Drone and Groundbot Communication Technical Report

Steve Gillet
Texas Tech University
May 2023

1 Acknowledgement

I would like to thank Isaac Mondragon and Rishikesh for being my long-time group mates and for all of the work and support they put into this project. A lot of work went into this project and a lot of frustration, a lot of pieces that were made and fussed over and weren't even used. This is probably the most work I've ever done on a project and certainly the most lessons learned. I'd also like to thank Dr. Hemmert and Dr. Nutter and all of the other supporting staff that were so willing to help us and accomodate what we were trying to accomplish, this really allows me to get the most out of my education when I can get these unique experiences and work on something that I'm really passionate about inside and outside of school.

2 Abstract

This paper describes our drone and groundbot submission for the IEEE Region 5 Conference Robotics Competition 2023. The objective of the competition is to have the drone and groundbot communicate with eachother to maneuver through a course with obstacles autonomously.

Contents

List of Figures

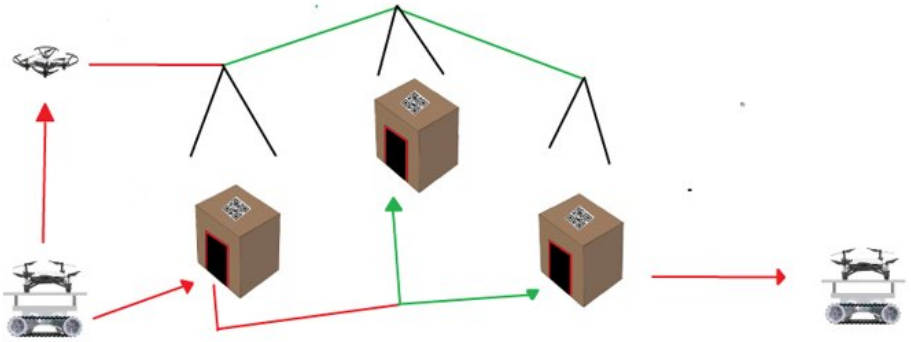

Figure 1: Project Sketch

3 Introduction

This paper describes the technical details of our drone and groundbot project for the IEEE Region 5 Conference Robotics Competition. The rules of the competition are that you must have a ground robot and a drone communicate and navigate a course autonomously. The drone must be a DJI Tello model. The obstacles are extra large Home Depot moving boxes that have holes carved in them for the ground robot to drive through. On the inside of the box is a QR code that tells you which box is the next box to go to and on the top of the box is the QR code that identifies that box. So the groundbot goes into the first box and reads which box to go to next, then the drone searches for that box from above and then the groundbot goes to that box and finds the next box and this continues until the groundbot goes through all of the boxes (Figure 1). There is also a second round where two teams go at the same time and whenever a team finishes all of their boxes they may land their drone on their groundbot and then land it on the other teams groundbot to take away their points.

The drone we used is the standard Tello DJI model. We modified the camera to allow it to see the QR codes more effectively. The groundbot is primarily made up of a Rover 5 Chassis, an Orange Pi 5 single-board-computer, a camera on a pan and tilt mount, 2 standard Rover 5 Chassis batteries, a buck converter to power the computer, an L298N H-bridge to drive the motors, and a I2C PWM breakout board to increase the natural PWM pins available.

4 Ethics and Safety

The primary ethical concerns for this project involve the originality of ideas and the proper credit to sources and the following of Texas Tech and FCC drone

flying regulations and the primary safety concern is for the quick-moving projectile nature of the drone and the electricity being used. All images and codes used from outside sources are carefully cited in the reference section.

Electrical safety is addressed in our lab safety training and the only major source of electricity we are dealing with is the lipo battery for which regular lab safety protocols are followed including handling and disposal. There is a bin in the lab where we work dedicated to burning batteries that is located near the entrance.

The FCC and Texas Tech regulations are currently being avoided by keeping the drone under half a pound in weight and only flying it indoors for testing. There is an outdoor area that we have already designated at a nearby retired Air Force base.

5 Software

5.1 Main Control Script

The Orange Pi 5 is flashed with the Armbian 11 operating system. All of the operating code comes in the form of a Python script using several modules and libraries to control the various aspects of the project. The Python version used in 3.10.

The main script that is run is called runCode.py. As you can see in Figure 2 there is a few libraries that were used the first few are for timing and threading coordination, then we have a few that are used to control the GPIOs and PWMs needed to control the motors and servos on the groundbot and pan-tilt camera (board, digitalio, busio, adafruit), and the rest are functions that were created and abstracted to other Python scripts to modularize and seperate the code.

```python
import cv2
import numpy as np
import time
import threading
import board
import digitalio
import busio
from adafruit_pca9685 import PCA9685
from adafruit_servokit import ServoKit
from simple_pid import PID
# from ultralytics import YOLO
# from TelloPython.Single_Tello_Test.tello import Tello
# from TelloPython.Single_Tello_Test.tello_test import flyDrone
import sys
from datetime import datetime
from detectDoor import readBox
from drone_detection.supervision2 import supervision2
import subprocess
from drone_detection.robotmove import moveToPosition
import adafruit_mpu6050
from gridCode.workingCodeCustomTello import droneGrid
```

