JACOBS UNIVERSITY

Introduction to Robotics and Intelligent Systems Lab
Lab 2

Aryans Rathi
a.rathi@jacobs-university.de

Suraj Giri
s.giri@jacobs-university.de

Riley Edwin Sexton
r.sexton@jacobs-university.de

Instructors name
Dr. Fangning Hu

2.1 Light Dependent Resistor

Task 2.1

In this task we were instructed to measure the range of resistances in the LDR (light dependent resistor seen in figure 1).We did this by attaching a multimeter to the LDR (figure 2) and adjusting levels of light to simulate bright light and a thick sheet of paper with results shown in table 1.

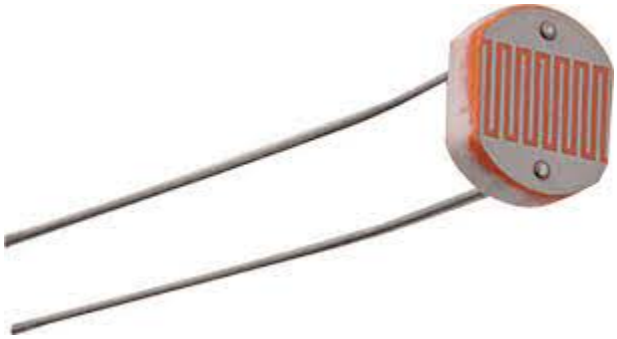

Figure 1, the LDR (light dependent resistor)

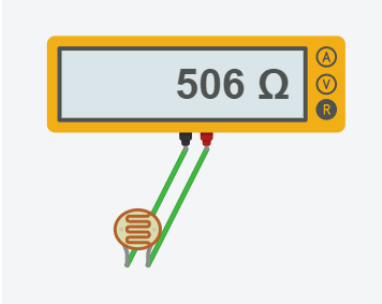

Figure 2, Resistance test on LDR

Percent of light	0%	25%	50%	75%	100%
Resistance recorded	180 KOhms	1.33 KOhms	940 Ohms	686 Ohms	506 Ohms

Table 1, findings from testing the setup in figure 3 with different levels of simulated light

Task 2.2

For task 2.2 we were instructed to create a circuit with a LDR as a voltage divider using the code and schematic provided (figures 3-5). Given the option between using a 1 KOhm resistor or a 10 KOhms resistor, the 10 KOhm resistor is more desirable as it allows there to be a much more pronounced difference between the bright and covered states of the photoresistor. This still works with the threshold set to 650 as the LED will still light up when the photoresistor is covered.

```
int input_pin= A0;
int LED = 10;
int sensor_sample = 0;

void setup(){
 Serial.begin(9600);
 pinMode (LED, OUTPUT);
}

void loop(){
  sensor_sample = analogRead(input_pin);
  Serial.print("sensor value = " );
  Serial.println(sensor_sample);

  if (sensor_sample < 650 ){
    digitalWrite(LED, HIGH);
  }else{
     digitalWrite(LED, LOW);
  }
  delay (50);
}
```

