



WM-3 — Intelligent Robot Arm

CS 4850 - Section 03 – Fall 2024

Professor Sharon Perry

Nov 13, 2024

| | |
|---|---|
|  <p>Zhiwen Zheng Team Leader</p> |  <p>Ellie Ireland Developer</p> |
|---|---|

Team Members:

| Name | Role | Cell Phone / Alt Email |
|--------------------------|---------------------------|--|
| Zhiwen Zheng (Team Lead) | Documentation and Test | zhiwen20010111@gmail.com |
| Ellie Ireland | Developer | elirel973@outlook.com |
| Waqas Majeed | Project Owner and Advisor | wmajeed@kennesaw.edu |

Website Link: <https://sites.google.com/view/wm-3intelligentrobotarm?usp=sharing>

Github Link: <https://github.com/WM-3-Inrelligent-Robot-Arm/wm3ira.github.io>

Number of Lines of code in project: ~1700

Number of Project Components and Tools: 12

Total Work Hours: 242.55

Table of Contents

| | |
|--|----|
| 1.0 Introduction | 3 |
| 2.0 Requirements | 3 |
| 2.1 Functional Requirements | 3 |
| 2.2 Hardware Interface Requirements | 3 |
| 2.3 Software Interface Requirements | 4 |
| 3.0 Analysis | 4 |
| 4.0 Design | 5 |
| 4.1 Dependencies | 5 |
| 4.2 Detailed System Design | 5 |
| 4.3 Constraints | 6 |
| 4.4 Integration | 6 |
| 4.5 Operation Flowchart | 6 |
| 5.0 Development | 7 |
| 5.1 Gantt Chart | 7 |
| 5.2 Hardware Delegations | 7 |
| 5.3 ROS and Gazebo | 7 |
| 5.31 Autonomous Navigation | 7 |
| 5.32 Object Recognition | 7 |
| 5.4 Nvidia Isaac Sim | 8 |
| 5.41 Autonomous Navigation | 8 |
| 5.42 Pose Estimation and Grasp Selection | 8 |
| 5.5 Setup | 8 |
| 6.0 Test Plan and Report | 9 |
| 7.0 Version Control | 9 |
| 8.0 Summary | 11 |
| 9.0 Appendix | 11 |
| 9.1 Definitions | 11 |
| 9.2 Mobile Robot Sensors | 11 |
| 9.3 Machine Learning Algorithms and Policies | 12 |

1.0 Introduction

The goal of this project was to develop an intelligent mobile manipulator by integrating a mobile robot with a robotic manipulator. This was done using the Turtlebot3 Waffle Pi as a mobile base and the Open ManipulatorX as the robot arm. Peripherals of this unit included a Raspberry Pi micro computer, a camera module, and a LIDAR sensor. The intelligent aspect of the mobile manipulator was implemented with machine learning, taking input from the real world using machine vision. The functionality of this included autonomous navigation, real time object recognition, and pose estimation based grasp selection.

2.0 Requirements

2.1 Functional Requirements

- Implementation
 - Mobile manipulator created from a mobile robot and robotic manipulator
- Autonomous Movement
 - Manipulator may move to a user-defined location with its mobile robot attachment
- Object Recognition
 - Manipulator unit may see via camera input
 - Manipulator unit may recognize some set of objects using its vision
- Pose Estimation and Grasp Selection
 - Mobile manipulator relocates to be near the object it intends to pick up
 - Mobile manipulator is positioned over the object in the optimal orientation
 - Mobile manipulator tightens grip to grasp and pick up the object
- Manual Control
 - Mobile manipulator may be controlled via user input
- Hardware Implementation
 - Designs and software are implemented into hardware

2.2 Hardware Interface Requirements

The hardware required for this project includes the TurtleBot3 Waffle Pi and the OpenMANIPULATOR-X. Additionally, a PC will be needed.

