EE 324L Project Report

“Wireless Gesture Controlled Vehicle using HC-05 Module”

Group Members
Sadaan Tahir - [redacted]
[redacted]
[redacted]

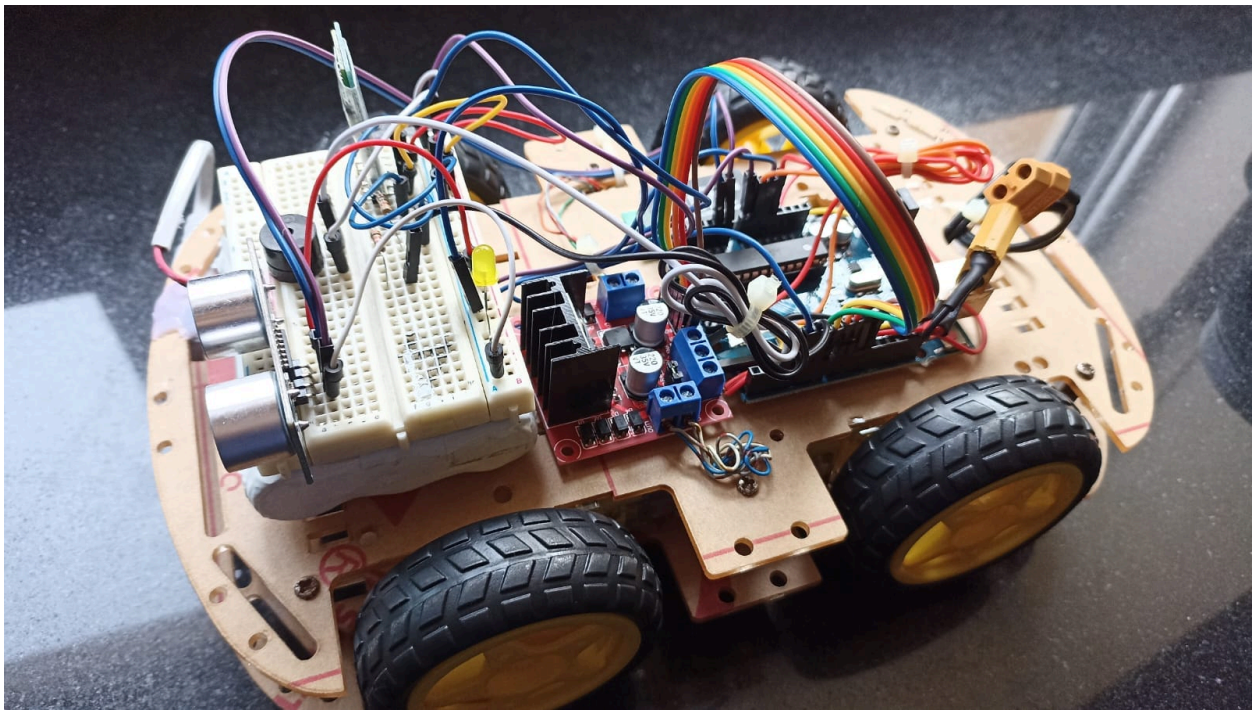


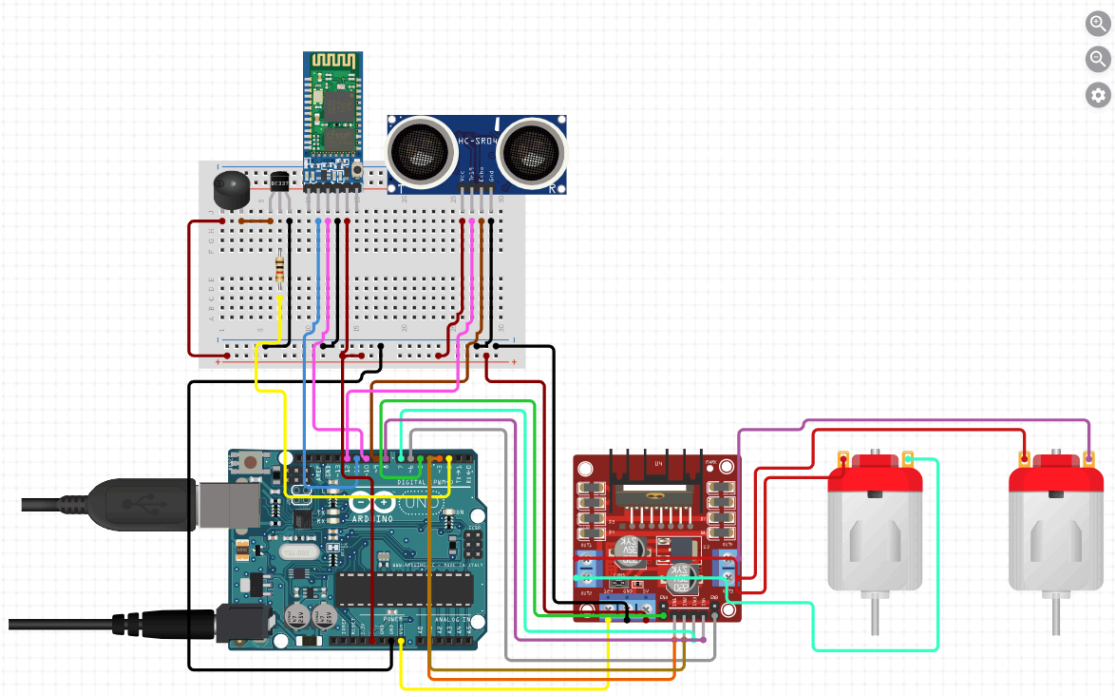
Image 1: A picture of the near finished car

Table of Contents
Introduction
Background Knowledge
Objective
Methodology
Experiment / Results
Discussions
Future Work
References: Links to relevant source material
Appendix: Includes Arduino code for both transceivers

