Final Report

Department of Systems and Computer Engineering

Carleton University

By:

Changlin Liu 101048980

Frank Xu 101050120

Yisheng Li 101028686

Colin McKay 101028772

Date: April 9, 2021

Table of Contents

1.0 Project Charter

1.1 Problem Motivation

In the age of automation, robots can cut costs and improve safety in the workplace with increased accuracy. Amazon has created robots to streamline the fulfillment process by having robots deliver shelves of items to workers for packaging and delivery. By using robots, Amazon reduces the time workers need to search for items on the warehouse floor. Our team seeks to develop a similar delivery robot but with the goal of being able to pick up a package from a destination then self-determining a path to follow to deliver the package. With this robot we hope to:

- Reduce the time used to find the correct path from pick up to drop off destination
- Increase safety by removing human error in driving
- Increase uptime by not requiring a human worker to be available
- Reduce costs by removing the cost to hire a human driver

1.2 Objective

The objective of University of Pennsylvania is to create an autonomous package delivery robot for SYSC 4805 that follows coloured lines on the ground and can find, pick up, and drop off packages.

1.3 Deliverable

The autonomous package delivery robot will consist of the main vehicle, a robotic arm to pick up and drop off packages, and sensors. The two types of sensors will be vision sensors and proximity sensors. The vision sensors will be used to detect the colour of the line the robot is following, and the proximity sensor is a safety measure that detects obstacles that may appear on the track. At

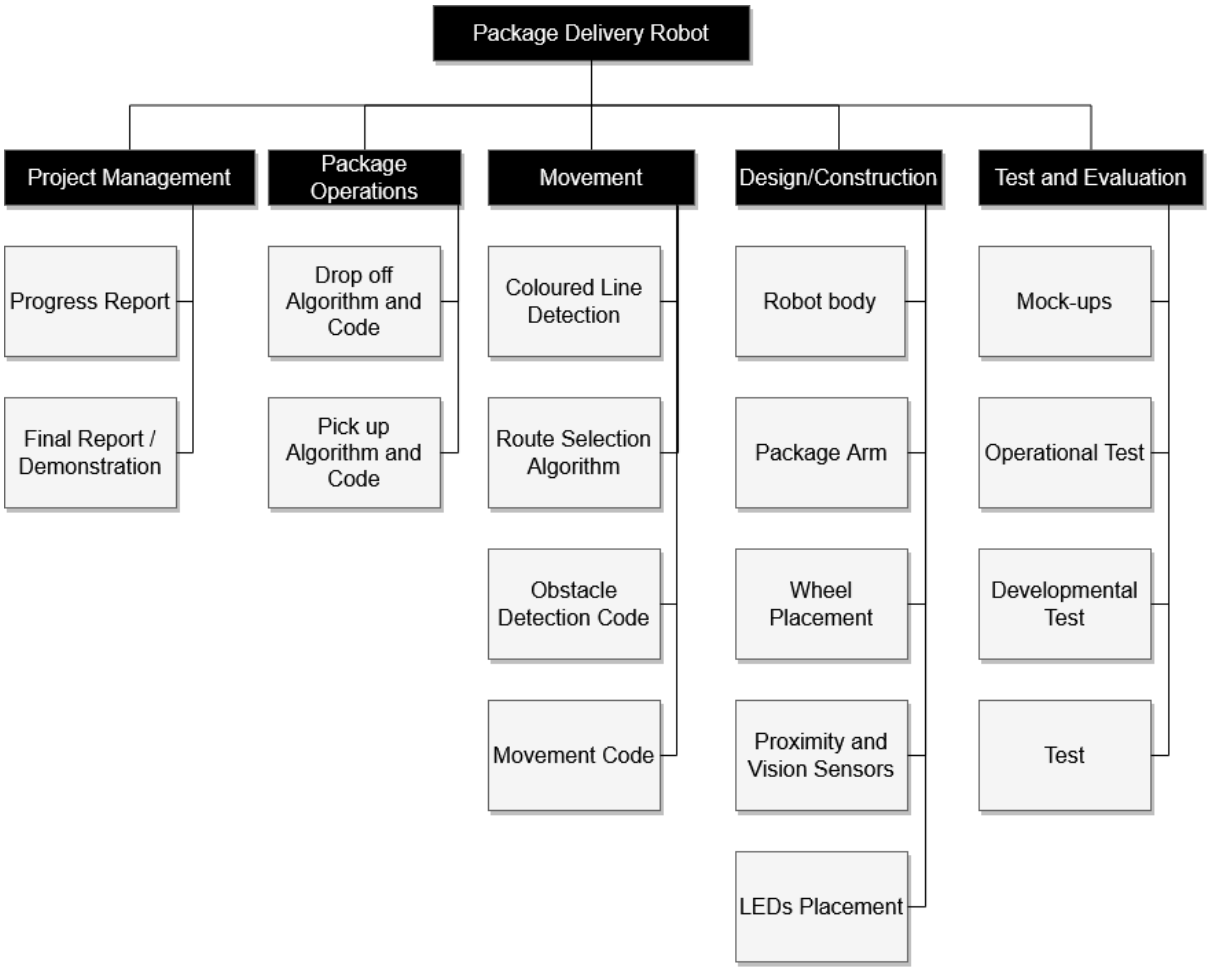
an intersection, the branching lines will be different colours. The main body of the robot will have an area to store picked up packages and house the electronics although it will not be shown as this project will strictly be done in a simulation. The robotic arm will be responsible for the pick-up and drop-off of packages. Package stations will notify the robot when packages are available for pickup.

