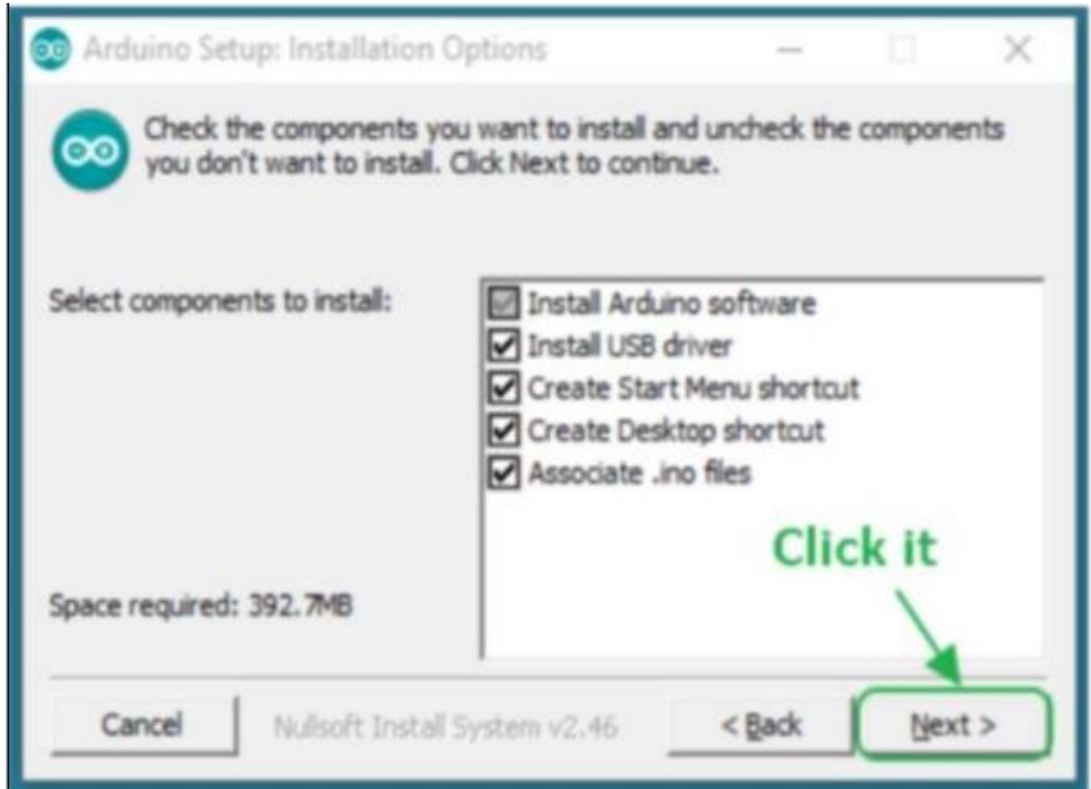


Experiment 1:

Study and Installation of Arduino IDE

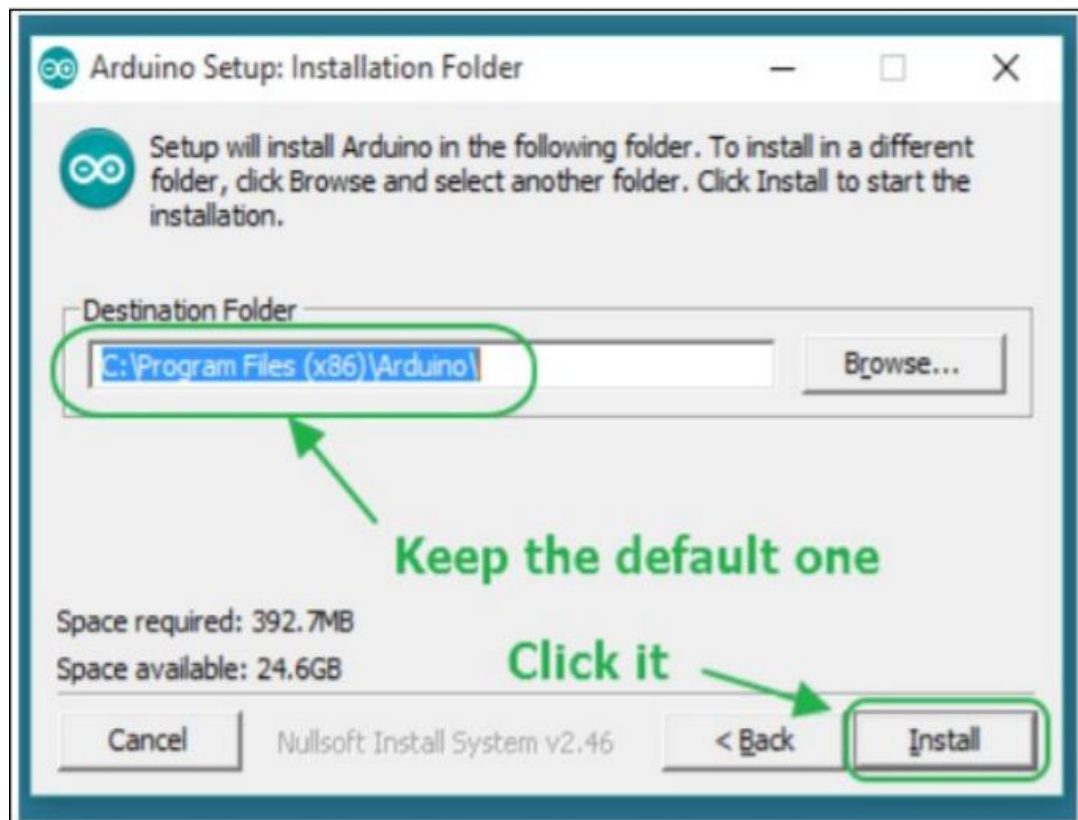
Step1: Downloading

To install the Arduino software, download this page: <http://arduino.cc/en/Main/Software> and proceed with the installation by allowing the driver installation process.



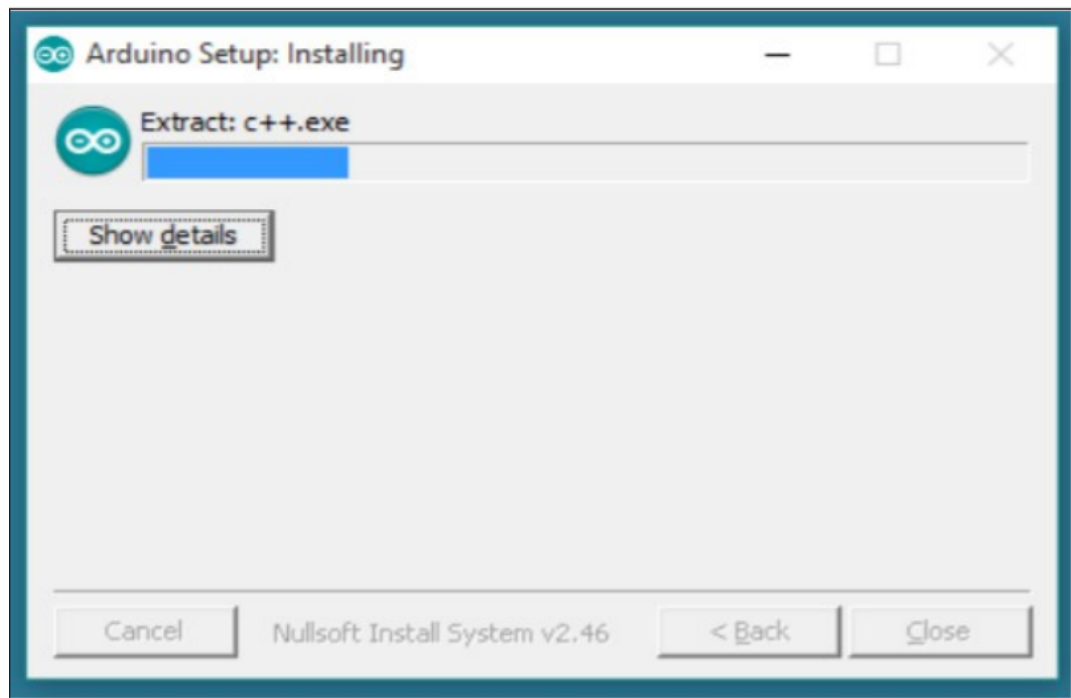
Step 2: Directory Installation

Choose the installation directory.