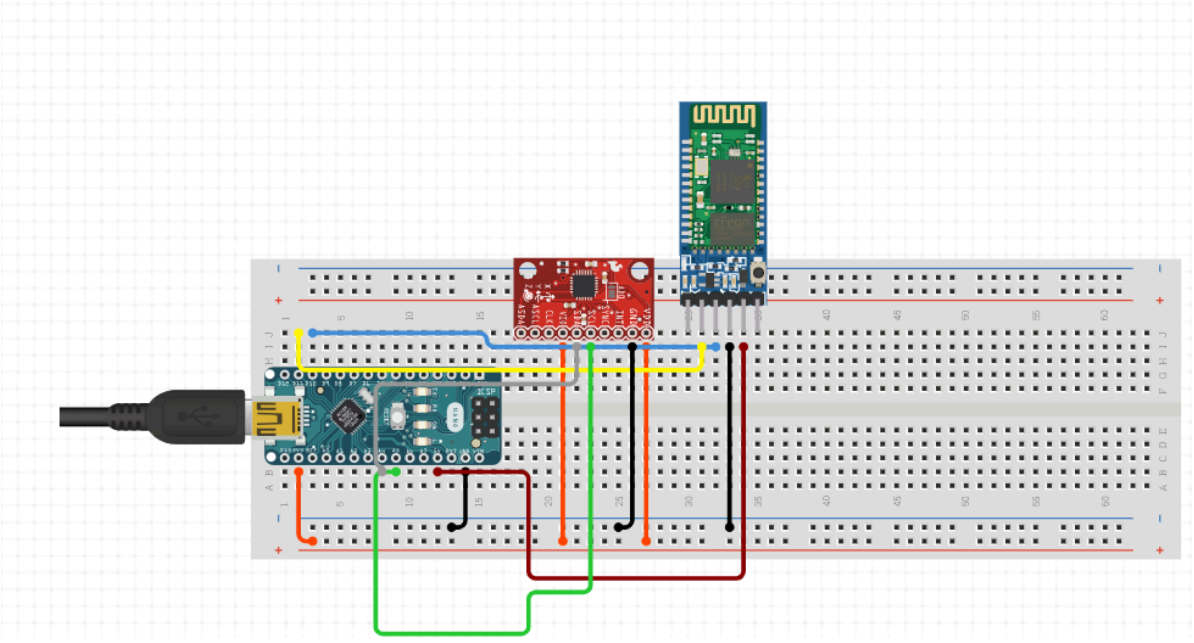
Introduction

Controlling vehicles remotely using aerial communication and through short gestures by hand can aid differently abled people in carrying out daily chores like traveling. This project hopes to build a model of such a vehicle that allows people to control vehicles more simply, different from the traditional driving.

Our gesture-controlled car project was driven by the vision of making vehicles controllable by greater ease so that they could be used by a larger demographic. Our focus was to create a simple model for this using Arduino (Uno and Nano) and Bluetooth modules that enables individuals to control vehicles through intuitive gestures. We ultimately accomplished a working prototype of a gesture-controlled car that not only showcased the capabilities of modern microcontrollers but also offered a glimpse into the future of transportation.



Circuit Diagram: Motor Driver and Receiver



Circuit Diagram: Gyroscope and Transmitter

Background Knowledge:

The project centers around the development of a gesture-controlled car, diverging from conventional control methods such as remote controllers or mobile applications. Unlike the prevalent technology, our approach leverages gestures, which are symbolic actions with specific meanings, executed by the user. The primary innovation lies in the utilization of an accelerometer to identify and interpret these gestures. The project is structured into two integral components: the transmitter circuit, employing the gyroscope for gesture recognition, and the receiver circuit, utilizing a motor driver to facilitate the motion of the car prototype. The user interfaces with the robot car through intuitive gestures, facilitated by a gesture device worn on the hand, housing an embedded sensor (a gyroscope). This sensor captures and interprets the hand movements, enabling seamless interaction with the robot. The shift from traditional input devices to gesture-based control enhances the user experience by providing a more natural and engaging means of communication with the robot.

The wireless connectivity between the gesture instrument and the robot car is achieved through Bluetooth technology. This wireless link eliminates the need for physical connections and enables a more user-friendly interaction with the robot. The user gains the ability to control the car effortlessly using accelerometer sensors integrated into a hand glove, offering a departure from conventional remote control methods.

The core of the robot's mobility lies in the implementation of a differential mechanism. This mechanism enables the rotation of the left and the right wheels (both wheels on either side) independently. This independent control facilitates precise movements, allowing the car to execute sharp turns without difficulty. The differential mechanism is pivotal in enabling the robot to rotate about its own

axis without any linear motion, adding a layer of agility and maneuverability to the vehicle.

The Gesture-Controlled Car project introduces an innovative paradigm in human-robot interaction, redefining the conventional methods of control. Through the fusion of gyrometer-based gesture recognition, wireless connectivity, and a differentially controlled mechanism, the project aims to enhance the user experience by providing a more intuitive, engaging, and precise means of controlling a robot toy car. This novel approach opens doors to a new era of user-friendly and responsive robotic devices.

Objectives:

The primary aim of this model is to develop an accessible and user-friendly human-machine interface for the control of various robotic applications, ranging from vehicles and robotic arms to toys and wheelchairs. The overarching objective is to create a cost-effective gadget that can cater to a broad audience, making advanced robotic control technology affordable and widely available.

Key Goals:

1. Convenience and Ease of Operation: The central focus is on creating a control interface that is convenient and easy for users to operate. This objective ensures that individuals with varying levels of technical expertise can seamlessly interact with and control the robot.

2. Affordability and Accessibility:The goal is to design a cost-effective solution, ensuring that the technology remains within

an affordable range for the general public. This accessibility contributes to the potential for widespread adoption across diverse user groups.

