LungTidalSonic

```
/*
/ LungTidalSonic 4/2018, original LungTidal 4/2009
/ A program for graphing lung tidal data collected by an Arduino PIC connected to a sonic distance
/ sensor mounted to a spirometer. The Arduino places the data on the serial port for LungTidalSonic
to see.
*/
import processing.serial.*;
import processing.pdf.*;

//MovieMaker mm;  // Declare MovieMaker object

Serial port;
String dataStr; // input string from serial port
String myName;
int MARKER = 65; // the letter A used to decode the serial input sent by the Arduino
boolean printScrn;
float dataMin, dataMax;
float plotX1, plotY1;
float plotX2, plotY2;
float labelX, labelY;
float colP, colW;

float[] arysonicV, arysonicMax;
float tRng;
float litersPerDataVal;
float plotVertRange;
float spiroMinData;
float spiroMaxData;

int curTest; // the current trial
int qtyTests = 3; // number of automatic trials
float volumeInterval = 1.0; // used for graph
float volumeIntervalMinor = 0.1; //used for graph
float volAtLeastThis = 0.5; // current trail max value must exceed this before moving tonext trial

int breathColor = #5AA4D2;
int completedColor = #0106FC;
int spuriousColor = #FF0000;

PFont plotFont;

int butQuitLeft;
int butQuitTop;
int butRestartLeft;
int butRestartTop;
int butZeroLeft;
```

```
int butZeroTop;
int butPrintLeft;
int butPrintTop;
PushButton butQuit;
PushButton butRestart;
PushButton butZero;
PushButton butPrint;

CalTable calibrationTable; // calibration text file is in the Data folder

void setup(){
 size(720,605);
 // Create MovieMaker object with size, filename,
 // compression codec and quality, framerate
 // mm = new MovieMaker(this, width, height, "LungTidal.mov",
 // 56, MovieMaker.ANIMATION, MovieMaker.BEST);
 myName = "Your Name Here";
 calibrationTable = new CalTable("calibration_data.tsv");
 dataStr = "0";
 dataMin = 0;
 dataMax = 1023;
 plotVertRange = 6;
 float litersAtDataMax = 6.0;
 spiroMinData = 390;
 spiroMaxData = 82;
 litersPerDataVal = litersAtDataMax/dataMax;
 plotX1 = 120;
 plotX2 = width - 80;
 labelX = 50;
 plotY1 = 60;
 plotY2 = height -70;
 labelY = height - 25;

 plotFont = createFont("SansSerif",20);
 textFont(plotFont);
 colP = 0.15;
 colW = colP*(plotX2-plotX1);
 curTest = 0;
 arysonicV = new float[qtyTests];
 arysonicV[0] = 0;
 arysonicV[1] = 0;
 arysonicV[2] = 0;
 arysonicMax = new float[qtyTests];
 arysonicMax[0] = 0;
 arysonicMax[1] = 0;
 arysonicMax[2] = 0;
 sonicV = 0;
 tRng = qtyTests*2;
```

```
 smooth();
 setupSerial();

 butQuitLeft = int(plotX2)+8;
 butQuitTop = int(plotY1)-50;
 butRestartLeft = int(plotX2)+8;
 butRestartTop = int(plotY1);
 butZeroLeft = int(plotX2)+8;
 butZeroTop = int(plotY1)+35;
 butPrintLeft = int(plotX2)+8;
 butPrintTop = int(plotY1)+70;
 setupButtons();
 //PrintIt p = new PrintIt();

}

void draw(){
 background(224);
 fill(255);
 rectMode(CORNERS);
 noStroke();
 rect(plotX1,plotY1,plotX2,plotY2);
 drawTitle();
 drawAxisLabels();
 drawVolumeLabels();
 drawButtons();
 if (myArd != -1){
   drawDataArea();
   drawMaxDataLine();
