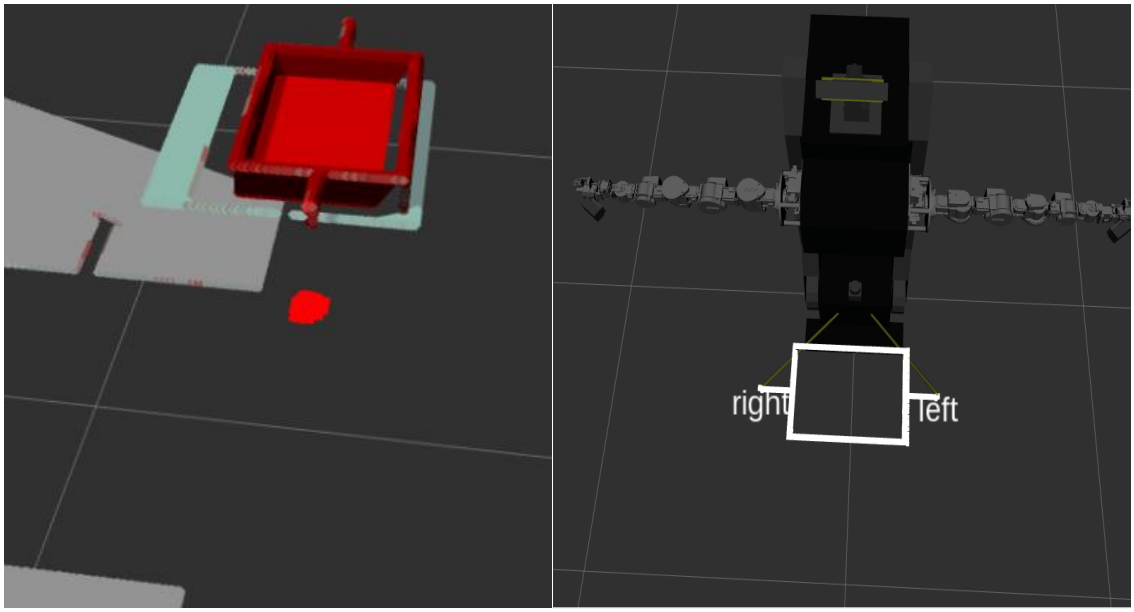


Figure 19: Handle Location from point cloud

4.2.2 Manipulation pipeline implementation

Since we have decided to use MoveIt as a manipulation interface therefore the manipulators need to be configured to be integrated with MoveIt.

4.2.2.1 Phoebe MoveIt! configuration

MoveIt! *Setup Assistant* has been used for integrating the manipulators with MoveIt! This package generates the required configuration and launch files. As MoveIt! uses the planning groups for motion planning therefore, during the setup of MoveIt configuration package six planning groups have been created: three for the arms and three for the end effectors.

- Right Arm group
- Right gripper group
- Left arm group
- Left gripper group
- Dual-arm group: Contains two sub-groups of Right arm and left arm
- Grippers group: Contains two sub-groups of Right gripper and left gripper

The point cloud data from the Xtion Pro Live 3D sensor has also been integrated with MoveIt!. The point cloud data is used by MoveIt! Planning Interface to update the

planning scene which is subsequently used during motion planning and collision detection and avoidance.

4.2.2.2 Motion Planning and Execution

MoveIt provides multiple ways to plan motion i.e. Pose goal, joint-space goal and Cartesian Paths etc. For a Pose goal we need to provide the quaternion, i.e. position and orientation, of the end effector as a Pose message. Using the perception pipeline we can get the position of handles of the tray. However, in order to pick the object, we also need to provide the correct orientation of the end effector. This orientation depends upon the geometry of the object to be manipulated. In our use case, we want the orientation to be such that the gripper's fingers are perpendicular to the handle of the tray so that while lifting it up it remains horizontal. Therefore, fixed orientation of the end effector has been selected.

The objective of our manipulation pipeline is to pick and place the tray using the dual-arm group while keeping it horizontal and without breaking it apart. The motion planning and IK solving plugins of MoveIt works only for connected chains [5]. However, the dual-arm group consists of two sub-groups that do not form a chain, therefore, this group cannot be directly used for motion planning using the *Pose Goal* function of MoveIt. There are two possible solutions: use two instances of MoveIt! *move_group* nodes for each arm or to come up with a custom solution. The former was not feasible due to the following reasons:

- it requires double computation resources
- it is very difficult to synchronize both the instances to carry out the task
- IK solvers produce random solution so the arms may not be able to pick the tray

Keeping in view these issues it was decided to solve the task using a custom solution.

