Annexe 1

Code du module «Pedale»
distortionSensorsV2.ino

```
/***********************************************************************
***********   distortion Effect for the MovingTones pedal   **********
***********      made by the MovingTones team               **********
***********              website :)                         **********
* Licensed under a Creative Commons Attribution 3.0 Unported License *
**********************************************************************
***********  effect adapted from the distortion effect by  ***********
***********       www.electrosmash.com/pedalshield          **********
***********  Based on rcarduino.blogspot.com previous work  **********
***********************************************************************/

/* To make this code compile you will need the following libraries:
        * Adafruit_ST7735
        * Adafruit_GFX
        * SPI.h
        * EEPROM.h */
// Libraries
#include "EEPROM.h"                                 //
#include <Adafruit_GFX.h>                           // Core graphics library
#include <Adafruit_ST7735.h>                        // Hardware-specific library
#include <SPI.h>                                    // Used during debugging to show messages
on the Serial print

// Variables from the original effects' code
int in_ADC0, in_ADC1;                            // variables for 2 ADCs values (ADC0, ADC1)
int POT0, POT1, POT2, out_DAC0, out_DAC1;        // variables for 3 pots (ADC8, ADC9, ADC10)
const int LED = 3;                               // LED in pin 3
const int FOOTSWITCH = 7;                        // Footswitch in pin 7
const int TOGGLE = 2;                            // Toggle in pin 2
int upper_threshold, lower_threshold;            // Variable for the distortion effect


const int SAVE_BUTTON = 1;                      // pins of the arduino
const int TFT_CS      = 10;
const int TFT_RST     = 8;
const int TFT_DC      = 9;

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS,  TFT_DC, TFT_RST);  // instantiate the
screen

// New variables

const int STANDBY1_MODE = 0;                    // effects will only be applied when the
pedal's footswitch is on HIGH
const int BUTTON_MODE   = 1;                    // sequence of pressing produces: low-high-
low-high or high-low-high-low
const int STANDBY2_MODE = 2;                    // if initially footswitch is HIGH, we will go
to BUTTON_MODE
const int SENSOR_MODE   = 3;                    // if intiially footswitch is LOW, we will go
to STANDBY1_MODE

const int DEBOUNCE_DELAY = 50;                  // delay to be applied to debounce

const int MIN =     0;                          // range of the parameters p0, p1, p2 for the
effects' code
const int MAX =  4096;

const int MIN_SCREEN =   0;                      // length of the bar graph to be displayed on
the screen
const int MAX_SCREEN = 115;

const int MAX_POT   =  4096;                    // readings from the bottons will be scaled to
this range
```

```
const int MIN_POT   =     0;
const int LIMIT_POT = 4096;

const int MAX_SENSORS = 180;                // sensors send angles in this range throught
the XBees
const int MIN_SENSORS = 0;

int footswitch_detect;                      // variables to fascilitate the toggle
int footswitch_detect_last;
int footswitch_detect_previous;

int save_button_detect;                     // currently not used
int save_button_press_time;
boolean save_button_pressed;

int footswitch_mode = STANDBY1_MODE;        // instantiate footswitch_mode to
STANDBY1_MODE

int last_debounce_time = 0;                 // variables to fascilitate debouncing
int last_update_time = 0;

int p0 = 0, p1 = 0, p2 = 0;                 // parameters to be used by the code to apply
effects accordingly
int p0_old, p1_old, p2_old;
int MEMORYPOTMOD0 = 0, MEMORYPOTMOD1 = 0, MEMORYPOTMOD2 = 0;



// Variables needed for the comunication with the ArduinoNano who is sending information
in Serial Port

byte incomingByte;      //Bytes that are going to be used to add them and to see if there
is any header between they two.
byte lastByte;          // This will be just the byte before incomingByte


// Variables to assemble 16-bits ints from bytes received form the Arduino Nano through
the XBees
// !!! Note that int on an Arduino DUE is 32-bit but in many other smaller Arduinos it is
16-bit
// Thus to avoid negative-wrapping-around-to-positive, use int16_t for all int variables
in this program
int16_t currentInt;
int16_t int1;
byte byteArray[3];
int16_t angles[3];      // the three angles read by the sensor as received from XBees
int16_t mask;



void setup()
{

  //Configuration from the original effect from pedalShield

  //ADC Configuration
  ADC->ADC_MR |= 0x80;              // DAC in free running mode.
  ADC->ADC_CR = 2;                  // Starts ADC conversion.
