Final Project Report

ME 102B: Mechatronic Design
Group 33: Eric Kang, Mirabelle Huang, Sebastian Baehr
Dec. 12, 2021

The purpose of this device is to provide entertainment and set the mood for different occasions with its 6 different presets. The different presets involve different LED patterns and predetermined panel angles. It's a convenient and portable product that enhances the mood for various environments.

Initially we wanted to have the LED pattern sync to the vibrations of the music but due to some mechanical issues with the microphone that we bought we didn't have enough time to get a new microphone and therefore could not sync the LEDs to the music in some of the presets. Originally we planned to have multiple rows of LEDs on each panel but unfortunately due to soldering and connectivity issues we had to just go with 1 strip each side. We also wanted to have a laser originally but due to space issues within our electronic housing box and an immense amount of wiring we couldn't get the wires for the laser and motor to fit inside in time so we decided to take out the laser as well. The predetermined panel rotation in some of the presets was supposed to be about 45 degrees, however, the encoder with the mega microcontroller was too noisy and thus we tried to have the panels rotate for a certain amount of time until it reached about 45 degrees and stop it but the issue we had with that was that we couldn't get the motors to stop rotating. Since we used the switch case function, the function would continuously run the code within the case that it was in so our panel rotation function kept getting called, causing the panels to continuously rotate. We attempted to write a counter into the cases that would only let the panel rotation function only be called once but sometimes the button would debounce and skip a preset despite our code to prevent debouncing and this would then mess up the counter code as it's dependent on going through the states in order.

Strategies that worked well for our group was assigning specific work to teammates and asking if anyone needed help in order to try to mitigate one person from being overly burdened with work. We also had everyone working on every part so that we all were able to contribute to the mechanical, electrical, and software parts of the device. This way we all knew what was going on with our project and understood all parts of it. What we wished we did differently was just start the project earlier and allow more time for error and failure during the project because we ended up procrastinating some parts of it which led to an immense amount of stress at the end. We also had issues with mechanical parts not working so we wish we had tested them earlier or bought them earlier or bought extra to mitigate this.

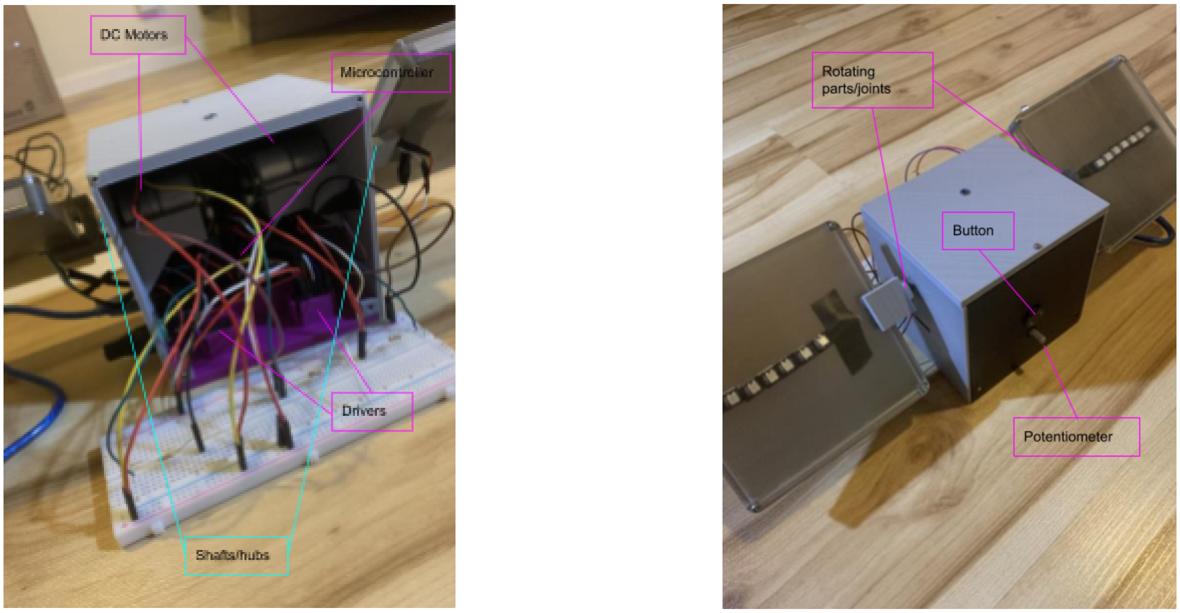
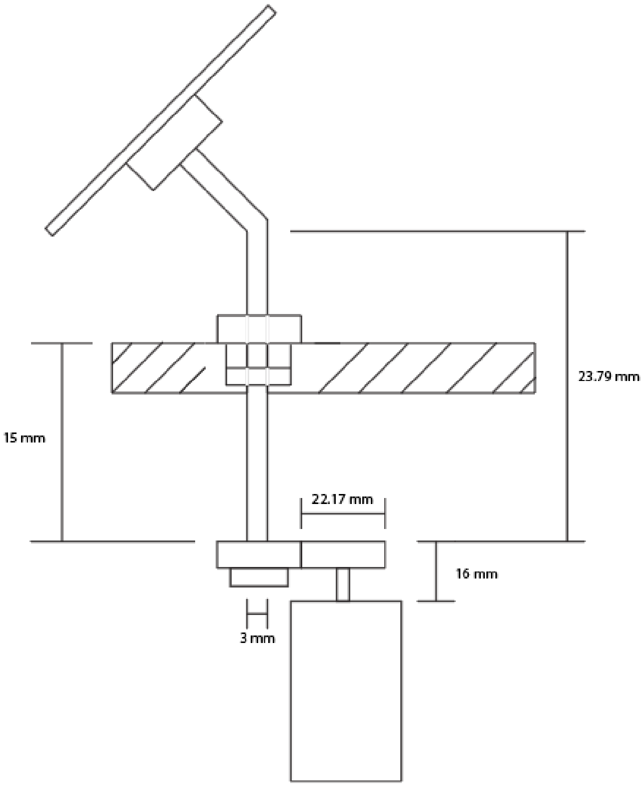


Figure 1: Photos of physical assembly from back side (left) and front side (right).

2. Function Critical Analysis

Motors: Due to low noise needing to be emitted from the motors and having position control, a brushless DC motor with a magnetic encoder was used for all 3 motors. Attempted control type is velocity PID control on the laser (LED) motor since both a fast time response and low steady state error is wanted, and position PI control on the panel motors since the desired time response will be underdamped and slow and to have low error at steady state. The torque on the panel motors can be considered negligible since no external forces exist and the gravitational force of the panel acts on the axis of rotation. The radial force on the panel motors is the weight of panels, but the ball bearings handle this force. The inertia of the panel motors must drive is $J_{tot} = J_{hub} + J_{connector} + J_{panel} = 388.69\ g{\cdot}in^2$. This causes the inertia ratio to be well below the rated value for the motor and thus not a concern. Since the panels only need to rotate slowly, current draw is also not a concern.

For the laser (LED) motor, the calculations for torque and current draw are shown below.



Torque:

Total Inertia of Shaft 1

$$J_{tot} = J_{LED} + J_{LED,connector} + 3 \cdot J_{hub} + J_{bearing} + J_{gear}$$

$$J_{LED} = J_{bearing} = 0$$

$$J_{LED,connector} = 97.98\ g{\cdot}in^2$$

$$J_{hub} = 0.197\ g{\cdot}in^2$$

$$J_{gear} = 0.270\ g{\cdot}in^2$$

$$J_{tot} = 98.781\ g{\cdot}in^2$$

Max acceleration needed = $100\ \frac{rev}{s^2}$

$$\tau_{shaft} = J_{tot}\alpha = (98.781\ g{\cdot}in2)(628\ \frac{rad}{s^2})$$

$$\tau_{shaft} = 0.04\ N \cdot m$$

Since motors are same size

$$\tau_{shaft} = \tau_{motor} = 0.04\ N \cdot m\ <\ \tau_{stall} = 0.15\ N \cdot m$$

FOS = 3.75

Current: Running at 26.67% of stall torque, and stall current rated at 2.0 A:

Current Max = $(2.0\ A)(.2667) = 533\ mA$

$V_{rates} = 7.4\ V$, but will be running on 9 V

$P = IV = (533\ mA)(7.4\ V) = 3.95\ mW = I_{expected}(9\ V)$

$I_{expected} = 439\ mA$

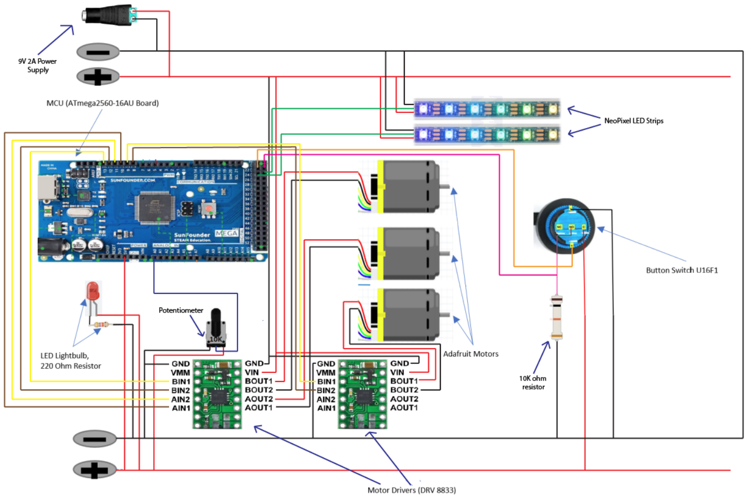
Stall Current not exceeded

Push Button: The button is rated for 5A but will just connect to microcontroller and attach to 10 kΩ pull-down resistor. It will mount in a M16 hole on the front plate using M16 threads and nut on button. Is momentary since states will change based on length of button press.

