Shahjalal University of Science and Technology
Department of Computer Science and Engineering

Course No.: CSE-367

Course Name: Microprocessors and Interfacing

Course Teacher: Abdullah Al Noman

Lecturer, Dept. of CSE, SUST.

Lab Project Name: Bluetooth Control Car with Automatic Braking System.

Lab Report Submission Date: 10.07.24

Group: 09

Github Link of Project: Khalidgithub2020331007/Bluetooth-Control-Car-with-Automatic-Braking-System

Registration Number	Name
2020331007	SHEIKH KHALID AZAD
2020331021	MD. MUZTAHID HASSAN
2020331031	ABU KAWSAR MD GOLAM SARWAR
2020331047	MD. MINHAJUL HAQUE
2020331049	SAHIDUR RAHMAN

Introduction:

A Bluetooth control car with an automatic braking system is a sophisticated remote-controlled vehicle that integrates Bluetooth technology and advanced safety features. Through a smartphone or a Bluetooth-enabled controller, users can wirelessly manage the car's movements, including acceleration, steering, and stopping. The automatic braking system enhances safety by using sensors to detect obstacles and automatically apply the brakes to prevent collisions. This combination of Bluetooth control and automatic braking makes the car both convenient to operate and safer to use, offering an engaging and secure driving experience.

Materials:

We used some materials to complete our work. And they are:

Basic Components:

1. A Chassis
2. Two Motors
3. Two Wheels

Electronics:

1. A Microcontroller
2. A Bluetooth Module.
3. A Ultrasonic sensors .
4. A Battery.

Additional Components:

1. Breadboard and Jumper Wires.
2. Soldering Iron and Solder.
3. Multimeter.
4. Screwdrivers.
5. Double Sided Tape.

Software:

1. Arduino IDE.
2. Bluetooth Control App.

Project Diagram:

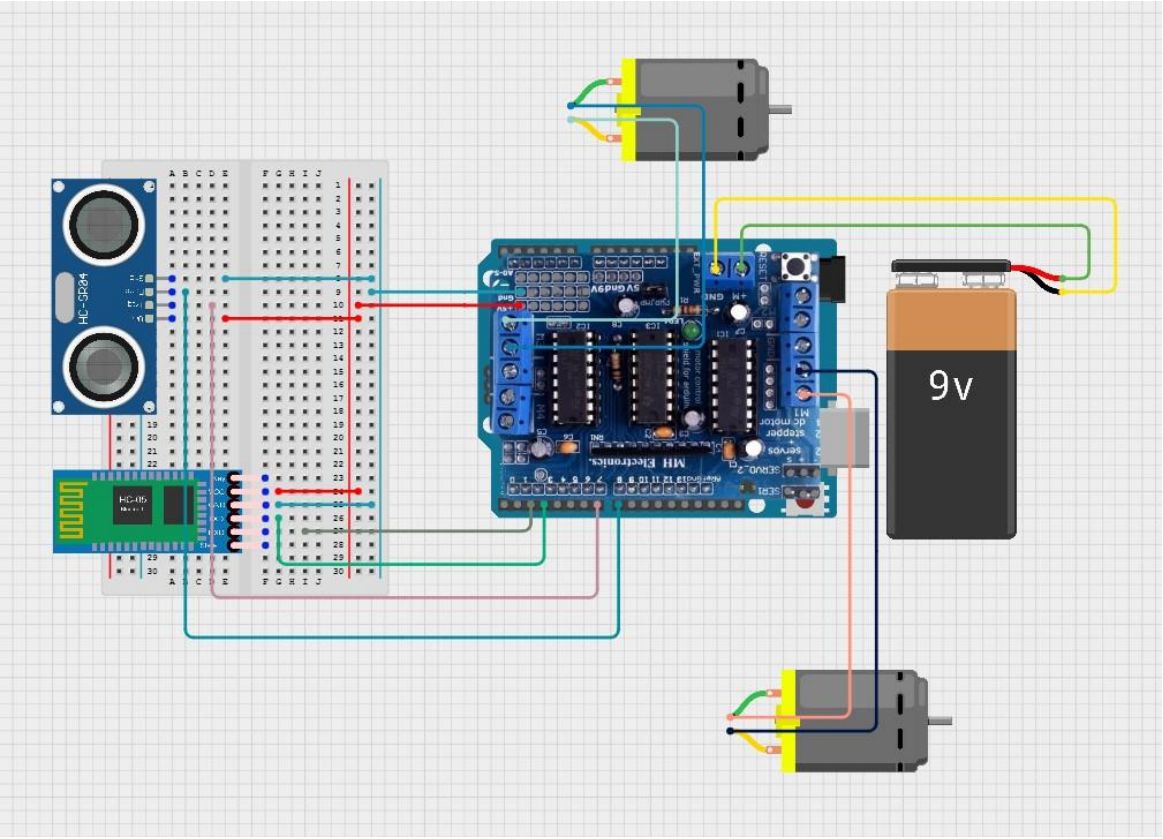

Figure: Diagram of Bluetooth Control Car with Automatic Braking System.

Project Description:

This project involves building a Bluetooth-controlled car with an automatic braking system using an Arduino Uno, an Adafruit Motor Shield, and an ultrasonic sensor. The car can be controlled via a Bluetooth-enabled device and will automatically stop and reverse when an obstacle is detected within a certain distance.

Components and Their Functions:

Arduino Uno: The microcontroller that serves as the brain of the car, processing inputs and controlling outputs.

Adafruit Motor Shield: A shield that allows easy control of DC motors, including setting speed and direction.

Bluetooth Module (HC-05): Enables wireless communication between the car and a Bluetooth-enabled device.

DC Motors: Provide movement for the car. In this setup, two motors are used, one for each side of the car.

Ultrasonic Sensor (HC-SR04): Detects obstacles by emitting ultrasonic waves and measuring the time it takes for the waves to bounce back.

SoftwareSerial Library: Allows serial communication on other digital pins of the Arduino, enabling Bluetooth communication.

AFMotor Library: Provides an easy interface to control the motors using the Adafruit Motor Shield.

Code:

Combine Code of Bluetooth and ultrasonic sensor that we used:

```
#include <AFMotor.h>
#include <SoftwareSerial.h>

// Define motor ports on the Adafruit Motor Shield
AF_DCMotor motor1(1, MOTOR12_1KHZ);
AF_DCMotor motor3(3, MOTOR34_1KHZ);

// Define Bluetooth module pins (using SoftwareSerial)
SoftwareSerial Bluetooth(2, 3); // RX, TX

// Define speed for the motors
const int SPEED = 200; // Adjust speed as necessary

// Define ultrasonic sensor pins
#define trigPin 7
#define echoPin 8

void setup() {
  Serial.begin(9600);
  Bluetooth.begin(9600);
  Serial.println("Motor, Bluetooth, and Ultrasonic Test");

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  long distance = getDistance();
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
```

```
  if (distance > 0 && distance < 15) { // Check for valid distance
    Serial.println("Obstacle detected! Stopping and reversing...");

    back();
    delay(2000);
    Stop();
  } else {
    if (Bluetooth.available() > 0) {
      char command = Bluetooth.read();
      Serial.print("Received: ");
      Serial.println(command);

      switch (command) {
        case 'F':
          forward();
          break;
        case 'B':
          back();
          break;
        case 'L':
          left();
          break;
        case 'R':
          right();
          break;
        case 'T':
          Stop();
          break;
        default:
          Serial.println("Invalid command");
          break;
      }
    }
  }

  delay(500);
}
```

```
long getDistance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  if (duration == 0) {
     Serial.println("Warning: No echo received.");
     return -1; // Indicate error
  }
  long distance = duration / 58.2;
  return distance;
}

void forward() {
  Serial.println("Forward");
  motor1.setSpeed(SPEED);
  motor1.run(FORWARD);
  motor3.setSpeed(SPEED);
  motor3.run(FORWARD);
}

void back() {
  Serial.println("Backward");
  motor1.setSpeed(SPEED);
  motor1.run(BACKWARD);
  motor3.setSpeed(SPEED);
  motor3.run(BACKWARD);
}

void left() {
  Serial.println("Left");
  motor1.setSpeed(0);
  motor1.run(RELEASE);
```

```
  motor3.setSpeed(SPEED);
  motor3.run(FORWARD);
}

void right() {
  Serial.println("Right");
  motor1.setSpeed(SPEED);
  motor1.run(FORWARD);
  motor3.setSpeed(0);
  motor3.run(RELEASE);
}

void Stop() {
  Serial.println("Stop");
  motor1.setSpeed(0);
  motor1.run(RELEASE);
  motor3.setSpeed(0);
  motor3.run(RELEASE);
}
```

