Cajonament

By Tolou Pharokhipanah, Tzuyi Chang, Anya Shrivastava, Sam Phillips

I. Opportunity

Cajonament is a device which can accompany a live musician. It takes in a rhythm played on a Cajon and then plays it back on loop, with the ability to re-record or pause the tune as the musician wants.

II. Strategy and Functionality

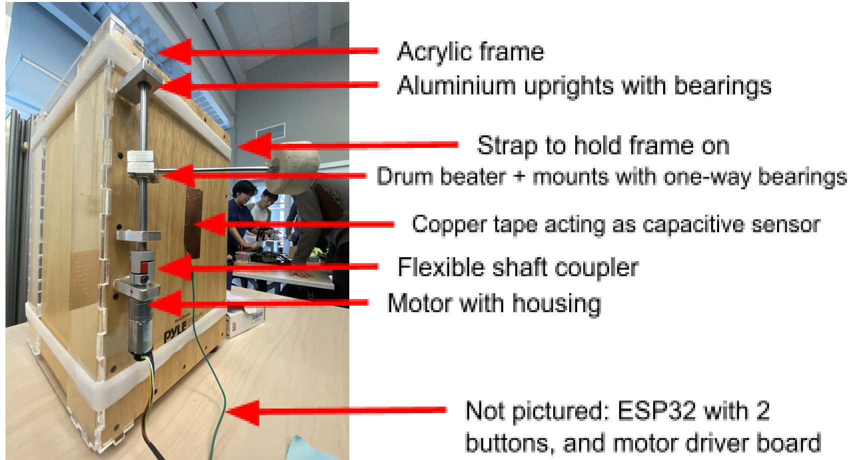
Sensing: The cajon has two main hitting areas (a bass and a snare). To locate where a hit had been played, we originally intended to use two microphones placed at opposite ends of the drum. We would take the difference in volume between the mics to infer the hit location. This turned out to be a poor strategy as the volume difference was very low. On the suggestions of Michael (GSI) and Alex (Machine Shop legend) we instead opted for capacitive touch sensors which used copper tape and the ESP32 touch pins.
Actuation: We used a brushed DC motor to drive a drum beater. We debated between a chain driven or direct drive system, and ultimately landed on a direct drive approach after seeking Professor Stuart's advice. The powertrain consisted of three machined aluminum uprights. The first held the motor using 2 M3 bolts. The motor was mounted to a flexible shaft coupling using a machined coupler, and the other end of the flexible shaft coupler was mounted to a 200mm long 8mm diameter rod which was suspended between the other two aluminum uprights, which had bearings press fit into them. This entire assembly was mounted to an acrylic frame, held in place by two Velcro straps.
We had planned to create two identical beaters to be able to hit the snare and bass section, but later decided to create one beater with two DOFs. Unfortunately, due to time constraints we were only able to manufacture one DOF, though our beater (attached to the 200mm long rod between the two aluminum uprights) is clamped into place between two aluminum plates using six 4-40 bolts. These plates each have a one-way bearing press fit into them, mounted in opposing directions. This means that the beater is constrained to rotate with the shaft but is free to slide along it. We had intended to add a second motor which would use a GT2 timing belt to actuate on this 2nd DOF.

<u>Initial desired functionality vs. Achieved functionality</u>
Force of hit: We wanted a hit of at least 1.5N to create a drum beat loud enough. Whilst we have not measured the output force from our mechanism directly, it definitely produces a loud-enough drum hit.
Speed: We wanted to be able to play a beat every 0.250 seconds (4 beats per second) as 95% of songs on Spotify fall between 70 bpm (1/4 beat = 0.214 seconds) and 169 bpm (1 beat = 0.355 seconds)[1]. Our drum was able to play 4 notes a second, however with very fast tunes like this we sometimes saw it skip beats.
Be able to hit Cajon in two different places: We did not meet this functionality
Sense and record repeated pattern: Our final product was able to do this using our capacitive sensor and a timestamps array.
Actuate beaters from Arduino: Our final product was able to do this well, with the controller design taking up much more of our time than we had expected at the start of this project.

