Digitronix Mechatronic Wrist

Group 22: Cynthia Valdez, Danielle Hernandez, Ricardo Cano, Fernando Cardenas

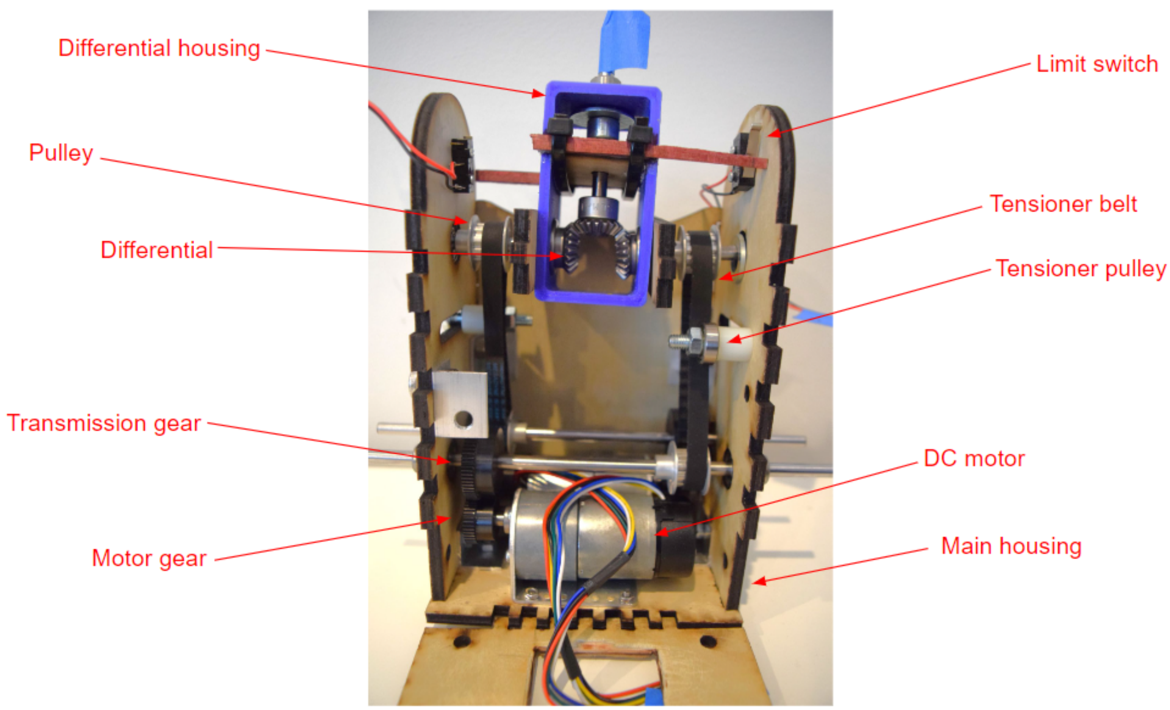
Opportunity

Numerous prosthetic hands and gripping devices for amputees are available on the market today. However, many of these devices are rigid at the wrist and do not allow users to control the angle at which they grip objects. Our solution employs a differential drive that empowers the user to move their hand with two degrees of freedom at the wrist with flexion/extension and supination/pronation.

High-Level Strategy

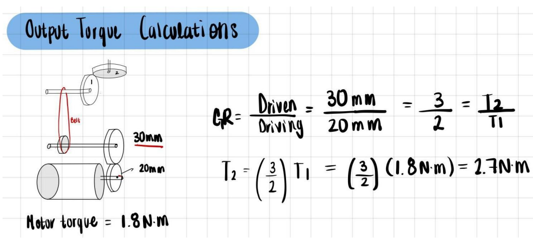
Our prosthetic wrist includes two DC brushless motors, geared power transmission shafts, and a differential to achieve two degrees of freedom: extension and flexion; supination and pronation. The user can adjust the position of two potentiometers to actuate the motion of the wrist. One is for supination and pronation, and the other is for extension and flexion. Once the appropriate potentiometer is rotated, the data is processed by an ESP32 microcontroller to calculate the speed and direction in which the DC motors should move. The rotational motion is transferred to the differential by the geared transmission shafts and pulleys. Our team intended to utilize myoelectric sensors to control the prosthetic wrist. However, achieving a stable signal from the myoelectric sensors became a challenge due to the proper positioning of the sensors and the fluctuations that are associated with the flexing of muscles. Adding a gain and proportional control may mitigate these fluctuations.

