DEVELOPMENT AND TESTING OF A MOBILE MANIPULATION ROBOT

by

JIANG Dechao

ZHANG Tong

B.Sc. in Electromechanical Engineering

2023/2024

Faculty of Science and Technology
University of Macau

This page intentionally left blank

Development and Testing of a Mobile Manipulation Robot

by

Jiang Dechao (D-C0-2802-1)
Zhang Tong (D-C0-2876-0)

Final Year Project Report submitted in partial
fulfillment of the requirements for the degree of

B.Sc. in Electromechanical Engineering

Faculty of Science and Technology
University of Macau

2023/2024

University of Macau

Abstract

Development and testing of a mobile manipulation robot

by Jiang Dechao (D-C0-2802-1)
Zhang Tong (D-C0-2876-0)

Project Supervisor
PhD, FASME, Professor Qingsong Xu

B.Sc. in Electromechanical Engineering

In this project, we present the development and testing of Libot, a mobile manipulation robot, designed specifically for automating tasks in library environments. Libot combines the functionalities of the UR5 robotic arm and MiR250 mobile platform, enhanced with a Realsense 435i depth camera, to autonomously navigate and interact within a library setting. The core of Libot's operation is a sophisticated book detection system using the YOLO v8 deep learning model, which recognizes book spine labels from RGB images to facilitate effective book cataloging. Additionally, the robot employs a Grasp Pose Detection (GPD) algorithm integrated with the MoveIt platform to optimize book handling through precise pick-and-place operations. Enhanced with Simultaneous Localization and Mapping (SLAM) techniques, Libot proficiently adapts to dynamic library layouts, improving its navigation and operational efficiency. The system architecture, built around the Robot Operating System (ROS), ensures robust performance in handling routine library tasks and emergent scenarios without disruptions. We tested various functions in the Gazebo environment, demonstrating the system's high 92.3% object detection accuracy, better grasping success rate and navigation capability. This study not only demonstrates the feasibility of deploying mobile manipulation robots in public service environments but also provides insights into the potential enhancements for interactive robotic systems in complex indoor spaces.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DOF	Degree of Freedom
GPD	Grasp Pose Detection
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IK	Inverse Kinematics
IMU	Inertial Measurement Unit
LLM	Large Language Model
MMR	Mobile Manipulation Robot
OS	Operation System
RAM	Random Access Memory
RFID	Radio Frequency Identification
RGB-D	Red Green Blue-Depth
ROS	Robot Operation System
RVIZ	ROS Visualizer
SLAM	Simultaneous Localization And Mapping Techniques
SURF	Speeded Up Robust Features
UR5	Universal Robot 5
URDF	Universal Robotic Description Format
XML	Extensible Markup Language

ACKNOWLEDGEMENTS

We extend our gratitude and appreciation to our supervisor, Professor Qingsong Xu. His support and dedication have been pivotal in the culmination of our final year project. We are immensely blessed to have experienced his mentorship and encouragement throughout the duration of this project.

We will always remember our Alma Maters:

University of Macau

Xi'an Jiaotong University and *University of Southern Denmark*

They not only enriched our knowledge but also forged the values of academic integrity and the pursuit of excellence.

Our thanks also go to Mr. Yuning Cao for his generous assistance in our research.

Lastly, we must acknowledge the unwavering support and encouragement of our families, friends, and loved ones. Their steadfast belief in our potential and continuous encouragement have fueled our ambitions and sustained us through challenging periods. We are immensely grateful for their patience, understanding, and unwavering support during our academic journey.

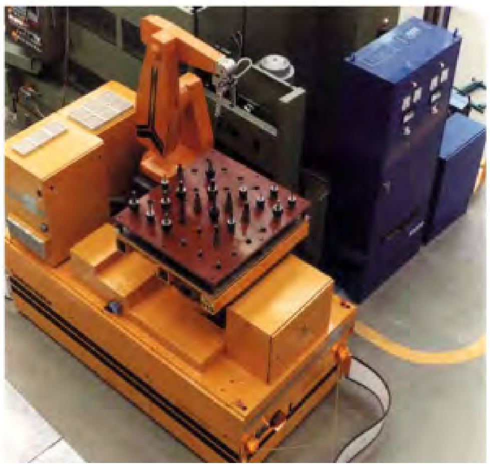
CHAPTER 1 : INTRODUCTION

1.1 RESEARCH BACKGROUND

Over the last sixty years, robotics has evolved from a novel engineering concept to a crucial component of modern industrial and service sectors. Initially confined to automotive manufacturing [1], robots are now integral in fields ranging from military operations and aerospace to logistics and public services. This broad adoption has spurred significant technological advancements, enabling robots to perform increasingly complex tasks. nature Mobile robots, which autonomously navigate and perform tasks in environments like hotels and hospitals, showcase the remarkable progress in this field. These robots began their journey in the late 1960s, with early models demonstrating basic navigation capabilities using algorithms like A* [2]. The 1980s saw the prototype of mobile manipulation robots (MMRs), which combined the mobility of these early robots with the ability to physically interact with their environment. For instance, the MORO robot, developed in 1984, could complete the loading and unloading of parts, whereas the MORO can only move along a buried wire in the floor [3]. This technology was furthered deeply in the 1990s through the Stanford Manipulation Group. Their team combined the PUMA 560 arm with a novel Powered Caster Vehicle, which gives the robot smooth, accurate motion and coordination integrating with an onboard manipulator [4].

Today, advancements in sensor technology [5] [6], miniaturization, and artificial intelligence [7] [8] have enabled MMRs to execute tasks that match or exceed human capabilities in precision and efficiency. These robots are now vital in managing complex, dynamic environments such as hospitals and libraries [9] [10] [11], where they adapt to new challenges in real-time.

(a) Shakeythe first Mobile Robot (Image credit: © Mark Richards. Courtesy of the Computer History Museum)



(b) MORO Robot [3]

(c) The Stanford Robotics Platform [4]

Figure 1.1: Robots in early era

1.2 INITIATIVE

In typical library operations, tasks such as cataloging, locating, and returning books are often repetitive and monotonous. Employing a library robot to handle these tasks can significantly alleviate the burden on human staff, freeing them to engage in more complex interactions with readers. This not only enhances the efficiency of library services but also addresses potential labor shortages by reallocating human resources to areas that require more human-nature skills and personal attention.

Building upon decades of robotics research, our project introduces "Libot" a Mobile Manipulation Robot (MMR) engineered specifically for library services. Libot embodies a significant leap in addressing complex tasks within public libraries, leveraging state-of-the-art robotics technologies encapsulated in the Robot Operating System (ROS). This system not only enhances the robot's operational efficiency but also integrates sophisticated object detection and deep learning methodologies to streamline book handling processes. Libot is design to have the capabilities of automating navigation to bookshelves, collecting returned books, and recognizing individual books through advanced vision systems. This initiative promises to revolutionize library services by improving accessibility and efficiency, providing a scalable model that could extend to other life scenarios where similar needs are evident.

1.3 RESEARCH OBJECTIVES AND METHODOLOGIES

This work proposes a novel Mobile Manipulation Robot (MMR) called Libot, designed specifically for library applications such as automated navigation to specific locations (e.g., bookshelves), collecting returned books, recognizing individual books, and executing pick-and-place operations. The main objectives of this work are outlined as follows:

(1) **Design and Simulate the Library Environment:** Develop and assemble the Libot, within a virtual Gazebo environment tailored to library settings. Design a specialized component, termed 'booksink', for collecting books. Assemble components including the UR5 robotic arm, MiR250 mobile robot, Realsense 435i depth camera, and necessary accessories. Adjust the physical parameters to simulate realistic scenarios, ensuring the virtual library environment is accurately constructed for testing and development purposes.

(2) **Develop a Book Detection System:** Utilize the YOLO-v8 deep learning model to recognize information on the spine of each book using RGB images captured by the Realsense 435i camera. This system aims to validate the effectiveness of the detection model in accurately identifying and cataloging books based on visible spine labels.

(3) **Implement a Pick-and-Place Strategy:** Design a pick-and-place strategy for the customized booksink of Libot. After recognizing the location and orientation of books, employ the Grasp Pose Detection (GPD) algorithm to determine the precise positioning for the robot's end effector. These pose instructions are then processed through the MoveIt platform to manage the grasping tasks. The effectiveness of this strategy is to be tested within the simulated environment.

(4) **Enhance Navigation Capabilities:** Develop intelligent navigation strategies for Libot, enabling the robot to autonomously navigate to the location

of returned books based on their placement within the library. Incorporate and test Hector mapping techniques to enhance SLAM functionalities, improving the robot's ability to maneuver in complex environments.

(5) **Construct an Integrated Operational Workflow:** Build a comprehensive workflow for Libot that supports efficient and robust operations using the ROS. This system should be capable of adapting to emergency situations without disrupting the overall workflow, dynamically adjusting operational paths based on real-time sensor data to optimize task execution.

CHAPTER 2 : LITERATURE REVIEW

This literature review section aims to introduce the related work of service robots with mobile manipulators, highlighting their vital role in various sectors. Specifically, the recent work of the Mobile Manipulation Robot in the Library are also presented.

2.1 RELATED WORK OF THE SERVICE ROBOT

The service robot is defined as "a robot executes useful tasks for humans or equipment in personal or professional use" [12]. In recent fifteen years, a considerable amount of scientists and researches focus on the applications of MMRs in various service area [10, 13–20], some of which are shown in Figure 2.1. The team at Waseda University has successfully developed the human symbiotic robot TWENDY-ONE, which has a head, a torso, arms with compact passive mechanisms, anthropomorphic hands with joints and skin mechanical flexibility, and an all-wheel vehicle. The researchers experimented with evaluating waitpersons' care and kitchen support (e.g., making breakfast and serving trays) for people with reduced mobility and achieved good results [13]. Toyota has developed a Human Support Robot that can operate in a natural home environment and independently provide living support to disabled people or the elderly, or it can also be remotely controlled manually to provide care support [14]. Moreover, researchers from the Politecnico di Torino developed a robot consisting of a 6-degree-of-freedom robotic arm, a vital sign sensor, a patient interaction tablet, and a PAQUITOP mobile platform to help hospital blood bank operators perform blood transfusions and patient transportation, demonstrating the effectiveness of these MMRs in the hospital [9]. Meeussenn et al. developed a domestic-used MMR for the office area that can autonomously navigate and complete simple tasks such as opening doors and socket-plugging with vision and laser sensors [15]. In the agricultural field, the MMR also plays a significant role in sowing, harvesting, and other aspects [16–18]. A robot proposed by Ivo et al. for pesticide spraying in viticulture using an all-terrain mobile manipulator consisting of a custom-made flipper-tracking mobile base and a torque-controlled 7-degree-of-freedom Kinova Gen3 robot arm. The nozzle is controlled at the end of the arm so that it can accurately spray to the designated area [16].

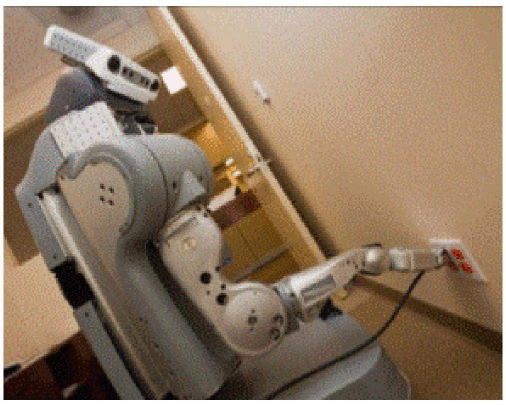
2.1.1 Related Advanced Technology in Service Robot

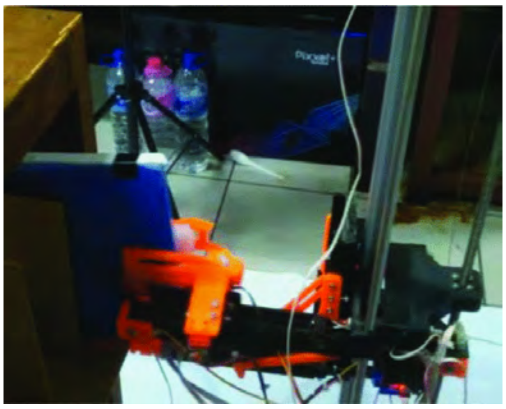
There have been significant advancements in field of object detection, autonomous navigation and path planning applied in the service robot with mobile manipulator. In this section, a review of a series of these techniques is presented.

2.1.1.1 Object Detection

In the previously mentioned study by Meeussenn et al., they used segmentation and clustering methods in conjunction with the geometric properties of the door for recognition. However, they could not recognize the case where the door was flush with the wall. To detect handles, they used the Viola and Jones object classifier cascade and 3D laser point cloud segmentation in parallel to recognize [15]. According to the working environment of intelligent service robots, Li et al. adopted a series of comprehensive recognition methods, which locked the region by segment-

(a) The TWENDY-ONE Robot performed wash-up at home [13]


(b) Robot recharged itself in the office environment [15]


(c) Library robot performed book-returning operations [20]

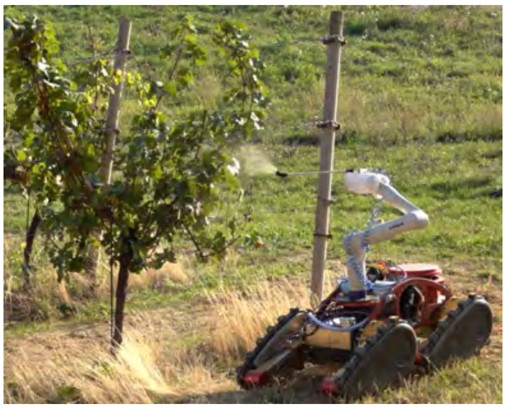

(d) Agricultural robot performed pesticide spraying in a field [16]

Figure 2.1: Tasks for service-based MMRs in different scenarios

ing the image using hierarchical models and multi-size filters and then compared the region's color features and shape features with those of the database. Finally, the Speeded Up Robust Features (SURF) algorithm is necessary to match the target [21]. A group of researchers in Japan proposed a human care robot system that mainly used a cooperative object detection method. Humans can choose the desired object to request to grasp it from a captured picture through matching template images [22]. Some Convolutional Neural Network (CNN) models are also the main algorithms for individual robots to finish the object recognition task [23–25]. Sun et al. in 2018 proposed a CNN with multi-task objectives for scene recognition and target detection, using global inference to understand the scene and global scene features to improve the recognition rate, which was tested in an office scene to achieve good recognition results [23]. Inspired by the feature pyramid network FPN [26] and grasping network DrGNet, Yu et al. cascaded them into a grasping CNN with image segmentation for grasping detection and verified the effectiveness in grasping objects on a table [24].

2.1.1.2 Localization and Mapping

Marder-Eppstein et al. (2010) describe a robust navigation system using a voxel-based 3D mapping algorithm, capable of dynamically modeling unknown spaces [27]. Their method ensures the robot can navigate efficiently around temporary obstacles. The ROS navigation stack is later forked from this research. A method separates odometry and mapping to improve real-time processing and mapping contributes advancements in Lidar Odometry and Mapping (LOAM), crucial for real-time applications such as autonomous driving and indoor navigation of robots in libraries [28]. Kim et al. (2018) focus on robotic mapping and registration of 3D point clouds using SLAM, specifically addressing the challenges posed by environments with multiple occlusions, like libraries. They introduce a mobile robot that utilizes a hybrid scanning system to enhance the automation and accuracy of data collection [29]. The hector mapping method presented as a scalable SLAM system which capable of 3D motion estimation, ideal for indoor navigation systems in robots [30]. This system integrates inertial sensing with robust scan matching, providing high reliability in library environments where non-planar movement across different floor levels may be required. ORB-SLAM3, a versatile system that supports multiple map reuse and real-time relocalization, which could be particularly beneficial in a library setting where different areas might be revisited at various times [31]. The system's flexibility in handling various sensor inputs and environmental conditions makes it adaptable to the structured yet dynamic nature of libraries.

2.2 RECENT WORK OF THE MOBILE MANIPULATION ROBOT IN THE LIBRARY

MMR for libraries is a sub-topic of MMR and service robots. In recent years, robots designed for libraries have been deployed in some public libraries, and many new technologies have been widely used in MMR for library use [19,20,32–35]. This kind of robot can usually navigate, locate, and place books. Some of the MMRs in the library have the ability to find books that are lying down or even guide readers in finding the books they desire. A novel robotic system for libraries was developed by Anita et al. The robot can transport and place books by scanning barcodes against a database of book information and then using a global positioning system to locate the designated location where the books are to be placed [19]. However, the interference and accuracy of using GPS signals for indoor positioning and navigation are debatable. In another design of a library robotic system, Tarkesh et al. detected RFID tags on books for book identification. However, due to the proximity of the books, the interference from the RFID transceiver may be very noticeable, and there would be a misrecognition of the library book area [32]. Regarding improving library robots, researchers from IIT have developed a low-cost book robot based on a similar detection method. It utilized a crank-slider mechanism and a two-crossed scissor lift mechanism for book gripping [33]. Finding the desired location is vital for the MMR for library use. Often, the solution for these robots is a line following robot [34] [20], i.e., moving along a line on the floor. Subsequently, the A* algorithm [20] and Dijkstra's algorithm [35], to name a few, have been applied to the field of library robotics, making path planning has become much more capable and flexible in libraries. However, these methods usually have the disadvantages of poor real-time performance and long computation time, such that they cannot change their routes in real-time for unexpected situations.

CHAPTER 3 : EXPERIMENTAL DESIGN

3.1 HARDWARE OF LIBOT

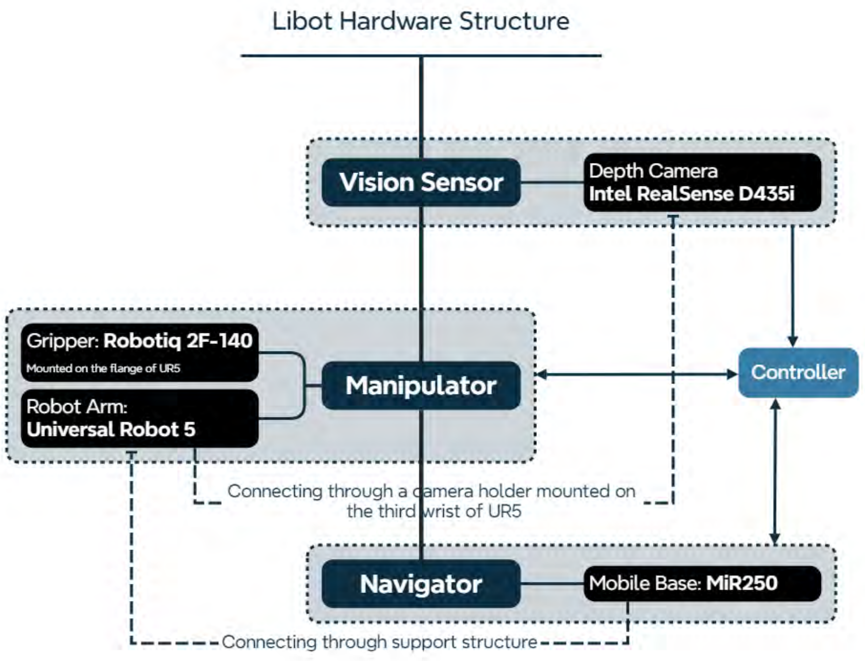


Figure 3.1: Libot Hardware Structure

Libot's hardware consists of three main modules: vision sensor, manipulator, and navigator. These modules connect to a central control computer.

The navigator module utilizes a Mobile Industrial Robot 250 (MiR250). MiR250 acts as the foundation of Libot, providing mobility and supporting a customized book sink. The book sink functions as both a storage unit for books during operation and a mounting platform for the manipulator module.

