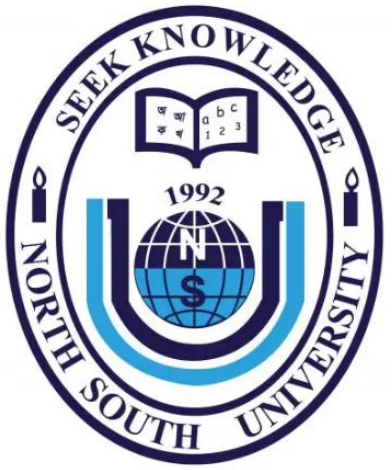


NORTH SOUTH UNIVERSITY

PROJECT REPORT

GROUP 06
GESTURED CONTROL AUTONOMOUS ROBOT CAR

NAME	ID
Safwan Ul Islam	2112173642
Sara Sultana	2132283042
Syed Abrar Reshad	2212277042
Nafis Fuad Bin Hafiz	2212654042

Table of Contents

Abstract

This project outlines the development of an advanced robot car that integrates hand gesture control with autonomous navigation capabilities, providing a versatile and interactive user experience. The robot operates in three distinct modes: gesture control, obstacle avoidance, and line following. Using an MPU6050 sensor for real-time gesture recognition and Arduino microcontrollers for processing, users can intuitively guide the robot's movements through hand gestures while easily switching between autonomous modes based on environmental conditions. The obstacle avoidance mode employs ultrasonic sensors to navigate around obstacles, ensuring safe and efficient movement, while the line following mode allows the robot to autonomously track pre-defined paths. This project aims to merge natural human-machine interaction with sophisticated autonomous technology, demonstrating the potential of gesture-based control in enhancing both manual and automated robotic systems. It also highlights the growing trend towards interactive and adaptive robotic systems, providing a practical solution that can be applied to various real-world applications, such as manufacturing, logistics, and personal assistance.

1. Introduction

1.1 Overview

The robot car project focuses on creating a robotic vehicle that combines manual control via hand gestures and sophisticated autonomous navigation features. Users can control the car's movements using gestures detected by an MPU6050 sensor connected to an Arduino Nano. The car operates in three modes: Gesture Control, Obstacle Avoidance, and Line Following.

1.2 Problem Statement

The increasing demand for interactive and adaptive robotic systems in fields such as manufacturing, logistics, and personal assistance highlights the need for systems that can efficiently perform tasks with minimal human intervention. Traditional robotic systems, which often rely on fixed, manual controls or basic autonomous features, lack the flexibility and responsiveness required for dynamic environments. Additionally, conventional control interfaces may not provide an intuitive or ergonomic experience, making it challenging for users to effectively operate robotic systems. To address these challenges, this project aims to develop a robotic car that integrates intuitive gesture-based control with advanced autonomous capabilities, enhancing both usability and functionality.

1.3 Project Aim & Objectives

The project aim is to develop a robot car that integrates gesture-based manual control with advanced autonomous navigation modes, including obstacle avoidance and line following.

The project objectives are given below:

- Gesture Control: Implement gesture recognition using an MPU6050 sensor.
- Obstacle Avoidance: Develop obstacle detection using ultrasonic sensors.
- Line Following: Implement line-following algorithm using line sensors.
- Mode Switching: Design a method to switch between Gesture Control mode, Obstacle Avoidance mode and Line Following mode.

2. Background Study

2.1 Gesture Control in Robotics

Gesture control technology has been increasingly explored in robotics due to its intuitive nature. Sensors like the MPU6050, which combines an accelerometer and gyroscope, enable precise gesture detection, allowing users to control robots in a natural and interactive manner.

2.2 Obstacle Avoidance in Robotics

Obstacle avoidance is a critical aspect of autonomous robotics, allowing robots to navigate safely in environments with obstacles. Various methods, such as ultrasonic sensors, infrared sensors have been used for obstacle detection. Ultrasonic sensors, like those used in this project, are cost-effective and provide reliable distance measurements by emitting and receiving sound waves. The main challenges in obstacle avoidance include accurately detecting obstacles in different environmental conditions and responding quickly to prevent collisions.

2.3 Line Following in Robotics

Line following is another significant aspect of autonomous navigation, where robots follow a predefined path marked on the ground. This is commonly achieved using line sensors that detect the contrast between the line and the background surface. The primary challenge in line following is maintaining accuracy at varying speeds and handling complex paths, such as sharp turns or intersections. Line following has applications in automated material handling, logistics, and assembly line operations, providing a robust solution for repetitive path-following tasks.

2.4 Related Projects

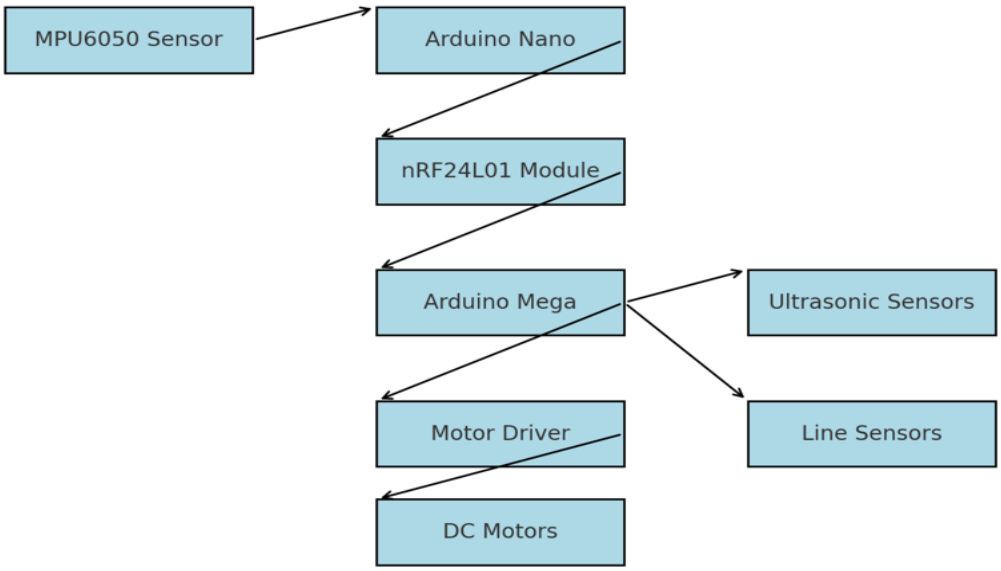
This project is inspired by various robotics projects that utilize gesture control, obstacle avoidance, and line following. Existing projects often implement one or two of these features independently, but our project aims to combine all three features into a single cohesive system. This integration makes the robot car more versatile, adaptable, and capable of addressing multiple real-world challenges efficiently.

