ME102B Final Project Report: Monkeybot

Project Opportunity

For our final project for ME102 Fall 2022, we wanted to create a robotic monkey that could climb the monkey bars. We chose a robot that can swing on monkey bars by mimicking the pendulum-type swing that primates employ to advance on tree branches. Based on our weighted matrix and group discussions, we agreed this project would be the best intersection between enthusiasm and feasibility, since the brachiating movement we are designing provides a fun challenge while being easily scalable. Non-flat terrain and above-air projects particularly stood out to us, and we found an opportunity to implement biomimicry with a swinging monkeybot.

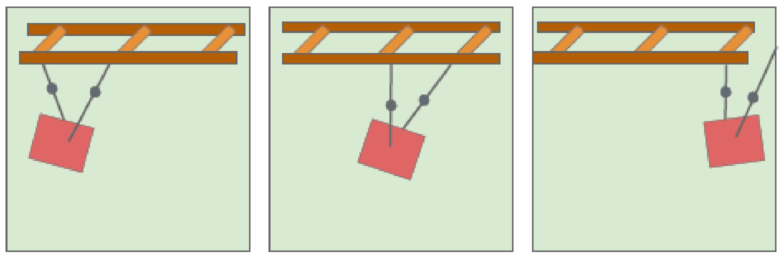
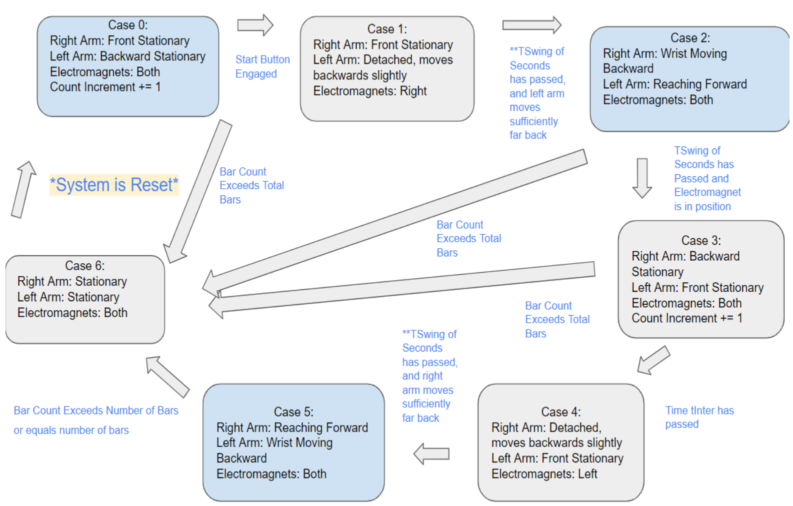


Figure 1. Arm movement of the monkeybot going across the monkey bars.

High-Level Strategy

Our monkeybot utilizes electromagnets as hands[1] that attach and detach from steel-plated monkey bars when current is turned off. DC motors and servo motors are combined to support the required arm movements to swing. The robot begins in a set position, with its right arm forward and left arm backwards on adjacent bars. Once initialized, the left electromagnets turn off and detach from the bar. The left arm's servo motor, located in the shoulder joint of the robot, shifts slightly back to create space for its arcing motion and prevent the arm from hitting the bar. When a set time, tSwing, has passed, and the left encoder position reaches sufficiently far back, the right DC motor wrist joint shifts forward and allows the left to finally reach the next bar. This cycle is repeated until the monkey bars are traversed, recognized by the software through a bar count.



[1] Gears were originally tested as the form of gripper for the monkeybot. However, this method proved to be too difficult and further complicated the simulation calculations necessary. We decided to pivot to electromagnets instead.

Figure 2. State Diagram of the Ideal Swinging Monkeybot

Simulations and encoder readings were used to plan the arm movement from one state to another. In a high level overview of our initial desired functionality and the achieved one, we were unable to mount the monkey strong enough to swing across the bars with our 2 electromagnets on each hand. However, our monkeybot demonstrates each subsystem successfully working and a solid transmission setup with the motors.

Manual and Actuation

We used a combination of materials and fabrication techniques in order to build the monkeybot's arms and body. The main body of the robot was laser cut with ¼" plywood, and it housed all the electronics and batteries. 3D-printed housing was designed to hold the 2 electromagnets of varying heights flush with each other on each of the monkey's hands. One of the pulleys controlling wrist movement was rigidly attached to the side of this 3D-printed piece, and a shaft connected it to the rest of the monkey arm. Steel (¼") and aluminum plates ($^{1}/_{32}$") were cut in Jacobs Hall's Metal Shop with the Fablight Machine to create the monkey bar and additional support around the arms, respectively. The arms were laser cut with ¼" wood and assembled with the aforementioned aluminum plates on each side to mitigate deflection.

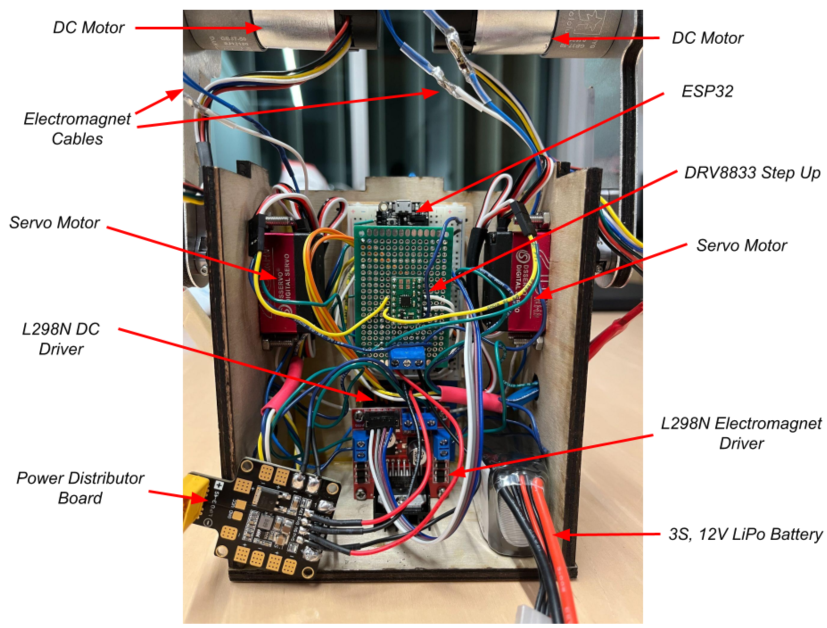


Figure 3. Outlined Assembled Robot

Each arm utilized a DC motor-driven pulley to control wrist angle for the electromagnets. Hence, the arm was manufactured with slots to properly tension the timing belt. Smooth bearings installed external to the belt provided additional tension and helped prevent sagging. High torque servo motors were mounted to the shoulder joint to control arm position and stabilize the body.

Function Critical Calculations

I. First, we calculated the required gripper torque. The monkeybot will hang on a single arm in the most extreme situation. Approximating the robot as a point mass, hanging from a bar with a ¾" diameter (0.019 m), we find an equilibrium state and solve for the single-arm grip strength.

