ME102B Final Project Report - sPrint

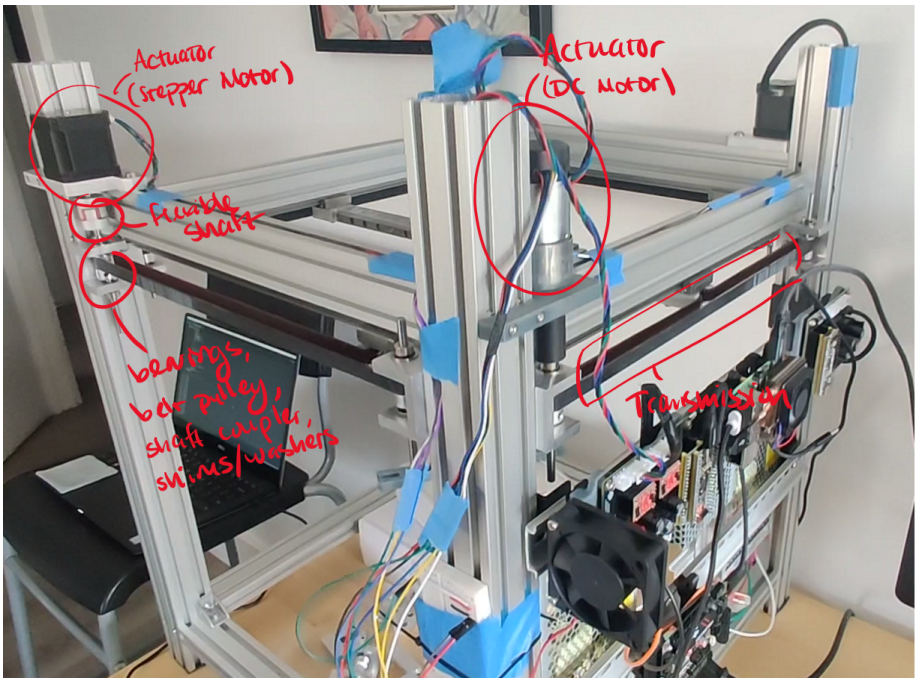
Anthony Chan, Chris Huang, Ty Schultz, Zac Gwennap

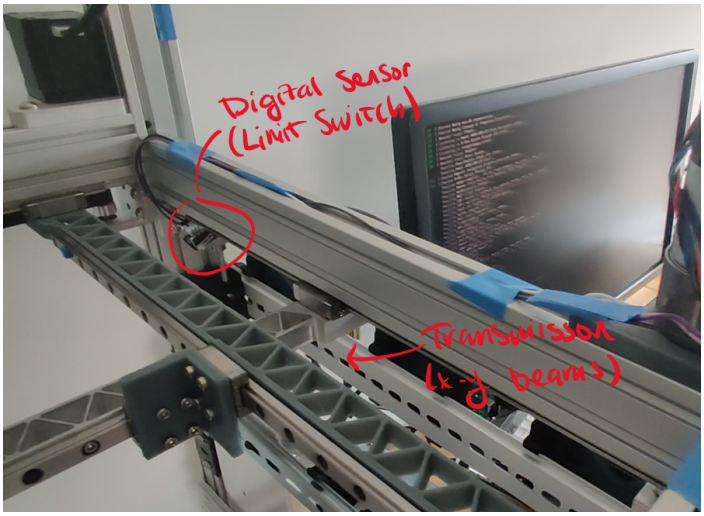
For our product, sPrint, we hope to build a rigid, high speed 3D printer that will save a lot of time and increase efficiency for rapid prototyping without diminishing overall print quality.