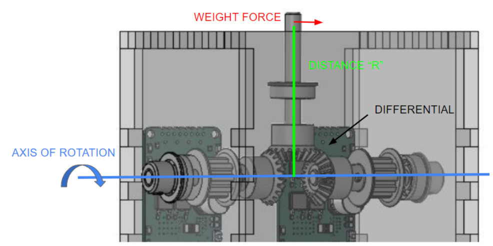
Integrated Physical Device



Function-critical decisions

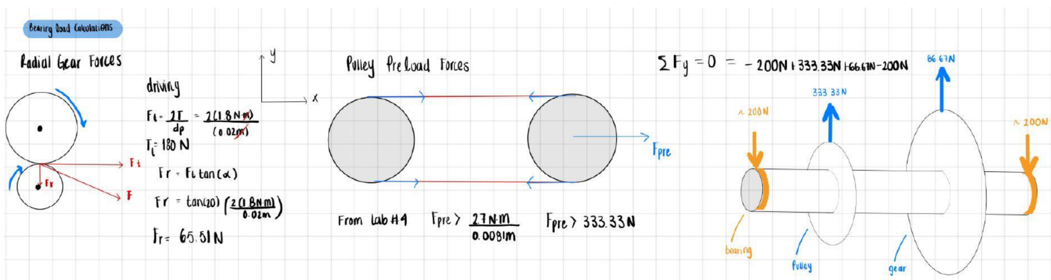
The main function-critical decisions our team made were the sizing of the motors and the selection of the bearings. Our team decided that designing for a 22.2 N load was appropriate for our product because that is the approximate weight of a five-pound dumbbell. The torque that our motor needs to counteract and overcome is as τ_{LOAD} = R x $f_{LOAD,}$ where R is the distance between the load's center of gravity and the center of the differential about which the load would be rotating. The weight force f_{LOAD} acting at distance R is 22.2 N. Assuming a worst-case scenario where the wrist is in flexion, or extension, at 90 degrees, the torque τ_{LOAD} calculated is approximately 9 kg·cm. The required torque of the motor is then calculated to be 1.6 times τ_{LOAD}, so that the torque due to the load would not exceed sixty percent of the motor's stall torque. Hence, the calculated τ_{MOTOR} is approximately 15 kg·cm leaving a factor of safety of 1.6. The motor we acquired was rated at a stall torque of 18 kg·cm, giving us a safety factor of 2.0, and the motor included an encoder. We also obtained a power supply that could provide the twelve volts needed to run the motor. The motor's stall current is seven amps, and the power supply provides up to 13.4 amps. To further increase the torque output of our system, we included a transmission gear that was 1.5 times larger than our motor gear, increasing the output torque to 27 kg·cm.





$$GR = \frac{driven}{driving} = \frac{30mm}{20mm} = \frac{3}{2} = \frac{T_2}{T_1} \qquad T_2 = \left(\frac{3}{2}\right)T_1 = \left(\frac{3}{2}\right)(1.8Nm) = 2.7Nm$$

