ELEC 291 Section 20C

LAB #5: INTERNET - ENABLED RANGE FINDING

(ARDUINO SEES THE WORLD AROUND IT LIKE A BAT AND TWEETS ABOUT IT)

LAB SECTION: L2D

TEAM #: D\_5C

|  |  |  |
| --- | --- | --- |
| *Student name* | *Student number* | *Contribution percentage* |
| LINDA MUNISI | 22535158 | 33.3% |
| SIRINE TRIGUI | 46443157 | 33.3% |
| YIXIN ZHAO | 49427140 | 33.3% |

*Contribution summary*:

We worked together in this lab. During the lab session, three of us implemented the Arduino code and tested it. In our two meetings, we all worked on Processing and Twitter.

We discussed together on implementation of taking and separating the signals. Yixin implemented the twitter function. We then all fixed the processing to receive two values and plot the graph, then worked on implementing the GUI and the pointer used to display the distance on the radar-like graph. All of us then participated in improving our code and demoed it to our TA.

**A. Introduction and motivations**

The objective of this lab is to give Arduino a bat-like eye to sense its surrounding using an ultrasonic range finder. The information from the sensor is sent through the Internet using Twitter. We considered the fact that Arduino can communicate via the internet through the computer when it is connected to it. That was the simplest method to have internet-enabled Arduino. The components used in this lab are;

* Ultrasonic range finder module HC-SR04
* Temperature sensor LM-335

The Ultrasonic range sensor is used to measure the distance from an object by measuring the time taken from a transmitted signal through the trig pin until it is reflected then sent back to be received by the echo pin . For the timing of the sensor, it is told to start to transmit or measure by a trigger signal. The sensor may be triggered by a 10 microsecond pulse on its Trig input pin. When triggered, it transmits a burst of 8 ultrasonic pulses and then it releases its echo output pin. The width of the echo signal is then used to measure the distance. A new signal can be sent only once the echo fades; this is called the cycle period and according to the datasheet should not be less than 50ms. We were also supposed to be aware of the range of angles for which the sensor has best performance. These ranges were from approximately 15 degrees (to be accurate) both sides.

Following the above, we had to use a combination of Processing, Twitter java library and twitter to make our Arduino tweets.

**B. Lab Description**

**Connection:**

We began our lab with connecting the circuit. First we connected the Ultrasonic range sensor. The left most pin was connected to the ground and right most pins was connected to the VCC pin. The trig pin is connected to pin 8 of the Arduino while the echo pin is connected to pin 6 of the Arduino. For the temperature sensor, the middle pin of the temperature sensor is connected to the resistor on one side and the analog pin on the other side. The last pin of the temperature sensor is connected to the ground. The adj pin of the temperature sensor is not connected (kindly note that we are using LM335 temp sensor instead of LM-35).

**Using Arduino**

We wrote the code for enabling the temperature sensor and the range sensor in Arduino. The code written was easy since it is the same as the code from the previous labs. We initialized communication with the serial monitor and initialized the pinMode pins of the Arduino. The echo and temperature pins were inputs while the trig pin was output. The trig pin was triggered by setting it to high and then setting it to low after a delay of 2 milliseconds (10 us are required). Since the value that results from the temperature sensor is in volts, we had to convert the value to read it in Kelvin. The LM-335 temperature sensor is different from the LM-35 in a sense that the temperature values are read in Kelvins and hence we had to convert the values to Celsius. This was done in the following line;

*float tempc = tempk - 273.15;*

Then the temperature obtained was used to calculate the duration and hence the distance of the object placed in front of the range sensor. The line of code used to calculate the distance is;

*float distance = duration / (speedS \* 2);* here the speed used being the speed of sound(since it is the one associated with the echo signal). We added an if statement to test the conditions when the distance measured is out of range. We set the if statement to display -1 as distance if the distance measured is out of range.

Testing of this part included, checking if the values that are displayed for temperature and distance are correct. For the temperature, the values displayed at first was wrong, we got a temperature that was ranging from 100 degrees and above which was wrong since that was not close to the values of the temperature of the room. After trying different connections of the temperature sensor, we realized that the sensor was burnt out and hence we had to get a new sensor. That’s when we got the LM-335 in place for the LM-35. It is slightly different with the connections and the values read as explained above. After changing the temperature sensor we got the correct values. We tested the range sensor by putting an object at a distance that was approximately equal to a sheet of A4 paper. The resulting distance was approximately 28 which was correct and corresponding to the length of the paper. We also tried to measure the distance of objects that were very far and we got -1 since the object becomes out of bound as the limit set was 100cm.