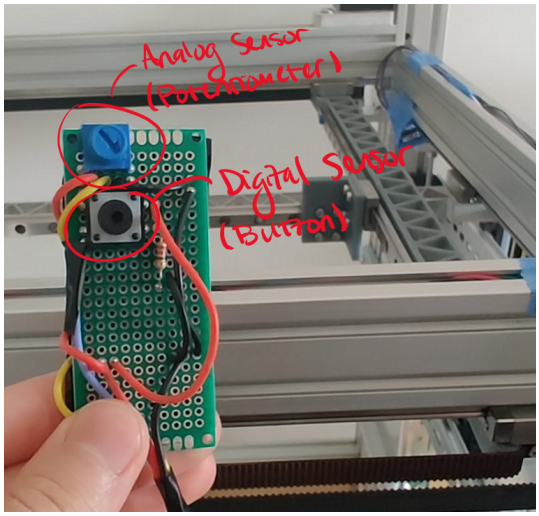
Our high-level strategy to achieve this goal is by increasing the stability of our frame and transmission system, while also optimizing our motors and power system by purchasing the right equipment and using software that can optimize pathing and acceleration. Ways that we hope to design to support these improvements are by shortening the transmission belts to reduce vibrations, by manufacturing all our parts to be custom and perfectly fit for our purpose, and by using two beams to move the printhead to reduce weight on the transmission. Some qualitative goals to hope to achieve initially were to reach a top speed of 1 m/s and a top acceleration of 9G's, which can be mutually exclusive. We surprised ourselves by attaining a top speed of 1.3 m/s and a top acceleration of 10G's, proving that the mechanical design choices were successful and we made the right decisions on our optimization and electrical hardware for our purpose.

Some initial desired functionalities that we achieved in our current machine include more powerful motorized, better path optimization, more robust transmission belts, and a sturdy frame to reduce vibration. Some functionalities we desired but have not yet implemented include additional mechanical trusses to reduce vibration, optimizing slicing software, and assembling the thermal chamber. \One goal we hoped to achieve was reaching 1 m/s max velocity, as well as a 9G acceleration (not mutual). After assembling our product and pushing the limits of our whole design, we were able to reach a max velocity of 1.3 m/s and a 10G acceleration, which surprised us as we managed to overshoot our goals.

Assembly Components







Critical Calculations

<u>Motor Max Acceleration (from max torque of desired stepper and DC motors)</u>

Nema (Stepper):

Nema max torque: 55 N-cm

2x motors → 110 N-cm max

Pulley gear diameter = 1.25 cm

Max force = $\frac{110\ N \cdot cm}{0.625\ cm}$ = 176 N → 0.6 safety = 105.6 N

Printhead and beam weight = 750 g = 0.75 kg

$F = ma \Rightarrow a_{max} = \frac{F}{m} = \frac{105.6\ N}{0.75\ kg} = 140\ m/s^2$ (theoretical max acceleration)

DFRobot (DC Motor):

DFRobot max torque = 18 kg-cm = 176.5 N-cm

2x motors → 353 N-cm

Pulley gear diameter = 1.25 cm

Max force = $\frac{353\ N \cdot cm}{1.25\ cm}$ = 282.4 N → 0.6 safety = 169.44 N

Printhead and beam weight = 750 g = 0.75 kg

$a_{max} = \frac{F}{m} = \frac{169.44\ N}{0.75\ kg} = 225.92\ m/s^2$ (theoretical max acceleration)

Conclusion: we will likely reach our goal of high acceleration even with additional factors such as friction and drag being in play.