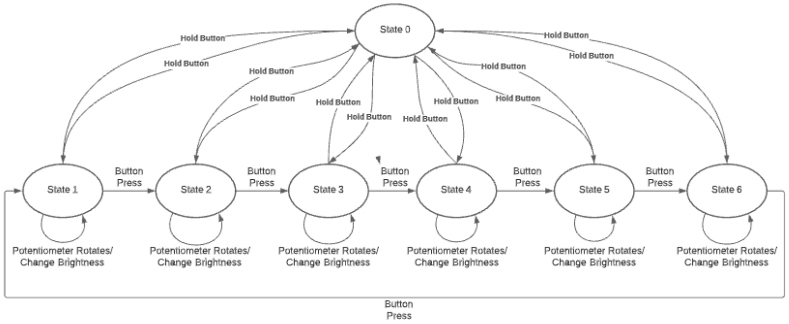
Auto-Gain Microphone: It has auto-gain amplification so detecting sound from far away and really close will be easier. The audio reading has a max value of 2 V with 1.25 V DC bias, so the 5 V input pins on the MEGA 2560 will suffice to read the values. The microphone casing will be mounted and glued to the mounting hole in the front plate.

LED Strips: They run on 9 V and 2 A, so they will be connected to the power supply of their own. The LEDs run on Neopixel's WS2812B chip allowing for individually addressable LEDs, and prevents having to PWM red,green, and blue channels. They will be attached to the bottom plate of panels, and a hole exists for wiring to travel out of the panel.

Electrical Diagram:



State Diagram



State	Description
State 0	OFF mode. LED panels OFF and @ 0^0, laser (LED) OFF, all motors OFF, potentiometer OFF, Button LED OFF, Button ON
State 1	ON mode. LED panels w/ solid color and @ 45^0, laser (LED) OFF, laser motor OFF, potentiometer ON, Button LED ON, Button ON
State 2	ON mode. LED panels w/ Color Cycle and @ 45^0, laser (LED) OFF, laser motor OFF, potentiometer ON, Button LED ON, Button ON
State 3	ON mode. LED panels w/ LED Flash and @ 0^0, laser (LED) OFF, laser motor OFF, potentiometer ON, Button LED ON, Button ON
State 4	ON mode. LED panels w/ Rainbow March and @ 45^0, laser (LED) ON, laser motor ON @half speed, potentiometer ON, Button LED ON, Button ON
State 5	ON mode. LED panels w/ Sliding Bar and @ 45^0, laser (LED) ON, laser motor ON @half speed, potentiometer ON, Button LED ON, Button ON
State 6	ON mode. LED panels w/ Color Cycle and @ 0^0, laser (LED) ON, laser motor ON @half speed, potentiometer ON, Button LED ON, Button ON

Appendix

Bill of Materials:

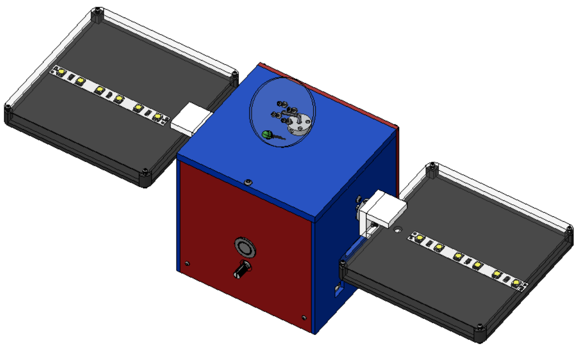
Item Name	Description	Price [ea.]	Quantity	Link to Item	Notes
Adafruit DC Motor	7V DC motor with magnetic encoder	$13.50	3	Adafruit	
DRV8833 Dual Motor Driver	Dual H-bridge motor driver IC	$6.95	2	Pololu	
LED Light Strips	Neopixel LED strip w/ individually addressable LEDs.	$14.99	1	Amazon	Neopixel only requires a power, ground, and data wire
Pololu Universal Aluminum Mounting Hub	3 mm universal mounting hub w/ set screw	$5.95	3	Pololu	
M2 Bolts/Nuts	310 Pieces M2 x 4mm/6mm/8mm/10mm/12mm/16mm/20mm	$9.99	1	Amazon	
Ulincos Momentary Push Button Switch	Momentary push button switch w/ LED	$8.38	1	Amazon	
Pot 1K ohm 1/5W Carbon Linear	1K ohm potentiometer	$1.22	1	Digi-Key	
Mega2560 R3 ATmega16AU	Arduino microcontroller w/ ATmega16 chip	$16.99	1	Amazon	
LitStar 9V 2A AC DC Power Supply	Power supply adapter (100-240V to 9V 2A)	$11.99	1	Amazon	
Uxcell MR63-2RS Deep Groove Ball Bearings	4 ball bearing w/ 3 mm ID/6 mm OD	$9.49	1	Amazon	

Item Name	Description	Price [ea.]	Quantity	Link to Item	Notes
Objet VeroClear	Objet 3D printing material	$0.34 /gram	388	Jacobs	Used to print both the LED panels, and the laser mount
Objet Tango Black	Objet 3D printing material	$0.34 /gram	46	Jacobs	Used to print both the LED panels, and the laser mount
Objet Support	Objet 3D printing support material	$0.14 /gram	140	Jacobs	Used to print both the LED panels, and the laser mount
Standard PLA	Standard 3D printing material for FDM	$27.00 /kilogram	1	AnyCubic	Used to manufacture the device housing

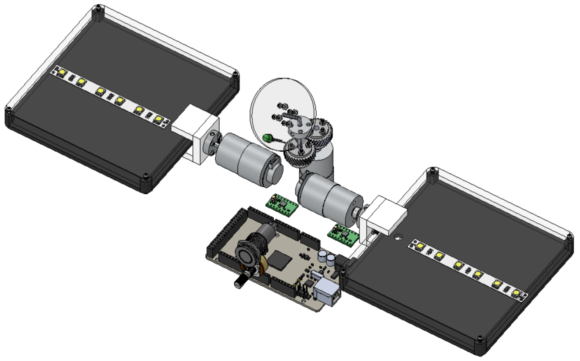
CAD Drawings:

Isometric Views

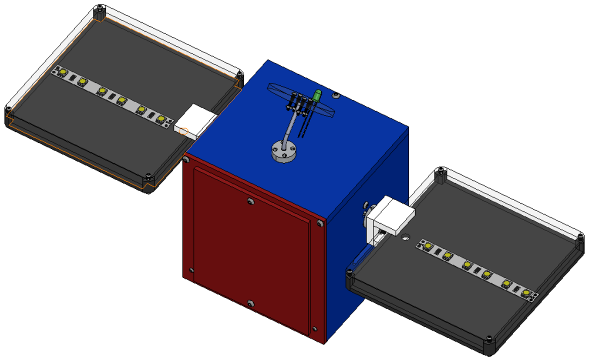
With Device Housing



W/out Device Housing

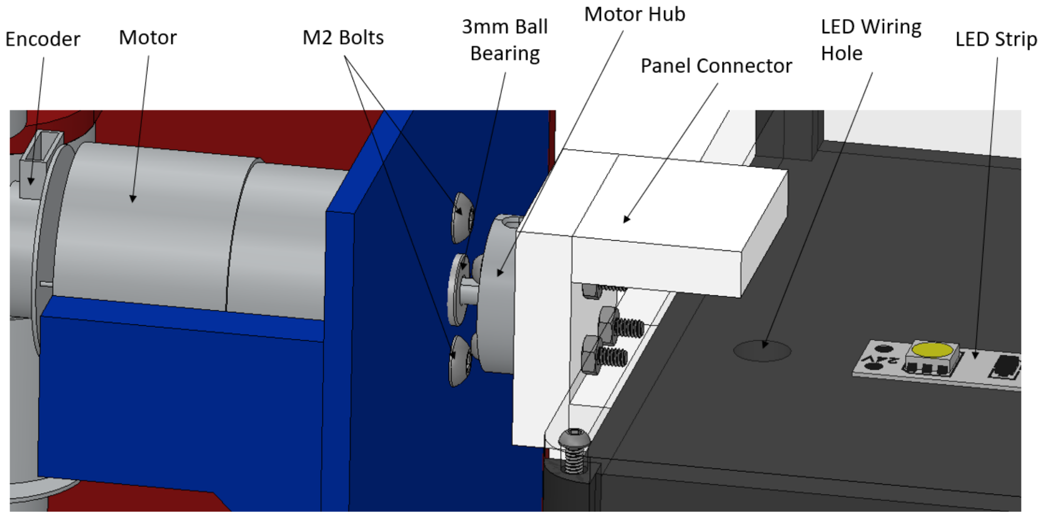


Back Side



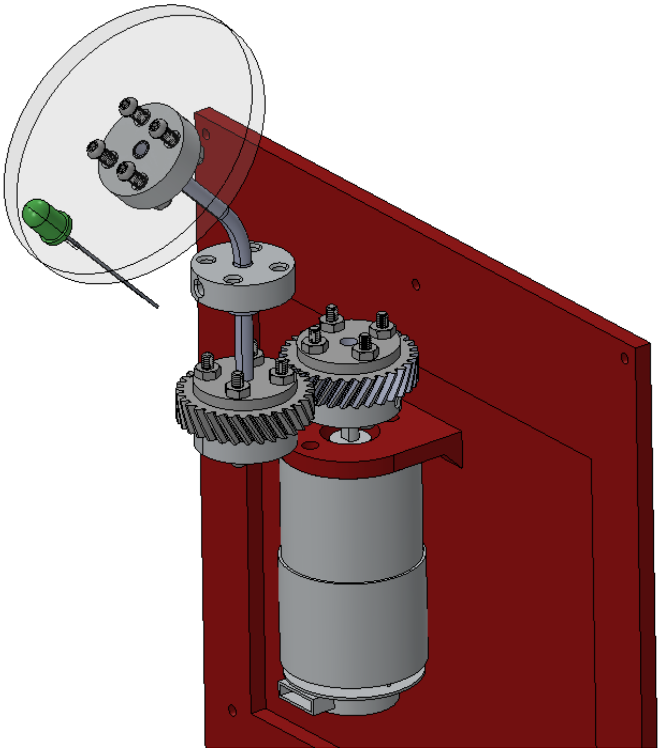
Close Up of Panel Assembly

- Connector for the encoder attached to the motor will point down in reality.
- Same assembly for both panels