2.0 Scope

2.1 Requirements

- The user shall input a sequence of pick-up and drop-off destinations and the robot shall automatically determine the path to make the deliveries.
- When the robot arrives at a destination, the robot shall drop off a box into the designated area or pick up an available box from the designated area.
- When a destination is specified, the robot shall automatically navigate to the destination.
- While the robot is traveling, the robot must stay within the designated lines and not veer off course with an error rate of less than 1%.
- While traveling, the robot must be able to detect obstacles on the path, stop, and wait for the obstacle to clear before continuing.
- While traversing, the robot must be able to detect the different colored lines on the ground to select the correct path, with a success rate of 99.5%.
- The robot's cruising speed must be at least 2 robot lengths per 10 seconds.
- The map must have multiple colours paths detectable by the robot to guide the robot.
- The robot arm shall be able to pick up boxes in the designated area with an error rate of less than 5%.

2.2 Work Breakdown Structure:



2.3 Test Cases

- Design a hard coded test so that the robot runs through every state but does not perform any pickups or drop offs.
- Drive the robot with a package. Ensure cruising speed is at least 2 robot lengths per 10 second.
- Set the robot at a package station with a package available. Ensure the package is loaded into storage correctly.

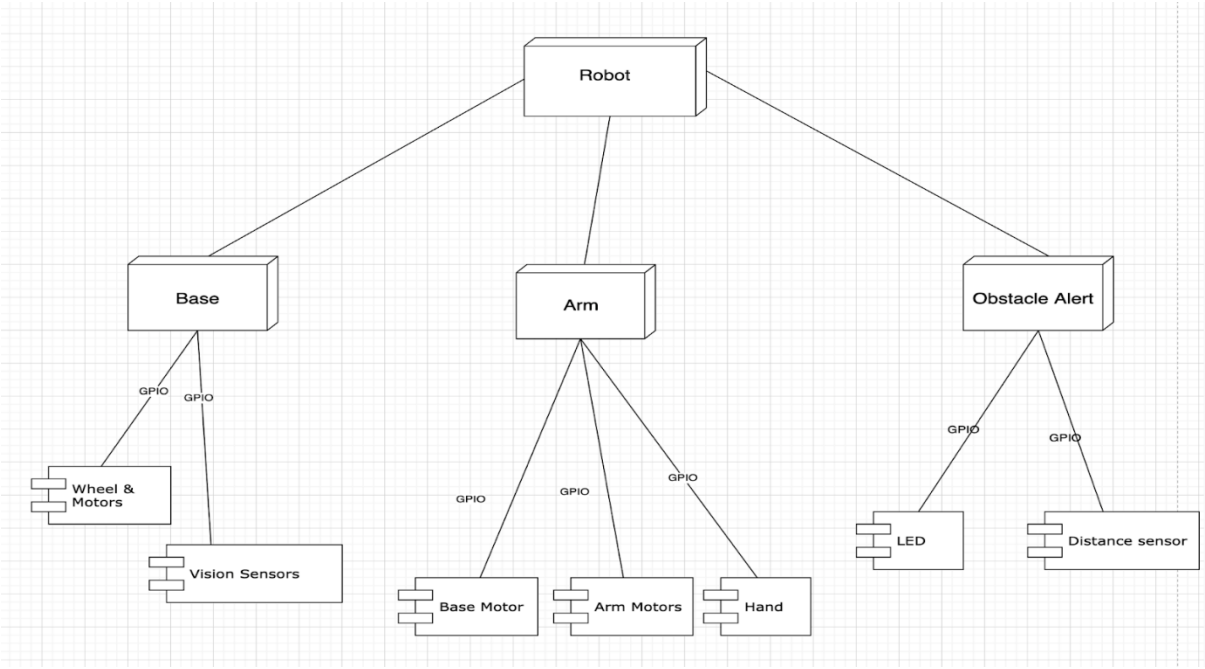
- Set the robot at a package station with a package available. Ensure the robot can successfully navigate to the destination.
- Start the robot preloaded with a package and test the dropped off ability at every package station. Time the speed of the drop offs and pickups. Ensure time taken is less than 30 seconds.
- Design a test so that the robot visits every destination and travels down every path. Ensure the robot never veers off course, and always traverses the intersecting nodes correctly.
- Run the robot along a path with an object in the way. Ensure robot detects the object and stops before hitting the object.
- Start robot with a designated area with a package available for pickup. Ensure the robot goes directly to the correct destination.