Step 3: Extraction of Files

The process will extract and install all the required files to execute properly the Arduino Software (IDE)

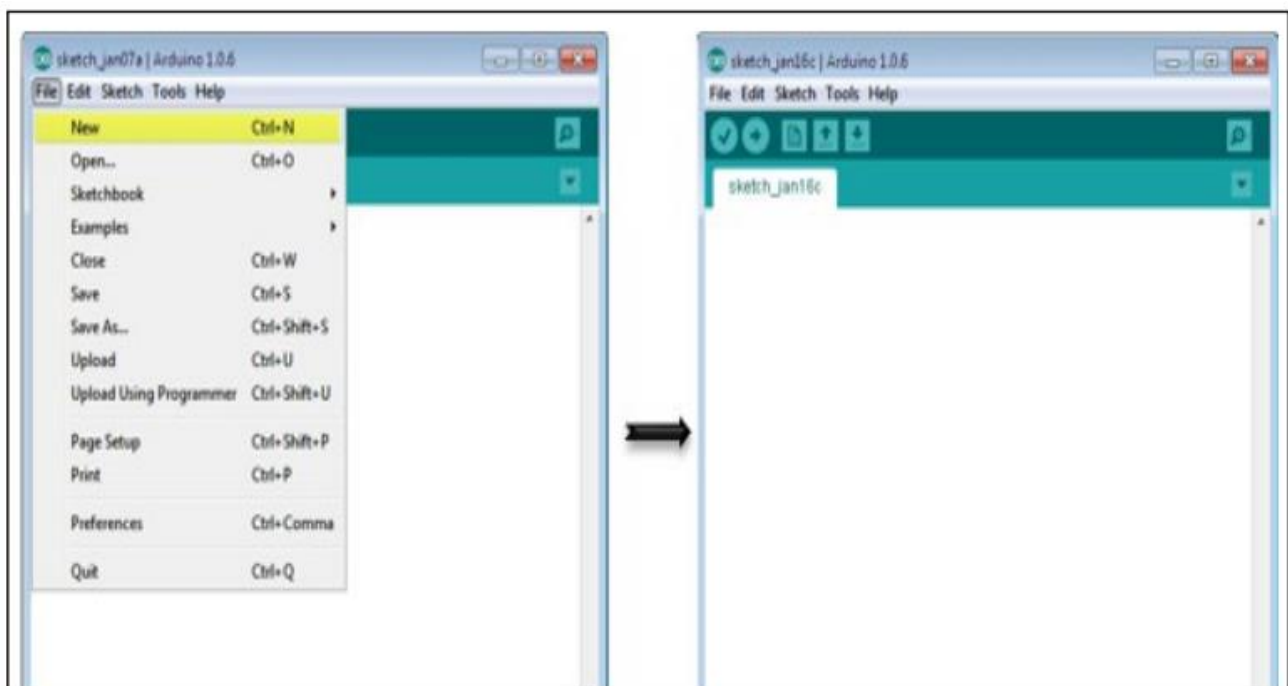


Step 4: Connecting the board

The USB connection with the PC is necessary to program the board and not just to power it up. The Uno and Mega automatically draw power from either the USB or an external power supply. Connect the board to the computer using the USB cable. The green power LED (labelled PWR) should go on.

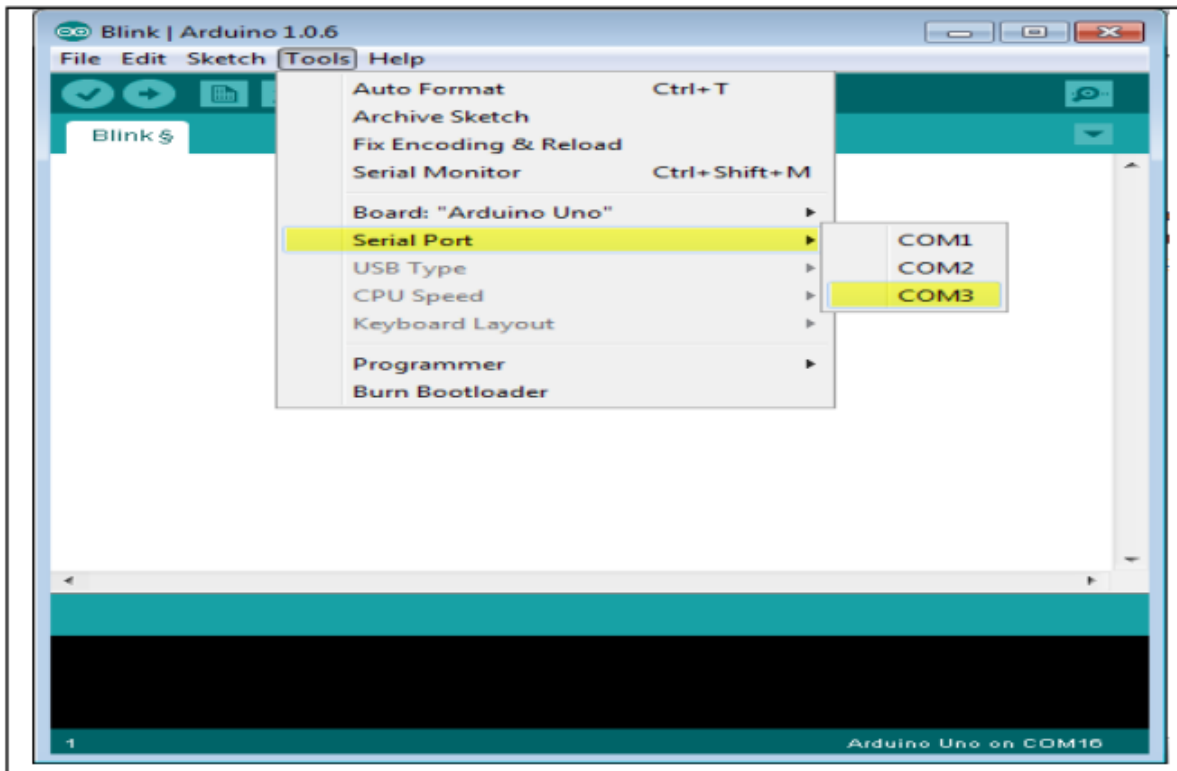
Step 5: Working on the new project

- Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit.
- Open a new sketch File



Step 6: Working on an existing project

To open an existing project example, select File → Example → Basics → Blink.



- **Step 9: Upload the program to your board.** Click the "Upload" button in the environment.
- Wait a few seconds; you will see the RX and TX LEDs on the board, flashing.
- If the upload is successful, the message "Done uploading" will appear in the status bar.

A	Verify
B	Upload
C	New
D	Open
E	Save
F	Serial Motor

Experiment 2:

Study and Installation of Tinkercad

Tinkercad is a web-based computer-aided design (CAD) software that allows users to create 3D models easily. Here are the steps to study and install Tinkercad:

Studying Tinkercad:

- **Explore Tinkercad's Official Website:** Visit Tinkercad's official website to get an overview of its features, capabilities, and applications.
- **Watch Tutorials:** Look for tutorials on platforms like YouTube or educational websites. Tinkercad provides its own set of tutorials for beginners.
- **Practice:** Start with simple projects to get acquainted with the interface and tools. Tinkercad offers a variety of sample projects to get started.
- **Join Communities:** Participate in forums, online communities, or social media groups related to Tinkercad. This is a great way to learn from experienced users, ask questions, and share your creations.
- **Read Documentation:** Familiarize yourself with Tinkercad's documentation, which includes guides, FAQs, and troubleshooting tips.

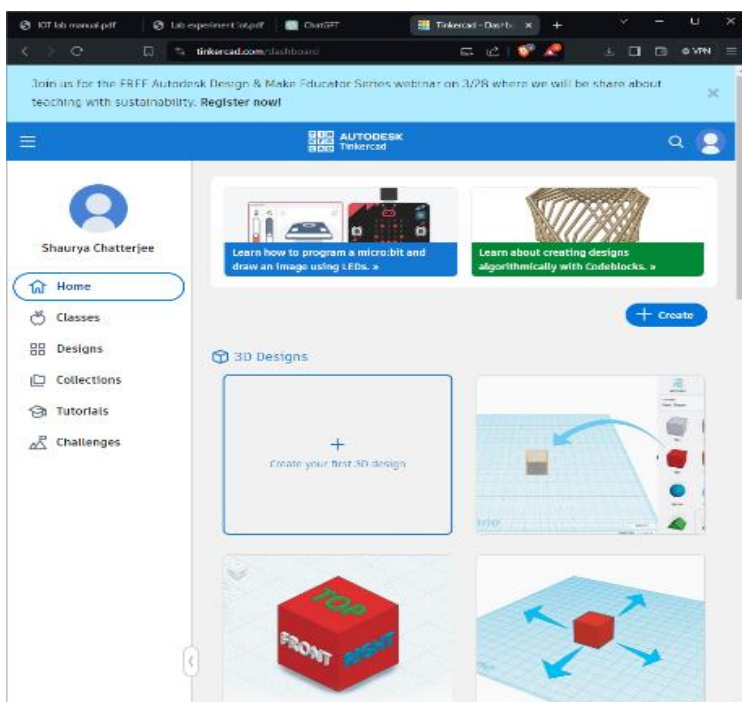
Installing Tinkercad:

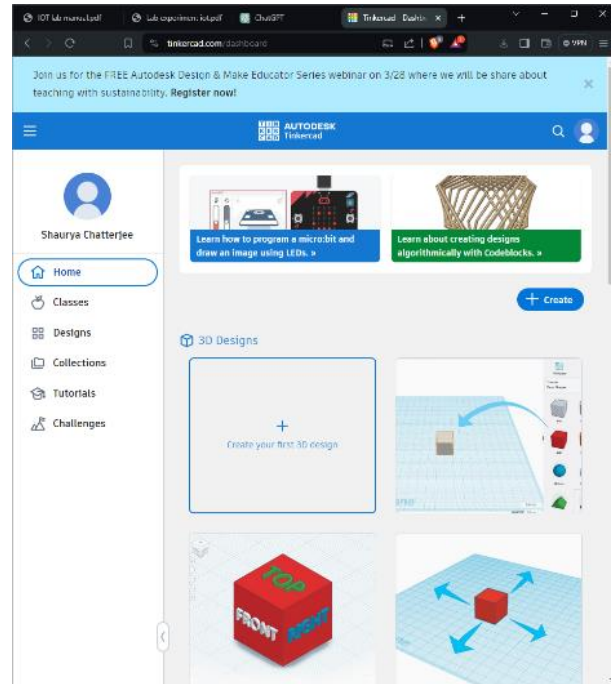
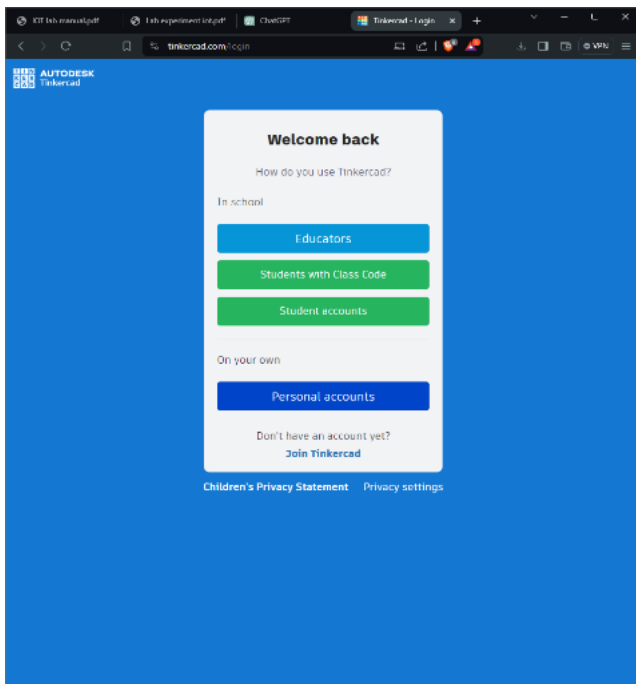
Tinkercad is a web-based application, so there's no need to download or install any software. However, you'll need a modern web browser with WebGL support.

System Requirements: Ensure your computer meets the system requirements to run Tinkercad smoothly. Generally, any modern computer with a standard web browser should suffice.

Create an Account: Visit the Tinkercad website and sign up for an account. You can sign up using your email address or through third-party services like Google or Facebook.

- **Login:** After creating an account, log in to Tinkercad using your credentials.
- **Start Designing:** Once logged in, you can start creating your 3D designs right away. Tinkercad provides a user-friendly interface with drag-and-drop functionality for easy designing.
- **Save Your Designs:** Remember to save your designs regularly to your Tinkercad account to avoid losing any work.
- **Export or Share:** After completing your design, you can export it in various formats or share it directly from Tinkercad.





Experiment 4:

Write a program using Tinkercad for RGB LED

Code:

```
const int redPin = 9;
const int greenPin = 10;
const int bluePin = 11;

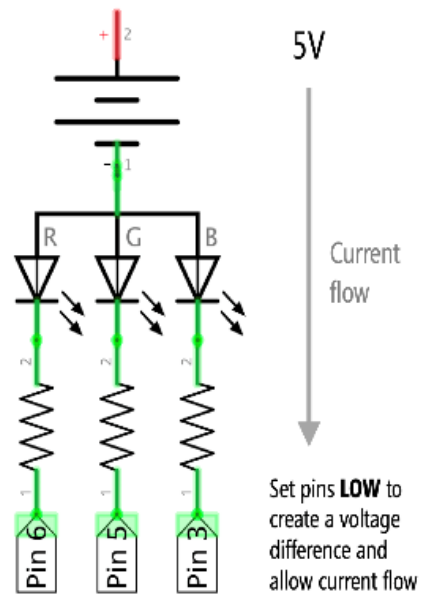
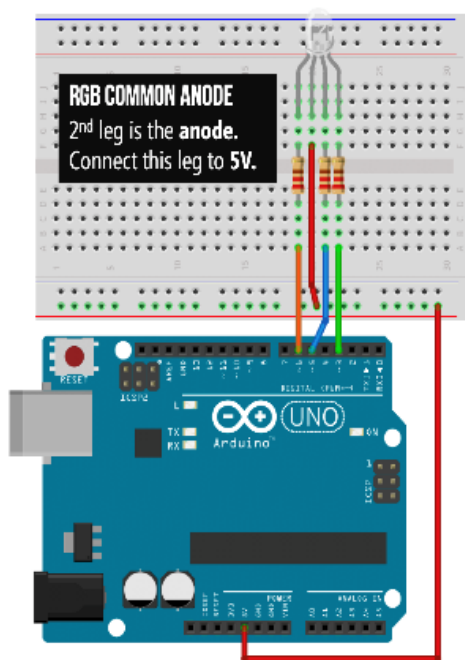
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  for (int i = 0; i < 255; i++) {
    analogWrite(redPin, i);
    delay(10);
  }
  for (int i = 255; i > 0; i--) {
    analogWrite(redPin, i);
    delay(10);
  }
  for (int i = 0; i < 255; i++) {
    analogWrite(greenPin, i);
    delay(10);
  }
  for (int i = 255; i > 0; i--) {
    analogWrite(greenPin, i);
    delay(10);
  }
  for (int i = 0; i < 255; i++) {
    analogWrite(bluePin, i);
    delay(10);
  }
  for (int i = 255; i > 0; i--) {
    analogWrite(bluePin, i);
```

```

delay(10);
}
}

```



Experiment 5:

Led control using push button in Arduino Board

Controlling an LED using a push button on an Arduino board is a common beginner project. Below are the steps to achieve this:

Hardware Required:

- Arduino Uno
- LED
- Push button
- Resistor (220 ohms recommended for the LED)
- Breadboard
- Jumper wires

Circuit Connections:

- Connect the longer leg (anode) of the LED to a digital pin on the Arduino (e.g., pin 13).
- Connect the shorter leg (cathode) of the LED to a current-limiting resistor (e.g., 220 ohms).
- Connect the other end of the resistor to the ground (GND) pin on the Arduino.
- Connect one leg of the push button to any digital pin on the Arduino (e.g., pin 2).
- Connect the other leg of the push button to the ground (GND) pin on the Arduino.
- Add a pull-up resistor (e.g., 10k ohms) between the digital pin connected to the push button and the 5V pin on the Arduino.

```
const int ledPin = 13;

const int buttonPin = 2;

int ledState = LOW;

int lastButtonState = LOW;

int buttonState;

unsigned long lastDebounceTime = 0;

unsigned long debounceDelay = 50;

void setup() {
  pinMode(ledPin, OUTPUT);

  pinMode(buttonPin, INPUT);

  digitalWrite(buttonPin, HIGH);
}

void loop() {
  int reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {

    lastDebounceTime = millis();

  }

  if ((millis() - lastDebounceTime) > debounceDelay) {

    if (reading != buttonState) {
```

```

        buttonState = reading;

    if (buttonState == LOW) {

        ledState = !ledState;

        digitalWrite(ledPin, ledState);

    }

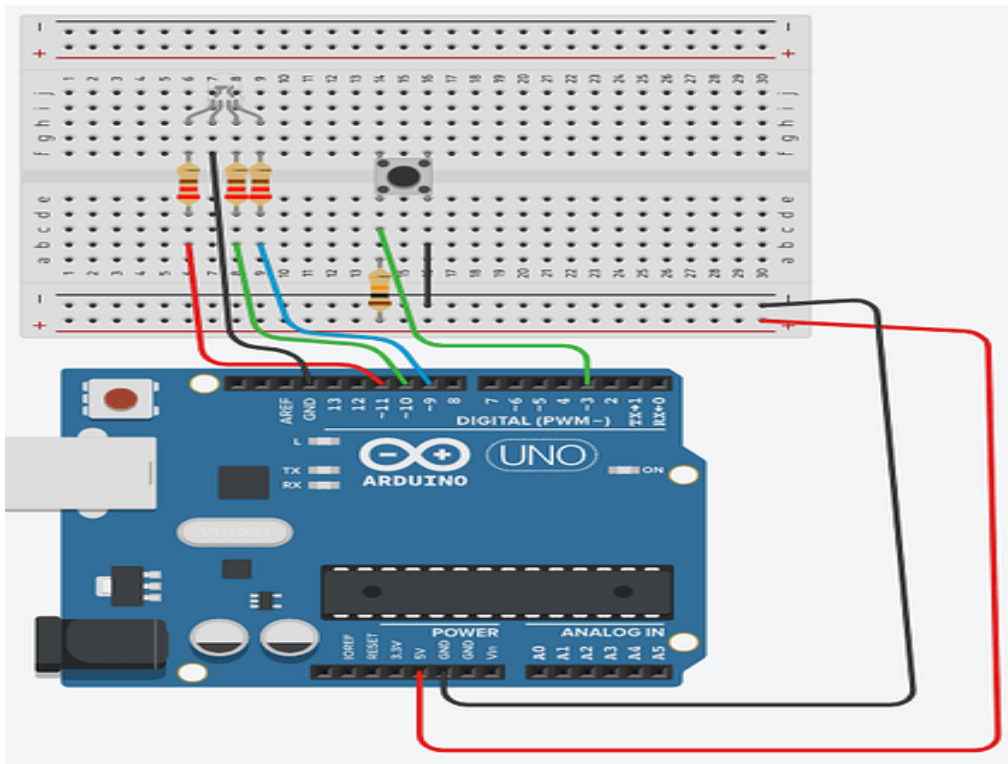
}

}

lastButtonState = reading;

}

```



Explanation:

The code sets up pin modes for the LED pin and push button pin.

