# FEE175AB Final Report

# Fruit Picking Robot

### EE 175AB Final Report

### Department of Electrical Engineering, UC Riverside

| | |
| --- | --- |
| **Project Team Member(s)** | Ulises Lazaro, Jason Weiser, Ronny Anariva, Jimmy Le |
| **Date Submitted** | March 15, 2021 |
| **Section Professor** | Konstantinos Karydis |
| **Revision** | Revision 1.0 |
| **URL of Project Wiki/Webpage** | N/A |
| **Permanent Emails of all team members** | Ulaza001@ucr.edu<br>ranar001@ucr.edu<br>jle071@ucr.edu<br>jweis012@ucr.edu |

**Summary:**

Robotics is an emerging field that has applications in all walks of life. This report introduces the ideas and experiments that were performed in order to design and code a Fruit Picking Robot. This paper explains the types of software, hardware, and algorithms that were implemented in our senior design project. A more in depth introduction will be provided in section 2. While this is a technical paper, we hope that the reader will find this paper a source of inspiration/research that will allow

them to understand the work that it takes to build and code a robot base and Arm. More specifically, the Turtlebot and ReactorX arm serve as the platform for the experimentation.

# Revisions

| Version | Description of Version | Author(s) | Date Completed | Approval |
| --- | --- | --- | --- | --- |
| 1.0 | Filling out all of the sections of the report | Ulises Lazaro<br>Jason Weiser<br>Ronny Anariva<br>Jimmy Le | 03/15/21 | X |
| 2.0 | ENGR181W Revisions | Ulises Lazaro<br>Jason Weiser | 6/10/2021 | X |

# Table of Contents

**Revisions** 3

**Table of Contents** 4

**1 * Executive Summary** 6

**2 * Introduction** 7

2.1 * Design Objectives and System Overview 7
2.2 * Backgrounds and Prior Art 7
2.3 * Development Environment and Tools 7
2.4 * Related Documents and Supporting Materials 7
2.5 * Definitions and Acronyms 7

**3 * Design Considerations** 11

*3.1* * Realistic Constraints 11
3.2 System Environment and External Interfaces 11
3.3 * Industry Standards 12
3.4 * Knowledge and Skills 12
3.5 * Budget and Cost Analysis 13
3.6 * Safety 14
3.7 Performance, Security, Quality, Reliability, Aesthetics etc. 14
3.8 * Documentation 14
3.9 Risks and Volatile Areas 14

**4 * Experiment Design and Feasibility Study** 15

4.1 * Experiment Design 15
4.2 * Experiment Results, Data Analysis and Feasibility 18

**5 * Architecture and High Level Design** 27

5.1 * System Architecture and Design 27
5.2 * Hardware Architecture 28
5.3 * Software Architecture (only required if your design includes software) 29
5.4 * Rationale and Alternatives 29

**6 Data Structures (include if used)** 30

6.1 Internal software data structure 30
6.2 Global data structure 30
6.3 Temporary data structure 30
6.4 Database descriptions 30

**7 * Low Level Design** **31**

7.1 *Module i:* Motor Control 31
*7.1.1 Processing narrative for module i (change italics to the name of your module)* 31
*7.1.2 Module i interface description (change italics to the name of your module)* 31
*7.1.3 Module i processing details (change italics to the name of your module)* 31
7.2 Module 2: The A* algorithm 32
7.3 Module 3: Obstacle Avoidance 34
7.4 Module 4: ReactorX 150 Arm 35
7.5 Module 5: Orbbec Astra Pro+ Camera 36
7.6Module 6: The entire Integrated System 38

**8 * Technical Problem Solving** **40**

8.1 The Original mobile base problem 40
8.2 Solving the Original Mobile base Problem
8.3 The End Effector Problem 40
8.4 Solving the End effector Problem 40
8.5 The Computer Vision Implementation Problem 40
8.6 Solving the Computer Vision Implementation Problem 41
8.7 The Gazebo-Turtlebot integration Problem 41
8.8 Solving the Gazebo-Turtlebot integration problem 41

**9 User Interface Design** 42

9.1 Application Control 42
9.2 User Interface Screens 42

**10 * Test Plan** **43**

10.1 * Test Design 43
10.2 * Bug Tracking 45
10.3 * Quality Control 45
10.4 * Identification of critical components 45
10.5 * Items Not Tested by the Experiments 46

**11 * Test Report** **47**

11.1 * Test 1 47
11.2 * Test i (one section for each test i = 2, …., n) 47
11.3

**12 * Conclusion and Future Work** **51**

12.1 * Conclusion 51

12.2 Future Work 51
12.3 * Acknowledgement 51

**13 * References 53**

**14 * Appendices 55**

# 1. * Executive Summary

This project implements an autonomous picker robot arm on a mobile base, intended for use in a rugged agricultural setting. It is optimized to pick cherry tomatoes. It aids human labor in cultivation fields.

One motivation behind this project is to help farmers harvest tomatoes during difficult times, for example during hot days or perhaps during fire season in California. Additionally, another motivation is to increase the efficiency of the fruit picking process. Some scientific articles show that there will be a shortage in farm labor in the near future. This causes issues with the supply of crops since it is a known fact that the human population keeps increasing. Cherry tomatoes are a delicate crop that is consumed by many people.

The robot is able to collect tomatoes with very little human interaction. The details for this technology will be explained in detail throughout the paper. This robot offers the key features of robustness, autonomy, and a wide range of motion.

The frame of the robot is made from aluminum, and it also includes a suspension system that allows the robot to reach difficult terrains. Having a metal frame ensures that the robot will stand the difficulties of most terrains without breaking. The feature of autonomy is one of the key components necessary for optimization. Several algorithms give the robot the ability to traverse farm maps without a human controlling the robots direction. The robot base uses the A-star algorithm for path planning based on a grid map, ideal for farm use since the mapping does not change.

Equally important, the robot uses computer vision to recognize when the tomatoes are ripe and ready for harvest, as well as ultrasonic sensors used for object avoidance. This design offers a wide range of motion provided by the 5 degree of freedom robotic arm that has a reach of .4 meter and is able to rotate 360 degrees. The arm uses state of the art software libraries which allow users to modify the arm movements with very little code.

Several configurations of the robot were constructed in the Gazebo physics simulation program, which allowed for rapid prototyping. Simulations in MoveIt allowed for remote development of waypoint generation procedures. ROS allowed for integration with control systems through the easy portability of python into C and Arduino code. Autonomous control was only partially developed in this project. Further work is speculated to include total autonomous control of the robots daily tasks.

In later sections, the test results for each section are provided. The tests show the accuracy of the electrical components as well as the algorithms optimized. The robot is able to traverse simple maps efficiently. The robotic arm experiments clearly demonstrate that the robot can pick any average size cherry tomatoes efficiently without damaging the crop.

This project experienced many technical challenges, delays, and failures. It also successfully implemented path generation, arm manipulation, image extraction and filtering, along with several physics simulations.

# 2. * Introduction

## 2.1.* Design Objectives and System Overview

### 2.1.1 Concept and application of the design

The concept of the project was to build and program a fully autonomous robot that aids in the task of harvesting tomatoes. The robot is complex enough that it can be used in a wide range of applications in the farming industry. The mobile base was built with a rugged frame that allows the robot to travel through tough terrain. It functions by executing the trajectory generation algorithm, the A star. The 5 degree of freedom arm has a long reach, which allows the robot to have the reach and strength necessary to pick most tomatoes. Next, the robot's computer vision identifies the tomatoes which are red and ready to be harvested. The components all work together to achieve the maximum efficiency.

### 2.1.2 What are you designing?

The mobile base, arm, and sensors incorporated numerous controllers, which all needed prototyping and testing. All systems were programmed to operate according to autonomous algorithms. The pick and place arm was purchased from a manufacturer but all other components were selected individually. All components were programmed in Python or C++. Appropriate power systems were selected.

### 2.1.3 Technical principles

**Autonomous Robot Technical Principles:**

Arduino Mega- used for mobile based input/output control.

L298 Driver- drives the motors using PWM signal with Arduino.

50:1 gearbox motor - provides base with enough torque to move around.

ReactorX 150 Manipulator - 5 DoF robot arm, 100 grams strength.

Orbbec Astra Pro 3D Camera - Object Detection.

**Software:**

Ubuntu - operating system

Arduino Ide - Arduino Code

ROS - Robot communication

### 2.1.4 Why is this a meaningful project?

This project has the potential to discover new efficiencies in labor intensive farming by inexpensive robotics parts and advancements in vision systems.

### 2.1.5 What are the intended applications?

The project is intended to operate in indoor or outdoor farms for pruning or harvesting.

### 2.1.6 How is it related to subjects in electrical engineering?

Areas of Control Theory are utilized with PID controllers. Many concepts from Computer Vision are used relating to image thresholding, image segmentation, and image recognition. Power systems, microcontrollers, lidar, cameras, and ultrasonic sensors are used.

### 2.1.7 What are the overall goals of the project?

The overall goals of the project are to incorporate as much autonomy as possible, while designing around many practical cases for robustness and efficient movement. It's movement should be tracked through the environment to allow for optimization.

### 2.1.8 High level description of the system structure

Functionalities, Interactions with external systems, System issues, Operating environment, User environment, inputs, outputs, etc.

### 2.1.9 Quantitative design objectives

Robotic Arm Accuracy: +/-2.5 mm error

Motor Speed: 95 % accuracy.

Goal Point Accuracy: 10% deviation from goal.

Successful vision recognition rate of fruit with CV system

PID controller effectiveness on prototyped cart

Travel time implemented on Turtlebot2 using trajectory generation algorithms and cameras.

### 2.1.10 Responsibilities

Ulises Lazaro

- A-Star Algorithm
- Robotic Electronics
- Coding - C, Python
- Path Planning

Jimmy Le

- A-Star Algorithm
- Robotic Electronics
- Coding - C, Python
- Path Planning

Ronny Anariva

- Object avoidance

Jason Weiser

- Prototyping camera arms
- Image processing in OpenCV, python
- Machine learning and reinforcement learning
- Gazebo modeling of arm and cart for image servoing

## 2.2.* Backgrounds and Prior Art

The main motivation behind this project is well captured by the article "Research and development in agricultural robotics: A perspective of digital farming". The article mentions how it estimates the world population to be around 9.8 billion people. This means that there would be an increased demand for food to compensate for the increase in population. An application of the project is

helping out with the efficiency of fruit picking and will ship out produce quicker. Many fruit picking robots exist, yet the other designs are much larger. This design is smaller since it must reach places close to the ground, for this particular prototype tomatoes are the desired object to be harvested. The small size has several advantages, the most important being power efficiency. The relatively small design and components consume less power than the other robots on the market.

## 2.3.* Development Environment and Tools

**Equipment**:

- Siglent Oscilloscope
- DD Signal Generator
- Atmega 2560(Arduino)
- Lithium Batteries
- Raspberry Pi 4
- Motor Drivers
- Power converters

**Software tools**:

- Linux( UBuntu)
- Arduino IDE
- ROS
- Gazebo for Simulation Environment

## 2.4.* Related Documents and Supporting Materials

Industry standard references are in section 13.

REP Standard 000: Index of ROS Enhancement Proposals (REPs)[7]
REP Standard 105: Coordinate Frames for Mobile Platforms[8]
REP Standard 118: Depth Images[9]
REP Standard 137: ROS distribution files[10]
REP Standard 141: ROS distribution files[1]
REP Standard 142: ROS Indigo and Newer Metapackages[11]
REP Standard 143: ROS distribution files[13]
REP Standard 144: ROS Package Naming[14]
REP Standard 153: ROS distribution files[15]
OSHA 1910.212: General requirements for all machines[16]
ISO 10218-1:2011: Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots[17]

## 2.5.* Definitions and Acronyms

A Star Search Algorithm = A*

Computer Vision = CV

Robot Operating System = ROS

RIBS= Riverside-Irvine Builder of State Machines

RIMS= Riverside-Irvin Microcontroller Simulator

SM= State Machine

OA= Obstacle Avoidance

# 3. * Design Considerations

## *3.1.** Realistic Constraints

**Robot constraints:**

There were three main constraints in this project: Weight of the Robot, power consumption, and the robots environment.

Weight:

The desired size of the robot came with a constraint on weight, for the robot to be robust it needed to be built with an aluminum frame. This added much stress on the motors and consumed power that otherwise could be used to harvest. Due to motor torque capabilities, the robot cannot exceed a weight of 16 Kg, including the load provided by tomatoes.

Power Consumption:

The weight of the robot puts much strain on the system, as a result the motors function at their maximum capabilities all the time. The 20 V lithium battery can provide up to 3 amps, which results in maximum power of 60 Watts. The motors alone consume 24 Watts, and the robotic arm consumes 60 watts, this is beyond the limits of the lithium battery. A second battery was needed to power the arm separately.

Robot's Environment:

The environment was another major concern. Originally the design was meant to be tested outside,due the covid 19 virus, and the testing environment was constructed in a laboratory, an area of 15'x10' was improvised. Another Robot was used for demonstration purposes. The team built an indoor environment using stools to mimic an environment of tomato plants. Plastic balls and wires were used instead of actual plants. This environment was not ideal to test the ruggedness of the robot, yet it was sufficient to test the search and harvesting algorithm.

## 3.2. System Environment and External Interfaces

The original prototype:

- Arduino IDE
- Interbotix Library
- RIBS/RIMS
- ROS
- Gazebo

Turtlebot and ReactorX 150:

- Ubuntu
- Gazebo
- Rviz
- Interbotix Library

## 3.3.* Industry Standards

The robotic arm is built with industry leading smart servos. First, the Arm's servos have unique IDs that allow the user to either communicate with one or several groups of motors. The servos are also Daisy chained. The Dynamixel are connected in series making it easy to limit the amount of wires needed. One feature of the ReactorX is that it has microcontrollers embedded in every motor. This allows the user to track position, velocity, current, torques at every joint. The motors also include multiple registers for setting velocity/acceleration limits and PID gains, features that make it ideal for industry standards. The MCU controlling these features is the ST CORTEX- M3, ARM controllers are industry rated ICs. As far as the communication, the protocol being used is the half duplex asynchronous serial communication(8 bit, 1 stop, no parity).

## 3.4.* Knowledge and Skills

Ulises Lazaro:

- Prior knowledge:
    - EE123: Power electronics
    - Python3
    - Mechanics
    - EE144: Robotics
    - C programming.
- New knowledge:
    - ROS
    - Linux Environment
    - Modeling.

