Winnie the Robot: A Robotic Drawing Arm

Aashika Nair, Mira Shah, Kimia Sattary, Marina Jew

Final Project Report– Group 2

MECENG 102B: Mechatronics Design

Department of Mechanical Engineering, University of California at Berkeley

Professor Hannah Stuart

December 19th, 2024

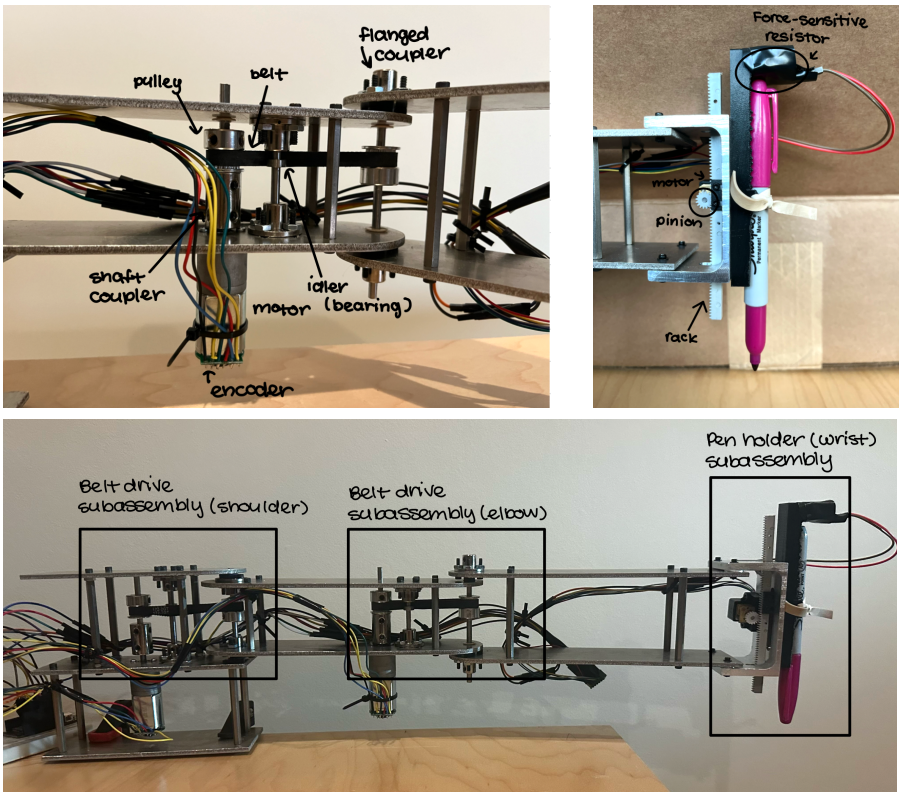
Opportunity

This project aimed to design a robot arm with two joints that can draw shapes by moving the motors to specified positions. Our motivation for this project was to build a machine that has the precision and control to draw shapes with more accuracy than humans, as a robot arm is capable of creating perfect arcs and perfectly straight lines. We intend for this to be a tool for people who wanted to create designs they would have trouble drawing on their own, allowing them to better express their creativity.

Strategy

Our base goal for this project was to have velocity control for all our motors, consistently draw an arc, and detect when the pen made contact with the paper. Our reach goal was to implement position control, program set positions, and use the arm to draw different shapes. Our strategy to address these objectives consisted of two major components. The first was a double-jointed arm with two motors that each had a belt drive system to actuate each joint of the arm. The second was a linearly-actuated wrist with a motor controlling a rack and pinion that lowers a pen until it makes contact with the drawing surface and raises the pen after the drawing is complete. We were able to complete our base goal. The double-jointed arm was controlled by two potentiometers, each controlled the speed of one motor, and we could draw arcs of different radii consistently. The linearly-actuated wrist had a set speed, and was stopped once a signal from a force sensitive resistor, installed near the top of the pen, was received.

Integrated Device



Figures 1, 2, & 3: These figures show the mechanical components and subsystems of the robotic arm

Function-Critical Decisions

Material: We chose Aluminium 6061-T1 for the material for the frame of our arm due to cost effectiveness, ease of machining, and yield strength. To choose the thickness, we conducted beam bending analysis. We started with the smallest thickness, 1/16”, available to us.

The calculation assumes the beam is a single unit with both arms, the point load includes only the metal bracket's weight, the cross-section with standoffs is approximated as a rectangular tube, and the uniform load accounts for the dead weight of the metal plates.