The manipulator module consists of a Universal Robot 5 (UR5) robotic arm mounted on the book sink. Two key components are attached to the UR5: a

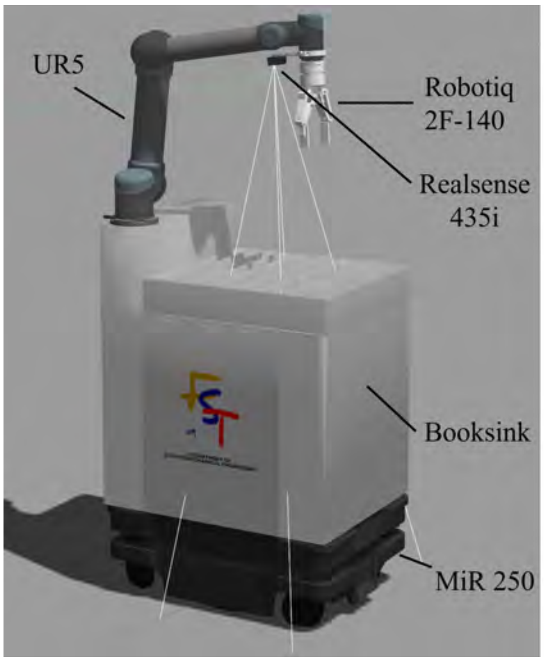


Figure 3.2: Libot

Robotiq 2F-140 gripper, which enables Libot to grasp and manipulate objects, and an Intel RealSense D435i depth camera which belongs to vision sensor module. This camera is mounted on the UR5's third wrist through a dedicated camera holder, providing real-time depth perception for the system.

3.1.1 Navigator

Mobile Industrial Robot 250

The MiR250, developed by Mobile Industrial Robots, is an advanced autonomous mobile robot (AMR). With a width of 580 mm, it can fit its footprint into some narrow corridors which is ideal for library working operation. Loaded up to 250kg, which is abundant for carrying booksink, UR5 robot arm and book.

Embedded with autonomous navigation function. Equipped with a comprehensive array of sensors, the MiR250 utilizes advanced LiDAR and 3D cameras to sense its environment. This array enables the robot to perform real-time mapping, obstacle detection, and dynamic path planning. MiR250 automatically maps its surroundings to navigate efficiently. It dynamically adjusts its route to avoid obstacles and ensure timely and safe deliveries.

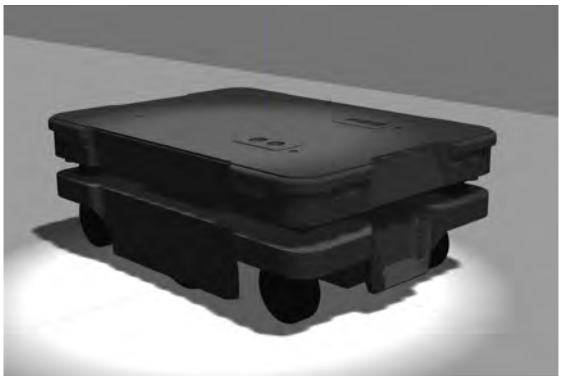


Figure 3.3: MiR250 in Gazebo simulation

3.1.2 Manipulator

3.1.2.1 Universal Robot 5

The UR5, shown in Figure 3.4 is a flexible light payload industrial collaborative robot that is widely used in the field of industrial automation. Developed by the Danish company Universal Robots, it supports integration with various programming environments and languages like C++ and Python, and can perform a variety of complex tasks. Also, the ROS driver for the UR series allows it to be controlled by ROS. Embedded with various force sensors and the safety system, it can sense outer force and actuate its force to work together with human operators without extra protection methods.

As shown in the Figure 3.5, the design of the UR5 robotic arm consists of six revolute joints and six links, with six degrees of freedom. It has an arm length of 850 mm and a total weight of 18.4 kg, making it ideal for maneuvering in tight

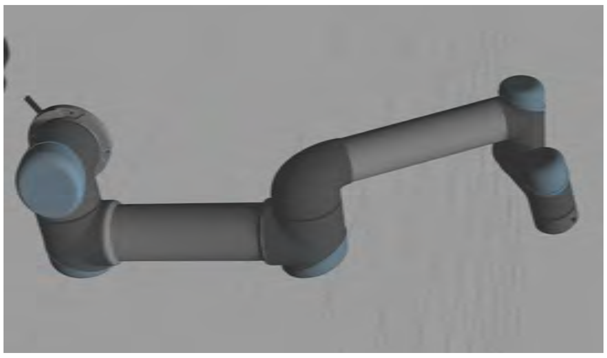


Figure 3.4: UR5 in Gazebo simulation

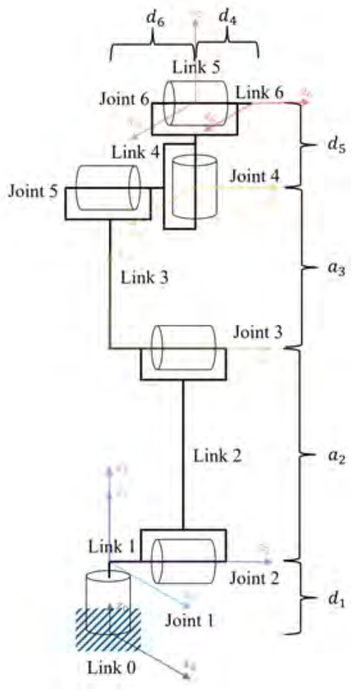


Figure 3.5: Frames assignment of UR5 [36]

workspaces like libraries. It is worth noting that because it has a not-monocentric wrist which means the three joint axes of the wrist (the last three joint axes) do not intersect at one point, resulting in the movement of operating the end effector will be more complicated (Each joint axis must participate in the movement).

The inverse kinematics (IK) solution of robot kinematics, i.e., knowing the position and direction of the end effector of the robot arm and finding the variables $P = p_i$ of each joint, where $p_i = [\theta_1^i, \ldots, \theta_6^i]^T \in [0,2\pi)^6$, is a problem that the robot system must face. It is considered to be a relatively difficult problem to solve the IK solution within a robot arm. Usually, the researchers should figure out nonlinear transcendental equations, in which they need to determine whether the solution exists, whether the solution is unique, and how to solve it. The IK solution will be introduced in the following section.

The process of finding the IK solution begins with the forward kinematics of UR5, which is expressed as a homogeneous transform matrix (including the position and orientations of the end effector frame x_6-y_6-z_6 relative to the base frame x_0-y_0-z_0) [37]:

$$ {}^0_6T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T {}^5_6T = \begin{bmatrix} u_x & v_x & w_x & {}^0p_x \\ u_y & v_y & w_y & {}^0p_y \\ u_z & v_z & w_z & {}^0p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

where

$u_x = c_1c_{234}c_5c_6 - c_1s_{234}s_6 + s_1s_5c_6$

$u_y = s_1c_{234}c_5c_6 - s_1s_{234}s_6 + c_1s_5c_6$

$u_z = s_{234}c_5c_6 + c_{234}s_6$

$v_x = -c_1c_{234}c_5s_6 - c_1s_{234}c_6 - s_1s_5s_6$

$v_y = -s_1c_{234}c_5s_6 - s_1s_{234}c_6 + c_1s_5s_6$

$v_z = -s_{234}c_5s_6 + c_{234}c_6$

$w_x = -c_1c_{234}s_5 + s_1c_5$

$w_y = -s_1 c_{234} s_5 - c_1 c_5$

$w_z = -s_{234} s_5$

${}^0p_x = a_2 c_1 c_2 + a_3 c_1 c_{23} + d_5 c_1 s_{234} - d_6 c_1 c_{234} s_5 + d_4 s_1 + d_6 s_1 c_5$

${}^0p_y = a_2 s_1 c_2 + a_3 s_1 c_{23} + d_5 s_1 s_{234} - d_6 s_1 c_{234} s_5 - d_4 c_1 - d_6 c_1 c_5$

${}^0p_z = d_1 + a_2 s_2 + a_3 s_{23} - d_5 c_{234} - d_6 s_{234} s_5$

Then the θ_1, which is the 5th joint angle, can be computed according to tranformation from frame 1 to frame 5 and Equation 3.2:

$$ {}^0_1T^{-1}\,{}^6_0T^{d}\,{}^5_6T^{-1} = \begin{bmatrix} a_x & b_x & c_x & (p_y - d_6 w_y)s_1 + (p_x - d_6 w_x)c_1 \\ a_y & b_y & c_y & (p_y - d_6 w_y)c_1 - (p_x - d_6 w_x)s_1 \\ u_z c_6 - v_z s_6 & w_z & -u_z s_6 - v_z c_6 & -d_1 + p_z - d_6 w_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

where

$a_x = u_x c_1 c_6 + u_y s_1 c_6 - v_x c_1 s_6 - v_y s_6$

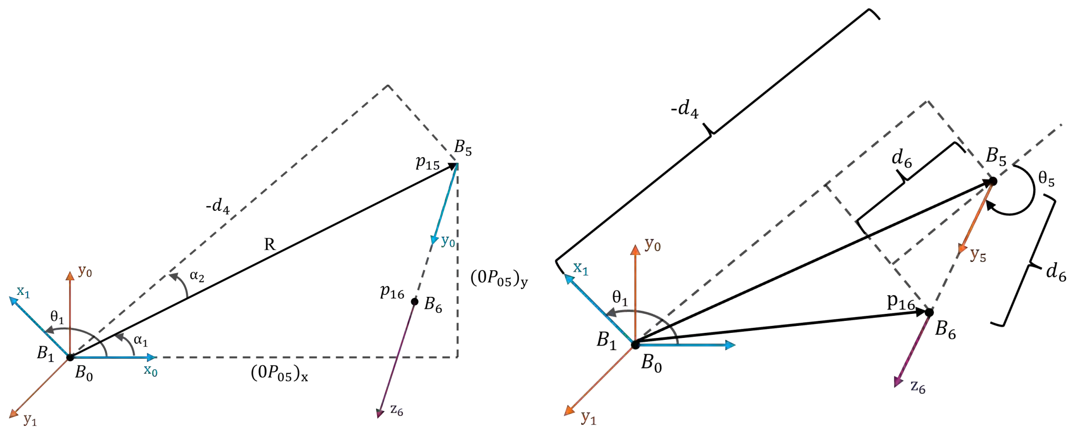
$a_y = -u_x s_1 c_6 + u_y c_1 c_6 + v_x s_1 s_6 - v_y c_1 s_6$

$b_x = w_x c_1 + w_y s_1$

$b_y = -u_x s_1 + w_y c_1$

$c_x = -u_x c_1 s_6 - u_y s_1 s_6 - v_x c_1 c_6 - v_y s_1 c_6$

$c_y = u_x s_6 - u_y c_1 s_6 + v_x s_1 c_6 - v_y c_1 c_6$



(a) An overhead view of the robot, looking down on the x-y plane for finding θ_1

(b) An overhead view of the robot, looking down on the x-y plane for finding θ_5

Figure 3.6: Geometric IK Analysis for finding θ_1 and θ_5 [38]

$$
{}^{1}_{5}T ={}^{1}_{4} T^{4}_{5}T = \begin{bmatrix} c_{234}c_5 & -c_{234}s_5 & s_{234} & a_2c_2 + a_3c_{23} + d_5s_{234} \\ -s_5 & -c_5 & 0 & -d_4 \\ s_{234}c_5 & -s_{234}s_5 & -c_{234} & a_2s_2 + a_3s_{23} - d_5c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}
$$

$$
{}^{0}_{1}T^{-1}T^{5}_{6}T^{-1} ={}^{1}_{5} T \tag{3.4}
$$

Using the equality of Equation 3.4, the relationship between the cells can be solved:

$$
(p_y - d_6w_y)c_1 - (p_x - d_6w_x)s_1 = -d_4 \tag{3.5}
$$

Known the Auxiliary Angle Formula:

$$
\begin{aligned}
a\sin x + b\cos x &= \sqrt{a^2+b^2}\sin\left(x + \arctan\frac{b}{a}\right),(a>0) \\
a\sin x + b\cos x &= \sqrt{a^2+b^2}\cos\left(x - \arctan\frac{a}{b}\right),(b>0)
\end{aligned}
\tag{3.6}
$$

θ_1 can be carried out with following trigonometric functions,which have two solutions:

$$r = \sqrt{(p_y - d_6 w_y)^2 + (p_x - d_6 w_x)^2} \tag{3.7}$$

$$\gamma = \text{Atan2}(p_y - d_6 w_y, p_x - d_6 w_x) \tag{3.8}$$

$$sin(\gamma - \theta_1) = \frac{-d_4}{r} \quad cos(\gamma - \theta_1) = \frac{\pm\sqrt{r^2 - d_4^2}}{r} \tag{3.9}$$

$$\theta_1 = \text{Atan2}(p_y - d_6 w_y, p_x - d_6 w_x) - \text{Atan}\Big(-d_4, \pm\sqrt{r^2 - d_4^2}\Big) \tag{3.10}$$

Regarding to θ_5 and θ_6, the key of solver is the equality of Equation 3.11:

$${}^0_1T^{-1}{}^0_6T^d = {}^1_6T \tag{3.11}$$

$$
{}^0_1T^{-1}{}^0_6T^d = \begin{bmatrix}
u_x c_1 + u_y s_1 & v_x c_1 + v_y s_1 & w_x c_1 + w_y s_1 & p_x c_1 + p_y s_1 \\
-u_x s_1 + u_y c_1 & -v_x s_1 + v_y c_1 & -w_x s_1 + w_y c_1 & -p_x s_1 + p_y c_1 \\
u_z & v_z & w_z & p_z - d_1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\tag{3.12}
$$

$$
{}^{1}_{6}T = {}^{1}_{5}T\,{}^{5}_{6}T = \begin{bmatrix} c_{234}c_5c_6 - s_{234}s_6 & -c_{234}c_5s_6 - s_{234}c_6 & -c_{234}s_5 & n_x \\ -s_5c_6 & s_5s_6 & -c_5 & n_y \\ s_{234}c_5c_6 + c_{234}s_6 & -s_{234}c_5s_6 + c_{234}c_6 & -s_{234}s_5 & n_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}
$$

where

$n_x = a_2c_2 + a_3c_{23} + d_5s_{234} - d_6c_{234}s_5$

$n_y = -d_4 - d_6c_5$

$n_z = a_2s_2 + a_3s_{23} - d_5c_{234} - d_6s_{234}s_5$

Inside the matrices 3.12 and 3.13, the elements 3.14 and 3.15 are calculated.

$$
-p_xs_1 + p_yc_1 = -d_4 - d_6c_5 \tag{3.14}
$$

$$
\begin{aligned} -u_xs_1 + u_yc_1 &= -s_5c_6 \\ -v_xs_1 + v_yc_1 &= s_5s_6 \end{aligned} \tag{3.15}
$$

$$
c_5 = \frac{p_xs_1 - p_yc_1 - d_4}{d_6}, \quad s_5 = \frac{\pm\sqrt{d_6^2 - (p_xs_1 - p_yc_1 - d_4)^2}}{d_6} \tag{3.16}
$$

Thus,

$$
\theta_5 = \mathrm{Atan2}\Big(\pm\sqrt{d_6^2 - (p_xs_1 - p_yc_1 - d_4)^2}, p_xs_1 - p_yc_1 - d_4\Big) \tag{3.17}
$$

$$
\theta_6 = \mathrm{Atan2}\Big(\frac{-v_xs_1 + v_yc_1}{s_5}, \frac{-(-u_xs_1 + u_yc_1)}{s_5}\Big) \tag{3.18}
$$

For the last three parallel joints (θ_2, θ_3, and θ_4), it is not complicated to find them since they form a classical 3R planar robot arm. There are two ways for the results: with the elbow "up" or "down." When the distance to the fourth joint is smaller than the difference $|a_2 - a_3|$ or exceeds the total $|a_2 + a_3|$, there are no solutions. When $\theta_3 = \pi$, a displacement singularity occurs, rendering $\theta_2 = \pi$ arbitrary in the case when $a_2 = a_3$.

For the other mathematical modals and and parameters of UR5, please refer to Appendix A.

3.1.2.2 Robotiq 2F-140

The Robotiq 2F adaptive gripper is a professional gripper for collaborative robots like Universe Robot Series. For this product, the Robotiq company provides two editions: Robotiq 2f-85 and Robotiq 2f-140. In the project, the robot adopted Robotiq 2F-140 shown in Figure 3.7 as the grasping tool since it has a longer stroke of 140mm that the Libot can pick various sizes of books from the sink and place each of them on the bookshelves in the library-use scenario.

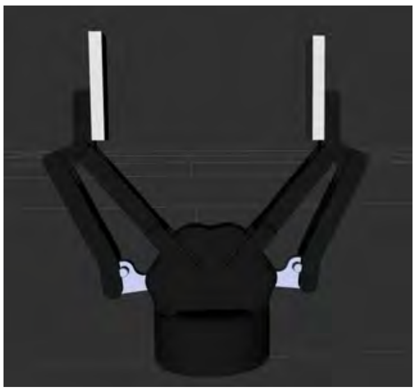


Figure 3.7: Robotiq 2F-140 in Rviz

3.1.3 Vision Sensor

Intel RealSense D435i

D435i is a new generation of depth cameras developed by Intel.Inc. As shown in Figure 3.8, it is characterized by providing both a depth sensor module (RGB-D Camera) and an inertial measurement unit (IMU). REALSENSE D435i uses a structured light-depth camera. The four cameras, which are known as the RGB-D camera, on the front are, from left to right, the left infrared camera, the infrared dot projector, the right infrared camera, and the RGB camera. The purpose of using it is to easily determine the pose and shape of the target, thereby achieving better grasping as well as barrier avoidance.

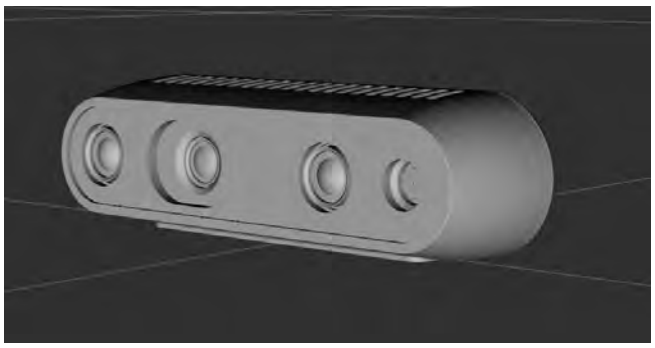


Figure 3.8: RealSense D435i in Rviz

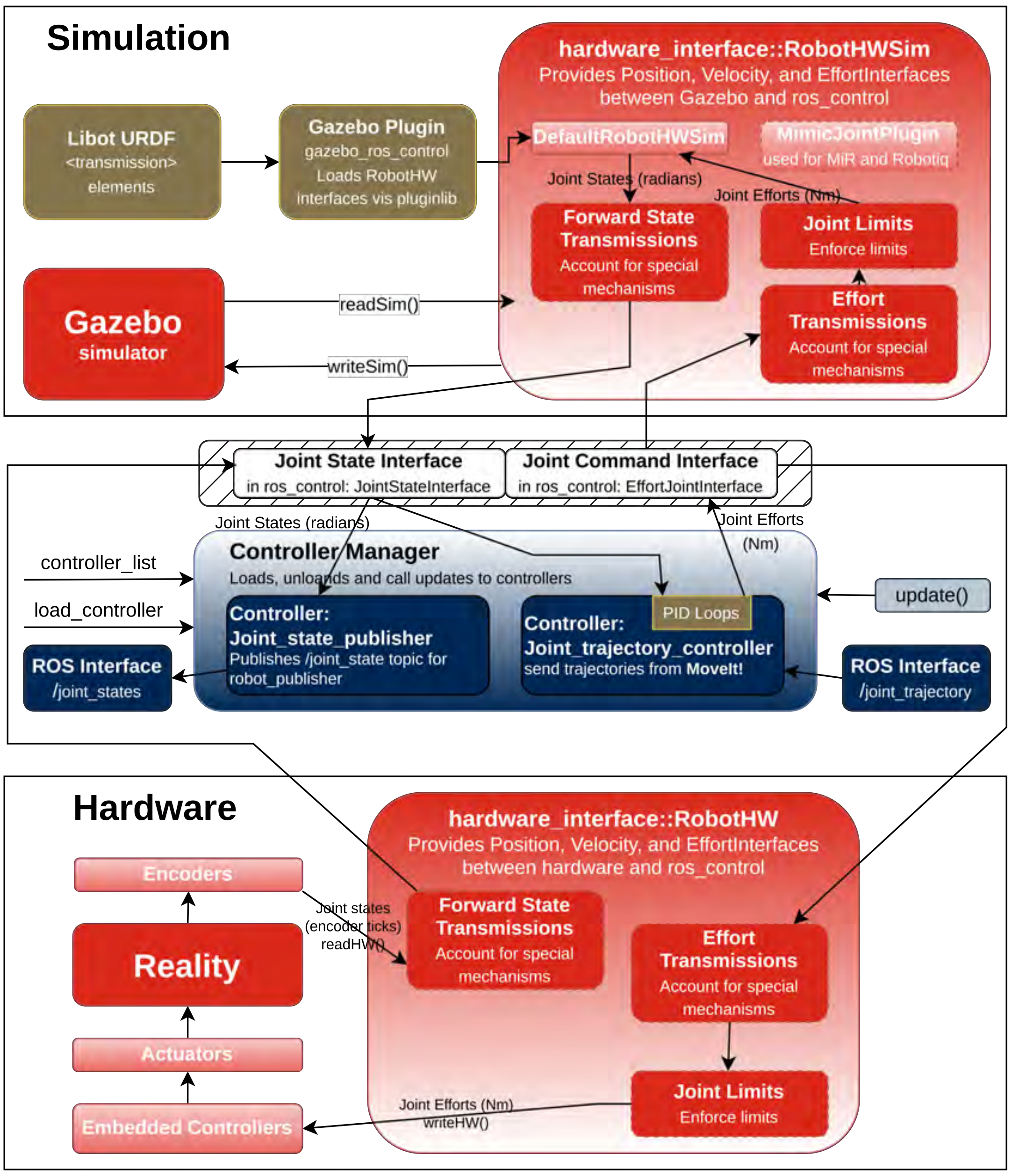

Simulation
hardware_interface::RobotHWSim
Provides Position, Velocity, and EffortInterfaces between Gazebo and ros_control
Libot URDF
<transmission> elements
Gazebo Plugin
gazebo_ros_control
Loads RobotHW interfaces vis pluginlib
DefaultRobotHWSim
MimicJointPlugin
used for MiR and Robotiq
Joint States (radians)
Joint Efforts (Nm)
Forward State Transmissions
Account for special mechanisms
Joint Limits
Enforce limits
Effort Transmissions
Account for special mechanisms
Gazebo
simulator
readSim()
writeSim()
Joint State Interface
in ros_control: JointStateInterface
Joint Command Interface
in ros_control: EffortJointInterface
Joint States (radians)
Joint Efforts (Nm)
controller_list
load_controller
Controller Manager
Loads, unloads and call updates to controllers
Controller:
Joint_state_publisher
Publishes /joint_state topic for robot_publisher
PID Loops
Controller:
Joint_trajectory_controller
send trajectories from MoveIt!
update()
ROS Interface
/joint_states
ROS Interface
/joint_trajectory
Hardware
hardware_interface::RobotHW
Provides Position, Velocity, and EffortInterfaces between hardware and ros_control
Encoders
Reality
Actuators
Embedded Controllers
Joint states (encoder ticks) readHW()
Forward State Transmissions
Account for special mechanisms
Effort Transmissions
Account for special mechanisms
Joint Limits
Enforce limits
Joint Efforts (Nm)
writeHW()

Figure 3.9: Data flow of `ros_control`, Gazebo, and reality in Libot

Libot was developed and controlled with a renowned systems: Robot Operating System (ROS) [39], simulated and tested in Gazebo, a primary tool used in ROS community. An overview of relationships between ROS services and simulation environment, ROS services between real hardware are presented in Figure 3.9 [40]. Technical in-depth of custom ROS networking of Libot, Construction of Universal

Robotic Description Format (URDF) of Libot used in simulation are presented in this chapter.

