ROB-GY 6103: Advanced Mechatronics
Integrated Project Report
Team 25

KaiYuan Ma
Ifedayo Olusanya
Justin Ramlall

Table Of Contents

Introduction:

The purpose of this project is to show understanding of making a robot that uses two microcontrollers integrated together. To do this, we are using the Raspberry Pi 3B in tandem with the Parallax Propeller activity board. The goal of our robot is to help identify defective and non-defective widgets on a warehouse floor in order to help management. Using directional aids in the form of triangles and markers in the form of ArUco tags, our robot must be able to navigate the warehouse and follow directions to identify these markers. In order to complete the objectives that have been outlined for us, we will be using a variety of components. These include the two previously mentioned microcontrollers, an infrared sensor array, two ultrasonic sensors, continuous servos, a Raspberry Pi (RPi) camera, two light emitting diodes (LEDs), and a liquid crystal display (LCD). A complete circuit diagram will be included in this report detailing the electrical connections as well as a flowchart and commented code that detail the control logic. Below is a figure of the map that the robot will traverse in testing.

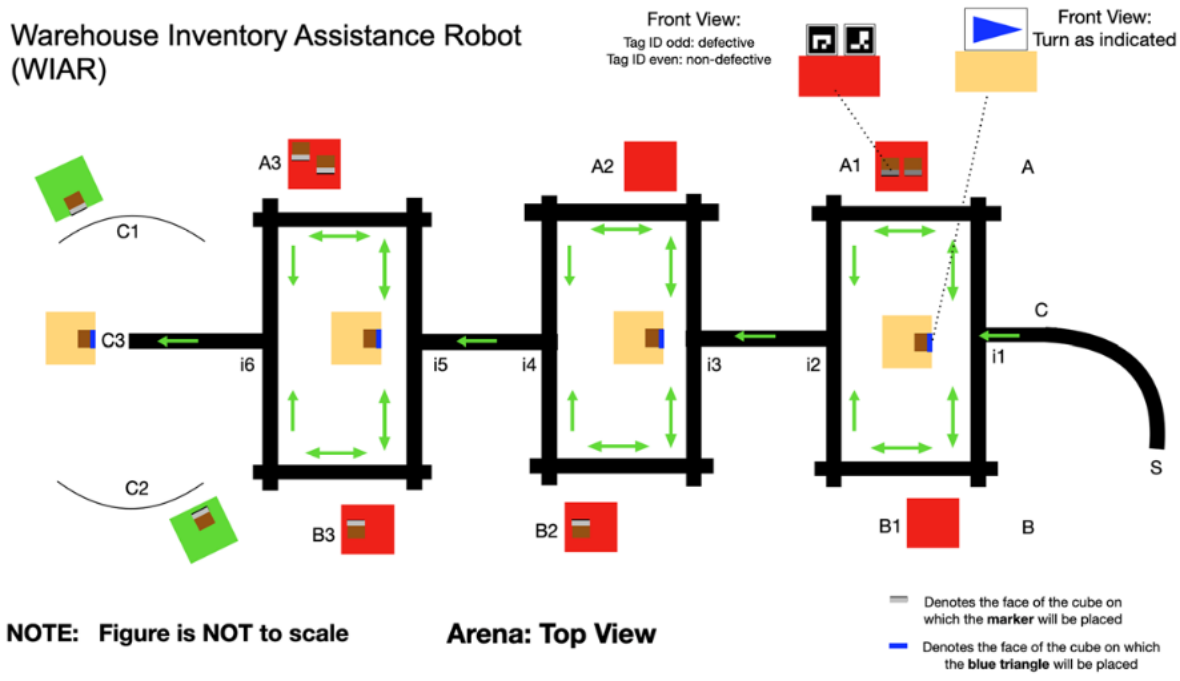

Figure 1: A map of the test area for the robot to traverse. The robot starts at position S and will traverse through the squares following directional aids. Once it reaches the end, the robot goes to a widget station and identifies how many widgets it has counted. Image courtesy of the ROB-GY 6103 instructional team.

Objectives:

A list of objectives for our robot to complete have been created in order for us to ensure functionality of the robot. By completing these objectives, we will know that the robot can function exactly as desired. This list is also used in order to determine the hardware that will be used to create the robot. The objectives are as follow:

1. The robot must line follow along the designated path
2. The robot must be able to detect and indicate when it reaches intersections i1 through i6
3. When the robot reaches a directional aid, it must turn in the indicated direction
4. When the robot reaches a widget station, it must detect the widgets by their markers
5. When the robot detects the markers, it must indicate the number of the defective and non-defective widgets at each station
6. The robot must reach the docking station by following the tag at the end of the course
7. Finally, the robot must display the total number of defective and non-defective widgets

Based on these objectives, we have decided to use the chassis and servos used in the Boe-Bot Kit from Parallax along with an infrared sensor array, two ultrasonic sensors, a LCD screen, two LEDs, and a RPi camera. Each of these objectives have been demonstrated as successful by the robot in testing.

System Design:

The system consists of a mechanical system, electrical system, RPi code and Propeller code integrated together to complete the objectives. The mechanical design will detail the components that were chosen for the robot. The electrical design will detail the electrical connections for all of the components, the microcontrollers, and the power supply. The RPi code and the Propeller code will be included in Appendix A with comments and a flowchart detailing the logic will be included in the integrated code section.