   A. $\tau_{gripper} = F_{gripper} \cdot d_{bar} = mglsin\theta \rightarrow F_{gripper} = \mu_S F_N = mglsin\theta / d_{bar} \rightarrow F_N = mglsin\theta / \mu_S d_{bar}$

   B. $F_N = (0.75kg)(9.81m/s^2)(0.2m)(sin30deg)/(0.95)(0.019m) \rightarrow$ $\mathbf{F_N \cong 40.76\ N}$

II. Then, we calculate the required torque of the motor at the base by considering the swing period of the system, Tsys. Within Tsys/2 seconds, the arm should be able to sweep through 360 - 30 - 30 = 300° to swing to the next bar. We approximate the arm as a simple pendulum driven by the body (point torque), so $T_{sys}/2 = \pi\sqrt{l/g}$.

A. Equation of motion is $mglsin\theta + \tau_{motor} = I_{zz}\ddot{\theta}$. Since point mass, $I_{zz} = ml^2 \rightarrow mglsin\theta + \tau_{motor} = ml^2\ddot{\theta}$. Using a linear approximation, $\tau_{motor}(t) \approx ml^2\ddot{\theta} - mgl\theta$. The solution is of the form $acos(\omega t + \phi)$ for constant torque.

B. A function that accomplishes the interpolation is: $\theta(t) = \frac{3\pi}{4}(1 - cos(\frac{\pi}{Tsys}t)$. Taking the derivative, we get $\dot{\theta} = (\frac{3\pi^2}{4Tsys})\, sin(\frac{\pi}{Tsys}t)$ and $\ddot{\theta} = (\frac{3\pi^3}{4Tsys^2})\, cos(\frac{\pi}{Tsys}t)$.

C. Plugging in, we get $\tau_{motor}(t) = ml^2(\frac{3\pi^3}{4Tsys^2})\, cos(\frac{\pi}{Tsys}t) - mgl\left[\frac{3\pi}{4}(1 - cos(\frac{\pi}{Tsys}t)\right]$.

D. Plotting this with m = 0.25 kg (arm mass) and l = 0.2 m, with a total revolution of $\Delta\theta$ (degrees), we find $\tau_{max} = (\frac{2.598}{270deg})\Delta\theta \rightarrow$ **$\tau_{motorMAX} \cong$ 0.577 N·m**. We would not require another gearbox.

III. Now, we approximate the motor shaft and bearing loads. The highest velocity encountered will be at the bottom of the pendulum rotation.

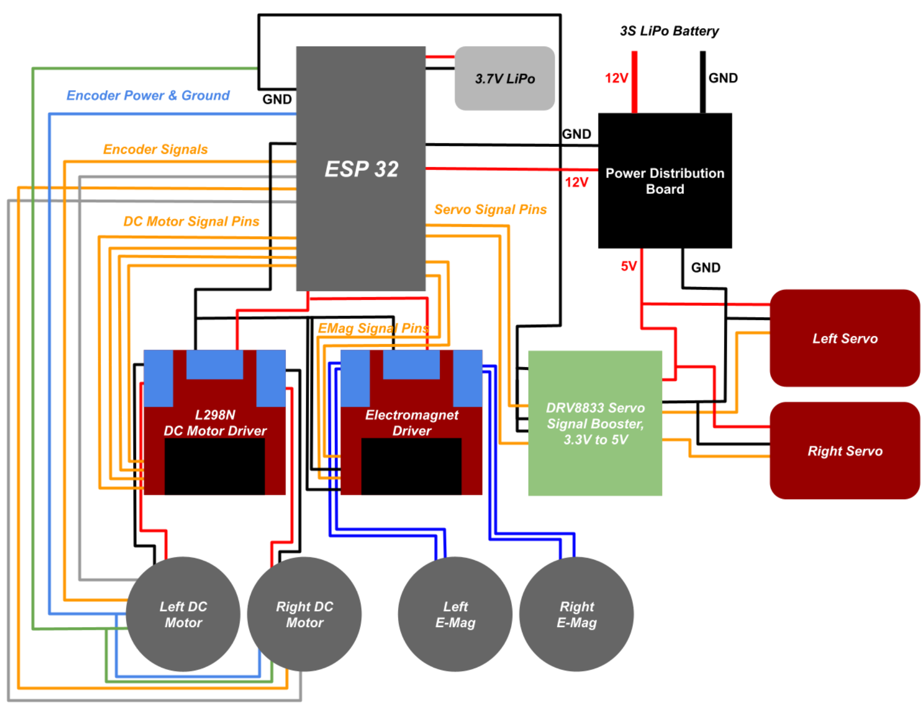
A. For one half of an oscillation, $\dot{\theta}_{avg} \approx \frac{2(\pi/3)}{Tsys} = 2.33\ rad/s$. Using the average angular velocity, we compute a centripetal force, $F_{centr} = \frac{mv^2}{l} = \frac{ml^2\dot{\theta}^2}{l} = ml\dot{\theta}^2$.

B. With $m_{robot} = 0.75kg$ and $l = 0.2m$, $F_{centr} = (0.75kg)(0.2m)(2.33\ rad/s = 0.814\ N$

C. $F_{shaft} = F_{centr} + mg = 8.17\ N\ total!$

D. On the gearbox free body diagram, $\sum F_z$: $F_{shaft} + F_{bearing1} + F_{bearing2} = 0$ and $\sum M_{bearing1}$: $F_{shaft}l_1 = F_{bearing2}l_2$. For our design, $l_1 = l_2$, so solving for bearing forces we get **$F_{bearing2} = F_{shaft}$ = 8.17 N** and **$F_{bearing1} = -2F_{shaft}$ = -16.34 N**. These are both within the allowed force on the shaft and bearings.

