LAB 2 REPORT

By

Qais Qamhia

(NB-427)

02/04/2020, Bremen

Jacobs University Bremen

Introduction to Robotics and Intelligent Systems Lab

Dr. Fangning Hu, instructor

Introduction:

In this lab I got to try more sensors (Thermometer, Ultrasonic distance sensor), the Servomotor, and how they all could be implemented in our intelligent system managed by the Arduino.

In addition, this lab introduced a new programming language (Processing), which is basically modified Java.
This language enables us to make computer programs that use data, such as mouse coordinates, as input for our intelligent system. Or use our circuit's output as input for this computer program, and change the outcome of this program accordingly.

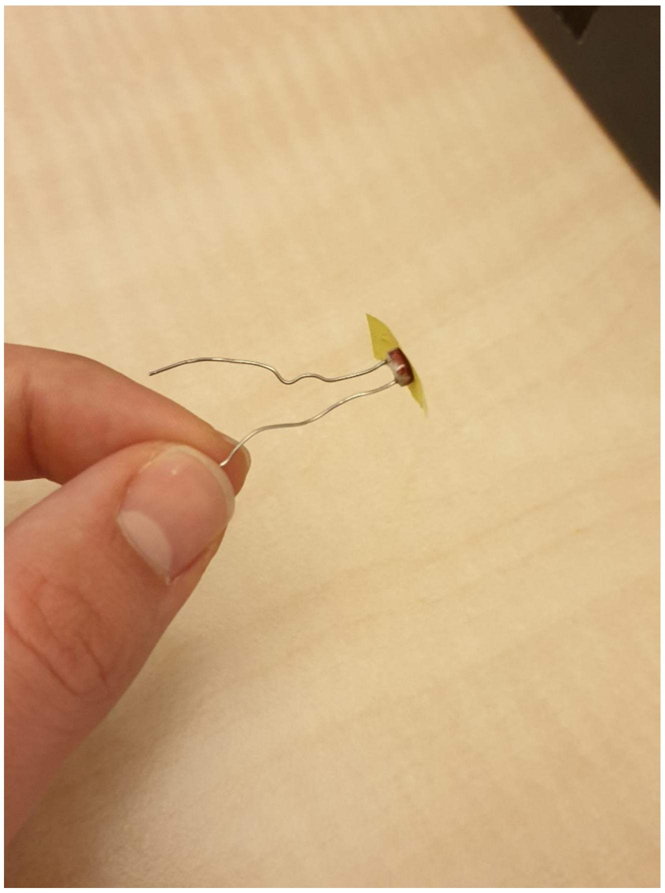
Execution:

Task 2.1:

When the light is high the resistance of the LDR is low. And when it's dark, the resistance of the LDR becomes high.

I connected the Multimeter legs properly (Red into HzVΩ and the black one into COM), then covered the LDR with a piece of paper and measured the resistance:

When I covered it and dimmed the lights the resistance became really high => 0.2 mOhm
When I removed the paper and turned the lights on, the resistance became low => 3.35 kOhm