Mechanical Design

For the base of our robot, we will start with the two Parallax Continuous Servos (Figure 2) and the wheel attachments along with the chassis provided in the Parallax Boe-Bot kit. Our

two microcontrollers include the Raspberry Pi 3 Model B from Raspberry Pi (Figure 3) and the Parallax Propeller Activity Board WX (Figure 4). These two will work together to control the rest of the components. From Raspberry Pi, we will also be using the Raspberry Pi Camera Module 3 (Figure 5) as the eyes of the robot. From Parallax, we are using the Parallax Ping))) Sensor (Figure 6) along with the Adafruit Industries LLC 3942 Ultrasonic Sensor (Figure 7). In order to line follow, the Pololu QTR-8RC Reflectance Sensor Array (Figure 8) is used.When we need to indicate, we will use a Parallax 2x16 Serial LCD with Piezo Speaker (Figure 9) with two generic LEDs. To power the RPi and connected components, we are using a generic 5000 mAh rechargeable battery bank, and for the Propeller board and its connected components, we will be using two 9V batteries connected in series. The components are all mounted to the chassis, either directly or using custom laser cut acrylic pieces.

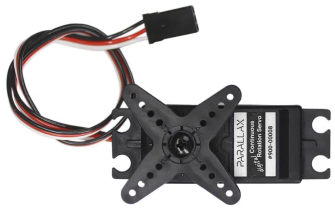

Figure 2: The continuous servo from Parallax. It is used to add motion to the robot. Two of them are used with attached wheels.

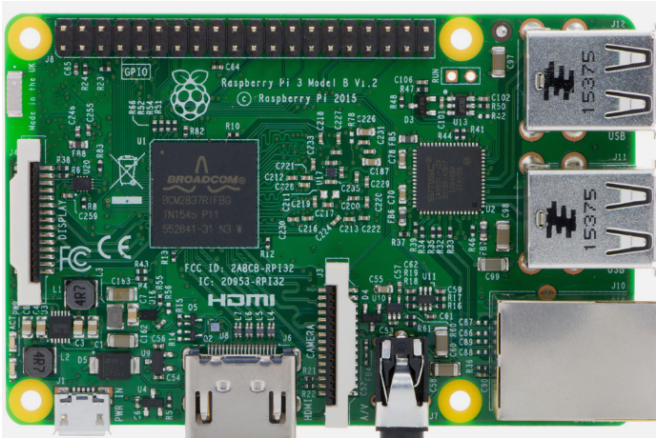

Figure 3: The Raspberry Pi 3 Model B. This controls some of our components with the important function of implementing computer vision. With the RPi camera we can interpret ID numbers from the ArUco tags as well as the direction from the directional aid.

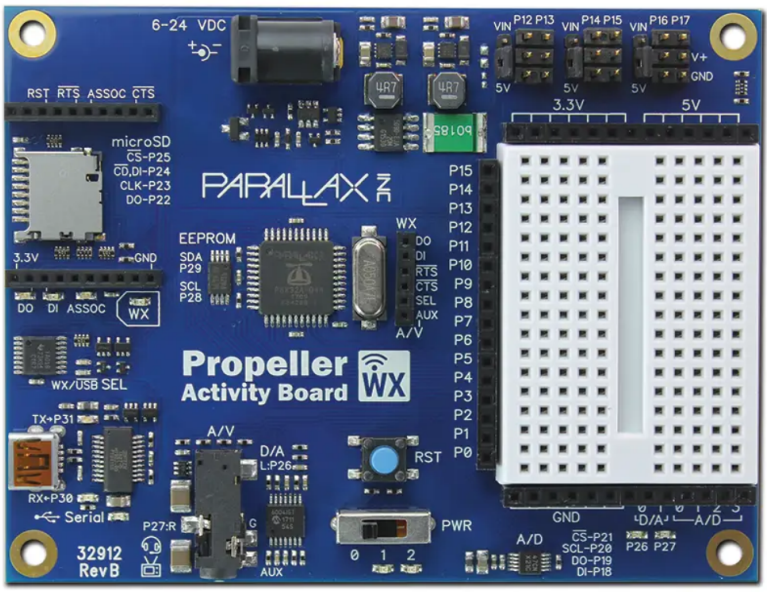

Figure 4: The Propeller Activity Board WX. This microcontroller controls most of our components. It has an attached breadboard for smaller circuits and 15 digital pins. It also has header pins to power our motors. It is powered by 18V DC.

Figure 5: The Raspberry Pi Camera Module 3. The “eyes” of the robot. Image capture input is taken from this module and interpreted to understand the information given by the ArUco tags and the directional aid.

Figure 6: The PING))) sensor from Parallax. It has 3 pins for 5V power supply, Ground, and SIG. SIG gets connected to a digital pin for sending and receiving the ultrasonic signal.

Figure 7: The Adafruit Industries LLC 3942 Ultrasonic Sensor. It has 4 pins for VCC power, Trigger, Echo, and Ground. Connecting Trigger and Echo with a 220 Ohm resistor allows this to function just like the PING))) sensor.

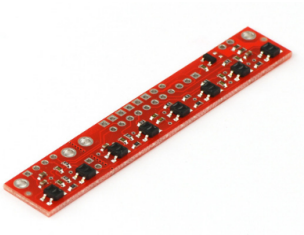

Figure 8: The Reflectance Sensor Array from Pololu. It has 8 Infrared Sensors. There is a pin for 5V power supply and Ground, and 8 pins to send and receive data for each infrared sensor.

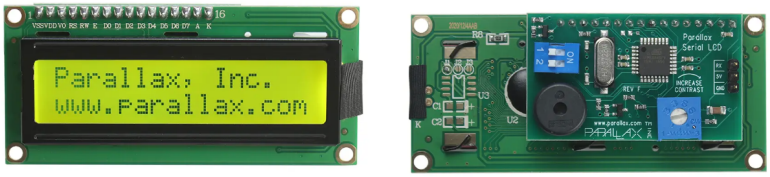

Figure 9: The Serial LCD from Parallax with two rows. The rear header board allows us to display text using three pins: 5V power supply, Ground, and RX for receiving serial data.