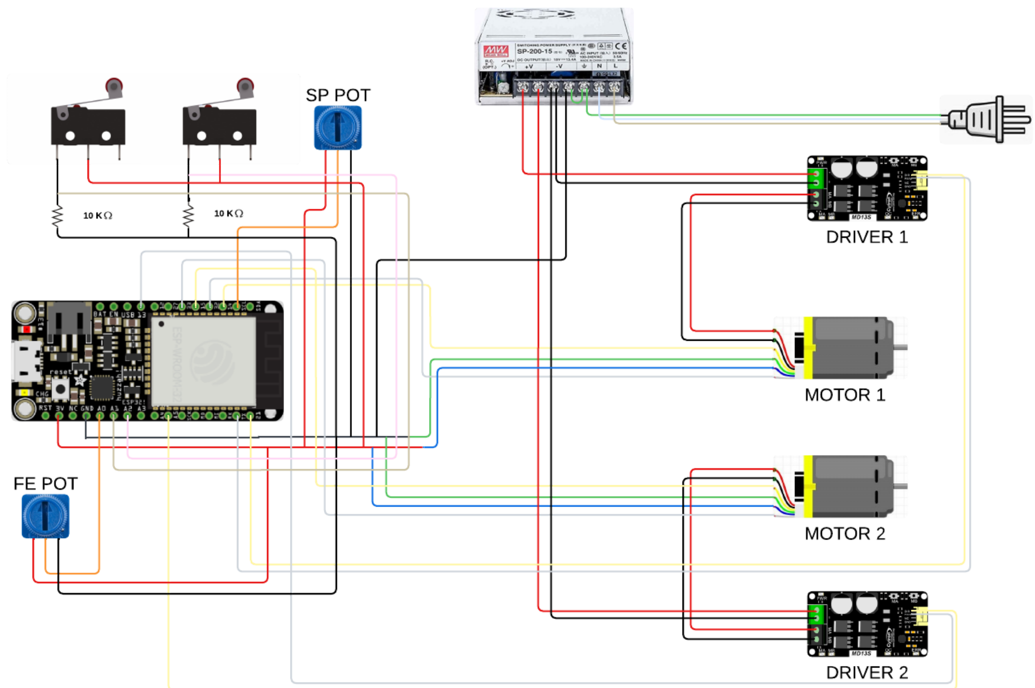
Using a pulley system meant that lateral forces experienced by our bearings would be due to radial gear forces and pulley preload forces. For the radial gear forces, we calculated the tangential force produced by the motor gear as $F_t = \frac{2T}{d}$ where T is the torque produced by the motor, and d is the diameter of the motor gear. The motor gear produces a radial force that can be calculated as $F_r = F_t tan(\alpha)$ where α is the pressure angle. Thus, the motor gear outputs a radial force of approximately 65.5 N. The pulley preload force is calculated as $F_{PRE} = \frac{T}{r}$ where T is the torque of the transmission gear and r is the radius of the pulley. The pulley preload force is calculated to be 333.3 N. Using a sum of forces, we calculated that each bearing would experience approximately 200 N. Since the bearings were rated for 845 N each, we were within the operating radial force limit of our bearings.



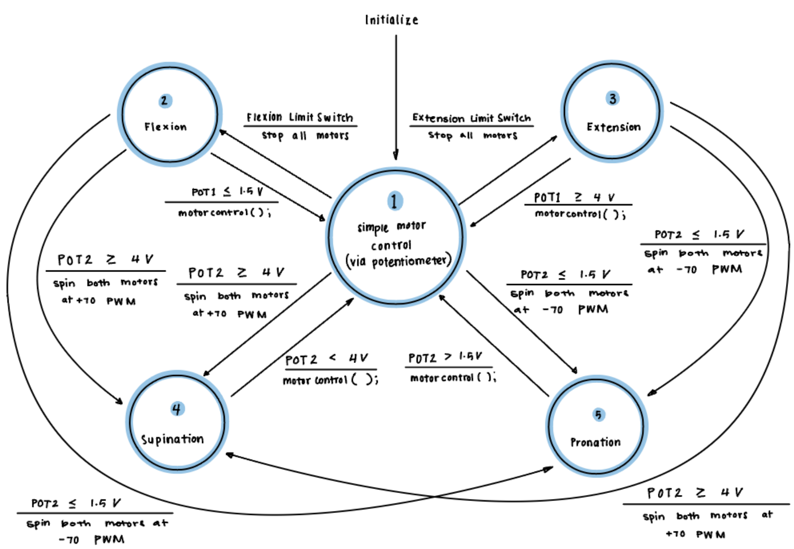
$$F_t = \frac{2T}{dp} = \frac{2(1.8Nm)}{(0.02m)} = 180N \qquad F_r = F_t \tan(\alpha) = (180N)\tan(20) = 65.51N$$

$$F_{pre} > \frac{2.7Nm}{0.0081m} > 333.33N \qquad \sum F_y = 0 = -200N + 333.33N + 66.67\,N - 200N$$