Circuit Diagram

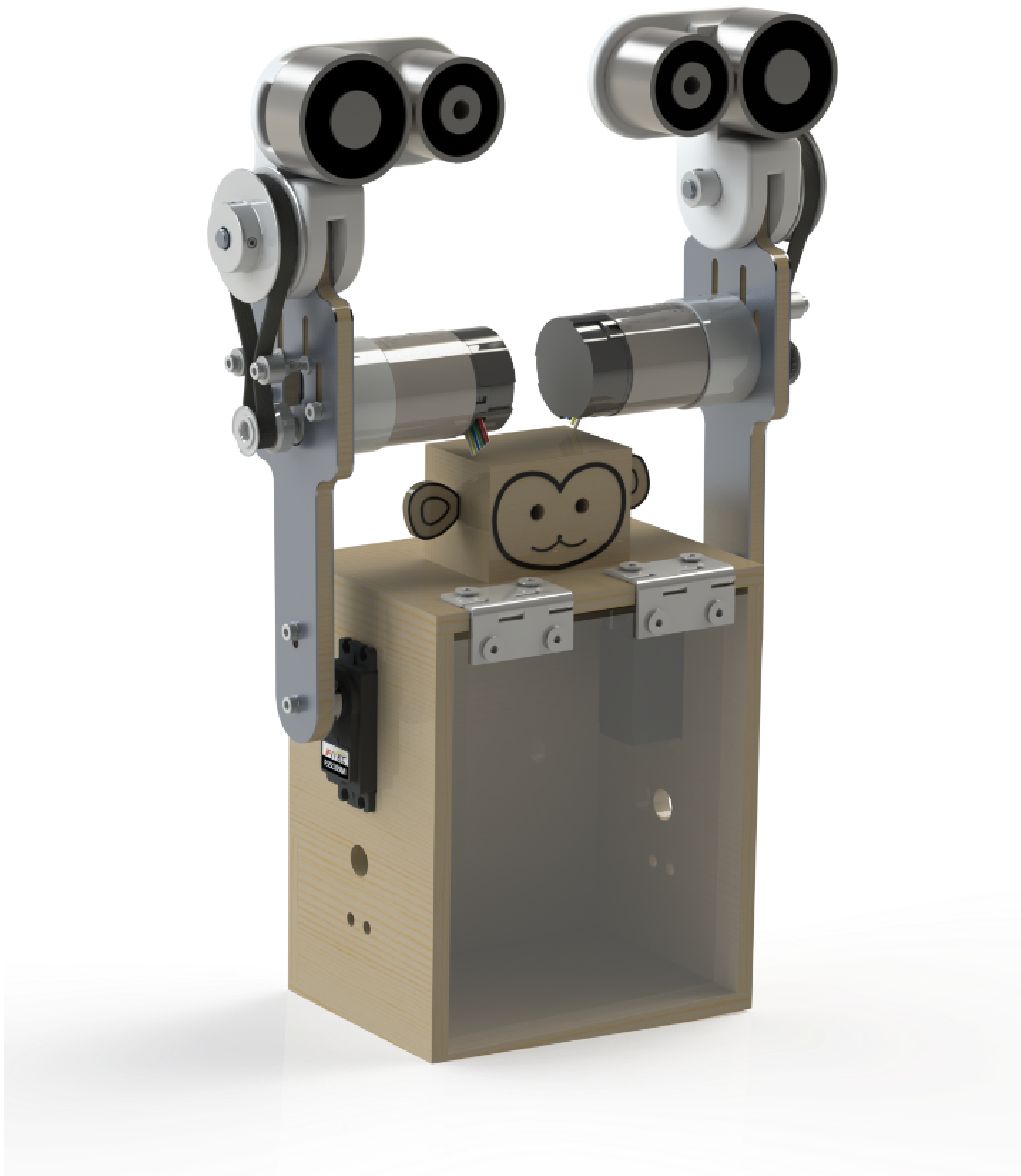


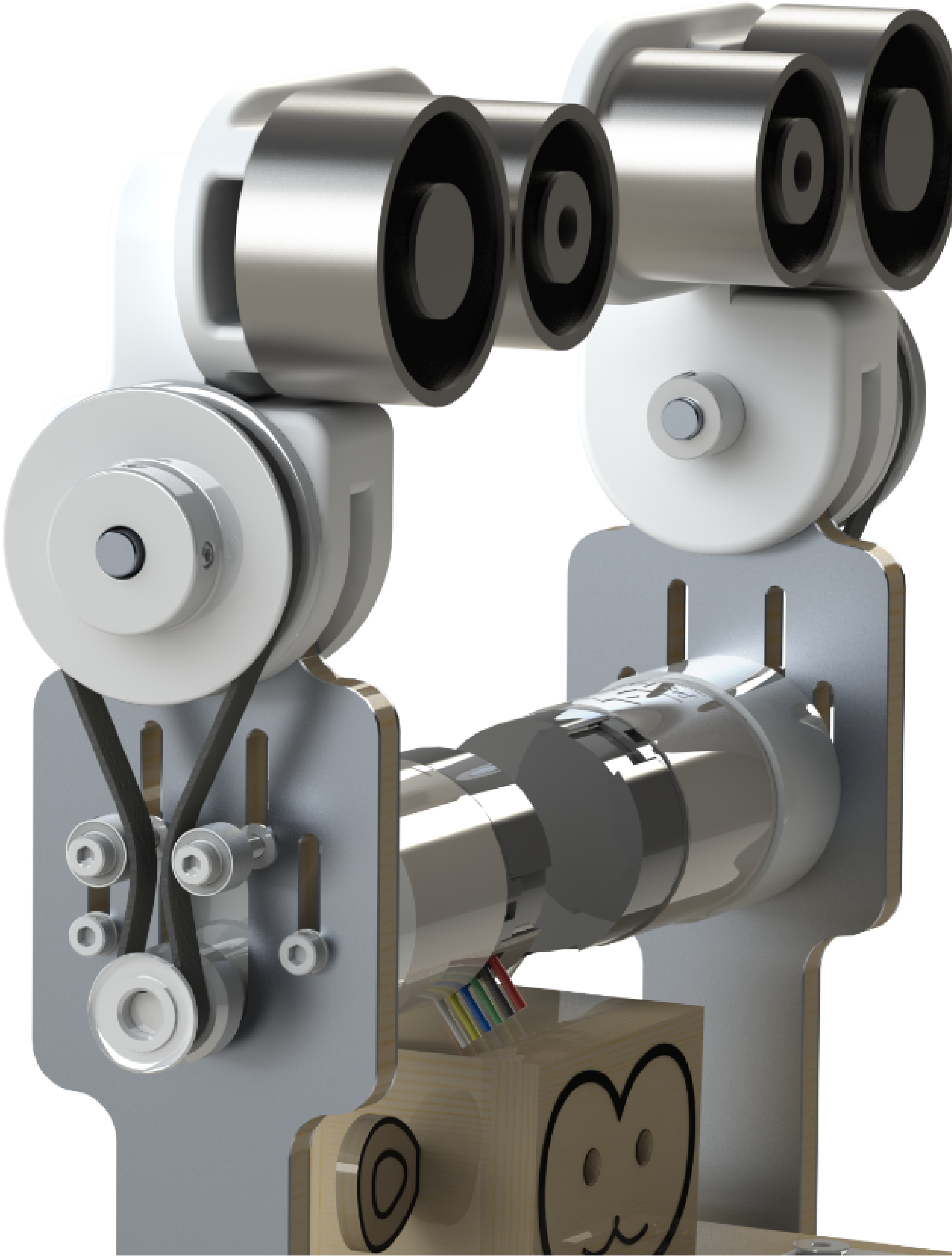
Reflection

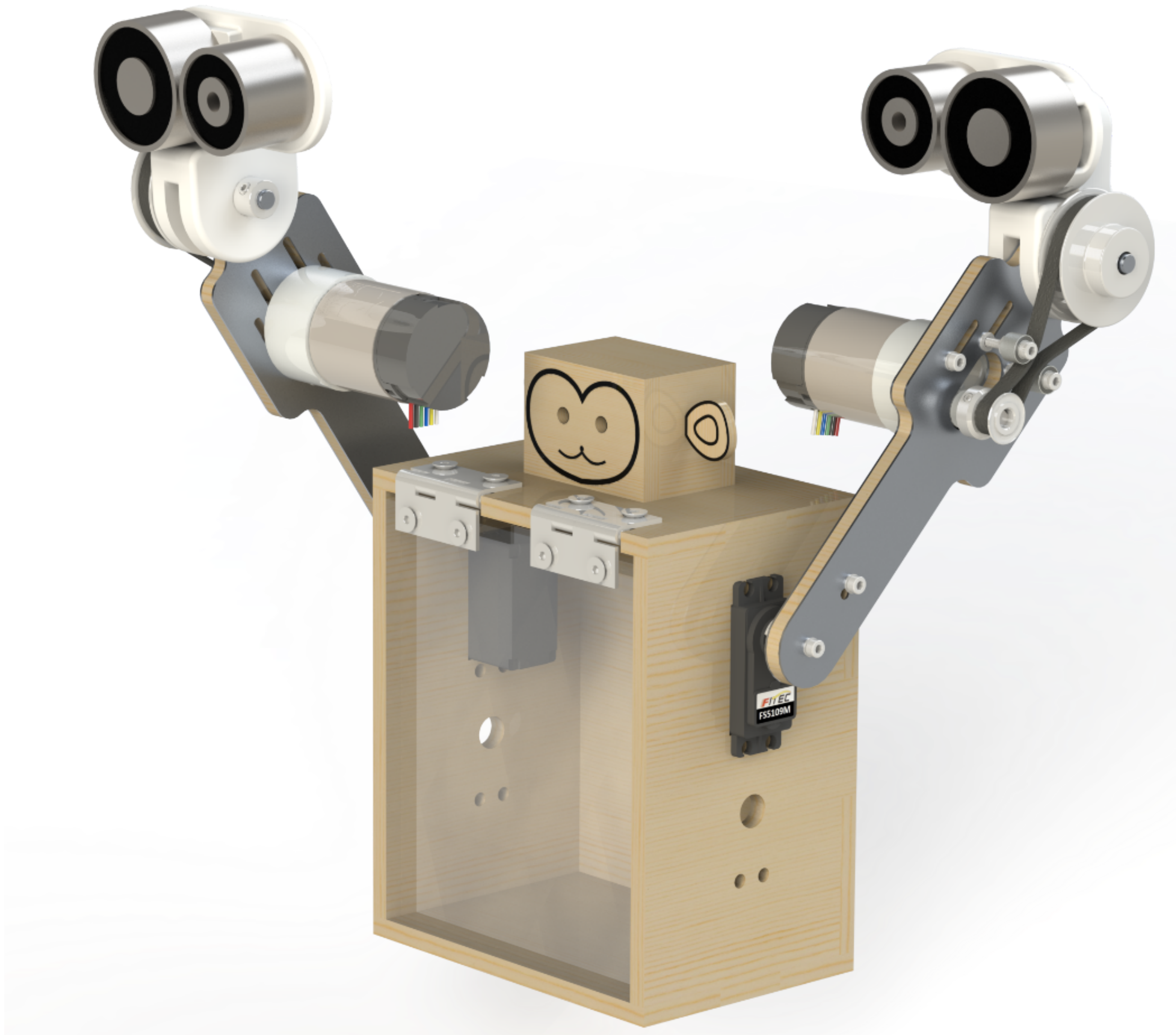
Consistent communication between group members was key throughout this project in not only creating a cohesive end product, but also in fostering an environment of inclusivity, mutual respect, and ultimately, greater productivity. We held weekly meetings from the beginning of the semester right up until the showcase to set goals and check in with each other, which worked really well for us. In retrospect, we believe it would have been helpful to manufacture and assemble all components sooner, so that more time could be spent purely debugging and troubleshooting. The controls and simulations for our robot's brachiating movement proved to be more complex than we anticipated, and we realized a bit too late that our electromagnets were not strong enough to handle the full swinging motion.