Electrical Design

To understand all of the electrical connections that were made, the below circuit diagram was created (Figure 10). The RPi is powered with the 5000 mAh battery bank, and is in charge of controlling the RPi camera, the two LED's, the LCD, and has four pins for communicating with the Propeller board. The Propeller board also uses four pins to communicate with the RPi, and controls both ultrasonic sensors, the infrared array, and the servo motors. This is powered by two 9V batteries in series, giving a voltage of 18V. The choice for two batteries was to improve performance issues that we noticed when we only had 9V supply to the board. In order for

proper communication between the microcontrollers, a common ground is used for all the components. The RPi camera connects to the RPi via the CSI Camera Port and is mounted with acrylic to the chassis. Each LED from the RPi is connected in series with a 220 Ohm resistor and then to ground on the Propeller. This utilizes the breadboard on the Propeller board. The LCD screen is connected to the TX pin on the RPi as well as ground and the 5V pin of the RPi. GPIO Pins 11, 13, 15, and 16 on the RPi connect to digital pins 12, 13, 15, 14 on the Propeller board, respectively. The PING))) sensor is connected to digital pin 10 on the Propeller and is also the only component that is supplied with 3.3V. The adafruit ultrasonic sensor connects the Trigger and Echo pins together with a 220 Ohm resistor and is connected to digital pin 10 on the Propeller board. The Pololu infrared array uses digital pins 0 through 7 and the left and right servo motors are connected to digital pins 16 and 17 respectively. The adafruit ultrasonic sensor, the Pololu infrared array, and the two servo motors are all supplied with 5V.

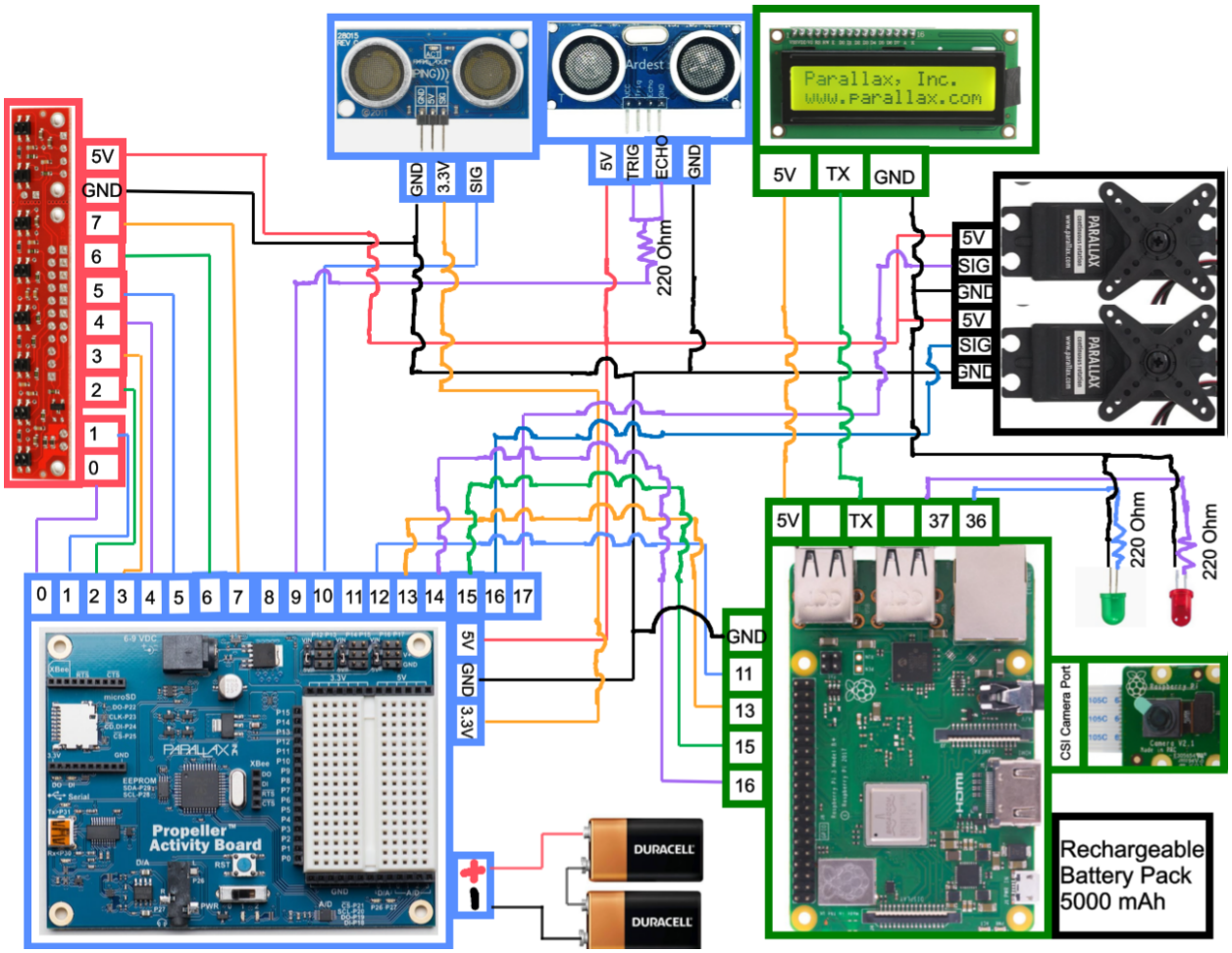

Figure 10: The complete circuit diagram for our robot. All electrical connections between the two microcontrollers and their components are detailed. The resistors used for the adafruit ultrasonic sensor and the two LED's are labeled as 220 Ohms. The servos are powered directly by the Propeller board using the motor header pins. A common ground is used for all components

Integrated Code

Included below is the flowchart describing the control logic for the integrated code (Figure 11). The code is split between the RPi and the Propeller board. They share inputs and outputs between each other in order to trigger specific events across the microcontrollers. The RPi code mainly handles the computer vision to detect the directional aids and the ArUco tags, as well as the LED and LCD indications and displays. The propeller mainly handles all of the sensor inputs from the ultrasonic sensors and the infrared array, as well as controls the servo motors to navigate the test area.