Figure 3, Code provided for voltage divider by the lab manual

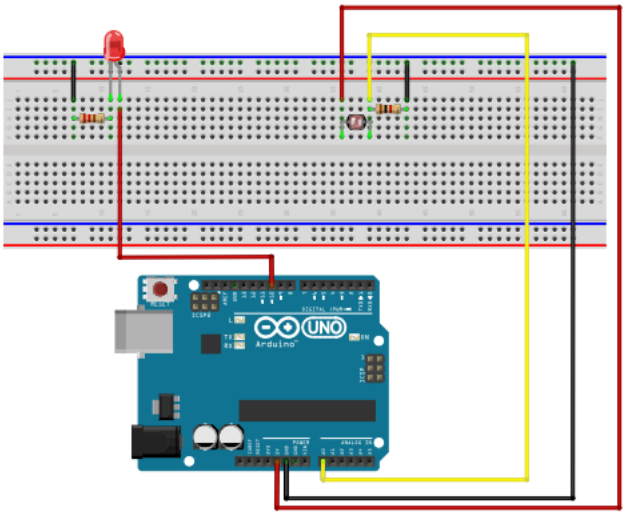

Figure 4, example setup provided by lab manual

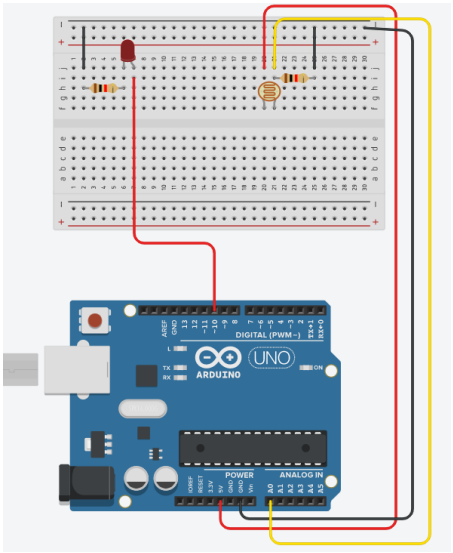

Figure 5, our setup made in tinkercad

2.2 Temperature Sensor

Task 2.3

In task 2.3 we are instructed to use the temperature sensor (figure 6) to create the circuit displayed in the lab manual (figures 7 and 8). We also use the code provided (figure 9) in the lab manual to create a program that detects temperature through the sensor and displays it on the serial monitor.

Figure 6, Arduino Temperature sensor

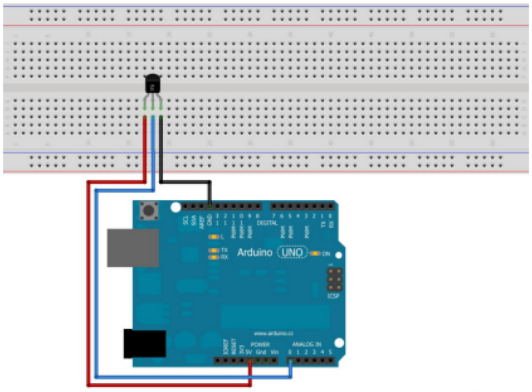

Figure 7, Temperature sensor circuit from lab manual

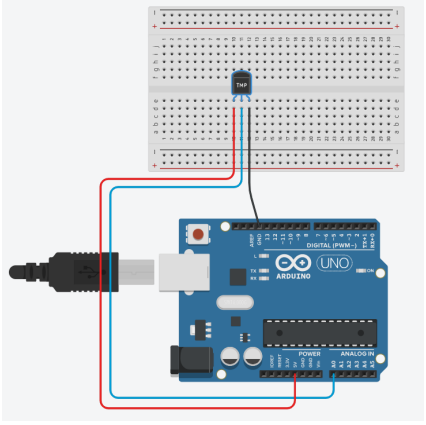

Figure 8, our recreation of the temperature Sensor from the lab manual

```
int TMP36= A0; // The middle lead is connected to analog-in 0.
int temperature= 0;

int wait_ms= 20; // wait time betweem measurements in millisec.

#define NR_SAMPLES 10
int samples[NR_SAMPLES]; // array of samples

void setup(){
  Serial.begin(9600);
}

void loop(){
  float sum= 0.0;
  for (int i=0; i< NR_SAMPLES; ++i){
    // map values from range [0, 410] to [-50, 150]
    samples[i]= map(analogRead(TMP36), 0, 410, -50, 150);
    sum += samples[i];
    delay(wait_ms);
  }
  float mean= sum/NR_SAMPLES;
  float sum_square_deviation= 0.0;
  for (int i=0; i< NR_SAMPLES; ++i){
    sum_square_deviation += (samples[i] - mean)*(samples[i] - mean);
  }
  float standard_deviation= sqrt(sum_square_deviation/NR_SAMPLES);
  Serial.print("mean: ");
  Serial.print(mean, 3);
  Serial.print(" C, \t std: ");
  Serial.println(standard_deviation);

}
```

Figure 9, Code for temperature sensor from the lab Manual

Task 2.4

The value 410 (from the code in figure 9) is used as we need to remap the value we get from the temperature sensor which is in the range 0 to 410 into a real temperature in the range -50 to 150 that we can display.