- TurtleBot3 WafflePi
 - Supports a maximum of a 30kg payload
 - Equipped with a Raspberry Pi microcomputer
 - 32-bit ARM Cortex®-M7 with FPU
 - RC-100B + BT-410 Set Remote Controller
 - XM430-W210 Actuator
 - 360 Laser Distance Sensor LDS-01 Laser Distance Sensor
 - Raspberry Pi Camera Module v2.1
 - Gyroscope 3 Axis and Accelerometer 3 Axis
 - GPIO 18 pins and Arduino 32 pin

Peripheral: UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
 Inputs: Push buttons x 2, Reset button x 1, Dip switch x 2

- OpenMANIPULATOR-X
 - Supports open-source software (ROS, MoveIt! package)
 - DYNAMIXEL-X Series actuators (quadruple jointed)
 - Open-source hardware allows for simple modification via 3D printed parts
 - Aluminum frames
- PC
 - Capable of running Linux or Windows Subsystem for Linux on Windows 10 or 11 (WSL)
 - Runs on an Nvidia GPU

2.3 Software Interface Requirements

The software IDEs and environments needed for this project include ROS2 run on Linux, the Gazebo simulation platform, and the Nvidia Isaac Sim.

- ROS 2 Humble
 - Set of software libraries and tools
 - Run on Ubuntu 22.04
- Gazebo simulation platform hosted on ROS 2
 - Linux requirement
- Nvidia Isaac Sim 4.1.0
 - Tool suited for sensor testing and simulation
- Nav2 Library
 - Behavior tree based navigation planning
 - Used for autonomous navigation
- YOLOv8 model
 - CNN object recognition model
 - Provides oriented bounding boxes for object classification
- MoveIt! package
 - Uses point clouds to determine object location
 - Used for pose estimation and grasp selection

3.0 Analysis

The project produced successful results, with the following statistical analyses:

- 100% accuracy for autonomous navigation system
- 86% accuracy for object recognition system
- 82% accuracy for optimal grasp selection based on pose estimation.

Though these results are acceptable, it is believed that modification of models and machine learning algorithms used can enhance both accuracy and speed of the mobile manipulator's capabilities.

4.0 Design

4.1 Dependencies

The software creates and calls variables pertaining to the hardware which it will be run on. Such statements and invocations control the hardware system. The robotic manipulator utilized in this project was the OpenMANIPULATOR-X technology. The mobile robot which the manipulator will sit upon the TurtleBot3 Waffle Pi technology. Both pieces of equipment are developed and marketed by the ROBOTIS company. All software was written with the intention of being run with this physical combination of units.

The software was run on the robot operating system (ROS) supported by the OpenMANIPULATOR-X and TurtleBot3 Waffle Pi robots. Therefore, it was developed in a Linux system, and subsequent files were written in the Python programming language and referenced libraries provided by ROS2.

4.2 Detailed System Design

Listed below is each piece of hardware and its comprised components that were used and developed in simulation for this project.

- OpenMANIPULATOR-X: robot manipulator
 - DYNAMIXEL X430 Actuators
- OpenCR 1.0: Control module for ROS
 - 32-bit ARM Cortex-M7 with FPU Microcontroller
 - ICM-20648 (Gyroscope)
- TurtleBot3 Waffle Pi: mobile robot
 - Raspberry Pi 4 micro computer
 - 32-bit ARM Cortex-M7 with FPU Microcontroller
 - RC-100B + BT-410 Set (Bluetooth 4, BLE) Remote Controller
 - XM430-W210 Actuators
 - LSD-01 360 Laser Distance Sensor
 - Raspberry Pi Camera Module v2.1
 - 3 Axis Gyroscope and Accelerometer IMU

Listed below is each software package and its comprised components that were created and developed for this project.

- turtlebot3_manipulation_bringup: main package with launch files and world specifications and designs
- turtlebot3_manipulation_description: package containing URDF, XACRO, and visual mesh files for simulation of the robot model
- turtlebot3_manipulation_moveit_config: control package generated by the MoveIt! library after defining the joint relationships and planning groups in the setup assistant
- turtlebot3_manipulation_navigation2: navigation package for development of autonomous navigation
- yolov8_obbb: object recognition package for development and training of the model
- gpd: grasp pose detection library for determining optimal grasp selection based on the targeted object's positioning

4.3 Constraints

Constraints relevant to this project primarily involved the environment of operation for the robot unit. Because the system will be developed in an ideal or nearly ideal space, it was not be trained to overcome any weather, terrain, lighting, or suboptimal environmental difficulties. The manipulator was trained on a certain subset of objects ranging from different sizes and orientations. However, it is impossible to encapsulate all types of objects in a small subset. Therefore, though it is likely the OpenMANIPULATOR-X can grasp items similar to those it has seen before, it is not guaranteed that it would work on all other objects.