3.2.1 Control Methodology

3.2.1.1 ROS Network

A ROS network is a distributed system made up of independent programs called nodes. These nodes can control different machines within the system. In Libot application, for example, some nodes control a UR5 robot, responsible for reading encoder data and controlling its pose. Other nodes are distributed on a MiR robot, and additional nodes can handle camera information.
Crucially, these nodes can communicate with each other by publishing messages to ROS Topic and subscribing to different topics, as long as a connection is established within the ROS network, communication flows.

- Basic Nodes

  Figure 3.10 illustrates the ROS nodes active during Libot operation (excluding navigation nodes). These nodes are independent programs that communicate within the ROS network, each fulfilling a specific role.

  - Gazebo Simulation Nodes

    In figure 3.10, nodes:

    `/gazebo`, `/gazebo_gui` and `/gazebo_controller_spawner`

    are managed by Gazebo, a robot simulation software. Libot will run in Gazebo's simulated world, during the simulation, Gazebo publishes joint states of Libot to joint state publisher of MiR and robot state

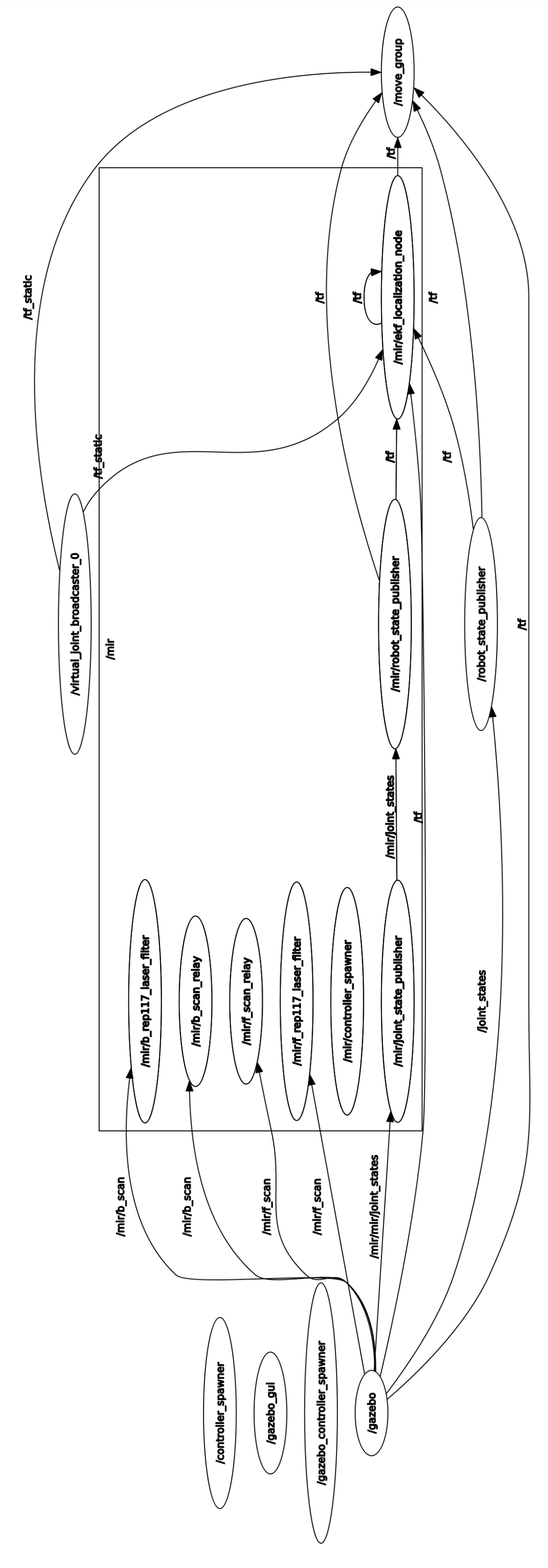


Figure 3.10: Flow of information between ROS nodes

publisher of a move group made up by UR5 robot arm and Robotiq gripper.

  - Joint State Publisher & Robot State Publisher

    Libot is controlled by two distinct groups inside ROS network, one is MiR group and the other is move group. Due to this configuration, there are two robot state publisher nodes in Libot ROS network which are:

    `/mir/robot_state_publisher` and `/robot_state_publisher`

    Joint state publisher read from Gazebo, publishes messages which contain name, position, velocity, effort of each non-fixed joints [41]. It can be used in conjunction with a robot state publisher node to also get information of transforms from each joint [42].

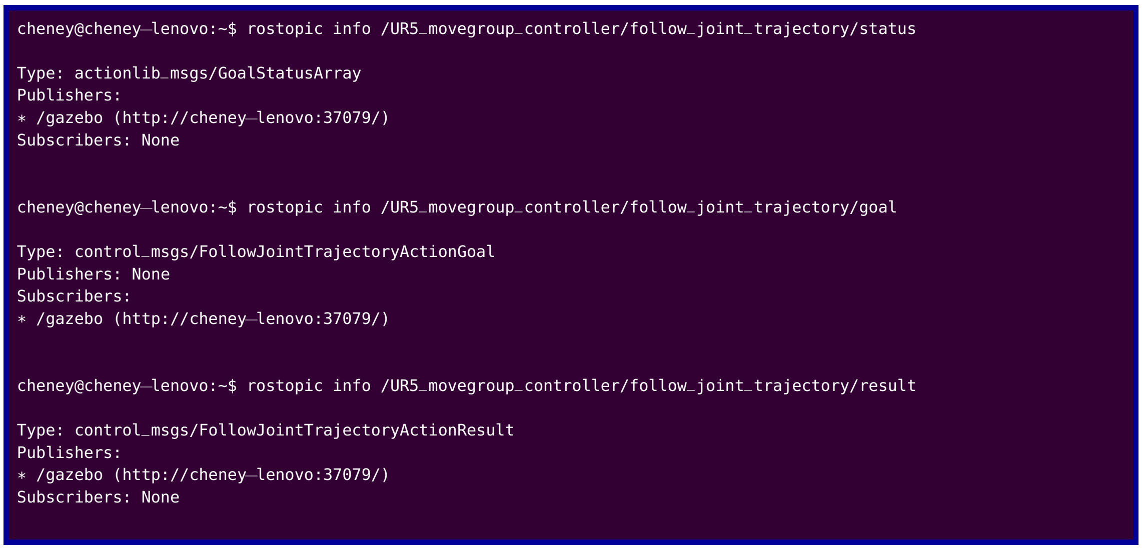
  - Controller Spawner

    The controller spawner is a crucial node within the Libot ROS network. Its primary function is to load and activate various robot controllers. In this project, the controller spawner is responsible for launching the following controllers: UR5 Move Group Controller, Robotiq Move Group Controller and MiR Controller. Details of these controllers will be discussed in Chapter 3.2.1.3 - Manipulation and Chapter 3.2.1.4 - Navigation.

- Topic Messaging

  ROS topics facilitate message exchange between nodes. One or multiple nodes can publish messages to a topic, while one or more nodes can subscribe to the same topic to receive information from the publishing node(s). However, to make communication successful, the message type of the subscriber must match that of the publisher. Although custom message types can be defined, throughout our project, we utilized predefined message types from the ROS Client Library. This library includes `rospy` (for Python) and `roscpp` (for C++) for not only creating nodes, publishing and subscribing to topics using the `NodeHandle` class within a ROS node.

An overview of message types used in this project is presented in this part. Additionally, Figure 3.10 shows messages flow network using topic publishing and subscribing.



```
cheney@cheney-lenovo:~$ rostopic info /UR5_movegroup_controller/follow_joint_trajectory/status

Type: actionlib_msgs/GoalStatusArray
Publishers:
* /gazebo (http://cheney-lenovo:37079/)
Subscribers: None


cheney@cheney-lenovo:~$ rostopic info /UR5_movegroup_controller/follow_joint_trajectory/goal

Type: control_msgs/FollowJointTrajectoryActionGoal
Publishers: None
Subscribers:
* /gazebo (http://cheney-lenovo:37079/)


cheney@cheney-lenovo:~$ rostopic info /UR5_movegroup_controller/follow_joint_trajectory/result

Type: control_msgs/FollowJointTrajectoryActionResult
Publishers:
* /gazebo (http://cheney-lenovo:37079/)
Subscribers: None
```

Figure 3.11: Terminal output of selected topic listing

– Topics in Move Group

  Topics facilitate communication within ROS for move groups like UR5 and Robotiq in the Libot workflow. Each group typically initializes three important topics: goal, status, and result. This section explains the message types associated with these topics and their contribution to Libot operation, using the UR5 move group as an example.

During operation, type, publisher, and subscriber can be retrieved using the `rostopic` in ROS Command Line Tool.

As shown in Figure 3.11, a goal topic under UR5 move group carries a message of type `control_msgs/FollowJointTrajectoryActionGoal` and send it to its subscriber Gazebo (the simulated world). This message type is a composite of several predefined basic message in ROS. It comprises: a `Header` storing time stamp, frame ID, and a sequence number; a `goal_ID` provides a unique identifier for the goal; and the `trajectory` information containing positions, accelerations, velocities along the planed path, it also includes tolerances of the path and goal, this will be read and executed by Gazebo or real robot.

Topic `/status` subscribe to our Gazebo node to get the status of the execution of the goal by returning a indicating number, The `/result` topic is for checking whether the goal is completed or failed from Gazebo.

- RViz: A Powerful Tool for Robotic Visualization and Interaction

  ROS Visualizer (RViz) is an invaluable visualization tool within the ROS ecosystem, offering a dynamic 3D visual interface for monitoring and interacting with robotic systems. Its versatility makes it a staple in both research and practical applications. This section explores Rviz's key functionalities and its integration with the robotics systems we built for Libot.

  – Visualization of Robot Models

    Rviz leverages the URDF of Libot to render accurate representations of Libot in a virtual 3D space. This visualization capability is crucial for ensuring that the physical attributes of a Libot, such as dimensions, joint articulations, and range of motion, are accurately reflected. It provides an intuitive understanding of the robot's structure and operational capacities without requiring physical prototypes, facilitating rapid development and iteration of robotic designs.

– Interpretation of Sensor Data

  Robotics heavily relies on sensor input for interaction with the environment. Rviz do not rendering any environmental features, however, it can perceive the environment by subscribing the sensor topics published by Libot's sensor, whether it be laser scans, point clouds, or camera feeds. By converting data stream which in particular: point clouds generate by Intel Realsense Depth camera, into visual elements, our team can directly observe how Libot perceives its surroundings.

– Path Planning and Navigation

  In motion planning, Rviz acts as a bridge between abstract algorithms and tangible outcomes. It visualizes planned trajectories, waypoints, and the robot's current path within the environment. This function is particularly useful when working with planning frameworks such as **MoveIt**, as it allows for the adjustment and optimization of paths before they are executed in the real world, thus ensuring safer and more efficient navigation.

– Simulation and Testing in a Controlled Environment

  By integrating with simulation tools like Gazebo, along with the `gazebo _ros_control` package, Rviz can replicate real-world conditions within a simulated environment. This capability allows our team to test and refine robot behaviors in a controlled setting.

3.2.1.2 Universal Robotic Description Format

The Universal Robotic Description Format (URDF) is an XML-based file format that serves as a comprehensive description of a robot's specifications. It plays a crucial role in various robotic applications, particularly those involving collision checking and motion planning. URDF provides not only the geometry information but also assembly hierarchy, inertial properties, and joint connection types to the motion planner (MoveIt and RViz) and visualizer (Gazebo).

There are two elements are required when forming URDF, `<joint/>` and `<link/>`. The `<link/>` initiate the link involved in Libot, geometry information or STL file can be declared inside this tag. The `<joint/>` specifies relationship between two joint by assigning `<child link>` and `<parent link>`. A 6-DoF information includes x,y,z, row, pitch, yaw are declare in this tag to specify position information. One of six joint types is required to choose for the `type` attribute, Libot's URDF use fixed or revolute joints. A segment of code in Libot's URDF to illustrate the construction of URDF shown below.

```
<xacro:include filename="$(find booksink)/urdf/booksink.urdf"/>
<xacro:include filename="$(find ur_description)/urdf/ur5.xacro"/>
<joint name="sink_ur5_joint" type="fixed">
    <origin xyz="-0.308 -0.2105 0.875" rpy="0.0 0.0 -1.57"/>
    <parent link="booksink_link"/>
    <child link="base_link"/>
</joint>
```

Figure 3.12: A segment from Libot's master URDF

In this segment of Master URDF of Libot showing in figure 3.12, the supporting structure `booksink` and all links and joints inside `UR5` are declared using `xacro:include` referring to sub URDF files. UR5 arm is expected to be installed on the supporting structure, so booksink link is set to parent and the base_link of UR5 is the child. The joint type is fixed since there will not be any relative motion between those two links.

Noted that the MiR moving base's description is not included in Libot's Master URDF since they are two distinct parts of Libot but they will eventually cooperate with each other through ROS network. Detailed work of assembling them will be explained in Chapter 3.2.2 - Simulation Environment using a customized ROS launch file.

To make Libot manipulable in our simulation environment: Gazebo, a segment of code which enables `gazebo_ros_control` is added to the URDF at the very last part of Libot's Master URDF. Please refer to the Appendix B or complete code of Libot's URDF.

3.2.1.3 Manipulation

- MoveIt Motion Planning Framework

  MoveIt: The Foundation for Libot's Manipulation Planning

  The MoveIt Motion Planning Framework serves as the core engine for planning the motions of Libot's manipulators. As illustrated in Figure 3.14, MoveIt utilizes Move Groups as central components. These Move Groups act as integrators, gathering all relevant information about Libot's kinematic and dynamic capabilities. Additionally, they provide sets of ROS actions that can be used to control Libot's movements.

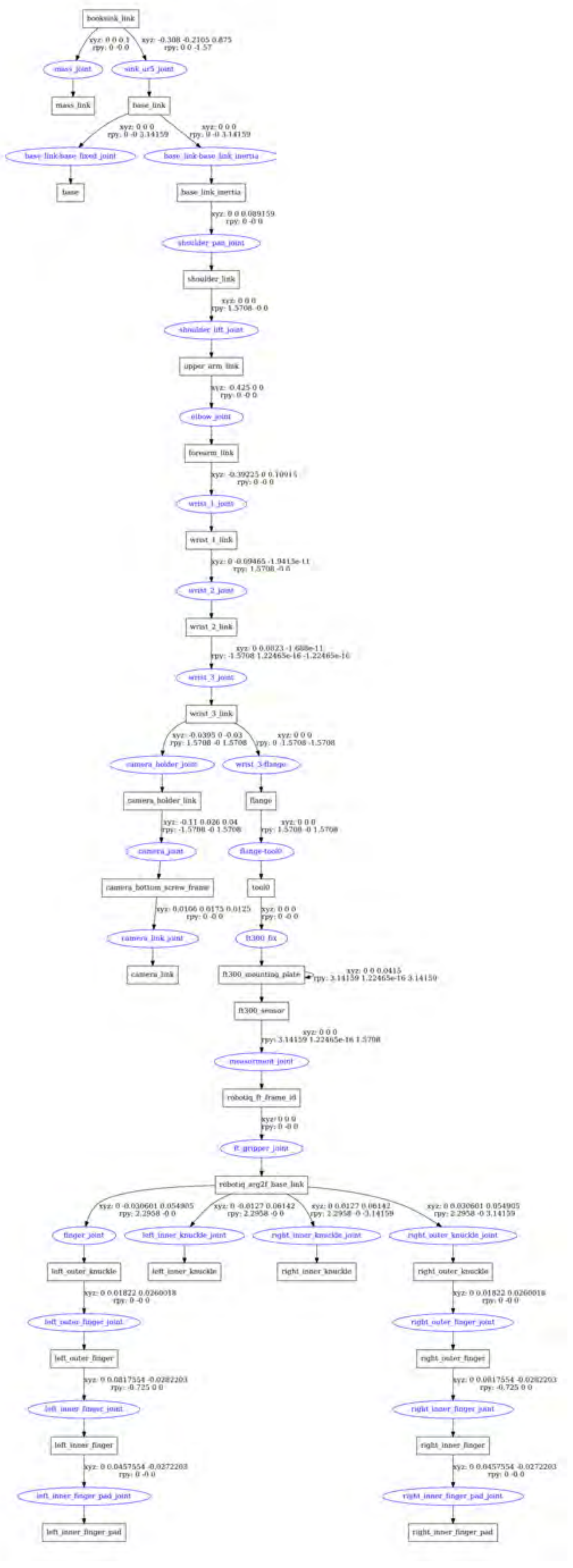

booksink_link
xyz: 0 0 0.1 rpy: 0 -0 0
xyz: -0.308 -0.2105 0.875 rpy: 0 0 -1.57
mass_joint
sink_ur5_joint
mass_link
base_link
xyz: 0 0 0 rpy: 0 -0 3.14159
xyz: 0 0 0 rpy: 0 -0 3.14159
base_link-base_fixed_joint
base_link-base_link_inertia
base
base_link_inertia
xyz: 0 0 0.089159 rpy: 0 -0 0
shoulder_pan_joint
shoulder_link
xyz: 0 0 0 rpy: 1.5708 -0 0
shoulder_lift_joint
upper_arm_link
xyz: -0.425 0 0 rpy: 0 -0 0
elbow_joint
forearm_link
xyz: -0.39225 0 0.10915 rpy: 0 -0 0
wrist_1_joint
wrist_1_link
xyz: 0 -0.09465 -1.9413e-11 rpy: 1.5708 -0 0
wrist_2_joint
wrist_2_link
xyz: 0 0.0823 -1.688e-11 rpy: -1.5708 1.22465e-16 -1.22465e-16
wrist_3_joint
wrist_3_link
xyz: -0.0395 0 -0.03 rpy: 1.5708 -0 1.5708
xyz: 0 0 0 rpy: 0 -1.5708 -1.5708
camera_holder_joint
wrist_3-flange
camera_holder_link
flange
xyz: -0.11 0.026 0.04 rpy: -1.5708 -0 1.5708
xyz: 0 0 0 rpy: 1.5708 -0 1.5708
camera_joint
flange-tool0
camera_bottom_screw_frame
tool0
xyz: 0.0106 0.0175 0.0125 rpy: 0 -0 0
xyz: 0 0 0 rpy: 0 -0 0
camera_link_joint
ft300_fix
camera_link
ft300_mounting_plate
xyz: 0 0 0.0415 rpy: 3.14159 1.22465e-16 3.14159
ft300_sensor
xyz: 0 0 0 rpy: 3.14159 1.22465e-16 1.5708
measurment_joint
robotiq_ft_frame_id
xyz: 0 0 0 rpy: 0 -0 0
ft_gripper_joint
robotiq_arg2f_base_link
xyz: 0 -0.030601 0.054905 rpy: 2.2958 -0 0
xyz: 0 -0.0127 0.06142 rpy: 2.2958 -0 0
xyz: 0 0.0127 0.06142 rpy: 2.2958 -0 -3.14159
xyz: 0 0.030601 0.054905 rpy: 2.2958 -0 3.14159
finger_joint
left_inner_knuckle_joint
right_inner_knuckle_joint
right_outer_knuckle_joint
left_outer_knuckle
left_inner_knuckle
right_inner_knuckle
right_outer_knuckle
xyz: 0 0.01822 0.0260018 rpy: 0 -0 0
xyz: 0 0.01822 0.0260018 rpy: 0 -0 0
left_outer_finger_joint
right_outer_finger_joint
left_outer_finger
right_outer_finger
xyz: 0 0.0817554 -0.0282203 rpy: -0.725 0 0
xyz: 0 0.0817554 -0.0282203 rpy: -0.725 0 0
left_inner_finger_joint
right_inner_finger_joint
left_inner_finger
right_inner_finger
xyz: 0 0.0457554 -0.0272203 rpy: 0 -0 0
xyz: 0 0.0457554 -0.0272203 rpy: 0 -0 0
left_inner_finger_pad_joint
right_inner_finger_pad_joint
left_inner_finger_pad
right_inner_finger_pad