4.2.2.3 Pick operation using dual-arms

The dual-arm pick operation has been divided into number of steps:

I. **Planning pre-grasp pose for a single arm:** In the first step we want the arms to move close to the handles so that the grippers are perpendicular to the handles in the horizontal axis. The position of the tray's right side handle, received from the

perception pipeline, along with the pre-calculated orientation of the end effector is sent to MoveIt as *pose goal* target for calculating pre-grasp pose for right arm group. If the planning is successful, the move group node returns a plan trajectory for the right arm.

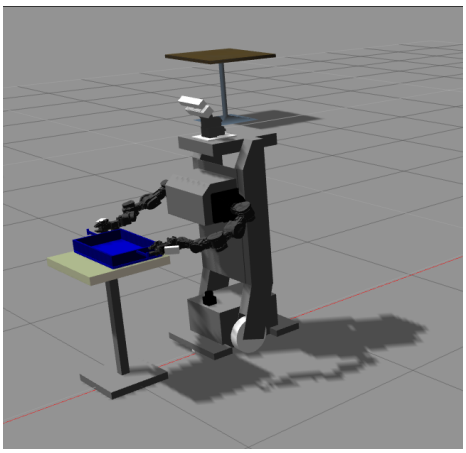


Figure 20: Pre-grasp Pose of dual-arm group

II. **Planning and executing pre-grasp pose for dual-arm:** The trajectory received in step I contains the joint position values for all the joints of the right arm. Since both right and left manipulators of Phoebe are similar therefore the trajectory calculated for the right arm can be mirrored for the left arm, provided we negate the roll and yaw joint values. So we use the end point of the trajectory of the right arm and mirror it for the left arm with proper sign changing. In this way we have the final joint positions for dual-arm group to go to the pre-grasp position. We use the *set_joint_value_target* method of *move_group* node to move both arms to the pre-grasp position simultaneously while both arms follow similar trajectory and the end effectors have the same orientation.

III. **Moving Both arms forward to grasp the handles:** Whence the pre-grasp pose has been achieved successfully by dual-arm group, the grippers are opened and each arm is moved separately in forward direction using Cartesian path planning. If both the arms move forward successfully and the both the handles are within the grippers, then the grippers are closed.

IV. **Lifting up the tray using the dual-arm group:** This is the critical step as we need both arms to move synchronously. To perform this step, we plan a Cartesian path in the upward direction for the right arm and then use mirror method to get the joint values for the left arm and finally use the dual-arm group to lift the tray up.

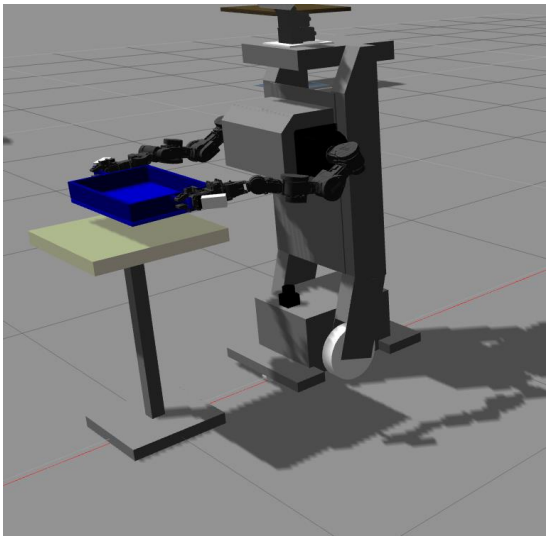


Figure 21: Pick Operation using dual-arms

4.2.2.4 Place Operation using dual-arms

During the place operation, in order to bring the tray down smoothly, the motion needs to be synchronized. To achieve this, we use a Cartesian path and the mirror method to get joint values for dual-arm group and place the tray on the table. Once the tray is on the table, the grippers are opened and each arm is retracted backward individually to pre-grasp position. From the pre-grasp position, the arms can be brought down individually or synchronously using the dual-arm group.

4.3 Phoebe indoor navigation using ROS navigation stack

ROS provides a powerful 2D navigation stack that can be used for mobile robots. It consists of multiple packages like *amcl, move_base, global_planner, dwa_local_planner and costmap_2d.* It uses robot odometry, sensor data and goal pose to calculate the velocity commands for the robot's mobile base. ROS navigation stack can be used on differential drive and holonomic wheeled robots only. It requires a laser or 3D sensor for map building, localization and obstacle avoidance.