III. Integrated Physical Device

---

1 https://blog.musiio.com/2021/08/19/which-musical-tempos-are-people-streaming-the-most/#:~:text=90%2D99%20BPM%20is%20the,a%20very%20similar%20tempo%20range

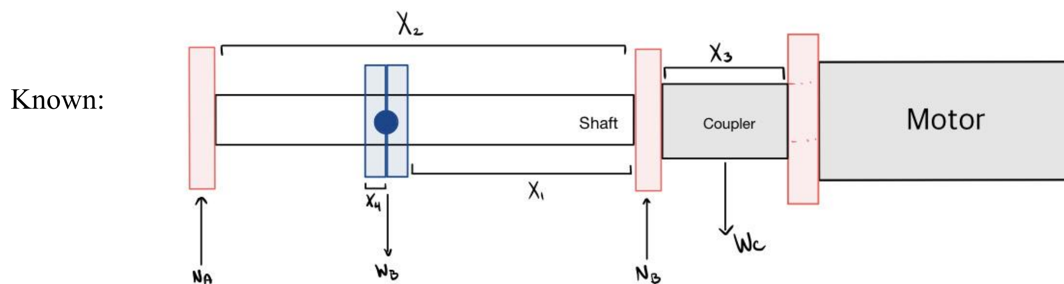


IV. Function Critical Decisions

A. Force Diagram and Load Calculations

We have a range of forces and moments depending on where the beater is placed, therefore we found the maximum and minimum values of this range.

Known:



$x_2 = 4\ in$; $x_3 = 0.5\ in$; $B_w = 0.3625\ lb$; $W_c = 0.1\ lb$

Values when beater is placed all the way to the left: $x_1 = 3.6\ in$

$$\sum M_B = 0 = -W_c * x_3 + B_w * x_1 - x_2 * N_A$$

$$4N_A = 1.255 \rightarrow N_A = 0.31375\ lb$$

$$\sum F_y = 0 = N_A + N_B - B_w - W_C \rightarrow N_B = B_w + W_C - N_A \rightarrow N_B = 0.14875\ lb$$

Values when beater is placed all the way to the right: $x_1 = 0.4\ in$

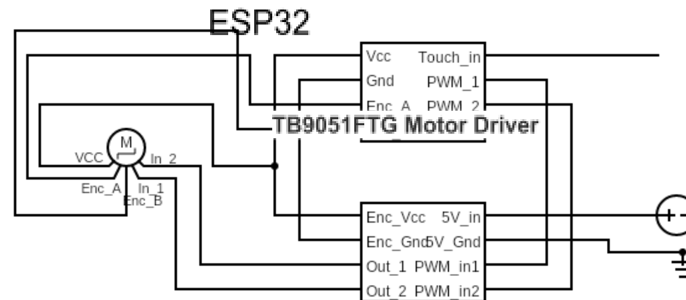
$$\sum M_B = 0 = -W_c * x_3 + B_w * x_1 - x_2 * N_A$$

$$4N_A = 0.095 \rightarrow N_A = 0.02375\ lb$$

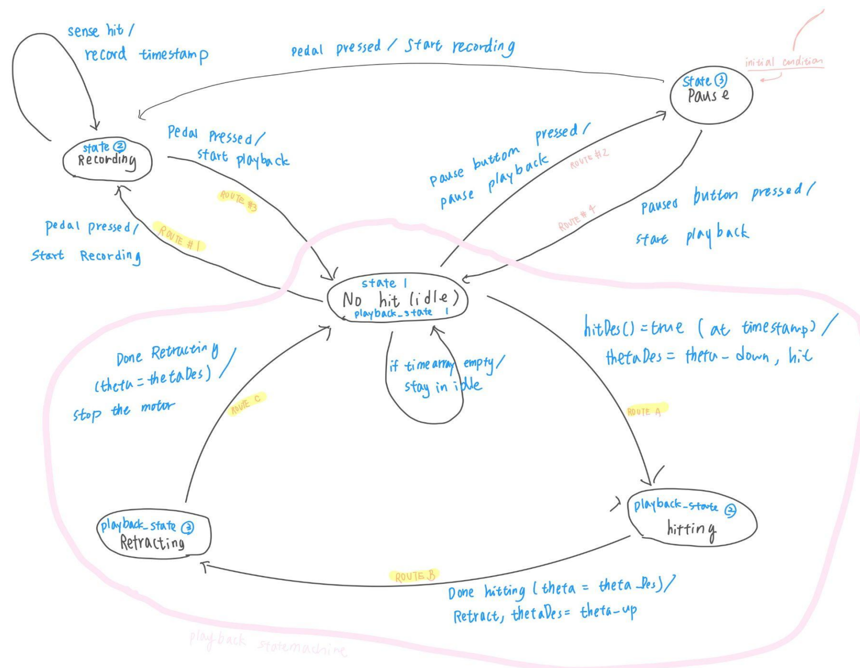
$$\sum F_y = 0 = N_A + N_B - B_w - W_C \rightarrow N_B = B_w + W_C - N_A \rightarrow N_B = 0.43875\ lb$$

