Lebanese University

Faculty of Engineering

Electrical and Electronic Dep.

RL4Eng

Development of Remote and Virtual Laboratories for Teaching and Training Engineering Students in the South Mediterranean and Sub-Saharan Higher Education Institutions

TRAINING MATERIAL ASSIGNMENTS

COMMUNICATION AND MINI-PROJECT (SHS307)

LUTHL4EEE

Editors:

Abed Ellatif Samhat
Rida EL Chall
Hussein Choueib
Mohammad Hammoud
Rasha Hammoud
Faten Mortada
Kassem Elhajj
Mohammad Karaki
Ali Farhat
Mostafa Roumiyeh
Deema Azzam
 Mariam Farhat
Aya hassan
Amar farshoukh
Nour Mourtada
Zahraa Tarhini
Ahmad Abu Zainab
Roaa Darwich
Fatima Moselmani
Lara Ibrahim
Hussien Rammal
Mahdi Abou Hamdan,
Fatima Ismail
Zaynab Baalbaki
Jad Ayach
Assaad Taleb
Ali Makki
Mohmmad Sweidan
Hassan Khrayzat
Batoul Abbass
Shebli Rakka
Hussein Ahmad
Fadel Morda
Mhamad Rayed
Kawthar berjawi
Houssein Darwish
Mohammad Atwi
Ali Rahme
Alaa Yehia
Ali Jaafar
Rokaya Alameen
Fatima Atwi
Mohamed Sabaayoun

ASSIGNMENT #1: COLOR MIXING USING RGB LED

1. Introduction

In recent years, the integration of RGB (Red, Green, Blue) LEDs has revolutionized the field of lighting and visual display technology. These compact, energy-efficient devices offer a spectrum of colors by varying the intensity of each primary color component. From decorative lighting to advanced digital displays, RGB LEDs have found extensive applications across various domains.

Our experiment aims to explore the color-mixing capabilities of RGB LEDs, endeavoring to generate a diverse array of new colors and shades. Our experimental design entails a sequential process, initiating with the LED emitting red light, followed by a gradual transition to green, then blue, and concluding with a return to the red color state. This cyclic progression facilitates the demonstration of a wide spectrum of hues and shades achievable through the manipulation of RGB components. By systematically cycling through these color transitions, our experiment seeks to showcase the extensive range of colors attainable, thereby enriching our understanding of RGB LED functionality and its potential applications.

2. Required components

a. (1) x Uno R3 Board

Our chosen microcontroller, used for program execution and signal control.

b. (1) x 830 Tie Points Breadboard

The platform used for circuit prototyping and component connection without the need to solder the connections.

c. (4) x Male to Male jumper wires

Conductive links for establishing electrical connections between components.

d. (1) x RGB LED

Light-emitting diode capable of emitting red, green, and blue light.

e. (3) x 220 ohm resistors

Components used to regulate current flow and protect the RGB LED from excess voltage.

3. Implementation Steps

a. Connection diagram

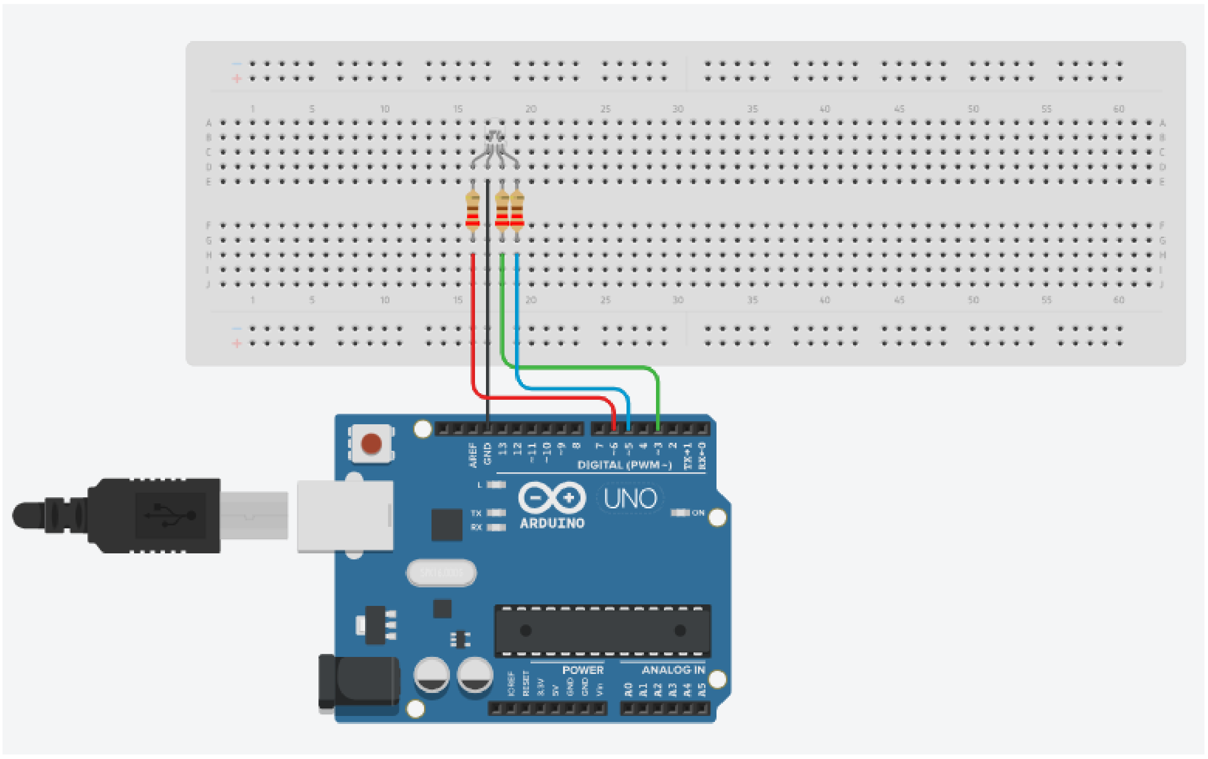


Figure 1: *Connection Diagram*

b. Code

```
#define GREEN_PIN 3
#define BLUE_PIN 5
#define RED_PIN 6

#define delayTime 10

void setup()
{
pinMode(RED_PIN, OUTPUT);
pinMode(GREEN_PIN, OUTPUT);
pinMode(BLUE_PIN, OUTPUT);
}
```

```
int redValue;
int greenValue;
int blueValue;

void loop()
{
  redValue = 255;
  greenValue = 0;
  blueValue = 0;

  for(int i = 0; i < 255; i++)
  {
  analogWrite(RED_PIN, redValue);
  analogWrite(GREEN_PIN, greenValue);
  redValue--;
  greenValue++;
  delay(delayTime);
  }

  for(int i = 0; i < 255; i++)
  {
  analogWrite(GREEN_PIN, greenValue);
  analogWrite(BLUE_PIN, blueValue);
  greenValue--;
  blueValue++;
  delay(delayTime);
  }

  for(int i = 0; i < 255; i++)
  {
  analogWrite(BLUE_PIN, blueValue);
  analogWrite(RED_PIN, redValue);
  blueValue--;
  redValue++;
  delay(delayTime);
  }
}
```

4. Results

We effectively blended RGB colors, adjusting their intensity to produce various shades. The transitions between these hues were seamless, facilitated by the implemented code. Additionally, a comprehensive simulation video accompanies this report.

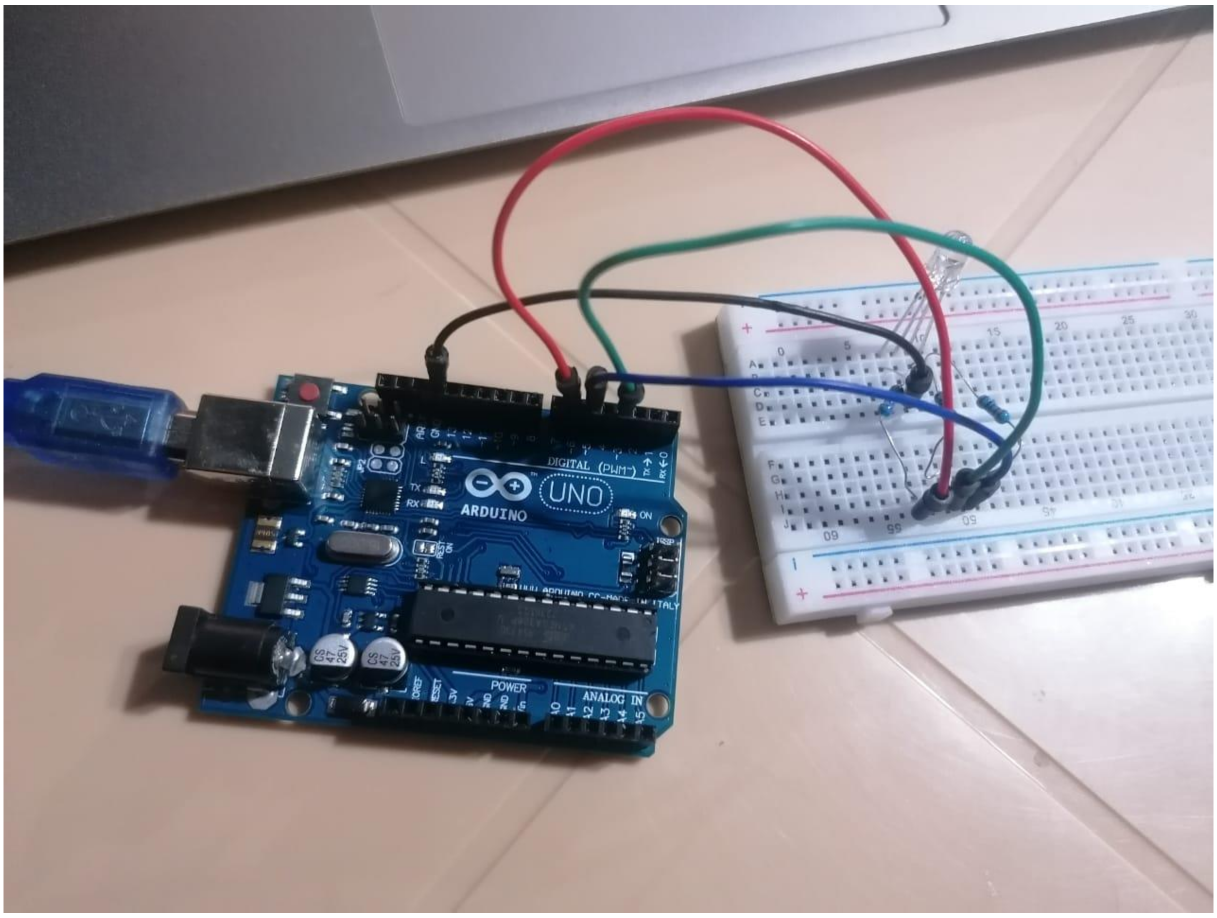


Figure 2: *Real Wiring Diagram*

5. Conclusion

The RGB LED included in the kit simplifies and streamlines the process of blending colors, yielding a spectrum of shades with practical applications in lighting and visual display. It is imperative to pair this LED with suitable resistances to regulate the current and safeguard it from damage during operation.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] TinkerCad, https://www.tinkercad.com/

ASSIGNMET #2: DIGITAL INPUTS

1. Introduction

In the world of electronics and microcontrollers, Arduino provides an accessible platform for learning and experimentation. This project focuses on exploring the basics of digital inputs and outputs by using push buttons to control an LED. Push buttons, common in electronic interfaces, provide a straightforward means of user interaction.

Our objective is to understand digital input processing in Arduino, integrate push buttons into a circuit to toggle the LED state, and develop code to interpret button presses and control the LED accordingly. Through this project, we aim to gain practical experience in Arduino programming and circuit design while highlighting the interaction between physical inputs and digital outputs.

2. Required components

a. (1) x Uno R3 Board

The heart of the project, It provides the necessary processing power and input/output interfaces to control the LED and read inputs from the push buttons.

b. (1) x 830 Tie-points Breadboard

Used for prototyping and creating circuits without soldering. It allows for easy connections and rearrangement of components.

c. (1) x 5mm red LED

The light-emitting diode (LED) emits red light when powered. In this project, it serves as the output device, indicating the state of the system.

d. (1) x 220 ohm resistor

Protects the LED from excessive current flow, ensuring its longevity and preventing damage. It's connected in series with the LED to limit the current passing through it.

e. (2) x push switches

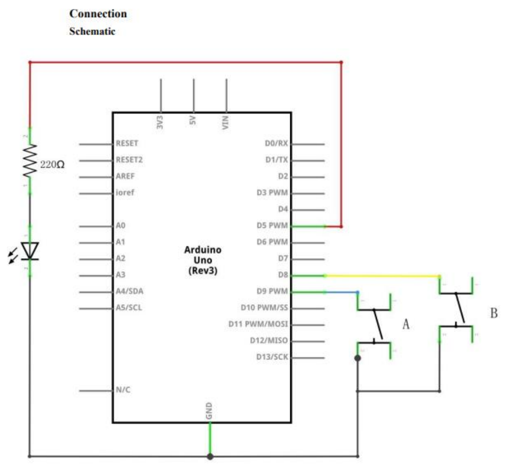
These switches are used as input devices. Pressing them changes their state, allowing the Arduino to detect user inputs.

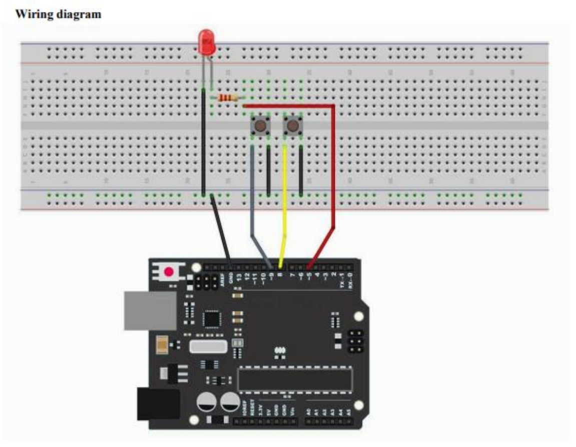
f. (7) x M-M wires (Male to Male jumper wires)

These wires are used to make connections between components on the breadboard and between the breadboard and Arduino.

3. Implementation Steps

a. Connection diagram



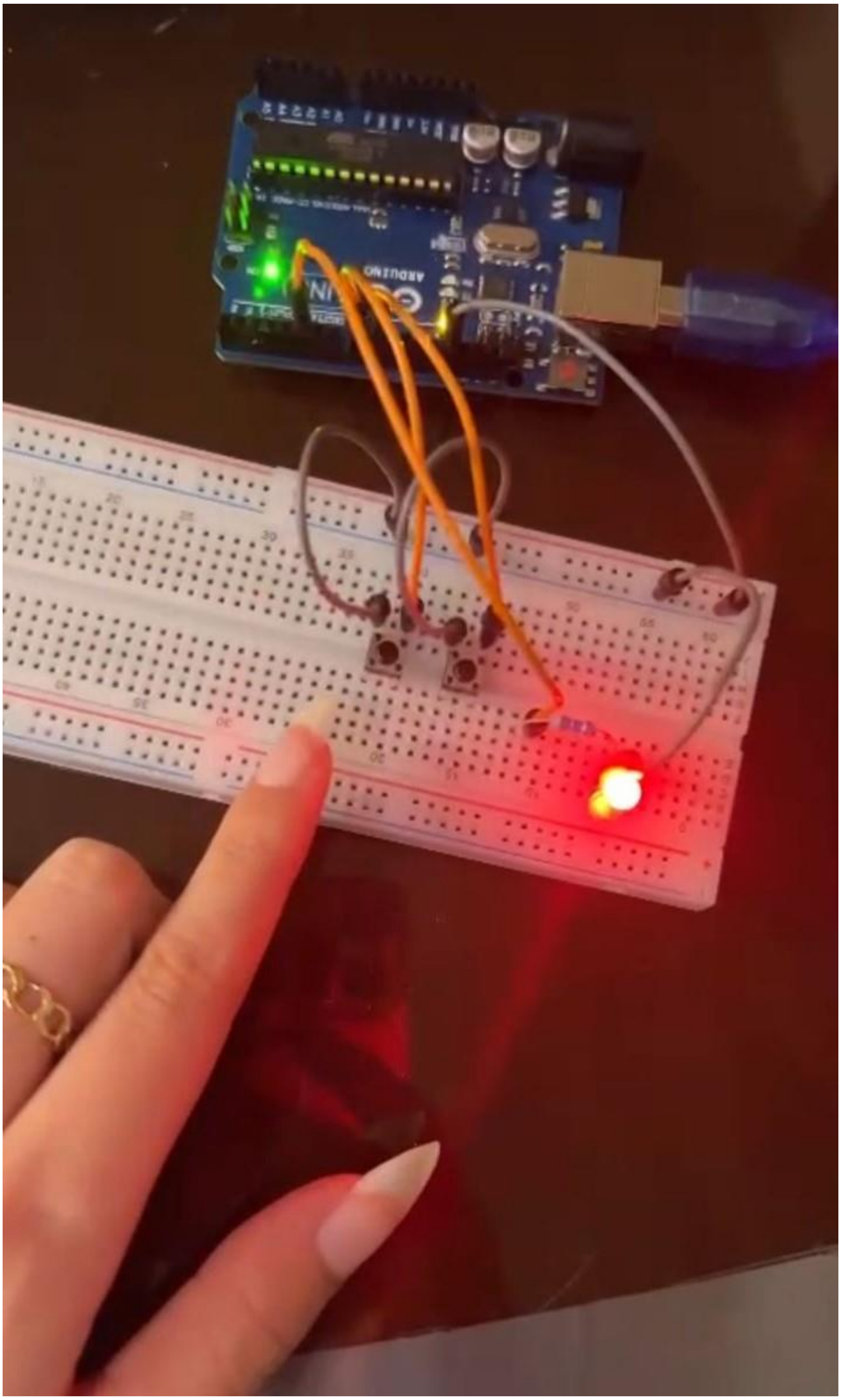


b. Code

```
int ledPin = 5; (this is the output pin)
int buttonApin = 9; (refer to the switch nearer the top of the breadboard)
int buttonBpin = 8; (refer to the other switch)

byte leds = 0;

void setup()
{
  pinMode(ledPin, OUTPUT); (defines the ledPin as being an OUTPUT)
–      pinMode(buttonApin, INPUT_PULLUP);
–      pinMode(buttonBpin, INPUT_PULLUP);  (means that the pin is to be used as an input, the default value for the input is HIGH, unless it is pulled LOW by the action of pressing the button)
–
–    }
–
–    void loop()
–    {
–      if (digitalRead(buttonApin) == LOW)
–      {
–        digitalWrite(ledPin, HIGH);
–      }
–      if (digitalRead(buttonBpin) == LOW)
–      {
–        digitalWrite(ledPin, LOW);
–      }
–    } (there are two 'if' statements. One for each button. Each does an 'digitalRead' on the appropriate input)
```

4. Results

We've successfully crafted a circuit using Arduino where the state of an LED is controlled by push buttons. When activated, the LED lights up, and pressing the buttons again switches it off.



5. Conclusion

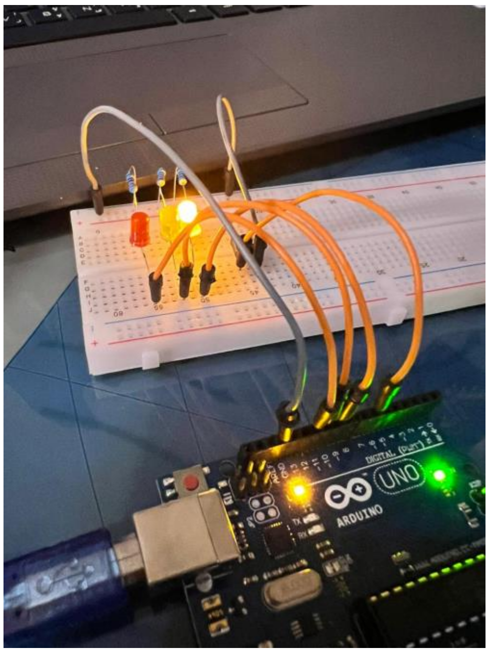
Through this project, we've gained a comprehensive understanding of Arduino's digital input/output capabilities. Working with push buttons and LEDs has enhanced our ability to translate physical interactions into digital signals effectively. We've learned how to control the state of an LED using push buttons, which has provided valuable insights into the practical application of microcontroller programming. This hands-on experience has equipped us with essential skills for future projects involving physical computing and Arduino development.

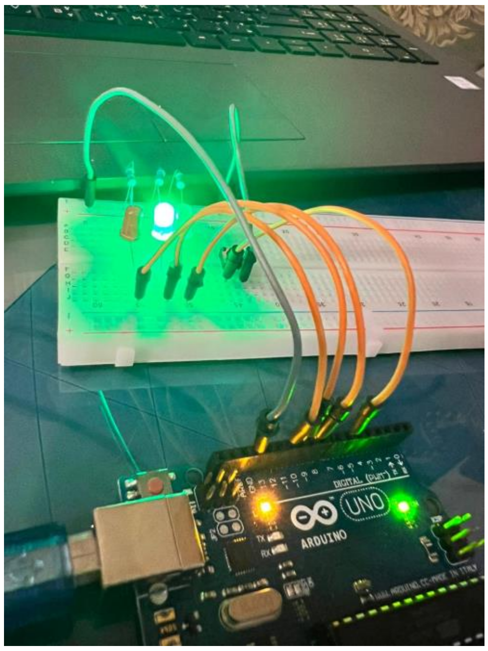
6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

7. Extra Work

In addition to the primary project, we explored Arduino Basics Handling Multiple States. This extension allowed us to delve deeper into the versatility of Arduino programming by managing multiple states within our circuit. By implementing this feature, we gained a deeper understanding of state management and conditional programming in Arduino. This enhancement provided valuable insight into more complex control structures and expanded our toolkit for future projects requiring dynamic state handling.





Code:

```
int button=6;

int redLED=11;

int greenLED=10;

int yellowLED=9;


int state = 0;

int old = 0;

int buttonPoll = 0;


void setup() {

pinMode(button, INPUT_PULLUP);

pinMode(redLED, OUTPUT);

pinMode(greenLED, OUTPUT);

pinMode(yellowLED, OUTPUT);

digitalWrite(redLED, LOW);

digitalWrite(greenLED, LOW);

digitalWrite(yellowLED, LOW);

}


void loop() {


buttonPoll = digitalRead (button);

if(buttonPoll == 1){

delay (50);

buttonPoll = digitalRead (button);
```

```
if (buttonPoll == 0){

state = old + 1;

}}

else{

  delay (100);

}

switch(state){

case 1:

digitalWrite(redLED, HIGH);

digitalWrite(greenLED, LOW);

digitalWrite(yellowLED, LOW);

old = state;

break;

case 2:

digitalWrite(redLED, LOW);

digitalWrite(greenLED, HIGH);

digitalWrite(yellowLED, LOW);

old = state;

break;

case 3:

digitalWrite(redLED, LOW);

digitalWrite(greenLED, LOW);

digitalWrite(yellowLED, HIGH);

old = state;

break; default:

digitalWrite(redLED, LOW);
digitalWrite(greenLED, LOW);

digitalWrite (yellowLED, LOW);

old = 0;

break;

}

}
```

ASSIGNMENT #3: CONTROLLING LED USING TILT BALL SWITCH

1. Introduction

The objective of this experiment is to manage to control a red LED using a tilt ball switch where the intensity of the LED is changing to the tilting of the ball switch.