Figure 2: Main Script Libraries

The next piece of code, Figure 3, is a function that connects automatically to the drone, an important part of this project is that it all starts and runs autonomously and this is a piece of that which will be discussed more in later sections. The function checks if a password is required (in the case of the drone we used there is not) and then uses the 'subprocess' library to run a bash terminal command to connect to the Tello drone's wifi which is put into the function as the 'ssid' argument.

```python
def connect_to_tello_wifi(ssid, password=None):
    try:
        # If a password is provided, use it to connect to the Wi-Fi
        if password:
            command = f'nmcli device wifi connect "{ssid}" password "{password}"'
        else:
            command = f'nmcli device wifi connect "{ssid}"'

        result = subprocess.run(command, shell=True, check=True, text=True, capture_output=True)
        print(f"Connected to {ssid}")
    except subprocess.CalledProcessError as e:
        print(f"Error connecting to {ssid}: {e.output}")
```

Figure 3: Main Code Wifi Function

The next part of the main code, Figure 4, is the most important component and is what allows the different pieces of the code to communicate with each other and control the ground robot. This code is the 'GroundBot' class that contains all of the controls and variables that when instantiated is passed into the main functions so that they all access the same variables. The '__init__' method is what is called when the GroundBot object is initialized and is vital

to the accessing and changing of variables from various pieces of code.

```
class GroundBot:
    def __init__(self):
        self.panPin = 9
        self.tiltPin = 8
        self.in1 = digitalio.DigitalInOut(board.D24)
        self.in2 = digitalio.DigitalInOut(board.D15)
        self.in3 = digitalio.DigitalInOut(board.D22)
        self.in4 = digitalio.DigitalInOut(board.D23)

        self.in1.direction = digitalio.Direction.OUTPUT
        self.in2.direction = digitalio.Direction.OUTPUT
        self.in3.direction = digitalio.Direction.OUTPUT
        self.in4.direction = digitalio.Direction.OUTPUT
        self.i2c = busio.I2C(board.SCL, board.SDA)

        self.pca = PCA9685(self.i2c)

        # Set the PWM frequency to 60hz.
        self.pca.frequency = 60
        self.kit = ServoKit(channels=16)
        self.kit.servo[self.panPin].set_pulse_width_range(500,2500)
        self.kit.servo[self.tiltPin].set_pulse_width_range(500,2500)

        self.cam1 = Cam(self.kit, self.panPin, self.tiltPin)

        # Set the PWM duty cycle for channel zero to 50%. duty_cycle is 16 bits to match other PWM objects
        # but the PCA9685 will only actually give 12 bits of resolution.
        self.ena = 14
        self.enb = 15
```

Figure 4: Main GroundBot Class Declaration

As you can see in Figure 5, there are several methods that are defined for moving the robot, they simply use the 'digitalio' library to control the input pins for the motor driver which determine the direction and the 'PCA9685' I2C breakout board library to control the enable pins of the motor driver that determine the speed. The speed is measured in hexadecimal with 'FFFF' being the maximum speed and as you can see in the code it never goes above '7FFF' which is 50% because the robot does not need to move quickly and the multiple aspects of the project are easier to manage at slow speed. The slow turn for example, which is used when the groundbot is trying to turn and keep the camera pointed at a stationary object at the same time, is at '47FF' which is about 33% speed.

```
def stop(self):
    self.in1.value = False
    self.in2.value = False
    self.in3.value = False
    self.in4.value = False
    self.pca.channels[self.ena].duty_cycle = 0x0000
    self.pca.channels[self.enb].duty_cycle = 0x0000

def turnRight(self):
    self.in1.value = True
    self.in2.value = False
    self.in3.value = False
    self.in4.value = True
    self.pca.channels[self.ena].duty_cycle = 0x7FFF
    self.pca.channels[self.enb].duty_cycle = 0x7FFF

def turnLeft(self):
    self.in1.value = False
    self.in2.value = True
    self.in3.value = True
    self.in4.value = False
    self.pca.channels[self.ena].duty_cycle = 0x7FFF
    self.pca.channels[self.enb].duty_cycle = 0x7FFF

def slowRight(self):
    self.in1.value = True
    self.in2.value = False
    self.in3.value = False
    self.in4.value = True
    self.pca.channels[self.ena].duty_cycle = 0x47FF
    self.pca.channels[self.enb].duty_cycle = 0x47FF
```

Figure 5: GroundBot Move Functions

The next piece of code is the function to control the pan and tilt of the camera, Figure 6. 'dx' and 'dy' are passed into the function from image processing piece of code that comes later and defines how much to move the camera in the x and y direction (pan and tilt). Then those amounts are put into a PID algorith implemented with the 'simple_pid' library in order to smooth out the movements and then the object variables are set to that resulting value and then the servos are set to the new values using the 'servokit' library. The 'np.clip' function is there to make sure the values don't go above or below the max and min permissible values.

```python
# Update the adjust_pan_tilt_servos function to use the PID controller
def adjust_pan_tilt_servos(self, dx, dy):
    try:
        # Calculate the pan and tilt output using the PID controller
        pan_output = self.pan_pid(dx)
        tilt_output = self.tilt_pid(dy)

        self.cam1.panAngle += pan_output
        self.cam1.tiltAngle -= tilt_output
        # if cam1.panAngle < 0 or cam1.tiltAngle > 180:
        #     turnRight()
        # elif cam1.panAngle > 180
        #     turnLeft()
        self.cam1.panAngle = np.clip(self.cam1.panAngle, 0, 180)
        self.cam1.tiltAngle = np.clip(self.cam1.tiltAngle, 0, 180)

        self.kit.servo[self.panPin].angle = self.cam1.panAngle
        self.kit.servo[self.tiltPin].angle = self.cam1.tiltAngle
    except Exception as e:
        print(f"Error: {e}")
```