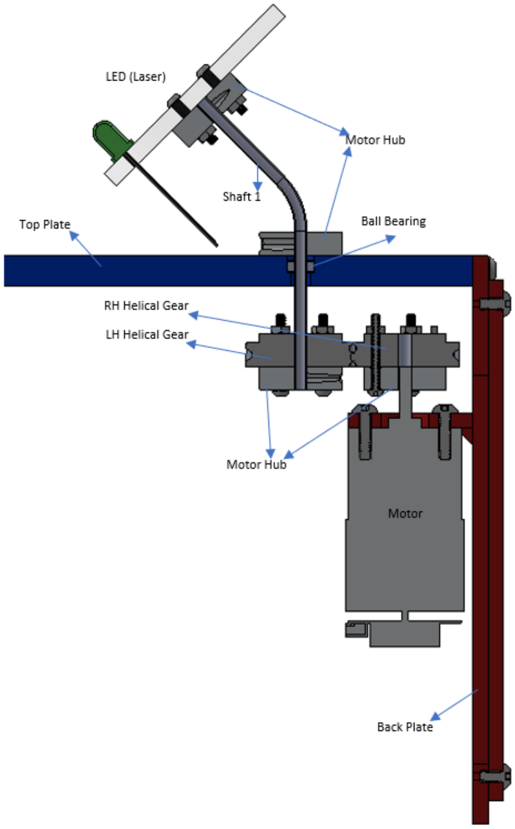


Spinning LED (Laser) Assembly

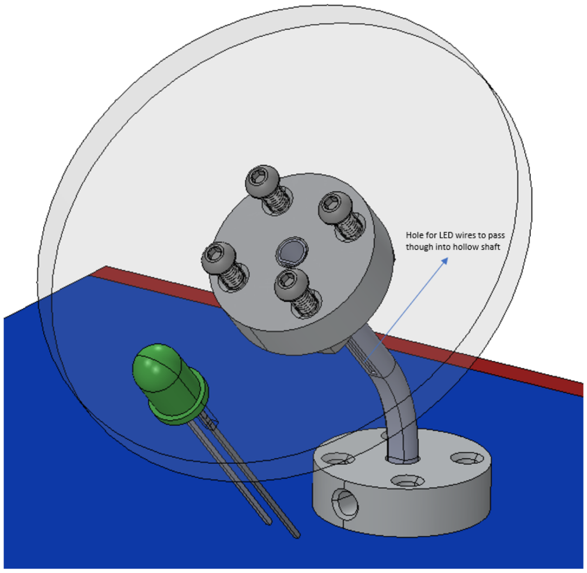
Isometric Close Up



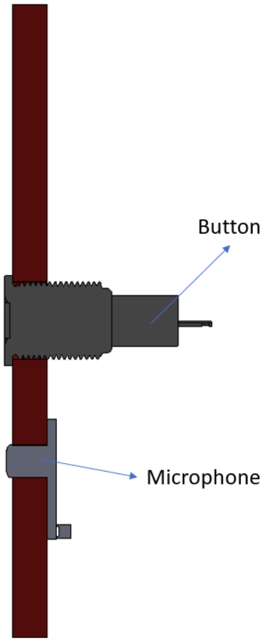
Cross Section



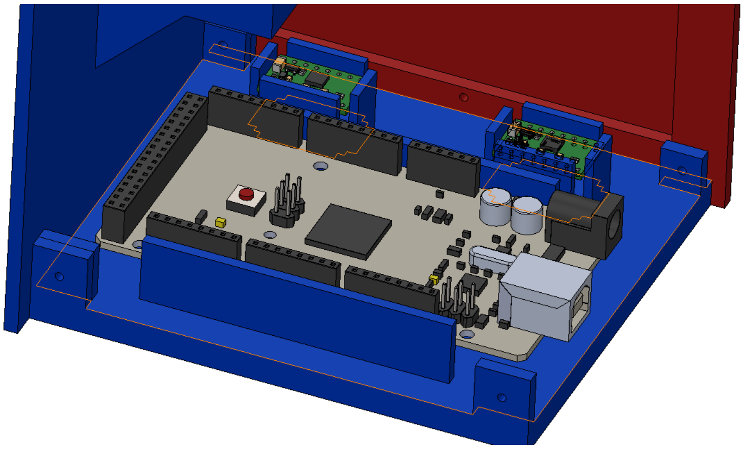
Close up of Top Section



Front Plate Cross Section

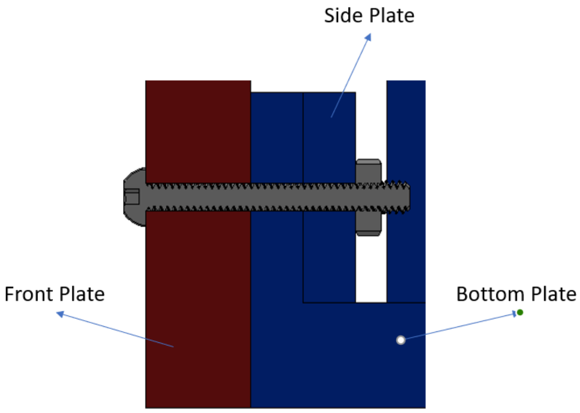


Close Up of Bottom Plate

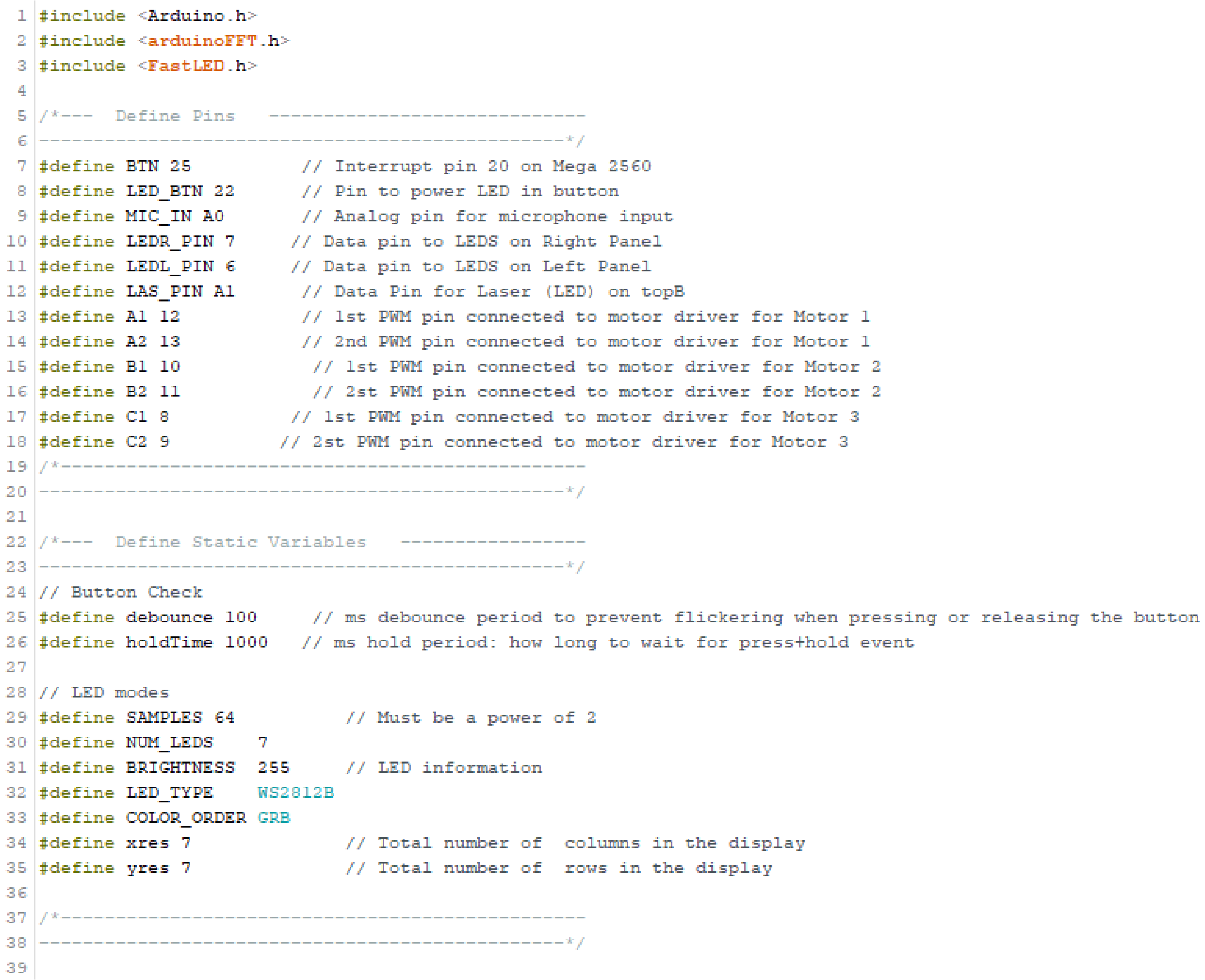


Close up of Plate Connection

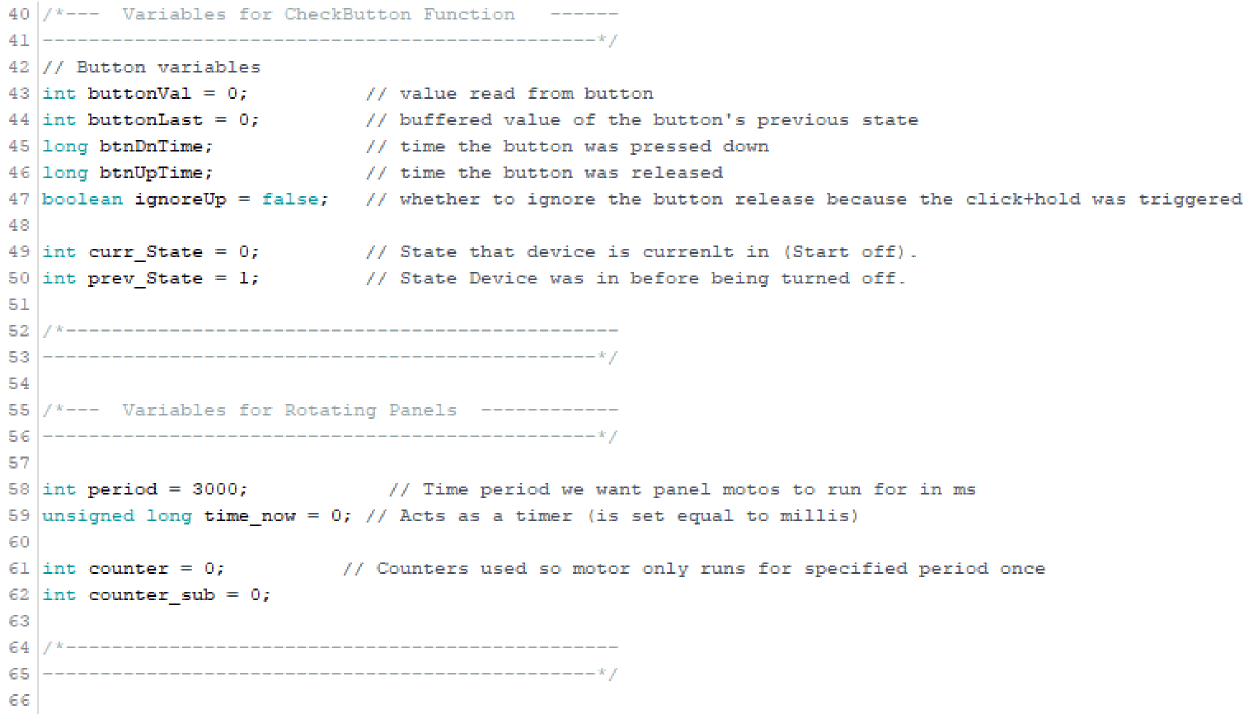
- Essentially the same on four corners at the bottom of the device.



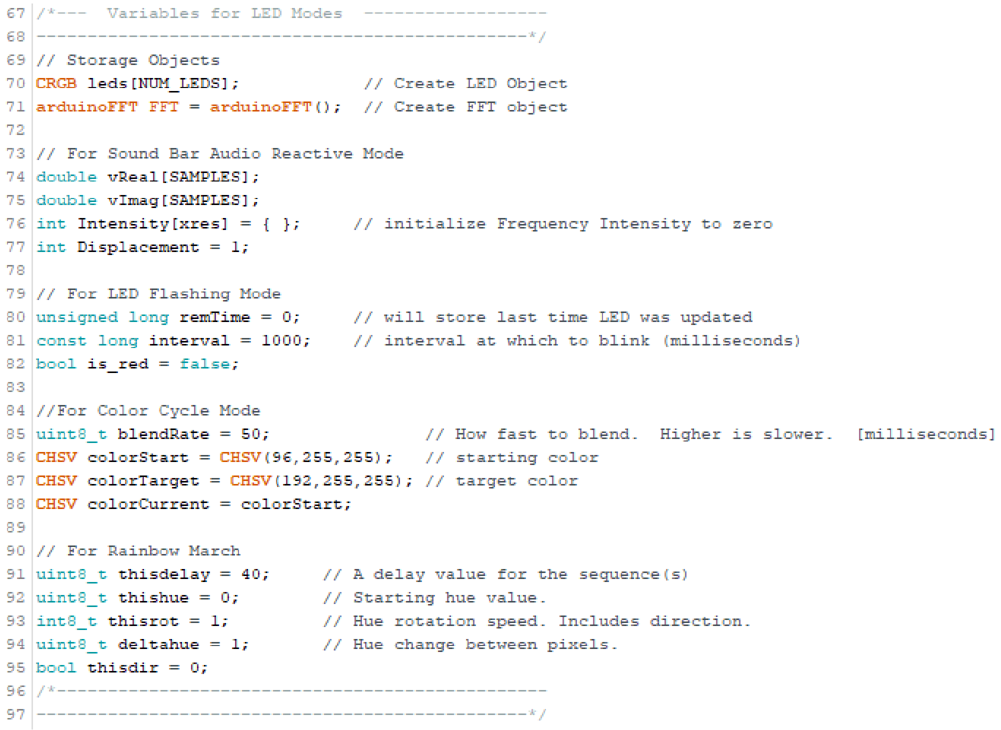
Code:



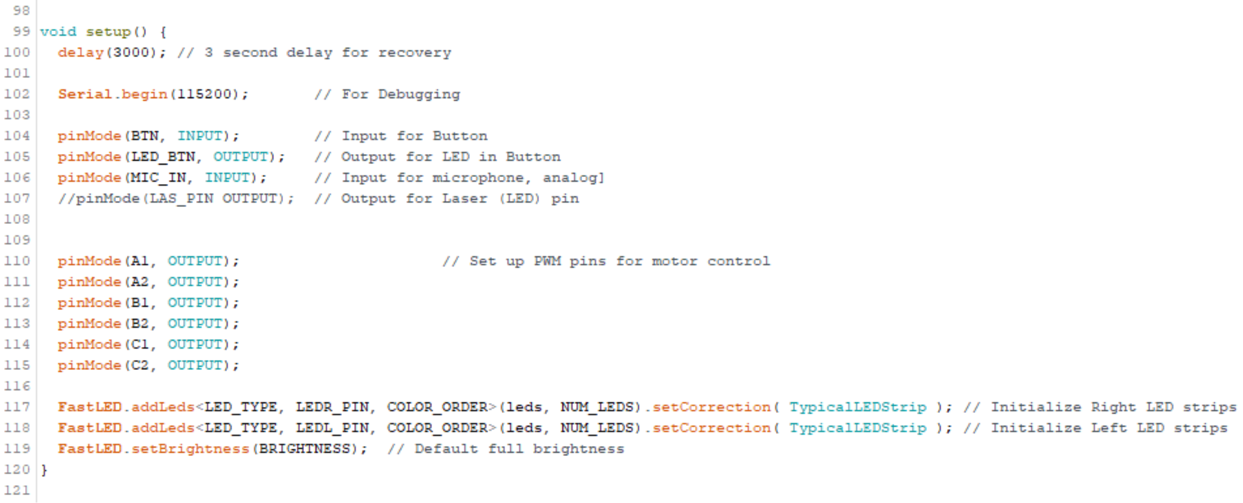
```
#include <Arduino.h>
#include <arduinoFFT.h>
#include <FastLED.h>

/*---  Define Pins   -----------------------------
-------------------------------------------------*/
#define BTN 25          // Interrupt pin 20 on Mega 2560
#define LED_BTN 22      // Pin to power LED in button
#define MIC_IN A0       // Analog pin for microphone input
#define LEDR_PIN 7     // Data pin to LEDS on Right Panel
#define LEDL_PIN 6     // Data pin to LEDS on Left Panel
#define LAS_PIN A1      // Data Pin for Laser (LED) on topB
#define A1 12           // 1st PWM pin connected to motor driver for Motor 1
#define A2 13           // 2nd PWM pin connected to motor driver for Motor 1
#define B1 10            // 1st PWM pin connected to motor driver for Motor 2
#define B2 11            // 2st PWM pin connected to motor driver for Motor 2
#define C1 8           // 1st PWM pin connected to motor driver for Motor 3
#define C2 9          // 2st PWM pin connected to motor driver for Motor 3
/*-----------------------------------------------
-------------------------------------------------*/

/*---  Define Static Variables   ----------------
-------------------------------------------------*/
// Button Check
#define debounce 100     // ms debounce period to prevent flickering when pressing or releasing the button
#define holdTime 1000   // ms hold period: how long to wait for press+hold event

// LED modes
#define SAMPLES 64          // Must be a power of 2
#define NUM_LEDS    7
#define BRIGHTNESS  255     // LED information
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB
#define xres 7              // Total number of  columns in the display
#define yres 7              // Total number of  rows in the display

/*-----------------------------------------------
-------------------------------------------------*/