2.3 Ultrasonic distance sensor

Task 2.5

In this task we are instructed to create a car sensor (figures 9 and 10) that beeps if something is within 100 cm then it will beep in rapid succession, if its less than 20 cm then it will play a constant tone.

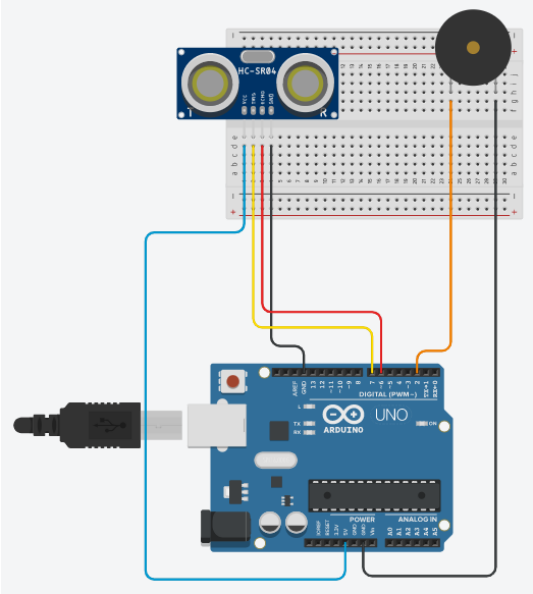

Figure 9, Circuit for car sensor

```
const int trig = 7;
const int echo = 6;
const int buzzer = 2;
void setup ()
{
    pinMode(buzzer, OUTPUT);
    pinMode (trig , OUTPUT );
    pinMode (echo , INPUT );
    digitalWrite (trig , LOW);
    Serial . begin (9600) ;
}
void loop ()
{
    // send an impulse to trigger the sensor start the measurement
    digitalWrite (trig , HIGH );
    delayMicroseconds (15) ; // minimum impulse width required by HC -SR4 sensor
    digitalWrite (trig , LOW);
    long duration = pulseIn (echo , HIGH ); // this function waits until the sensor
    //outputs the result
    // the result is encoded as the pulse width in microseconds
    // 'duration ' is the time it takes sound from the transmitter back to the
    //receiver after it bounces off an obstacle
    const long vsound = 340; // [m/s]
    long dist = ( duration / 2L) * vsound / 10000L; // 10000 is just the scaling
    //factor to get the result in [cm]
    // REMEMBER : when doing operations with 'long ' variables , always put 'L' after
    //constants otherwise you will have bugs !
    Serial.println (dist);
    if(dist < 100L && dist > 20L)
    {
        noTone(buzzer);
        tone(buzzer, 1000);
        delay(dist * 3);
        noTone(buzzer);
        delay(dist * 3);
    }
    else if(dist < 20L)
    {
    tone(buzzer, 1000);
    }
}
```

Figure 10, code for car sensor

2.4 Servomotor

Task 2.6

The servo library does not allow the use of PWM pins 9 and 10.

Task 2.7

The servo motor circuit in figure 11 uses the code in figure 12 to make the servo stop at 0 degrees.

