

**Summer Projects  
Science and Technology Council  
IIT Kanpur**

*AR Gesture Controlled Car*

*Documentation*

Date of Submission : 16/07/2024

# Contents

1. Acknowledgements
2. Introduction
  - 2.1. Aim
  - 2.2. Problem Statement
3. Plan of Action
4. Theory
  - 4.1. IMU Sensor - MPU6050
  - 4.2. NODEMCU Micro-controller board
  - 4.3. L298 Motor Driver Module
  - 4.4. DC Motors
5. Implementation of IMU controlled Car
  - 5.1. Component Assembly
  - 5.2. Circuit Engineering
  - 5.3. Codes for Sender and Receiver Modules
6. Implementation of Augmented Reality (AR)
  - 6.1. Marker-Based AR
  - 6.2. Markerless AR
  - 6.3. Code used to implement marker-based AR
7. Conclusion
  - 7.1. Key Achievements
8. Future Aspects
9. References

# 1.Acknowledgements:

We would like to express our heartfelt gratitude to the Electronics Club and the Science & Technology Council at IIT Kanpur for providing us with the opportunity to work on this summer project . We are deeply thankful to our mentors, Devansh Bansal , Devansh Agrawal , Dhruv Mittal and Kanika, as well as the Coordinators of the Electronics Club. Their invaluable guidance and unwavering support were instrumental in helping us complete this project on time.

## 2.Introduction

### 2.1 AIM

The purpose of this project is implementation of hand gesture recognition technology for intuitive control of the race car using IMU sensor-MPU 6050, along with integration of a first-person view (FPV) camera within the race car, providing an immersive perspective that transports the user directly to the car's location via a headset. We also got a chance to try AR app development capable of overlaying obstacles and checkpoints onto the FPV camera feed, effectively transforming any ordinary floor into a dynamic race track environment.

### 2.2 PROBLEM STATEMENT

The main objectives in this project were to implement gesture control and AR app development which can be broken down into the following :

- Calculation of roll and pitch and accordingly powering the motors to move in some specific direction
- Using the camera feed and overlaying obstacles over it using marker and markerless AR

## 3.Plan of action

The approach for the project was simple since it was divided in two parts : hardware which included the gesture control which was done by us in the first part of the project duration , involving use of imu sensors, and motor driver to direct the car's movement , and the second Integration of a camera situated within the car with the application in the phone using CV which enables us to view the car's usual FPV (First Person View) as a thrilling race track taking the car into AR. We carried out the AR integration for the rest of the project duration, finally enabling us to complete the project.

# 4. THEORY

## 4.1 IMU Sensor - MPU6050

In this project we have used MPU6050 which is an Inertial Measurement Unit (IMU) sensor module that combines a 3-axis gyroscope and a 3-axis accelerometer on a single chip. It is widely used in various applications due to its compact size, cost-effectiveness, and reliable performance. Here are some key features and details about the MPU6050:



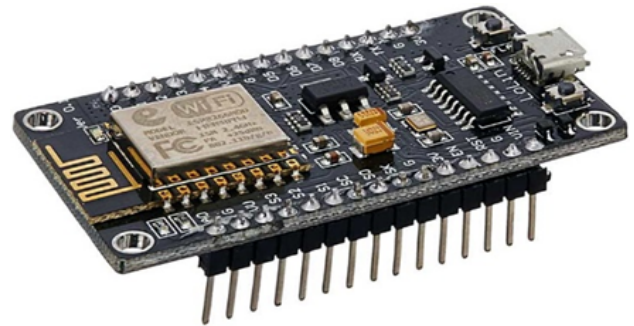
1. 6 Degrees of Freedom (DoF): It provides six degrees of freedom by combining a 3-axis accelerometer and a 3-axis gyroscope, which allows it to measure both linear acceleration and angular velocity.
2. Digital Motion Processor (DMP): The MPU6050 includes a built-in Digital Motion Processor, which can process complex motion fusion algorithms, reducing the computational load on the host microcontroller.
3. I2C Interface: It communicates with microcontrollers using the I2C protocol, which is simple and widely supported.
4. Temperature Sensor: It includes an internal temperature sensor for more accurate readings and compensation.
5. Applications: The MPU6050 is commonly used in drones, robotics, wearable devices, gaming controllers, and smartphones for motion detection, gesture recognition, and orientation sensing.
6. Low Power Consumption: It is designed to be energy-efficient, making it suitable for battery-powered applications.

7. Motion Sensing Capabilities: The sensor can detect motion and orientation changes, making it useful for applications requiring precise motion tracking and control.

The MPU6050 is valued for its versatility and ease of integration into various electronic projects, providing reliable motion sensing capabilities in a small package.