<u>Bearing Max Load vs Belt Tension (bearing load vs ideal belt pre-tension)</u>

Bearing max radial load = 77.85 lbf

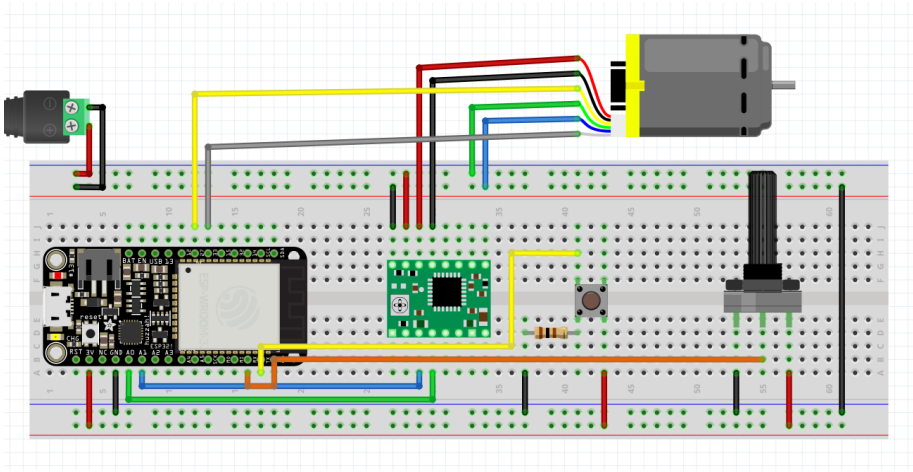
Common belt tensions: 6 to 8 lbf

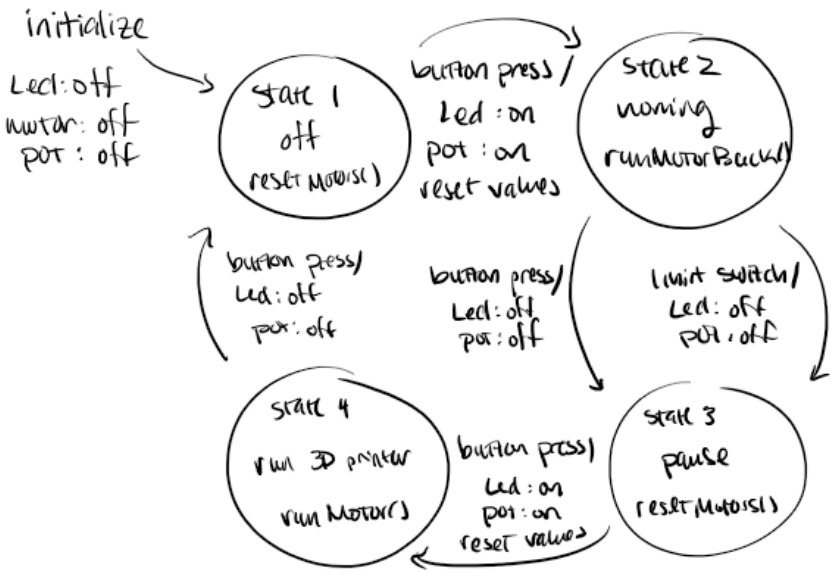
2x bearings → max load of 155.7lbf → 0.6 safety = 93.42 lbf

Belt tension of 8 lbf = 16 lbf on bearing
16 lbf << 93.42 lbf ∴ bearings will be fine

Calculations conclusion: The bearings can withstand the load of the belt pre-tension. We can reach ideal tensions without worrying about the structure of our motors and bearings, as they are within reason for the forces that will be applied on them.

Circuit and State Diagram:





For the Future

A couple strategies we would recommend for future students in the class that worked well for us is to get started early in designing, picking parts, and machining to help reduce stress later in the course. What we would have done differently if given this project again is ordering our parts/materials earlier as manufacturing can have mishaps, or ordering parts can go differently than planned, so we can have more time to correct for these mistakes before the deadlines.

