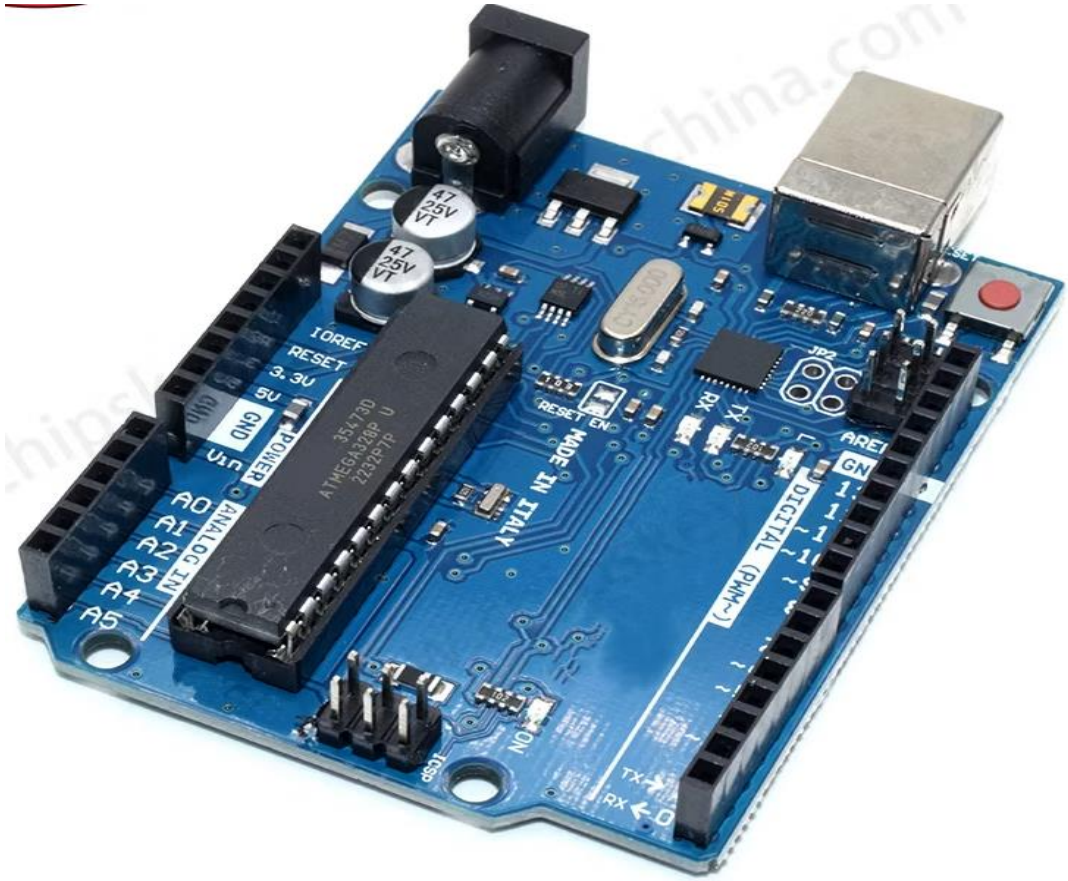
ASSIGNMENT 1



Name : Mohamed Ehab Sayed

Ass Microprocessor1

Dr: Usama

Task 1

Explain the main components and working principals of microprocessor and microcontroller and their differences.

Component/Aspect	Microprocessor	Microcontroller
Definition	A central processing unit (CPU) designed for general-purpose computing.	A compact integrated circuit that includes a CPU, memory, and peripherals for specific control tasks.
Core Components	ALU (Arithmetic and Logic Unit), Control Unit, Registers, Cache, System Bus	CPU, ROM, RAM, EEPROM, I/O Ports, Timers, ADC (Analog-to-Digital Converter)
Memory	Primarily relies on external memory (RAM, ROM).	On-chip memory (ROM, RAM, EEPROM) for program and data storage.
I/O Support	Typically requires external components (I/O chips).	Built-in I/O ports for direct interfacing with sensors, actuators, etc.
Clock Speed	Higher clock speed (up to GHz range), suited for general computing tasks.	Lower clock speeds (typically in kHz to MHz range), optimized for control tasks.
Complexity	More complex, typically used in computers, workstations, etc.	Less complex, designed for embedded systems and specific applications.
Peripherals	Requires external peripherals like timers, serial ports, etc.	Integrated peripherals like timers, ADCs, serial communication ports, etc.
Power Consumption	Higher power consumption due to external components.	Low power consumption, as most components are integrated into a single chip.
Cost	More expensive due to the need for external components.	Generally cheaper, as it integrates most components on a single chip.
Use Cases	Used in general-purpose computing (PCs, laptops, servers).	Used in embedded systems (e.g., robotics, home appliances, automotive controls).
Size	Larger in size as it needs external memory and peripherals.	Smaller and more compact, integrating everything on a single chip.
Example	Intel Core i7, AMD Ryzen, ARM Cortex-A series.	Arduino, PIC microcontrollers, Atmel AVR.

