MEMORANDUM

6/16/2023

TO: Charlie Refvem, Professor of Mechanical Engineering
FROM: Gabriel Ahern, Trevor Foley
SUBJECT: **ME 507 Term Project**

INTRODUCTION/OVERVIEW:

The term project assigned in ME 507 for Spring Quarter of 2023 consisted of designing and manufacturing a robot for a large-scale version of the game Hungry-Hungry Hippos. In this game, our robot was required to start at one of four colored stations, then find and collect balls of the same color. It was then supposed to deposit the collected balls at either its home station or launch them into a center bucket completely autonomously. Details on our design, as well as the design of the software for controlling it, are specified in the following sections.

MAJOR HARDWARE – MOTORS/ACTUATORS:

Our intention when creating the mechanical design for our hippo was to have an agile robot that could easily change directions to either pick up balls or avoid other bots. In order to meet this criterion, we chose to use four DC gearbox motors ("TT Motors") to drive a set of mecanum wheels. This gave our bot the ability to strafe, move diagonally, and easily rotate in place.

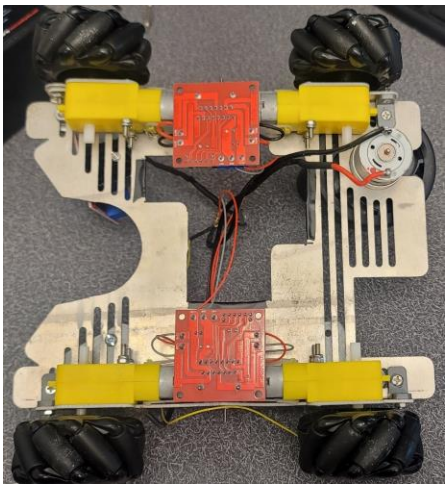


Figure 1. Bottom of Hippo bot showing TT motors connected to their motor drivers (red board) and mecanum wheels.

In order to collect balls, we decided to use a vacuum to suck up any ball in front of our bot. This vacuum was located at the back of our bot and designed to pull the balls first to a sorting junction with a color sensor and then up into the collection tank if they were of the correct color.

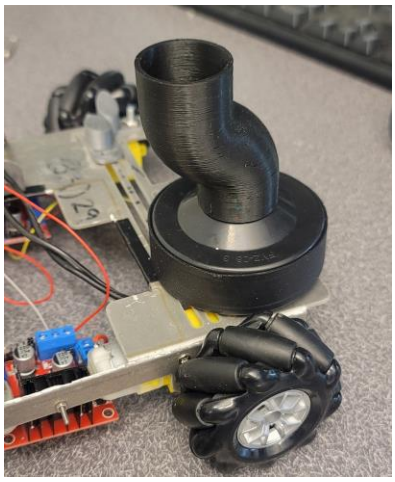


Figure 2. Vacuum with initial tubing attachment at back of bot.

To control the flow and processing of balls inside of our robot, we used 2 servo motors. At the color sensor inside, one servo arm ensured balls were stopped to have their color verified before moving onward.

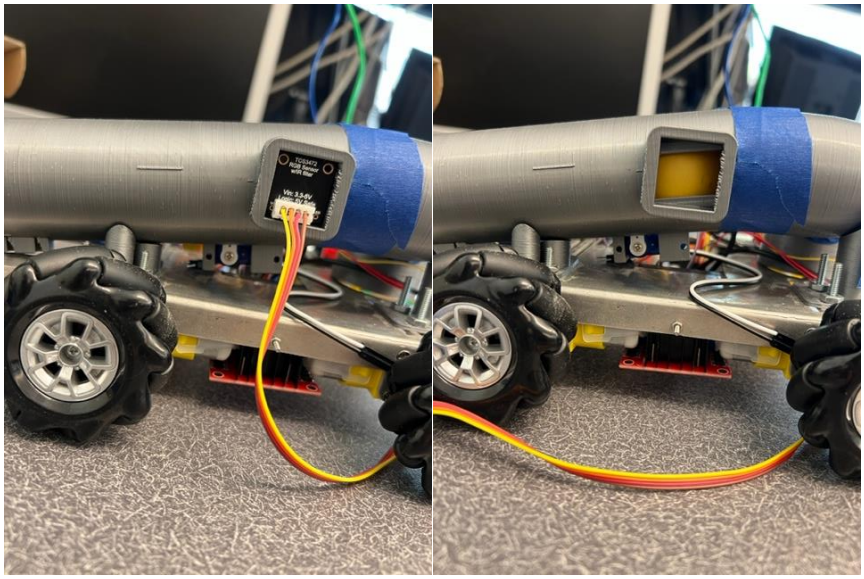


Figure 3. The vacuum line leading from the front of the robot to the rear with the color sensor (left) and with color sensor removed to show the positioning of a ball stopped at the first servo (right). In both images, the servo is visible directly under the sensor/ball, the arm of which is white.

A second servo controlled the “flapper arm,” a plastic paddle capable of blocking either the tube up into the ball collection dome or the exit tube. When blocking the tube to the dome, the vacuum no longer had pull on the ball that failed color processing, so it was necessary to angle the processing tube downwards to let gravity pull rejected balls out the back end.

Figure 4. The flapper arm blocking the exit of the processing tube. Balls of the correct color exit on the vacuum side, where a vertical tube would lead to the storage dome.

MAJOR HARDWARE – SENSORS:

While we initially intended on using a color sensor, IR sensors, motor encoders, and an accelerometer to perform line following, ball/bot/arena detection, and keep track of our bot’s location in the arena, we ended up deciding to use the overhead camera and its visual processing code – provided by Nathan – and proximity sensors for movement and location tracking. We did keep the color sensor, however, and aimed to use it to confirm that all balls collected were the correct color.

MAJOR HARDWARE – PCB:

Our PCB design consists of 4 layers: 2 interiors for GND and 3v3 Power, a top layer for the STM MCU and its necessary components, and a bottom layer for power regulation (converting a 12V input down to 5V for sensor/motor rails and down again to 3v3 for the STM and other sensors). The reason why we put components on both the top and bottom layers was to reduce the overall dimensions of the board and conserve space inside the robot. While this did yield a much smaller footprint, it made it difficult to solder components as we could not just use a template for all; we were able to use a template for the components on the top layer, but had to hand-solder the bottom layer components. We also chose to shape it like the United States for aesthetics.

After all our components came in, we realized that we were missing one of our resistors. Due to lack of time, we were unable to order a replacement. However, we were able to use one of the through-hole resistors in the mechatronics lab as a replacement; we soldered on a through-hole resistor to the SMD pads as an emergency fix.

After soldering our components to our pcb we found that the 5V and 3v3 rails were outputting the correct voltage, but the Power LED (Orange LED on the top of the board designed to indicate that power is connected) was not on. While probing the components on the power side of the board, we forgot to disconnect the power supply and accidentally caused a short in the board. Although we replaced the switching and linear regulators (the switching was shorted during probing and the linear was suspected of potentially not being entirely connected since the Power LED was not lit) and found that all power regulators were continuous (checked using the continuity feature on a multimeter), we were unable to find and correct the short in time. We suspect that either the board itself became damaged or that one of the other SMD components was shorted. Because of this, we switched back to the Blackpill for the remainder of the project. For images of the complete pcb and of its design in Fusion360 see appendix B. For details on the calculations of components for the pcb, see appendix **X**.

MAJOR HARDWARE – ROBOT DESIGN:

The robot design was primarily based around two large components, the vacuum and the battery. Great care was taken to ensure that the vacuum line used to pick up the balls, as well as affixed sensors and servos, were routed around the battery while still being enclosed by the given 250 x 250 x 300 mm envelope. The battery itself is 66 x 106 x 84 mm, taking up a huge amount of space on top of the main chassis.

The general design of the bot is to use suction to draw balls up into the processing tube, where their color will be verified. If of the correct color, the first servo will lower and allow the vacuum to further suck the ball into the storage dome. If incorrect, the flapper arm servo will rotate to block the entrance to the storage tube, and the first servo will again lower to allow the ball to exit the robot via gravity.