The code starts off with the calibration of the infrared array and the setup of the main loops for both microcontrollers. We then enter our line following code, which we return to frequently as the robot navigates the course. Now the first thing that we check for is if we reach an intersection and there is an object in front of us. This tells us that we are at intersections i1, i3, or i5. Each of these start the rectangular sections of the test area, and the movement and computer vision codes work the same for each. The camera takes an image of the directional aid and determines if it points left or right. It sends the direction to the Propeller board so that the robot may either turn left or right. Once the robot turns, the robot line follows until we reach one of the widget stations. Here, the robot takes an image of the ArUco tags, and determines if it is defective or not. It then outputs the defective and non-defective counts on the LCD as well as with the LED's. The robot then backtracks and moves to the other side of the rectangle to the next station. Depending on the directional aid provided, the robot is now either moving from side A to B or side B to A. It again does the same thing at this widget station. Once it is done, it moves forward to the next section. Once it has moved past the three rectangular areas, we reach intersection i6, the start of No Man's Land. Once the robot line follows from i6 to point C3, it will eventually detect the directional aid in front of it with no more black tape underneath. Here, the directional aid tells it which way to turn. The robot then turns in that direction, using the camera to scan for the final ArUco tag at either C1 or C2. Once it finds it, it drives straight forward until it reaches the station, and stops. It then displays the final count for defective and non-defective widgets.

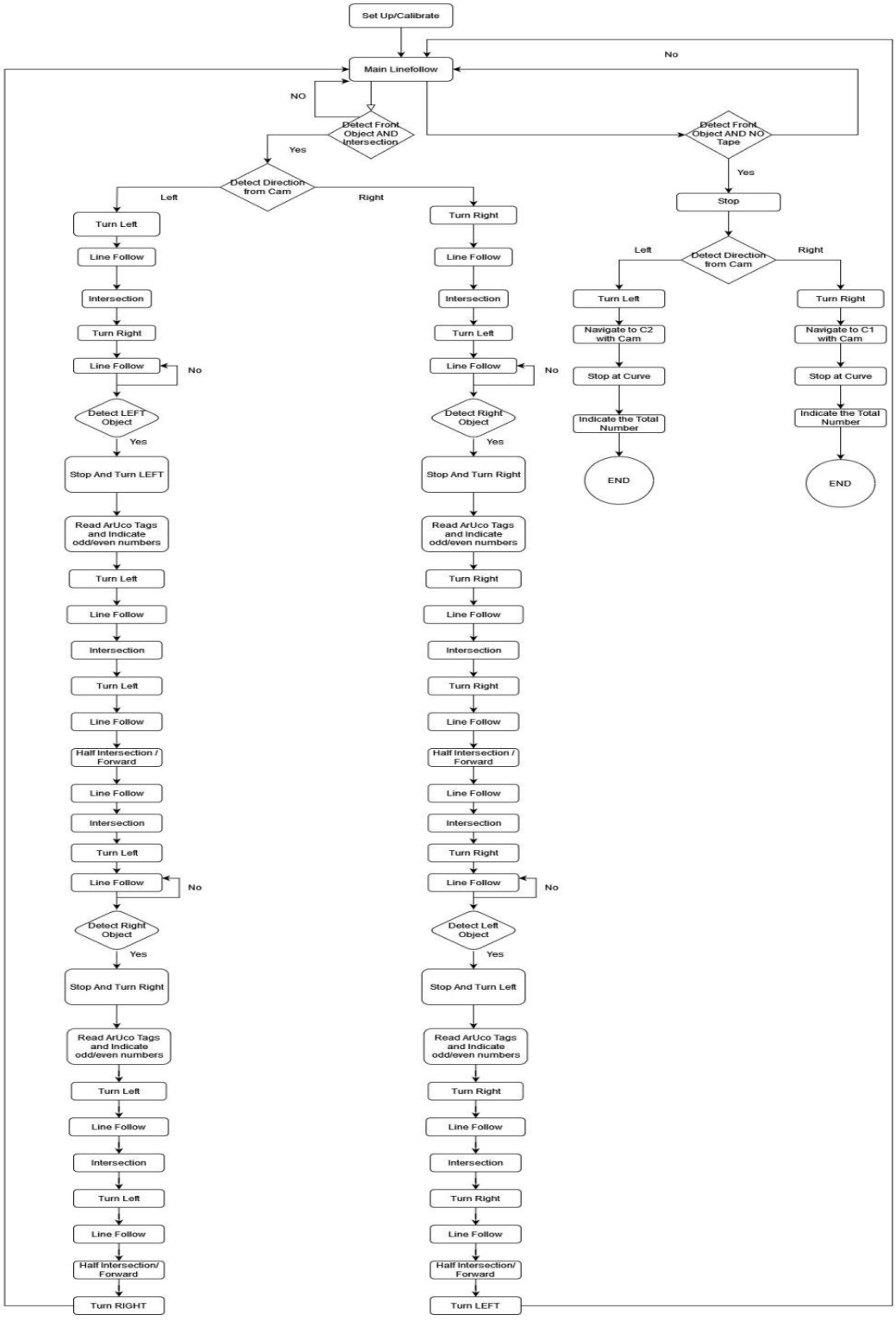

Figure 11: This flowchart shows the behavior of the robot as it navigates the test area. The main inputs for its actions are determined by the computer vision, the infrared sensors, and the ultrasonic sensors. The main behavior occurs in the first three rectangular sections and the final behavior is for No Man's Land.