L = 400 mm, $W_{pen\ holder}$ = 0.050 kg × g = 0.1635 N, E = 70GPa, I = 338290 mm^4

$w_{plates} = density\ of\ Al\ *\ (\ width\ *\ thickness\ of\ plate)\ *\ g$

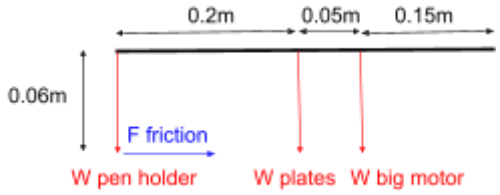
$= 0.0000027\ kg/mm^3 \times 3.175mm \times 50mm \times g\ =\ 4.2\ N/m$

$\delta_{max,\ point\ load} = \frac{W_{pen\ holder} L^3}{3EI} = 0.00014mm$ $\delta_{max,\ distributed\ load} = \frac{w_{plates} L^4}{8EI} = 0.00114mm$

$\delta_{max,\ total} = 0.00128mm$

With such a small deflection, we validated that 1/16’’ 6061 Al would be suitable for our project.

Motor choice: The motor for the shoulder joint experiences the highest load in this analysis. The forces acting on it primarily arise from the combined weight of the metal plates, the motor itself, and the pen holder section



$W_{plates} = 2\ *\ density\ of\ Al\ *\ (length\ *\ width\ *\ thickness\ of\ plate)\ *\ g\ =\ 1.68N$

$W_{pen\ holder} = W_{pen\ bracket} + W_{small\ motor} * g = 0.598N$

$W_{big\ motor} = mass\ *\ g\ =$0.461N

The frictional force exerted by the pen is estimated with a coefficient of friction of 0.5, based on previous studies. The normal force is approximated as half of the total dead weight of the arm.

$F_{friction} = \mu_R * N = 0.5 * 0.5 * (W_{plates} + W_{pen\ holder} + W_{big\ motor}) = 0.685N$

$\tau_{motor\ 1} = 0.15m * W_{big\ motor} + 0.2m * W_{plates} + 0.4m * W_{pen\ holder} + 0.06 * F_{friction}$

$= 0.685Nm$

Thus, we chose a motor with a stall torque value of 2.06Nm which gives us a factor of safety of 3. Since the motor on the shoulder bears the most load, we chose the stall torque for both shoulder and elbow motors based on the torque that the shoulder motor will experience.

The torque required for the motor to actuate the rack and pinion for the motion of the pen is determined by the weight of the rack and pen holder assembly and the diameter of the pinion’s pitch diameter.

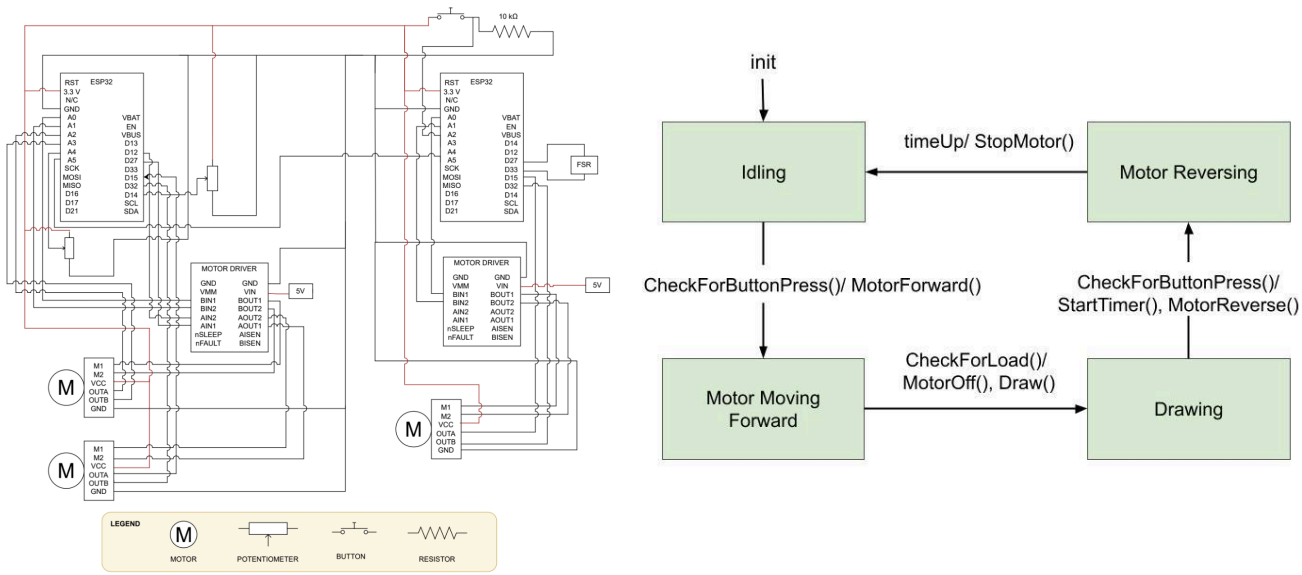
$\tau_{motor} = F\ *\ d\ =\ g\ *\ (M_{rack} + M_{pen\ holder} + M_{pen} + M_{fasteners})\ *\ r$

$\tau_{motor} = g\ *\ (0.002kg + 0.02kg + 0.01kg + 0.001\ kg)\ *\ 0.004\ m$

$\tau_{motor} = 0.00129\ Nm$

We determined based on this that the lab motors were able to provide sufficient torque for this purpose. They have a stall torque of 0.0078 Nm, which would give us a factor of safety larger than 2.