## 4.2 ESP8266 Node MCU

The Node MCU is a microcontroller capable of communicating wirelessly with other devices. The Node MCU ESP8266 development board comes with the ESP-12E module containing the ESP8266 chip which is a Tensilica Xtensa 32-bit LX106 RISC microprocessor. This



microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. It has high processing power with in-built Wi-Fi / Bluetooth. It also has a Deep Sleep Operating feature that makes it ideal for IoT projects. Since an IoT device has to operate 24 hours a day and 7 days a week, this Deep Sleep feature wakes the device whenever there is a command, therefore helps save power consumption. NodeMCU can be powered using a Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.

Table 2.1: NodeMCU Development Board Pinout Configuration

Pin Category	Name	Description
Power	Micro-USB, 3.3V, GND, Vin	<p>Micro-USB: NodeMCU can be powered through the USB port</p> <p>3.3V: Regulated 3.3V can be supplied to this pin to power the board</p> <p>GND: Ground pins</p> <p>Vin: External Power Supply</p>
Control Pins	EN, RST	The pin and the button resets the microcontroller
Analog Pin	A0	Used to measure analog voltage in the range of 0-3.3V
GPIO Pins	GPIO1 to GPIO16	NodeMCU has 16 general purpose input-output pins on its board
SPI Pins	SD1, CMD, SD0, CLK	NodeMCU has four pins available for SPI communication.

UART Pins	TXD0, RXD0, TXD2, RXD2	NodeMCU has two UART interfaces, UART0 (RXD0 & TXD0) and UART1 (RXD1 & TXD1). UART1 is used to upload the firmware/program.
I2C Pins		NodeMCU has I2C functionality support but due to the internal functionality of these pins, you have to find which pin is I2C.

## 4.3 L298N Motor Driver

The L298N Motor Driver module consists of an L298N Motor Driver IC, 78M05 Voltage Regulator, resistors, capacitor, Power LED, 5V jumper in an integrated circuit 78M05 Voltage regulator will be enabled only when the jumper is placed. When the power supply is less than or equal to 12V, then the internal circuitry will be powered by the voltage regulator and the 5V pin can be used as an output pin to power the microcontroller. The jumper should not be placed when the power supply is greater than 12V and separate 5V should be given through 5V terminal to power the internal circuitry. ENA and ENB pins are speed control pins for Motor A and Motor B while IN1& IN2 and IN3 & IN4 are direction control pins for Motor A and Motor B.



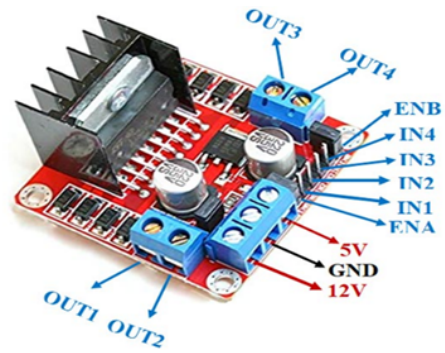


Figure 2.2: A L298N Motor Driver and its pinouts

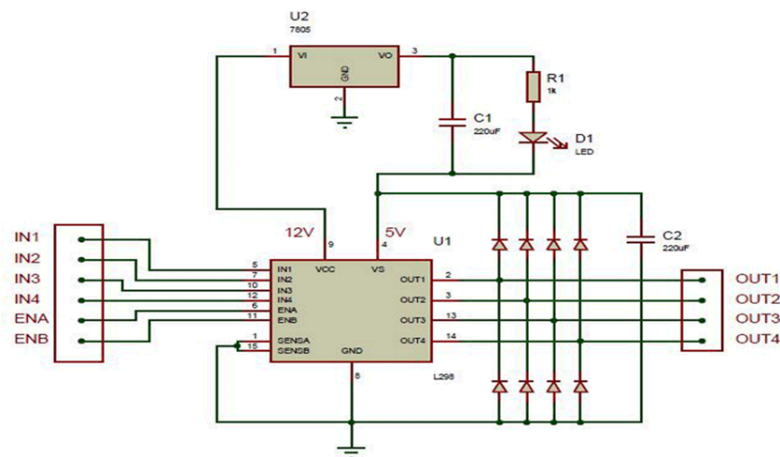


Figure 2.3: An L298N Module Pinout Configuration

Table 2.2: Details of the pins of L298N

Pin Name	Description
IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B

ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A
OUT3 & OUT4	Output pins of Motor B
12V	12V input from DC power Source
5V	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin

## 4.4 DC Motors

A machine that converts DC power into mechanical power is known as a DC motor. Its operation is based on Faraday's principle which states that when a current carrying conductor is placed in a magnetic field, the conductor experiences a mechanical force. DC motors have a revolving armature winding but non-revolving armature magnetic field and a stationary field winding or permanent magnet. Different connections of the field and armature winding provide different speed/torque regulation features. The speed of a DC motor can be controlled by changing the voltage applied to the armature or by changing the field current.



