#include "Particle.h"

#ifndef PARTICLE

#include <Wire.h>

#include <SPI.h>

#endif

#include <Adafruit\_LIS3DH.h>

// led include

#include <neopixel.h>

#include "math.h"

// Used for software SPI

#define LIS3DH\_CLK 13

#define LIS3DH\_MISO 12

#define LIS3DH\_MOSI 11

// Used for hardware & software SPI

#define LIS3DH\_CS 10

#define G\_EPSILON 0.05 //unit is m/s^2

#define ROCKING\_BIAS 0

#define G 0.98 //unit is m/s^2

#define UPDATE\_TIME 200 //ms

#define PROCESS\_TIME 1000 //ms

#define RECOVER\_TIME 2000 //ms

#define SEND\_TIME 3000 //ms

#define EGGA //egg a

//LED defines

#define CAM\_PIN D3

#define VIBE\_PIN D4 //changed from D2

#define PIXEL\_PIN D2

#define PIXEL\_COUNT 12

#define PIXEL\_TYPE WS2812B

// max frame count in a clip

#define FRAME\_COUNT\_MAX 10

struct vec4

{

float x;

float y;

float z;

float w;

vec4() : x(0), y(0), z(0), w(0)

{

}

vec4(float a, float b, float c, float d) : x(a), y(b), z(c), w(d)

{

}

vec4 operator=(const vec4 &a)

{

x = a.x;

y = a.y;

z = a.z;

w = a.w;

return a;

}

vec4 operator+(const vec4 &a)

{

return vec4(x + a.x, y + a.y, z + a.z, w + a.w);

}

vec4 operator-(const vec4 &a)

{

return vec4(x - a.x, y - a.y, z - a.z, w - a.w);

}

vec4 operator\*(const float a)

{

return vec4(x \* a, y \* a, z \* a, w \* a);

}

vec4 operator/(const float a)

{

return vec4(x / a, y / a, z / a, w / a);

}

float length()

{

return sqrt(x \* x + y \* y + z \* z + w \* w);

}

};

struct vec3

{

float x;

float y;

float z;

vec3() : x(0), y(0), z(0)

{

}

vec3(float a, float b, float c) : x(a), y(b), z(c)

{

}

vec3 operator=(const vec3 &a)

{

x = a.x;

y = a.y;

z = a.z;

return a;

}

vec3 operator+(const vec3 &a)

{

return vec3(x + a.x, y + a.y, z + a.z);

}

vec3 operator-(const vec3 &a)

{

return vec3(x - a.x, y - a.y, z - a.z);

}

vec3 operator\*(const float a)

{

return vec3(x \* a, y \* a, z \* a);

}

vec3 operator/(const float a)

{

return vec3(x / a, y / a, z / a);

}

float length()

{

return sqrt(x \* x + y \* y + z \* z);

}

};

vec3 operator\*(const float a, vec3 v)

{

return vec3(v.x \* a, v.y \* a, v.z \* a);

}

vec4 operator\*(const float a, vec4 v)

{

return vec4(v.x \* a, v.y \* a, v.z \* a, v.w \* a);

}

float dot(const vec3 &a, const vec3 &b)

{

return a.x \* b.x + a.y \* b.y + a.z \* b.z;

}

vec3 cross(const vec3 &a, const vec3 &b)

{

return vec3(a.y \* b.z - a.z \* b.y, a.z \* b.x - a.x \* b.z, a.x \* b.y - a.y \* b.x);

}

vec3 normalize(vec3 a)

{

float l = a.length();

return a / l;

}

vec4 normalize(vec4 a)

{

float l = a.length();

return a / l;

}

template <class T>

class Clip

{

private:

T frames[FRAME\_COUNT\_MAX];

float durations[FRAME\_COUNT\_MAX];

int frameCount;

float tMax;

public:

Clip();

void AddFrame(T frame, float duration);

T Lerp(float t);

};

template <class T>

Clip<T>::Clip()

{

frameCount = 0;

tMax = 0;