4.3.1 Configuration of ROS Navigation stack

The overview of ROS navigation stack is shown in Figure 22

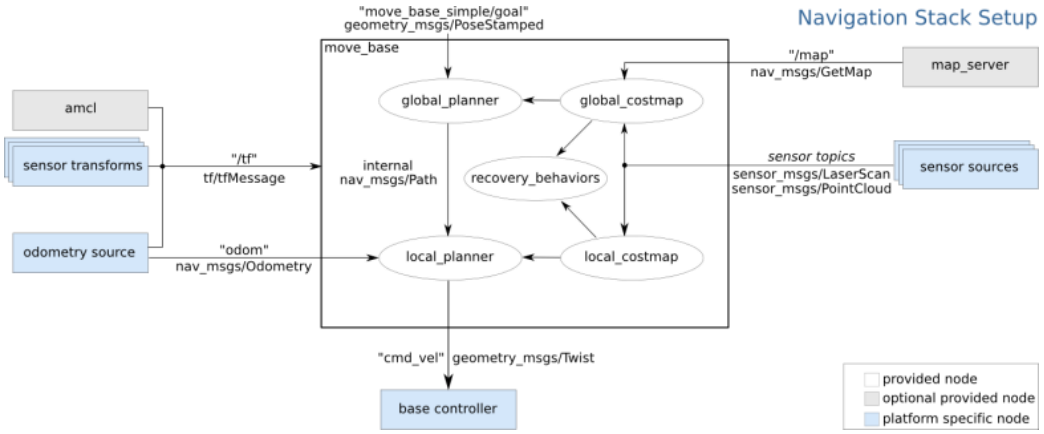


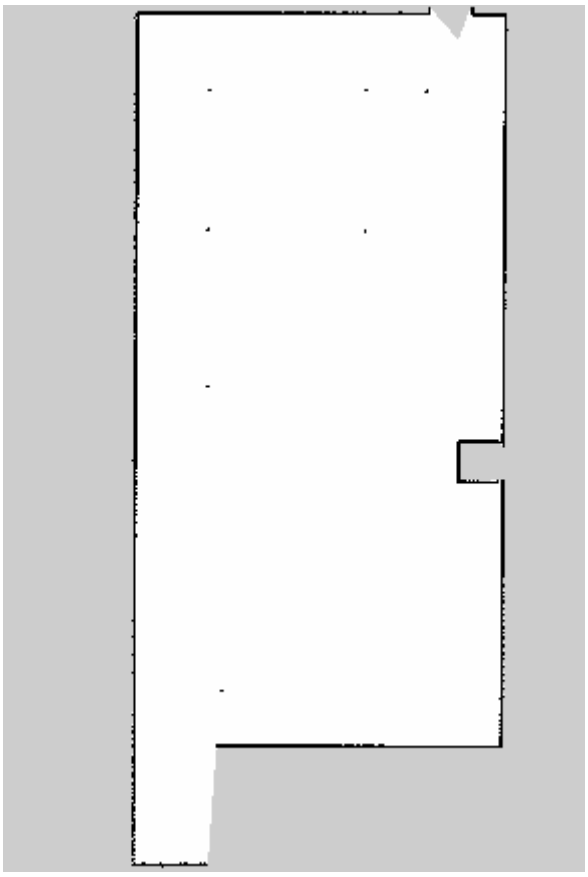
Figure 22: Overview of ROS Navigation Stack [35]

The *amcl* node of navigation stack implements the Adaptive Monte Carlo Localization system for moving the robot in 2D against a known map. It uses map, laser scans and the robot transforms to provide the estimated pose of the robot in the map. This node has a lot of parameters that need to be configured for proper operation of the node. The documentation for configuration of the *amcl* node can be found at [36].

4.3.1.1 Creating a map using ROS

The *gmapping* [1] package of ROS can be used to create a map. This package implements a laser based SLAM (Simultaneous Localization and Mapping) algorithm as *slam_gmapping* node. This node can be used to create 2D occupancy grid map using laser scan and robot odometry. In order to create the map of the area using SLAM, the robot needs to be moved in that area. For this work, *turtlebot_teleop* [1] package has been used by remapping our robot's odometry to the node's odometery input parameters in order to move the robot. Figure 23 (a) show the Gazebo cafe world that has been used as an indoor environment for the testing of robotic waiter. Figure 23 (b) depicts the map created using *slam_gmapping* node. The obstacles, like tables, and walls of the cafe are shown black in the map while the unoccupied area is represented by white space.

(a)



(b)

Figure 23: Cafe World (a), Map of the café world created using SLAM

4.3.1.2 Path Planning using LiDAR and 3D Point Cloud

The *move_base* package is the main package that implements the path planning and execution tasks of navigation stack. It implements global planning, local planning and robot steering. The global planner is used to plan the entire path from start to end using the information from the static map created through SLAM. The local planner uses the path generated by the global planner, scans the path on runtime, using LIDAR and/or RGB-D camera, for any obstacle. If it detects any obstacle it creates a new path and steers the robot to avoid this obstacle. The local planner tries to return to the global plan after any deviation in the path due to obstacles. Both local and global maps use *costmaps* to keep all the information of the map. The costmaps are configured using three configuration files:

- *costmap_common_params.yaml:* used by both planners. Contains information about the sensors to be used for obstacle detection, scan range of the sensors, cost scaling factor to tune the behavior of planner around obstacles, and geometry of the robot. In this work we have integrated the LiDAR and XtionPro Live 3D sensor with navigation stack therefore *voxel* type map has been used to detect obstacles. High obstacles like table tops cannot be detected using sensors which are mounted near the ground therefore the high mounted RGB-D camera complements the LiDAR sensor.

