Little Drummer Bot

ME102B Fall 2024 Final Report

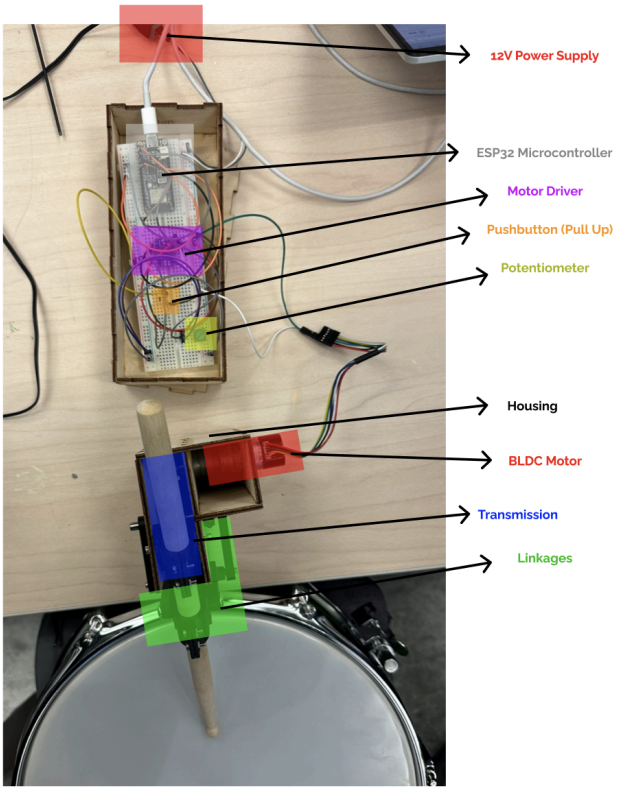
Group 12: Jasper Hsu, Meghna Sharma, Layne Werle

Opportunity: Our project goal was to create a robotic device that assists people with limited mobility/ability in playing the drums.

High-Level Strategy

The high-level goal of our project was to create a robotic device that controls a drumstick that plays a drum. The mechanical system is powered by our ESP32 that drives the 12V DC motor when the button is pushed. A small gear is attached to the motor which rotates a larger gear in a ratio of 2:1. A keyed shaft holds the large gear in place as it spins, which drives the keyed shaft, which drives the 2 horizontal linkages, which drives the 1 vertical linkage, which pivots the drumstick downward (as it is held fixed at the pivot point of the further back drumstick holder). The final component is the springs attached to the bottom of the linkage. Incorporating these into our design adds elasticity to the vertical linkage drive to replicate a 'bounce' motion.

Assembled Device



Our initial desired functionality was to achieve consistent, rhythmic tapping of a singular drumstick which we were able to produce. With more time, we hoped to have incorporated a servo motor attached to the bottom of our housing in order to be able to rotate the mechanism in order to play multiple drums. In addition, adding more sensors/actuation methods would allow us to have more states such as pre programmed beats or songs.

Function-Critical Decisions

To ensure the machine can provide beats for most songs, we aim for a maximum clicking speed of over 200 BPM (beats per minute). Therefore, we need to calculate the required speed and torque of the motor to achieve this performance.

The mass and length of the drumstick are approximately: $l = 25cm,\ m = 50g$.

Assuming the drumstick's rotation axis is at the center, its moment of inertia is: $I = \frac{1}{12}ml^2$

The angle that the shaft needs to rotate from the clicking position to the highest lifting position is about: $\alpha = 45°$.

The Gear ratio provided by gear transmission is: $K = 2$.

The motor needs to rotate an angle of $\frac{\alpha}{K}$ twice for each click, which needs to be done in $\frac{1}{BPM}$ min.

Therefore, the average speed required for the motor is: $\omega_r = \frac{2\alpha}{K} \cdot BPM = 25rpm$.

Assuming the motor undergoes uniform acceleration, it should accelerate to $2\omega_r$ within $t_a = \frac{1}{2BPM}$ min.