3.0 Design

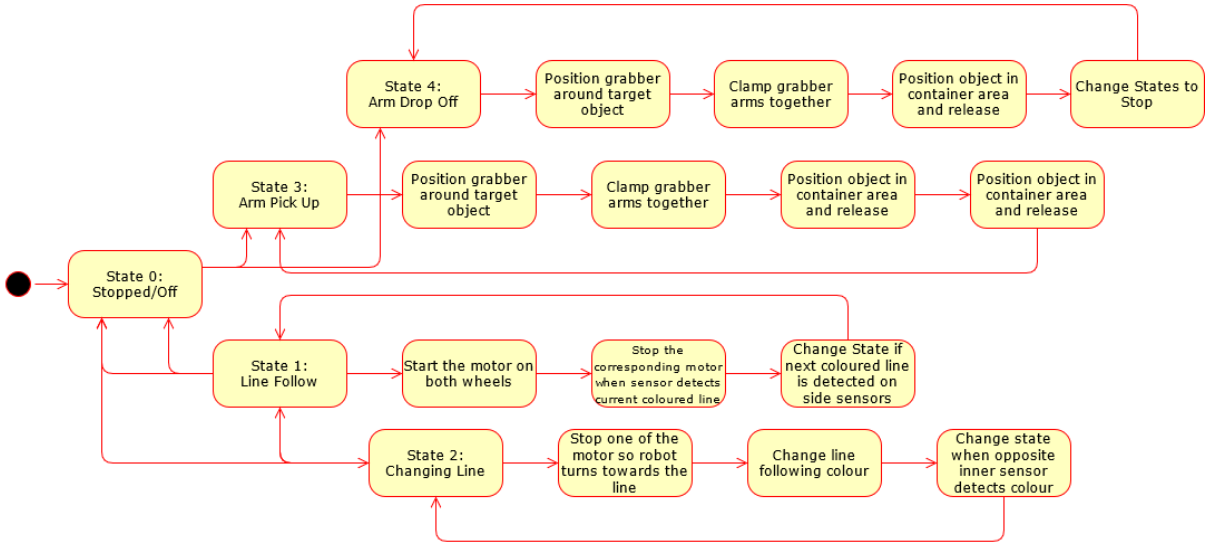
3.1 Trigger Type

The robot is designed to use event triggers. We chose to use events instead of time as triggers because there is a lot less room for error. If we had to implement a time trigger design, we would have to record the time the robot takes to travel so we can know when to program the turn. With event triggers, we do not have to achieve an infeasible amount of precision to function correctly. Most of the actions taken by the robot are triggered by the vision sensors detecting a change.