- *global_costmap_params.yaml:* used by global planner only. Contains the base_frame, map_frame and some other parameters.

- *local_costmap_params.yaml:* used by the local planner only. Contains local planner window size in addition to the frames. Further details can be found at [37].

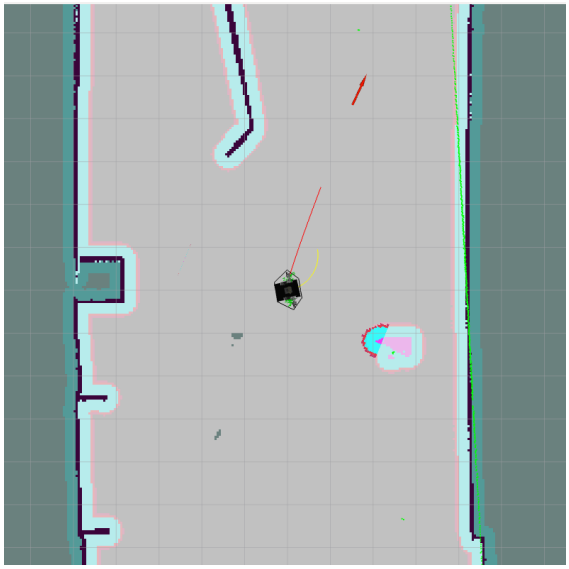


Figure 24: Path Planning using move_base. Red line: global plan, Yellow arc: local plan

ROS provides *global_planner* [38] package that is used as global planner for navigation. It can be configured to use A* or Dijkstra's algorithm for path planning. It has different parameters that need to be configured.

ROS provides various options for the selection of local planner. In this work *dwa_local_planner* [39] package has been used. Further documentation and configuration can be found [39]. Tuning of navigation stack has been carried out as per the guideline provided in [40].

5 Experiments and Results

During the development and testing of this work, a number of tests have been carried out. This chapter will summarize those tests. As the task requires the integration of perception, manipulation and navigation subtask, each subtask has been tested separately before final integrated testing.

5.1 Experimental Setup

Following system configuration has been used for the development and testing:

- Computer with Ubuntu 16.04 Xenial
- ROS Kinetic
- MoveIt 1.0
- PCL 1.4.1
- Gazebo 7.1

5.2 Perception pipeline test

The main objective of the perception pipeline is to calculate position, (x, y, z), of tray handles in the point cloud that is used in the manipulation task as target position. To test the accuracy of the computed point we need to know the actual values of that point. In the Gazebo world each link of every element has fixed coordinates. These coordinates, transformed to robot's reference frame, have been used as reference values for the testing of perception pipeline. Lighting conditions, robot and tray positions, and PTU pose have been kept constant during the tests. During development and testing the perception pipeline has been tuned by adjusting voxel grid size, Euclidean cluster size and tolerance and plane segmentation threshold. Following tables shows the results of the tests:

Test No	Left Handle Actual Position (x, y, z)m	Left handle Computed Position (x, y, z)m	Right Handle Actual Position (x, y, z)m	Right Handle Computed Position (x, y, z)m
1	0.5645, 0.2625, 0.705	0.5692,0.2636,0.6848	0.5645, -0.2725, 0.705	0.5646,-0.2755,0.7050
2		0.5691,0.2634,0.7041		0.5645,-0.2756,0.7048
3		0.5692,0.2633,0.6784		0.5709,-0.2761,0.7056
4		0.5692,0.2634,0.6784		0.5708,-0.2762,0.7062
5		0.5692,0.2634,0.6783		0.5642,-0.2761,0.7050
6		0.5693,0.2628,0.6948		0.5644,-0.2763,0.7051
7		0.5694,0.2627,0.6949		0.5642,-0.2765,0.7050
8		0.5694,0.2626,0.6844		0.5643,-0.2767,0.7052
9		0.5695,0.2624,0.6945		0.5641,-0.2769,0.7052
10		0.5695,0.2624,0.6843		0.5626,-0.2802,0.7066