# 5. Implementation of IMU Controlled Car

## 5.1 Components Assembly

### 5.1.1 Chassis, motors and wheels

There are four wheels and four motors. Each of the wheels and motors are connected in pairs and attached to the chassis.

### 5.1.2 Powering the motors

Connecting wires are soldered to the positive and negative part of the motors and connected to the outputs of the motor driver. The two left wheels are connected together in positive-to-positive and negative-to-negative manner. The same form of connection is done for the two right wheels. In total, there is now one positive and one negative from the left wheels. This is connected to one section of the L298N motor controller. Similarly, there is one positive and one negative wire from the right wheels. This is connected to another section of the L298N motor controller.

### 5.1.3 Powering the L298N motors driver (or controller)

The motor controller is powered by a DC-DC Buck converter. This Buck converter is an adjustable DC-DC step down module. It was set to 9V.

### 5.1.4 Powering the NodeMCU

The L298N motor controller has a 5V output source. Since the NodeMCU operates on 5V, the NodeMCU was connected to the 5V output source on the L298N motor controller.

### 5.1.5 Remotely controlling the L298N motors driver (or controller)

The movement Data is taken from an Inertial Measurement Unit (IMU), which is Connected to another NodeMCU. This NodeMCU then relays this movement data to the other NodeMCU connected to the motor driver module.

## 5.2 CIRCUIT ENGINEERING

In this section a circuit will be modeled to enable the motor driver to receive instructions from the node MCU to control the motors to move in the specified directions.

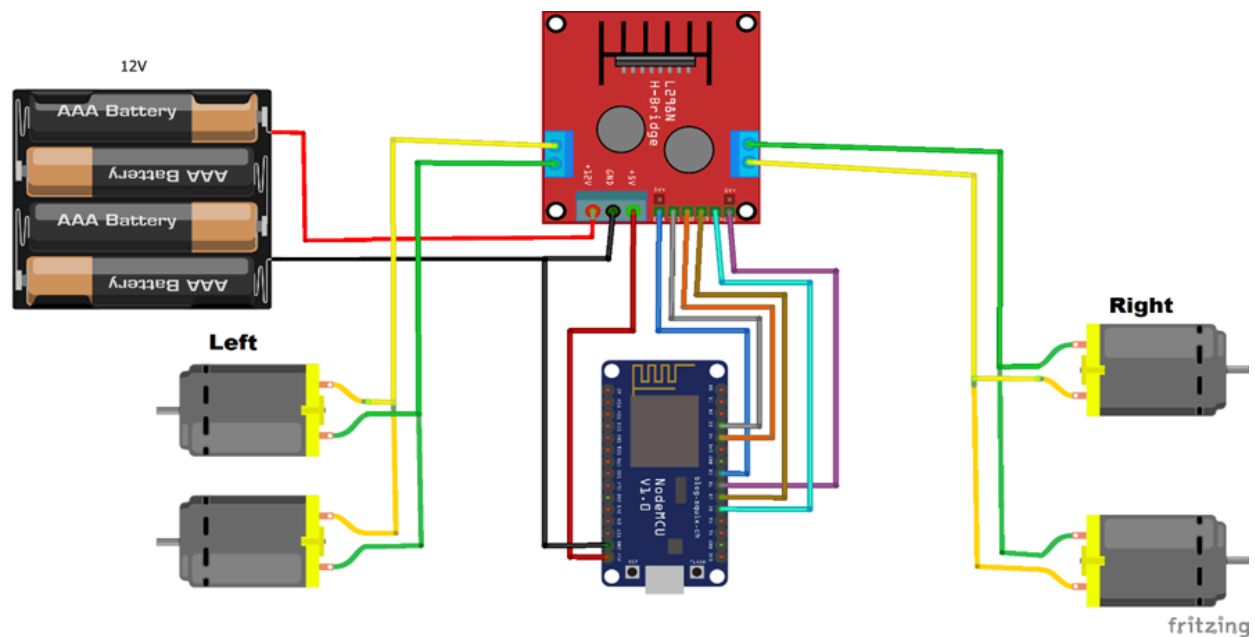


Figure 4.1: Circuit Diagram

### 5.2.1 Description of Circuit

The circuit above has been designed to function according to the aim of this project which is to wirelessly control the movement of a robotic car with the aid of gestures from the gyroscope sensor of an IMU Sensor. In the circuit above the motors are connected to the motor driver, the positive and negative of the two left motors are connected to the output 1 and 2 of the motor driver respectively, the positive and negative of the two right motors are connected to the output 3 and 4 of the motor driver respectively. The motor driver is connected to the NodeMCU, IN1 and IN2 of the motor driver are connected to pin D1 and D2 of the node MCU to control the left motors, IN3 and IN4 of the motor driver are connected to pin D3 and D4 of the NodeMCU to