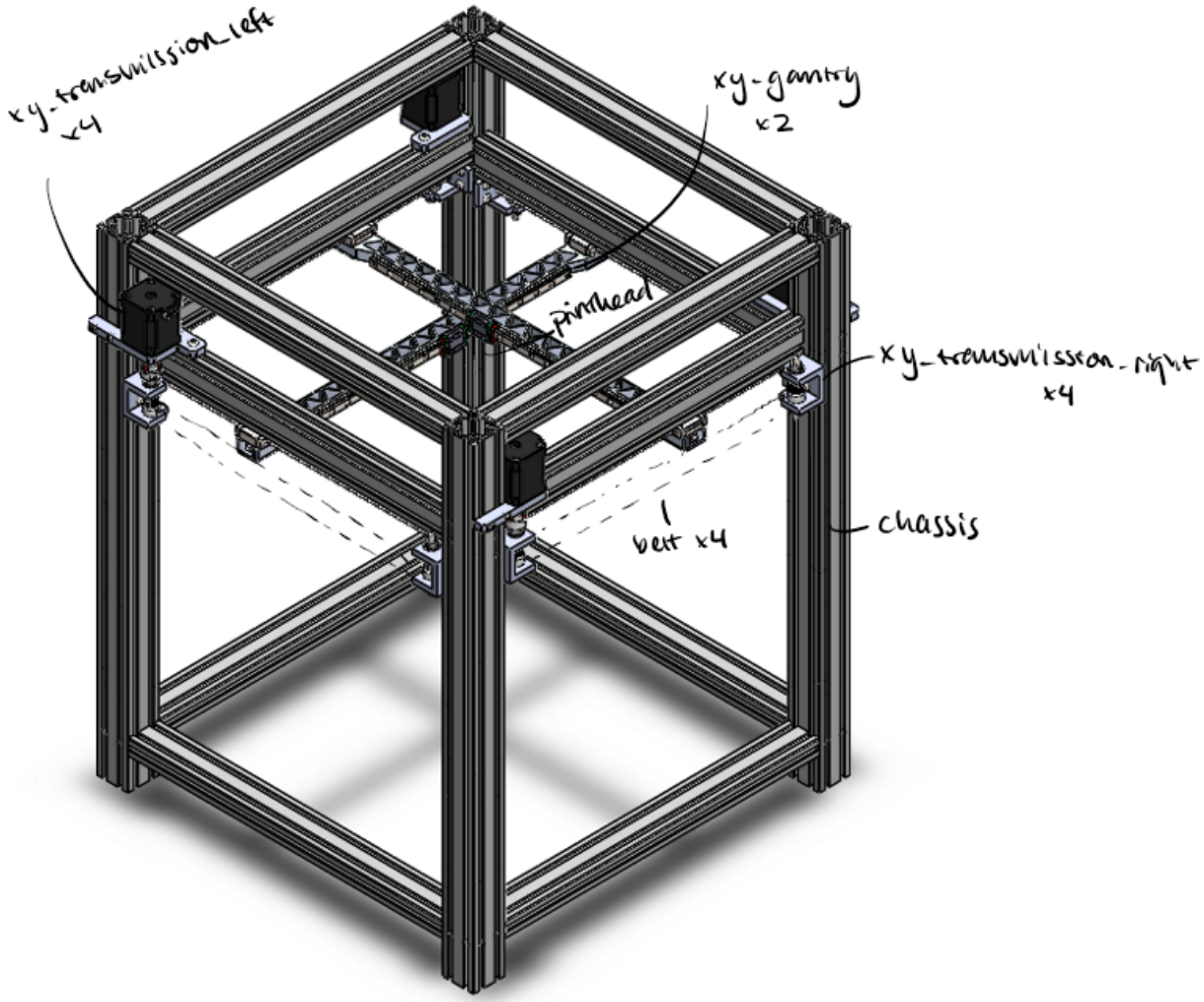
Appendices

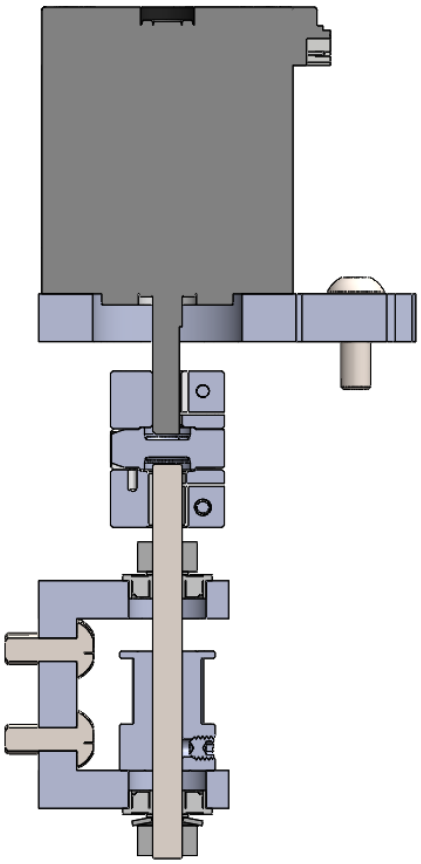
BOM

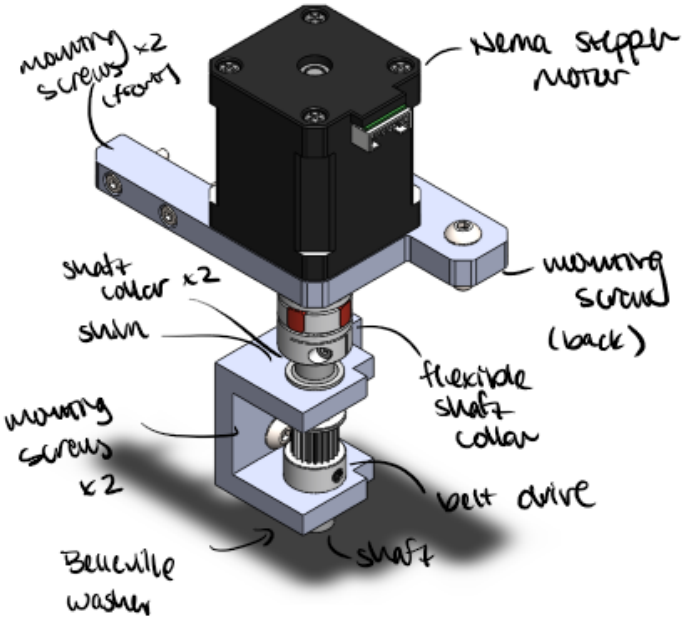
PART NUMBER	DESCRIPTION	QTY.	PRICE	TOTALS
HFS5-4040-600	4040 Extrusion	4	$ 15.66	$ 62.64
HFS5-2040-400-TPW	2040 Extrusion	12	$ 7.32	$ 87.84
MGN12_350	MGN12 Linear Rail	4	$ 33.25	$ 133.00
F695_bearing	Bearings	16	$ 1.15	$ 18.40
5x90mm_shaft	Shaft	8	$ 4.92	$ 39.36
5x5mm_Spider_Coupling	Coupler	4	$ 2.87	$ 11.48
stepper_mount_gantry_top	Machined 6061 Aluminum	4	$ 7.25	$ 29.00
GT2 Timing Pulley 20T 10mm	Timing Pulley	8	$ 2.98	$ 23.84
LDO-42STH48-2504AC	Stepper Motor	4	$ 33.95	$ 135.80
bearing_block_v2	Machined 6061 Aluminum	8	$ 3.75	$ 30.00
6056N13	Shaft Collar	16	$ 0.90	$ 14.40
91235A317	Belleville Spring Lock Washer	8	$ 0.11	$ 0.88
4M-10MM-GT2	GT2 10mm Belt	4	$ 4.19	$ 16.77
92334A113	Washer	8	$ 0.10	$ 0.80
B18.3.1M - 3 x 0.5 x 16 Hex SHCS -- 16NHX	Screw	16	$ -	$ -