Jason Weiser:

- Prior knowledge:
    - EE123: Power electronics
    - EE146: Computer Vision
    - EE144: Robotics
    - Python3
- New knowledge:
    - Gazebo/ROS
    - Linux
    - Visual Servoing
    - Machine Learning

Ronny Anariva:

- Prior knowledge:
    - CS170: Artificial Intelligence
    - EE/CS122A: Embedded Systems
    - CS100: Software Construction
    - Circuit Design
    - Unix

- New knowledge:
    - Python
    - ROS
    - Simulation w/Gazebo
    - Inverse Kinematics
    - OpenCV

Jimmy Le:

- Prior knowledge
    - CS170
    - EE146
- New knowledge
    - ROS
    - Inverse/forward kinematics
    - Differential Drive model/Unicycle model
    - Simulation with Gazebo/MoveIt
    - Linux/Ubuntu environment

## 3.5.* Budget and Cost Analysis

The cost and budget analysis are shown in **Table 3.1.**

| Part | Price |
| --- | --- |
| PhantomX Reactor Arm Kit | $599.95 |
| Raspberry Pi 4 8GB Starter Kit - 8GB RAM | $129.29 |
| HiLetgo 3pcs ESP8266 (6) | $30.42 |
| Arduino Uno (3) | $14.99 |
| HC-SR04: Ultrasonic Sensor (5) | $8.29 |
| Arduino Mega | $15.99 |
| Aluminum | $100 |
| L298 | $5 |
| LM2598 | $5 |
| Motors | $120 |
| Cables | $5 |
| PCB | $5 |

| | |
| --- | --- |
| Wheels and shocks | $200 |
| Bolts, screws, washers | $20 |
| Jetson Nano Dev Kit | $84.99 |
| Total | $1,343.92 |

**Table 3.1**

## 3.6.* Safety

The most critical safety concerns are regarding the electrical components. The frame of the robot is mostly aluminum, a requirement needed to make the structure robust. To prevent electric shock, the circuit and power components were isolated from the frame. In prototyping, wood and cardboard worked well.. A safety switch was also installed.

The construction of the frame required potentially dangerous industry tools. Most of the construction was done following OSHA regulations since the person building the robot, Ulises, has experience in the field.

Pathing for Turtlebot has potentially sporadic movement, so we developed procedures to give the robot adequate area for testing. Object avoidance sensors also mitigated these risks.

## 3.7.* Performance, Security, Quality, Reliability, Aesthetics, etc(N/A)

Procedures to enforce quality standards were limited to informal discussions in the group chat. Weekly meetings with Professor Karydis emphasized integration of subsystems. Performance, Quality, and Reliability were natural test factors in the experiments we ran. Security and Aesthetics were not considered in this prototype.

## 3.8.* Documentation(Everyone)

A GitHub repository was used to maintain the documentation of the code, allowing for editing and saving when necessary. A Google drive was also heavily used to store important information, weekly progress reports, designs, and notes on random useful information. The link to this github can be found in the reference section. This abundance of documentation allowed for better debugging.

A Discord server was heavily utilized, on a daily basis, to discuss problems and maintain accountability. This system of organizing meetings recognized the conditions of working remotely and evolved to react to deadlines and pressures.

## 3.9.Risks and Volatile Areas(N/A)

N/A

# 4. * Experiment Design and Feasibility Study

## 4.1.* Experiment Design

### 4.1.1 Differential drive motor:

Objective:

The objective of the differential drive is to have a mathematical model for the dynamics of the mobile base. The system is modeled as if it only had 2 wheels, the left and right. The differential model uses the encoder sensors to accurately measure the velocity of each wheel depending on its radius. The mathematical relation can be observed in figure 4.1 below.

Setup:

The motors come equipped with the encoders, to test the features of the motors the frame of the robot had to be constructed first in order to accurately test the efficiency of the design. Once installed, several tests were conducted with varying loads to test the response in relation to the weight.

Procedure:

The differential drive model was programmed using the C language, and the Atmega 2650 as the main controller. The encoders provide feedback to measure speed. The tests were conducted in an enclosed environment, using cement flooring, this environment minimized the amount of slippage on the wheels. To test the way the system is able to turn, tests were conducted in the same manner.

Result:

The results for the velocities were successful only in the forward and backward direction. This is because all four electric motors are working in phase, providing the maximum power delivered. The model does not work for turning, this model does not take friction into consideration, the amount of friction on the system was greater than the system could handle. In conclusion, the robot only moves forward and backward with an accuracy of 95%.

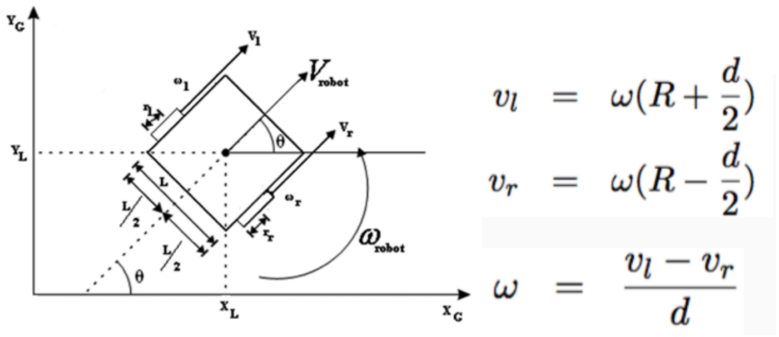



**Figure 4.1, Unicycle Model**

### 4.1.2 Ultrasonic Sensors Experiment:

Objective :

The objective of the ultrasonic sensor experiment was to measure the accuracy of the sensors when used for object detection.

Setup:

A base was constructed with 4 sensors mounted, 2 on the sides and the other on the front and back.
Each one facing in a different direction to simulate the sides of the robot.

Procedure:

The sensors utilized the Atmega to test the range at which the objects would be detected.

Results:

The result from the experiment indicated that the range of accuracy was far from desired, the sensors were not accurate. Although part of the main design, the module was not implemented in the final product.

### 4.1.3 Motor/Encoder Testing:

Objective

To test the amount of torque output by each motor.

Setup:

The motors were installed on the robot base with a full load, meaning all components were integrated to account for the total weight of the machine.

Procedure:

A wide range of voltages were tested, from 0 to 12 V.

Results:

The results were unsatisfactory, the motors did not perform well. The weight was exceeding the capabilities of the motors. Motors with a lower gear ratio are needed.

### 4.1.4 Waypoints of the Physical Turtlebot:

*Higher Level Experiments were done Using a the TurtleBot

Objective

To test the accuracy of the higher level algorithm using a robot with a similar design, the Turtlebot. The A* algorithms is the main module being tested.

Setup

A lab room was used to construct a maze, the robot was to travel the maze without crashing to test the accuracy of the waypoint navigation system.

Procedure

Objects were placed in an area of 10’ by 20’. The code included the map of the environment, This test only tested the amount of slippage. The A* Algorithm was used to calculate the accuracy of the system.

Results:

The TurtleBot provided satisfactory results, the motor and sensors work as intended, the experiment will be described in detail in the next section. Every goal was reached within the desired time.

### 4.1.5 ReactorX 150 Robot Arm Experiment:

Objective:

The objective is to accurately track the position of the end effector. The robotic arm has 5 degrees of freedom which allows the end effector to move freely in all directions.

Setup:

Setting the Robotic arm was done by the Department of Electrical. The setup consists of installing the arm on top of the mobile base, along with all the electrical components. These are provided with power by the base which has its own battery source.

Procedure:

To test the arm, code generated in python programming language help was used to translate the forward and inverse kinematics. There exists a set of libraries called Interbotix, which were a vital part of the project. These help to code the procedure, which consisted of testing the limits of the arm, as well as the time it took to go from one point to another.

Results:

The data in table 4.1 shows all the points that caused inconsistencies.Here the points are within the workspace of the robot, yet they do not seem to work properly. The program crashes, this was due to the time constraints. The rest of the points worked as expected, including the unit circle.

### 4.1.6 Orbbec Astra Pro Experiment:

Objective:

To detect objects based on color and shape using computer vision.

Setup:

The camera was mounted on top of the robot base, giving 180 degrees of vision that makes it efficient to inspect the tomato plant.

Procedure:

Several colors of objects were placed in front of the camera at different lengths and angles. The experiment was to calculate the accuracy to which the camera would stop detecting objects.

Results:

The results were accurate for short distances. The colors were detected with an accuracy of 90% up to 2 meters. After 2 meters the camera did not accurately measure the distance. The algorithm/code was tested on many different colors, although the color red seemed to be the most accurate.

#### 4.1.7 Centroid identification in a simulated environment

Objective:

To identify centroids of Gazebo Models using robot models and camera packages in ROS. This would aid in the automation of the fruit picker control processes.

Setup:

Initialize the ROS environment in RVIZ and Gazebo. Prepare python scripts which input instructions for targets and vision filtering code.

Procedure:

Run through tree models and visually inspect identified results. Refine filtering technique to maximize positive identifications.

Result:

Through our familiarity with similar experiments and physical prototyping we've done, we can assume 95%+ ability to identify red cherry tomatoes on the models we've created.

## 4.2.* Experiment Results, Data Analysis and Feasibility

#### 4.2.1 Differential drive motor:

The turtlebot is based on the unicycle model. A model much simpler than the differential drive model. It is more natural to think about the magnitude of velocity and its direction. The PID controller was implemented with that idea in mind. In the code, the ROS messages included a 6x1 vector which describes components of velocity and direction. These are the inputs to the PID controller, we tell how fast we are trying to go, and the code calculates the error and therefore adjusts to the proper velocity. The desired average velocity for the experiments was ½ m/s. It is worth noting that the velocity is not constant. There is some slow down between waypoints, but this is due to the trajectory implementation used to calculate the smoothest trajectory to the goal.

The Figure below, 4.2.1, demonstrates how the simulated fall would be arranged using cylinders as plant locations. From the start to the first cylinder, the Turtlebot took T = 34 seconds. The reason why it took so long was because the Turtlebot was spinning for 20 seconds trying to get to the first cylinder. From the first cylinder to the second cylinder, the Turtlebot took T = 26 seconds. The Turtlebot was slow to figure out its next objective when looking at the video. From the second to the third cylinder, it took T = 12 seconds. From the third to fourth cylinder, the Turtlebot took T = 15 seconds. From the fourth to the fifth cylinder, the Turtlebot took T = 12 seconds. The time it took for the Turtlebot to go from the fifth cylinder to the sixth is T = 16 seconds. Finally, from the sixth cylinder to the lonely cylinder, the Turtlebot takes T = 14 seconds. From all of the times, then the Turtlebot runs through it's obstacle in 2 minutes and 9 seconds.

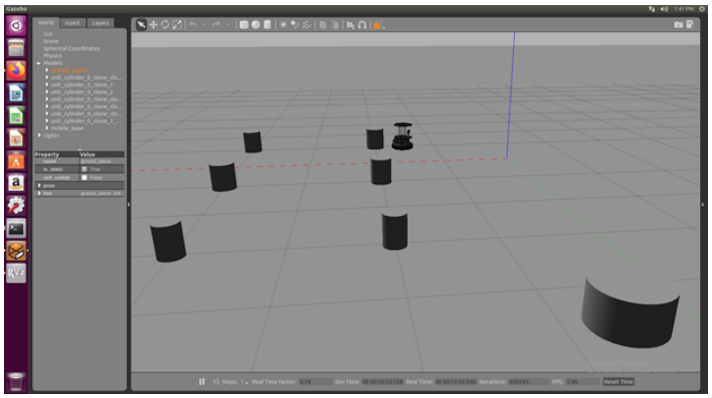
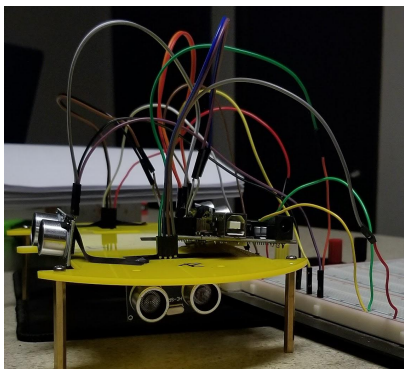


**Figure 4.2.1, Simulated farm using cylinders as plant locations**

### 4.2.2 Ultrasonic Sensors Experiment:

Four ultrasonic sensors were installed as it can be observed in figure 4.2a. The logic of the device could also be obtained from the table 4.2a. The data shows the possible obstruction in front of the robot and its actions. The robot will stop if it detects an object anywhere near its parameter with a tolerance of 1 meter. This will give the robot enough leeway to be able to recalculate the path desired with the aid of the computer vision algorithm.



| Sensor Definition | A3 | A2 | A1 | A0 | Result |
| --- | --- | --- | --- | --- | --- |
| A0: Rear | 0 | 0 | 0 | 0 | ALL |
| A1: Right | 0 | 0 | 0 | 1 | ! Reverse |
| A2: Left | 0 | 0 | 1 | 0 | ! Right |
| A3: Front | 0 | 0 | 1 | 1 | Forward \|\| Left \|\| BOTH |
| | 0 | 1 | 0 | 0 | ! Left |
| | 0 | 1 | 0 | 1 | Forward \|\| Right \|\| BOTH |
| | 0 | 1 | 1 | 0 | Forward \|\| Reverse |
| | 0 | 1 | 1 | 1 | Forward |
| | 1 | 0 | 0 | 0 | ! Forward |
| | 1 | 0 | 0 | 1 | Left \|\| Right |
| | 1 | 0 | 1 | 0 | Left \|\| Reverse \|\| BOTH |
| | 1 | 0 | 1 | 1 | Left |
| | 1 | 1 | 0 | 0 | Right \|\| Reverse \|\| BOTH |
| | 1 | 1 | 0 | 1 | Right |
| | 1 | 1 | 1 | 0 | Reverse |
| | 1 | 1 | 1 | 1 | STOP!!! |

Figure 4.2 and Table

### 4.2.3 Motor/Encoder Testing and voltage regulation:

The encoder is the main component for the feedback controller. Each of the motors has a PID controller that reads the velocity and calculates the error. To test the actual hardware a signal generator was first used to generate a PWM signal, and tested a range of signals until they reached the maximum RPM of 180. THis was a surprise since the motors are rated for 220 RPM. It is possible that the motor needs thicker cable to allow for more current to flow. The encoder can be seen in Fig 4.2, with the oscilloscope the outputs were checked to make sure the two signals were 90 degrees apart. The yellow signal is to measure the ticks. To calculate the velocity, we measure the amount of ticks the encoder has produced during a 10 ms period. The pink signal is used to tell the controller if the wheel is spinning clockwise or counterclockwise. The encoder can produce 3220 counts per revolution when both rising and falling edges are accounted for.