Circuit diagram



State transition diagram



Reflection:

One task we wish we had worked on sooner is the coding porting of this project. We spend a significant amount of time debugging our code, and ensuring the code executes the indeed actions. Another task we should have completed sooner is the ordering of the parts and assembling of the components. We encounter a few challenges during the assembly process such as securing the differential housing, and tolerance issues that cause there to be play in the differential housing support structure.

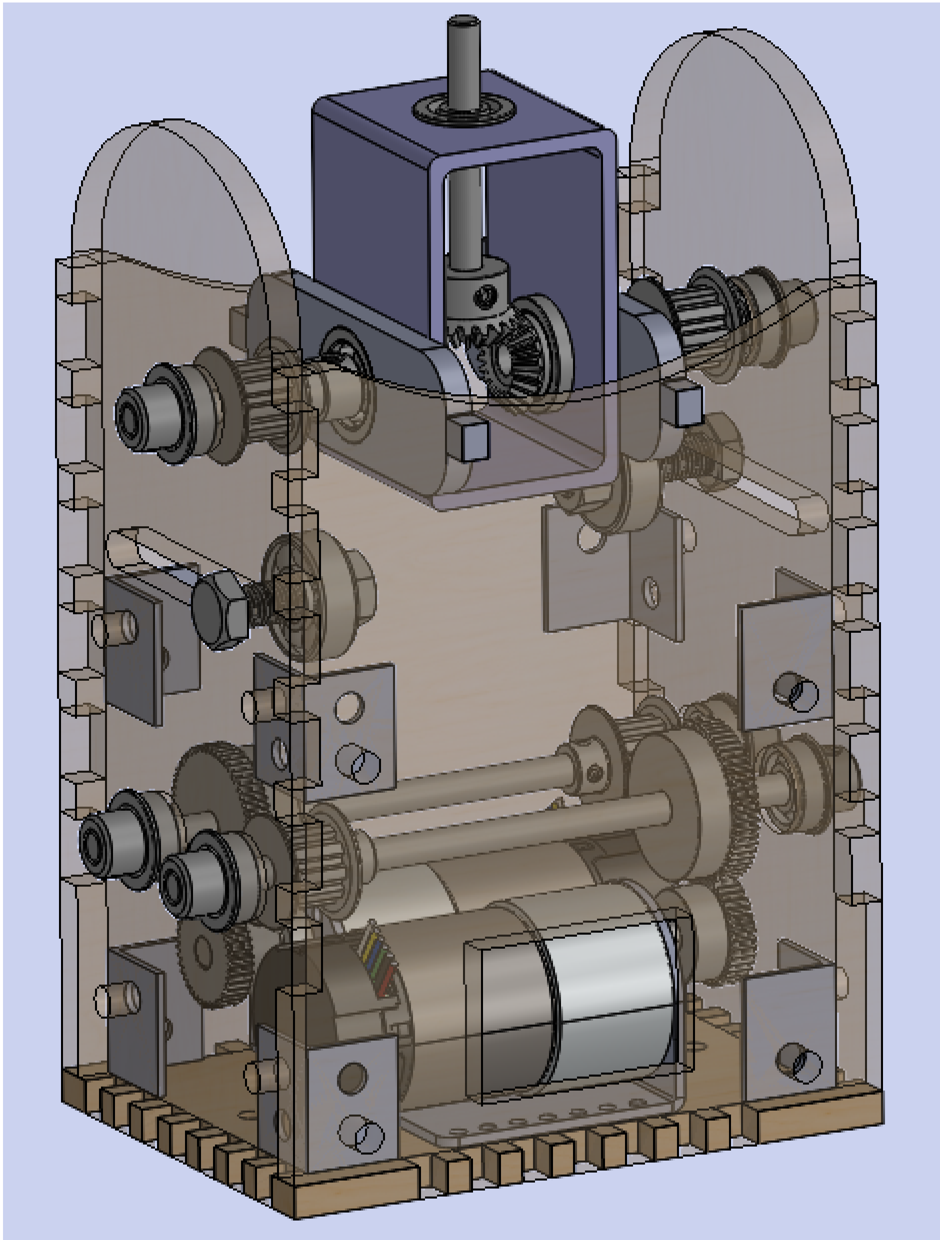
Appendix

Bill of Materials:

Vendor	Part No.	Description	Req.Qty	Order Qty	Unit measure	Price	Total
Berkeley Ace Hardware	N/A	Collar	1	1	Each	3.49	3.49
Berkeley Ace Hardware	N/A	Washer	1	1	Each	0.33	0.33
Berkeley Ace Hardware	5118088	Aluminum Angle 1/6"X3/4 X 48"	1	1	Each	9.59	9.59
Berkeley Ace Hardware	900086	Bearings	2	2	Each	14.99	29.98
Berkeley Ace Hardware	25106	Metric Hex Key Set	1	1	Set of 7 pc	4.99	4.99
Berkeley Ace Hardware	2299907	Cm Combination Wrench 8 mm	1	1	Each	7.59	7.59
Berkeley Ace Hardware	299881	Cm Combination Wrench 10 mm	1	1	Each	8.99	8.99
McMaster-Carr	6484K29 6	Timing Belt Pulley, XL Series, Trades Number 90xL031,5.6" Wide	2	2	Each	7.03	14.06