   calcStatusValues();
   incrementTest();
   }else{
   drawNoArduino();
 }
}


void calcStatusValues(){
 float avgOfTests,maxOfTests,minOfTests;
 maxOfTests = 0;
 minOfTests = MAX_INT;
 avgOfTests = 0;
 for (int i = 0; i <= min(curTest,(qtyTests-1)); i ++){
   maxOfTests = max(arysonicMax[i],maxOfTests);
   minOfTests = min(arysonicMax[i],minOfTests);
   avgOfTests = (arysonicMax[i]+avgOfTests);
 }
 avgOfTests = avgOfTests/(min(curTest,(qtyTests-1))+1);
```

```
 drawStatusValues(maxOfTests, minOfTests, avgOfTests);
}

void incrementTest(){
 if (curTest < qtyTests){ // curTest will reach qtyTest
  if ((arysonicV[curTest]==dataMin) & (arysonicMax[curTest]> (volAtLeastThis+
arysonicV[curTest]))){
    curTest++;
  }
 }
}

void drawDataArea(){
 for (int colm = 0; colm <= curTest; colm ++){
  if ((colm == curTest) &(curTest < qtyTests)){ // an ongoing trial
    fill(breathColor);
    beginShape();
    float x = map((colP+colm),0,tRng,plotX1,plotX2);
    float y = map(arysonicV[colm],plotVertRange,0,plotY1,plotY2);
    float yy = min(y,plotY2);
    vertex(x,yy);
    vertex(x+colW,yy);
    vertex(x+colW,plotY2);
    vertex(x,plotY2);
    endShape(CLOSE);
    if(arysonicV[colm] <= arysonicMax[colm]-(30*litersPerDataVal)){
      textSize(10);
      textAlign(CENTER);
      text(nfc(arysonicV[colm],2),x+colW/2,yy-3);
    }
  }else{ // a finished trial
    fill(completedColor);
    if (colm < qtyTests){
      beginShape();
      float x = map((colP+colm),0,tRng,plotX1,plotX2);
      float y = map(arysonicMax[colm],plotVertRange,0,plotY1,plotY2);
      float yy = min(y,plotY2);
      vertex(x,yy);
      vertex(x+colW,yy);
      vertex(x+colW,plotY2);
      vertex(x,plotY2);
      endShape(CLOSE);
    }
  }
 }
}

void drawMaxDataLine(){
```

```
 textSize(15);
 stroke(0);
 for (int colm = 0; colm <= curTest; colm ++){
   if (colm < qtyTests){
     float x = map((colP+colm),0,tRng,plotX1,plotX2);
     float y = map(arysonicMax[colm],plotVertRange,0,plotY1,plotY2);
     float yy = y ; //max(y,0);
     fill(0);
     textAlign(CENTER);
     text(nfc(arysonicMax[colm],2),x+colW/2,yy-5);
     fill(#0200CB);
     if (colm == curTest ){
       strokeWeight(2);
     } else{
       strokeWeight(1);
     }
     line(x,yy,x+colW,yy);// draw max line
     line(x,plotY2,x,yy);
     line(x+colW,plotY2,x+colW,yy);
     fill(0);
     text("Trial "+ (colm+1),x+colW/2,plotY2+25);
   }
 }
}
```

Buttons

```
void setupButtons(){
 butQuit = new PushButton("Quit",30,60,butQuitLeft,butQuitTop,150,200);
 butRestart = new PushButton("Restart",30,60,butRestartLeft,butRestartTop,150,200);
 butPrint = new PushButton("Print",30,60,butPrintLeft,butPrintTop,150,200);

}

void drawButtons(){
 butQuit.display();
 butRestart.display();
 if (curTest >= qtyTests){ // curTest will reach qtyTest
   butPrint.display();
 }
}

/*
/ This class creates a button and handle the mouseover chanhe in look. It was not possible
/ to put the mousepressed within this class. It sort of worked but was slow and the mousepressed
/ event seemed to stay activated.