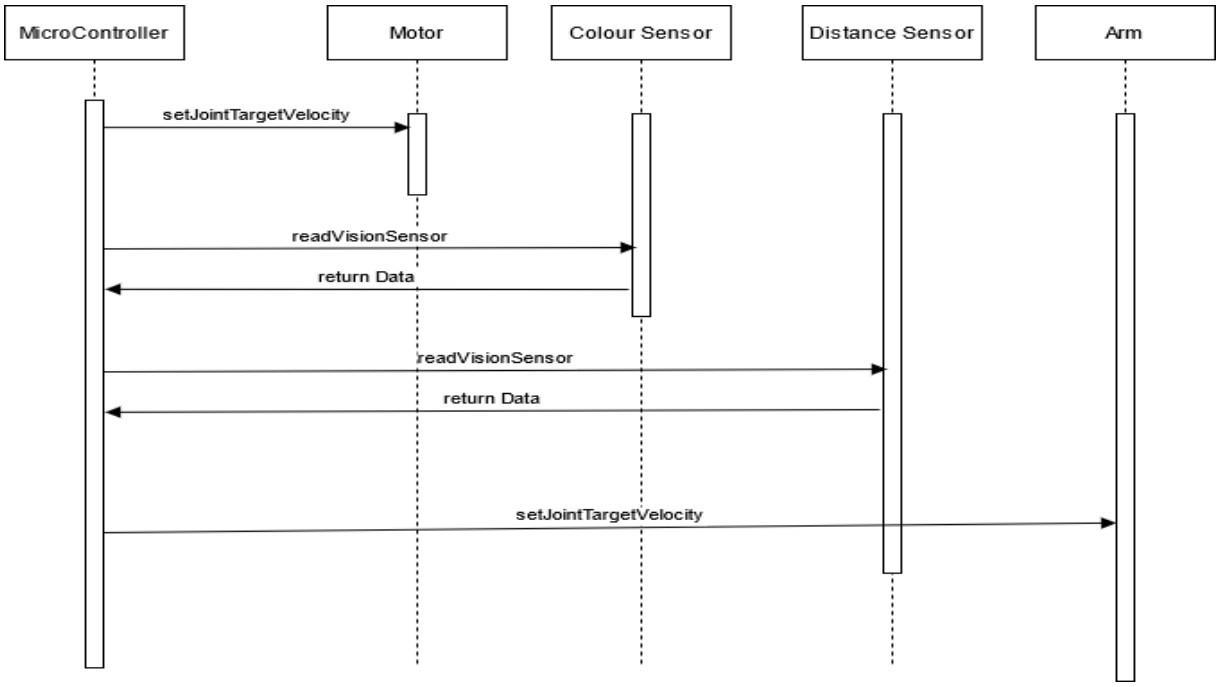
3.2 System Architecture



3.3 State Diagram



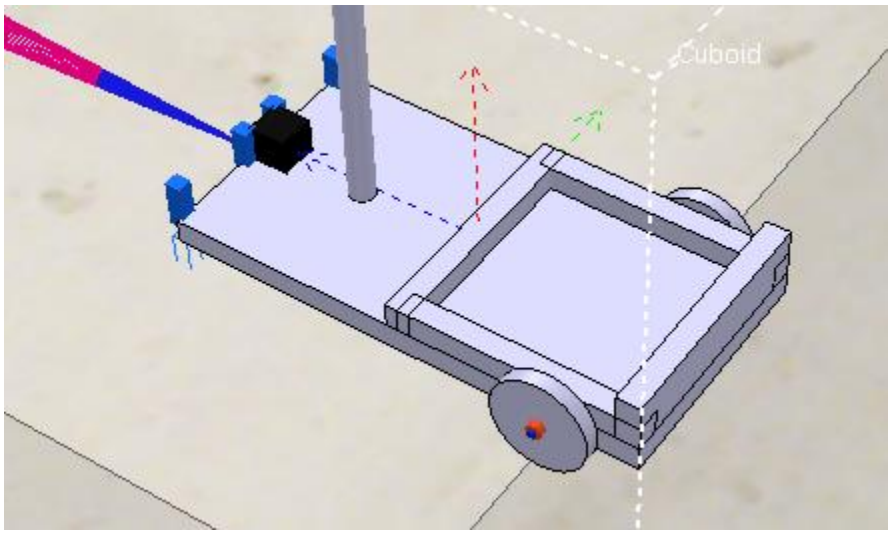
3.4 Sequence Diagram



4.0 Implementation

4.1 Robot Chassis – Dennis Liu, Colin McKay, Frank Xu

4.1.1 Picture of Robot Chassis



4.1.2 Chassis Design – Everyone

The robot uses 2 motorized wheels and one castor wheel to move. The robot also features a cargo area to hold packages and 4 coloured sensors to detect lines. The package sits in the back half of the robot leaving the front half for the robotic arm.

4.1.3 Colour Detection – Dennis Liu

The RGB values of coloured lines are converted into either 1, 2 or 3 depending on the intensity of the value and is stored in the robot program. The robot will then convert the RGB values from the colour sensors and try to match the RGB value from the sensor to the legend to determine the colour.

```
--Colour Legend
ground = {3, 3, 3}
black = {1, 1, 1}
red = {3, 1, 1}
blue = {1, 1, 3}
green = {1, 3, 1}
pink = {3, 1, 3}
yellow = {3, 3, 1}
turquoise = {1, 3, 3}
fuchsia = {3, 1, 2}
dgreen= {1, 2, 1}
test = {0, 0, 0}
```

4.1.4 Line Follow – Dennis Liu

The coloured line follow logic of the robot is similar to the logic of a regular line follow logic. The robot has two sensors set on either side of the robot with the line in between the sensor. Once the line the robot is trying to follow is detected by either of the sensor, the robot will turn to keep the line between the sensors. The colour of the line it is following is saved as a variable for by the robot.

4.1.5 Colour Swap – Dennis Liu

When the next coloured line is detected by the outside sensors, the current line colour will be swapped to the new colour and a turn will be initiated until the inner sensors detect the coloured line. Once the inner sensor detects the coloured line, the robot will switch back to line following mode. Logic has been implemented to then determine the next colour to look for.

4.1.6 Get Next Colour – Colin McKay

To determine what the next colour the outside sensor should look for, the robot will take the current colour it is following and the destination colour and output the next colour using if statements.

4.1.7 Proximity Sensor – Frank Xu

A proximity sensor is attached to the front of the robot to detect any obstacles that may appear in front of the path of the robot. As a safety feature, the robot will detect the obstacle, blink an LED to display that an obstacle is detected, and wait until the obstacle is removed before proceeding.

4.1.8 Turning around – Frank Xu

Once the robot has arrived at the end of a path and triggered the pick-up or drop-off sequence, the robot will initiate a turn around sequence. The robot first determines the destination colour by reading the next queued destination in an index. The robot will then turn the wheels in opposite directions so that the robot will turn around its center. The robot will stop the turn once the inner sensor detects the coloured line again and will initiate line follow logic.

4.1.9 Successive Deliveries – Colin McKay

After the robot has completed a delivery, it checks for the next package available for delivery from an array. It loads the next pickup location, and the next drop-off location. The robot then starts it's next delivery.