According to the impulse-momentum theorem($T \cdot t = I \cdot \Delta\omega$), the required torque of the motor is:

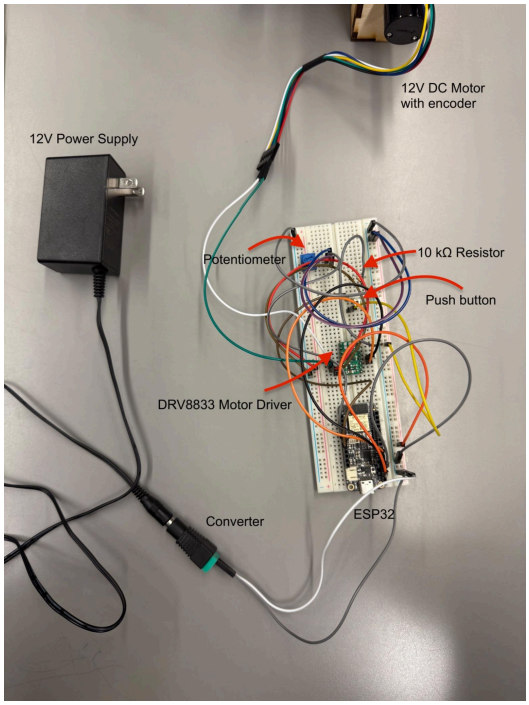
$T_r = \frac{2KI\omega_r}{60/t_a} = \frac{2K\frac{1}{12}ml^2\omega_r}{60/BPM/2}$ =0.0182Nm=0.19Kgcm.

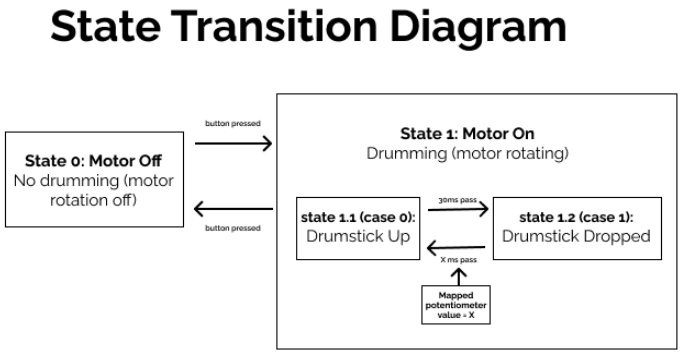
The motor we use has no load speed of 600 rpm and no speed torque of 0.25kg cm. The speed of the motor under required torque is:

$\omega = 600\frac{0.25-0.19}{0.25} = 144rpm > \omega_r$.

Therefore, the motor we selected can meet our speed requirements.

Diagrams (Circuit and State Transition)





Reflection

Upon reflection, we are glad we decided to hone in on one degree of freedom and make that function with high quality, instead of aiming for more quantity in functions; it resulted in a more technically valid product after receiving more of our focus. We also wish we had looked for more research and inspiration in P1/P2 though to find a more exciting idea that we're all challenged by!