System Testing:

An incremental approach was taken in order to test the functionality of all the systems together. First, we assembled the robot using the chassis, servo motors, RPi, Propeller board, and infrared array. We then began with the basic line following code. Once the robot was able to follow the lines completely, we then moved to implement the ultrasonic sensors. At first, three sensors were used, one in the front and one on each side. Once the mounting hardware was created and the ultrasonic sensors were fixed in place, We could begin testing these. The front ultrasonic sensor was later disconnected because the RPi camera could serve the same function. The ultrasonic sensors were also calibrated to only send and receive signals from up to 10cm away, since this is how far the widget stations on sides A and B would be. Once all of these were working together to a certain degree, the RPi camera module was mounted and the code for this could start to be developed. Preliminary testing of the RPi camera and RPi board was done to ensure that the directional aids and the ArUco tags could be properly identified. Once it was on the robot, we tested the robot moving around the course while making sure that the camera was detecting the aids and markers. Next came one of the most important parts of the testing process. The RPi board and Propeller board were connected electrically and the codes were adjusted to send and receive signals across each other. An example of how to test this with the other components, the code was adjusted so that when the infrared array detects an intersection and the RPi camera detects the directional aid, the RPi sends an output to the Propeller board so that it can turn the robot. At this point we also wanted to ensure that when the robot reached point C3 on the map, it was able to navigate to the final station with just the camera. A rough framework for this showed basic functionality and would later be refined. The codes were both built upon to include all other necessary communications based on our logic flowchart. After more testing to achieve a basic level of the objectives we could, we finally added on the LCD and LEDs to the robot. Now that it was fully assembled, we finished our testing with fine tuning of certain sensor values and adjusted the RPi camera for the right lighting conditions, and did a final test. In this final test, the robot performed all of the objectives with no problems.

Conclusion:

This project was intended to show our understanding of the use of computer vision with a Raspberry Pi as well as integrate it with another microcontroller, the Parallax Propeller Board, in order to design a robot that would navigate the testing area shown in Figure 1. After understanding all of the objectives that were laid out for our robot, we created a flowchart to detail the logic of the code as well as began our mechanical design using the components shown in Figures 2 through 9. Once all of the components were connected as described in Figure 10, we were able to iterate and test the functionality of these components with the code for the microcontrollers and ensure proper functionality of the robot. After testing, our team can confidently conclude that our robot has a robust design in its integrated system that allows us to perform all of the required tasks.

Appendix:

Python code (on PI)

```
import cv2       #import cv2 library
import numpy as np    #import numpy
import time      #import timer module
import RPi.GPIO as GPIO      #Import pi GPIO controls
from cv2 import aruco            #import aruco library
import serial    #import usart module

serial_port ='/dev/serial0'       #port to use for serial communication
baud_rate = 9600       # set baud rate to 9600
ser = serial.Serial(serial_port,baud_rate, timeout=1)           #attach serial


GPIO.setmode(GPIO.BOARD)                    #gpio pin type
GPIO.setwarnings(False)


defective_lf=15          #pin for left and defective signaling with propeller
nondefective_rt= 13    #pin for right and non defective signaling with propeller
inaruco=16      #pin for aruco detection
indir = 11        #pin for triangle detection
noman_aruco = 22     #no man's land aruco detection pin
arucofollow = 18        #pin for no man's land line follow
red_led = 37             #indicate defective
green_led =36           #indicate nondefective

tot_def =0       #count total defective
tot_nondef =0  #count total defective

#set all pins
```

```
GPIO.setup(defective_lf,GPIO.OUT)
GPIO.setup(nondefective_rt,GPIO.OUT)
GPIO.setup(arucofollow,GPIO.OUT)

#send all output pins low
GPIO.setup(noman_aruco,GPIO.IN)
GPIO.setup(inaruco,GPIO.IN)
GPIO.setup(indir,GPIO.IN)



#leds
GPIO.setup(red_led,GPIO.OUT)
GPIO.setup(green_led,GPIO.OUT)

#lcd function for displaying text and numbers
def send_command(command):
    ser.write(command.encode())
def send_command2(command):
    ser.write(str(command).encode('utf-8'))



def nothing(x):
    pass

#function to detect triangle
def get_direction(frame):
    direction = 0  # left =1, right =2
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    l_b = np.array([100, 51, 51])
    u_b = np.array([130, 255, 255])
    mask = cv2.inRange(hsv, l_b, u_b)
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # apply thresholding to convert the HSV image to a binary image
```

```
    ret,thresh = cv2.threshold(mask,150,255,0)

    # find the contours
    contours,hierarchy = cv2.findContours(thresh ,
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        x,y,w,h = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt)<100:
            continue
        approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt, True), True)
        #print (approx)
        if len(approx) == 3:
            res  = cv2.drawContours(res, [cnt], -1, (0,255,255), 3)
            # compute the center of mass of the triangle
            M = cv2.moments(cnt)
            if M['m00'] != 0.0:
                x = int(M['m10']/M['m00'])
                y = int(M['m01']/M['m00'])
            cv2.putText(res, 'Triangle', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 0), 2)

            if w > h:
                if approx[0][0][0] < x:
                    direction =2
                else:
                    direction = 1
            else:
                continue
    return direction

#function to detect aruco tag
def detect_aruco(frame):
    defective = 0
    nondefective = 0
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

```
    key=getattr(aruco,f'DICT_{markerSize}X{markerSize}_{totalMarkers}')
    arucoDict = aruco.Dictionary_get(key)
    corners,ids,rejected=aruco.detectMarkers(gray,arucoDict)
    frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
    #cv2.imshow('frame_marker', frame_markers)

    if ids is not None:
        for i in range(len(ids)):
            if ids[i][0]%2 ==1:
                defective +=1
                GPIO.output(red_led,GPIO.HIGH)
                time.sleep(0.3)
                GPIO.output(red_led,GPIO.LOW)
                time.sleep(0.3)

            else:
                nondefective +=1
                GPIO.output(green_led,GPIO.HIGH)
                time.sleep(0.3)
                GPIO.output(green_led,GPIO.LOW)
                time.sleep(0.3)
    else:
        'Nothing Here'

    return defective,nondefective,ids


#Function to detect aruco in no man's land
def noman_aruco(frame):
    defective = 0
    nondefective = 0
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{markerSize}X{markerSize}_{totalMarkers}')
    arucoDict = aruco.Dictionary_get(key)
```