V. Diagrams

Circuit diagram



State Transition Diagram



VI. Reflection

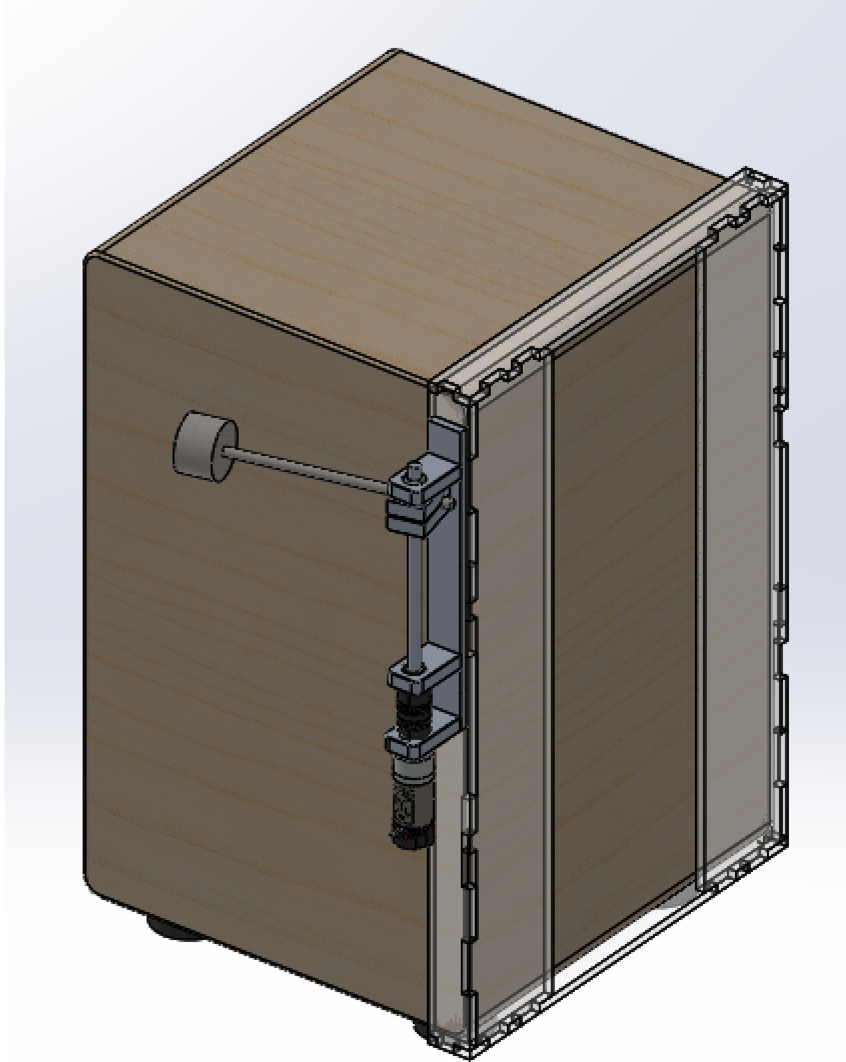
Overall, we are really proud of this project. On reflection though there are some things that we could have done better:

1. Work on integrating separate parts of our system earlier. Integration took us much longer than expected, and we wish we brought our code and physical components together 3-4 weeks earlier
2. Better compatibility of component selection: Our motor driver was not perfectly matched to our motor and we experienced many issues with stalling as a result, and had to narrow our range of PWM values in our code to accommodate for this
3. More willing to change code approach: We spent a long time trying to debug position control and did not consider swapping to velocity control until we spoke to the professor. We wish we'd been more open minded in changing our fundamental approach rather than trying to find a 'one-line' fix.

