**ELEC 291 Section 20C**

Lab 1

L2C

Team 2A

*Student name Student number Contribution percentage*

Andy Ruan 36863141 33.3%

Kevin Wong 32105132 33.3%

Clarence Su 36387132 33.3%

Contribution Summary:

Andy Ruan wrote the Processing code to display the sensor data in the form of a GUI resembling a RADAR.

Kevin Wong wrote Processing code to send sensor data to Twitter. He also organized the lab report.

Clarence Su wired the Arduino and wrote Arduino code to collect the data from the LM35DZ sensor and HC-SR04 sensor.

**Introduction and motivations**

The main objective of this lab is to use the HC-SR04 sensor in order to detect objects two meters away, to display the object’s location on a user-friendly GUI and sending a tweet with data from the sensor using the Twitter API.

This report outlines the process in which Team 2A used to complete lab 5 of ELEC 291-20C. It is organized in the following manner: First, it will describe the lab and its sub-components. Then the report will conclude lessons learned from the lab. In addition, three appendices are included providing a Fritzing breadboard schematic of the circuit used and all the code used during the lab respectively.

**Lab Description**

Lab 5 comprised of three parts:

1. Implementing Arduino code to collect data from HC-SR04 sensor
2. Creating GUI on Processing
3. Adding Twitter component to send tweets

These four components resulted in a complete circuit, which incorporated the Arduino UNO, the HC-SR04 sensor, and the LM35 sensor. The circuit and the Arduino code will allow users to detect objects within a 200 centimeter range of the HC-SR04 sensor and send the data to Processing where the object’s location will be displayed on a GUI and its distance will be sent as a tweet on Twitter.

**Implementing Arduino code to collect data from HC-SR04 sensor**

Using the information from the datasheet of HC-SR04, we initialized the pin connected to Tri terminal as output, and the pin connected to the Echo terminal as input. The data reading cycle starts with triggering the sonar by sending out a pulse, after it senses the echo pulse and returns the time the pulse travels. With the travel time, we can calculate the distance of the object, which reflects the pulse from the sonar, using the distance equation d=vt. Since the time returned by the sonar is the time traveled in a complete cycle, we divided the time by 2. Additionally, sound travels at a different speed in different temperatures. Therefore, our team used a LM35 sensor to record the real time temperature of the environment and use this measurement to calibrate the speed of sound to get a more precise measurement. In the data sheet of HC-SR04, it states that the range of sonar is 4 meters. We decided to cap the distance up to 2 meters, considering the range of the lab and the fact that having a smaller range could emphasize the change of distance on our radar GUI. Therefore, before sending data to Processing, our Arduino program will check if the data exceed the predetermined range and replace the data with zero (the GUI will display out of range for zero) if it exceeds the range of 2 meters. However, we have the radar range as a global variable so that our team could easily change one line of code in order to change the range.

**Creating GUI on Processing**

Using drawing methods similar to those of lab 4, our team was able to implement a functional RADAR-eque GUI to display the ultrasonic sensor data. We encountered some difficulty trying to figure out the mathematical formulas to draw each line from origin to the edge of the radar in combination with the axis system that Processing uses. Before interfacing with the serial, we tested with a random number generator while manipulating the variable for “currentWedge” to ensure the correct data was being displayed at the correct point. Most implementation decisions involved graphical design. We were aiming for an authentic radar, so we minimized the amount of different colours used and used a standard font. This also meant that we wanted to fade the scanning line as it moved in addition to display and fade red dots as the scanning past by. The speed of fading took some tinkering as that would also affect how often our displayed positional data updated.

While the real time sensor data was indeed being read by the serial input, to keep the radar authentic, we decided to only update the displayed distance once every scan rotation and display a red dot at this distance when the scanning line past over the correct scanning angle. We realized later that this radar GUI is better suited to a setup with an automated method of scanning such as having the ultrasonic on a motor which we did not have time to set up. The team did not want to throw away the work on the radar design however as we wanted to use the opportunity to work with moving and fading lines. An alternative design decision may have been to implement a vertical line scanning across a rectangular radar screen. This, like our current implementation would be suited to a specific setup involving the ultrasonic on a rail moving back and forth. As a compromise, our team decided to manually pivot the ultrasonic and implement buttons on the GUI to switch the sensor angle on the radar to simulate a wider range of effectiveness (180 degrees in front). The selected angle would be highlighted by a thicker line on the radar. Ideally we would have wanted multiple ultrasonics facing each one of these angles so that we could display red dots at multiple angles during one scanning rotation.