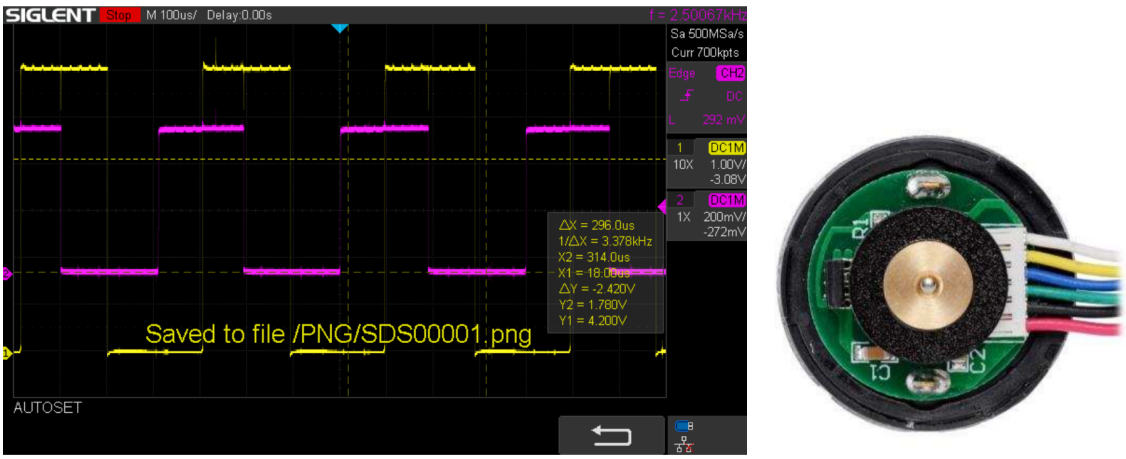



**Figure 4.2b**

**Voltage Regulation Circuits:**

The LM2598 step down transformer was tested and installed in the robot. The buck converter works fine for this application since the motors do not consume more than 1 ampere at 12 volts. The converters are rated for voltages up to 37 volts. More intrinsic information is found on the link to the datasheet. Many of the components other than the motors are rated for 5 volts, the module works better than any voltage divider, it is controlled by frequency and does consume extra energy. It is controlled by the frequency provided by the potentiometer. As it can be seen on figure 4.2c, the input to the device is assumed to be unregulated, the output is regulated with an accuracy of 5%. The desired output is constant with a range of 3-20 volts.

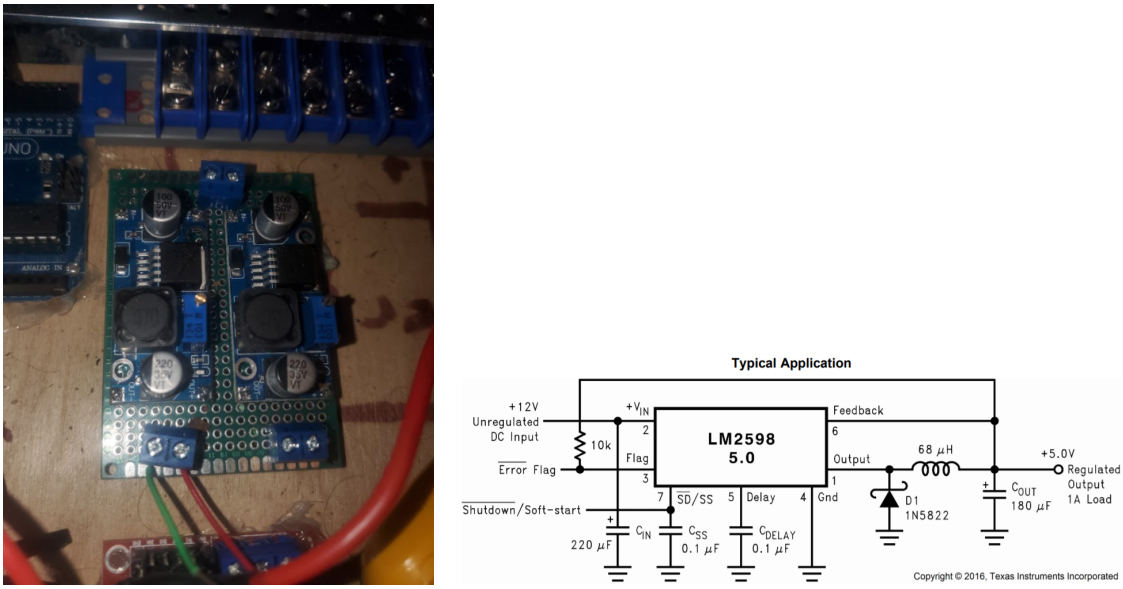



**Figure 4.2c. The figure shows the schematic of the LM2598**

### 4.2.4 Waypoints of the Physical Turtlebot:

From the video above, the Turtlebot moves from (1,1) to (2,2) in 8 seconds, and after that, the Turtlebot moves from (1,1) to (4,2) in another 8 seconds. This is an interesting problem because we set the trajectory generation variable T to be 5 seconds. This means that the Turtlebot should get from waypoint to waypoint in 5 seconds. As mentioned and from counting in the video, you will see that it takes a few more seconds from both waypoints. From (1,1) to (2,2), the Turtlebot moved 2 meters over 8 seconds, so that would mean the rate would be 0.25 m/s. This is calculated by doing rate = distance/time. When the Turtlebot went from (1,1) to (4,2), then it moved (2+sqrt(2)) over 8 seconds. This means that the rate is 0.4268 m/s.

### 4.2.5 ReactorX 150 Robot Arm Experiment:

When a coordinate was given to the arm script, both the RVIZ and physical robots achieved movement. This experiment passed or failed based on which coordinate points created movement which achieved the target point. The robot manipulation libraries have several set positions. In Fig 4.2 all motors are at their zero position, referenced as it's 'home' position. Moving time for the arm is 2 seconds. We found some points failed to converge when the algorithm's calculated trajectory took longer than 2 seconds. Unlike the Turtlebot2, the RX150 library used acceleration controllers. This was visible during experimentation from clearly smoother trajectories.

As any other robot, the ReactorX has its limitations. The workspace of the robot is not optimal for our design but it does work for testing the higher level algorithm. In the original robot we have a workspace that allows the arm to rotate 300 degrees. For the Turtle the base motor is only able to rotate 180 degrees. This is enough space for the robot to pick anything in front of it. In this particular experiment we tested the height of the robot and its capabilities. From the base, the tomato or goal point desired will only be able to reach up to a distance of .4 meters, when the x parameter is .15 meters, At this point and position it is not recommended to pick a tomato since the shoulder joint be forced to take all the strain given by the system.

When the Turtlebot is going through the map searching for a tomato, there is no need to operate the arm so it goes into a sleeping position. Not to say that the arm is off, the arm is functional but this position is optimal for the arm not to interfere with any obstacle along the way.

Testing the Home position in Figure 4.2

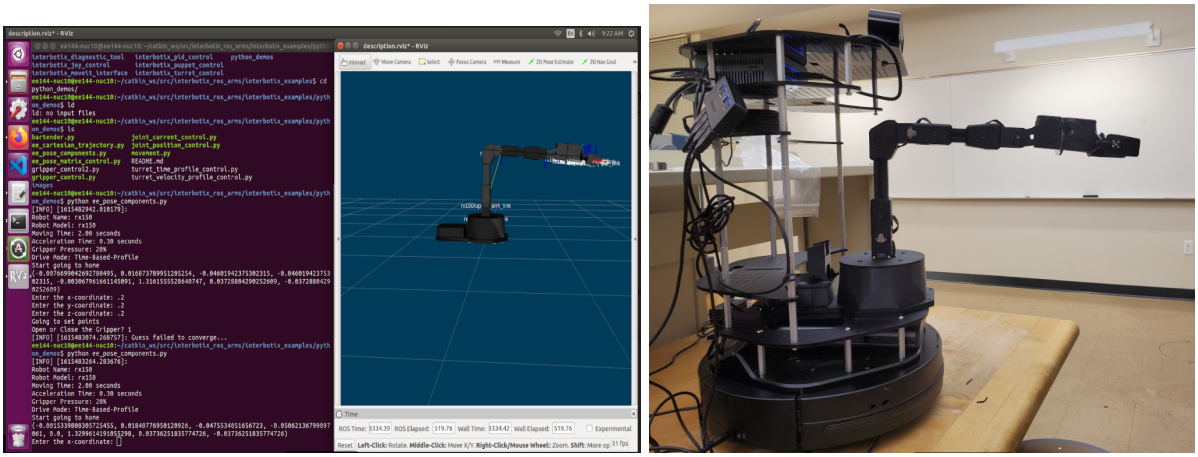


**Figure 4.2,** Testing the Home position in Figure 4.2

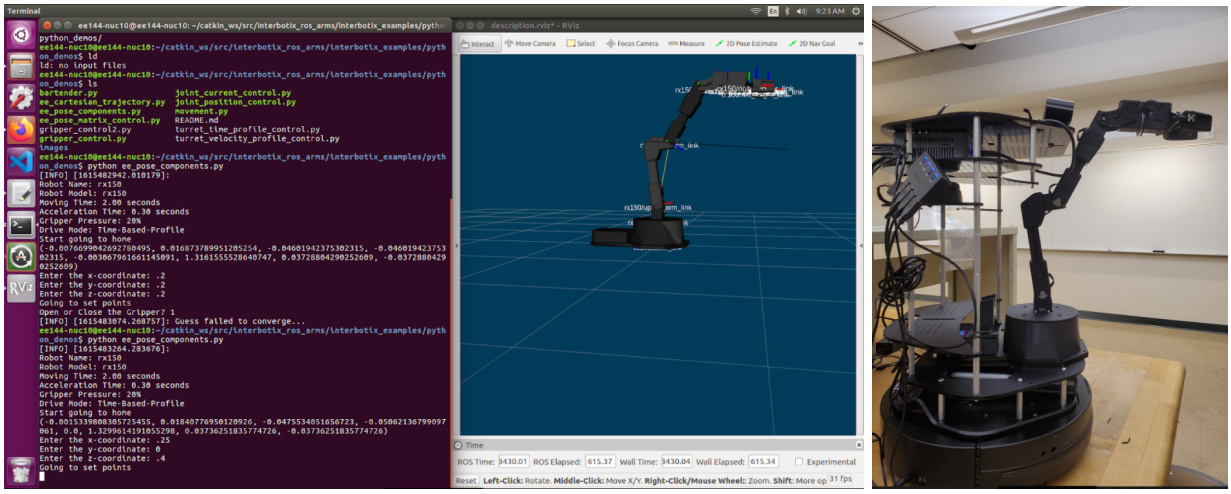


**Figure 4.3** Testing a point relating to vertical limits in Figure 4.3.

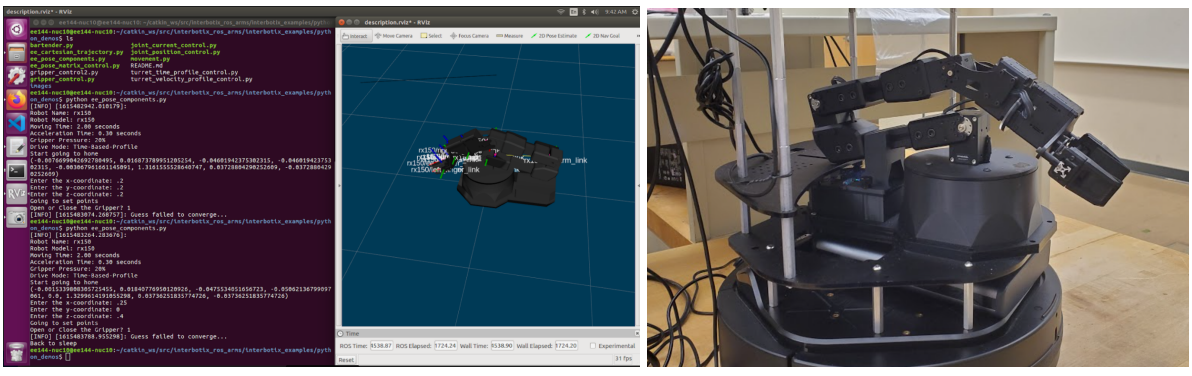


**Figure 4.4** Testing sleep position in Figure 4.4.

The following table below shows the results of the given x,y, and z coordinates. From the results of the experiment, these are some of the test coordinates that we did to ensure that the arm either goes to the given coordinate or fails to go to that coordinate.

| x | y | z | Results |
| --- | --- | --- | --- |
| 0.3 | -0.1 | 0.1 | Fails |
| 0.25 | -0.1 | 0.1 | Works |
| 0.15 | -0.2 | 0.3 | Fails |
| 0 | -0.15 | 0.2 | Works |
| 0.2 | 0 | 0.2 | Works |
| -0.2 | 0 | 0.2 | Works |
| 0.1 | -0.1 | 0.25 | Works |
| 0.2 | 0 | 0.25 | Works |
| 0.23 | 0.2 | 0.3 | Fails |
| 0.23 | 0.1 | 0.3 | Fails |
| 0.12 | 0.2 | 0.3 | Works |
| 0.4 | 0 | 0.25 | Works |

**Table 4.1,** Experimental results shown

After running these points, then the pass rate would be 66.667%. This was calculated by taking the number of tests that passed and then dividing it by the total number of tests. From the tests, we can get the workspace for each coordinate. Note that each of the following numbers are in meters, and the square brackets mean the numbers inside of them are included.

x: [-0.3, 0.3]
y: [-0.2, 0.2]
z: [0, 0.3]

**4.2.6 Orbbec Astra Pro Experiment:**

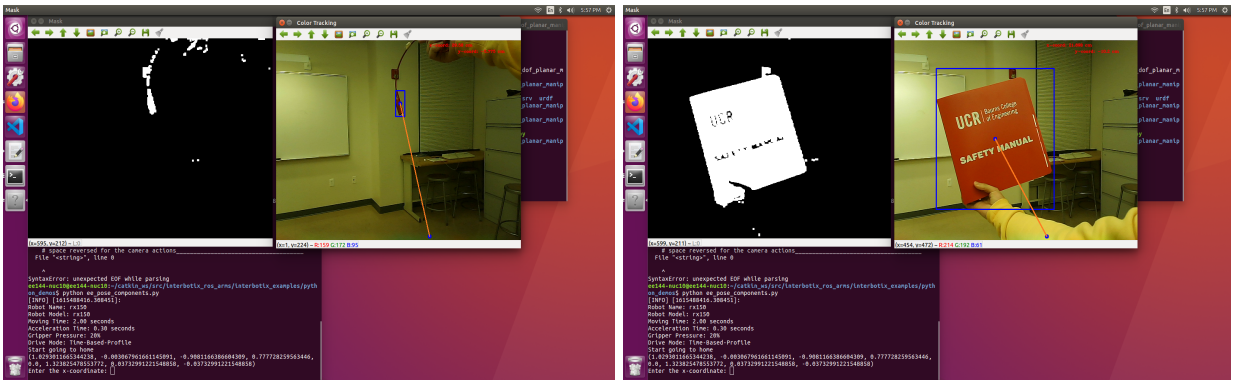



Figure 4.5 Color detection

When performing this test, the goal was to be able to detect the color red. As shown on the images above, the object of interest is enclosed within a box showing the actual detection of the object whose primary color is red. This would then be used to detect tomatoes, red tomatoes in this case, During testing I encountered a few erroneous results. It would sometimes enclose/focus on the desk to the right of the image. For this experiment, different objects were tested such as banana plugs, as the ones used in the lab, a red binder, and even the fire alarm on the walls served as a testing device. The arrived conclusion was that if the color of the object was vibrant enough, the algorithm would prioritise it as the object to focus on. After concluding the experiment, the algorithm was able to detect objects of the color red with an accuracy of ~92%. This was calculated by: total number of tests passed/ total number of tests.

| Object | Distance from Camera(m) | Detection | Expected Result |
|---|---|---|---|
| Marker | 1.5 | Yes | PASS |
| Binder | 2.0 | Yes | PASS |
| Fire Alarm | 4.0 | Yes | PASS |
| Banana Plug | 2.5 | Yes | PASS |
| Pen | 2.0 | No | FAILED |
| Can | 3.0 | Yes | PASS |
| Ball | 2.0 | Yes | PASS |
| Phone Case | 2.0 | Yes | PASS |
| Backpack | 3.0 | Yes | PASS |
| Keychain | 2.0 | Yes | PASS |
| Water Bottle | 2.0 | Yes | PASS |

| | | | |
| --- | --- | --- | --- |
| Nothing | NA | Yes | FAILED |

**Table 4.2**

### 4.2.7 Centroid identification in a simulated environment

I was able to develop centroid detection and image filtering scripts that reliably identified basic orange and red fruit.