```



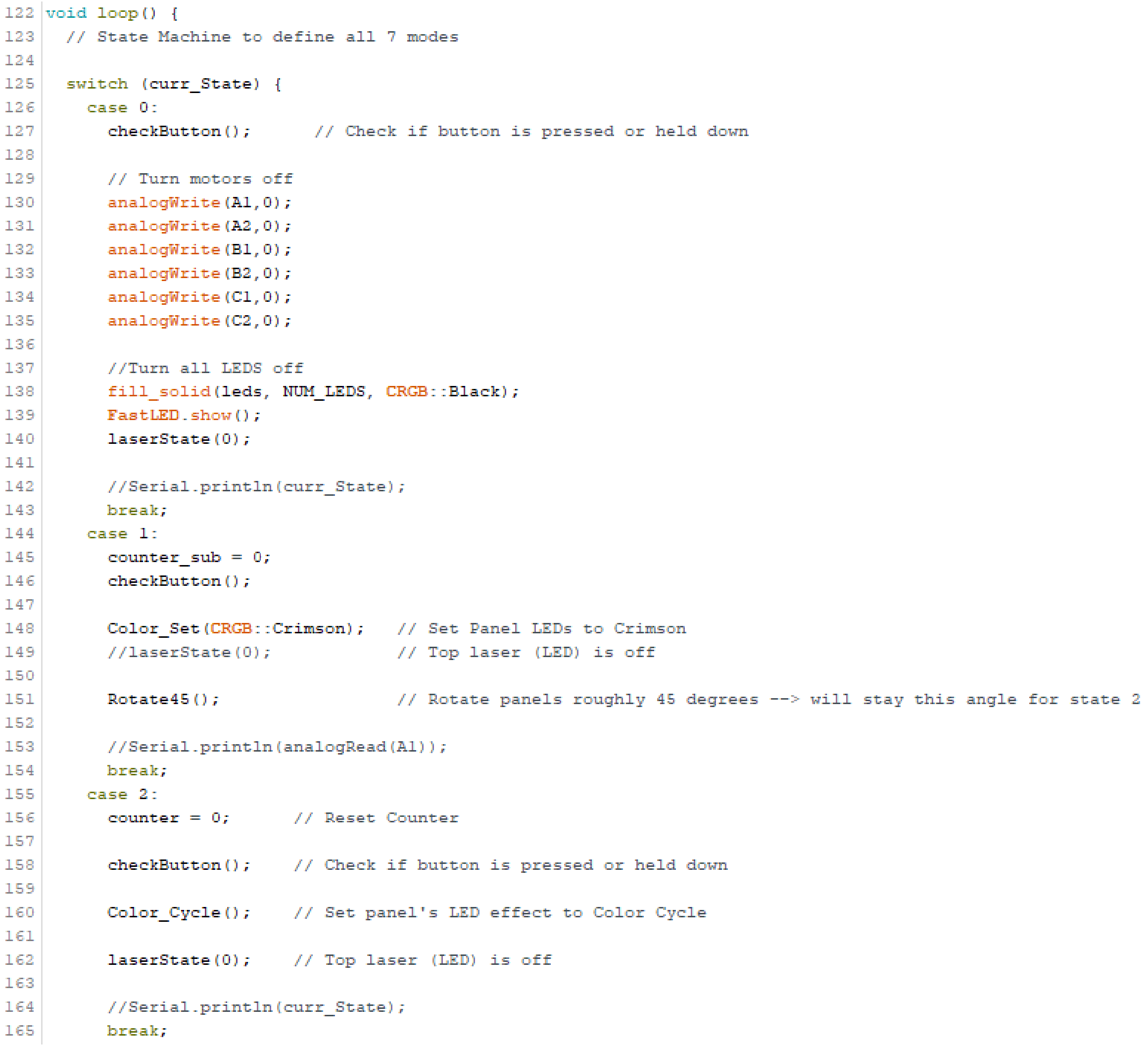
```
/*---  Variables for CheckButton Function   ------
-------------------------------------------------*/
// Button variables
int buttonVal = 0;           // value read from button
int buttonLast = 0;          // buffered value of the button's previous state
long btnDnTime;              // time the button was pressed down
long btnUpTime;              // time the button was released
boolean ignoreUp = false;    // whether to ignore the button release because the click+hold was triggered

int curr_State = 0;          // State that device is currenlt in (Start off).
int prev_State = 1;          // State Device was in before being turned off.

/*-----------------------------------------------
-------------------------------------------------*/

/*---  Variables for Rotating Panels  ------------
-------------------------------------------------*/

int period = 3000;             // Time period we want panel motos to run for in ms
unsigned long time_now = 0; // Acts as a timer (is set equal to millis)

int counter = 0;          // Counters used so motor only runs for specified period once
int counter_sub = 0;

/*-----------------------------------------------
-------------------------------------------------*/

```



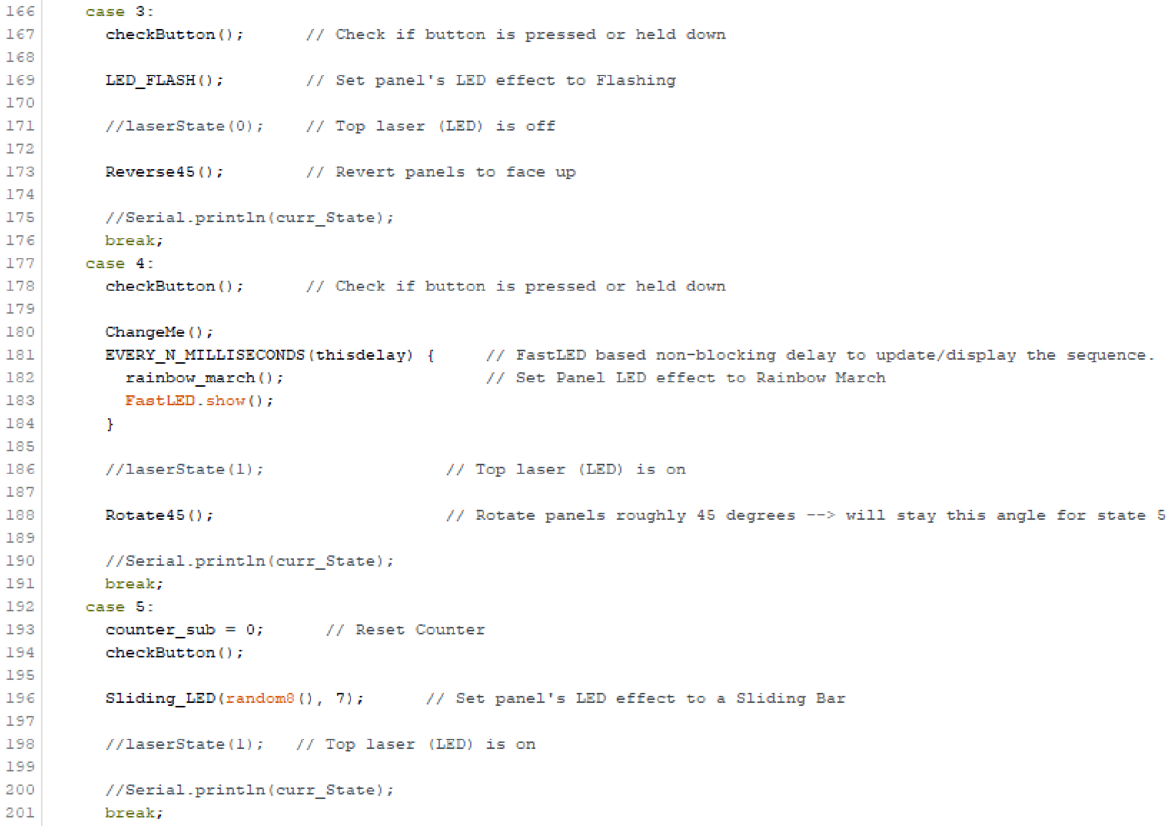
```
/*---  Variables for LED Modes  ------------------
------------------------------------------------*/
// Storage Objects
CRGB leds[NUM_LEDS];            // Create LED Object
arduinoFFT FFT = arduinoFFT();  // Create FFT object

// For Sound Bar Audio Reactive Mode
double vReal[SAMPLES];
double vImag[SAMPLES];
int Intensity[xres] = { };     // initialize Frequency Intensity to zero
int Displacement = 1;

// For LED Flashing Mode
unsigned long remTime = 0;     // will store last time LED was updated
const long interval = 1000;    // interval at which to blink (milliseconds)
bool is_red = false;

//For Color Cycle Mode
uint8_t blendRate = 50;               // How fast to blend.  Higher is slower.  [milliseconds]
CHSV colorStart = CHSV(96,255,255);   // starting color
CHSV colorTarget = CHSV(192,255,255); // target color
CHSV colorCurrent = colorStart;

// For Rainbow March
uint8_t thisdelay = 40;     // A delay value for the sequence(s)
uint8_t thishue = 0;        // Starting hue value.
int8_t thisrot = 1;         // Hue rotation speed. Includes direction.
uint8_t deltahue = 1;       // Hue change between pixels.
bool thisdir = 0;
/*------------------------------------------------
------------------------------------------------*/
```



