BRAC UNIVERSITY

Inspiring Excellence

CSE461

Introduction to Robotics Lab

Lab No. : 01
Group : 06
Section : 03
Semester : Fall_2025

Group members :

Muaz Mohammad Zimam	23101529
Abdullah Al Mahmud	22201502
Safin Ahamed Ifty	22299283
Razia Marzan Mou	22299123
Quazi Sumaya Sultana Prionti	22101454

Submitted Date: 28-10-2025
Submitted to - Mahadi Ibne Bakar & Aabrar Islam

1. **Objectives:**

   The aim of this lab is to give us an understanding of the Arduino Uno's digital I/O and serial communication capabilities by building simple circuits. More precisely, we will: Blink an LED at a fixed interval to explore how digital output pins are configured and driven. Read a push‑button input (using the internal pull‑up resistor) and print "Switch pressed" to the Serial Monitor to see how to handle inputs and use serial communication for feedback. Combine the above so that the LED lights only while the button is held down, illustrating how to link input sensing with output actuation in one sketch. This exercise offers hands-on practice with the Arduino's pin layout, the Arduino IDE's programming model, and the integration of hardware control with real-time software feedback.

2. **Equipments:**

   A. Arduino Uno B. 1 pc LED C. 1 pc 220 Ω Resistor D. 1 pc Push-button Switch E. Connecting Wires (Male–Male) F. Breadboard

3. **Experimental Setup:**

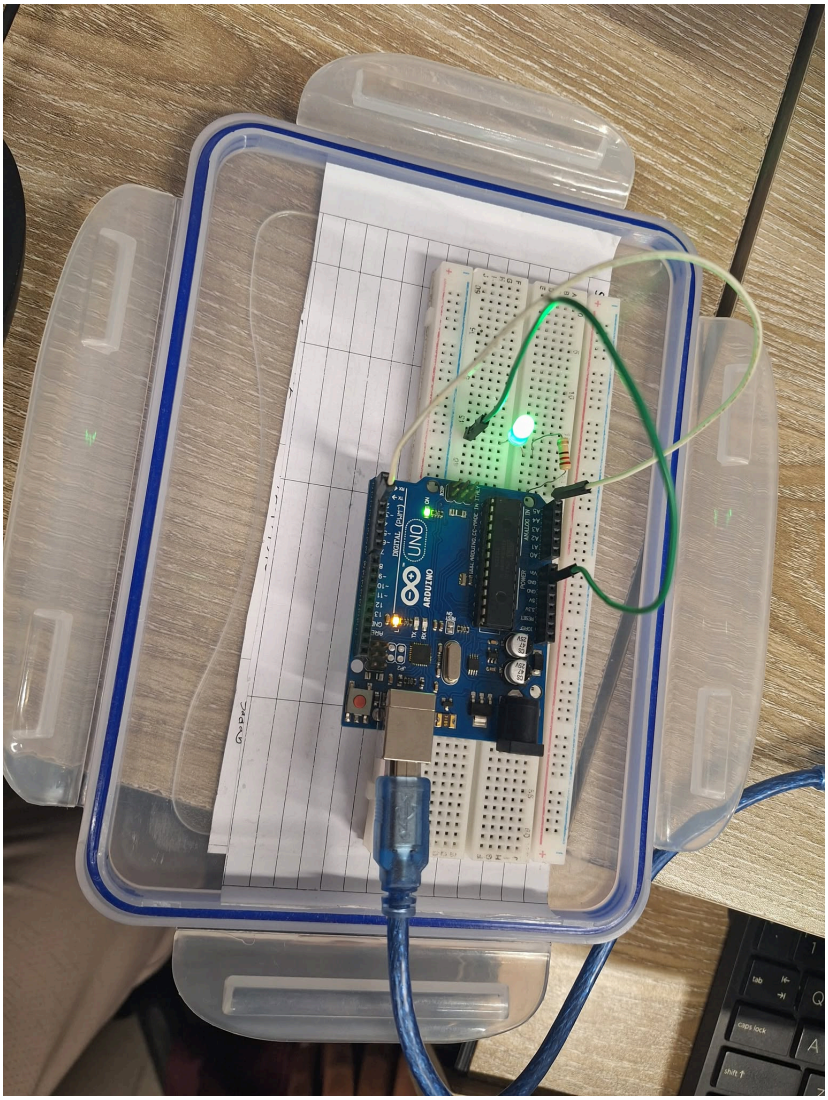
   (**Picture + Explanation**)

   In this lab, we interfaced an Arduino Uno with an LED and a push-button switch to explore digital I/O and serial communication. Here's what we did:

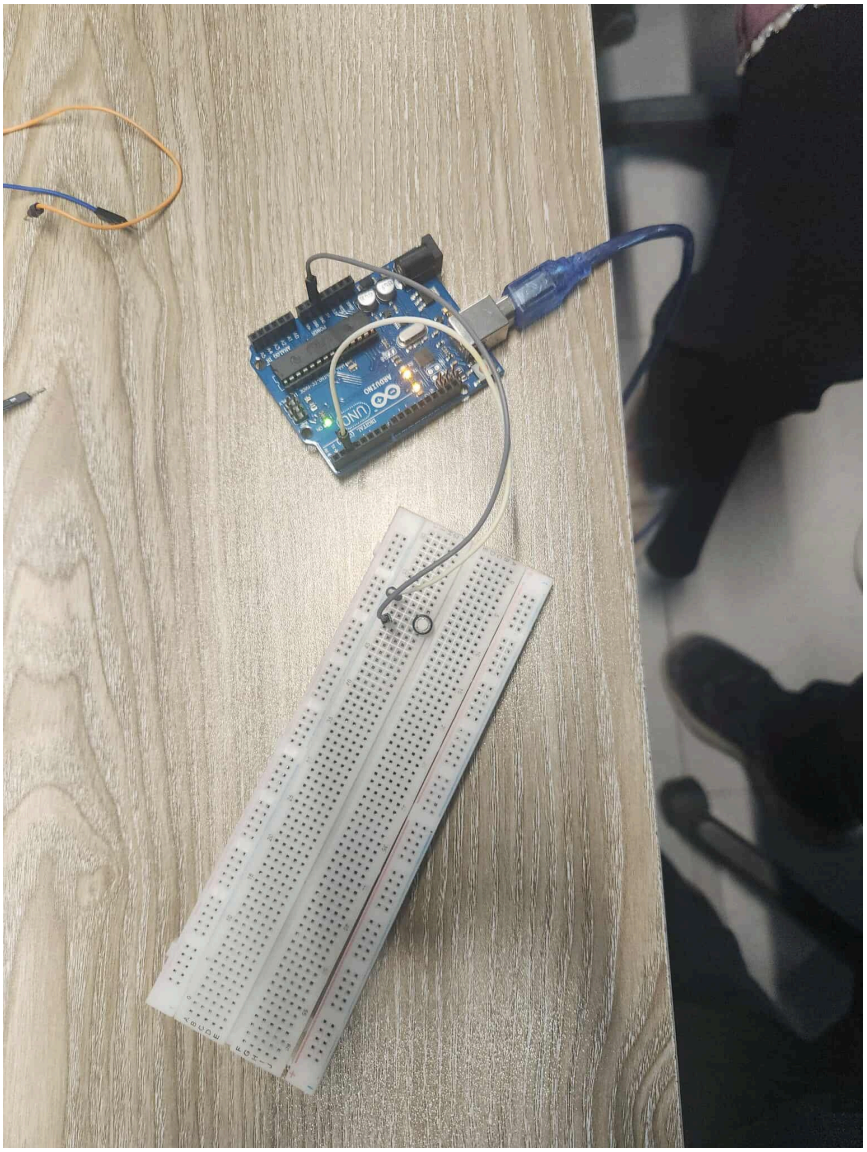
   Circuit setup: For the LED blink, we wired the LED (with a 220 Ω resistor) between digital pin 2 and GND.

   For the switch experiments, we connected one side of the push-button to digital pin 2 (or pin 3 for the combined task) and the other side to GND, relying on the Arduino's internal pull-up resistor.

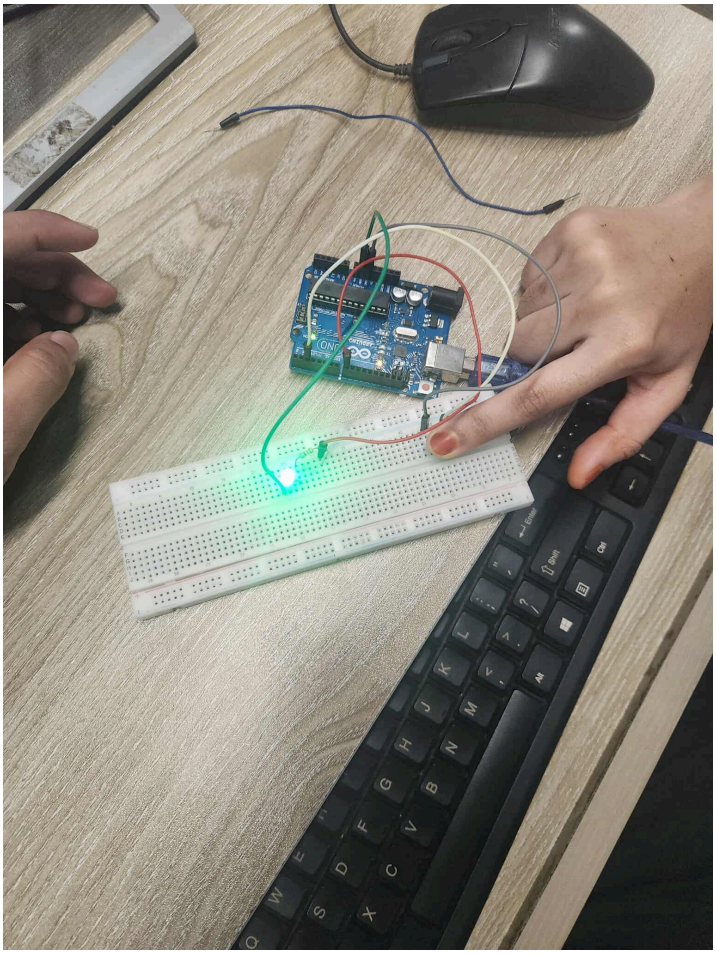
   Experiment 1 (LED blinking): We declared LED_PIN as an OUTPUT, then in loop() toggled it HIGH and LOW with delay(1000), printing "LED ON"/"LED OFF" over Serial at 9600 baud to confirm each state.



Experiment 2 (Switch-activated serial messaging): We set BUTTON_PIN as INPUT_PULLUP and in the loop(), polled digitalRead(). When the button read LOW (pressed), we sent “Button pressed.”; otherwise “Button not pressed.”, with a short delay for debouncing.



Combined task (Switch-controlled LED): We merged input and output: on a LOW read at BUTTON_PIN, we lit the LED (on LED_PIN) and printed “Button pressed.”When released, we turned the LED off and printed “Button not pressed.” This demonstrates linking input sensing directly to output actuation with real-time serial feedback.

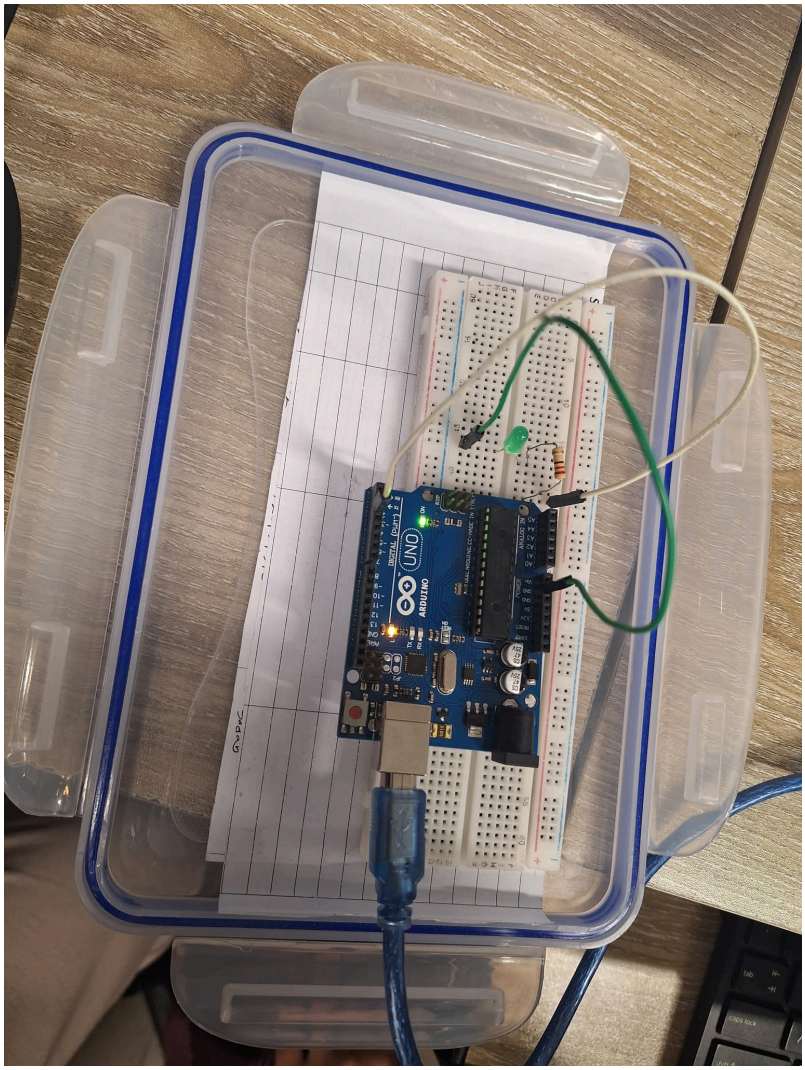


4. Code: (If Applicable)

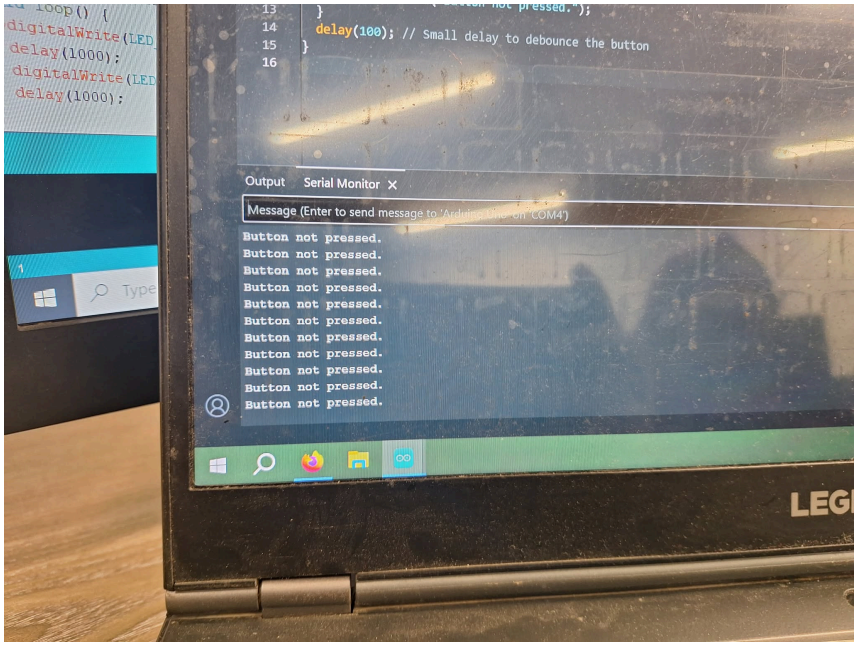
```
# Enter Code Here


// Experiment 1
#define LED_PIN 2 // Define the LED pin
void setup() {
pinMode(LED_PIN, OUTPUT); // Set the LED pin as an output
Serial.begin(9600); // Initialize serial communication
}
void loop() {
digitalWrite(LED_PIN, HIGH); // Turn the LED on
Serial.println("LED ON"); // Print LED status
delay(1000); // Wait for 1 second
digitalWrite(LED_PIN, LOW); // Turn the LED off
Serial.println("LED OFF"); // Print LED status
delay(1000); // Wait for 1 second
}


// Experiment 2
```

```
#define BUTTON_PIN 2 // Define the button pin
void setup() {
pinMode(BUTTON_PIN, INPUT_PULLUP); // Set the button pin as input
Serial.begin(9600); // Initialize serial communication
}
void loop() {
if (digitalRead(BUTTON_PIN) == LOW) { // Button is pressed
Serial.println("Button pressed."); // Print message to S. Monitor
}else{
Serial.println("Button not pressed.");
}
delay(100); // Small delay to debounce the button
}


// Lab Task
#define BUTTON_PIN 2 // Define the button pin
#define LED_PIN 3
void setup() {
pinMode(LED_PIN, OUTPUT);
pinMode(BUTTON_PIN, INPUT_PULLUP); // Set the button pin as input
Serial.begin(9600); // Initialize serial communication
}
void loop() {
if (digitalRead(BUTTON_PIN) == LOW) { // Button is pressed
Serial.println("Button pressed."); // Print message to S. Monitor
digitalWrite(LED_PIN, HIGH);
}else{
digitalWrite(LED_PIN, LOW);
Serial.println("Button not pressed.");
}
delay(100); // Small delay to debounce the button
}
```

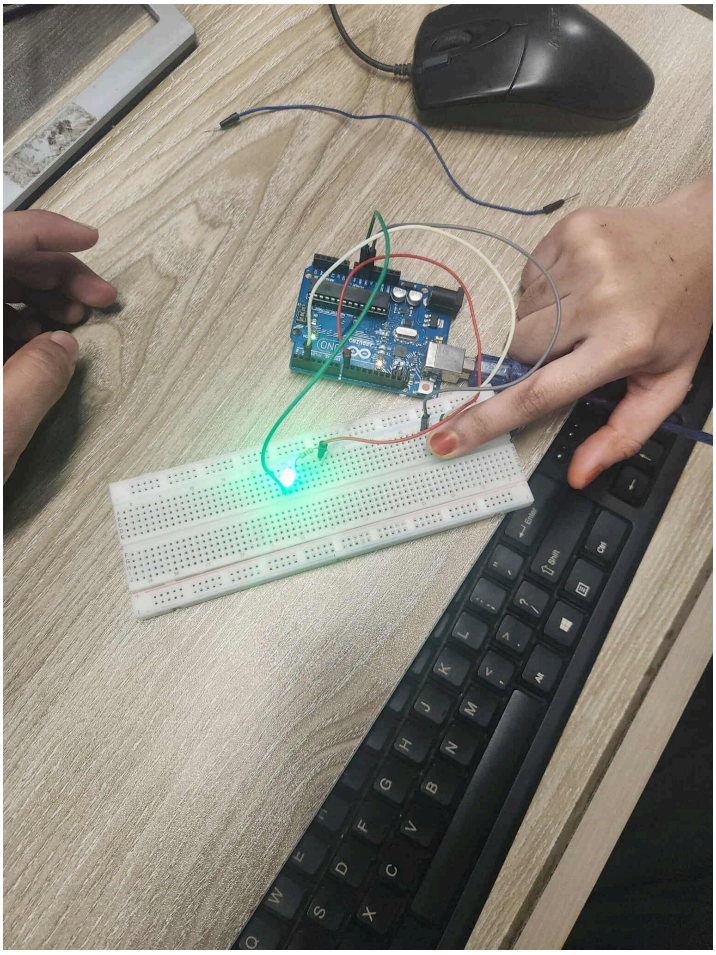
5. **Results (Output of the experiment):**
   The LED turned on for one second, then off for one second in a continuous cycle, and the Serial Monitor printed “LED ON” and “LED OFF” at each state change. This confirms that the digital output pin was correctly configured and timed using delay().



Experiment 2 – Switch-Activated Serial Messaging: Whenever the push-button was pressed, the Serial Monitor displayed “Button pressed.”; when the button was released, it showed “Button not pressed.” This verifies that the input pin (with internal pull-up) was read correctly and that serial communication was functioning as expected.



Lab Task – Switch-Controlled LED: Pressing the button simultaneously lit the LED and printed “Button pressed” to the Serial Monitor; releasing it turned the LED off and printed “Button not pressed.” This demonstrates successful integration of input sensing with output control and real-time feedback via serial messages.



6. Discussions/Answers:

(May contain conclusions or learnings from the result along with problems faced and solving methodology.)

Through these experiments, we confirmed our understanding of the Arduino's digital I/O and serial interface. By first blinking the LED, we validated that configuring a pin as OUTPUT and using delay() reliably toggles hardware. Introducing the push-button taught us about the internal pull-up resistor and inverted logic (LOW when pressed, HIGH when released) and reinforced how serial messages provide real-time debugging feedback. During testing, we encountered a few issues:

Button bounce: Rapid, unintended state changes. We mitigated this with a 100 ms delay() in the loop for basic debouncing. Wiring errors: Incorrect resistor placement or reversed LED polarity initially prevented the LED from lighting. Careful review of the circuit diagram and breadboard layout resolved these. Our stepwise methodology—verifying each component in isolation before integration—proved effective for both troubleshooting and demonstrating how software can directly control hardware actions.