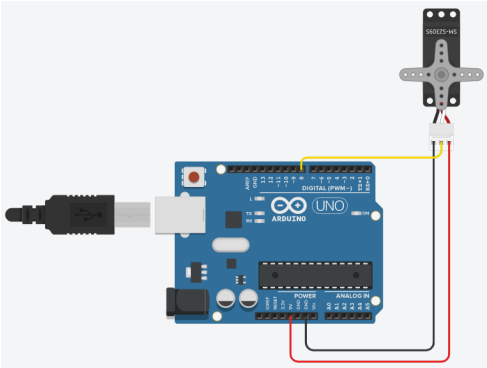

Figure 11, circuit used for the servomotor

```
#include <Servo.h>
Servo s; // create servo object ;

void setup()
{
    s.attach(8); // control signal on pin 8
}

void loop()
{
    // now we can control the servo with write ( angle )
    // angle of the servo 0 -180 degrees
    // 90 degrees = servo is centerd
    s.write(0);
    delay(3000);
}
```

Figure 12, code used to make the servo stop at 0 degrees

Task 2.8

In the code in figure 13 the servo transitions between 0 degrees and 180 degrees in a cycle that in total takes 6 seconds to complete.

```
#include <Servo.h>
Servo s; // create servo object ;

void setup()
{
    s.attach(8); // control signal on pin 8
}

void loop()
{
    // now we can control the servo with write ( angle )
    // angle of the servo 0 -180 degrees
    // 90 degrees = servo is centerd
    //full cycle is 6 seconds as it has two delays of 3 seconds
    s.write(0);
    delay(3000);
    s.write(180);
    delay(3000);
}
```

Figure 13, code for a servo that transitions between 0 and 180 degrees

2.5 Get Started with Processing

Task 2.9

In figure 14 is the car I made using processing and figure 15 is the code to draw the car.

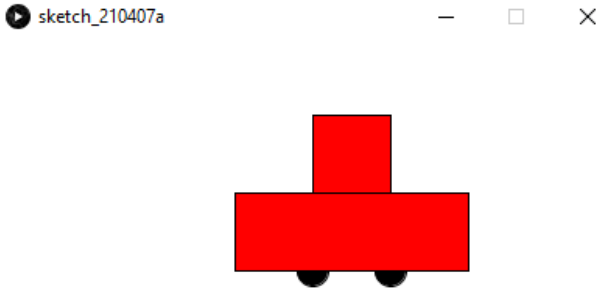

Figure 14, 2d car drawn using processing

```
const int trig = 7;
const int echo = 6;
const int buzzer = 2;
void setup ()
{
    pinMode(buzzer, OUTPUT);
    pinMode (trig , OUTPUT );
    pinMode (echo , INPUT );
    digitalWrite (trig , LOW);
    Serial . begin (9600) ;
}
void loop ()
{
    // send an impulse to trigger the sensor start the measurement
    digitalWrite (trig , HIGH );
    delayMicroseconds (15) ; // minimum impulse width required by HC -SR4 sensor
    digitalWrite (trig , LOW);
    long duration = pulseIn (echo , HIGH ); // this function waits until the sensor
    //outputs the result
    // the result is encoded as the pulse width in microseconds
    // 'duration ' is the time it takes sound from the transmitter back to the
    //receiver after it bounces off an obstacle
    const long vsound = 340; // [m/s]
    long dist = ( duration / 2L) * vsound / 10000L; // 10000 is just the scaling
    //factor to get the result in [cm]
    // REMEMBER : when doing operations with 'long ' variables , always put 'L' after
    //constants otherwise you will have bugs !
    Serial.println (dist);
    if(dist < 100L && dist > 20L)
    {
        noTone(buzzer);
        tone(buzzer, 1000);
        delay(dist * 3);
        noTone(buzzer);
        delay(dist * 3);
    }
    else if(dist < 20L)
    {
    tone(buzzer, 1000);
    }
}
```

Figure 15, code used to draw the car

Task 2.10

The car in figure 16 is improved by using noStroke(); and increased polygons and more colors.