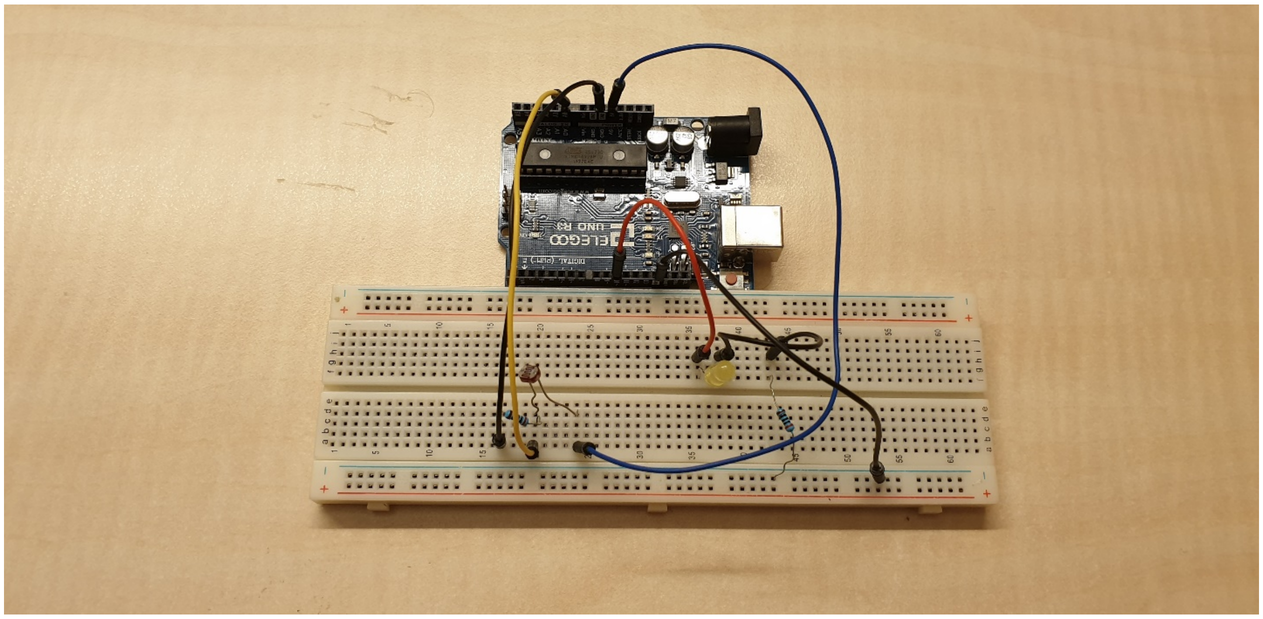


Task 2.2:

I connected the LDR to a 5V voltage from one leg. And to a resistor that goes to GND, and to input analog A0 with the other leg. And the rest of the circuit is a normal LED circuit as the one made in previous tasks.

The light around the LDR affects its resistance, and the resistance affects the voltage that goes through the LDR. This voltage will be given to us as analog input (0 – 1023 numbers).
When it's below a certain number (Low voltage; High resistance; Low ambient light) we're going to give HIGH output for our LED. (Turn it on)
And when it's above this number (High voltage; Low resistance; High ambient light) we're going to give LOW output to our LED. (Turn it off)
Now the sensor values that I got were around 50-200. So the LED was always one as the sensor reading would never go above 650. And this is due to the fact that voltage is being cut off by the extra resistor we put (R2). That's why picking a lower R2 (1 kOhm) would be more suitable, and overall better for this circuit.

I had to change the first IF-statement (sensor_sample < 650) into (< 60) instead, and it worked perfectly then!
The light turned off when the ambient light is high, and turned back on when I covered it with my finger or a piece of paper.



Task 2.3:

I connect the temperature sensor (Flat side pointing towards me) such that the left pin is connected to the 5V from the Arduino board, the pin on the right is connected with GND, and the middle pin with the analog input A0 pin.

The sensor will send back voltage in the form of analog signal as input to our Arduino board. And the program will take these values and map them into the corresponding Celsius

temperatures between -50 and 150.
And it will also calculate and show the standard deviation of our temperatures.

The temperature shown will be rather stable, until we touch it with our finger, then it'll go up. And we take our hand off it'll go down again.

Task 2.4:

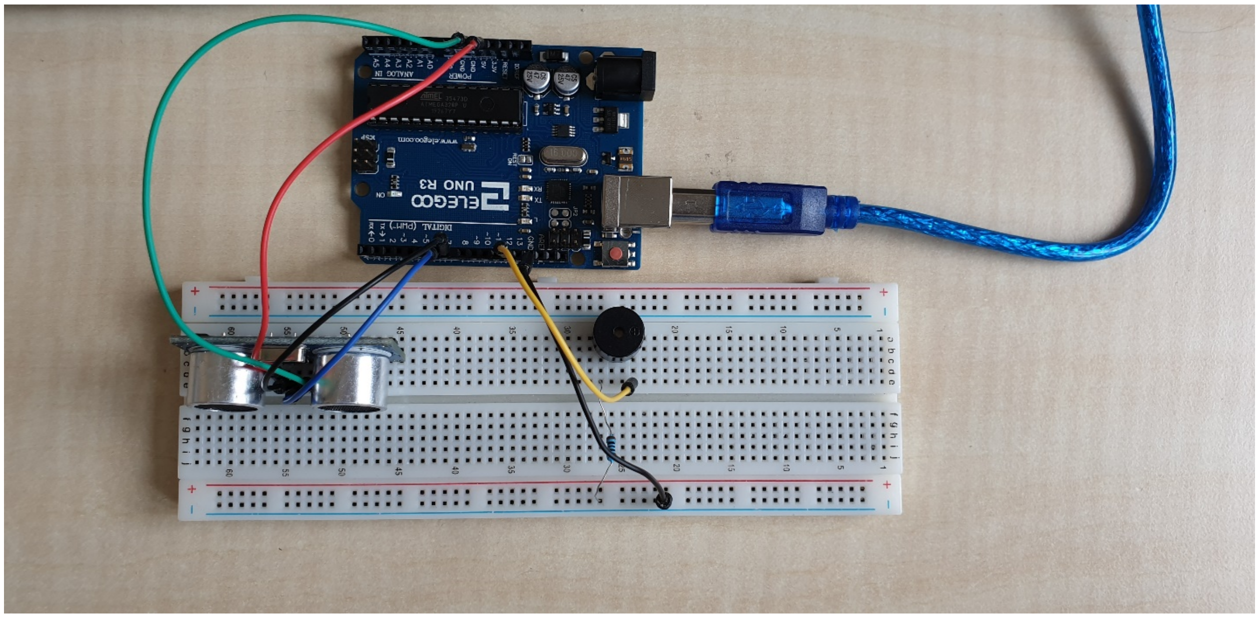
The map function will link the analog values from the sensor into the corresponding temperature values in Celsius. In the map function we enter: the number to map, the lower bound of the value's current range, the upper bound of the current range, the lower bound of the value's target range and the upper bound of its target range.