}

template <class T>

void Clip<T>::AddFrame(T frame, float duration)

{

if (frameCount < FRAME\_COUNT\_MAX)

{

if (duration < 0)

duration = -duration;

frames[frameCount] = frame;

durations[frameCount] = duration;

tMax += duration;

frameCount++;

}

}

template <class T>

T Clip<T>::Lerp(float t)

{

//abs

if (t < 0)

t = -t;

//float mod

if (t > tMax)

{

int nt = t / tMax;

t = t - nt \* tMax;

}

//evaluate

int left = 0;

for (int i = 0; i < frameCount; i++)

{

if (durations[i] == 0)

continue;

if (t > durations[i])

t -= durations[i];

else

{

left = i;

break;

}

}

int right = (left + 1) % frameCount;

float u = t / durations[left];

T result = u \* frames[right] + (1 - u) \* frames[left];

return result;

}

enum State

{

Depressed,

Lonely,

Chill,

Neutral,

Annoyed,

Agitated,

MadMax,

StateCount

};

//multithread

SYSTEM\_THREAD(ENABLED);

//accelerometer

Adafruit\_LIS3DH lis = Adafruit\_LIS3DH();

//led strip

Adafruit\_NeoPixel strip(PIXEL\_COUNT, PIXEL\_PIN, PIXEL\_TYPE);

void UpdateTimerHandler();

void ProcessTimerHandler();

void RecoverTimerHandler();

void SendTimerHandler();

//void AngerEventHandler(const char \*event, const char \*data);

Timer updateTimer(UPDATE\_TIME, UpdateTimerHandler);

Timer processTimer(PROCESS\_TIME, ProcessTimerHandler);

Timer recoverTimer(RECOVER\_TIME, RecoverTimerHandler);

Timer sendTimer(SEND\_TIME, SendTimerHandler);

Thread threadMotor("threadMotor", threadMotorProc);

Thread threadLed("threadLed", threadLedProc);

volatile float anger; //volatile for shared variable

volatile unsigned int curTimeColor; //use integer so that it's easier to spot a bug

volatile unsigned int curTimeMotor; //use integer so that it's easier to spot a bug

volatile int curState;

vec4 curColor;

vec3 down;

vec3 curAccel;

bool readyToSend;

bool isG;

float angerDeltaSend;

const int angerRecoverStep = 15;

const float angerInit = 50.f;

const float angerMax = 100.f;

const float angerMin = 0.f;

const float angerDeltaSendFactor = 0.8f;

const float angerDeltaHorizontalFactor = 30.f;

const float angerDeltaVerticalFactor = 9.f;

Clip<vec4> colorClips[StateCount];

#ifdef EGGA

const char \*angerSendEvent = "Egg\_A\_Send";

const char \*angerReceiveEvent = "Egg\_B\_Send";

#endif

#ifdef EGGB

const char \*angerSendEvent = "Egg\_B\_Send";

const char \*angerReceiveEvent = "Egg\_A\_Send";

#endif

#if defined(ARDUINO\_ARCH\_SAMD)

// for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!

#define Serial SerialUSB

#endif

void setup(void)

{

#ifndef ESP8266

while (!Serial)

; // will pause Zero, Leonardo, etc until serial console opens

#endif

Serial.begin(9600);

// motor initiali

pinMode(CAM\_PIN, OUTPUT);

pinMode(VIBE\_PIN, OUTPUT);

delay(100);

// strip initialization

strip.begin();

strip.setBrightness(200);

strip.show();

// initialization

down = vec3(0, 0, -1);

curAccel = vec3(0, 0, -1);

readyToSend = false;

anger = angerInit;

angerDeltaSend = 0.f;

curTimeColor = 0;

curState = Neutral;

curColor = vec4(0, 0, 0, 0);

//TODO:

//You can add 20 frames maximum for each clip

//vec4 stands for RGB and transparency, the range is from 0 to 1

//second parameter here is how long the frame is going to last

//It is not a point in time, because this way I don't need to do the sorting

//and you don't need to specify the order of the frames. The order is the same as in which they are added.