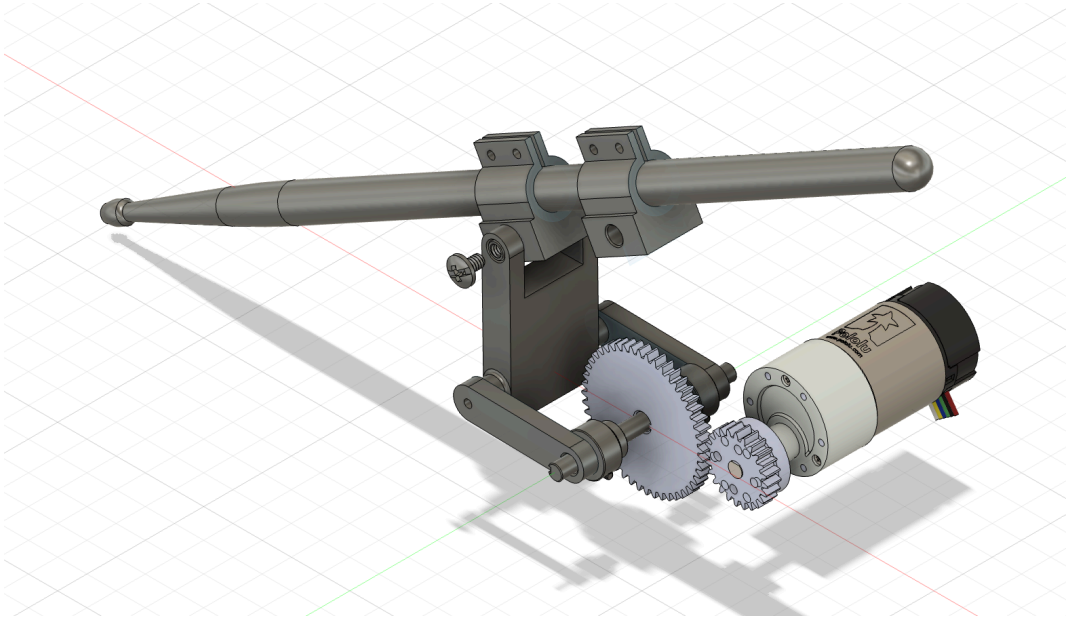
Appendices

Bill of Materials (BOM)

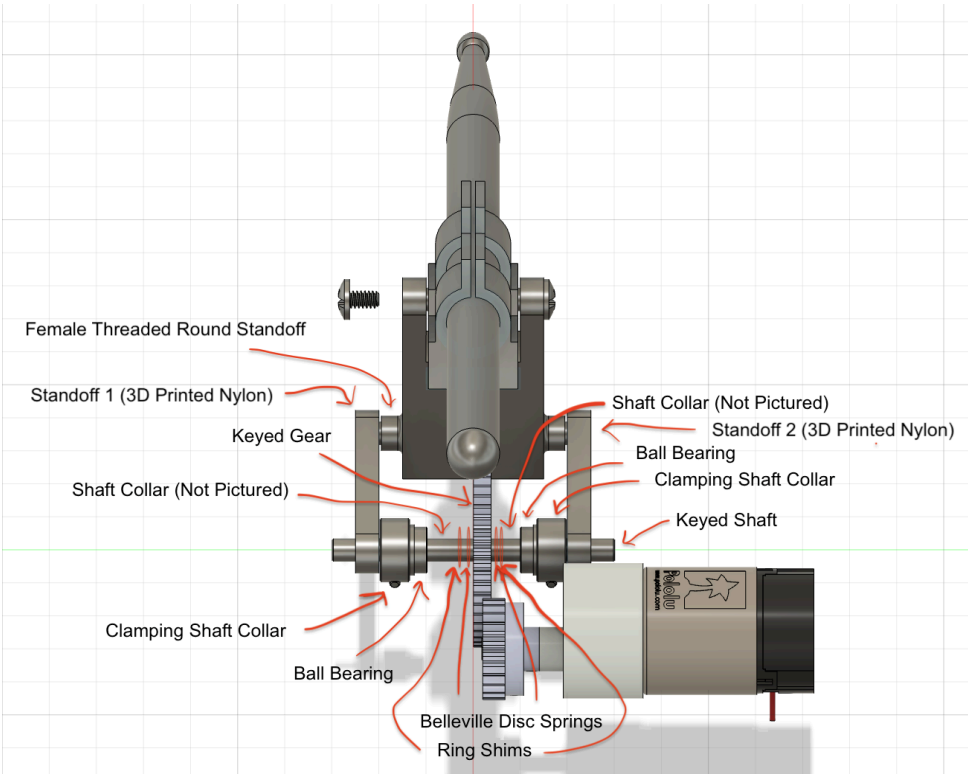
Little Drummer Bot's Bill of Materials								Total (Projected):	$ 263.86
Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity	Vendor	Link to Item	Notes	Subtotal
1/4"D 4-40 2"L Round Standoff	descriptions available in product links	for transmission	91125A473	$4.94	1	McMaster Carr	https://www.mcr		$ 4.94
1/4"D 3"L Keyed Rotary Shaft		for transmission	8488T1	$16.50	1	McMaster Carr	https://www.mcr		$ 16.50
1/4"D 4-40 1"L Round Standoff		for transmission	91125A439	$4.11	1	McMaster Carr	https://www.mcr		$ 4.11
Stainless Steel Ball Bearing 1/4"		for transmission	57155K323	$5.70	2	McMaster Carr	https://www.mcr		$ 11.40
Clamping Shaft Collar 1/4"		for transmission	6435K12	$5.43	2	McMaster Carr	https://www.mcr		$ 10.86
Universal Aluminum Mounting Hub		for motor mointing	B08JTYB8VK	$9.99	1	Amazon	https://www.ama		$ 9.99
4mm Flange Shaft Coupling		for transmission	B0BZ57SPX5	$7.70	1	Amazon	https://www.ama		$ 7.70
Stainless Steel Binding Barrel and Screw		for drumstick holder	99637A308	$3.61	1	McMaster Carr	https://www.mcr		$ 3.61
M3 Steel Hex Nuts		for transmission	90591A250	$2.81	1	McMaster Carr	https://www.mcr		$ 2.81
M3 Screw 10mmL		for transmission	91274A105	$5.43	1	McMaster Carr	https://www.mcr		$ 5.43
4-40 3/8L Screw		for drumstick holder and housing	92196A108	$5.84	1	McMaster Carr	https://www.mcr		$ 5.84
4-40 nuts		for drumstick holder and housing	91841A005	$3.89	1	McMaster Carr	https://www.mcr		$ 3.89
Plywood for Housing		for housing	--	$8.32	1	Jacobs Hall	https://store.jac		$ 8.32
Snare Drum		for product to play on	B0CKW5F652	$47.40	1	Amazon	https://www.ama		$ 47.40
PLA for 3D Printed Parts		for drumstick holder and linkages	--	$42.31	1	Etch Shop	https://store.jac		$ 42.31
Ring Shim 0.01" Thick, 1/4" ID		for transmission	97022A372	$8.63	1	McMaster Carr	https://www.mcr		$ 8.63
Belleville Disc Springs		for transmission	94065K26	$5.00	1	McMaster Carr	https://www.mcr		$ 5.00
Compression Spring		for elastic bounce	9657K32	$5.92	1	McMaster Carr	https://www.mcr		$ 5.92
Compression Spring Used		for elastic bounce	SP 9718	$4.40	1	Amazon	https://www.ama		$ 4.40
gearmotor		for driving motion	B07GNDG2NC	$14.88	1	Amazon	https://www.ama		$ 14.88
mcmaster shipping + tax costs		The additionally paid costs for each M	--	$39.92	1	McMaster Carr	--		$ 39.92