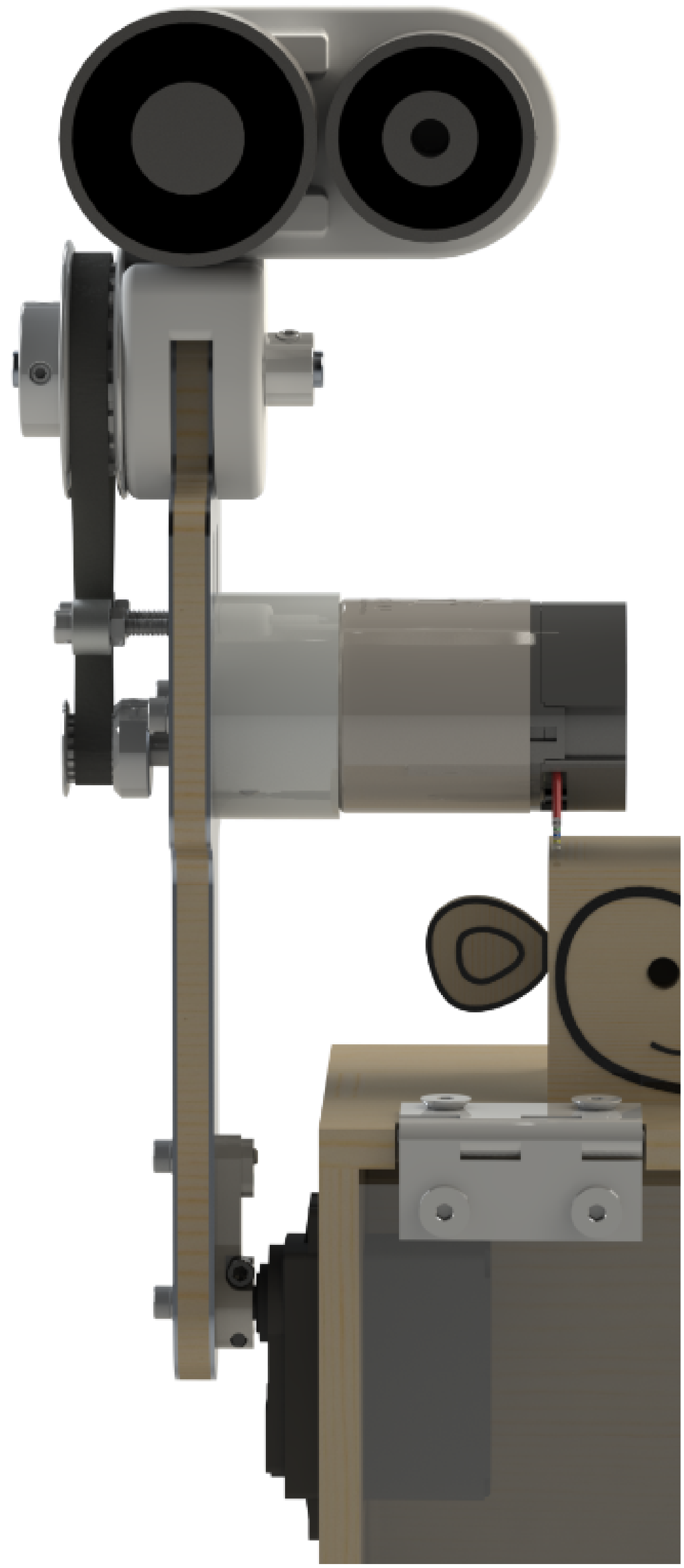
Bill of Materials

Part + Link	Supplier	Quantity	Orders	Total Cost to Acquire
24" x 24" x 0.125" Mild Steel Plate	Jacobs Hall	1	1	$45
24" x 24" x 0.063" 6061 Aluminum Plate	Jacobs Hall	1	1	$35
L298N Motor Driver Control Board	Jacobs Hall	2	1	$6
ESP32	ME Lab Kit	1	1	
DRV8833 Motor Driver	ME Lab Kit	1	1	
Brushed DC Motors	Pololu	2	1	$104
3S LiPo Battery, 2200 mAh	Amazon	2	1	$32
Drone Power Distribution Board	Amazon	2	1	$12
DS 3218 Servo	Amazon	2	1	$31
6 x 356 mm Aluminum Shaft	Amazon	2	1	$10
PCB Boards	Amazon	1	1	$13
20 & 60 Tooth Timing Pulleys	Amazon	2	1	$14
12V 250N Electromagnet	Amazon	2	1	$31
12V 200N Electromagnet	Amazon	2	1	$32






FITEC
FS5109M



communication.ino (main Arduino INO file)

```
#include "Electromagnet.h"
#include "Motors.h"
#include "StateMachine.h"

/*
* This file currently contains unit tests for the arduino libraries written.
* Note: default constructors are currently not working, do not use them.
*/
//create electromagnet objects for DEBUGGING ONLY (Comment out when using
state machine)
Electromagnet emagL(17);
Electromagnet emagR(21);

//create a motors object for DEBUGGING ONLY (Comment out when using state
machine)
Motors motors(33, 27, 13, 12, 15, 32, 34, 39, 26, 25);

int numBars = 2;

//Create a state machine object
StateMachine stateMachine(33, 27, 13, 12, 15, 32, 34, 39, 26, 25, 17, 21,
numBars);

void setup() {
 //Begin a serial monitor
 Serial.begin(115200);
 Serial.println("Starting");
}

void loop() {
 //update the state machine state
 stateMachine.updateState();
 //drive the motors to their desired position
 stateMachine.motors.driveDes();
}



/*
```

```
* Unit test cases - to be used for debugging ONLY when the state machine has
not been initalized.
* Place in the main loop() to use.
* emagTests() -> test case for the electromagnet units on both arms
* hangTest() -> writes both electromagnets high for a hanging test
*/

void detachTest() {
 //test detaching and arm moving back sequence

 //turn L electromagnet off
 emagL.switchOff();

 //move the left arm back and the left DC motor to flop back position
 motors.setDesPos(20, 41, 534, 0);

 //drive the motors
 motors.driveDes();
}

void servoTests() {
 //run the servo to a desired angle, DC motors to 0.
 int desAngle = 41;
 motors.setDesPos(desAngle, desAngle, 300, -300);
 motors.driveDes();
 Serial.println(motors.getAngleMotorLeft());
 Serial.println(motors.getAngleMotorRight());
 delay(15);
}

void dcTests() {
 //digitalWrite(13, HIGH);
 //digitalWrite(12, LOW);
 motors.setDesPos(60, 60, 300, 300);
 motors.driveDCR(300);
 motors.driveDCL(300);
 printEncoder();
}

void encoderTest() {
```

```
 //print the encoder readings to the serial monitor
 Serial.println(motors.getAngleMotorLeft());
 Serial.println(motors.getAngleMotorRight());
}

void emagTests() {
 //Run the electromagnet tests
 emagL.emagTest();
 emagR.emagTest();
}

void hangTest() {
 //move the servos to a default 41 deg position
 motors.setDesPos(60, 60, 0, 0);
 motors.driveDes();
  //turn on the electromagnets for a simple hang
 emagR.switchOn();
 emagL.switchOn();

 printEncoder();
}

void printEncoder() {
 Serial.print("enc left: ");
 Serial.print(motors.getAngleMotorLeft());
 Serial.println();
 Serial.print("enc right: ");
 Serial.print(motors.getAngleMotorRight());
 Serial.println();
}

void encoder41Deg() {
 //Start in the 41, 41 position with both wrists vertical. this is the zero
position.
 //(41, 41) is left arm forwards, left arm backwards.
 motors.setDesPos(41, 41, 0, 0);

 motors.driveDes();
}
```

```
void encoderCalibrate() {
 //to be called AFTER setup()
 //move to the 76, 76 position. (76, 76) is left arm forwards, right arm
backwards.
 motors.setDesPos(76, 76, 0, 0);
 motors.driveDes();
 //Now, move the wrists to vertical in this config and print out the encoder
reading.
 //This gives desired state.
 printEncoder();
}
```

Electromagnet.cpp

```
#include "Electromagnet.h"
/*
This file contains an implementation of the Electromagnet class
*/

/*
Initialize an electromagnet object
Inputs:
   s_pin (int) : pin number of signal pin used to switch the electromagnet on
and off
*/
Electromagnet::Electromagnet() {
   //default constructor - calls the other constructor
   Electromagnet(13);
}

Electromagnet::Electromagnet(int s_pin) {
   sPin = s_pin; //store the signal pin
   testPeriod = 3000; //set the test period

   //set the pin mode to OUTPUT
   pinMode(sPin, OUTPUT);
}

/*
```

```
Function to turn the electromagnet on. To turn the electromagnet on, set the
forward signal
pin on the L298N Motor driver HIGH.
*/
void Electromagnet::switchOn() {
   digitalWrite(sPin, HIGH);
}

/*
Function to turn the electromagnet off. To turn the electromagnet off, set the
forward signal
pin on the L298N Motor driver LOW.
*/
void Electromagnet::switchOff() {
   digitalWrite(sPin, LOW);
}

/*
Function to provide a test case for the electromagnet. Runs the electromagnet
high for
testPeriod ms, then low for testPeriod ms. Place in loop() to test.
*/
void Electromagnet::emagTest() {
   switchOn();
   delay(testPeriod);
   switchOff();
   delay(testPeriod);
}
```

Electromagnet.h

```
#ifndef Electromagnet_H
#define Electromagnet_H

#include <Arduino.h> //required to access arduino functionality
/*
This file contains a class for interacting with a single electromagnet, or a
set of
electromagnets wired to the same terminal on the relay board.
```

```
*/

class Electromagnet {
   private:
       //electromagnet signal pin from the ESP32
       int sPin;
       int testPeriod;
   public:
       Electromagnet(); //define a default constructor
       Electromagnet(int s_pin);
       void switchOn();
       void switchOff();
       void emagTest();
};

#endif
```