4.1.10 Remote API – Colin McKay

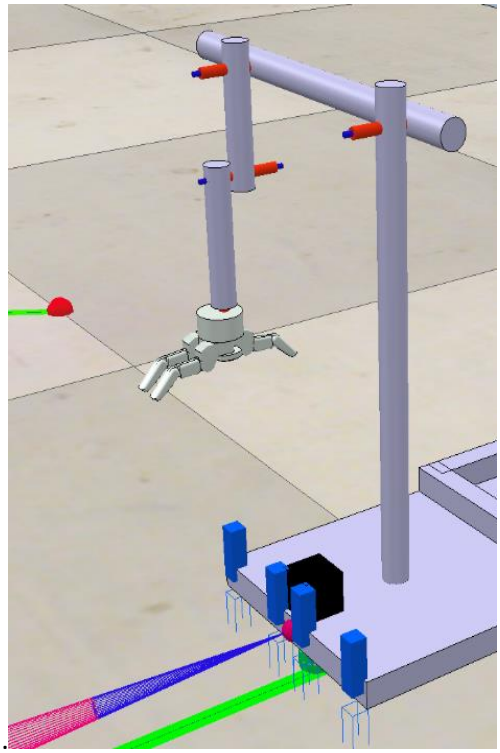
Originally, we had planned to use the remote API to control our robot. We developed a UI where the user could specify where the robot would drive to. This would have been extended to allow for a user to specify a package's pick up and drop off points. However, due to limitations on the speed of the remote API, the car needed to drive approximately 10x slower to correctly follow the lines compared to running Lua code natively. The remote API is working to specify a location to drive to but was abandoned close to the end of the project.

4.1.11 Testing and Debugging – Colin McKay

Extensively tested the robot in complex multi delivery scenarios. Ensured that the vehicle never strayed off the path far enough to disrupt package pickup. One problem encountered was with the obstacle simulator, where the car would crash occasionally. This was due to the proximity sensor being detected too early, combined with the vehicle slowly drifting while stopped.

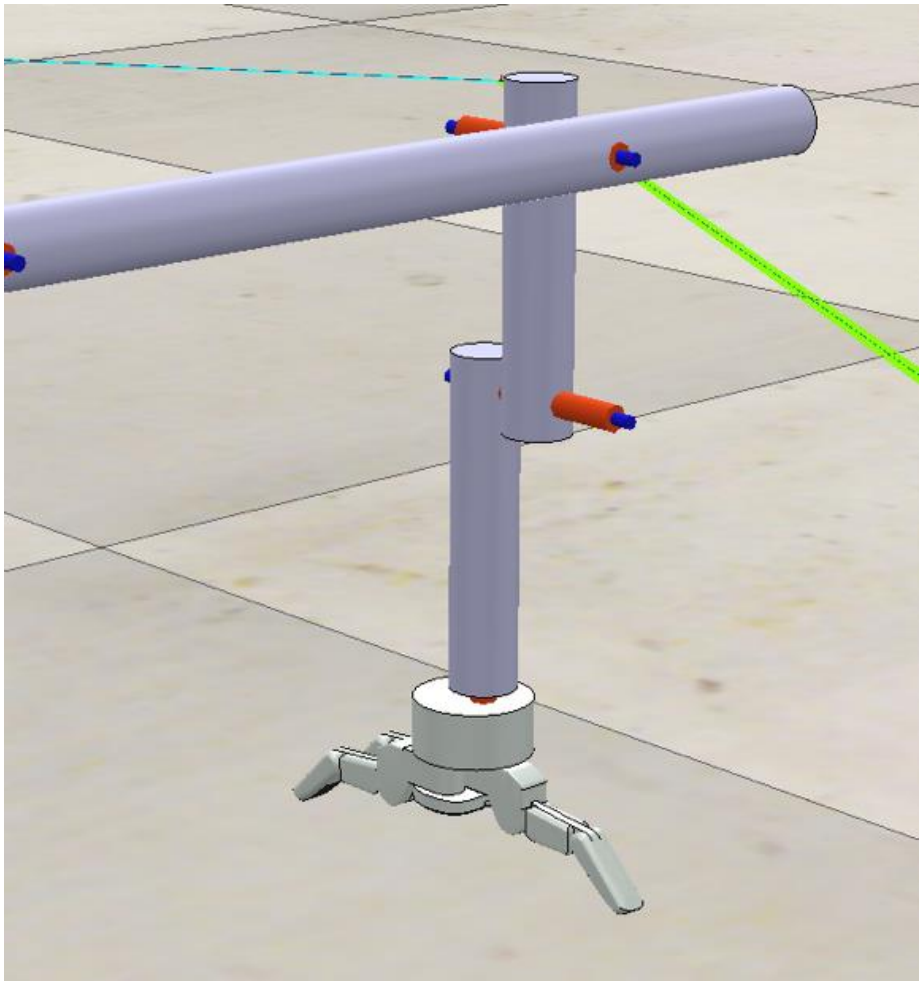
4.2 Robotic Arm – Yisheng Li

The arm is made up of three cylinders, joints which are the motors that make the arm to act and reach the specified position and a robotic hand that grabs packages.



The motors at the bottom of the center shaft of the arm allow the arm to rotate in a horizontal direction, the motors at the top of the center shaft allow the arm to swing up and down, and the motors on the outside of the arm allow the arm to bend outward or inward.