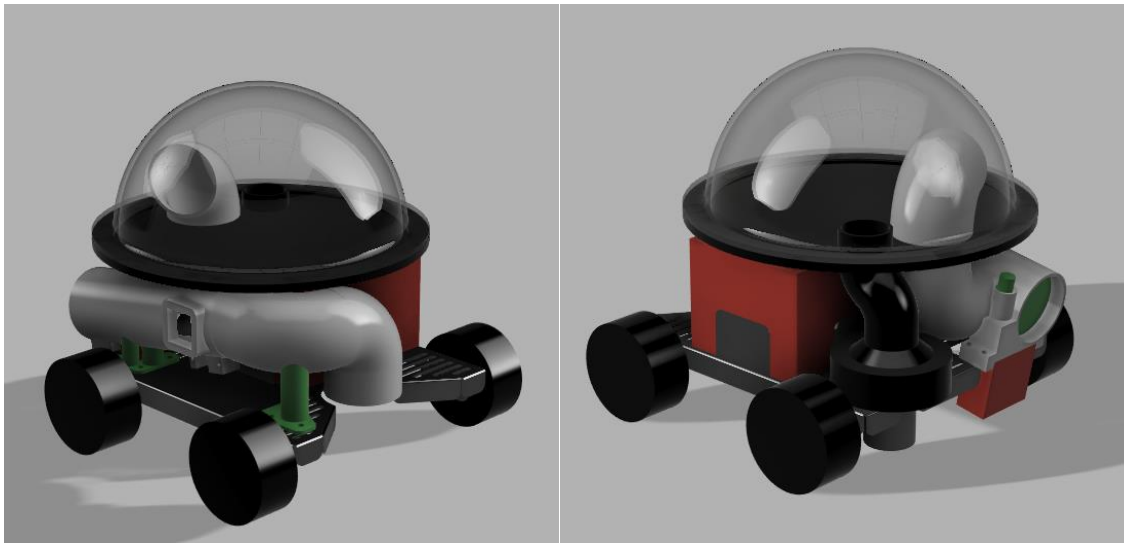


Figure 5. A rendering of the complete robot from the front right (left) and the rear left (right). The relative size of the battery (large red box) is apparent.

The frame of the robot was designed to be made out of a single sheet of metal, with numerous folds lending to the rigidity of the robot and keeping the battery from shifting.

CAL POLY

SOFTWARE – DRIVERS:

Color Sensor Driver:

The color sensor is a TCS34727, which is driven through I2C protocol. Reading from each of the three color registers (red, green, and blue), requires first writing to a master register on the device to specify which color register to pull from, then reading this value into an array on the microcontroller for processing/display etc. This driver was written by interpreting multiple functions from the Adafruit driver for Arduino into equivalent STM32 HAL, largely using the HAL functions HAL_I2C_Master_Transmit() to write to the master register (specifying color register) and HAL_I2C_Master_Receive() to read this data. Later, we discovered a driver made in STM32 HAL by ipa64, and revised our functions to mimic his style, which used the single function HAL_I2C_Mem_Read() instead of the multiline read and write functions we were calling.

Mecanum Wheels/Motor Driver:

The motor drivers used here are very similar to those written for lab 2, the notable exception being that the off-channel is held low instead of high. Apart from this, it is the same PWM setup.

The mecanum driver takes in four motor driver objects and controls all PWM simultaneously. Functions include going forward, backward, and each of the side directions. The direction of the wheels is controlled within the motor driver by either sending a positive or negative speed value, and a summary of movements for mecanum wheels is summarized below.

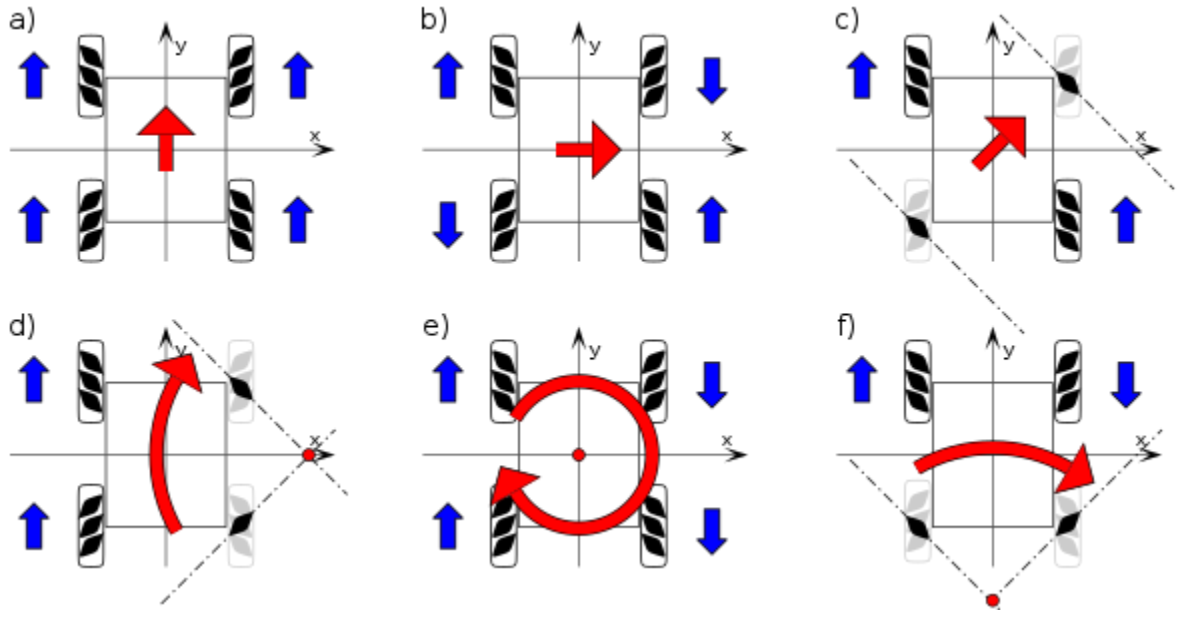


Figure 6. Mecanum wheel motion controls with wheel directions. Source: Wikipedia commons

Communication:

Because of the use of Nathan's overhead camera as our main sensor for controlling our robot, we developed two external files (one in Python using VSS and another in C++ using the Arduino IDE) to access the camera data, process it to plot a trajectory to the nearest ball of the correct color, then send commands to the Master Mind/main file on the STM MCU over first Bluetooth serial (from our laptop to the ESP32) and then through USART (from the ESP32 to STM). The code for the Bluetooth Serial communication on the ESP32 is simple and is set up to grab the characters sent to it from the laptop and echo them to the STM. This code – and the code for the camera data processor - can be seen in Appendix **X**, and a deeper dive into the data processing and its setup occurs in the Data Interpretation section.

SOFTWARE – MASTERMIND/MAIN LOOP:

For our main loop/Master Mind file in the CubeIDE, we set it up so that it first checks the proximity sensors, then reads the commands from the topCamData file on our laptop to move accordingly, and finally checks the color sensor to see if a ball has been detected.

For the proximity sensor state, we first determine which of the two front sensors has been activated, then strafe in the opposite direction to avoid collision. If both have been activated, we then wait a second before checking again (in this case it is assumed that a bot has been detected in front of ours). Ideally, this state would confirm with the upper camera whether the object being detected is the central bucket or a robot and move in a more complex manner than just strafing. However, due to time constraints, we decided it would be best just to implement a strafing or waiting mechanism to take care of most potential collisions. This state ends with a delay before continuing to check movement so as to ensure its strafing has occurred.

After the sensor has been activated, the main loop checks the character received through USART from the topCamData file and moves accordingly (either rotating, driving, or disabling motors/stopping).

At the end of this loop, the color sensor is read and, if a color is detected (by checking its level compared to the empty tube calibration value), it then determines whether it's the correct color or not. If it is, the servo used to hold the balls back is turned on to allow the ball to continue up to storage. If not, the hold servo is still called to allow it forward, but the flapper servo is also called to kick the ball out of the bot.

Because we were unable to finish putting our hippo together, and were focused primarily on that towards the end, this file is mainly a schematic; the file was created as a structure that details all the states and their conditions and specifies what occurs in each (using comments) but is not populated with the code to call the completed drivers. It does have all the pins and timers set up correctly to control each component, as well as the code for processing USART values (which has been tested and confirmed to be working correctly).