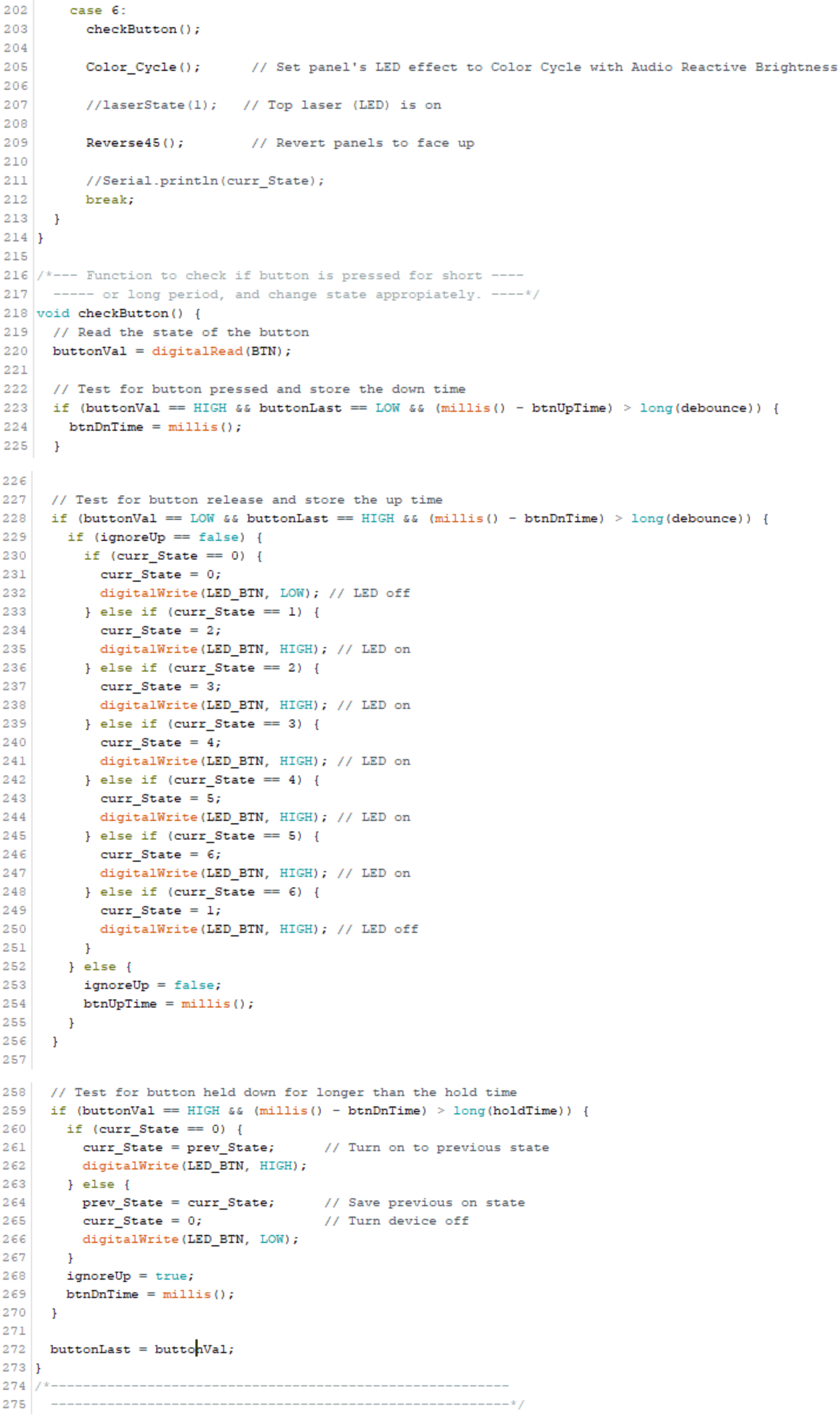
```

void setup() {
  delay(3000); // 3 second delay for recovery

  Serial.begin(115200);       // For Debugging

  pinMode(BTN, INPUT);        // Input for Button
  pinMode(LED_BTN, OUTPUT);   // Output for LED in Button
  pinMode(MIC_IN, INPUT);     // Input for microphone, analog]
  //pinMode(LAS_PIN OUTPUT);  // Output for Laser (LED) pin


  pinMode(A1, OUTPUT);                      // Set up PWM pins for motor control
  pinMode(A2, OUTPUT);
  pinMode(B1, OUTPUT);
  pinMode(B2, OUTPUT);
  pinMode(C1, OUTPUT);
  pinMode(C2, OUTPUT);

  FastLED.addLeds<LED_TYPE, LEDR_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection( TypicalLEDStrip ); // Initialize Right LED strips
  FastLED.addLeds<LED_TYPE, LEDL_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection( TypicalLEDStrip ); // Initialize Left LED strips
  FastLED.setBrightness(BRIGHTNESS);  // Default full brightness
}

```



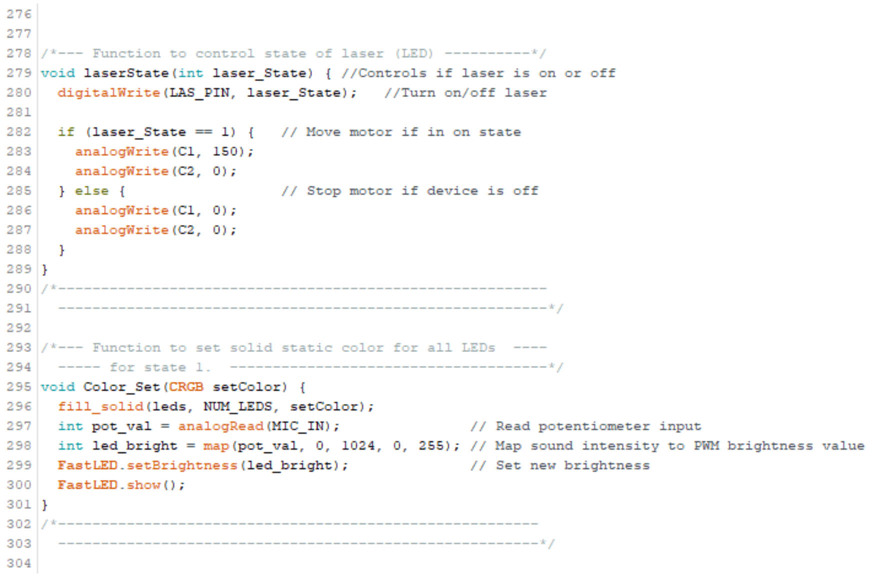
```
void loop() {
  // State Machine to define all 7 modes

  switch (curr_State) {
    case 0:
      checkButton();      // Check if button is pressed or held down

      // Turn motors off
      analogWrite(A1,0);
      analogWrite(A2,0);
      analogWrite(B1,0);
      analogWrite(B2,0);
      analogWrite(C1,0);
      analogWrite(C2,0);

      //Turn all LEDS off
      fill_solid(leds, NUM_LEDS, CRGB::Black);
      FastLED.show();
      laserState(0);

      //Serial.println(curr_State);
      break;
    case 1:
      counter_sub = 0;
      checkButton();

      Color_Set(CRGB::Crimson);   // Set Panel LEDs to Crimson
      //laserState(0);            // Top laser (LED) is off

      Rotate45();                 // Rotate panels roughly 45 degrees --> will stay this angle for state 2

      //Serial.println(analogRead(A1));
      break;
    case 2:
      counter = 0;      // Reset Counter

      checkButton();    // Check if button is pressed or held down

      Color_Cycle();    // Set panel's LED effect to Color Cycle

      laserState(0);    // Top laser (LED) is off

      //Serial.println(curr_State);
      break;
```



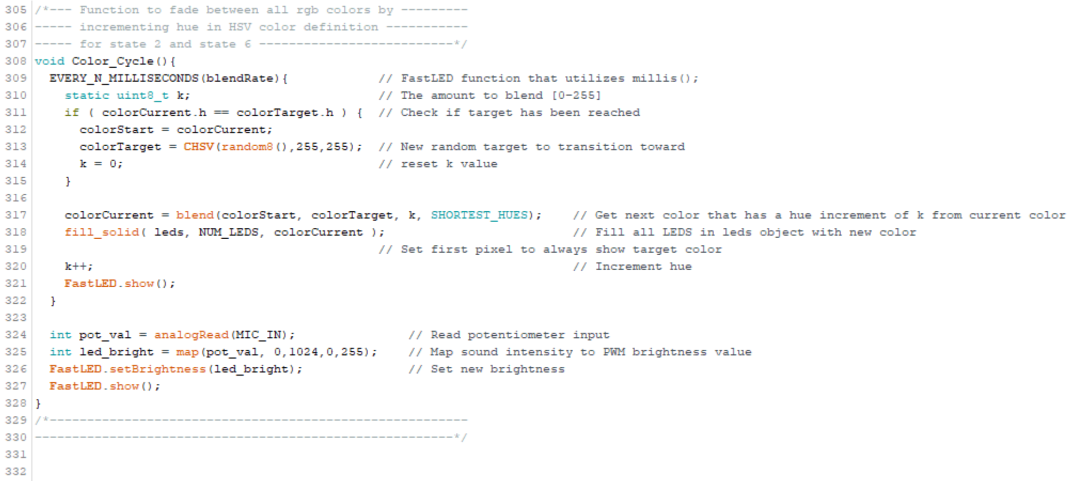
```
    case 3:
      checkButton();      // Check if button is pressed or held down

      LED_FLASH();        // Set panel's LED effect to Flashing

      //laserState(0);    // Top laser (LED) is off

      Reverse45();        // Revert panels to face up

      //Serial.println(curr_State);
      break;
    case 4:
      checkButton();      // Check if button is pressed or held down

      ChangeMe();
      EVERY_N_MILLISECONDS(thisdelay) {     // FastLED based non-blocking delay to update/display the sequence.
        rainbow_march();                    // Set Panel LED effect to Rainbow March
        FastLED.show();
      }

      //laserState(1);                  // Top laser (LED) is on

      Rotate45();                       // Rotate panels roughly 45 degrees --> will stay this angle for state 5

      //Serial.println(curr_State);
      break;
    case 5:
      counter_sub = 0;      // Reset Counter
      checkButton();

      Sliding_LED(random8(), 7);      // Set panel's LED effect to a Sliding Bar

      //laserState(1);   // Top laser (LED) is on

      //Serial.println(curr_State);
      break;
```