//\*\*\*\*\*\*\*\*\*\*\*\*record color clips begin\*\*\*\*\*\*\*\*\*\*\*\*//

//Depressed

colorClips[Depressed].AddFrame(vec4(0.12, 0, 1, 0.9), 500.f); //blue

colorClips[Depressed].AddFrame(vec4(0, 0, 0, 0), 200.f);

colorClips[Depressed].AddFrame(vec4(0.7, 0, 1, 0.3), 500.f); //purple

colorClips[Depressed].AddFrame(vec4(0, 0, 0, 0), 200.f);

colorClips[Depressed].AddFrame(vec4(0.12, 0, 1, 0.9), 500.f); //blue

//Lonely

colorClips[Lonely].AddFrame(vec4(0.1, 1, 0.45, 0.4), 400.f); //green

colorClips[Lonely].AddFrame(vec4(0, 0, 0, 0), 200.f);

colorClips[Lonely].AddFrame(vec4(0.7, 0, 1, 0.3), 400.f); //purple

colorClips[Lonely].AddFrame(vec4(0, 0, 0, 0), 200.f);

colorClips[Lonely].AddFrame(vec4(0.12, 0, 1, 0.9), 400.f); //blue

//Chill

colorClips[Chill].AddFrame(vec4(0.1, 1, 0.45, 0.6), 350.f); //green

colorClips[Chill].AddFrame(vec4(0, 0, 0, 0), 100.f);

colorClips[Chill].AddFrame(vec4(0.3, .8, 1, 0.8), 350.f); //cyan

colorClips[Chill].AddFrame(vec4(0, 0, 0, 0), 100.f);

colorClips[Chill].AddFrame(vec4(0.12, 0, 1, 0.9), 350.f); //blue

colorClips[Chill].AddFrame(vec4(0, 0, 0, 0), 100.f);

colorClips[Chill].AddFrame(vec4(0.7,0,1, 0.8), 350.f); //purple

colorClips[Chill].AddFrame(vec4(0, 0, 0, 0), 100.f);

//Neutral

colorClips[Neutral].AddFrame(vec4(0, 1, 0.15, 0.8), 300.f); //green

colorClips[Neutral].AddFrame(vec4(0, 0, 0, 0), 80.f);

colorClips[Neutral].AddFrame(vec4(0.3, .8, 1, 0.8), 300.f); //cyan

colorClips[Neutral].AddFrame(vec4(0, 0, 0, 0), 80.f);

colorClips[Neutral].AddFrame(vec4(0.8, 0.2, 0.1, 0.9), 300.f); //peach

colorClips[Neutral].AddFrame(vec4(0, 0, 0, 0), 80.f);

colorClips[Neutral].AddFrame(vec4(0.7,0,1, 0.8), 300.f); //purple

colorClips[Neutral].AddFrame(vec4(0, 0, 0, 0), 80.f);

//Annoyed

colorClips[Annoyed].AddFrame(vec4(0.8, 0.2, 0.1, 0.9), 200.f); //peach

colorClips[Annoyed].AddFrame(vec4(0, 0, 0, 0), 90.f);

colorClips[Annoyed].AddFrame(vec4(0.7,0,1, 0.8), 200.f); //purple

colorClips[Annoyed].AddFrame(vec4(0, 0, 0, 0), 90.f);

colorClips[Annoyed].AddFrame(vec4(1,0,0.48, 1), 200.f); //magenta

colorClips[Annoyed].AddFrame(vec4(0, 0, 0, 0), 90.f);

//Agitated

colorClips[Agitated].AddFrame(vec4(1, 0.56, 0, 0.9), 100.f); //orange

colorClips[Agitated].AddFrame(vec4(0, 0, 0, 0), 90.f);

colorClips[Agitated].AddFrame(vec4(1, 0, 0, 0.8), 100.f); //red

colorClips[Agitated].AddFrame(vec4(0, 0, 0, 0), 90.f);

colorClips[Agitated].AddFrame(vec4(1, 0.56, 0, 0.9), 90.f); //orange

colorClips[Agitated].AddFrame(vec4(0, 0, 0, 0), 90.f);

colorClips[Agitated].AddFrame(vec4(1, 0, 0.48, 1), 100.f); //magenta

colorClips[Agitated].AddFrame(vec4(0, 0, 0, 0), 90.f);