Circuit Diagram and State Diagram



Figures 4 & 5: Figure 4 (left) shows the full circuit diagram of our system. Figure 5 (right) shows the state diagram that we implemented into the code.

Reflection

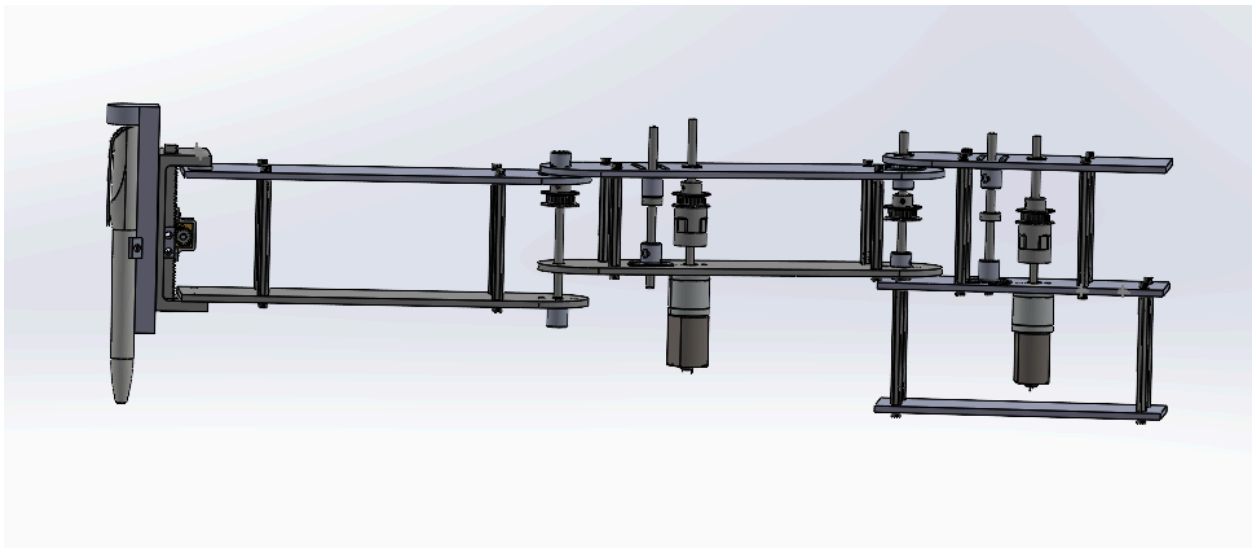
Overall, the execution of this project went relatively smoothly. The brainstorming and designing stage took more time than we had anticipated, but it was necessary to iterate through different design concepts. It was important for us to design easily manufacturable parts as we only had a semester to create our project. This allowed us to have time to remake parts, such as 3D printing our pen holder. When assembling the arm, components fit together well. However, we found that our mechanical tolerances were not tight enough to allow precise use of the arm. If we were to build this again, we would design parts with smaller tolerances and give ourselves more time to test and iterate mechanical subsystems. Ideally, we would test the movement and control of each section of the arm before integrating everything.