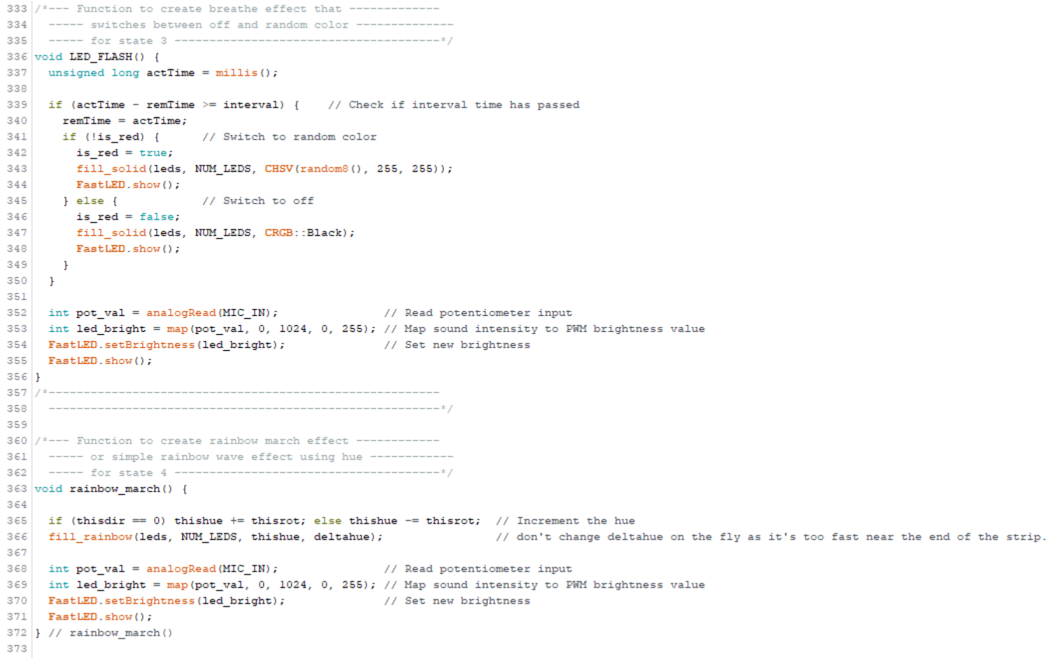
```
    case 6:
      checkButton();

      Color_Cycle();      // Set panel's LED effect to Color Cycle with Audio Reactive Brightness

      //laserState(1);   // Top laser (LED) is on

      Reverse45();        // Revert panels to face up

      //Serial.println(curr_State);
      break;
  }
}

/*--- Function to check if button is pressed for short ----
  ----- or long period, and change state appropiately. ----*/
void checkButton() {
  // Read the state of the button
  buttonVal = digitalRead(BTN);

  // Test for button pressed and store the down time
  if (buttonVal == HIGH && buttonLast == LOW && (millis() - btnUpTime) > long(debounce)) {
    btnDnTime = millis();
  }

  // Test for button release and store the up time
  if (buttonVal == LOW && buttonLast == HIGH && (millis() - btnDnTime) > long(debounce)) {
    if (ignoreUp == false) {
      if (curr_State == 0) {
        curr_State = 0;
        digitalWrite(LED_BTN, LOW); // LED off
      } else if (curr_State == 1) {
        curr_State = 2;
        digitalWrite(LED_BTN, HIGH); // LED on
      } else if (curr_State == 2) {
        curr_State = 3;
        digitalWrite(LED_BTN, HIGH); // LED on
      } else if (curr_State == 3) {
        curr_State = 4;
        digitalWrite(LED_BTN, HIGH); // LED on
      } else if (curr_State == 4) {
        curr_State = 5;
        digitalWrite(LED_BTN, HIGH); // LED on
      } else if (curr_State == 5) {
        curr_State = 6;
        digitalWrite(LED_BTN, HIGH); // LED on
      } else if (curr_State == 6) {
        curr_State = 1;
        digitalWrite(LED_BTN, HIGH); // LED off
      }
    } else {
      ignoreUp = false;
      btnUpTime = millis();
    }
  }

  // Test for button held down for longer than the hold time
  if (buttonVal == HIGH && (millis() - btnDnTime) > long(holdTime)) {
    if (curr_State == 0) {
      curr_State = prev_State;      // Turn on to previous state
      digitalWrite(LED_BTN, HIGH);
    } else {
      prev_State = curr_State;      // Save previous on state
      curr_State = 0;               // Turn device off
      digitalWrite(LED_BTN, LOW);
    }
    ignoreUp = true;
    btnDnTime = millis();
  }

  buttonLast = buttonVal;
}
/*--------------------------------------------------------
  --------------------------------------------------------*/
```



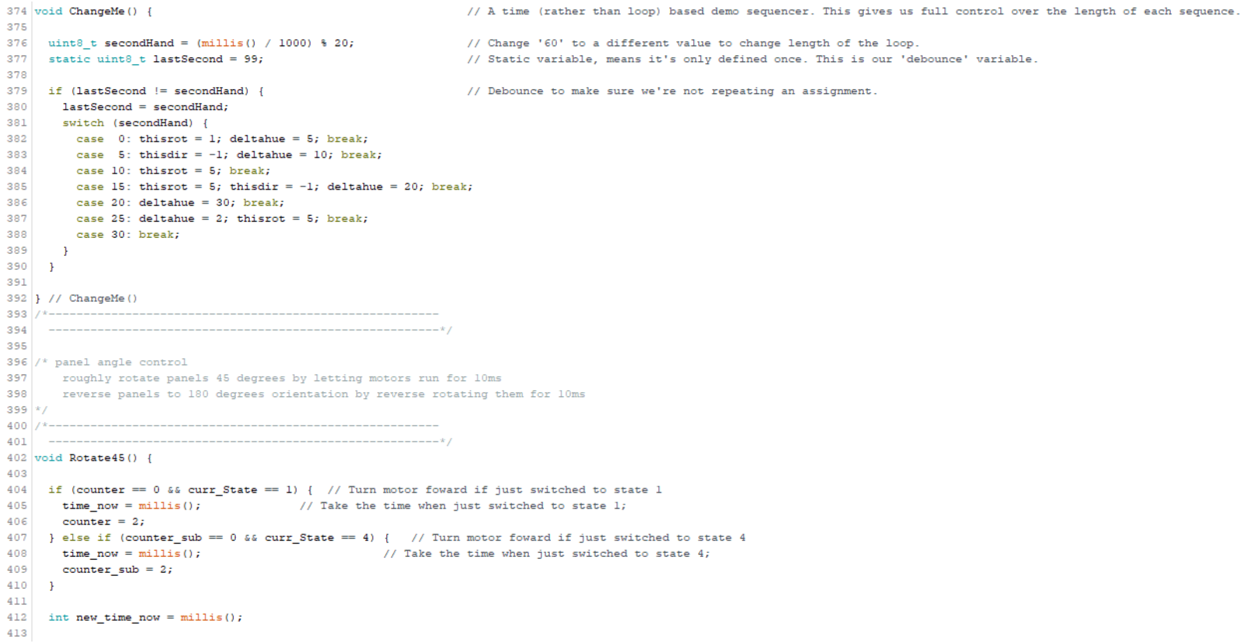
```


/*--- Function to control state of laser (LED) ----------*/
void laserState(int laser_State) { //Controls if laser is on or off
  digitalWrite(LAS_PIN, laser_State);   //Turn on/off laser

  if (laser_State == 1) {   // Move motor if in on state
    analogWrite(C1, 150);
    analogWrite(C2, 0);
  } else {                  // Stop motor if device is off
    analogWrite(C1, 0);
    analogWrite(C2, 0);
  }
}
/*--------------------------------------------------------
  --------------------------------------------------------*/

/*--- Function to set solid static color for all LEDs  ----
  ----- for state 1.  ------------------------------------*/
void Color_Set(CRGB setColor) {
  fill_solid(leds, NUM_LEDS, setColor);
  int pot_val = analogRead(MIC_IN);                 // Read potentiometer input
  int led_bright = map(pot_val, 0, 1024, 0, 255); // Map sound intensity to PWM brightness value
  FastLED.setBrightness(led_bright);              // Set new brightness
  FastLED.show();
}
/*-------------------------------------------------------
  -------------------------------------------------------*/