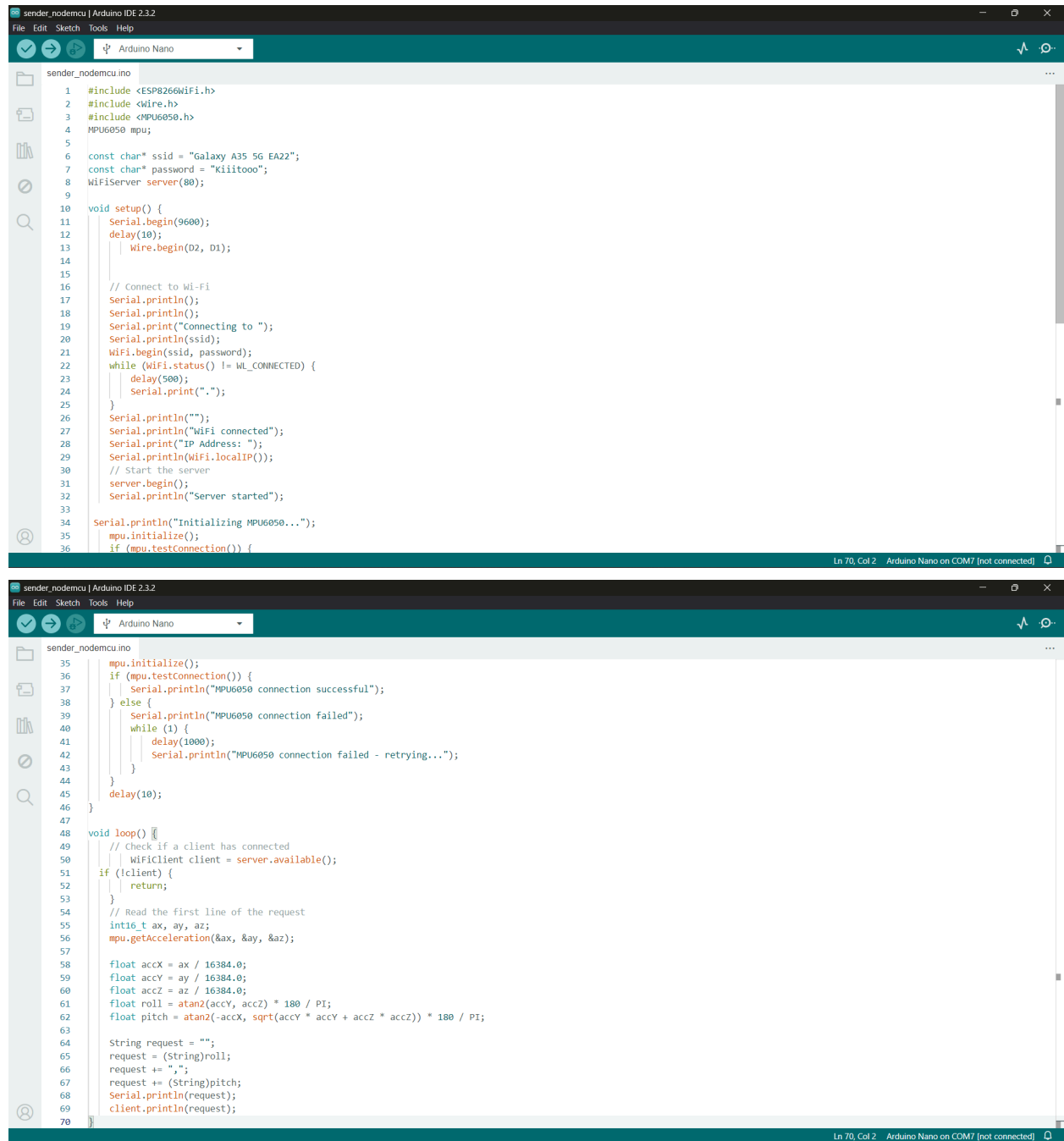
control the right motors, ENA and ENB of the motor driver are connected to 5V supply. The Vin pin of the node MCU is connected to the 5V outlet of the motor driver, the ground pin of the node MCU is connected to the ground outlet of the motor driver and negative of the battery and 12V of the motor driver to the positive of the battery.

## 5.3 Codes for Sender and Receiver Modules

The objective of this section is to write and upload the code for programming the node MCU to enable it to receive instructions from the android device, interpret it and control the motor driver according to the interpretation which is to move the motors in the specified direction. An Arduino IDE with board esp8266 will be used to program the node MCU.

When the code has been written, it will be compiled to check for errors, if there are no errors, the node MCU is then connected to a computer via a USB cord, once the device is detected, the appropriate port is selected then the code is uploaded. After the code has been successfully uploaded, the node MCU will automatically connect to the mobile hotspot of the android device via its WIFI module to receive instructions, the serial monitor on the Arduino IDE is opened to confirm the successful connection of the node MCU to the android device's hotspot and an IP address is also displayed which will be put into the application on the android device to interface it with the node MCU.