Using an upscaled TNR font, we also displayed the numerical value of the temperature and distance sensors on the right of the radar. Keeping to the theme of authenticism, we decided to used abbreviated labels and only update the distance at the same time that a red dot appear instead of in real time. A small bug we encountered and fixed here was that we did not take into account that the last data point would continued to be displayed if something were to move out of range. We also considered displaying the humidity as that is another factor that affects the speed of sound, however we did not have time to implement the calculations for this. Finally after some debate, the team decided to display the direction of the selected angle in terms of clock position instead of some arbitrary angle that required a reference point to be labeled.

A small note that we will take into account for next time is that a window size of 1000 by 1000 was very large on screens and small on other screens due to different team member’s personal laptop resolutions. As the Processing code for the radar was written by the team member with a large resolution, viewing work became difficult for the smaller resolutions. We will make sure to settle on a window size that everyone can work with next time.

**Adding Twitter comp**o**nent to send tweets**

To be able to send tweets to Twitter, the team needed to import the Twitter4j library to Processing and create a Twitter account. To enable developer settings, the team needed to input a mobile phone number for their Twitter account. The team then created a new Twitter application on the Twitter Developers; the new application will create application keys and access tokens for the Processing code to use in order to communicate with the Twitter API. Next, the team implemented a button that would tweet the object’s current location when pressed. The button simply calls a function that uses a twitter object created with application keys and access tokens to update twitter status with a message when pressed.

A problem the team had with Twitter was the tweet would not send. After debugging, the team realized Twitter’s restriction on the number of characters was causing this problem. By fixing the length of our tweet, the problem was solved.

**Conclusion**

This lab allowed students to improve their skills in Processing by creating a user-friendly GUI to display an object’s location up to 200 cm from the HC-SR04 sensor. Additionally, students learned to use external APIs such as Twitter to share and broadcast the data onto the Internet. Learning to use the Twitter API allows students to learn other APIs such as Facebook, Amazon, Youtube much easier.

Some important lessons the team learned from this lab was that writing modular code allowed the team to divide the tasks much easier and also allowed the team to work more efficiently. As a result, the team completed the lab much faster than the previous labs. Furthermore, the team learned that by writing modular code, the application was much easier to scale. For instance, when combining the Twitter button code to the radar GUI code in Processing, the team was less prone to bugs because the two codes were modular and did not interfere with each other’s functionalities.

**Reference**

[1] http://www.flaticon.com/free-icon/twitter\_23681#term=twitter&page=1&position=4

[2] http://www.flaticon.com/free-icon/left-arrow\_62069#term=left&page=1&position=6

**Appendix I**

Below is the Fritzing diagram used for Lab 5:

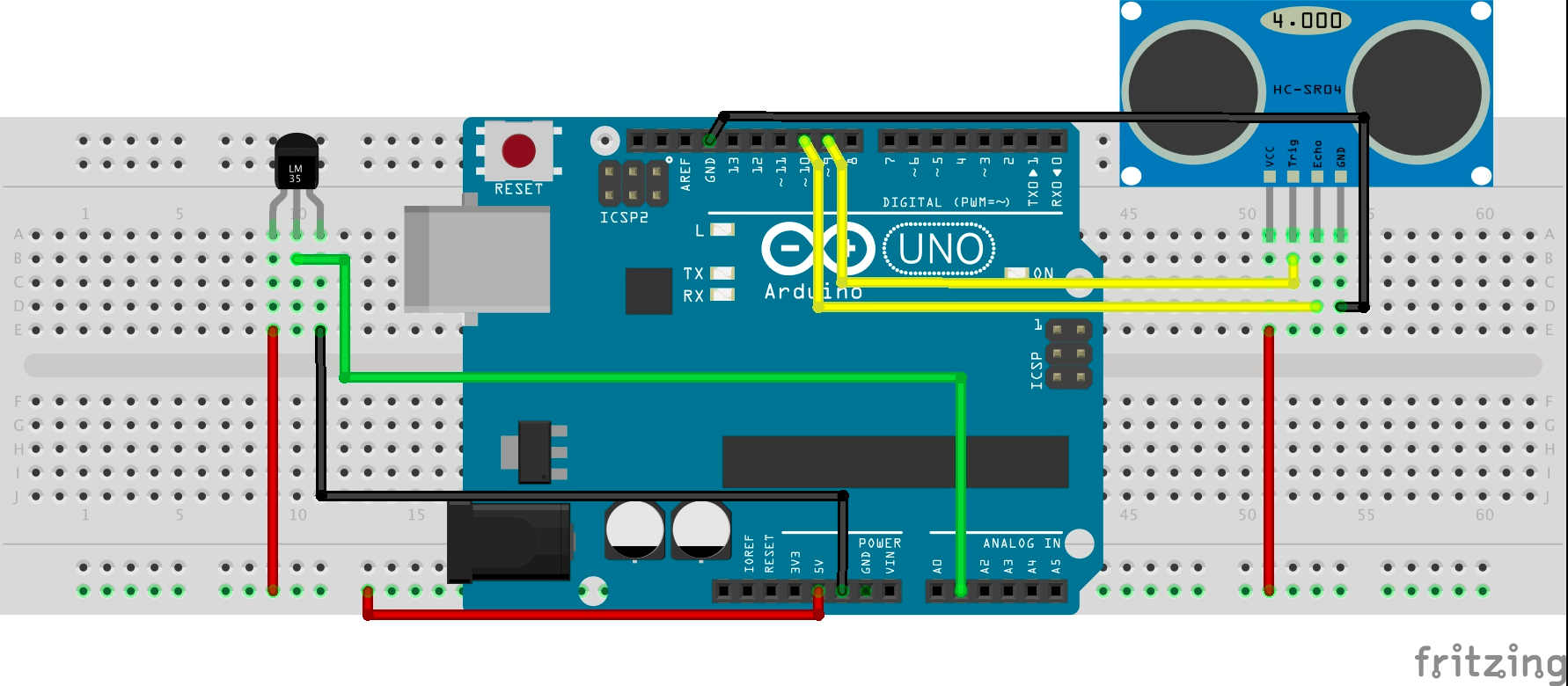


Figure 1. Fritzing Diagram

**Appendix II**

Below is the Arduino code used for Lab 5:

#define ultraTri 9

#define ultraEcho 10

#define maxRange 200 //unit cm

const int LM35 = A1;

const float soundSpeedinZero = 331;

void setup() {

Serial.begin(9600);

pinMode(LM35, INPUT);

pinMode(ultraTri,OUTPUT);

}

void loop() {

delay(1000);

// The sensor is triggered by a HIGH pulse of 10 or more microseconds.

// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

digitalWrite(ultraTri, LOW);

delayMicroseconds(5);

digitalWrite(ultraTri, HIGH);

delayMicroseconds(10);

digitalWrite(ultraTri, LOW);

// Read the signal from the sensor: a HIGH pulse whose

// duration is the time (in microseconds) from the sending

// of the ping to the reception of its echo off of an object.

pinMode(ultraEcho, INPUT);

float ultraSonReading = pulseIn(ultraEcho, HIGH);

float LM35reading = analogRead(LM35);

float temperature = calibrateLM35(LM35reading);

float distance = calibrateDistance(ultraSonReading,temperature);

if(distance <= 200.0){

Serial.print(distance);

Serial.print(",");

}else {

Serial.print(0.0);

Serial.print(",");

}

Serial.print(temperature);

Serial.print(",");

Serial.print(0.0);

Serial.print(",");

Serial.print("\n");

}

//calibrate the raw analog value from LM35

float calibrateLM35(float rawValue){

return (5.0 \* rawValue \* 100.0) / 1024;

}

//calibrate the distance with the temperature

float calibrateDistance(float travelTime, float temperature){

float soundSpeed = 331 + 0.6\* temperature; //sound speed in the particular temperature

float timeInSec = travelTime / 1000000.0; //travelTime in second

return timeInSec \* soundSpeed / 2.0 \* 100; //distance = time \* speed divide the value by 2 to find the one trip distance

}

**Appendix III**

Below is the Processing code used for Lab 5:

import processing.serial.\*;

import twitter4j.conf.\*;

import twitter4j.\*;

import twitter4j.auth.\*;

import twitter4j.api.\*;

import java.util.\*;

// button images

PImage leftButton;

PImage rightButton;

PImage twitterButton;

Twitter twitter;

// button positions

int leftButtonX=950, leftButtonY=800; // Position of left button

int rightButtonX=1100, rightButtonY=800;

int twitterButtonX=1250, twitterButtonY=800;