**Using Processing**

We exploited our skills from lab4 to draw a “normal” graphs displaying the temperature and the measured distances (snapshot is provided below: Graph for Temperature and Distance vs Time).In the graph, we used the same procedure as the previous lab to display the graph. We used *import processing.serial.\** to initialize the communication between Processing and the serial monitor*;* then we initialized the serial port and opened the serial port that we were using. There were other parameters as the horizontal position of the graph, the size of the window that we set in the *setup()*. The line, *myPort.bufferUntil('\n'),* indicated generated a serial event every after getting a new line with the code since the characters received were put in the buffer, and the program read from the port and refresh the buffer after every “\n”.Then we set our initial background color to white for a better view since we were using different colors.

In the draw function, first it drew x and y axes using the line function that specifies the start point and end point of a line. We also set the color of the line using the *stroke()* function. The loop:

*if (xPos >= 1200) {*

*xPos = 100;*

*background(255);*

*} else {*

*xPos++;}*

makes the graph run only in correspondence with our x axis that we drew. Once the graph reaches the end of the axis, it goes to the beginning of x axis; otherwise it increases the x so that the graph moves. We also called *drawStuff()* in the draw to put numbers besides our axes. This function was modified from an online resource.

The *serialEvent()* function reads data sent from Arduino through the port as String. After research and discussion, we decided to use array *input* to store the data that Processing received. The program receives three numbers each time it reads from *myPort*, which are distance and temperature, in order. We separated those data by “,”. In this way, the program gives us two independent graphs.

After finished the basic functions for plotting, we added the buttons to switch between different graphs, and all the texts to make the plot more user-friendly. We used the Button example provided by the Processing and modified it according to our code. We set all their parameters at the beginning and used *stroke()*, *rect()* to draw them as rectangles. The *update()* reads the coordinates of the mouse, and uses *overT() and overD()* to check which button is selected. The *mousePressed()* detects the action of clicking mouse. Combining *update()* and *mousePressed()* together, we can check which button is pressed. After the button is pressed, the corresponding flag is set as 1, then the *serialEvent()* updated the current graph with the corresponding data. We set the button functions in the following way:

Left click on it once, graphs the corresponding data;

Right click on it, stops reading new data and stable at the last data it receives;

Left click on another button.

The three buttons have different colors, for instance, the color for light level is green. When the user chooses the print light level, the title and the labels of x and y axes changes alone with it. Also, the units are printed in different color, so the unit for light level is in green, which makes it easier for the user to read.

The *text()* function takes the content of a text and the coordinates, then displays it at the chosen coordinates. The function *textSize(32);* is used to set the size of the text that we want by specifying the size in parenthesis, example 32. Then we used *pushMatrix()* and *popMatrix()* functions to rotate the text while labelling the y-axis. We first set the location where we want the text to begin, and then put the text is the stack(push in stack). After that we make a translation beginning from the starting point and rotate it in the desired direction using the *rotate()* function. We then use the text function to state the text that we want to be stated and rotated and lastly, pop the matrix. Below is the code that labels the y-axis with text that is rotated;

*int x = -450; // x- Location of start of text.*

*int y = 500; //y- Location of start of text.*

*pushMatrix();*

*translate(x, y);*

*rotate(-HALF\_PI);*

*text("Temp/Hum/Light", x, y);*

*popMatrix();*

On testing the graphs by variating the values of temperature and distance, the values graph displayed was corresponding to the correct values(that are also printed in the console of processing) and hence everything was working fine.

**Internet enabled Arduino**

Using an Internet-enabled computer that is attached to the Arduino: This is the simplest way of connecting an Arduino to the Internet and it is the approach used in this lab. A program that is running on the computer communicates with Arduino via the serial port; whenever Arduino needs Internet access, the program on the computer deals with it. Arduino was connecting to Twitter through the internet.

In order to add Twitter support to Processing, we had to install a Twitter java library. We downloaded and installed the Twitter4J library. After unzipping the Twitter4J zip file, we found the twitter4j-core-4.0.4.jar file under the lib folder then we dragged it to our processing sketch so that a “code” subfolder has been created for us containing the file.