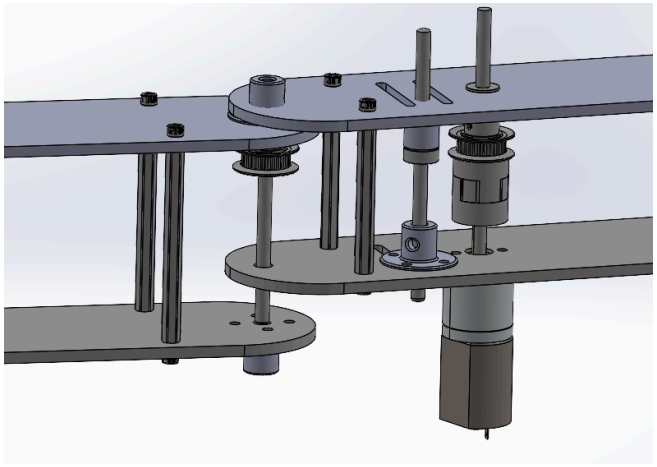
Appendix A: Bill of Materials

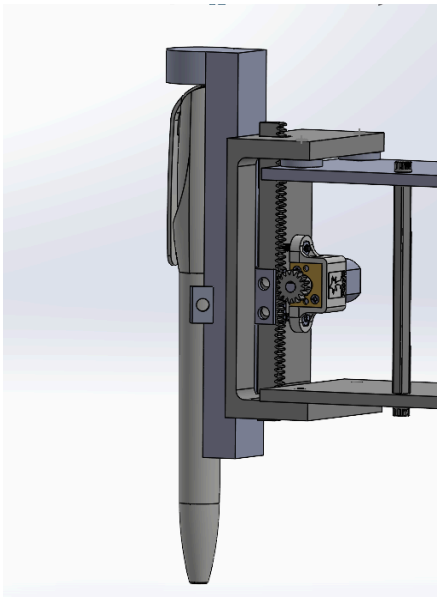
Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity
Motor Bracket	Pololu 20D mm Metal Gearmotor Bracket Pair	used for motor mounting	1138	$ 7.95	1
Encoder	Magnetic Encoder Pair Kit for 20D mm Metal Gearmotors, 20 CPR, 2.7-18V	encoders for arm motors	3499	$ 8.95	1
Motor Driver	TB6612FNG Dual Motor Driver Carrier	motor drivers for arm motors	713	$ 4.95	1
Gear Rack	20 Degree Pressure Angle Gear Rack, 0.5 Module	gear rack for pen motion	2662N55	$ 3.44	1
Plastic Gear	21 Degree Pressure Angle Plastic Gear, Round Bore, 0.5 Module, 16 Teeth Acetal	gear for pen motion	2662N312	$ 5.54	1
Sleeve Bearing	Light Duty Dry-Running Flanged Sleeve Bearing, Thermoplastic-Blend, for 4 mm Shaft Diameter, 3 mm Long	sleeve bearing to align shafts in plate and allow rotation	2705T113	$ 2.12	6
U-Channel	Multipurpose 6061 Aluminum U-Channel, 0.2" Leg x 0.13" Base Thickness, 3" Outside Width, 1/2 Foot Long	uchannel bracket for pen mounting	1630T31	$ 10.99	1
Hex Standoffs	Female Threaded Hex Standoff, 18-8 Stainless Steel, 3/16" Hex, 1-3/4" Long, 4-40 Thread	standoffs for between the plates (shorter)	91115A822	$ 3.32	4
Hex Standoffs	Female Threaded Hex Standoff, 18-8 Stainless Steel, 3/16" Hex, 2-1/4" Long, 4-40 Thread	standoffs for between the plates (longer)	91115A155	$ 4.00	12
4-40 Socket Head Screws	Black-Oxide Alloy Steel Socket Head Screw, 4-40 Thread Size, 1/2" Long, packs of 100	screws for standoffs, shaft couplers	91251A110	$ 11.96	1
4-40 Hex Nuts	Low-Strength Steel Hex Nut, Zinc-Plated, 4-40 Thread Size, packs of 100	nuts for 4-40 screws	90480A005	$ 1.10	1
4 mm Shaft	Rotary Shaft, 12L14 Carbon Steel, 4 mm Diameter, 400 mm Long	shafts for pulleys	1327K508	$ 12.40	1
Motors	391:1 Metal Gearmotor 20Dx46L mm 12V CB with Extended Motor Shaft	motors for arm with extended shaft for encoders	3496	$ 29.95	2
GT2 Pulley	WINSINN GT2 Pulley 20 Teeth 4mm bore 6mm Width 20T Timing Belt Pulley Wheel Aluminum for 3D Printer	pulley for belt drive	B07CXSSGF8	$ 5.99	1
Shaft Coupler	uxcell 4mm to 4mm Bore Rigid Coupling Set Screw L22XD14 Stainless Steel,Shaft Coupler Connector for 3D Printers, Motor Accessories,2pcs	to attach shafts to motor shafts	L22XD14	$ 9.99	1
4mm ID Bearings	uxcell MR84-2RS Deep Groove Ball Bearings 4mm Inner Dia 8mm OD 3mm Bore Double Sealed Chrome Steel Z2 10pcs	to use as idlers for the belt drive	MR84-2RS	$ 9.18	1
172 mm belt	BEMONOC 2GT Rubber Timing Belt 172-2GT-6 L=172mm W=6mm 86 Teeth in Closed Loop for 3D Printer Pack of 10pcs	belt for belt drive	B01A3Z05WE	$ 9.99	1
Power Supply	SmoTecQ 12V 2A Power Supply AC Adapter 2 Pack, AC 100-240V to DC 12 Volt Transformers, 2.1mm X 5.5mm Wall Plug	power supply for motors	n/a	$ 8.99	1
Sheet Metal	24" x 24" x 0.125" 6061 Aluminum Plate (Omax)	sheet metal to cut plates (split with another group)	n/a	$ 20.64	1
Loctite	Threadlocker Blue 242	loctite for set screws in shaft couplers to prevent loosening	B0001IRSNS	$ 7.68	1
Electronics Components	ESP, breadboard, wires, potentiometer, button, etc.	From ME 100 microkit (free)	n/a	$ -	1
Force Sensitive Resistor	Thin Film Pressure Sensor 20g-2Kg DF9-16	force sensor to detect if there is load on pen (contact with paper)	B07T1CHY58	$ 9.58	1
Sharpie	for drawing	already had (free)	n/a	$ -	2
Pen Holder	mount for attaching pen to rack and bracket	3d printed (free)	n/a	$ -	1
Motor Spacer Block	spacer to align motor with rack	3d printed (free)	n/a	$ -	1
Zipties & Electrical Tape	to organize wires	already had (free)	n/a	$ -	1

Figure 6: Shows the components and parts that we purchased and the justification behind the purchase.