  ADC->ADC_CHER = 0xFFFF;           // Enable all ADC channels

  //DAC Configuration
  analogWrite(DAC0, 0);             // Enables DAC0
  analogWrite(DAC1, 0);             // Enables DAC1
  // We are going to read from the FootSwitch to make our 3 modes.

  pinMode(FOOTSWITCH, INPUT);       // Enables to read from the footswitch
  // pinMode(SAVE_BUTTON, INPUT);
  pinMode(LED, OUTPUT);

  // record the state of the footswitch when turned on

  /*HERE WE SHOULD DETECT IF HI IS ALEREADY IN BYPASSMODE OR NOT. AND THEN START FROM HIS
ORIGINAL STATE
  ***************************************************************************************
***************/
  footswitch_mode = digitalRead(FOOTSWITCH);
  footswitch_detect_last = digitalRead(FOOTSWITCH);
  footswitch_detect_previous = footswitch_detect_last;

  //Screen (add description and comments)

  tft.initR(INITR_BLACKTAB);
  tft.fillScreen(ST7735_BLACK);

// Here we set the screen parameters: orientation, text size, color and text display
  tft.setRotation(-1);
  tft.setTextSize(2);
  tft.fillScreen(ST7735_WHITE);
  tft.setTextColor(ST7735_RED);
  tft.println("*Distortion*");
  tft.println("");
  tft.println("          P1");
  tft.println("");
  tft.println("          P2");
  tft.println("");
  tft.println("          P3");


  p0_old = p0;
  p1_old = p1;
  p2_old = p2;


  // Initialize the Serial ports for message transmissions
  Serial.begin(57600);                  //Initialize the port for comunication with the
PC to send messages useful for coding (Serial)
  Serial1.begin(57600);                 //Initialize the port for comunication with the
XBee (Serial1)

  // From XBee
  lastByte = (byte)0;     // We inicialize the lastBythe with 0 so it can make its first
comparison.

}

void loop()
{

  readFootSwitch();                                     //We read the FootWwitch State and we
see if it's pressed.
  updateScreen();                                       //We upload the screen information
  // readSaveButton();

  // From the original effect code
  // Read the ADCs
  while ((ADC->ADC_ISR & 0x1CC0) != 0x1CC0); // wait for ADC 0, 1, 8, 9, 10 conversion
complete.
  in_ADC0 = ADC->ADC_CDR[7];              // read data from ADC0
  in_ADC1 = ADC->ADC_CDR[6];              // read data from ADC1


  // POT0=ADC->ADC_CDR[10];                 // read data from ADC8
  // POT1=ADC->ADC_CDR[11];                 // read data from ADC9
  // POT2=ADC->ADC_CDR[12];                 // read data from ADC10


  // Changing the parameters values depending of the mode

  switch (footswitch_mode) {
    case STANDBY1_MODE:
    case STANDBY2_MODE:
      Serial.print("Standby Mode ");
      break;
    case BUTTON_MODE:
      Serial.print("Button Mode: ");
      readPotentiometer();
      break;

    case SENSOR_MODE:
      Serial.print("Sensor Mode: ");
      readSensor();
      break;

    default:
      Serial.print("Invalid state ");
  }


  // We display the parameters values on the computer screen
  Serial.print(p0);
  Serial.print("|");
  Serial.print(p1);
  Serial.print("|");
  Serial.print(p2);
  Serial.print("|");

  Serial.println("");

  // Code for the effect 'distortion' - distortion does not use parameter p1
  // Code from the original effect: This is the effect, taking the parameters p0, p1 and
p2
  upper_threshold = map(p0, 0, 4095, 4095, 2047);
  lower_threshold = map(p0, 0, 4095, 0000, 2047);

  if (in_ADC0 >= upper_threshold) in_ADC0 = upper_threshold;
  else if (in_ADC0 < lower_threshold)  in_ADC0 = lower_threshold;

  if (in_ADC1 >= upper_threshold) in_ADC1 = upper_threshold;
  else if (in_ADC1 < lower_threshold)  in_ADC1 = lower_threshold;

  //adjust the volume with POT2
  out_DAC0 = map(in_ADC0, 0, 4095, 1, p2);
  out_DAC1 = map(in_ADC1, 0, 4095, 1, p2);

  //Write the DACs
  dacc_set_channel_selection(DACC_INTERFACE, 0);          //select DAC channel 0
  dacc_write_conversion_data(DACC_INTERFACE, out_DAC0);   //write on DAC
  dacc_set_channel_selection(DACC_INTERFACE, 1);           //select DAC channel 1
  dacc_write_conversion_data(DACC_INTERFACE, out_DAC1);    //write on DAC
}



// reading the FOOTSWITCH on the pedalShield and implement a debouncing mechanism
void readFootSwitch() {