B18.3.4M - 5 x 0.8 x 12 SBHCS --N	Screw	32	$ -	$ -
B18.3.5M - 3 x 0.5 x 12 Socket FCHS -- 12N	Screw	16	$ -	$ -
B18.3.4M - 5 x 0.8 x 16 SBHCS --N	Screw	4	$ -	$ -
XY_beam	Machined 6061 Aluminum	2	$ 19.75	$ 39.50
MGN12C Carriage	MGN12C Carriage	4	$ -	$ -
XY_belt_cover	Machined 6061 Aluminum	4	$ 1.25	$ 5.00
XY_belt_tensioner	Machined 6061 Aluminum	4	$ 1.00	$ 4.00
MGN9_Rail	MGN9 Linear Rail	2	$ 30.40	$ 60.80
B18.3.1M - 3 x 0.5 x 6 Hex SHCS -- 6NHX	Screw	12	$ -	$ -
B18.3.1M - 3 x 0.5 x 30 Hex SHCS -- 18NHX	Screw	4	$ -	$ -
printhead_mockup	Machined 6061 Aluminum	2	$ -	$ -
MGN9C Carriage	MGN9C Carriage	4	$ -	$ -
Raspberry Pi 4b	Raspberry Pi	1	$ 63.81	$ 63.81
Annex Supernova	Stepper Driver	2	$ 35.00	$ 70.00
Meanwell 48V 350W Power Supply	48V Power Supply	1	$ 34.47	$ 34.47
Meanwell 24V 200W Power Supply	24V Power Supply	1	$ 29.74	$ 29.74
DFRobot GB37Y3530-12V-251R	Brushed Motor	2	$ -	$ -
Mobilux EP2 Grease	Linear Rail Grease	1	$ 12.90	$ 12.90
				$ 924.43