In this case the current range is our analog readings of the voltages given by the sensor. These values normally range from 0 – 1023 when we have 0V – 5V range. But in this case the sensor only reads 0V – 5V, that's why our current range's higher bound is 410 (The number in analog that corresponds to 2V)!

Task 2.5:

I connected the Ultrasonic distance sensor as described (Vcc – 5V, Trig – pin #7, Echo – pin #6, GND – GND). With the program given, the sensor sends signals from the speaker, and receives the bouncing signals back through the microphone, and based one the speed of the sound waves, and the time it took to return it'll decide how far away the object (Our hand for example) is from the device.
These results will be printed out in our serial monitor.

For the second part of this task I had to connect a buzzer on this circuit. I connected the positive leg with pin #12 (from which our buzzer will get the power needed), and the negative leg to a resistor that goes to GND.

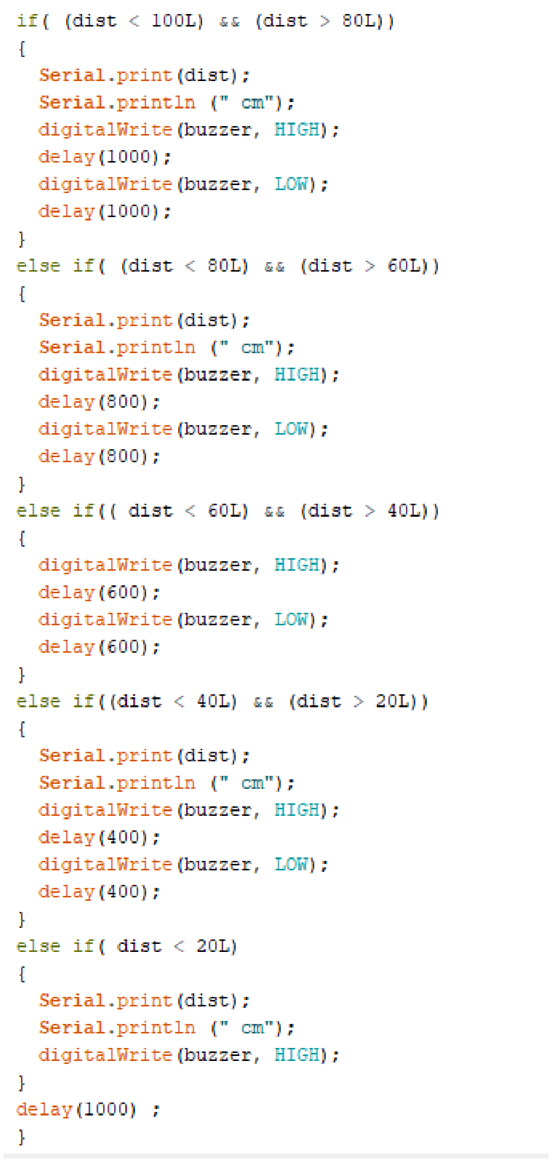


For the programming part, I kept the first part of the program the same, as I want the Distance sensor to do the exact same thing, and just give the distance as output.

For the second part of the program, I used this distance output inside some IF-statements to make the buzzer beep at specific distances with a certain frequency.
When the distance becomes less than 100cm but still more than 80cm, the buzzer will have HIGH output (will beep) for a second, then it will stop, and have a one second delay after it does. So that this way we'll get slow beeps when the distance is rather large.