McMaster-Carr	91235A4 15	Belleville Spring Lock Washer, 18-8 stainless steel, for M6 screw size, 6.20mm ID, 14.300mm OD, Pack of 10	1	1	Set of 10 pc	12.91	12.91
McMaster-Carr	5905K76	Needle-Roller Bearing, Open, for 16 mm shaft diameter	2	2	Each	10.97	21.94
McMaster-Carr	6056N14	Carbon Steel Screw collar for 6mm Shaft diameter , DIN 705	1	1	Each	2	2
McMaster-Carr	2810N1	Plastic Miter gear, 0.5 Module	3	3	Each	3.2	9.6
McMaster-Carr	4138N71	1045 Carbon Steel Rotary shaft, 6mm diameter, 200 mm	3	3	Each	18.11	54.33

		long					
McMaster-Carr	1277N71 1	Corrosion-Resistant Timing Belt Pulley, XL Series, Hub, 2 Flags, 9.5 maximum Belt Width, 22mm OD	4	4	Each	12.83	51.32
McMaster-Carr	57155K4 81	Flanged Ball Bearing, Steel, open Trade Number 606	7	7	Each	9.2	64.4
McMaster-Carr	6056N14	Carbon Steel Screw collar for 6mm Shaft diameter , DIN 705	6	6	Each	2	12
McMaster-Carr	3560N13	Metal Miter Gear, 303 Stainless Steel, Round Bore, 1 Module, 20 teeth, 21.4 mm OD	3	3	Each	45.27	135.81
McMaster-Carr	2664N32 3	Metal Gear - 20 Degree Pressure Angle, Round with Set Screw, 0.5 Module, 40 teeth, 6 mm shaft	2	2	Each	23.21	46.42
McMaster-Carr	2664N32 8	Metal Gear - 20 Degree Pressure Angle, Round with Set Screw, 0.5 Module, 60 teeth	2	2	Each	25.06	50.12
TOTAL							539.87

<u>CAD</u>:



Code:

```
//Include Libraries ---------------------------------------------
#include·<ESP32Encoder.h>
#include·<Arduino.h>
#include <CytronMotorDriver.h>

//Define Pins ---------------------------------------------------
#define POT1 26      // Potentiomeneter reading acts as a substitute for flex/ext myolectric sensor
#define POT2 14      // Potentiomeneter reading acts as a substitute for sup/pro myolectric sensor
#define BTN_Flex 34  // Limit switch for Flexion
#define BTN_Ext 25   // Limit switch for Flexion

//Setup Interrupt Variables ------------------------------------
volatile bool flexionLimit = false;     // flexion flag
volatile bool extensionLimit = false;   // extension flag
volatile bool deltaT = false;           // encoder timer flag
int state = 1;                          // initialize state of the system as state 1 / Idle
hw_timer_t* timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;


//Define motors and variables -----------------------------------
int omegaMax = 20;  //CHANGE THIS VALUE

// Red Motor
CytronMD motor1(PWM_DIR, 17, 21);
ESP32Encoder encoder1;
// Green Motor
CytronMD motor2(PWM_DIR, 13, 4);
ESP32Encoder encoder2;

int count = 0;
int omega = 0;
```

```
int omega = 0;
int omegaDes = 0;
int D = 0;
int Kp = 3.5;


//Setting PWM Properties ----------------------------------------
const int MAX_PWM = 70;

//Initialize Interrupts -------------------------------------------
void IRAM_ATTR isrFlexion() {  // the function to be called when flexion interrupt is triggered
  flexionLimit = true;
}
void IRAM_ATTR isrExtension() {  // the function to be called when extension interrupt is triggered
  extensionLimit = true;
}
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  getEncoderCount();
  deltaT = true;  //function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

//Timer Initialization Function ---------------------------------
// initializing the timer for periodically checking the encoder count every 100 ms (when deltaT = true)
void TimerInit() {
  timer0 = timerBegin(0, 80, true);               // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80
  timerAttachInterrupt(timer0, &onTime0, true);  // edge (not level) triggered
  timerAlarmWrite(timer0, 10000, true);          // 10000 * 1 us = 10 ms, autoreload true
  timerAlarmEnable(timer0);                      // enable timer 0
}
```

```
//Setup -----------------------------------------------------------
void setup() {
  //Input pins
  pinMode(POT1, INPUT);
  pinMode(BTN_Flex, INPUT);
  pinMode(BTN_Ext, INPUT);
  pinMode(POT2, INPUT);
  setEncoderPins();

  //Interrupts
  attachInterrupt(BTN_Flex, isrFlexion, RISING);
  attachInterrupt(BTN_Ext, isrExtension, RISING);
  TimerInit();

  Serial.begin(115200);
}
void loop() {
  timerStart(timer0);
  if (deltaT) {
    Timer0Reset();
    //Map Potentiometer signal to speeds
    omega = count;
    omegaDes = map(analogRead(POT1), 0, 4095, -omegaMax, omegaMax);

    switch (state) {
      case 1:  // basic speed control
        // we start off with our motor speed controlled by the potentiometer
        motorControl();
        Serial.println("Welcome! Use a potentiometer to adjust your motor speed.");
        // If the hand flexes too far, we have to stop the motors from damaging the hardware
        if (CheckForFlexionLimit()) {
          motor2.setSpeed(0);
          motor1.setSpeed(0);
          state = 2;
        }
        // If the hand extends too far, we must also stop the motors from damaging the hardware
        if (CheckForExtensionLimit()) {
          motor2.setSpeed(0);
          motor1.setSpeed(0);
          state = 3;
        }
        // If POT2 is in HIGH voltage, we are in supination
        if (CheckforSupMyo() == true) {  // EVENT CHECKER
          state = 4;                     // SERVICE FUNCTION
        }
        if (CheckforProMyo() == true) {  // EVENT CHECKER