Bill of Materials

Name	**Number**	**Cost/item**	**Total cost**	**Purchaser**	**Link**	**Part number**	**Additional info**
2pk Drum beaters	1	$11.01	$11.01	Sam	Link		
8mm Needle Roller bearings	1	$4.99	$4.99	Sam	Link		
One way roller bearing	1	$6.99	$6.99	Sam	Link		
8mm OD rods	1	$4.77	$4.77	Sam	Link		
Rod 200mm length 8mm dia	1	$11.01	$11.01	Sam	Link		
Capacitive touch sensor	1	$7.95	$7.95	Anya	Link	1982	
Piezo sensors	5	$0.95	$4.75	Anya	Link	1739	
Copper tape	1	$7.71	$7.71	Anya	Link		
Pack of alligator clips	1	$5.28	$5.28	Anya	Link		
Aluminium stock 10mm by 25mm by 3 ft	1	$25.79	$25.79	Sam	Link	9146T6	10mm by 25mm by 3 ft
Drum rocker spring	1	$10.99	$10.99	Sam	Link		
Flexible shaft coupler 8mm dia	1	$9.99	$9.99	Sam	Link		
Motor (25D, 4804)	1	$55.00	$55.00	Sam	Link		
Motor driver	1	$19.00	$19.00	Sam	Link		
M3 bolts	20	$0.00	$0.00	Sam has box of from previous project			
1/4" Acrylic Sheet	1	$38.85	$38.85	Joy	Link		
Total			**$231.08**				