We followed the instructions on Connect to set the app for Twitter. In that app website, we created an application and set its name, description, and the website. Since none of us had a website, we put a placeholder there in URL form. Then we checked our API Key, API Secret, Access Token Secret and Access Token. Then we began to write the code for twitter, we imported all the twitter configurations files that we were going to use in processing. We also needed to import the java.util package as we will make use of the List and Date classes when dealing with tweets we retrieve from Twitter and so we imported it using the line of code *import java.util;* Testing this included running the code with setup and draw functions in processing and check whether any error shows up. While testing this, there was no error which means twitter library was properly configured in our sketch folder. By referring to Codasign for instructions, we created our twitter application as follows:

* First we created an instance of the Twitter Object, the instance was named twitter and the line of code is *Twitter twitter;* then we configured our application with the API configuration keys. These are the “Consumer Key”, The “Consumer Secret”, The “Access Token” and The “Access Token Secret”. These were created in the beginning when we first created the application. Then we proceeded with authenticating our code with twitter,
* To authenticate ourselves with Twitter we created a ConfigurationBuilder object and initialise it with our application keys. In the setup() function, a new ConfigurationBuilder instance is created as follows: *ConfigurationBuilder cb = new ConfigurationBuilder();* Then after that, we initialised the object with our application keys example for the consumer Key we used the line of code;

*cb.setOAuthConsumerKey("consumerKey");*

* Then we created a TwitterFactory object and passed it our ConfigurationBuilder object:

*TwitterFactory tf = new TwitterFactory(cb.build());*

Finally we initialized our Twitter object by retrieving an instance from the twitter Factory

*twitter = tf.getInstance();*

Since proper programming practice is testing the code each step before proceeding so we tested this part first. We ran the code above in processing and despite the fact that we did not see any results since we did not have complete code, we did not get any errors we proceeded to assume that so far the code we had was correct.

Then we had a tweet method that contains the message to be tweeted. In that message we make an attempt to update our status on twitter and print the message on the console if it works. We had a try- catch block that handles the exception in the case that we fail to update our status in twitter instead of freezing up the computer. We used a twitter exception to catch the error and print a statement indicating an error on the console, using the following line of code;

*catch(TwitterException te){*

*System.out.println("Error: " + te.getMessage());*

After that, we were able to control out twitter with the button that controls the temperature graph since we combined the twitter code in processing and the twitter code. We tested the code by testing the sensor at different distances to see whether it would tweet. Not exactly what we expected, we got only one tweet after the object got into the range. And after that, we can’t get any new tweets even if we plugged it and run all the code again (we got few but it was very random). We searched the error information online. Some said we need to update our Twitter4j but we were using the latest version. The useful one said that Twitter does not take same message several times so we needed to make all of them distinguishable from one another. During our demo, we encountered another problem: our program could not tweet as it did the night before. Turned out there is a limit of how many tweets one can send per day.

**Designing Radar GUI**

In addition with our GUI from the last lab we designed another GUI with radar screen to display the distance in this lab. At first, we designed it using *arc()* function. However, the arcs did not look good and we did not figure out how to draw the arc as curves. Then we started researching online. We chose one which contains sweep function and good labels from several other projects(included in the references). The reason was although the others also had radar screen, in this lab we only needed a simple one because we did not have a servo to control the angle. Besides, we did not need to display any shape since we were not using any angles. Therefore, the one with better display screen, good labels, and without any complex function of displaying became our first choice.

We kept the original size and the basic lines and deleted everything about degree as well as sweep. Though we thought about keeping the sweep function, it did nothing but made the GUI fancier. Reading data from the port was easy since we did it in the last lab. The red circle on the screen displays the distance and we changed the scale so it suited our sensor range. The program sent a tweet whenever the object is closer than 15 cm to it. The twitter function is the same as above.

We learned that the line (*ellipse(radius, radius, 750, 750);* ) inside the *draw()* not only sketches a circle, but it draws it every time the *draw()* function is called, which means it refreshes the screen.Also, we figured how to enlarge the scale to make the graph more readable. The biggest problem we encountered was that we needed to adjust some parameters many times in order to make a better looking graph.

**C. Conclusions**

The wiring was kind of troublesome since we did not know we burned out our LM35 sensor since we connected it the opposite way(pin to ground was connected to VCC and VCC to ground). We kept getting wrong data. The TAs helped us and we replaced it with LM335, which fixed the problem. The Radar GUI was not that hard after we figured out what the original code means. However it was time consuming to make a good display. We spent a lot of time on Twitter function. First of all, the better way to use Twitter4j is to put the core file into your building path. Moreover, there are two things we learnt about Twitter:

1. Twitter does not take duplicate messages; so we created a variable “count”, which counts how many time the *draw()* is called, and sent it with our message to make each one different from the previous one.
2. There is a daily limit in Twitter. One can only make a certain number of tweets per day. If the limit is reached, the messages can still be sent but not every one of them.

