Design of a simulation framework for POMDP based autonomous object grasping with a mobile robot manipulator

Aldiyar Dabarov, B.Eng

Submitted in fulfilment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

School of Engineering and Digital Sciences
Department of Electrical and Computer Engineering
Nazarbayev University

53 Kabanbay Batyr Avenue,
Nur-Sultan, Kazakhstan, 010000

Lead Supervisor: Almas Shintemirov
Co-supervisor: Ablaikhan Akhazhanov

March 2022

Declaration

I hereby, declare that this manuscript, entitled "Design of a simulation framework for POMDP based autonomous object grasping with a mobile robot manipulator", is the result of my own work except for quotations and citations which have been duly acknowledged.

I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

Name: Aldiyar Dabarov

Date: 28.04.2022

Abstract

The ALARIS laboratory is developing a mobile manipulator for POMDP-based object grasping tasks. It is necessary to develop a simulation environment for testing purposes. However, the task can be repeated using different robotic platforms after the completion of the project. As a result, it was decided to develop a simulation framework that meets the requirements of the project. The requirements for such a framework were defined through the analysis of the resources with similar setups and goals. These works were collected through a systematic literature review. With the combination of the requirements from the initial project and the ones obtained from the literature, the simulation framework was designed, developed, and tested.

Acknowledgements

After seven years, my life as a student at Nazarbayev University comes to the finish line. It has been a time full of theoretical and practical experience. Over this time, I have worked in different subfields of Electrical Engineering, starting from mathematical modeling, transferring to RF circuits design, and finally finding my passion for Robotics and Software Development.

Thus, in the first place, I would like to thank my supervisor Dr. Almas Shintermirov who introduced me to this topic and had confidence that I would handle this relatively challenging task. Facilities provided in the ALARIS lab allowed me to work on different projects, including mobile robots and indoor GPS, before deciding to focus on the topic of this thesis. Without my supervisor, I would have not explored the robotics field that well.

Then, I would like to thank the professors with whom I worked during my bachelor's, Dr. Aresh Dadlani and Dr. Mohammad Hashmi. The work with Dr. Dadlani was my first research experience. It was the reason for my getting interested in academia. His expertise and support were the reason for my personal growth in research and software development. Then, the work with Dr. Hashmi introduced me to the field of RF circuit design, which was another great opportunity to grow as an Electrical Engineer.

Most importantly, I am grateful to my friends and family, especially my laboratory peers, for always supporting and helping me during my studies. I would not have finished this thesis without all these people.

Table of Content

Abstract 3

Acknowledgments 4

List of Abbreviations 7

List of Tables 8

List of Figures. 9

Chapter 1 - Introduction 10

1.1 Aim and Objectives 11

1.2 Thesis Structure 11

Chapter 2 - Background 13

2.1 Software Framework 13

2.1.1 Value of Using a Framework 14

2.2 Understanding POMDP 15

2.2.1 MDP 15

2.2.2 POMDP 17

Chapter 3 - Literature Review 20

3.1 Literature Review Method 20

3.2 Core parts 23

3.3 Additional parts 23

Chapter 4 - Framework Design. 27

4.1 Technology Choice 27

4.1.1 System 28

4.1.2 Simulation Environment 28

4.1.3 Default Functionality 30

Chapter 5 - Framework Implementation 34

5.1 Installing the Dependencies 34

5.2 Building Environment 38

5.3 Adding Control Functionality 42

Chapter 6 - Framework Testing 46

6.1 Testing Simulation Environment 46

6.2 Testing Control Services 47

Chapter 7 - Conclusion 52

7.1 Summary 52

7.2 Future Work 52

References 55

List of Abbreviations and Symbols

ALARIS	Astana LAboratory for Robotic and Intelligent Systems
API	Application programming interface
CAD	Computer-aided design
DOF	Degrees of freedom
GUI	Graphical user interface
MOMDP	Mixed observability Markov decision process
POMDP	Partially observable Markov decision process
ROS	Robot Operating System
RPC	Remote procedure call
SEO	Search engine optimization
YCB	Yale-CMU-Berkeley

List of Tables

Table 1 Search queries and results for each database utilized 21

Table 2 Search results after filtering by year and publication type 21

Table 3 Number of relevant papers based on title, abstract and content . . . 22

Table 4 Selected works for the literature review 23

Table 5 Summary of differences in the selected literature 24

Table 6 Results of the POMDP planners used in the developed interface . . . 31

List of Figures

Figure 1 Systematic literature review method 20

Figure 2 Gazebo interface created for a rock-sample task 31

Figure 3 Framework design structure 32

Figure 4 The office world with robot in Gazebo 46

Figure 5 The Gazebo world after spawning bookshelf and YCB objects . . . 47

Figure 6 The robot in the home state 48

Figure 7 The state in front of the orange object 49

Figure 8 The example of calling the grab service from therminal 49

Figure 9 The orange object keeping the offset after grab command 50

Chapter 1 - Introduction

Applications like unknown environment inspection and emergency responses often expose hazards to acting people. The utilization of robots in those can significantly minimize risks. Mobile manipulators are popular platforms for cases when there is a need for interaction with the environment, such as substance collection and disaster rescue [1]. Robots should have fast actions in response to environmental changes in such scenarios. It often means that the platform should be autonomous.

The development of such robots included several steps. The important part is testing the platform in the simulated environment. This process helps to avoid the costs of testing the robots in the physical environment, where expensive equipment can be damaged. Thus, it is a necessity to do simulation testing. However, the development of the software environment is time-consuming. There exists a lot of ready-to-use simulation environments, but even those should be configured for the project's needs.

Currently, ALARIS is developing an autonomous mobile manipulator. The platform consists of Jackal mobile robot by Clearpath Robotics and Gen3 lite robotic arm by Kinova Robotics. This project aims to develop a platform for autonomous object grasping with the use of POMDP. As a result, there is a need for the development of the simulation environment. However, since it is planned to replicate the task using different robotic platforms in the future, the better solution would be the development of the simulation framework for the concrete problem.

1.1 Aim and Objectives

This thesis focuses on the development of the simulation framework. The thesis aim can be formulated as to *design and implement a simulation framework for POMDP-based object grasping task with mobile manipulator*. It is divided into several objectives. First, it is necessary to find the in-demand features for the framework through the literature review. The next goal is to develop a framework that meets the needs of the laboratory project.

1.2 Thesis Structure

The remaining of the thesis is structured as follows:

- **Chapter 2 - Background** gives a detailed overview of concepts that are necessary for an understanding of the thesis.
- **Chapter 3 - Literature Review** includes the methodology and analysis of the systematic literature review on the topic of the thesis.
- **Chapter 4 - Framework Design** discusses the choice of technologies, the list of features, and the reasoning behind these decisions.
- **Chapter 5 - Framework Implementation** gives step by step guide on how the framework was developed in the tutorial-like form.
- **Chapter 6 - Framework Testing** shows that the framework is functional and meets the set requirements.
- **Chapter 7 - Conclusion** summarizes this thesis work and suggests a possible extension of this project.

Chapter 2 - Background

This chapter covers background information to understand the topics covered in this work. First, the concept of software framework is explained, including its definition, benefits, and drawbacks of its utilization. Then, the concepts of MDP and POMDP are defined since these are the futures involved in the project.

2.1 Software Framework

Development of the software application is a complex and time-consuming process. It involves the design, development, and production of high-quality code [2]. Projects in the same area usually have similar design and development patterns. This is where software frameworks are useful. A software framework is a collection of components that are intended to produce reusable code for specific applications. The usage of frameworks reduces the amount of boilerplate code in a program. The framework usually comes with a well-defined application program interface (API) that users can build applications on top of [3]. There are a few features of the framework that make it a standalone concept:

- **Core codebase.** Frameworks have a codebase of reusable components that users are not intended to change. It includes some initial behavior that the user can extend. This is done by passing user data as parameters into the framework rather than changing its architecture of it.

- **The control logic.** In other software libraries, the logic of how components interact is defined by the user. In software frameworks, however, the control of modules within the code base is decided by the framework itself [3].

These two points can be summarized as bringing the development environment to the user. This leads to the point that software frameworks consists of *frozen* and *hot* spots. *Frozen spots* are the immutable architectural part of the codebase. It defines the reusable components and the interactions between them. *Hot spots*, on the other hand, are the parts that users utilize to add their domain logic to the application [3].

2.1.1 Value of Using a Framework

There are several clear advantages of using the framework from the definitions given above rather than creating and developing the product from scratch. A framework is a complete application that lacks functionality. The functionality part comes from the user, which results in ready to use application. It comes with several benefits to the application developers, including a lower cost of production. Less time is needed to finish the application. It means that the product needs less time to market [3].

Furthermore, well-designed frameworks bring quality benefits to the product [3]. The code that comes with it is usually pretested, which means that this part of the application code should be reliable. In addition, the frameworks are usually opinionated on how they should be utilized. It establishes the best programming practices for the code. The development pattern in the code base of the application will be more consistent with such an approach.