Motors.cpp

```
#include "Motors.h"

/*
This file contains an implementation of the motors class.
*/

/*
Init function for a motor class
*/
Motors::Motors() {
   Motors(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
}
Motors::Motors(int servo_pin_L, int servo_pin_R, int m_pin_1_L, int m_pin_2_L,
int m_pin_1_R, int m_pin_2_R, int enc1_L, int enc2_L, int enc1_R, int enc2_R){
   //define the servo pins
   servoPinL = servo_pin_L;
   servoPinR = servo_pin_R;

   //define the motor pins
   mPin1L = m_pin_1_L;
   mPin2L = m_pin_2_L;
```

```
  mPin1R = m_pin_1_R;
  mPin2R = m_pin_2_R;

  //define the encoder pins
  encPin1L = enc1_L;
  encPin2L = enc2_L;
  encPin1R = enc1_R;
  encPin2R = enc2_R;

  //set the pin modes to OUTPUT
  pinMode(servoPinL, OUTPUT);
  pinMode(servoPinR, OUTPUT);
  pinMode(mPin1L, OUTPUT);
  pinMode(mPin2L, OUTPUT);
  pinMode(mPin1R, OUTPUT);
  pinMode(mPin2R, OUTPUT);

  //set the encoder pins to INPUT
  pinMode(encPin1L, INPUT);
  pinMode(encPin2L, INPUT);
  pinMode(encPin1R, INPUT);
  pinMode(encPin2R, INPUT);

  //set up encoders
  ESP32Encoder::useInternalWeakPullResistors = UP; //Enable weak pullups
  encoderL.attachHalfQuad(encPin1L, encPin2L);
  encoderL.setCount(0); //INITIALIZE the encoder count to be zero!!
  encoderR.attachHalfQuad(encPin1R, encPin2R);
  encoderR.setCount(0); //INITIALIZE the encoder count to be zero!!

  //set the default desired servo positions, 60 degrees (vertical) - 41
degrees is a bar starting position.
  desPosServoL = 60;
  desPosServoR = 60;

  //set default desired motor positions
  desPosMotorL = 0;
  desPosMotorR = 0;

  //define frequencies
```

```
  servoFreqLow = 500; //500 for the Pololu servo
  servoFreqHigh = 2500; //2500 for the Pololu servo

  //define PWM constants
  MAX_PWM = 180;
  MIN_PWM = 0;

  //define controller constants
  Kp = 1;

  //allocate the timers
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);

  //set the servo periods
  servoL.setPeriodHertz(50);
  servoR.setPeriodHertz(50);

  //set the motor pin periods
  motorL1.setPeriodHertz(50);
  motorL2.setPeriodHertz(50);
  motorR1.setPeriodHertz(50);
  motorR2.setPeriodHertz(50);

  //attach the servos to their PWM pins
  servoL.attach(servoPinL, servoFreqLow, servoFreqHigh);
  servoR.attach(servoPinR, servoFreqLow, servoFreqHigh);

  //attach the motors pwm pins
  motorL1.attach(mPin1L, servoFreqLow, servoFreqHigh);
  motorL2.attach(mPin2L, servoFreqLow, servoFreqHigh);
  motorR1.attach(mPin1R, servoFreqLow, servoFreqHigh);
  motorR2.attach(mPin2R, servoFreqLow, servoFreqHigh);

  //drive the servos to their initial position!
  int startPos = 41; //option to begin it in the start position with one fwd
one back.
  servoL.write(startPos); //writes it NOT to the vertical but to 60
```

```
    servoR.write(startPos);
}

/*
Function to set the desired positions of the two servo motors.
Inputs:
    des_pos_1 (int) : desired position in radians of the first servo
    des_pos_2 (int) : desired position in radians of the second servo
*/
void Motors::setDesPos(int des_pos_servo_L, int des_pos_servo_R, int
des_pos_motor_L, int des_pos_motor_R) {
    desPosServoL = des_pos_servo_L;
    desPosServoR = des_pos_servo_R;
    desPosMotorL = des_pos_motor_L;
    desPosMotorR = des_pos_motor_R;
}

/*
Function to drive the servo motors to the desired position stored in the class
*/
void Motors::driveDes() {
    driveServos(desPosServoL, desPosServoR);
    driveDCL(desPosMotorL);
    driveDCR(desPosMotorR);
}

/*
Function to drive the two servo motors
Inputs:
    desPos1 (int) : desired position from 0 to 180 of the first servo
    desPos2 (int) : desired position from 0 to 180 of the second servo
*/
void Motors::driveServos(int desPosL, int desPosR) {
    //drive the first servo motor to desPos1
    servoL.write(desPosL);
    //derive the second servo motor to desPos2
    servoR.write(desPosR);
    return;
}
```

```
/*
Function to return the current angle of the DC motor, as read by the encoder
in ticks.
Returns:
   motorAngle (int) : positional reading of the encoder in encoder units
*/
int Motors::getAngleMotorLeft() {
   return (int32_t)encoderL.getCount();
}

int Motors::getAngleMotorRight() {
   return (int32_t)encoderR.getCount();
}

/*
Function to drive the DC motor to the correct position using FB control
The right arm has a negative feedback gain as it is flipped.
Inputs:
   des_pos (double) : desired position in radians. NOT the class attribute
*/
void Motors::driveDCL(int des_pos) {
   //next, calculate the control input
   int controlInput = floor(Kp*(des_pos - (int32_t)encoderL.getCount()));

   //drive the motor with a PWM according to the control input
   if (controlInput >= 0) {
       //clamp the control input
       controlInput = min(MAX_PWM, max(MIN_PWM, controlInput));

       //write one pin high, the other low
       motorL1.write(controlInput);
       digitalWrite(mPin2L, LOW);
       // motorL2.write(0);
   } else {
       //flip the sign of the control input
       controlInput = -1*controlInput;

       //clamp the control input
       controlInput = min(MAX_PWM, max(MIN_PWM, controlInput));
```

```
        //drive the motor
        // motorL1.write(0);
        digitalWrite(mPin1L, LOW);
        motorL2.write(controlInput);
    }
}

void Motors::driveDCR(int des_pos) {
    //next, calculate the control input SWITCHING the sign of Kp to (-)
    int controlInput = floor(-Kp*(des_pos - (int32_t)encoderR.getCount()));

    //drive the motor with a PWM according to the control input
    if (controlInput >= 0) {
        //clamp the control input
        controlInput = min(MAX_PWM, max(MIN_PWM, controlInput));

        Serial.println(controlInput);

        //write one pin high, the other low
        motorR1.write(controlInput);
        // motorR2.write(0);
        digitalWrite(mPin2R, LOW);
    } else {
        //flip the sign of the control input
        controlInput = -1*controlInput;

        //clamp the control input
        controlInput = min(MAX_PWM, max(MIN_PWM, controlInput));

        Serial.println(controlInput);

        //drive the motor
        // motorR1.write(0);
        digitalWrite(mPin1R, LOW);
        motorR2.write(controlInput);
    }
}

void Motors::driveOLTest() {
    // motorR1.write(0);
```

```
    digitalWrite(mPin2R, LOW);
    motorR1.write(180);
}

void Motors::dcControllerTest() {
    //give the controller a small setpoint that it must reach.Assume it starts
at zero.
    int setpt = 300;
    driveDCR(setpt);
}
```