          state = 5;                     // SERVICE FUNCTION
        }
        break;

      case 2:  // flexion limit reached
        Serial.println("You've flexed as far as you can! You're so stronk.");
        if (map(analogRead(POT1), 0, 4095, 0, 5) <= 1.5) {  //I kind of prefer to use the value of D as an event checker inst
          motorControl();
          state = 1;
        }
        // If POT2 is in HIGH voltage, we are in supination
        if (CheckforSupMyo() == true) {  // EVENT CHECKER
          state = 4;                     // SERVICE FUNCTION
        }
        if (CheckforProMyo() == true) {  // EVENT CHECKER
          state = 5;                     // SERVICE FUNCTION
        }
```

```
    break;

  case 3:  // extension limit reached
    Serial.println("You've extended as far as you can! How flexible <.<");
    if (map(analogRead(POT1), 0, 4095, 0, 5) >= 4) {
      motorControl();
      state = 1;
    }
    // If POT2 is in HIGH voltage, we are in supination
    if (CheckforSupMyo() == true) {  // EVENT CHECKER
      state = 4;                     // SERVICE FUNCTION
    }
    if (CheckforProMyo() == true) {  // EVENT CHECKER
      state = 5;                     // SERVICE FUNCTION
    }
    break;

  case 4:
    if (CheckforSupMyo() == true) {  // EVENT CHECKER
      Serial.println("Wrist is in Supnation");
      motor1.setSpeed(70);
      motor2.setSpeed(70);
    }
    else if (map(analogRead(POT1), 0, 4095, 0, 5) >= 4) {
      motorControl();
      state = 3;
    }
    else if (map(analogRead(POT1), 0, 4095, 0, 5) <= 1.5) {
      motorControl();
      state = 2;
    }
    else {
```

```
          state = 1;
        }
        break;
      case 5:                                   //
        if (CheckforProMyo() == true) {  // EVENT CHECKER
          Serial.println("Wrist is in pronation");
          motor1.setSpeed(-70);
          motor2.setSpeed(-70);
        }
        else if (map(analogRead(POT1), 0, 4095, 0, 5) >= 4) {
          motorControl();
          state = 3;
        }
        else if (map(analogRead(POT1), 0, 4095, 0, 5) <= 1.5) {
          motorControl();
          state = 2;
        }
        else {
          state = 1;
        }
        break;
    }
  }
}

//Other Functions ----------------------------------------------------
void getEncoderCount() {
  count = encoder2.getCount();
  encoder2.clearCount();
}
void setEncoderPins() {
  ESP32Encoder::useInternalWeakPullResistors = UP;  // Enable the weak pull up resistors
```

```
  encoder1.attachHalfQuad(27, 33);                    // Attache pins for use as encoder pins
  encoder1.setCount(0);                               // set starting count value after attachi
  encoder2.attachHalfQuad(15, 32);                    // Attache pins for use as encoder pins
  encoder2.setCount(0);
}
void plotControlData() {
  Serial.print("Speed:");
  Serial.print(omega);
  Serial.print("  ");
  Serial.print("Desired_Speed:");
  Serial.print(omegaDes);
  Serial.print("  ");
  Serial.print("PWM:");
  Serial.println(D);
}
void motorControl() {
  int error = omegaDes - omega;
  D = Kp * error;

  if (D > MAX_PWM) {
    D = MAX_PWM;
  } else if (D < -MAX_PWM) {
    D = -MAX_PWM;
  }
  motor2.setSpeed(D);
  motor1.setSpeed(-D);
}
void Timer0Reset() {
  portENTER_CRITICAL(&timerMux0);
  deltaT = false;
  portEXIT_CRITICAL(&timerMux0);
}
```

```
  portEXIT_CRITICAL(&timerMux0);
}
bool CheckForFlexionLimit() {
  if (flexionLimit == true) {
    flexionLimit = false;
    return true;
  } else {
    return false;
  }
}
bool CheckForExtensionLimit() {
  if (extensionLimit == true) {
    extensionLimit = false;
    return true;
  } else {
    return false;
  }
}
bool CheckforSupMyo() {
  if (map(analogRead(POT2), 0, 4095, 0, 5) >= 4) {
    return true;
  } else {
    return false;
  }
}
bool CheckforProMyo() {
  if (map(analogRead(POT2), 0, 4095, 0, 5) <= 1.5) {
    return true;
  } else {
    return false;
  }
}
```