However, there are some drawbacks to using the framework instead of from-scratch development. First, there is less control over the control flow. The logic of interaction could limit the development of the application. For example, React JavaScript framework, which is initially developed to be client-side rendered [4]. The client-side rendering loses to the server-side rendering in speed, and SEO optimization [5]. Thus, the application is limited by the framework's design choices it is developed on top of.

Moreover, the usage of frameworks can bring another issue if the space is an important aspect of the project. Software frameworks are developed for general use in a specific area. A project might not need all the functionality that the framework provides. In that case, it is better to develop the core logic of the product from scratch, which takes additional time and production cost.

2.2 Understanding POMDP

This section explains the brief theory to understand POMDP. It starts with the explanation of Markov Decision Processes. This sets up the background for the definition of the POMDP. Both definitions include the application and the solution goals of both frameworks.

2.2.1 MDP

Markov Decision Processes is a modeling framework for problems that require sequential decision-making. There is an agent that makes the decision based on the probabilistic dynamics of the environment where it acts. As a result, it is assumed that the agent has a perfect vision of the environment. These features of the framework make it practical for a variety of real-world tasks, including robotics, trading, and gaming industry.

MDP can be expressed as a tuple $\{S, A, T, R, \gamma\}$, where

- S is a set of all possible states of the environment
- A is a set of all possible actions agent can decide to make
- $T(s,a,s')$ is a transition function, which calculates the probability of transitioning from state s to s' after taking action a

- *R(s,a,s')* is a reward function, which is similar to transition function, it rather defines the reward value of the transition from s to *s'* after taking action a
- γ is a discount factor

S and A are finite states since the environment is fully visible to the agent. However, the transition between states is not definitive. The result of the action has a certain probability defined by transition function T. Finally, there is a need to determine a more desirable outcome of actions. These values are defined by the reward function R. Nonetheless, the agent should think about the time taken to get the higher reward. This means that the actions closer to the present time are more favorable than those taken later. There, the discount factor γ comes in useful.

After defining the model, there is a need to get a solution for it. For MDP, it is to find an optimal policy π^*. It is a function that considers the set of actions and states to maximize the final reward. Thus, the solution to MDP can be considered an optimization problem. The optimal policy is defined as:

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}}\ \Omega(s,a), \forall s \in S \tag{1}$$

where

$$\Omega(s,a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{2}$$

where V^* is optimal value function, defined as:

$$V^*(s) = \max_{a \in A} \Omega(s,a), \forall s \in S \tag{3}$$

It can be seen that optimal policy π is dependent on the optimal value function

of the possible next state. Thus, the optimal policy function defines the optimal action to take on each state of the environment. The most popular solution is a recursive value iteration in the value function V.

2.2.2 POMDP

Partially Observable Markov Decision Processes (POMDPs) are an extension of regular MDP. It is utilized when the environment includes uncertainties and is not fully observable by the agent. It makes POMDP more applicable for real-world tasks, most of which are uncertain by their nature. The inclusion of the uncertainties leads to the point that the agent cannot be completely confident about the state of the environment. Thus, the MDP tuple should be modified to be more applicable to POMDP.

Instead, POMDPs are defined with a tuple $\{S, A, T, R, O, Z, \gamma, b\}$. There S, A, T, R and γ have a similar purpose as in MDP, but other variables are interpreted as:

- O is a set of all possible observations that the agent can have a perception of
- $Z(s,a,o)$ is a an observation function, which calculates the probability of having an observation o, when the agent takes action a after being in the state s
- $b(s)$ is a belief point, that defines the probability of being in the state s with the current observation o

The belief point determines the most possible state that the agent is currently in. The probability of the each possible state sums up to 1 $\sum_{s \in S} = 1$. Each probability is calculated based on the agent's observation. With that information and the set of possible actions, the agent calculates all the possible observations that it can get to with the observation function $Z(s,a,o)$. Then, this collection of possible observations is used to calculate possible belief states, called a set of reachable belief points.

Similar to MDP, the final goal is to obtain the maximum reward. However, unlike MDP, the function that is needed to be solved is not the policy but the optimal value function, which is defined as:

$$V^*(b) = \max_{a \in \Gamma^*} \sum_{s \in S} b(s)\alpha(s) = \max_{a \in \Gamma^*} b^T \alpha \tag{4}$$

where Γ^* is an optimal set of belief state vectors, where each vector has a size of state space. Those vectors are called α-vectors. It can be seen that the optimal function is calculated for the belief point b rather than the state as in MDP.

It is clear that with another level of uncertainty, the computation power required for the solution increases. As a result, POMDPs are usually solved either with some sort of approximation derived for a specific task or through the machine learning algorithms for the prediction of the current state.

Chapter 3 - Literature Review

3.1 Literature Review Method

There is a need to collect an overview of the object manipulation works that use POMDP to understand the core requirements for the developed framework. A systematic literature review, described in [6], was conducted. It is a process of consistent and non-biased literature selection. These features are possible due to the protocol's criteria that the review should be repeatable. The brief summary of the selection criteria and its output is shown in figure 1.

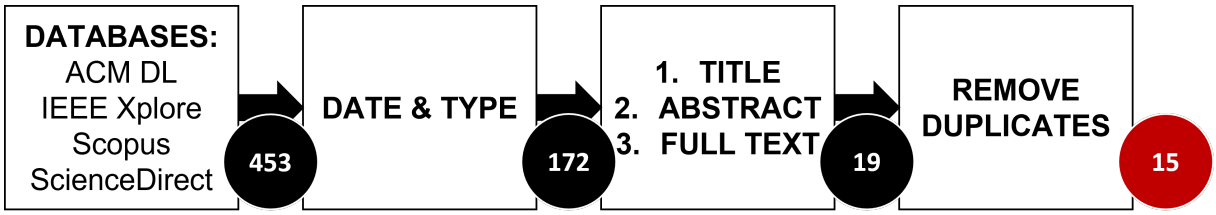

Figure 1: Systematic literature review method

First, major scientific databases were utilized to assure the quality of the conducted review. The resources that include robotics-related articles were selected since the object manipulation task is a subject of a robotics field. Nazarbayev University provides access to four major databases of that type, ACM Digital Library, IEEE Xplore, Scopus, and ScienceDirect.

The framework is developed specifically for an object manipulation task that involves the usage of POMDP to model the problem. Thus, the search expression that was used to collect the literature was *ALL (POMDP) AND ALL (object) AND ALL (manipulation)* , where *ALL* means that the terms may occur anywhere in the article and *AND* is an intersection operation. Table 1 shows specific queries for each database, which

Table 1: Search queries and results for each database utilized

Database	Search query	Number of papers
ACM Digital Library	[All: pomdp] AND [All: object] AND [All: manipulation]	100
IEEE Xplore	("All Metadata":POMDP) AND ("All Metadata":Object) AND ("All Metadata":Manipulation)	10
Scopus	ALL (pomdp) AND ALL (object) AND ALL (manipulation)	249
ScienceDirect	POMDP object manipulation	94
Total		453

differs slightly for each one. It also shows the output of each database on these queries in several papers. However, this amount will be different for future searches since new papers will come up. Thus, it is necessary to bring the first exclusion criteria, which is years of publication. The articles published in the current year should be excluded since new works can get published this year. In addition, it was decided to remove papers published before 2018. ROS Melodic, on top of which the framework works, was released in 2018. It means that the systems developed before that year are not supported by it and do not need to be considered. Then, only journal articles and conference papers are considered to be included as the next filtering step. Other types of publications like books and review papers summarize the state of the topic. The results of the exclusion by both criteria are shown in table 2.

Table 2: Search results after filtering by year and publication type

Database	Year criteria	Publication type criteria
ACM Digital Library	51	26
IEEE Xplore	3	3
Scopus	131	121
ScienceDirect	34	22
Total	219	172

The next step is to exclude papers that are not relevant to the topic. This process was held in three steps. First, the articles with nonrelevant titles were excluded. Titles should include terms related to either general robotic tasks or object manipulation tasks. In addition, titles that state that the paper is a review are also excluded. POMDP is not necessary to be mentioned in the title since it might be the method utilized but not the focus of the project. Subsequently, the abstracts of the remaining papers were scanned, and those that focus on the development of manipulation tasks with the use of POMDP were selected. The abstract should include the information that manipulator robots are utilized and that it functions in an uncertain environment. The uncertain environment should be defined explicitly or with the use of POMDP. Other types of MDPs are excluded as well, including the Mixed observability Markov decision process (MOMDP). These exclusion criteria can be summarized as relevant to the topic of the thesis project. Finally, the content of those was read, and the non-relevant ones were removed. The papers that do not show empirical results are excluded since they do not bring value to the development of the framework.