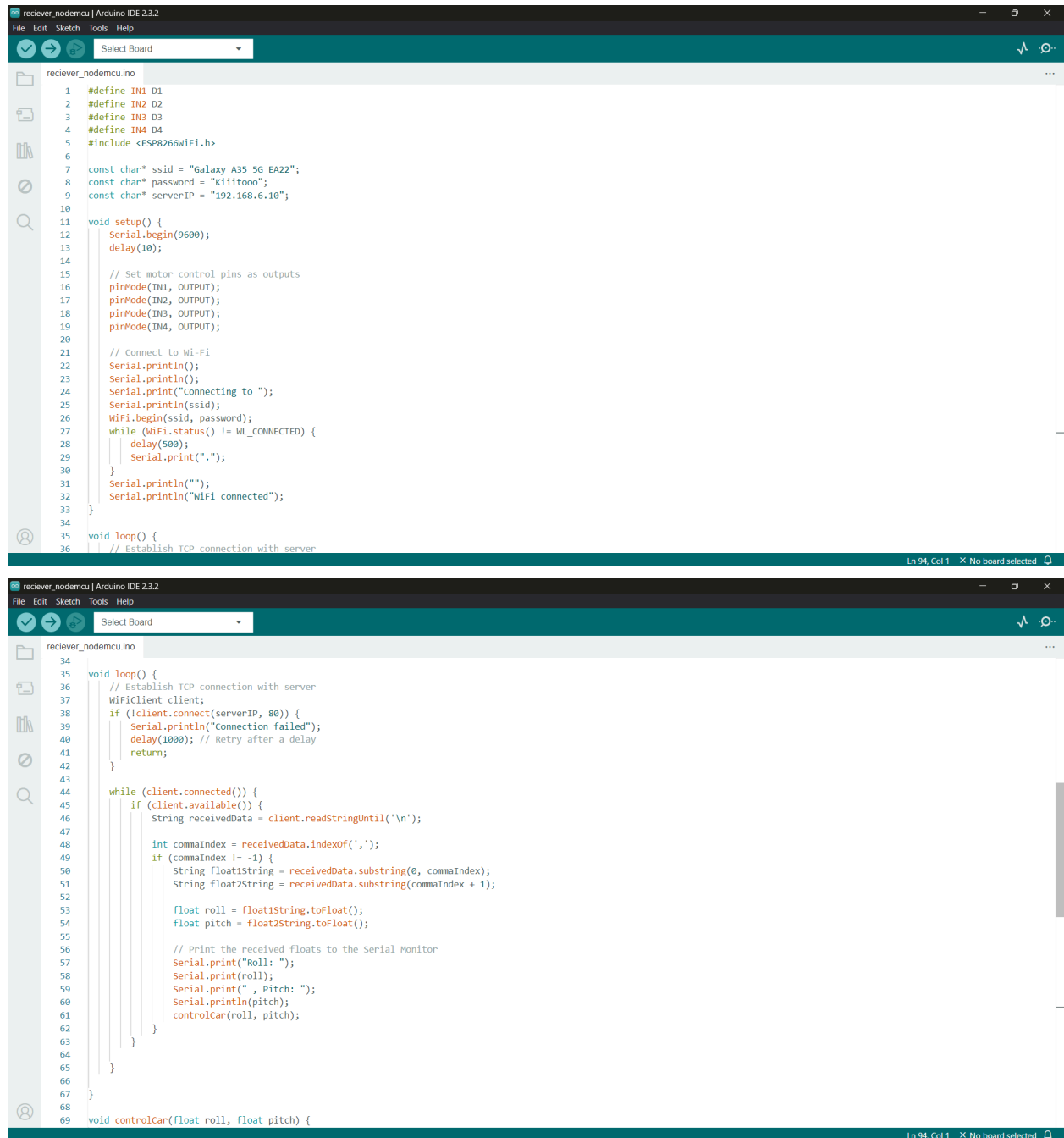
## 5.3.1 Code For Sender NodeMCU



```
sender_nodemcu | Arduino IDE 2.3.2
File Edit Sketch Tools Help

sender_nodemcu.ino
1 #include <ESP8266WiFi.h>
2 #include <Wire.h>
3 #include <MPU6050.h>
4 MPU6050 mpu;
5
6 const char* ssid = "Galaxy A35 5G EA22";
7 const char* password = "Kiiitooo";
8 WiFiServer server(80);
9
10 void setup() {
11   Serial.begin(9600);
12   delay(10);
13   Wire.begin(D2, D1);
14
15   // Connect to Wi-Fi
16   Serial.println();
17   Serial.println();
18   Serial.print("Connecting to ");
19   Serial.println(ssid);
20   WiFi.begin(ssid, password);
21   while (WiFi.status() != WL_CONNECTED) {
22     delay(500);
23     Serial.print(".");
24   }
25   Serial.println("");
26   Serial.println("WiFi connected");
27   Serial.print("IP Address: ");
28   Serial.println(WiFi.localIP());
29   // Start the server
30   server.begin();
31   Serial.println("Server started");
32
33   Serial.println("Initializing MPU6050...");
34   mpu.initialize();
35   if (mpu.testConnection()) {
36
37     mpu.initialize();
38     if (mpu.testConnection()) {
39       Serial.println("MPU6050 connection successful");
40     } else {
41       Serial.println("MPU6050 connection failed");
42       while (1) {
43         delay(1000);
44         Serial.println("MPU6050 connection failed - retrying...");
45       }
46     }
47   }
48   delay(10);
49
50 void loop() {
51   // Check if a client has connected
52   WiFiClient client = server.available();
53   if (!client) {
54     return;
55   }
56   // Read the first line of the request
57   int16_t ax, ay, az;
58   mpu.getAcceleration(&ax, &ay, &az);
59
60   float accX = ax / 16384.0;
61   float accY = ay / 16384.0;
62   float accZ = az / 16384.0;
63   float roll = atan2(accY, accZ) * 180 / PI;
64   float pitch = atan2(-accX, sqrt(accY * accY + accZ * accZ)) * 180 / PI;
65
66   String request = "";
67   request = (String)roll;
68   request += ",";
69   request += (String)pitch;
70   Serial.println(request);
71   client.println(request);
72 }
```