Figure 6: Pan and Tilt Function

The last couple of methods which you can see in Figure 7 use the MPU 6050 to keep track of the yaw of the groundbot in order to be able to return to the same heading consistently and a wait function for the groundbot to wait to move on to the next box until it's location has been found by the drone. 'updateYaw' grabs the yaw from the gyroscope using the 'mpu6050' library, converts it to rads and then updates the object attribute offset by a drift correction attribute. The 'waitForBoxPosition' simply sleeps until the designated box key position is no longer (0,0) which indicates that it's position has been updated by the drone search code.

```python
def updateYaw(self):
    # adjust this value based on how much the yaw drifts over time

    while self.running:
        gyro = self.mpu.gyro
        gyro_rad = [g * np.pi / 180 for g in gyro]  # Convert gyro data to radians

        self.yaw += (gyro_rad[2] + self.yaw_drift_correction) * self.dt  # Subtract the correction from the yaw
        time.sleep(self.dt)

def waitForBoxPosition(self, boxKey):
    while self.box_positions[boxKey] == (0, 0):
        time.sleep(0.1)
```

Figure 7: Update Yaw and Wait Methods

The camera class, Figure 8, is instantiated inside of the groundbot 'init' method and it is simply used to initialize and keep track of the pan and tilt values.

```
class Cam:
    def __init__(self, kit, panPin, tiltPin):
        self.kit = kit
        self.panAngle = 90
        self.tiltAngle = 80
        self.panPin = panPin
        self.tiltPin = tiltPin
        self.kit.servo[self.panPin].angle=self.panAngle
        self.kit.servo[self.tiltPin].angle=self.tiltAngle
    def camLeft(self):
        self.panAngle = 180
        self.kit.servo[self.panPin].angle=self.panAngle
    def camRight(self):
        self.panAngle = 0
        self.kit.servo[self.panPin].angle=self.panAngle
    def camForward(self):
        self.panAngle = 90
        self.kit.servo[self.panPin].angle=self.panAngle
```

Figure 8: Camera Class

Finally we move into the actual main loop of the code which executes everything, Figure 9. All of the main pieces are preceded by a print statement which briefly describes what is happening at any given time. First the drone takes off and begins its search pattern using the 'droneGrid' function. Then it falls into the main loop which starts with the 'readBox' function where the groundbot goes into the first box and looks for the first QR code. When it finds the next QR code value it backs out a safe distance to clear the box and then corrects it's orientation using the yaw values and the next couple of turn loops. Then the groundbot waits using the 'waitForBoxPosition' method while the drone finds the 'nextQRcode' position.

```
print('drone searching')
droneGrid(groundBot)

while True:
    # Read the first box and get the nextQRcode value
    print('groundbot searching')
    readBox(groundBot)
    # break
    print('groundbot backing out')
    time.sleep(1)
    groundBot.backward()
    time.sleep(3)

    if groundBot.yaw < 0.000:
        while groundBot.yaw < 0.000:
            groundBot.turnLeft()
            time.sleep(0.1)
        groundBot.stop()

    elif groundBot.yaw > 0.000:
        while groundBot.yaw > 0.000:
            groundBot.turnRight()
            time.sleep(0.1)
        groundBot.stop()

    print('groundbot waiting')
    # Wait for the corresponding box_positions variable to be updated
    groundBot.waitForBoxPosition(groundBot.nextQRcode)
```

Figure 9: First Half of the Main Control Loop

The second half of the loop, Figure 10, checks to see if the 'nextQRcode' value is 'DONE' in which case it moves to the last position and then breaks out of the loop and stops or else it moves to the next position and then goes back to the top of the loop and goes into the box and continues the code. The if statement there is to make sure that the groundbot stops a meter before it gets to the position so that it is looking at the right box.

```
# Move to the position specified by the nextQRcode
print('groundbot moving to next box')
nextPosition = (groundBot.box_positions[groundBot.nextQRcode][0] - groundBot.currentPosition[0], groundBot.box_positions[groundBot.nextQRcode][1]
if (nextPosition[1]) < -100:
    positionToMove = (nextPosition[0], nextPosition[1] + 100)
elif (nextPosition[1]) > 100:
    positionToMove = (nextPosition[0], nextPosition[1] - 100)
else:
    if (nextPosition[0]) < -100:
        positionToMove = (nextPosition[0] + 100, nextPosition[1])
    elif (nextPosition[0]) > 100:
        positionToMove = (nextPosition[0] - 100, nextPosition[1])
    else:
        positionToMove = nextPosition

# Check if the nextQRcode value is 'done'
if groundBot.nextQRcode == 'DONE':
    # Perform the action for the 'done' QR code
    moveToPosition(groundBot, nextPosition)

    # Enter and read the last box
    print('groundbot searching')
    readBox(groundBot)

    # Break the loop
    break

moveToPosition(groundBot, nextPosition)
groundBot.currentPosition = groundBot.box_positions[groundBot.nextQRcode]
```

Figure 10: Second Half of the Main Control Loop

5.2 Drone Code

The drone code consists of two main functions, the 'tellopath' function which defines the drones search path and moves through it and the 'QR_tello' function which processes the frames coming from the Tello and searches them for QR codes and then sets the position attribute to it's current position based up which QR code value is read.