CAD Screenshots

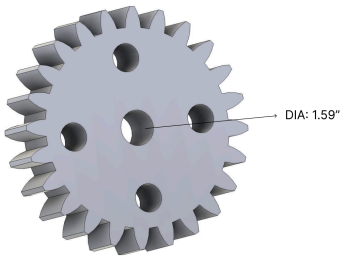
Full Assembly:

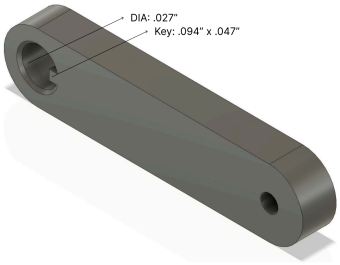


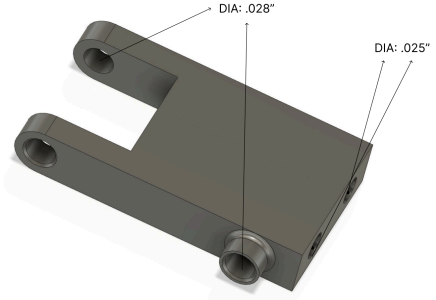
Labeled Transmission

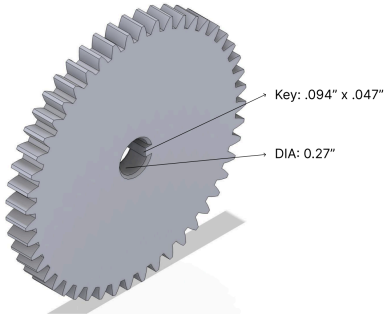

Female Threaded Round Standoff
Standoff 1 (3D Printed Nylon)
Keyed Gear
Shaft Collar (Not Pictured)
Clamping Shaft Collar
Ball Bearing
Belleville Disc Springs
Ring Shims
Shaft Collar (Not Pictured)
Standoff 2 (3D Printed Nylon)
Ball Bearing
Clamping Shaft Collar
Keyed Shaft