## 5.3.2 Code for Receiver NodeMCU



```
reciever_nodemcu.ino
1  #define IN1 D1
2  #define IN2 D2
3  #define IN3 D3
4  #define IN4 D4
5  #include <ESP8266WiFi.h>
6
7  const char* ssid = "Galaxy A35 5G EA22";
8  const char* password = "Kiiiitooo";
9  const char* serverIP = "192.168.6.10";
10
11 void setup() {
12     Serial.begin(9600);
13     delay(10);
14
15     // Set motor control pins as outputs
16     pinMode(IN1, OUTPUT);
17     pinMode(IN2, OUTPUT);
18     pinMode(IN3, OUTPUT);
19     pinMode(IN4, OUTPUT);
20
21     // Connect to Wi-Fi
22     Serial.println();
23     Serial.println();
24     Serial.print("Connecting to ");
25     Serial.println(ssid);
26     WiFi.begin(ssid, password);
27     while (WiFi.status() != WL_CONNECTED) {
28         delay(500);
29         Serial.print(".");
30     }
31     Serial.println("");
32     Serial.println("WiFi connected");
33 }
34
35 void loop() {
36     // Establish TCP connection with server
37     WiFiClient client;
38     if (!client.connect(serverIP, 80)) {
39         Serial.println("Connection failed");
40         delay(1000); // Retry after a delay
41         return;
42     }
43
44     while (client.connected()) {
45         if (client.available()) {
46             String receivedData = client.readStringUntil('\n');
47
48             int commaIndex = receivedData.indexOf(',');
49             if (commaIndex != -1) {
50                 String float1String = receivedData.substring(0, commaIndex);
51                 String float2String = receivedData.substring(commaIndex + 1);
52
53                 float roll = float1String.toFloat();
54                 float pitch = float2String.toFloat();
55
56                 // Print the received floats to the Serial Monitor
57                 Serial.print("Roll: ");
58                 Serial.print(roll);
59                 Serial.print(" , Pitch: ");
60                 Serial.println(pitch);
61                 controlCar(roll, pitch);
62             }
63         }
64     }
65 }
66
67
68
69 void controlCar(float roll, float pitch) {
```

```
reciever_nodemcu | Arduino IDE 2.3.2
File Edit Sketch Tools Help

Select Board

reciever_nodemcu.ino
68
69 void controlCar(float roll, float pitch) {
70   // Forward/Backward control based on Y-axis acceleration
71   if (pitch > 20.0) { // Threshold value for forward motion
72     moveForward();
73   } else if (pitch < -20.0) { // Threshold value for backward motion
74     moveBackward();
75   } else {
76     stopMoving();
77   }
78
79   // Left/Right control based on X-axis acceleration
80   if (roll > 20.0) { // Threshold value for left turn
81     turnRight();
82   } else if (roll < -20.0) { // Threshold value for right turn
83     turnLeft();
84   }
85 }
86
87 void moveForward() {
88   digitalWrite(IN1, HIGH);
89   digitalWrite(IN2, LOW);
90   digitalWrite(IN3, HIGH);
91   digitalWrite(IN4, LOW);
92   Serial.println("Moving Forward");
93 }
94
95 void moveBackward() {
96   digitalWrite(IN1, LOW);
97   digitalWrite(IN2, HIGH);
98   digitalWrite(IN3, LOW);
99   digitalWrite(IN4, HIGH);
100   Serial.println("Moving Backward");
101 }
102
103 void turnLeft() {
```

```
reciever_nodemcu | Arduino IDE 2.3.2
File Edit Sketch Tools Help

Select Board

reciever_nodemcu.ino
90   digitalWrite(IN3, HIGH);
91   digitalWrite(IN4, LOW);
92   Serial.println("Moving Forward");
93 }
94
95 void moveBackward() {
96   digitalWrite(IN1, LOW);
97   digitalWrite(IN2, HIGH);
98   digitalWrite(IN3, LOW);
99   digitalWrite(IN4, HIGH);
100   Serial.println("Moving Backward");
101 }
102
103 void turnLeft() {
104   digitalWrite(IN1, LOW);
105   digitalWrite(IN2, HIGH);
106   digitalWrite(IN3, HIGH);
107   digitalWrite(IN4, LOW);
108   Serial.println("Turning Left");
109 }
110
111 void turnRight() {
112   digitalWrite(IN1, HIGH);
113   digitalWrite(IN2, LOW);
114   digitalWrite(IN3, LOW);
115   digitalWrite(IN4, HIGH);
116   Serial.println("Turning Right");
117 }
118
119 void stopMoving() {
120   digitalWrite(IN1, LOW);
121   digitalWrite(IN2, LOW);
122   digitalWrite(IN3, LOW);
123   digitalWrite(IN4, LOW);
124   Serial.println("Stop Moving");
125 }
```