2. Illustrate the Arduino open source prototyping kit.

The **Arduino Open Source Prototyping Kit** is a versatile platform used for building interactive projects and prototypes, integrating both hardware and software. It consists of several key components:

1. **Arduino Boards**: Microcontroller-based boards (like Arduino Uno, Nano, etc.) that process data and control devices.
2. **IDE**: A software development environment used to write and upload code to the Arduino board.
3. **Microcontroller**: The central processing unit that runs the uploaded programs.
4. **Sensors**: Devices that gather data from the environment, such as temperature or motion sensors.
5. **Actuators**: Devices that respond to Arduino signals, like LEDs, motors, and displays.
6. **Breadboard and Wires**: Tools for connecting components without soldering.
7. **Power Supply**: Arduino can be powered via USB or an external source.
8. **Shields**: Add-on boards that extend the functionality of the Arduino.

Working Process:

- Connect the Arduino board to a computer and sensors/actuators to the board.
- Write a program (sketch) using the Arduino IDE, which reads sensor inputs and controls actuators.
- Upload the program to the Arduino board, which starts interacting with the environment based on the code.

Applications:

- **Home Automation**: Control appliances based on sensor data.
- **Robotics**: Build robots with sensors and actuators.
- **Wearables**: Create fitness trackers or health-monitoring devices.

This open-source kit allows for quick experimentation and prototyping, making it ideal for learning and innovation in electronics and embedded systems.

3. Write briefly on the Arduino Integrated Development Environment (IDE).

The **Arduino Integrated Development Environment (IDE)** is a software platform used to write, compile, and upload code to Arduino boards. It features:

- **Code Editor**: Allows you to write and edit code with syntax highlighting and error detection.
- **Libraries**: Simplifies programming by offering built-in and external libraries for sensors and devices.
- **Compiler**: Converts code into machine-readable instructions for the Arduino.
- **Uploader**: Uploads the compiled code to the Arduino board.
- **Serial Monitor**: Enables communication between the Arduino and the computer for debugging and monitoring data.
- **Cross-Platform**: Available for Windows, Mac, and Linux.

The Arduino IDE simplifies the process of programming and prototyping with Arduino boards, making it accessible for users of all skill levels.

1. **Arduino sketch** written in **Embedded C** for the **ATmega microcontroller** consists of two main functions:

2. `setup()`: This function runs once when the program starts. It is used to initialize pins, configure settings, and set up any necessary components (e.g., sensors, serial communication).

3. `loop()`: This function runs continuously after `setup()`. It contains the main logic of the program, executing tasks repeatedly, such as reading sensor values, controlling outputs, or handling timing.

Key elements in an Arduino sketch include:

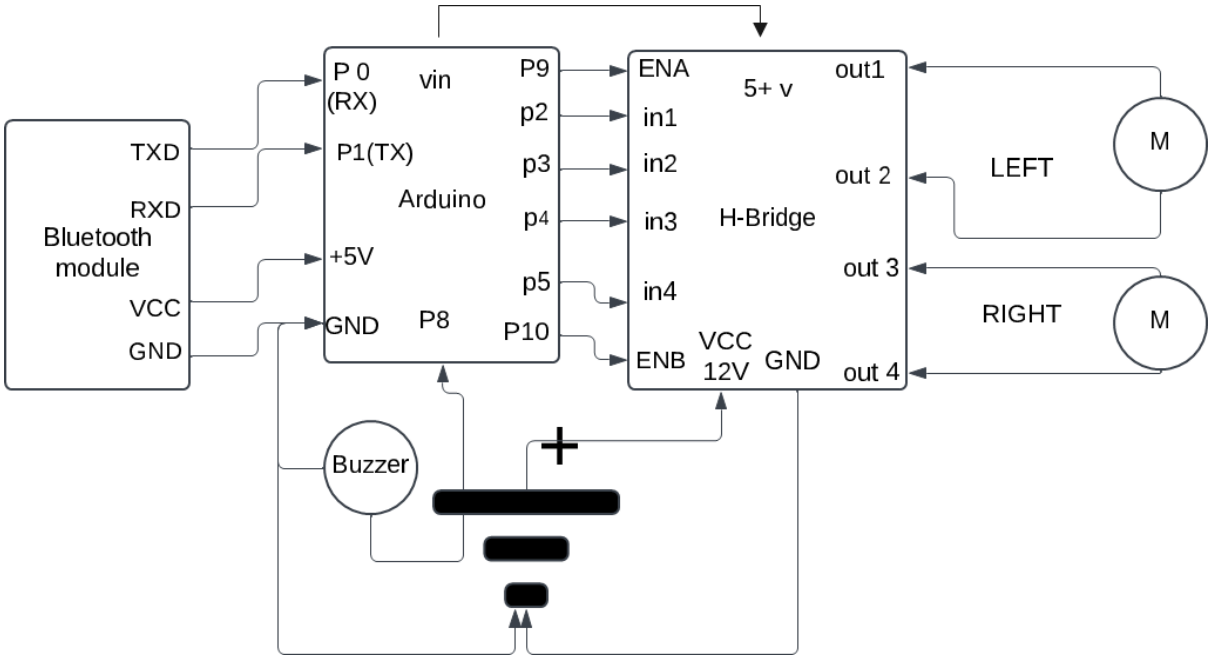
- **Pin initialization** with `pinMode()`.
- **Digital and analog I/O operations** with `digitalWrite()`, `digitalRead()`, `analogWrite()`, and `analogRead()`.
- **Timing** with `delay()`.
- **Serial communication** with `Serial.begin()` and `Serial.print()`.

Example: A basic sketch to blink an LED involves setting the LED pin in `setup()` and continuously turning it on and off in `loop()` using `digitalWrite()` and `delay()`.

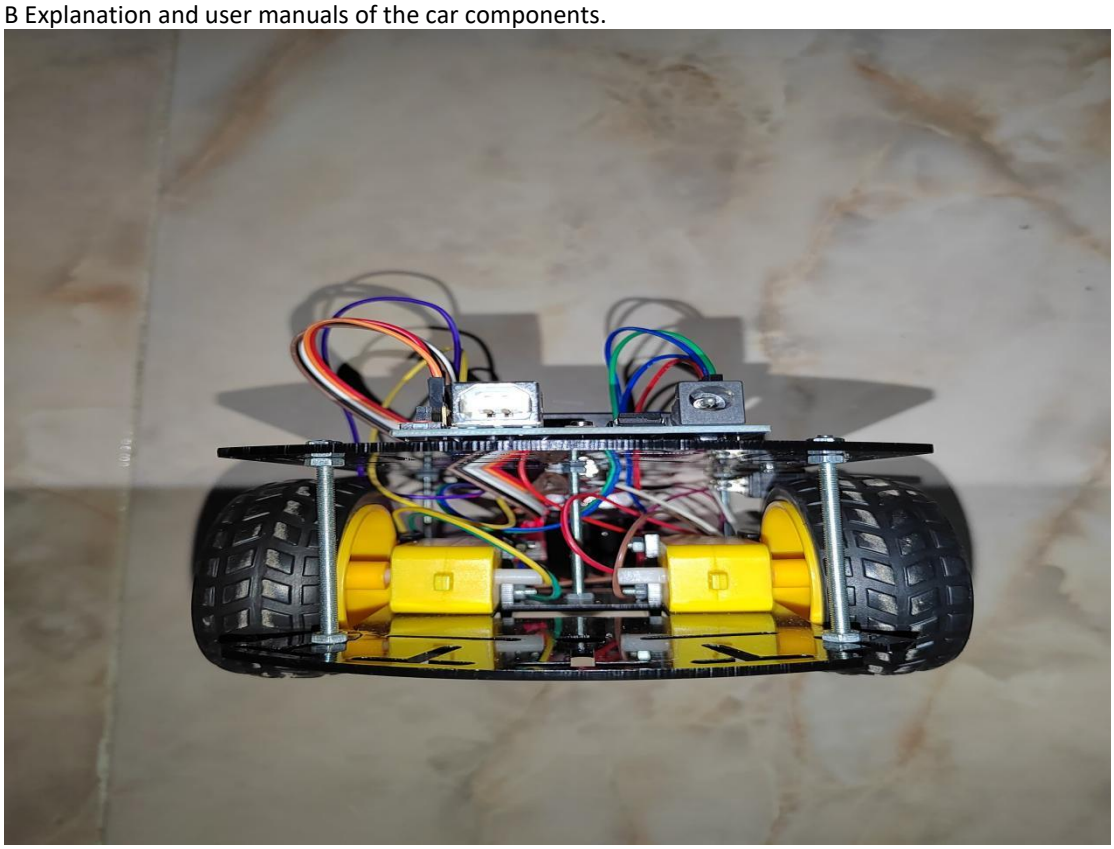
This simple structure makes Arduino programming easy and efficient for embedded system projects.