Code Explanation:

Libraries and Definitions:

AFMotor.h: Controls the motors via the Adafruit Motor Shield.

SoftwareSerial.h: Enables serial communication with the Bluetooth module on pins 2 (RX) and 3 (TX).

motor1 and motor3: Instances of AF_DCMotor class representing two motors connected to ports 1 and 3 of the motor shield.

trigPin and echoPin: Pins for the ultrasonic sensor.

Setup Function:

Initializes serial communication for debugging and Bluetooth communication. Sets the trigPin as an output and echoPin as an input for the ultrasonic sensor.

Loop Function:

Continuously measures the distance using the ultrasonic sensor.
If an obstacle is detected within 15 cm, the car stops and reverses for 2 seconds.If no obstacle is detected, the car checks for commands from the Bluetooth module and executes corresponding movements (forward, backward, left, right, stop).

getDistance Function:

Sends an ultrasonic pulse and measures the time it takes for the echo to return.
Converts the time into distance in centimeters.
Returns -1 if no echo is received, indicating an error.

Movement Functions:

forward(): Sets both motors to move the car forward.
back(): Sets both motors to move the car backward.
left(): Stops the left motor and runs the right motor forward to turn left.
right(): Stops the right motor and runs the left motor forward to turn right.
Stop(): Stops both motors.

How It Works:

Initialization: When the car is powered on, it initializes the motors, Bluetooth module, and ultrasonic sensor.

Distance Measurement: The ultrasonic sensor continuously measures the distance to any obstacle in front of the car.

Automatic Braking: If an obstacle is detected within 15 cm, the car automatically stops and reverses to avoid a collision.

Bluetooth Control: The car listens for commands from a paired Bluetooth device. Based on the received command ('F', 'B', 'L', 'R', 'T'), it moves forward, backward, left, right, or stops.

Feedback: The car sends feedback to the serial monitor, displaying the received commands and the measured distance.

This project demonstrates a combination of manual control via Bluetooth and autonomous safety features using an ultrasonic sensor, making it a practical and educational example of robotics and embedded systems.

Result:

We were unable to complete our project because it is not feasible with the Arduino Uno.

The Arduino Uno uses a single-core microcontroller, specifically the ATmega328P, which is an 8-bit AVR RISC-based microcontroller. This means it can execute only one instruction at a time. As a result, it cannot effectively handle both Bluetooth communication and ultrasonic sensor measurements simultaneously.

While we were able to perform tasks individually, such as controlling the car via Bluetooth or using the ultrasonic sensor for obstacle detection, we could not perform both tasks concurrently due to the limitations of the single-core microcontroller.