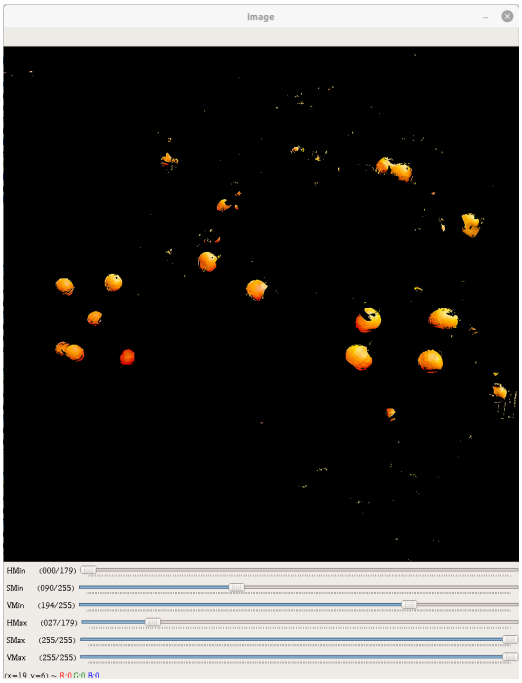



**Fig. 4.2.7a**

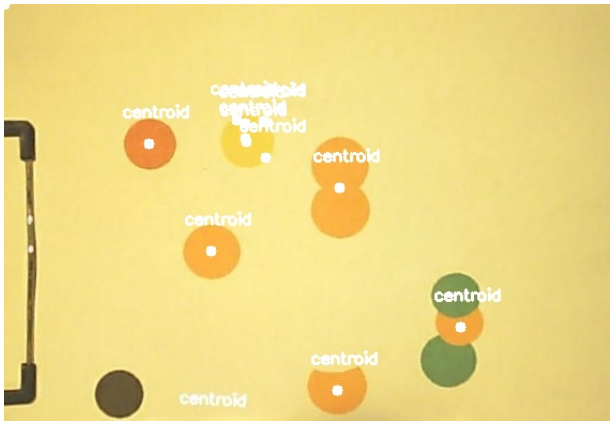



**Fig. 4.2.7b**

Fig 4.2.7a shows results from basic filtering, while Fig 4.2.7b shows results from centroid filtering using a camera on motors and filtering algorithms. Filtering for orange color ranges was partially successful at separating overlapping images and demonstrated limited results with colors outside of targeted color ranges.

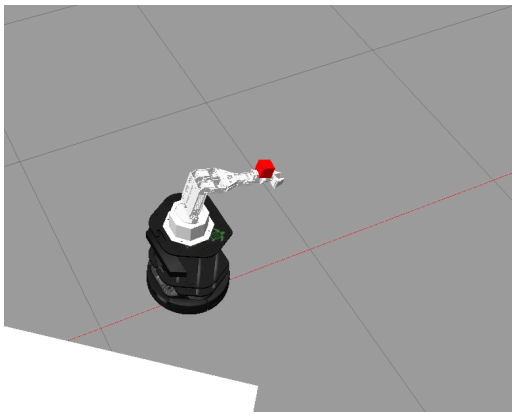


Fig 4.2.7c

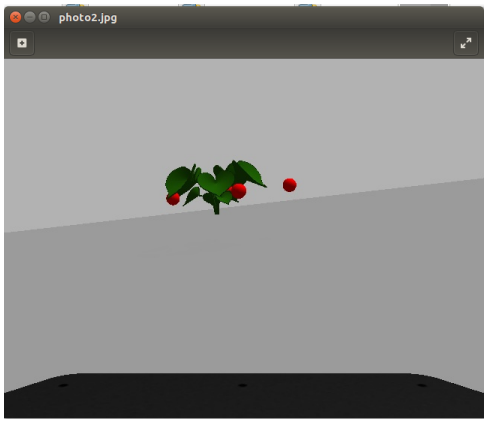



Fig 4.2.7d

Figure 4.2.7c and Figure 4.2.7d show efforts in integrating Turtlebot2 models with ReactorX models using ROS control. Images were extracted, but the development of integration experienced technical delays.

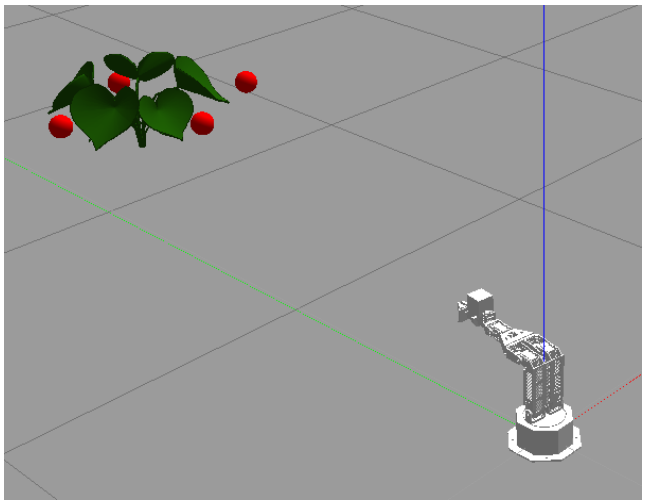


Figure 4.2.7e

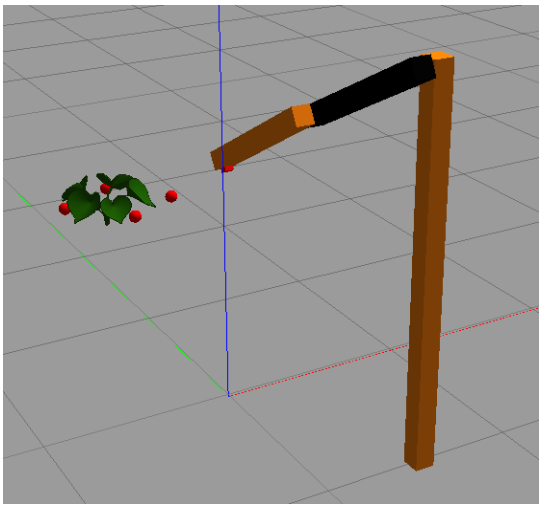


Figure 4.2.7f

Figure 4.2.7e and Figure 4.2.7f show workarounds which isolate camera control to subsystems. Time ran out in integrating centroid extraction, but application of the algorithms to extracted image files is an easy next step.

# 5. * Architecture and High Level Design

## 5.1.* System Architecture and Design

The following diagrams show the high-level overview of the original mobile base. The system shows the flow of logic from the controller to the motors and encoders. The differential drive system cannot function without feedback. The controller initially reads the feedback from the encoders to measure the initial speed of the motors. Depending on such speed the controller will use a PID controller control to accurately track the velocity of each wheel. The final velocity is a predetermined value in the code, depending on how fast the user wants the robot to go. For this specific application the motors have a maximum speed of .5 meters per second. As it can be seen in figure 5.1 the motors are not directly controlled by the Arduino. Each motor receives power through a motor driver

As it can be seen in Figure 5.1, an Arduino Mega was used because it provides over 50 input/output pins. The figure describes how the mobile base functions, specifically how the logic to the motors is coordinated. This is necessary because the robot needed to use 4 encoders, 4 DC motors, and 2 motor drivers to power up the motors. The Arduino Mega will generate a PWM signal to tell the motors the direction they need to spin. High level diagram of motors and controllers shown in Fig. 5.1.

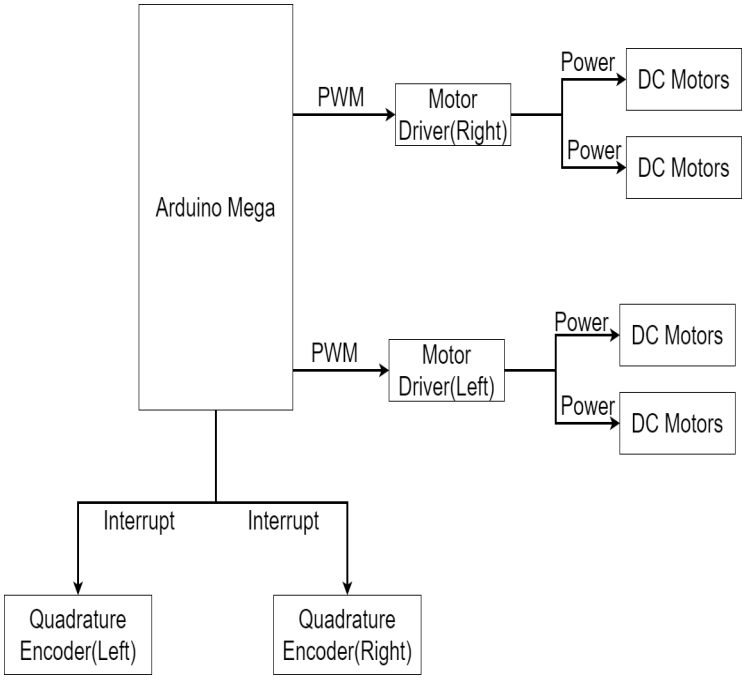



**Figure 5.1 Flow diagram of motor control**

In Figure 5.2, seen below, the system block diagram is shown for the Turtlebot design. The main tool for communication for all of the components is ROS. ROS helps set up the nodes which are needed to boot up the arm, mobile base, and computer vision. Initially the nodes are set up when the script is activated. The turtlebot main script is the master, while every other node acts as servers. Without permission from the master, communication between components is not possible. Here, the ROS Master provides naming and registration services to the rest of the nodes or components in the ROS system. When the nodes are set up, components can communicate with each other individually. Their goal is to

integrate all of the nodes together into one script as seen in the diagram below. The project was done this way because it minimizes the level of abstraction. Sensors can be very complex, so this system allows the sensors to have coordinated peer-to-peer communication.

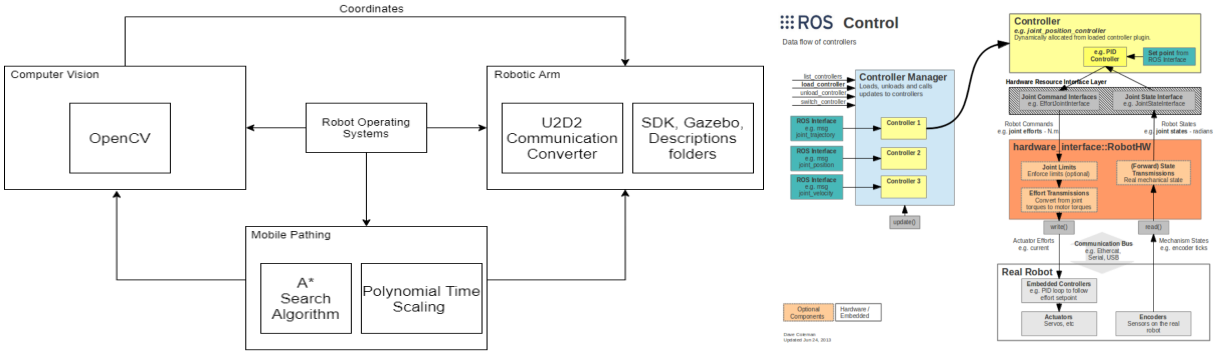



**Figure 5.2 Node communication**

## 5.2.* Hardware Architecture

In Figure 5.4, it shows the block diagram for the voltage distribution for the prototype robot. To distribute the power, several transformers are required. The voltage varies among the components, some require 5 Volts and some required 12 Volts .A 20 volt battery is used to provide 3 amps of current. To distribute the voltage properly 3 different step down transformers are added that limit the voltage to 12 volts and 5 volts from the microprocessor and other miscellaneous components. Most of the power is consumed by the motors. Together the four motors consume 2 Amps, at 12 Volts this is equivalent to 24 Watts. The battery can only deliver 60 watts optimally. The distribution breadboard is only for testing, but it was adequate for prototyping. A custom PCB will be incorporated in future designs that contains the transformer and all wires going from the power supply to the controller.

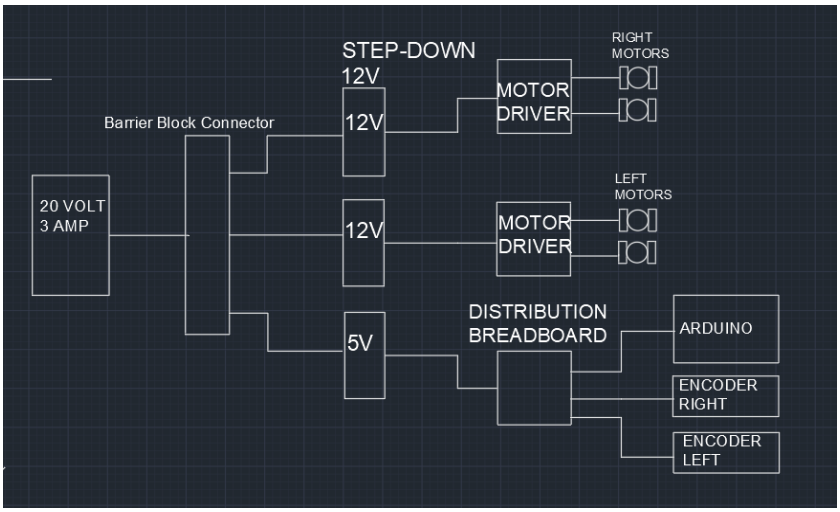



**Figure 5.4 Power flow of system**

## 5.3.* Software Architecture

In Figure 5.5 below, the flowchart for the software architecture can be seen. First, the main operating system used was a windows machine. The ROS interface only works with Linux OS, to overcome this, a virtual machine was used to emulate the environment. To be more specific, Ubuntu 16.6 Distro is needed to communicate with the version of ROS used to program robots. In Ubuntu is where the scripts are launched, these are first set up by turning on the master node (Turtlebot). Once on, all of the robotic components are able to communicate with each other. From this point, the robot will be autonomous until the goal has been met.