Table 5: Perception Pipeline Tests

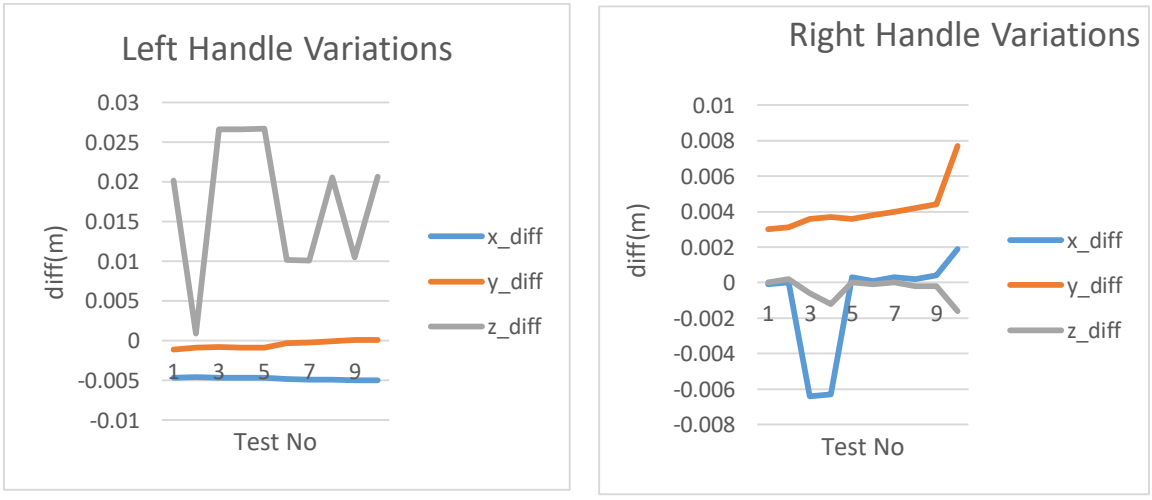


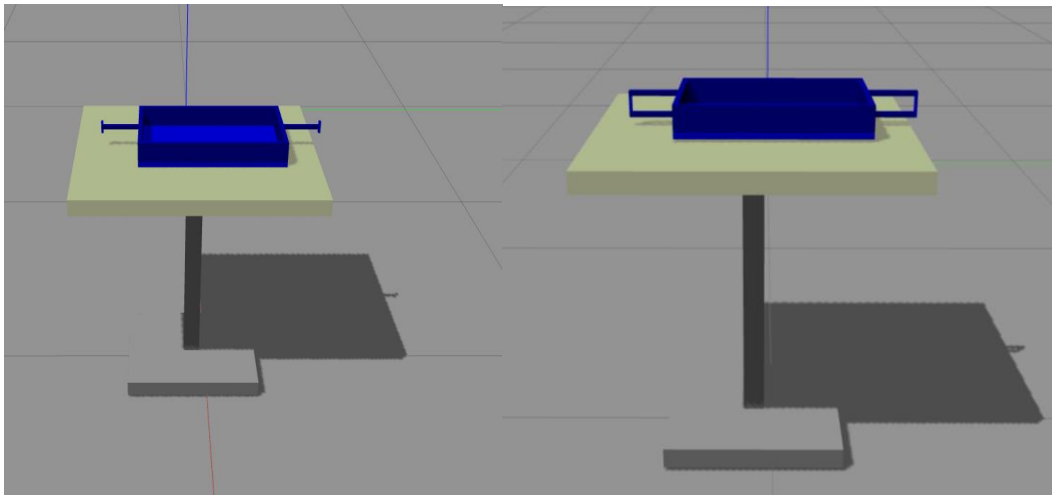
Figure 25: Graph of the variation (difference between actual and computed values) in perception readings

From the graphs with the limited number of tests, in the unchanging test conditions, it can be seen that there is a significant variation in the output of the perception pipeline. In order to improve the accuracy of the readings, an average of 3 readings has been used for the integration.

5.3 Manipulation Pipeline testing

The objective of the manipulation pipeline is to pick and place the tray using dual arms while maintaining its level and avoiding collision with the surrounding objects like table etc. As previously described, MoveIt has been used in this work. During the development phase of the work, different motion planners, e.g. *PRMstarConfigDefault*, *RRTConnectkConfigDefault* etc., have been used for single and dual arm manipulation. Similarly, the default IK Solver *KDL* as well as *IKFast Kinematics Solver* have also been used.

For testing of the manipulation pipeline, two trays with different types of handles have been used as shown in the following Figure 26. For each tray, different orientations of the end effector have been used for testing. For the configuration (named Tray2) depicted in Fig 25 (b) two orientations of the end effector are used and for configuration (Tray1) in Fig 25(a) one orientation of the end effector.



(a) (b)

Figure 26: Trays used for Testing of manipulation pipeline

Since the manipulation pipeline requires the *Pose Goal* target as input which in turn is calculated based upon the output of the perception pipeline, therefore, the variations in the output of the perception pipeline have caused some failures in the pick task of manipulation.