Motors.h

```
#ifndef Motors_H
#define Motors_H

#include <Arduino.h> //required to access the arduino functions.
#include <ESP32Servo.h> //Servo library
#include <ESP32Encoder.h> //encoder library

/*
This file contains the utilities for driving all motors.
*/
class Motors {
    public:
        Motors(); //default constructor
        Motors(int servo_pin_L, int servo_pin_R, int m_pin_1_L, int m_pin_2_L,
int m_pin_1_R, int m_pin_2_R, int enc1_L, int enc2_L, int enc1_R, int enc2_R);
        void setDesPos(int des_pos_servo_L, int des_pos_servo_R, int
des_pos_motor_L, int des_pos_motor_R);
        void driveDes();
        int getAngleMotorLeft();
        int getAngleMotorRight();
        void driveServos(int desPosL, int desPosR);
        void driveDCL(int des_pos);
        void driveDCR(int des_pos);
        void driveOLTest();
        void dcControllerTest();
    private:
        //Servo motors
```

```
Servo servoL;
Servo servoR;

//DC Motor pins
Servo motorL1;
Servo motorL2;
Servo motorR1;
Servo motorR2;

//PWM pins for the servos
int servoPinL;
int servoPinR;

//PMW pins for the DC motors
int mPin1L;
int mPin2L;
int mPin1R;
int mPin2R;

//Encoder pins for DC motors
int encPin1L;
int encPin2L;
int encPin1R;
int encPin2R;

//Encoder object
ESP32Encoder encoderL;
ESP32Encoder encoderR;

//motor parameters
int MAX_PWM;
int MIN_PWM;

//controller paramters
double Kp = 1;

//desired angular position of the servos
int desPosServoL;
int desPosServoR;
```

```
        //desired angular position of the motors
        int desPosMotorL;
        int desPosMotorR;

        //Servo freqeuncies
        int servoFreqLow; //500 for the Pololu servo
        int servoFreqHigh; //2500 for the Pololu servo
};

#endif
```

StateMachine.cpp

```
#include "StateMachine.h"
/*
This file contains an implementation of the State Machine class
*/

/*
Initialization function for a state machine
Inputs:
   dc_motor_left (DCMotor) : DC motor object, left arm
   dc_motor_right (DCMotor) : DC motor object, right arm
   servo_motors (ServoMotors) : Servo motors (L and R) object
   num_bars (int) : number of total bars on the monkey bars
*/
StateMachine::StateMachine() {
   //DEFAULT constructor
   StateMachine(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13);
}
StateMachine::StateMachine(int servo_pin_L, int servo_pin_R, int m_pin_1_L,
int m_pin_2_L, int m_pin_1_R, int m_pin_2_R, int enc1_L, int enc2_L, int
enc1_R, int enc2_R, int s_pin_L, int s_pin_R, int num_bars) {
   //store the motor objects in the class attributes
   motors = Motors(servo_pin_L, servo_pin_R, m_pin_1_L, m_pin_2_L, m_pin_1_R,
m_pin_2_R, enc1_L, enc2_L, enc1_R, enc2_R);

   //store the electromagnet objects
   eMagLeft = Electromagnet(s_pin_L);
   eMagRight = Electromagnet(s_pin_R);
```

```
   //initialize system to be in the zero state
   state = 0;

   //set the number of bars traversed to be zero
   barCounter = 0;

   //set the number of total bars
   numBars = num_bars;

   //set the time constants
   tInter = 1500; //intermediate stopping time between swings in ms.
}

/*
Function to update the state based on the current values of the system
*/
void StateMachine::updateState() {
   //Finite state machine switch statement
   switch (state) {
       case 0:
           /*
           Case 0 is the initialization state. In state 0, the following
parameters are satisfied:
           1. The right arm is in the forwards position on a bar, the left arm
is in the backwards
              position on a bar.
           2. The electromagnets on both arms are on.
           3. Increment bar count by 1.

           Entrance conditions:
               ...
           Exit conditions:
           This state is only visited for tInter time. It is the start state
of the system and is exited once
           the start button has been engaged. If bar count exceeds total bars,
go to state 6 (terminal).
           */

           Serial.println("hi");
```

```
            //update the desired states
            updateDesStates();

            //First, ensure that both electromagnets are on
            eMagLeft.switchOn();
            eMagRight.switchOn();

            if (firstRound == true) {
                firstRound = false; //set it such that it is not the first
traversal
                delay(100000);
            } else {
                //delay for tInter seconds. This is a debugging feature that
may be shortened in period.
                delay(tInter);
            }

            if (barCounter >= numBars) {
                //go to the terminal state, as the number of bars have been met
                state = 6;
            } else {
                //go to state 2 and swing to the next bar
                state = 1;
                barCounter++; //increment the bar counter
            }
            break;

        case 1:
            /* FIRST part of a swing
            In state 1, the following parameters are satisfies
            1. Right arm is not moving.
            2. Left arm has detached from the bar and is moving backwards
slightly to avoid hitting the bar.
            3. Right electromagnets are on
            4. Left electromagnets are off.
            5. Wrist goes to flop position.

            Entrance conditions:
            - Case 1 may be entered only from case 0.
```

```
            Exit conditions:
            - Case 1 exits after a time period tSwing of seconds has passed and
the left arm has moved sufficiently far back. (moves to case 2)
            */

            //update the desired states
            updateDesStates();

            //turn the left electromagnet off
            eMagLeft.switchOff();

            //turn the right electromagnet on
            eMagRight.switchOn();

            callCounter ++;
            if (callCounter >= callCutoff) {
                //reset the call counter
                callCounter = 0;

                //move to the next state
                state = 2;
            }
            break;

        case 2:
            /* SECOND part of a swing
            In state 2, the following parameters are satisfies
            1. Right arm is moving to the backwards position and attached to a
bar
            2. Left arm has detached from the bar and is moving forwards
            3. Right electromagnets are on
            4. Left electromagnets are on.

            Entrance conditions:
            - Case 1 may be entered only from case 0.

            Exit conditions:
            - Case 1 exits after a time period tSwing of seconds has passed and
the electromagnet is in position. (moves to case 3)
            */
```

```
            //update the desired states
            updateDesStates();

            //turn the left electromagnet on.
            eMagLeft.switchOn();

            //turn the right electromagnet on (should already be on)
            eMagRight.switchOn();

            callCounter ++;
            if (callCounter >= callCutoff) {
                //reset the call counter
                callCounter = 0;