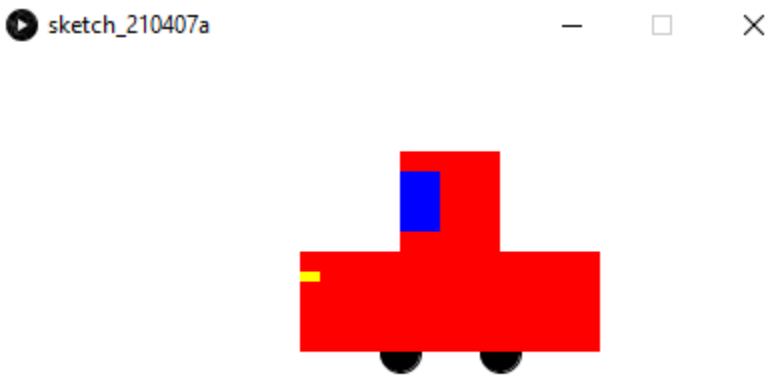

Figure 16, Improved car drawn in processing

```
void setup()
{
  size(400,200);
  background(255);
  fill(0,0,0);
  ellipse(200,150,20,20);
  ellipse(250,150,20,20);
  fill(255,0,0);
  noStroke();
  rect(150,100,150,50);
  rect(200,50,50,50);
  fill(255,255,0);
  rect(150,110,10,5);
  fill(0,0,255);
  rect(200,60,20,30);
}
```

Figure 17, code for drawing the improved car.

2.6 Moving Object

Task 2.11

Moving background to setup means it only runs at the start so when the draw function updates only the ellipse gets redrawn.

```
float x = 60;
float y = 60;
int diameter = 60;
void setup ()
{
  size (480 , 360) ;
  frameRate (30) ;
}
void draw ()
{
  background (102) ;
  x = x +2.8;
  y = y +2.2;
  ellipse (x , y , diameter , diameter ) ;
}
```

Figure 18, Working code for Task 2.11

```
float x = 60;
float y = 60;
int diameter = 60;
void setup ()
{
  size (480 , 360) ;
  frameRate (30) ;
  background (102) ;
}
void draw ()
{
  x = x +2.8;
  y = y +2.2;
  ellipse (x , y , diameter , diameter ) ;
}
```

Figure 19, Modified code for Task 2.11

Figure 20, Background is redrawn along with the ellipse

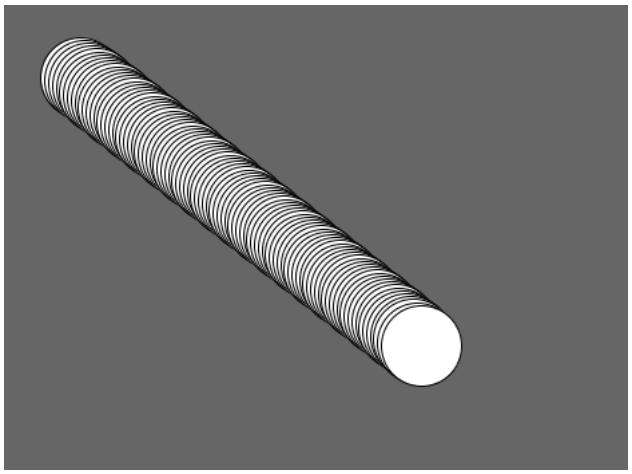

Figure 21, Background is not redrawn along with Ellipse

2.7 Interact with the Mouse

In figure 22 the code uses mouseX and mouseY to create a 9 x 9 ellipse at the mouse and the background updates before the ellipse is drawn giving the impression that the ellipse is moving along with the mouse.

```
void setup () {
size (480 , 360) ;
smooth () ;
noStroke () ;
fill (102) ;
}
void draw ()
{
background(204);
ellipse ( mouseX , mouseY , 9 , 9) ;
}
```

Figure 22, code of ellipse moving with mouse

2.8 Communication from Processing to Arduino

Task 2.12

The processing code (figure 23) sends the mouseX to the arduino code (figure 24) with the port.write command and the arduino code receives it with serial.read and sets the brightness to that.

```
import processing.serial.*;
Serial port ; // Create object from Serial class
int val ; // Data received from the serial port
void setup ()
{
  size (200 , 200) ;
  // Open the port that the board is connected to and use the same speed (9600 bps)
  port = new Serial (this,"COM3",9600);
}
void draw ()
{
  // draw a gradient from black to white
  for (int i = 0; i < 256; i ++)
  {
    stroke ( i ) ;
    line (i , 0 , i , 150) ;
   }
  // write the current X- position of the mouse to the serial port as
  // a single byte
  port.write(mouseX) ;
}
```

