WM-3 - Intelligent Robot Arm

Software Design Document (SDD)

CS 4850 – Section 03

Fall 2024

Professor Perry

Aug, 27, 2024

 Zhiwen Zheng Team Leader	 Ellie Ireland Developer

Team Members:

Name	Role	Cell Phone / Alt Email
Zhiwen Zheng (Team Lead)	Documentation and Test	678-818-5275 zhiwen20010111@gmail.com
Ellie Ireland	Developer	863-221-3999 elirel973@outlook.com
Waqas Majeed	Project Owner and Advisor	470-578-6005 wmajeed@kennesaw.edu

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Ellie Ireland	08/29/24	First draft	v0

1. Introduction and Overview

The intelligent robot arm project entails the design and development of a robotic manipulator arm in an industry standard environment. The goal of this robot is to achieve the grasping and manipulation of objects functionalities. Once the software is established, it will be uploaded to the system on the physical hardware, allowing for testing and verification.

The purpose of this SDD document is to monitor the development process of this project and ensure the design of both the hardware and software is kept in a forward direction.

2. Design Considerations

2.1. Assumptions and Dependencies

The software will create and call variables pertaining to the hardware which it will be run on. Such statements and invocations will be what controls the hardware system. The robotic manipulator utilized in this project will be the OpenMANIPULATOR-X technology [1]. The mobile robot which the manipulator will sit upon will be the TurtleBot3 Waffle Pi technology [2]. Both pieces of equipment are developed and marketed by the *ROBOTIS* company. All software will be written with the intention of being run with this physical combination of units.

It is assumed that the software will be run on the robot operating system (ROS) supported by the OpenMANIPULATOR-X and TurtleBot3 Waffle Pi robots. Therefore, it will be written in the C programming language and reference libraries provided by ROS. The robot operating system is hosted on Linux.

It is possible that additional features be added to the scope of this project. The primary functionality that will be considered is the implementation of artificial intelligence and machine learning to the robot unit, which will be applied to the visual input from the unit's camera and distance sensors. Such an extension will provide autonomous decision-making pertaining to objects that will be grasped, and where to go to grasp them, and how to grasp them correctly. This capability will be included depending on the development of the project over its predetermined lifespan.

2.2. General Constraints

Constraints applied to the software of the system are given by the limitations of the physical system and differences in the simulated environment and real-world environment.

Firstly, constraints for this project exist in the hardware environment portion of the system. The robots used maintain a standard of accuracy that is acceptable to the market;

however, that does not mean it is perfect. The actuators used in the manipulator's hardware are the DYNAMIXEL-X430 encoders, responsible for the robot's four-jointed movement. These are instructed by a PID control system with a 4096 pulse/rev resolution. Further details can be referenced in the operation specification site on the *ROBOTIS* website [3]. The TurtleBot3 Waffle Pi robot consists of a significantly larger amount of hardware that may produce error. The most important of these being the XM430-W210 actuator, the 360 LSD-01 laser distance sensor, and the Raspberry Pi Camera Module v2.1. As the most influential in determining the path and direction which the mobile robot may move, these components have the ability to produce a large amount of error should they at any time fail. More information concerning the parts and pieces of the Waffle Pi robot may be found on the ROBOTIS website [2].

Within this project, there also exist software environment constraints. Due to the timeframe we will be given, it is required that design of the software system must begin before the hardware will be accessible for testing. Therefore, in the initial stages of development, the system will be simulated in a software supported by ROS called *Gazebo*. While this platform offers a valuable virtual environment, ideals upheld in simulation are often impossible to achieve in the real world. Therefore, once the hardware is received, it is likely that some reconfiguration must be performed in order to translate the software to the hardware as intended in development.

2.3. Development Methods

Development for this project will begin in an industry standard virtual environment called Gazebo. This simulation platform is compatible with ROS and offers tools pertaining to both OpenMANIPULATOR-X and TurtleBot3 Waffle Pi. Therefore, because its offerings support the hardware we will use for this project, a large portion of development may occur completely virtually, making it a desirable alternative while we await the ordered hardware to be delivered. This will be the selected method of software design and testing for two thirds of the project life cycle.

Once the hardware components are made available to us, the robotic unit will be assembled, and the software will be uploaded to it. Development will then resume, and we will use the physical hardware behavior as feedback rather than Gazebo's output simulation.