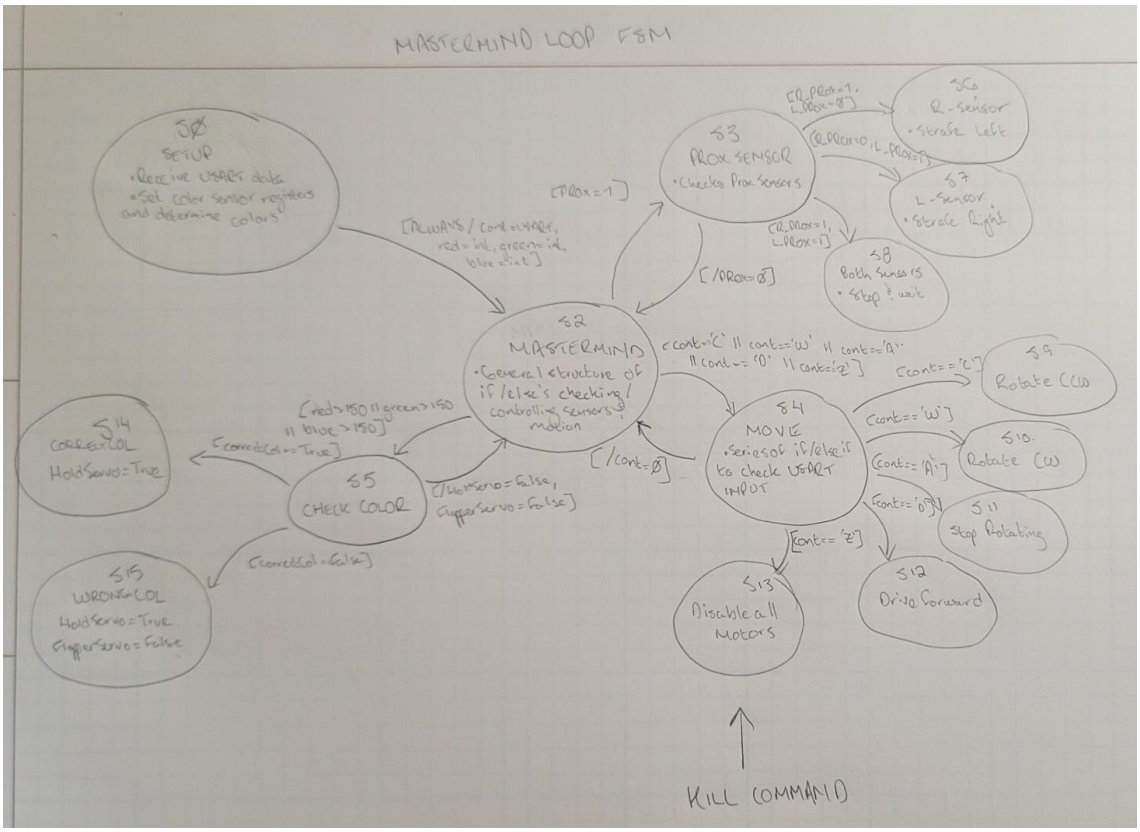


Figure . Initial FSM for Master Mind

Note: the above FSM was used as a draft of the intended implementation of Master Mind. While its structure was followed, because we were unable to complete Master Mind in time there are some slight changes in values/variable names. If we had completed it, a finalized FSM would instead be provided.

SOFTWARE – CODING STRATEGY:

Our strategy when creating our code was to first create all the necessary drivers for controlling each sensor and motor, then develop and test the code for communication and robot/ball position processing, and only then create the main loop/Master Mind file to integrate everything.

With the drivers, the main strategy was to break up its necessary functionality into small methods that could be called either in a main loop (as in the case of the topCamData file for position/trajectory processing and communication) or other methods within the driver.

For the Master Mind file, an FSM was used to plan each necessary state and the flow for controlling the hippo. Its details – as well as its FSM - are in the previous section.

DATA INTERPRETATION:

CAL POLY

Top Camera Data Processor:

In the Python file (which was run on our laptop) we first pull data from Nathan's overhead camera which gives us two lists filled with dictionaries: one for all the balls, and another for the robots. We then take this data and run it through a method named 'trajectory', which returns the distance to the nearest, correctly colored ball, the angle from our bot to the ball, the dictionary holding our bot's data, and the index in the balls list for the nearest ball. To determine the closest ball (of the desired color) we first check the color of every ball in the *balls* data list (through a for-loop) and, if it is the correct color, find the distance between our bot and it. The equation used was the vector distance equation and is below.

//insert equation here sqrt(Xhipp – ballX)^2 +(Yhipp – ballY)^2)

We store this distance value as well as the index of the ball in the *balls* list in variables *dist* and *closeball*, respectively, then repeat the process, updating these values every time a ball is found to be closer.
The next step in the method is to find the angle from our bot to ball. Because we know the coordinates of both the ball and our bot, we use the arctan function (where the numerator is the difference in Y positions and the denominator is the difference in X), then convert its radian output to degrees. The distance equation is below.

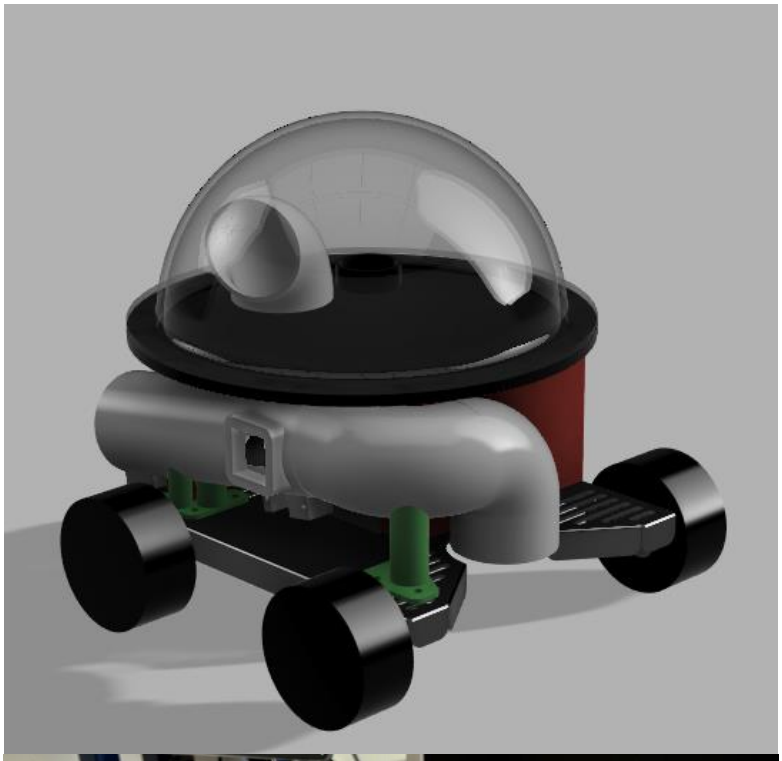
```
//reqdAngle = math.atan2((balls[closeBall]['y'] - Y_hip),(balls[closeBall]['x'] - X_hip)) #gives number b/t pi and -pi
   reqdAngle = math.degrees(reqdAngle)
```

After determining the angle from our bot to the ball, we run a method named 'angController', which returns letter values corresponding to commands for rotating the bot (sent through the ESP32 to Master Mind) until its orientation matches that of the required angle to the ball. It does so by first finding the difference between the angles, then determining if said difference is positive, negative, or zero (where positive and negative values correlate to clockwise and counterclockwise rotation commands and zero tells the bot it is in line). If the bot is orientated toward the ball, a command is then sent to the STM to have it drive forward.

An important note about this driver is that there is no command to stop the robot when it reaches the ball; once it reaches the ball and sucks it up, it should immediately find the next closest, correctly colored ball and repeat the above process.

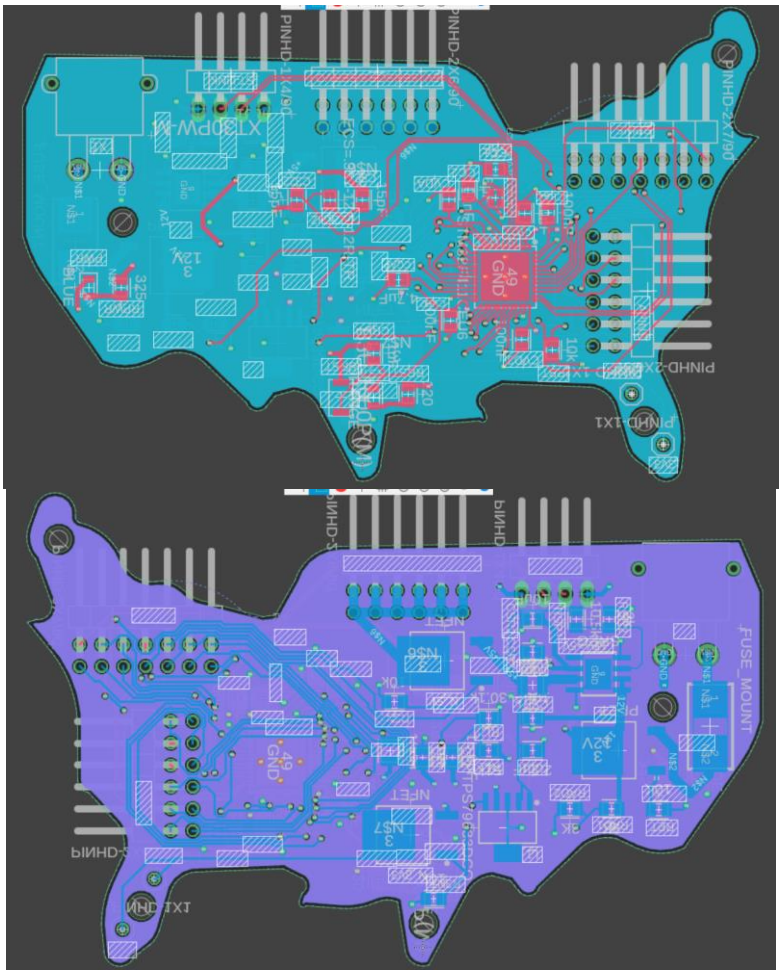
APPENDICES:

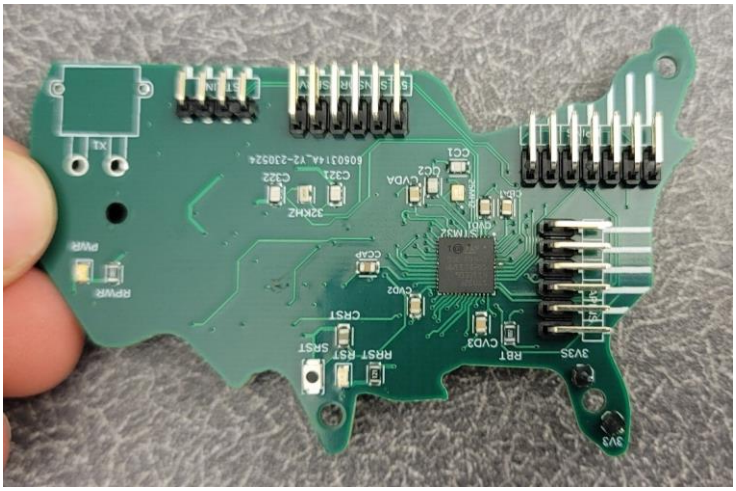
Appendix A: MECHANICAL DESIGN:

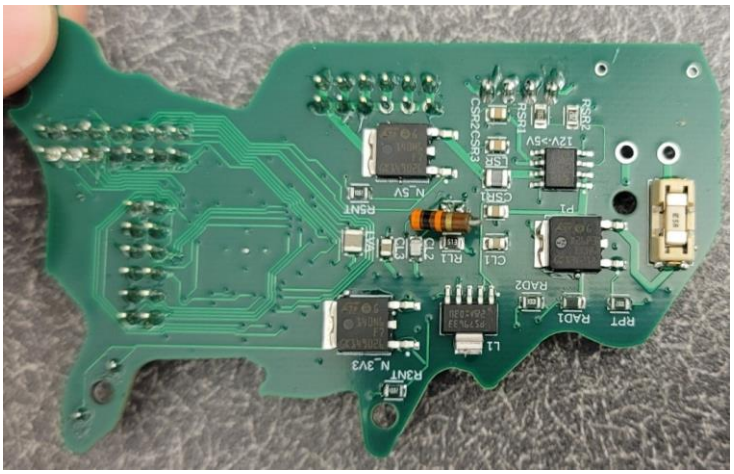


CAL POLY

Appendix B: PCB Design:






CSR2CSR3
RSR1
RSR2
12V->5V
N_5V
RSNT
CSR1
P1
RAD2
RAD1
RPT
N_3V3
RSNT

CAL POLY

Appendix C: TOP CAM DATA PROCESSING AND COMMUNICATION CODE:

TopCamData.py File:

```
import requests
import json
#from serial import Serial
import serial
import math
#import sys
import time
import threading


def topCamData():
    #Access cameras website (make sure connected to hotspot...joemama123)
    #response = requests.get('http://10.144.129.5:5000/info')
    #data = response.json()

    #Get balls and robots lists of dictionaries from data
    #balls = data['balls']
    #robots = data['robots']

    #Returns the balls list of dicts and robots list of dicts
    #return balls, robots

    balls = [{'color': 'green', 'x': 264, 'y': 218}, {'color': 'blue', 'x': 264,
'y': 218}, {'color': 'red', 'x': 280, 'y': 230},
             {'color': 'blue', 'x': 50, 'y': 70}, {'color': 'green', 'x': 200,
'y': 218}, {'color': 'yellow', 'x': 240, 'y': 28}]
    robots = [{'center': [195,180], 'decoded_info': '2', 'orientation': -90},
{'center': [26,30], 'decoded_info': '27', 'orientation': 170},
              {'center': [300,180], 'decoded_info': '5', 'orientation': 90},
{'center': [195,300], 'decoded_info': '8', 'orientation': 0}]
    return balls, robots


def trajectory(balls,robots):
#This method determines trajectory to closest ball
    for botDict in robots:
        #botDict = robots[i] #Temp storage of each dictionary in robots list
        if(botDict['decoded_info'] == '27'): #If the 'decoded_info' key equals
our bots number, 27, its our hippo
            hippo = botDict #robots[i]
```

```
            X_hip, Y_hip = hippo['center']
            Ang_hip = hippo['orientation']

    dist = 10000
    i = 0
    closeBall = 0
    for ballDict in balls:
        #calculates the vector dist between each ball and hippo, then records
shortest length and index of closest
        '''ballX = balls[j]['x']
        ballY = balls[j]['y']
        ballCol = balls[j]['color']'''
        ballX = ballDict['x']
        ballY = ballDict['y']
        ballCol = ballDict['color']
        if ballCol == 'blue': #Change to whatev color for match
            tempDist = math.sqrt((X_hip-ballX)**2 + (Y_hip-ballY)**2)
            #compares distances to determine which is smallest/closest ball and
its index
            if(tempDist<dist):
                dist = tempDist
                closeBall = i

        i = i + 1

    reqdAngle = math.atan2((balls[closeBall]['y'] - Y_hip),(balls[closeBall]['x']
- X_hip)) #gives number b/t pi and -pi
    reqdAngle = math.degrees(reqdAngle) #should convert radian to
degrees...hopefully

    return dist, reqdAngle, hippo, closeBall

    #Find angle between each bot and hippo, determine if that's similar to the
reqd angle of ball within a few degrees, then
    #extend it in the direction of the bots orientation to account for intended
trajectory


'''def botCheck(robots, hippo, dist, reqdAngle):
#checks to see if any bots are in the way of our trajectory. Finish last/if have
enough time
    X_hip, Y_hip = hippo['center']
```

```
    for k in robots:
        if(robots[k]['decoded_info'] != '27'):
            Xbot,Ybot = robots[k]['center']
            botToBot = math.sqrt((X_hip-Xbot)**2 + (Y_hip-Ybot)**2)
            botAng = math.degrees(math.atan2((Ybot - Y_hip),(Xbot - X_hip)))
            if(reqdAngle>=(botAng-5) or reqdAngle<=(botAng+5)): #check this logic
dumbass
                if(botToBot<=dist):
                    #SHIEET BRA WE GOT A BOT IN THE WAY
                    time.delay(1000) #don't have enough time to calculate
trajectory, so wait for it to move


        #also put in code for going around center bucket

def arenaBounds()'''


def angController(reqdAngle, hippo):
#P controller for determining if angle of bot matches redq angle
    Ang_hip = hippo['orientation']
    diffAng = reqdAngle - Ang_hip
    diffAng = diffAng*10
    if(diffAng<0): #reqdAngle < Ang_hip
        #tell bot to rotate towards angle
        return 'C' #C = CCW
    elif(diffAng>0): #reqdAngle > Ang_hip
        return 'W' #W = CW
    else:
        return 'A' #A = At angle, stop rotating


'''def distController(hippo, atAng, closeBall, balls):
#P controller for checking dist from bot to ball and making sure is decreasing
#may not use this and instead will just drive towards ball until picked up
    X_hip, Y_hip = hippo['center']
    ballX = balls[closeBall]['x']
    ballY = balls[closeBall]['y']
    diffDist = math.sqrt((X_hip-ballX)**2 + (Y_hip-ballY)**2)
    if(diffDist > 0 and atAng == 'A'):
        return 'D' #D = Drive
    else:
```

```
        return 'B' #B = at ball, stop driving
    '''


def main():
    #Set up Serial Comms [use BT connected Port]
    #ser = serial.Serial()
    #ser.baudrate = 115200
    #ser.port = 'COM17' #Rename to COM whatever the BT COM number is
    #ser.open()


    #ser.write(str(ColorInput).encode('ascii'))
    #ser.write(b'B')

    #gets initial posisition of hippo and records it so it can later go home
    sepuku = True
    home = True
    ballData, botData = topCamData()
    dist, reqdAngle, hippo, closeBall = trajectory(ballData, botData)
    print(closeBall)
    col = ballData[closeBall]['color']
    print(col)
    X_home,Y_home = hippo['center']
    Ang_home = hippo['orientation']

    #continues to run until KILL command inserted
    while(sepuku==True):
        try:
            #gets balls and robots location data for processing, then finds
nearest ball
            balls, robots = topCamData()
            dist, reqdAngle, hippo, closeBall = trajectory(balls, robots)
            #print(dist)
            #print(reqdAngle)

            #try:
            #calculate trajectory to nearest ball and send driving commands
            command = angController(reqdAngle, hippo)
            #print(command)
            if(command == 'A'):
                #command = distController(hippo, command, closeBall, balls)
```

```
                command = 'D'
                print(command)
                #ser.write(str(command).encode('ascii'))
        except KeyboardInterrupt:
            print("Interrupt")
            userInput = input()

            if(userInput == 'K'):
                #Exits while loop and kills all motors
                sepuku = False
                command = 'Z' #Z = kill all motors
                print(command)
            elif(userInput == 'H'):
            #sendd bot home and exits while loop and kills motors if at home
                while(home == True):
                    X_hip,Y_hip = hippo['center']
                    homeAngle = math.atan2((Y_home - Y_hip),(X_home - X_hip))
#gives number b/t pi and -pi
                    homeAngle = math.degrees(reqdAngle)
                    command = angController(homeAngle, hippo)
                    if(command == 'A' and X_hip != X_home and Y_hip != Y_home):
                        command = 'D'
                    elif(X_hip == X_home and Y_hip == Y_home):
                        sepuku = False
                        command = 'Z'
                        home = False
                    print(command)
                    #ser.write(str(command).encode('ascii'))

    '''#gets user input if any entered to check if 'KILL' or 'HOME' entered
        userInput = input()

        if(userInput):
            if(userInput == 'KILL'):
                #Exits while loop and kills all motors
                sepuku = False
                command = 'Z' #Z = kill all motors
            elif(userInput == 'HOME'):
            #sendd bot home and exits while loop and kills motors if at home
                X_hip,Y_hip = hippo['center']
                homeAngle = math.atan2((Y_home - Y_hip),(X_home - X_hip)) #gives
number b/t pi and -pi
```

```
                homeAngle = math.degrees(reqdAngle)
                command = angController(homeAngle, hippo)
                if(command == 'A' and X_hip != X_home and Y_hip != Y_home):
                    command = 'D'
                elif(X_hip == X_home and Y_hip == Y_home):
                    sepuku = False
                    command = 'Z'
            else:
                userInput = False
        else:
            #calculate trajectory to nearest ball and send driving commands
            command = angController(reqdAngle, hippo)
            if(command == 'A'):
                #command = distController(hippo, command, closeBall, balls)
                command = 'D' '''

    '''#gets balls and robots location data for processing, then finds nearest
ball
        balls, robots = topCamData()
        dist, reqdAngle, hippo, closeBall = trajectory(balls, robots)
        print(dist)
        print(reqdAngle)

        if(userInput == 'KILL'):
            #Exits while loop and kills all motors
            sepuku = False
            command = 'Z' #Z = kill all motors
        elif(userInput == 'HOME'):
            #sendd bot home and exits while loop and kills motors if at home
            X_hip,Y_hip = hippo['center']
            homeAngle = math.atan2((Y_home - Y_hip),(X_home - X_hip)) #gives
number b/t pi and -pi
            homeAngle = math.degrees(reqdAngle)
            command = angController(homeAngle, hippo)
            if(command == 'A' and X_hip != X_home and Y_hip != Y_home):
                command = 'D'
            elif(X_hip == X_home and Y_hip == Y_home):
                sepuku = False
                command = 'Z'
        else:
            #calculate trajectory to nearest ball and send driving commands
            command = angController(reqdAngle, hippo)
```

```
            if(command == 'A'):
                #command = distController(hippo, command, closeBall, balls)
                command = 'D' '''

        #send command for driving to blackpill mastermind
        #print(command)
        #ser.write(str(command).encode('ascii'))


if __name__ == '__main__':
        print('running')
        main()
        #print('running')
```