I divided the distance from 100cm – 0cm into 5 intervals, with the delay between the beeps getting shorter and shorter, until it is below 20cm, then there is no delay at all (continuous beeping).

Here is a picture of the code for extra clarification:



```
if( (dist < 100L) && (dist > 80L))
{
  Serial.print(dist);
  Serial.println (" cm");
  digitalWrite(buzzer, HIGH);
  delay(1000);
  digitalWrite(buzzer, LOW);
  delay(1000);
}
else if( (dist < 80L) && (dist > 60L))
{
  Serial.print(dist);
  Serial.println (" cm");
  digitalWrite(buzzer, HIGH);
  delay(800);
  digitalWrite(buzzer, LOW);
  delay(800);
}
else if(( dist < 60L) && (dist > 40L))
{
  digitalWrite(buzzer, HIGH);
  delay(600);
  digitalWrite(buzzer, LOW);
  delay(600);
}
else if((dist < 40L) && (dist > 20L))
{
  Serial.print(dist);
  Serial.println (" cm");
  digitalWrite(buzzer, HIGH);
  delay(400);
  digitalWrite(buzzer, LOW);
  delay(400);
}
else if( dist < 20L)
{
  Serial.print(dist);
  Serial.println (" cm");
  digitalWrite(buzzer, HIGH);
}
delay(1000) ;
}
```

Task 2.6:

Use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins.

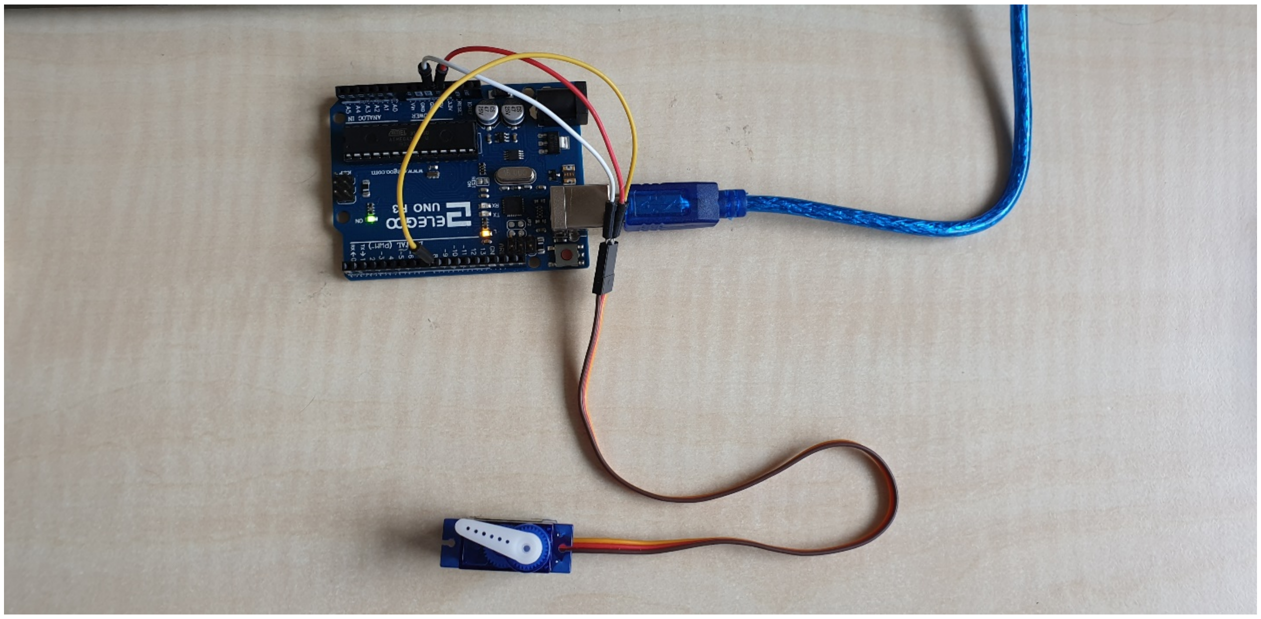
Task 2.7:

I properly connected the Servomotor to the Arduino board (Brown wire - GND. Red wire – 5V. Orange wire – signal). The Servomotor receives the instructions from our code as input through the signal (Orange) wire, and does the rotations needed.