// button variables for indicating if the button selected

boolean twitterButtonOver = false;

boolean leftButtonOver = false;

boolean rightButtonOver = false;

int buttonWidth = 80, buttonHeight = 80;

// radar variables

final float scanSpeed = PI/70; // speed of scanning, lower => faster

Serial myPort;

PFont font;

float scanAngle = 0; // current angle of the scanning line

int currentWedge = 3; // abstraction for programmer to decide which angle ultrasonic is facing

float wedge5Mem = 0; // 2 o'clock edge case

// real time data

float realTimeItemDistance = 0;

float realTimeTemperature = 0;

// displayed data

float itemDistance = 0; // used to calculate position of red dot

float dispDist; // used to display the position of object as red dot is shown

void setup() {

size(1400, 1000);

smooth();

font = loadFont("TimesNewRomanPSMT-12.vlw");

//Declaring Serial port to run

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

myPort.bufferUntil('\n');

background(0);

//Load images to buttons

leftButton = loadImage("leftArrow.png");

leftButton.resize(100, 80);

rightButton = loadImage("rightArrow.png");

rightButton.resize(100, 80);

twitterButton = loadImage("twitter.png");

twitterButton.resize(100, 80);

size(1400, 1000);

//Input Tokens for Twitter API

ConfigurationBuilder cb = new ConfigurationBuilder();

cb.setOAuthConsumerKey("/\*ADD CONSUMER KEY\*/");

cb.setOAuthConsumerSecret("/\*ADD CONSUMER SECRET\*/");

cb.setOAuthAccessToken("/\*ADD ACCESS TOKEN\*/");

cb.setOAuthAccessTokenSecret("/\*ADD ACCESS TOKEN SECRET\*/");

// Create a twitter object to use to send tweet

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

twitter = tf.getInstance();

}

void draw() {

noStroke();

fill(0, 30);

rect(0, 0, width, height); // semi-transparent rectangle drawn over window to fade anything that isn't redrawn

drawRadar();

drawTextInfo();

drawScanLineAndDot();

updateButtonStatus();

image(leftButton,leftButtonX, leftButtonY);

image(twitterButton,twitterButtonX, twitterButtonY);

image(rightButton,rightButtonX, rightButtonY);

}

// draws the radar bounded by 1000x1000 area

void drawRadar() {

// all the rings

strokeWeight(1);

noFill();

stroke(#109856);

ellipse(500, 500, 1000, 1000);

ellipse(500, 500, 750, 750);

ellipse(500, 500, 500, 500);

ellipse(500, 500, 250, 250);

// all the lines center at (500, 500) to the edge of the radar

strokeWeight(2);

line(500, 500, 500+500\*cos(0), 500+500\*sin(0));

line(500, 500, 500+500\*cos(PI/6), 500+500\*sin(PI/6));

line(500, 500, 500+500\*cos(PI/3), 500+500\*sin(PI/3));

line(500, 500, 500+500\*cos(PI/2), 500+500\*sin(PI/2));

line(500, 500, 500+500\*cos(2\*PI/3), 500+500\*sin(2\*PI/3));

line(500, 500, 500+500\*cos(5\*PI/6), 500+500\*sin(5\*PI/6));

line(500, 500, 500+500\*cos(PI/3), 500+500\*sin(PI/3));

line(500, 500, 500+500\*cos(PI), 500+500\*sin(PI));

line(500, 500, 500+500\*cos(7\*PI/6), 500+500\*sin(7\*PI/6));

line(500, 500, 500+500\*cos(4\*PI/3), 500+500\*sin(4\*PI/3));

line(500, 500, 500+500\*cos(3\*PI/2), 500+500\*sin(3\*PI/2));

line(500, 500, 500+500\*cos(5\*PI/3), 500+500\*sin(5\*PI/3));

line(500, 500, 500+500\*cos(11\*PI/6), 500+500\*sin(11\*PI/6));