It activates the internal pull-up resistor for the push button pin to ensure a stable reading when the button is not pressed.

The code implements debounce logic to ensure that a single button press is registered despite any electrical noise or bouncing that might occur when the button is pressed or released.

When the button is pressed, the LED state toggles.

Upload:

- Connect the Arduino board to your computer using a USB cable.
- Open the Arduino IDE.
- Copy and paste the provided code into a new sketch.
- Click on the "Upload" button to compile and upload the code to the Arduino board.

Testing:

Once uploaded, you should see the LED turning on and off each time you press the push button.

Experiment 6:

Sensor interfacing (distance sensor and buzzer)

Interfacing a distance sensor and a buzzer involves connecting the sensor to a microcontroller, reading its output, and triggering the buzzer based on certain conditions (such as detecting an object within a certain range). Below, I'll outline a basic example using an ultrasonic distance sensor (like the HC-SR04) and a buzzer interfaced with an Arduino microcontroller.

Components Needed:

- Arduino board (e.g., Arduino Uno)
- Ultrasonic distance sensor (e.g., HC-SR04)
- Buzzer
- Jumper wires

Circuit Wiring:

- Connect VCC of the distance sensor to 5V on the Arduino.
- Connect GND of the distance sensor to GND on the Arduino.
- Connect Trig pin of the distance sensor to a digital pin (e.g., Pin 7) on the Arduino.
- Connect Echo pin of the distance sensor to another digital pin (e.g., Pin 6) on the Arduino.
- Connect one terminal of the buzzer to a digital pin (e.g., Pin 8) on the Arduino.
- Connect the other terminal of the buzzer to GND on the Arduino.

Code:

```
const int trigPin = 7;

const int echoPin = 6;

const int buzzerPin = 8;

long duration;

int distance;

int threshold = 20;

void setup() {

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(buzzerPin, OUTPUT);

  Serial.begin(9600); // Initialize serial communication
}

void loop() {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
```

```
distance = duration * 0.034 / 2; // Convert time to distance in cm
```

```
Serial.print("Distance: ");
```

```
Serial.print(distance);
```

```
Serial.println(" cm");
```

```
if (distance < threshold) {
```

```
  digitalWrite(buzzerPin, HIGH);
```

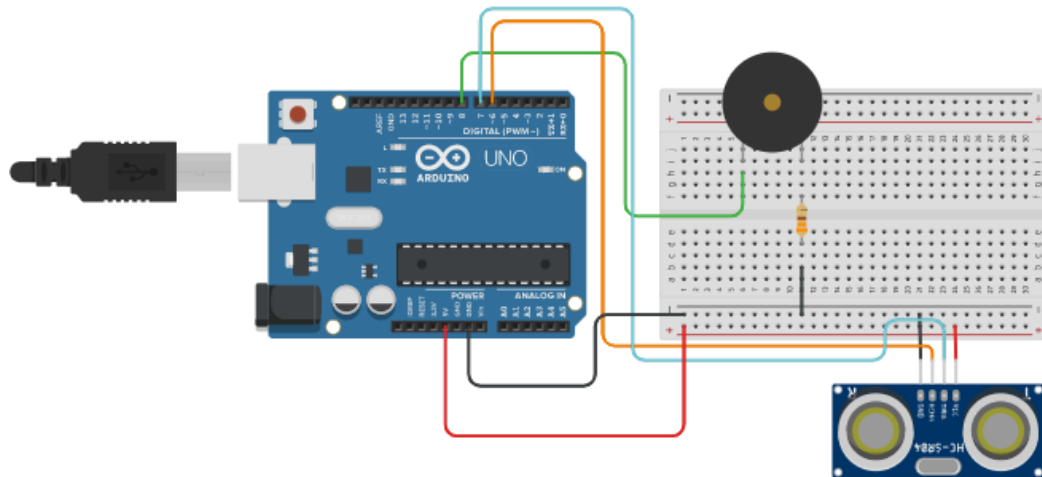
```
} else {
```

```
  digitalWrite(buzzerPin, LOW);
```

```
}
```

```
delay(100);
```

```
}
```



Explanation:

- The code continuously measures the distance using the ultrasonic sensor and calculates it based on the time taken for the signal to bounce back.
- If the measured distance is less than the threshold (in this example, 20 cm), it turns on the buzzer; otherwise, it turns it off.
- The serial monitor can be used to view the measured distance for debugging purposes.

Note: Ensure that you've installed the necessary Arduino libraries for interfacing with the ultrasonic sensor if you're using a different model than the HC-SR04. Additionally, remember to upload the code to your Arduino board after wiring the components properly.

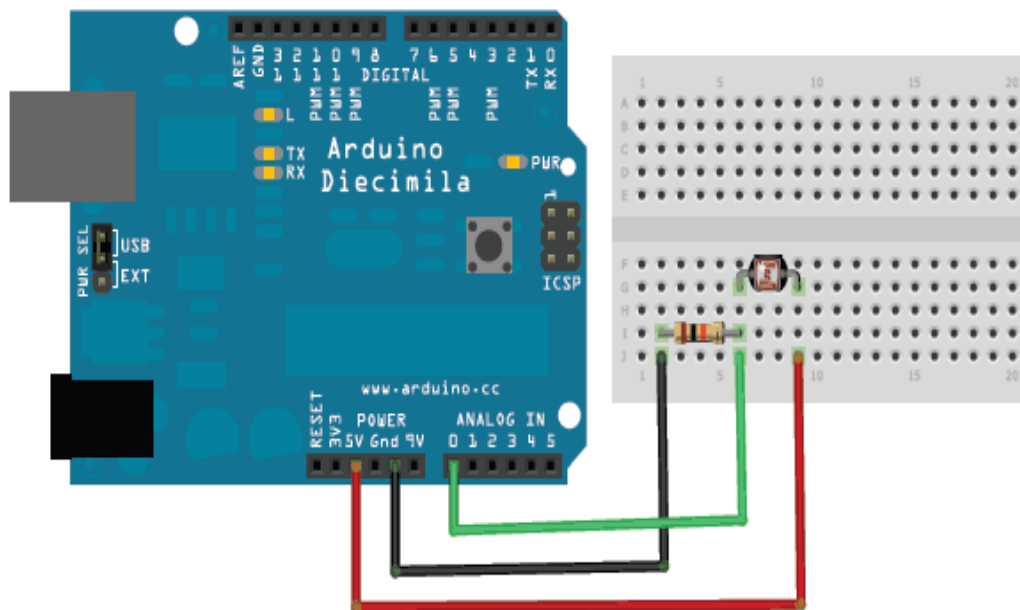
Experiment No: 7

Detection of the light using photo resistor

Hardware Required:

1. Arduino Uno
2. Photoresistor
3. 10k ohm resistor
4. Breadboard
5. Jumper wires

Connection Diagram:



Code:

```
const int photoresistorPin = A0;

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

void loop() {
  int lightLevel = analogRead(photoresistorPin);

  Serial.print("Light Level: ");
  Serial.println(lightLevel);

  delay(500);
}
```

Output:

With Light On:

Light Level: 712

Light Level: 689

Light Level: 654

Light Level: 630

With Light Off:

Light Level: 30

Light Level: 27

Light Level: 25

Light Level: 23

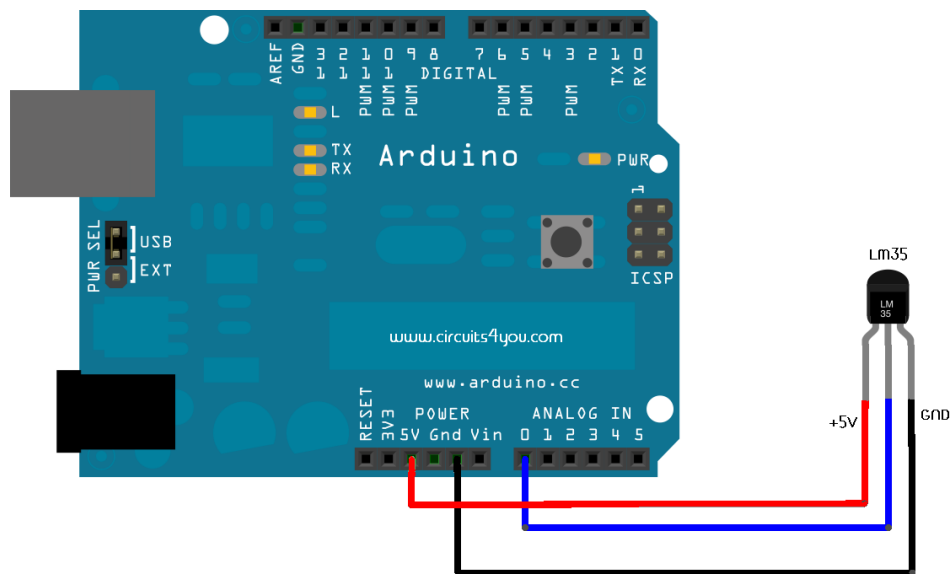
Experiment No: 8

Interfacing of temperature sensor with Arduino

Hardware Required:

1. Arduino Uno
2. LM35 temperature sensor
3. Jumper wires
4. Breadboard

Connection Diagram:



Code:

```
const int lm35Pin = A0;

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

void loop() {
    int sensorValue = analogRead(lm35Pin);
    float temperature = (sensorValue * 0.48875); // LM35 has a sensitivity of 10mV/°C
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println("°C");
    delay(1000);
}
```

Output:

Temperature: 23.50°C

Temperature: 23.52°C

Temperature: 23.48°C

Temperature: 23.55°C

Temperature: 23.49°C

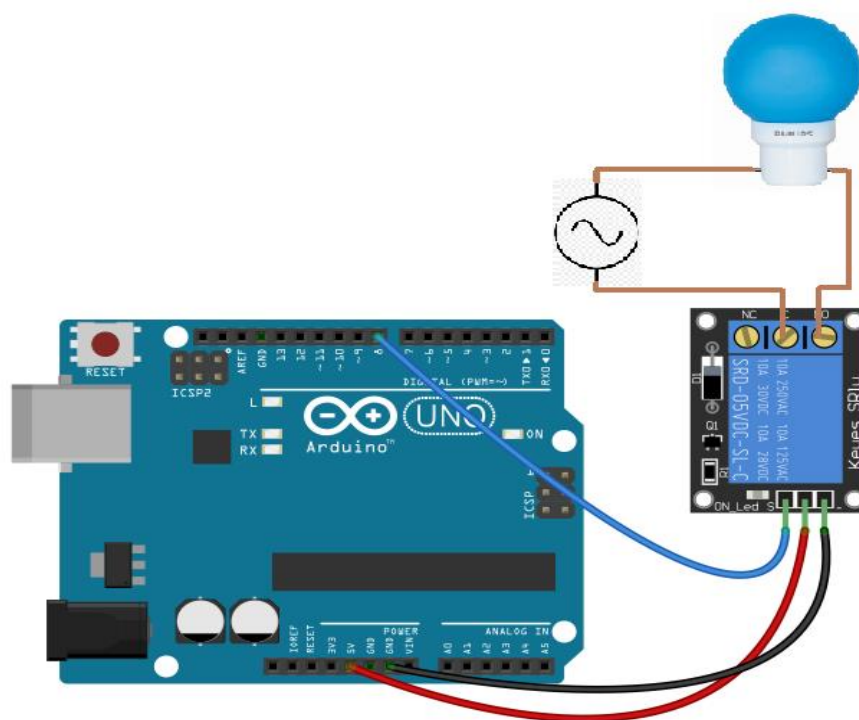
Experiment No: 9

Interfacing of the Relay with Arduino.

Hardware Required:

1. Arduino Uno
2. Relay module
3. Jumper wires
4. Power source

Connection Diagram:



Code:

```
#define RELAY_PIN 2

void setup() {
  pinMode(RELAY_PIN, OUTPUT);
}

void loop() {
  digitalWrite(RELAY_PIN, HIGH);
  delay(2000);
  digitalWrite(RELAY_PIN, LOW);
  delay(2000);
}
```

Output:

Sr. No.	Delay	Relay Status
1	2s	ON
2	2s	OFF

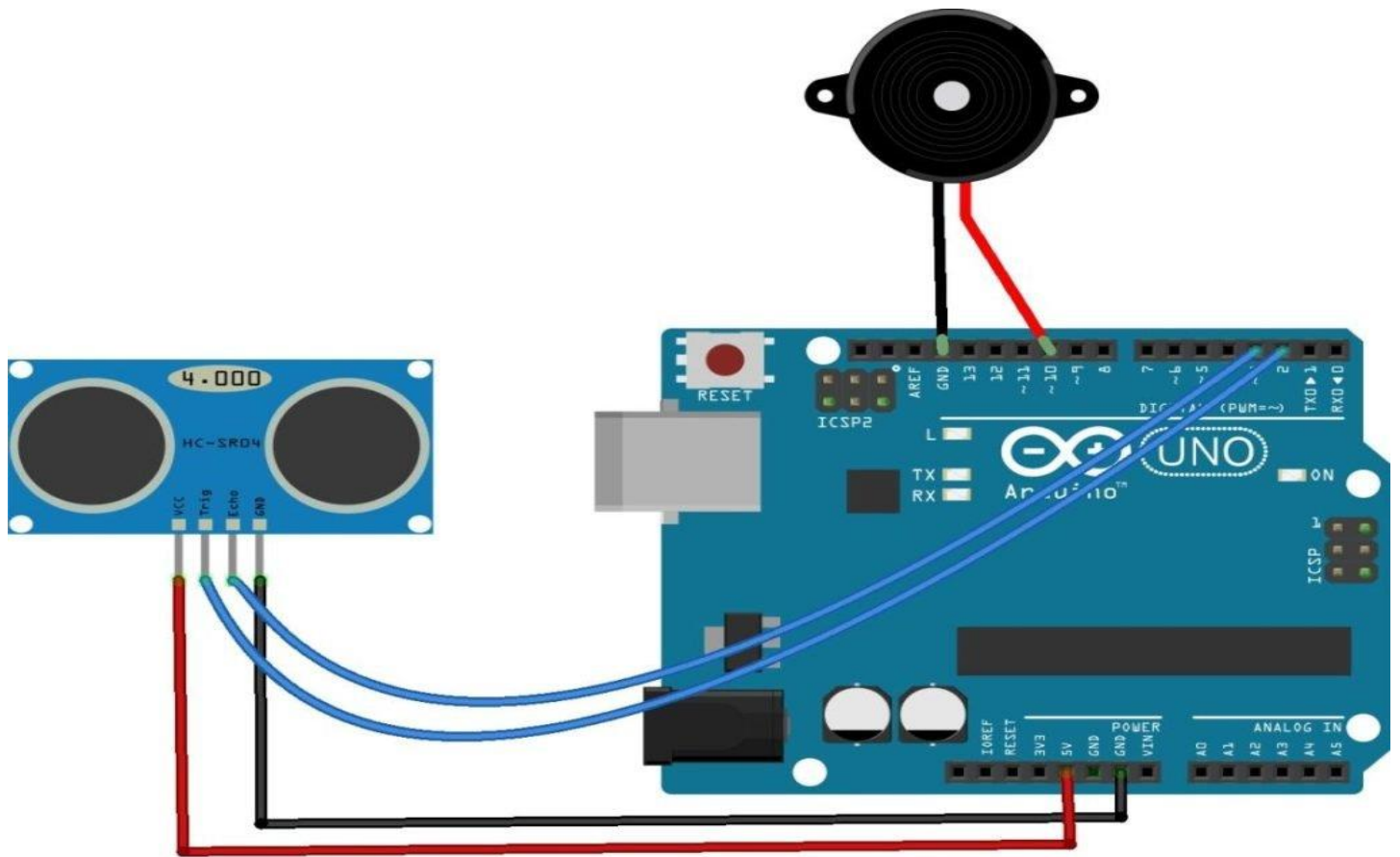
Experiment No: 10

Building Intrusion Detection System with Arduino and Ultrasonic Sensor

Hardware Required:

1. Arduino Uno
2. Ultrasonic sensor (HC-SR04)
3. Buzzer or LED for alarm indication
4. Breadboard and jumper wires for connections

Connection Diagram:



Code:

```
const int trigPin = 7;
const int echoPin = 8;
const int alarmPin = 9;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(alarmPin, OUTPUT);
}
```

```
void loop() {  
  
    long duration, distance;  
  
    digitalWrite(trigPin, LOW);  
  
    delayMicroseconds(2);  
  
    digitalWrite(trigPin, HIGH);  
  
    delayMicroseconds(10);  
  
    digitalWrite(trigPin, LOW);  
  
    duration = pulseIn(echoPin, HIGH);  
  
    distance = duration * 0.034 / 2;  
  
    Serial.print("Distance: ");  
  
    Serial.println(distance);  
  
    if (distance < 100) {  
  
        digitalWrite(alarmPin, HIGH);  
  
    } else {  
  
        digitalWrite(alarmPin, LOW);  
  
    }  
  
  
  
    delay(1000); // Delay for stability  
  
}
```

Output:

Distance: 50

Distance: 60

Distance: 90

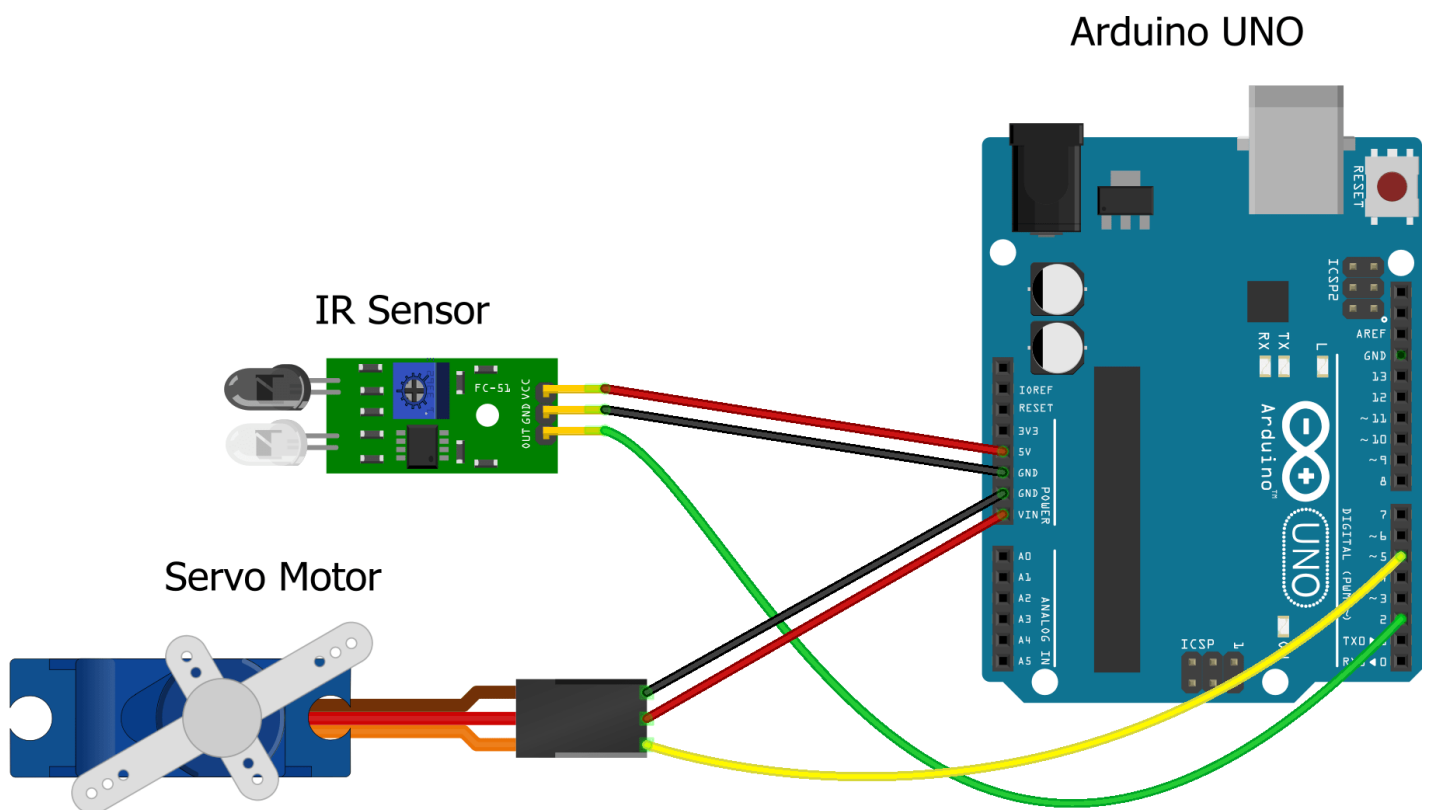
Experiment No: 11

Actuator interfacing (IR sensor, Servo motor)

Hardware Required:

1. Arduino Uno
2. IR sensor module
3. Servo motor
4. Jumper wires
5. Breadboard

Connection Diagram:



Code:

```
#include <Servo.h>

Servo myservo;

int IR_PIN = 2;

int val;

void setup() {
  myservo.attach(9);
  Serial.begin(9600);
}

void loop() {
```

```
val = digitalRead(IR_PIN);  
if (val == LOW) {  
    myservo.write(0);  
    delay(1000);  
} else {  
    myservo.write(90);  
    delay(1000);  
}  
}
```

Output:

No obstacle detected. Servo rotated to 90 degrees.

Obstacle detected! Servo rotated to 0 degrees.

No obstacle detected. Servo rotated to 90 degrees.

No obstacle detected. Servo rotated to 90 degrees.

Obstacle detected! Servo rotated to 0 degrees.

No obstacle detected. Servo rotated to 90 degrees.

Experiment No: 12

Sending message using MQTT from broker to server by node red and using Mosquitto

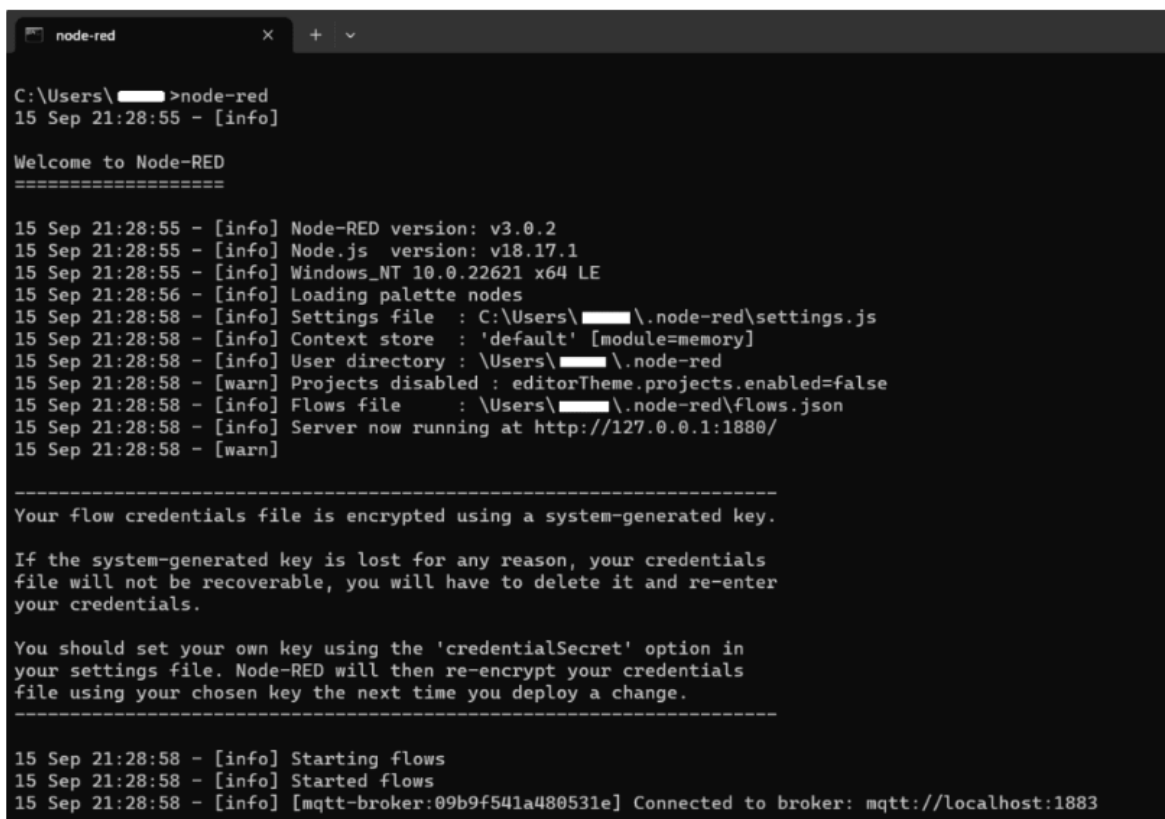
Installing Node-RED

Now, I will demonstrate the installation of Node-RED by running it locally from a Windows PC. The following instructions will assume you are using a Windows operating system.

Node-RED is available for different platforms and operating systems. Here, you can find a comprehensive list of supported environments with setup instructions.

Node-RED requires Node.js to work. Therefore, download this first tool from [here](#).