3. Versatility of Applications:The project is intended to be applicable across a range of robotic devices. Whether controlling a vehicle, a robotic arm, a toy, or a wheelchair, the interface should be adaptable and versatile to meet the diverse needs of users.

Functional Goals:

The implementation involves a systematic process to achieve the overarching objective:

1. **Hand Gesture Measurement:** Employ sensors to accurately measure and capture hand gestures. This step is crucial for interpreting user commands effectively.

2. **Gesture Identification:** Calibrate the gyroscope to decide the specific movement direction associated with each hand gesture. This ensures precision in understanding user inputs.

3. **Gesture Data Encoding and Transmission:** Encode the identified gesture data and transmit it via bluetooth. This step involves converting the gesture information into a format suitable for wireless transmission.

4. **Wireless Reception and Decoding**: Implement a corresponding bluetooth receiver to capture the transmitted data. Decode the received information to extract the original gesture data accurately.

5. **Robot Movement Control:** Utilize the decoded gesture data to control the movement of the robot. This step ensures a direct correlation between user gestures and the robot's responsive actions.

By achieving these objectives and functional goals, the project strives to introduce a versatile, user-friendly, and cost-effective human-machine interface for robotic control. The success of this undertaking holds the promise of making advanced robotics accessible to a broader audience, paving the way for diverse applications in real-world scenarios.

Methodology:

The project successfully accomplished a series of systematic steps to achieve its functional goals for implementing a versatile and user-friendly human-machine interface for robotic control. Firstly, two HC-05 Bluetooth modules were paired, configuring them in a master-slave setup to ensure secure and stable communication between the hand gesture sensor and the robot. Following this, the gyroscope was calibrated, by experimenting on different values of the gyro angles to accurately interpret a variety of hand gestures and ensure enough sensitivity while preventing hyper-sensitivity to small fluctuations in gestures. This involved capturing baseline data for each gesture and fine-tuning the gyroscope's sensitivity after several experimentations (we finally decide on a tolerance of +-3 angle scores on each axis (x and y) beyond which it starts considering the motion, (any values within this range is mapped to ‘STOP’).

In the subsequent phase, the project advanced to the construction of the robot, focusing on chassis design and integrating the LM398N motor driver for precise control over the robot's movement. A 12 V power supply system with battery level indication was established to ensure continuous and stable operation. Simultaneously, an ultrasonic sensor was integrated into the robot's design to detect obstacles in its path. The feedback mechanism incorporated a buzzer to alert the user of detected obstacles, enhancing user awareness. To complete the implementation, a Python script was developed to receive and interpret transmitted gesture data on the receiver end over the serial port

(Code attached in Appendix). Pyserial and Matplotlib's Animations were employed to display real-time information about the situation at hand as well as to print the current state of the car ("Normal operation" or "Obstacle in front") using data from the ultrasonic sensor. This comprehensive approach to implementing the project has realized a cost-effective and accessible human-machine interface for robotic control, paving the way for diverse applications in real-world scenarios.

Discussion on Methods and Results:

AT Commands (HC-05):

The utilization of AT commands to pair of two Bluetooth modules has proven to be an effective and streamlined process. AT commands served as a set of imperative instructions, accessible through a serial interface, empowering us to easily configure and control the Bluetooth modules. The initial steps involved designating one module as the master and the other as the slave, binding the master to the address of the slave, configuring their respective addresses, and specifying roles through AT commands. Activating the pairing mode was achieved seamlessly through these commands. Subsequently, the master module could initiate a connection request to the slave module, leading to a successful pairing with full-duplex communication.