  footswitch_detect = digitalRead(FOOTSWITCH);

  if (footswitch_detect != footswitch_detect_last) {
    last_debounce_time = millis();
  }

  if (footswitch_detect != footswitch_detect_previous && millis() - last_debounce_time >
DEBOUNCE_DELAY) {
    footswitch_mode = (footswitch_mode + 1) % 4;
    footswitch_detect_previous = footswitch_detect;
  }

  footswitch_detect_last = footswitch_detect; // we detect changes in the FOOTSWITCH by
comparing these two variables
}


void readSaveButton() {
  save_button_detect = digitalRead(SAVE_BUTTON);

  if (save_button_detect == HIGH) {
    last_debounce_time = millis();

    if (save_button_pressed = true) {
      if (millis() - save_button_press_time > 3000) {
        EEPROM.write(0, p0);
        EEPROM.write(1, p1);
        EEPROM.write(2, p2);
      } else {
        p0 = EEPROM.read(0);
        p1 = EEPROM.read(1);
        p2 = EEPROM.read(2);
      }

      save_button_pressed = false;
    }
  }

  if (save_button_detect == LOW && millis() - last_debounce_time > DEBOUNCE_DELAY &&
save_button_pressed == false) {
    save_button_press_time = millis();
    save_button_pressed = true;
  }
}

void readPotentiometer() {

  int POTMOD0 = ADC->ADC_CDR[2];                           //reading first potentiometer
  POT0 = updatePot(POT0, MEMORYPOTMOD0, POTMOD0);          //update first parameter using
current reading of potentiometer
  p0 = POT0;                                               //update globally
  MEMORYPOTMOD0 = POTMOD0;                                 //now the potentiometer reading
becomes the 'previous' reading
  //Serial.println(p0);
  int POTMOD1 = ADC->ADC_CDR[1]; //read from pot1          //reading second potentiometer
  POT1 = updatePot(POT1, MEMORYPOTMOD1, POTMOD1);       //update second parameter using
current reading of potentiometer
  p1 = POT1;                                            //update globally
  MEMORYPOTMOD1 = POTMOD1;                              //now the potentiometer reading
becomes the 'previous' reading

  int POTMOD2 = ADC->ADC_CDR[0]; //read from pot2       //reading third potentiometer
  POT2 = updatePot(POT2, MEMORYPOTMOD2, POTMOD2);       //update third parameter using
current reading of potentiometer
  p2 = POT2;                                            //update globally
  MEMORYPOTMOD2 = POTMOD2;                              //now the potentiometer reading
becomes the 'previous' reading
}


int updatePot(int POT, int MEMORYPOTMOD, int POTMOD) {   //update the given parameter
  int VALUE = 0;                                         //Value to be added to previous
values of parameters
  if ((POTMOD - MEMORYPOTMOD) < -0.9 * MAX_POT) {        // case of more than one complete
turn in the positive direction
    VALUE = MAX_POT + POTMOD - MEMORYPOTMOD;

    if ((POT + VALUE) > LIMIT_POT) {
      POT = LIMIT_POT;
    }
    else {
      POT = POT + VALUE ;
    }
  }
  else if ((POTMOD - MEMORYPOTMOD) > 0.9 * MAX_POT) {   // case of more than one complete
turn in the negative direction
    VALUE = - (MAX_POT) + POTMOD - MEMORYPOTMOD;
    if ((POT + VALUE) < MIN_POT) {
      POT = MIN_POT;
    }
    else {
      POT = POT + VALUE ;
    }
  }
  else if ((POTMOD - MEMORYPOTMOD) > 0) {               //case of less than one complete
turn in the positive direction
    VALUE = POTMOD - MEMORYPOTMOD;
    if ((POT + VALUE) > LIMIT_POT) {
      POT = LIMIT_POT;
    }
    else {
      POT = POT + VALUE ;
    }
  }
  else if ((POTMOD - MEMORYPOTMOD) < 0) {               //case of less than one complete
turn in the negative direction
    VALUE = POTMOD - MEMORYPOTMOD;//+POTSENSOR
    if ((POT + VALUE) < MIN_POT) {
      POT = MIN_POT;
    }
    else {
      POT = POT + VALUE ;
    }
  }
  return POT;
}
```

```
// Read the 3 angles transmitted by the XBees if there is message available
// The 3 angles (6 bytes) are always sent together following an int header == -21846
void readSensor() {
  if (Serial1.available()) {

    boolean header = false;                       // before a valid header is detected,
header is set to false
    lastByte = Serial1.read();                    // read one byte

    while (Serial1.available() && !header) {
      incomingByte = Serial1.read();    // read another byte

