Thrash Mixer

A project by Tom Yaniv



The Thrash Mixer was a project that I worked on in my 2022 Spring Semester at the College of San Mateo for a class project devolving around creating new musical interfaces and technologies. My device intended to use a Gyroscope sensor to output X, Y, and Z variables from 0 to 300 depending on its axis and distance against the ground. In return, the current location of the gyroscope would be transcribed and transmitted through MIDI signals into Logic Pro that would send knob messages based on the MIDI amount, the number corresponding to the gyroscope's location, and in turn, changing the Crossfade controls between two tracks on the preset Logic Program. I built the entire project on C++ and used Arduino's programs through an Arduino Duo connected to the Gyroscope sensor patched on the Skateboard.

My goal with this project was to develop a more interactive method for performances and generate more engagement value for smaller artists and performers who want a more adaptive way to work with their music. Although wired up and unable to be used portably, the end goal of this project was to have the Skateboard work entirely off a wireless connection to a computer anchor that would run all the programs and crossfade the queued tracks. Transportability and mobility would allow performers to use the technology anywhere they pleased, not just in stationary positions locked to a standing computer, and would allow for more extensive performance opportunities depending on the planned performance method, whether in a concert venue or an outdoor setting like a Skatepark, Parking Lot or other such gathering areas. I plan to finish my work to have the Skateboard properly wireless with Java programming or continue working with boards using C++.

Code for XYZ input to MIDI value:

```
/*
 * MIDIUSB_test.ino
 *
 * Created: 4/6/2015 10:47:08 AM
 * Author: gurbrinder grewal
 * Modified by Arduino LLC (2015)
 */

#include "MIDIUSB.h"

#include "SparkFunLSM6DSO.h"
#include "Wire.h"
//#include "SPI.h"

LSM6DSO myIMU; //Default constructor is I2C, addr 0x6B



// First parameter is the event type (0x09 = note on, 0x08 = note off).
// Second parameter is note-on/note-off, combined with the channel.
// Channel can be anything between 0-15. Typically reported to the user as
1-16.
// Third parameter is the note number (48 = middle C).
// Fourth parameter is the velocity (64 = normal, 127 = fastest).

void noteOn(byte channel, byte pitch, byte velocity) {
  midiEventPacket_t noteOn = {0x09, 0x90 | channel, pitch, velocity};
  MidiUSB.sendMIDI(noteOn);
}

void noteOff(byte channel, byte pitch, byte velocity) {
  midiEventPacket_t noteOff = {0x08, 0x80 | channel, pitch, velocity};
  MidiUSB.sendMIDI(noteOff);
}

void setup() {
  Serial.begin(115200);
  delay(500);
```

```
  Wire.begin();
  delay(10);
  if( myIMU.begin() )
    Serial.println("Ready.");
  else {
    Serial.println("Could not connect to IMU.");
    Serial.println("Freezing");
  }

  if( myIMU.initialize(BASIC_SETTINGS) )
    Serial.println("Loaded Settings.");

}

// First parameter is the event type (0x0B = control change).
// Second parameter is the event type, combined with the channel.
// Third parameter is the control number number (0-119).
// Fourth parameter is the control value (0-127).

void controlChange(byte channel, byte control, byte value) {
  midiEventPacket_t event = {0x0B, 0xB0 | channel, control, value};
  MidiUSB.sendMIDI(event);
}

void loop() {

  //Get all parameters
  Serial.print("\nAccelerometer:\n");
  Serial.print(" X = ");
  Serial.println(myIMU.readFloatAccelX(), 3);
  Serial.print(" Y = ");
  Serial.println(myIMU.readFloatAccelY(), 3);
  Serial.print(" Z = ");
  Serial.println(myIMU.readFloatAccelZ(), 3);

  Serial.print("\nGyroscope:\n");
  Serial.print(" X = ");
  Serial.println(myIMU.readFloatGyroX(), 3);
  Serial.print(" Y = ");
```

```
  Serial.println(myIMU.readFloatGyroY(), 3);
  Serial.print(" Z = ");
  Serial.println(myIMU.readFloatGyroZ(), 3);

  Serial.print("\nThermometer:\n");
  Serial.print(" Degrees F = ");
  Serial.println(myIMU.readTempF(), 3);


  int potVal =
 (myIMU.readFloatAccelZ()) * 1000;//read data from Z axis

  byte parameterz = map(potVal, -1000, 1000, 0, 127);

  controlChange(0, 21, parameterz);   // Channel 0, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

  controlChange(0, 21, parameterz);  // Channel 0, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

    potVal =
 (myIMU.readFloatAccelX()) * 1000;//read data from X axis

  byte parameterx = map(potVal, -1000, 1000, 0, 127);

  controlChange(1, 21, parameterx);   // Channel 1, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

  controlChange(1, 21, parameterx);  // Channel 1, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

  potVal =
```

```
 (myIMU.readFloatAccelY()) * 1000;//read data from Y axis

  byte parametery = map(potVal, -1000, 1000, 0, 127);

  controlChange(2, 21, parametery);   // Channel 2, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

  controlChange(2, 21, parametery);  // Channel 2, middle C, normal
velocity
  MidiUSB.flush();
  delay(100);

Serial.print(" Z = ");
  Serial.println(parameterz);

 Serial.print(" X = ");
  Serial.println(parameterx);

 Serial.print(" Y = ");
  Serial.println(parametery);

  delay(1000);
  // controlChange(0, 10, 65); // Set the value of controller 10 on
channel 0 to 65
}
```

The rest of the programming was done through Logic's MIDI controller assignment by way of assigning the MIDI outputs of the Arduino to a new controller interface on Logic and using the MIDI messages as signals for a digital Crossfade fader controller between two tracks.