The 'send_command' method is how the drone is controlled and as you can see in Figure 11, is done with simple strings such as 'forward 20' which tells the drone to move forward 20 cm. The drone will do a certain number of forward movements and update it's position based on those movements and then move right and go backward a number of times and in this way moves in a lawnmower pattern.

```python
def tellopath(tello):
    tello.send_command("takeoff")
    time.sleep(7)
    tello.send_command("speed 10")
    # height = tello.get_height()
    # Rheight = 40
    tello.send_command(f"up 70")
    waitfordrone()
    tello.send_command("forward 20")
    waitfordrone()
    yp += 20

    print("Current Position: {}, {}".format(groundBot.xp, groundBot.yp))

    while True:
        # Move forward in increments of 76 cm on the groundBot.yp axis
        while groundBot.yp < 292 and (groundBot.xp == 0 or groundBot.xp == 152 or groundBot.xp == 304 or groundBot.xp == 456
            tello.send_command("forward 68")
            waitfordrone()
            groundBot.yp += 68

            print("Current Position: {}, {}".format(groundBot.xp, groundBot.yp))
            if groundBot.yp == 292:
                break
```

Figure 11: The Tello Path Function

The 'QR_tello' function then uses those positions to update the 'box_position' attribute using the barcode data from the 'decode' function imported from the 'pyzbar' library, Figure 12.

```python
def QR_tello(tello):
    while True:
        # Get the Tello drone's camera feed
        frame = tello.frame

        # If the frame is not None, decode barcodes and display the frame
        if frame is not None:
            gray_img = cv2.cvtColor(frame,0)
            barcode = decode(gray_img)

            for obj in barcode:
                points = obj.polygon
                (x,y,w,h) = obj.rect
                pts = np.array(points, np.int32)
                pts = pts.reshape((-1, 1, 2))
                cv2.polylines(frame, [pts], True, (0, 255, 0), 3)

                barcodeData = obj.data.decode("utf-8")
                if barcodeData == 'A':
                    groundBot.box_positions['A'] = (groundBot.xp, groundBot.yp)
                    print(groundBot.box_positions['A'])
                elif barcodeData == 'B':
                    groundBot.box_positions['B'] = (groundBot.xp, groundBot.yp)
                    print(groundBot.box_positions['B'])
                elif barcodeData == 'C':
                    groundBot.box_positions['C'] = (groundBot.xp, groundBot.yp)
                    print(groundBot.box_positions['C'])
```

Figure 12: Drone QR Code Reading Function

The two functions run at the same time using the 'threading' library which

allows you to run separate pieces of code at the same time as threads. These threads currently run forever until the drone reaches the end of it's movement pattern. The lawnmower pattern goes forward and to the right 20 feet and 30 feet and is designed so that if the groundbot starts at one of the corners of the playing field then the drone will cover the whole area. The entire drone code being run on threads allows the main body of code that controls the groundbot to continue running the whole time while the drone is moving simultaneously.

5.3 Find QR Code Groundbot Script

The 'readBox' function defines the code for moving the groundbot, aiming the camera, and reading the QR code when the groundbot is close enough to the inidivual box it's looking for to single that out and find the next QR code. The script starts (Figure 13) with a few definitions needed specifically in that script. There's a couple of functions that are important including the 'timer_function' that defines how long the movements of the groundbot will last, this usually depends on how offcenter the groundbot is from the center of the box. Then there is the 'equalizeHistograms' function which exists to attempt to control for the changes in color that occur in different lightings. When a frame comes from the camera it is converted to HSV color values and those values are used to search for specific colors, especially the red color of the tape that outlines the entry of the box.

```
def readBox(groundBot):

    movement_pid = PID(1, 0, 0, setpoint=0, output_limits=(-1, 1))

    # Define a function that will be executed by the timer thread
    def timer_function(duration):
        # print(duration)
        time.sleep(abs(movement_pid(duration)))
        groundBot.stop()
        groundBot.turnFlag = False

    def equalizeHistograms(hsvImage):
        h, s, v = cv2.split(hsvImage)
        v = cv2.equalizeHist(v)
        return cv2.merge((h,s,v))

    cap = cv2.VideoCapture(0)

    frameCenterX = 320
    frameCenterY = 240
```

Figure 13: Beginning of Box Reading Function Definition

Another important piece of the definitions is the color masks which were fine tuned to the competition lighting, Figure 14. These masks work by taking a range of HSV values and either singling them out or blocking them out. For

example, you can see the color definitions for the red color are quite broad (H value of 0-179) but the orange colors are pretty narrow (H value of 163-179) which is because in the primary search for the box entrance you want to be able to look for the red of the tape while blocking out the orange branding on the box.

```
frameCenterX = 320
frameCenterY = 240

# lowerRedLow = np.array([0, 50, 50])
# upperRedLow = np.array([10, 255, 255])
# lowerRedHigh = np.array([175, 50, 50])
# upperRedHigh = np.array([180, 255, 255])
# lowerOrange = np.array([2,50,80])
# upperOrange = np.array([15, 210, 120])
# lowerCardboard = np.array([165, 25, 80])
# upperCardboard = np.array([180, 90, 165])

lowerRed = np.array([0, 186, 75])
upperRed = np.array([179, 255, 247])

lowerOrange = np.array([163, 98, 70])
upperOrange = np.array([179, 202, 200])

lowerCardboard = np.array([11, 34, 88])
upperCardboard = np.array([32, 81, 215])

redFlag = 0
minWidthCardboard = 50

look_for_qr_code = False
frameCounter = 0
foundFlag = 0
flagThresh = 0
```