Figure 3.13: Structure tree of Libot's URDF

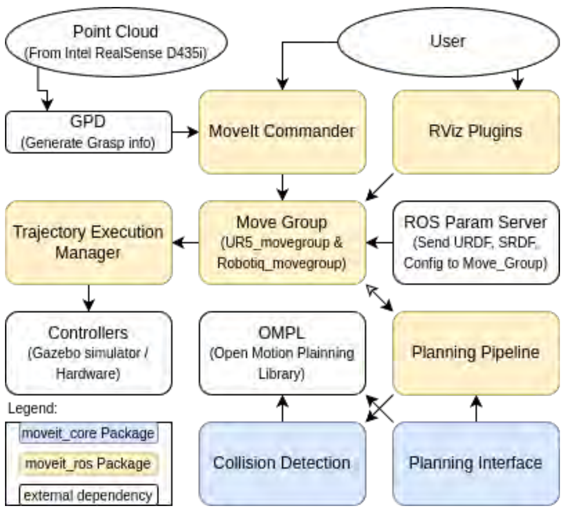


Figure 3.14: MoveIt Systems Architecture

Integration with Deep Learning-based Grasp Pose Detection

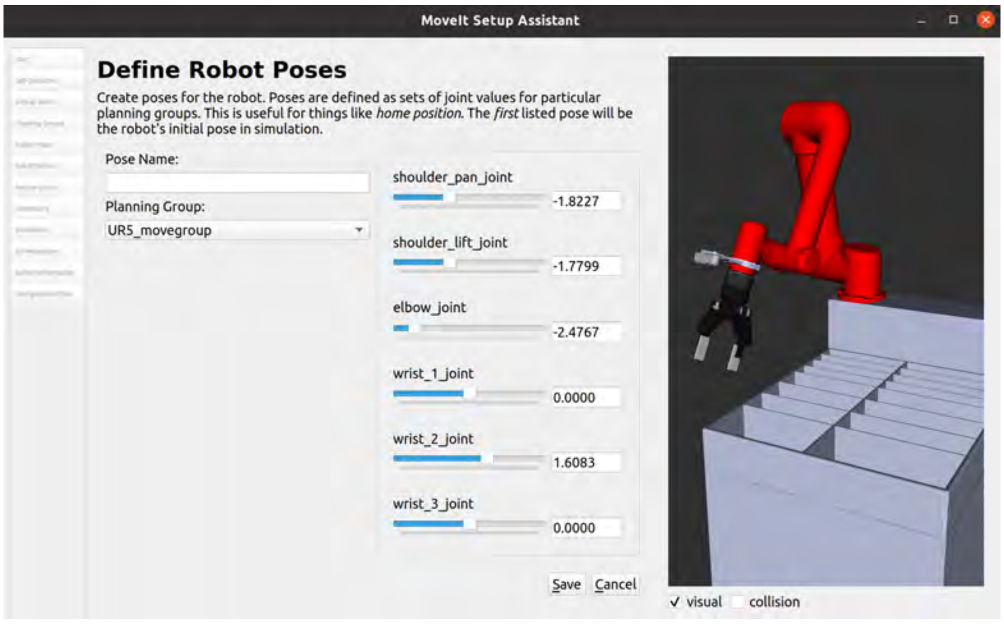
Libot leverages a deep learning-based grasp pose detection method (GPD) [43], which will be detailed in Chapter 3.2.1.3 - Object Detection. This method works seamlessly with MoveIt for book grasping tasks, design of information flows between GPD and MoveIt will is discussed later in this section.

MoveIt Setup Assistant: Parameterization and Configuration with MoveIt Setup Assistant

Before motion planning can occur, the MoveIt core service requires knowledge of Libot's specific parameters. To facilitate this configuration, a user-friendly GUI application provided by MoveIt, called MoveIt Setup Assistant, is employed. This tool streamlines the generation of all necessary parameters by directly parsing Libot's URDF file.

– Move Group Setup

  As mentioned in Figure 3.1, Manipulator of Libot can be decomposed in to two part, Robotiq gripper and UR5 robot arm. Thus, it is logical



Define Robot Poses

Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *home position*. The *first* listed pose will be the robot's initial pose in simulation.

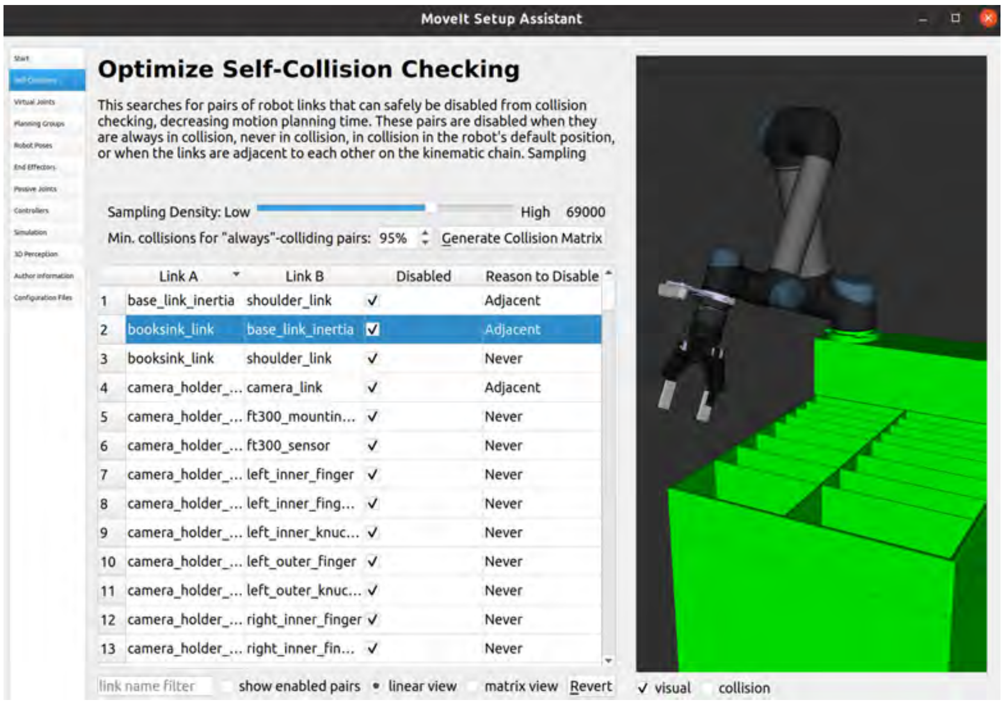
Pose Name:

Planning Group: UR5_movegroup

shoulder_pan_joint	-1.8227
shoulder_lift_joint	-1.7799
elbow_joint	-2.4767
wrist_1_joint	0.0000
wrist_2_joint	1.6083
wrist_3_joint	0.0000

Save Cancel

(a)



Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling

Sampling Density: Low High 69000

Min. collisions for "always"-colliding pairs: 95% Generate Collision Matrix

	Link A	Link B	Disabled	Reason to Disable
1	base_link_inertia	shoulder_link	✓	Adjacent
2	booksink_link	base_link_inertia	✓	Adjacent
3	booksink_link	shoulder_link	✓	Never
4	camera_holder_...	camera_link	✓	Adjacent
5	camera_holder_...	ft300_mountin...	✓	Never
6	camera_holder_...	ft300_sensor	✓	Never
7	camera_holder_...	left_inner_finger	✓	Never
8	camera_holder_...	left_inner_fing...	✓	Never
9	camera_holder_...	left_inner_knuc...	✓	Never
10	camera_holder_...	left_outer_finger	✓	Never
11	camera_holder_...	left_outer_knuc...	✓	Never
12	camera_holder_...	right_inner_finger	✓	Never
13	camera_holder_...	right_inner_fin...	✓	Never

link name filter show enabled pairs ● linear view matrix view Revert

(b)

Figure 3.15: Screenshots of MoveIt Setup Assistant

to set two move groups in MoveIt Setup Assistant for these two component as `UR5_movegroup` and `Robotiq_movegroup`. Moreover, Robotiq gripper is attached on the third wrist of UR5 as an end-effector, so we set `Robotiq_movegroup` as the subgroup of `UR5_movegroup`. Configuration file of two move groups added correspondingly and stored in a new ROS package.

- Define Poses

  Pre-defined pose stores values of each joint, and the name of this pose. it allows MoveIt planning to move to a `named_target` by only pass the name of the pose to MoveIt commander. It is useful when the robot need to frequently visit some certain positions, it is also convenient to set the pose in Setup Assistance with a friendly user interface shown in Figure 3.15a. We defined each three named poses for both `UR5_movegroup` and `Robotiq_movegroup`.

- Self-Collision Checking

  Generate Self-Collision Matrix: The assistant can automatically generate a self-collision matrix based on the robot's URDF file as shown in Figure 3.15b. This matrix identifies pairs of links (robot segments) that are known to collide, never collide, or might collide depending on the robot's configuration.

Controlling Libot with MoveIt Tools

Libot's control through MoveIt can be achieved using various methods. One approach involves MoveIt Commander, which encompasses both the MoveIt Python API and the MoveIt Command Line Tool. This method offers programmatic control over Libot's movements. Additionally, ROS Visualizer (RViz) provides an interactive alternative for manipulating Libot. RViz allows users to visually plan and execute trajectories within a simulated environment. A more detailed discussion of Control of Motion is listed in

Chapter 3.2.1.3 Motion Planning and Task Integration.

- TF2: Transform Library

TF2 is a novel and powerful generation of the transform library, enabling the user and researchers to utilize each coordinate frame freely. It is a distributed system with no bottleneck process and low latency [44]. In this system, the coordinate frame information of a robot is accessible to all ROS nodes on the computer. Tf2 offers two options for managing this information: utilizing a central server that holds all the transform data or allowing each component to construct its own transform database within the distributed system. One of the main characteristics of it is the tree-like structure between coordinate frames. This structure facilitates users to transform between two coordinate frames in a standard way [45]. It is also this structure that users are not required to obtain the configuration of each coordinate frame and can modify by redefining a link.

As a ROS important package, the TF2 could enable users to let each node communicate with each other like Moveit via tf topic. For instance, in figure 3.10, the `/mir/ekf_localization_node` node communicates with `/move_group` and via `/tf`, which converts the geometric data to the desired type of data of each joint at the same time. Two standard modules, including Broadcaster and Listener, are produced for the ROS network and also work outside of ROS. The Broadcaser (transform broadcasting) is a simple module that broadcasts and updates the transform at a certain frequency. The Listener (tranform listening) hears the values and puts them into a list. If there is a need to obtain the value, this Listener do interpolation between two adjacent values. Noted that frequency should be considered since the listener usually does not regard the present coordinate system as the future one, so users should set the frequency high enough to keep the motion of the joint real.

In the computing transform part, the listener follows the edge of the tree up or down to find the parent node, which creates the spinning set. If so, the listener may compute the net transform from the source fame to the target frame in terms of the order of the spinning set. For example, given the transform b respect to a T_a^b and transform c respect to b T_b^c, the transform c respect to a is:

$$T_a^c = T_a^b * T_b^c \tag{3.19}$$

which is chain-like and straightforward; the case is true in multiple frame calculation.

- Object Detection

  In order to achieve the grasping goal, Libot should first sense what type of books are in the booksink. Therefore, embedding a machine vision system is necessary for the robot. The following paragraph will present the experimental design of the object detection method in the Libot.

  - Model-YOLO v8

    This project, YOLO v8 adopted as the deep learning detection model [46]. YOLO stands for "You look only once, " meaning higher detection accuracy and faster detection speed. The YOLO v8 model consists of three parts: Backbone, Neck, and Head. The network model is shown in Figure 3.16. Among them, the backbone part adopts a Cross Stage Partial Network and a pyramid pooling module, which can reduce the amount of calculation and better extract features. The newly replaced C2F module of the Neck part allows for both lightweight and adaptability to different objects. As for the Head part, it adopted the famous Decoupled Head, which can effectively reduce the amount of calculation and also enhance robustness. It is worth noticing that the Anchor-Free detection method used by v8 can directly predict the target center point and the aspect ratio of the image, further reducing the number of frames, thereby improving the detection speed and accuracy of the model.

Based on the novel Anchor-Free detection method, YOLO v8 regards object detection as a spatially separated bounding box and related probability-like regression problem, where the bounding box and classification scores can be predicted directly from the entire image. The picture is divided into S*S grids by YOLO. In this case, the target's center point falling into one of the grid cells is responsible for target detection. Then, each grid cell can predict X bounding boxes and the confidence of each grid cell.

Thus, class-specific confidence scores of each grid $\mathrm{Pr}(\mathrm{Class}_i) * \mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}}$, which imply the probability of that class may appear in that image as well as the quality of predict box fitting the objectives [47]:

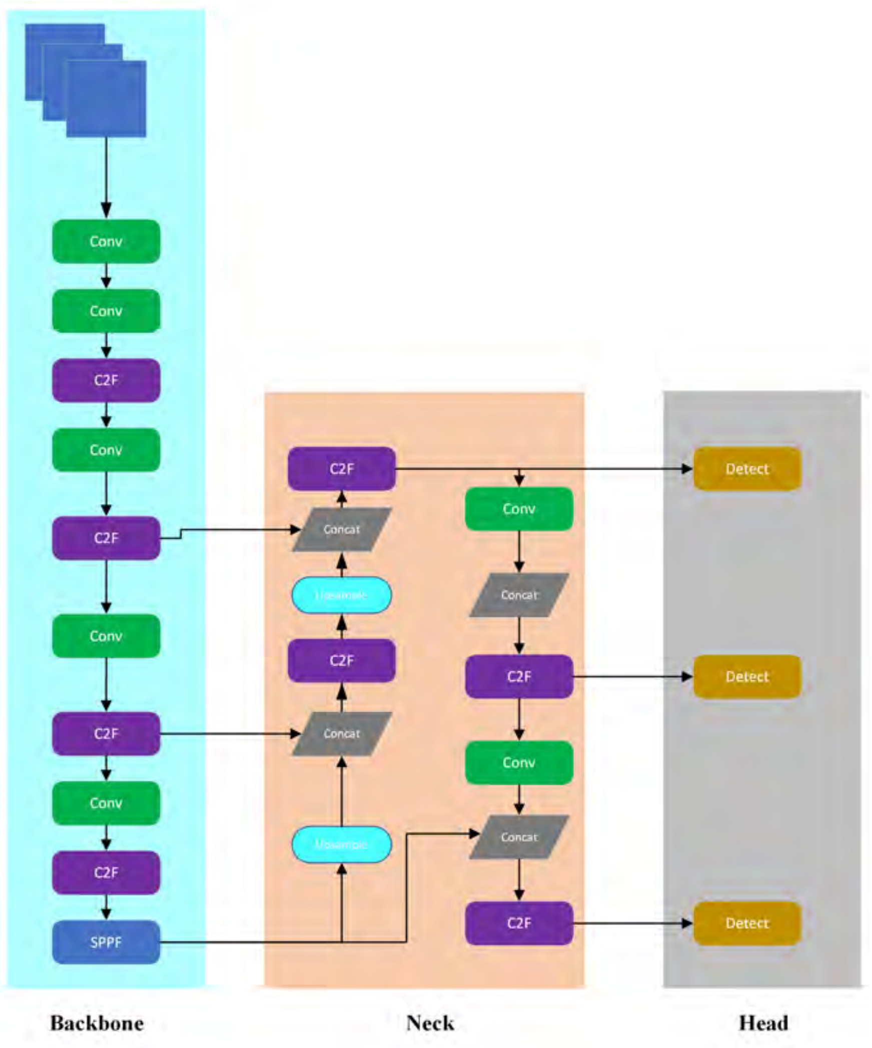


Figure 3.16: YOLO v8 structure

$$\Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i|\text{Obj}) * \Pr(\text{Obj}) * \text{IOU}_{\text{pred}}^{\text{truth}} \tag{3.20}$$

where

$\Pr(\text{Obj}) * \text{IOU}_{\text{pred}}^{\text{truth}}$ is the confidence of the object,

$\Pr(\text{Class}_i|\text{Obj})$ is the conditional class probabilities of each grid (consider the grid cell containing an object).

– Grasping Posture Analysis

The grasping posture analysis is a critical part of vision grasping. In this project, the Libot adopted the Grasp Pose Detection (GPD) Algorithm to detect 6-DoF grasp poses based on 3D point clouds [43]. Using the two-finger robot manipulator, this algorithm has the main advantage compared to the traditional method, i.e., the GPD algorithm can produce different grasps without the real CAD files. Its mechanism enables it to run in the deep cluster, which is beneficial for grasping different postures of books in the book in real scenarios. Also, according to the work of Gualtieri et al., the program on a Baxter Research Robot achieved the average grasp success rate of 93% for the dense clutter environment, which implies its high accuracy [43].

The GPD algorithm includes two steps: selecting a vast number of candidates for grasps and determining whether or not they are feasible grasps. In the first step, the points are selected randomly from parts of the point cloud containing objects to be grasped, $\mathcal{C}_{\mathcal{G}}$. Then, the surface is normal, and an axis of major principle curvature of the object surface of adjacent points is figured out. Similarly to the work of Kappler et al. [48], orientations orthogonal to the curvature axis are regarded as the main direction to move the hand toward the point cloud without collision. Checking whether the point cloud is included is required after obtaining the contact configuration; if not, we can delete them and keep the desired grasping candidates.

Step two comes with a four-layer convolution neural network (CNN).

This kind of CNN is actually from LeNet [49] with two convolutional or pooling layers followed by one inner product layer. This step aims to evaluate whether the grasp candidates are valid or not, which affects the final grasp choice. The information of the part of the object (geometry and appearance) is encoded to train into a CNN network. The input of the CNN is a multi-view encoded representation of the object's geometric information; i.e., the voxelized point cloud information is projected onto the orthogonal planes of the X, Y, and Z axes of the robot hand reference coordinate system. For each projection direction, The average height map of the observation point, the average height map of the occlusion point, and the average map of the surface normal direction are classified by CNN, and the output is whether the grasp candidate is a valid grasp, as shown in Figure 3.17.