For this task, the program defines pin #8 as the signaling pin for the Servomotor. Then it commands it to rotate with specific angles (0°, 45°, 90°, 135°, 180°) with 3 seconds delay between each rotation.

I added the following lines of code at the end of the program for the Servomotor to turn back to 0°, and then I added s.detach() to stop the movement of the Servomotor at 0°:

```
s.write (0);
delay (3000) ;
s.detach();
```



Task 2.8:

I used a for loop to make a continuous motion for the servo shaft, defining a variable for the angle, starting from 0 and incrementing 1 each loop to cover all angles between 0 and 180.

I also added delay after each movement which controls the speed for which the shaft is rotating, as it basically defines the time it takes to increment the angle by 1 and change the shaft's position slightly.

Therefore, the higher the delay, the slower the shaft gets and vice versa.

We want the cycle to be 6 seconds long (6000 ms), i.e. each half cycle (from 0° – 180°) should take 3 seconds. For that we need to divide this number by the number of positions (angles) in each half cycle, so we could decide the delay needed after each one to make the full length of the cycle equal to 6 seconds.

=> 3000/180 = 16 ms => the delay after each position change

I also added 6 more seconds between every cycle and the next.

So, the final code will look as the following:

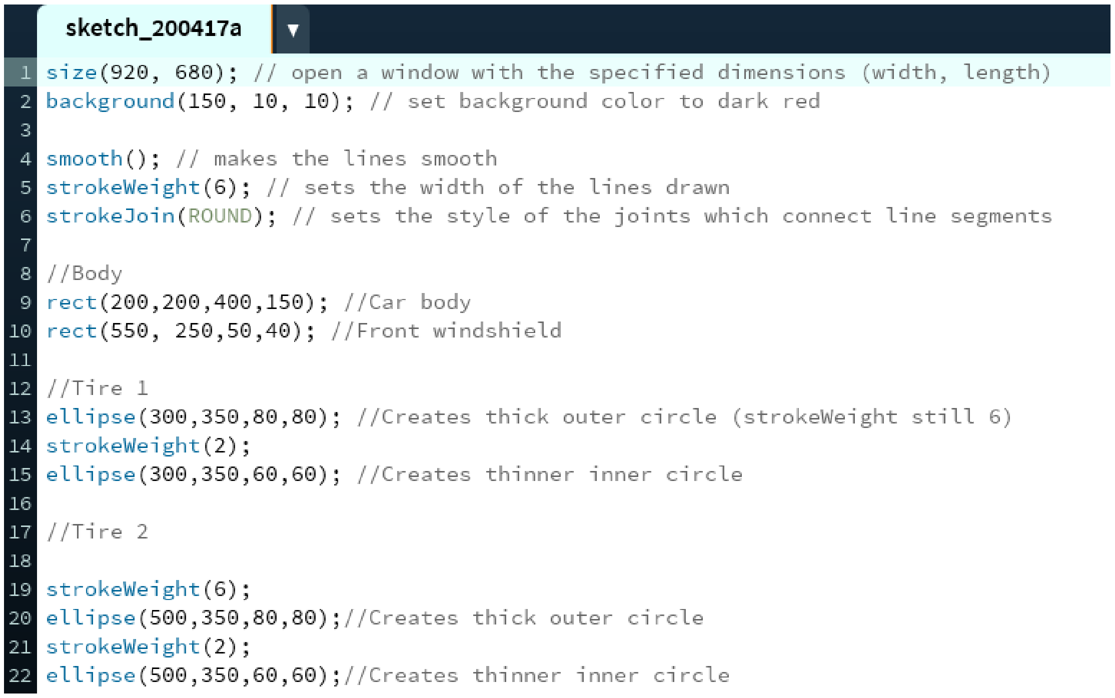
```
#include <Servo.h>
Servo s; // create servo object ;
int angle = 0;
void setup ()
{
s.attach (8); // control signal on pin 8
}
void loop ()
{

for(angle = 0; angle < 180; angle++)  // Goes from 0 degrees to 180 degrees
{
 s.write(angle);                      // Tell the Servo to go to position in variable 'angle'
 delay(16);                           // Waits 16ms for the Servo to reach the position
}
for(angle = 180; angle > 0; angle--)
{
 s.write(angle);
 delay(16);
}

delay(6000);
}
```

Here is also a video demonstration: (Note: This video was taken before me changing the delay values to 16, so the cycle might be faster than 6 seconds):