Figure 14: Definition of Color Masks

In Figure 15, you can see the piece of code where the red frame is found. The masks are applied to the frame to find the color and then the contours are found by the opencv 'findContours' function and then it goes through the contours using the 'approxPolyDP' function to find those with a certain number of vertices which indicate that a square shape has been found and then if it is a certain width in pixels the center is found and sent to the groundbot and camera movement functions so that they can orient themselves around the center of the entrance of the box. The 'flagThresh' variable is set if the red shape takes up 85% of the screen implying that the groundbot is close enough to the entrance to start looking for the QR code.

```python
for contour in contours:
    approx = cv2.approxPolyDP(contour, 0.01*cv2.arcLength(contour, True), True)
    if len(approx) >= 3:
        x,y,w,h = cv2.boundingRect(contour)
        aspect_ratio = float(w)/h
        # print(w)
        if w > 100:
            # print(w*h)
            print(float(w)/frame_width)
            if 0.85 <= float(w)/frame_width:
                if flagThresh > 0:
                    look_for_qr_code = True
                flagThresh +=1
            # red_pixels = cv2.countNonZero(mask[y:y+h, x:x+w])
            if True:
                redFlag += 1
                cv2.drawContours(frame, [contour], 0, (0, 255, 0), 3)
                centerX = x + w / 2
                centerY = y + h / 2

                dx = centerX - frameCenterX
                dy = centerY - frameCenterY
                groundBot.adjust_pan_tilt_servos(dx, dy)
                if groundBot.cam1.panAngle < 87 and not groundBot.turnFlag:
                    groundBot.turnRight()
                    # print('right')
                    # # Create a timer thread that will execute the timer_function after 5 seconds
                    timer = threading.Thread(target=timer_function, args=(.01080*dx,))
                    # # Start the timer thread
                    timer.start()
                    groundBot.turnFlag = True
```

Figure 15: Find the Red Entrance

The code to search for the QR code (Figure 16) is located in an 'if' statement at the top of the loop and if it is activated the camera is tilted up to its maximum tilt and then the groundbot moves slowly forward until it finds the QR code. The 'try-catch' block searches the frames for the QR code using the same function from 'pyzbar' and if it is detected, the 'nextQRcode' object attribute is set and the loop is broken out of to return to the main code.

```
if look_for_qr_code:
    print('looking for qr code')
    # Set the tilt angle to look up
    groundBot.cam1.tiltAngle = 0
    groundBot.kit.servo[groundBot.tiltPin].angle = groundBot.cam1.tiltAngle

    # Move forward slowly
    groundBot.slowForward()
    forwardTimer = threading.Thread(target=timer_function, args=(0.1,))
    forwardTimer.start()

    # Search for the QR code
    try:
        gray_img = cv2.cvtColor(frame,0)
        barcode = decode(gray_img)

        for obj in barcode:
            points = obj.polygon
            (x,y,w,h) = obj.rect
            pts = np.array(points, np.int32)
            pts = pts.reshape((-1, 1, 2))
            cv2.polylines(frame, [pts], True, (0, 255, 0), 3)

            qrCodeValue = obj.data.decode("utf-8")
            barcodeType = obj.type
            string = "Data " + str(qrCodeValue) + " | Type " + str(barcodeType)

            cv2.putText(frame, string, (x,y), cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,0,0), 2)
            print("Barcode: "+qrCodeValue +" | Type: "+barcodeType)

            if qrCodeValue:
                print(qrCodeValue)
                groundBot.nextQRcode = qrCodeValue
                groundBot.stop()
                foundFlag = 1
    except Exception as e:
        print(e)
```

Figure 16: Groundbot Search for QR Code

There is also a secondary piece of code in the case that the entrance to the box cannot be seen from the drone's vantage point, in which case the side of the box is searched for and then the groundbot turns until the camera is pointed at the side of the box and the groundbot is pointed perpendicularly and then the groundbot moves in a forward arc to move around until the entrance can be seen. In Figure 17, you can see how the orange mask is used to look for the branding on the side of the box and how the groundbot is controlled to turn and move around the box using the 'slowRight' and 'angleLeft' functions.

```
if redFlag <= 3:

    # Convert the frame to the HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    hsv = equalizeHistograms(hsv)
    # Apply a Gaussian blur to reduce noise
    blur = cv2.GaussianBlur(hsv, (5, 5), 0)
    # Threshold the frame to extract the lighter object
    mask = cv2.inRange(blur, lowerCardboard, upperCardboard)
    orangeMask = cv2.inRange(blur, lowerOrange, upperOrange)

    # Find contours in the binary mask
    contours, hierarchy = cv2.findContours(orangeMask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Loop through each contour
    for cnt in contours:
        # Calculate the bounding box of the contour
        x, y, w, h = cv2.boundingRect(cnt)
        aspect_ratio = float(w)/h
        # If the width of the bounding box is greater than the minimum width and the height is less
        if w >= minWidthCardboard and 0.8 < aspect_ratio < 1.2:
            # Draw a rectangle around the bounding box
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

            centerX = x + w / 2
            centerY = y + h / 2
            dx = centerX - frameCenterX
            dy = centerY - frameCenterY
            groundBot.adjust_pan_tilt_servos(dx, dy)
            if groundBot.cam1.panAngle < 170:
                print('cardboard turn')
                groundBot.slowRight()
            else:
                print('cardboard angle')
                groundBot.angleLeft()
```