```
    corners,ids,rejected=aruco.detectMarkers(gray,arucoDict)
    #cv2.imshow('frame_marker', frame_markers)

    if ids is not None:
        for i in range(len(ids)):
            if ids[i][0]%2 ==1:
                GPIO.output(red_led,GPIO.HIGH)
                time.sleep(0.3)
                GPIO.output(red_led,GPIO.LOW)

            else:
                nondefective +=1
                GPIO.output(green_led,GPIO.HIGH)
                time.sleep(0.3)
                GPIO.output(green_led,GPIO.LOW)
    else:
        'Nothing Here'
    return ids

#function for no man's land drve
def aruco_drive(frame):
    defective = 0
    nondefective = 0
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{markerSize}X{markerSize}_{totalMarkers}')
    arucoDict = aruco.Dictionary_get(key)
    corners,ids,rejected=aruco.detectMarkers(gray,arucoDict)
    frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)

    if ids is not None:
        corner_mid_point = corners[0][0][1][0] + (corners[0][0][0][0] - corners[0][0][1][0])/2

    else:
        corner_mid_point = 0
```

```
    return corner_mid_point

#set up pi cam
cap = cv2.VideoCapture(0);
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 480)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
markerSize=6
totalMarkers=250

GPIO.output(defective_lf,GPIO.LOW)
GPIO.output(nondefective_rt,GPIO.LOW)
GPIO.output(arucofollow,GPIO.LOW)

while(True):
#read all input pins waiting for signal from properller
    check_direction = GPIO.input(11)
    check_aruco = GPIO.input(16)
    check_noman_aruco = GPIO.input(22)

# if only direction pin is set high read triangle direction
    if(check_direction == 1 and check_aruco ==0):
        for i in range(50):
            _, frame = cap.read()
            a = get_direction(frame)
            if a >0:
                if a ==  1:
                    GPIO.output(defective_lf,GPIO.HIGH)
                    print(a)
                    time.sleep(1)
                    break
                elif a == 2:
                    GPIO.output(nondefective_rt,GPIO.HIGH)
                    print(a)
                    time.sleep(1)
```

```
                break
            else:
                continue
            print(a)
            time.sleep(0.3)
        GPIO.output(defective_lf,GPIO.LOW)
        GPIO.output(nondefective_rt,GPIO.LOW)

    , #if only aruco pin is set high read tags
     elif(check_aruco ==1 and check_direction == 0):
        ret,frame = cap.read()
        def_count,nondef_count,identity = detect_aruco(frame)
        tot_def +=def_count
        tot_nondef += nondef_count
        print(identity)
        send_command(chr(12))
        message = "def = "
        message2 = "nondef = "
        send_command(message)
        send_command2(tot_def)

        send_command(chr(13))

        send_command(message2)
        send_command2(tot_nondef)

#if no man's land pin is high search for aruco
    elif(check_noman_aruco ==1):
        ret,frame = cap.read()
        identity = noman_aruco(frame)

        if identity is not None:
            GPIO.output(arucofollow,GPIO.HIGH)
```

```
        else:
            GPIO.output(arucofollow,GPIO.LOW)
            continue


    #read both aruco and direction at no man's land
    elif(check_direction ==1 and check_aruco ==1):
        ret,frame = cap.read()
        a = aruco_drive(frame)
        print(a)
        if(a == 0):
            GPIO.output(nondefective_rt,GPIO.LOW)
            GPIO.output(defective_lf,GPIO.LOW)
        if(a >0 and a <150):
            GPIO.output(nondefective_rt,GPIO.LOW)
            GPIO.output(defective_lf,GPIO.HIGH)
            time.sleep(0.3)
            GPIO.output(defective_lf,GPIO.LOW)
        elif(a>350):
            GPIO.output(defective_lf,GPIO.LOW)
            GPIO.output(nondefective_rt,GPIO.HIGH)
            time.sleep(0.3)
            GPIO.output(nondefective_rt,GPIO.LOW)
        else:
            GPIO.output(defective_lf,GPIO.LOW)
            GPIO.output(nondefective_rt,GPIO.LOW)


    else:
        continue


cap.release()
cv2.destroyAllWindows()
```

Parallax code

```
#include "simpletools.h" // Include simpletools
#include "servo.h" // Include servo library
#include "ping.h" // Include Ultrasonic Sensor
void forward(void);
void reverse_(void); //forward declaration of function to move robot forward at fastest speed
void stopmotors(void); //forward declaration of function to stop robot
void turnright(void); //forward declaration of function to turn robot
void turnleft(void); //forward declaration of function to turn robot left
void turnright_lf(void);
void turnleft_lf(void);
void spinleft(void); //forward declaration of function to spin robot left
void spinright(void); //forward declaration of function to spin robot right
void linefollow(void);//forward declaration of function to enable robot line follow at fastest speed
void calibrate(void *par1); //forward declaration of function to calibrate the Pulolo reflectance
sensor
void intersection_blink(void *par2); // Intersection code
void pingfromcog2(void *par3);//forward declaration of function to read obstacle at the left of
robot
void pingfromcog3(void *par4);//forward declaration of function to read obstacle at the right of
robot
void aruco_follow(void);
void spinright_slow(void);
void spinleft_slow(void);
void turnleft_slow(void);
void turnright_slow(void);
static volatile int blink;
volatile int sensor_mean; // intersection variable
//volatile int front; // distance reading for front ultrasonic sensor
volatile int left;// distance reading for left ultrasonic sensor
volatile int right;// distance reading for right ultrasonic sensor
```