ESP32 BT Serial to USART code:

```
//This example code is in the Public Domain (or CC0 licensed, at your option.)
//By Evandro Copercini - 2018
//
//This example creates a bridge between Serial and Classical Bluetooth (SPP)
//and also demonstrate that SerialBT have the same functionalities of a normal
Serial

#include "BluetoothSerial.h"
#include <HardwareSerial.h>

#define RX0 12
#define TX0 13
HardwareSerial SerialPort(0);

#define USE_PIN // Uncomment this to use PIN during pairing. The pin is specified
on the line below
const char *pin = "8008"; // Change this to more secure PIN.
//#define STM32_Pin 12
//#define testPin 13
String command;
String device_name = "Fuck Me";

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif
```

```
#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Bluetooth not available or not enabled. It is only available for
the ESP32 chip.
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin(device_name); //Bluetooth device name
  Serial.printf("\"%s\" is started.\nPair it with Bluetooth!\n",
device_name.c_str());
  //Serial.printf("The device with name \"%s\" and MAC address %s is
started.\nNow you can pair it with Bluetooth!\n", device_name.c_str(),
SerialBT.getMacString()); // Use this after the MAC method is implemented
  #ifdef USE_PIN
    SerialBT.setPin(pin);
    Serial.println("Using PIN");
  #endif

  SerialPort.begin(115200,SERIAL_8N1,RX0,TX0);
}

void loop() {
  // Read received messages (LED control command)
  if (SerialBT.available()){
    char phoneChar = SerialBT.read();
    if (phoneChar != '\n'){
      command += String(phoneChar);
    }
    else{
      command = "";
    }
    Serial.write(phoneChar);
  }
  // Check received message and control output accordingly
  /*if (command =="stop" || command == "Stop" || command == "STOP"){
    digitalWrite(STM32_Pin, HIGH);
  }
  else if (command =="test" || command == "Test" || command == "TEST"){
    digitalWrite(testPin, HIGH);
  }*/
```

```
  //Serial.println(command);
  //SerialPort.write('B');
  SerialPort.print(command);
  delay(100);
  Serial.println(SerialPort.read());
  delay(1000);

}
```

CAL POLY

Appendix D: Drivers

RGB Driver:

Header File:

```
/*
 * rgb_driver.h
 *
 *  Created on: May 30, 2023
 *      Author: Trevor Foley
 */

#include "stdint.h"
#include "stm32f4xx_hal.h"

#ifndef INC_RGB_DRIVER_H_
#define INC_RGB_DRIVER_H_

typedef struct rgb_sensor *RGBPtr;
typedef struct rgb_sensor {
      I2C_HandleTypeDef *hi2c;
} RGB;

void ConstructRGBSensor(RGBPtr sensor, I2C_HandleTypeDef *hi2c);

uint16_t ReadColorRegister(RGBPtr sensor, uint8_t reg);

void GetRawColors(RGBPtr sensor, uint16_t* red, uint16_t* green, uint16_t*
blue, uint16_t* clear);

void GetColors(RGBPtr sensor, uint16_t* r, uint16_t* g, uint16_t* b);

void SetControlRegister(RGBPtr sensor, uint8_t reg, uint8_t value);

uint8_t ReadControlRegister(RGBPtr sensor, uint8_t reg);

#endif /* INC_RGB_DRIVER_H_ */
```

CAL POLY

Class File:

```
/*
 * rgb_driver.c
 *
 *  Created on: May 30, 2023
 *      Author: Trevor Foley
 */

#include "rgb_driver.h"

void ConstructRGBSensor(RGBPtr sensor, I2C_HandleTypeDef *hi2c)
{
	sensor->hi2c = hi2c;
	SetControlRegister(sensor, 0x00, 0x01);
}

void SetControlRegister(RGBPtr sensor, uint8_t reg, uint8_t value)
{
	uint8_t data = value;
	HAL_I2C_Mem_Write(sensor->hi2c, (uint16_t)(0x29<<1), 0x80 | reg, 1,
&data, 1, 100);
}

uint8_t ReadControlRegister(RGBPtr sensor, uint8_t reg)
{
	uint8_t data;
	HAL_I2C_Mem_Read(sensor->hi2c, (uint16_t)(0x29<<1), 0x80 | reg, 1,
&data, 1, 100);
	return data;
}

uint16_t ReadColorRegister(RGBPtr sensor, uint8_t reg)
{
	uint16_t datal, datah;
	uint8_t data[2];
	HAL_I2C_Mem_Read(sensor->hi2c, (uint16_t)(0x29<<1), 0x80 | reg, 1,
data, 2, 100);
	datal = (uint16_t) data[0];
	datah = (uint16_t) data[1];
	datah <<= 8;
	return datah | datal;
}

void GetRawColors(RGBPtr sensor, uint16_t* red, uint16_t* green, uint16_t*
blue, uint16_t* clear)
{
	*red = ReadColorRegister(sensor, 0x16);
	*green = ReadColorRegister(sensor, 0x18);
	*blue = ReadColorRegister(sensor, 0x1A);
	*clear = ReadColorRegister(sensor, 0x14);
	HAL_Delay((256 - 0xEB) * 12 / 5 + 1);
}
```

CAL POLY

```
void GetColors(RGBPtr sensor, uint16_t* r, uint16_t* g, uint16_t* b)
{
	uint16_t red, green, blue, clear;
	GetRawColors(sensor, &red, &green, &blue, &clear);
	uint32_t sum = clear;

	if (clear == 0) {
		*r = *g = *b = 0;
		return;
	}

	*r = red*255/sum;
	*g = green*255/sum;
	*b = blue*255/sum;
}
```

CAL POLY

Motor Driver

Header File:

```
/*
 * motor_driver.h
 *
 *  Created on: Apr 27, 2023
 *      Author: Trevor Foley
 */
#include "stdint.h"
#include "stm32f4xx_hal.h"

#ifndef INC_MOTOR_DRIVER_H_
#define INC_MOTOR_DRIVER_H_

typedef struct motor *MotorPtr;
typedef struct motor {
      int32_t duty_cycle;
      uint32_t forward_channel;
      uint32_t reverse_channel;
      TIM_HandleTypeDef *htim;
} Motor;

void ConstructMotor(MotorPtr motor, TIM_HandleTypeDef *htim, uint32_t
forward, uint32_t reverse);

void EnableMotor(MotorPtr motor);

void CrippleMotor(MotorPtr motor);

void SetDutyCycle(MotorPtr motor, int32_t duty);

#endif /* INC_MOTOR_DRIVER_H_ */
```