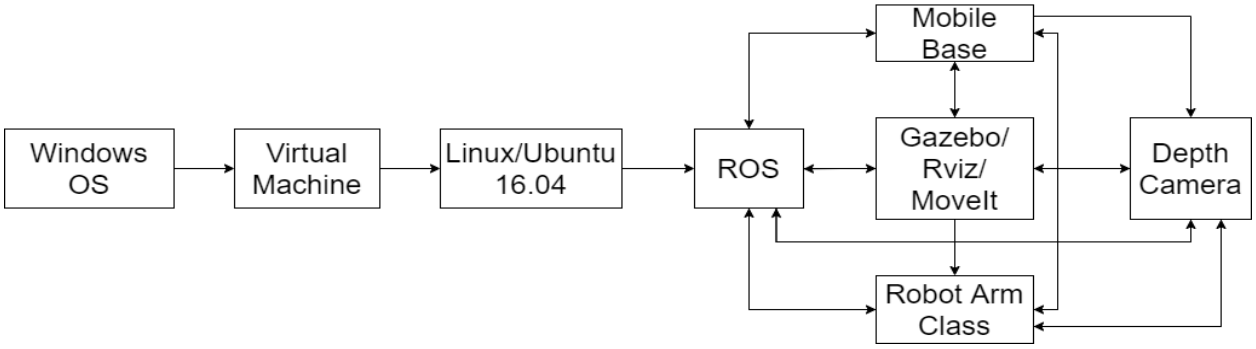



**Figure 5.5 Software flow chart**

## 5.4.* Rationale and Alternatives

The architecture of the robot was designed with feasibility in mind. The system is able to be implemented cheaply compared to other approaches. The software is open source, making the system easy to update and no paid software is needed. Alternatively, the robot base was pre-designed to have 6 wheels instead of 4, the extra 2 wheels provided more ruggedness. This way the robot will not be stuck in tough terrains. The discouragement came with the cost of the extra wheels. It was not cost effective to have them, the motors are an expensive part of the system.

# 6. Data Structures

## 6.1. Internal software data structure

N/A

## 6.2. Global data structure

N/A

## 6.3. Temporary data structure

N/A

## 6.4. Database descriptions

N/A

# 7. * Low Level Design

## 7.1.*Module 1: Motor Control

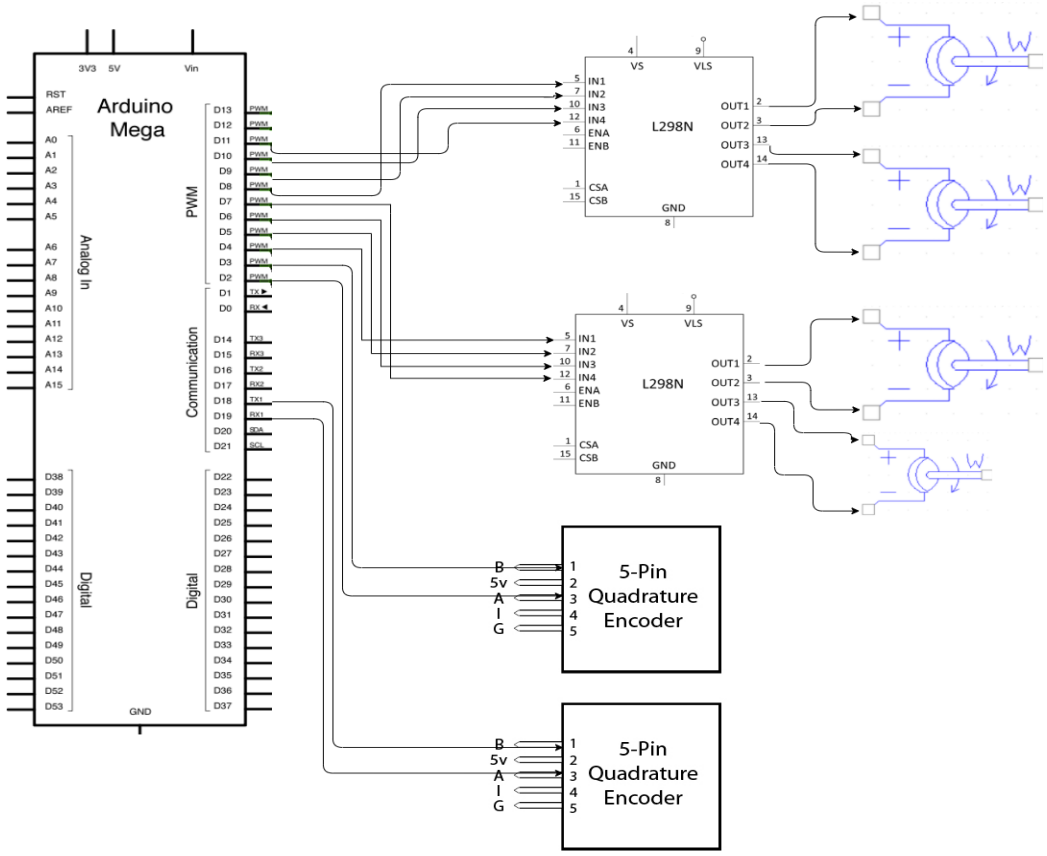



**Figure 7.1 Motor Logic Control**

### 7.1.1. Processing narrative for *module 1*

Fig. 7.1 shows the circuit schematic of the wheel velocity controller. The interrupt pins of the Arduino Mega sample every 10 seconds for an encoder tick, which is converted into linear velocity. This data is given to PID controllers to regulate the motor's performance.

### 7.1.2. *Module 1* interface description

The controller is an Arduino Mega, which receives control from a Raspberry PI. The Arduino communicates with L298N motor controllers, which are connected to motors. Encoders return measurements of velocity, which are given to the PID controller written in the Arduino.

### 7.1.3. *Module 1* processing details

The diagram shown in Fig. 7.1 uses an Arduino Mega, Quadrature encoders, L298N motor drivers, and the motors. These components are controlled by interrupt pins and while loops which continuously monitor the system. It is active while the system is powered.

### 7.1.4. *Module 1* designer

Jimmy and Ulises designed, assembled, and tested this module.

## 7.2. **Module 2:* The A* algorithm

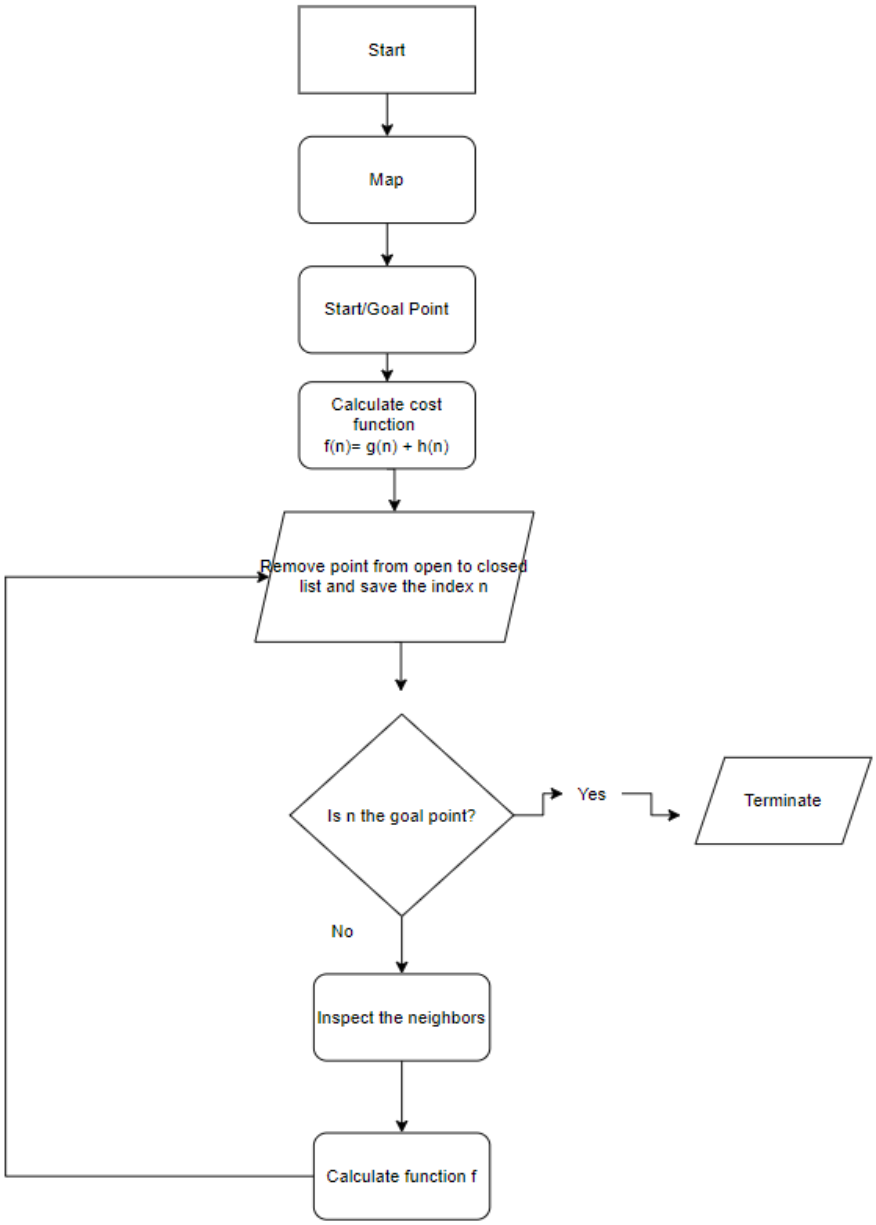



**Figure. 7.2 A* Algorithm**

### 7.2.1. Processing narrative for the A* Algorithm

This A* implementation takes a completed grid map and finds an optimal path through any obstacles. It uses Manhattan distance to calculate the heuristic function.

### 7.2.2. *Module 2* interface description

Input for the A* algorithm is a start point, an end point, and obstacles which are all represented on a grid. The outputs are the waypoints for the generated path, which are sent to the Kobuki motor base controllers.

### 7.2.3. *Module 2* processing details

The A* algorithm runs on the Kobuki Mobile Base. A* is an optimal path search algorithm. The algorithm generates potential paths around obstacles and removes suboptimal results. The waypoint generation algorithm takes this recommended path and generates smooth trajectories through these targets using matrices which contain state equations.

### 7.2.4. *Module 2* designer

Jimmy and Ulises designed, assembled, and tested this module.

## 7.3. *Module 3:* Obstacle Avoidance

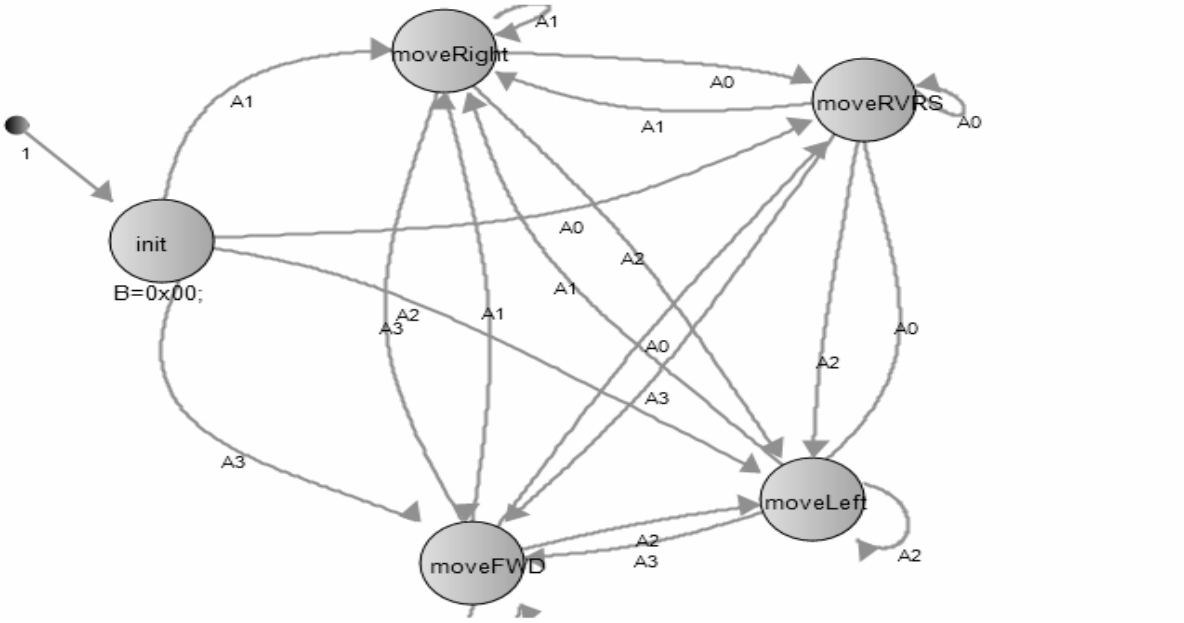



**Figure 7.3 Finite State Machine:** A0= front sensor; A1= left sensor; A2= right sensor; A3= rear sensor.

### 7.3.1. Processing narrative for *Module 3*

The obstacle avoidance feature uses four ultrasonic sensors to detect potentially dangerous objects. The finite state machine of this module is shown in Fig. 7.3. The sensor controller returns a binary signal when an object is detected within a specified range.

### 7.3.2. *Module* Interface Description

Sensory input from the HC-SRO4 ultrasonic sensors feed into the system. Movement instructions are communicated to the main controller.

### 7.3.3. Module 3 processing details

This algorithm protects the cart from damage with random obstacles. HC-SRO4 sensors connect to the edges of the cart, attaching to the main movement controller. Once a sensor detects an object within a distance threshold, it flags the algorithm. The base then moves away from the detected object.

### 7.3.4. *Module 3* designer

Ronny designed, assembled, and tested this module.