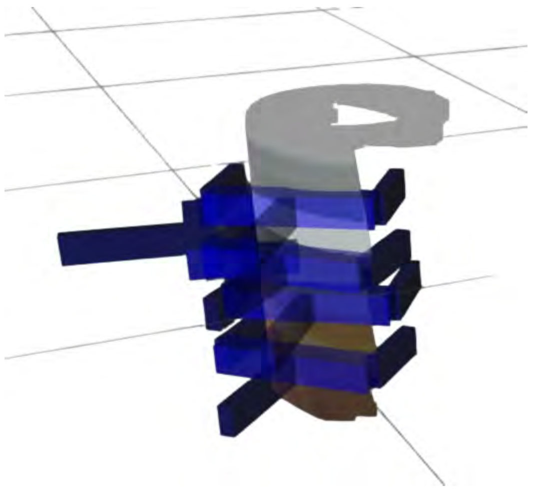


Figure 3.17: Generated grasps using GPD

- Motion Planning and Task Integration

Figure 3.18 shows the conceptual program design of Libot. There are four

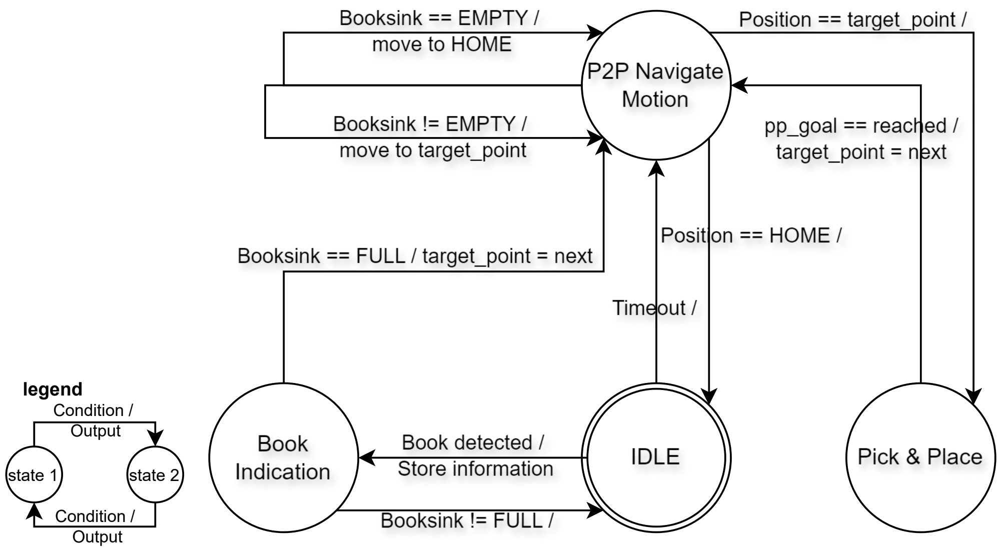

Booksink == EMPTY / move to HOME
P2P Navigate Motion
Position == target_point /
Booksink != EMPTY / move to target_point
pp_goal == reached / target_point = next
Position == HOME /
Booksink == FULL / target_point = next
Timeout /
legend
Condition / Output
state 1
state 2
Condition / Output
Book Indication
Book detected / Store information
IDLE
Pick & Place
Booksink != FULL /

Figure 3.18: Finite-State Machine diagram of Libot's working logic

presented states: `IDLE`, `Book_Indication`, `P2P_Navigation` and `Pick_Place`.

1. `IDLE`: IDLE means Libot not being used or active. No task is execute during Libot's `IDLE` state
2. `Book_Indication`: Detection of book placed in booksink can trigger the transition from `IDLE` state to `Book_Indication` state. This transition will output tag information recognized by our vision model 3.16 of the newly placed book and stored in a shared memory. Ultimately, the tag info will be compared with a library database then return the location information of the book, Indication task completes by this point.
3. `P2P_Navigation`: This state execute the task of the navigation to a given point. It has three entries corresponding to three other states. From `Book_Indication` to this state requires the condition of the booksink is full, the first location point which obtained from indication task store in a queue (shared memory) will be assign to a variable name `target_point` which is the point will be given to navigation task.

Direct transition from IDLE to this state is possible, this design is to make Libot work at a certain frequency. For example, the timeout value can set to be 4 hours, with this entry to navigation, Libot will be in return book operation every 4 hours even the booksink is not full. Once it enters P2P_Navigation from IDLE, a re-entry will be trigger with the condition of the booksink is not full, it will assign target_point in the way mentioned earlier.

4. Pick_Place: After the complement of the navigation task, Libot will be in this state to perform Pick and Place task, upon the completion of this task, it will returns to P2P_Navigation state. Detailed description of Pick and Place task will be discussed later in this section.

Once the Libot return all the books after the transition between P2P_Navigation and Pick_Place state, the emptiness of booksink will trigger the re-entry of assigning the target_point to HOME position. Libot will move to it's HOME position first and turn to IDLE again.

Pick and Place task

The "Pick and Place" task is a fundamental operation that Libot perform object manipulation tasks, specifically in library environments. In the designated state known as Pick_Place, Libot is tasked with the transfer of books between two specific points: the booksink and the bookshelf.

This operation involves two distinct approaches:

1. Fixed Position Grasping: Initially, Libot identifies and grabs a book placed in a predetermined, fixed position within the block of the booksink.

This method relies on pre-programmed joint states within Libot's operational environment, minimizing the need for real-time positional adjustments.

2. Dynamic Grasping Using GPD Algorithm: More advanced than the fixed position approach, this method utilizes Libot's depth sensors coupled with a GPD algorithm. The GPD algorithm processes the input from the depth sensors to analyze the spatial orientation and positioning of books. Based on this analysis, Libot dynamically calculates the optimal grasp for each book, regardless of its initial position. This approach significantly enhances Libot's adaptability to varied book placements and orientations, which is crucial in less structured environments like disorganized book returns.

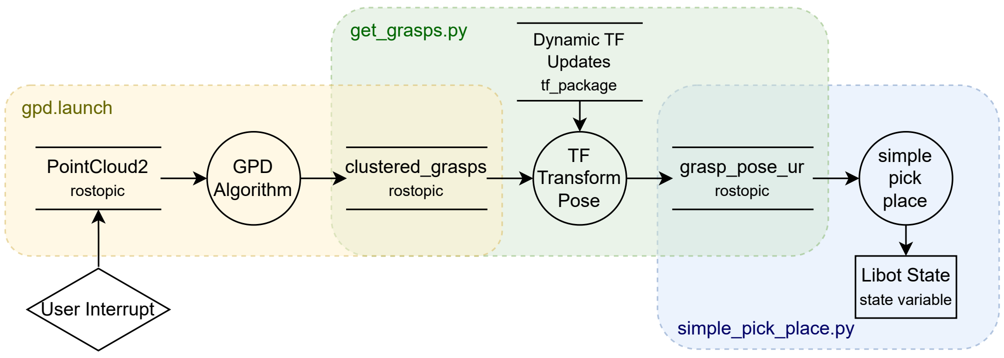


Figure 3.19: Task diagram of GPD pick & place

Figure 3.19 presents a schematic overview of the dynamic grasping process during the `Pick_Place` state through a task flow diagram. The diagram begins with an operator-initiated user interrupt or an automated trigger from the `P2P_Navigation` state, represented by a isometric square shape, which activates the grasping sequence.

Within this sequence, there are three core subprocesses outlined:

1. GPD Algorithm: This module processes the `PointCloud2` data, received as a ROS topic, to compute potential grasps. These grasps are identified and then published as a `clustered_grasps` topic, which aggregates the grasping points identified by the GPD algorithm.

2. TF Transform Pose: Subsequent to grasp identification, the `clustered_grasps` must undergo a transformation to align with the arm of Libot's operational frame of reference. This step involves converting the grasps from the `camera_depth_optical_frame` to the `booksink_link`, which serves as the reference frame for the MoveIt motion planning framework. This transformation is pivotal as it ensures the grasps are contextualized within the robot's spatial understanding and MoveIt's planning pipeline.

3. Simple Pick Place Task: The final module, depicted as a rounded rectangle, utilizes the transformed grasp data to inform the motion planning for the UR5 robotic arm. This `simple_pick_place.py` script orchestrates the physical actions required for the robotic arm to move to the desired positions and execute the pick and place task, finally returns a new state to `Libot State` in a rectangle which represents the shared memory that can be pass to and edit in other state.

The diagram highlights the information flow through partial rectangles, indicating the buffering and exchange of information via ROS topics between tasks. This structured flow ensures that each subprocess receives the necessary input to perform its function effectively within the overall grasping task.

Figure 3.17 breaks down the complex grasping task into a series of interconnected steps, from initial interrupt to the execution of a pick and place action, detailing the data transformations and communications required to accomplish dynamic grasping with Libot in its operational environment.

3.2.1.4 Navigation

MiR is a commercial product which has its own mapping and navigation technology. However, in this project, Libot uses open-sourced system ROS and open-sourced packages created for ROS to perform the navigation task. The core in this navigation task is the `move_base` node initialized by `ros_navigation` package [27], further enhanced by an approach called hector mapping [30] to SLAM with the help of a LiDAR and depth cameras on Libot's MiR base.

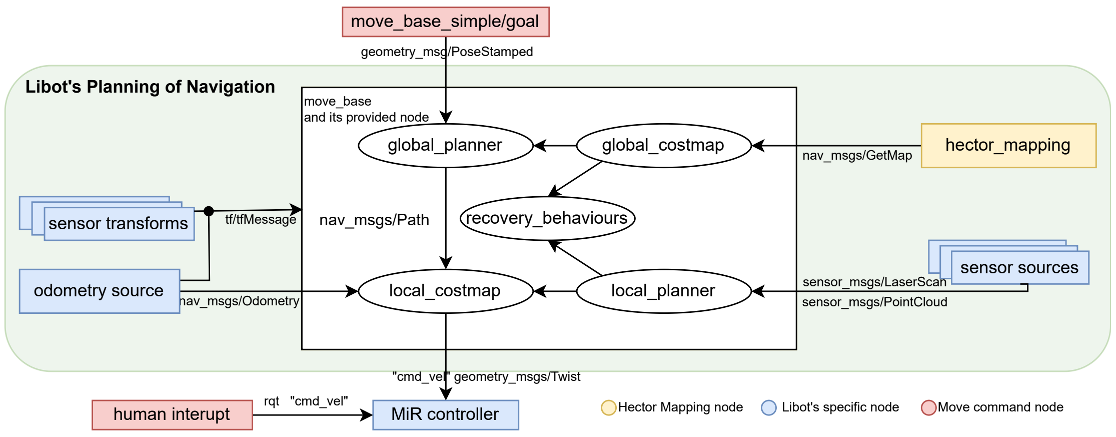


Figure 3.20: ROS structure of Libot's navigation [27]

In the process of Libot's navigation task, the flow of information is integral for Libot to understand its environment, plan its path, and move accordingly. Here's a step-by-step breakdown of the information flow during navigation, with reference to the provided Figure 3.20.

1. Sensory Input: Libot's journey begins with gathering data about its surroundings. This information comes from two primary sources:

   - Odometry Source: This provides data about Libot's movement over time, such as how far and fast it's traveled, based on encoders in its wheels or other motion sensors.

   - Sensor Sources: These are the inputs from the robot's perception hardware, like LIDAR or depth cameras, which provide `sensor_msgs/LaserScan` and `sensor_msgs/PointCloud` data.

2. Sensor Transforms: The data from various sensors are then transformed using `tf/tfMessage`, which helps maintain the relationship between coordinate frames. This step is crucial as it aligns all sensory data to a common reference frame, allowing Libot to have a coherent understanding of its environment relative to its current position and orientation.

3. Costmaps Generation: Both the `global_costmap` and `local_costmap` are updated with the transformed sensor data. The `global_costmap` reflects the overall environment, usually based on a pre-existing map, while the `local_costmap` is more dynamic, reflecting immediate obstacles and changes around Libot.

4. Global Planner: Utilizing the `global_costmap`, the `global_planner` computes an initial path to the destination. This path is laid out as a series of waypoints which are then passed down as `nav_msgs/Path` to the `local_planner`.

5. Local Planner: The `local_planner` refines the path provided by the `global_planner`, considering the real-time data from the `local_costmap`. It generates a short-term path that Libot can follow safely, adjusting for unexpected

obstacles and ensuring the robot's movements are kinematically feasible.

6. Execution of Movements: Commands for movement, encapsulated in `cmd_vel` messages of type `geometry_msgs/Twist`, are sent from the `local_planner` to the MiR controller. This component is responsible for the direct control of Libot's actuators, translating the velocity commands into physical motion.

7. Hector Mapping: In parallel, the `hector_mapping` node performs SLAM to update the map of the environment and locate Libot within it. This information is also used to update the `global_costmap` and aids in continuous navigation.

8. Recovery Behaviours: If at any point, Libot encounters a problem, like an impassable obstacle or a localization error, the `recovery_behaviours` are triggered. These behaviors aim to resolve the issue by re-planning a path or clearing the costmaps to restart the mapping process.

9. Human Interaction: Throughout this process, there's the potential for a human operator to intervene, either by providing new commands or by interrupting the current operation.

The flow of information through these components ensures Libot can navigate autonomously and efficiently, adapting to dynamic changes in its environment and fulfilling the task at hand.

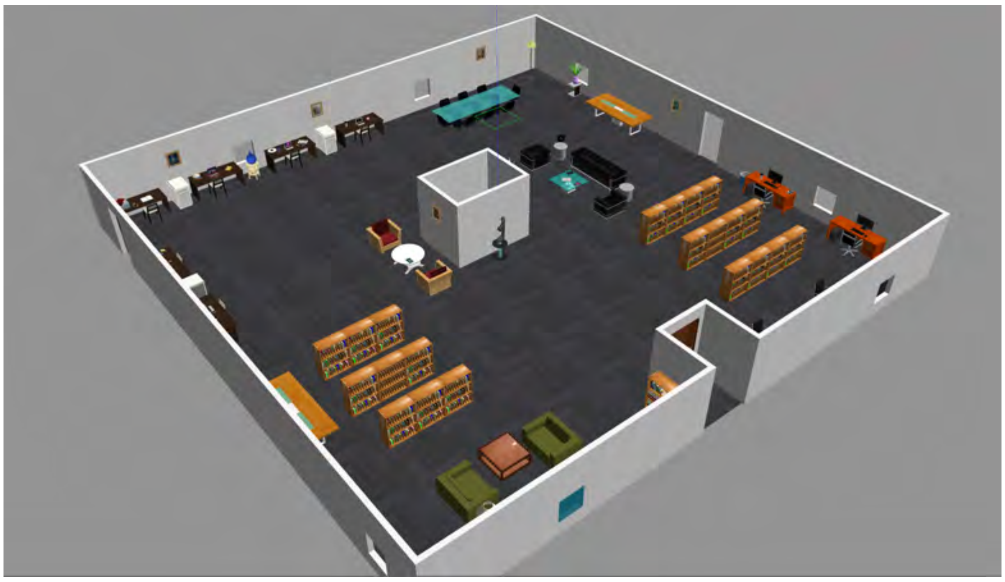
3.2.2 Simulation Environment

In robotics research, Gazebo (Figure 3.21) is a very important simulation tool that allows for the emulation of robots' activities, even for the most complex systems, in a well-represented three-dimensional environment. It draws its strength from enabling the correct duplicate of physics from reality. This capability is also instrumental in the development of robotic algorithms and systems and will have been tested in advance of the deployment of these on real, manifested, physical robots.

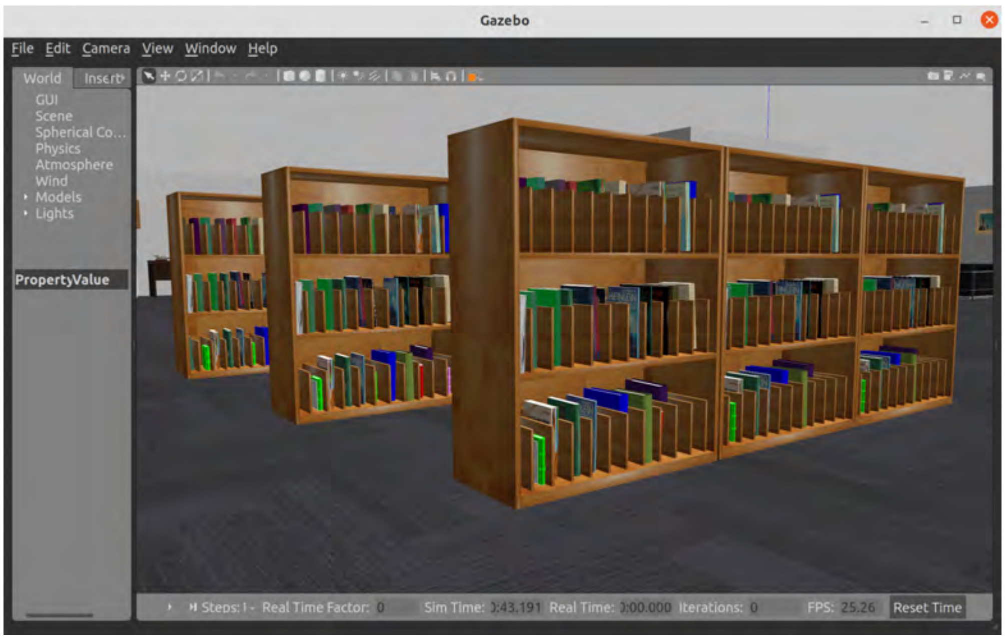
Gazebo gives a hard-coded platform in which physical parameters like mass, inertia, friction, and collision, among others, can be defined and even finetuned.

This is the simulation-driven physics, which contains powerful engines such as ODE (Open Dynamics Engine), Bullet, Simbody, and DART to really control the same in the dynamic interactions of the robot with the environment during the simulations. A key part of our project involved the integration of Gazebo with ROS, which was facilitated by `gazebo_ros_pkgs`. Such integration allows an easy sync between the simulated environment and ROS functionalities. In our project, particularly, we have used this feature to manipulate and control our Libot.

URDF defines the physical characteristics and behavior of the Libot in the simulation environment. This form of XML format enabled us to detailed describe the physical parameters of Libot : the distribution of weight, joint constraints, and values of the body's dimensions. These are used by Gazebo in simulating the physical interactions the robot exhibits with the environment. We set those parameters in URDF with an aim to simulate as closely as possible real-world conditions, in order to get reliable data for the further development and refinement



(a)


Gazebo
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0 Sim Time: 3:43.191 Real Time: 0:00.000 Iterations: 0 FPS: 25.26 Reset Time

(b)

Figure 3.21: Gazebo simulation of library environment

of Libot functionalities.

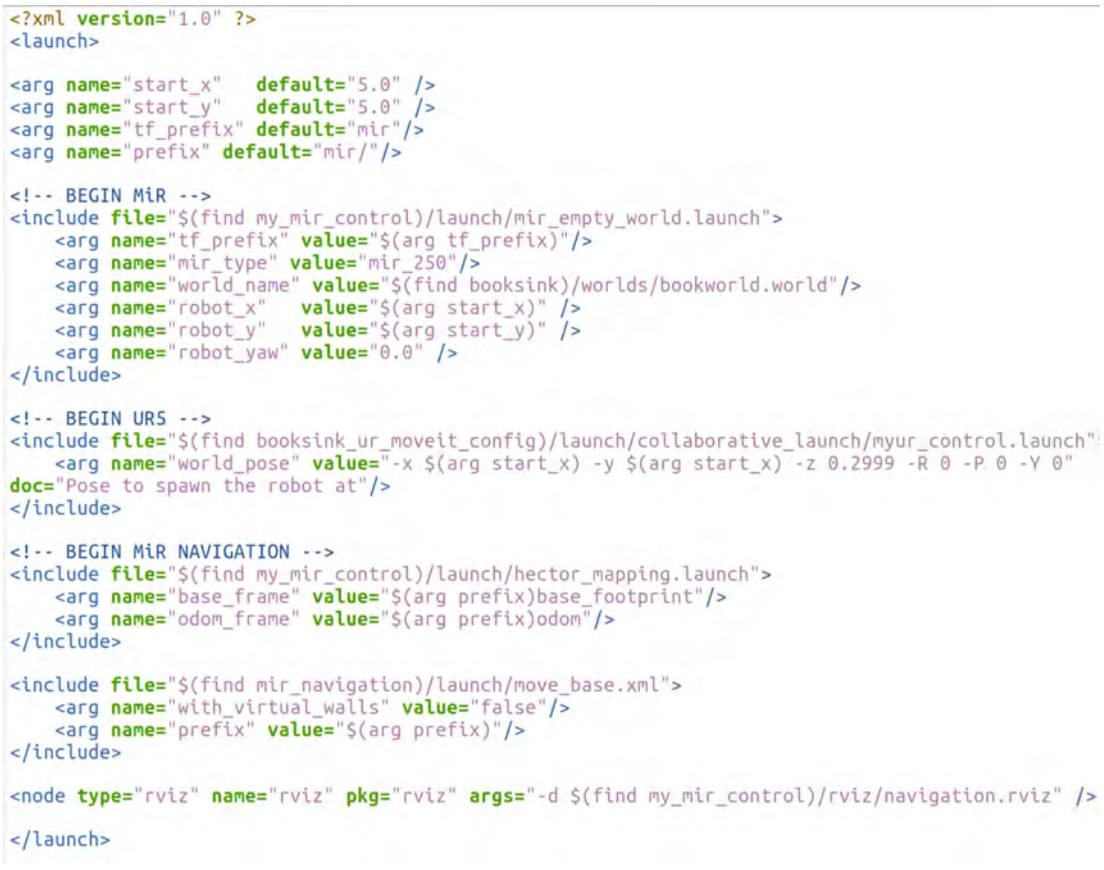
In summary, Gazebo is a lighthouse tool that provides a meager set of functionality. All the URDF specifications details, Gazebo and ROS helped expand limits in robotics, opening opportunities for such advanced robotic systems that would be both efficient and effective in their appointed tasks.