After this lab we realized how powerful Processing is and it was a lot of fun to see the Arduino working with the software, with the internet.

**D. References**

- The following are the references that were used in this report;

Arduino Reference for the functions:<http://arduino.cc/en/Reference/HomePage>

Temperature sensor IC datasheet: [LM35\_datasheet.pdf](https://connect.ubc.ca/bbcswebdav/pid-3136547-dt-content-rid-14227028_1/courses/SIS.UBC.ELEC.291.20C.2015W2.59753/ELEC291_15W2/L5/291_L5_LabInfo_images/LM35_datasheet.pdf)

HC-SR04 datasheet 1: [HC\_SR04\_1.pdf](https://connect.ubc.ca/bbcswebdav/pid-3136547-dt-content-rid-14227028_1/courses/SIS.UBC.ELEC.291.20C.2015W2.59753/ELEC291_15W2/L5/291_L5_LabInfo_images/HC_SR04_1.pdf)

HC-SR04 datasheet 2: [HC-SR04\_Manual.pdf](https://connect.ubc.ca/bbcswebdav/pid-3136547-dt-content-rid-14227028_1/courses/SIS.UBC.ELEC.291.20C.2015W2.59753/ELEC291_15W2/L5/291_L5_LabInfo_images/HC-SR04_Manual.pdf)

Processing download website:<https://processing.org/download/>

Processing Tutorial:<https://processing.org/tutorials/>

Twitter:<https://twitter.com/?lang=en>

Twitter Application Management website: <https://apps.twitter.com/>

Twitter4J:[http://twitter4j.org/en/index.html](http://twitter4j.org/en/index.htmlHow)

How to set up Twitter Java library in processing:

<https://youtu.be/gwS6irtGK-c?list=PLiHuW09DwTen50Jj_fjwHnmNIlXKDSitn>

Setting up GUI using controlP5: <http://www.sojamo.de/libraries/controlP5/>

How to fix the 403 error in processing console:

<https://twittercommunity.com/t/status-is-a-duplicate/16928>

Source of the sensor part for fritzing: <http://fritzing.org/projects/hc-sr04-project>

Learning how to authenticate and configure twitter:

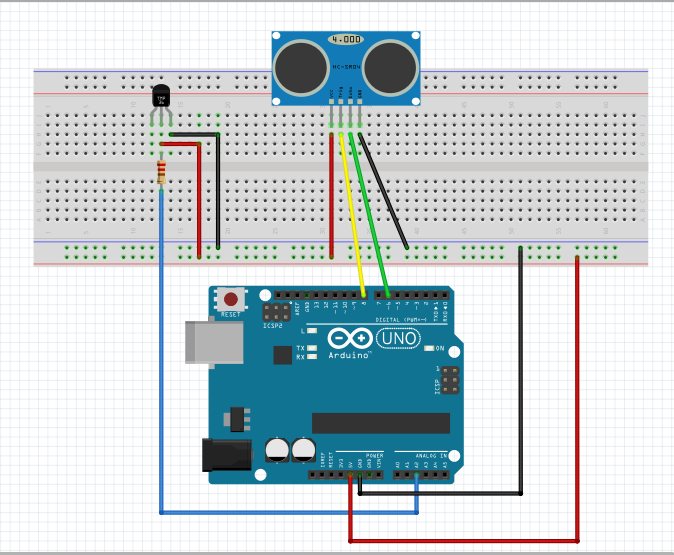
[http://codasign.com/tutorials/processing-and-twitter/searching-twitter-for-tweets/](http://fritzing.org/projects/hc-sr04-project)

Radar Screen

<http://www.instructables.com/id/How-to-make-a-simple-ultrasonic-radar-system-/>

**Appendix I**

- The fritzing schematic of the circuit is shown below;



**Appendix II**

Arduino code:

# define trig 8 // trig(sender) pin is assigned to pin 8 of the Arduino

# define echo 6 // echo(receiver) pin is assigned to pin 6 of the Arduino

int temperature = A2; //temperature sensor is linked to analog pin A2

float duration = 0; //a variable to calculate the distance

float vofTemp = 1; //initializing the value that we get from the pin assigned to the temperature sensor to 1

void setup() {

pinMode(trig, OUTPUT); //set the trig as output in the Arduino to get the signal

pinMode(echo, INPUT); //set the echo as input to Arduino

pinMode(temperature, INPUT); //set temperature sensor as input to `Arduino

Serial.begin(9600); //initialize communication with the serial monitor

}

void loop() {

duration = pulseIn(echo, HIGH); // PulseIn returns the length of the pulse in microseconds

delay(500);

vofTemp = analogRead(temperature); // getting the voltage from the temperature sensor pin

float tempk = ((5.0 \* vofTemp \* 100.0) / 1024); // convert the voltage to temperature in Kelvin

float tempc = tempk - 273.15; // get the equivalent temperature in Celcius

delay(2);

float S = 331.5 + (0.6 \* tempc); // get the Sound

float speedS = 1 / ( S \* 0.0001); // get the speedSound

float distance = duration / (speedS \* 2); // getting the distance

Serial.print(tempc); //printing the temperature on the serial monitor in celcius

Serial.print(","); // to seperate the two values

if(distance >= 100){ //setting the distance range u to 100

Serial.println("-1");// if out of range, -1 will be printed

}

else{

Serial.println(distance); // printing the measured distance

}

delay(100); //delay before starting the next loop

}

**Appendix III**

Processing Code:

import processing.serial.\*; //initializing communication with the serial monitor

import twitter4j.conf.\*;

import twitter4j.\*;

import twitter4j.auth.\*;

import twitter4j.api.\*;

import java.util.\*;

Twitter twitter;

Serial myPort; // initialize The serial port

int xPos = 100; // set the horizontal position of the graph

float inByte = 0; //initialize the value used to hold the byte from the serial monitor

float inputs[]; //store the data read from myPort in order to distribute them later

//create flags for which button is pressed and set the loop in serialEvent()

int flagT, flagD, count = 0;

int tX, tY, dX, dY; // Position of temp and distance button

// Position of distance button

int tSize = 50;

int dSize = 50; //size of the temperature and distance-controlling button

//change the title and labels color according to which data is printed(temp/ distance)

color current;

color tColor, dColor, tHighlight, dHighlight; //setting up different colors for each variable

//initialize the state of the buttons. All not presssed

boolean tOver = false;

boolean dOver = false;

void setup() {

//set application keys and create new instance and initialise ir with applicatio keys

ConfigurationBuilder cb = new ConfigurationBuilder();

cb.setOAuthConsumerKey("oDrf60UwSZqHX3zrkVHRrOKFZ");

cb.setOAuthConsumerSecret("8XMXCJF54UkQVDYCAON3ucEOTCKklZ9WcIjKsS0ISfKW5T9Qq4");

cb.setOAuthAccessToken("4865065038-Lp8riluWFcnF4nsOgkbYVDfWnjygpRLynAtd69s");

cb.setOAuthAccessTokenSecret("sKUk76croih3NblzTapb2OgyPJb24JKBLKnGapF7fSSUg");

//create a TwitterFactory object and pass it our ConfigurationBuilder

twitter = new TwitterFactory(cb.build()).getInstance();

size(1500, 1350); //set the size of the display screen

println((Object)Serial.list()); //enabling printing of the values on the console window

//initialize serial communication and the port being used

myPort = new Serial(this, Serial.list()[0], 9600);

// generate a new serialEvent() when gets a newline character:

myPort.bufferUntil('\n');

// set initial background:

background(255);

//set up the different colors for each button to distinguish them

tColor = color(150, 100, 200);

dColor = color(50, 200, 100);

//set the position of the Temperature button

tX = width - tSize - 10;

tY = height - tSize / 2 - 100;

//set the position of the distance button

dX = width - dSize \* 2 - 30;

dY = height - dSize / 2 - 100;

}

//The function that enables drawing

void draw() {

drawStuff();// draw the coordinate system, numbers and the letters on the buttons

update(mouseX, mouseY); //check the position of the mouse

stroke(0); //setting the outline to black

rect(tX, tY, tSize, tSize);//draw the button as a tSize \* tSize rectangle at (tX, tY) position

stroke(0);

rect(dX, dY, dSize, dSize);

textSize(32); //size of the title of the graph

//fill the following text with the color of the pressed button(also the same color of the graph)

fill(current);

//the title and its position

text(" D-5C Graph of Temperature and Distance Against time", 400, 100 );