                //move to the next state
                state = 3;
            }
            break;

        case 3:
            /*
            In state 3, the following parameters are satisfied:
            1. The right arm is in the backwards position on a bar
            2. Left arm is in the forwards position on a bar
            3. Right electromagnets are on
            4. Left electromagnets are on
            5. Increment bar count by 1
            Entrance conditions:
            - Case 3 may be enetered only from case 2
            Exit conditions:
            - Case 3 is exited a time tInter after entrance to prepare for the
next swing.
            - Case 3 is exited to state 4 if the bar count exceeds number of
bars
            */

            //update the desired states
            updateDesStates();

            //turn the left electromagnet on
```

```
            eMagLeft.switchOn();

            //turn the right electromagnet on
            eMagRight.switchOn();

            //wait on the bar for tInter seconds
            delay(tInter);

            if (barCounter >= numBars) {
                //go to the terminal state, as the number of bars have been met
                state = 6;
            } else {
                //go to state 2 and swing to the next bar
                state = 4;
                barCounter++; //increment the bar counter
            }
            break;

        case 4:
             /* FIRST part of a swing
            In state 1, the following parameters are satisfies
            1. Right arm is moving slightly back to avoid hitting the bar
            2. Left arm stays still.
            3. Right electromagnets are off
            4. Left electromagnets are on.
            5. Move right wrist to "flop" configuration

            Entrance conditions:
            - Case 4 may be entered only from case 3.

            Exit conditions:
            - Case 1 exits after a time period tSwing of seconds has passed and
the left arm has moved sufficiently far back. (moves to case 2)
            */

            //update the desired states
            updateDesStates();

            //turn the left electromagnet on
            eMagLeft.switchOn();
```

```
            //turn the right electromagnet off
            eMagRight.switchOff();

            callCounter ++;
            if (callCounter >= callCutoff) {
                //reset the call counter
                callCounter = 0;
                //move to the next state
                state = 5;
            }
            break;

        case 5:
            /* SECOND part of a swing
            In state 4, the following parameters are satisfies
            1. Right arm is moving freely forwards
            2. Left arm is attached to a bar and moving to the backwards
position
            3. Right electromagnets are on
            4. Left electromagnets are on.

            Entrance conditions:
            - Case 5 may be entered only from case 4.

            Exit conditions:
            - Case 5 exits after a time period tSwing of seconds has passed and
the electromagnet is in position. (moves to case 0)
            */

            //update the desired states
            updateDesStates();

            //turn the left electromagnet on
            eMagLeft.switchOn();

            //turn the right electromagnet on
            eMagRight.switchOn();

            callCounter ++;
```

```
            if (callCounter >= callCutoff) {
                //reset the call counter
                callCounter = 0;
                //move to the next state
                state = 0;
            }
            break;

        case 6:
            /*
            In case 4, the following parameters are satisfied:
            1. The right arm is fixed to a bar
            2. The left arm is fixed to a bar
            3. Right electromagnets are on
            4. Left electromagnets are on
            5. The bar count exceeds or equals the number of monkey bars.
            Entrance conditions:
            - State is entered if bar count exceeds or equals number of bars,
from case 0 or case 2.
            Exit conditions:
            - Robot must be reset to exit case 4, this is the terminal case
            */
           eMagLeft.switchOn();
           eMagRight.switchOn();
           break;
    }
}

void StateMachine::state0() {
   Serial.println("hi");
   //update the desired states
   updateDesStates();

   //First, ensure that both electromagnets are on
   eMagLeft.switchOn();
   eMagRight.switchOn();

   if (firstRound == true) {
       firstRound = false; //set it such that it is not the first traversal
       delay(100000);
```

```
   } else {
       //delay for tInter seconds. This is a debugging feature that may be
shortened in period.
       delay(tInter);
   }
}

void StateMachine::state1() {
   //update the desired states
   updateDesStates();

   //turn the left electromagnet off
   eMagLeft.switchOff();

   //turn the right electromagnet on
   eMagRight.switchOn();
}

void StateMachine::state2() {
   //update the desired states
   updateDesStates();

   //turn the left electromagnet on.
   eMagLeft.switchOn();

   //turn the right electromagnet on (should already be on)
   eMagRight.switchOn();
}
void StateMachine::state3() {
   //update the desired states
   updateDesStates();

   //turn the left electromagnet on
   eMagLeft.switchOn();

   //turn the right electromagnet on
   eMagRight.switchOn();

   //wait on the bar for tInter seconds
   delay(tInter);
```

```
}
void StateMachine::state4() {
   //update the desired states
   updateDesStates();

   //turn the left electromagnet on
   eMagLeft.switchOn();

   //turn the right electromagnet off
   eMagRight.switchOff();
}
void StateMachine::state5() {
   updateDesStates();

   //turn the left electromagnet on
   eMagLeft.switchOn();

   //turn the right electromagnet on
   eMagRight.switchOn();
}

/*
Function to retrieve the state of the system
*/
int StateMachine::getState() {
   return state;
}

/*
Function to update the desired states of each motor depending on the state
*/
void StateMachine::updateDesStates() {
   if (state == 0) {
       //set desired motor states
       motors.setDesPos(backwardSlightServoL, forwardServoR,
backwardSlightDCL, forwardDCR);
   } else if (state == 1) {
       //set the desired motor states
       motors.setDesPos(backwardFullServoL, forwardServoR, backwardFullDCL,
forwardDCR);
```

```
    } else if (state == 2) {
        //set the desired motor states
        motors.setDesPos(forwardServoL, backwardSlightDCR, forwardDCL,
backwardSlightDCR);
    } else if (state == 3) {
        //set the desired motor states
        motors.setDesPos(forwardServoL, backwardSlightDCR, forwardDCL,
backwardSlightDCR);
    } else if (state = 4) {
        //set the desired motor states
        motors.setDesPos(forwardDCL, backwardFullDCR, forwardDCL,
backwardFullDCR);
    } else {
        //set the desired DC states to be whatever they currently are
        return;
    }
}
```