The role of Tilt sensors (tilt ball switch) is to detect orientation or inclination.

2. Required components

a. **(1) x Uno R3 Board**

b. **(1) x 830 Tie-points Breadboard**

c. **(1) x Tilt Ball Switch**

d. **(2) x F-M wires**

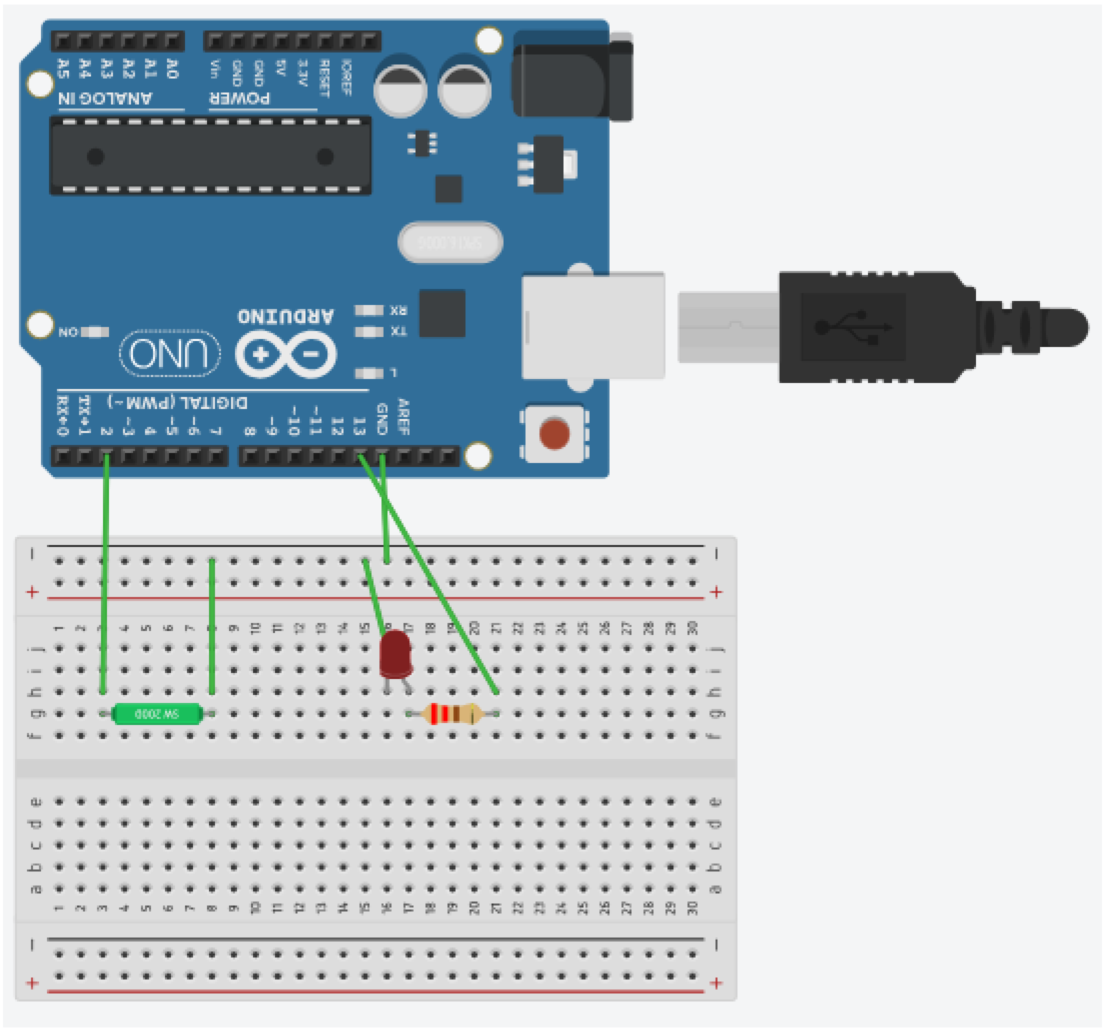
e. **(3) x M-M wires**

f. **(1) x Red LED**

g. **(1) x 220Ω Resistor**

3. Implementation Steps

a. Connection diagram



b. Code

```
/*******************************************/
const int ledPin = 13;//the led attach to


void setup()
{
  pinMode(ledPin,OUTPUT);//initialize the ledPin as an output
  pinMode(2,INPUT);
  digitalWrite(2, HIGH);
}
/****************************************/
void loop()
{
  int digitalVal = digitalRead(2);
  if(HIGH == digitalVal)
  {
    digitalWrite(ledPin,HIGH); //turn the led on
  }
  else
  {
    digitalWrite(ledPin,LOW); //turn the led off
  }
}
/*******************************************/
```

4. Conclusion

Tilt ball switches can detect orientation or inclination. They are small, inexpensive, low-power, and easy to use. Unlike LED, the ball switch has no polarity which means they do not have positive or negative poles.

Resistor is a necessary component in this circuit as it prevents the LED from burning out due to high current.

5. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] Tinkercad, https://www.tinkercad.com/

ASSIGNMET #4: ACTIVE BUZZER

1. Introduction

A **passive buzzer** is a fundamental electronic component that generates sound when an alternating current (AC) signal is applied. Unlike active buzzers, which produce sound independently, passive buzzers require external circuitry to create audible tones. They find applications in alarms, timers, and musical instruments.

1.1 Problem Statement

In this assignment, we explore the interaction between a **passive buzzer** and an **Arduino Uno**. Our goal is to understand how to control the buzzer using digital pins and generate different tones. Specifically, we aim to create a sequence of musical notes, each lasting for a fixed duration.

1.2 Objective

1. Wire up a passive buzzer to the Arduino Uno.
2. Write a simple Arduino sketch to generate specific musical notes using the buzzer.
3. Explore the relationship between frequency, duration, and resulting sound.

2. Required components

a. **Passive Buzzer**

The working principle of passive buzzer is using PWM generating audio to make the air to vibrate. Appropriately changed as long as the vibration frequency, it can generate different sounds. For example, sending a pulse of 523Hz, it can generate Alto Do, pulse of 587Hz, it can generate midrange Re, pulse of 659Hz, it can produce midrange Mi.

b. **Arduino UNO**

The Arduino Uno is a popular microcontroller board based on the ATmega328P chip. It serves as the brain of our project, allowing us to control various components, including the buzzer.

c. **Breadboard**
A breadboard provides a convenient platform for prototyping and connecting electronic components without soldering. It allows us to create temporary circuits for testing and experimentation.

d. **Jumper Wires**
These wires connect the components on the breadboard to the Arduino Uno. We'll use male-to-female (M-F) jumper wires for this setup.

3. Implementation Steps

1. Wiring the Passive Buzzer

1. Connect the positive terminal (+) of the passive buzzer to a digital pin on the Arduino (D8).

2. Connect the negative terminal (-) of the passive buzzer to the GND (ground) pin on the Arduino.

2. Write the Arduino Sketch

Now let's create a simple Arduino sketch to generate different tones using the passive buzzer.

3. Upload the Sketch

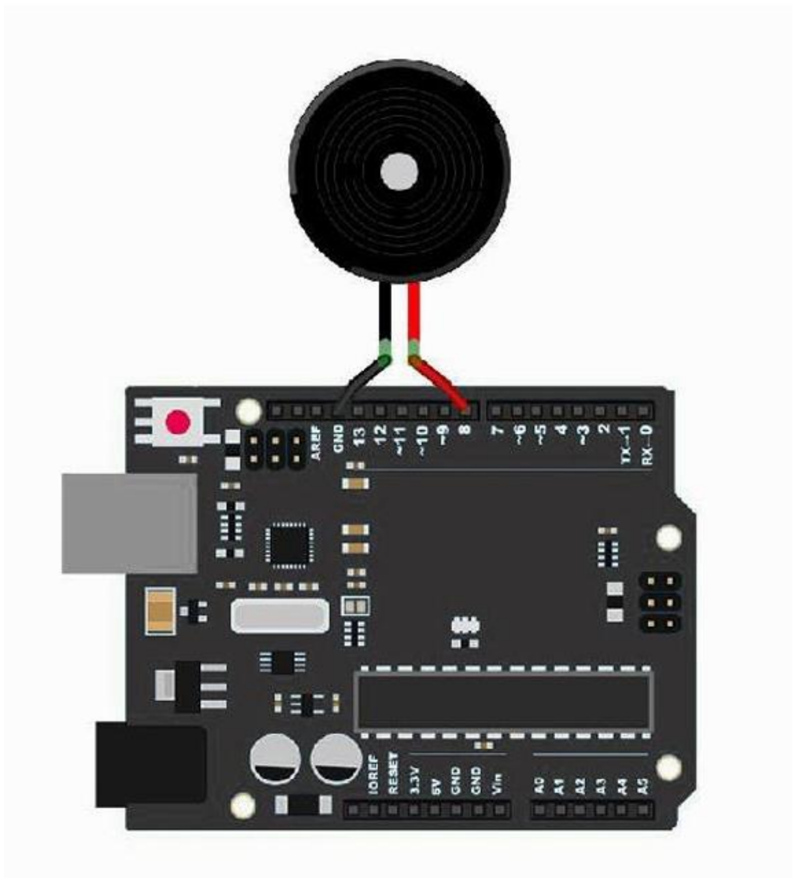
1. Open the Arduino IDE.

2. Create a new sketch and paste the code snippet.

3. Select the correct board (Arduino Uno) and port.

4. Click the Upload button to upload the sketch to the Arduino.

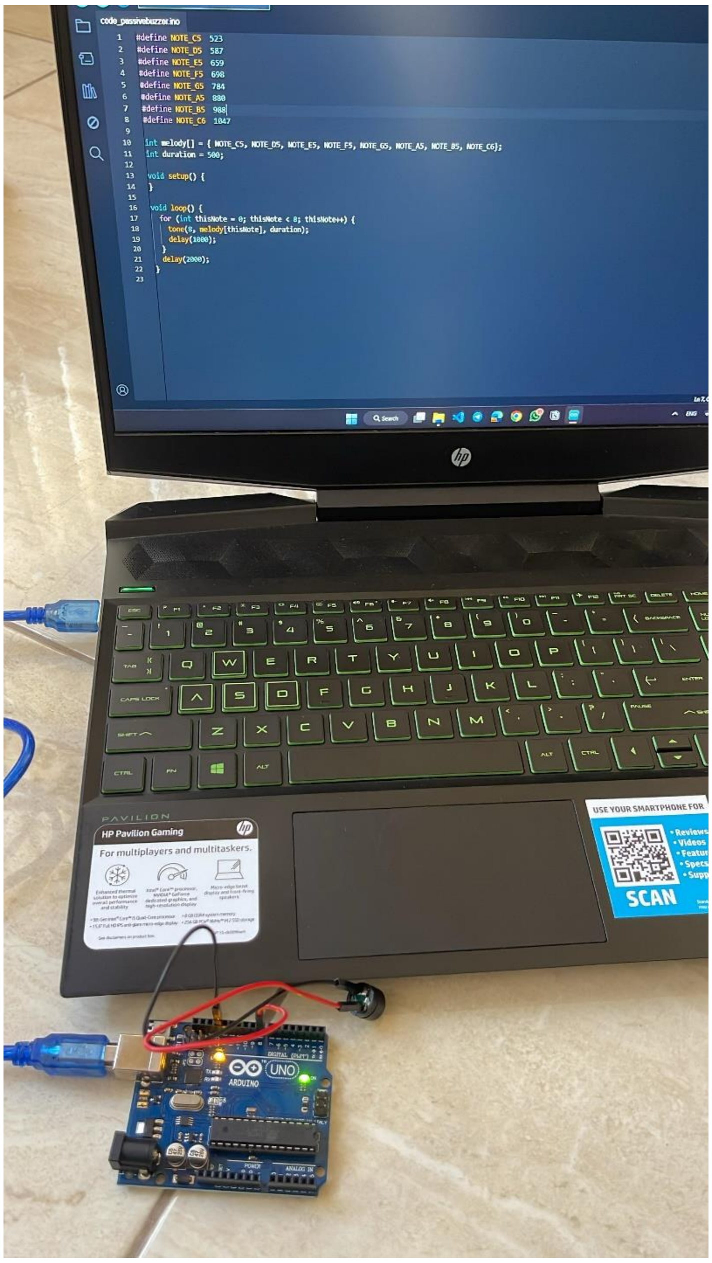
a. Connection diagram



b. Code

```
#define NOTE_C5  523
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_G5  784
#define NOTE_A5  880
#define NOTE_B5  988
#define NOTE_C6  1047

int melody[] = { NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5,
NOTE_B5, NOTE_C6};
int duration = 500;

void setup() {
}

void loop() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    tone(8, melody[thisNote], duration);
    delay(1000);
  }
  delay(2000);
}
```

4. Results



code_passivebuzzer.ino

```
#define NOTE_C5  523
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_G5  784
#define NOTE_A5  880
#define NOTE_B5  988
#define NOTE_C6  1047

int melody[] = { NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_B5, NOTE_C6};
int duration = 500;

void setup() {
}

void loop() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    tone(8, melody[thisNote], duration);
    delay(1000);
  }
  delay(2000);
}
```

5. Conclusion

Learnings:

1. **Passive Buzzer Functionality**:

    - We explored the difference between **active buzzers** (with built-in oscillators) and **passive buzzers** (which rely on external signals).
    - Passive buzzers are compact, simple, and widely used for generating sound in various applications.
2. **Arduino Uno Basics**:
    - The Arduino Uno serves as the **microcontroller** for our project.
    - We learned about digital pins, which can be configured as input or output.
    - The Arduino IDE allows us to write and upload code to the board.
3. **Wiring and Prototyping**:
    - We connected the passive buzzer to the Arduino using jumper wires.

Key Findings:

1. **External Signal Dependency**:
    - Passive buzzers require an external signal (usually a square wave) to produce sound.
    - They do not beep if DC signals are used; square waves are essential.
2. **Tone Generation**:
    - By using the tone() function in Arduino, we can generate specific musical notes.
    - Adjusting the frequency values allows us to play different tones.
3. **Creative Exploration**:
    - We can create melodies, alarms, or custom sound effects by combining different tones.
    - The passive buzzer opens up possibilities for sound-based projects.
4. **Conclusion:**

- Working with the passive buzzer and Arduino Uno was an exciting journey into the world of sound manipulation. We learned how to wire components, write code, and create audible tones. Whether it's a simple alarm or a musical melody, the passive buzzer adds a fun dimension to our Arduino projects.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #5: TEMPARETURE AND HUMIDITY SENSOR

1. Introduction

Monitoring temperature and humidity is important for many applications, such as industrial, environmental monitoring and agriculture. Technology has advanced and because microcontrollers like Arduino are affordable and simple to use, the integration of sensors with them has grown in popularity. This assignment's main goal is to investigate the temperature and humidity sensor while addressing the difficulties in their implementation. We aim to gain a comprehensive understanding of the capabilities and limitations of this sensor integrated with Arduino platforms.

2. Required components.

a. Arduino UNO

b. Temperature and humidity sensor

c. Breadboard

d. LCD Screen

e. Wires

3. Implementation Steps

a. Connection diagram

In the figures below you'll find the wiring diagrams of each of DHT11 and LCD Screen alone and a brief explanation of the wiring procedure we followed.

DHT11 sensor:

. 1st pin is the data pin connected to digital pin 7

. 2nd pin (Vcc) power: connected to %v pin on the Arduino using the breadboard we connected the 5v pin to the + side so that we can use the 5V for the DHT11 and the LCD screen together.

. GND pin: connect it to the GND pin.

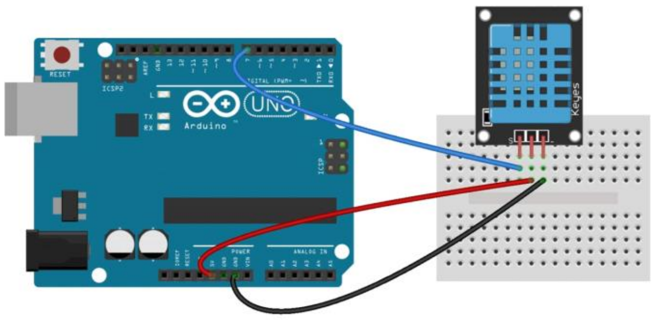


Fig.1: DHT11 with Arduino connection.

LCD with I2cC module:

. Vcc: connected to 5v pin.

. GND: connected to GND pin on the Arduino.

. SDA: Connected to A4 analog pin on the Arduino (for I2C communication)

. SCL: Connected to A5 analog pin on the Arduino.

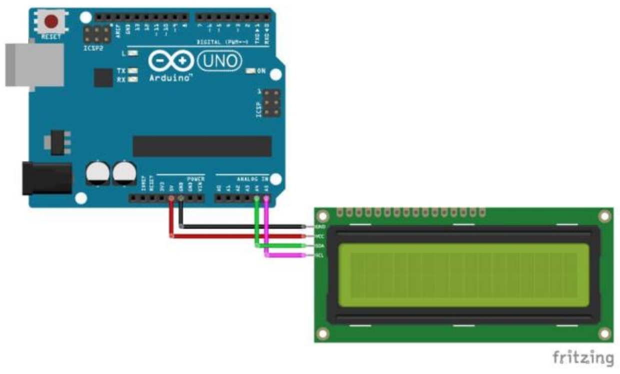


Fig.2: LCD I2C connections with Arduino.

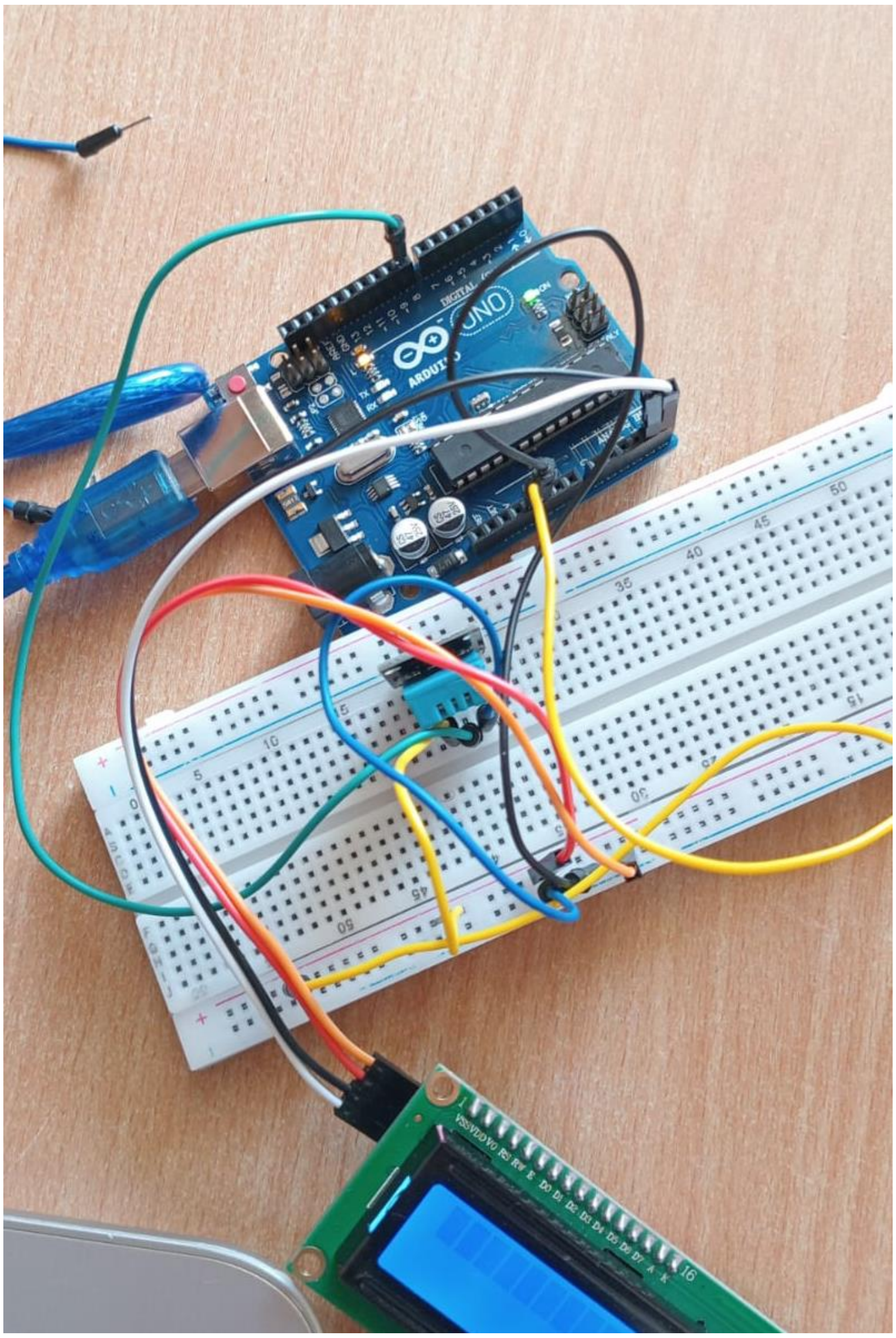


Fig.3: Wiring of the DHT11 , LCD and the Arduino

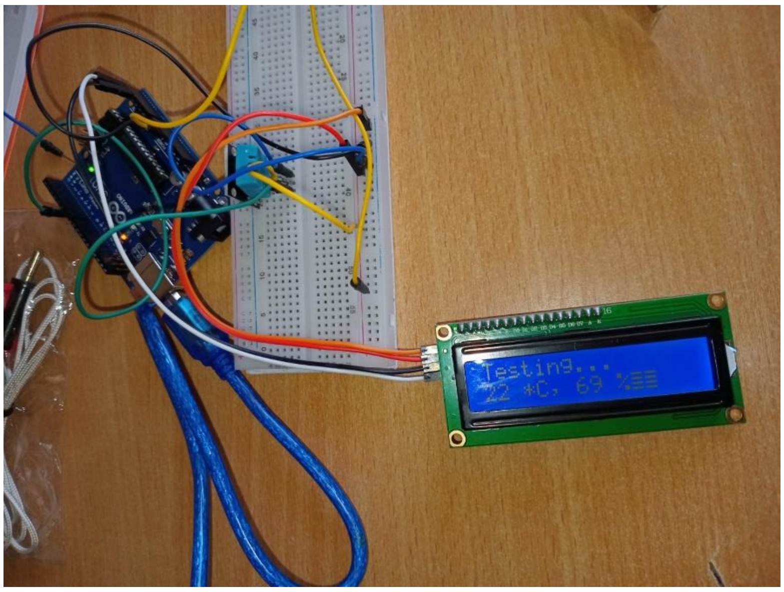
b. Code

```
#include <LiquidCrystal_I2C.h>
#include <SimpleDHT.h>
int pin = 7;
SimpleDHT11 dht;
LiquidCrystal_I2C lcd(0x27,16,2);
void setup() {
  Serial.begin(9600);
  lcd.backlight();
  lcd.begin(16,2);
}

void loop(){
  byte temperature = 0;
  byte humidity = 0;
  lcd.setCursor(0,0);
  lcd.print("Testing...");
   lcd.setCursor(0,1);
  if (dht.read(pin, &temperature, &humidity, NULL)) {
    lcd.print("Read DHT failed.");
    return;
  }
  Serial.print((int)temperature); Serial.print(" *C, ");
 Serial.print((int)humidity); Serial.println(" %");
  lcd.print((int)temperature); lcd.print(" *C, ");
lcd.print((int)humidity); lcd.println(" %");
  delay(2000); //to work properly the sensor needs 2 seconds delay
 lcd.clear();
}
```

4. Results

After wiring the DHT11 sensor and the LCD to the Arduino, uploading the appropriate code and proving the Arduino with a power supply (from the PC), the sensor successfully read the temperature and humidity accurately. The LCD displayed the temperature and humidity reading in real-time, providing a clear and an easy-to-read interface.