Task 2

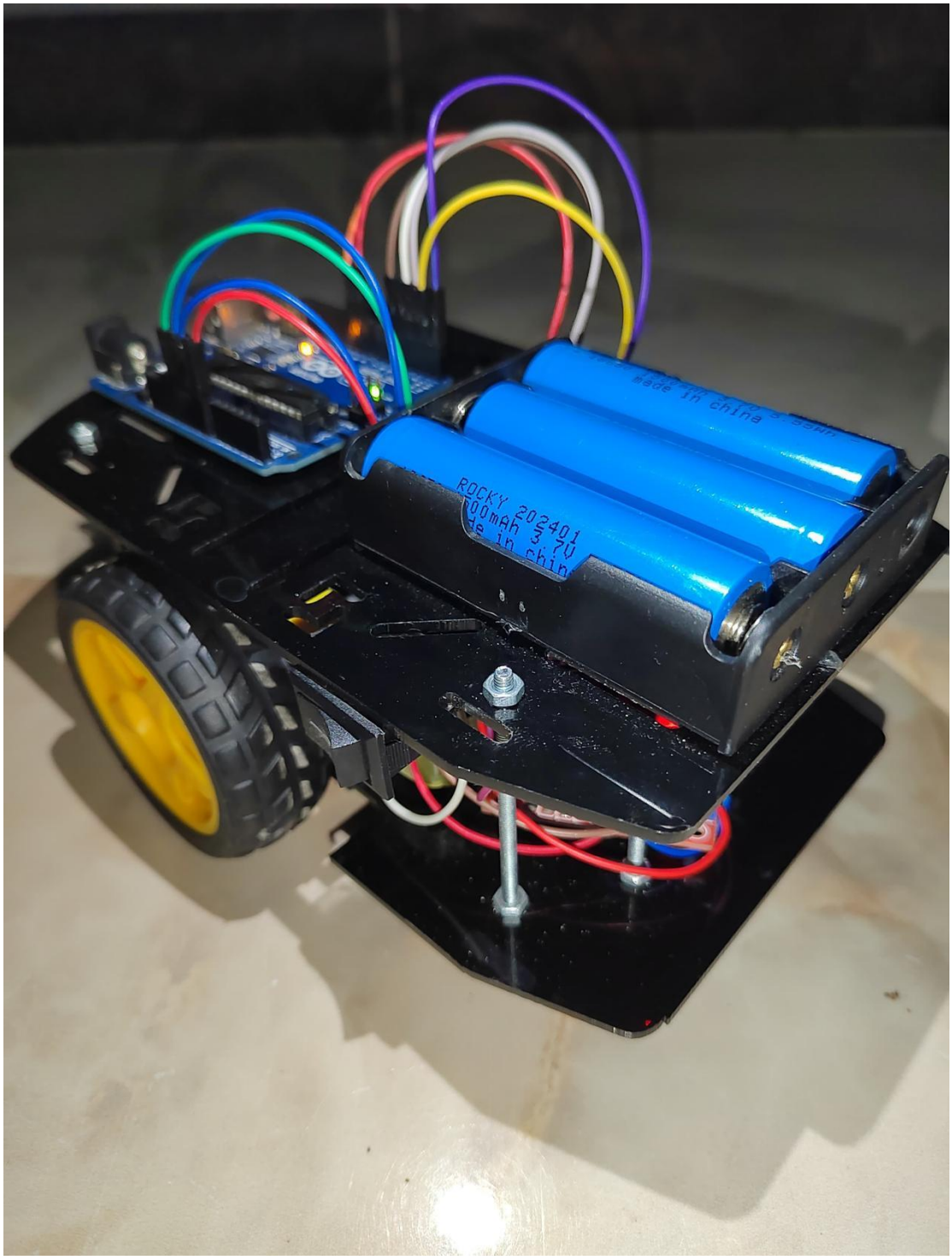
A. Schematic diagram of the robot car.



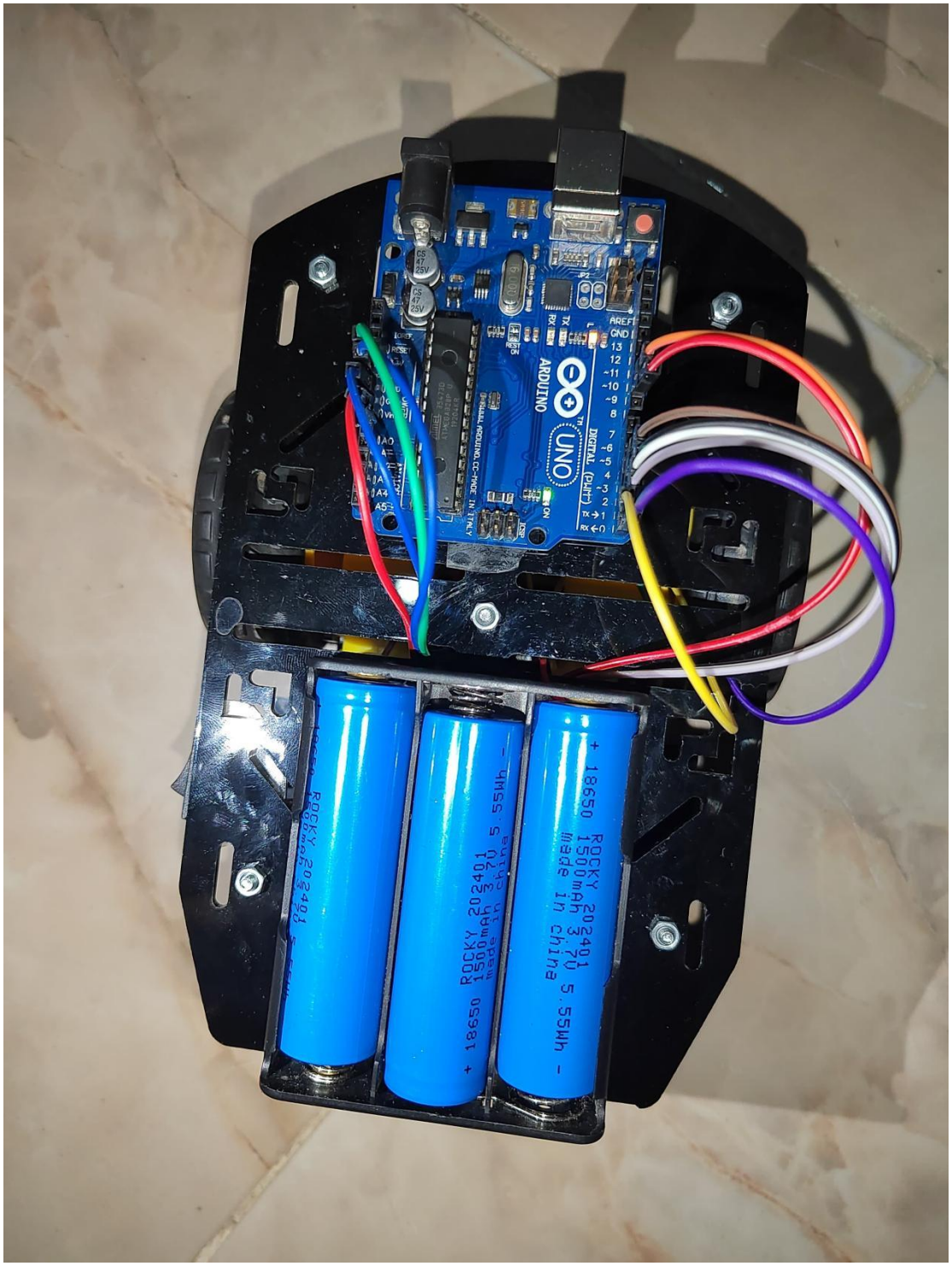
B Explanation and user manuals of the car components.



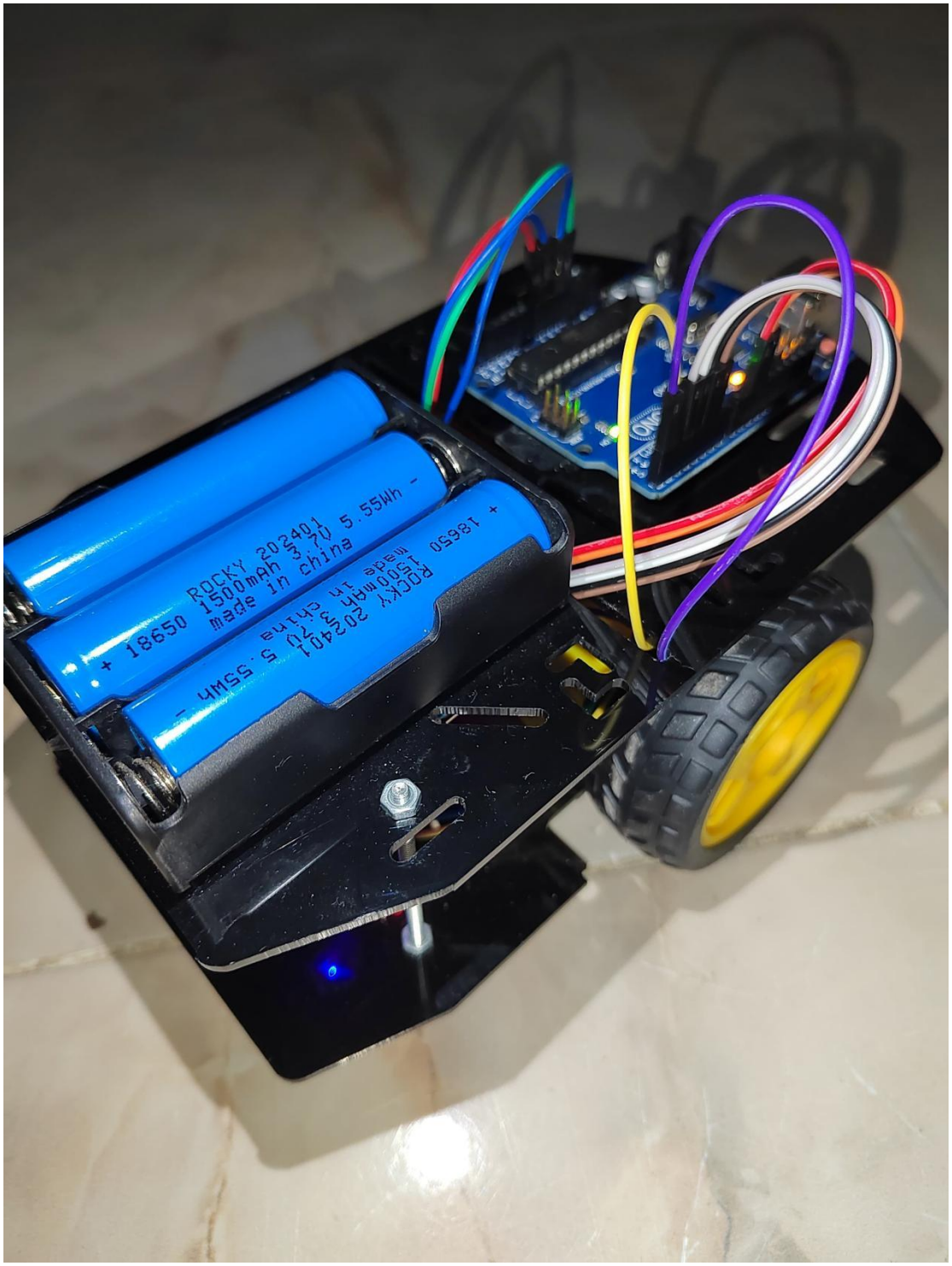
ASSIGNMENT 1



ASSIGNMENT 1



ASSIGNMENT 1



Source code of the system (with comments)

```
#include <SoftwareSerial.h>

// تعيين المداخل التي سيتم استخدامها للاتصال بالبلوتوث
SoftwareSerial BluetoothSerial(0, 1); // RX, TX

// تعريف مداخل H-Bridge
const int IN1 = 2; // المحرك اتجاه 1
const int IN2 = 3;
const int ENA = 9; // المحرك سرعة 1 (PWM)

const int IN3 = 4; // المحرك اتجاه 2
const int IN4 = 5;
const int ENB = 10; // المحرك سرعة 2 (PWM)

const int buzzerPin = 8;  // الدبوس إلى البازر توصيل 8

// البيانات لتخزين متغيرات
int speed = 0;      // السرعة
char direction = 's'; // الاتجاه (افتراضيًا: توقف)

void setup() {
 // كمخرجات المداخل إعداد
 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 pinMode(ENA, OUTPUT);

 pinMode(IN3, OUTPUT);
 pinMode(IN4, OUTPUT);
 pinMode(ENB, OUTPUT);

 pinMode(buzzerPin, OUTPUT);  // كمخرج البازر تحديد

 // التسلسلي الاتصال بدء
 Serial.begin(9600);        // مع الاتصال Serial Monitor
 BluetoothSerial.begin(9600); // البلوتوث مع الاتصال
 Serial.println("Bluetooth Connected!");
}

void loop() {
 // البلوتوث من واردة بيانات هناك كانت إذا
 if (BluetoothSerial.available()) {
  // البلوتوث من المرسلة البيانات قراءة
  char incomingByte = BluetoothSerial.read();

  // (للسرعة 5 إلى 0) رقمًا البيانات كانت إذا
  if (incomingByte >= '1' && incomingByte <= '8') {
   speed = (incomingByte - '0') * 10; // 0-100 نطاق إلى السرعة تحويل
   Serial.print("Speed: ");
   Serial.println(speed);
  } else { // للاتجاه حرفًا البيانات كانت إذا
   direction = incomingByte;

   // الاتجاه على بناءً العمليات تنفيذ
   controlMotors(direction, speed);

   // على الاتجاه طباعة Serial Monitor
```

```
    switch (direction) {
      case 'u':
        Serial.println("Forward");
        break;
      case 'd':
        Serial.println("Backward");
        break;
      case 'l':
        Serial.println("Left");
        break;
      case 'r':
        Serial.println("Right");
        break;
      case 's':
        Serial.println("Stop");
        break;
      case 'k':
        Serial.println("shoot");
        break;
      case 'a':
        Serial.println("Alarm");
        playSiren();  // الشرطة صفارة صوت تشغيل
        break;
      case 'm':
        Serial.println("Mute");
        noTone(buzzerPin); // الصوت إيقاف
        break;
      default:
        Serial.println("Unknown command");
        break;
    }
   }
  }
}

// المحركات في التحكم دالة
void controlMotors(char dir, int spd) {
  int motorSpeed = map(spd, 0, 80, 0, 255); // 0-255 نطاق إلى السرعة تحويل

  switch (dir) {
    case 'u': // للأمام
      digitalWrite(IN1, HIGH);
      digitalWrite(IN2, LOW);
      analogWrite(ENA, motorSpeed);

      digitalWrite(IN3, HIGH);
      digitalWrite(IN4, LOW);
      analogWrite(ENB, motorSpeed);
      break;

    case 'd': // للخلف
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, HIGH);
      analogWrite(ENA, motorSpeed);

      digitalWrite(IN3, LOW);
      digitalWrite(IN4, HIGH);
```

```
  analogWrite(ENB, motorSpeed);
  break;

case 'l': // لليسار (يعمل واحد محرك)
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, motorSpeed);

  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, 0); // الآخر المحرك إيقاف
  break;

case 'r': // لليمين (يعمل واحد محرك)
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0); // الآخر المحرك إيقاف

  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, motorSpeed);
  break;

case 's': // توقف
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);

  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, 0);
  break;

case 'k': // توقف
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 255);

  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, 255);
  delay(500);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 255);

  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(ENB, 255);
  delay(500);
  break;

default: // توقف - أخرى حالة أي
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);
```

```
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    analogWrite(ENB, 0);
    break;
  }
}

// الشرطة صفارة لتشغيل دالة
void playSiren() {
  // الشرطة صفارة ترددات
  tone(buzzerPin, 1000);  // تردد أول
  delay(200);             // الترددات بين تأخير
  tone(buzzerPin, 1500);  // ثاني تردد
  delay(200);             // الترددات بين تأخير
  tone(buzzerPin, 1000);  // أول تردد
  delay(200);             // الترددات بين تأخير
  tone(buzzerPin, 1500);  // ثاني تردد
  delay(200);             // الترددات بين تأخير
  noTone(buzzerPin);      // الصوت إيقاف
}
```