Figure 17: Groundbot Code to Move Around the Side of a Box

5.4 Groundbot Move to Position Code

The last integral piece of code is the code that defines how the robot should move to the position of the next box given to it by the drone. This code (Figure 18) is pretty simple in that the groundbot moves the designated distance in the x direction and then turns and moves in the designated y position, the x and y direction are maintained from the yaw attribute. So, you'll see the groundbot turn until the yaw corresponds with a right or left turn 90 degrees based on whether x is positive or negative, then it moves forward with the 'move' function which simply moves the robot forward unless an obstacle is detected in front of it, which is defined as a large contour in the frame, and if there is an obstacle the groundbot makes a simple square movement and then the normal movement is continued. Then the robot moves in the y direction.

```
def move():
    groundBot.forward()

    while check_obstacle():
        groundBot.stop()
        avoidObstacle()

x, y = coord

timePerCM = 0.033

if x > 0:
    while groundBot.yaw > -0.019:
        groundBot.turnRight()
        time.sleep(0.1)
    groundBot.stop()
elif x < 0:
    while groundBot.yaw < 0.019:
        groundBot.turnLeft()
        time.sleep(0.1)
    groundBot.stop()

move()
```

Figure 18: Code Defining How to Get Groundbot to Next Position

6 Hardware

The primary pieces of hardware are the DJI Tello Drone, Rover 5 Chassis, the Orange Pi 5, The L298N H-Bridge Motor Driver, 2 Tenergy 9.6V Flat NiMH Batteries, the pan and tilt servos, camera, and PCA9685 breakout board.

6.1 DJI Tello Drone

The DJI Tello Drone is an educational drone that can be controlled using the Tello app or, in the case of this project, the DJI Tello SDK which takes simple strings sent from a wifi connection. This project uses a modified Tello code from the 'dji-tello' Github to make the connections and process the video frames and send the commands. The drone has approximately 13 minutes of flight time per battery. The drone for this project was modified so that the camera faces straight down because the only need for the camera here was to read QR codes directly beneath the drone. This drone is prized for it's safety and simplicity and it is very easy to fly in the right conditions. All of the stabilization and the precision of the movements is done internally with the drone's firmware. The drone does this with two simple cameras on the bottom which allow it to locate itself and move in reference to the information it gets from those cameras. This makes the drone easy to fly but unfortunately heavily limits its ability to fly in different lighting conditions or outdoors in windy conditions.

Figure 19: DJI Tello Drone

6.2 Rover 5 Chassis

The Rover 5 Chassis (Figure 20) is a simple rover body that includes tread, two motors, and a motor encoder that is commonly used in early project labs at Texas Tech University. In this project the motor encoder is unused.

Figure 20: Rover 5 Chassis

6.3 Orange Pi 5

The Orange Pi 5 is a single board computer with a Rockchip RK3588s octa-core 64-bit processor, ARM Mali-G619 GPU, and an NPUfor 6Tops AI computing power. The main advantage of this computer and why it was chosen is that it has a lot of bang for buck with a significantly more powerful CPU than other computers in its price-range. The abundance of image processing in this project necessitates a powerful computer to get the job done. The Armbian 11 OS was used here for ease of use and compatibility with the various libraries used.

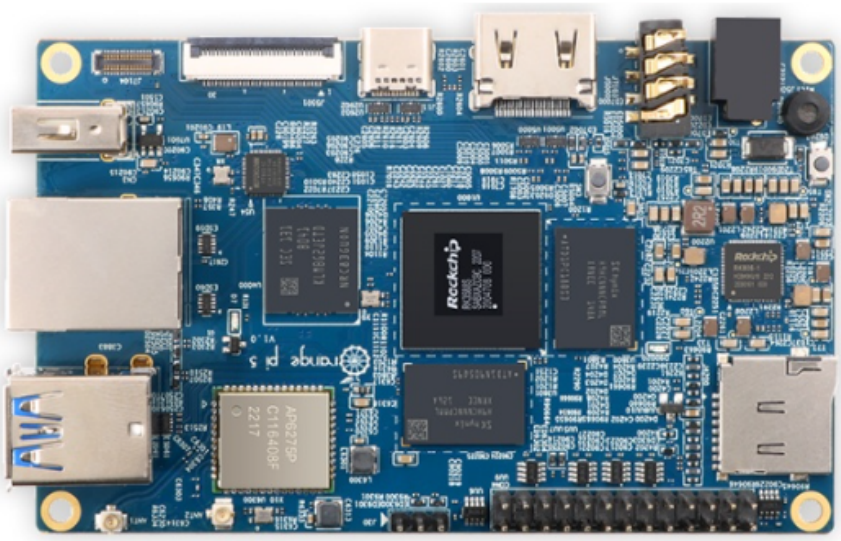

Figure 21: ESC

6.4 L298N H-Bridge Motor Driver

The L298N H-Bridge Motor Driver (Figure 22) is a dual channel full H-Bridge which allows 2 DC motors to be controlled independently. Motor input voltage of 5 to 35 volts. Drive current 2 amps. Logic voltage 5 volts and 0 to 36 milliamp current. This motor driver was chosen because of its affordability and simplicity as well as the logic power matching the requirements for the Orange Pi.