Table 3: Number of relevant papers based on title, abstract and content

Database	Title	Abstract	Full text
ACM Digital Library	10	2	1
IEEE Xplore	3	3	3
Scopus	84	45	11
ScienceDirect	14	7	4
Total	111	57	19

Lastly, the duplicates should be removed. Some of the articles are stored in several databases. This results in a total of **15 papers** as references for the literature review. Table 4 shows the mapping from the selected works and the databases.

Table 4: Selected works for the literature review

Database	Article
ACM Digital Library	[7]
IEEE Xplore	[8–10]
Scopus	[8–18]
ScienceDirect	[11, 19–21]

3.2 Core parts

It is necessary to figure out the pattern in the systems from the selected works. These system similarities are the essential parts that would form the core of the framework. Those will be *frozen spots* of the framework that were mentioned in chapter 2. However, it is hard to find similarities between all of the papers. These works have similar commands that are controlled by different planners. These actions are the basic grasping commands, including the movement to certain coordinates with different orientations possibilities and controlling the gripper's fingers for grasping and releasing the object.

In addition, all the papers use some sort of movement restriction for the robots. It either optimizes the movement of the robots or defines collision objects for the arms to avoid. [16], for example, sets the height constraint for the movement of the robot to avoid inefficient lifting and the consistent model for further comparisons. [19], on the other hand, introduce the objects in the environment to the robot for collision avoidance purposes.

3.3 Additional parts

Even though all the systems utilize the points to approach, they handle it differently. [11], for instance, have predefined states between which the robotic arm operates. Others do not specify a location and let the planner decide on the position and orientation

of the approach, like in [10]. Both methods are viable and dependent on the needs of the system.

Table 5: Summary of differences in the selected literature

Paper	Goal handling	Robot Used	Objects Manipulated	POMDP solver
[7]	Position	Kinova MICO	Cylindrical objects	trust-POMDP
[8]	Position	Kinova MICO	Stapler	MICAH (proposed)
[9]	States	ROBOTIQ 2-finger	Cylindrical objects	Hierarchical belief tree search (proposed)
[10]	Position	fetch Robot	YCB Benchmarks objects [22]	PA-POMCP (proposed)
[11]	States	ROBOTIQ 2-finger	Cylindrical objects	Hierarchical POMDP (proposed)
[12]	Position	fetch Robot	COCO [23] and VMRD [24] objects	INVIGORATE (proposed)
[13]	Position	FRANKA robot	RAM stick	visual RL (proposed)
[14]	Position	Kinova MOVO	Not specified	Object-Oriented POMDP (proposed)
[15]	Position	Kinova MICO	G3DB objects [25]	DESPOT
[16]	Position	7 DoF Kuka LWR	Cubic objects	LPPGI (proposed)
[17]	States	7 DOF Barrett WAM	Single cubic object	ConCERRT (proposed)
[18]	Position	Kinova MICO	Canteen, block and cup	QMDP approximation
[19]	States	7 DoF Kuka LWR	Cylindrical objects	LoOP (proposed)
[20]	Position	Kinova Jaco	Cubic objects	POMCP
[21]	States	6 DoF Kuka arm	Cylindrical objects	LfD (proposed)

Another difference between papers is the variety of setups. It includes the various robotic platforms, objects for grasping, and finally, environments. [11] tests its algorithm

on ROBOTIQ 2-finger gripper robot to grasp cylindrical objects, while [8] utilizes Kinova MICO to place a stapler on top of the book stack. These two are just examples, and there are other robots used in the remaining papers.

Lastly, the works utilize different POMDP solving tools, including online and offline solvers. These differ depending on the focus of the work. The article can propose a new solver, in which case it will utilize its model [11]. On the other hand, the paper can utilize popular existing solvers like DESPOT [15]. Table 5 summarizes the differences between selected works.

Chapter 4 - Framework Design

The design process of the framework follows the protocol from [26]. The first in this protocol is *design planning*. First, the scope should be defined. For the developed framework, the scope is defined in the title of the thesis project. It is a simulation framework for POMDP-based autonomous object grasping with a mobile manipulator. This excludes any other modeling concepts like MDP and MOMDP.

Then, all of the requirements for the software should be stated clearly. From the literature review, sections 3.2 and 3.3 define the features that should be available in the framework. For the *frozen spots* of the framework, there should be functions that handle the movement and grasping actions, and ones that handle the movement restriction appendage. For the *hot spots* there should be an ability to control the movement of the robot either through predefined states or the position and orientation values. Then, the framework should have a feature of changing the environment, robot, and objects. Finally, there should be a possibility to connect a POMDP solver that could control the arm. However, it is important to emphasise that the framework provides an interface for the POMDP. It does not aim to develop POMDP and model and solve it.

4.1 Technology Choice

After the definition of the scope and features, it is necessary to choose the technologies that will be used to develop the framework. Since this thesis is a part of the ALARIS project, there are some project requirements predefined by ALARIS members.

4.1.1 System

The project is built on top of ROS, which is a de-facto standard in the robotics research area [27–31]. ROS is an open-source Robot Operating System that was initially developed by Ph.D. students at Stanford University [32]. Despite the name, it is not an operating system in the conventional meaning of this term [33]. It is rather a collection of software libraries, called *ROS packages*, for robotics software development. Other than package management, it provides far more features as an ecosystem, including communication of different processes on one or more machines. The process is often referred to as *ROS nodes*. Communication is a core feature that allows robots to communicate in the same environment and tell each other about their current state. There are different distributions of ROS. ROS updates usually come with major changes in functionalities and package compatibility. Everything in the project is set and used on top of Ubuntu 18.04 LTS distribution. For this Ubuntu version, the compatible ROS distribution is Melodic. Thus, the framework is built on top of Ubuntu 18.04 and ROS Melodic.

4.1.2 Simulation Environment

The framework will be built on top of some existing simulators. Hence, it is necessary to review options on the market. There are plenty of existing simulation environments when it comes to robotics tasks. There are proprietary simulators that target commercial organizations and research facilities. And there are open-source projects that are maintained by the community of regular users. Some of the popular projects are Marilou Robotics Studio, RobotBuilder, Webots, and Gazebo [34].

Marilou Robotics Studio Marilou Robotics Studio belongs to the proprietary software group. It is a commercial simulator developed by anyCode. It has its physics engine that allows customizing the environment rules, such as removing friction and slips [35]. It has a large set of robots and sensors that can be customized with its modeler. In addition, it allows users to add their models through CAD-like instruments [34]. Marilou, however, does not upgrade frequently. The latest version is *anyKode Marilou 2010*, and it was updated on 20th November 2020. In addition, the community behind the simulator is small compared to the open-source projects. It means that there are fewer robotic platforms and plugins.

RobotBuilder On the other side, there are plenty of open-source simulators. RobotBuilder is a GUI-oriented software. It can be utilized to prototype robots with models and actions. The code for the physical robots is generated based on this configuration of parameters. It only supports Java and C++ programming languages. These features of the software bring some limitations to the development process.

Webots Webots, in contrast, is a fully configurable tool. Webots was developed by Cyberbotics Ltd. in 2004 as a commercial mobile robotics simulator [36]. However, currently, it is an open-source project with an optional subscription for the support services [37]. It is constantly updated and has a community behind it that supports the open-source project [37]. As a result, it has a variety of both built-in and add-on features developed by the community. It includes support of different programming languages and technical tools. In addition, it has direct compatibility with ROS. Nevertheless, the main focus of the simulator is mobile robots. There is much less support for other types of robots such as manipulators [37].

Gazebo Finally, there is Gazebo simulation software. It is a fully open-source simulator with the physics engine [27]. As a simulator, it allows the addition of robot models and environment construction. It is a de facto simulator when it comes to robotic tasks [27–31]. It is developed by the Open Robotics team, which also develops ROS. As a result, it is directly compatible with ROS [38]. It has all of the advantages of Webots, such as fully configurable physics, a large community, and a vast range of accessible technical tools. Moreover, it does not only focus on mobile robots. Thus, because of the clear advantages of Gazebo over other simulation environments, the framework utilizes Gazebo as a simulation environment.

4.1.3 Default Functionality

The final step is a choice of the default functionality of the framework. This part of the system could be then changed or extended by the user. In the case of the developed framework, it is a simulation world, robot, manipulated objects, and a POMDP algorithm. Thus, these parts can be chosen arbitrarily but should be compatible with the remaining system. The combination of the robots that are available in the laboratory was chosen to form a mobile manipulator. These are Clearpath Jackal mobile robot, Kinova Gen3 lite manipulator, and Intel RealSense camera. All three have several ROS packages for a Melodic distribution. For the simulation environment, the office world by Clearpath Robotics was chosen. For the manipulated objects the YCB benchmark dataset that was mentioned in [9, 11] were selected because they were specifically designed to standardize the grasping task.