//VERTICAL AXIS

textSize(32);

int x = -450; // Location of start of text.

int y = 500;

pushMatrix(); //rotating the text to make it vertical

translate(x, y);

rotate(-HALF\_PI);

//y-axis

text("Temp/Distance", x, y);

popMatrix();

//HORIZONTAL AXIS

text("Time/s", width - 200, height - 200 );

// draw the graph;

stroke(current);// use the same color of the pressed button(and the text above)

line(xPos, (height - 200), xPos, (height - 200 - inByte)); //position to start the graph

//draw the coordinates

stroke(255, 0, 0); //set the background color

line(xPos, (height - 200), width - 200, (height - 200)); //horizontal line drawing

line(100, (height - 200), 100, 150); //vertical line drawing

//label buttons accordingly

text("T", tX + 13, tY + 35);

text("D", dX + 13, dY + 35);

// at the edge of the screen, go back to the beginning of the x axis

if (xPos >= 1200) {

xPos = 100;

background(255);

} else {

// increment the horizontal position:

xPos++;

}

}

//if the mouse is on one of the button, set the “over” flag of this button as true, the others’ as false

void update(int x, int y) {

if ( overT(tX, tY, tSize, tSize) ) {

tOver = true; // the mouse is over temperature button

dOver = false; //set “over” flag of humidity and light as false

} else if ( overD(dX, dY, dSize, dSize) ) {

dOver = true; // the mouse is over humidity button

tOver = false; //set “over” flag of the other two as false

}

}

void serialEvent (Serial myPort) {

// get the ASCII string:

String inString1 = myPort.readStringUntil('\n'); //serial port reads until end of new line

if (inString1 != null) {

// trim off any whitespace:

inString1 = trim(inString1);

// convert to an float and map to the screen height:

inputs = float(split(inString1, ','));//separate the String with “,” to read different data

//the temperature button is pressed so that temperature values can keep looping until stopped

if (flagT == 1) {

println("temperature"); //print temperature on the console

if (inputs.length == 2) { //three values read from the serial port

inByte = inputs[0]; //the first value of the array is temperature value

float tempc = inByte; //receive the temperature value

println(tempc);

tempc = map(tempc, 0, 800, 0, height); //graph the temperature values

}

}

//the distance button is pressed so that light values can keep looping until stopped

if ( flagD == 1) {

println("distance");

if (inputs.length == 2) {

inByte = inputs[1]; //the third value of the array is distance value

float distance = inByte;

println(distance);

distance = map(distance, 0, 800, 0, height); //graph the distance values

}

}

//tweet a tweeter if the object is closer than 15 cm

if(inputs[1] <= 15)

tweet();

count ++;

}

}

//check which button is pressed and set the print flag to 1, change the current color,

//LEFT CLICK STARTS PLOTTING, RIGHT CLICK STOPS UPDATING THE GRAPH

void mousePressed() {

if (tOver) {

flagT = 1;

current = tColor;

if (mouseButton == RIGHT)

flagT = 0;

}

if (dOver) {

flagD = 1;

current = dColor;

if (mouseButton == RIGHT)

flagD = 0;

}

}

//check if the mouse is over the temperature button

boolean overT(int x, int y, int width, int height) {

if (mouseX >= x && mouseX <= x + width &&

mouseY >= y && mouseY <= y + height) {

return true;

} else {

return false;

}

}

//check if the mouse is over the distance button

boolean overD(int x, int y, int width, int height) {

if (mouseX >= x && mouseX <= x + width &&

mouseY >= y && mouseY <= y + height) {

return true;

} else {

return false;

}

}

//FUNCTION THAT SETS THE LABELS OF THE AXES AND THEIR POSITIONS

void drawStuff() {

textSize(17);

fill(tColor);//using the color of temperature button

text("Temperature\*C", 90, 75);

fill(dColor);//using the color of humidity button

text("Distance/cm", 90, 95);

//print the numbers besides of axes in black

fill(50);

textSize(11);

//x axis

for (int i = 0; i <= 1200; i += 50) {

text(i , 100 + i, height - 185);

stroke(300);

}

for (int j = 0; j < 100; j += 50) {

text(j, 80, height - 200 - j);

stroke(300);

}

}

//attempt to update our status on Twitter and print a message to the console if it works

void tweet( ){

try{

Status status = twitter.updateStatus("Found item close to me" + count);

System.out.println("Status updated to [" + status.getText() + "].");