2.5 Unique Contribution of This Work

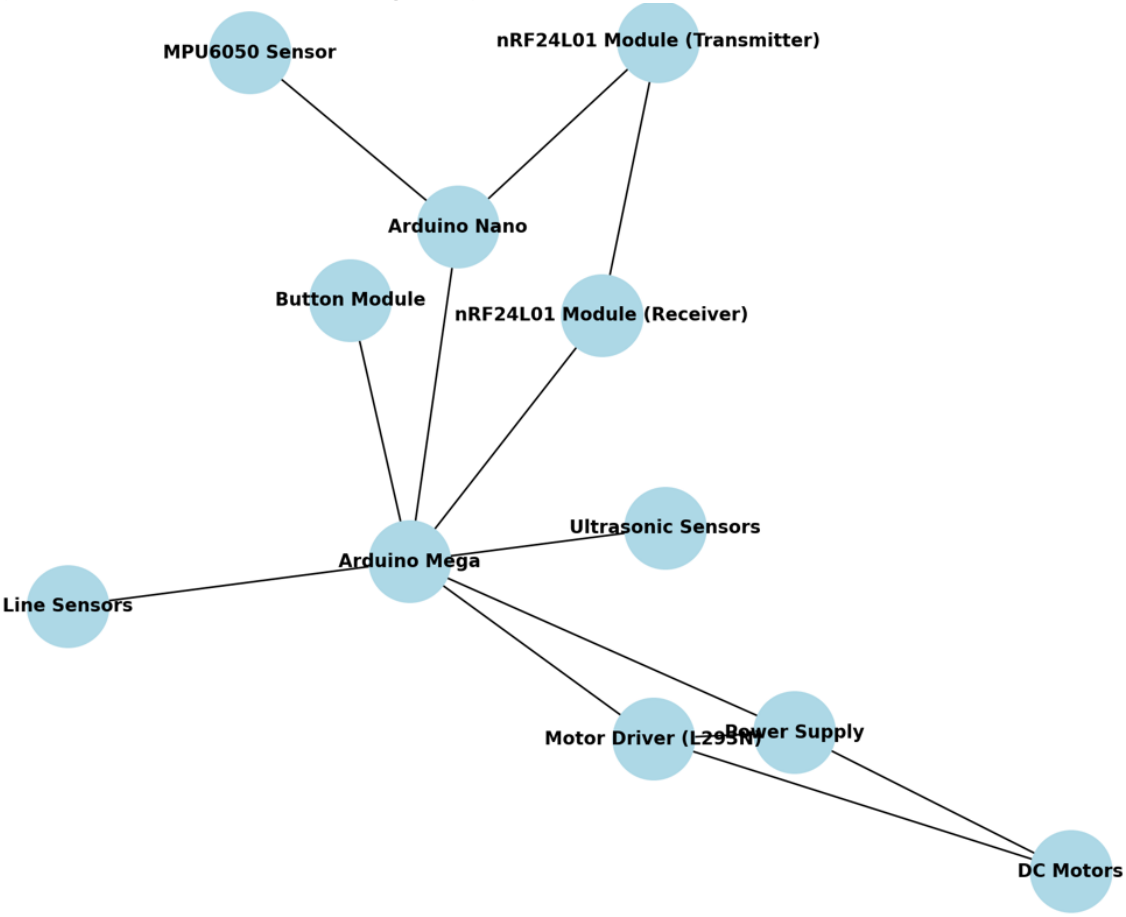
The unique contribution of this project lies in the seamless integration of gesture-based control with autonomous features such as obstacle avoidance and line following, which are typically implemented separately in most projects. Unlike existing systems that focus solely on one or two functionalities, our project offers a complete solution by combining gesture control, obstacle avoidance, and line following into a unified system. This approach demonstrates an innovative blend of natural human-machine interaction with advanced autonomous capabilities, setting our work apart from other projects in terms of versatility, adaptability, and real-world applicability.