## **6. Augmented Reality (AR)**

### **Introduction**

Augmented Reality (AR) is a technology that superimposes computer-generated enhancements atop an existing reality to make it more meaningful through the ability to interact with it. This technology blends digital components into the real world in such a way that they enhance one another but can also be easily distinguished apart. AR is utilized in various fields, from gaming and entertainment to education and healthcare. It allows users to see and experience their surroundings in an enriched manner by overlaying visual, auditory, and sensory information onto the physical world. For instance, in education, AR can bring textbooks to life with interactive 3D models, while in healthcare, it can assist surgeons by providing real-time data during operations. The technology relies on hardware like smartphones, tablets, and AR glasses, coupled with software that includes apps and platforms designed to deliver these augmented experiences.

We have developed an AR application capable of overlaying obstacles and checkpoints onto the FPV camera feed, effectively transforming any ordinary floor into a dynamic race track environment.

There are two approaches to implementing Augmented Reality, which are described below.

### **6.1 Marker - Based AR**

Marker-based augmented reality (AR) is a type of AR technology that uses a physical marker or object as a reference point to overlay virtual content onto the real world. The marker can be an image, symbol, QR code, or any other recognizable object that the AR software can detect and track.

When the AR software recognizes the marker, it uses its position and orientation to determine where to place the virtual content in the real world. This creates the illusion that the virtual content is interacting with the real world and appearing to be part of it.

### 6.1.1 Steps for Marker-Based AR

1. **Identify the objective:** Define the objective of the AR application and identify the target audience.
2. **Create AR content:** Create 3D models, animations, videos, images, and sounds that will be used in the AR application.
3. **Choose an AR development tool:** Choose an AR development tool that is suitable for your project. Popular AR development tools include Vuforia, ARToolKit, and Wikitude.
4. **Design the marker:** Design the marker that will be used to trigger the AR experience. The marker can be a printed image, a QR code, or any other recognizable object.
5. **Develop the AR app:** Use the AR development tool to create the AR app. This includes integrating the AR content with the marker and the AR software, designing the user interface and user experience, and programming the AR features.
6. **Test the AR app:** Thoroughly test the AR app to ensure that it functions correctly and meets the desired objectives.
7. **Deploy the AR app:** Once the AR app is fully tested and refined, it can be deployed to the target audience.
8. **Maintenance and updates:** AR applications require ongoing maintenance and updates to ensure that they remain functional and relevant. This includes fixing bugs, updating the AR content, and integrating new features as needed.

### 6.1.2 Benefits of Marker-Based AR

1. **Accuracy:** Marker-based AR is very accurate because it relies on a physical marker to provide a reference point for the AR software. This means that the virtual content is precisely placed in the real world, creating a seamless and immersive experience.
2. **Interactivity:** Marker-based AR allows users to interact with virtual content in real-time, making the experience more engaging and interactive. This can be particularly useful for educational or training applications where hands-on learning is important.
3. **Ease of use:** Marker-based AR is relatively easy to use because the marker provides a clear reference point for the AR software. Users do not need to calibrate or configure the app, making it more accessible to a wider audience.