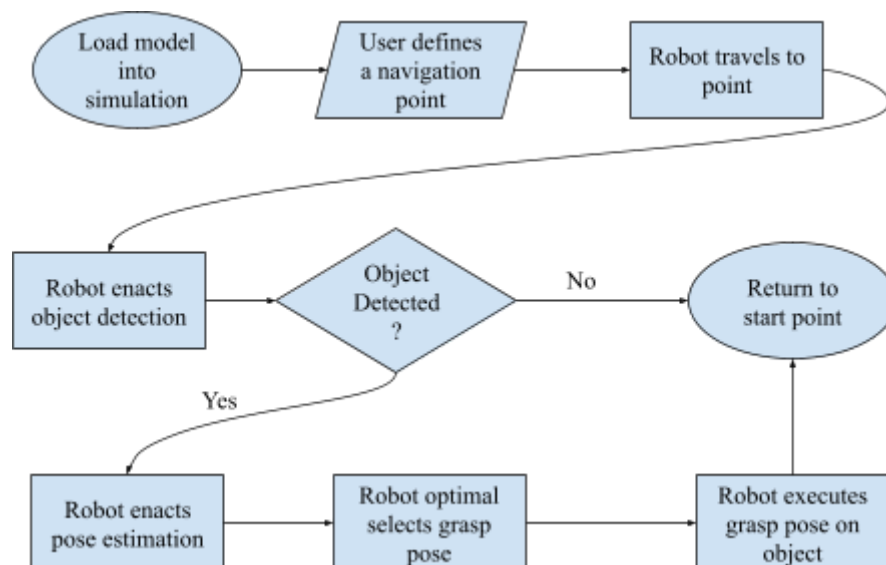
The operations of the Waffle Pi offer a maximum velocity of 0.26 m/s, and a maximum payload of 30 kg. The OpenMANIPULATOR-X system exercises a maximum payload of 500 grams, with a joint speed of 46 RPM. It can reach a maximum of 380 mm, and its hand may grasp anything as large as 75 mm and hold anything as small as 20 mm

4.4 Integration

Design for this project was based on the already established model of the integrated Turtlebot3 Waffle Pi and OpenManipulatorX. This includes the mobile robot as the base, incorporating differential drive, and the 4 DoF manipulator including an additional degree for the end effector. The base of the robot provides two joints, one for each wheel. The manipulator is fixed atop the mobile base to fully integrate the system and act as the parent link for which the manipulator's base joint may rotate upon. These design features were specified in the URDF and XACRO files in the description portion of the package.

Once the model was established and its URDF imported, the parent-child joint relationships and connecting links were defined in the MoveIt! so that interactions between each joint could be sound. This also enabled collision and the ability to place the robot in the Gazebo simulation.

4.5 Operation Flowchart



5.0 Development

5.1 Gantt Chart

Shown below is the approximate Gantt chart schedule this project followed, showing details pertaining to where and how time was invested.

[illegible]

5.2 Hardware Delegations

With the implementation of the mobile manipulator, certain tasks and functionalities were delegated to different parts of the robot. Autonomous navigation and object recognition was primarily a task of the mobile robot base, as that was the vessel for which the robot was able to navigate, and it was equipped with a camera module. Pose estimation based grasp selection was based on the manipulator.

Autonomous navigation functionality took place in both the Gazebo simulator and Isaac Sim. However, following tasks were then appointed based on resources available and simulation features. This also allowed for more efficient development.

5.3 ROS2 and Gazebo

5.31 Autonomous Navigation

In ROS2, the autonomous navigation was handled with the navigation library, Nav2. This allowed the robot to travel from a start point to a designated endpoint using behavior trees, which were customized based on the obstacles in the map the robot resides in using independent modular servers computing each path and control system. The Turtlebot3 Waffle Pi, being a differential driven mobile robot, was then able to handle the commands given by the navigation stack to avoid collisions with different objects.

5.32 Object Recognition

The object recognition in ROS was implemented using the YOLOv8 CNN model. This was developed with training on a specified dataset. To begin, a simple dataset of different bolts was used and tested. Once accuracy was obtained for this task, the dataset was expanded to

include detection of balls and bottles. Accuracy was once again tested and improved upon with further epochs.