MP4

20200416_181101.mp4

Task 2.9/Task 2.10:



```
size(920, 680); // open a window with the specified dimensions (width, length)
background(150, 10, 10); // set background color to dark red

smooth(); // makes the lines smooth
strokeWeight(6); // sets the width of the lines drawn
strokeJoin(ROUND); // sets the style of the joints which connect line segments

//Body
rect(200,200,400,150); //Car body
rect(550, 250,50,40); //Front windshield

//Tire 1
ellipse(300,350,80,80); //Creates thick outer circle (strokeWeight still 6)
strokeWeight(2);
ellipse(300,350,60,60); //Creates thinner inner circle

//Tire 2

strokeWeight(6);
ellipse(500,350,80,80);//Creates thick outer circle
strokeWeight(2);
ellipse(500,350,60,60);//Creates thinner inner circle
```

In these tasks, I used Processing -with the help of their reference page- to create a simple 2D car.

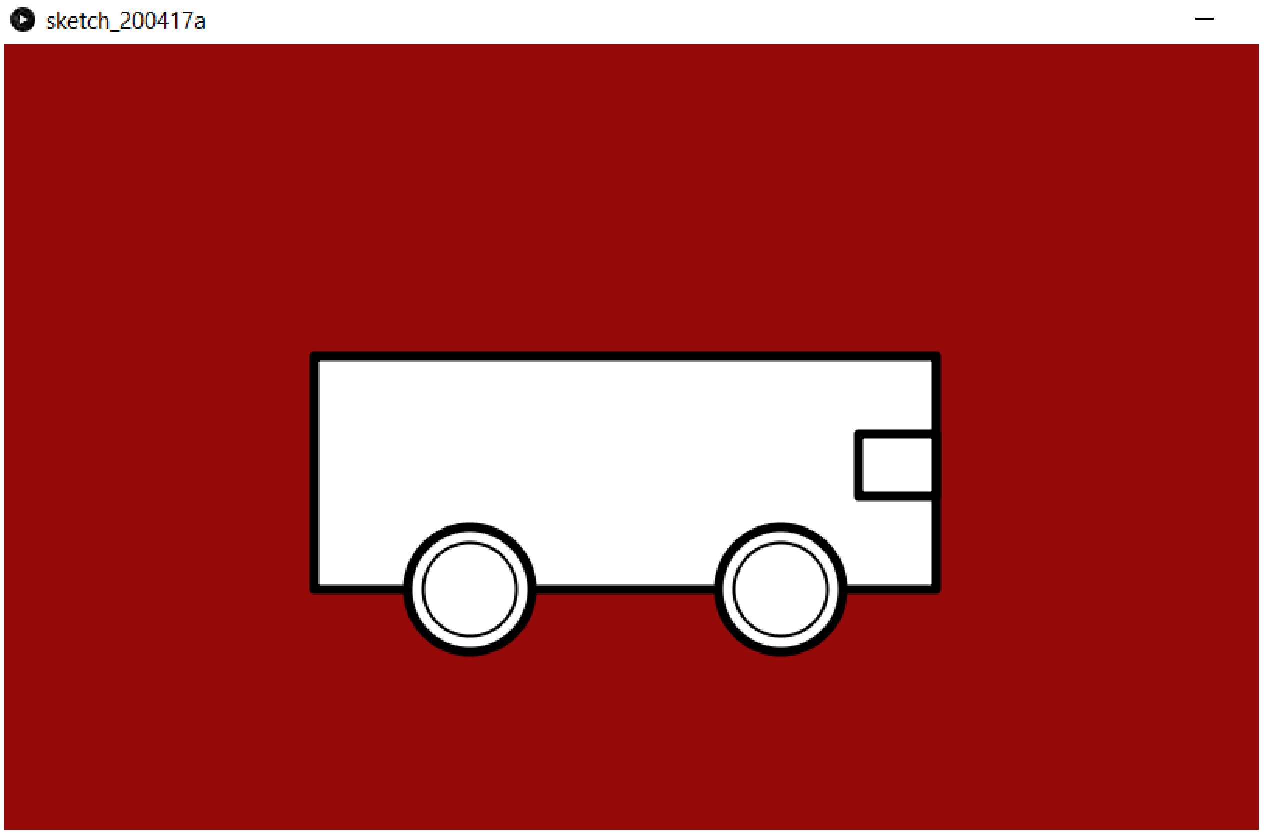
I first defined the window that I used to visualize my data, with the dimensions of 920 pixels for width, and 680 pixels for height. Then I chose a color for the background using the function background (R, G, B).
Next, I used the function smooth(), which makes all the lines and shapes created after this function has been called smoothened out, and give them more aesthetically pleasing look.
Note that in processing this function, and similar ones that decide the shapes, looks, colors of strokes (lines), caps, joints…etc, will have a continuous effect on all the lines of code mentioned after, until stated otherwise.
For example, you would need to use nosmooth() after we used smooth() the first time, in order to reverse its effect.