*/
```

```
class PushButton {
 String caption;
 int butHt ;
 int butWd ;
 int butLeft;
 int butTop ;
 int fcR;
 int fcP;
 PushButton(String _caption, int _butHt, int _butWd, int _butLeft, int _butTop, int _fcR, int _fcP){
   caption = _caption;
   butHt = _butHt;
   butWd = _butWd;
   butLeft = _butLeft;
   butTop = _butTop;
   fcR = _fcR;
   fcP = _fcP;
 }
 void display(){
   if (mouseOver()){
     fill(fcR+50);
     stroke(0, 255, 0);
   }
   else{
     fill(fcR);
     stroke(0,0,255);
   }
   rectMode(CORNERS);
   rect(butLeft,butTop,butLeft+butWd,butTop + butHt);
   fill(0);
   textFont(plotFont);
   textSize(10);
   textAlign(CENTER,CENTER);
   text(caption, butLeft + butWd/2, butTop + butHt/2);
 }
 boolean mouseOver(){
   if((mouseX > butLeft) & (mouseX < (butLeft+butWd))){
     if ((mouseY < (butTop+butHt)) & (mouseY > butTop)){
       return true;
     }
     else{
       return false;
     }
   }
   else{
     return false;
   }
 }
}
```

```
// This assumes all the buttons are at the same X
void mousePressed(){
 if((mouseX > butQuitLeft) & (mouseX < (butQuitLeft+60))){
   if ((mouseY < (butQuitTop+30)) & (mouseY > butQuitTop)){
     //mm.finish();
     quitProgram();
   }
   if ((mouseY < (butRestartTop+30)) & (mouseY > butRestartTop)){
     resetTesting();
   }
   if (curTest >= qtyTests){
     if ((mouseY < (butPrintTop+30)) & (mouseY > butPrintTop)){
       printTrials();
     }
   }
 }
}

void quitProgram(){
 exit();
}

void resetTesting(){
 sonicVrawPrev = Float.NaN;
 if (myArd != -1){
   text("Inializing and Restarting Trials",plotX1+ 20,plotY1 + 40);
   port.stop();
   for (int i = 0; i < qtyTests; i ++){
     arysonicV[i] = 0;
     arysonicMax[i] = 0;
   }
   curTest = 0;
   setupSerial();
 }
 else{
   exit();
 }
}
```

CalTable

```
int myCal;

class CalTable {
 float[][] data;
 int rowCount;

 CalTable() {
```

```
  data = new float[10][10];
}

CalTable(String filename) {
  println("Reading calibration table " + filename);
  String[] rows = loadStrings(filename);
  if (rows != null){
    data = new float[rows.length][];
    for (int i = 0; i < rows.length; i++) {
      if (trim(rows[i]).length() == 0) {
        continue; // skip empty rows
      }
      println(rows[i]);
      if (rows[i].startsWith("#")) {
        continue; // skip comment lines
      }
      // split the row on the tabs
      float[] pieces = float(split(rows[i], TAB));
      // copy to the table array
      data[rowCount] = pieces;
      rowCount++;
      // this could be done in one fell swoop via:
      //data[rowCount++] = split(rows[i], TAB);
    }
    // array is one row too large, resize the &apos;data&apos; array as necessary
    rowCount--;
    data = (float[][]) subset(data, 0, rowCount);
    println("Read " + rowCount + " rows of calibration pairs.");
  }
  else {
    myCal = -1;
    println("... Calibration table " +filename + " is not found!");
  }
}


int getRowCount() {
  return rowCount;
}

// return calibrated value
// 5/2018 modified from 2009 version to project beyond the
// maximum calibration value and account for raw sonic data
// being reversed.
float getCalValue(Float sensorVal) {
  float interP = 0;

  // handle outlying sensorVal first.