Motor Driver:

We utilized the L298N dual H-bridge motor driver to control the speed and direction of two DC motors independently. The L298N is a versatile and popular choice for motor control due to its simple operation and built-in features. It consists of two H-bridges, each controlling one motor. An H-bridge is a circuit made of four transistors arranged in a way that allows current to flow to the motor in different directions.
Since we used four motors, we joined the left two motors and the right two motors and connected them to the H-bridge (which was powered using

a 12V battery). To start the motors, we set the ENABLE pin to HIGH, activating the H-bridge and allowing current to flow. We controlled the direction of the motor by setting the IN1 and IN2 (for left motors) or IN3 and IN4 (for right motors) pins HIGH or LOW, determining the direction of current flow. Setting both HIGH or both LOW stopped the motors in one direction (left or right). Turning the car left or right involved turning one set of motors on and the other off. Adjusting the speed was achieved by applying a Pulse-Width Modulation (PWM) signal to the PWM pin, controlling the average power delivered to the motor.

Ultrasonic Sensor:

The ultrasonic sensor worked by sending out an ultrasonic pulse at 40 kHz, which travels through the air. If there is an obstacle or object, the sound wave will bounce back to the sensor. The sensor then measures the time it takes for the echo to return and uses that information to calculate the distance to the object. This value was then sent over the serial port to the controller 'master' board, and subsequently graphed using Python script.

Using these methods, we were able to get a fully functional gesture controlled car working with 4-directional motion control (forward, backward, left, right, but not omni motion) as well as obstacle sensing "alarm" mechanism, with good responsiveness over a large range and a speed of around 7.2 km/h (2 m/s).

Conclusions:

This was a valuable learning experience for us. We learnt that choosing the right sensors was critical for accurate gesture recognition. We also realized that sometimes it is better to trust our own capabilities rather than a sensor's output and it is sometimes best to get a new sensor rather than debugging for hours.

For a responsive and intuitive user experience, real-time processing of gesture data was crucial. We implemented different approaches to

minimize latency between gesture detection and car movement but still delays were inevitable. Overall, it was a great learning experience.

Future Work:

1. Advanced Sensory Integration:

Visual Enhancement: Enhance the model's perception by incorporating advanced visual sensors. Consider installing a pair of cameras to give the robot a stereoscopic vision, allowing it to navigate and interact with its environment more effectively.

Integration of LiDAR or Depth Sensors: Explore the integration of LiDAR (Light Detection and Ranging) or depth sensors to provide the robot with a three-dimensional understanding of its surroundings. This would contribute to improved obstacle detection and navigation capabilities.

2. Navigation and Autonomy:

AI-based Navigation Algorithms: Integrate artificial intelligence (AI) algorithms for autonomous navigation. This would enable the robot to make intelligent decisions based on the data gathered from sensors, optimizing its path planning in dynamic environments.

3.Energy-Efficient Power Systems:

Develop and implement advanced power management systems to optimize energy consumption. Explore the use of energy-efficient components and technologies to extend the operational life of the robot.

Energy Harvesting Solutions:Investigate energy harvesting solutions, such as regenerative braking or piezoelectric systems, to capture and utilize ambient energy for powering the robot. This would further reduce the dependence on traditional power sources.

4. Communication and Remote Operation:

Extend the robot's capabilities for remote operation by incorporating telepresence features. This could include improved two-way communication, allowing users to interact with the environment through the robot in real-time.

Upgrade the wireless communication system to support higher bandwidth and lower latency. This improvement ensures a more robust connection for live streaming and remote control, especially in areas with challenging communication conditions.

5. Environmental Adaptability:

Enhance the model's robustness by incorporating weatherproofing measures. This would enable the robot to operate in various environmental conditions, making it suitable for outdoor applications.

The proposed future work aims to elevate the model's capabilities in diverse domains, ranging from advanced perception and navigation to energy efficiency and communication. By exploring these avenues, the gesture-controlled robot car can evolve into a highly adaptable and sophisticated robotic platform with expanded utility across military, exploration, medical, and various other fields.