CAD Screenshots

Note: certain items such as the transmission belt cannot be added in CAD as they are flexible components






mounting screws x2 (front)
Nema stepper motor
shaft collar x2
shim
mounting screw (back)
flexible shaft collar
mounting screws x2
belt drive
Belleville washer
shaft

Code Screenshots

```
#include <ESP32Encoder.h>
#define LIMIT 17
#define BTN 16
#define BIN_1 26
#define BIN_2 25
#define LED_PIN 13
#define POT 14

ESP32Encoder encoder;

// position values
int thetaPrev = 0;
int theta = 0;
int thetaDes = 0;
int thetaDel = 0;
int thetaMax = 455*19;   // 455
int DTheta = 0;

// velocity values
int omegaSpeedPrev = 0;
int omegaSpeed = 0;
int omegaAccel = 0;
int omegaDes = 0;
int omegaMax = 50;       // 19
int D = 0;
int dir = 1;
int potReading = 0;
float D_scale = 1.0;

// position control values
float KpTheta = 0.25;    // 0.25
float KiTheta = 0.01;    // 0.01
float KdTheta = 2;       // 2
float errorTheta = 0;
float errorThetaSum = 0;

// velocity control values
float KpOmega = 15;      // 15
float KiOmega = 0.01;    // 0.01
float KdOmega = 6;       // 6
float errorOmega = 0;
float errorOmegaSum = 0;

```

```
// buttons
volatile bool limitIsPressed = false;
volatile bool buttonIsPressed = false;
int state = 1;

//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;
// hw_timer_t * timer2 = 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 = 255;  // Max = 255
const int NOM_PWM_VOLTAGE = 150;

//Initialization -----------------------------------
// button interrupt
void IRAM_ATTR isr() {
  buttonIsPressed = true;
  timerStart(timer0);
}

// limit switch interrupt
void IRAM_ATTR isr1() {
  limitIsPressed = true;
}

// void IRAM_ATTR onTime0() {
//   portENTER_CRITICAL_ISR(&timerMux0);
//   interruptCounter = true; // the function to be called when timer interrupt is triggered
//   portEXIT_CRITICAL_ISR(&timerMux0);
// }

```

```
// deltaT
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);
}

// debounce
void IRAM_ATTR onTime0() { // end timer
  timerStop(timer0);     // stop debounce timer
}

// setup
void setup() {
  // put your setup code here, to run once:
  // onboard setup
  pinMode(POT, INPUT);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // sets the initial state of LED as turned-off
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  attachInterrupt(BTN, isr, RISING); // CHANGE -> RISING
  pinMode(LIMIT, INPUT);
  attachInterrupt(LIMIT, isr1, RISING);

  // serial + encoder
  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
  encoder.attachHalfQuad(33, 27); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);

```

```
  // initilize timer
  // timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  // timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  // timerAlarmWrite(timer0, 5000000, true); // 5000000 * 1 us = 5 s, autoreload true

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

  timer0 = timerBegin(0, 80, true);
  timerAttachInterrupt(timer0, &onTime0, true);
  timerAlarmWrite(timer0, 500000, true);        // 500 ms debounce

  // enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable
  // timerAlarmEnable(timer2);

  // stop debounce timer
  timerStop(timer0);
}

void loop() {
    switch (state) {
      case 1:  // off state
        Serial.println("State 1: Off State");
        if (CheckForButtonPress()) {
          // Services
          led_on();
          theta = 0;
          errorThetaSum = 0;
          errorOmegaSum = 0;
          limitIsPressed = false;
          state = 2;
        }
        break;
```

```
      case 2:  // init state
        Serial.println("State 2: Initialize (if necessary)");
        runMotorBack();
        if (CheckForButtonPress()) {
          // Services
          led_off();
          limitIsPressed = false;
          state = 3;
        } else if (CheckForLimitPress()) {
          // Services
          led_off();
          limitIsPressed = false;
          state = 3;
        }
        break;

      case 3:  // off state
        Serial.println("State 3: Reset State");
        resetMotor();
        if (CheckForButtonPress()) {
          // Services
          led_on();
          theta = 0;
          errorThetaSum = 0;
          errorOmegaSum = 0;
          limitIsPressed = false;
          state = 4;
        }
        break;

      case 4:  // on state
        Serial.println("State 4: Motor Controlled by Potentiometer");
        runMotor();
        if (CheckForButtonPress()) {
          // Services
          led_off();
          limitIsPressed = false;
          state = 1;
        }
        break;
  }
}