Once the installation completes, it is possible to verify the procedure's correctness by typing the following command into a Powershell console:



```
C:\Users\>node-red
15 Sep 21:28:55 - [info]

Welcome to Node-RED
=====

15 Sep 21:28:55 - [info] Node-RED version: v3.0.2
15 Sep 21:28:55 - [info] Node.js version: v18.17.1
15 Sep 21:28:55 - [info] Windows_NT 10.0.22621 x64 LE
15 Sep 21:28:56 - [info] Loading palette nodes
15 Sep 21:28:58 - [info] Settings file : C:\Users\...\node-red\settings.js
15 Sep 21:28:58 - [info] Context store : 'default' [module=memory]
15 Sep 21:28:58 - [info] User directory : \Users\...\node-red
15 Sep 21:28:58 - [warn] Projects disabled : editorTheme.projects.enabled=false
15 Sep 21:28:58 - [info] Flows file : \Users\...\node-red\flows.json
15 Sep 21:28:58 - [info] Server now running at http://127.0.0.1:1880/
15 Sep 21:28:58 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

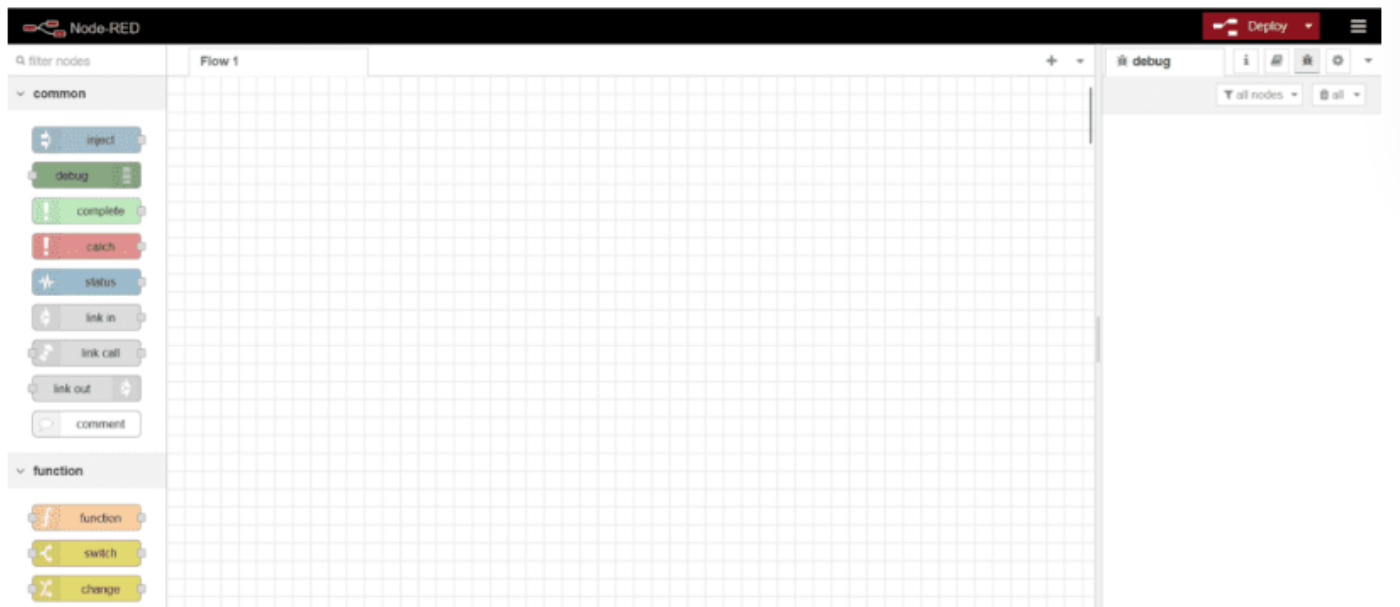
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

15 Sep 21:28:58 - [info] Starting flows
15 Sep 21:28:58 - [info] Started flows
15 Sep 21:28:58 - [info] [mqtt-broker:09b9f541a480531e] Connected to broker: mqtt://localhost:1883
```

The console will stay open with Node-RED now running. If the console closes, this will also terminate the Node-RED process.

While this is the default “start-up” instruction provided in the official tutorial, running it at start-up or as a service is still possible using third-party tools.

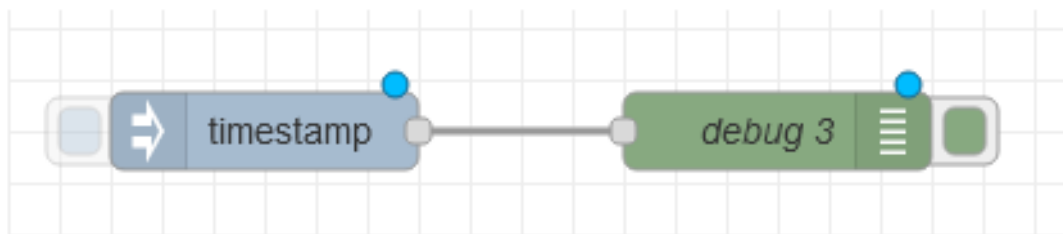


This configuration panel consists of three parts:

- A grid in the middle where I can design my project.
- A menu panel on the left with a list of objects, called nodes, which I can drag and drop on the grid to design my configuration.
- And an information panel on the right, which holds the debug panel, among others.

Node-RED offers many kinds of nodes for implementing different functionalities or system resources.

Nodes exchange messages comprising a topic and a payload, carrying the message information.



Preparing the MQTT broker infrastructure

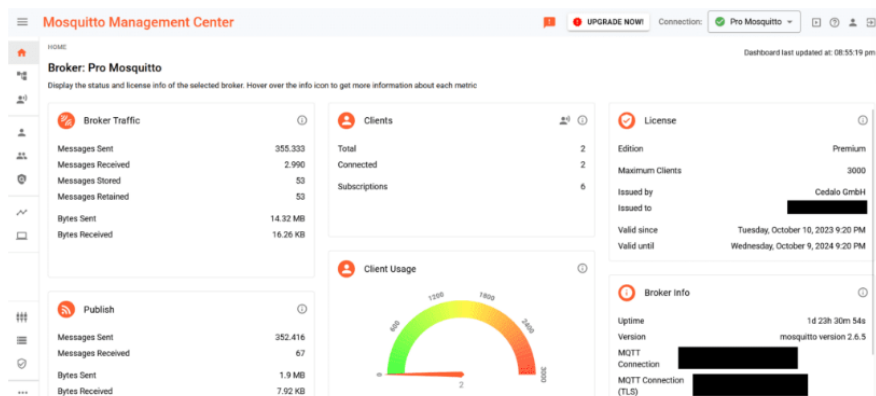
After this first introduction to Node-RED, I can proceed with configuring the MQTT infrastructure. In this article, I used the free 14-day cloud trial of Pro Edition for Eclipse Mosquitto. However, a 30-day on-premises trial version with basic MQTT HA is also available. It grants access to advanced features such as enhanced security, management APIs, or a high-performance local persistence database.

To set up a Pro Mosquitto broker, I must enter my email address and the server location where I would host my broker. There are several options to choose from.

Complete the form by clicking the “Submit” button and wait for a confirmation mail. The Pro Mosquitto setup procedure will begin upon confirming the mail address – requiring 10 to 15 minutes.

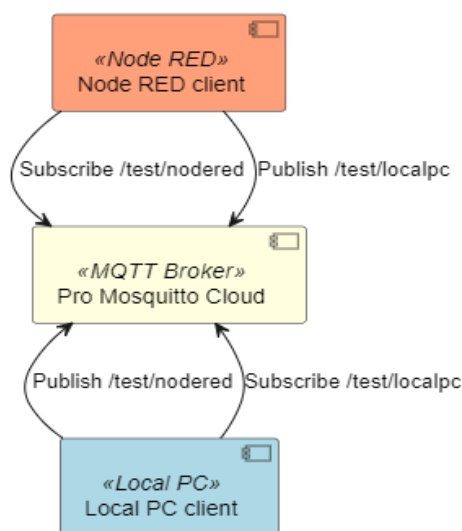
I will receive a second email once the configuration is complete (Figure 6). It will contain important information like the MQTT broker address or the credentials required to access the private configuration area of Mosquitto, called Mosquitto Management Center (MMC).

How to set up clients in Mosquitto Management Center (MMC)



This is the central control dashboard where it is possible to monitor relevant information, like the number of connected clients, exchanged messages, etc. Another essential functionality accessible from this dashboard is “Clients creation,” which I will use in my example.

To demonstrate the Node-RED integration with Mosquitto, I will create two clients: one for Node-RED, named “nodered”, and one used locally from my PC, named “localpc” – where a publisher and a subscriber are located.