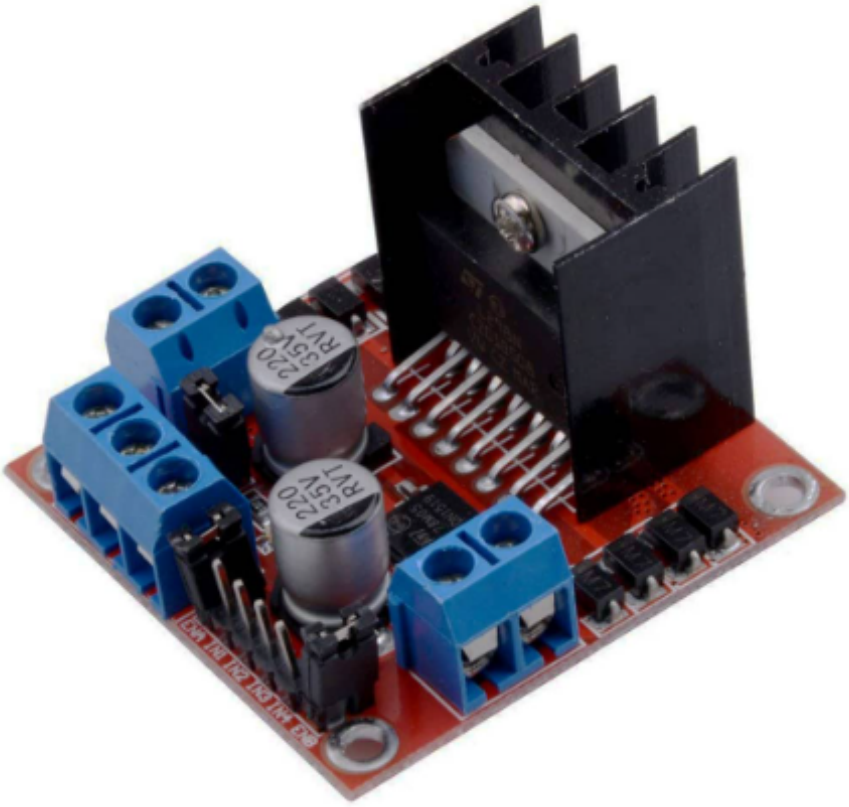

Figure 22: L298N H-Bridge Motor Driver

6.5 Batteries

The Tenergy 9.6V Flat NiMH Battery (Figure 23) was chosen for accessibility and voltage that works well with the Rover 5 Chassis. There were two batteries used on groundbot, one that powers the motors and one that powers the computer.

Figure 23: Tenergy 9.6V Flat NiMH Battery

6.6 Miuzei Servo Motor

The Miuzei SG90 Servo Motors (Figure 24) are used for the pan and tilt camera mount, one for panning and the other connected to it at 90 degrees for tilting. They are controlled with PWM from the PCA breakout board and were chosen for their accessibility and 180 degree turning ability.

Figure 24: Miuzei Servo Motor

6.7 Camera

The camera used was a Raspberry Pi camera from the Texas Tech ECE stockroom and was chosen for its USB connection because the Orange Pi 5 uses a different CSI connection to most cameras available in the area.

6.8 PCA9685 16-Channel Module PWM

The PCA9685 16-Channel PWM Module (Figure 25) is 5 volt compatible and can be run at 3.3 volts. It was used in this project because only 3 PWM pins were available on the Orange Pi 5 and 4 were needed for the 4 motors (2 DC, 2 Servo) and all you need for the PCA is one available I2C channel.

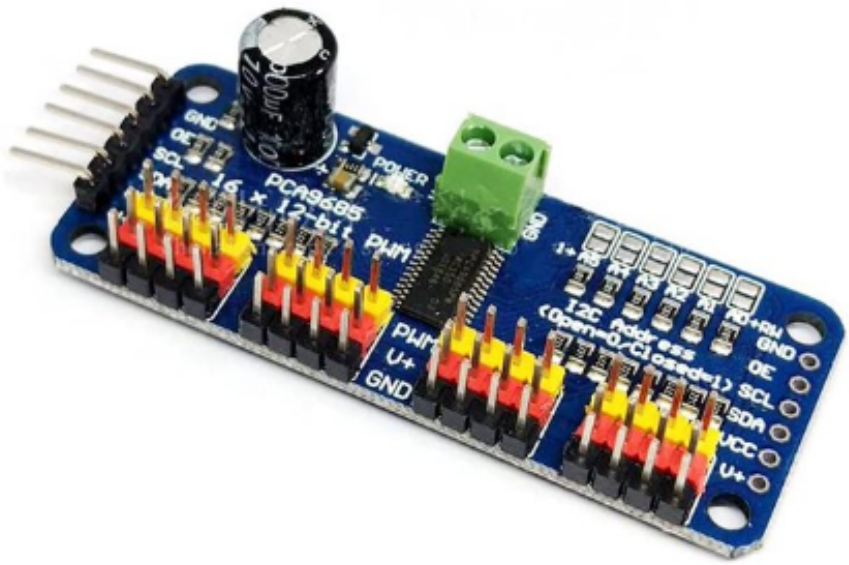

Figure 25: PCA9685 16-Channel Module PWM

7 Conclusion

The project at this point is able to complete all of the objectives independently and needs to be integrated. The code for the groundbot to find the entrance of the box is still inconsistent mostly due to lighting, the reflectiveness of the red tape appears to hurt the ability to find the entrance rather than help as it increases the variability to lighting. Other methods have been conceived including machine learning and shape matching, but the competition draws near and the time for experimentation and implementation is little. The hope is that the color values and lighting can be fine tuned at the competition grounds to match those conditions. The drone search code appears to work great although there are concerns that the method is highly dependent on the starting position which is unknown. The integration of all of the pieces and implementation of an automatic starting mechanism should be easy to implement.