Figure 23, processing code for Task 2.12

```
const int ledPin = 9; // the pin that the LED is attached to
void setup ()
{
    // initialize the serial communication :
    Serial . begin (9600) ;
    // initialize the ledPin as an output :
    pinMode ( ledPin , OUTPUT ) ;
}
void loop ()
{
    byte brightness ;
    // check if data has been sent from the computer :
    if ( Serial . available () )
    {
        // read the most recent byte ( which will be from 0 to 255) :
        brightness = Serial . read () ;
        // set the brightness of the LED:
        analogWrite ( ledPin , brightness ) ;
    }
}
```

Figure 24, arduino code for Task 2.12

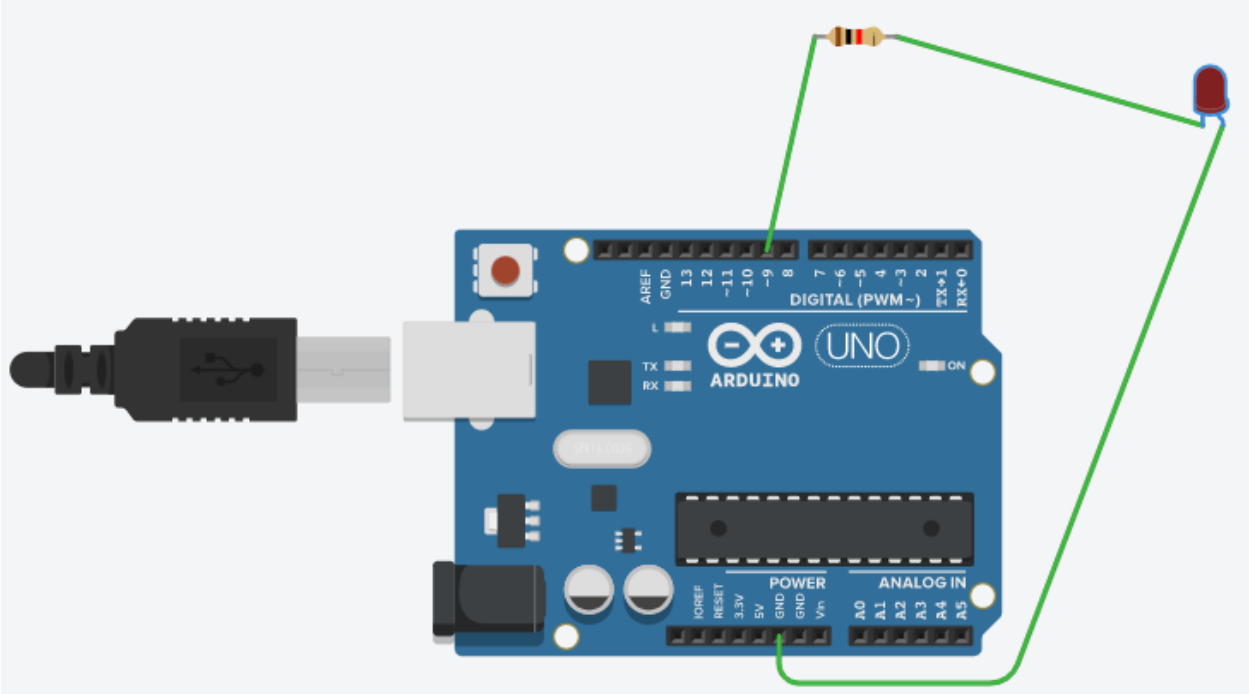

Figure 25, Circuit for Task 2.12