The pictures below will provide you with the sample reading of the sensor along with handling procedure when the sensor can’t read a value (when we remove the data pin)



Fig,4: Result

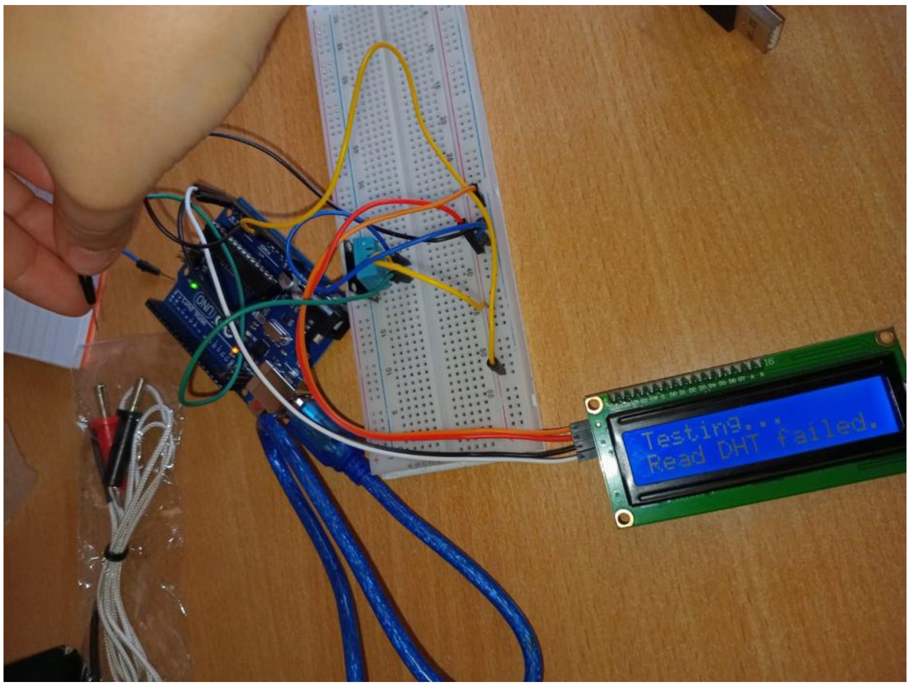


Fig.6: Error handling

5. Conclusion

Integrating the DHT11 sensor and LCD display with the Arduino microcontroller allows for real-time monitoring of temperature and humidity levels. This setup provides a cost effective and user-friendly solution for obtaining environmental data, enabling users to make informed decisions based on accurate measurements. With further enhancements and additions, such as data logging capabilities or remote monitoring features, this system can be customized to meet specific requirements and serve a wide range of purposes effectively.

6. References

Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #6: IR RECEIVER MODEL

1. Introduction

The IR Receiver Model experiment is about using a special sensor called an infrared (IR) receiver with Arduino. This experiment helps beginners understand how IR sensors work and how they can be used in projects. By following simple steps, you can learn to connect the sensor to Arduino, write code, and make it respond to IR signals. This hands-on experience teaches basic electronics and coding skills, empowering you to create things like remote-controlled gadgets or smart devices for your home.

2. Required components

a. **Computer:** required for programming the Arduino board and uploading code

b. **Arduino board:** microcontroller platform for controlling electrical components

c. **Jumper wires(M to M)** : to make all necessary connections

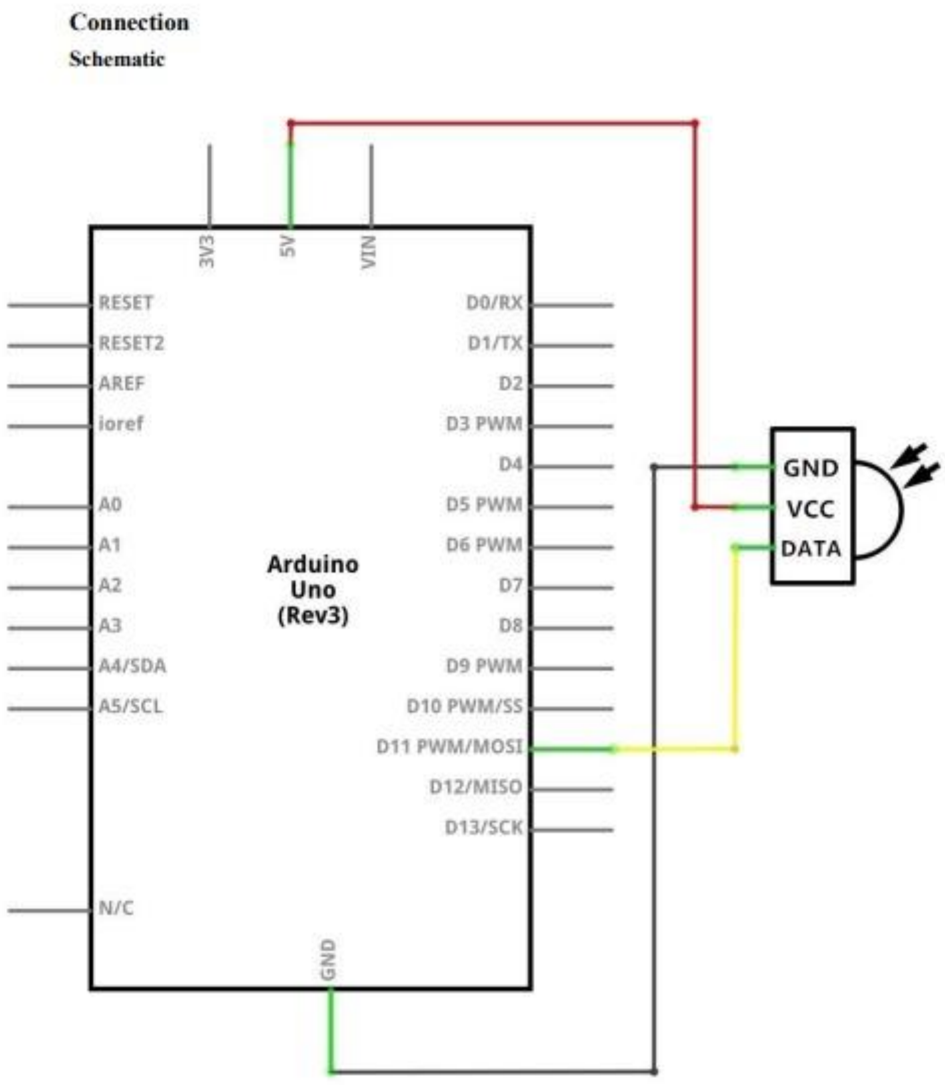
d. **IR receiver module:** sensor for detecting and receiving infrared signals

e.IR transmitter: device emitting infrared signals

3. Implementation Steps

Talking about the hardware side we connect the IR sensor via jumping wires (M to M),after connecting the Arduino board to the computer via a cable.we then open the required application (Arduino IDE) write the required program and making sure to include the needed library, we then open the serial moniter to observe the output

a. Connection diagram



b. Code

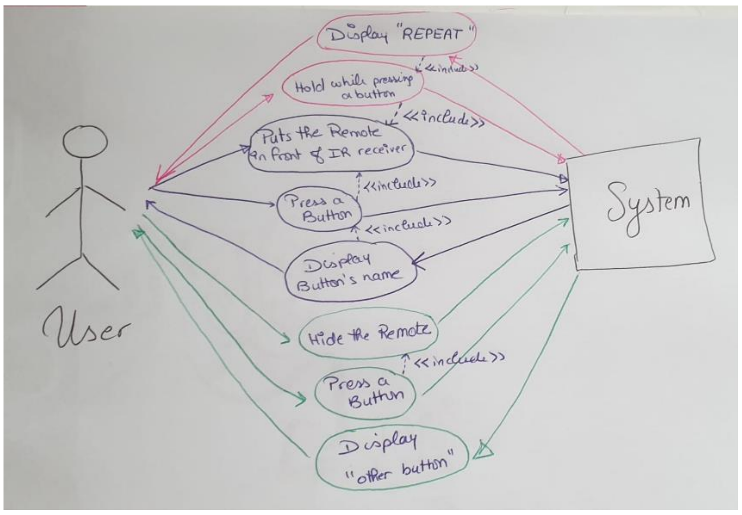
```
#include "IRremote.h"


int receiver = 11; // Signal Pin of IR receiver to Arduino Digital Pin 11


/*-----( Declare objects )-----*/

IRrecv irrecv(receiver);     // create instance of 'irrecv'
```

```
decode_results results;      // create instance of 'decode_results'


/*-----( Function )-----*/
void translateIR() // takes action based on IR code received


// describing Remote IR codes
{
  switch(results.value)
  {
  case 0xFF629D: Serial.println("UP"); break;
  case 0xFF22DD: Serial.println("LEFT");    break;
  case 0xFF02FD: Serial.println("OK");    break;
  case 0xFFC23D: Serial.println("RIGHT");   break;
  case 0xFFA857: Serial.println("DOWN");    break;
  case 0xFF9867: Serial.println("2");    break;
  case 0xFFB04F: Serial.println("3");    break;
  case 0xFF6897: Serial.println("1");    break;
  case 0xFF30CF: Serial.println("4");    break;
  case 0xFF18E7: Serial.println("5");    break;
  case 0xFF7A85: Serial.println("6");    break;
  case 0xFF10EF: Serial.println("7");    break;
  case 0xFF38C7: Serial.println("8");    break;
  case 0xFF5AA5: Serial.println("9");    break;
  case 0xFF42BD: Serial.println("*");    break;
  case 0xFF4AB5: Serial.println("0");    break;
  case 0xFF52AD: Serial.println("#");    break;
  case 0xFFFFFFFF: Serial.println(" REPEAT");break;


  default:
    Serial.println(" other button   ");
  }// End Case


  delay(500); // Do not get immediate repeat


} //END translateIR
void setup()   /*----( SETUP: RUNS ONCE )----*/
{
  Serial.begin(9600);
  Serial.println("IR Receiver Button Decode");
  irrecv.enableIRIn(); // Start the receiver


}/*--(end setup )---*/




void loop()   /*----( LOOP: RUNS CONSTANTLY )----*/
{
  if (irrecv.decode(&results)) // have we received an IR signal?


  {
    translateIR();
    irrecv.resume(); // receive the next value
  }
```

}/* --(end main loop)-- */

4. Results

we get(the keywords:“**UNKNOWNBUTTON**” when the IR receiver a.ka. remote is directed toward the sensor and we get “**REPEAT**” for pressing a button holding on it, otherwise we get the pressed button to be displayed.



N.B: there’s a video showing the results of the experiment

5. Conclusion

We can conclude that even if the remote is not directed towards the IR sensor , we will obtain an output which means that the sensor will still detect the infrared signals but the information from them will not be obtained

6. References

https://lafvintech.com/

ASSIGNMET #7: SERVO MOTOR

1. Introduction

Servo is a type of geared motor that can only rotate 180 degrees. It is controlled by sending electrical pulses from your UNO R3 board. These pulses tell the servo what position it should move to. The Servo has three wires, of which the brown one is the ground wire and should be connected to the GND port of UNO, the red one is the power wire and should be connected to the 5v port, and the orange one is the signal wire and should be connected to the Dig #9 port.

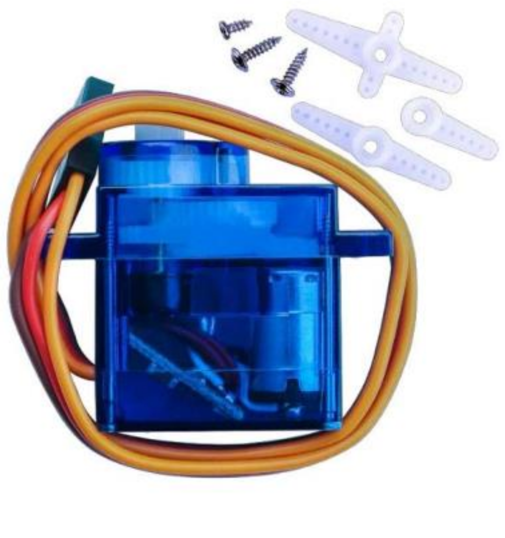
2. Required components

a. **(1)x Uno R3 Board**

b. **(1)x Servo (SG90)**

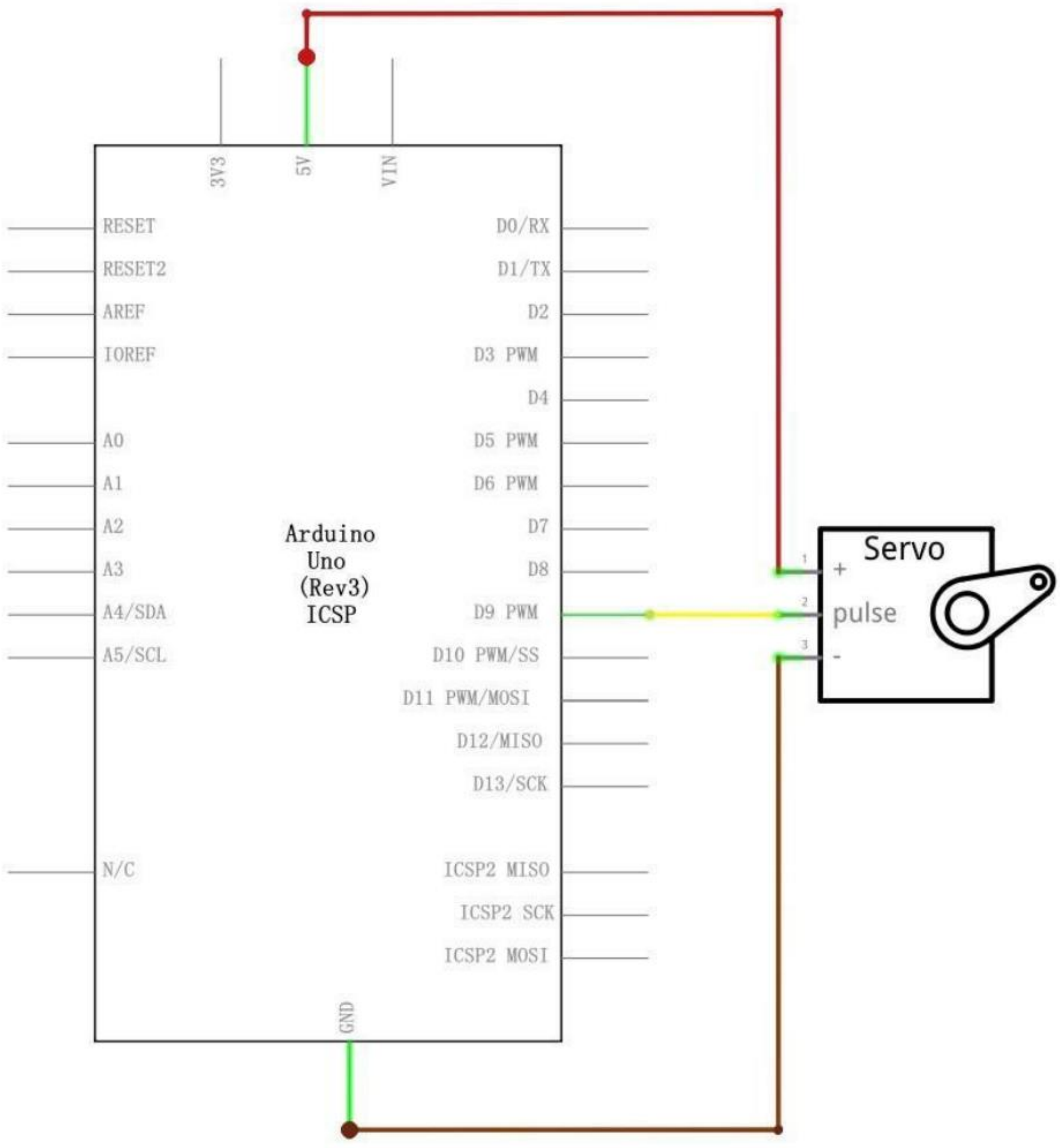
c. **(3) x M-M wires (Male to Male jumper wires)**

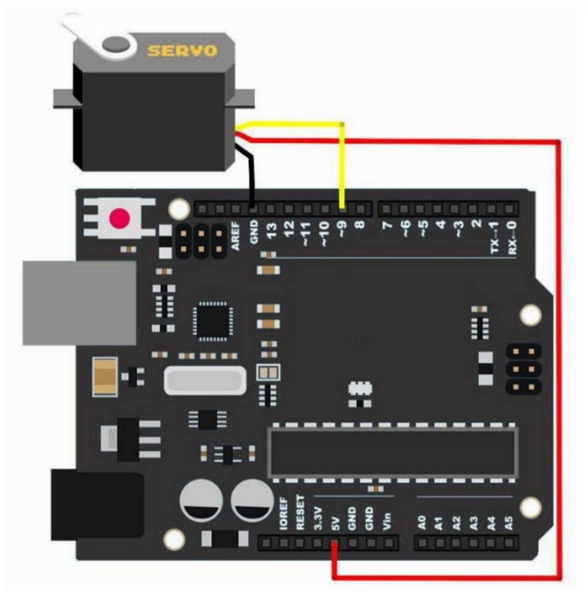
Universal for JR and FP connector Cable length : 25cm No load; Operating speed: 0.12 sec / 60 degree (4.8V), 0.10 sec / 60 degree (6.0V) Stall torque (4.8V): 1.6kg/cm Temperature : -30~60'C Dead band width: 5us Working voltage: 3.5~6V Dimension : 1.26 in x 1.18 in x 0.47 in (3.2 cm x 3 cm x 1.2 cm) Weight : 4.73 oz (134 g).



3. Implementation Steps

a. Connection diagram

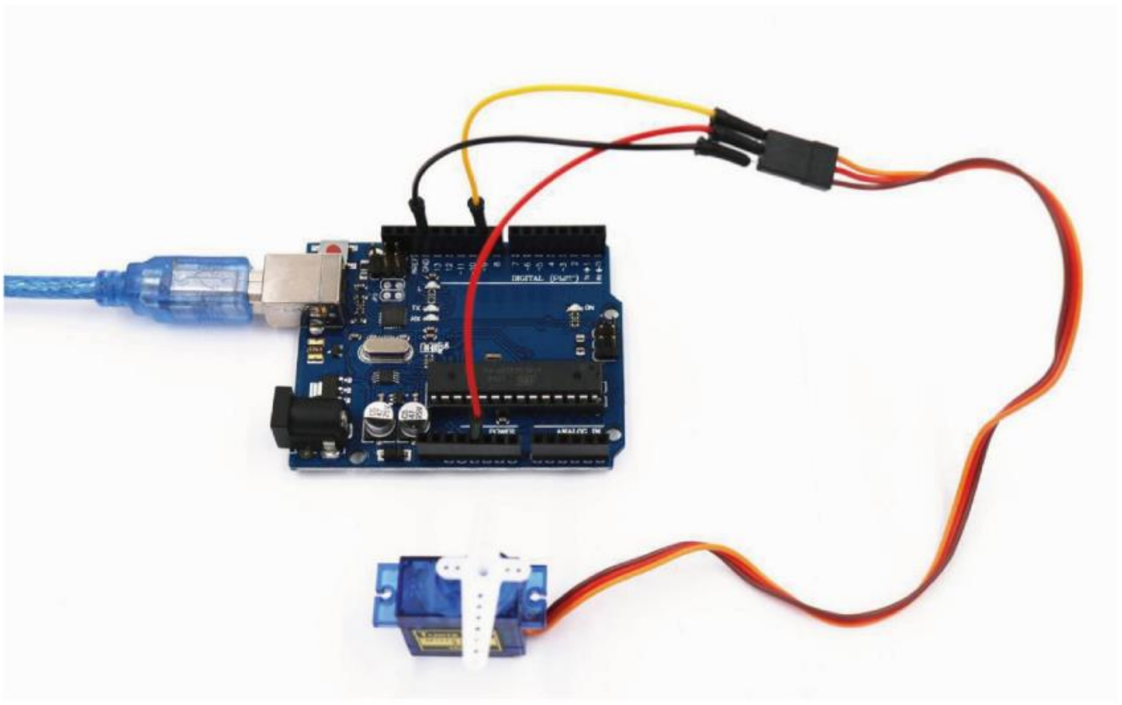




b. Code

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

26. Results



27. Conclusion

Servo motors offer several advantages:

Precision Control: Servo motors provide precise control over position, velocity, and acceleration, making them suitable for applications requiring high accuracy.

High Torque-to-Size Ratio: Despite their compact size, servo motors can deliver high torque, making them ideal for applications where space is limited but high power is needed.

Feedback Mechanism: Most servo motors incorporate feedback mechanisms such as encoders or resolvers, allowing for accurate position feedback. This enables closed-loop control systems, enhancing accuracy and stability.

Fast Response Time: Servo motors have rapid response times, making them suitable for dynamic applications where quick adjustments are necessary.

Customizable Performance: Servo motors can be customized to meet specific performance requirements such as speed, torque, and precision, making them versatile across various industries and applications.

In conclusion, servo motors offer precise control, high torque-to-size ratio, feedback mechanisms for accuracy, fast response times, and customizable performance, making them a preferred choice for applications requiring precise and dynamic motion control.

28. References

You can add references if anyone is used rather than the one provided.

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #8: ANALOG JOYSTICK MODULE

1. Introduction

This report examines how the analog joystick module for Arduino can be used to control projects effectively. The objective is to optimize its usage for smoother project control and enhanced user interaction.