Appendix B: CAD Screenshots







Figures 7, 8, & 9: Pictures of the fully completed CAD, accurate to what we manufactured. We modeled the CAD on Solidworks.

Appendix C: Code

Code for the ESP32's controlling the motors in the arm:

```
#include <ESP32Encoder.h>
#define BIN_1 26
#define BIN_2 25
#define AIN_1 27
#define AIN_2 12
#define LED_PIN 13
#define POT 14  // CHANGE POTENTIOMETER PIN HERE TO MATCH CIRCUIT IF NEEDED
#define POT2 36
#define FSR_SOURCE 32
#define FSR A3
#define ESP_1 4

ESP32Encoder encoder1;
ESP32Encoder encoder2;

int state = 0;
volatile bool signalSent = false;
volatile bool signalSent2 = false;
int omega1 = 0;
float omega2 = 0;
//int thetaMax = 455;  // 75.8 * 6 counts per revolution
int omegaMax = 25;
int D1 = 0;
int D2 = 0;
float potReading = 0;
float potReading2 = 0;

float omegaDes1 = 0;
float omegaDes2 = 0;

int list_length = 3;
int omegaDesList1[] = {0, 0, 0};
int omegaDesList2[] = {2, 0, 0};

int index1 = 0;
int index2 = 0;

float omegaDesired1 = omegaDesList1[index1];
float omegaDesired2 = omegaDesList2[index2];

bool omegaDesIsChanged = true;

int Kp_12V = 50;  // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int Ki_12V = 0.5;
int IMax = 0;
```

```
46  int sumerror1 = 0;
47  float error1;
48
49  int sumerror2 = 0;
50  float error2;
51  bool timeUp = false;
52
53  //Setup interrupt variables ----------------------------
54  volatile int count1 = 0;                  // encoder count
55  volatile int count2 = 0;                  // encoder count
56  volatile bool interruptCounter = false;  // check timer interrupt 1
57  volatile bool deltaT = false;            // check timer interrupt 2
58  int totalInterrupts = 0;                 // counts the number of triggering of the alarm
59  hw_timer_t* timer0 = NULL;
60  hw_timer_t* timer1 = NULL;
61  portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
62  portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
63
64  // setting PWM properties ------------------------------
65  const int freq = 5000;
66  const int ledChannel_1 = 1;
67  const int ledChannel_2 = 2;
68  const int resolution = 8;
69  const int MAX_PWM_VOLTAGE = 255;
70  const int NOM_PWM_VOLTAGE = 150;
71
72  //Initialization ---------------------------------------
73  void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
74    signalSent = true;
75  }
76  // void IRAM_ATTR isr2() {  // the function to be called when interrupt is triggered
77  //   signalSent2 = true;
78  // }
79
80  //Initialization ---------------------------------------
81  void IRAM_ATTR onTime0() {
82    portENTER_CRITICAL_ISR(&timerMux0);
83    interruptCounter = true;  // the function to be called when timer interrupt is triggered
84    omegaDesIsChanged = true;
85    //changeThetaDes();
86    //timeUp = true;
87    portEXIT_CRITICAL_ISR(&timerMux0);
88    //D2 = 0;
89    //moveMotor();
90
```

```
90
91  }
92
93  void IRAM_ATTR onTime1() {
94    portENTER_CRITICAL_ISR(&timerMux1);
95    count1 = encoder1.getCount();
96    encoder1.clearCount();
97    count2 = encoder2.getCount();
98    encoder2.clearCount();
99    if (index1 == 2){
100     encoder1.setCount(0);
101     encoder2.setCount(0);
102     omega1 = 0;
103     omega2 = 0;
104   }
105   deltaT = true;  // the function to be called when timer interrupt is triggered
106   portEXIT_CRITICAL_ISR(&timerMux1);
107   //Serial.println("In TIMERRRRR");
108   //Serial.print("Count 1: ");
109   //Serial.println(count1);
110 }
111
112 void setup() {
113   // put your setup code here, to run once:
114   pinMode(POT, INPUT);
115   pinMode(LED_PIN, OUTPUT);
116   pinMode(ESP_1, INPUT);
117   attachInterrupt(ESP_1, isr, RISING);
118   //attachInterrupt(ESP_1, isr2, FALLING);
119
120   digitalWrite(LED_PIN, LOW);  // sets the initial state of LED as turned-off
121   digitalWrite(FSR_SOURCE, HIGH);
122
123   Serial.begin(115200);
124   ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the weak pull up resistors
125   encoder1.attachHalfQuad(34, 39);                           // Attache pins for use as encoder pins
126   encoder1.setCount(0);                                      // set starting count value after attaching
127
128   encoder2.attachHalfQuad(15, 32);                           // Attache pins for use as encoder pins
129   encoder2.setCount(0);
130
131
132
133   // Method 1
134   // configure LED PWM functionalitites
```

```
131
132
133     // Method 1
134     // configure LED PWM functionalitites
135     // ledcSetup(ledChannel_1, freq, resolution);
136     // ledcSetup(ledChannel_2, freq, resolution);
137
138     // attach the channel to the GPIO to be controlled
139     // ledcAttachPin(BIN_1, ledChannel_1);
140     // ledcAttachPin(BIN_2, ledChannel_2);
141
142     // initilize timer