3. Model

3.1 Block Diagram



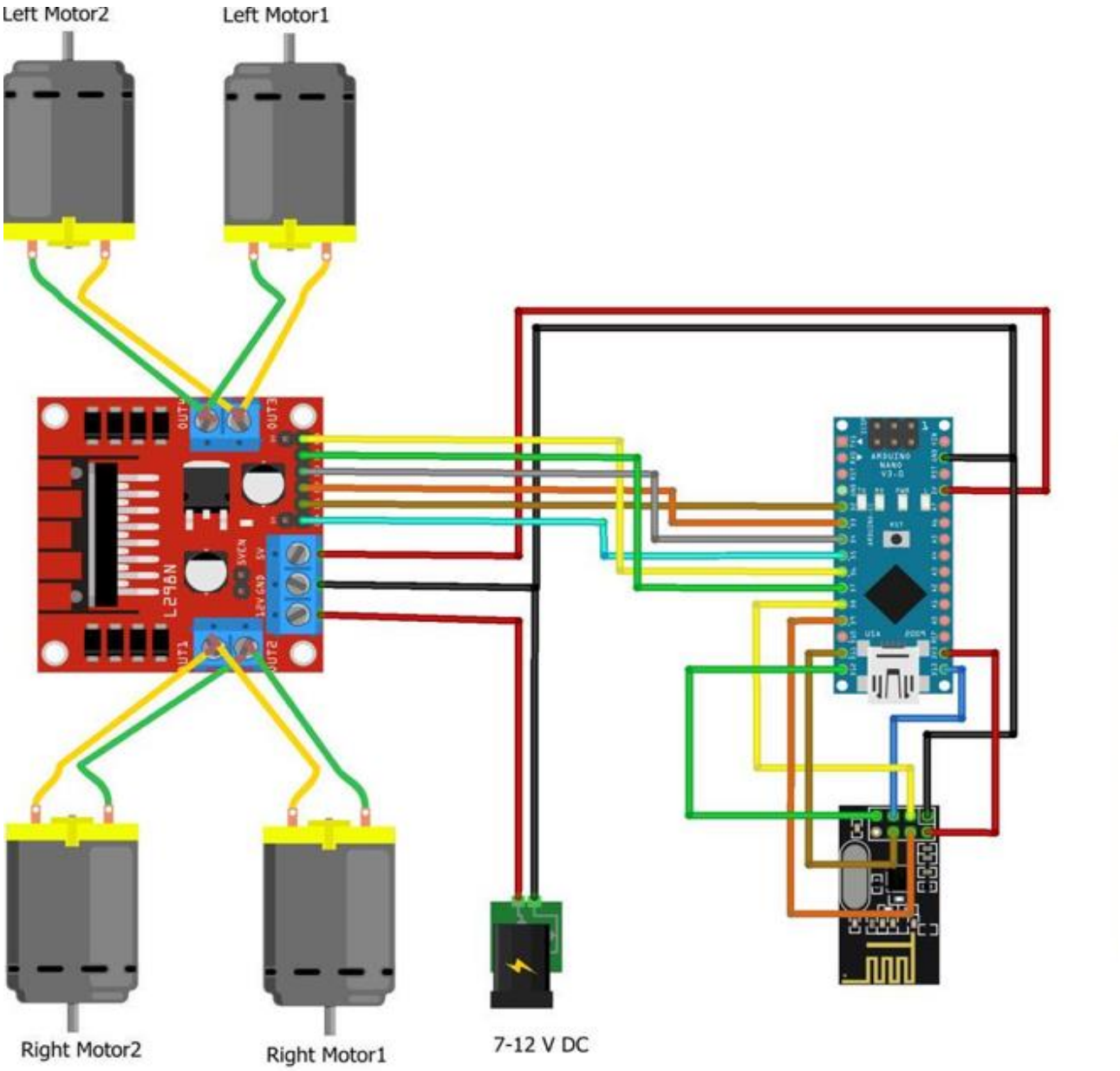
(More in detail block diagram)



3.2 Circuit Schematic

The hardware integration includes:

- MPU6050 connected to Arduino Nano for gesture control.
- Ultrasonic sensors and line sensors connected to Arduino Mega.
- Motor control using L293N motor driver.
- Wireless communication via nRF24L01 modules.



4. Implementation

4.1 Hardware Setup

- **Transmitter Wire Connections**

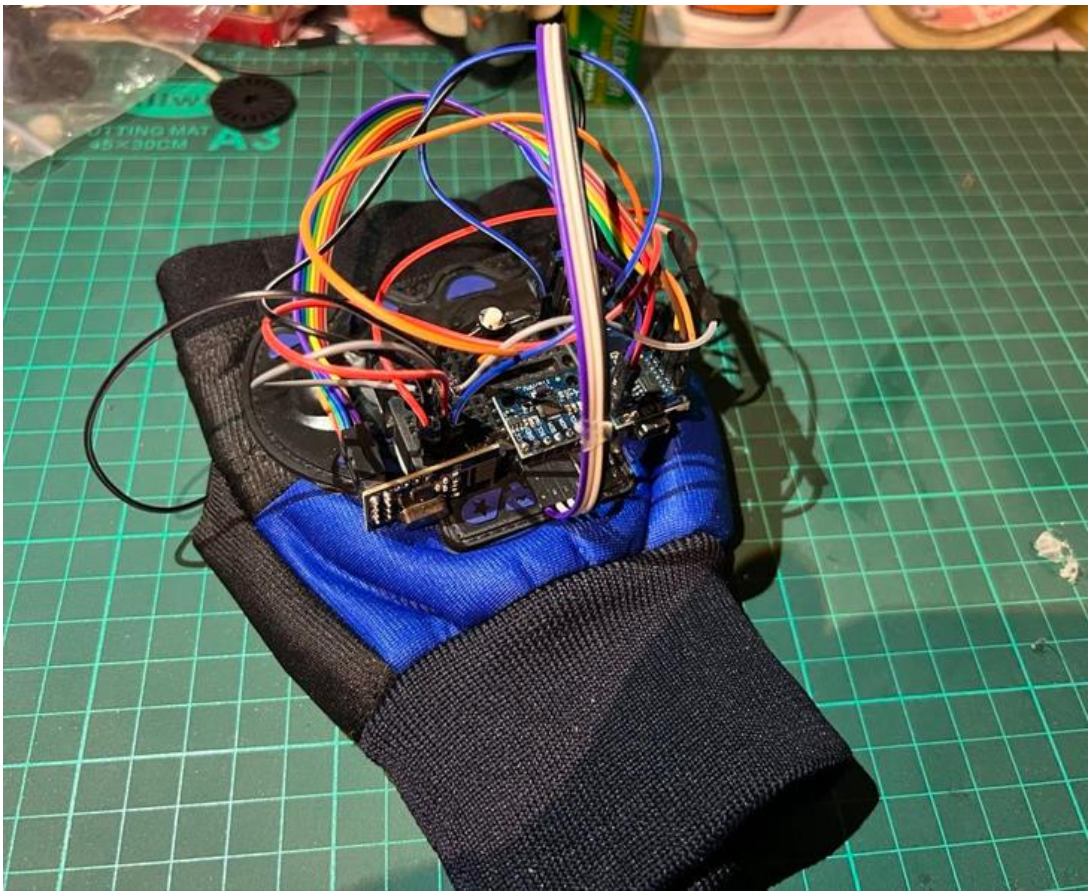
1. **Arduino Nano**
   I. Connect the VIN pin of the Arduino Nano to a 5V power source.
   II. Connect the GND pin to the ground of the power supply.