2. Required components

a. Arduino Uno R3 Board

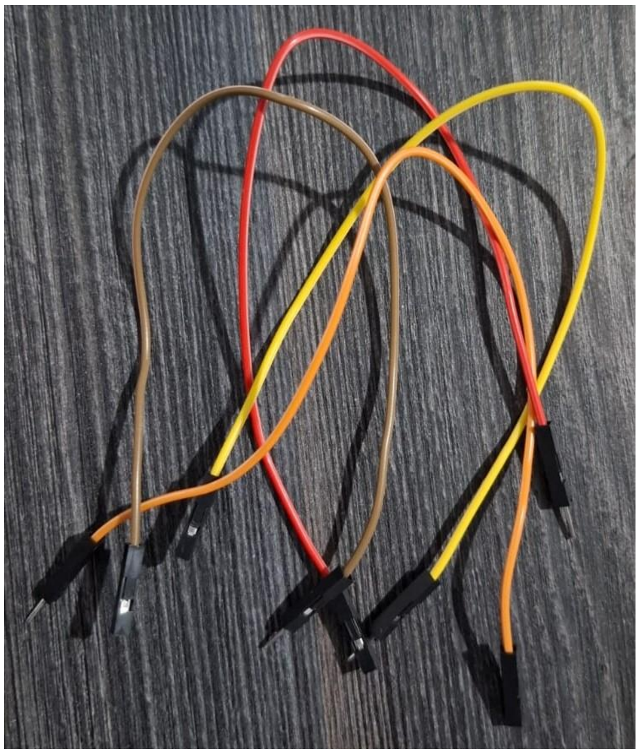
The Arduino Uno R3 serves as the central processing unit for testing the input of the joystick module.

b. Joystick Module

The joystick modules two potentiometers translate physical movements into analog voltage signals, representing the joystick's position along the X and Y axes. Additionally, its tactile switch provides a digital signal indicating button presses.

c. 5 F-M wires (Female to Male DuPont wires)

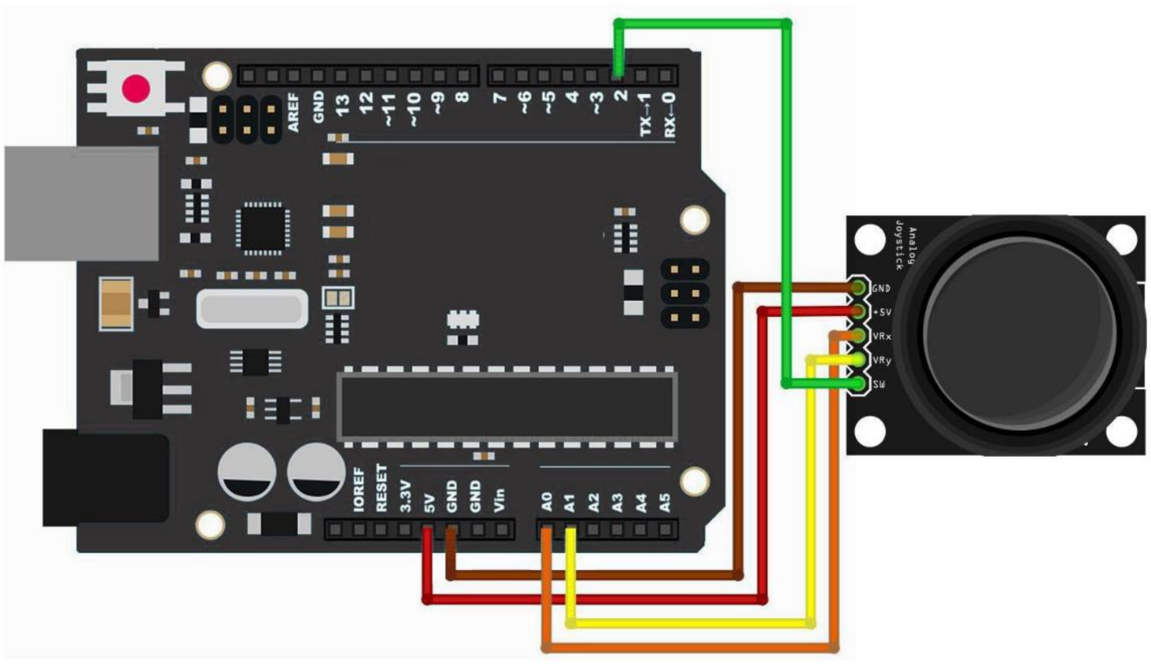
These wires establish connections between the Arduino board and the joystick module.



3. Implementation Steps

To test the joystick module, we simply connect it to the analog and digital pins of the arduino and observe the output on the serial monitor.

a. Connection diagram

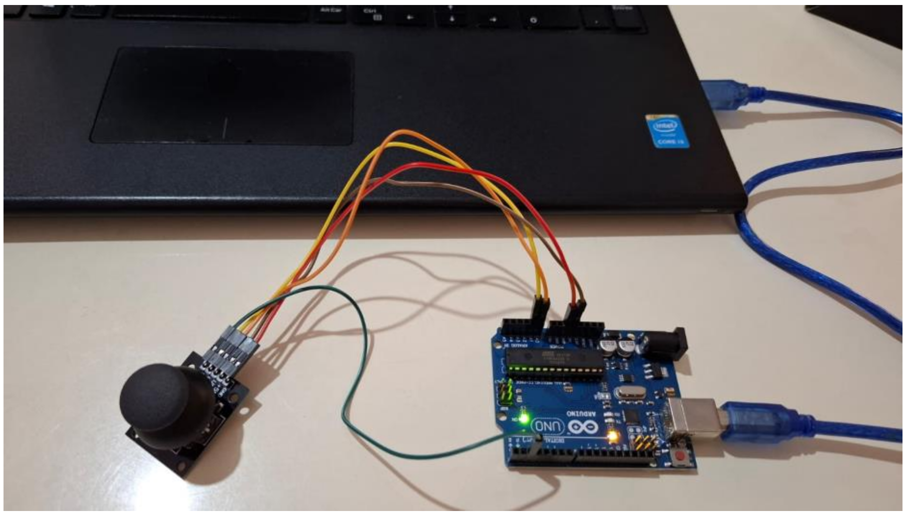


b. Code

```
// Arduino pin numbers
const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output

void setup() {
    pinMode(SW_pin, INPUT);
    digitalWrite(SW_pin, HIGH);
    Serial.begin(9600);
}

void loop() {
    Serial.print("Switch:  ");
    Serial.print(digitalRead(SW_pin));
    Serial.print("\n");
    Serial.print("X-axis: ");
    Serial.print(analogRead(X_pin));
    Serial.print("\n");
    Serial.print("Y-axis: ");
    Serial.println(analogRead(Y_pin));
    Serial.print("\n\n");
    delay(500);
}

```

4. Results

Link to video



5. Conclusion

The joystick module offered precise control over two axes, allowing for nuanced input that can be translated into various actions within a project. Working with the Arduino Uno R3 board and joystick module provided insights into sensor integration and control mechanisms.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMENT #9: WATER LEVEL DETECTION SENSOR MODULE

1. Introduction

Water level detection is crucial in various applications such as water tanks, industrial processes, and environmental monitoring. Traditional methods often involve manual monitoring or complex systems. However, with advancements in sensor technology and microcontrollers like Arduino, automated and efficient solutions are now feasible.

Background Information:

Water level detection systems traditionally rely on float switches, ultrasonic sensors, or capacitive sensors. Each method has its advantages and limitations in terms of accuracy, reliability, and cost.

Problem Statement:

Despite the availability of different sensing technologies, there is a need for easy-to-implement water level detection system suitable for various applications.

Objective:

The objective of this project is to develop a water level detection sensor module using Arduino .

2. Required Components:

Arduino Uno R3 Board

Female to Male DuPont wires (3 pieces)

Water level detection sensor module

Explanation of main functionalities:

Arduino Uno R3 Board : This serves as the microcontroller that receives input from the water level detection sensor module and processes it according to the programmed logic. It controls the overall operation of the system.

Female to Male DuPont wires : These wires are used to establish connections between the Arduino board and the water level detection sensor module. They transfer signals and power between the components.

Water level detection sensor module : This sensor module detects the level of water using conductivity. It typically consists of probes or sensors that come into contact with the water. The main functionalities of this module include:

. Sensing water level: The sensor detects the presence and level of water at its probes.

. Signal output: It generates electrical signals corresponding to the water level detected.

. Interface with Arduino: The module communicates with the Arduino board, typically through digital or analog pins, to transmit the water level information.

3. **Implementation Steps:**

The water level has three pins :

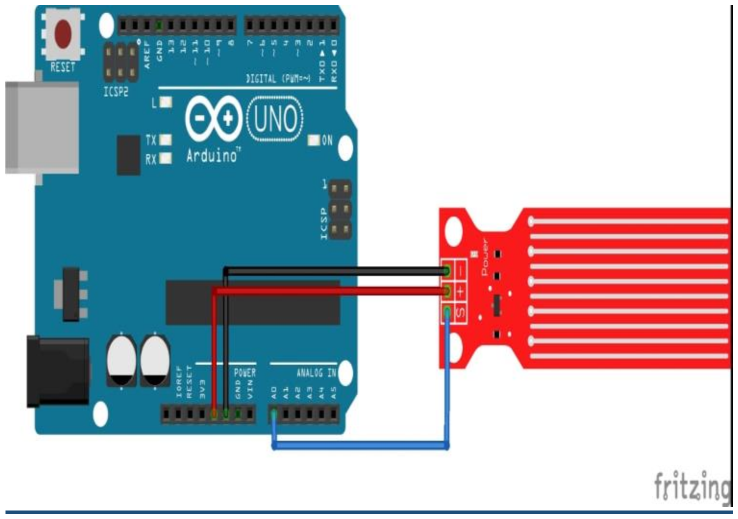
1. power pin
2. ground pin
3. output analog signal pin that must be connected to the board

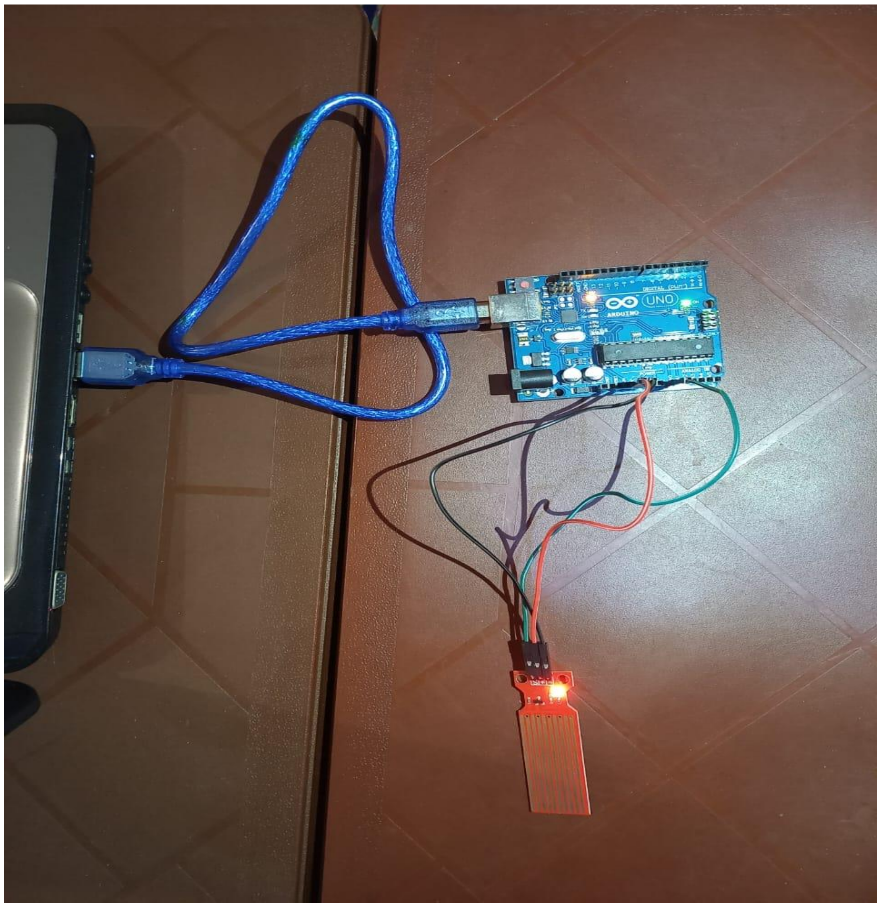
Functionality:

The more the water sensor is immersed in water the higher is the conductivity meaning less resistivity. Hence more water level leads to lower voltage and vice versa.

The output received is then interpreted according to limits (found by experiment) as high, medium, low or empty.

a. Connection Diagram

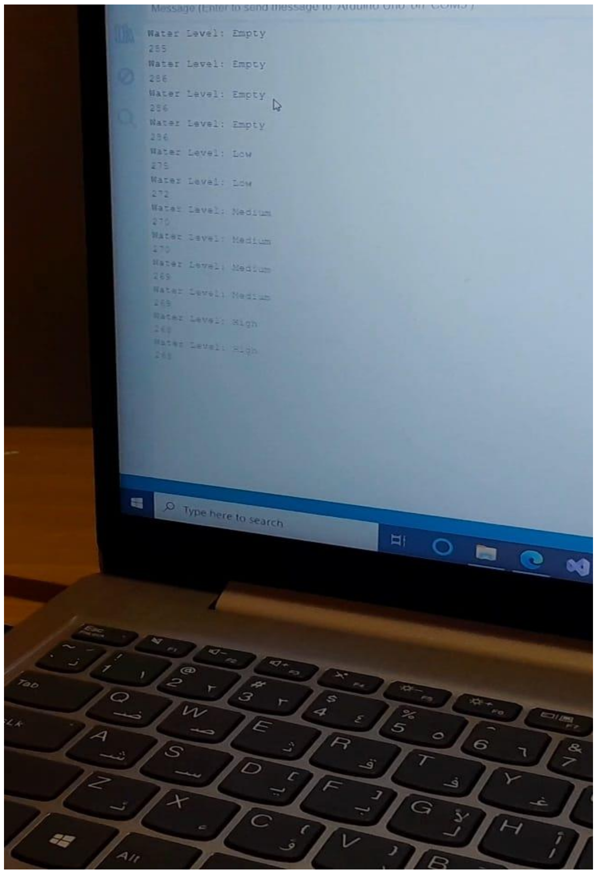




b. Code

```
int sensorvalue=0;
int upperThreshold=282;
int lowerThreshold=271;
#define readingpin A0
#define PowerSensor 7
void valueclassification(int value)
{
  if (value <= 268) {
           Serial.println("Water Level: High");
   Serial.println(value);
  }
  else if (value > 268 && value <= lowerThreshold) {
            Serial.println("Water Level: Medium");
   Serial.println(value);
  }
  else if (value > 268 && value <= upperThreshold) {
            Serial.println("Water Level: Low");
   Serial.println(value);
  }
  else if (value>282) {
            Serial.println("Water Level: Empty");
   Serial.println(value);
  }
}
int readfromsensor()
{
  digitalWrite(PowerSensor,HIGH);
  delay(1000);
  int value=analogRead(PowerSensor);
  delay(1000);
  digitalWrite(PowerSensor,LOW);
  delay(1000);
  return value;
}
void setup() {
  pinMode(PowerSensor,OUTPUT);
  digitalWrite(PowerSensor,LOW);
}

void loop() {
  Serial.begin(9600);
  sensorvalue=readfromsensor();
  delay(1000);
  valueclassification(sensorvalue);
  delay(1000);
}
```

4. Results



```
Water Level: Empty
255
Water Level: Empty
256
Water Level: Empty
256
Water Level: Empty
256
Water Level: Low
275
Water Level: Low
272
Water Level: Medium
270
Water Level: Medium
270
Water Level: Medium
269
Water Level: Medium
269
Water Level: High
269
Water Level: High
268
```

5. Conclusion

. Reading electric signals using analog pin.

. Employing the reading using Arduino software to display the output (physical measurement).

. Controlling type of voltage (high or low) sent at digital pin.

6. References

Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #10: ULTRASONIC SENSOR MODULE

1. Introduction

Ultrasonic sensor is great for all kind of projects that need distance measurements, avoiding obstacles as examples. The HC-SR04 is inexpensive and easy to use since we will be using a Library specifically designed for these sensor.

2. Required components

a. **(1) x Uno R3 Board**

b. **(1) x Ultrasonic sensor module**

c. **(4) x F-M wires (Female to Male DuPont wires)**

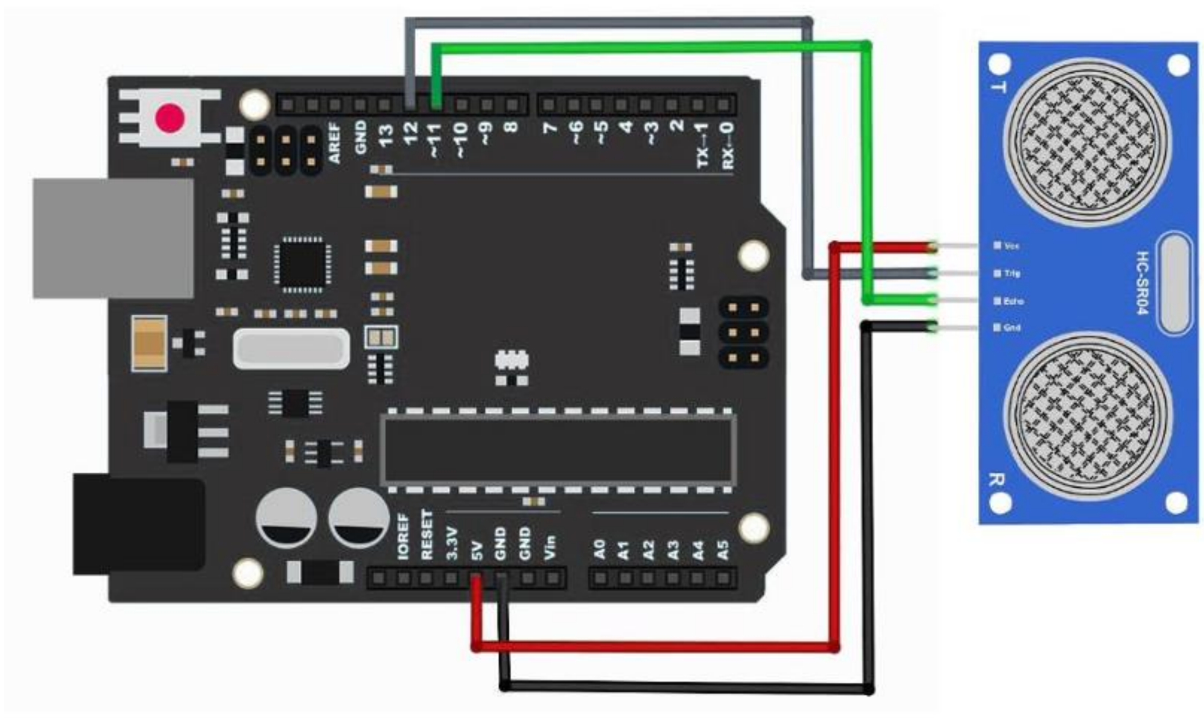
3. Implementation Steps

1_Make all hardware connections.

2_Write the code needed on software.

3_Upload the code to the Arduino.

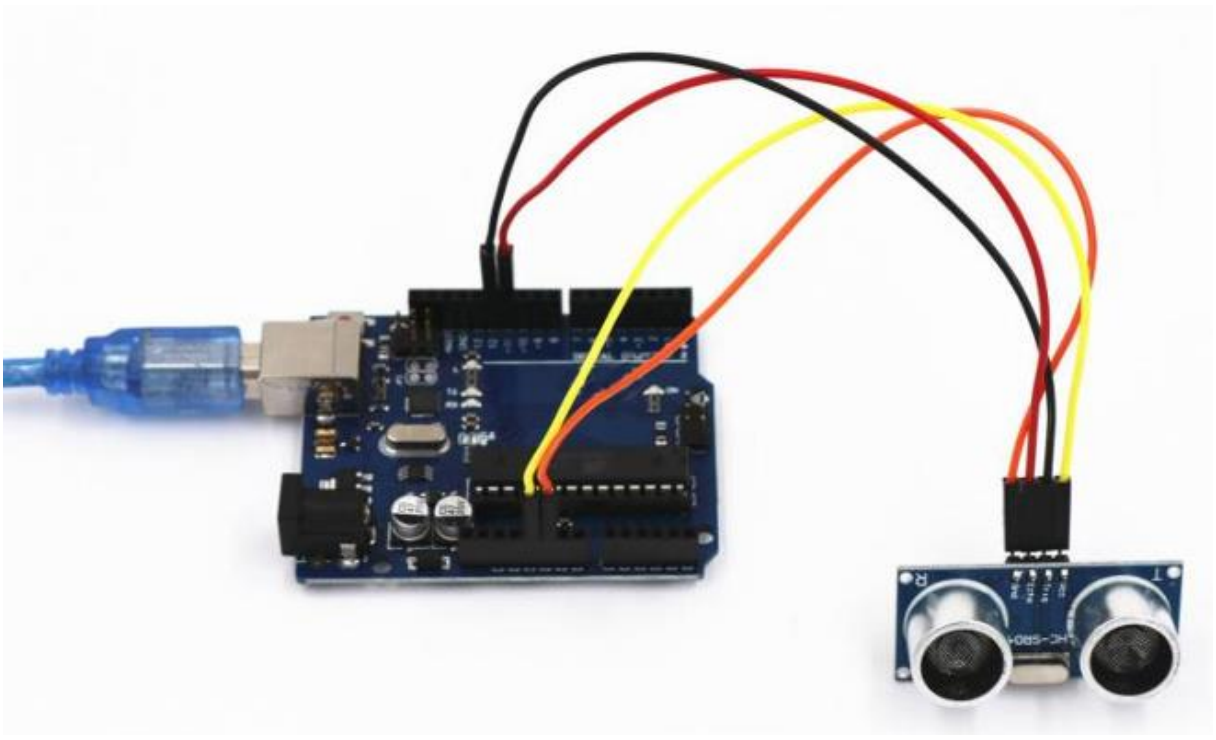
a. Connection diagram



b. Code

```
–      #include <Ultrasonic.h>
–      #define trigPin 12
–      #define echoPin 11
–      Ultrasonic ultrasonic (trigPin , echoPin);
–      void setup() {
–      Serial.begin(9600);
–      }
–      void loop(){
–      float distance_cm=ultrasonic.distanceRead(CM);
–      Serial.print("Distance: ");
–      Serial.print(distance_cm);
–      Serial.println(" cm");
–      delay(1000);
}
```

4. Results



5. Conclusion

We have learned how to make a small project by doing connections in a write way, implementing the code, solving problem issues, uploading code to Arduino, and enjoying successful results.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #11: HC-SR501 PIR SENSOR

1. Introduction

In this session, we were introduced to the HC-SR501 PIR Sensor and explored its functionality: it detects movement, activating the light for a set duration before deactivating to await further movement within its range. As a result, our task is to experiment with integrating it with an UNO board.

2. Required components

We need 4 components to achieve the schematic circuit:

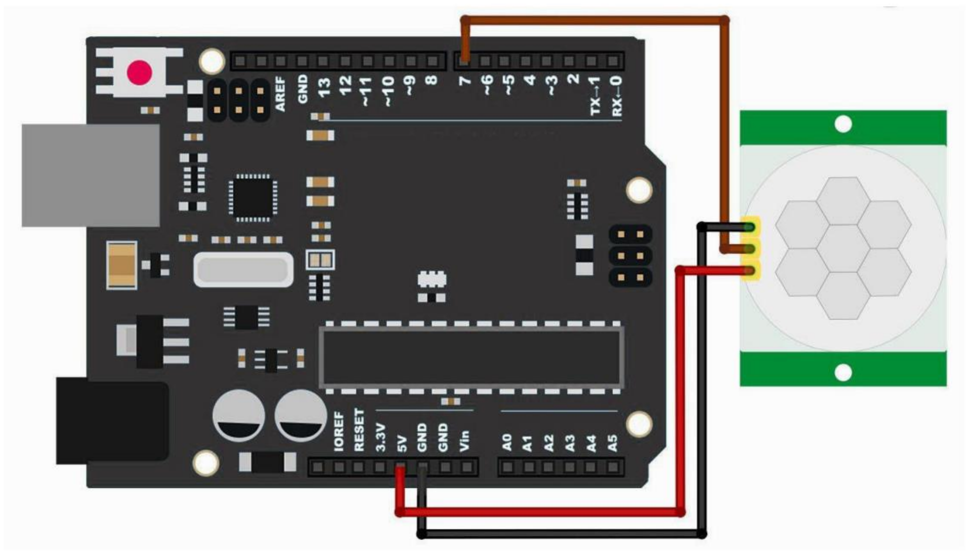
- **(1)x Uno R3 Board:** The UNO R3 board has rows of connectors along both sides that are used to connect to several electronic devices and plug-in 'shields' that extends.
- **(1)x HC-SR501 PIR motion sensor:** The SR501 will detect infrared changes and if interpreted as motion, will set its output low. What is or is not interpreted as motion is largely dependent on user settings and adjustments. The device will detect motion inside a 110-degree cone with a range of 3 to 7 meters.
- **(3)x F-M wires (Female to Male DuPont wires):** to connects between the motion sensor and UNO.
- **Red led:** The function of this led is to radiate when the sensor detects movement.