StateMachine.h

```
#ifndef StateMachine_H
#define StateMachine_H

#include <Arduino.h> //required to access the arduino functions.
#include "Electromagnet.h"
#include "Motors.h"
/*
This file contains a class for the state machine of the system, which manages
the different
trajectories based on the state of the system.
*/
class StateMachine {
    private:
        int barCounter;
        int numBars;
        int tInter;

        //Store desired states for each stage
        int forwardServoR = 41; //right front, left back
```

```
        int forwardServoL = 76; //left front, back right position
        int backwardFullServoR = 90; //in this position, it's winding back to
avoid hitting a bar.
        int backwardFullServoL = 20;
        int backwardSlightServoR = 76; //In this position, it's sitting on a
bar
        int backwardSlightServoL = 41;
        int forwardDCR = 0; //right arm forward, left arm back
        int forwardDCL = 942; //DC motor position when left arm is forward, WRT
a zero at left back
        int backwardFullDCR = 534; //don't really care about wrist at full back
position.
        int backwardFullDCL = -1593;
        int backwardSlightDCR = 1045; //DC motor position when right arm sits
back on a bar
        int backwardSlightDCL = 0;
        int flopL = 534; //flop back angle to minimze the profile of the
electromagnet
        int flopR = -1593;

        //store information about first traversal
        bool firstRound = true;

        //store a call counter for the number of calls to each feedback loop
        int callCounter = 0;
        int callCutoff = 50; //gives around 1s of motion.

        void updateDesStates();
    public:
        Motors motors;
        Electromagnet eMagLeft;
        Electromagnet eMagRight;

        int state;
        StateMachine();
        StateMachine(int servo_pin_L, int servo_pin_R, int m_pin_1_L, int
m_pin_2_L, int m_pin_1_R, int m_pin_2_R, int enc1_L, int enc2_L, int enc1_R,
int enc2_R, int s_pin_L, int s_pin_R, int num_bars);
        void updateState();
        void state0();
```

```
        void state1();
        void state2();
        void state3();
        void state4();
        void state5();
        int getState(); //retrieve the state of the system
};

#endif
```