143     // timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE ->
        countUp
144     // timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
145     // timerAlarmWrite(timer0, 5000000, true); // 5000000 * 1 us = 5 s, autoreload true
146
147     // timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE ->
        countUp
148     // timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
149     // timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true
150
151     // at least enable the timer alarms
152     // timerAlarmEnable(timer0); // enable
153     // timerAlarmEnable(timer1); // enable
154
155     // Method 2
156     // configure PWM functionalitites with attaching the channel to the GPIO to be controller
157     ledcAttach(BIN_1, freq, resolution);
158     ledcAttach(BIN_2, freq, resolution);
159     ledcAttach(AIN_1, freq, resolution);
160     ledcAttach(AIN_2, freq, resolution);
161
162     // initialize timer
163     timer0 = timerBegin(1000000);            // Set timer frequency to 1Mhz
164     timerAttachInterrupt(timer0, &onTime0);  // Attach onTimer0 function to our timer.
165     //timerAlarm(timer0, 4000000, true, 0);    // 5000000 * 1 us = 5 s, autoreload true
166     timerAlarm(timer0, 10000000, true, 0);
167     timerStop(timer0);
168
169     timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
170     timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our timer.
171     timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload true
172
173 }
```

```
}

void loop() {

  switch (state) {

    case 0:
      Serial.println("S0");
      if (CheckForSignal() == true) {
        led_on();
        state = 1;
        timerRestart(timer0);
        timerStart(timer0);
        index1 = 0;
        index2 = 0;
        omegaDesired2 = omegaDesList2[index2];
      }
      break;
    case 1:
      Serial.println("S1");
      delay(1000);
      updateVars();

      if (omegaDesIsChanged) {
        Serial.println("THETA DES CHANGED");
        Serial.print("ThetaDesired: ");
        Serial.println(omegaDesired2);
        omegaDesIsChanged = false;
      }
      moveMotor();
      // if (CheckForSignal2() == true) {
      //   led_off();
      //   portENTER_CRITICAL_ISR(&timerMux0);
      //   timeUp = false;
      //   portEXIT_CRITICAL_ISR(&timerMux0);
      //   timerStop(timer0);
      //   D1 = 0;
      //   D2 = 0;
      //   moveMotor();
      //   state = 0;
      // }

      if (timeUp == true) {
        state = 0;
        led_off();
```

```
217             led_off();
218             Serial.println("stop");
219             portENTER_CRITICAL_ISR(&timerMux0);
220             timeUp = false;
221             portEXIT_CRITICAL_ISR(&timerMux0);
222             timerStop(timer0);
223
224
225
226
227         }
228         }
229
230     }
231     void updateVars(){
232
233       if (deltaT) {
234
235         portENTER_CRITICAL(&timerMux1);
236         deltaT = false;
237         portEXIT_CRITICAL(&timerMux1);
238
239         omega1 = count1;
240         omega2 = count2;
241
242         Serial.print("Omega1: ");
243         Serial.println(omega1);
244
245         potReading = analogRead(POT);
246         potReading2 = analogRead(POT2);
247
248         omegaDes1 = (-(float(potReading/4095) - 0.5) * 2) * 15;
249         omegaDes2 = (-(float(potReading2/4095) - 0.5) * 2) * 15;
250
251         // if (omegaDes1 > 0){
252         //   omegaDes1 += 100;
253         // }
254         // else if (omegaDes1 < 0){
255         //   omegaDes1 -= 100;
256         // } else {
257
258         // }
259
260         // if (omegaDes2 > 0){
261         //   omegaDes2 += 100;
```

```
261     //   omegaDes2 += 100;
262     // }
263     // else if (omegaDes2 < 0){
264     //   omegaDes2 -= 100;
265     // } else {
266
267     // }
268     Serial.print("OmegaDes1: ");
269     Serial.println(omegaDes1);
270
271   error1 = omegaDes1 - omega1;
272   sumerror1 = sumerror1 + error1/10;
273
274   int duty1 = Kp_12V*error1 + Ki_12V*sumerror1;
275
276   error2 = omegaDes2 - omega2;
277   sumerror2 = sumerror2 + error2/10;
278
279   int duty2 = Kp_12V*error2 + Ki_12V*sumerror2;
280
281   if (sumerror1 > 6){
282     sumerror1 = 6;
283     } else if (sumerror1 < -6){
284       sumerror1 = -6;
285     }
286
287   if (duty1 > MAX_PWM_VOLTAGE) {
288       duty1 = MAX_PWM_VOLTAGE;
289     } else if (duty1 < -MAX_PWM_VOLTAGE) {
290       duty1 = -MAX_PWM_VOLTAGE;
291     }
292
293   if (sumerror2 > 6){
294     sumerror2 = 6;
295     } else if (sumerror2 < -6){
296       sumerror2 = -6;
297     }
298
299   if (duty2 > MAX_PWM_VOLTAGE) {
300       duty2 = MAX_PWM_VOLTAGE;
301     } else if (duty2 < -MAX_PWM_VOLTAGE) {
302       duty2 = -MAX_PWM_VOLTAGE;
303     }
304   Serial.print("Duty1: ");
305   Serial.println(duty1);
```