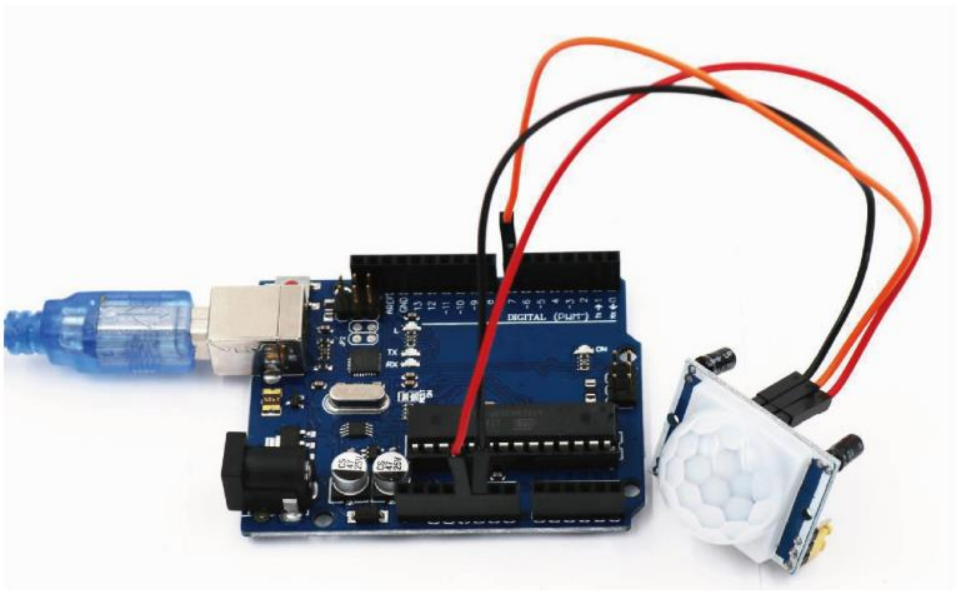
Pin or Control	**Function**
Time Delay Adjust	Sets how long the output remains high after detecting motion .... Anywhere from 5 seconds to 5 minutes.
Sensitivity Adjust	Sets the detection range .... from 3 meters to 7 meters.
Trigger Selection Jumper	Set for single or repeatable triggers.
Ground pin	Ground input.
Output Pin	Low when no motion is detected. High when motion is detected. High is 3.3V.
Power Pin	5 to 20 VDC Supply input.

3. Implementation Steps

1. Prepare all the required components.
2. Power the PIR with 5V and connect ground to ground. Then connect the output to a digital pin, such as pin 7.
3. Place the red LED between pin 13 and ground.
4. Connect UNO to the laptop via USB.
5. Open IDE and write the code of implementation.
6. Run it and testing the sensor.

a. Connection diagram



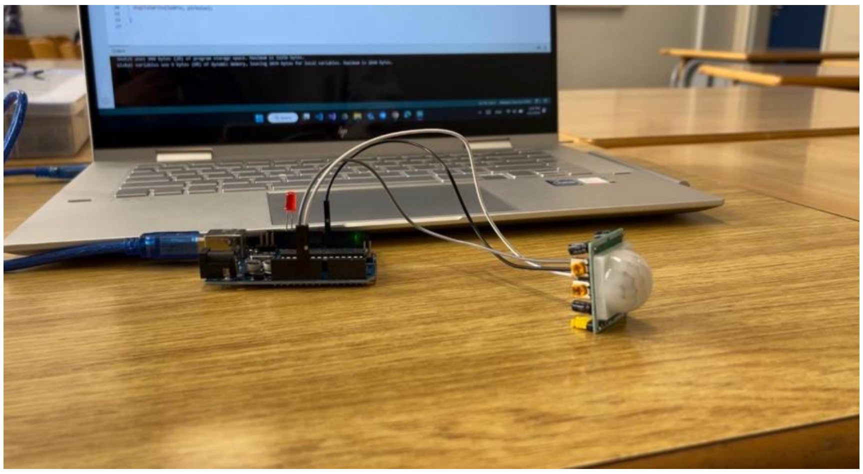


b. Code

```
int ledPin = 13;  // LED on Pin 13 of Arduino
int pirPin = 7; // Input for HC-S501

int pirValue; // Place to store read PIR Value

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(pirPin, INPUT);
  digitalWrite(ledPin, LOW);
}

void loop() {
  pirValue = digitalRead(pirPin);
  digitalWrite(ledPin, pirValue);
}
```

4. Results

- The Arduino will monitor the output of PIR sensor.
- When motion is detected, the LED connected to pin 13 will turn on.



5. Conclusion

In conclusion, the integration of a PIR sensor with an Arduino offers a straightforward and effective solution for motion detection applications. Studies have shown that this combination significantly enhances motion detection capabilities, demonstrating high accuracy rates in various environmental conditions while remaining cost-effective and user-friendly. Its passive nature and simple interface make it accessible for various projects, providing a reliable method for detecting movement efficiently and affordably.

6. References

Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

https://youtu.be/FxaTDvs34mM?si=FdGuMT7ec3soOx0e

ASSIGNMET #12: MEMBRANE SWITCH MODULE

1. Introduction

In this project, we will go over how to integrate a keyboard with an UNO R3 board so that the UNO R3 can read the keys being pressed by a user.

Keypads are used in all types of devices, including cell phones, microwaves, ovens, door locks, etc. They're practically everywhere. Tons of electronic devices use them for user input.

2. Required components

a. Component 1

(1) x Uno R3 Board: it is a microcontroller board. It has 20 digital input/output pins of which 6 can be used as PWM outputs and 6 can be used as analog inputs. Programs can be loaded on to it from the Arduino IDE.

b. Component 2

(1) x Membrane switch module: the membrane switch module is a 4x4 matrix keypad where numbers from 0 till 9 and letters A till D as well as the symbols '*' and '#' are present on it as buttons with the ability to press them.

c. Component 3

(8) x M-M wires (Male to Male jumper wires): wires used to transfer electricity and obtain the wanted connections.

3. Implementation Steps

Firstly, we prepare the needed components. Then, we make the necessary connections as present in the figures below. Next, we start working on the code, downloading the libraries needed if any and showing the serial monitor, and make it as to get the required objective of getting the keys pressed by the user. Finally, we upload the finished code to the Arduino board and make sure that it is working correctly on the serial monitor.

a. Connection diagram

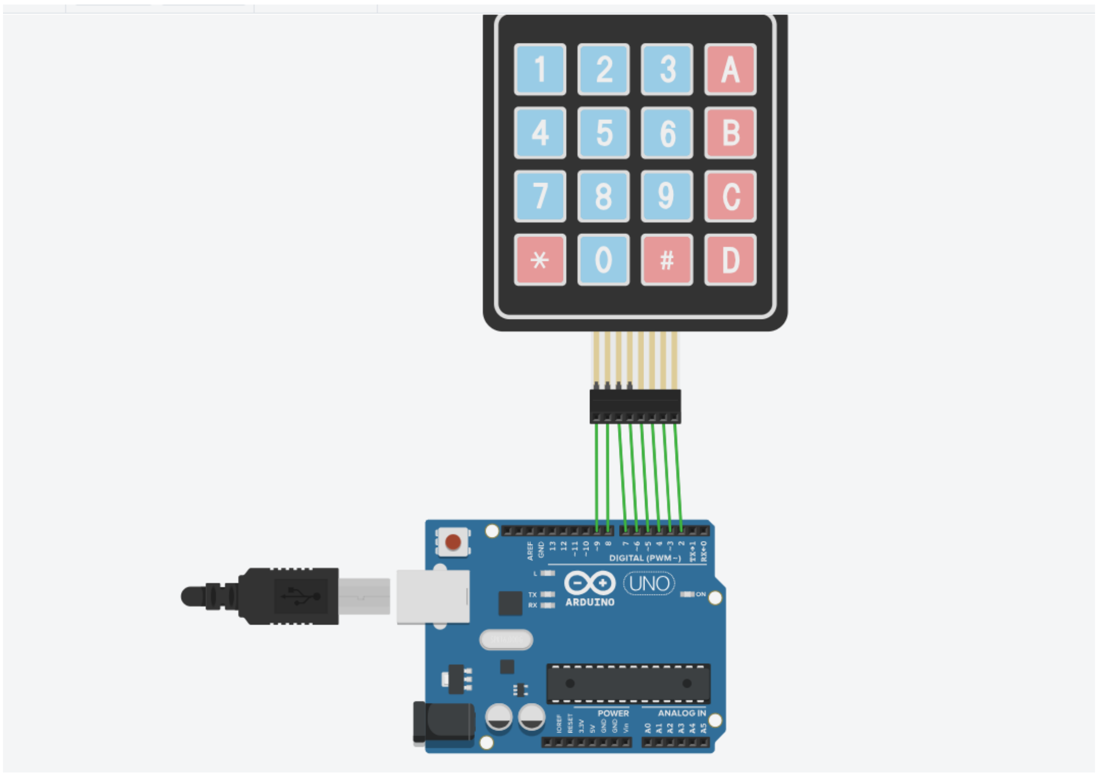
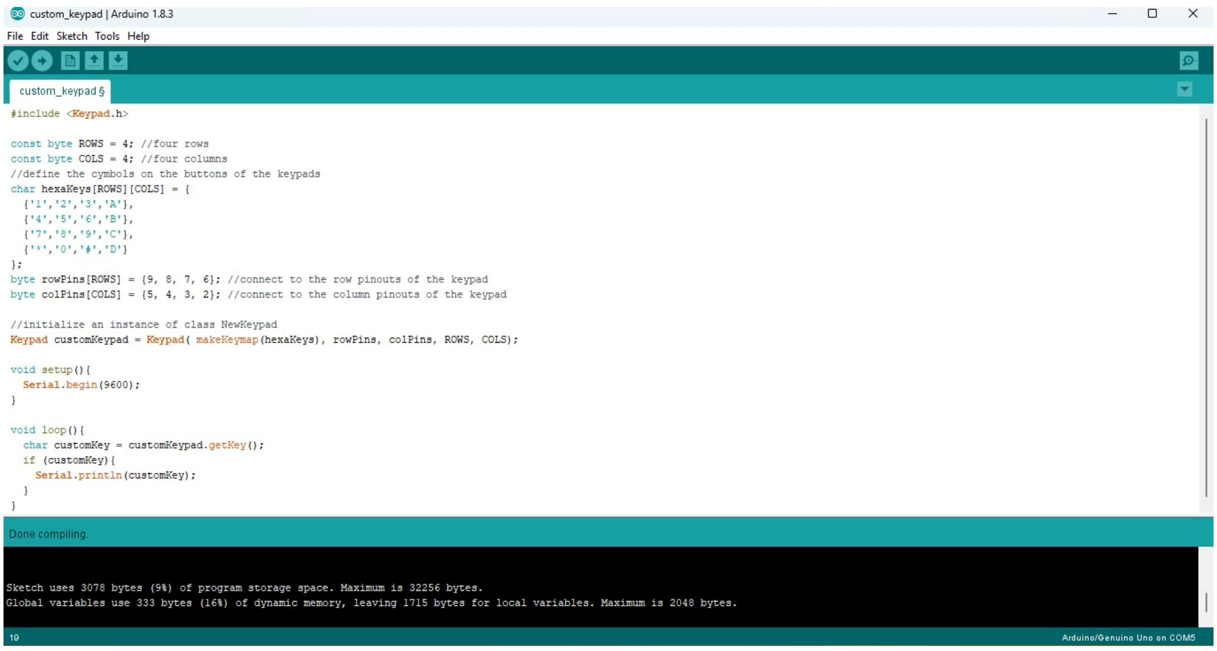


Figure 1. Connections on Tinker Cad

b. Code

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);
```

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

void loop(){
  char customKey = customKeypad.getKey();

  if (customKey){
    Serial.println(customKey);
  }
}
```



```
#include <Keypad.h>

const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
//define the cymbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; //connect to the column pinouts of the keypad

//initialize an instance of class NewKeypad
Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

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

void loop(){
  char customKey = customKeypad.getKey();
  if (customKey){
    Serial.println(customKey);
  }
}
```

Done compiling.

```
Sketch uses 3078 bytes (9%) of program storage space. Maximum is 32256 bytes.
Global variables use 333 bytes (16%) of dynamic memory, leaving 1715 bytes for local variables. Maximum is 2048 bytes.
```

Figure 2. Code in Arduino IDE

4. Results

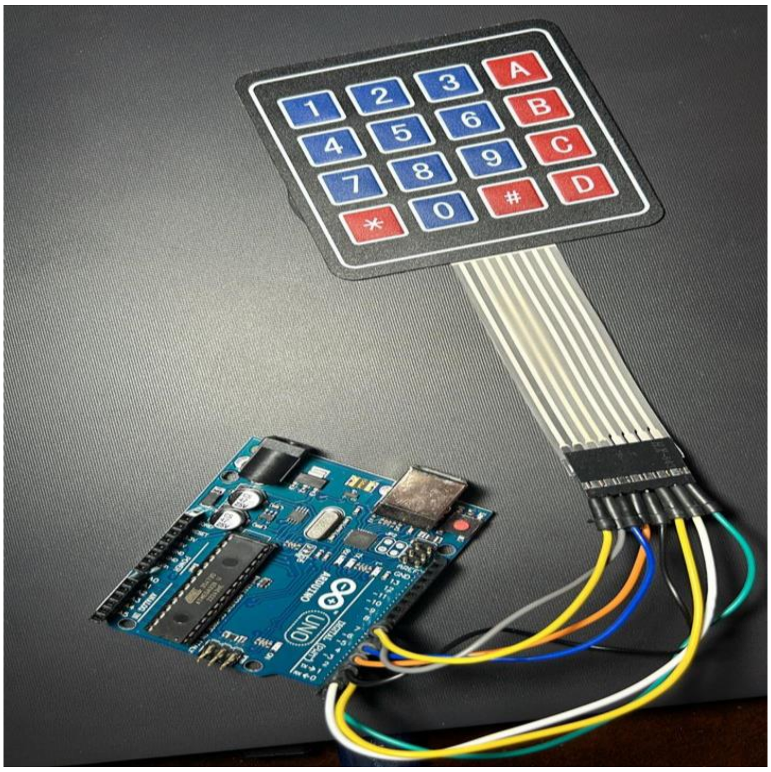
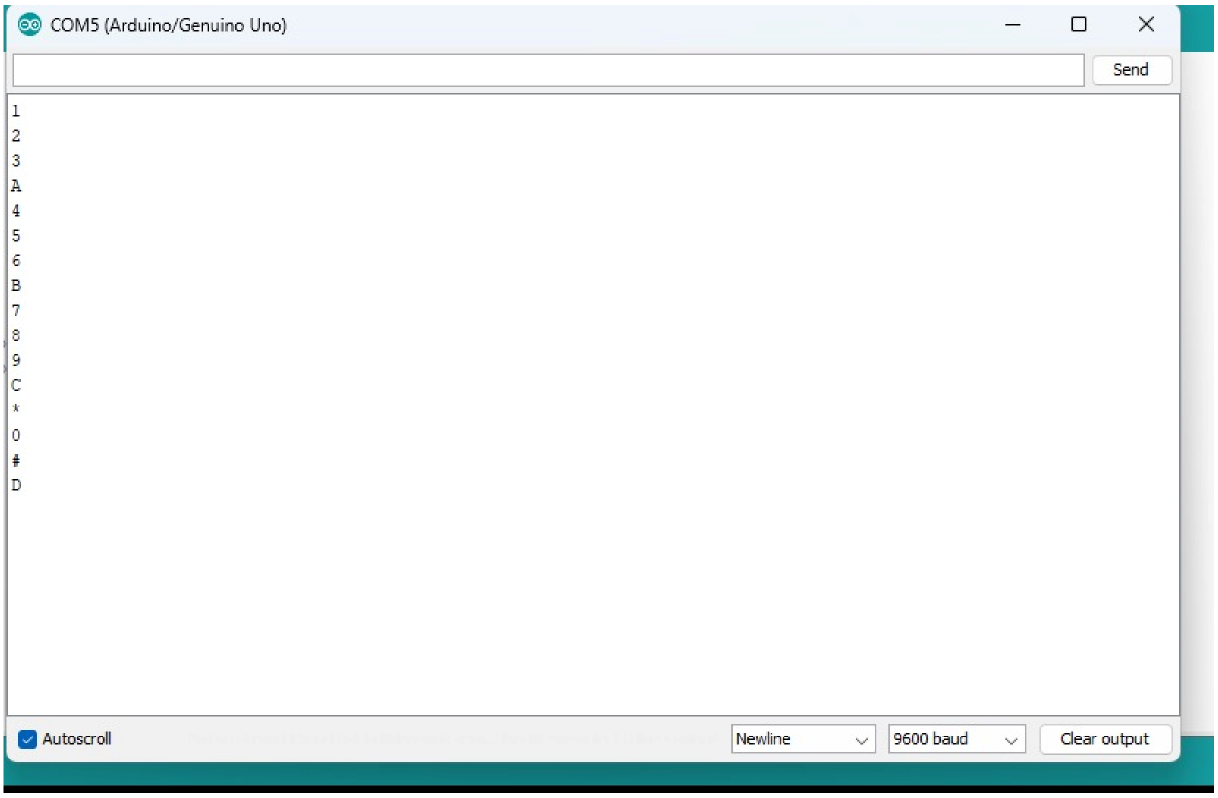


Figure 3. Connections in real time



```
1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
```

Newline
9600 baud

Figure 4. Results on serial monitor

5. Conclusion

This kit has allowed us to learn about the compatibility and integration of software and hardware together. As well as the importance and opportunities that can be created with them.

This lab specifically has helped us in learning to use the serial monitor, as well as identify how a keypad works and its many uses and ideas that it can be implemented in.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #13: GY-521 MODULE

1. Introduction

The module has MPU6050 on it, which has a 3-axis accelerometer, a 3-axis gyroscope, and a Digital Motion Processor on it. This module works on the principle of MEMS (Micro electro-mechanical system).

Before running into the details of working with the module, first, we should have an idea about what an accelerometer or a gyroscope is: it gives a measure of acceleration along all 3 axes- x,y, and z.

Every accelerometer module has a micro-electromechanical structure that suffers a displacement due to acceleration this displacement is sensed in terms of a change in capacitance which changes out voltage given by the accelerometer unit. The GY-521 module has registers that store the acceleration value of all axes.

Similarly, the gyroscope gives the amount of rotation along each of the x,y, and z-axes. Rotation along x,y, and z-axes are popularly known as roll, pitch, and yaw respectively. The Gyroscope sensor also has a special structure that vibrates due to the rotation of the sensor leading to a change in the capacitance and hence output voltage changes.

2. Required components

a. Uno R3 Board

b. GY-521 module

c. F-M wires

To connect The GY-521 module to Arduino.

d. LCD

To display the result.

3. Implementation Steps

Using jumper wires, connect the GY-521 and LCD to the appropriate pins on the Arduino board. For the GY-521, connect VCC to 5V, GND to GND, SDA to A4, and SCL to A5. For the LCD, connect VCC to 5V, GND to GND, and SDA/SCL to A4/A5 respectively if it supports I2C communication.

Once the hardware connections are made, ensure that the necessary libraries are installed in your Arduino IDE. If using an I2C-enabled LCD, make sure the LiquidCrystal_I2C library is installed. Additionally, install any required libraries for the GY-521 module depending on the functionalities you intend to use.

a. Connection diagram

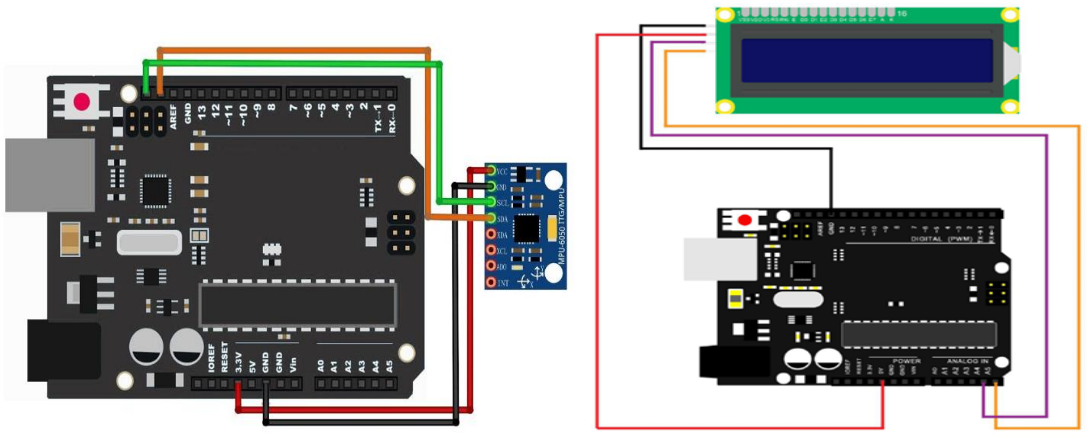


Figure 1: Wiring Diagram

b. Code

```
// MPU-6050 Short Example Sketch

#include<Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  lcd.init(); // initialize the lcd
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for temperature in degrees C from
datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);
lcd.backlight();
lcd.setCursor(10,0);
lcd.print("T=");
lcd.setCursor(12,0);
lcd.print(Tmp/340.00+36.53);
lcd.setCursor(0,0);
lcd.print("AcX=");
lcd.setCursor(4,0);
lcd.print(AcX);
lcd.setCursor(0,1);
lcd.print("GyX=");
lcd.setCursor(4,1);
lcd.print(GyX);
delay(2000);
lcd.init();
lcd.setCursor(10,0);
lcd.print("T=");
lcd.setCursor(12,0);
lcd.print(Tmp/340.00+36.53);
lcd.setCursor(0,0);
lcd.print("AcY=");
lcd.setCursor(4,0);
lcd.print(AcY);
lcd.setCursor(0,1);
lcd.print("GyY=");
lcd.setCursor(4,1);
lcd.print(GyY);
delay(2000);
lcd.init();
lcd.setCursor(10,0);
lcd.print("T=");
lcd.setCursor(12,0);
lcd.print(Tmp/340.00+36.53);
lcd.setCursor(0,0);
lcd.print("AcZ=");
lcd.setCursor(4,0);
lcd.print(AcZ);
lcd.setCursor(0,1);
lcd.print("GyZ=");
lcd.setCursor(4,1);
lcd.print(GyZ);

//lcd.setCursor(7,0);
//lcd.print("Tmp=");
//lcd.setCursor(12,0);
//lcd.print(Tmp/340.00+36.53);
  delay(2000);
  lcd.init();
}
```

4. Conclusion

In conclusion, through the process of interfacing with the GY-521 and an LCD, I have gained valuable insights and skills. This experience has not only enhanced my understanding of hardware integration but has also equipped me with the ability to effectively communicate between components in a simple yet efficient manner

5. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] STEMpedia,Tutorial, https://aithestmpedia.com/

ASSIGNMET #14: 74HC595 SHIFT REGISTER

1. Introduction

This report explores the integration of the 74HC595 shift register to efficiently control eight LEDs within a circuit. By leveraging the shift register's serial-in, parallel-out functionality, the circuit simplifies LED control, reducing wiring complexity while providing practical insights into sequential and simultaneous LED operation..