Then I set the stroke's (drawing lines') width to 6 pixels and set the style of the joints that connect the lines together into ROUND.

For the body part, I created one big rectangle, and then a smaller rectangle that represents a windshield which I placed at the front of the bigger one.
Also, the fact that I made the smaller parts (windshield, and later on the tires) after the main body, allow these parts to be placed on top of the first one, and therefore be visible, and not hidden by the main body of the car.

Then I created the tires, a big circle with stroke weight kept at 6, then a thinner and smaller circle inside of it (stroke weight 2 and smaller radius).

And below is the outcome:

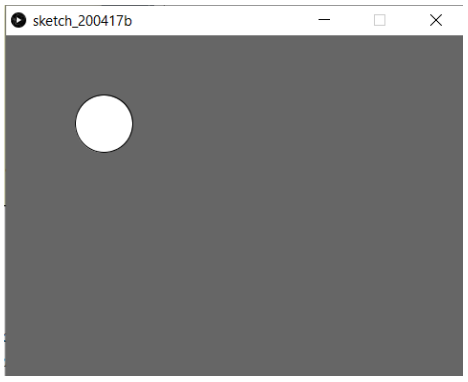


Task 2.11:

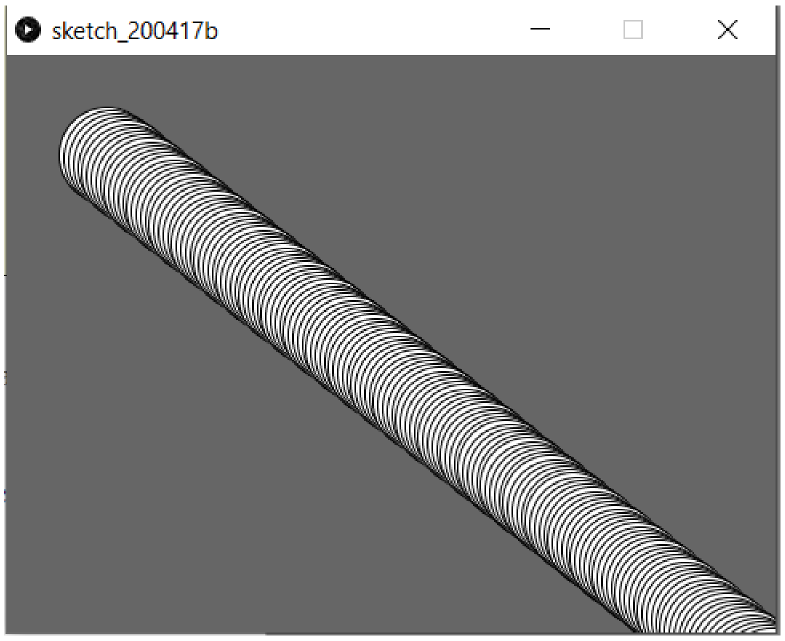
This code will define coordinates and dimensions of a circle, then it'll open a new window and set the frame rate 30 frames per second to be shown on this window.

Inside the draw function, the commands will be looped. The program will set the background into grey color for the next set of operations, and then apply translation into the circle's coordinates by adding 2.8 to its x-coordinate and 2.2 to its y-coordinate. After that the circle will be drawn for the first time with the new coordinates

The loop will start over again, setting the background once again into grey (for the next set of commands), while at the same time applying the translation into the circle's coordinates and drawing a new circle, and by looping this background color change, it'll delete the trace of the old circle each loop, as the new commands, and new visualizations will always be put on top of the old ones. Therefore, the old circle will be covered by the new background and the new circle with the new updated coordinates will be drawn on this new background, making a smooth translation for this circle from the top left corner into the bottom right.



By putting the background() function in setup, we'll run it only once at the beginning of the execution, setting the background into grey the for the first time. But then after entering the loop, we'll have nothing to delete the traces of the old circles after each loop cycle, which will cause the circle to leave all it's traces behind (Each new circle will be drawn over the old one, covering most of it, but leaving a small part which will cause the whole track to be visible at the end). And it'll also go from the top left corner into the bottom right corner (as we changed nothing in that)



Task 2.12:

For this task I connected a basic LED circuit (Positive leg LED with resistor – Pin #9, Negative leg LED – GND).