Finally, DESPOT as the POMDP planning algorithm was chosen that was mentioned in [7, 10, 11, 15, 20]. It is a relatively popular algorithm with ready-to-use open-source code that is compatible with ROS Melodic. In addition, last year, I have already

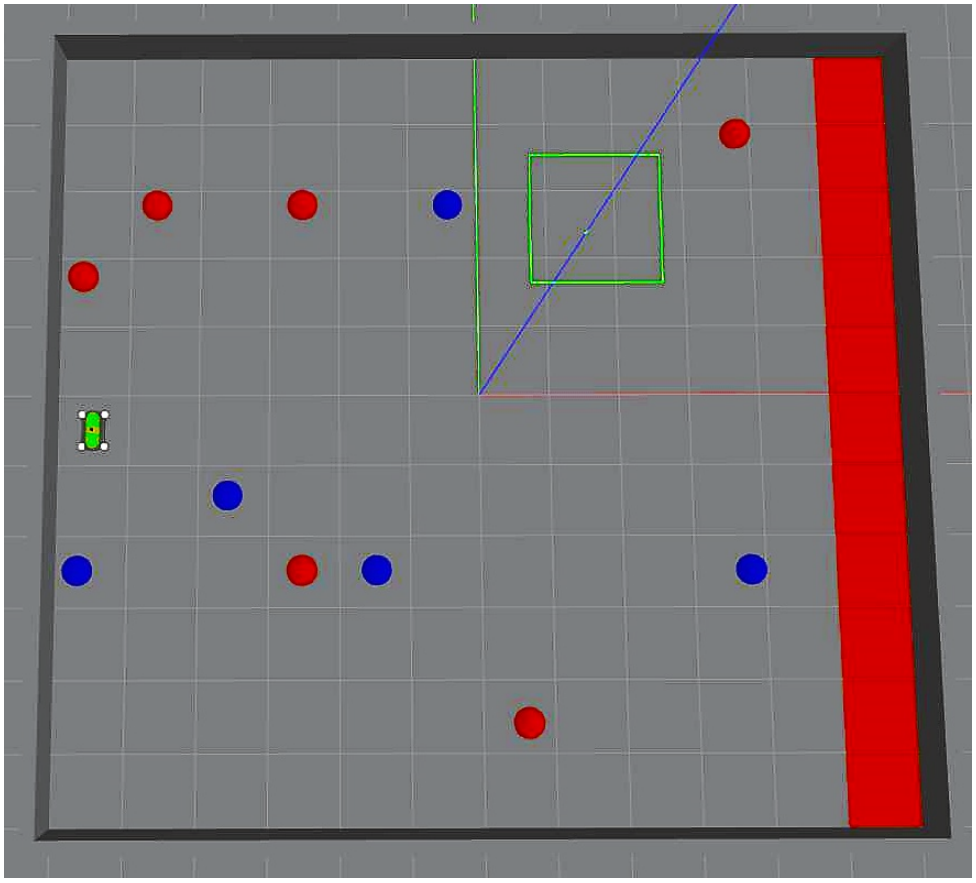

Figure 2: Gazebo interface created for a rock-sample task

developed an interface for another work that involved a POMDP task. It was a master thesis of a current Ph.D. student, where I implemented a Gazebo interface for a rock-sample task [39]. The task involved the movement of the mobile robot in the 7x11 grid world, where each square could have rocks to sample. Figure 2 demonstrates this Gazebo interface. The interface properly interacted with the DESPOT planner through ROS services. Table 6 shows the result of the DESPOT against POMCP testing in the developed interface. Thus, it is necessary to develop ROS services for the movement and grasping actions for compatibility with DESPOT.

Table 6: Results of the POMDP planners used in the developed interface

Planner	Total Discounted Reward
DESPOT	18.604
POMCP	21.652

The final design of the framework is shown in figure 3. The frozen spots of the framework are ROS services that provide functionality and the *MoveIt* package, which backs the *Move* service. Everything else, including spawned models and POMDP solver, can be replaced by the user in the code.

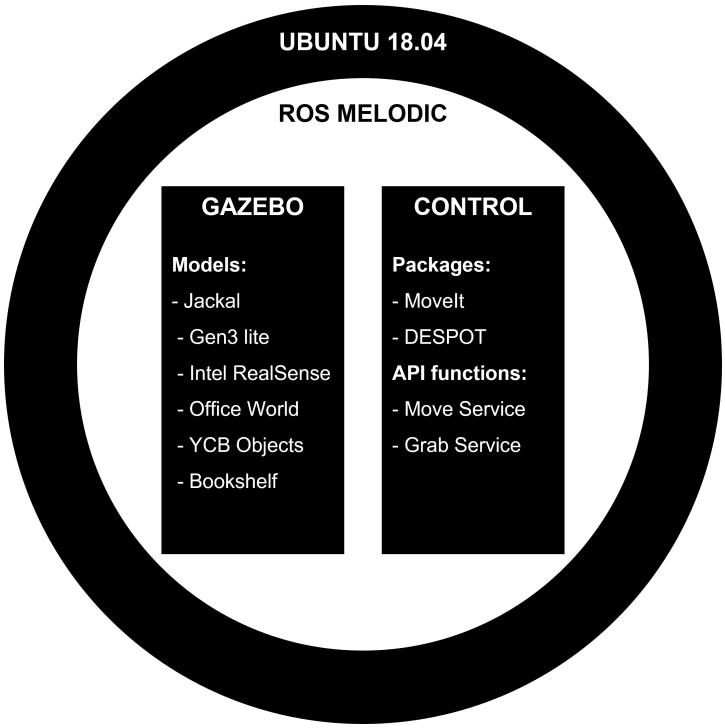

Figure 3: Framework design structure

Chapter 5 - Framework Implementation

In this section, the setup of the environment and the configuration of the framework files are described. The explanation is written in the form of instructions, so the process could be repeated. It is important that to replicate the steps provided below, Ubuntu 18.04 is required.

First, it is necessary to install the ROS Melodic distribution. The full desktop version should be installed since it comes with the Gazebo simulation environment which is a part of the technical stack of the framework. ROS can be installed with the following commands:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu  $(
lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc
| sudo apt-key add -
$ sudo apt install ros-melodic-desktop-full
```

5.1 Installing the Dependencies

The next is setting up the project. This includes downloading and installing necessary ROS packages. To begin with, it is necessary to create the workspace for the project using the following command:

```
$ mkdir -p catkin_workspace/src
```

The *src* folder will include all of the ROS packages that are utilized in the project. First, install the required ROS packages inside *src* folder by navigating into it and running

the following commands:

```
$ git clone https://github.com/clearpathrobotics/cpr_gazebo.git

$ cd cpr_gazebo

$ git checkout melodic_devel

$ cd ..

$ git clone https://github.com/okabedev/ycb_gazebo_sdf.git

$ git clone https://github.com/Kinovarobotics/ros_kortex.git

$ cd ros_kortex

$ git checkout melodic_devel

$ cd ..

$ git clone https://github.com/issaiass/realsense_gazebo_plugin.git

$ git clone https://github.com/issaiass/realsense2_description.git

$ git clone https://github.com/jackal/jackal_simulator.git

$ cd jackal_simulator

$ git checkout melodic_devel

$ cd ..

$ git clone https://github.com/jackal/jackal.git -b melodic-devel

$ git clone https://github.com/jackal/jackal_manipulation.git --recursive
```

All the packages are downloaded into the workspace. These are the packages for the Kinova Gen3 lite, Intel RealSense camera, Jackal mobile robot, office environment, and YCB objects dataset. The dependencies for the download packages should be installed then. It is done by navigating into the workspace folder and running the following command:

```
$ rosdep install --from-paths src --ignore-src -y
```

Kinova Gen3 lite manipulator's ROS package is called *ros_kortex*, which includes the simulation model, the driver to control the robot, and examples codes. Then, there are two packages for Intel RealSense camera. *realsense2_description* is a ROS package that includes the SDF model of the camera. *realsense_gazebo_plugin* is a package that allows the model to interact with the environment around it. The Jackal mobile robot requires three packages to be installed. The latter one is utilized to create an OctoMap, which is a 3D map of the environment [40]. Those include *jackal*, *jackal_simulator*, and *jackal_manipulator*. The first package is the main one, which includes the control scripts and the description of the simulation model. The second one provides the integration of the Jackal model into the Gazebo simulator. Finally, *jackal_manipulator* allows connecting Jackal with the manipulator robot, which is Kinova Gen3 lite in this project.

The *jackal_manipulator* has a built-in module for connecting Kinova arms to the Jackal robot. However, the description of this model should be included in the model file of the Jackal. It is needed for the mobile manipulator to appear in the simulation environment. Furthermore, the Intel RealSense camera needs to be attached to this construction. As a result, the model files *jackal.urdf.xacro* should be changed to include the model of the camera. Thus, the following code was inserted in that file:

```
<xacro:property
  name="robotFileName"
  value="jackal_gen3_lite_description.urdf.xacro"
/>
```

```
<xacro:property

  name="cameraFileName"

  value="_d435.urdf.xacro"

/>



<xacro:include

  filename="$(find jackal_kinova_description)/urdf/$robotFileName"