2. Required components

(1) x Uno R3 Board

(1) x 830 tie-points breadboard

(8) x leds

(8) x 1K ohm resistors

(1) x 10 ohm resistor

(1) x 74hc595 IC

(14) x M-M wires (Male to Male jumper wires)

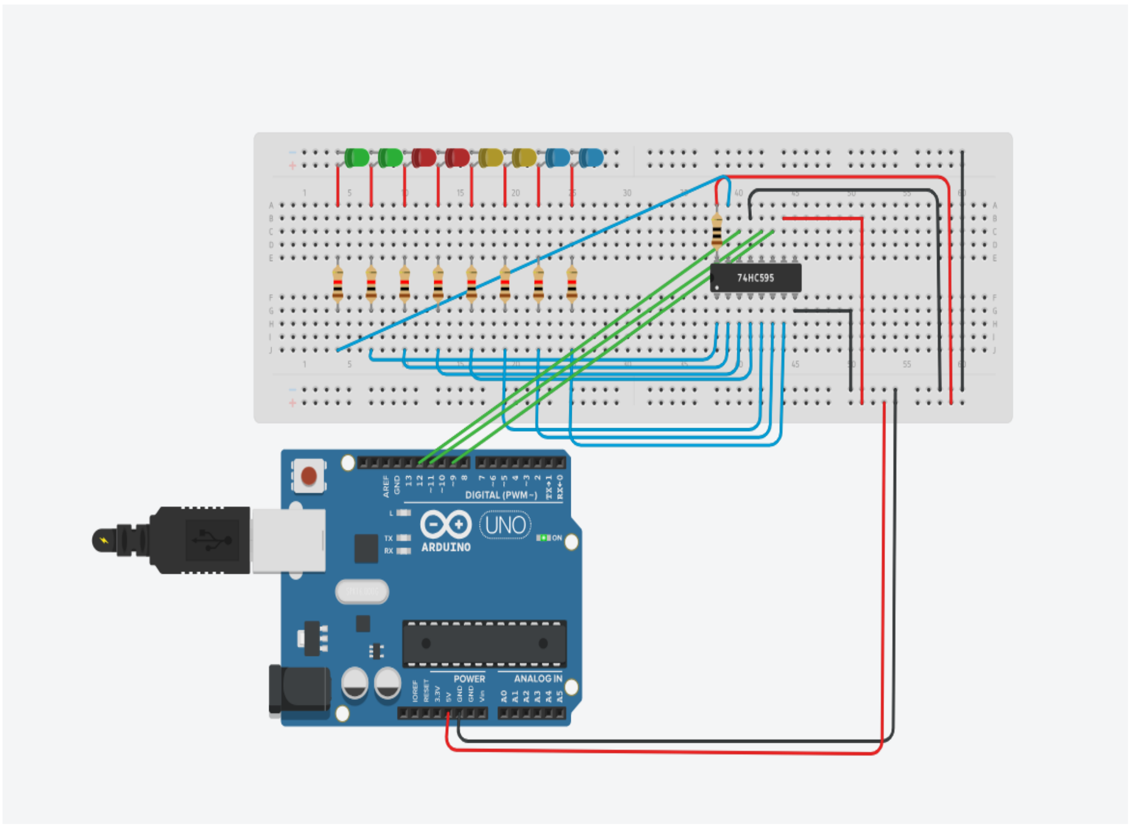
3. Implementation Steps

a. Connection diagram

- Insert the 74HC595 shift register into the breadboard with the notch facing upwards, ensuring it's securely positioned.
- Use jumper wires to connect digital pins 12, 11, and 9 from the Arduino UNO to pins 14, 12, and 11 of the shift register respectively.
- Connect the latch and output enable pins of the shift register as required for stable LED output and brightness control.
- Connect the Vcc pin of the shift register to a 10 ohm resister to prevent it from damaging.
- Connect one end of each resistor to the cathode (shorter lead) of each LED, and the other end to the ground (GND) rail of the breadboard. Connect the anode (longer lead) of each LED to an output pin of the shift register.

- Ensure all connections are secure and there are no short circuits or loose connections that could cause issues.
- Load up the sketch listed a bit later and try it out. Each LED should light in turn until all the LEDs are on, and then they all go off and the cycle repeats.

Online simulator implementation:

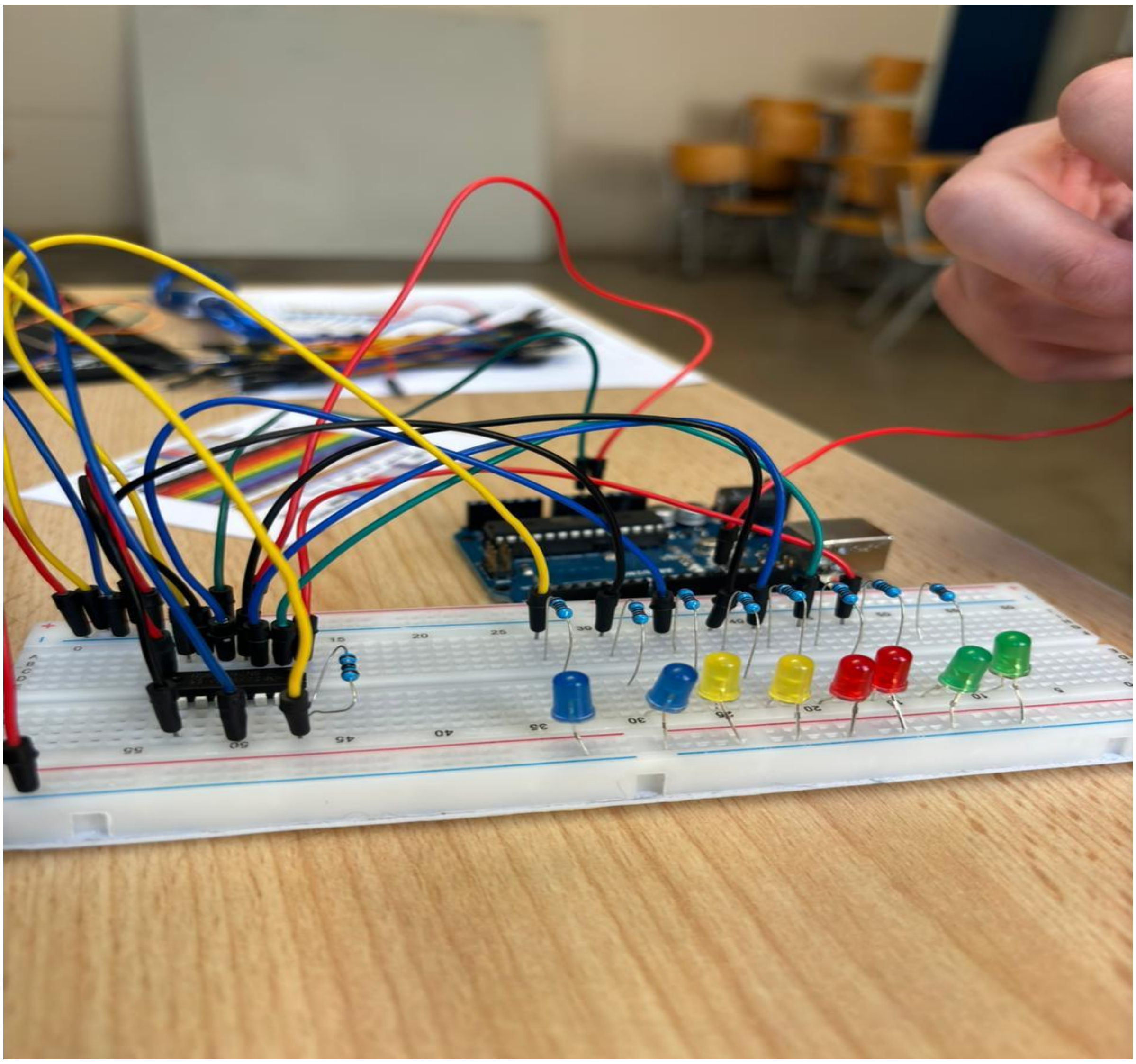


b. Code

```
int tDelay = 100;
int latchPin = 11;      // (11) ST_CP [RCK] on 74HC595
int clockPin = 9;      // (9) SH_CP [SCK] on 74HC595
int dataPin = 12;     // (12) DS [S1] on 74HC595

byte leds = 0;

void updateShiftRegister()
{
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, leds);
  digitalWrite(latchPin, HIGH);
}

void setup()
{
 pinMode(latchPin, OUTPUT);
```

```
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

void loop()
{
  leds = 0;
  updateShiftRegister();
  delay(tDelay);
  for (int i = 0; i < 8; i++)
  {
    bitSet(leds, i);
    updateShiftRegister();
    delay(tDelay);
  }
}
```

4. Results



5. Conclusion

Using this kit we've learned how can we control 8 leds using shift register with 16 pins having 8 as output pins which are pin Q_0 till pin Q_7 by which a delay time of 100ms is presented between each led and another while operating.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] Circuit design Surprising Bojo-Jarv - Tinkercad .

ASSIGNMENT #15: FOUR DIGITAL SEVEN SEGMENT DISPLAY

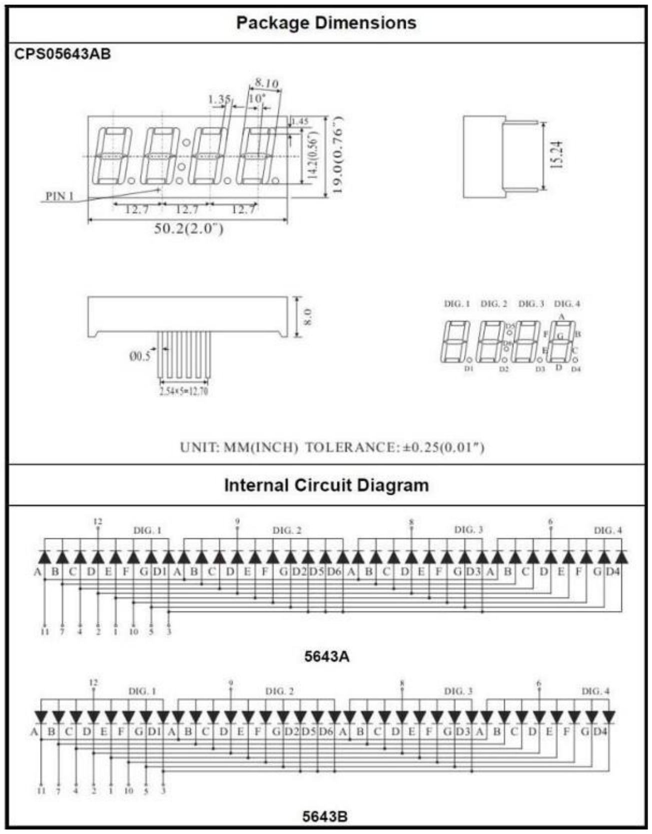
1. Introduction

In this lesson, we will use a 4-digit 7-segment display. We will display numbers from 1 to 9 followed by letters from A to K (which represent numbers from 10 to 15). The duration between each display is half a second.

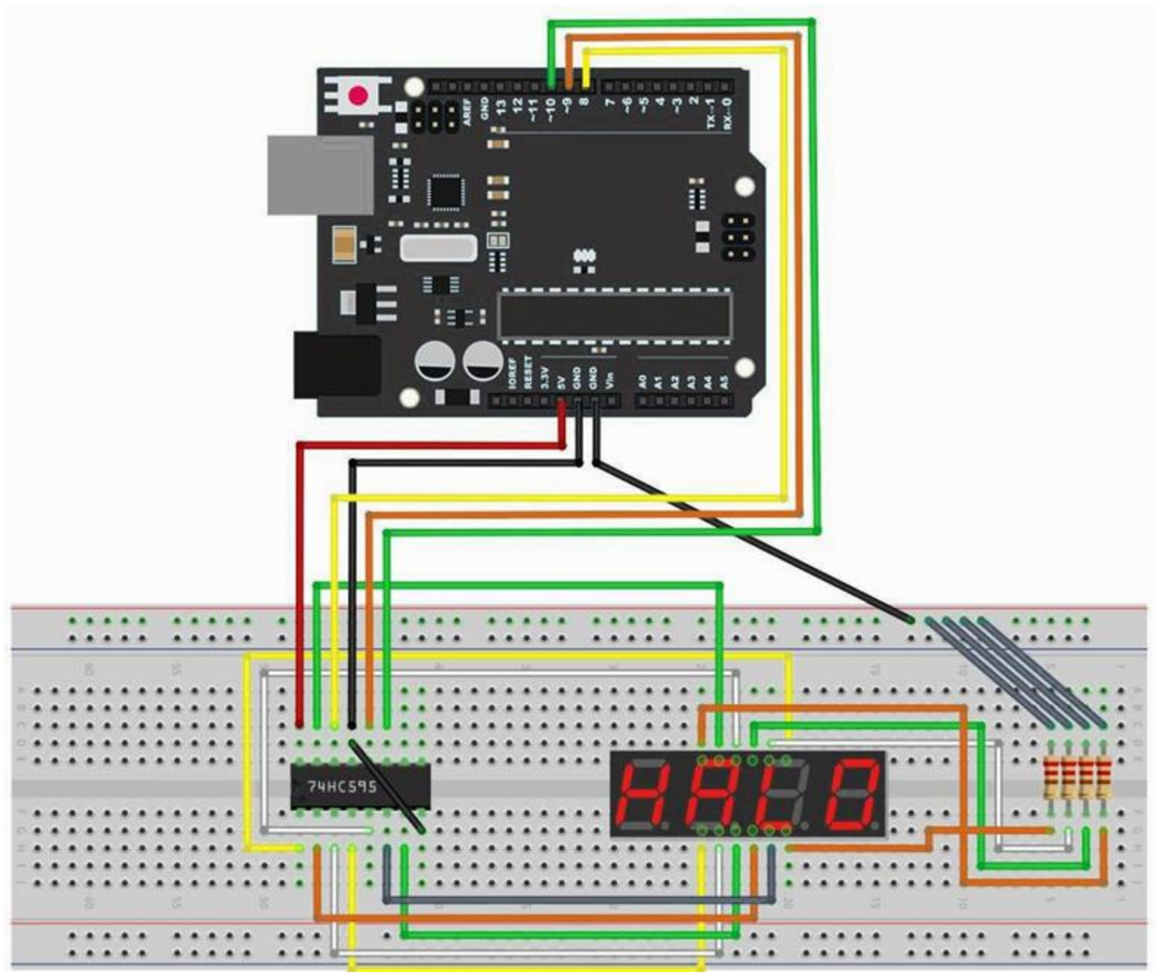
Component Required:

- (1) Uno R3 Board
- (1) 830 tie-points breadboard
- (1) 74HC595 IC
- (1) 4 Digit 7-Segment Display
- (4) 220 ohm resistors
- (23) M-M wires (Male to Male jumper wires)

Component Introduction: Four Digital Seven segment display



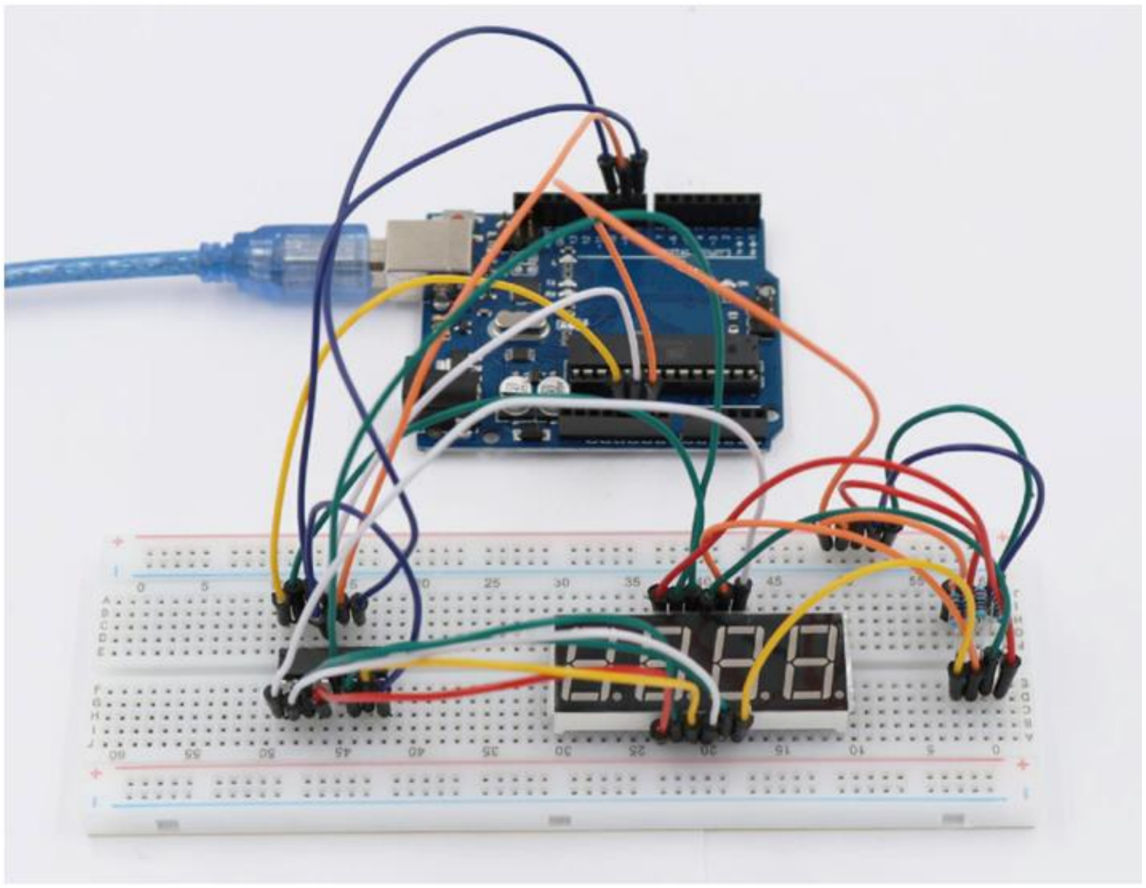
Wiring diagram :



Arduino Code:

```
int latch=9;  //74HC595  pin 9 STCP
int clock=10; //74HC595  pin 10 SHCP
int data=8;   //74HC595  pin 8 DS

unsigned char table[]=
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c
,0x39,0x5e,0x79,0x71,0x00};

void setup() {
  pinMode(latch,OUTPUT);
  pinMode(clock,OUTPUT);
  pinMode(data,OUTPUT);
}
void Display(unsigned char num)
{

  digitalWrite(latch,LOW);
  shiftOut(data,clock,MSBFIRST,table[num]);
  digitalWrite(latch,HIGH);

}
void loop() {
  Display(1);
  delay(500);
  Display(2);
  delay(500);
  Display(3);
  delay(500);
  Display(4);
  delay(500);
  Display(5);
  delay(500);
  Display(6);
  delay(500);
  Display(7);
  delay(500);
  Display(8);
  delay(500);
  Display(9);
  delay(500);
  Display(10);
  delay(500);
  Display(11);
  delay(500);
  Display(12);
  delay(500);
  Display(13);
  delay(500);
  Display(14);
  delay(500);
  Display(15);
  delay(500);
}
```

Example Picture :



ASSIGNMET #16: STEPPER MOTOR WITH ROTARY ENCODER

1. Introduction

Stepper motors are a type of electric motor that divide a full rotation into discrete steps. This precise positioning ability makes them ideal for a variety of applications, from 3D printers and CNC machines to robotic arms and disk drives. However, for these applications to function accurately, the motor's position needs to be constantly monitored. This is where encoders come in.

Encoders are sensors that are attached to the shaft of a stepper motor. They detect the motor's rotation and provide feedback on its angular position. This feedback loop allows for much more precise control of the motor's movement compared to simply sending step pulses.

mhamad rayed: In this project we are testing and programming a simple stepper motor.

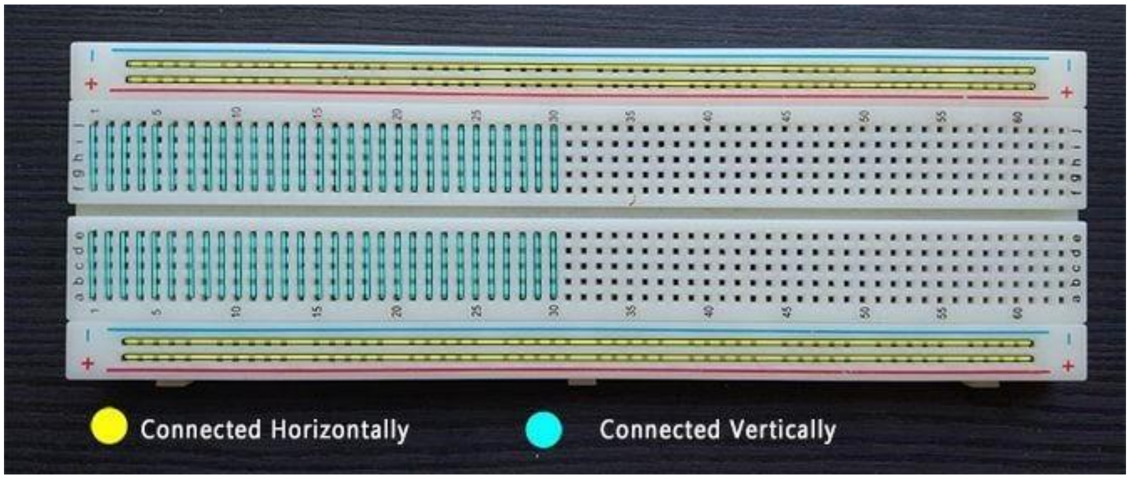
2. Required components

a. Uno R3 board

The Arduino Uno R3 is a widely popular open-source microcontroller board based on the ATmega328P microcontroller.The board provides 14 digital input/output (I/O) pins, among which 6 can be used as PWM (Pulse Width Modulation) output pins. There are also 6 analog input pins, enabling you to interface with analog sensors or read voltages from the environment.