//MadMax

colorClips[MadMax].AddFrame(vec4(1, 0.8, 0, 0.9), 100.f); //yellow

colorClips[MadMax].AddFrame(vec4(0, 0, 0, 0), 20.f);

colorClips[MadMax].AddFrame(vec4(1, 0, 0, 0.9), 100.f); //red

colorClips[MadMax].AddFrame(vec4(0, 0, 0, 0), 20.f);

colorClips[MadMax].AddFrame(vec4(1, 0, 0.48, 1), 100.f); //magenta

colorClips[MadMax].AddFrame(vec4(0, 0, 0, 0), 20.f);

colorClips[MadMax].AddFrame(vec4(1, 0.8, 0, 0.9), 100.f); //yellow

colorClips[MadMax].AddFrame(vec4(0, 0, 0, 0), 20.f);

//\*\*\*\*\*\*\*\*\*\*\*\*record clips\*\*\*\*\*\*\*\*\*\*\*\*//

Serial.println("LIS3DH test!");

if (!lis.begin(0x18))

{ // change this to 0x19 for alternative i2c address

Serial.println("Couldnt start");

while (1)

;

}

Serial.println("LIS3DH found!");

lis.setRange(LIS3DH\_RANGE\_2\_G); // 2, 4, 8 or 16 G!

Serial.print("Range = ");

Serial.print(2 << lis.getRange());

Serial.println("G");

//start timers

updateTimer.start();

processTimer.start();

recoverTimer.start();

sendTimer.start();

#if defined EGGA || defined EGGB

Particle.subscribe(angerReceiveEvent, AngerEventHandler);

#endif

}

bool IsG(vec3 a)

{

float diff = a.length() - G;

//Serial.println(diff);

float diffAbs = diff > 0 ? diff : -diff;

//Serial.print(diffAbs);

//Serial.print(" ? ");

//Serial.println(G\_EPSILON);

if (diffAbs < G\_EPSILON)

return true;

else

return false;

}

State GetCurrentState(float anger)

{

if (anger < 20)

{

return Depressed;

}

else if (anger < 35)

{

return Lonely;

}

else if (anger < 45)

{

return Chill;

}

else if (anger < 55)

{

return Neutral;

}

else if (anger < 65)

{

return Annoyed;

}

else if (anger < 75)

{

return Agitated;

}

else

{

return MadMax;

}

}

void loop()

{

if (readyToSend)

{

#if defined EGGA || defined EGGB

Particle.publish(angerSendEvent, String(angerDeltaSend), PUBLIC);

#endif

angerDeltaSend = 0;

readyToSend = false;

}

int newState = GetCurrentState(anger); //convert State to integer

if (curState != newState)

{

curState = newState;

curTimeColor = 0;

}

}

float GetDeltaAnger(vec3 a)

{

vec3 norDown = normalize(down);

float projDownScalar = dot(norDown, curAccel); //we need this

vec3 projDown = projDownScalar \* norDown;

vec3 projPlane = curAccel - projDown;

float projPlaneLength = projPlane.length(); //we need this

//Serial.print("projDownScalar:"); Serial.println(projDownScalar, 6);

//Serial.print("projPlaneLength:"); Serial.println(projPlaneLength, 6);

float verticalAbs = projDownScalar - down.length();

if (verticalAbs < 0)

verticalAbs = -verticalAbs;

float horizontalAbs = projPlaneLength;

//Serial.print("vertical:"); Serial.println(verticalAbs, 6);

//Serial.print("horizontal:"); Serial.println(horizontalAbs, 6);