## 7.4. *Module 4: ReactorX 150 Arm

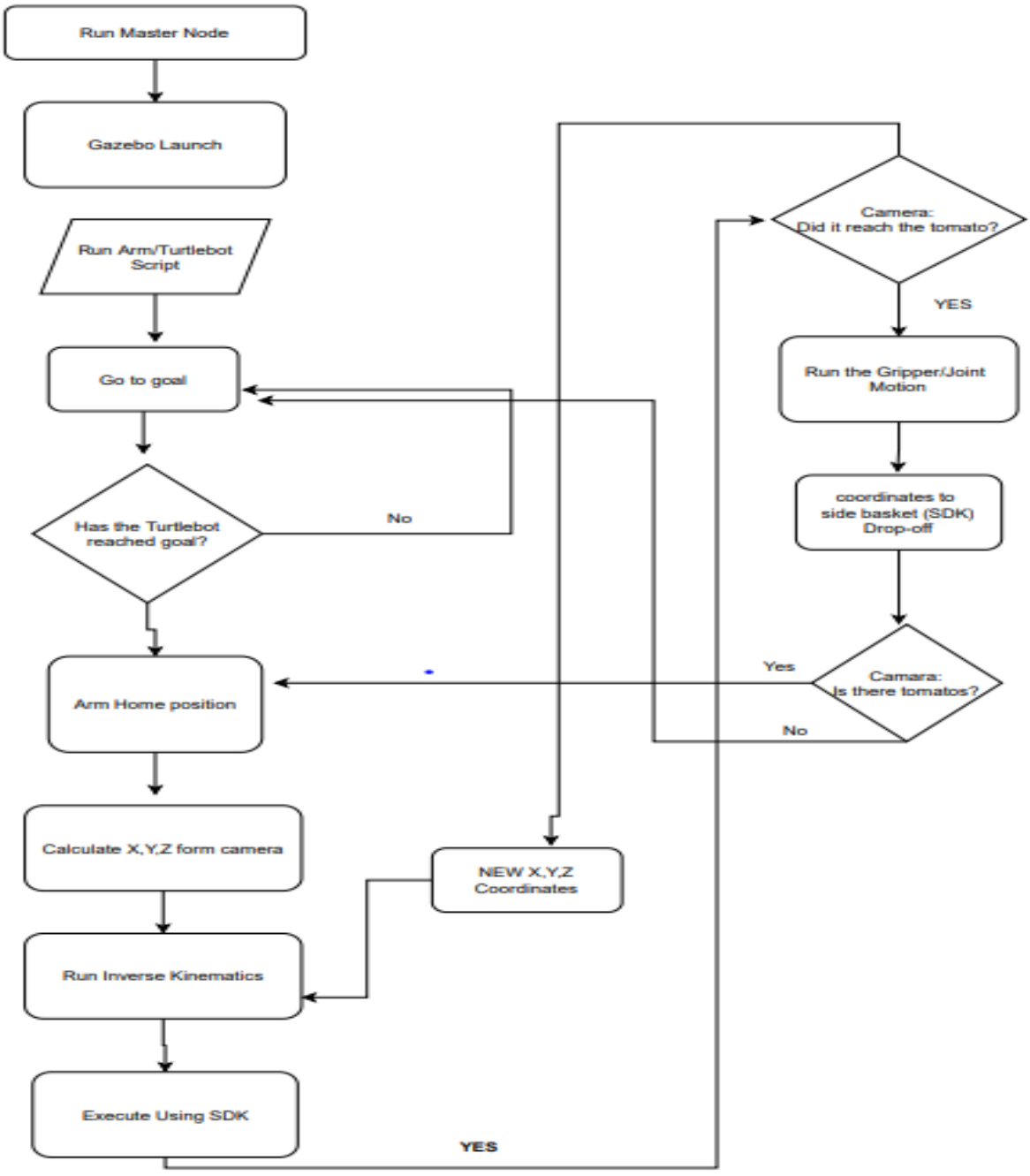



**Figure 7.4**

### 7.4.1. Processing narrative for *module 4*

Fig. 7.4 shows the robot arm and camera flowchart. ROS and Gazebo are first launched to activate the master node. A Python script feeds target locations into ROS which sends control signals to motors. When Turtlebot2 reaches a target position, the arm camera locates a tomato and feeds position feedback information to the arm to complete the motion. The arm calibrates using a home position of the camera. It uses inverse kinematics scripts from the Interbotix libraries. The target object is then picked and delivered to the case on the side of the robot. Once all the tomatoes are picked then the turtlebot will go to the next pot and start the algorithm again until the whole map has been traversed.

### 7.4.2. *Module 4* interface description

The robot arm takes inputs from the camera and target coordinates so it can work on the environment. It outputs information which updates the controller on it's status.

### 7.4.3. *Module 4* processing details

This system uses an Orbbec Astra Pro Camera and ReactorX 150 robotic arm. The ReactorX 150 is limited in its range of motion. The Astra Pro camera has a resolution of 640x480, which limits its ability to detect objects.

### 7.4.4. *Module 4* designer

Jimmy and Ulises designed, assembled, and tested this module.

## *7.5. *Module 5: Orbbec Astra Pro+ Camera*

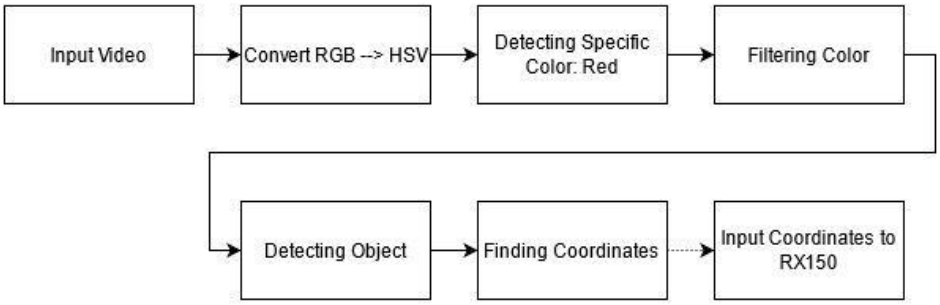



**Figure 7.5**

### 7.5.1. Processing narrative for Orbbec Astra Pro+

The Orbbec Astra camera starts once the robot reaches a pot. Then the color detection algorithm begins detecting objects in specific color ranges. When an object is detected, the terminal will display a box enclosing this object as well as a line connecting the origin with the object. Finally, the image

displayed on the terminal will also display the x,y coordinates of the object with respect to the end effector of the ReactorX 150.

### 7.5.2. *Module 5* interface description

The algorithm for color detection takes as input live video being retrieved by the Orbbec camera. It uses the color detection algorithm to return xyz coordinates.

### 7.5.3. *Module 5* processing details

This module utilizes an Orbbec Astra camera and RX150 arm, as shown in Figure 7.5. The algorithm converts camera input from RGB to HSV format, returns a specific range of color properties, places the values into a mask, and stores it as a binary image. This filters out unnecessary data and focuses on the object of interest (OoI). The algorithm selects the object with the largest area to minimize the effects of noise, and bounds it with a box, returning the coordinates of the location. These coordinates are now the origin which relates distance to the end defector of the RX150 arm. This makes the calculations of the distance to the OoI less complex.

### 7.5.4. *Module 5* designer

Ronny designed, built, tested, and assembled this module.

## *7.6.*Module 6: The entire Integrated System*

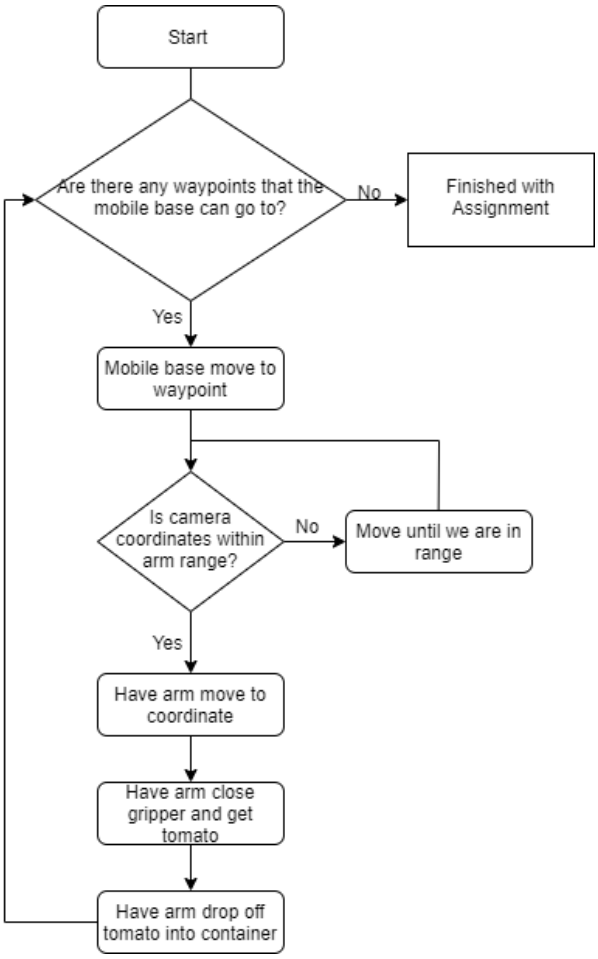



**Figure 7.6**

### 7.6.1. Processing narrative for *module 6*

The flowchart of the overall system is seen in Fig. 7.6. The system checks for start/end points. Without any, the Turtlebot is terminated. Otherwise the model needs to develop a path to the target. Once the Turtlebot is at its destination, the camera locates the cherry tomato and reports x,y, and z coordinates to the robot arm. If those coordinates are valid, then the robot arm will pick up the cherry tomato, otherwise, the camera would have to move around to get valid coordinates. When the arm has the tomato in the gripper, the turtlebot goes to the location of the container and drops off the tomato. If there are no more tomatoes, it stops.

### 7.6.2. *Module 6* interface description

This model takes in start and stop waypoint coordinates, similarly to Section 7.2.2, along with any map data. Arm configuration coordinates may also be entered into another interface. The only output interface that is present is printing out waypoints from the A* algorithm is Section 7.2.2.

### 7.6.3. *Module 6* processing details

The main hardware that was used for the Turtlebot design are the DYNAMIXEL XM430-W350-T and DYNAMIXEL XL430-W250-T servo motors for the robot arm, the Orbbec Astra Depth Camera, the Intel NUC, and the Kobuki Mobile Base. The main design constraint for this Turtlebot design is the fact that the Kobuki Base only operates indoors. The lack of integration with the camera systems limits behavior to a set of dummy instructions.

### 7.6.4. *Module 5* designer

Ronny designed, built, tested, and assembled this module.

# 8. * Technical Problem Solving

## 8.1.* The Original Mobile Base Problem

A problem encountered was the issue of weight. The frame of the robot was too heavy to turn right or left. The motors did not have enough torque to move the robot once all the components were installed. It was able to move forward and back perfectly. Since the robot has no steering mechanism, the differential drive model turns wheels in opposite directions in order to turn. As a result it needs to overcome friction, the motors were not strong enough for this procedure. Ulises was in charge of building the robot platform and concluded that bigger motors were needed, specifically ones with a higher torque ratio. Motors with at least twice the torque since the weight of the tomatoes was not accounted for. The robot alone ended up weighing 15 lbs, the motors are designed to push 16 lbs. The robot should work ideally yet such circumstances are not obtainable.

## 8.2.* Solving the Original Mobile Base Problem

To fix the problem of the motors bigger motors were needed, unfortunately, there were budget constraints and this option was dismissed. On the other hand, Professor Karydis allowed a robot from the department to be used. This worked as a solution because the team only needed to test the high level algorithms. The robot build was a small portion of the project, the major part was the autonomous aspect. The team used the Turtlebot for the rest of the project and achieved the desired results.

## 8.3.* The End Effector Problem

In this problem, the arm did not work with the desired interface called MoveIt. This interface was supposed to allow the robot to be programmed without much code but had bugs that could not be fixed. It was not compatible with the gripper motors for unknown reasons. Ulises was able to code the arm but had issues manipulating the end effector gripper. The MoveIt interface was supposed to work with the whole robot arm. The MoveIt interface has many features that make it possible for the robot arm to be manipulated easily. The robot arm is able to move to the end effector to any position desired, but the gripper did not open or close properly. As a result, hard code was created to control the gripper.

## 8.4.* Solving the End effector Problem

The gripper problem was solved by taking another approach, by using the Interbotix libraries to manipulate the robotic arm. The manufacturer provided the team with several examples that demonstrated opening and closing the gripper, with speed and torque control for the gripper. The only issue is that the arm must be controlled purely by using Python scripts. Once the arm issue was resolved, Ulises was in charge of coding the kinematics of the arm.

## 8.5.* The Computer Vision Implementation Problem

One of the main tasks of the project was implementing CV so that the camera could send coordinates to the robot arm. The issue arose when the actual distance was calculated. The algorithm developed was not accurate. The coordinates returned to the computer were off by meters.

## 8.6.* Solving the Computer Vision Implementation Problem

The team was not able to fully implement computer vision into the project successfully, but was able to get coordinates, albeit not accurately and up to scale. The algorithm's result is close but not enough to obtain acceptable results. The color detection is the only part that works as desired. The algorithm can differentiate only objects but cannot calculate the distance from the robotic arm to the desired goal.

## 8.7.* The Gazebo-Turtlebot integration Problem

ROS can be very particular about mixing different robots together in the same file. Jason thought incorporating the Turtlebot2 with the PhantomX arm would be an important and easy first step in developing our simulation models. Unfortunately an attempt to link two URDF's together led to confusing errors that required a thorough knowledge of ROS control to solve. The models display properly but control of the arm doesn't work.

## 8.8.* Solving the Gazebo-Turtlebot integration problem

Eventually Jason created a custom cart in two separate URDF namespaces, merged everything into one URDF, and simulated necessary behavior in other systems to address this problem. Jason used a pendulum and the PhantomX by itself to study the laser rangefinder and custom camera systems.

# 9. User Interface Design

N/A

## 9.1. Application Control

N/A

## 9.2. User Interface Screens

N/A

# 10. * Test Plan

## 10.1.* Test Design

**Test Case 1: Trajectory Accuracy**. Executed by Ulises and Jimmy

**1.1 Objective**

This experiment verifies that the mobile base trajectory is correctly created from the waypoint generation algorithm. The A* algorithm is further refined with these results.

**1.2 Experiment setup**

An environment is used to describe the farm requirements. The laboratory was used because there was no access to a farm with moderate flat ground. Several obstacles were placed to test the flexibility of the algorithm.

**1.3 Experiment procedure and how to collect data**

As the robot executed the A* algorithm, it set the proper waypoints. This was coupled with the gathered data, this data was visible real time from the virtual machine. The data was saved in the robot's computer then compared with the physical data gathered by measuring the position of the robot. Several experiments were conducted with different start and end goals. Once it reached a destination and stopped, the robot measured the discrepancy between actual and predicted coordinates.