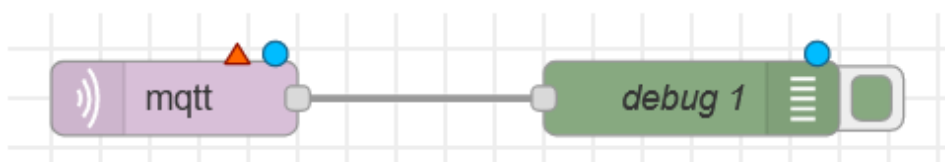
For a detailed guide on creating clients, refer to Cedalo’s extended guides [like this one](#).

Once created, the two clients are visible on the **Clients** page.

Name	ID	Text Name	Description	Groups	Roles
admin		Admin user			super-admin, read-clients, read-brokers
localpc					client
nodered					client

Connecting Node-RED to the MQTT broker

With everything now set up to connect with the MQTT broker, open the Node-RED dashboard as demonstrated in the chapter “Starting the Node-RED environment.” Drag an “MQTT in” node and a “Debug” node into the canvas, as shown in the following figure, and make sure to connect the two nodes.



The “MQTT in” node receives messages from a broker, much like a subscribing client. The “Debug” node enables printing all the received data from the “MQTT in” node in a debug window, demonstrating correct data reception.

Properties

Server: [Redacted] ✓

Action: Subscribe to single topic ▼

Topic: /test/nodered

QoS: 2 ▼

Output: auto-detect (parsed JSON object, string or buffer) ▼

Name: Name

The “MQTT out” node publishes messages over a specific topic, while the “Inject” node creates and sends data to other nodes. In this case, the “Inject” node will cyclically send the current timestamp as an output every second. This output will then be published by the “MQTT-out” node, as shown in Figure 14.

Now, I can double-click on the “MQTT-out” node and perform the configuration, which will almost be identical to the “MQTT-in” node configuration. (Therefore, the specific step-by-step description will be omitted here). The only difference is the topic where the node will publish the message, which is set to /test/localpc in this case.

Properties

Name: Name

msg. payload = timestamp ✕

msg. topic = a_2 ✕

+ add inject now

☒ Inject once after 0.1 seconds, then

Repeat: interval ▼

every 1 seconds ▼

Test the Node-RED configuration

Node-RED is now configured to act as a subscriber and publisher. However, my setup is still not running: I must click the “Deploy” button in the upper-right corner of the Node-RED console window to activate my design.

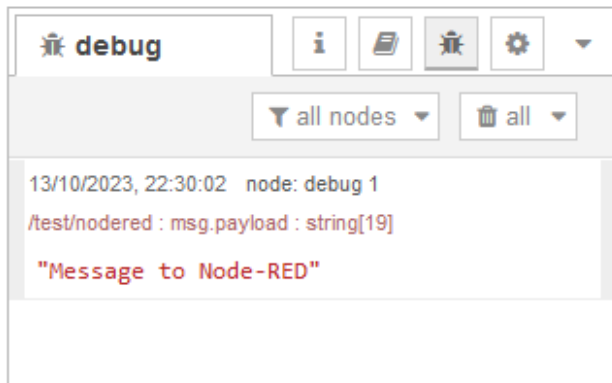
Once the Node-RED configuration runs, I can create a local publisher and subscriber hosted by my PC, which will interact with Node-RED.

I can start by testing the “MQTT-in” node operation, which means I will try to publish a message from a local publisher client to the Node-RED client. I will create a publisher client using the MQTT.js client library through the following command into a Powershell console:

```
npx mqtt pub -h hg13dtohpgob-tvqx2k3tdgzf.cedalo.dev -t /test/nodered -m  
"Message to Node-RED" -u localpc -P testpwd
```

If the library is not installed on your machine, the packet manager will automatically prompt a request for installation. Another possibility would be to use the client utilities provided by the open-source version of Mosquitto, for which professional technical support is available.

This command will publish a message on the topic `/test/nodered`, logging as the `localpc` client. The message will reach the Node-RED “MQTT-in” node and display it on the debug console, courtesy of the debug node.



Now, let's test the second part of my configuration to verify the “MQTT-out” node effectively publishes the injected data.

I will create a subscriber client on the PC using the MQTT.js client library with the following command:

```
npx mqtt sub -h hg13dtohpgob-tvqx2k3tdgzf.cedalo.dev -t /test/localpc -u  
localpc -P testpwd
```

The subscriber will subscribe to the specified topic and receive the data published from Node-RED, then depict a list of published timestamps (Figure 17).



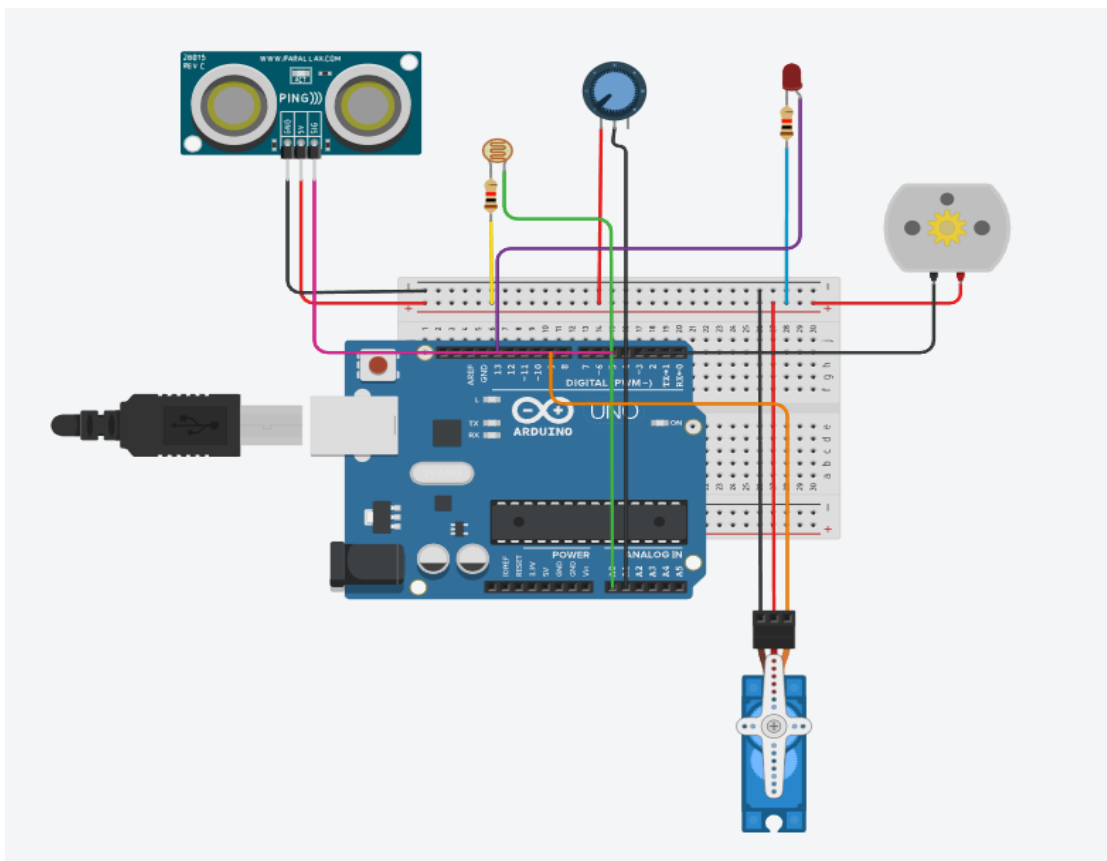
Experiment No: 3

How to write a program using Tinkercad for different electronic components

Hardware Required:

1. Arduino UNO
2. LED
3. Servo Motor
4. LDR
5. DC Motor
6. Ultrasonic Sensor
7. Potentiometer
8. Breadboard
9. Jumper wires

Connection Diagram:



Code:

```
#include <Servo.h>

Servo myservo;

int ledPin = 2;

int servoPin = 9;
```

```
int ldrPin = A0;

int motorPin = 3;

int trigPin = 11;

int echoPin = 12;

int potPin = A1;

int distance = 0;

int brightness = 0;

void setup() {

  pinMode(ledPin, OUTPUT);

  pinMode(servoPin, OUTPUT);

  pinMode(motorPin, OUTPUT);

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  Serial.begin(9600);

  myservo.attach(servoPin);

}

void loop() {

  int ldrValue = analogRead(ldrPin);

  if (ldrValue < 500) {

    digitalWrite(ledPin, HIGH); // turn the LED on

  } else {

    digitalWrite(ledPin, LOW); // turn the LED off

  }

  int potValue = analogRead(potPin);

  int servoAngle = map(potValue, 0, 1023, 0, 180);

  myservo.write(servoAngle);

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);

  distance= duration*0.034/2;

  if(distance < 10) {

    analogWrite(motorPin, 255); // turn the motor on at full speed
```

```
} else {  
    analogWrite(motorPin, 0); // turn the motor off  
}  
  
brightness = analogRead(potPin);  
brightness = map(brightness, 0, 1023, 0, 255);  
Serial.print("Brightness: ");  
Serial.println(brightness);  
delay(1000);  
}
```