Separate code of bluetooth part:

```
#include <AFMotor.h>
#include <SoftwareSerial.h>

// Define motor ports on the Adafruit Motor Shield
AF_DCMotor motor1(1, MOTOR12_1KHZ);
AF_DCMotor motor3(3, MOTOR34_1KHZ);

// Define Bluetooth module pins (using SoftwareSerial)
SoftwareSerial Bluetooth(2, 3); // RX, TX

// Define speed for the motors
const int SPEED = 255;

void setup() {
  Serial.begin(9600);
  Bluetooth.begin(9600);
  Serial.println("Motor and Bluetooth Test");
}

void loop() {
  if (Bluetooth.available() > 0) {
    char command = Bluetooth.read();
    Serial.print("Received: ");
    Serial.println(command);

    switch (command) {
      case 'F':
        forward();
        break;
      case 'B':
        back();
        break;
      case 'L':
        left();
        break;
      case 'R':
        right();
        break;
      case 'T':
        Stop();
        break;
      default:
```

```
      Serial.println("Invalid command");
      break;
    }
  }
}

void forward() {
  Serial.println("Forward");
  motor1.setSpeed(SPEED);
  motor1.run(FORWARD);
  motor3.setSpeed(252);
  motor3.run(FORWARD);
}

void back() {
  Serial.println("Backward");
  motor1.setSpeed(SPEED);
  motor1.run(BACKWARD);
  motor3.setSpeed(252);
  motor3.run(BACKWARD);
}

void left() {
  Serial.println("Left");
  motor1.setSpeed(0); // Stop the left motor
  motor1.run(RELEASE);
  motor3.setSpeed(SPEED); // Move the right motor forward
  motor3.run(FORWARD);
}

void right() {
  Serial.println("Right");
  motor1.setSpeed(SPEED); // Move the left motor forward
  motor1.run(FORWARD);
  motor3.setSpeed(0); // Stop the right motor
  motor3.run(RELEASE);
}

void Stop() {
  Serial.println("Stop");
  motor1.setSpeed(0);
  motor1.run(RELEASE);
  motor3.setSpeed(0);
  motor3.run(RELEASE);
}
```

Discussion:

Project Overview

Our project aimed to build a Bluetooth-controlled car with an automatic braking system using an Arduino Uno. The goal was to control the car via a Bluetooth-enabled device and have it automatically stop and reverse when an obstacle was detected by an ultrasonic sensor. The key components included an Arduino Uno, an Adafruit Motor Shield, a Bluetooth module, and an ultrasonic sensor.

Challenges and Limitations

While working on this project, we encountered a significant limitation due to the Arduino Uno's microcontroller. The ATmega328P microcontroller used in the Arduino Uno is an 8-bit AVR RISC-based single-core microcontroller. This design allows it to execute only one instruction at a time, which posed a challenge for our project requirements.

Single-Core Limitation: The single-core nature of the ATmega328P meant that it could not handle Bluetooth communication and ultrasonic sensor measurements simultaneously. When attempting to process data from both the Bluetooth module and the ultrasonic sensor concurrently, we faced performance issues and unreliable operation.

Task Execution:

We were able to successfully complete individual tasks such as:

Controlling the car via Bluetooth, allowing it to move forward, backward, left, right, and stop based on commands received from a Bluetooth-enabled device.

Using the ultrasonic sensor to detect obstacles and stop the car to prevent collisions.

However, combining these tasks proved infeasible due to the microcontroller's inability to multitask effectively.

Timing and Interrupts: Handling precise timing for both Bluetooth communication and ultrasonic sensor readings required careful

management of interrupts and delays, which further complicated the implementation.

Lessons Learned

Hardware Constraints: The project highlighted the importance of understanding hardware constraints and choosing the right microcontroller for multitasking applications. For projects requiring simultaneous processing of multiple tasks, a more powerful microcontroller or a multicore processor might be necessary.

Modular Design: Designing the system in a modular way, where each component (Bluetooth module and ultrasonic sensor) was tested independently, helped in isolating and identifying the limitations.

Potential Solutions:

Upgrade Microcontroller: Using a more powerful microcontroller such as the ESP32, which has dual cores and integrated Bluetooth, could handle multiple tasks more efficiently.

Optimize Code: Optimizing the code to reduce the processing load and improve the handling of tasks through better use of interrupts and efficient programming practices.

Additional Microcontroller: Incorporating an additional microcontroller to handle specific tasks separately, thus distributing the workload.

Conclusion

While the project demonstrated the feasibility of controlling a car via Bluetooth and implementing an automatic braking system using an ultrasonic sensor, the single-core limitation of the Arduino Uno's ATmega328P microcontroller hindered our ability to execute both tasks concurrently. This experience underscores the need to match project requirements with the appropriate hardware capabilities. For future projects with similar multitasking needs, exploring more advanced microcontrollers or alternative architectures will be crucial for success.

Overall, despite the challenges faced, the project provided valuable insights into the limitations of single-core microcontrollers and the importance of hardware selection in embedded systems design.