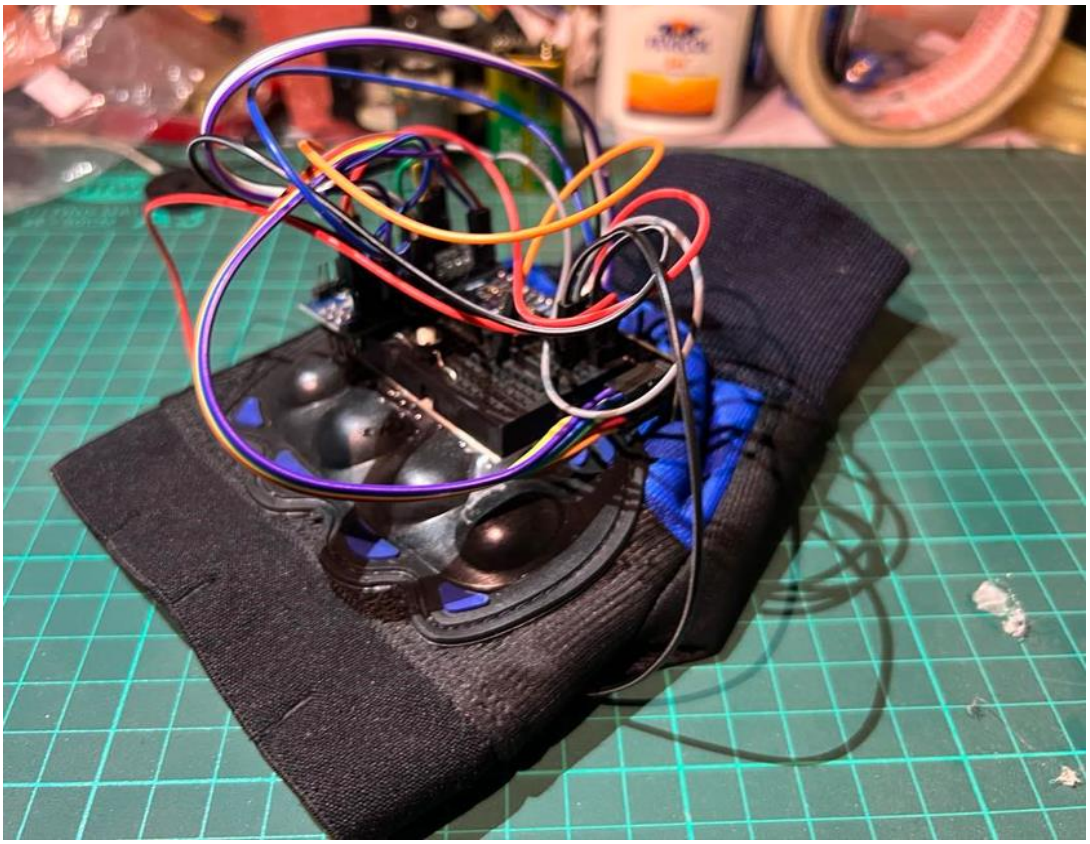
2. **Nrf24L01 Module**
   I. GND: Connect to GND on the Arduino.
   II. VCC: Connect to 3.3V on the Arduino (Do NOT connect to 5V, as it can damage the module).
   III. CE: Connect to D8 on the Arduino.
   IV. CSN: Connect to D10 on the Arduino.
   V. SCK: Connect to D13 on the Arduino.
   VI. MOSI: Connect to D11 on the Arduino.
   VII. MISO: Connect to D12 on the Arduino.

3. **MPU6050 (Gyroscope and Accelerometer Sensor)**
   I. VCC: Connect to 3.3V or 5V on the Arduino Nano (MPU6050 is compatible with both voltages).
   II. GND: Connect to GND on the Arduino.
   III. SCL: Connect to A5 on the Arduino (I2C Clock line).
   IV. SDA: Connect to A4 on the Arduino (I2C Data line).

4. **Mode Selector (Push Button)**
   I. Connect one leg of the push button to D2 (digital input pin) on the Arduino.
   II. Connect the other leg of the button to GND.
   III. Use a 10kΩ pull-up resistor to connect the D2 pin to 5V, or use the built-in pull- up resistor in the Arduino by declaring INPUT_PULLUP in the code

- **Transmitter Model**





- **Receiver Wire Connections**

1. **Arduino Mega**
   I. Connect the VIN pin to a 7-12V power source.
   II. Connect the GND pin to the ground of the power source.