In the Processing part:
I imported the Processing library for serial communication by using the command:
import processing.serial.*;
Then I created and opened a serial object that is connected to same port and serial our Arduino chip is attached to, which also runs at the same speed as the Arduino serial.

We open a window and run a loop function, which draws a line from the beginning till the end of the window (Each loop the beginning and end point of the line will be increased by one, making it span the whole x-axis of the window), and changing color each loop making it brighter and brighter each loop (We increment the RGB value in the stroke() function, starting from 0 which corresponds to black, until 255 which corresponds to white).
This whole step is just for aesthetics, to make us visualize when the led is going to become brighter, and when it'll be dimmed more.

Then we'll write and store the x-value of our mouse position on the port that we previously created.

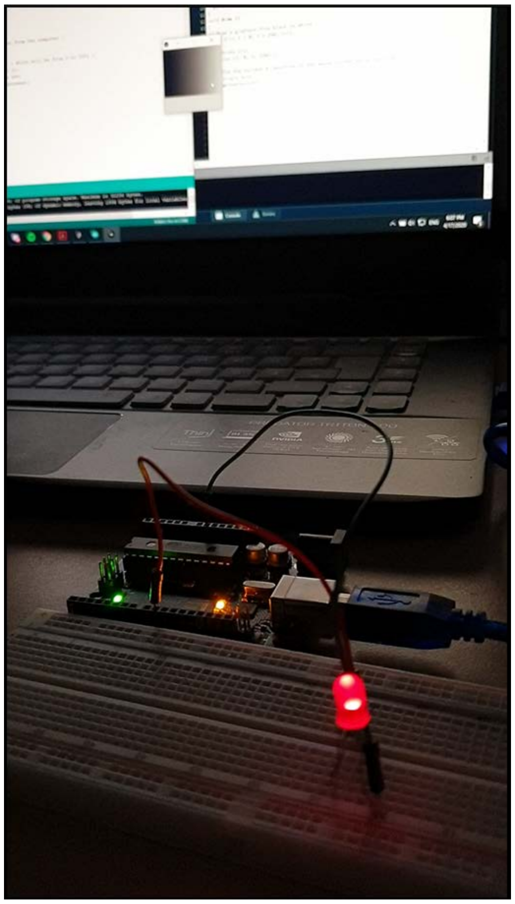
In the Arduino part:
We start the serial at the same speed as the other one. So that both of them will now be connected to the same PC port and running in the same speed (Both serials are connected to one another; Both are connected to one serial).

We define a variable called brightness as a byte (stores an 8-bit unsigned number, from 0 to 255). And whenever there is data coming from our Processing serial (Mouse x-coordinate) we'll take that value and store it in this variable.
Then we'll use this value in analogWrite() function to provide voltages to our LED accordingly. As analogWrite() takes its second argument as a value from 0-255, such that analogWrite(255) requests a 100% duty cycle (maximum voltage will go to the led pin), and analogWrite(127) is a 50% duty cycle (half the maximum voltage will go to the led pin), and analogWrite(0) is 0% duty cycle (no voltage will go to our LED).

Note that our window size in Processing is 200 by 200 pixels, so the maximum x value possible of the mouse coordinate will be 200. And this will guarantee that we won't try to store values larger than one byte could handle, or try to putting a value larger than 255 in our analogWrite() function.

The final result of all this will be that we'll have a window in our Processing program with a gradient from black to white, and depending on our mouse placement over this window, the LED brightness will become higher or lower.

MP4
20200417_180734.mp4



Conclusion:

The Arduino part of this lab was rather clear and simple. The only issues I faced were with the distance sensor, and they were easily fixed by rechecking the code, and fixes some small syntax errors.

For the Processing part, it was not easy to follow up to all the new info all once. I had to look up almost every single function in order to understand what it takes as arguments and properly understand what it does.
On top of that the general logic behind this language's syntax and its code structure was confusing at first. I had to test around for a while, and continuously check their references site in order to comprehend how to properly write the codes.

References:

https://www.arduino.cc/reference/en
https://www.arduino.cc/en/Reference/Servo
https://www.arduino.cc/en/Tutorial/Sweep
https://www.arduino.cc/reference/en/language/variables/data-types/byte/
https://processing.org/reference/