```

```
// Other functions

// Event Checker Function
bool CheckForButtonPress() {
  // with debouce
  if (timerStarted(timer0)) {
    return false;
  } else {
    if (buttonIsPressed){
      buttonIsPressed = false;
      return true;
    } else {
      return false;
    }
  }
}

// Limit Switch Checker Function
bool CheckForLimitPress() {
  if (limitIsPressed && digitalRead(LIMIT) == 1) {
    limitIsPressed = false;
    return true;
  } else {
    return false;
  }
}

// Main Service Function
void runMotor() {
  if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      // Position
      theta += count;
      potReading = analogRead(POT);
      thetaDes = (map(potReading, 0, 4095, 0, thetaMax));
      thetaDel = thetaPrev - theta;
      thetaPrev = theta;

```

```
    errorTheta = thetaDes - theta;
    // Noise Reduction
    if (errorTheta < 0.04*thetaMax && errorTheta > -0.04*thetaMax) {
      errorTheta = 0;
    }
    errorThetaSum += errorTheta;

    // Anti-Windup
    if (errorThetaSum > 300) {
      errorThetaSum = 300;
    } else if (errorThetaSum < -300) {
      errorThetaSum = -300;
    }

    // Position Value (PID)
    DTheta = ((KpTheta * errorTheta) + (KiTheta * errorThetaSum) + (KdTheta * thetaDel));

    // Velocity
    omegaSpeed = count;
    omegaAccel = omegaSpeed - omegaSpeedPrev;
    omegaSpeedPrev = omegaSpeed;
    omegaDes = DTheta;

    errorOmega = omegaDes - omegaSpeed;
    errorOmegaSum += errorOmega;

    // Anti-Windup
    if (errorOmegaSum > 400) {
      errorOmegaSum = 400;
    } else if (errorOmegaSum < -400) {
      errorOmegaSum = -400;
    }

    // Velocity Value (PID)
    D = ((KpOmega * errorOmega) + (KiOmega * errorOmegaSum) + (KdOmega * omegaAccel));

    //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;
    }