Launch File: Launch the simulation in one step

To start the simulation of Libot operation, multiple ROS nodes need to be start to run manually, however some of the work can be simplified by writing a launch file. The launch is written in XML language and can be exceute in terminal by running `roslaunch` command. Here, we present one of a launch file (show in Figure 3.22) responsible for

- Spawn Libot's URDF into ROS parameter server
- Start Gazebo simulator
- Start controllers for UR5, Robotiq, and MiR and their corresponding RViz interface.

Note that this shown launch file will gathering command from its included launch files when it is being execute, for other relating launch files, please refer to Appendix C.



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

<arg name="start_x"   default="5.0" />
<arg name="start_y"   default="5.0" />
<arg name="tf_prefix" default="mir"/>
<arg name="prefix" default="mir/"/>

<!-- BEGIN MiR -->
<include file="$(find my_mir_control)/launch/mir_empty_world.launch">
    <arg name="tf_prefix" value="$(arg tf_prefix)"/>
    <arg name="mir_type" value="mir_250"/>
    <arg name="world_name" value="$(find booksink)/worlds/bookworld.world"/>
    <arg name="robot_x"   value="$(arg start_x)" />
    <arg name="robot_y"   value="$(arg start_y)" />
    <arg name="robot_yaw" value="0.0" />
</include>

<!-- BEGIN UR5 -->
<include file="$(find booksink_ur_moveit_config)/launch/collaborative_launch/myur_control.launch"
    <arg name="world_pose" value="-x $(arg start_x) -y $(arg start_x) -z 0.2999 -R 0 -P 0 -Y 0"
doc="Pose to spawn the robot at"/>
</include>

<!-- BEGIN MiR NAVIGATION -->
<include file="$(find my_mir_control)/launch/hector_mapping.launch">
    <arg name="base_frame" value="$(arg prefix)base_footprint"/>
    <arg name="odom_frame" value="$(arg prefix)odom"/>
</include>

<include file="$(find mir_navigation)/launch/move_base.xml">
    <arg name="with_virtual_walls" value="false"/>
    <arg name="prefix" value="$(arg prefix)"/>
</include>

<node type="rviz" name="rviz" pkg="rviz" args="-d $(find my_mir_control)/rviz/navigation.rviz" />

</launch>
```

Figure 3.22: Simulation launch file

CHAPTER 4 : RESULT AND DISCUSSION

4.1 YOLO-V8 EXPERIMENTAL RESULT

4.1.1 Dataset

In this project, our Libot can detect information about books via format book spine labels for public library use. The book's label can help us find where books should be placed through the library's database. Therefore, to ensure the data quality, we arbitrarily sampled the over 300 book spines from University of Macau Library. The purpose of the experiment was to verify that the method could detect the book with high accuracy, so we took about 1000 photos under different lighting and angle conditions. After manual annotation on the *Roboflow* website, there are 312 classes in terms of the strings of each book, like "B" 2005" L484" Mes".

Then, the dataset was classified into three categories according to a certain proportion: 70 percent of the training set, 20 percent of the verification set, and 10 percent of the test set in case of occurrence of overfitting. The training set is used to train the model's internal parameters, and the validation set is responsible for testing the status and convergence of the model during the training process. Based on the performance of several sets of model validation sets, which set of hyperparameters has the best performance is determined. The test set is used to evaluate the model's generalization ability, using a data set that has never been seen before to determine whether the model works.

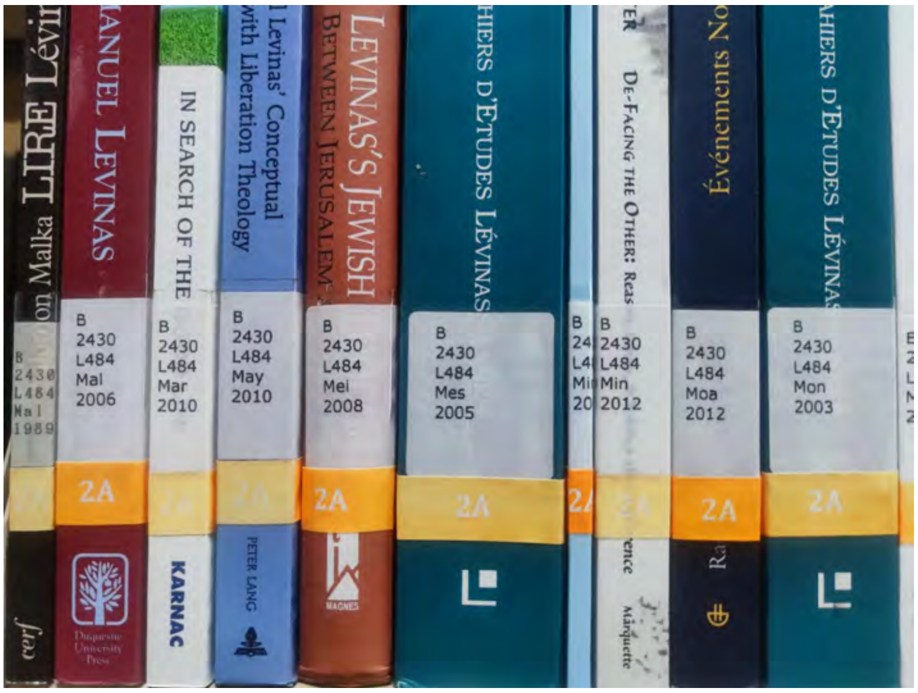


Figure 4.1: Sample books

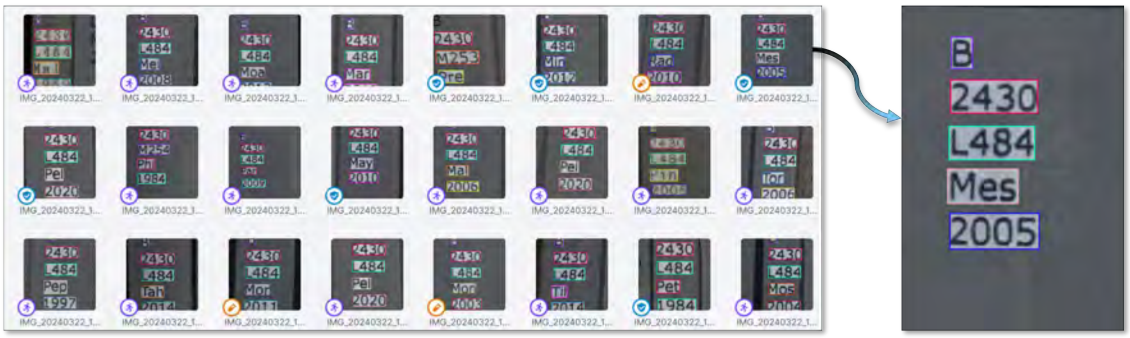


Figure 4.2: Annotated database

4.1.2 Training Environment

In this project, we used the YOLOv8l to train the model without the pre-trained weights, and the whole process was conducted for 195 epochs, 2 batch sizes with an input image size of modified 640 × 640 pixels. The hardware environment parameters used in the experimental training step are shown in Table 4.1. Besides, the Libot adopted the `ultralytics_ros` Package [50] as the real-time object detection method based on the Ultralytics YOLO model.

Table 4.1: Hardware environment parameters of the training process

Parameters	Configuration
CPU	Intel i7-12700H
GPU	NVIDIA GeForce GTX 3060 8GB
RAM	16GB
OS	Ubuntu 20.04
Python	3.8
Framework	Ultralytics YOLO v8.1.47 , PyTorch 2.2.2+Anaconda 3 2024.02

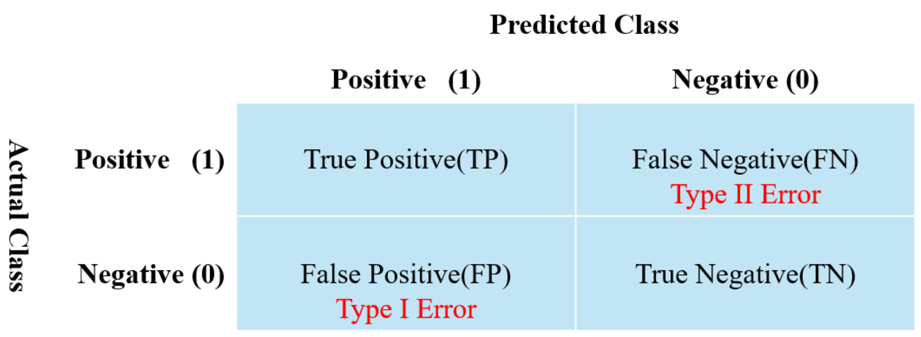
4.1.3 Evaluation Metrics

Several parameters are mainly adopted for the YOLO-based detection task: precision(P), recall(R), F1 score, and average precision(AP). Precision (P) is a measure of accuracy that represents the proportion of examples classified as positive that are true, which can be defined as:

$$P = \frac{TP}{TP + FP} \times 100\% \tag{4.1}$$

where TP is True Positive, FP is False Positive, both of which are illustrated in Figure 4.3.

Recall (R) starts from the perspective of real results and describes how many



Actual Class	Predicted Class: Positive (1)	Predicted Class: Negative (0)
Positive (1)	True Positive(TP)	False Negative(FN) Type II Error
Negative (0)	False Positive(FP) Type I Error	True Negative(TN)

Figure 4.3: Illustration of Confusion Matrix for detection objects

real positive examples in the test set were selected by the two classifiers, i.e., how many real positive examples were recalled by the two classifiers,which can be described as shown in Equation 4.2.

$$R = \frac{TP}{TP + FN} \times 100\% \tag{4.2}$$

where FN stands for False Negative, also shown in Figure 4.3

The F1 score is the harmonic mean of precision and recall, with a maximum of 1, where 1 is the best and 0 is the worst. It is defined as:

$$F_1 - Score = 2 \cdot \frac{\mathrm{P} \ \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{4.3}$$

This parameter can control recall and precision at the same time to evaluate the quality of the model.

AP is the abbreviation of Average Precision. It is the area under the Precision-Recall curve (P-R curve), which can be calculated according to Equation 4.4:

$$\mathrm{AP} = \int_0^1 \mathrm{P_i}(\mathrm{R_i})\mathrm{dR_i} \tag{4.4}$$

where P_i is the accuracy of identifying the i-th object and P_i denotes the recall of the i-th object.

And the Mean Average Precision (mAP) is the average AP of the each class. This parameter can be solved by Equation 4.5.

$$mAP = \frac{\sum_{i=1}^{N} AP_i}{N} \times 100\% \tag{4.5}$$

where AP_i represents the average precision of the i-th class, and N is the total class number.

In this section, the the mean average precision at an intersection over union (IoU) threshold of 0.5 (mAP50) and mean average precision across IoU thresholds ranging from 0.5 to 0.95 (mAP50-95) are adopted as two of the evaluation parameters to check the trained weight is precision by comparing the bound box.

4.1.4 Detection Result and Discussion

Here, dectection results trained by YOLOv8l model are illustrated in Table 4.2. The model showcases its capability to effectively detect objects with various degrees of overlapping bounding boxes, as evidenced by an overall class mAP50 of 92.3% and a noteworthy mAP50-95 of 82.8% among all string classes. This highlights the model's ability in precisionly capturing varying string object instances. Also, "2003" "2012" shows good performance in accuracy with 99.5% . Nevertheless, "Col" "Cow" perform poor in the accuracy in terms of their mAP values. The main reason for the low accuracy of some classes is that the string was not captured in enough training sets, which is also caused by the fact that the string is too sparse in the library. For the result of all classes, please refer to Appendix C.

Table 4.2: Detection result using YOLOv8l Model

Class	Precision/%	Recall/%	mAP50/%	mAP50-95/%
All	81.3	81	92.3	82.8
"2003"	98.7	60.2	99.5	85.9
"2012"	96.7	69.5	99.5	90.8
"2015"	96.5	93.1	97.3	96.4
"2430"	99.7	98.7	99.5	91.7
"B"	97.8	98.4	99.1	82.6
"Col"	13.8	20.7	49.7	43.1
"Cow"	72.3	22.4	35.9	33.0
"D454"	95.7	100	92.3	92.1
"R554"	98.6	92.7	91.4	86.7

The visual detection results for static data collected from the library are displayed in Figure 4.4, showing the perfect ability of the YOLO-v8l model to identify various strings on the book spine. Strong generalization capabilities are also displayed in this model. However, there is still also some false detection appearing. For example, in Figure 4.4, "G474" in the first column of the second row was mistakenly recognized as "D454", and "S752" in the fourth column of the third row was mistakenly recognized as "S554". These problems may also be attributed to the insufficient number of samples these classes learn from images.

Figure 4.5a shows F1-Confidence curves for all classes in our book-label string datasets. These curves depict the relationship between the confidence threshold and the F1 score, which measures the balance between precision and recall. When the confidence threshold is set low, a significant number of low-confidence samples are classified as true positives, leading to a high recall rate but a low precision rate. Conversely, a high confidence threshold only considers samples with high confidence as true positives. This means that higher confidence is necessary for

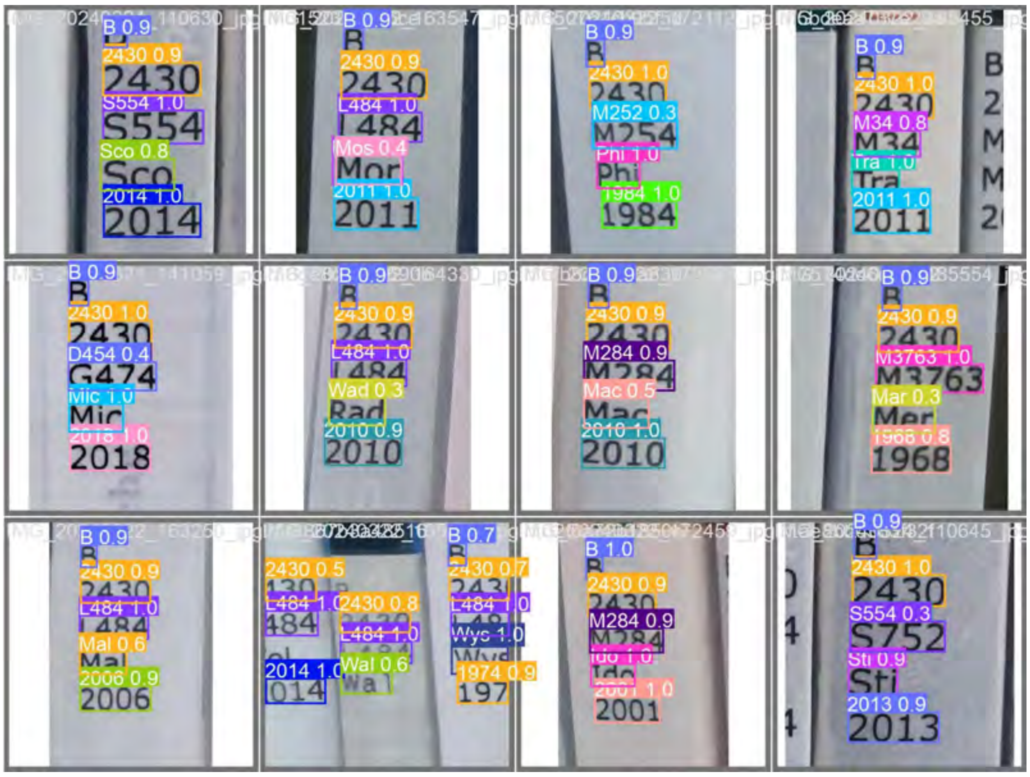
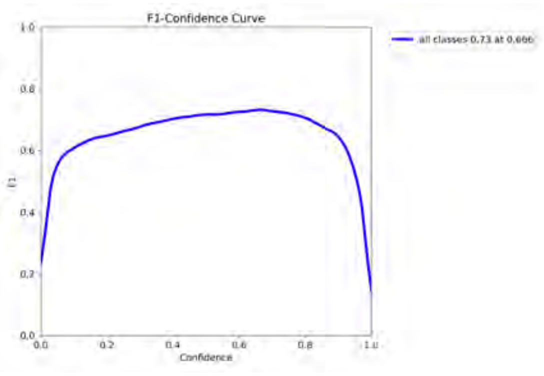


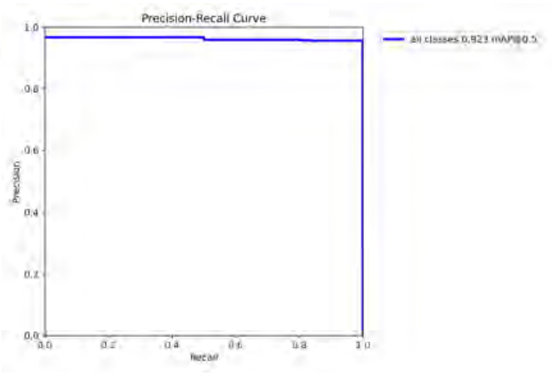
Figure 4.4: The result of proposed model for static label detection

precise category detection, resulting in lower F1 scores at both ends of the confidence spectrum. The figure indicates that the best F1 scores can be achieved within the confidence range of 0.6-0.8. In this range, the category detection is more accurate, as samples with higher confidence are classified correctly, resulting in a better balance between precision and recall.

The Precision-Recall curve (P-R curve) is shown in Figure 4.6. The P-R curve reflects the relationship between precision and recall. As mentioned in Chapter 4.1, the area enclosed under the P-R curve (blue curve) is mAP. The curve in this figure after training is close to (1,1), achieving a higher accuracy.



(a) F1 Curve



(b) P-R Curve

Figure 4.5: F1 and P-R Curve for all classes on the collected dataset

The evolution of several parameters, such as box loss, object loss, class loss, and metrics at the end of each epoch, such as accuracy and recall, is shown in Figure 4.6 during training and validation. As can be seen from the figure, the precision gradually approaches 0.8 and the recall rate gradually converges to around 0.8 after 100 epochs. The fluctuation situation is asymptotically stable, indicating that there is a better performance on the training effect.

The simulated real-time book recognition results on the booksink of the Libot

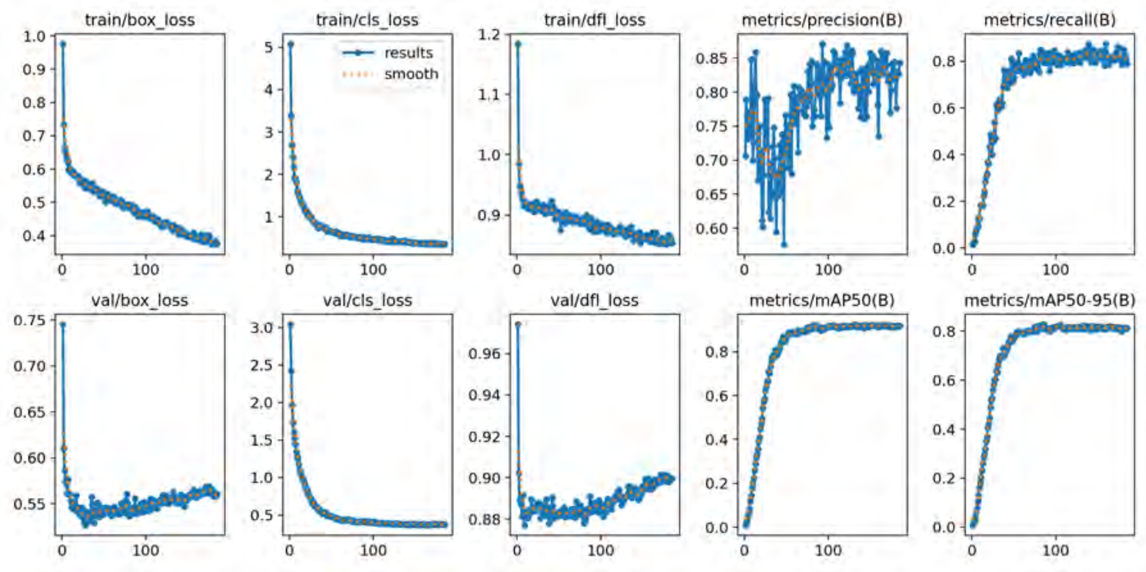


Figure 4.6: Network convergence on the collected dataset

by `ultralytics_ros` Package [50] are also illustrated in Figure 4.7. Although some real scenarios like shades and occlusions may affect the detection results, the overall effectiveness is satisfied, showing the robustness and anti-interference ability with relatively high accuracy.

Overall, the trained YOLO-v8 model demonstrated a high degree of performance in detecting various classes of book spines, although some of the classes may not be satisfactory due to the incomplete dataset. Real-world scenarios such as darker or extreme shaded environments may also influence recognition accuracy.

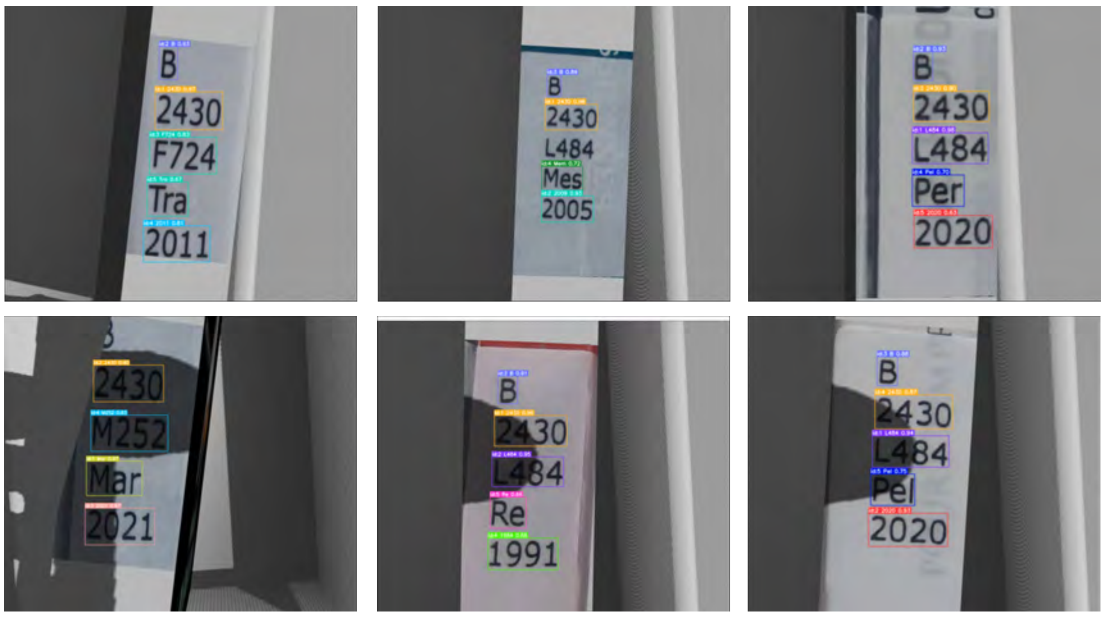
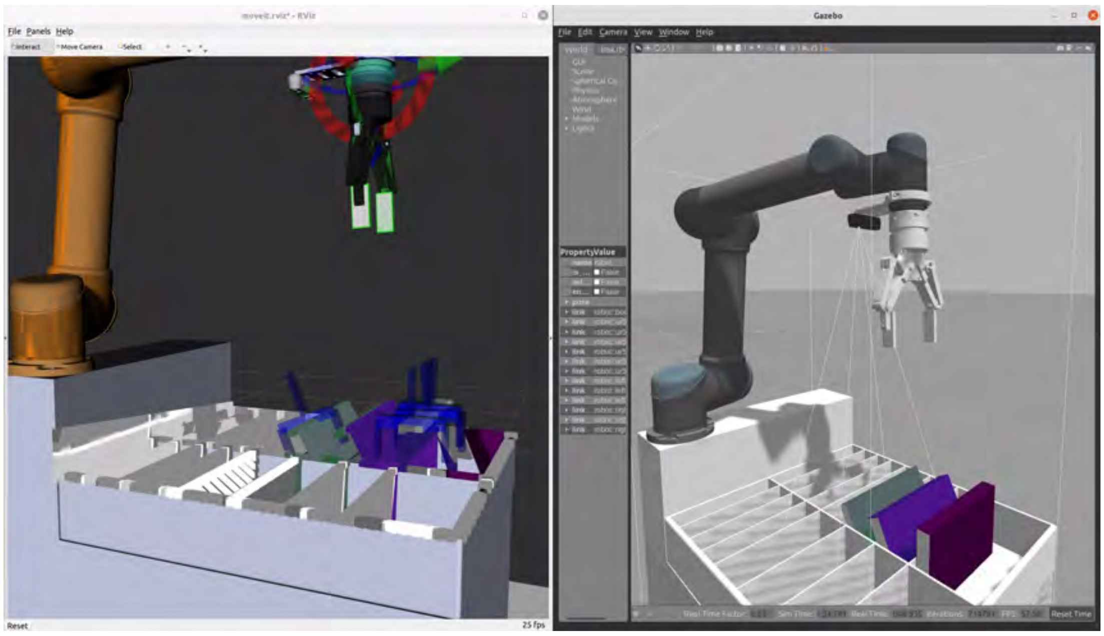


Figure 4.7: Screenshots of real-time detection results in the Gazebo environment

4.2 GRASP TASK AND GRASP POSE DETECTION

Simulation Outcomes

The simulation performed using ROS, with the visual tools of Rviz and Gazebo, provided a clear visualization of the robot's ability to detect and execute grasp poses. As depicted in the Figure 4.8(a), the RViz is visualizing the point cloud and the generated grasp pose by GPD algorithm, while the Gazebo on the right side of the figure shows the simulation environment which mimics the world.



(a)

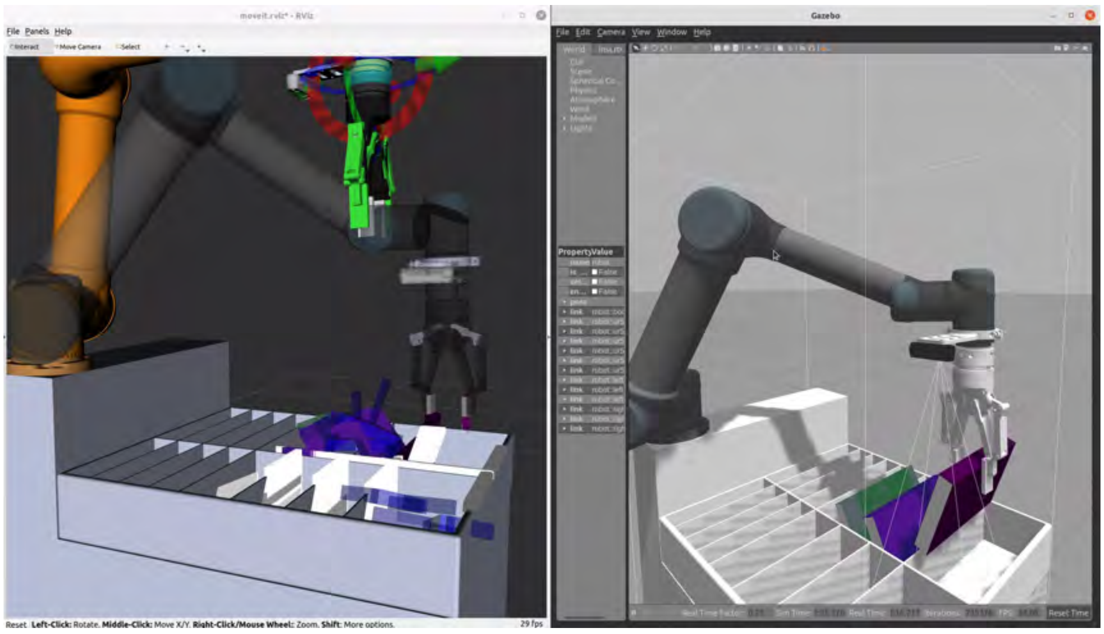

moveit.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
29 fps
Gazebo
File Edit Camera View Window Help
(b)

Figure 4.8: GPD result

Analysis of Grasp Pose Detection

The process begins with the robot scanning the environment to generate a detailed 3D map composed of point clouds. These point clouds are then processed to identify potential grasping points. Each point is evaluated based on predefined metrics such as surface normals and curvature, which are essential for determining the feasibility of a successful grasp.

In the scenario shown in the Figure 4.8(b), the robot successfully identifies and aligns its gripper with the optimal grasp point, as indicated by the alignment of the gripper's orientation with the highlighted blue area.

Discussion

The successful detection and execution of grasp poses in a simulated environment underscore the potential of integrating advanced machine learning techniques with robotic systems. However, the simulation also highlights several challenges and areas for improvement:

- **Computational Demand:** The processing of high-resolution 3D point clouds requires significant computational resources, which can limit the speed and efficiency of the task execution.

- **Real-world Application:** While the simulation provides valuable insights, the transition from a controlled environment to real-world applications involves additional challenges, such as dealing with unexpected obstacles and variable object textures, which can affect sensor performance.

- **Adaptability and Learning:** The current system performs well within the predefined scenarios but integrating adaptive learning mechanisms could enhance the robot's performance by enabling it to learn from past experiences and adjust its grasp strategies accordingly.

The grasp pose detection simulation demonstrates the feasibility of sophisticated robotic manipulation in controlled environments.

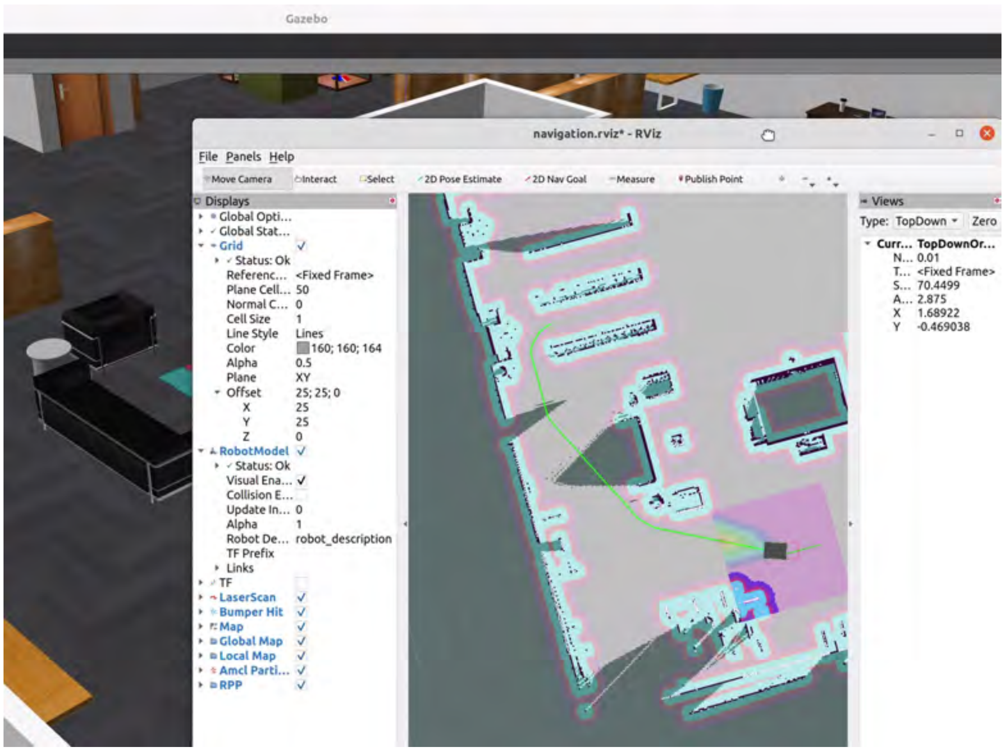
4.3 HECTOR MAPPING AND NAVIGATION

Hector mapping is an approach for real-time indoor mapping and navigation that does not require odometry data. This method utilizes a fast and robust scan matching algorithm to estimate the robot's pose which is especially effective in environments with distinctive geometric structures. As demonstrated in our setup, Hector mapping effectively utilizes the laser scans provided by the robot's LiDAR sensor to dynamically create a detailed map of the environment.

Implementation and Results

The implementation of Hector mapping in our study involved integrating it with the ROS navigation stack. The process begins with the robot autonomously scanning the surrounding area. The ROS navigation stack processes these scans to iteratively construct and update a 2D occupancy grid map, as shown in the Figure 4.9(a). This map indicates open areas, obstacles, and unknown spaces with different shades, effectively representing the spatial layout.

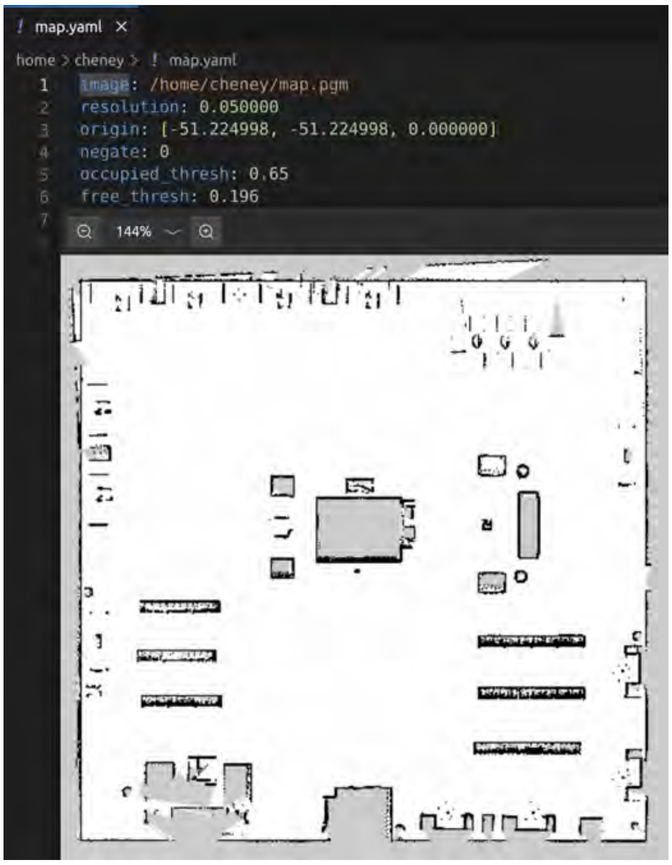
As the robot navigates through the environment, it continuously updates this map, which also assists in path planning and obstacle avoidance. Figure 4.9(b) captures a typical navigation scenario where the robot was tasked with reaching a specific location (2D Nav Goal) in Rviz. The path planning module calculates an optimal path shown in green, which dynamically adjusts as the robot moves and the map