In the specific design of the software, we will utilize the Software Development Life Cycle model. This process breaks down the software design and implementation into seven phases: planning, requirements, design, development, testing, deployment, and maintenance [4]. Separating each of these activities ensures all who participate in the creation of new software do not overlook any step in favor of another. Therefore, we will uphold this model in the development of our own project as well.

3. Architectural Strategies

3.1. Platform and Programming Language

The architecture for this project will be based on constraints set by the platforms and tools we will be working with. In order to interface the robot's physical peripherals, we will be utilizing the C programming language. Actions performed by the robot will be called using a library extension of the ROS system called MoveIt!. Additional resources offered by ROS may also be referenced in the creation of the framework of the system.

3.2. Expansion and Future Plans

There is potential to expand the software to reach the territory of artificial intelligence and machine learning. Should the project's scope expand this far, the algorithm used for object detection and self-direction will be a neural network. This model will subsequently be trained based on data we will collect and modify to indicate good and bad performance. The specific structure of the neural network will be decided based on the initial performance and recognition patterns of the robot.

3.3. Error Detection

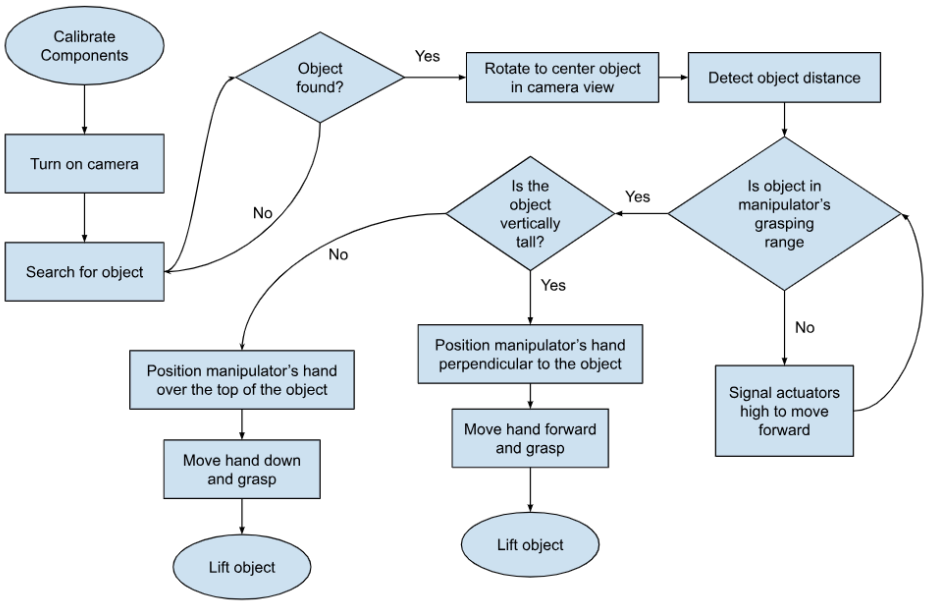
Because there are no sensors in the robotic manipulator's grasping region, the system is open looped. In order to close this loop and provide feedback pertaining to whether a grasping action was successful or not, we will stand in place to modify the sensitivity or direction of the movement. It will also be possible to close this loop using machine vision and additional implementation of machine learning; however, this will be an optional feature implemented based on convenience and need.

3.4. Concurrency and Parallelization

There is no direct need for parallelized processing in this system. The step-by-step actions remain straightforward. Once calibrated, the robot will scan for objects to pick up and, upon detection, move in the direction of that object. Computationally, the robot accepts the visual or distance input from the object and, once pictorially centered, decides to move its actuators; if the object becomes larger, it continues to move in that direction; if the object becomes smaller, it moves in the opposite direction. Though there is not direct parallelization in the program, there may exist interrupts, whereby if an incorrect action is detected to have taken place, the robot should halt performance and rectify the mistake.