// thicken the line for the currently selected angle

strokeWeight(8);

switch( currentWedge ) {

case 0: line(500, 500, 500+500\*cos(PI), 500+500\*sin(PI)); break;

case 1: line(500, 500, 500+500\*cos(7\*PI/6), 500+500\*sin(7\*PI/6)); break;

case 2: line(500, 500, 500+500\*cos(4\*PI/3), 500+500\*sin(4\*PI/3)); break;

case 3: line(500, 500, 500+500\*cos(3\*PI/2), 500+500\*sin(3\*PI/2)); break;

case 4: line(500, 500, 500+500\*cos(5\*PI/3), 500+500\*sin(5\*PI/3)); break;

case 5: line(500, 500, 500+500\*cos(11\*PI/6), 500+500\*sin(11\*PI/6)); break;

case 6: line(500, 500, 500+500\*cos(0), 500+500\*sin(0)); break;

}

// display axis in terms of cm

textFont(font);

fill(#109856);

text("50", 505, 390);

text("100", 505, 265);

text("150", 505, 140);

text("200", 505, 15);

text("50", 505, 620);

text("100", 505, 745);

text("150", 505, 870);

text("200", 505, 995);

text("50", 380, 515);

text("100", 255, 515);

text("150", 130, 515);

text("200", 5, 515);

text("50", 610, 515);

text("100", 730, 515);

text("150", 855, 515);

text("200", 980, 515);

noFill();

}

// draws the scanning line and displays a red dot(if sensor data is valid) at the correct angle and distance on the radar

void drawScanLineAndDot() {

strokeWeight(10);

stroke(#109856);

line(500, 500, 500+500\*cos(scanAngle), 500+500\*sin(scanAngle)); // draw the new position of the scanning line

//float dist = random(0, 200); // RNG used for testing

if( itemDistance != 0 ) { // ignore data distances that are 0, which means out of range according to Arduino code

// determine the currently selected angle and draw a red dot at the distance calculated

// until the scanning line moves PI/3 radians past the selected angle,

// at which point the dot will start to fade

// also update the displayed distance for drawTextInfo()

if( currentWedge == 0 && scanAngle > PI && scanAngle < 4\*PI/3 ) {

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(PI), 500+(itemDistance\*2.5)\*sin(PI), 12, 12);

dispDist = itemDistance;

}

else if( currentWedge == 1 && scanAngle > 7\*PI/6 && scanAngle < 3\*PI/2 ) {

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(7\*PI/6), 500+(itemDistance\*2.5)\*sin(7\*PI/6), 12, 12);

dispDist = itemDistance;

}

else if( currentWedge == 2 && scanAngle > 4\*PI/3 && scanAngle < 5\*PI/3 ) {

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(4\*PI/3), 500+(itemDistance\*2.5)\*sin(4\*PI/3), 12, 12);

dispDist = itemDistance;

}

else if( currentWedge == 3 && scanAngle > 3\*PI/2 && scanAngle < 11\*PI/6 ) {

stroke(#ff0000);

ellipse(500-(itemDistance\*2.5)\*cos(3\*PI/2), 500+(itemDistance\*2.5)\*sin(3\*PI/2), 12, 12);

dispDist = itemDistance;

}

else if( currentWedge == 4 && scanAngle > 5\*PI/3 && scanAngle < 2\*PI ) {

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(5\*PI/3), 500+(itemDistance\*2.5)\*sin(5\*PI/3), 12, 12);

dispDist = itemDistance;

}

else if( currentWedge == 5 && scanAngle > 11\*PI/6 && scanAngle < 2\*PI ) {

wedge5Mem = itemDistance;

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(11\*PI/6), 500+(itemDistance\*2.5)\*sin(11\*PI/6), 12, 12);

dispDist = itemDistance;

}

else if( wedge5Mem != 0 && currentWedge == 5 && scanAngle > 0 && scanAngle < PI/6 ) {

stroke(#ff0000);

ellipse(500+(wedge5Mem\*2.5)\*cos(11\*PI/6), 500+(wedge5Mem\*2.5)\*sin(11\*PI/6), 12, 12);

}

else if( currentWedge == 6 && scanAngle > 0 && scanAngle < PI/3 ) {

stroke(#ff0000);

ellipse(500+(itemDistance\*2.5)\*cos(0), 500+(itemDistance\*2.5)\*sin(0), 12, 12);

dispDist = itemDistance;

}

}

// update the red dot distance every time the scan line moves 2\*PI radians

// also reset the scanAngle back to 0 rads

if ( scanAngle > 2\*PI ) {

scanAngle = 0;

itemDistance = realTimeItemDistance;

//itemDistance = dist; // for RNG testing

}

else { // otherwise just increment the scanAngle for the next scanning line to be drawn

scanAngle += scanSpeed;