**1.4 Expected results**

The trajectory was quite not as expected, not optimal. But when reaching the waypoints the robot was off by +/- 0.1 meter due to the slips of the wheels. The slip cannot be reduced, in fact it will only increase in a more uncontrolled environment. Overall the robot reached its destination regularly.

**Test Case 2: Gripper Strength**: Executed by Jimmy and Ulises

**2.1 Objective**

The purpose of this experiment is to measure the pressure that the gripper can apply on the cherry tomato without crushing it. This tests the InterbotixRobot Class inside of the "robot_manipulation.py" script. This script allows the user to choose a wide range of pressure sensitivity.

**2.2 Experiment setup**

The experiment was conducted in a controlled environment, a laboratory. The robotic arm was installed on the base, no need to remove to test the unit. Cherry tomatoes were tested by manually putting the fruit inside the gripper forks.

**2.3 Experiment procedure and how to collect data**

The gripper pressure parameter inside of the "robot_manipulation.py" script was adjusted to the desired force. The collected data indicated that 50% of the motors maximum torque was enough to pick any tomato up to 5 ounces. The robotic arm is equipped with a feedback system that returns the real time torque from each motor. The feedback system data is all saved in the robot's computer and may be retrieved by the user.

**2.4 Expected results**

As expected the results were satisfactory, the motors have enough torque to pick most cherry tomatoes. The smart feedback system coupled with the robots interface allow the robotic gripper to be versatile. The torque values retrieved from the experiments and parameters can be set to keep the arm from crushing the tomatoes.

**Test Case 3: Color detection**. Executed by Ronny

**3.1 Objective**

The goal of this experiment is to test the accuracy of the color detection algorithm.

**3.2 Experiment setup**

Objects were placed near the camera's field of view. A range of distances helped define the accuracy of the camera. The distances used varied from 0 to 4 meters away from the camera's sensors. Using different color artifacts to tune the algorithm and test its accuracy.

**3.3 Experiment procedure and how to collect data**

Collection of data was done by observation as well as by the competitors algorithm. Verifying if the color detected was the desired one.

**3.4 Expected results**

The experiment showed that the robot could detect only the color red accurately, other colors were distorted.

**Test Case 4: Simulation of Obstacle Avoidance**. Executed by Ronny

**4.1 Objective**

The goal of this experiment is to verify the logic behavior of the obstacle avoidance feature. This was only possible to be done by setting the parameters through keyboard input and printing out onto the terminal the direction of motion.

**4.2 Experiment setup**

The obstacle avoidance was taking four(4) inputs defined as A0, A1, A2, and A3, if either of these were to equal to 1 then this simulated the detection of an object resulting in the robot moving in direction away from this obstacle.

**4.3 Experiment procedure and how to collect data**

After running the algorithm, it requested input data for the sensors through the terminal. The user passed on the requested info and the algorithm would return a direction of motion if a "1" was passed onto any of the sensors. Inputs are of the form: A0= 0; A1= 1; A2= 0; A3=1; chosen randomly. This example input indicates obstacle detection on the front and right side of the mobile base.

**4.4 Expected results**

Expected results are motion in the forward and/or left direction.

## 10.2.* Bug Tracking

A bug was discovered in the A* algorithm which caused the trajectory of the mobile base to behave in unexpected ways. For example, there was a test where the mobile base would have to go to two waypoints. It reached the first waypoint fine, but when it reached the second waypoint, it moved diagonally into the opposite direction from the goal. It would randomly execute paths that were not optimal, hence not satisfying results of the A* algorithm.

There were not any bugs with regards to testing the gripper pressure because the results are either the arm crushes the cherry tomato, or that the arm doesn't crush the cherry tomato. There should be no bug that arises when testing gripper pressure.

There were many bugs in the color detection algorithm. Often the camera would test positive when no objects were in front of the camera. Sometimes when holding a smaller object the algorithm would focus on the hand rather than the red object of interest. These bugs were identified by Jimmy and Ronny. Time did not allow for a proper solution, must be delayed for future work,

## 10.3.* Quality Control

All of the test cases mentioned in Section 10.1 will be reviewed.

For test case 1, the waypoints start = (1,1), end = (3,2) succeeded. The waypoints start = (3,3), end = (1,1) were unable to move backwards, so the test failed. After working on that for a considerable amount of time the bugs could not be fixed, but the problem was pinpointed by the polynomial time scaling function.

For test case 2, the team tested a couple of gripper pressure percentage values to find the best one. Tested 3 percentages: 20%, 30%, and 40% . All tests passed for this section, which was completed by Ulises.

For test case 3, the camera was tested to detect only red objects. The test failed in multiple tries, the last attempt was spring quarter.

For test case 4, the different inputs of the simulation need to make sense in terms of which direction the robot should move. If there is an obstacle in front of the original mobile base, then it should either move backward, left, or right. When testing out an obstacle that was in the front, and to the right, then it should go backward and/or left. The test failed.

## 10.4.* Identification of critical components

This robot is composed of three major components: mobile base, robotic arm and computer vision. Each of them are composed of sub-components that require attention. For example the mobile base runs the A* algorithm, which needs the necessary attention to make sure it runs/behaves as expected. Similarly, the robotic arm runs an inverse kinematics algorithm that demands attention.

From the gripper pressure experiment, the critical component that involves this test is the gripper of the arm. To run the gripper of the ReactorX 150, the "InterbotixRobot" Class uses the function that is necessary to control the gripper. The functions that were used to test the gripper strength are the open function, close function, and the gripper strength function.

## 10.5.* Items Not Tested by the Experiments

Depth perception was not tested, nor was it needed because the robot avoidance algorithm was not implemented.

# 11. * Test Report

## 11.1.* Test 1: Trajectory Accuracy

Figure 11.1 below shows a picture of the robot executing the path finding algorithm. The green points on the floor are equivalent to the x and y points in a cartesian coordinate system.

**Figure 11.1 Room test: Testing actual path trajectory**

1. Test results, person performing the test

   It depends on where the mobile base starts, but when testing out the motion pathing, the mobile base ends up within +/- 0.1 m difference. Ulises and Jimmy performed the experiment.
2. Comparison with expected results

   The test results match with the expected results in section 10.1.
3. Analysis of Test results

   The summary of the test showed that the A* algorithm cannot make a satisfactory trajectory backwards. The code only works going forward, to be specific, the code does not fully take into account the vector angle related to velocity and it has trouble creating a proper set of trajectory points. For instance, when the robot is at a non home point and it is programmed to go back home, the algorithm returns a set of points outside the map. As a consequence the robot drives off the path.
4. Corrective actions taken

   Unfortunately, there was not enough time to fix this incorrect behavior. If given more time, then the team can change the order of the polynomial time scaling to increase the accuracy of the pathing of the Turtlebot.

## * Test 2: Testing the Gripper Strength and Delivery Method

Person(s) performing the experiment: Ulises and Jimmy

Testing the gripper strength of the ReactorX 150. This test is to figure out what percentage of the gripper strength is used so that it can safely grab a cherry tomato without crushing it. The different "gripper_pressure" values that were tried are: 40%, 30%, and 20%.

**Figure 11.2 Testing the force necessary to pull tomato**

1. Test results, person performing the test

From the gripper pressures listed above, all three gripper pressures did not crush the tomato. Anything above 40% caused the tomato to compress and split. Test results, Jimmy and Ulises both performed the test.

2. Comparison with expected results

The comparison from grippers strengths did not vary much, the gripper does not need much torque to hold a tomato.

3. Analysis of Test results

The data from the motors demonstrated that with 20% gripper strength the arm could hold most size tomates while being efficient in power.

4. Corrective actions taken

If needed the robot's gripper strength can be updated with one line of code. The corrective action will only be needed if the size of the tomatoes increases.

Testing delivery methods:

To deliver the tomato the team fabricated a small container that will store the products until it is ready for delivery to the final goal. The motion of the picking is mimicking the motions performed by a real human being. The process is not abrupt, the harvesting of the fruit is done with a twisting motion of the wrist. The wrist holds the tomato tightly with enough force to pluck the tomato without damaging it.

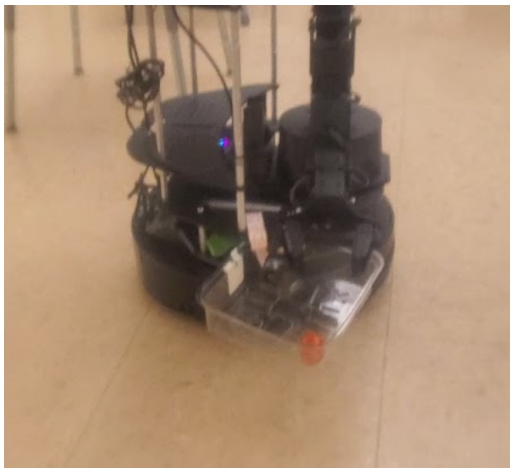


**Figure 11.3**

## Test 3: Color detection

Person(s) performing the experiment Ronny

1. Test results, person performing the test
   Testing for the accuracy of the color detection algorithm was performed by Ronny. The result was actual detection of red objects, giving priority to larger sized objects first and once these items have been removed it moves onto the next largest object.
2. Comparison with expected results
   As shown previously in section: , the algorithm detects colors with a ~92% accuracy.
3. Analysis of Test results
   The results showed that this algorithm with the parameters that were set worked more with brighter shades of red. It would detect more of the reflective colors rather than the matte/opaque colors.
4. Corrective actions taken
   New high and low H,S,V parameters were passed onto the algorithm for better accuracy of detection

## Test 4: Simulation of Obstacle Avoidance

Person(s) performing the experiment Ronny

1. Test results, person performing the test
   The person in charge of performing this test was Ronny, and the outcome was the following: Given the following input: A0= 0; A1= 1; A2= 0; A3=1, the result was for a message displayed on the terminal describing possible motion in the: left direction.
2. Comparison with expected results.
   Table 11.4 below shows the logic of the sensors for object detection.

| Sensor Definition | A3 | A2 | A1 | A0 | Possible Direction of Motion |
| --- | --- | --- | --- | --- | --- |
| A0: Rear | 0 | 0 | 0 | 0 | ALL |

| | | | | | |
| --- | --- | --- | --- | --- | --- |
| A1: Right | 0 | 0 | 0 | 1 | Reverse |
| A2: Left | 0 | 0 | 1 | 0 | Right |
| A3: Front | 0 | 0 | 1 | 1 | Forward \|\| Left \|\| BOTH |
| | 0 | 1 | 0 | 0 | Left |
| | 0 | 1 | 0 | 1 | Forward \|\| Right \|\| BOTH |
| | 0 | 1 | 1 | 0 | Forward \|\| Reverse |
| | 0 | 1 | 1 | 1 | Forward |
| | 1 | 0 | 0 | 0 | Forward |
| | 1 | 0 | 0 | 1 | Left \|\| Right |
| | 1 | 0 | 1 | 0 | Left \|\| Reverse \|\| BOTH |
| | 1 | 0 | 1 | 1 | Left |
| | 1 | 1 | 0 | 0 | Right \|\| Reverse \|\| BOTH |
| | 1 | 1 | 0 | 1 | Right |
| | 1 | 1 | 1 | 0 | Reverse |
| | 1 | 1 | 1 | 1 | STOP!!! |

**Table 11.4 The above is a truth table representing every possible expected output.**

The test results partially satisfied the expected outcome. When the output was displayed onto the terminal, it was missing the possible scenario in which it could move in reverse or both left and reverse simultaneously.

3. Analysis of Test results

   Test result satisfied our goal, which is to avoid obstacles. Now it is just a matter of refinement onto what is a better instruction for the robot to follow.

4. Corrective actions taken

   Rather than giving it more than one possible path to follow, we will just use one option for it to follow. Once it executes and will now test to see if there are more obstacles and if there aren't then it will keep moving in its current direction.

# 12* Conclusion and Future Work

## * Conclusion

**Ulises**:

In conclusion, the project was mostly successful, the only part missing was the computer vision. As part of the team, I single handedly constructed the robotic base for the first prototype, mechanically it was sturdy, the issue was the chosen motors. The differential drive model was coded by Jimmy and I. The code was a complete success. While having the robot elevated the system works as desired. The PID controller takes care of the accuracy. Unfortunately, the implemented the higher level algorithm using the Turtlebot and ReactorX worked as required by the specification and needs. For the higher level algorithms, I wrote the A* algorithms, Jimmy helped implement the goal points. We were successful in getting a robot from point A to B. The Arm was also a successful part of the project. Jimmy and I figured out how to manipulate the arm using the interbotix libraries, we failed on how to figure out the MoveIt interface when it came to the gripper. Personally, I learned much about ROS and Interbotix libraries. I feel comfortable programming any robot, I learned many skills, but the most important is robot manipulation. The project would have been successful if we had a few more sensors integrated into the system. Overall, the implementation of individual components was present, just not completed, it will be in the upcoming months.

**Ronny**:

This project had a huge impact on my college career. The life lesson learned with this project was perseverance. No matter how things seem to be going wrong, one has got to keep trying until reaching a goal. This is basically a summary of the project, things seemed to not be going how we wanted but we managed to push through. Now on a more technical level, I learned about computer vision, robotics(which I was always interested in but never had the opportunity to explore it as we did during this class), the process of designing/developing a project, integration of previous knowledge from previous courses on this project, and even how different things are now with this new “online” format. Also, while doing research I came across how much this field of robotics is growing and how applicable our project was to different scenarios with only minor changes.

**Jimmy**:

My knowledge of the aspects of the project and my professional growth grew significantly. In terms of the technical aspect, I learned a lot about the ROS libraries and how it is the foundation to many robot applications. I also learned about the differential drive model for a robot to move around and adjust to different environments. In addition, I also learned a little bit about the robot manipulator, specifically, the forward kinematics, the inverse kinematics, and the “workspace” of the robot arm. I also went into the simulations that were also compatible with ROS and learned about how to use those simulations to alter our project with the environment created in the simulation. I have rarely used Linux before this class, so it was good to learn about the Linux environment and the certain ROS distribution Ubuntu Kinetic. In terms of personal growth, this project taught me to have some type of backup plan when the original plan does not work out. This project has also taught me to communicate more frequently and effectively with my

group members to make sure that we are aware of what we are doing. Essentially, I learned the importance of teamwork and communication so that the project runs smoothly.

**Jason**:

Most of my work involved reading papers on Computer Vision and reverse engineering Turtlebot2 and PhantomX ROS packages. URDF files which describe model behavior interact with packages which define dependencies and python files which interact with Topics. My largest goal was to integrate a Turtlebot2 model with a PhantomX model, which proved to be very difficult. Each system utilized dozens of files and required a thorough development process to test changes. It was necessary to have functional simulation models to implement the CV strategies though. The difficulty with integration was disappointing. Extraction and filtering of images was completed on other models though, which included a pendulum and much simpler cart design. Basic filtering and control strategies were developed on mockup physical models. I outlined several relevant papers in the field, for instance “Survey on Visual Servoing for Manipulation” by Kragic and Christensen, or “Learning Visual Servoing with Deep Features and Fitted Q-Iteration” by Lee, Levine, and Abbeel.