As the image below shows, the outermost part of the arm is made up of two cylinders and equipped with two motors. This is to give the arm a greater range of motion so that the hand can reach more positions.



As the screenshot below shows, the arm waits for the pick or drop command from robot and implement pick_a and drop_a method respectively.

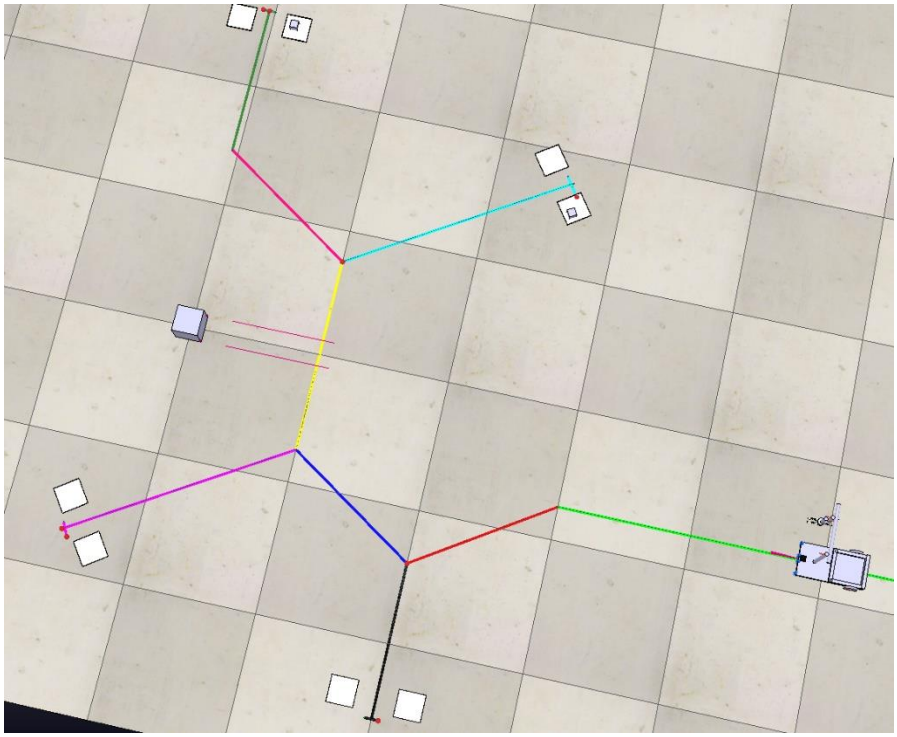
```
function sysCall_threadmain()
    baseMotor = sim.getObjectHandle("base_motor")
    armMotor1 = sim.getObjectHandle("arm_motor1")
    armMotor2 = sim.getObjectHandle("arm_motor2")
    armMotor3 = sim.getObjectHandle("arm_motor3")



    sim.setJointMaxForce(armMotor1, 1000)
    sim.setJointMaxForce(armMotor2, 1000)
    sim.setJointMaxForce(armMotor3, 1000)
    sim.setIntegerSignal("close",0)


    sim.waitForSignal("pick")
    sim.clearIntegerSignal("pick")
    pick_a()

    sim.setIntegerSignal("turn_around", 1)

    sim.waitForSignal("drop")
    sim.clearIntegerSignal("drop")
    drop_a()

    sim.setIntegerSignal("picked", 0)


end
```

The screenshot below are the details of pick_a methods. The way that makes the arm to pick up the package is setting specific positions for those four joints (motors) to reach the package, set close signal to hand to grab the package, setting specific positions for joints to move to the cargo area and clean close signal of hand to put the package in the cargo space. The implementation of drop_a method is similar.

```
function pick_a()
    sim.wait(5)
   -- down close
    x = 337
    sim.setJointTargetPosition(armMotor1, 2*math.pi*(x/360))
    sim.setJointTargetPosition(armMotor2, 2*math.pi*((360-x)/360))
    sim.wait(10)
    sim.setIntegerSignal("close",1)
    sim.wait(5)


    -- up turn
    sim.setJointTargetPosition(armMotor1, 0)
    sim.setJointTargetPosition(armMotor2, 0)
    sim.wait(5)
    sim.setJointTargetPosition(baseMotor, 2*math.pi*(270/360))


    -- down open
    sim.wait(10)
    x = 350
    sim.setJointTargetPosition(armMotor1, 2*math.pi*(x/360))
    sim.setJointTargetPosition(armMotor2, 2*math.pi*((360-x)/360))
    sim.wait(10)
    sim.setIntegerSignal("close",0)


    -- up turn back
    sim.wait(5)
    sim.setJointTargetPosition(armMotor1, 0)
    sim.setJointTargetPosition(armMotor2, 0)
    sim.setJointTargetPosition(baseMotor, 0)
    sim.wait(5)

end
```

4.3 Map – Frank Xu

4.3.1 Picture of Map



4.3.2 Coloured Paths and Packages – Frank Xu

The map features 10 different coloured paths for the robot to follow with protrusions at the end to determine a stop. Package stations are placed beside path ends for the robot to pick-up and drop-off.