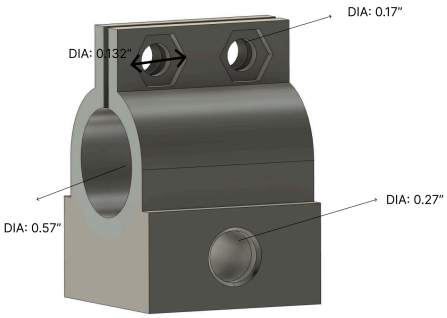
3D Printed CAD Dimensioned Screenshots (Small Gear, Link 1, Link 2, Large Gear, Drumstick Holder)











Code Screenshots

```
void TimerInterruptInit() {
  timer = timerBegin(1000000);           // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer, &onTime);  // Attach onTimer function to our timer.
  timerAlarm(timer, 500000, true, 0);
}

void setup() {
  Serial.begin(115200);
  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);
  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);
  pinMode(POT, INPUT);
}

void motorUp(int speed) {
  // scale the speed to 0-255
  int scaledSpeed = map(speed, 0, 100, 0, 255);
  ledcWrite(BIN_1, scaledSpeed);
  ledcWrite(BIN_2, 0);
}

void motorStop() {
  ledcWrite(BIN_1, 0);
  ledcWrite(BIN_2, 0);
}

#include <Arduino.h>
#define BIN_1 26
#define BIN_2 25
#define BTN 27
#define POT 33 // go back and make sure this is the right pin

volatile bool buttonIsPressed = false; // default state of button
hw_timer_t* timer0 = NULL;
int MAX_PWM_VOLTAGE = 255;
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;

bool drummingActive = false;
unsigned long motorStartTime = 0;

// define the durations for the different states
int slowUpStateDuration = 30;
int slowDownStateDuration = 750;
int fastUpStateDuration = 30;
int fastDownStateDuration = 350;


void IRAM_ATTR isr() {
  buttonIsPressed = true;
}

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

```
void loop() {

  switch (state) {
    case 1: // off state
      Serial.println("State0: button not pressed motor off");
      motorStop();

      if (CheckForButtonPress()) {
        state = 1; // Transition to armed state
        Serial.println("Transitioned to ON state.");
      }
      break;

    case 2: // on state
      Serial.println("State1: button pressed motor starting");
      unsigned long currentTime = millis() - motorStartTime;
      int upStateDuration = map(analogRead(POT), 0, 4095, slowUpStateDuration, fastUpStateDuration);
      int downStateDuration = mappedDownDuration();

      switch (currentTime) {
        case 0: // Motor up state
          if (currentTime < upStateDuration) {
            Serial.println("State1: Motor Up");
            motorUp(100); // Run the motor
          } else {
            // Transition to stop state when upStateDuration is reached
            Serial.println("Transitioning to Motor Stop state");
            currentTime = upStateDuration;
          }
          break;

        case 1: // Motor stop state
          if (currentTime >= upStateDuration && currentTime < (upStateDuration + downStateDuration)) {
            Serial.println("State1: Motor Stop");
            motorStop(); // Stop the motor
          } else if (currentTime >= (upStateDuration + downStateDuration)) {
            // Reset time for the next cycle
            Serial.println("Resetting motorStartTime for next cycle");
            motorStartTime = millis();
          }
          break;
  }

      if (CheckForButtonPress()) {
        state = 0; // Transition to off state
        Serial.println("Transitioned to OFF state.");
      }
      break;
  }
}


bool CheckForButtonPress() {
  if (interruptCounter)
  {           // Check if the timer interrupt occurred
    portENTER_CRITICAL(&timerMux);
    interruptCounter = false;       // Reset the interrupt counter
    portEXIT_CRITICAL(&timerMux);
    return false;                   // Timer was active, so ignore button press
  }
  else if (buttonIsPressed)
  {     // Check if button was pressed
    return true;                    // Valid button press
  }
  return false;
}

int mappedDownDuration() {
    // read the pot value
  int potValue = analogRead(POT);
  Serial.println(potValue);
  // scale the pot value to 0-100
  // pot goes from 0 to 4095
  int scaledPot = map(potValue, 0, 4095, 0, 100);
  // scale the duration based on the pot value
  int downStateDuration = map(scaledPot, 0, 100, slowDownStateDuration, fastDownStateDuration);
  return downStateDuration;
}
```