References:

https://www.researchgate.net/publication/354777772_Gesture_Controlled_Robot_using_Accelerometer

Appendix

The code for Arduino Uno used in the bluetooth controlled car:

```
#include <SoftwareSerial.h>

// Motor control pins
int ENA = 11; // PWM signal for speed control of MOTORS A (left)
int ENB = 10; // PWM signal for speed control of MOTORS B (right)

// Ultrasonic sensor pins
const int ECHO_PIN = A0; // Echo pin A0
const int TRIG_PIN = A1; // Trig pin A1

// Buzzer control
int buzzerPin = 3; // Buzzer PWM pin
int beepFrequency = 2000; // Adjust the frequency based on your preference

// Bluetooth communication
SoftwareSerial EEBlue(4, 5); // RX | TX

// Distance measurement variables
unsigned long duration;
float distance = 0;

// Time tracking for sensor updates
unsigned long time;

void setup() {
  Serial.begin(9600);
  EEBlue.begin(9600); // Baud Rate for command Mode

  // Motor control pins
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);

  // Buzzer pin
  pinMode(buzzerPin, OUTPUT);

  // Ultrasonic sensor pins
  pinMode(ECHO_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);

  // Extra LEDs
  pinMode(2, OUTPUT);
```

```
  pinMode(12, OUTPUT);

  // Initialize time for sensor updates
  time = millis();
}

void loop() {
  if (millis() - time >= 1000 && !EEBlue.available()) {
    // Ultrasonic sensor measurement
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    duration = pulseIn(ECHO_PIN, HIGH);
    distance = (duration * 0.034 / 2.0);
    EEBlue.write(distance);

    // Activate the buzzer if the distance is less than 15m
    if (distance < 15) {
        analogWrite(buzzerPin, 255); // Adjust the PWM value based on the desired
beep intensity
    } else {
      analogWrite(buzzerPin, 0); // Turn off the buzzer
    }

    time = millis(); // Update the time for the next sensor update
  }

  if (EEBlue.available()) {
    Serial.println(EEBlue.read());
  }

  // Motor control based on Bluetooth commands
  int command = EEBlue.read();

  if (command == 2) { // Left
    digitalWrite(2, LOW);
    digitalWrite(12, LOW);
    analogWrite(ENA, 150);
    analogWrite(ENB, 150);
    digitalWrite(9, HIGH);
    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
    digitalWrite(6, HIGH);
  }

  if (command == 3) { // Right
    digitalWrite(2, LOW);
```

```
    digitalWrite(12, LOW);
    analogWrite(ENA, 150);
    analogWrite(ENB, 150);
    digitalWrite(9, LOW);
    digitalWrite(7, HIGH);
    digitalWrite(8, HIGH);
    digitalWrite(6, LOW);
  }

  if (command == 1) { // Stop
    digitalWrite(2, LOW);
    digitalWrite(12, LOW);
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
    digitalWrite(9, LOW);
    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
    digitalWrite(6, LOW);
  }

  if (command == 4) { // Forward
    digitalWrite(2, HIGH);
    digitalWrite(12, HIGH);
    analogWrite(ENA, 255);
    analogWrite(ENB, 255);
    digitalWrite(9, HIGH);
    digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
    digitalWrite(6, LOW);
  }

  if (command == 5) { // Backward
    digitalWrite(2, LOW);
    digitalWrite(12, LOW);
    analogWrite(ENA, 255);
    analogWrite(ENB, 255);
    digitalWrite(9, LOW);
    digitalWrite(7, LOW);
    digitalWrite(8, HIGH);
    digitalWrite(6, HIGH);
  }
}
```

The code for the Arduino Nano used in the gyro powered control glove:

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

// Initialize MPU6050 sensor and Bluetooth serial communication
Adafruit_MPU6050 mpu;
SoftwareSerial BTSerial(2, 3);

void setup() {
  // Initialize serial communication for debugging and Bluetooth
  Serial.begin(9600);
  BTSerial.begin(9600);

  // Initialize MPU6050 sensor
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }

  // Configure MPU6050 settings
  mpu.setAccelerometerRange(MPU6050_RANGE_16_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  delay(100);  // Wait for sensor initialization
}

void loop() {
  // Read accelerometer and gyroscope data from MPU6050 sensor
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Check if data is available from Bluetooth
  if (BTSerial.available()) {
    Serial.write(BTSerial.read());
  } else if (!BTSerial.available()) {
      // Process accelerometer data and send corresponding signals via
Bluetooth
    if (abs(a.acceleration.x) <= 3 && abs(a.acceleration.y) <= 3) {
      BTSerial.write(1);  // Signal for no movement
    } else if (a.acceleration.x < -3) {
```

```
      BTSerial.write(4);  // Signal for left tilt
    } else if (a.acceleration.x > 3) {
      BTSerial.write(5);  // Signal for right tilt
    } else if (a.acceleration.y < -3) {
      BTSerial.write(2);  // Signal for backward tilt
    } else if (a.acceleration.y > 3) {
      BTSerial.write(3);  // Signal for forward tilt
    }
  }

  // Optional: Print additional sensor data for debugging
  // BTSerial.print("Temperature:");
  // BTSerial.print(temp.temperature);
  // Serial.println("Temperature:");
  // Serial.println(temp.temperature);
  // BTSerial.print("\tx-acceleration:");
  // BTSerial.print(a.acceleration.x);
  // BTSerial.print("\ty-acceleration:");
  // BTSerial.print(a.acceleration.y);
  // BTSerial.print("\tz-acceleration:");
  // BTSerial.print(a.acceleration.z);
  // BTSerial.print("\tx-gyro:");
  // BTSerial.print(g.gyro.x);
  // BTSerial.print("\ty-gyro:");
  // BTSerial.print(g.gyro.y);
  // BTSerial.print("\tz-gyro:");
  // BTSerial.println(g.gyro.z);

  // Optional: Delay to control loop frequency
  // delay(10);
}
```

Plotting Python Script:

```
import time
import serial
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def animate(i, dataList, ser, text):
    ser.write(b'g')
# Transmit the char 'g' to receive the Arduino data point
    arduinoData_string = ser.readline().decode('ascii')
# Decode received Arduino data as a formatted string
```

```
    try:
        arduinoData_float = float(arduinoData_string)   # Convert to float
        dataList.append(arduinoData_float)
 # Add to the list holding the fixed number of points to animate
        if arduinoData_float < 15.0:
            text.set_text("Too close to an obstacle, buzzer sounded, STOP")
        else:
            text.set_text("Normal operation")
    except:
# Pass if data point is bad
        pass

    dataList = dataList[-50:]
# Fix the list size so that the animation plot 'window' is x number of points
    ax.clear()                                          # Clear last data frame
    ax.plot(dataList)                                   # Plot new data frame
    ax.set_ylim([0, 200])
# Set Y axis limit of the plot
    ax.set_title("Distance in front of Car")            # Set title of the figure
    ax.set_ylabel("Distance")                           # Set title of the y-axis
    ax.set_xlabel("Time axis")
      ax.text(0.5, 0.95, text.get_text(), transform=ax.transAxes, ha='center',
va='center', bbox=dict(facecolor='white', alpha=0.5))
dataList = []
# Create an empty list variable for later use
fig = plt.figure()
# Create Matplotlib plots, fig is the 'higher level' plot window
ax = fig.add_subplot(111)
# Add subplot to the main fig window
text = ax.text(0.5, 0.95, '', transform=ax.transAxes, ha='center', va='center',
bbox=dict(facecolor='white', alpha=0.5))

ser = serial.Serial("COM9", 9600)
# Establish a Serial object with COM port and BAUD rate to match Arduino
Port/rate
time.sleep(2)
# Time delay for Arduino Serial initialization

# Matplotlib Animation Function that takes care of real-time plot.
```

```
# Note that 'fargs' parameter is where we pass in our dataList and Serial object.
ani = animation.FuncAnimation(fig, animate, frames=100, fargs=(dataList, ser,
text), interval=100)

plt.show()
 # Keep Matplotlib plot persistent on screen until it is closed
ser.close()
 # Close Serial connection when the plot is closed
```