}

//catch any problem and print the error message

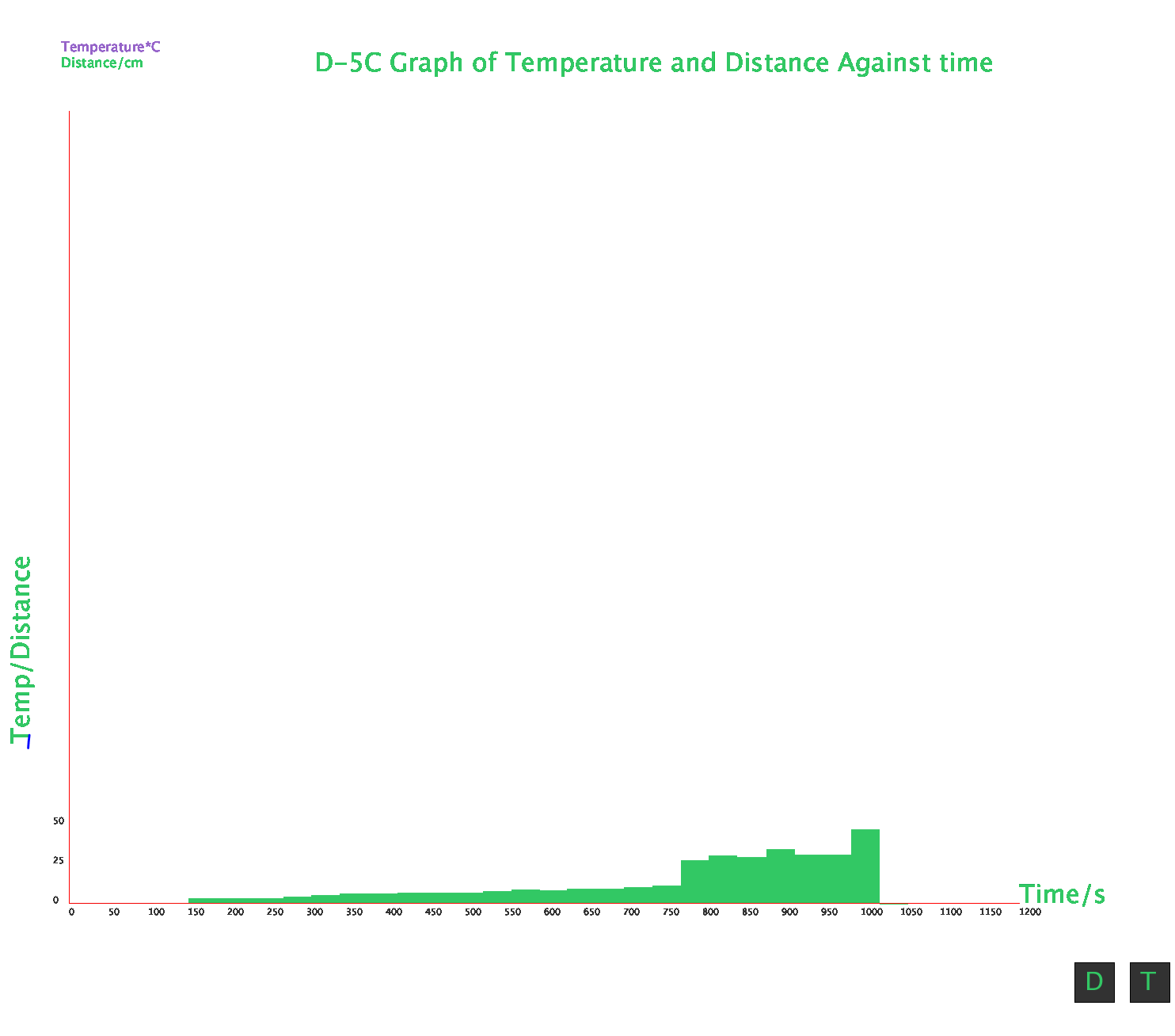
catch(TwitterException te){

System.out.println("Error: " + te.getMessage());

}

}

Screenshot for the graph display for distance in processing:



**Appendix IV**

GUI Code (radar):

import processing.serial.\*; // import serial library

import twitter4j.conf.\*;

import twitter4j.\*;

import twitter4j.auth.\*;

import twitter4j.api.\*;

import java.util.\*;

Twitter twitter;