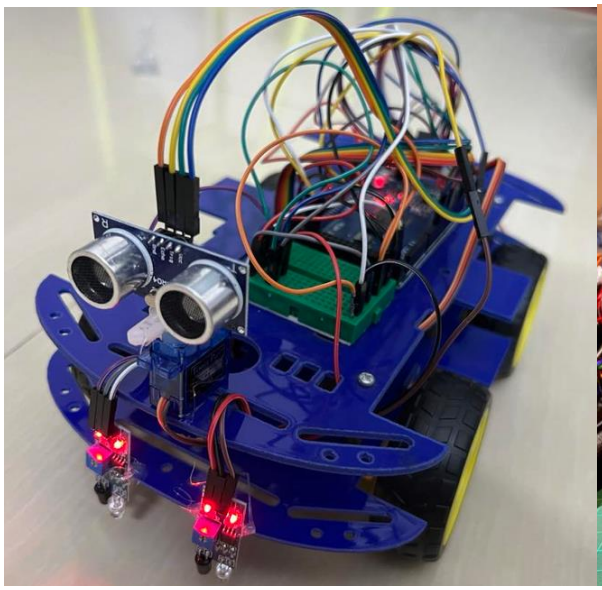
2. **Nrf24L01 Module**
   I. GND: Connect to GND on the Arduino.
   II. VCC: Connect to 3.3V on the Arduino (Do NOT connect to 5V, as it may damage the module).
   III. CE: Connect to D8 on the Arduino.
   IV. CSN: Connect to D10 on the Arduino.
   V. SCK: Connect to D52 on the Arduino.
   VI. MOSI: Connect to D51 on the Arduino.
   VII. MISO: Connect to D50 on the Arduino.

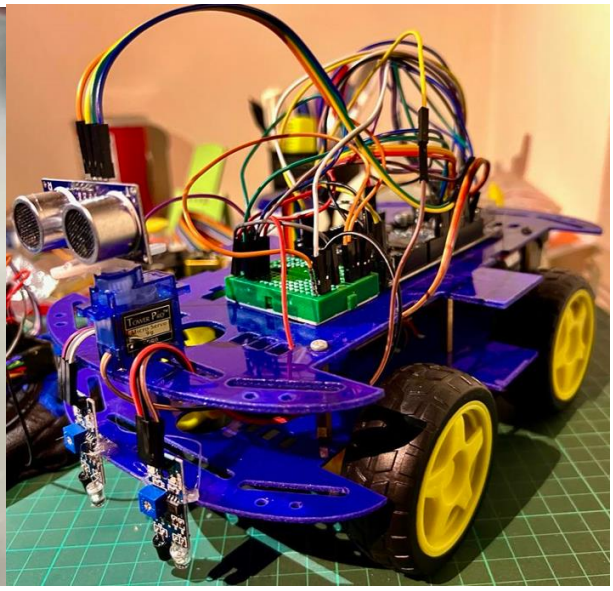
3. **L2G8N Motor Driver**
   I. Connect the +12V input of the L298N to the 12V power source.
   II. Connect the GND pin of the L298N to the ground of the power source and the Arduino.
   III. ENA (Enable A): Connect to D3 on the Arduino.
   IV. ENB (Enable B): Connect to DG on the Arduino.
   V. IN1: Connect to D4 on the Arduino.
   VI. IN2: Connect to D5 on the Arduino.
   VII. IN3: Connect to D6 on the Arduino.
   VIII. IN4: Connect to D7 on the Arduino.
   IX. Connect Motor A wires to OUT1 and OUT2 on the L298N.
   X. Connect Motor B wires to OUT3 and OUT4 on the L298N.

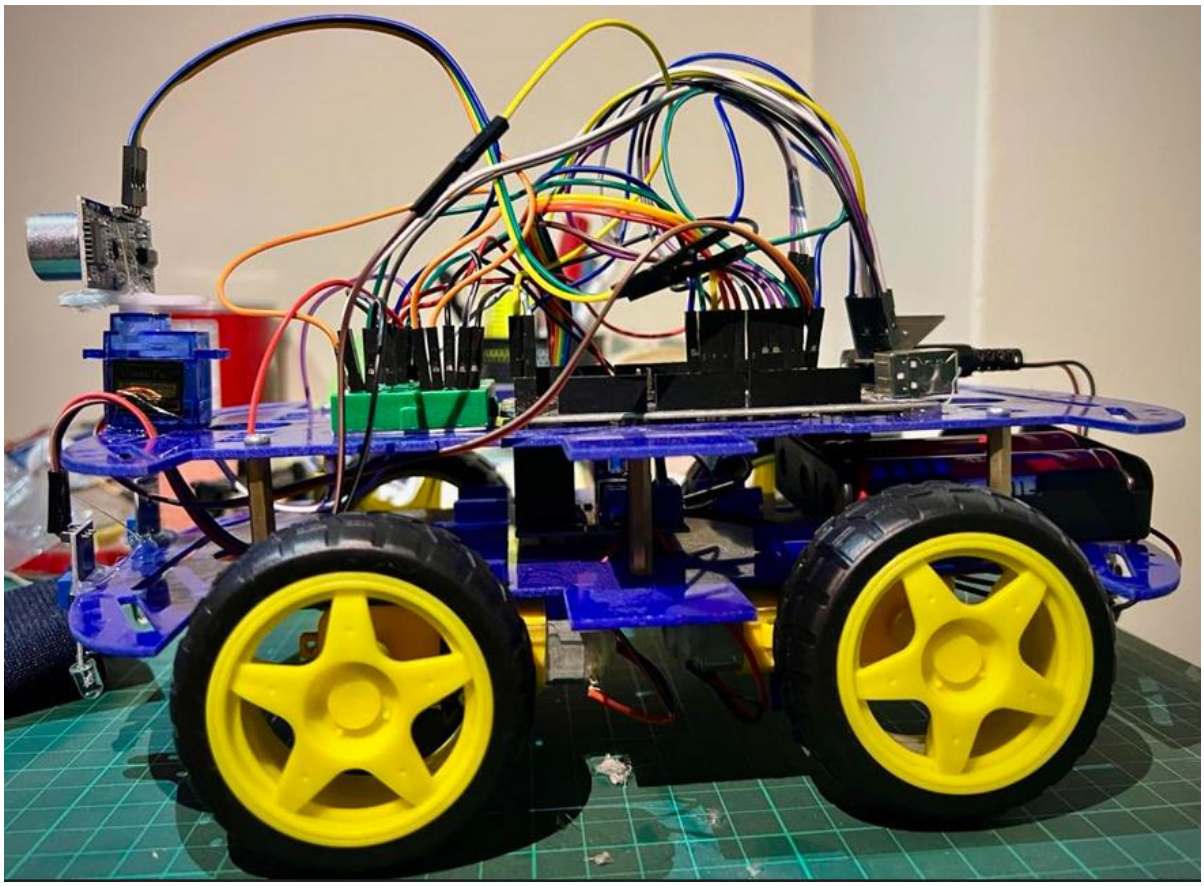
4. **Servo Motor**
   I. Connect to D13 on the Arduino.
   II. Connect the VCC pin to the 5V on the Arduino (or an external power source if servo power requirements are high).
   III. Connect the GND pin to the ground.