sketch_dec4b_velocity_control.ino

```
305     Serial.println(duty1);
306     D1 = duty1;
307     D2 = duty2;
308     }
309   }
310
311   void moveMotor(){
312     Serial.println("In Move Motor");
313
314     if (D1 > 40) {
315         ledcWrite(BIN_1, LOW);
316         ledcWrite(BIN_2, D1);
317       } else if (D1 < -50) {
318         ledcWrite(BIN_2, LOW);
319         ledcWrite(BIN_1, -D1);
320       } else {
321         ledcWrite(BIN_2, LOW);
322         ledcWrite(BIN_1, LOW);
323       }
324
325     if (D2 > 40) {
326         ledcWrite(AIN_1, LOW);
327         ledcWrite(AIN_2, D2);
328       } else if (D2 < -50) {
329         ledcWrite(AIN_2, LOW);
330         ledcWrite(AIN_1, -D2);
331       } else {
332         ledcWrite(AIN_2, LOW);
333         ledcWrite(AIN_1, LOW);
334       }
335
336   }
337
338   void changeThetaDes() {
339
340     if (index1 < list_length) {
341
342       index1 = index1 + 1;
343       index2 = index2 + 1;
344
345     } else {
346
347       index1 = 0;
348       index2 = 0;
349     }
```

sketch_dec4b_velocity_control.ino

```
341
342       index1 = index1 + 1;
343       index2 = index2 + 1;
344
345     } else {
346
347       index1 = 0;
348       index2 = 0;
349     }
350
351     if (index1 == 2){
352       portENTER_CRITICAL_ISR(&timerMux0);
353       timeUp = true;
354       portEXIT_CRITICAL_ISR(&timerMux0);
355     }
356
357     omegaDesired2 = omegaDesList1[index1];
358     omegaDesired2 = omegaDesList2[index2];
359     D2 = 0;
360
361
362   }
363
364   bool CheckForSignal(){
365     if (signalSent == true) {
366       signalSent = false;
367       Serial.println("signal");
368       return true;
369     } else {
370       return false;
371     }
372
373
374   }
375
376
377
378   void led_on() {
379     digitalWrite(LED_PIN, HIGH);
380   }
381
382   void led_off() {
383     digitalWrite(LED_PIN, LOW);
384   }
385
```

Code for the ESP32 controlling the rack and pinion (includes state machine):

pen_code_speed_control.ino

```
#include <ESP32Encoder.h>
#define BIN_1 26
#define BIN_2 25
#define FSR 33
#define FSR_SOURCE 27
#define BTN 34
#define ESP2 4

ESP32Encoder encoder;

//Speed control variables
int omegaSpeed = 0;
int omegaDes = 0;
int omegaMax = 20;  // CHANGE THIS VALUE TO YOUR MEASURED MAXIMUM SPEED
int D = 0;
int dir = 1;
int potReading = 0;
float error =0;
float sum = 0;

int Kp = 50;  // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int Ki = 30;
int IMax = 0;

//Setup interrupt variables ----------------------------
volatile int count = 0;                 // encoder count
volatile bool interruptCounter = false;  // check timer interrupt 1
volatile bool deltaT = false;            // check timer interrupt 2
int totalInterrupts = 0;                 // counts the number of triggering of the alarm
hw_timer_t* timer0 = NULL;
hw_timer_t* timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 150;
//const int NOM_PWM_VOLTAGE = 100;
int motor_PWM;
int state = 1;

//setup variables
volatile bool buttonIsPressed = false;
int thresh = 3000;
int fsr volt;
```

pen_code_speed_control.ino

```
int fsr_volt;
bool timeUp = false;

// encoder properties ---------------------------------
int v = 0;

//Initialization -------------------------------------
void IRAM_ATTR onTime0() {
  //Serial.println("ON TIMEEEE");
  portENTER_CRITICAL_ISR(&timerMux0);
  timeUp = true;  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  encoder.clearCount();
  deltaT = true;  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
  //Serial.println("ISR BUTTONNN");
}

void setup() {
  //pinMode(LED_PIN, OUTPUT);    // configures the specified pin to behave either as an input or an output
  //digitalWrite(LED_PIN, LOW);  // sets the initial state of LED as turned-off
  pinMode(FSR_SOURCE, OUTPUT);
  pinMode(FSR, INPUT);
  pinMode(ESP2, OUTPUT);

  Serial.begin(115200);

  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the weak pull up resistors
  encoder.attachHalfQuad(15, 32);                           // Attache pins for use as encoder pins
  encoder.setCount(0);                                      // set starting count value after attaching

  digitalWrite(FSR_SOURCE, HIGH);
  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, FALLING);

  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);

  // initialize timer
```