      int1 = assembleInt(lastByte, incomingByte);  // each int is 16-bit, so we assemble
the incoming int by applying bit-wise operations on the two bytes

      // Serial.println("waiting"); // uncomment this line to see how long it takes to
clear the buffer

      if (int1 == -21846) {
        header = true; // a correct header is detected
      }
      else {
        lastByte = incomingByte; // if there is not any header we take our second byte as
our first byte, to then make a comparison with the following
      }
    }

    // if the header is detected, the next 6 bytes are the 3 values of the angles (each
int is 16-bit)
    readAngles();
  }
}

// assemble an int from the two bytes just read into byteArray
int16_t assembleInt(byte byte1, byte byte2) {

  currentInt = (int16_t)byte1 << 8;
  mask = 0xFF;
  mask = mask & (int16_t)byte2; // mask the first 8 bit to zero, in case the byte is sign-
extended

  currentInt = currentInt | mask;

  // the byte1*256 + byte2 before is wrong because byte2 had been sign-extended - have to
mask the first 8 bits
  //mask = 0xFF;
  //currentInt = byte1*256 + (byte2 & mask);
  return currentInt;
}

//  Read the angles from the next 6 bytes (2 bytes for each 16-bit int angle)
void readAngles() {

  // Expect 3 angles
  for (int i = 0; i < 3; i++) {

    /* We know that we should have a message in the buffer, (because we have reconized an
header)
    but, it can happen that the pedal tries to read the angles form the Serial before the
sensor
    manage to send him the angles (it can also be because of the data transmission speed
v/s
    the pedal's processor speed)
    so here we wait for the next message if it's necessary

    TODO: Use a timout!!*/

    while (!Serial1.available()) {
      Serial.println("----Im waiting111111-");
    }

    byteArray[0] = Serial1.read();                        // We read the fist byte


    while (!Serial1.available()) {                        // We wait again in case it is
necesary  TODO: Use a timout!!

      Serial.println("----Im waiting222222-");
    }

    byteArray[1] = Serial1.read();                        // We read the second byte
    angles[i] = assembleInt(byteArray[0], byteArray[1]);  // We construct the angle using
both bytes
  }

  /*Sometimes, (and I don't know yet why.. and this is a ToDo:try to prevent this from
another way)
    there are some messages that are eathier read badly or sended badly.. so we delete any
message with an
    incorrect angle*/

  if (angles[0] >= 0 && angles[0] <= MAX_SENSORS && angles[1] >= 0 && angles[1] <=
MAX_SENSORS && angles[2] >= 0 && angles[2] <= MAX_SENSORS) {

    // Writing the angles into the global variables
    p0 = angles[0] * 4096 / MAX_SENSORS;                //Here we scale the angle into our
effect scale. here is until 4096
    p1 = angles[1] * 4096 / MAX_SENSORS;
    p2 = angles[2] * 4096 / MAX_SENSORS;



  }
}

void updateScreen() {
  //  only update the screen every 0.1 seconds
  if (millis() - last_update_time > 100) {
    last_update_time = millis();

    //  if the new parameter is bigger than the old one, add a blue bar
    //  else add a white bar. this scheme is applied for every parameter.
    if (p0_old < p0) {
      tft.fillRoundRect(MIN_SCREEN, 32, map(p0, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 16, 0,
ST7735_BLUE);
    }
    if (p0_old > p0) {
      tft.fillRoundRect(map(p0, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 32, map(p0, MIN, MAX,
MAX_SCREEN, MIN_SCREEN), 16, 0, ST7735_WHITE);
    }

    p0_old = p0;


    if (p1_old < p1) {
      tft.fillRoundRect(MIN_SCREEN, 64, map(p1, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 16, 0,
ST7735_BLUE);
    }
    if (p1_old > p1) {
      tft.fillRoundRect(map(p1, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 64, map(p1, MIN, MAX,
MAX_SCREEN, MIN_SCREEN), 16, 0, ST7735_WHITE);
    }

    p1_old = p1;

    if (p2_old < p2) {
      tft.fillRoundRect(MIN_SCREEN, 96, map(p2, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 16, 0,
ST7735_BLUE);
    }
    if (p2_old > p2) {
      tft.fillRoundRect(map(p2, MIN, MAX, MIN_SCREEN, MAX_SCREEN), 96, map(p2, MIN, MAX,
MAX_SCREEN, MIN_SCREEN), 16, 0, ST7735_WHITE);
    }

    p2_old = p2;
  }
}
```