5. **Ultrasonic Sensor (HC-SR04)**
   I. Trig Pin: Connect to D12 on the Arduino.
   II. Echo Pin: Connect to D11 on the Arduino.
   III. Connect the VCC pin to the 5V on the Arduino.
   IV. Connect the GND pin to the ground

- **Receiver Model**







4.2 Software Development

- **Data Processing:** Developed firmware for the Arduino Nano to interpret data from the MPU6050 sensor, translating it into recognizable gestures. Implemented algorithms to detect specific hand gestures and map them to car movements, enabling intuitive and responsive manual control of the robot.
- **Data Transmission:** Wrote code to transmit gesture data from the Arduino Nano to the receiver unit via the nRF24L01 module. Ensured that data packets were correctly formatted and transmitted with minimal delay, maintaining consistent communication between the transmitter and receiver.
- **Mode Handling:** Developed software for the Arduino Mega to handle different operational modes based on the received gesture data. Implemented logic to switch between Gesture Control, Obstacle Avoidance, and Line Following modes seamlessly. This allowed users to easily transition between manual and autonomous operations, enhancing the flexibility of the robot.
- **Autonomous Navigation:** Created and integrated algorithms for obstacle avoidance using ultrasonic sensors and line following using line sensors. Developed code to ensure that the ultrasonic sensors accurately detected obstacles and the line sensors consistently tracked the predefined path. The algorithms were designed to interact effectively with gesture-based controls, allowing smooth operation across different modes.
- **System Integration:** Combined all hardware and software components into a cohesive system. Ensured proper integration of sensors, controllers, and communication modules. Tested the interaction between gesture control and autonomous modes to guarantee seamless functionality.
- **Performance Testing:** Conducted extensive tests to validate the accuracy of gesture recognition, the efficiency of obstacle avoidance, and the precision of line following. Addressed any issues or bugs identified during testing to enhance the overall system reliability and performance. Collected data on gesture recognition rates, obstacle detection distances, and line-following success rates to evaluate the system's effectiveness.

5. Results and Discussions

5.1 Measurements and Data Analysis

- Gesture Recognition: The system achieved a gesture recognition accuracy of approximately 90% under controlled conditions.
- Obstacle Avoidance: Successful navigation in various environments, with obstacles detected within a range of 100 cm.
- Line Following: The car consistently followed a predefined path, demonstrating reliable sensor integration.

5.2 Challenges and Solutions

Week 1: Research and Initial Setup

1. Challenge: Identifying suitable components for the project.
    - Reason: There were many options for sensors and microcontrollers, and choosing the right ones required careful research.
    - Solution: Focused on components that were cost-effective, widely used, and supported by libraries (e.g., MPU6050, Arduino).

Week 2: Gesture Control Setup and Testing

2. Challenge: Understanding sensor data.
    - Reason: The MPU6050 outputs complex accelerometer and gyroscope readings, which were initially hard to interpret.
    - Solution: Used the Serial Monitor to observe data changes during different gestures.
3. Challenge: Defining thresholds for gestures.
    - Reason: Setting values too low caused false triggers, while setting them too high made the gestures unresponsive.
    - Solution: Adjusted thresholds through trial and error.
4. Challenge: Ensuring accurate stop gestures.
    - Reason: Setting zero for "stop" was unreliable due to sensor noise.
    - Solution: Used a range of values (e.g., -2000 to 2000) to define a neutral or stop state.

Week 3: Hardware Assembly and Motor Control Setup

5. Challenge: Ensuring stable motor connections.
    - Reason: Loose wires or unstable connections caused inconsistent motor movements.
    - Solution: Soldering the wires and verified each motor's wiring.
6. Challenge: Debugging motor movements.
    - Reason: Motors didn't always respond correctly to forward, backward, or turning commands.
    - Solution: Tested each motor individually and adjusted the code to match the wiring.
7. Challenge: Power management.
    - Reason: Motors drew too much power, causing voltage drops and unstable behavior.
    - Solution: Used a separate battery pack to power the motors.

Week 4: Wireless Communication and Obstacle Avoidance Development

11. Challenge: Setting up wireless communication.
    - Reason: The nRF24L01 module had connection stability issues and limited range.
    - Solution: Ensured proper wiring and added capacitors to stabilize the module.
12. Challenge: Tuning the ultrasonic sensor.
    - Reason: Obstacle detection was inconsistent, especially with smaller or angled objects.
    - Solution: Adjusted sensor placement and tested in different conditions.
13. Challenge: Prioritizing obstacle avoidance.
    - Reason: When obstacles were detected, other functionalities like line following conflicted.
    - Solution: Ensured obstacle avoidance had the highest priority in the logic.