```
Displays
  Global Opti...
  Global Stat...
  Grid                 ✓
    Status: Ok
    Referenc...        <Fixed Frame>
    Plane Cell...      50
    Normal C...        0
    Cell Size          1
    Line Style         Lines
    Color              160; 160; 164
    Alpha              0.5
    Plane              XY
    Offset             25; 25; 0
      X                25
      Y                25
      Z                0
  RobotModel           ✓
    Status: Ok
    Visual Ena...      ✓
    Collision E...
    Update In...       0
    Alpha              1
    Robot De...        robot_description
    TF Prefix
    Links
  TF
  LaserScan            ✓
  Bumper Hit           ✓
  Map                  ✓
  Global Map           ✓
  Local Map            ✓
  Amcl Parti...        ✓
  RPP                  ✓
```

```
Views
Type: TopDown   Zero
  Curr...  TopDownOr...
    N...   0.01
    T...   <Fixed Frame>
    S...   70.4499
    A...   2.875
    X      1.68922
    Y      -0.469038
```

(a)



```yaml
image: /home/cheney/map.pgm
resolution: 0.050000
origin: [-51.224998, -51.224998, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

(b)

Figure 4.9: Result of hector mapping and route planning

updates.

Discussion

The navigation system's efficacy is evidenced by its precise mapping and real-time responsiveness to environmental changes. The ability to dynamically adjust paths in response to real-time obstacle detection illustrates the robustness of Hector mapping integrated with ROS navigation. However, challenges such as dealing with dynamic obstacles or abrupt environmental changes remain areas for further research.

CHAPTER 5 : CONCLUSION

Libot, a MMR designed for the automation of library services. Throughout this study, Libot demonstrated smooth navigation and manipulation within a complex library environment, underscoring the efficacy of its integrated sensor suite and control strategies.

5.1 KEY OUTCOMES

A complete robot was created from scratch in the form of URDF. Including the booksink part which is built in Solidworks and transferred into a ROS package that can be included in Libot's URDF.

Successfully implementing MoveIt as the manipulation planning framework. MoveIt Setup Assistance is used for creating the package that initializes the ros controllers for UR5 and Robotiq move groups. Later, the `Move Group Interface (Python)` is used to form a Python script to generate motion planning path for pick & place task.

High accuracy YOLO v8 Model as the object detection method to differentiate books using `ultralytics_ros` package. The book spine dataset from the UM library was trained in the Yolov8l model. The trained model demonstrated high performance on static detection and sim-time simulated environment validation.

Grasp Pose Detection outputs visualized book grasp information in RViz and helps performed grasp by MoveIt. The GPD is an open-sourced algorithm but is hard to implement. Environment dependencies have been resolved on both testing computers in this project. Python scripts are created which are responsible for

- Initializing a node that subscribes to the GPD output topic
- Post-process the message from the topic into `Stamped_Pose` message, and perform the transform from camera frame to Libot's master frame (booksink_frame)
- Publish the topic of the `Pose` can be read and planned by MoveIt.

Hector mapping is used to generate a map description file from the customized library environment for Libot's navigation and path planning.

Build the simulated library world in Gazebo environment and integrate with ROS. Libot successfully finished each task and fulfilled the requirements of our project.

5.2 CHALLENGES ENCOUNTERED

Gaps Between Simulation and Real-World Application. Simulations often fail to fully replicate the complexities and unpredictability of real-world conditions. Factors such as varying lighting conditions, unanticipated obstacles and physical properties of the Libot can affect performance to some extent. This gap can lead to suboptimal decision-making and efficiency when the systems are deployed in real environments.

Integration of Object Detection and Grasp Pose Detection (GPD). The integration of Object Detection technologies with Grasp Pose Detection (GPD) systems is crucial for automating grasping tasks. However, achieving seamless integration that consistently performs at 100% efficiency is hard. Variations in object sizes, shapes, and orientations, coupled with environmental factors, can impede the system's ability to accurately detect and handle objects. This can result in decreased reliability and effectiveness in tasks such as automated sorting or assembly in industrial settings.

Mapping Challenges. Mapping environments is a fundamental aspect of navigation for autonomous systems. However, this process can become highly unreliable in environments with repetitive patterns. These conditions can confuse the mapping algorithm, leading to inaccurate or incomplete data being collected.

5.3 IMPLICATIONS FOR FUTURE DEVELOPMENTS

The insights gained from this project illuminate the path forward for enhancing autonomous capabilities in similar applications. Future work could explore the integration of more advanced machine learning models to further improve decision-making and adaptability of the system.

CHAPTER 6 : FUTURE WORK

With the continuous advancement in the fields of machine learning and large language models (LLMs), two major trends have emerged in the development of mobile manipulation robots: generalization and robot learning. This section outlines potential future work derived from these trends.

6.1 GENERALIZATION

Currently, the application scenarios of our Libot are limited to autonomous book returns in libraries, lacking versatility in functionality. In the short term, we plan to employ the YOLO v8 recognition system for training on elevator buttons to enable autonomous operation. This step is part of a broader strategy to achieve full automation in various environments. Additionally, this recognition technology has been adapted for autonomous scanning of library shelves, enabling the Libot to identify misplaced books and other irregularities. In the long term, our goal is to generalize this robotic platform for diverse everyday applications by modularizing its storage compartments and equipping it with various attachments for tasks like express delivery in different settings. This approach aims to surpass traditional robotic delivery systems by achieving true door-to-door contactless delivery.