5.4 Nvidia Isaac Sim

5.41 Autonomous Navigation

In the simulation environment of Isaac Sim, the autonomous navigation was based on the navigation package and the ROS 2 bridge in Isaac Sim. Because the models are ROS-based, the ROS 2 bridge allows Isaac Sim access to the control of the model and uses the ROS command and packages to support the simulation, like RVIZ. When setting up the navigation, the map of the environment, model definition, navigation parameters and some other files were defined before starting the navigation.

5.42 Pose Estimation and Grasp Selection

Pose estimation was developed using the MoveIt! library for individual joint control. This resource allowed for specification of each joint angle. However, using point cloud data and object location estimation from camera input, these joint angles can be produced, which is the essence of pose estimation.

5.5 Setup

Basic requirements for this project include a PC installed with either Windows 10 or later or Linux Ubuntu 22.04. This project was developed using PCs running Windows 10 and Windows 11. Therefore, the Windows subsystem for Linux was installed on both in order to run Ubuntu 22.04. ROS 2 Humble and relevant libraries were installed on each PC. Since one PC was unable to run Nvidia Isaac Sim due to lack of a graphics card, the Isaac Sim was only downloaded and used on one of the two PCs.

The Turtlebot3 manipulation package including both the Waffle Pi and OpenManipulatorX was installed on the ROS platform and simulated. The URDF files included in this package model the implementation of the two robots as one unit. The model was simulated in both Gazebo and Isaac Sim. Some time was taken in order to become familiarized with the platforms. Example nodes for movement and control were created, executed, and modified. Action graphs for commands were set up for effective joint communication.

Modification of the mobile manipulator model began after fluency was achieved in the ROS and Isaac Sim interfaces. This included defining the joint relationships, planning groups, end effectors, valid ranges of motion, and general behaviors of each subsystem. This was primarily done using the MoveIt! library and the setup assistant that it provides.

Development of autonomous navigation began in the Isaac Sim using the navigation package that was passed from ROS 2 using the Isaac Sim bridge. Implementation of the functionality followed in Gazebo, beginning with a twin delayed deep deterministic policy gradient machine learning model, which eventually shifted to a simpler concept using a known map. Complete capability was achieved using the Nav2 library in ROS 2 and Gazebo.

Object recognition development also began in the Isaac Sim using the ROS Object Detection package RT-DETR (Real-Time Detection Transformer). However, after complications and difficulties, it was later moved to become a ROS 2 task. This shifted the responsibilities to the YOLOv8 convolutional neural network based model. Initial object detection was trained with a dataset of bolts. This was later expanded upon by implementing more objects for the model to recognize.

On the contrary, the grasp planning and pose estimation began in ROS 2. Likewise, after many difficulties were met, an implementation with similar results was researched in Isaac Sim and implemented.

Unit testing and modifications were performed for all functionalities.

6.0 Test (Plan and report)

Listed below are the project's requirements and testing results of whether they passed or failed. Severity is assigned.

| Requirement | Pass | Fail | Severity |
|---|------|------|----------|
| Implement Base and Manipulator to Create Mobile Manipulator | X | | |
| Autonomous Navigation | X | | |
| Object Recognition | X | | |
| Pose Estimation Based Optimal Grasp Selection | X | | |
| Hardware Implementation | | X | Medium |
| Manual Control | | X | Low |

The gross functional requirements were achieved. Over the course of this project, the hardware implementation of the mobile manipulator shifted to have a focus in virtual environments, as the hardware did not arrive in time for effective development to take place. The manual control functionality was initially considered as an optional implementation, resulting in its severity being low for our scope.

7.0 Version Control

Due to the nature of the project, version control for the software was limited. That which took place included separate attempts or reanalysis of selected implemented libraries. Listed below is the log of these reconsiderations.