Week 5: Line Following and Mode Switching

14. Challenge: Adjusting line sensors.
    - Reason: The sensors sometimes failed to detect the line or followed the wrong path due to light conditions or uneven surfaces.
    - Solution: Calibrated the sensors by adjusting their sensitivity and tested on different surfaces.
15. Challenge: Switching between modes.
    - Reason: The robot sometimes tried to execute commands from multiple modes simultaneously.
    - Solution: Used a variable to track the current mode and ensured only one mode was active at a time.

Week 6: Combining All Features

17. Challenge: Code conflicts between functionalities.
    - Reason: Each feature had its own logic, and combining them led to overlapping commands.
    - Solution: Used a state machine approach to separate and prioritize functionalities.
18. Challenge: Sensor overlap issues.
    - Reason: Multiple sensors being active simultaneously caused conflicting behavior (e.g., obstacle avoidance overriding line following).
    - Solution: Set clear priorities, such as obstacle avoidance taking precedence over line following.
19. Challenge: Timing mismatches.
    - Reason: Delays in one part of the code caused lag in other parts.
    - Solution: Replaced delay() with non-blocking code using millis() to manage timing.
20. Challenge: Debugging the combined system.
    - Reason: Testing all functionalities together made it harder to identify specific issues.
    - Solution: Tested each mode separately before integrating and added debug messages to track the robot's behavior.

5.3 Improvements

1. **Better Sensor Integration**
    - Camera Module: Add a camera so the robot can see obstacles better or even detect path for navigation.
    - Combine Sensors: Use multiple sensors together (like an MPU6050 and GPS) to improve stability and precision when recognizing gestures and navigating.
2. **Use Machine Learning for Gestures**
    - Train a Model: Use a tool (like TensorFlow Lite) to create a small machine learning model that can recognize gestures more accurately. This will make the robot understand our gestures better.
    - Adaptive Thresholding: Instead of fixed thresholds, use adaptive thresholds that can adjust based on conditions, making gesture detection more reliable.
3. **Advanced Navigation**
    - Smart Path Planning: Add advanced algorithms to help the robot find the best path to a destination while avoiding obstacles.
    - Smart Path Mapping: The robot can create a real-time map of its surroundings and navigate more easily.
4. **User-Friendly Interface**
    - Mobile App Control: Create a simple mobile app to control the robot via Bluetooth or internet. The app could help users control the car manually, switch modes, or view data like battery levels.
    - Voice Commands: Add voice control to make switching modes or directing the robot easier.
5. **Better Power Management**
    - Battery Management System: To monitor battery usage better and improve runtime.
    - Solar Charging: Use a small solar panel so the robot can charge itself while idle, especially outdoors.
6. **Hardware Upgrades**
    - Better Motors: Upgrade to more efficient motors, like stepper or brushless motors, for smoother movement.
    - Sturdier framework: Make the robot body sturdier to handle uneven surfaces.
    - Shock Absorbers: Add shock absorbers to reduce vibrations, which helps sensors give more accurate data.

7. **Data Logging and Analysis**
    - Record Data: Add a module to log sensor data while the robot is running. This helps us analyze the robot's performance and improve it.
    - Cloud Storage: Store sensor data in the cloud for long-term analysis to spot patterns and prevent failures.
8. **Test in Real Environments**
    - Test in Different Conditions: Test the robot in different environments (indoors and outdoors) and adjust the sensors for better performance under different conditions.
    - User Feedback: Gather feedback from users to improve gestures, usability, and overall performance.

6. References

- Warren, J.-D., Adams, J., & Molle, H. (2013). *Arduino Robotics*. Wiley.
- Liu, J., Zhang, X., & Zhao, L. (2015). *Gesture Recognition Using the MPU6050 Sensor*.
- Arkin, R. C. (2005). *Obstacle Avoidance for Mobile Robots: A Review*. IEEE Transactions on Robotics, 21(3), 415-426.
- Arduino. (n.d.). *How to Control a Robot Car with Arduino Using Hand Gestures* [YouTube Video].
- Smith, J. (2020). *Building Gesture Controlled Robots with Arduino and MPU6050*. Robotics Journal, 10(4), 12-19.
- Brown, M. (2019). *Ultrasonic Sensors in Robotics: A Comprehensive Guide*. TechReview Press.
- RoboTechLab. (n.d.). *Arduino Robot Car Obstacle Avoidance and Line Following Tutorial* [YouTube Video].
- RoboticsProject.com. (2021). *Gesture Controlled Robots: Challenges and Solutions*. Available here
- Miller, L. (2018). *Line Following and Obstacle Avoidance in Robotic Systems*. Robotics and Automation Handbook.
- Hackster.io. (n.d.). *Gesture Controlled Car Using Arduino and MPU6050*. Available here

- Warren, J.-D., Adams, J., & Molle, H. (2013). *Arduino Robotics*. Wiley.
- Liu, J., Zhang, X., & Zhao, L. (2015). *Gesture Recognition Using the MPU6050 Sensor*.
- Arkin, R. C. (2005). *Obstacle Avoidance for Mobile Robots: A Review*. IEEE Transactions on Robotics, 21(3), 415-426.

7. Appendix

7.1 Code Listings

TRANSMITTER

```
#include <Wire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <MPU6050.h>

// nRF24L01 setup
RF24 radio(8, 10); // CE and CSN pins for nRF24L01
const byte address[6] = "00001"; // Communication address

// MPU6050 setup
MPU6050 mpu;

// Data structure for transmission
struct Data {
int xAxis; // Scaled x-axis acceleration
int yAxis; // Scaled y-axis acceleration
int mode; // 0: Gesture Control, 1: Obstacle Avoidance, 2: Line Following
};

Data sendData;

// Button pin and mode control
const int buttonPin = 2;
int currentMode = 0; // Initial mode: Gesture Control

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

// Initialize MPU6050
mpu.initialize();
if (!mpu.testConnection()) {
Serial.println("MPU6050 connection failed!");
while (1); // Stop execution if MPU6050 is not connected
}
Serial.println("MPU6050 connected successfully.");

// Initialize nRF24L01
radio.begin();
radio.openWritingPipe(address);
radio.setPALevel(RF24_PA_MIN);
radio.setDataRate(RF24_250KBPS);
```

```
radio.stopListening();

// Configure button as input with pull-up resistor
pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
// Check button press for mode switching
static bool lastButtonState = HIGH;
bool currentButtonState = digitalRead(buttonPin);

if (lastButtonState == HIGH && currentButtonState == LOW) {
currentMode = (currentMode + 1) % 3; // Cycle through modes: 0, 1, 2
switch (currentMode) {
case 0:
Serial.println("Gesture Control Mode");
break;
case 1:
Serial.println("Obstacle Avoidance Mode");
break;
case 2:
Serial.println("Line Following Mode");
break;
}
delay(200); // Debounce delay
}
lastButtonState = currentButtonState;

// Read acceleration data from MPU6050
int16_t ax, ay, az;
mpu.getAcceleration(&ax, &ay, &az);

// Scale acceleration values by dividing by 200
sendData.xAxis = ax / 200;
sendData.yAxis = ay / 200;
sendData.mode = currentMode; // Send the current mode

// Send data over nRF24L01
radio.stopListening();
radio.write(&sendData, sizeof(Data));
radio.startListening();

// Print data and transmission status to Serial Monitor
```

```
Serial.print("X: ");
Serial.print(sendData.xAxis);
Serial.print(", Y: ");
Serial.print(sendData.yAxis);
Serial.print(", Mode: ");
switch (sendData.mode) {
case 0:
Serial.println("Gesture Control");
break;
case 1:
Serial.println("Obstacle Avoidance");
break;
case 2:
Serial.println("Line Following");
break;
}

delay(100); // Adjust delay as needed
}
```

RECEIVER

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Servo.h>

// Motor control pins
int ENA = 3;
int ENB = 9;
int MotorA1 = 4;
int MotorA2 = 5;
int MotorB1 = 6;
int MotorB2 = 7;

// Ultrasonic sensor pins
```

```
const int trigPin = 12;
const int echoPin = 13;
long duration;
int distance;

// Servo motor pin
int servoPin = 11;
Servo Myservo;

// IR sensor pins for line following
const int IR_SENSOR_RIGHT = 22; // Replace with the actual pin
const int IR_SENSOR_LEFT = 23; // Replace with the actual pin

// nRF24L01 setup
RF24 radio(8, 10); // CE and CSN pins for nRF24L01
const byte address[6] = "00001"; // Communication address

struct Data {
int xAxis;
int yAxis;
int mode; // 0: Gesture Control, 1: Obstacle Avoidance, 2: Line Following
};
Data receiveData;

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

// Initialize nRF24L01
radio.begin();
radio.openReadingPipe(0, address);
radio.setPALevel(RF24_PA_MIN);
radio.setDataRate(RF24_250KBPS);
radio.startListening();

// Motor pins setup
pinMode(ENA, OUTPUT);
pinMode(ENB, OUTPUT);
pinMode(MotorA1, OUTPUT);
pinMode(MotorA2, OUTPUT);
pinMode(MotorB1, OUTPUT);
pinMode(MotorB2, OUTPUT);

// Ultrasonic sensor setup
```

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

// IR sensor pins setup
pinMode(IR_SENSOR_RIGHT, INPUT);
pinMode(IR_SENSOR_LEFT, INPUT);

// Servo setup
Myservo.attach(servoPin);
Myservo.write(90); // Initialize servo to center position
}

void loop() {
while (radio.available()) {
radio.read(&receiveData, sizeof(Data));

// Print the received mode
Serial.print("Received Mode: ");
Serial.println(receiveData.mode);

if (receiveData.mode == 0) {
Serial.println("Gesture Control Mode");
gestureControl();
} else if (receiveData.mode == 1) {
Serial.println("Obstacle Avoidance Mode");
obstacleAvoidance();
} else if (receiveData.mode == 2) {
Serial.println("Line Following Mode");
lineFollowing();
}
}
}

void gestureControl() {
if (abs(receiveData.xAxis) < 49 && abs(receiveData.yAxis) < 49) {
stopMotors();
} else if (receiveData.xAxis < -50) { // Forward
moveForward();
} else if (receiveData.xAxis > 50) { // Backward
moveBackward();
} else if (receiveData.yAxis < -50) { // Left
turnLeft();
} else if (receiveData.yAxis > 50) { // Right
```

```
turnRight();
}
}

void obstacleAvoidance() {
// Ultrasonic sensor distance calculation
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = duration / 58.2;

Serial.print("Distance: ");
Serial.println(distance);

if (distance > 15) {
Serial.println("Path Clear - Moving Forward");
Myservo.write(90);
moveForward();
} else if (distance <= 15 && distance > 0) {
Serial.println("Obstacle Detected - Avoiding");
stopMotors();
delay(100);

Myservo.write(0); // Check left
delay(500);
Myservo.write(180); // Check right
delay(500);
Myservo.write(90); // Reset to center
delay(500);

moveBackward(); // Reverse
delay(500);
stopMotors();

turnLeft(); // Turn left to avoid
delay(500);
}
}

void lineFollowing() {
```

7.2 Simulations