#include "CurieIMU.h"

#include "CuriePME.h"

#include <Wire.h>

#include "ir\_gesture\_vector.h"

#define gesturePin 7

#define IMULow -32768

#define IMUHigh 32767

/\* reading the accelerometer 100 times per second \*/

const unsigned int sampleRateHZ = 100;

enum ir\_command {Undefined, Forward, Reverse, Stop,

LeftSpin, RightSpin, LeftForward, RightForward, LeftReverse, RightReverse,

Record, Replay1, Replay2, Replay3, Replay4};

enum ir\_command category = Undefined;

byte vector\_imu[126];

unsigned int numSamples = 126;

void setup()

{

Serial.begin(9600);

while(!Serial);

Serial.println("IR Gesture Controller");

/\* Set button pin as input \*/

pinMode(gesturePin, INPUT);

// initialize digital pin LED\_BUILTIN as an output.

pinMode(LED\_BUILTIN, OUTPUT);

/\* Start the IMU (Inertial Measurement Unit), enable accelerometer only \*/

CurieIMU.begin(ACCEL);

CurieIMU.setAccelerometerRate(sampleRateHZ);

CurieIMU.setAccelerometerRange(4);

/\* Start the PME (Pattern Matching Engine) \*/

CuriePME.begin();

// I2C master sends category to I2C slave

Wire.begin();

learning();

}

void learning()

{

CuriePME.learn(vector\_forward0, 126, Forward);

CuriePME.learn(vector\_forward1, 126, Forward);

CuriePME.learn(vector\_forward2, 126, Forward);

CuriePME.learn(vector\_forward3, 126, Forward);

CuriePME.learn(vector\_reverse0, 126, Reverse);

CuriePME.learn(vector\_reverse1, 126, Reverse);

CuriePME.learn(vector\_reverse2, 126, Reverse);

CuriePME.learn(vector\_reverse3, 126, Reverse);

CuriePME.learn(vector\_stop0, 126, Stop);

CuriePME.learn(vector\_stop1, 126, Stop);

CuriePME.learn(vector\_stop2, 126, Stop);

CuriePME.learn(vector\_stop3, 126, Stop);

CuriePME.learn(vector\_leftspin0, 126, LeftSpin);

CuriePME.learn(vector\_leftspin1, 126, LeftSpin);

CuriePME.learn(vector\_leftspin2, 126, LeftSpin);

CuriePME.learn(vector\_leftspin3, 126, LeftSpin);

CuriePME.learn(vector\_rightspin0, 126, RightSpin);

CuriePME.learn(vector\_rightspin1, 126, RightSpin);

CuriePME.learn(vector\_rightspin2, 126, RightSpin);

CuriePME.learn(vector\_rightspin3, 126, RightSpin);

}

void learn\_gesture(enum ir\_command category)

{

Serial.println("====================================================");

Serial.println("I2C\_master: learning gesture " + String(category));

Wire.beginTransmission(4);

Wire.write(category);

Wire.endTransmission();

for (int idx=0; idx<4; idx++) {

// learn from sample 4 times

Serial.println(String(idx) + ": learning gesture " + String(category));

/\* Record IMU data while button is being held \*/

readFromIMU(vector\_imu);

CuriePME.learn(vector\_imu, 126, category);

print\_vector();

}

}

void print\_vector()

{

Serial.print("{");

for (int idx=0; idx<numSamples; idx++) {

if ((idx%18)==0) {

Serial.println();

}

Serial.print(vector\_imu[idx]);

Serial.print(",");

}

Serial.println("}");

}

void loop ()

{

Serial.println("====================================================");

Serial.println("Ready to roll...");

/\* Record IMU data while button is being held \*/

readFromIMU(vector\_imu);

unsigned int gesture = CuriePME.classify(vector\_imu, 126);

if (gesture == CuriePME.noMatch) {

Serial.println("Gesture = NoMatch!");

} else {

Serial.println("Gesture = " + String(gesture));

Serial.println("I2C\_master: gesture = " + String(gesture));

Wire.beginTransmission(4);

Wire.write(gesture);

Wire.endTransmission();

}

}

void readFromIMU(byte buf[])

{

byte byte\_imu[1000];

int raw\_imu[3];

unsigned int i = 0;

/\* Wait until button is pressed \*/

while (digitalRead(gesturePin) == LOW);

Serial.println("Recording motion... ");

digitalWrite(LED\_BUILTIN, HIGH);

/\* While button is being held... \*/

while (digitalRead(gesturePin) == HIGH) {

/\* Wait for new accelerometer data to be ready \*/

if (CurieIMU.dataReady()) {

/\* Read the new x, y & z values into the buffer \*/

CurieIMU.readAccelerometer(raw\_imu[0], raw\_imu[1], raw\_imu[2]);

/\* Map raw values to 0-255 \*/

byte\_imu[i] = (byte) map(raw\_imu[0], IMULow, IMUHigh, 0, 255);

byte\_imu[i + 1] = (byte) map(raw\_imu[1], IMULow, IMUHigh, 0, 255);

byte\_imu[i + 2] = (byte) map(raw\_imu[2], IMULow, IMUHigh, 0, 255);

i += 3;

/\* If the buffer doesn't have enough space for the

\* next x, y & z values, we're finished. \*/

if (i + 3 > 1000) {

break;

}

}

}

digitalWrite(LED\_BUILTIN, LOW);

Serial.println("Proccessing samples... ");

undersample(byte\_imu,i/3,buf);

}

byte getAverageSample(byte samples[], unsigned int num, unsigned int pos, unsigned int step)

{

unsigned int ret;

/\* This is the number of samples that will be used to create each

\* average sample; i.e. all the skipped samples before and after

\* the current sample \*/

unsigned int size = step \* 2;

/\* Don't do any averaging, if we are at the beginning or end

\* of the sample window \*/

if (pos < (step \* 3) || pos > (num \* 3) - (step \* 3)) {

ret = samples[pos];

} else {

ret = 0;

pos -= (step \* 3);

/\* Calculate the sum of 'step' samples before, and after,

\* the current sample \*/

for (unsigned int i = 0; i < size; ++i) {

ret += samples[pos - (3 \* i)];

}

ret /= size;

}

return (byte)ret;

}

void undersample(byte input[], int numSamples, byte output[])

{

/\* Number of processed samples (1 sample == byte\_imu x, y, z)

\* that can fit inside a neuron which has 128byte,

\* so 128byte/(3byte per sample) = 42sample.

\*/

const unsigned int samplesPerVector = (CuriePME.maxVectorSize / 3);

unsigned int oi = 0; /\* Current position in output sample buffer \*/

unsigned int ii = 0; /\* Current position in input sample buffer \*/

/\* No. of samples to skip for each iteration \*/

unsigned int step = numSamples / samplesPerVector;

for (unsigned int i = 0; i < samplesPerVector; ++i) {

for (unsigned int j = 0; j < 3; ++j) {

/\* Get an average sample for the current position

\* in the sample window \*/

output[oi + j] = getAverageSample(input, numSamples, ii + j, step);

}

/\* Skip 'step' samples \*/

ii += (step \* 3);

oi += 3;

}

}