```

```
  // sensorVal is larger than the largest (smallest volume)
  if (sensorVal >= data[0][1]){
    return data[0][0];
  }

  // sensorVal is smaller than the smallest (largest volume)
  if (sensorVal <= data[rowCount-1][1]) {
    float delta = data[rowCount-2][1]-data[rowCount-1][1];
    float amt = (data[rowCount-1][1]-sensorVal)/delta;
    interP = lerp(data[rowCount-2][0],data[rowCount-1][0],amt);
    return interP;
  }

  // At this point sensorVal is within the table
  for (int i = rowCount -1; i > -1; i--) {
    if( data[i][1] > sensorVal) {
      float delta = data[i][1]-data[i+1][1];
      float amt = (data[i][1]-sensorVal)/delta;
      interP = lerp(data[i][0],data[i+1][0],amt);
      break;
    }
  }
  return interP;
 }

 //float getFloat(int rowIndex, int column) {
 // return data[rowIndex][column];
 //}

}
```

ExportApplicationInstructions

```
/*
As of Processing version 3:

Use File -> Export Application

Pick the platform Mac OS X and make sure to select Embed Java.

Processing will create a folder named "application.macosx". In that
folder will be the exported application.

There will also be a folder called "source". It will contain copies
of the source files and a copy of the java file Processing created.
This can be confusing. That folder and its contents are not needed.
It can be deleted.

*/
```

GraphComposition

```
void drawTitle(){
 fill(0);
 textFont(plotFont);
 textSize(20);
 textAlign(LEFT);
 String title = "Lung Tidal Volume";
 if (curTest < qtyTests){
   title = title + " - Starting Trial " + str(curTest+1) + " of " + str(qtyTests);
 }else{
   title = title + " - Finished";
 }
 text(title,plotX1,plotY1 - 25);
}

void drawAxisLabels(){
 fill(0);
 textFont(plotFont);
 textSize(13);
 textLeading(15);
 textAlign(CENTER,CENTER);
 text("Spirometer\nVolume\nIn\nLiters",labelX,(plotY1+plotY2)/2);
}

void drawVolumeLabels(){
 fill(0);
 textFont(plotFont);
 textSize(10);
 stroke(128);
 strokeWeight(1);
 float magnify = 10; // required due to a problem with the modulo function
 for (float v = dataMin; v <=plotVertRange*magnify; v+= volumeIntervalMinor*magnify){
   float y =map(v,dataMin,plotVertRange,plotY2,plotY1);
   if (v % volumeInterval ==0) { // if a major tick
     if(v==dataMin){
       textAlign(RIGHT); //align by the bottom
     }
     else if(v==dataMax){
       textAlign(RIGHT,TOP); //align by the top
     }
     else{
       textAlign(RIGHT,CENTER);//center vertically
     }
     text(nfc(v,1),plotX1-10,y);
     stroke(0);
     line(plotX1-4,y,plotX1,y);// draw major tick
```

```
      stroke(192);
      line(plotX1,y,plotX2,y);// draw major line
      stroke(128);
    }
    else{
      line(plotX1-2,y,plotX1,y);
    }
  }
}

void drawStatusValues(float maxOfTests, float minOfTests, float avgOfTests){
 fill(0);
 textFont(plotFont);
 textSize(20);
 textAlign(RIGHT,BASELINE);
 String maxT = nfc(maxOfTests,2);
 String minT = nfc(minOfTests,2);
 String avgT = nfc(avgOfTests,2);
 String curP = nfc(sonicV,2);
 if (mouseOnMyName()) {
   fill(255,0,0);
 }else{
   fill(0);
 }

 text(myName, plotX2-5, plotY1+25);
 fill(0);
 text("Maximum: "+ maxT + " L", plotX2-5, plotY1+50);
 text("Minimum: "+ minT + " L", plotX2-5, plotY1+75);
 text("Average: "+ avgT + " L", plotX2-5, plotY1+100);

 textFont(plotFont);
 textSize(15);
 textLeading(18);
 text(timeStamp(),plotX2-5,plotY1+125);

 textAlign(RIGHT,BASELINE);
 text("Spirometer Reading: "+ curP, plotX2, plotY2+25);
 text("Raw Reading: "+ dataStr, plotX2, plotY2+40);
 if (dataStr ==null){
   drawNeedToReset();
 }
 if (myCal == -1){
   drawNoCalibrationTable();
 }
}

boolean mouseOnMyName(){
```

```
if((mouseX >plotX2-200) & (mouseX < plotX2-5)){
   if ((mouseY < ( plotY1+25)) & (mouseY > plotY1)){
     return true;
   }else{
     return false;
   }
}else{
  return false;
}
}


String timeStamp(){
 String amPm;
 String d = String.valueOf(day()); // Values from 1 - 31
 String m = String.valueOf(month()); // Values from 1 - 12
 String y = String.valueOf(year()); // 2003, 2004, 2005, etc.