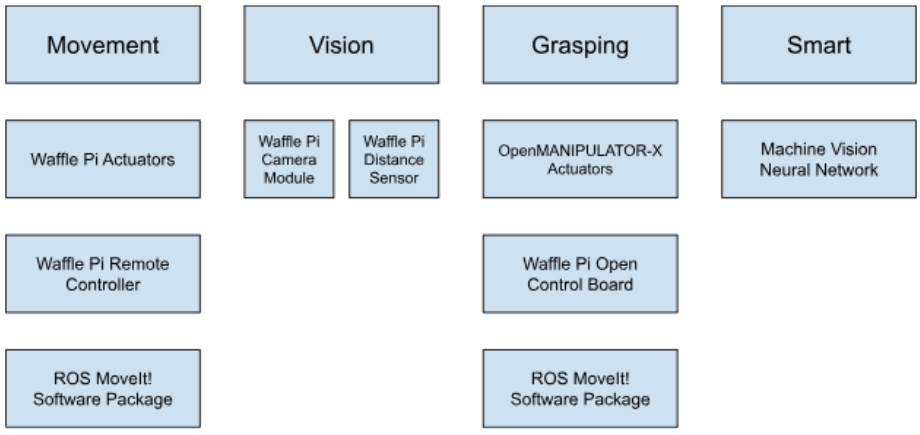
3.5. Operating Flowchart

Depicted below is the abstracted order of operations of the entire system. After each computation, a decision must be made, seen in the physical output of the robot unit.



4. System Architecture

As per the software requirements specification, the functional requirements of this project include movement, vision, grasping, and possibly autonomous smart capabilities. The general hierarchy and responsibility assignment for these functionalities can be depicted in the figure below. Most of these delegations were intuitive. While there does exist additional potential smart features, the most likely candidate for implementation would be simple robot autonomy from human direction and dependency.



The movement of the robot unit is the responsibility of the TurtleBot3 Waffle Pi. This mobile robot will allow the manipulator to relocate, expanding its range and what it has access to. The vision portion of the unit if given with components built into the Waffle Pi, including a camera module and a distance sensor. The grasping of the unit will be done with the

OpenMANIPULATOR-X robot, controlled by its actuators at four different joint points. Lastly, the smart features will be implemented using a machine learning neural network based on the vision input that the system receives.

5. Detailed System Design

5.1. Classification

Listed below is each piece of hardware and its comprised components that will be used and developed in 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

[5]

Listed below is each expected piece of software and its comprised components that will be created and developed in this project. Additional resources that we expect to be using include MoveIt! package from ROS, Gazebo simulation environment from ROS, and MIVisionX machine vision toolkit.
main.c: C program file
- Calibrate hardware
- Control both Waffle Pi and OpenMANIPULATOR-X actuators
- Grab Waffle Pi camera module and distance sensor input
- Generate decisions concerning the orientation at which the manipulator should grasp

machineVision.c: C program file
- Grab Waffle Pi camera module and distance sensor input
- Generate decisions concerning movement and direction

5.2. Definition

Listed below are simple definitions of the meaning and purpose behind each component.
- Actuators: motors that drive the movement of the system.
- Microcontroller: chip that processes the commands given to the system by our software, translating them into physical movements.
- Gyroscope: sensing unit that detects angular velocity and orientation.

- Microcomputer: computer system with similar, yet more complex functionalities to the microcontroller. Provides additional optional user interfacing between user external components.
- Remote controller: Bluetooth enabled device that allows the user to make changes to the direction and actions of the system in real time, rather than by a software.
- Laser Distance Sensor: sensor measuring space in time based on a the time of flight of an emitted laser beam.
- Camera Module: camera provides visual input of the robot unit's surroundings.
- Accelerometer: sensor that provides the acceleration information of its host.

5.3. Constraints

Constraints relevant to this project primarily involve the environment of operation for the robot unit. Because the system will be developed in an ideal or nearly ideal space, it will not be trained to overcome any weather, terrain, lighting, or suboptimal environmental difficulties. The manipulator will be 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 will be able to grasp items similar to those it has seen before, it is not guaranteed that it will work on all other objects.

The operating life the Waffle Pi offers with its lithium polymer 11.1V battery is two hours. It allows for a maximum velocity of 0.26 m/s, and a maximum payload of 30 kg [2]. 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 [6].