CAD



Code

```
#include <Arduino.h>
#include <ESP32Encoder.h>

#define PDL 4  // declare the pedal pin number MAUYVE CANG THIS
#define BTN 5//Declare the button pin number
#define LED_PIN 13 // declare the builtin LED pin number
#define BIN_1 26
#define BIN_2 25

ESP32Encoder encoder;

//Setup variables ------------------------------------
//From playback_state_machine:
volatile int theta = 90; // current position
```

```
int thetaDes_Up = 30; // Retracted Position
int thetaDes_Down = 0; // Hitting position
volatile int thetaDes = 0;
volatile int playback_state = 1;
volatile int direct = 1;
volatile double firstHitTime;

///////////////////////////V Control
int omegaSpeed = 0;
int omegaDes = 50;
int omegaPrev = 0;
int omegaMax = 19;
//int D = 0;

volatile int D1 = 100; // Calculated PWM value that will apply to the motor to reach the desired
position
int thetaMax = 1600;     // 75.8 * 6 counts per revolution
volatile int D = 100; // Calculated PWM value that will apply to the motor to reach the desired position
volatile int e = 0; // Desired - current position
volatile int sum_e ; // For Integrap control

//PID constants
double kp = 0.1;
double ki = 0.1;
double kd = 1;
double k = 0.1;
int KiMax = 2;

unsigned long currentTime, previousTime;
double elapsedTime;
double error;
double lastError;
double input, output, setPoint;
double cumError, rateError;

// From Cajonament:
const int freq = 5000; //LED PMW from lab
const int pwmChannel = 0; //LED PMW channel from Lab
const int resolution = 8; // Resolution from Lab
volatile bool pedalIsPressed = false; //Flag for loop pedal
volatile bool pauseButtonIsPressed = false; //Flag for pause playback button
int state = 3; // initial condition : pause
float playback_timestamps[50] = {}; // Initialize playback timestamps variable. Size pre-set to 50 as
```

```
dynamic sizing too hard in Arduino
int curr_val = 0;
int last_val = 0;
int next_val = 0;
const int ledChannel_1 = 1;
int timeArrayIndex = 0;
int t0 = 0; //the beginning time of each array loop

//Touch sensor
const int touchPin =14;
const int threshold = 20; // variable for storing the touch pin value
int touchValue;
int numHits = 0;
volatile bool touchDebounceT = 1; //debounce Flag
hw_timer_t * touchDebounceTimer = NULL;

volatile bool debounceT0 = 1; //debounce Flag for PDL

hw_timer_t * timer0 = NULL;

//Initialization ------------------------------------
//From Playback_state_machine:
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
volatile bool hitDone = false;
volatile int dir = -1;

hw_timer_t * timer_encoder = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;


// setting PWM properties ---------------------------
const int ledChannel_2 = 2;
const int MAX_PWM_VOLTAGE = 255;
const int MIN_PWM_VOLTAGE = 230;

const int NOM_PWM_VOLTAGE = 150;

void IRAM_ATTR onTime_encoder() {
  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);
}
//From Cajonament:
void IRAM_ATTR isr_pedal() {  // the function to be called when interrupt is triggered
  pedalIsPressed = true;
}
void IRAM_ATTR onTime0() {
  debounceT0 = true;
  timerStop(timer0);
}
void IRAM_ATTR isr_pauseButton() {  // the function to be called when interrupt is triggered
  pauseButtonIsPressed = true;
}
void TimerInterruptInit0() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer0 = timerBegin(0, 80000, true); // divides the frequency by the prescaler: 80,000,000 / 80,000 = 1,000 tics / sec(1 tic = 1ms)
  timerAttachInterrupt(timer0, &onTime0, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer0, 800, true);          // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer0); // Enables timer
}

void IRAM_ATTR onTimeTouch() {
  portENTER_CRITICAL_ISR(&timerMux0);
  touchDebounceT = true;
  timerStop(touchDebounceTimer);
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void TouchTimerInterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  touchDebounceTimer = timerBegin(0, 80000, true); // divides the frequency by the prescaler: 80,000,000 / 80,000 = 1,000 tics / sec(1 tic = 1ms)
//  timerAttachInterrupt(touchDebounceTimer, &onTimeTouch, true);    // sets which function do you want to call when the interrupt is triggered
//  timerAlarmWrite(touchDebounceTimer, 300, true);          // sets how many tics will you count to trigger the interrupt
//  timerAlarmEnable(touchDebounceTimer); // Enables timer
}
```

```
void setup() {
  pinMode(PDL, INPUT);
  pinMode(BTN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  attachInterrupt(PDL, isr_pedal, RISING); //CHANGE = defines interrupt to occur on rising/falling
edge
  attachInterrupt(BTN, isr_pauseButton, RISING);
  Serial.begin(115200);

  // Position Reading Setup
  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

  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);

  TimerInterruptInit0(); // Initiates timer interrupt
  timerStart(timer0);

  timer_encoder = 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(timer_encoder, &onTime_encoder, true); // edge (not level) triggered
  timerAlarmWrite(timer_encoder, 5000, true); // 5000 * 1 us = 5 ms, autoreload true

  timerAlarmEnable(timer_encoder); // enable
}

//MAIN LOOP
void loop() {
  delay(100);
  switch (state) {
    case 1: //Playback
      Serial.println("State 1: playback");

      // ROUTE 1
      if (CheckForPedalPress()) {
        resetDebounceFlags();
```

```
      state = 2;
      delay(800);
    }

    // ROUTE 2
    if (CheckForPauseButtonPress()) {
      resetDebounceFlags();
      state = 3;
    }

    playback_state_machine();
    break;

  case 2: //recording state
    recordHits(); //Function, defined elsewhere, that records and stores hits
    // ROUTE 3
    if (CheckForPedalPress() == true){
      resetDebounceFlags();
      state = 1;
    }
    break;
  case 3: //Pause state
    //ROUTE 4
    if (CheckForPauseButtonPress() == true) { // to playback
      resetDebounceFlags();// ??
      state = 1;
    }

    if (CheckForPedalPress()) { // to recording