```



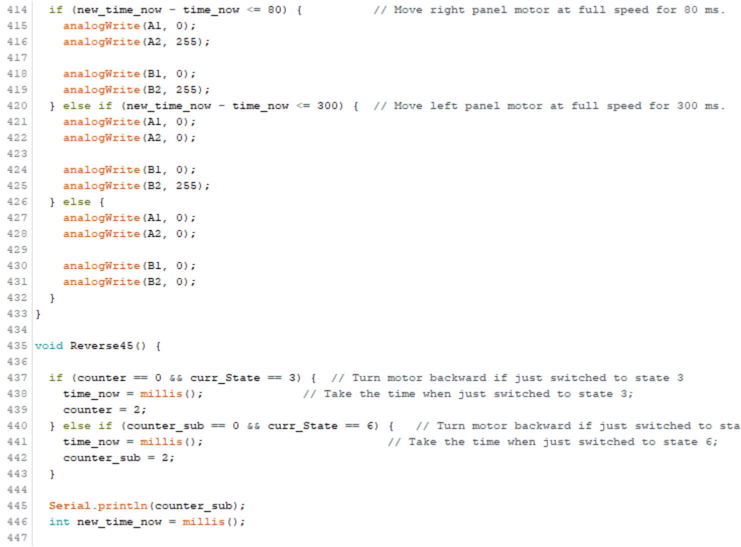
```
/*--- Function to fade between all rgb colors by ---------
----- incrementing hue in HSV color definition -----------
----- for state 2 and state 6 -------------------------*/
void Color_Cycle(){
  EVERY_N_MILLISECONDS(blendRate){            // FastLED function that utilizes millis();
    static uint8_t k;                         // The amount to blend [0-255]
    if ( colorCurrent.h == colorTarget.h ) {  // Check if target has been reached
      colorStart = colorCurrent;
      colorTarget = CHSV(random8(),255,255);  // New random target to transition toward
      k = 0;                                  // reset k value
    }

    colorCurrent = blend(colorStart, colorTarget, k, SHORTEST_HUES);    // Get next color that has a hue increment of k from current color
    fill_solid( leds, NUM_LEDS, colorCurrent );                          // Fill all LEDS in leds object with new color
                                              // Set first pixel to always show target color
    k++;                                                                 // Increment hue
    FastLED.show();
  }

  int pot_val = analogRead(MIC_IN);               // Read potentiometer input
  int led_bright = map(pot_val, 0,1024,0,255);    // Map sound intensity to PWM brightness value
  FastLED.setBrightness(led_bright);              // Set new brightness
  FastLED.show();
}
/*-------------------------------------------------------
--------------------------------------------------------*/


```



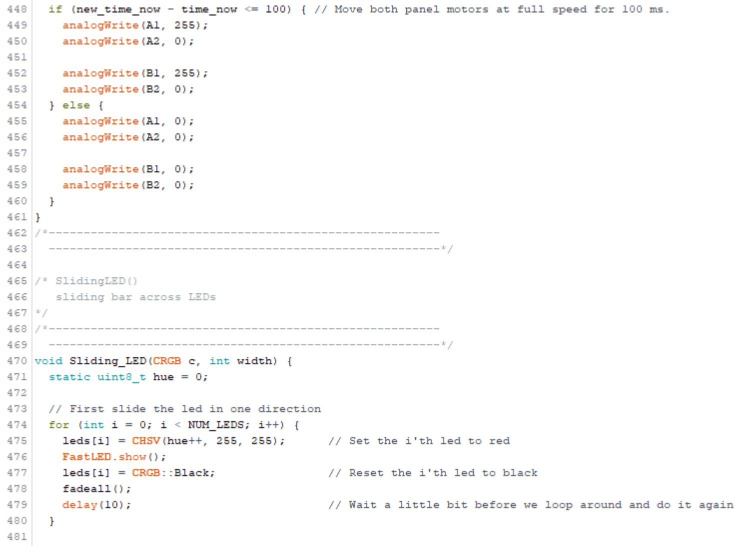
```
/*--- Function to create breathe effect that ------------
  ----- switches between off and random color -------------
  ----- for state 3 -------------------------------------*/
void LED_FLASH() {
  unsigned long actTime = millis();

  if (actTime - remTime >= interval) {    // Check if interval time has passed
    remTime = actTime;
    if (!is_red) {      // Switch to random color
      is_red = true;
      fill_solid(leds, NUM_LEDS, CHSV(random8(), 255, 255));
      FastLED.show();
    } else {            // Switch to off
      is_red = false;
      fill_solid(leds, NUM_LEDS, CRGB::Black);
      FastLED.show();
    }
  }

  int pot_val = analogRead(MIC_IN);               // Read potentiometer input
  int led_bright = map(pot_val, 0, 1024, 0, 255); // Map sound intensity to PWM brightness value
  FastLED.setBrightness(led_bright);              // Set new brightness
  FastLED.show();
}
/*-------------------------------------------------------
  -------------------------------------------------------*/

/*--- Function to create rainbow march effect -----------
  ----- or simple rainbow wave effect using hue ------------
  ----- for state 4 -------------------------------------*/
void rainbow_march() {

  if (thisdir == 0) thishue += thisrot; else thishue -= thisrot;  // Increment the hue
  fill_rainbow(leds, NUM_LEDS, thishue, deltahue);                // don't change deltahue on the fly as it's too fast near the end of the strip.

  int pot_val = analogRead(MIC_IN);               // Read potentiometer input
  int led_bright = map(pot_val, 0, 1024, 0, 255); // Map sound intensity to PWM brightness value
  FastLED.setBrightness(led_bright);              // Set new brightness
  FastLED.show();
} // rainbow_march()

```



```
void ChangeMe() {                                             // A time (rather than loop) based demo sequencer. This gives us full control over the length of each sequence.

  uint8_t secondHand = (millis() / 1000) % 20;                // Change '60' to a different value to change length of the loop.
  static uint8_t lastSecond = 99;                             // Static variable, means it's only defined once. This is our 'debounce' variable.

  if (lastSecond != secondHand) {                             // Debounce to make sure we're not repeating an assignment.
    lastSecond = secondHand;
    switch (secondHand) {
      case  0: thisrot = 1; deltahue = 5; break;
      case  5: thisdir = -1; deltahue = 10; break;
      case 10: thisrot = 5; break;
      case 15: thisrot = 5; thisdir = -1; deltahue = 20; break;
      case 20: deltahue = 30; break;
      case 25: deltahue = 2; thisrot = 5; break;
      case 30: break;
    }
  }

} // ChangeMe()
/*--------------------------------------------------------
  --------------------------------------------------------*/

/* panel angle control
    roughly rotate panels 45 degrees by letting motors run for 10ms
    reverse panels to 180 degrees orientation by reverse rotating them for 10ms
*/
/*--------------------------------------------------------
  --------------------------------------------------------*/
void Rotate45() {

  if (counter == 0 && curr_State == 1) {  // Turn motor foward if just switched to state 1
    time_now = millis();              // Take the time when just switched to state 1;
    counter = 2;
  } else if (counter_sub == 0 && curr_State == 4) {   // Turn motor foward if just switched to state 4
    time_now = millis();                          // Take the time when just switched to state 4;
    counter_sub = 2;
  }

  int new_time_now = millis();

```



```
  if (new_time_now - time_now <= 80) {          // Move right panel motor at full speed for 80 ms.
    analogWrite(A1, 0);
    analogWrite(A2, 255);

    analogWrite(B1, 0);
    analogWrite(B2, 255);
  } else if (new_time_now - time_now <= 300) {  // Move left panel motor at full speed for 300 ms.
    analogWrite(A1, 0);
    analogWrite(A2, 0);

    analogWrite(B1, 0);
    analogWrite(B2, 255);
  } else {
    analogWrite(A1, 0);
    analogWrite(A2, 0);

    analogWrite(B1, 0);
    analogWrite(B2, 0);
  }
}

void Reverse45() {

  if (counter == 0 && curr_State == 3) {  // Turn motor backward if just switched to state 3
    time_now = millis();               // Take the time when just switched to state 3;
    counter = 2;
  } else if (counter_sub == 0 && curr_State == 6) {   // Turn motor backward if just switched to state 6
    time_now = millis();                           // Take the time when just switched to state 6;
    counter_sub = 2;
  }

  Serial.println(counter_sub);
  int new_time_now = millis();

```



```
  if (new_time_now - time_now <= 100) { // Move both panel motors at full speed for 100 ms.
    analogWrite(A1, 255);
    analogWrite(A2, 0);

    analogWrite(B1, 255);
    analogWrite(B2, 0);
  } else {
    analogWrite(A1, 0);
    analogWrite(A2, 0);

    analogWrite(B1, 0);
    analogWrite(B2, 0);
  }
}
/*--------------------------------------------------------
  --------------------------------------------------------*/

/* SlidingLED()
   sliding bar across LEDs
*/
/*--------------------------------------------------------
  --------------------------------------------------------*/
void Sliding_LED(CRGB c, int width) {
  static uint8_t hue = 0;

  // First slide the led in one direction
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = CHSV(hue++, 255, 255);      // Set the i'th led to red
    FastLED.show();
    leds[i] = CRGB::Black;                // Reset the i'th led to black
    fadeall();
    delay(10);                            // Wait a little bit before we loop around and do it again
  }

  // Now go in the other direction.
  for (int i = (NUM_LEDS) - 1; i >= 0; i--) {
    leds[i] = CHSV(hue++, 255, 255);      // Set the i'th led to red
    FastLED.show();
    leds[i] = CRGB::Black;                // Reset the i'th led to black
    fadeall();
    delay(10);                            // Wait a little bit before we loop around and do it again
  }

  int pot_val = analogRead(MIC_IN);               // Read potentiometer input
  int led_bright = map(pot_val, 0, 1024, 0, 255); // Map sound intensity to PWM brightness value
  FastLED.setBrightness(led_bright);              // Set new brightness
  FastLED.show();
}

void fadeall() {
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i].nscale8(250);
  }
}
/*---------------------------------------------------------
  --------------------------------------------------------*/
```