Version 4.0.0 (11/13/24)

- Implemented pose estimation and grasp selection

- Used the MoveIt! library for grasp planning
- Contributors: Zhiwen Zheng

Version 3.1.0 (11/12/24)

- Added complexity to YOLOv8 object recognition by implementing a larger dataset
- Dataset includes bottles, bolts, and balls
- Contributors: Ellie Ireland

Version 3.0.1 (11/11/24)

- Created an object detection model based on a set of images of bolts
- Uses the YOLOv8 CNN model
- Contributors: Ellie Ireland

Version 3.0.0 (11/10/24)

- Developed object recognition model in Isaac Sim
- Used the Isaac ROS Object Detection package RT-DETR (Real-Time Detection Transformer)
- Achieved 2D recognition capabilities
- Contributors: Zhiwen Zheng

Version 2.1.0 (11/09/24)

- Migrated navigation task to perform without the need for training
- Created a known map for the model
- Implemented the Nav2 library
- Contributors: Ellie Ireland

Version 2.0.0 (11/03/24)

- Developed navigation model based on a twin delayed deep deterministic policy gradient machine learning model
- Contributors: Ellie Ireland

Version 1.3.0 (10/13/24)

- Implemented Turtlebot3 Waffle Pi and OpenManipulatorX robot models together in ROS
- Contributors: Ellie Ireland

Version 1.2.0 (10/08/24)

- Built action map for Turtlebot3 Waffle Pi movement control
- Contributors: Zhiwen Zheng

Version 1.0.1 (10/07/24)

- Combined Waffle Pi and OpenManipulatorX models together in one model in Isaac Sim
- Contributors: Zhiwen Zheng

Version 1.0.0 (09/30/24)

- Imported Turtlebot3 Waffle Pi robot model in Isaac Sim
- Imported OpenManipulatorX robot model in Isaac Sim
- Contributors: Zhiwen Zheng

Version 0.3.1 (09/29/24)

- Added inertia to URDF model

- Added collision to URDF model
- Contributors: Ellie Ireland

Version 0.3.0 (09/24/24)

- Created URDF model of a 4DoF manipulator
- Contributors: Ellie Ireland

Version 0.2.0 (09/23/24)

- Modeled a robot and tested in Isaac Sim
- Modeled a manipulator and tested in Isaac Sim
- Contributors: Zhiwen Zheng

Version 0.1.0 (09/03/24)

- Model created and designed in ROBOTC platform
- Contributors: Ellie Ireland

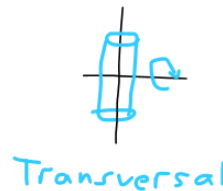
8.0 Summary

This project aimed to develop a mobile manipulator by integrating a mobile robot and robotic manipulator. Capabilities of the robotic unit included autonomous navigation, object recognition, and pose estimation based grasp selection. These functionalities were developed using the ROS 2 Humble, Gazebo, and Nvidia Isaac Sim platforms. The project yielded successful results, with a perfect navigation system, an 86% accuracy for object recognition, and an 82% accuracy for optimal grasp selection based on pose estimation.

9.0 Appendix

9.1 Definitions

- Node – any program that has access to the ROS functionality and communications
- Topics – used to send data streams
- Services – used for interactions
- Differential Drive – motors are independent of each other
- Joint Rotations



9.2 Mobile Robot Sensors

- Proprioceptive Sensors - tell how much the robot has moved (shaft encoders)
 - Dead Reckoning – tells how far the robot has gone and how much it has rotated (use IMU)

- Odometry - use of data from motion sensors to estimate change in position over time (can be faulty)
- Exteroceptive Sensors – gives perception of world (camera)
- Exteroceptive Sensors – allow for operation in dynamic environments (lidar, radar, sonar)

9.3 Machine Learning Algorithms and Policies

- Q-value function – function that estimates the expected future rewards an agent can obtain by taking specific action α in a specific state s
 - Critic network – neural network that approximates the Q-value function
 - Actor – neural network that represents the policy of the agent
 - Policy – strategy that the agent employs to determine its actions given a state
 - Can be deterministic or stochastic
 - Deterministic – actor directly outputs a specific action for each state
 - Stochastic – policy outputs a probability distribution over possible actions
 - Target networks – copies of the main actor and critic networks used to stabilize training
- Twin Delayed Deep Deterministic Policy – good for continuous control problems like autonomous robot driving
 - Twin (two) critic networks (Q value functions)
 - Delayed updates from the actor for the target and policy
 - Action noise regularization so policy is less likely to exploit actions with high Q-value estimates