Serial myPort;

int radius = 350; // set the radius of objects

int w = 300; // set an arbitary width value

int radarDist, count = 0;

float value ; // value from sensor

PFont myFont; // setup fonts in Processing

float input[];

/\* create background and serial buffer \*/

void setup() {

//create new instance and initialise ir with applicatio keys

ConfigurationBuilder cb = new ConfigurationBuilder();

cb.setOAuthConsumerKey( "consumerkey");

cb.setOAuthConsumerSecret("consumersecret");

cb.setOAuthAccessToken( "4865065038-accesstoken");

cb.setOAuthAccessTokenSecret( "accesstokensecret");

//create a TwitterFactory object and pass it our ConfigurationBuilder

twitter = new TwitterFactory(cb.build()).getInstance();

// setup the background size, colour and font.

size(750, 450);

background (0); // 0 = black

myFont = createFont("verdana", 12);

textFont(myFont);

// setup the serial port and buffer

myPort = new Serial(this, Serial.list()[0], 9600);

}

/\* draw the screen \*/

void draw() {

fill(0); // set the following shapes to be black

noStroke(); // set the following shapes to have no outline

ellipse(radius, radius, 750, 750); // draw a circle with a width/ height = 750 with its center position (x and y) set by the radius

rectMode(CENTER); // set the following rectangle to be drawn around its center

rect(350, 402, 800, 100); // draw rectangle (x, y, width, height)

/\* set the radar distance rings and out put their values, 50, 100, 150 etc.. \*/

for (int i = 0; i <= 6; i++) {

noFill();

strokeWeight(1);

stroke(0, 255 - (30 \* i), 0);

ellipse(radius, radius, (100 \* i), (100 \* i));

fill(0, 100, 0);

noStroke();

text(Integer.toString((radarDist / 50 + 1) \* 20), 380, (305 - radarDist), 50, 50);

radarDist += 50;

}

radarDist = 0;

/\* draw the grid lines on the radar every 30 degrees and write their values 180, 210, 240 etc.. \*/

for (int i = 0; i <= 6; i++) {

strokeWeight(1);

stroke(0, 55, 0);

line(radius, radius, radius + cos(radians(180 + (30 \* i)))\*w, radius + sin(radians(180 + (30 \* i)))\*w);

fill(0, 55, 0);

noStroke();

if (180 + (30 \* i) >= 300) {

text(Integer.toString(180 + (30 \* i)), (radius + 10) + cos(radians(180 + (30 \* i))) \* (w + 10), (radius + 10) + sin(radians(180 + (30 \* i))) \* (w + 10), 25, 50);

} else {

text(Integer.toString(180 + (30 \* i)), radius + cos(radians(180 + (30 \* i)))\*w, radius + sin(radians(180 + (30 \* i)))\*w, 60, 40);

}

}

/\* Write information text and values. \*/

noStroke();

fill(0);

rect(350, 402, 800, 100);

if (value == -1) {

fill(0, 100, 0);

text("Distance: OUT OF RANGE! ", 120, 400, 200, 50);

}

else {

//tweet a tweeter if the object is closer than 15 cm

if (value <= 15.0)

tweet();

count ++;

fill(150, 0, 0);

ellipse(350, (350 - value) - value / 20 \* 30, 10, 10);

fill(0, 100, 0);

text("Distance: " + Float.toString(value), 120, 400, 200, 50); // text(string, x, y, width, height)

}

fill(0);

rect(70, 60, 150, 100);

fill(0, 100, 0);

text("Screen Key:", 100, 50, 150, 50);

noFill();

stroke(150, 0, 0);

strokeWeight(1);

ellipse(29, 53, 10, 10);

fill(150, 0, 0);

text(" D-5C Position", 115, 70, 150, 50);

println(value);

}

/\* get values from serial port \*/

void serialEvent (Serial myPort) {

String inString = myPort.readStringUntil('\n'); // read the serial port until a new line

if (inString != null) { // if theres data in between the new lines

inString = trim(inString); // get rid of any whitespace just in case

input = float(split(inString, ','));//separate the String with “,” to read different data

if (input.length == 2) {

value = input[1];

}

}

}

//attempt to update our status on Twitter and print a message to the console if it works

void tweet( ) {

try {

Status status = twitter.updateStatus("Too close " + count);

System.out.println("Status updated to [" + status.getText() + "].");

}

//catch any problem and print the error message

catch (TwitterException te) {

System.out.println("Error: " + te.getMessage());

}

}

Screenshot for the radar Screen:

