Southeast University

Department of Computer Science & Engineering (CSE)

Assignment

Introduction to Embedded System Lab

Course Code: CSE 382.1

SUBMITTED BY

Name: Morsalin Miah

Id:2020000000043

SUBMITTED TO

Ms. Arpita Das

Initial: AD

Lecturer, Department of Computer Science & Engineering

Southeast University

Experiment 1

Title: Using Ultrasonic Sensor to measure the distance between the sensor and any object.

Abstract
The purpose of this project is to learn about the working and application of Arduino and Ultrasonic sensors. These instrumentational tools play a vital role in the construction of integrated and complex real-life systems. We used an ultrasonic sensor along with a microprocessor(Arduino) to measure the distance of the object in the range of the particular sensor. We will code our microprocessor to work for the particular application. As a result, we will be able to measure critical distance which as an application will be used to calculate distance.

Objective

- To understand the working principle of ultrasonic sensor
- To implement the theoretical knowledge of sensors
- To get to know about Arduino Uno use IDE to write the codes for Arduino Uno.
- To Explore Sensor Range and Response Time

Theory
Ultrasonic sensors are widely employed in robotics projects for accurate distance measurement and obstacle detection. These sensors are readily available in the market and come in various configurations, with 3-pin, 4-pin, and 5-pin modules. Generally, the most common ones are the 4-pin or 5-pin modules.
The ultrasonic module comprises two transducers one for transmitting ultrasonic waves and the other for receiving them. Both transducers are mounted on a single PCB alongside control circuitry, facilitating seamless integration into robotic projects.

Ultrasonic distance sensors offer precise, non-contact distance measurements ranging from approximately 2 cm to 400 cm. These sensors operate using ultrasound, which is a high-frequency sound with a frequency of 40 kHz.

The working principle of ultrasonic distance measurement mirrors that of RADAR. To initiate ranging, Arduino generates a brief 10 μs pulse to the trigger input. The Ultrasonic Module responds by emitting an 8-cycle burst of ultrasound at 40 kHz and elevating its

echo line. The module then listens for an echo, lowering the echo line as soon as it detects one. The width of the echo pulse is proportional to the distance to the object.

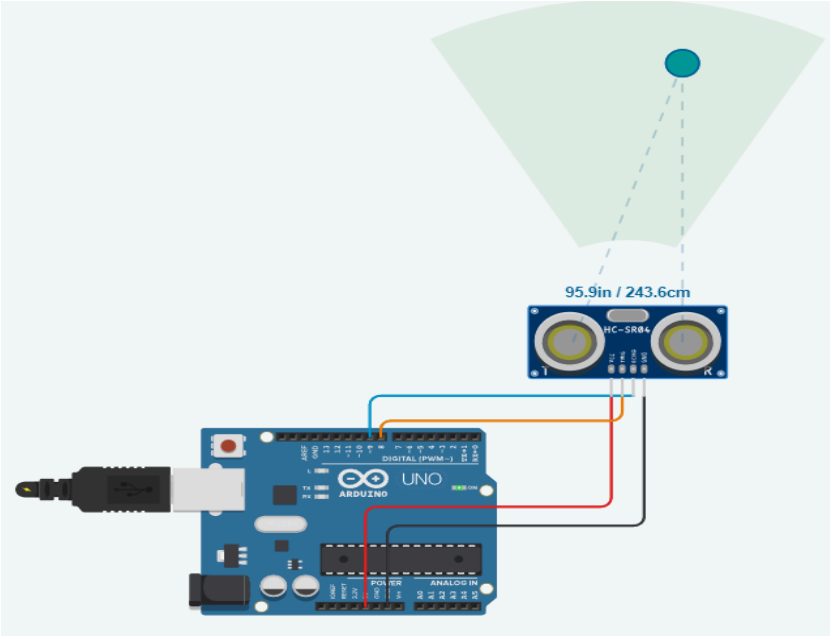
If an object is detected, the echo pulse width can be measured in microseconds. Dividing this value by 58 provides the distance in centimeters while dividing by 148 yields the distance in inches. If no object is detected, the module automatically lowers its echo line after approximately 30 ms.

The Ultrasonic Module can be triggered as frequently as every 50 ms, allowing for up to 20 measurements per second. It is crucial to wait for 50 ms before the next trigger, even if a close object is detected and the echo pulse is shorter. This delay ensures that any residual ultrasonic signals dissipate, preventing false echoes in subsequent measurements. The sensor is capable of detecting objects within a range of 3 cm to 3 m, making it versatile for various applications.

Apparatus

- Jumper Wires
- Arduino UNO
- Ultrasonic sensor

Circuit Diagram



Code:

```
int trig =8, echo=9;
float time=0, distance=0;
void setup(){

 pinMode(trig, OUTPUT);
 pinMode(echo, INPUT);
 Serial.begin(9600);
}

void loop(){
  digitalWrite(trig, LOW);
          delay(3);
          digitalWrite(trig,HIGH);
          delayMicroseconds(10);
          digitalWrite(trig, LOW);
          time= pulseIn(echo,HIGH);
          distance= (time*0.034)/2;
          Serial.println(distance);
          delay(2000);
 }
```

Conclusion

From this particular experiment, we learned that ultrasonic sensors can detect and measure any object in its range irrespective of the material applied. It can even detect transparent material but the limitation is against any black body object that absorbs sound signals.

Experiment 2

Experiment Name: Measuring temperature and humidity using a DHT11 sensor.

Abstract:
DHT11 sensors are widely used for their low cost and ease of use in monitoring temperature and humidity. They combine a thermistor for temperature measurement and a humidity-sensitive polymer capacitor for humidity sensing. This experiment aimed to evaluate the sensor's capabilities and limitations in various environmental settings.

Objective :

- Calibrate the DHT11 sensor by exposing it to a controlled environment with known temperature and humidity levels.
- Verify the accuracy and reliability of the sensor readings by comparing them with reference values obtained from calibrated instruments.
- Ensure that the DHT11 sensor provides consistent and repeatable measurements under controlled conditions.

Theory

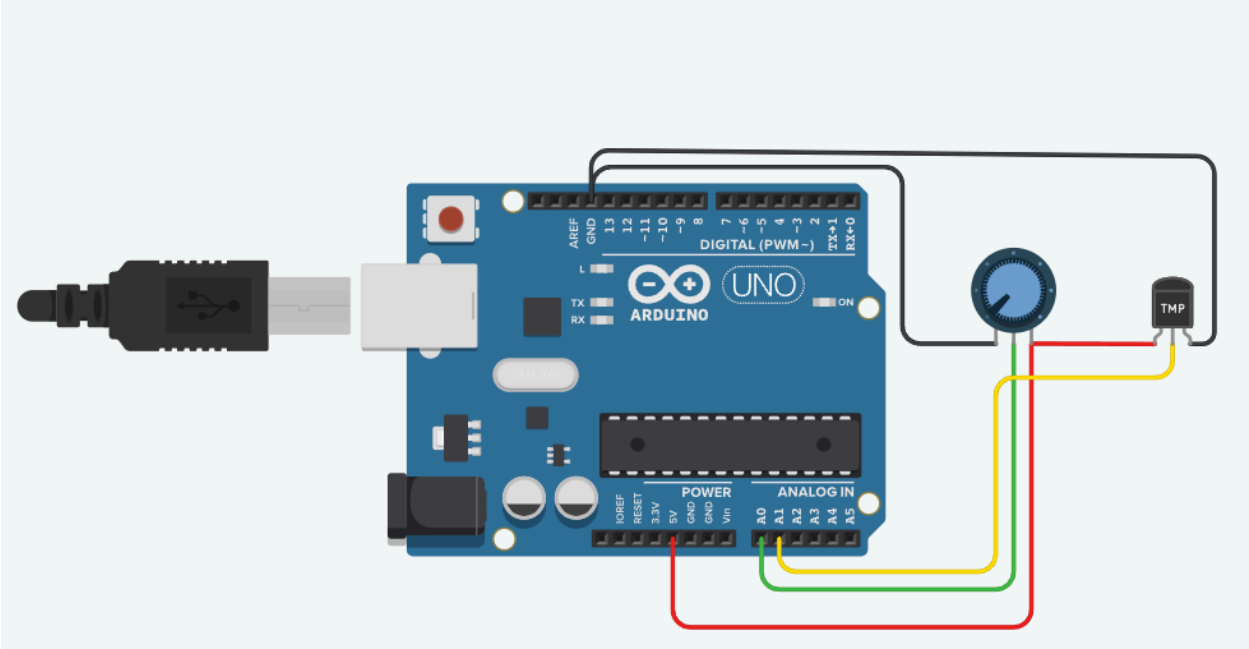
The DHT11 sensor operates based on the principles of capacitive humidity sensing and calibrated digital temperature sensing. It is designed to provide accurate and reliable measurements of both temperature and humidity in a compact, cost-effective package.
Humidity Sensing: The DHT11 sensor incorporates a humidity sensing element based on the capacitive humidity sensing principle. This element consists of a moisture-absorbing substrate sandwiched between two electrodes. The capacitance of this sensing element changes with variations in ambient humidity levels.
When exposed to air with different humidity levels, the moisture-absorbing substrate absorbs or releases water molecules, leading to changes in the capacitance of the sensing element. The DHT11 measures the time it takes for a capacitor to discharge through a resistor, and this time is directly proportional to the relative humidity of the surrounding environment. The sensor converts this analog humidity signal into a digital format for ease of processing.

Temperature Sensing: The DHT11 utilizes a calibrated digital temperature sensor to measure ambient temperature accurately. This sensor provides a digital output corresponding to the temperature value.
The digital temperature sensor converts the analog temperature signal into a digital format, making it suitable for integration with digital systems such as microcontrollers.
The DHT11 sensor employs a mathematical model to convert the digital signal into temperature readings, ensuring accurate and calibrated temperature measurements.

Apparatus:

- Arduino Uno R3
- DHT11 sensor
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE software

Circuit Diagram



Setup:

Connect the DHT11 sensor to the Arduino Uno as follows:
VCC pin to 5V pin
GND pin to GND pin
Data pin-to-pin 2

Code

```
DHT dht(DHTPIN, DHTTYPE);

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

void
loop()
{
  // Wait 2 seconds between measurements
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!


float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to avoid NaN)
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print humidity and temperature values
  Serial.print("Humidity: ");
```

```
  Serial.print(h);
  Serial.print("%\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println("°C");
}
```

Results:

1. Analyze the recorded data, comparing the DHT11 sensor readings with reference values (if available) or across different environments.
2. Calculate the average temperature and humidity readings and any observed variations.
3. Assess the sensor's accuracy based on the comparisons, noting any significant deviations.
4. Evaluate the sensor's response time by observing how quickly its readings change with environmental shifts

Experiment 3:

Experiment Name: Rotate 180 degrees with a Servo Motor

Abstract:

This experiment investigated the capability of a servo motor to perform a continuous 180-degree rotation. The objective was to measure its accuracy, speed, and limitations under this specific action.

Theory

Servo motors are widely used in robotics and automation applications due to their precise control and angular positioning capabilities. They receive pulse width modulation (PWM) signals from a microcontroller, determining the angle at which the shaft holds. This experiment aimed to explore the servo's behavior and efficiency when commanded to rotate through its full range (180 degrees) and back to its starting position.

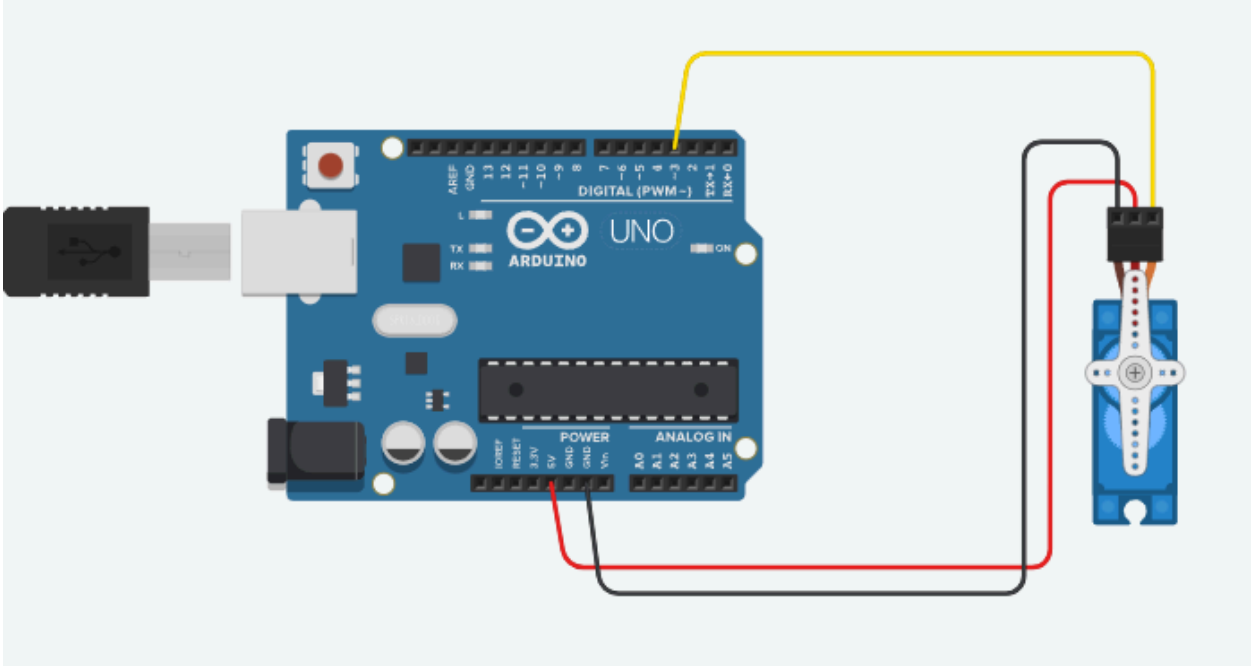
Apparatus

- Arduino Uno R3
- Standard servo motor
- Breadboard
- Jumper wires
- USB cable
- Stopwatch or timer
- Computer with Arduino IDE software

Setup:

- Servo ground (black wire) to GND pin
- Servo power (red wire) to 5V pin
- Servo signal (yellow wire) to pin 8 Assemble the circuit on the breadboard, ensuring secure connections.

Circuit diagram



Code

```
#include <Servo.h>

Servo myservo; // Create servo object

const int angleGoal = 180; // Target rotation angle (180 degrees)

void setup() {
  myservo.attach(8); // Attach servo to pin 8
}

void loop() {
  // Rotate to angleGoal slowly
  for (int angle = 0; angle <= angleGoal; angle++) {
    myservo.write(angle);
    delay(10); // Adjust delay for desired speed
  }
```

```
  // Pause for observation
  delay(2000);

  // Return to starting position slowly
  for (int angle = angleGoal; angle >= 0; angle--) {
    myservo.write(angle);
    delay(10);
  }

  // Repeat the cycle
  delay(2000);
}
```

Procedure:

1. Run the uploaded code on the Arduino.
2. Observe the servo motor's movement throughout the 180-degree rotation and back to its starting position.
3. Use the stopwatch or timer to measure the total time taken for one complete cycle (180° and back).
4. Repeat the measurement several times and record the data.

Results:

1. Analyze the recorded data, calculating the average rotation time and noting any significant variations.
2. Based on observation, assess the accuracy of the servo's movement, if it reached the full 180-degree range and returned precisely to its starting position.
3. Identify any limitations encountered during the experiment, such as delays, wobbly movement, or audible noise.

Discussion:

1. Compare the obtained results with theoretical expectations and discuss any discrepancies.
2. Analyze the potential sources of error, such as the delay time in the code, variations in servo performance, or external factors like friction.
3. Discuss the implications of the findings for various real-world applications of servo motors where precise and efficient continuous rotations are required.

Conclusion:

1. Summarize the key findings from the experiment, highlighting the achieved rotation accuracy, speed, and any identified limitations.
2. Briefly discuss the suitability of servo motors for applications requiring 180-degree continuous rotations based on the obtained results.
3. Suggest potential improvements for future experimentation, such as testing different servo models, adjusting software parameters, or controlling external factors.