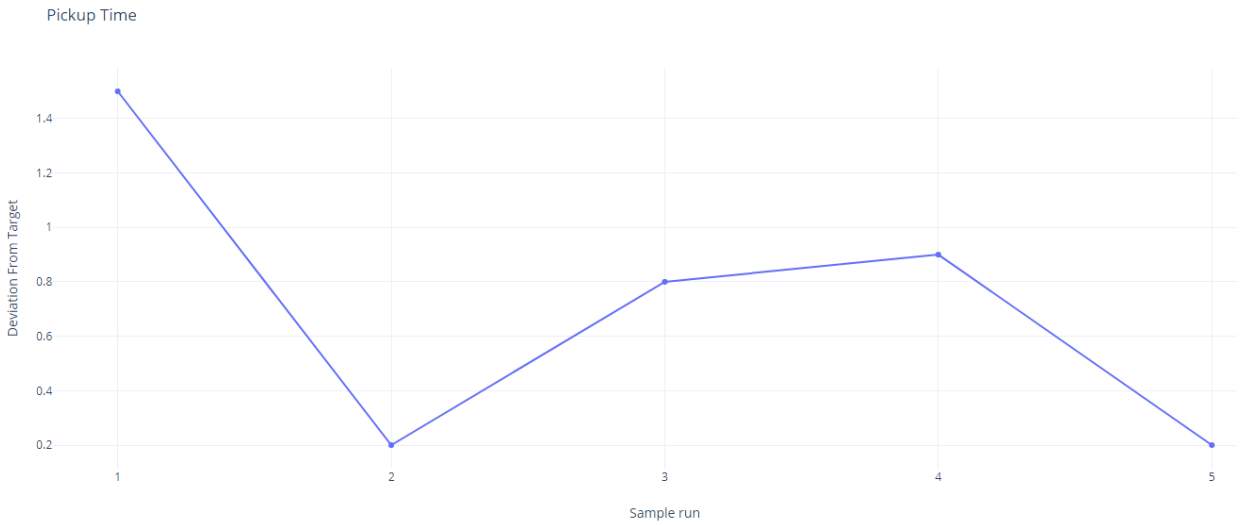
4.3.3 Obstacle Simulator – Frank Xu

A big cube will periodically move onto and away from the path to simulate an obstruction in the path. The robot will have to stop and wait for the obstruction to clear before continuing. The cube however will not appear while the robot is occupying the space where the cube would appear.