 int hr = hour();
 String h;
 if (hr > 12){
   h = String.valueOf(hour()-12);
   amPm = "pm";
 }else{
   h = String.valueOf(hour());
   amPm = "am";
 }
 String mn = String.valueOf(minute());
 String s = String.valueOf(second());
 return h+":"+mn+":"+s + " " + amPm + "\n" +m+"/"+d+"/"+y;
}

void drawNoArduino(){
 fill(#FF0004);
 textFont(plotFont);
 textSize(20);
 textLeading(28);
 textAlign(LEFT,BASELINE);
 String msg = arduinoUSB + "\nUSB driver NOT detected.\nEither the Arduino device is not
connected\nor the device driver is not loaded.";
 msg = msg + "\nClose this program. Fix the problem.\nRestart the program.";
 msg = msg + "\n";
 msg = msg + "\nUSB drivers that were detected:";
 String[] devList;
 devList = Serial.list();
 for (int i= 0; i < devList.length;i++){
   String devN = devList[i];
   msg = msg + "\n" + devN;
 }
```

```
 text(msg, width*7/24-40,height/2-200);
}

void drawNeedToReset(){
 fill(#FF0004);
 textFont(plotFont);
 textSize(24);
 textLeading(28);
 textAlign(LEFT,BASELINE);
 String msg = "Reset the spirometer to the bottom.\n\nPress the Restart button.\n\nWait until there is a non-null raw reading.";
 text(msg, width*0.2,height/2-40);
}


void drawNoCalibrationTable(){
 fill(#FF0004);
 textFont(plotFont);
 textSize(15);
 textLeading(18);
 textAlign(LEFT,BASELINE);
 String msg = "The calibration table seems to be missing. It belongs\nin a folder named \"Data\" one level deeper than this\napplicaton.";
 msg = msg + "\n\nThe calibration table contains pairs of dial readings and\ncorresponding raw data values at every 0.5 L increments";
 msg = msg + "\nin Tab separated, plain text file format. This program\nuses that file to interpolate raw data values between";
 msg = msg + "\nthe pairs intervals. The spirometer readings displayed\nbelow are not calibrated.\n";
 msg = msg + "\nRecord the spirometer dial readings and corresponding\ndisplayed raw readings. Create the calibration file\nand then restart the program.";
 text(msg, width*15/48-10,height/2-60);
}
```

NameEntry

```
void keyPressed() {
 if (mouseOnMyName()){
   if ((keyCode >=32) && (keyCode <= 126)) {
     myName = myName + key;
   }
   else {
     if (keyCode == 8){
       if (myName.length()>0){
         myName = myName.substring(0, myName.length()-1); // strip off the last char
       }
     }
   }
 }
```

```
}
```

Printing

```
void printTrials(){
  println("printing");
  String myDir = whereAmI();
  println(myDir);
  String myPDFJob = myDir +"/LungTidalPrintJob.pdf";
  println(myPDFJob);
  beginRecord(PDF, myPDFJob);
  background(224);
  fill(255);
  rectMode(CORNERS);
  noStroke();
  rect(plotX1,plotY1,plotX2,plotY2);
  drawTitle();
  drawAxisLabels();
  drawVolumeLabels();
  drawButtons();
  if (myArd != -1){
    drawDataArea();
    drawMaxDataLine();
    calcStatusValues();
    incrementTest();
  }
  else{
    drawNoArduino();
  }
  endRecord();
  try{
    Runtime.getRuntime().exec("open -a Preview " + myPDFJob);
  }
  catch (IOException e){
    e.printStackTrace();
  }
 }

String whereAmI(){
 String myExecPath;
 myExecPath = System.getProperty("user.dir");
 String myPath = myExecPath.replace("\\", "/");
 return myPath;
}
```

Serial

```
String arduinoUSB = "/dev/tty.usbserial-A7006Tac";
//String arduinoUSB = "/dev/tty.usbmodem1411";
```

```
int myArd;
float sonicV;
float sonicVraw;
float sonicVrawPrev = Float.NaN;
float sonicVrawMaxDelta = 180.0;
float sonicVrawDelta = Float.NaN;
float MAX_DISTANCE_PRAC = 400;
// Sensor readings above this value are ignored so that jitter at the
// down position does not read as data.