}

}

// draws all the text that is displayed in the area x > 1000 and y < 500

// text includes name of project, range, temperature, and direction data

void drawTextInfo() {

// title

fill(#109856);

textFont(font, 48);

text("LAB 5 RADAR", 1010, 100);

// labels & units

textFont(font, 28);

text("dist: ", 1010, 150);

text("dir: ", 1010, 200);

text("temp: ", 1010, 250);

text(realTimeTemperature, 1100, 250);

text("°C", 1275, 250);

text(getDirection(), 1100, 200);

if( dispDist == 0 ) { // data of 0 means out of range as decided by Arduino code

text("OUT OF RANGE!", 1100, 150);

}

else {

text(str(dispDist), 1100, 150 );

text("cm", 1275, 150);

}

noFill();

}

// returns a String to indicate the current direction of scanning

String getDirection() {

switch( currentWedge ) {

case 0: return "900 o'clock";

case 1: return "1000 o'clock";

case 2: return "1100 o'clock";

case 3: return "1200 o'clock";

case 4: return "100 o'clock";

case 5: return "200 o'clock";

case 6: return "300 o'clock";

default: return null;

}

}

// parse and update the distance and temperature data from the sensors

void serialEvent(Serial myPort) {

String inString = myPort.readString();

String[] sensorData;

if ( inString != null ) {

sensorData = split(inString, ',');

realTimeItemDistance = Float.parseFloat(sensorData[0]);

realTimeTemperature = Float.parseFloat(sensorData[1]);

}

}

// sends a tweet displaying the current direction and distance away from an object

// if not object, tweets that there is nothing in range

// throws TwitterException if a problem is encountered while tweeting

void tweet()

{

try

{

//Declare a Status object in order to update Twitter status

Status status;

if (dispDist == 0) {

//Status on Twitter will be "No Object in range!" if not in our defined range

status = twitter.updateStatus("No object in range!");

}

else {

//Otherwise display Object's current location

status = twitter.updateStatus("Object " +dispDist + " cm away at " +getDirection() + "!");

}

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

}

catch (TwitterException te)

{

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

}

}

//check if one of the button is selected, if yes, resize the button to indicate the selection

//effect and set that button status to true

void updateButtonStatus() {

if (overTwitterButton()) {

twitterButtonOver = true;

leftButtonOver = false;

rightButtonOver = false;

twitterButton.resize(110, 90);

leftButton.resize(100, 80);

rightButton.resize(100, 80);

}else if(overLeftButton()){

twitterButtonOver = false;

leftButtonOver = true;

rightButtonOver = false;

leftButton.resize(110, 90);

rightButton.resize(100, 80);

twitterButton.resize(100, 80);

}else if(overRightButton()){

twitterButtonOver = false;

leftButtonOver = false;

rightButtonOver = true;

rightButton.resize(110, 90);

leftButton.resize(100, 80);

twitterButton.resize(100, 80);

}

else {

leftButtonOver = rightButtonOver = twitterButtonOver = false;

leftButton.resize(100, 80);

twitterButton.resize(100, 80);

rightButton.resize(100, 80);

}

}

//mouse click event and check wichi button is clicked

void mousePressed() {

if (twitterButtonOver) {

tweet();

}

else if(leftButtonOver){

if(currentWedge-1>=0){

--currentWedge; //move the scan edge to left

}

}

else if(rightButtonOver){

if(currentWedge+1<=6){

++currentWedge; //move the scan edge to right

}

}

}

//check if the the mouse is on the left button

boolean overLeftButton() {

if (mouseX >= leftButtonX && mouseX <= leftButtonX+buttonWidth &&

mouseY >= leftButtonY && mouseY <= leftButtonY+buttonHeight) {

return true;

} else {

return false;

}

}

//check if the the mouse is on the right button

boolean overRightButton() {

if (mouseX >= rightButtonX && mouseX <= rightButtonX+buttonWidth &&

mouseY >= rightButtonY && mouseY <= rightButtonY+buttonHeight) {

return true;

} else {

return false;

}

}

//check if the the mouse is on the twitter button

boolean overTwitterButton() {

if (mouseX >= twitterButtonX && mouseX <= twitterButtonX+buttonWidth &&

mouseY >= twitterButtonY && mouseY <= twitterButtonY+buttonHeight) {

return true;

} else {

return false;

}

}