A total of 30 repetitions of the pick and place task of manipulation pipeline have been performed for both the tray of Fig 25. No extra obstacle has been added in the planning scene except the table. The path planning was carried out under fixed orientation constraint of the end effector for pre-grasp pose. The robot was positioned at a fixed

distance, 57cm, from the table facing directly toward the tray to be lifted. This distance has to be within the manipulator reach of 68cm. The exact value was selected based on trial and error method of performing number a of planning tasks. The test results are summarized in Table 6.

Test Configuration	**No of Tests**	**Pass**	**Fail**	**Failure Stage/Reason**
Tray1	30	24	6	3 failed due to collision with tray handles due to inaccuracy of perception calculations
				1 failed during Cartesian path planning of MoveIt
				1 failed due to unequal movement of arms. Robot orientation was not aligned with the tray
				1 failed during place stage. Tray collided with table due to extra downward motion
Tray2	30	17	13	9 failed due to collision with tray handles due to inaccuracy of perception calculations
				2 failed during Cartesian path planning of MoveIt
				2 failed due to unequal movement of arms. Robot orientation was not aligned with the tray.

Table 6: Pick and Place task test data for Tray1 and Tray2

As can be observed from the above data, the majority of the failures, in each case, were due to the position reported by the perception pipeline which is based upon the PCL library. This can be mitigated by recalculating the position of the handles from the pre-grasp pose by using the wrist-mounted cameras of the manipulators.

Furthermore, the maximum opening range of the gripper of the Cyton Gamma 1500 manipulator is only 3.5cm and the width of handles is approximately 1.2cm. In the case of tray2, the handles are quite complex, for a robotic manipulation task using this gripper, as the grippers have to be guided accurately between the 3 sides of the handles to grasp. This complex handle shape, smaller gripper opening range and the variations in the computed position caused the higher failure rate for Tray2.

During the motion of the manipulators, the tray needs to be kept stable. As mentioned in section 3.3 ROS controllers use PID feedback mechanism to keep the joint stable therefore the PID values for each joint of simulated manipulator have also been tuned using *rqt Dynamic Reconfigure tool* [41].

5.4 Navigation Pipeline testing

For the navigation pipeline, the pass scenario is the collision free motion of robot from kitchen to the serving table. The testing scene is Gazebo *Cafe world* as shown in Figure 23. The major part of the implementation of the navigation pipeline is the configuration of local and global planners. The *dwa_local_planner* and *global_planner* using Dijkstra's algorithm have been used in this work. The maximum linear velocity, during the tests, was 1.85 m/sec and the maximum rotation velocity was 2.5 rad/sec.

The objective of the testing is to evaluate the following capabilities:

- Navigation: The ability of the robot to plan and follow a path, for different conditions. The repeatability has been tested by giving a same goal pose for 10 iterations under same conditions without any additional obstacle. The following table shows the results. The achieved goal has been read from the robot's pose in the Gazebo world.

Test No	**Goal Pose (x, y, Ɵ)**	**Achieved Pose (x, y, Ɵ)**
1	2.0, 5.25, 1.5708	1.985, 5.215, 1.562
2		1.975, 5.236, 1.554
3		2.05, 5.234, 1.564
4		2.05, 5.271, 1.557
5		1.97, 5.195, 1.552
6		1.98, 5.232, 1.563
7		2.07, 5.274, 1.564
8		2.04, 5.226, 1.557
9		1.98, 5.208, 1.559
10		1.99, 5.262, 1.562

Table 7: Path planning and execution test results

- Collision avoidance: Will the robot avoid new obstacles added in the environment that are not in the static map?

Following table summarizes the test and the results.

Test Configuration	**No of Tests**	**Pass**	**Fail**	**Failure Stage/Reason**
Goal between the tables	5	5	0	Nil
Goal too close to the wall	5	0	5	The inflation layer around the walls forced the robot to stop before the goal. This is the required behavior, since it ensures safety against collision with the walls
Path too close to the table edge	5	3	2	1 failed. PTU was pointed too high and the table surface was higher than LiDAR range. Both sensors could not detect the obstacle.
				1 failed due to high speed, table surface was detected too late to avoid the collision
Addition of obstacle in the path	5	4	1	Robot speed was high and the added obstacle was close to the robot.

Table 8: Summary of navigation tests

Since some of the tests failed due to the speed of the robot, therefore, the maximum linear and rotation velocities have been reduced to 0.35m/sec and 1.0 rad/sec respectively.

5.5 Integration testing

After the development and testing of individual components, the same were integrated to carry out the complete task of robotic waiter: to pick the tray, from kitchen, using both arms, move across the cafe, while carrying the tray in both arms, to the serving table and place it there. A total of 20 repetitions have been carried out for the complete task and the results are summarized in the following table:

Test Configuration	No of Tests	Pass	Fail	Failure Stage/Reason
Locate the handles of the tray from the point cloud, pick the tray using both arms, while stably holding the tray move to the serving table and place the tray.	20	14	6	3 failed due to collision of the manipulator with tray handles
				2 failed during forward motion planning of right arm
				1failed during place operation

Table 9: Summary of Integrated testing

The results show that the proposed system can be used for dual arm manipulation. However, the system is not very robust as there are two weak links in the proposed solution that are the apparent cause of most of the failures. The position estimation based upon the 3D point cloud using PCL is not very accurate. This erroneous position, when used for motion planning and IK solutions causes failures. Motion planning and IK solutions themselves fail occasionally.

6 Conclusion and Future Work

6.1 Conclusion

- **Perception**: The perception subtask has been implemented using PCL library. As can be observed from the test results from the previous chapter, the output of perception pipeline varies considerably. This output is used in manipulation. In most of the cases this output is within the limits of gripper range of the manipulator. When this variation exceeds the limit, it causes failures. In conclusion, the perception pipeline is good enough to get the approximate pose of the handles from the point cloud.

- **Manipulation**: MoveIt has been used for motion planning and IK solution of the manipulation subtask. The overall performance of manipulation pipeline can be graded good when the failures due to erroneous input coordinates are taken into account.

- **Navigation**: ROS navigation stack has been used for robot navigation in the simulated indoor environment. Due to the sensor fusion of LiDAR and RGB-D camera, and low speed of the robot, the navigation stack has shown good performance.

Based on the results presented in previous chapter, it can be concluded that the performance of proposed solution is satisfactory in the simulated indoor environment. However, the solution is not robust. It is very sensitive to errors in perception and also to the pose of the robot and tray. The errors get magnified during motion planning and IK solution and cause failures.

6.2 Future Work

The solution developed in this thesis can be extended and improved in multiple directions as discussed below.

- **Perception**:
    - 2D image processing capability can be integrated with 3D point cloud processing to improve the accuracy of location estimation.

- Object detection and recognition capability can be added so that different objects can be handled by the robot. This can further be used for automatic grasp pose generation capability based on the type of recognized object.

- **Manipulation**
  - Automatic grasp pose selection capability for different objects will be beneficial for object manipulation.
- A GUI (Graphical User Interface) can also be developed that will allow the user to select different simulation environments, selection of different sensor combinations for the robot and different robotic platforms as well. This interface can also be used to select robot speed during the navigation task etc.

7 Summary

The main objective of this thesis was to develop a solution for dual-arm manipulation for a robotic waiter. The development task has been simplified to the use case of picking a single object using dual-arms simultaneously while maintaining the orientation of the object, moving the robot to another location, and placing the object

The task has been divided into three steps of locating two pick points on the object from 3D point cloud, lifting the object using both the manipulators simultaneously, and moving the robot in the indoor environment to the place location. The task has been carried out using a modular approach of developing individual components separately and then integrating them to perform as a mobile robotic waiter. The perception and navigation packages developed during the task can be used independently. The proposed solution has been developed for the Phoebe robot. Open source tools like ROS, MoveIt, PCL, and Gazebo have been used for the solution of the task.

The developed solution has been tested in the simulated environment using the Gazebo simulator. On the basis of the results of these experiments it is concluded that the developed solution is capable of achieving the required objective. However, the solution is very sensitive to errors in perception and the relative pose of the robot and tray. Furthermore, there are certain limitations for solution: it cannot be directly used for the manipulation of any arbitrary shape object.

References

[1] "ROS Documentation," wiki.ros.org, [Online]. Available: http://wiki.ros.org/Documentation. [Accessed 2020].

[2] A. H. Nathan Koenig, "Design and Use Paradigms for Gazebo, An Open-Source Multi-robot Simulator," in *IEEE/RSJ International Conference on Intellegent Robots and Systems*, Sendai, Janpan, 2004.

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

[4] "PCL," [Online]. Available: https://pointclouds.org/documentation/. [Accessed 2020].

[5] I. S. S. C. Sachin Chitta, "MoveIt! [ROS Topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18-19, March 2012.

[6] "MoveIt," [Online]. Available: https://moveit.ros.org/documentation. [Accessed 2020].

[7] "OMPL," [Online]. Available: https://ompl.kavrakilab.org. [Accessed 2020].

[8] "KDL," [Online]. Available: https://www.orocos.org/wiki/Kinematic_and_Dynamic_Solvers.html. [Accessed 2020].

[9] "TRAC-IK," [Online]. Available: http://wiki.ros.org/trac_ik. [Accessed 2020].

[10] "BehaviorTree.CPP," [Online]. Available: https://www.behaviortree.dev/. [Accessed 2020].

[11] M. C. C. S. a. P. Ö. A. Marzinotto, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014.