float zeroCutOff = 380;


void setupSerial(){
 println("Available serial ports:");
 println((Object[])Serial.list());
 // Use the port corresponding to your Arduino board. The last
 // parameter (e.g. 19200) is the speed of the communication. It
 // has to correspond to the value passed to Serial.begin() in your
 // Arduino sketch.
 println("Looking for " +arduinoUSB);
 myArd = check4SerialDevice(arduinoUSB);
 //println(myArd + " check4");
 if (myArd != -1){
   println("Found " +arduinoUSB + " at " +myArd);
   port = new Serial(this, Serial.list()[myArd], 19200);
   port.bufferUntil(MARKER);
 }
 else{
   println("Did not find Arduino USB driver " + arduinoUSB);
 }
}

void serialEvent(Serial port) {
 dataStr = (port.readStringUntil(MARKER));
 if (dataStr.length() >=2){
   dataStr = dataStr.substring(0, dataStr.length()-1); // strip off the last char
   sonicVraw = float(dataStr);

   // detect and ignore zinger values
   if (sonicVraw < MAX_DISTANCE_PRAC){
         sonicVrawDelta = sonicVraw - sonicVrawPrev;
         // println(sonicVraw + " <= now | prev => " + sonicVrawPrev );
         if ( sonicVrawDelta > sonicVrawMaxDelta){
           // println(sonicVrawDelta + " Ingorning " + sonicVraw );
           return;
         }
         // println(sonicVrawDelta + " is current delta");
   } else {
```

```
    // println("Spurious serial data. " + sonicVraw );
    return;
   }
   sonicVrawPrev = sonicVraw;
   if (myCal != -1){ // normal condition
      sonicV = calibrationTable.getCalValue(sonicVraw);
      if (sonicVraw > zeroCutOff){
        sonicV = 0;
      }
   }
   else { // abnormal, calibration table not found
      sonicV = map(sonicVraw,spiroMinData,spiroMaxData,0,6);
      sonicV = max(0,sonicV);
      if (sonicVraw > zeroCutOff){
        sonicV = 0;
      }
   }

   if ((dataStr != null) & (curTest < qtyTests)) { // ie stop after last test
    arysonicV[curTest] = sonicV;
    if(arysonicV[curTest] > arysonicMax[curTest]){
      arysonicMax[curTest] = arysonicV[curTest];
    }
   }
  }
}

int check4SerialDevice(String devName){
 String[] devList;
 devList = Serial.list();
 for (int i= 0; i < devList.length;i++){
   String devN = devList[i];
   if (devN.equals(devName) == true){
     return i;
   }
 }
 return -1;
}
```

Z_TheAssociatedArduinoCode

```
   ///*
   // The Arduino side for the processing LungTidal. This version uses
   // the HC-SR04 sonic distance sensor along with with the DHT11 Temp
   // and Humidity sensor for correction. Revised 5/2018 from the original
   // LungTidal that read a potentiometer.

   // Arduino code to send HC-SR04 distance reads out to the LungTidal
   // processing program
```

```
  // You can use the Arduino serial monitor to view the sent data, or it can
  // be read by Processing.

  // The LungTidal Processing code (sonic version) graphs the data received
  // so you can see the value of the sonic distance reads changing over time.