5.0 Control Charts

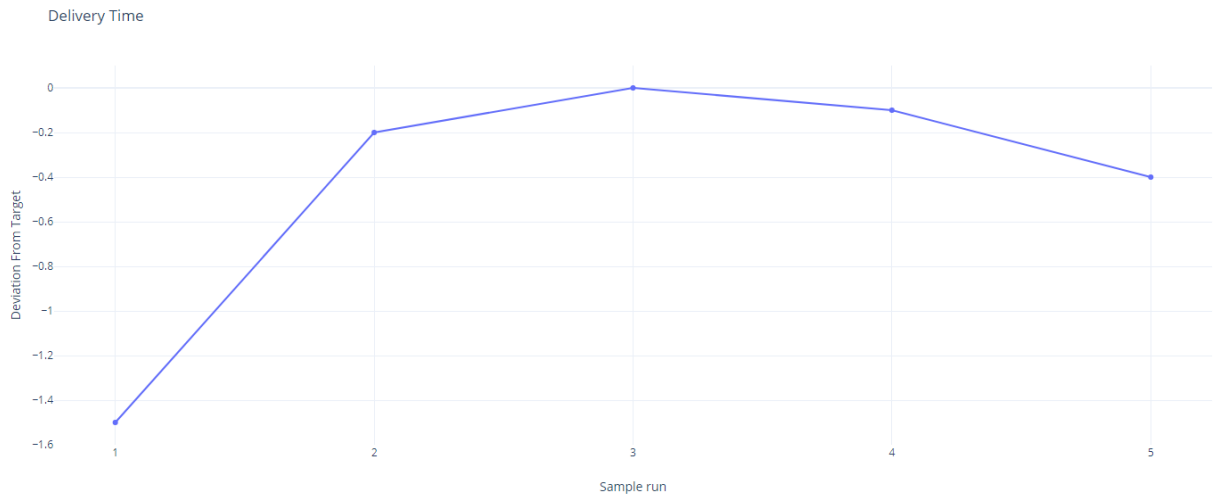
5.1 Pickup time

Target time: 20 seconds



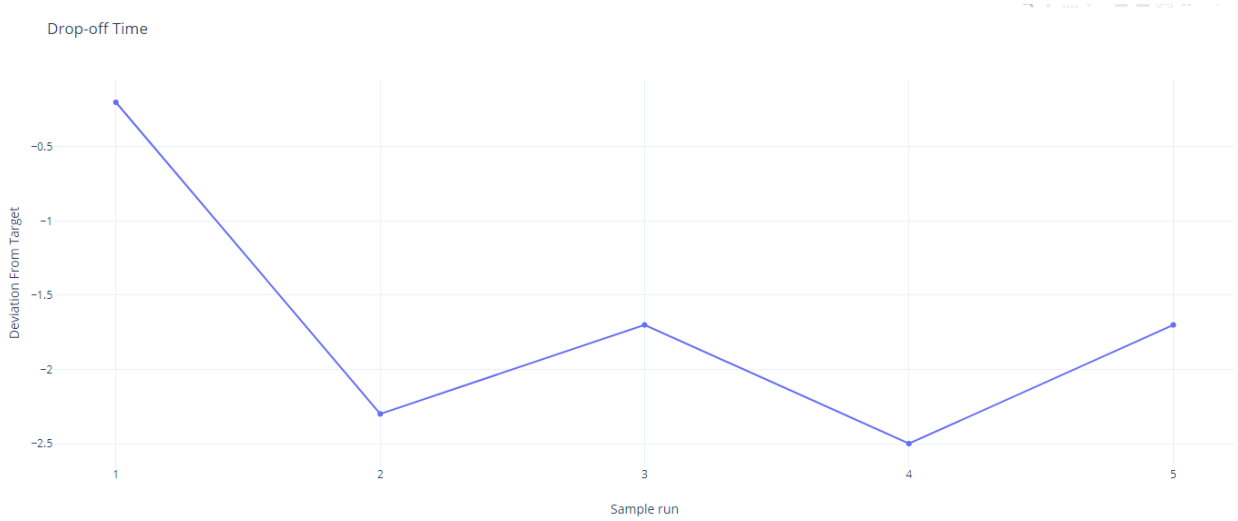
5.2 Delivery Time

Target time: 25 seconds



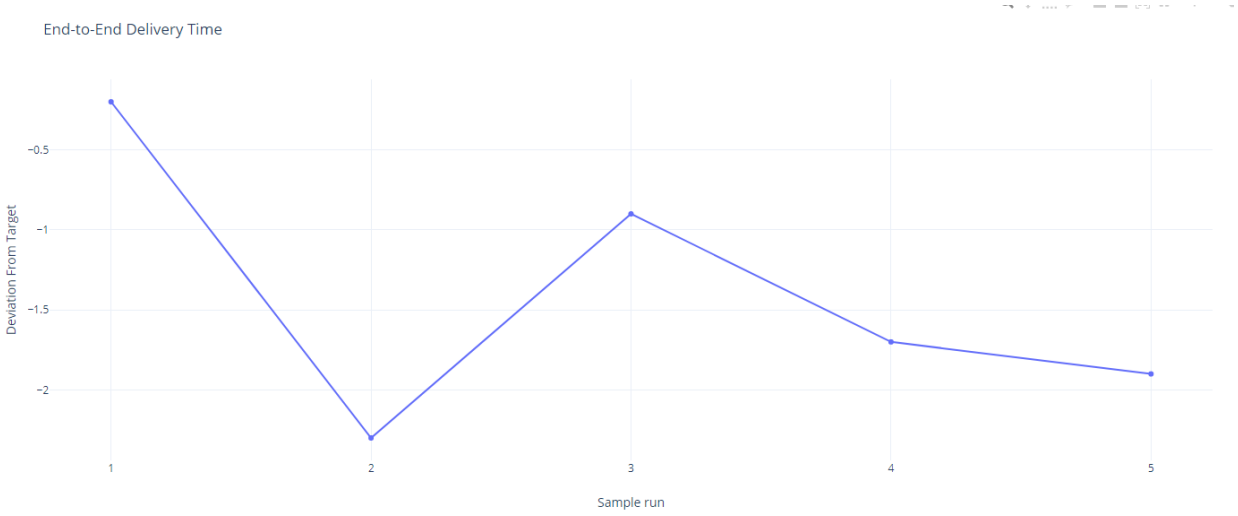
5.3 Drop-off Time

Target time: 20 seconds



5.4 End-to-End Delivery Time

Target time: 65 seconds



6.0 Project management

6.1 Schedule Network Diagram

6.2 Gantt Chart

Tasks		Week of lab					
		4	5	6	7	8	9
Component Development	Test environment development						
	Visual sensors and movement programming						
	Arms programming						
	Distance sensors programming						
Component Test and Review	Visual sensors and movement test and review						
	Arms test and review						
	Distance sensors test and review						
	Status LED test and review						
	Input device test and review						
Integration	All components Integration						
	Robot test and review						
Final report							

6.3 Team members contribution

Tasks		Frank Xu	Dennis Liu	Yisheng Li	Colin McKay
Component Development	Test environment development	R	I	A	I
	Visual sensors and movement programming	I	R	I	A
	Arm programming	A	I	R	I
	Distance sensors programming	R	I	A	I
Component Test and Review	Visual sensors and movement test and review	I	R	I	A
	Arms test and review	A	I	R	I
	Distance sensors test and review	I	A	I	R
	Status LED test and review	I	A	I	R
	Input device test and review	I	R	I	A
Integration	All components Integration	R	I	A	I
	Robot test and review	I	A	I	R
Final report		R	R	R	R

* R - Responsible, A - Approver, I - Informed