CAL POLY

Class File:

```
/*
 * motor_driver.c
 *
 *  Created on: Apr 27, 2023
 *      Author: Trevor Foley
 */

#include "motor_driver.h"

void ConstructMotor(MotorPtr motor, TIM_HandleTypeDef *htim, uint32_t
forward, uint32_t reverse)
{
	motor->htim = htim;
	motor->forward_channel = forward;
	motor->reverse_channel = reverse;
	SetDutyCycle(motor, 0);
}

void EnableMotor(MotorPtr motor)
{
	HAL_TIM_PWM_Start(motor->htim, motor->forward_channel);
	HAL_TIM_PWM_Start(motor->htim, motor->reverse_channel);
}

void CrippleMotor(MotorPtr motor)
{
	HAL_TIM_PWM_Stop(motor->htim, motor->forward_channel);
	HAL_TIM_PWM_Stop(motor->htim, motor->reverse_channel);
}

void SetDutyCycle(MotorPtr motor, int32_t duty)
{
	if (duty > 0 && duty < 100){
		__HAL_TIM_SET_COMPARE(motor->htim, motor->forward_channel,
duty*47);
		HAL_TIM_PWM_Stop(motor->htim, motor->reverse_channel);
	}
	else if (duty < 0 && duty > -100){
		__HAL_TIM_SET_COMPARE(motor->htim, motor->reverse_channel, duty*-
47);
		HAL_TIM_PWM_Stop(motor->htim, motor->forward_channel);
	}
	else {
		HAL_TIM_PWM_Stop(motor->htim, motor->reverse_channel);
		HAL_TIM_PWM_Stop(motor->htim, motor->forward_channel);
	}
}
```

CAL POLY

Mecanum Driver

Header File:

```
/*
 * mecanum_driver.h
 *
 *  Created on: Jun 14, 2023
 *      Author: Trevor Foley
 */

#include "stdint.h"
#include "stm32f4xx_hal.h"
#include "motor_driver.h"

#ifndef INC_MECANUM_DRIVER_H_
#define INC_MECANUM_DRIVER_H_

typedef struct chassis *ChassisPtr;
typedef struct chassis {
      MotorPtr front_right;
      MotorPtr front_left;
      MotorPtr back_left;
      MotorPtr back_right;
} Chassis;

void ConstructChassis(ChassisPtr chassis, MotorPtr motor1, MotorPtr motor2,
MotorPtr motor3, MotorPtr motor4);

void EnableChassis(ChassisPtr chassis);

void CrippleChassis(ChassisPtr chassis);

void GoForward(ChassisPtr chassis);

void GoBackward(ChassisPtr chassis);

void GoLeft(ChassisPtr chassis);

void GoRight(ChassisPtr chassis);

#endif /* INC_MECANUM_DRIVER_H_ */
```

CAL POLY

Class File:

```
/*
 * mecanum_driver.c
 *
 *  Created on: Jun 14, 2023
 *      Author: Trevor Foley
 */

#include "mecanum_driver.h"

void ConstructChassis(ChassisPtr chassis, MotorPtr motor1, MotorPtr motor2,
MotorPtr motor3, MotorPtr motor4)
{
	chassis->front_right = motor1;
	chassis->back_right = motor2;
	chassis->front_left = motor3;
	chassis->back_left = motor4;
}

void EnableChassis(ChassisPtr chassis)
{
	EnableMotor(chassis->front_right);
	EnableMotor(chassis->front_left);
	EnableMotor(chassis->back_left);
	EnableMotor(chassis->back_right);
}

void CrippleChassis(ChassisPtr chassis)
{
	CrippleMotor(chassis->front_right);
	CrippleMotor(chassis->front_left);
	CrippleMotor(chassis->back_left);
	CrippleMotor(chassis->back_right);
}

void GoForward(ChassisPtr chassis)
{
	SetDutyCycle(chassis->front_right, 35);
	SetDutyCycle(chassis->front_left, 35);
	SetDutyCycle(chassis->back_left, 35);
	SetDutyCycle(chassis->back_right, 35);
}

void GoBackward(ChassisPtr chassis)
{
	SetDutyCycle(chassis->front_right, -35);
	SetDutyCycle(chassis->front_left, -35);
	SetDutyCycle(chassis->back_left, -35);
	SetDutyCycle(chassis->back_right, -35);
}

void GoLeft(ChassisPtr chassis)
```

CAL POLY

```
{
	SetDutyCycle(chassis->front_right, 35);
	SetDutyCycle(chassis->front_left, -35);
	SetDutyCycle(chassis->back_left, 35);
	SetDutyCycle(chassis->back_right, -35);
}

void GoRight(ChassisPtr chassis)
{
	SetDutyCycle(chassis->front_right, -35);
	SetDutyCycle(chassis->front_left, 35);
	SetDutyCycle(chassis->back_left, -35);
	SetDutyCycle(chassis->back_right, 35);
}
```

CAL POLY

Servo Driver

Header File

```c
/*
 * servo_driver.h
 *
 *  Created on: May 30, 2023
 *      Author: Trevor Foley
 */

#include "stdint.h"
#include "stm32f4xx_hal.h"

#ifndef INC_SERVO_DRIVER_H_
#define INC_SERVO_DRIVER_H_

typedef struct servo *ServoPtr;
typedef struct servo {
	int32_t position;
	uint32_t drive_channel;
	TIM_HandleTypeDef *htim;
} Servo;

void ConstructServo(ServoPtr servo, TIM_HandleTypeDef *htim, uint32_t
drive_channel);

void GoToPosition(ServoPtr servo, int32_t position);

#endif /* INC_SERVO_DRIVER_H_ */
```

CAL POLY

Class File:

```
/*
 * servo_driver.c
 *
 *  Created on: May 30, 2023
 *      Author: Trevor Foley
 */

#include "servo_driver.h"

void ConstructServo(ServoPtr servo, TIM_HandleTypeDef *htim, uint32_t
drive_channel)
{
	servo->htim = htim;
	servo->drive_channel = drive_channel;
	GoToPosition(servo, 0);
}

void GoToPosition(ServoPtr servo, int32_t position)
{
	__HAL_TIM_SET_COMPARE(servo->htim, servo->drive_channel,
(int)(position/1.8 + 50));
}
```

Appendix E: Main Code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "mecanum_driver.h"
#include "servo_driver.h"
```

```
#include "rgb_driver.h"
#include <stdio.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------
---------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------
---------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------
---------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------
---------------*/
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
```

```
char cont;
char temp;
int red, green, blue;
char rbuffer [50] = {0};
char gbuffer [50] = {0};
char bbuffer [50] = {0};
char buffer [10] = {0};
/* USER CODE END PV */

/* Private function prototypes -------------------------------
---------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM4_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----------------------------------------
---------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
```

```
{
  /* USER CODE BEGIN 1 */
  RGB rgb1;
  RGBPtr rgbSensor1 = &rgb1;
  ConstructRGBSensor(rgbSensor1, &hi2c1);

  uint16_t red, green, blue, time;

  Servo s1;
  ServoPtr servo1 = &s1;
  ConstructServo(servo1, &htim4, TIM_CHANNEL_1);

  Servo s2;
  ServoPtr servo2 = &s1;
  ConstructServo(servo2, &htim4, TIM_CHANNEL_2);

  Motor mot1;
  MotorPtr motor1 = &mot1;
  ConstructMotor(motor1, &htim2, TIM_CHANNEL_1, TIM_CHANNEL_2);

  Motor mot2;
  MotorPtr motor2 = &mot2;
  ConstructMotor(motor2, &htim2, TIM_CHANNEL_3, TIM_CHANNEL_4);

  Motor mot3;
  MotorPtr motor3 = &mot3;
  ConstructMotor(motor3, &htim3, TIM_CHANNEL_1, TIM_CHANNEL_2);

  Motor mot4;
  MotorPtr motor4 = &mot4;
  ConstructMotor(motor4, &htim3, TIM_CHANNEL_3, TIM_CHANNEL_4);

  Chassis chas;
```

```
  ChassisPtr chassis = &chas;
  ConstructChassis(chassis, motor1, motor2, motor3, motor4);
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------
---------------*/

  /* Reset of all peripherals, Initializes the Flash interface
and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_USART1_UART_Init();
  MX_TIM4_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  char msg_buff[100] = {0};
  int32_t len = sprintf(msg_buff,"EVERYTHING IS ON");
  HAL_UART_Transmit(&huart1, (uint8_t*) msg_buff, len, 10000);
```