[12] R. F. E. N. M. A. P. G.-D.-S. Delia Sepúlveda, "Robotic Aubergine Harvesting Using Dual-Arm Manipulation," *IEEE Access, vol. 8, pp. 2020, doi: 10.1109/ACCESS.2020.3006919.*, vol. 8, pp. 121889-121904, 2020.

[13] S. K. T. M. T. Y. W. W. Y. O. N. C. N. K. Y. N. I. M. T. S. Ryo Sakai, "A mobile dual-arm manipulation robot system for stocking and disposing of items in a convenience store by using universal vacuum grippers fo grasping items," *Advanced Robotics*, vol. 34, pp. 219-234, 2020.

[14] I. R. a. D. O. P. Sven Cremer, "Robotic Waiter with Physical Co-manipulation Capabilities," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Taipei, Taiwan, August, 2014.

[15] A. J. Ruchik Mishra, "ROS Based Service Robot Platform," in *4th International Conference on Control, Automation and Robotics (ICCAR)*, Auckland, 2018.

[16] J. T. M. R. Sukkpranhachai Gatesichapakorn, "ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera," in *2019 First*

International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), Bangkok, Thiland, 2019.

[17] T. A. V. K. F. S. Kenta Takaya, "Simulation Environment for Mobile Robots Testing Using ROS and Gazebo," in *20th International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, 2016.

[18] R. L. A. G. I. A. E. M. Maxim Sokolov, "3D modelling and simulation of a crawler robot in ROS/Gazebo," in *4th International Conference on Control Mechatronics and Automation*, Barcelona Spain, 2016.

[19] "Collada," [Online]. Available: https://www.khronos.org/collada/. [Accessed 2020].

[20] "STL," [Online]. Available: http://www.fabbers.com/tech/STL_Format.

[21] "URDF," [Online]. Available: http://wiki.ros.org/urdf/Tutorials/.

[22] "PeopleBot DataSheet," [Online]. Available: https://www.generationrobots.com/media/PeopleBot-PPLB-RevA.pdf.

[23] "ros_diff_drive," [Online]. Available: http://wiki.ros.org/diff_drive_controller. [Accessed 2020].

[24] "CytonGammaDatasheet," [Online]. Available: https://datasheet.octopart.com/CYTON-GAMMA-1500-Robai-datasheet-67184669.pdf. [Accessed 2020].

[25] "Flea3Camera," [Online]. Available: https://www.flir.eu/products/flea3-usb3/. [Accessed 2020].

[26] "BumbleBee," [Online]. Available: https://www.flir.com/support/products/bumblebee2-firewire/#Overview.

[27] "XtionProLIve," [Online]. Available: : https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/. [Accessed 2020].

[28] "HokuyoLiDAR," [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=233. [Accessed 2020].

[29] E. M.-E. W. M. V. P. A. R. T. J. B. D. C. B. M. Sachin Chitta, "ros_control: A generic and simple control framework for ROS," *The Journal of Open Source Software*, 2017.

[30] "Gazebo," [Online]. Available: http://gazebosim.org/tutorials?tut=ros_control&cat=connect_ros. [Accessed 2020].

[31] M. Ferguson, "Simple Grasping," [Online]. Available: https://github.com/mikeferguson/simple_grasping. [Accessed 2020].

[32] A. t. Pas, M. Gualtieri, K. Saenko and . R. Platt, "Grasp Pose Detection in Point Clouds," *The International Journal of Robotics Research,*, vol. 36, no. 13-14, pp. 1455-1473, 2017.

[33] "GPD_ROS," [Online]. Available: https://github.com/atenpas/gpd_ros/.

[34] R. R.-M. F. Pollefeys, "A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus," in *European Conference on Computer Vision*, 2008.

[35] "move_base," [Online]. Available: http://wiki.ros.org/move_base. [Accessed 2020].

[36] "ROS-AMCL," [Online]. Available: http://wiki.ros.org/amcl?distro=noetic. [Accessed 2020].

[37] "Config Navigation stack," [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed 2020].

[38] "Nav_global_planner," [Online]. Available: http://wiki.ros.org/global_planner?distro=kinetic. [Accessed 2020].

[39] "Nav_dwa_local," [Online]. Available: http://wiki.ros.org/dwa_local_planner?distro=kinetic. [Accessed 2020].

[40] K. Zheng, "ROS Navigation Tuning Guide," 2016. [Online]. Available: https://arxiv.org/abs/1706.09068. [Accessed 2020].

[41] "rqt," [Online]. Available: http://wiki.ros.org/rqt. [Accessed 2020].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Muhammad Qasim Imran

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Dual-Arm Manipulation in Robotic Waiter Use Case”, supervised by Gert Kanter
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

05.01.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2: Code Repository

The software packages developed for this work have been upload to the following repository:

https://github.com/qasimimran1/robotic_waiter