6.2 ROBOT LEARNING

The current version of Libot, while capable of sensing its environment, still lacks the depth of human perception and understanding. Therefore, our future endeavors will focus on enhancing its ability to interact effectively with both people and the physical world. Leveraging advancements in LLMs, we aim to integrate these models into Libot to enable autonomous work planning, moving away from traditional methods. A specific objective is to empower Libot to receive and interpret natural language instructions for complex tasks, such as navigating to specific library floors for book retrieval. For example, a command like “Please proceed to the designated floors to collect books," would be autonomously translated by the LLM-integrated Libot into actionable tasks, including comprehensive planning for obstacle avoidance and efficient elevator use to navigate between floors. This ambitious goal underscores our commitment to advancing robotic capabilities, although much work remains to be done.

REFERENCES

[1] S. Y. Nof, “Applications: Design and integration,” in *Handbook of Industrial Robotics*, pp. 109–150, Wiley, 1999.

[2] N. J. Nilsson, “A mobile automaton: An application of artificial intelligence techniques,” *International Joint Conference on Artificial Intelligence*, 1 1969.

[3] J. Schuler, *Integration von Förder- und Handhabungseinrichtungen.* Springer Berlin Heidelberg, 1 ed., 6 1987.

[4] R. Holmberg and O. Khatib, “Development and control of a holonomic mobile robot for mobile manipulation tasks,” *The International Journal of Robotics Research*, vol. 19, pp. 1066 – 1074, 2000.

[5] Y. Liu, S. Wang, Y. Xie, T. Xiong, and M. Wu, “A review of sensing technologies for indoor autonomous mobile robots,” *Sensors 2024, Vol. 24, Page 1222*, vol. 24, p. 1222, 2 2024.

[6] D. Xie, L. Chen, L. Liu, L. Chen, and H. Wang, “Actuators and sensors for application in agricultural robots: A review,” *Machines*, vol. 10, 10 2022.

[7] J. N. P. M. Hägele and K. Nilsson, “Industrial robotics,” in *Field and Service Robotics*, vol. 25, pp. 964–969, Springer, 2014.

[8] S. Cebollada, L. Payá, M. Flores, A. Peidró, and O. Reinoso, “A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data,” *Expert Systems with Applications*, vol. 167, 4 2021.

[9] L. Tagliavini, L. Baglieri, G. Colucci, A. Botta, C. Visconte, and G. Quaglia, “D.o.t. paquitop, an autonomous mobile manipulator for hospital assistance,” *Electronics*, vol. 12, no. 2, 2023.

[10] J. Varela-Aldás, J. Buele, S. Guerrero-Núñez, and V. H. Andaluz, “Mobile manipulator for hospital care using firebase,” in *HCI International 2022 - Late Breaking Papers. Multimodality in Advanced Interaction Environments* (M. Kurosu, S. Yamamoto, H. Mori, D. D. Schmorrow, C. M. Fidopiastis, N. A. Streitz, and S. Konomi, eds.), (Cham), pp. 328–341, Springer Nature Switzerland, 2022.

[11] R. Ramos-Garijo, M. Prats, P. J. Sansz, and A. P. D. Pobil, “An automous assistant robot for book manipulation in a library,” *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 3912–3917, 2003.

[12] *Robotics-Vocabulary, ISO Standard 8373:2021*, 2021.

[13] H. Iwata and S. Sugano, “Design of human symbiotic robot twendy-one,” in *2009 IEEE International Conference on Robotics and Automation*, pp. 580–586, 2009.

[14] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, “Development of the research platform of a domestic mobile manipulator utilized for international competition and field test,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7675–7682, 2018.

[15] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger, "Autonomous door opening and plugging in with a personal robot," in *2010 IEEE International Conference on Robotics and Automation*, pp. 729–736, 2010.

[16] I. Vatavuk, G. Vasiljević, and Z. Kovačić, "Task space model predictive control for vineyard spraying with a mobile manipulator," *Agriculture*, vol. 12, no. 3, 2022.

[17] L. F. P. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in agriculture robotics: A state-of-the-art review and challenges ahead," *Robotics*, vol. 10, no. 2, 2021.

[18] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, "Design, integration, and field evaluation of a robotic apple harvester," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1140–1159, 2017.

[19] Y. Angal and A. Gade, "Development of library management robotic system," in *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, pp. 254–258, 2017.

[20] H. N. Alim, N. Tamami, and A. H. Alasiry, "Automated library system mobile robot using a-star algorithms," *IES 2022 - 2022 International Electronics Symposium: Energy Development for Climate Change Solution and Clean Energy Transition, Proceeding*, pp. 355–360, 2022.

[21] "Object detection based on color and shape features for service robot in semi-structured indoor environment," *International Journal of Intelligent Robotics and Applications*, vol. 3, pp. 430–442, 12 2019.

[22] S. Takaki, J. K. Tana, and S. Ishikawa, "A human care system by a mobile robot employing cooperative objects recognition," *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2017-December, pp. 1148–1152, 12 2017.

[23] H. Sun, Z. Meng, P. Y. Tao, and M. H. Ang, "Scene recognition and object detection in a unified convolutional neural network on a mobile manipulator," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5875–5881, 9 2018.

[24] Y. Yu, Z. Cao, S. Liang, Z. Liu, J. Yu, and X. Chen, "A grasping cnn with image segmentation for mobile manipulating robot," *IEEE International Conference on Robotics and Biomimetics, ROBIO 2019*, pp. 1688–1692, 12 2019.

[25] H. Zhang, J. Tan, C. Zhao, Z. Liang, L. Liu, H. Zhong, and S. Fan, "A fast detection and grasping method for mobile manipulator based on improved faster r-cnn," *Industrial Robot*, vol. 47, pp. 167–175, 3 2020.

[26] S. Seferbekov, V. Iglovikov, A. Buslaev, and A. Shvets, "Feature pyramid network for multi-class land segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[27] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 300–307, 2010.

[28] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," *Robotics: Science and Systems*, 2014.

[29] P. Kim, J. Chen, and Y. K. Cho, “Slam-driven robotic mapping and registration of 3d point clouds,” *Automation in Construction*, vol. 89, pp. 38 – 48, 5 2018.

[30] S. Kohlbrecher, O. V. Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pp. 155–160, 2011.

[31] C. Campos, R. Elvira, J. J. Rodriguez, J. M. Montiel, and J. D. Tardos, “Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam,” *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, 12 2021.

[32] T. S. Pujari and S. B. Deosarkar, “Design of intelligent and robotic library system,” *RTEICT 2017 - 2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, Proceedings*, vol. 2018-January, pp. 1903–1908, 7 2017.

[33] S. V. Krishnan, V. Singh, P. Shah, A. Yadav, G. Panampilly, S. Saha, and H. D. Shukla, “Development of an rfid-based semi-autonomous robotic library management system,” *2020 4th International Conference on Automation, Control and Robots, ICACR 2020*, pp. 26–31, 10 2020.

[34] J. Thirumurugan, M. Vinoth, G. Kartheeswaran, and M. Vishwanathan, “Line following robot for library inventory management system,” *International Conference on "Emerging Trends in Robotics and Communication Technologies", INTERACT-2010*, pp. 1–3, 2010.

[35] R. Mehta and A. Sahu, “Autonomous robot for inventory management in libraries,” *2020 IEEE International Students’ Conference on Electrical, Electronics and Computer Science, SCEECS 2020*, 2 2020.

[36] J. Villalobos, I. Y. Sanchez, and F. Martell, “Singularity analysis and complete methods to compute the inverse kinematics for a 6-dof ur/tm-type robot,” *Robotics*, vol. 11, no. 6, 2022.

[37] J. J. Craig, “Inverse manipulator kinematics,” in *Intoduction to Robotics: Mechanics and Control*, pp. 117–150, Harlow, United Kingdom: Pearson, 2017.

[38] K. P. Hawkins, “Analytic inverse kinematics for the universal robots ur-5/ur-10 arms,” Georgia Institute of Technology, 2013. Available: https://repository.gatech.edu/server/api/core/bitstreams/e56759bc-92c8-43df-aa62-0dc47581459d/content.

[39] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[40] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. Fernández Perdomo, “ros_control: A generic and simple control framework for ros,” *The Journal of Open Source Software*, 2017. Available: `http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf`.

[41] S. Jagadeeshan, “Package summary: joint_state_publisher.” Jan. 17, 2020. [Online]. Available: http://wiki.ros.org/joint_state_publisher. [Accessed Mar. 12, 2024].

[42] GvdHoorn, “Package summary: robot_state_publisher.” Jan. 17, 2020. [Online]. Available: http://wiki.ros.org/robot_state_publisher. [Accessed Mar. 12, 2024].

[43] M. Gualtieri, A. ten Pas, K. Saenko, and R. W. Platt, "High precision grasp pose detection in dense clutter," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 598–605, 2016.

[44] Ros Documentation, "tf2/design." Jan. 17, 2015. [Online]. Available: http://wiki.ros.org/tf2/Design. [Accessed Apr. 4, 2024].

[45] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pp. 1–6, April 2013.

[46] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolo." ver. 8.0.0, 2023. Available: https://github.com/ultralytics/ultralytics.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[48] D. Kappler, J. Bohg, and S. Schaal, "Leveraging big data for grasp planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4304–4311, 2015.

[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[50] K. Mori, "ultralytics_ros." Noetic version, 2023. Available: https://github.com/Alpaca-zip/ultralytics_ros.

APPENDIX A : MATHEMATICAL MODEL AND PARAMETERS OF UR5

This experimental simulation is based on the Gazebo simulation platform, so in order to achieve accurate simulation data of the experiment, it is necessary to obtain a real mathematical model. The accuracy of simulation data also lays the foundation for physical experiments. This appendix introduces the technical specification, Denavit-Hartenberg parameter and other available parameters of UR 5.

Technical specification of UR5

Table A.1: Physical parameters of UR5

Weight (kg)	18.4
Payload (kg)	5
Joint ranges (°)	± 360
Reach (m)	0.85
Joint ranges (°/s)	180
Joint ranges (m/s)	1
Degree of freedom (DOF)	6
Repeatability (mm)	± 360
I/O Power supply	12V/24V 600mA
Power consumption (W)	150
Communication	TCP/IP, Ethernet socket & Modbus TCP
Materials	Aluminium, ABS plastic

Denavit-Hartenberg parameter of UR5

Denvavit-Hartenberg parameter (DH parameter) is a four-parameter associated with a specific convention management that fixes the reference frame to the space kinematic chain or robotic arm. This paper uses the standard DH parameter notation [38].

Table A.2: DH parameters of UR5

i	a_i (m)	α_i (rad)	$d_i(m)$	θ_i (rad)
0	0	0	-	-
1	0	$\frac{\pi}{2}$	0.08916	θ_1
2	0.425	0	0	θ_2
3	0.39225	0	0	θ_3
4	0	$\frac{\pi}{2}$	0.10915	θ_4
5	0	-$\frac{\pi}{2}$	0.009456	θ_5
6	-	-	0.0823	θ_6

Available parameters of UR5

Table A.3: Mass and center of mass of each link of UR5

Link	mass (kg)	center of mass (respect to i-th link reference frame) (m)
1	3.7	[0, -0.02561, 0.00193]
2	8.393	[0.2125, 0, 0.11336]
3	2.33	[0.15, 0, 0.0265]
4	1.219	[0, -0.0018, 0.01634]
5	1.219	[0, 0.0018, 0.01634]
6	0.1879	[0, 0, 0.001159]

Table A.4: Link inertia matrices in the Gazebo environment

(respect to i-th link reference frame) (kgm^2)

$$I_1 = \begin{bmatrix} 0.01497 & 0 & 0 \\ 0 & 0.01497 & 0 \\ 0 & 0 & 0.01041 \end{bmatrix} \quad I_2 = \begin{bmatrix} 0.13388 & 0 & 0 \\ 0 & 0.13388 & \\ 0 & 0 & 0.01511 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} 0.03121 & 0 & 0 \\ 0 & 0.03121 & 0 \\ 0 & 0 & 0.00410 \end{bmatrix} \quad I_4 = \begin{bmatrix} 0.00201 & 0 & \\ 0 & 0.00201 & 0 \\ 0 & 0 & 0.002194 \end{bmatrix}$$

$$I_5 = \begin{bmatrix} 0.00183 & 0 & 0 \\ 0 & 0.00183 & 0 \\ 0 & 0 & 0.00219 \end{bmatrix} \quad I_6 = \begin{bmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

APPENDIX B : LIBOT UNIVERSAL ROBOTIC DESCRIPTION FORMAT

Libot's Master URDF code

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://wiki.ros.org/xacro" name="myur5">

    <xacro:include filename="$(find booksink)/urdf/booksink.urdf"/>

    <link name="mass_link">
    <visual>
      <!-- <geometry>
        <cylinder length="0.1" radius="0.1"/>
      </geometry> -->
      <surface>
                <friction>
                    <ode>
                        <mu>100</mu>
                        <mu2>100</mu2>
                    </ode>
                </friction>
      </surface>
      <material name="blue">
        <color rgba="0 0 .8 1"/>
      </material>
    </visual>
    <mass value="200" />
    <inertial auto="true" />
    </link>

    <joint name="mass_joint" type="fixed">
        <origin xyz="0 0 0.1" rpy="0.0 0.0 0.0"/>
        <parent link="booksink_link"/>
        <child link="mass_link"/>
    </joint>

    <xacro:include filename="$(find ur_description)/urdf/ur5.xacro"/>
    <joint name="sink_ur5_joint" type="fixed">
        <origin xyz="-0.308 -0.2105 0.875" rpy="0.0 0.0 -1.57"/>
        <parent link="booksink_link"/>
        <child link="base_link"/>
    </joint>

    <xacro:include filename="$(find robotiq_ft_sensor)/urdf/robotiq_ft300.
        urdf.xacro" />
    <xacro:robotiq_ft300 prefix="" parent="tool0">
      <origin xyz="0 0 0" rpy="0 0 0"/>
    </xacro:robotiq_ft300>

    <xacro:include filename="$(find camera_holder)/urdf/camera_holder.urdf"
         />
    <joint name="camera_holder_joint" type="fixed">
        <parent link="wrist_3_link"/>
        <child link="camera_holder_link"/>
        <origin xyz="-0.0395 0 -0.03" rpy="1.5708 0 1.5708"/>
      </joint>

      <xacro:arg name="use_nominal_extrinsics" default="false" />
      <xacro:include filename="$(find realsense2_description)/urdf/_d435i.
          urdf.xacro"/>

      <xacro:sensor_d435i parent="camera_holder_link"
          use_nominal_extrinsics="$(arg use_nominal_extrinsics)">
      <origin xyz="-0.11 0.026 0.04" rpy="-1.5708 0 1.5708"/>
      </xacro:sensor_d435i>


    <xacro:include filename="$(find robotiq_2f_140_gripper_visualization)/
        urdf/robotiq_arg2f_140_model.xacro" />
      <joint name="ft_gripper_joint" type="fixed">
        <parent link="robotiq_ft_frame_id"/>
        <child link="robotiq_arg2f_base_link"/>
        <origin xyz="0 0 0" rpy="0 0 0"/>
      </joint>

    <!-- <gazebo>
        <plugin filename="libgazebo_ros_control.so" name="
            gazebo_ros_control">
            <robotNamespace>/</robotNamespace>
        </plugin>
    </gazebo> -->

  </robot>
```

Complete URDF code running in Gazebo simulation with MoveIt

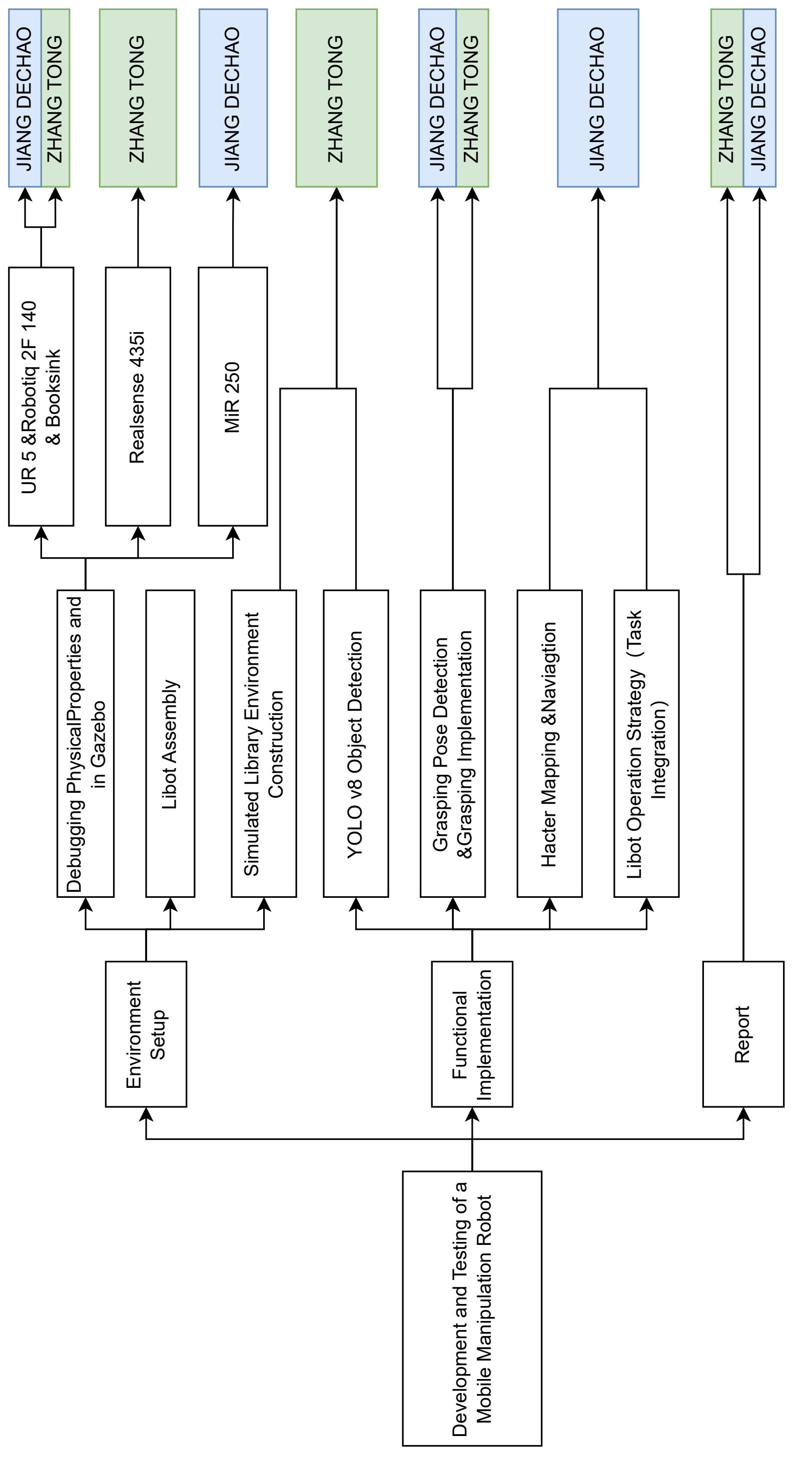
Please visit https://github.com/Cheney823/24EME_FYP/blob/main/src/myur_moveit_config/config/gazebo_myur5.urdf for details.

APPENDIX C : GITHUB REPOSITORY OF THE PROJECT DEVELOPMENT AND TESTING OF A MOBILE MANIPULATION ROBOT

Please visit our Github repository for full access to our project code.
Link: `https://github.com/Cheney823/24EME_FYP`
QR code:

24EME_FYP

APPENDIX D : WORK BREAKDOWN



Development and Testing of a Mobile Manipulation Robot
Environment Setup
Functional Implementation
Report
Debugging PhysicalProperties and in Gazebo
Libot Assembly
Simulated Library Environment Construction
YOLO v8 Object Detection
Grasping Pose Detection &Grasping Implementation
Hacter Mapping &Naviagtion
Libot Operation Strategy（Task Integration)
UR 5 &Robotiq 2F 140 & Booksink
Realsense 435i
MiR 250
JIANG DECHAO
ZHANG TONG
ZHANG TONG
JIANG DECHAO
ZHANG TONG
JIANG DECHAO
ZHANG TONG
JIANG DECHAO
ZHANG TONG
JIANG DECHAO