```
//Code to turn vaccuum on here
//Code to turn Hold Sensor on here

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_UART_Receive_IT(&huart1,(uint8_t*)&temp,1);

  if(prox sensor left is high or prox sensor right is high){
    if(left sensor == high and right sensor == low){
      //Call motor driver and strafe right
      GoRight(chassis);
    }else if(right sensor == high and left sensor == low){
      //Call motor driver and strafe left
      GoLeft(chassis);
    }else{
      //Call motor driver and stop bot
      CrippleChassis(chassis);
    }
  }

  //Checking for motor commands
  else if(cont == 'C'){
    //Call MotorDriver and Rotate CCW
    cont = '0';
  }else if(cont == 'W'){
    //Call MotorDriver and Rotate CW
    cont = '0';
  }else if(cont == 'A'){
```

```
      //Call MotorDriver and have hippo stop rotating; stop bot
      CrippleChassis(chassis);
      cont = '0';
    }else if(cont == 'D'){
      //Call MotorDriver and have hippo drive (forward)
      GoForward(chassis);
      cont = '0';
    }else if(cont == 'Z'){
      //turn off all power fets, or at least disable all motors
and stop bot
      //turn off vaccuum here
      CrippleChassis(chassis);
      //HAL_GPIO_WritePin(pin connected to vacuum MOSFET)
    }

    //checking color sensor
    SetControlRegister(rgbSensor1, 0x00, 0x01);
    HAL_Delay(50);
    SetControlRegister(rgbSensor1, 0x00, 0x03);
    SetControlRegister(rgbSensor1, 0x01, 0xEB);
    SetControlRegister(rgbSensor1, 0x0F, 0x00);
    HAL_Delay(50);

    GetColors(rgbSensor1, &red, &green, &blue);
    //compare color registers
    //psuedo-code, use actual registers once color is assigned
    if(color sensor == non-gray color){
      //determine ball color
      if(color == our color){
        //Call Hold Servo and turn off
        GoToPosition(servo1, 0);
      }else{
        //Call Flapper Servo and turn on
```

```
        GoToPosition(servo2, 90);
        //Call Hold Servo and turn off
        GoToPosition(servo1, 0);
      }
    }
    if(exit proximity sensor == high){
      //ball has exited the robot
      GoToPosition(servo1, 90);
      GoToPosition(servo2, 0);
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1)
;
```

```
  /** Initializes the RCC Oscillators according to the
specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 192;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYP
E_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3)
!= HAL_OK)
  {
    Error_Handler();
  }
```

```c
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
```

```
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 4799;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
```

```
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_2);
```

```
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_4);
  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 4799;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /* USER CODE BEGIN TIM3_Init 2 */
  HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4);
  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 1919;
```

```
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */
  HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_2);
  /* USER CODE END TIM4_Init 2 */
  HAL_TIM_MspPostInit(&htim4);
```

```c
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */
```

```
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
```

```
  /* Prevent unused argument(s) compilation warning */
  if(huart == &huart1){
  HAL_UART_Receive_IT(&huart1,(uint8_t*)&temp,1);
  HAL_UART_Transmit_IT(&huart1,(uint8_t*)&temp,1);
  cont = temp;
  /*if(temp == ''){
    toggle = true;
  }*/
  }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error
occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL
error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source
line number
```

```
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file
name and line number,
     ex: printf("Wrong parameters value: file %s on line
%d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Appendix F: PCB COMPONENT SELECTION CALCS/DETAILS:

Component:	**Voltage:**	**Current:**	**Quantity:**	**Total Current:**
Motor [TT]	5V	500mA (Nom), ~1.3A @stall Torque	4	2A (Expected)
Motor Driver [Qunqi L298N]	5V	12V Vdd, 5V Logic	2 (Each can drive 2 motors)	2A (Expected)
Servo [MG995]	5V	10mA (Idle), 170mA (No-Load), 1200mA (Max)	1-2	250mA (Expected)/per Servo
IR Sensor [Gikfun EK1254]	5V	<500mA; 100mA (Estimate)	4	400mA (Estimate)

Color Sensor []	5V	15mA	1	15mA
Photo Interrupter Encoder []				
Vacuum []	12V	10A	1	10A
Radio Receiver			1	

Rail:					
Direct off Battery [12V]		**5V**		**3.3V**	
Component:	**Current:**	**Component:**	**Current:**	**Component:**	**Current:**
Motor Drivers & Motors	2A	Servo	250mA		
Vacuum	10A	IR Sensor	400mA		
		Color Sensor	15mA		
Total I:	12A	**Total I:**	~1A	**Total I:**	<250mA (Assumption)

Direct off Battery:

These components will be wired OFF THE PCB BOARD; do not need to account for them on the board. DO need to account for their GPIO pins and NFETs.

5V Rail:

This rail is on the PCB board so the total current the sensors and servos that connect to it use MUST BE ACCOUNTED FOR.

3.3V Rail:

Since there will only be 1-2 sensors connected to this (and with low current) it SHOULD NOT NEED CURRENT TO BE SCALED UP MUCH.

NOTE: THESE VALUES/TOTALS ARE BASED ON THE ASSUMPTION THAT ALL COMPONENTS ARE RUNNING AT ONCE; IDEALLY THIS SHOULD NEVER OCCUR.

FUSE SELECTION:

- Actual Current (MAX; EVERYTHING ON): 2A
- Current Rating = Actual Current/.75 = 2A/.75 = 2.67A
- Chose a 3A fuse holder and fuse from Digikey:
    - Fuse Holder: https://www.digikey.com/en/products/detail/littelfuse-inc/0154003-DRL/2520361
    - Fuse: https://www.digikey.com/en/products/detail/littelfuse-inc/0451003-NRL/2023596

5V RAIL NFET SELECTION:

- Drain to Source Voltage: >5V
- Gate Voltage: 3.3V (MCU logic voltage)
- Current Through: >2A
- **USED SAME NFET AS THE ONE BEING USED FOR VACUUM AND 5V MOTORS…**

ADC:

- Input Voltage: 3V
    - Used Resistor Divider circuit:
    - R2/(R1+R2) = ¼; R2=3k; R1=9k;
    - Wanted higher resistors so lower current
- Using ADC1_0

SWITCHING REGULATOR CHOICE AND SETUP:

- 12V -> 5V, 2A
- Chose ST1S41PHR
    - IC Reg Buck Adjustable
    - 4-18V Input
    - 0.8-18V Output
    - 4A Output Current

- -40C ~150C Operating Temp (Tj)

Dmin = Voutmin/Vinmax = 5/12 = .417
Io = Max output Current = 4A
-> Will actually be around 2A max
delta_I_L = .25*Io = 1A
Vpp_max = .01*Vinmax = .01*18V
-> Specified Cin usually dimensioned to keep max peak-peak V ripple in order of 1% Vinmax
-> Max Operating Vin = 18V; will be 12V though
Fsw = 0.85MHz (Typ)
-> 0.7<Fsw<1

Input Capacitor Selection:
Using eff (n) = 1, and max Duty (D) of 0.5:
Cin_min = Io/(2*Vpp_max*Fsw) [Eq 18]

Cin_min = 2/(2*.12*.85E6) = 9.8uF
USED VALUE REPORTED IN TABLE 6 [10uF]

Inductor Selection:
Lmin = [Vout/delta_I_max]*[(1-Dmin)/Fswmin] [Eq 20]
I_L,PK = Io + delta_I_L/2 [Eq 21]

Lmin = [5/1]*[(1-.417)/.7] = 4.16uH
I_L,PK = 2 + ½ = 2.5A

USED 4.7uH FOR INDUCTOR

Output Capacitor Selection:
Delta_Vout = ESR*delta_Imax + delta_Imax/(8*Cout*fsw) [Eq 22]
ASSUMING ESR = .05 Ohm (Wikipedia says ceramic Caps b/t 0.01 and 0.1 Usually)
WANT DELTA_VOUT RIPPLE TO BE SMALL