References

[] *DJI Tello Quadcopter Drone Boost Combo with HD Camera and VR, comes 3 Batteries, 8 Propellers, Powered by DJI Technology and Intel 14-Core Processor, Coding Education, Throw and Go*. URL: `https://www.amazon.com/`

DJI-Quadcopter-Protective-Propellers-Technology/dp/B07K8ZM1H1. (accessed: 04.29.2023).

[] *L298N Motor Driver Module (2A)*. URL: https://digitalelectronics.lk/product/l298n-motor-driver-module-2a/. (accessed: 04.29.2023).

[] *Miuzei 10 Pcs 9G SG90 Micro Servo Motor Kit for RC Robot Arm/Hand/Walking Helicopter Airplane Car Boat Control, Mini Servos for Arduino Project*. URL: https://www.amazon.ca/Miuzei-Helicopter-Airplane-Remote-Control/dp/B07H85M78M. (accessed: 04.29.2023).

[] *Orange Pi 5*. URL: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html. (accessed: 04.29.2023).

[] *Rover 5 Chassis with 4 Encoders 4 Motors*. URL: https://digiwarestore.com/id/robot-kits/rover-5-chassis-with-4-encoders-4-motors-642325.html. (accessed: 04.29.2023).

[] *Treedix Compatible with Arduino PCA9685 16-Channel Module PWM/Steering Gear Driver Board IIC*. URL: https://www.walmart.com/ip/Treedix-Compatible-with-Arduino-PCA9685-16-Channel-Module-PWM-Steering-Gear-Driver-Board-IIC/1561206955?wmlspartner=wlpa&selectedSellerId=101292976&adid=22222222222000000000&wmlspartner=wmtlabs&wl0=e&wl1=o&wl2=c&wl3=10352200394&wl4=pla-1103028060075&wl5=&wl6=&wl7=&wl10=Walmart&wl11=Online&wl12=1561206955_10001312336&wl14=pca9685&veh=sem. (accessed: 04.29.2023).

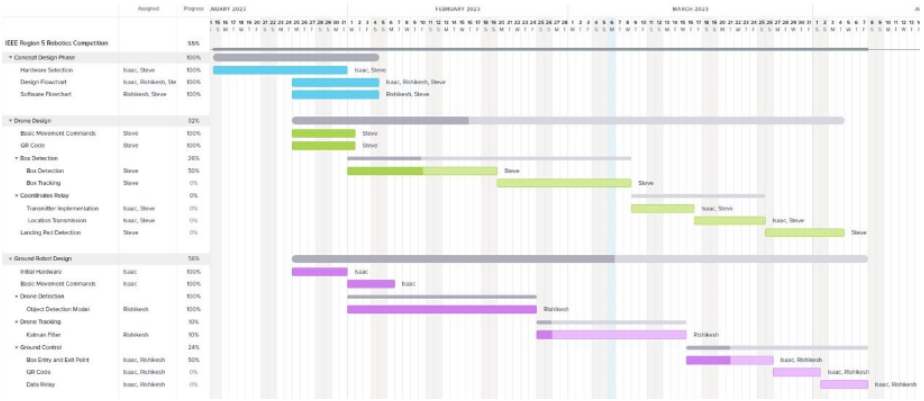

Figure 26: Gantt Chart

Project Lab 4						
	Running Total			*Estimate*		
Direct Labor:						
Individuals	**Rate/Hr**	**Hours**		**Rate/Hr**	**Hours**	
Isaac Mondragon	$25	135	$3,375.00	$25	120	$3,000.00
Stephen Gillet	$25	148	$3,700.00	$25	120	$3,000.00
Rishi Kesh	$25	157	$3,925.00	$25	120	$3,000.00
DL Subtotal (DL)		**Subtotal:**	$11,000.00		**Subtotal:**	$9,000.00
Labor Overhead	**Rate:**	100%	$11,000.00	*rate:*	100%	$9,000.00
Total Direct Labor (TDL)			$22,000.00			$18,000.00
Contract Labor:	**Rate/Hr**	**Hours**		**Rate/Hr**	**Hours**	
Lab Assistant	$40	0	$0.00	$40	0	$0.00
Classmates	$15	0	$0.00	$15	5	$75.00
Instructor	$200	0	$0.00	$200	5	$1,000.00
Total Contract Labor (TCL)			$0.00			$1,075.00
Direct Material Costs:						
(from Material Cost worksheet)						
Total Material Cost:			$541.92			$650.00
Equipment Rental Costs:	**Rental Rate**	**Days**		**Rental Rate**	**Days**	
Oscilloscope	0.20%	70	$396.20	0.20%	98	$554.68
Function Generator	0.20%	70	$62.86	0.20%	98	$88.00
DMM	0.20%	70	$18.20	0.20%	98	$25.48
Power Supply	0.20%	70	$33.10	0.20%	98	$46.34
Soldering Iron	0.20%	70	$3.50	0.20%	98	$4.90
Total Rental Costs: (TRM)			$513.86			$719.41
Total TDL+TCL+TDM+TRM			$23,055.78			$20,444.41
Business overhead	**Rate:**	55%	$12,680.68		55%	$11,244.42
Total Cost:		**Current**	$35,736.46		**Estimate**	$31,688.83

Figure 27: Budget Chart