  //*/
  //// DHT Libraries from Adafruit
  //// Dependant upon Adafruit_Sensors Library
  //#include "DHT.h";
  //// NewPing Library for HC-SR04
  //#include "NewPing.h"

  //// Constants
  //#define DHTPIN 7 // DHT-11 Output pin number
  //#define DHTTYPE DHT11 // DHT Type is DHT 11
  //#define TRIGGER_PIN 12 // DHT trigger pin number
  //#define ECHO_PIN 11 // DHT echo pin number

  //#define RDLEDPIN 13 // LED connected to digital pin 13, distance reading
  //#define DHTRDLEDPIN 8 // LED connected to digital pin 8, dht read

  //// distance units will be mm in this sketch so that the integer
  //// value can be sent with enough resolution
  //#define MAX_DISTANCE 440 // Sent to sensor library, reports 0 beyond this
  //#define MAX_DISTANCE_PRAC 400 // Practical distance in distance units
  //#define MIN_DISTANCE 2 // Practical minimum distance in distance units
  //#define SAMPLE_DELAY 80 // Main loop sample delay
  //#define SENSOR_INVERVAL_FACTOR 16 // Any loop counter multiple of this triggers
temp/hum read.
  //#define ITERATIONS 5 // Number of reads per averaging

  //// Note: SAMPLE_DELAY * SENSOR_INVERVAL_FACTOR = time between temp/hum
checks.
  //// This needs to be at least 500 ms. Reported to be 2000 ms required.
  //// This time interval allows the DHT-11 sensor can stabilize.

  //// Define Variables

  ////int value = LOW; // Previous value of the LED
  //int readValue = 0; // Variable to hold the read value
  //int lastValue = 0; // Previous readValue
  //int readDelta = 2; // Read threshold difference for reporting
  //float hum = 55; // Humidity value in percent, default in case of dht error
  //float temp = 23; // Temperature value in Celsius, default in case of dht error
  //float duration; // HC-SR04 pulse read duration value
  //float distance; // Calculated distance in cm
  //int counter = 0; // Cycle counter
```

```
//// Initialize sensors
//NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
//DHT dht(DHTPIN, DHTTYPE);

//void setup() {
// Serial.begin(19200);
// pinMode(RDLEDPIN, OUTPUT); // sets the digital pin as output
// dht.begin();
// GetTempHum();
// // Serial.println(""); // for debuging
// // Serial.println("Starting"); // for debuging
//}

//void loop() {

// counter ++;
// if (counter % SENSOR_INVERVAL_FACTOR == 0){
// GetTempHum();
// // Serial.println("Temp/Hum Just Read " ); // for debuging
// // Serial.println( hum); // for debuging
// // Serial.println( temp); // for debuging
// counter = 0;
// blinkLed();
// }

// // read vlaue on distance sensor
// // but return smaller of the reading or MAX_DISTANCE_PRAC
// readValue = fmin(distread(),MAX_DISTANCE_PRAC);
// // if dist has changed more than delta
// if (abs(readValue-lastValue) > readDelta) {
// Serial.print(readValue);
// Serial.print("A");
// lastValue = readValue;
// digitalWrite(RDLEDPIN, HIGH);
// }

// delay(SAMPLE_DELAY);
// digitalWrite(RDLEDPIN, LOW);
//}

//int distread(){
// float soundsp; // Stores calculated speed of sound in M/S
// float soundmm; // Stores calculated speed of sound in mm/ms
// // Calculate the Speed of Sound in M/S
// soundsp = 331.4 + (0.606 * temp) + (0.0124 * hum);
// // Convert to mm/ms
// soundmm = soundsp / 1000;
```

```
// duration = sonar.ping_median(ITERATIONS);
// // Calculate the distance in mm as integer
// return (int)(duration / 2) * soundmm;
//}

//void GetTempHum(){
// hum = dht.readHumidity(); // Get Humidity value
// temp= dht.readTemperature(); // Get Temperature value
// if (isnan(hum)) {
// hum = 40;
// }
// if (isnan(temp)) {
// temp = 23;
// }

//}

//void blinkLed(){
// digitalWrite(DHTRDLEDPIN, HIGH);
// delay(SAMPLE_DELAY);
// digitalWrite(DHTRDLEDPIN, LOW);
//}
```