      resetDebounceFlags();
      state = 2;
    }

    break;
  }
}

//PLAYBACK STATE MACHINE
void playback_state_machine() {

  //Record Time 0 when Entering Playback StateMachine
  t0 = millis();
```

```
while (CheckForPedalPress() == false && CheckForPauseButtonPress() == false) {

  ///////////////////////////////////////////////////////////////////////
  switch (playback_state) {

    case 1: // No hit, waiting to go down
      //ROUTE A - Start Hitting
      if (timeArrayEmpty() == false && hitDesired() == true) {
        Serial.println("Route A 1-2");
        dir = -1;
        playback_state = 2;
        omegaDes = abs(omegaDes) * dir;
        thetaDes = thetaDes_Down;
      }
      break;

    case 2: // hitting,(going down)
      //ROUTE B - Finished Hitting, start Retracting
      hit();

      if (reachDownPosition() == true) {
        ////////////////////////////////////////////
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
  //      delay(50);
        direct = 1;
        looping();
        ////////////////////////////////////////////////////////////////////////////////////////////////////
        Serial.println("Route B 2-3");
        dir = 1;
        omegaDes = abs(omegaDes) * dir;
        thetaDes = thetaDes_Up;
        playback_state = 3;
      }
      omegaPrev = omegaSpeed;
      break;

    case 3: //Retracting
      retract();

      //ROUTE C -finished retracting
      if (reachUpPosition() == true) {
        ledcWrite(ledChannel_1, LOW);
```

```
        ledcWrite(ledChannel_2, LOW);
        Serial.println("Reached high position. Theta high meas. = ");
        Serial.print(theta);
        if (playback_timestamps[timeArrayIndex + 1] != 0) {
          delay(playback_timestamps[timeArrayIndex + 1] - playback_timestamps[timeArrayIndex]);
        }
        else {
          /////////////reset theta after each loop
          if (theta > thetaDes_Up) {
            direct = -1;
          }
          else if (theta < thetaDes_Up) {
            direct = 1;
          }
          thetaDes = thetaDes_Up;
        }

        direct = -1;
        Serial.println("Route C 3-1");
        playback_state = 1;
        thetaDes = thetaDes_Up;
      }
      break;
  }

  if(CheckForPedalPress()==true){
    resetDebounceFlags();
    state = 2;
   ledcWrite(ledChannel_1, LOW);
   ledcWrite(ledChannel_2, LOW);
    break;
  }
  if(CheckForPauseButtonPress()==true){
    resetDebounceFlags();
    state = 3;
   ledcWrite(ledChannel_1, LOW);
   ledcWrite(ledChannel_2, LOW);
    break;
  }
 }

}
```

```
//FUNCTIONS FOR MAIN CAJOAMENT

void resetDebounceFlags() {
  Serial.println("Flags reset");
  pauseButtonIsPressed = false;
  pedalIsPressed = false;
  debounceT0 = false;
  timerRestart(timer0);
  timerStart(timer0);
}


//Event checker for pedal & pause button
bool CheckForPedalPress() {
  if (pedalIsPressed && debounceT0) {
    return true;
  }
  else {
    return false;
  }
}

bool CheckForPauseButtonPress() {
  if (pauseButtonIsPressed && debounceT0) {
    return true;
  }
  else {
    return false;
  }
}

//record hit Time_array
void recordHits() {
  while (CheckForPedalPress() == false) {
     if (CheckForPedalPress() == true) { // If we want to stop recording, Exit out of this function and go
back into state machine
       resetDebounceFlags();
       state = 1;
       Serial.println("we're changing state");
       break;
     }
     touchValue = touchRead(touchPin);
     if(touchValue < threshold && debounceT0){
```

```
        debounceT0 = false;
        timerRestart(timer0);
        timerStart(timer0);
//        portEXIT_CRITICAL(&timerMux0);
        if(numHits == 0){
          firstHitTime = millis();
          playback_timestamps[numHits] = 0;
        }
        else{
          playback_timestamps[numHits] = millis()-firstHitTime;
        }
        numHits += 1;
      }
     for (int i = 0; i < numHits; i++){
        Serial.print(playback_timestamps[i]);
        Serial.print(" ");
      }
    delay(50);
  }

}



//FUNCTIONS FOR PLAYBACK STATE MACHINE
bool hitDesired() {
  if ((millis() - t0) >= playback_timestamps[timeArrayIndex]) {
    return true;
  }
  else {
    return false;
  }
}

bool reachDownPosition() {
  if (omegaSpeed == 0 && omegaPrev !=0) { //finishing hitting
    theta = 0;
    return true;
  }
  else {
    return false;
  }
}
```

```
void looping() {
  if (playback_timestamps[timeArrayIndex + 1] != 0) {
    timeArrayIndex++;
  }
  else {
    timeArrayIndex = 0; ////CYCLE////
  }
}

bool reachUpPosition() {
  if (theta >= thetaDes) {
    return true;
  }
  else {
    return false;
  }
}


void hit() {
  dir = -1;
  velocityControl();
}
void retract() {
  dir = 1;
  velocityControl();
}

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

    omegaSpeed = count;
    theta += count;

    currentTime = millis();
    elapsedTime = currentTime - previousTime;
    e = thetaDes - theta;
    rateError = (e - lastError) / elapsedTime;
    lastError = e;
```

```
    e = omegaDes - omegaSpeed;
    sum_e = sum_e + e;
    //D = kp * e + ki * sum_e;
    D = kp * e + ki * sum_e + kd * rateError; ///////////I change and add kd///////////
    D = abs(D);
    //D = 255;
    D = D * dir;

    Serial.print("Theta is:");
    Serial.println(theta);

    //Ensure we do not stall motor
    if (D >= 0 && D < MIN_PWM_VOLTAGE) {
      D = MIN_PWM_VOLTAGE;
    }
    else if (D <= 0 && D > -MIN_PWM_VOLTAGE) {
      D = -MIN_PWM_VOLTAGE;
    }

    //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;
    }

    //Map the D value to motor directionality
    //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
    if (D > 0) {
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
      ledcWrite(ledChannel_1, -D);
      ledcWrite(ledChannel_2, LOW);
    }
    else {
      ledcWrite(ledChannel_1, LOW);
    }
  }
}
```

```
bool timeArrayEmpty() {
  if (playback_timestamps[1] == 0) {
    return true;
  }//Array always initilazes entries to 0
  else {
    return false;
  }
}
```