Experiment: 4

Experiment Name: Blinking a 5V LED with Arduino and Relay

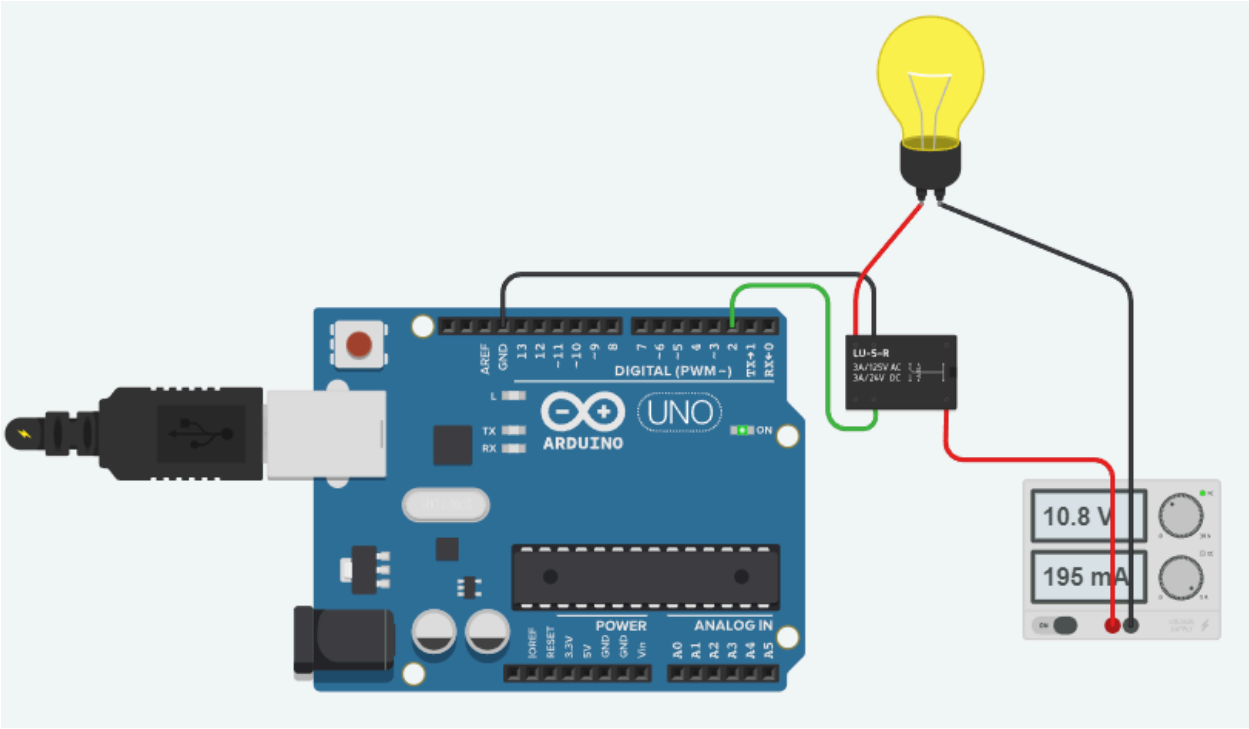
Objective

This lab report investigates the use of an Arduino UNO and a relay to control a 5V LED in a blinking circuit. We explore the basic principles of relay operation and Arduino programming to design and implement a circuit that achieves the desired blinking effect. Through experimentation and analysis, we gain a deeper understanding of how relays and Arduino can be used for timing, control, and interactive applications.

Theory

Relays act as electromechanical switches controlled by a low-power signal, capable of switching high-power circuits. We can create dynamic effects like blinking lights by combining them with timing circuits or microcontrollers like Arduino. This experiment explores using an Arduino UNO and a relay to control a 5V LED, offering precise control and customization compared to traditional timer IC circuits.

Circuit



Apparatus:

- Arduino UNO
- Breadboard
- Power Supply
- Relay (SPDT)
- 5V LED
- Resistor (220Ω)
- Connecting wires
- Computer with Arduino IDE installed

Circuit Design:

Connect the positive terminal of the relay coil to digital pin 2 of the Arduino through a transistor driver (e.g., 2N3904).
Connect the negative terminal of the coil to the Arduino's ground.
Connect the LED with the resistor in series across the relay's normally open (NO) and common (COM) contacts.
Power the Arduino and circuit with separate 5V supplies.

Code:

```
int relayPin = 2;
void setup() {
  pinMode(relayPin, OUTPUT);
}

void loop() {
  digitalWrite(relayPin, HIGH);
  delay(1000);
  digitalWrite(relayPin, LOW);
  delay(2000);
}
```

Procedure:

- Upload the code to the Arduino UNO.
- Observe the LED blinking behavior.
- Modify the code to change the blink frequency and duty cycle (ratio of on-time to total cycle time) by adjusting the delay values.
- Explore alternative blinking patterns by implementing if statements, loops, or functions within the loop() function.

Observations and Analysis:

- The experiment successfully demonstrated the use of an Arduino and relay to create a controlled blinking LED effect.
- Modifying the delay values in the code allowed for precise control of the blink frequency and on/off durations.
- The Arduino's flexibility enables the implementation of various blink patterns and dynamic responses to inputs like sensors or buttons.

Result and Discussion:

This experiment showcases the advantages of using an Arduino compared to timer ICs for controlling blinking LEDs.
The ease of programming and code modification empowers users to design custom blink patterns and integrate interactive elements.
When choosing components, consider the current requirements of the relay and LED alongside the Arduino's available digital pins and current capacities.
Conclusion:

Exploring the use of an Arduino and relay to control a blinking LED provided valuable insights into timing, control, and interactive possibilities. This foundation can be built upon to create more complex projects involving diverse sensors, actuators, and dynamic lighting effects.