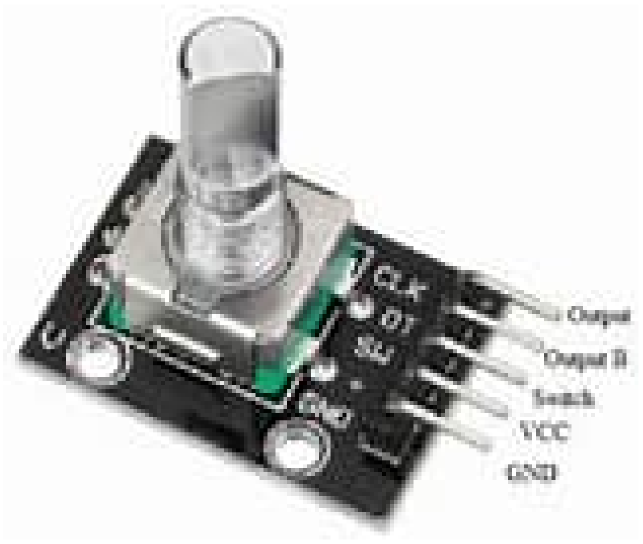
b. 830 tie_points breadboard

A breadboard is a construction base for electronics prototyping. It might look like a regular plastic board with many tiny holes in it, but it's much more than that. It is one of the main tools for circuit building and electronics in general.



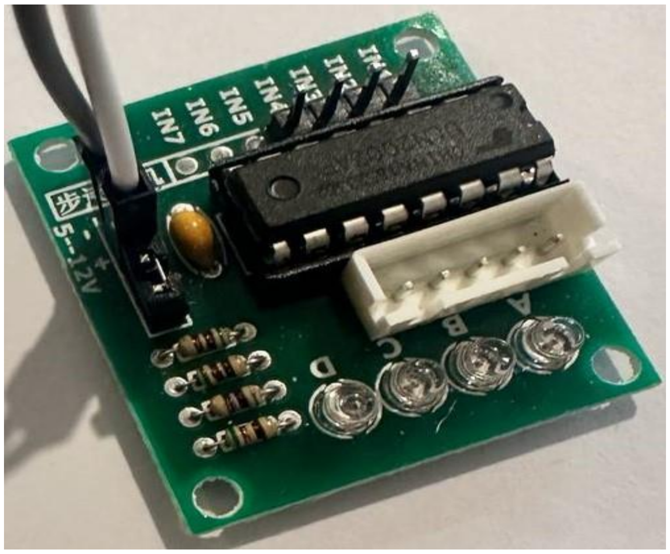
c. Rotary encoder

Rotary encoders are sensors which sense the rotation of a central shaft. Unlike a rotary potentiometer, encoders can turn infinitely, covering the full 360 degrees of the shaft's rotation. Many have built-in pushbuttons as well.



d. ULN2003 stepper motor driver

A stepper motor driver is considered as the driver circuit which facilitates the motor to operate in the method that it works. For instance, these motors need an adequate and regulated amount of energy for phases in the exact order. Because of this, the stepper motor drivers are termed to be more improvised versions of motors than the traditional types of motors.



e. Stepper motor

A stepper motor, also known as step motor or stepping motor, is an electrical motor that rotates in a series of small angular steps, instead of continuously. Stepper motors are a type of digital actuator. Like other electromagnetic actuators, they convert electric energy into mechanical position.

f. Power supply

g. 9V1A adapter

h. F-M wires (female to male dupont wires)

i. M-M wires (male to male jumper wires)

3. Implementation Steps

a. Connect the motor driver pins IN1-IN2-IN3-IN4 respectively to arduino digital pins 8-9-10-11

b. Connect the positive pole and GND of rotary encoder respectively to arduino pins 5V and GND

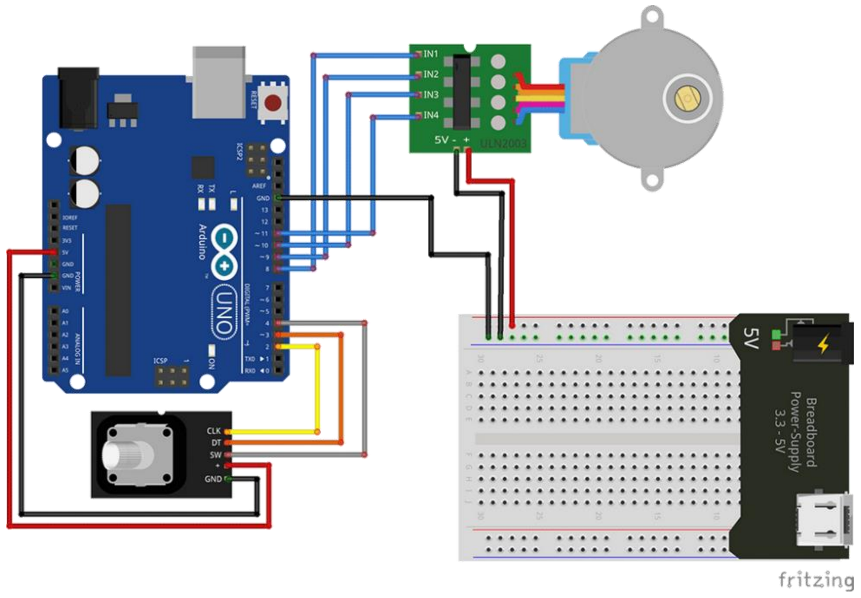
c. Connect the other three pins SW (push button), DT (pin B) and CLK (pin A) of rotary encoder respectively to arduino digital pins 4-3-2

d. Connect the positive and negative pins of motor driver respectively to breadboard positive and negative poles, then connect the arduino GND to breadboard GND

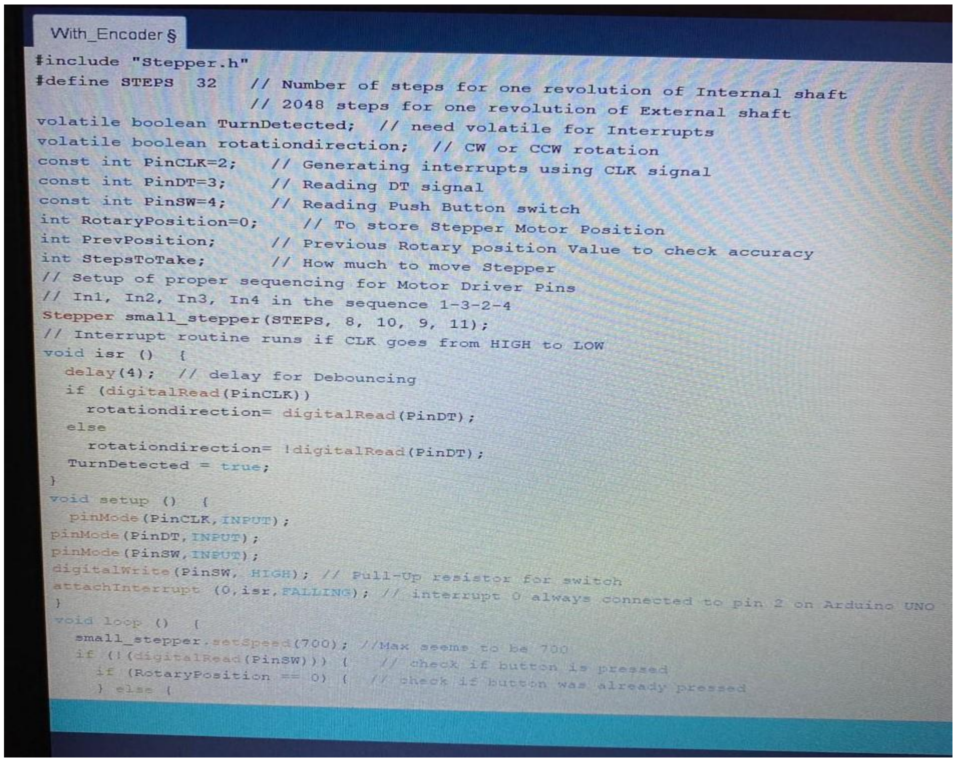
e. Connect the stepper motor to the motor driver, then connect the power supply to the breadboard in order to feed the motor driver and feed the power supply by the 9V1A adapter

f. write the code and upload it to the arduino using arduino IDE application

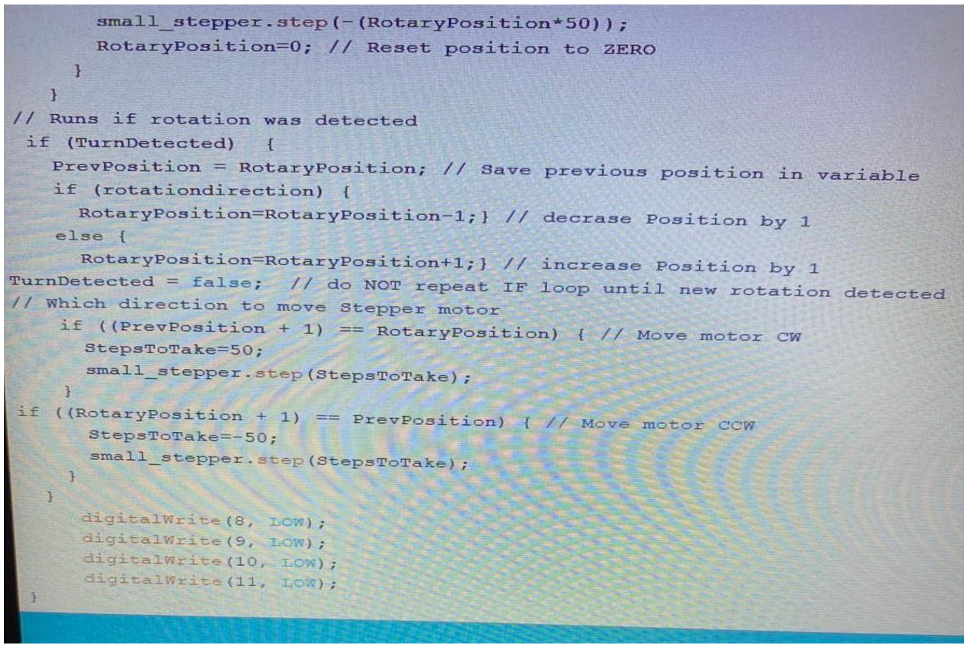
A. Wiring diagram



B. Code



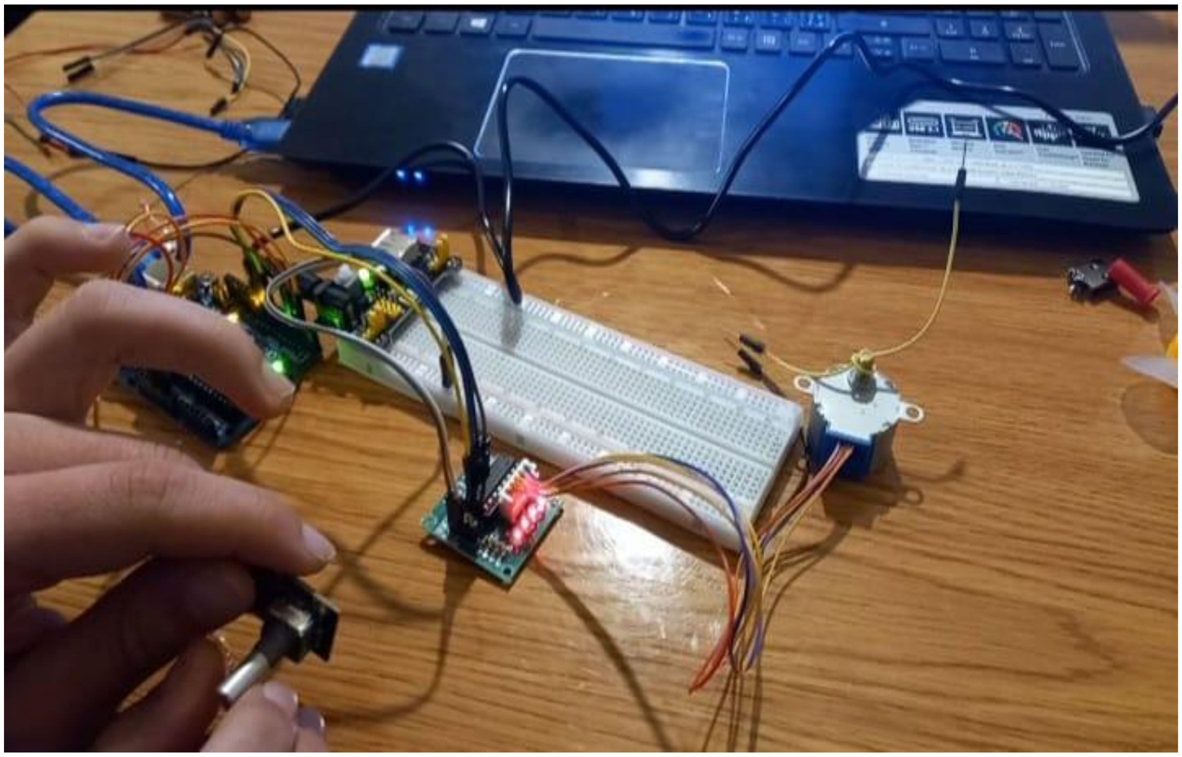
```
#include "Stepper.h"
#define STEPS  32   // Number of steps for one revolution of Internal shaft
                    // 2048 steps for one revolution of External shaft
volatile boolean TurnDetected;  // need volatile for Interrupts
volatile boolean rotationdirection;  // CW or CCW rotation
const int PinCLK=2;   // Generating interrupts using CLK signal
const int PinDT=3;    // Reading DT signal
const int PinSW=4;    // Reading Push Button switch
int RotaryPosition=0;    // To store Stepper Motor Position
int PrevPosition;     // Previous Rotary position Value to check accuracy
int StepsToTake;      // How much to move Stepper
// Setup of proper sequencing for Motor Driver Pins
// In1, In2, In3, In4 in the sequence 1-3-2-4
Stepper small_stepper(STEPS, 8, 10, 9, 11);
// Interrupt routine runs if CLK goes from HIGH to LOW
void isr ()  {
  delay(4);  // delay for Debouncing
  if (digitalRead(PinCLK))
    rotationdirection= digitalRead(PinDT);
  else
    rotationdirection= !digitalRead(PinDT);
  TurnDetected = true;
}
void setup ()  {
  pinMode(PinCLK,INPUT);
pinMode(PinDT,INPUT);
pinMode(PinSW,INPUT);
digitalWrite(PinSW, HIGH); // Pull-Up resistor for switch
attachInterrupt (0,isr,FALLING); // interrupt 0 always connected to pin 2 on Arduino UNO
}
void loop ()  {
  small_stepper.setSpeed(700); //Max seems to be 700
  if (!(digitalRead(PinSW))) {   // check if button is pressed
    if (RotaryPosition == 0) {  // check if button was already pressed
    } else {
```



```
      small_stepper.step(-(RotaryPosition*50));
      RotaryPosition=0; // Reset position to ZERO
    }
  }
// Runs if rotation was detected
 if (TurnDetected)  {
   PrevPosition = RotaryPosition; // Save previous position in variable
   if (rotationdirection) {
     RotaryPosition=RotaryPosition-1;} // decrase Position by 1
   else {
     RotaryPosition=RotaryPosition+1;} // increase Position by 1
TurnDetected = false;  // do NOT repeat IF loop until new rotation detected
// Which direction to move Stepper motor
    if ((PrevPosition + 1) == RotaryPosition) { // Move motor CW
      StepsToTake=50;
      small_stepper.step(StepsToTake);
    }
 if ((RotaryPosition + 1) == PrevPosition) { // Move motor CCW
      StepsToTake=-50;
      small_stepper.step(StepsToTake);
    }
  }
     digitalWrite(8, LOW);
     digitalWrite(9, LOW);
     digitalWrite(10, LOW);
     digitalWrite(11, LOW);
}
```

4. Results

Now we obtain the hardware and we can control the unipolar stepper motor using programmed Arduino UNO board and rotary encoder module.The motor rotates at a certain speed with the rotary encoder rotated manually, and it rotates in the opposite direction with the rotary encoder rotated in the opposite direction.



5. Conclusion

This laboratory provided valuable hands-on experience with stepper motors and their control using Arduino. We delved into the assembly process, gaining a practical understanding of the motor's physical components. Furthermore, we explored the programming aspect, learning to send control signals and interact with the motor through Arduino code. A key takeaway was the process of debugging errors encountered during the project. This troubleshooting process is a crucial skill for working with electronic components and microcontrollers. By successfully navigating these challenges,

we gained confidence in our ability to apply this knowledge to real-world projects that utilize stepper motors.

6. Reference

Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

ASSIGNMET #17: LASER SECURITY SYSTEM

1. Introduction

In today's world, security is a top priority. However, traditional security systems can be expensive and complex. That is where Arduino-based projects come in. One such project is the laser security system.

This system uses simple components like lasers, light sensors, and buzzers, all controlled by an Arduino microcontroller. When an object breaks the laser beam, the sensor detects the change in light intensity and triggers the buzzer, alerting of a potential intrusion.

In this report, we will explore how to design and implement this laser security system. It's a hands-on project that teaches us about electronics, programming, and the practical applications of security systems in everyday life.

2. Required components

a. Arduino Uno Microcontroller:

The brain of the system, the Arduino controls the operation of the entire setup. It receives input from the sensor, processes it, and triggers the buzzer when necessary.

b. Laser Module:

Emits a thin laser beam used as the tripwire. When uninterrupted, the laser beam falls directly onto the Light Dependent Resistor (LDR), maintaining a continuous circuit.

c. Light Dependent Resistor (LDR):

Detects changes in light intensity. When the laser beam is obstructed by an object, the light falling on the LDR decreases, causing its resistance to increase. The Arduino detects this change in resistance.

d. LED:

An additional visual indicator. It flashes when the buzzer is activated, providing a visual cue alongside the audible alert.

e. Active Buzzer:

Produces an audible alarm when triggered by the Arduino. In this setup, the buzzer alerts of an intrusion by emitting a buzzing sound when the laser beam is interrupted.

f. Connecting Wires:

They are used to establish connections between the Arduino board pins, the sensors (laser module and LDR), and the buzzer.

3. Implementation Steps

1. Prepare Components: Gather all the necessary components including the Arduino Uno, laser module, LDR, buzzer, LED, and connecting wires.

2. Connect Components to Arduino: Use the connecting wires to establish connections between the components and the Arduino Uno. Connect the laser module to 5V and GND pins, the LDR to an analog pin (e.g., A0), the buzzer to another digital pin (e.g., pin 3), and the LED to another digital pin (e.g., pin 4).

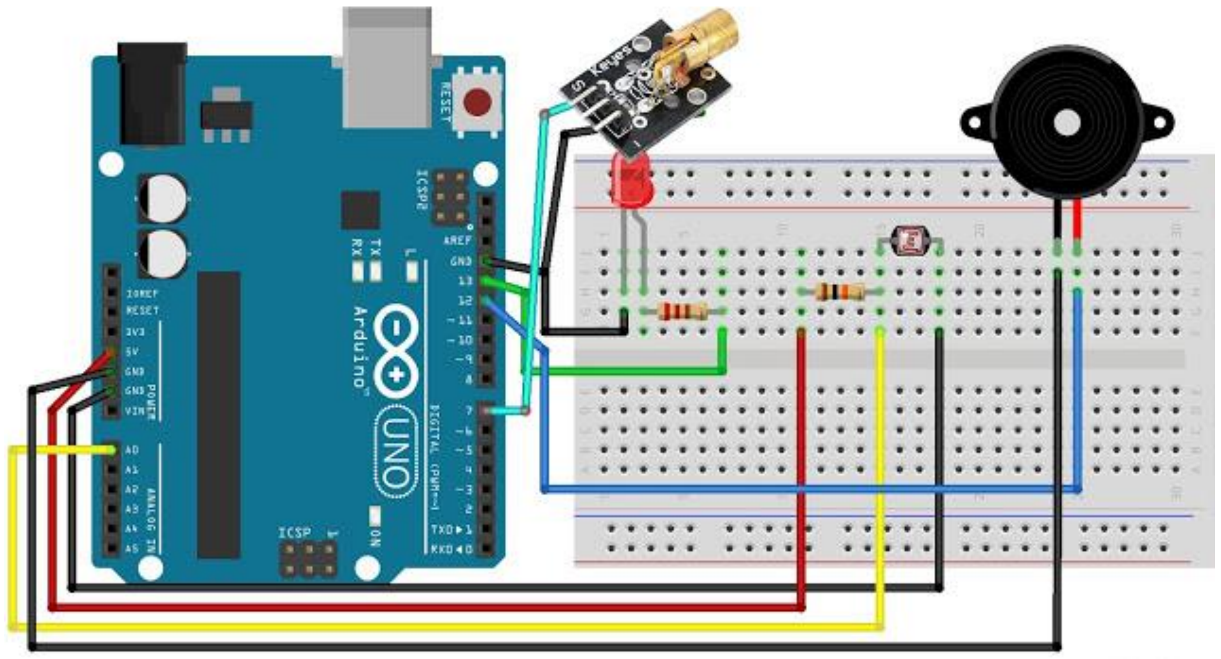
3. Write Arduino Sketch: Develop the Arduino sketch (program) to define the behavior of the system. This includes initializing pins, reading inputs from the LDR, detecting interruptions in the laser beam, activating the buzzer and LED when an intrusion is detected, and controlling the duration and frequency of the alerts.

4. Upload Sketch to Arduino: Connect the Arduino Uno to your computer using a USB cable and upload the developed Arduino sketch to the board using the Arduino IDE.

5. Test the System: Place the laser module and LDR in a suitable position, ensuring that the laser beam is aimed at the LDR. Observe the system's behavior as you interrupt the laser beam. Verify that the buzzer and LED activate appropriately, when an intrusion is detected.

6. Adjust Sensitivity and Timing: Fine-tune the system parameters as needed to achieve the desired sensitivity and response time. This may involve adjusting the placement of components, modifying the Arduino sketch, or tweaking hardware configurations.

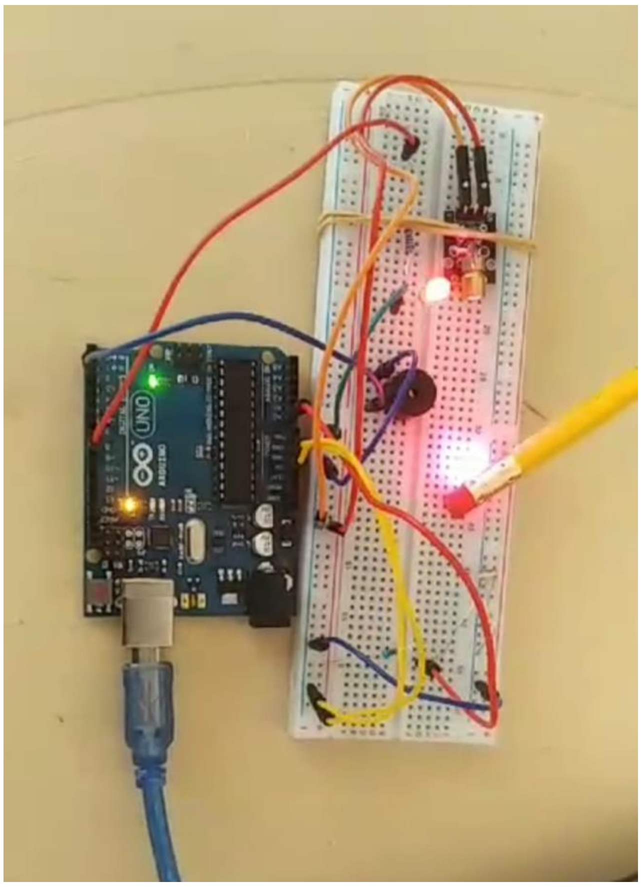
a. Connection diagram



b. Code

```
int  sensorPin  =  A0;     // select the input  pin for  the ldr
int buzzerpin = 2;  // select the pin for  the LED
int  sensorValue =  0;  // variable to  store  the value  coming  from  the
sensor
int ledPin= 8;

void setup()
{
pinMode(buzzerpin,OUTPUT);
pinMode(ledPin,OUTPUT);
Serial.begin(9600);
}
void loop(){
sensorValue =  analogRead(sensorPin);
Serial.println(sensorValue);

if(sensorValue > 20)
{
  digitalWrite(buzzerpin, HIGH);
  digitalWrite(ledPin, HIGH);
  delay(200);
  digitalWrite(buzzerpin, LOW);
  digitalWrite(ledPin,LOW);
  delay(200);
  digitalWrite(buzzerpin, HIGH);
  digitalWrite(ledPin, HIGH);
  delay(200);
  digitalWrite(buzzerpin, LOW);
  digitalWrite(ledPin, LOW);
  delay(200);
}
else{
  digitalWrite(buzzerpin, LOW);
  digitalWrite(ledPin, LOW);
}
}
```

4. Results



5. Conclusion

In wrapping up, working on the laser security system with Arduino was educational. We picked up skills in sensor connections, security system setup, and component usage. Though we encountered challenges, troubleshooting them improved our understanding. Overall, it was an enriching experiment.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] 37 Sensor Kit Tutorial, https://lafvintech.com/

ASSIGNMENT #18 SERVO MOTOR CONTROL USING ANALOG SENSOR INPUT

1. Introduction

In this project, we aim to create a system that controls a servo motor based on analog input from a sensor. The project integrates Arduino programming and servo motor control to achieve a specific objective.