/>



<xacro:include

  filename="$(find realsense2_description)/urdf/$cameraFileName"

/>



<sensor_d435 parent="kinova_arm_end_effector_link">

  <origin xyz="0.10 0 0" rpy="-3.1415 -1.57 0" />

</sensor_d435>
```

robotFileName is the file that includes the mobile manipulator model from the *jackal_manipulator* package. *cameraFileName* is the file that includes the model of Intel RealSense camera model. The final three lines are attaching the camera to the end-effector link of the manipulator. It has an offset of 0.1 above the arm. The orientation values are toggled so that the camera is directed the same way as the manipulator.

Lastly, a DESPOT planner should be installed. It is not necessary to install it in the *src* folder since it is not a ROS package. To install it run the following commands:

```
$ git clone https://github.com/AdaCompNUS/despot.git

$ cd despot

$ make
```

5.2 Building Environment

The newly created files should also be stored in a separate ROS package. It can be created by running:

```
$ catkin_create_pkg gen3lite_pickplace std_msgs rospy roscpp
```

where *gen3lite_pickplace* is the name of the package and the parameters after it is the dependencies of the package. In this package several folders should be created for the *launch, script* and *model* files. The information regarding the structure of the ROS projects can be found on their official website.

First, the launch file that creates the gazebo world and spawns the robot into it should be created. This file has the following content:

```
<launch>

  <param name="office_geom" command="$(find xacro)/xacro --inorder

  '$(find cpr_office_gazebo)/urdf/office_geometry.urdf.xacro'" />



  <arg name="use_sim_time" default="true" />

  <arg name="gui" default="true" />

  <arg name="headless" default="false" />



  <arg name="front_laser" default="true" />
```

```
<arg name="default_config" value="front_laser"/>

<arg name="default_config" value="base" unless="$(arg front_laser)" />


<!-- Configuration of Jackal which you would like to simulate.

     See jackal_description for details. -->

<arg name="config" default="$(arg default_config)" />


<!-- Optionally enable teleop for the simulation -->

<arg name="joystick" default="false" />


<!-- Launch Gazebo with the specified world -->

<arg name="world_name"

default="$(find cpr_office_gazebo)/worlds/actually_empty_world.world"/>


<include file="$(find gazebo_ros)/launch/empty_world.launch">

  <arg name="debug" value="0" />

  <arg name="gui" value="$(arg gui)" />

  <arg name="use_sim_time" value="$(arg use_sim_time)" />

  <arg name="headless" value="$(arg headless)" />

  <arg name="world_name" value="$(arg world_name)" />

</include>


<node name="office_world_spawner" pkg="gazebo_ros" type="spawn_model"

args="-urdf
```

```
  -model office_geom

  -param office_geom

  -x 0 -y 0 -z 0 -Y 0" />



  <!-- Spawn Jackal -->

  <include file="$(find jackal_gazebo)/launch/spawn_jackal.launch">

    <arg name="x" value="0" />

    <arg name="y" value="0" />

    <arg name="z" value="1.0" />

    <arg name="yaw" value="0" />

    <arg name="config" value="$(arg config)" />

    <arg name="joystick" value="$(arg joystick)" />

  </include>



</launch>
```

To spawn different objects into the gazebo world, the *spawn_model* script from the *gazebo_ros* package is utilized. Since this script will be used multiple times, it was put into a separate *roslaunch* file. *roslaunch* files are used to run ros nodes with arguments without a need to explicitly write shell commands. The file has the following content:

```
<launch>

  <arg name="robot_name" default=""/>

  <arg name="x" default="0.0"/>

  <arg name="y" default="0.0"/>
```

```
  <arg name="z" default="1.0"/>

  <arg name="roll" default="0"/>

  <arg name="pitch" default="0"/>

  <arg name="yaw" default="0.0"/>

  <arg name="sdf_robot_file" default=""/>

  <node name="$(arg robot_name)_spawn_urdf" pkg="gazebo_ros"

              type="spawn_model" respawn="false"

              output="screen"

              args="-file $(arg sdf_robot_file) -sdf

              -x $(arg x) -y $(arg y) -z $(arg z)

              -R $(arg roll) -P $(arg pitch) -Y $(arg yaw)

              -model $(arg robot_name)"/>

</launch>
```

In this code, *x, y, z* are the coordinates of the model, *roll, pitch, yaw* define the orientation of the model, *robot_name* is the name used in the simulation, and *sdf_robot_file* is the path to the model's file.

The initial task requires the pick and place functionality. Hence, there should be a place where the objects are held and where they should be transported. For this purpose, the models that come with the gazebo are utilized. Those models are called *cabinet* and *bookshelf*. To spawn those with the use of the above's code, the file with the following content is run:

```
<launch>

    <include file="$(find gen3lite_pickplace)/launch/spawn_sdf.launch">
```

```
        <arg name="robot_name" value="bookshelf"/>

        <arg name="x" value="0.0"/>

        <arg name="y" value="0.7"/>

        <arg name="z" value="0.0"/>

        <arg name="roll" value="0"/>

        <arg name="pitch" value="0"/>

        <arg name="yaw" value="0.0"/>

        <arg name="sdf_robot_file"

        value="$(find gen3lite_pickplace)/models/bookshelf/model.sdf"/>

    </include>

    <!--Similar for cabinet model-->

</launch>
```

It was an example of spawn code usage; other objects are spawned similarly (name the file for them as *spawn_objects_to_pick.launch*). The file is saved as *spawn_table.launch* and can be run through:

```
$ roslaunch gen3lite_pickplace spawn_table.launch
```

5.3 Adding Control Functionality

To add the control functionality, the ROS services were utilized. It is a concept in the ROS ecosystem that allows RPC request/reply interactions. It is also useful, since the DESPOT framework is specifically defined to execute actions through ROS services. Two separate services were created for moving the robot and for grabbing objects. For each of those, the type files that indicate the message format exchanged between service

and client were defined. Message for the *grab* service has the following form:

```
string targetName

---

int8 grabbed
```

where the *targetName* is the name of the file that is grabbed by the gripper and *grabbed* is a value that indicates the success of the action. In addition, the *grab* service also has a functionality of the release, and the release message has the following form:

```
---

int8 released
```

It does not need the object name since it either is holding an object or not. The *released* variable indicates whether the arm released the object or it was not holding anything. Next, the *move* service has the following message format:

```
geometry_msgs/Point offset

int8 state

float32 close

---

int8 success
```

This format was designed to handle both state and position movement. It can handle both positional movement and movement to the predefined states as it was discussed in chapter 4. *close* variable is needed to handle the gripper at the start of the motion so it could grab the target object.

The code for both services is available in the Github repository of the framework (https://github.com/dabarov/thesis-project). It is not pasted here for convenience. It

launches the script for grabbing the *box_to_hand.py* if the message of type *Grab* is received and kills it if *Release* message is received. Gazebo grippers have a bag during the collision with an object when it starts to shake and can lose the object. Thus, it was decided to imitate the grabbing by constantly changing the position of the object by the initial offset from the gripper. This logic is implemented in the aforementioned script *box_to_hand.py*, where it saves the initial offset when *Grab* message is received, changes it based on the position and orientation of the gripper, and stops only when *Release* message is received.

Chapter 6 - Framework Testing

6.1 Testing Simulation Environment

The simulation can be run through several commands since the framework is ready to use. To launch the world with the robot in it, run the following command:

```
$ roslaunch gen3lite_pickplace office_world_spawner.launch
```

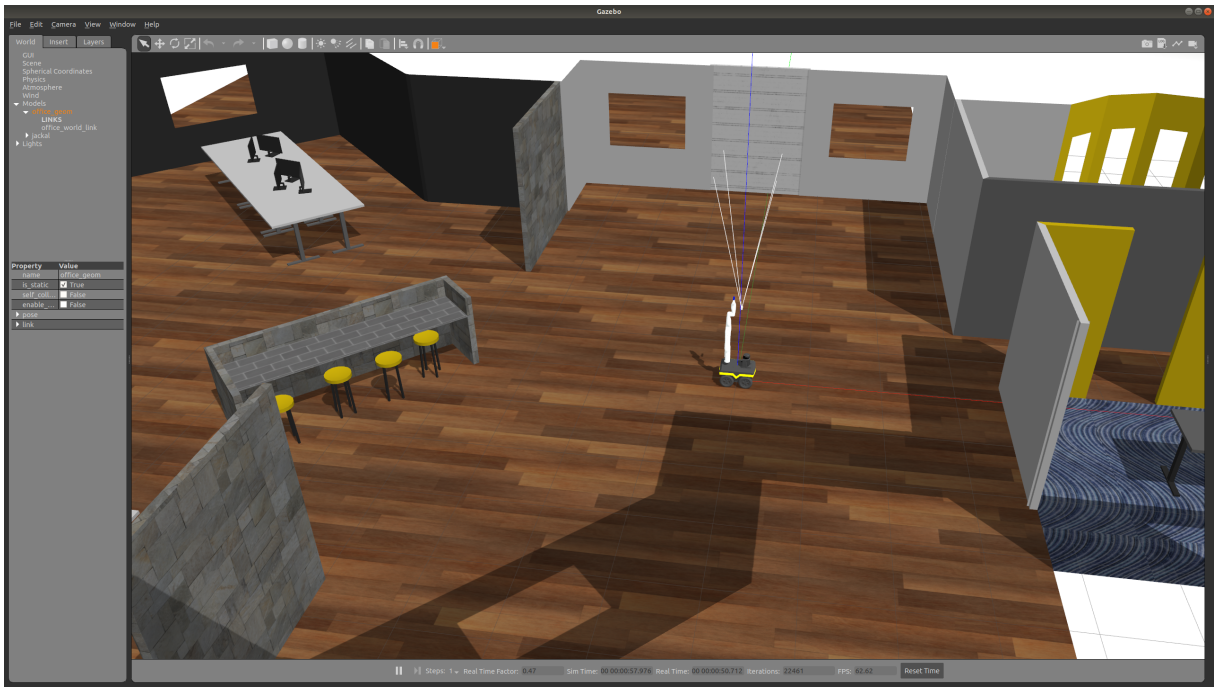

Figure 4: The office world with robot in Gazebo

Then it is necessary to launch the ros_kortex driver to control the robotic arm.