pen_code_speed_control.ino

```
  // initialize timer
  timer0 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer0, &onTime0);  // Attach onTimer0 function to our timer0.
  timerAlarm(timer0, 500000, true, 0);    // 2000000 * 1 us = 2 s, autoreload true //aashika changed
  timerStop(timer0);

  timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our timer1.
  timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload true

  //initialize motor as off
  MotorOff();
}

void loop() {
  //delay(100);
  fsr_volt = analogRead(FSR);
  Serial.println(fsr_volt);

  switch(state){
    case 1:
      MotorOff();

      timeUp = false;
      Serial.println("in state 1");
      if(CheckForButtonPress()== true){
        state = 2;
      }
      break;
    case 2:
      timeUp = false;
      Serial.println("in state 2");
      MotorForward();
      if(CheckForLoad()== true){
        Serial.println("motor off");
        state = 3;
      }
      break;
    case 3:
      timeUp = false;
      Serial.println("in state 3");
      MotorOff();
      Draw();
      if(CheckForButtonPress()== true){
        state = 4;
        timerRestart(timer0);
        timerStart(timer0);
        Serial.println("");
```

pen_code_speed_control.ino

```
            Serial.println("");
            Serial.println("START TIMER");
          }
          break;
        case 4:
          Serial.println("in state 4");
          MotorReverse();
          if (timeUp == true) {
            MotorOff();
            Serial.println("TIME UP MOTOR STOP");
            portENTER_CRITICAL_ISR(&timerMux0);
            timeUp = false;
            portEXIT_CRITICAL_ISR(&timerMux0);
            timerStop(timer0);
            state = 1;
          }
          // if(CheckForButtonPress()== true){
          //   state = 1;
          // }
          break;
        default:
          //Serial.println("Error");
          break;

      }

    }

    bool CheckForButtonPress() {
      if (buttonIsPressed == true) {
        Serial.println("button is pressed");
        buttonIsPressed = false;
        return true;
      } else {
        return false;
      }
    }

    bool CheckForLoad() {
      if (fsr_volt > thresh) {
        return true;
      } else {
        return false;
      }
    }

    void MotorForward() {
      if (deltaT) {
```

pen_code_speed_control.ino

```
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    omegaSpeed = count;
    omegaDes = -10;  // PLEASE SPECIFY OMEGAMAX VALUE ABOVE

    // D = map(omegaDes, -omegaMax, omegaMax, -NOM_PWM_VOLTAGE, NOM_PWM_VOLTAGE);  // REPLACE THIS LINE WITH P/PI CONT
    error = omegaDes - omegaSpeed;
    sum += (error/10);

    //anti-windup section
    if (sum > 6){
      sum = 6;
    }
    else if (sum < -6){
      sum = -6;
    }

    D = Kp * error + Ki * sum;

    //Ensure that you don't go past the maximum possible command
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }
    Serial.print("D: ");
    Serial.println(D);
    if (D > 0) {
      ledcWrite(BIN_1, LOW);
      ledcWrite(BIN_2, D);
    } else if (D < 0) {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, -D);
    } else {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, LOW);
    }

  }
}

void MotorReverse() {
  //Serial.println("motor reversing");
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
```

pen_code_speed_control.ino

```
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    omegaSpeed = count;
    omegaDes = 10;  // PLEASE SPECIFY OMEGAMAX VALUE ABOVE

    //A6 CONTROL SECTION
    //Stand-in mapping between the pot reading and motor command.
    //CHANGE THIS SECTION FOR P AND PI CONTROL

    // D = map(omegaDes, -omegaMax, omegaMax, -NOM_PWM_VOLTAGE, NOM_PWM_VOLTAGE);  // REPLACE THIS LINE WITH
    error = omegaDes - omegaSpeed;
    sum += (error/10);

    //anti-windup section
    if (sum > 6){
      sum = 6;
    }
    else if (sum < -6){
      sum = -6;
    }

    D = Kp * error + Ki * sum;
    //Serial.print("error: ");
    //Serial.println(error);
    //Ensure that you don't go past the maximum possible command
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }
    Serial.print("D: ");
    Serial.println(D);
    if (D > 0) {
      ledcWrite(BIN_1, LOW);
      ledcWrite(BIN_2, D);
    } else if (D < 0) {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, -D);
    } else {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, LOW);
    }

  }
  digitalWrite(ESP2, LOW);
  Serial.println("give low");
```

Upload

```
283     Serial.println("give low");
284   }
285
286   void MotorOff() {
287     Serial.print("turn off motor");
288     ledcWrite(BIN_1, 0);
289     ledcWrite(BIN_2, 0);
290   }
291
292   void Draw(){
293     digitalWrite(ESP2, HIGH);
294     Serial.println("Drawing");
295   }
296
297
```