```
volatile int position, a; //position helps tell us where the black line is during line follow, while a helps us control when to calibrate and when to read position values
//volatile int intersection; //this helps us determine when we are at an intersection
volatile float scaled_sensor_val[8]; //stores the final reading of sensor values
unsigned int stack1[40 + 95]; // controls the second cog
unsigned int stack2[40 + 25]; //controls the third cog
unsigned int stack3[40 + 25]; //controls the 4th cog
unsigned int stack4[40 + 25]; //controls the 4th cog
int movecount;
int maininter; // counter for the center lane
int mainobstacle; // obstacle variable for the main lane
int allinter;
int section;
int left_flag;
int right_flag;
int c;
int d;
int l;
int r;
int aruco;
int park;
int main() // Main function
{
a = 1; // begin calibration
cogstart((void*)calibrate, NULL, stack1, sizeof(stack1));// start up cog 1 and run calibrate function
pause(2000); // wait 2 seconds
movecount = 30;
spinright(); // spin ring
movecount = 60;
spinleft(); // spin left
movecount = 30;
spinright(); // spin right
stopmotors(); //stop motors
```

```
a=2; // stop calibration and just keep reading position value
// initial variables to operate sequences
maininter = 0;
mainobstacle =0;
section =1;
left_flag =0;
right_flag =0;
aruco =0;
park =0;
//variables to control turns
//start ultrasonic cogs (2 and 3)
cogstart((void*)intersection_blink, NULL, stack2, sizeof(stack2)); // start second cog
cogstart((void*)pingfromcog2, NULL, stack3, sizeof(stack3)); // start third cog
cogstart((void*)pingfromcog3, NULL, stack4, sizeof(stack4)); // start fourth cog

//setup pins
set_direction(11,0);
set_direction(15,0);
set_direction(13,0);
set_direction(8,1);
set_direction(14,1);
set_direction(12,1);
set_output(14,0);
set_output(12,0);
//set_direction(8,0);
l =0;
r =0;
while(1)
{
if(sensor_mean <300){
do{
if (park ==1){
aruco_follow();
}
```

```
else if(section ==4 && sensor_mean <50){
movecount =20;
forward();
pause(200);
stopmotors();
pause(50);
set_output(12,1);
l += input(15);
r += input(13);
set_output(12,0);
set_output(8,1);
if(l>0 && aruco ==0){
do{
aruco += input(11);
movecount =10;
spinleft_slow();
pause(5);
stopmotors();
pause(200);
park = 1;
}while(l>0 && aruco ==0);
}
else if (r>0 && aruco ==0){
do{
aruco +=input(11);
movecount =10;
spinright_slow();
pause(5);
stopmotors();
pause(200);
park = 1;
} while(r>0 && aruco ==0);
}
set_output(8,0);
```

```
movecount = 20;
stopmotors();
pause(100);
}
else if(maininter ==0){
set_output(12,1);
l += input(15);
r += input(13);
linefollow();
}
else if(maininter ==2 && left ==1 && mainobstacle ==0 && left_flag ==1){
mainobstacle +=1;
maininter +=1;
movecount = 20;
forward();
pause(750);
spinleft();
pause(170);
stopmotors();
reverse_();
pause(1100);
stopmotors();
set_output(14,1);
set_output(14,0);
pause(2000);
forward();
pause(1100);
spinleft();
pause(200);
}
else if(maininter ==6 && right ==1 && mainobstacle ==1 && left_flag ==1){
mainobstacle +=1;
maininter +=1;
movecount = 20;
```

```
forward();
pause(650);
spinright();
pause(200);
stopmotors();
reverse_();
pause(1100);
stopmotors();
set_output(14,1);
set_output(14,0);
pause(2000);
forward();
pause(1100);
spinleft();
pause(200);
}
else if(maininter ==2 && right ==1 && mainobstacle ==0 && right_flag ==1){
mainobstacle +=1;
maininter +=1;
movecount = 20;
forward();
pause(750);
spinright();
pause(200);
stopmotors();
reverse_();
pause(1100);
stopmotors();
set_output(14,1);
set_output(14,0);
pause(2000);
forward();
pause(1100);
spinright();
```

```
pause(200);
}
else if(maininter ==6 && left ==1 && mainobstacle ==1 && right_flag ==1){
mainobstacle +=1;
maininter +=1;
movecount = 20;
forward();
pause(750);
spinleft();
pause(200);
stopmotors();
reverse_();
pause(1100);
stopmotors();
set_output(14,1);
set_output(14,0);
pause(2000);
forward();
pause(1100);
spinright();
pause(200);
}
else{
linefollow();
}
}while(sensor_mean<300);
}
else if(sensor_mean>300){
if(park ==1){
do{
stopmotors();
pause(5);
blink =1;
}while(park ==1);
```

```
}
else if(maininter ==0){
maininter +=1;
movecount =20;
stopmotors();
set_output(12,1);
l += input(15);
r += input(13);
set_output(12,0);
blink =1;
if(l>0){
l=0;
left_flag =1;
movecount =20;
turnleft();
pause(600);
}
else if (r>0){
r =0;
right_flag =1;
movecount =20;
turnright();
pause(600);
}
}
else if(maininter ==1 && left_flag ==1){
maininter +=1;
movecount =20;
turnright();
pause(600);
blink =1;
}
else if ((maininter ==3||maininter==5 ||maininter ==7) && left_flag ==1){
maininter +=1;
```