```
$ roslaunch kortex_driver kortex_driver.launch
```

Next step is to spawn table and objects to manipulate. To accomplish this run:

```
$ roslaunch gen3lite_pickplace spawn_table.launch

$ roslaunch gen3lite_pickplace spawn_objects_to_pick.launch
```

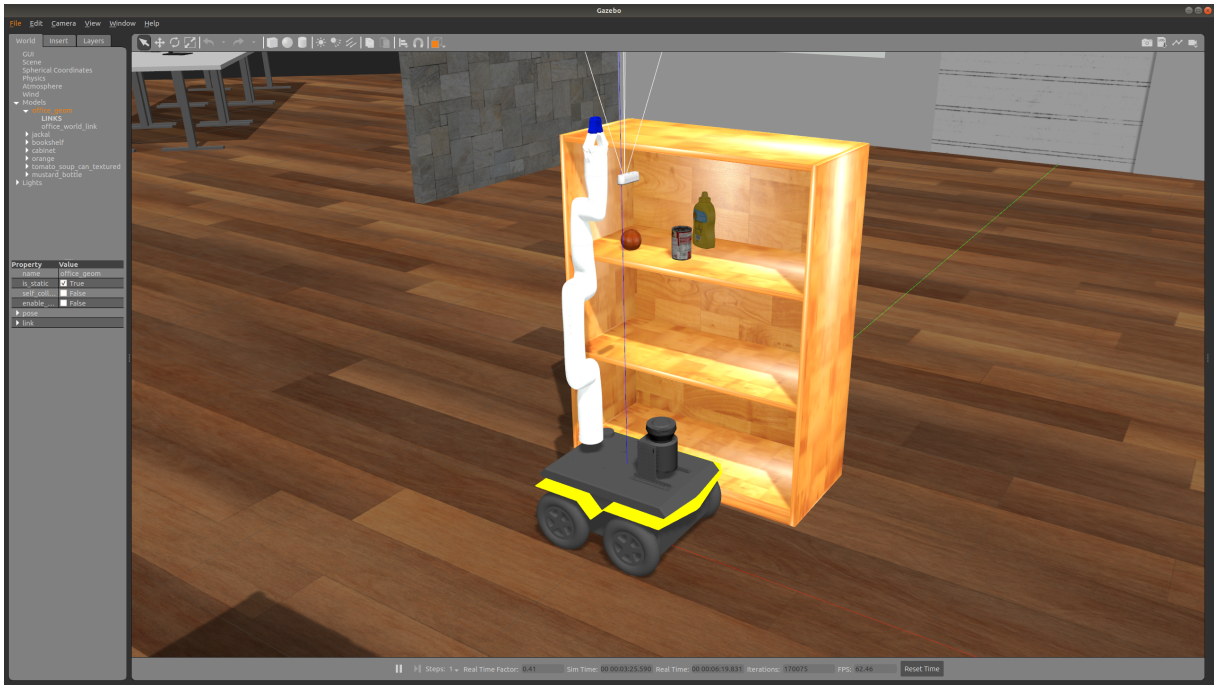

Figure 5: The Gazebo world after spawning bookshelf and YCB objects

6.2 Testing Control Services

After running the commands above, the Gazebo environment that is shown in Figure 1 should be created. This means that the environment is ready for testing of the services for movement and grabbing the objects. The services can be launched using the following commands:

```
$ rosrun gen3lite_pickplace grab_server.py

$ rosrun gen3lite_pickplace move_server.py
```

If you send the following command, the arm will move in the y-direction for 0.1 units after opening the gripper for 90%. It will ignore the state field because of the -1 value.

```
$ rosservice call /kinova_moveit/move "offset:

  x: 0.0

  y: 0.1

  z: 0.0

state: -1

close: 0.9"
```

The state commands can be utilized as well. There are few predefined states included in the framework. Figure 6 shows the robot in the home state.

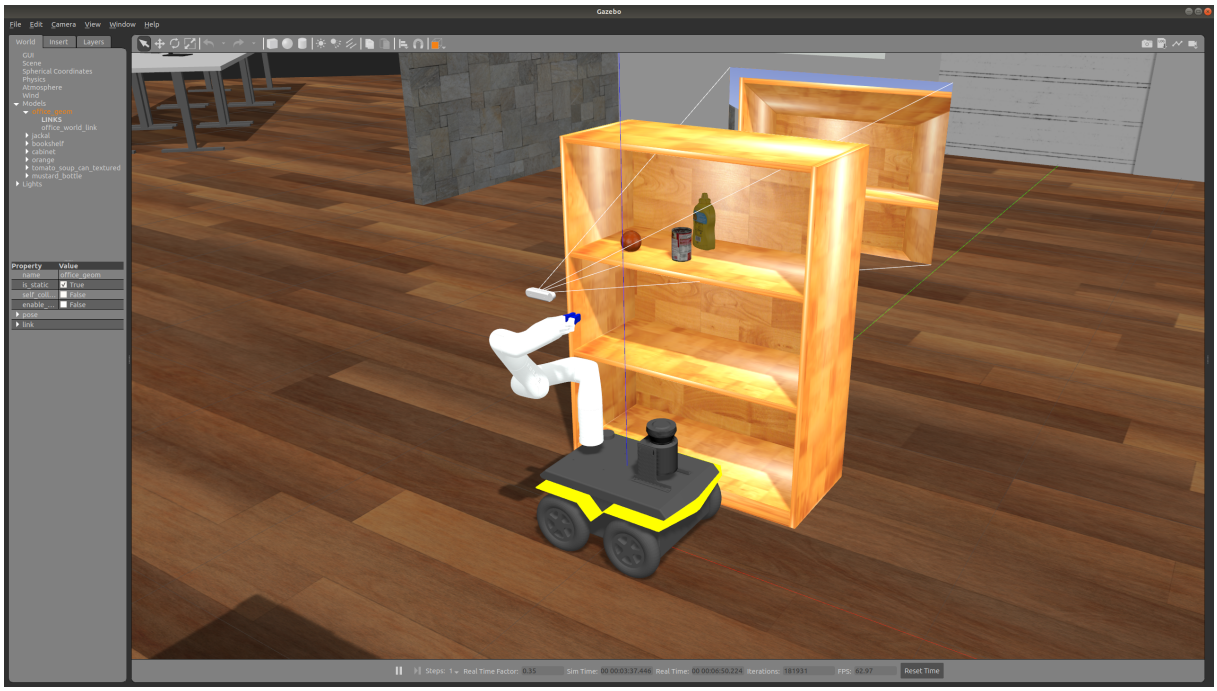

Figure 6: The robot in the home state

To grab spawned *orange* object, the following snippet can be used:

```
$ rosservice call /kinova_moveit/grab "targetName: 'orange'"
```

It can be seen that the offset is held during the movement of the arm with the use of the move commands. To release the object this command should be utilized:

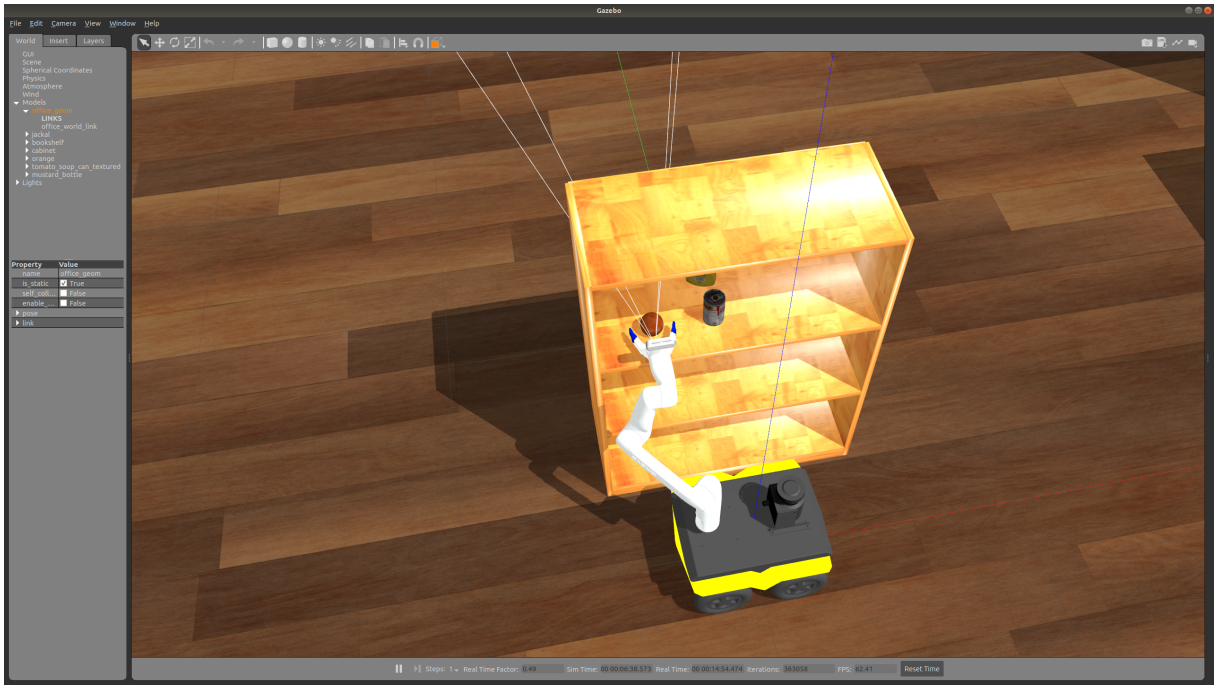

Figure 7: The state in front of the orange object

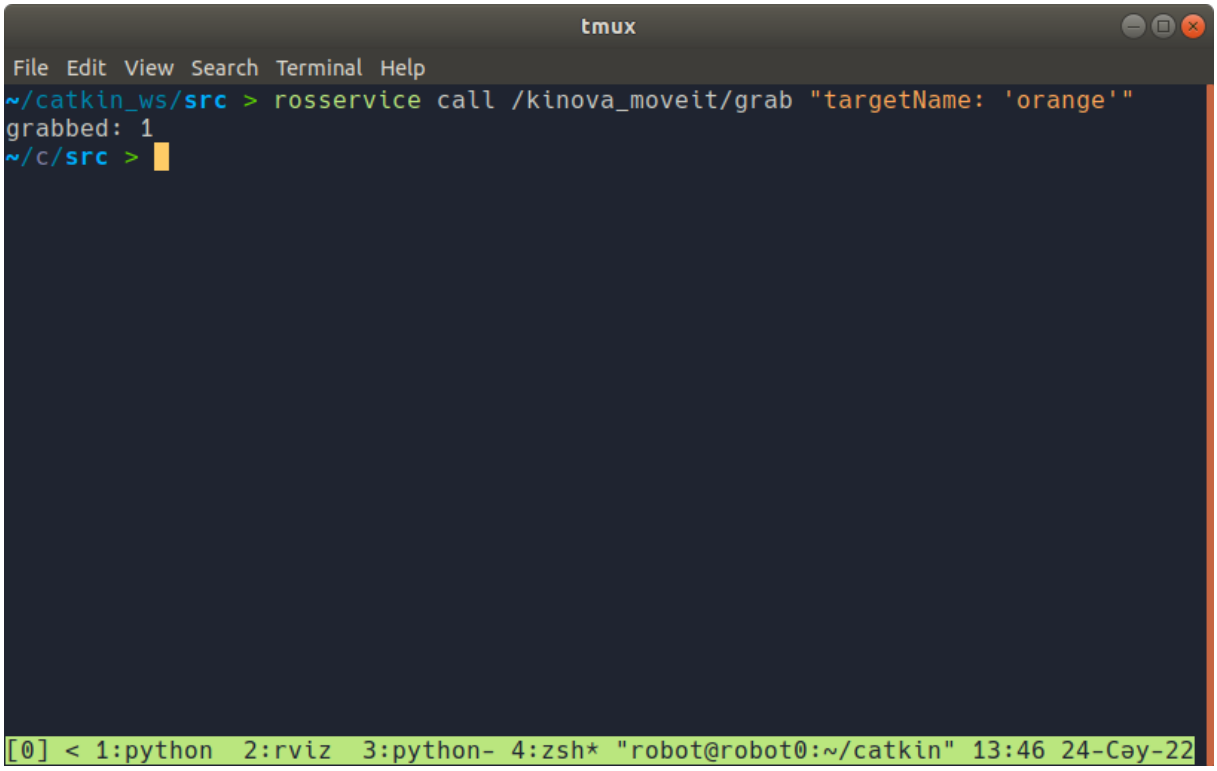

Figure 8: The example of calling the grab service from therminal