//decide shaking or rocking

float tangentSlope = 100; //init with infinity

if (horizontalAbs > 0)

tangentSlope = verticalAbs / horizontalAbs;

if ((verticalAbs < 0.3 || (tangentSlope > 0 && tangentSlope < 1.f + ROCKING\_BIAS)) //tan(45 degree) = 1

&& horizontalAbs < 0.5) //to prevent horizontal shake

{

//rocking

float deltaAnger = -angerDeltaHorizontalFactor \* horizontalAbs;

Serial.print(" Rocking = ");

Serial.println(deltaAnger, 6);

return deltaAnger;

}

else

{

//shaking

float deltaAnger = angerDeltaVerticalFactor \* verticalAbs;

Serial.print(" Shaking = ");

Serial.println(deltaAnger, 6);

return deltaAnger;

}

}

void UpdateTimerHandler()

{

lis.read(); // get X Y and Z data at once

vec3 raw(lis.x, lis.y, lis.z);

sensors\_event\_t event;

lis.getEvent(&event);

vec3 cooked(event.acceleration.x, event.acceleration.y, event.acceleration.z);

curAccel = cooked;

if (IsG(curAccel))

{

isG = true;

down = curAccel;

//Serial.println("isG");

}

else

isG = false;

}

void ProcessTimerHandler()

{

//Serial.print("X: "); Serial.print(curAccel.x);

//Serial.print(", Y: "); Serial.print(curAccel.y);

//Serial.print(", Z: "); Serial.print(curAccel.z);

//Serial.println(" m/s^2");

//if(IsG(curAccel))

//{

//down = curAccel;

//Serial.println("isG");

//}

if (isG)

{

//do nothing

}

else

{

float angerDelta = GetDeltaAnger(curAccel);

angerDeltaSend += angerDeltaSendFactor \* angerDelta;

anger += angerDelta;

if (anger < angerMin)

anger = angerMin;

if (anger > angerMax)

anger = angerMax;

}

Serial.printlnf("Anger = %f, curTimeColor = %d, curState = %d", anger, curTimeColor, curState);

}

void RecoverTimerHandler()

{

float angerRecover = (angerInit - anger) / angerRecoverStep;

//Serial.print(" \*Anger to be recovered = ");

//Serial.println(angerRecover, 6);

anger += angerRecover;

if (anger < angerMin)

anger = angerMin;

if (anger > angerMax)

anger = angerMax;

//Serial.print(" \*Anger after recovery = ");

//Serial.println(anger, 6);

}

void SendTimerHandler()

{

readyToSend = true;

}

void AngerEventHandler(const char \*event, const char \*data)

{

float angerDeltaReceive = atof(data);

Serial.print("AngerDeltaReceive = ");

Serial.println(angerDeltaReceive, 6);

}

//\*MULTI-THREADING STARTS HERE\*//

//motor thread

void threadMotorProc()

{

while (true)

{

AngerRanges(curState);

delay(100);

}

}

//led thread

void threadLedProc()

{

while (true)

{

vec4 targetColor = colorClips[curState].Lerp(curTimeColor++);

curColor = (curColor + targetColor) / 2.f; //targetColor;//

//curColor = newCurColor;

setLed(curColor.x, curColor.y, curColor.z, curColor.w);

//Serial.printlnf("targetColor = %f, %f, %f, %f", targetColor.x, targetColor.y, targetColor.z, targetColor.w);

//Serial.printlnf("newCurColor = %f, %f, %f, %f", newCurColor.x, newCurColor.y, newCurColor.z, newCurColor.w);

//Serial.printlnf("curColor = %f, %f, %f, %f\n", curColor.x, curColor.y, curColor.z, curColor.w);

}

}

void setLed(float r, float g, float b, float a)

{

for (int i = 0; i < PIXEL\_COUNT; i++)

{

strip.setPixelColor(i, 255 \* r \* a, 255 \* g \* a, 255 \* b \* a); //maintain color while fading brightness

}

strip.show();

delay(5); //this does not change pattern but just function as a way to throttle writing to led

}

void AngerRanges(int x)

{

Serial.printlnf("anger ranges %d", x);

if (x == Depressed)