```
movecount =20;
turnleft();
pause(600);
blink =1;
}
else if (maininter ==8 && left_flag ==1){
maininter =0;
mainobstacle =0;
left_flag =0;
movecount =20;
section +=1;
turnright();
pause(600);
blink =1;
}
else if(maininter ==1 && right_flag ==1){
maininter +=1;
movecount =20;
turnleft();
pause(600);
blink =1;
}
else if ((maininter ==3||maininter==5 ||maininter ==7) && right_flag ==1){
maininter +=1;
movecount =20;
turnright();
pause(600);
blink =1;
}
else if (maininter ==8 && right_flag ==1){
maininter =0;
mainobstacle =0;
right_flag =0;
movecount =20;
```

```
section +=1;
turnleft();
pause(600);
blink =1;
}
else{
maininter +=1;
movecount = 20;
forward();
pause(100);
blink =1;
}
}
}
}
void calibrate(void *par1){
int cal_ini_sensor_val[8];
int cal_sensor_val[8];
int max_cal[8];
int min_cal[8];
int cal_value;
int sensor_ini_val[8];
int sensor_val[8];
int sensor_scale_sum;
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
cal_ini_sensor_val[i] = rc_time(i,1);
max_cal[i] = cal_ini_sensor_val[i];
min_cal[i]= cal_ini_sensor_val[i];
}
while(a==1)
{
```

```
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
cal_sensor_val[i] = rc_time(i,1);
}
for(int i=0;i<8;i++){
if(min_cal[i] >cal_sensor_val[i]){
min_cal[i]= cal_sensor_val[i];
}
else if (max_cal[i]<cal_sensor_val[i]){
max_cal[i] = cal_sensor_val[i];
}
}
}
while(a==2){
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
sensor_ini_val[i] = rc_time(i,1);
sensor_val[i] = sensor_ini_val[i];
if(sensor_val[i]< min_cal[i]){
sensor_val[i] = min_cal[i];
}
else if (sensor_val[i] >max_cal[i]){
sensor_val[i] = max_cal[i];
}
}
sensor_scale_sum = 0;
for(int i=0;i<8;i++){
float A = (float)(max_cal[i] -sensor_val[i]);
float B =(float) (max_cal[i] - min_cal[i]);
float C = (A/B);
```

```
scaled_sensor_val[i] = 1000 -(1000*(C));
sensor_scale_sum += scaled_sensor_val[i];
}
sensor_mean = (int)(sensor_scale_sum/8);
float D = 0*scaled_sensor_val[0]+
1000*scaled_sensor_val[1]+2000*scaled_sensor_val[2]+3000*scaled_sensor_val[3]+4000*scal
ed_sensor_val[4]+5000*scaled_sensor_val[5]+6000*scaled_sensor_val[6]+7000*scaled_sensor
_val[7];
float E = scaled_sensor_val[0]+
scaled_sensor_val[1]+scaled_sensor_val[2]+scaled_sensor_val[3]+scaled_sensor_val[4]+scale
d_sensor_val[5]+scaled_sensor_val[6]+scaled_sensor_val[7];
position = (int)(D/E);
}
}
void turnright_lf(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,30);
servo_speed(17,0);
pause(20);
}
}
// turn robot right by stopping left motor and turning right motor forward
void turnleft_lf(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,-30);
pause(20);
}
}
void linefollow(void){
if (position < 3000) {
do {
movecount = 1;
turnright_lf();
```

```
} while (position < 3000);
} else if (position > 4000)
do {
movecount = 1;
turnleft_lf();
} while (position > 4000);
movecount = 2;
forward();
}
// blink led at intersection
void intersection_blink(void *par2) {
while(1){
//whenever blink is 0 turn off both leds
while(blink ==0){
low(26);
pause(10);
}
while(blink ==1){ //when blink is one turn on red led, wait for 0.2 seconds and turn it off
high(26);
pause(200);
blink =0;
}
}
}
// read distance with left ultrasonic sensor then convert it to an obstacle true or false.
void pingfromcog2(void *par3) {
int a;
while(1)
{
a =ping_cm(10);
if(a>7){
left = 0;}
else if(a>2 && a <7){
left = 1;
```

```
}
}
}
// read distance with right ultrasonic sensor then convert it to an obstacle true or false.
void pingfromcog3(void *par4) {
int a;
while(1)
{
a =ping_cm(9);
if(a>7){
right = 0;}
else if (a>2 && a<7) {
right = 1;
}
}
}
void forward(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,30);
servo_speed(17,-30);
pause(5);
}
}
// stop robot by stopping both motors
void stopmotors(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,0);
pause(20);
}
}
// turn robot right by stopping right motor and turning left motor forward
void turnright(void) {
for (int i = 0; i < movecount; i++) {
```

```
servo_speed(16,95);
servo_speed(17,0);
pause(20);
}
}
// turn robot right by stopping left motor and turning right motor forward
void turnleft(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,-95);
pause(20);
}
}
// spin robot right by turning both motors forward
void spinright(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,95);
servo_speed(17,95);
pause(20);
}
}
void spinright_slow(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,20);
servo_speed(17,20);
pause(20);
}
}
// spin robot left by turning both motors backwards
void spinleft(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,-95);
servo_speed(17,-95);
pause(20);
```

```
}
}
void spinleft_slow(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,-20);
servo_speed(17,-20);
pause(20);
}
}
void aruco_follow(void){
set_output(12,1);
set_output(14,1);
int pi = input(15);
int pi2 = input(13);
if (pi2 ==1) {
do {
pi2 = input(13);
movecount = 2;
turnright_slow();
pause(5);
} while (pi2 ==1);
} else if (pi ==1)
do {
pi = input(15);
movecount = 2;
turnleft_slow();
pause(5);
} while (pi ==1);
movecount = 2;
forward();
}
void turnright_slow(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,20);
```

```
servo_speed(17,0);
pause(5);
}
}
// turn robot right by stopping left motor and turning right motor forward
void turnleft_slow(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,-20);
pause(5);
}
}
void reverse_(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,-30);
servo_speed(17,30);
pause(5);
}
}
```