5.4. Resources

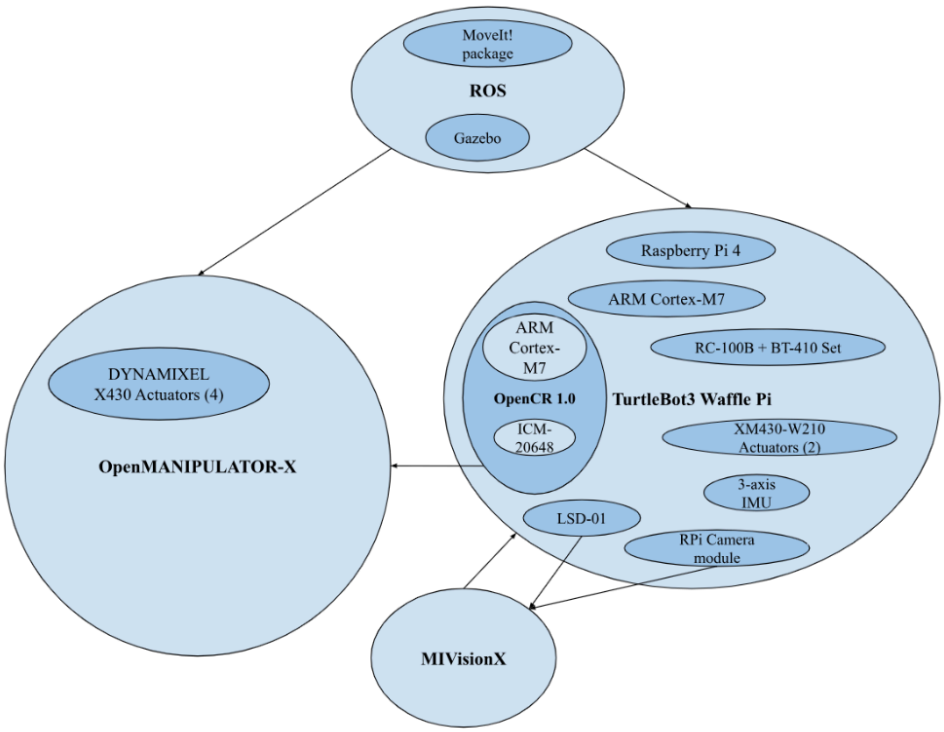
Covered below is a list of the external packages, libraries, and softwares called and referenced within the project.

- ROS: the operating system on which the OpenMANIPULATOR-X and Waffle Pi run.
- MoveIt!: ROS package that allows for simple control of each robot making up the complete unit [7].
- Gazebo: ROS simulation environment that allows for development of the robot system without the physical parts.
- MIVisionX: toolkit allowing for prototyping and development of computer vision applications [8].

Because this project has a linear execution path, parallel programming would not provide a significant speedup or process enhancement. Therefore, a large portion of possible issues that parallelizing the software may cause can be eliminated from our project. The robot's task of looking for objects

5.5. Interface/Exports

This robot will perform the action of grasping an object based on its detected shape and size. It is capable of maneuvering in different directions in order to reach an object that is out immediate range. The detailed components of each machine and how they are integrated have been disclosed in previous sections. Shown below is a diagram of each mentioned component and how they are interfaced to work together, where arrow pointer connections indicate a data transfer. The data transfer between the OpenCR 1.0 and OpenMANIPULATOR-X will occur via a physically wired connection in the system, whereas all other data transfers will be uploaded to the integrated processors before the operation begins.



6. Glossary

OpenMANIPULATOR-X: robotic manipulator
TurtleBot3 Waffle Pi: mobile robot which the manipulator will sit atop
Robot Operating System (ROS): software framework for robot software development
ROBOTIS: company that manufactures and produces the robotic pieces used in this project
DYNAMIXEL-X430 encoders: actuators used in the OpenMANIPULATOR-X robot
XM430-W210 actuators: actuators used in the TurtleBot3 Waffle Pi robot
LDS-01: laser distance sensor implemented in the TurtleBot3 Waffle Pi robot
Gazebo: virtual environment simulation for different ROBOTIS technology hosted on ROS
MoveIt!: the ROS package allowing for simple control of the OpenMANIPULATOR-X
OpenCR 1.0: embedded board for ROS used for controlling the OpenMANIPULATOR-X
MIVisionX: machine vision toolkit

7. Bibliography

[1] https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/
[2] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/
[3] https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/
[4] https://theproductmanager.com/topics/software-development-life-cycle/
[5] https://emanual.robotis.com/docs/en/parts/controller/opencr10/
[6] https://www.robotis.us/openmanipulator-x/
[7] https://moveit.ros.org/
[8] https://rocm.docs.amd.com/projects/MIVisionX/en/docs-5.3.3/README.html
[9] https://emanual.robotis.com/