```

```
292         //Map the D value to motor directionality
293         //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
294         if (D > 0) {
295             ledcWrite(ledChannel_1, LOW);
296             ledcWrite(ledChannel_2, D);
297         }
298         else if (D < 0) {
299             ledcWrite(ledChannel_1, -D);
300             ledcWrite(ledChannel_2, LOW);
301         }
302         else {
303             ledcWrite(ledChannel_1, LOW);
304             ledcWrite(ledChannel_2, LOW);
305         }
306 
307         plotControlData();
308     }
309 }
310 
311 // Initialization Setup
312 void runMotorBack() {
313     if (deltaT) {
314         portENTER_CRITICAL(&timerMux1);
315         deltaT = false;
316         portEXIT_CRITICAL(&timerMux1);
317 
318         // Position
319         theta += count;
320         potReading = analogRead(POT);
321         thetaDes = (map(potReading, 0, 4095, 0, -thetaMax)); // flip dir
322         thetaDel = thetaPrev - theta;
323         thetaPrev = theta;
324 
325         errorTheta = thetaDes - theta;
326         // Noise Reduction
327         if (errorTheta < 0.04*thetaMax && errorTheta > -0.04*thetaMax) {
328             errorTheta = 0;
329         }
330         errorThetaSum += errorTheta;
331 
```

```
    // Anti-Windup
    if (errorThetaSum > 300) {
      errorThetaSum = 300;
    } else if (errorThetaSum < -300) {
      errorThetaSum = -300;
    }

    // Position Value (PID)
    DTheta = ((KpTheta * errorTheta) + (KiTheta * errorThetaSum) + (KdTheta * thetaDel));

    // Velocity
    omegaSpeed = count;
    omegaAccel = omegaSpeed - omegaSpeedPrev;
    omegaSpeedPrev = omegaSpeed;
    omegaDes = DTheta;

    errorOmega = omegaDes - omegaSpeed;
    errorOmegaSum += errorOmega;

    // Anti-Windup
    if (errorOmegaSum > 400) {
      errorOmegaSum = 400;
    } else if (errorOmegaSum < -400) {
      errorOmegaSum = -400;
    }

    // Velocity Value (PID)
    D = ((KpOmega * errorOmega) + (KiOmega * errorOmegaSum) + (KdOmega * omegaAccel));

    //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;
    }
```

```
369         //Map the D value to motor directionality
370         //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
371         if (D > 0) {
372             ledcWrite(ledChannel_1, LOW);
373             ledcWrite(ledChannel_2, D);
374         }
375         else if (D < 0) {
376             ledcWrite(ledChannel_1, -D);
377             ledcWrite(ledChannel_2, LOW);
378         }
379         else {
380             ledcWrite(ledChannel_1, LOW);
381             ledcWrite(ledChannel_2, LOW);
382         }
383 
384         plotControlData();
385     }
386 }
387 
388 // Reset Motor to 0
389 void resetMotor() {
390     if (deltaT) {
391         portENTER_CRITICAL(&timerMux1);
392         deltaT = false;
393         portEXIT_CRITICAL(&timerMux1);
394 
395         // reset values
396         thetaPrev = 0;
397         theta = 0;
398         thetaDes = 0;
399         thetaDel = 0;
400         thetaMax = 455*20;   // 455
401         DTheta = 0;
402 
403         // velocity values
404         omegaSpeedPrev = 0;
405         omegaSpeed = 0;
406         omegaAccel = 0;
407         omegaDes = 0;
408         omegaMax = 50;       // 19
409         potReading = 0;
410 
```

```
411         D = 0;
412
413         //Ensure that you don't go past the maximum possible command
414         if (D > MAX_PWM_VOLTAGE) {
415             D = MAX_PWM_VOLTAGE;
416         }
417         else if (D < -MAX_PWM_VOLTAGE) {
418             D = -MAX_PWM_VOLTAGE;
419         }
420
421         //Map the D value to motor directionality
422         //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
423         if (D > 0) {
424             ledcWrite(ledChannel_1, LOW);
425             ledcWrite(ledChannel_2, D);
426         }
427         else if (D < 0) {
428             ledcWrite(ledChannel_1, -D);
429             ledcWrite(ledChannel_2, LOW);
430         }
431         else {
432             ledcWrite(ledChannel_1, LOW);
433             ledcWrite(ledChannel_2, LOW);
434         }
435
436         plotControlData();
437     }
438 }
439
440 // LED on
441 void led_on() {
442   digitalWrite(LED_PIN, HIGH);
443 }
444
445 // LED off
446 void led_off() {
447   digitalWrite(LED_PIN, LOW);
448 }
449
```

```
void plotControlData() {
  // Serial.print("Test Value:");
  // Serial.print(digitalRead(LIMIT));
  // Serial.print(" ");
  // Serial.println();
  // Serial.print("Test Value:");
  // Serial.print(limitIsPressed);
  // Serial.print(" ");
  // Serial.println();
  // Serial.print("errorOmega/20:");
  // Serial.print(errorOmega/20);
  // Serial.print(" ");
  // Serial.print("errorOmegaSum/20:");
  // Serial.print(errorOmegaSum/20);
  // Serial.print(" ");
  // Serial.print("Speed:");
  // Serial.print(omegaSpeed);
  // Serial.print(" ");
  // Serial.print("Desired_Speed:");
  // Serial.print(omegaDes);
  // Serial.print(" ");
  // Serial.print("errorTheta/20:");
  // Serial.print(errorTheta/20);
  // Serial.print(" ");
  // Serial.print("errorThetaSum/20:");
  // Serial.print(errorThetaSum/20);
  // Serial.print(" ");
  // Serial.print("Position:");
  // Serial.print(theta);
  // Serial.print(" ");
  // Serial.print("Desired_Position:");
  // Serial.print(thetaDes);
  // Serial.print(" ");
  // Serial.print("DTheta:");
  // Serial.print(DTheta);
  // Serial.print(" ");
  // Serial.print("PWM_Duty/10:");
  // Serial.println(D/10);  //PWM is scaled by 1/10 to get more intelligible graph
}
```