```
$ rosservice call /kinova_moveit/release "{}"
```

The sequence of an object grasping can be seen in Figures 6, 7 and 9. In figure

6 is in initial *Home* state. Then, it navigates with an open gripper to the target object, which is orange in this case (figure 7). Then the grab service is called as in figure 8, which attaches the object to the gripper by maintaining the offset. Finally, the object is lifted and transported to another position as in figure 9. The sequence takes 1 minute and 23 seconds in real-time, which is reasonable for the experiment.

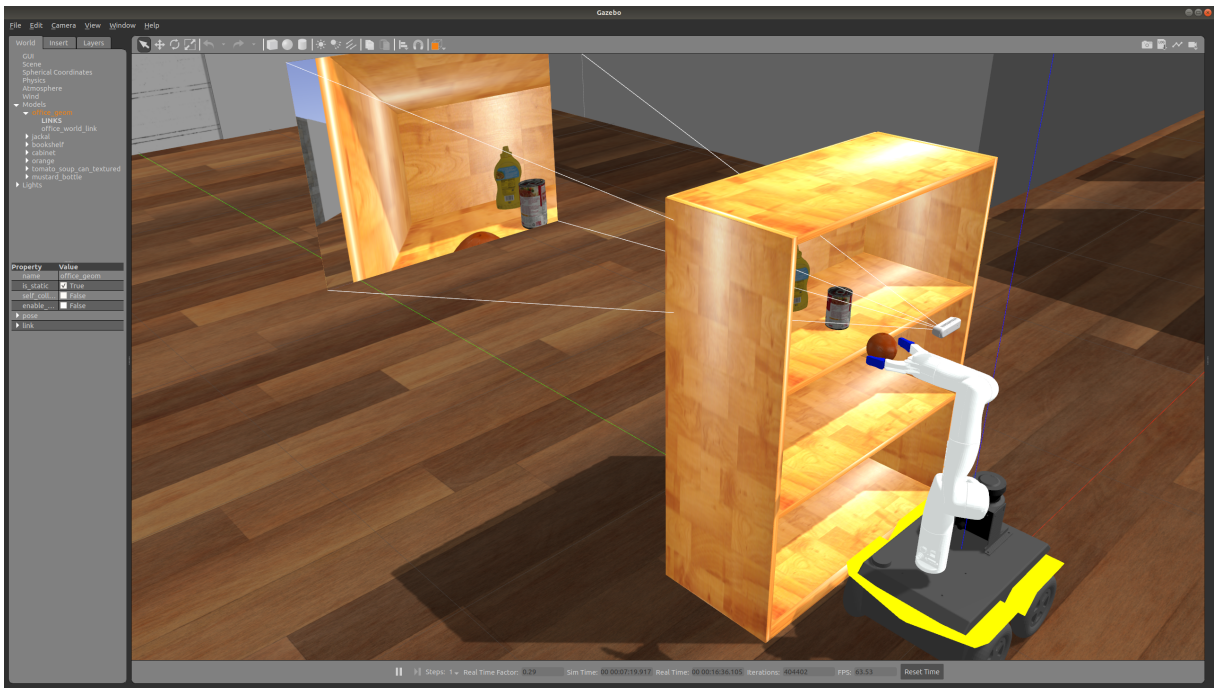

Figure 9: The orange object keeping the offset after grab command

Chapter 7 - Conclusion

7.1 Summary

The simulation framework for mobile manipulators that involve POMDP was created in this thesis. To accomplish this, the required features for the end product were identified. It was done through the analysis of related works, which were collected using the systematic literature review protocol. In addition, the needs of the ALARIS laboratory team dictated some decisions during the design process since it is a part of the laboratory project. The development and testing processes of the framework are written in the form of a tutorial so that they could be replicated. During the testing phase, the framework showed that it accomplishes to meets the requirements.

7.2 Future Work

There are several points to extend the project further. This thesis only used the technologies needed for the ALARIS project during the testing phase. To completely verify the viability of the framework it could be tested on different setups. Some robots that are available in the lab for testing are the Kinova Jaco manipulator, Jaguar 4x4 mobile robot, and 4 DOF manipulator built with Hebi Robotics' servo motors.

Then, this framework will be utilized with the POMDP task. The POMDP test was not done in this project since it was not included in the scope of the thesis. The development of the POMDP model is a time-consuming task. Thus, it can be a long-term extension of the project.

Final point is that the ROS Melodic will lose its support from Open Robotics in May 2023. This will lead to the point that some packages used in the project become

obsolete. Thus, it would be better to port the framework to newer versions of ROS.

References

[1] M. Naazare, F. G. Rosas, and D. Schulz, "Online next-best-view planner for 3d-exploration and inspection with a mobile manipulator robot," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3779–3786, 2022.

[2] S. S, "A study of software development life cycle process models," *SSRN Electronic Journal*, 2017. [Online]. Available: https://doi.org/10.2139/ssrn.2988291

[3] N. M. Edwin, "Software frameworks, architectural and design patterns," *Journal of Software Engineering and Applications*, vol. 07, no. 08, pp. 670–678, 2014. [Online]. Available: https://doi.org/10.4236/jsea.2014.78061

[4] M. Iglesias, "Using your library with react," in *Pro D3.js*. Apress, 2019, pp. 201–218. [Online]. Available: https://doi.org/10.1007/978-1-4842-5203-1_12

[5] T. F. Iskandar, M. Lubis, T. F. Kusumasari, and A. R. Lubis, "Comparison between client-side and server-side rendering in the web development," *IOP Conference Series: Materials Science and Engineering*, vol. 801, no. 1, p. 012136, May 2020. [Online]. Available: https://doi.org/10.1088/1757-899x/801/1/012136

[6] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele Univ.*, vol. 33, 08 2004.

[7] M. Chen, S. Nikolaidis, H. Soh, D. Hsu, and S. Srinivasa, "Trust-aware decision making for human-robot collaboration," *ACM Transactions on Human-Robot Interaction*, vol. 9, no. 2, pp. 1–23, Feb. 2020. [Online]. Available: https://doi.org/10.1145/3359616

[8] A. Jain and S. Niekum, "Learning hybrid object kinematics for efficient hierarchical planning under uncertainty," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2020. [Online]. Available: https://doi.org/10.1109/iros45743.2020.9340749

[9] W. Zhao and W. Chen, "Efficient planning for object search task based on hierarchical POMDP," in *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, Dec. 2021. [Online]. Available: https://doi.org/10.1109/robio54168.2021.9739277

[10] Y. Xiao, S. Katt, A. ten Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2019. [Online]. Available: https://doi.org/10.1109/icra.2019.8793494

[11] W. Zhao and W. Chen, "Hierarchical POMDP planning for object manipulation in clutter," *Robotics and Autonomous Systems*, vol. 139, p. 103736, May 2021. [Online]. Available: https://doi.org/10.1016/j.robot.2021.103736

[12] H. Zhang, Y. Lu, C. Yu, D. Hsu, X. Lan, and N. Zheng, "INVIGORATE: Interactive visual grounding and grasping in clutter," in *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, Jul. 2021. [Online]. Available: https://doi.org/10.15607/rss.2021.xvii.020

[13] Y. Shi, Z. Chen, H. Liu, S. Riedel, C. Gao, Q. Feng, J. Deng, and J. Zhang, "Proactive action visual residual reinforcement learning for contact-rich tasks using a torque-controlled robot," in *2021 IEEE International Conference*

on Robotics and Automation (ICRA). IEEE, May 2021. [Online]. Available: https://doi.org/10.1109/icra48506.2021.9561162

[14] K. Zheng, Y. Sung, G. Konidaris, and S. Tellex, "Multi-resolution POMDP planning for multi-object search in 3d," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, Sep. 2021. [Online]. Available: https://doi.org/10.1109/iros51168.2021.9636737

[15] N. P. Garg, D. Hsu, and W. S. Lee, "Learning to grasp under uncertainty using POMDPs," in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, May 2019. [Online]. Available: https://doi.org/10.1109/icra.2019.8793818

[16] J. Hoelscher, D. Koert, J. Peters, and J. Pajarinen, "Utilizing human feedback in POMDP execution and specification," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids).* IEEE, Nov. 2018. [Online]. Available: https://doi.org/10.1109/humanoids.2018.8625022

[17] E. Pall, A. Sieverling, and O. Brock, "Contingent contact-based motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, Oct. 2018. [Online]. Available: https://doi.org/10.1109/iros.2018.8594365

[18] S. Javdani, H. Admoni, S. Pellegrinelli, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization for teleoperation and teaming," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 717–742, Jun. 2018. [Online]. Available: https://doi.org/10.1177/0278364918776060

[19] F. Riccio, R. Capobianco, and D. Nardi, "LoOP: Iterative learning for optimistic planning on robots," *Robotics and Autonomous Systems*, vol. 136, p. 103693, Feb. 2021. [Online]. Available: https://doi.org/10.1016/j.robot.2020.103693

[20] A. Hayashi, D. Ruiken, T. Hasegawa, and C. Goerick, "Reasoning about uncertain parameters and agent behaviors through encoded experiences and belief planning," *Artificial Intelligence*, vol. 280, p. 103228, Mar. 2020. [Online]. Available: https://doi.org/10.1016/j.artint.2019.103228

[21] C.-J. Liang, V. R. Kamat, and C. C. Menassa, "Teaching robots to perform quasi-repetitive construction tasks through human demonstration," *Automation in Construction*, vol. 120, p. 103370, Dec. 2020. [Online]. Available: https://doi.org/10.1016/j.autcon.2020.103370

[22] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[23] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014. [Online]. Available: https://arxiv.org/abs/1405.0312

[24] H. Zhang, X. Lan, X. Zhou, Z. Tian, Y. Zhang, and N. Zheng, "Visual manipulation relationship network for autonomous robotics," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 118–125.

[25] A. Kleinhans, B. Rosman, M. Michalik, B. Tripp, and R. Detry, "G3db: A database of successful and failed grasps with rgb-d images, point clouds, mesh models and gripper parameters," 2015. [Online]. Available: http://rgdoi.net/10.13140/RG.2.1.1350.3122

[26] A. Tang, A. Aleti, J. Burge, and H. van Vliet, "What makes software design effective?" *Design Studies*, vol. 31, no. 6, pp. 614–640, Nov. 2010. [Online]. Available: https://doi.org/10.1016/j.destud.2010.09.004

[27] S. Suvarna, D. Sengupta, P. Koratikere, and R. S. Pant, "Simulation of autonomous airship on ros-gazebo framework," in *2019 Fifth Indian Control Conference (ICC)*, 2019, pp. 237–241.

[28] P. Anggraeni, I. Rokhim, and R. M. Salam, "Design and development of multiple mobile manipulator robots using gazebo-ros," in *2020 International Conference on Applied Science and Technology (iCAST)*, 2020, pp. 672–676.

[29] K. Kumar, S. I. Azid, A. Fagiolini, and M. Cirrincione, "Erle-copter simulation using ros and gazebo," in *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, 2020, pp. 259–263.

[30] A. Sagitov and Y. Gerasimov, "Towards dji phantom 4 realistic simulation with gimbal and rc controller in ros/gazebo environment," in *2017 10th International Conference on Developments in eSystems Engineering (DeSE)*, 2017, pp. 262–266.

[31] Y. Kim, S.-y. Lee, and S. Lim, "Implementation of plc controller connected gazebo-ros to support iec 61131-3," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1195–1198.

[32] E. Ackerman and E. Guizzo, "Wizards of ros: Willow garage and the making of the robot operating system," Jun 2021. [Online]. Available: https://spectrum.ieee.org/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system

[33] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," *ICRA Workshop on Open Source Software*, vol. 3, pp. 1–6, 01 2009.

[34] A. Harris and J. M. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *2011 Proceedings of IEEE Southeastcon*, 2011, pp. 243–249.

[35] G. J. Yang, R. Delgado, and B. W. Choi, "A practical joint-space trajectory generation method based on convolution in real-time control," *International Journal of Advanced Robotic Systems*, vol. 13, no. 2, p. 56, 2016. [Online]. Available: https://doi.org/10.5772/62722

[36] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf

[37] Webots, "http://www.cyberbotics.com," open-source Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[38] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, "Manipulation task simulation using ros and gazebo," *2014 IEEE International Conference on Robotics and Biomimetics, IEEE ROBIO 2014*, 12 2014.

[39] S. Shaldambayeva, "High level robot motion planning using pomdp," Master's thesis, School of Engineering and Digital Sciences, 2021.

[40] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at https://octomap.github.io. [Online]. Available: https://octomap.github.io