2. Required components

The required components are:

a. Arduino Uno

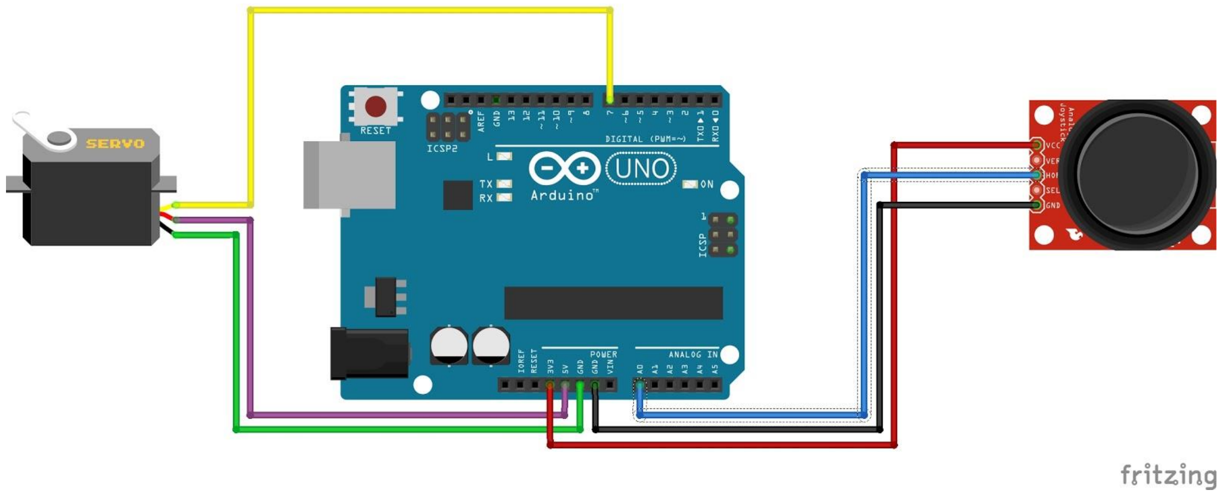
b. Servo Motor

c. Joystick

d. Jumper Wires

3. Implementation Steps

a. Connection diagram



b. Code

```
#include <Servo.h>
const int InputPin = 7;
const int Y_pin = A0;
Servo myservo;
int val;

void setup() {
myservo.attach(InputPin);
}

void loop() {
  val= analogRead(Y_pin);
  val= map(val,0,1023,0,180);
  myservo.write(val);
  delay(15);
}

```

21. Results

The system was successfully developed and tested, demonstrating effective control of a servo motor based on analog input from a sensor.

22. Conclusion

In conclusion, the project successfully achieved its objectives of developing a system for controlling a servo motor using analog input from a sensor. By leveraging Arduino programming and servo motor control techniques, we created a versatile and responsive system capable of precise motor positioning based on environmental input.

23. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] Arduino Book for Beginners, Mike Cheich

ASSIGNMET #19: INTERACTIVE SYSTEM

1. Introduction

The goal is to create an interactive system where the LED respond to the distance of an object detected by the ultrasonic sensor. The problem statement: Design and build a system that uses an ultrasonic sensor to measure the distance to an object. Based on the distance, control a set of LED and display a notification on an LCD screen and a buzzing sound.

2. Required components

a. **Ultrasonic Sensor :** calculates the distance to a target by emitting ultrasonic sound waves and converting those waves into electrical signals.

b. **LED :** it emits light

c. LCD Screen

d. Arduino

e. Resistors : protects electrical components

f. Wires

g. **Breadboard :** connect components together within the board.

h. **Buzzer:** produces sounds with certain frequencies.

3. Implementation Steps

Connect 5V of Arduino to positive of breadboard

Connect GND of Arduino to negative of breadboard

Pin Connections of **Ultrasonic Sensor**:

- Connect the sensor's TRIG pin to Arduino pin 12.
- Connect the sensor's ECHO pin to Arduino pin 11.
- Attach the sensor's VCC (power) pin to Arduino's 5V output.
- Connect the sensor's GND (ground) pin to Arduino's GND.

Pin Connections of **LED**:

- Plug the shorter leg of the LED into a hole on the breadboard which connects to GND.
- Plug the longer leg of the LED into a different hole on an independent line of the breadboard.
- Add a resistor between this longer leg and pin 5 on the breadboard.

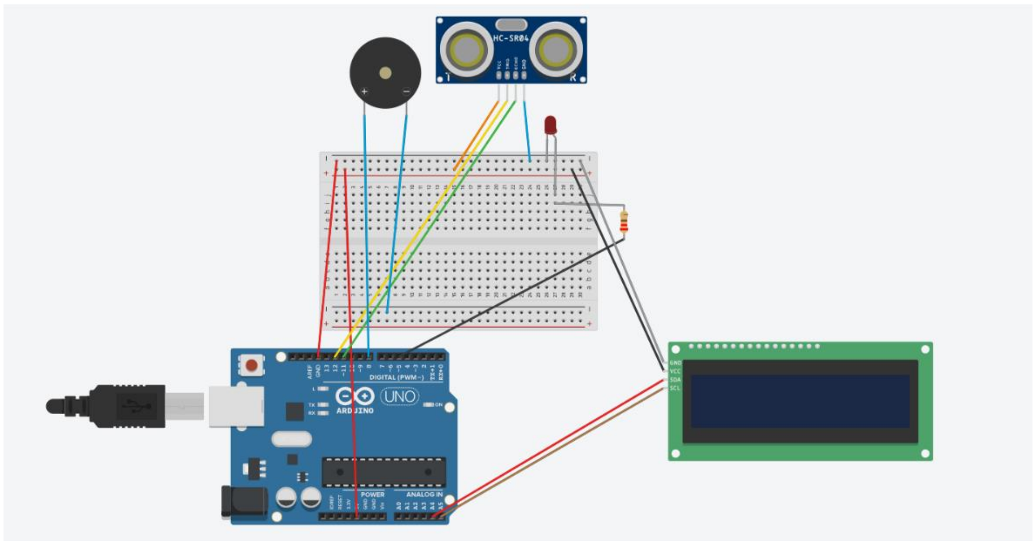
Pin Connections of **Buzzer**:

- Connect the positive (supply) wire of the buzzer to a digital pin on the Arduino.
- Connect the negative (ground) wire of the buzzer to the GND pin on the Arduino.

Pin Connections of **LCD**:

- Connect the SCL to the A5 pin, and the SCA to the A4 pin
- VCC to 5V and GND to GND

a. Connection diagram then code



```
#include "SR04.h"
#include "pitches.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
#define TRIG_PIN 12
#define ECHO_PIN 11
SR04 sr04 = SR04(ECHO_PIN,TRIG_PIN);
long d;

void setup() {
   pinMode(5,OUTPUT);
   lcd.init();
   lcd.backlight();
}

void loop() {
   d=sr04.Distance();
   Serial.print(d);
   Serial.println("cm");
   if(d<30){
    digitalWrite(5,HIGH);
    tone(8, NOTE_D5, 100);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("BISO DETECTED!"); //biso is my cat
   }
   else{
    digitalWrite(5,LOW);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("You're safe");
   }
}
```

4. Conclusion

From our experience with an **Arduino kit**, we've gained valuable insights into electronics, and programming.

Basics of Electronics: The Arduino kit introduced me to fundamental concepts like current, voltage, and resistance. I got hands-on experience with components like breadboards, wires, and sensors.

Programming Skills: Through guided lessons, I learned how to **code** using the Arduino platform. Whenever I faced issues, I turned to online resources and the Arduino community for help.

5. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] tinkercad https://www.tinkercad.com/

ASSIGNMET #20: TEMPERATURE SENSOR WITH EMERGENCY BUZZER

1. Introduction

In general, for Arduino:

Arduino is an open-source electronics platform that simplifies the process of creating interactive electronic projects. Developed in the early 2000s by Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. Arduino consists of a microcontroller board with input and output pins, a development environment (Arduino IDE), and a vast community of users and contributors. The Arduino board acts as the brain of the project, executing code written in the Arduino programming language, which is based on Wiring and C/C++. Overall, Arduino has revolutionized the world of electronics prototyping, democratizing access to technology and empowering individuals to turn their ideas into reality.

In specific for this assignment:

The utilization of temperature sensors with Arduino microcontrollers has become increasingly prevalent in various applications ranging from home automation to industrial monitoring. In this report, we delve into the integration of a temperature sensor with a buzzer using Arduino, exploring its functionality, implementation, and potential applications.

The primary objective of this assignment is to design and implement a system that can detect changes in temperature and provide auditory alerts through a buzzer based on predefined thresholds. By accomplishing this task, we aim to showcase the versatility of Arduino in creating practical solutions for real-world problems.

2. Required components

The required components for a system with temperature sensor with a buzzer alert are as follows:

a. Arduino Board:

Acts as the brain of the project, running the code and controlling the behavior of the temperature sensor and buzzer. It reads analog data from the temperature sensor and activates the buzzer when the temperature exceeds a certain threshold.

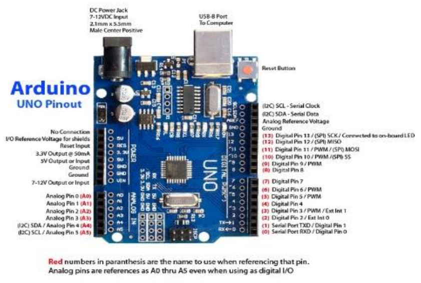

DC Power Jack
7-12VDC Input
2.1mm x 5.5mm
Male Center Positive
USB-B Port
To Computer
Reset Button
Arduino
UNO Pinout
No Connection
I/O Reference Voltage for shields
Reset Input
3.3V Output @ 50mA
5V Output or Input
Ground
Ground
7-12V Output or Input
Analog Pin 0 (A0)
Analog Pin 1 (A1)
Analog Pin 2 (A2)
Analog Pin 3 (A3)
(I2C) SDA / Analog Pin 4 (A4)
(I2C) SCL / Analog Pin 5 (A5)
(I2C) SCL - Serial Clock
(I2C) SDA - Serial Data
Analog Reference Voltage
Ground
(13) Digital Pin 13 / (SPI) SCK / Connected to on-board LED
(12) Digital Pin 12 / (SPI) MISO
(11) Digital Pin 11 / PWM / (SPI) MOSI
(10) Digital Pin 10 / PWM /(SPI) SS
(9) Digital Pin 9 / PWM
(8) Digital Pin 8
(7) Digital Pin 7
(6) Digital Pin 6 / PWM
(5) Digital Pin 5 / PWM
(4) Digital Pin 4
(3) Digital Pin 3 / PWM / Ext Int 1
(2) Digital Pin 2 / Ext Int 0
(1) Serial Port TXD / Digital Pin 1
(0) Serial Port RXD / Digital Pin 0
Red numbers in paranthesis are the name to use when referencing that pin.
Analog pins are references as A0 thru A5 even when using as digital I/O

Figure 1: Arduino Board

b. Temperature Sensor (Thermistor):

Measures the ambient temperature and converts it into a resistance value. As the temperature changes, the resistance of the thermistor changes accordingly. The Arduino reads this resistance value and calculates the temperature using a Steinhart-Hart equation.

Figure 2: Thermistor Module

c. Buzzer:

An electromechanical device that generates sound when activated. In this assignment, the buzzer is used to emit an emergency sound when the temperature exceeds a predefined threshold.

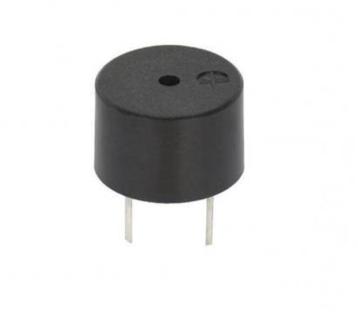


Figure 3: Buzzer for Arduino

d. Connecting Wires:

Produces sound when activated by the Arduino. In this assignment, the buzzer is used to emit an emergency sound when the temperature exceeds a certain threshold. The Arduino activates the buzzer by sending a signal to its control pin.

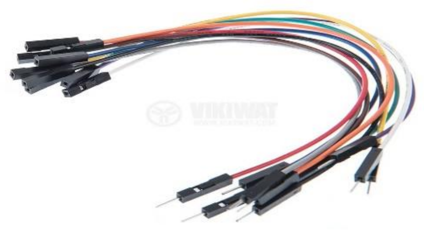


Figure 4: Connecting Wires

e. Power Source:

Provides power to the Arduino board and other components. In this assignment it is the USB connection to the computer. Ensuring an adequate and stable power supply is essential for the proper functioning of the entire system.

Figure 5: USB Cable

3. Implementation Steps

Gather Components: Collect all the necessary components mentioned earlier, including the Arduino board, temperature sensor (thermistor), buzzer, connecting wires, and power source.

Setup Arduino IDE: This software will be used to write, compile, and upload code to the Arduino board.

Connect Components: Wire up the components according to the circuit diagram. Connect the thermistor, buzzer, and any other necessary components to the appropriate pins on the Arduino board. Ensure that all connections are secure and correctly made.

Write Arduino Code: Open the Arduino IDE and create a new sketch. Write the code that reads data from the temperature sensor, calculates the temperature, and activates the buzzer when the temperature exceeds a certain threshold.

Upload Code to Arduino: Connect your Arduino board to your computer using a USB cable. Select the correct board and port in the Arduino IDE, then compile and upload your code to the Arduino board. Verify that the code uploads successfully and that there are no errors reported.

Test the System: Power up the Arduino board and observe the behavior of the system. Use a heat source (lighter) to increase the temperature around the thermistor and verify that the buzzer activates when the temperature exceeds the specified threshold. Adjust the threshold and other parameters in the code as necessary to achieve the desired functionality.

Optimize and Debug: Fine-tune your code and circuitry to improve performance and reliability. Test the system under various conditions and ensure that it behaves as expected. Debug any issues or errors that arise during testing and make necessary adjustments to the code or hardware.

a. Connection diagram:

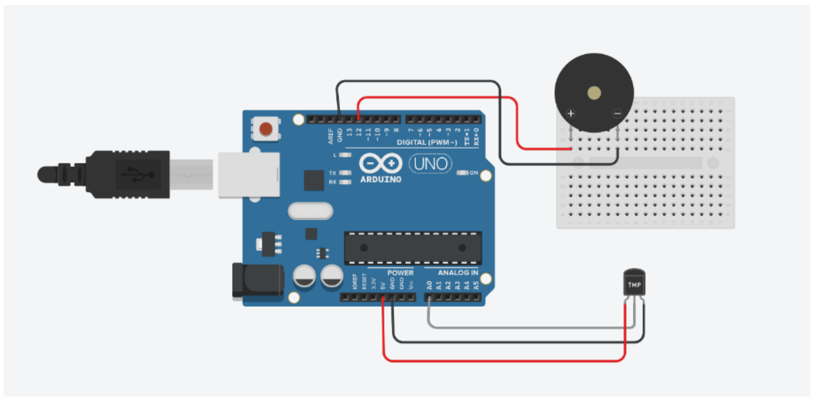


Figure 6: Diagram using TinkerCad

b. Code:

```
#define THERMISTORPIN A0

#define THERMISTORNOMINAL 10000

#define TEMPERATURENOMINAL 25

#define NUMSAMPLES 10

#define BCOEFFICIENT 3950

#define SERIESRESISTOR 10000

#define BUZZER_PIN 12

#define EMERGENCY_SOUND_DURATION 500

#define BUZZER_THRESHOLD 30.0

#define LED 13


unsigned long previousMillis = 0;

int emergencyState = LOW;
```

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

void emergencySound() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= EMERGENCY_SOUND_DURATION) {
    // Toggle the emergencyState
    emergencyState = !emergencyState;
    digitalWrite(BUZZER_PIN, emergencyState);

    // Update the previousMillis for the next tone
    previousMillis = currentMillis;
  }
}


void loop(){
float temperature = readTemperature();
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println("°C");
  // Activate emergency sound if temperature exceeds threshold
  if (temperature > BUZZER_THRESHOLD) {
    emergencySound(); // Activate emergency sound
  } else {
    digitalWrite(BUZZER_PIN, LOW); // Turn off buzzer
  }
}


float readTemperature() {
  uint16_t samples[NUMSAMPLES];

  // Saving values from input to pole
  for (int i = 0; i < NUMSAMPLES; i++) {
    samples[i] = analogRead(THERMISTORPIN);

  }

  // Calculate average value of input
  float average = 0;
  for (int i = 0; i < NUMSAMPLES; i++) {
    average += samples[i];
  }
  average /= NUMSAMPLES;
  // Calculate resistance

  average = 1023 / average - 1;

  average = SERIESRESISTOR / average;


  // Convert resistance to temperature

  float temperature = average / THERMISTORNOMINAL;

  temperature = log(temperature);

  temperature /= BCOEFFICIENT;

  temperature += 1.0 / (TEMPERATURENOMINAL + 273.15);

  temperature = 1.0 / temperature;

  temperature -= 273.15; // Convert to Celsius


 Serial.println(temperature);

  return temperature;

}
```

4. Results

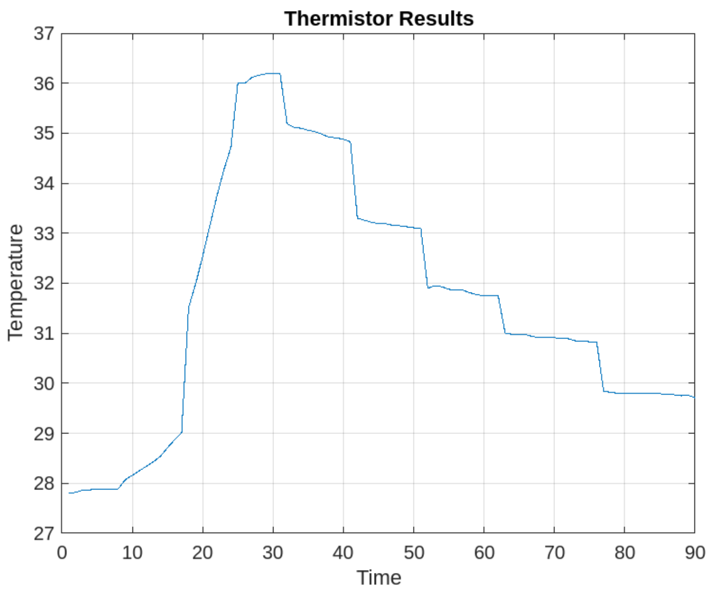
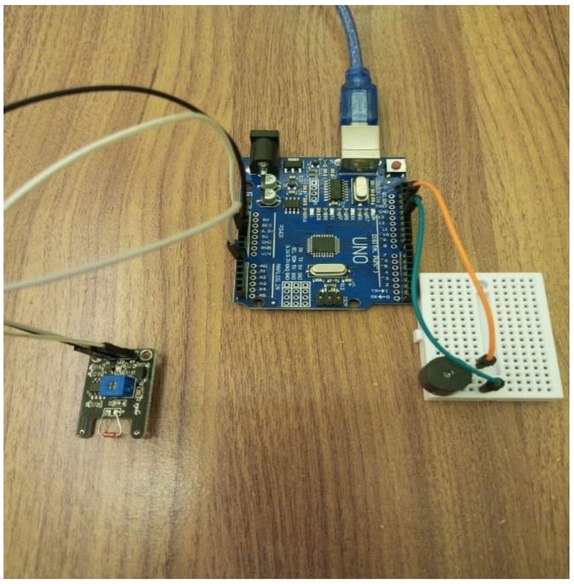


Figure 7: Graph showing the results using MATLAB

MATLAB Code showing the results of the thermistor sensor taking data from serial plotter and implementing them in MATLAB:

```
>> y_values = [27.80, 27.82, 27.87, 27.87, 27.88, 27.89, 27.89, 27.89, ...
        28.07, 28.16, 28.25, 28.34, 28.43, 28.54, 28.71, 28.87, ...
        29.01, 31.51, 31.99, 32.54, 33.14, 33.73, 34.26, 34.71, ...
        36.01, 36.0, 36.12, 36.16, 36.19, 36.20, 36.20, 35.19, ...
        35.12, 35.10, 35.06, 35.03, 34.98, 34.92, 34.91, 34.88, ...
        34.83, 33.30, 33.26, 33.22, 33.19, 33.19, 33.16, 33.15, ...
        33.13, 33.11, 33.10, 31.9, 31.95, 31.93, 31.88, 31.86, ...
        31.86, 31.81, 31.77, 31.75, 31.76, 31.75, 31.00, 30.98, ...
        30.98, 30.97, 30.93, 30.92, 30.91, 30.91, 30.90, 30.89, ...
        30.85, 30.85, 30.83, 30.83, 29.84, 29.82, 29.80, 29.80, ...
        29.80, 29.80, 29.80, 29.80, 29.79, 29.78, 29.77, 29.75, ...
        29.76, 29.71];
>> x_values = 1:length(y_values);
>> plot(x_values, y_values);
>> xlabel('Time');
>> ylabel('Temperature');
>> title('Thermistor Result');
>> grid on;
```

Figure 8: Real Wiring Diagram



5. Conclusion

In conclusion, the implementation of the temperature sensor with a buzzer in this assignment has provided valuable insights into sensor interfacing and real-time feedback mechanisms. The choice of this specific sensor was deliberate, as its integration aligns with the requirements of the final project I am working on. By incorporating this sensor, I aim to enhance the functionality and effectiveness of the final project, ensuring accurate temperature monitoring and timely alerts. Through this assignment, I have gained practical experience in Arduino programming, sensor calibration, and signal processing, which will undoubtedly contribute to the successful execution of the final project.

6. References

[1] Ultimate Starter Kit for Arduino UNO, Tutorial, https://lafvintech.com/

[2] YouTube Sessions