ENDED UP USING 10uF for Cap

Figure 1. Application circuit

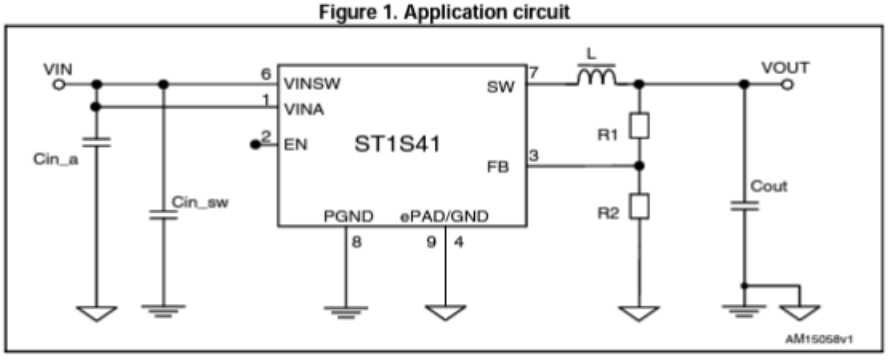
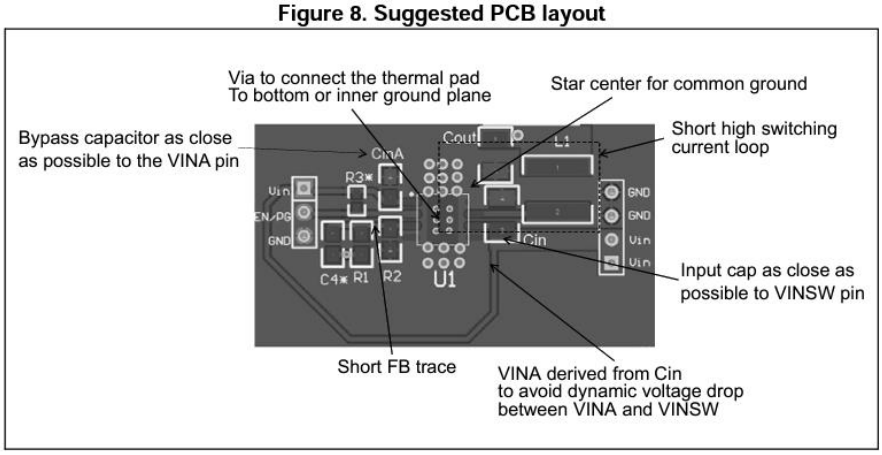
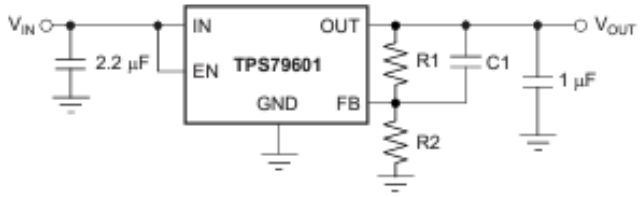


Figure 8. Suggested PCB layout



LINEAR REGULATOR CHOICE AND SETUP:

- 5v -> 3.3v, 2a
- Chose TPS79633DCQ
  - 5.5V Max Input (Our input is 5V)
  - 3.3V Output
  - 1A Output
  - -40C ~ 125C Operating Temp
  - GammaJB (Junction to Board) = 30.1 C/W
    - T = 25 + 30.1C/W * (5V-3.3V)*1A = 76.17 C
  - Found R1, R2, and C1 From equations below (detailed in data sheet)



OUTPUT VOLTAGE PROGRAMMING GUIDE

OUTPUT VOLTAGE	R1	R2	C1
1.8 V	14.0 kΩ	30.1 kΩ	33 pF
3.6 V	57.9 kΩ	30.1 kΩ	15 pF

Figure 25. Typical Application, Adjustable Output

$$R1 = \left(\frac{V_{OUT}}{V_{REF}} - 1\right) \times R2$$

$$C1 = \frac{\left(3 \times 10^{-7}\right) \times (R1 + R2)}{(R1 \times R2)}$$

CRYSTAL OSCILLATOR – 12MHZ:

25MHZ Cryst Caps:
CL = 8pF
ESTIMATE Cstray = 5pF
C1,C2 = 2*CL – 2*Cstray = 16 – 10 = 6pF

CRYSTAL OSCILLATOR – 32kHZ:

-

32kHZ Cryst. Cap.s
CL = 12.5pF
CL = (C1*C2)/(C1+C2) +Cstray
ESTIMATE Cstray = 5pF
CL - Cstray = (C1*C2)/(C1+C2)
RUT: C1,C2 = 2*CL - 2*Cstray = 25-10 = 15pF

LED RESISTOR CALCS:

ORANGE LED RESISTOR:
V=2.1V
Imax=30mA
Itest=10mA
Vdd=3.3V
Rmin = V/I = (3.3-2.1)/.03 = 40 Ohm
Rtest = 1.2/.01 = 120 Ohm [USING]

GREEN LED RESISTOR:
V=2.1V
Imax=30mA
Itest=20mA
Vdd=3.3V
Rmin= 40 Ohm
Rtest= 1.2/.02 = 60 Ohm [USING]

BLUE LED RESISTOR:
V=2.65V
Imax=30mA
Itest=2mA
Vdd=3.3V
Rmin = (3.3-2.65)/.03 = 21.7 Ohm

Rtest = .65/.002 = 325 Ohm [USING]

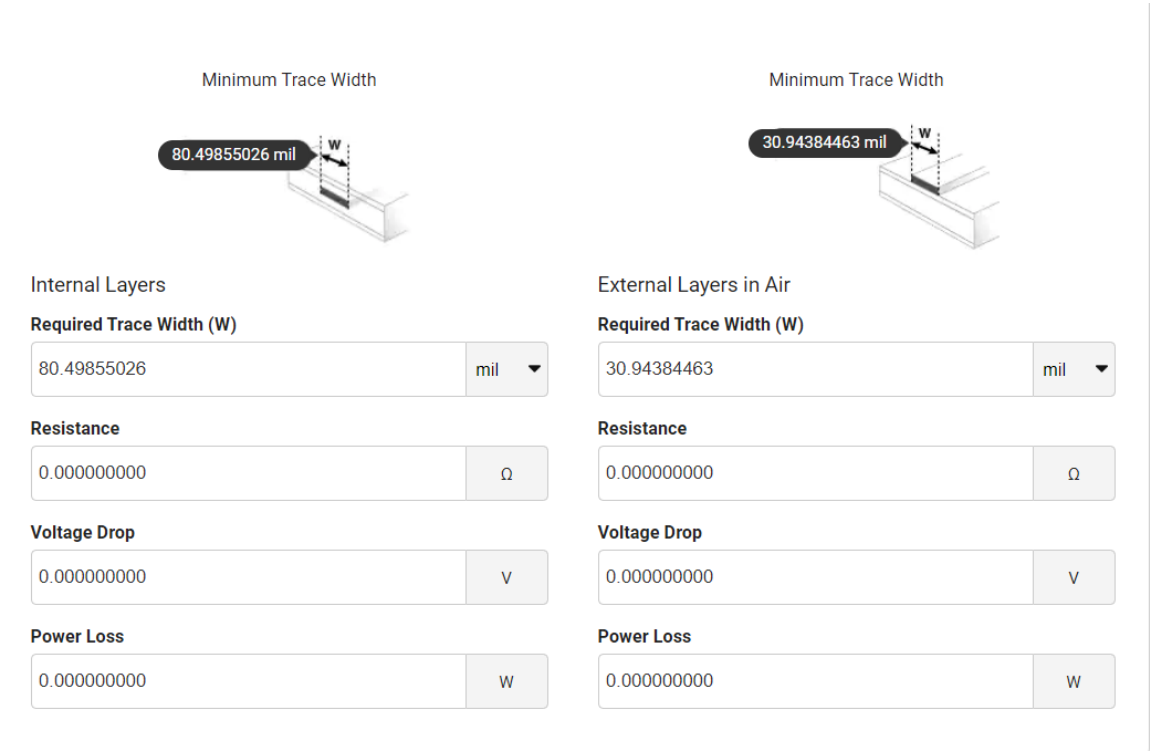
STM32 GPIO PINS:

Component Pin Info:				
Component	**Pin:**	**Timer:**	**Channel:**	**Type:**
Vacuum NFET				Analogue
Motor Drivers NFET				Analogue
	N/A	N/A	N/A	Vin – 12V
	A1	2	1	PWM A1
	A2	2	2	PWM A2
	A3	2	3	PWM B1
	A4	2	4	PWM B2
Motor Driver 1	N/A	N/A	N/A	GND
	N/A	N/A	N/A	Vin – 12V
	A5	3	1	PWM A1
	A6	3	2	PWM A2
	A7	3	3	PWM B1
	A8	3	4	PWM B2
Motor Driver 2	N/A	N/A	N/A	GND
	N/A			Vin – 5V
	A9			PWM
Servo	N/A			GND
	N/A			Vin – 5V
	A10			PWM
Servo 2	N/A			GND
Color Sensor:				

TRACE WIDTH CALCS:
CRITERIA:

- Current: 2A or .5A
- Thickness: 1.37 mil
- Temp Rise: 10C
- Ambient Temp: 25C

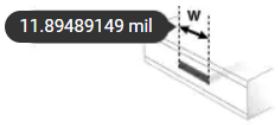
For the 2A Traces:



- Used the “External Layers in Air” values since these traces were on the external faces of the pcb (Top and Bottom layers)
- USED 40 mil FOR SECURITY

<u>For the <500mA Traces:</u>

Minimum Trace Width



Internal Layers

Required Trace Width (W)

11.89489149 mil

Resistance

0.000000000 Ω

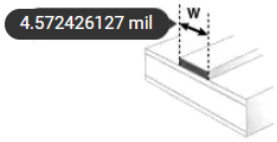
Voltage Drop

0.000000000 V

Power Loss

0.000000000 W

Minimum Trace Width



External Layers in Air

Required Trace Width (W)

4.572426127 mil

Resistance

0.000000000 Ω

Voltage Drop

0.000000000 V

Power Loss

0.000000000 W

- Used “External Layers in Air” values
- USED 10 – 20 mil TO BE CONSERVATIVE