## Future Work

- Build out autonomous decision making
- Build on mobile arm prototype
- Navigation sensors
- PID controllers
- Communication
- Convolutional neural network
- Training an object identification model using a dataset

## * Acknowledgement

We would like to give special thanks to Professor Karydis and Lu Shi for giving us the opportunity to ask them technical and/or personal questions. We would also like to give thanks to Abhishek Aich for also helping us in terms of technical questions when it comes to computer vision.

# 13* References

[1] R.Davies, J.Kollewe, “Robocrop: World’s first raspberry-picking robot set to work,” theguardian.com, para. 26, May 2019. [Online]. Available:

https://www.theguardian.com/technology/2019/may/26/world-first-fruit-picking-robot-set-to-work-artificial-intelligence-farming

[2] “Strawberry-picking robots to gather enough fruit for Wimbledon,” roboticsresearch, para 4, July 2019. [Online].Available:
https://www.roboticsresear.ch/articles/17565/strawberry-picking-robots-to-gather-enough-fruit-for-wimbledon

[3 ] Ubiquity Robotics Raspberry Pi Image(2020), [XY File]. Available:

https://downloads.ubiquityrobotics.com/pi.html

[4]] “Object Tracking Robot Using ROS MoveIt and OpenCV | Arduino” Robotics and ROS Learning. July, 2019. [Online]
https://www.youtube.com/watch?v=qTbHxIIyrYA&t=374s

[5] James Bruton. Building a ROS Robot for Mapping and Navigation #1. (Aug. 3, 2020). Available: https://www.youtube.com/watch?v=q1u_cC-5Sac

[6] “Industrial standards for robotic machinery,” Safety and Health Magazine, para 3, April 2018. [Online]. Available:
https://www.safetyandhealthmagazine.com/articles/16866-industrial-standards-for-robotic-machinery

[7] Index of ROS Enhancement Proposals (REPs), REP Standard 000, 2000

[8] Coordinate Frames for Mobile Platforms, REP Standard 105, 2010

[9] Depth Images, REP Standard 118, 2011

[10] ROS distribution files, REP Standard 137, 2013

[11] ROS distribution files, REP Standard 141, 2013

[12] ROS Indigo and Newer Metapackages, REP Standard 142, 2014

[13] ROS distribution files, REP Standard 143, 2014

[14] ROS Package Naming, REP Standard 144, 2015

[15] ROS distribution files, REP Standard 153, 2018

[16] General requirements for all machines, OSHA 1910.212, 1978

[17] Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots, ISO 10218-1:2011, 2011

[18] “3 Degrees of Freedom Manipulator” Robotics and ROS Learning, July 2019 [Online]
https://github.com/bandasaikrishna/3-DOF_Manipulator

[19] L.Todes, M.Trossen, S.Wiznitzer, “InterbotiX X-Series Arm & Turret ROS Packages,” GitHub, 27, October 2019 [Online] https://github.com/Interbotix/interbotix_ros_arms

## 14* Appendices

*** Appendix A:** Parts List

- When using the Turtlebot Design(Note that the only thing that was attached to the Turtlebot design was the plastic container):
    - Plastic Container
    - Alligator Clips
    - Cardboard box

*** Appendix B:** Equipment List

- When using the Original Design:
    - Raspberry Pi
    - Arduino Uno
    - HC-SR04 sensors
- When using the Turtlebot Design:
    - ReactorX 150 Robot Arm
    - Kobuki Mobile Base
    - Orbbec Astra Pro Camera
    - NUC

*** Appendix C:** Software List (URL to online drive or SVN server, with sharing set to Public. Can omit this appendix if your project didn't involving writing a program)

- Python Language
- ROS
    - Gazebo
    - MoveIt!
- VMWare(From Native Windows to virtual Linux/Ubuntu)
- OpenCV
- Cisco AnyConnect (VPN)
- RIMS/RIBS

**Appendix D:** Special Resources

- Video Demonstration of the Project
    - https://drive.google.com/file/d/1-pM2o9QWGu-WYmfWx90fODOk-Ovk3oxo/view?usp=sharing

# Fruit Picking Robot

Ulises Lazaro
Jason Weiser
Ronny Anariva
Jimmy Le

## Team Members

Ulises Lazaro
Undergrad
Department of ECE
University of California, Riverside

Jason Weiser
Undergrad
Department of ECE
University of California, Riverside

# Motivation

- Learn Robotics
- Introduce a Solution to the Farm workers Issue
  - Lack of Farmers, Retiring
  - Only 9% of farmers are younger than 35

[1]-Research and development in agricultural robotics: A perspective of digital farming

# Project Goals

- Pick cherry tomatoes and deliver it to a basket
- Design and optimize hardware
- Design Robot Operating System (ROS) control systems
- Integrate sensors
- Interact with map of environment

# Overview

- Turtlebot2
  - High Level Algorithms
  - Interbotix Libraries
- Prototype
  - Hardware focus
  - PID controllers
  - Power systems

Fig. 1 Mobile arm prototype with Reactor arm

Fig. 2: Turtlebot2 with RX150 arm

# Prototype Hardware

- Arduino Mega
- L298 Motor Driver
- LM2596 DC-DC Buck Converter
- 50:1 gear motors, 64 CPR, 16 kg-cm
- 20V, 3 Amp battery
- Raspberry Pi 4
- Misc.
  - Aluminum
  - Shocks

Fig. 1 Completed prototype

Fig. 2 Phantomx Reactor

Fig. 3 Arduino Mega

Fig. 4 Motor driver

Fig. 5 Gear motor

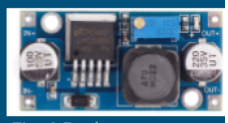


Fig. 6 Buck converter

# TurtleBot Hardware

- Kobuki Base
- Depth Perception Camera
- Netbook Computer (ROS Compatible)
- ReactorX 150 Robot Arm
  - 5 Dof

Fig. 1 TurtleBot2

# Pathing

- Unicycle model
- A* Algorithm
  - Calculates the shortest path.
- Generate smooth trajectories using polynomial time scaling.
- How to test?
  - Give it a map.
  - Give the Turtlebot a start and an end goal.

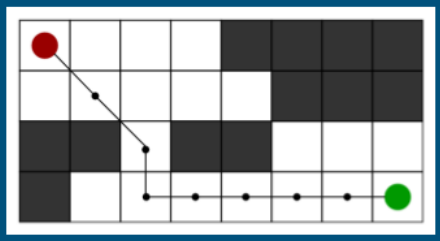


Fig. 1: A* algorithm example

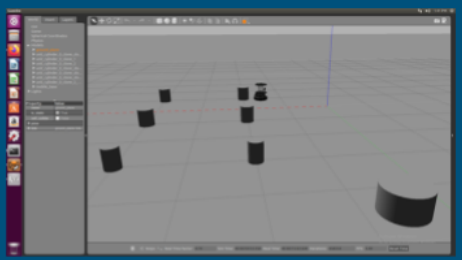


Fig 2. Gazebo model of environment

# Arm Kinematics

- Interbotix_sdk(Physical Arm)
  - Launches a specific node that boots up the physical arm
- Interbotix_Gazebo and Interbotix_descriptions(Simulation)
  - Launches the necessary model into the simulation
- How to test?
  - Input coordinates to the arm
  - Test the range of the arm(Works/fails)
  - Mimic movements of picking a cherry tomato

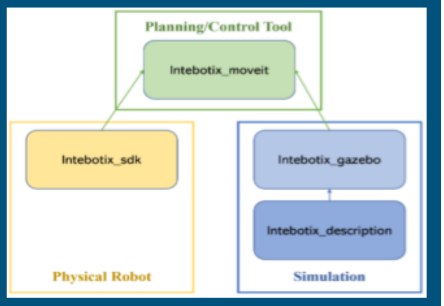



Fig 1. Flowchart

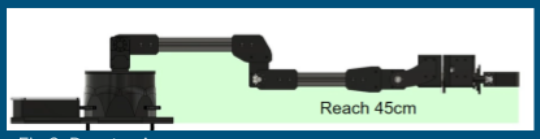



Fig 2. Reactor Arm

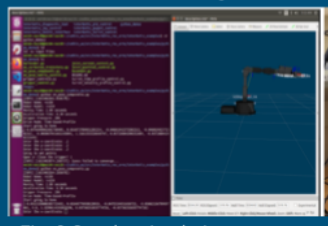


Fig. 3 Gazebo simulation

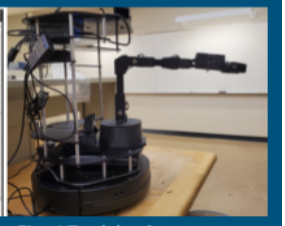


Fig. 4 Turtlebot2

# Pathing and Kinematics Results

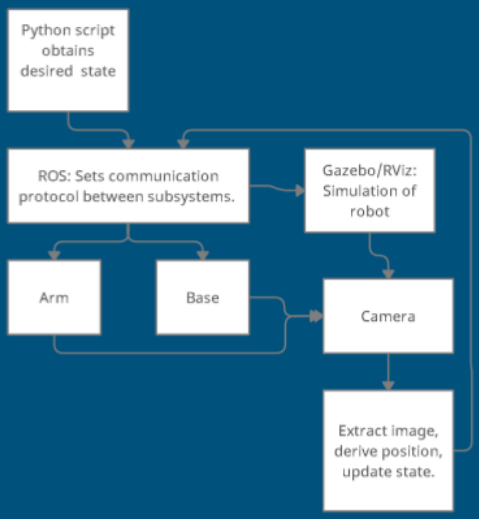
Experiment 1: Path generation

Experiment 2: Path generation, object retrieval

Experiment 3: Path generation, object retrieval

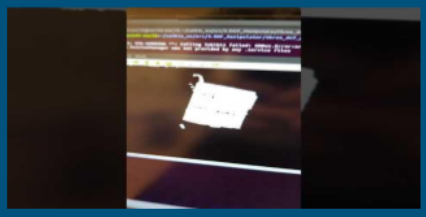
# Control flowchart

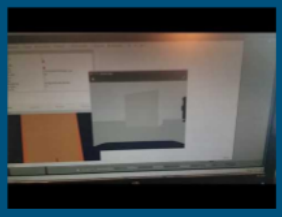
- Python scripts
- ROS
- Base and arm
- Gazebo simulation
- Computer Vision
- Motion planning

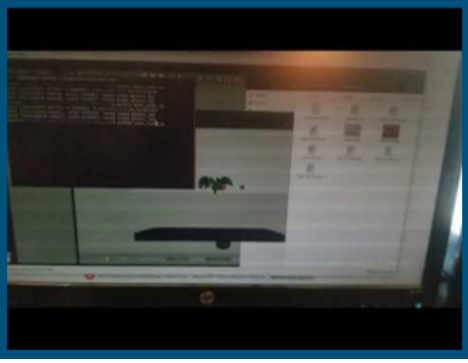


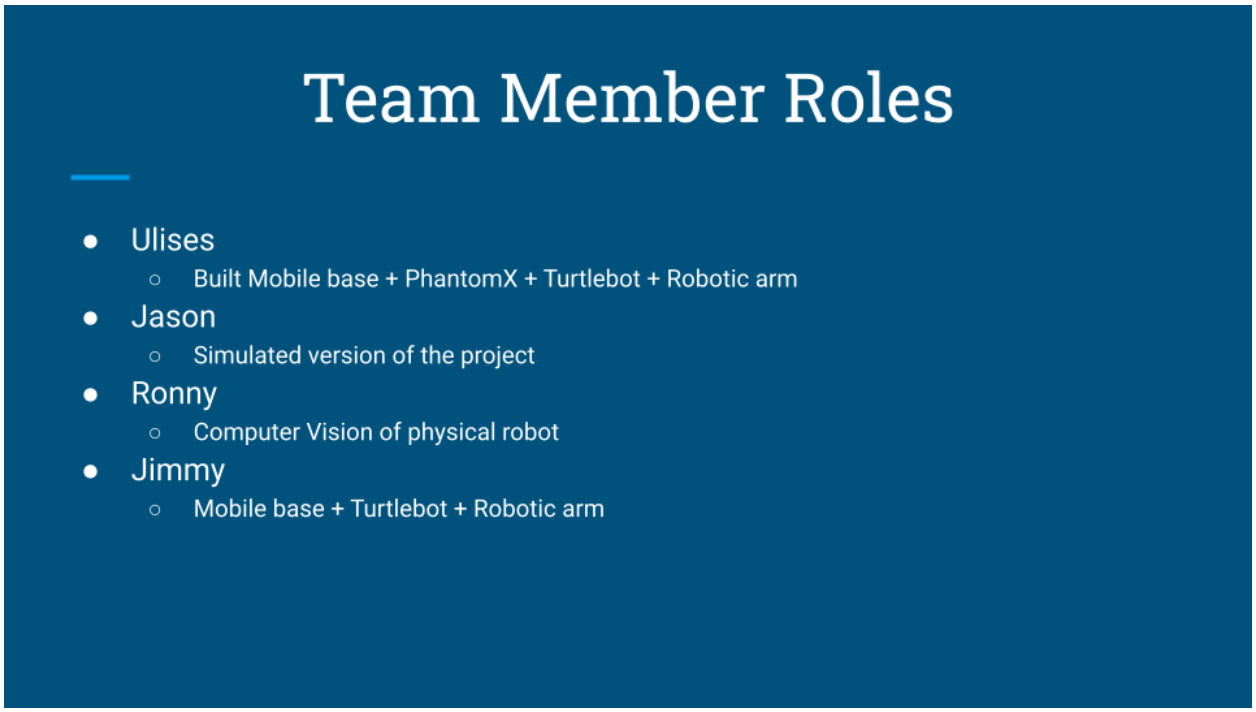

# CV with Turtlebot2, Gazebo

- Integrate camera data into control decisions.
- Extract pictures, filter, identify objects, return coordinates
- Simulate models, cameras, rangefinders, obstacles in Gazebo








Team Member Roles
Ulises
Built Mobile base + PhantomX + Turtlebot + Robotic arm
Jason
Simulated version of the project
Ronny
Computer Vision of physical robot
Jimmy
Mobile base + Turtlebot + Robotic arm

# Future Work

- Build out autonomous decision making
- Build on mobile arm prototype
  - Navigation sensors
  - PID controllers
  - Communication
- Convolutional neural network
  - Training an object identification model using a dataset

# Conclusion

- Constructed a mobile arm prototype
- Implemented Turtlebot2 ground control
- Implemented Turtlebot2 arm control with object manipulation
- Implemented CV algorithms on Gazebo simulations