{

DepressedMotor();

} // chilled out bro

else if (x == Lonely)

{

LonelyMotor();

} // calm + blue/darkpurple

else if (x == Chill)

{

ChillMotor();

} // calm/neut + green, cyan, blue

else if (x == Neutral)

{

NeutralMotor();

} // cyan, lite purple, peach

else if (x == Annoyed)

{

AnnoyedMotor();

} // mildly annoyed + magenta

else if (x == Agitated)

{

AgitatedMotor();

} // agitated + orange/magenta

else if (x == MadMax)

{

MadMaxMotor();

} // wtf dude + red/yellow

}

void DepressedMotor()

{ // breathes slowly, turning off for a bit each time

for (int i = 10; i < 30; i++)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

for (int i = 30; i > 10; i--)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

}

void LonelyMotor()

{ // slightly higher duty, quicker breathing cycle, light jolts from cam at regular intervals

for (int i = 20; i < 40; i++)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

if (i == 29)

{

analogWrite(CAM\_PIN, 75, 100);

delay(150);

}

else if (i == 31)

{

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

else

{

delay(150);

}

}

for (int i = 40; i > 20; i--)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

if (i == 31)

{

analogWrite(CAM\_PIN, 75, 100);

delay(150);

}

else if (i == 29)

{

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

else

{

delay(150);

}

}

}

void ChillMotor()

{ // ever so slightly higher duty cycle, jolts from cam twice as often at regular intervals

for (int i = 25; i < 50; i++)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

if (i == 45 | i == 35)

{

analogWrite(CAM\_PIN, 80, 100);

delay(150);

}

else if (i == 47 | i == 37)

{

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

else

{

delay(150);

}

}

for (int i = 50; i > 25; i--)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

if (i == 35 | i == 45)

{

analogWrite(CAM\_PIN, 80, 100);

delay(150);

}

else if (i == 33 | i == 43)

{

analogWrite(CAM\_PIN, 0, 100);

delay(150);

}

else

{

delay(150);

}

}

}

void NeutralMotor()

{ // similar to max chill state, but vibe is always on - stronger breathing

//Serial.println("Reach Neutral Breathe Function");

for (int i = 25; i < 70; i++)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

for (int i = 70; i > 25; i--)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

}

void AnnoyedMotor()

{ // slight vibe pulsing

for (int i = 60; i < 80; i++)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(30);

}

for (int i = 80; i > 60; i--)

{

int duty = (255 / 100) \* i;

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(30);

}

}

void AgitatedMotor()

{ // more aggressive pulsing, shocks from cam turning on/off

//ramp up

for (int i = 70; i < 90; i++)

{

int duty = (255 / 100) \* i;

if (i > 84 && i < 87)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 74 && i < 77)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 86)

{

analogWrite(VIBE\_PIN, 0, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

else

{

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

}

//ramp down

for (int i = 90; i > 70; i--)

{

int duty = (255 / 100) \* i;

if (i > 84 && i < 87)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 74 && i < 77)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i < 74)

{

analogWrite(VIBE\_PIN, 0, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

else

{

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

}

}

void MadMaxMotor()

{ // more aggressive pulsing, shocks from cam turning on/off

//ramp up

for (int i = 70; i < 100; i++)

{

int duty = (255 / 100) \* i;

if (i > 74 && i < 77)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 79 && i < 82)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 84 && i < 87)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 89 && i < 92)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 94 && i < 97)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else

{

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

}

//ramp down

for (int i = 100; i > 70; i--)

{

int duty = (255 / 100) \* i;

if (i > 74 && i < 77)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 79 && i < 82)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 84 && i < 87)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 89 && i < 92)

{

analogWrite(CAM\_PIN, 200, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else if (i > 94 && i < 97)

{

analogWrite(CAM\_PIN, 255, 100);

analogWrite(VIBE\_PIN, duty, 100);

delay(50);

}

else

{

analogWrite(VIBE\_PIN, duty, 100);

analogWrite(CAM\_PIN, 0, 100);

delay(50);

}

}

}