4. **Flexibility:** Marker-based AR can be used in a variety of settings and applications, including advertising, entertainment, education, and training. It can be used to display product information, provide additional context for exhibits, or offer immersive learning experiences.
5. **Cost-effectiveness:** Marker-based AR is generally more cost-effective than other types of AR because it does not require expensive hardware or sensors. It can be used with a smartphone or tablet, making it accessible to a wide range of users.

### 6.1.3 Limitations of Marker-based AR

1. **Limited tracking area:** Marker-based AR relies on a physical marker or object to provide a reference point for the AR software. This means that the tracking area is limited to the size of the marker, which can be a disadvantage if a larger tracking area is needed.
2. **Marker dependency:** Marker-based AR is dependent on the marker being in the field of view of the device's camera. If the marker is obscured or not visible, the AR content will not be displayed. This can be a limitation in some scenarios where the marker cannot be placed in a visible position or is not easily recognizable.
3. **Limited content:** Marker-based AR is limited to the content that can be displayed on the marker. This means that the amount of virtual content that can be displayed is limited by the size of the marker and the available space around it.
4. **Calibration:** The AR software must be calibrated to the specific marker used in the application. This means that each marker used in the application requires calibration, which can be time-consuming and tedious.
5. **User experience:** The user experience in marker-based AR applications can be affected by the quality of the camera and the lighting conditions. Poor lighting or low-resolution cameras can affect the accuracy of the AR experience and impact the user's overall experience.

## 6.2 Markerless AR

Markerless augmented reality (AR) is a type of AR technology that does not require the use of physical markers or visual cues to superimpose digital content onto the real world. Instead, it uses computer vision algorithms to recognize and track objects and their features in real-time, such as planes, edges, corners, and textures, and then overlay digital information onto them. This enables users to interact with virtual objects and information seamlessly in their natural environment without the need for special markers or codes.

### 6.2.1 Types of Markerless AR

1. **SLAM (Simultaneous Localization and Mapping):** This technology uses computer vision and depth sensors to create a 3D map of the user's environment and track their position and orientation in real-time. SLAM is commonly used in mobile AR applications, such as Pokemon Go and Snapchat filters.
2. **Object recognition:** This technology uses machine learning algorithms to recognize specific objects or patterns in the user's environment, such as a product or a logo, and overlay digital information onto them. Object recognition is commonly used in retail, advertising, and industrial design.
3. **Natural feature tracking:** This technology tracks the natural features of the user's environment, such as edges, corners, and textures, and uses them as reference points to overlay digital content onto the real world. Natural feature tracking is commonly used in education, gaming, and tourism.
4. **Spatial computing:** This technology combines computer vision, machine learning, and sensor data to create a more immersive and interactive AR experience that responds to the user's movements and gestures. Spatial computing is commonly used in industrial design, architecture, and engineering.

### 6.2.2 Real-Life Examples of Markerless AR

1. **Pokemon Go:** Pokemon Go is a popular mobile game that uses markerless AR to allow users to catch virtual creatures in the real world. The game uses the camera and sensors on a user's device to overlay virtual creatures onto the real world.
2. **IKEA Place:** IKEA Place is an AR app that allows users to virtually place furniture in their homes before making a purchase. The app uses markerless AR to scan and map the environment and accurately place virtual furniture in the real world.
3. **Snapchat filters:** Snapchat filters use markerless AR to overlay virtual filters onto a user's face in real-time, allowing them to transform their appearance and create fun and engaging content.
4. **AR navigation:** AR navigation apps, such as Google Maps and AR City, use markerless AR to provide users with directions and information in the real world. These apps use the camera and sensors on a user's device to overlay information onto the real world, such as street names and directions.
5. **Medical training:** Markerless AR is also being used in medical training to simulate surgeries and procedures in a virtual environment. This allows medical students to practice and refine their skills in a safe and controlled environment.

### 6.2.3 Software Development Kits (SDKs) for Markerless AR

1. **ARCore:** ARCore is a markerless AR SDK developed by Google. It allows developers to build AR applications that can track and map the environment without the need for markers or special sensors. ARCore is available for Android and iOS devices and supports a wide range of features, including motion tracking, environmental understanding, and light estimation.
2. **ARKit:** ARKit is a markerless AR SDK developed by Apple. It allows developers to create AR applications that can track and map the environment without the need for markers or special sensors. ARKit is available for iOS devices and supports a wide range of features, including motion tracking, environmental understanding, and light estimation.

3. **Vuforia:** Vuforia is a markerless AR SDK developed by PTC. It allows developers to create AR applications that can track and map the environment without the need for markers or special sensors. Vuforia supports a wide range of features, including image recognition, object recognition, and text recognition.
4. **Wikitude:** Wikitude is a markerless AR SDK that supports both image recognition and location-based AR. It allows developers to create AR applications that can track and map the environment without the need for markers or special sensors. Wikitude is available for Android and iOS devices and supports a wide range of features, including image recognition, object recognition, and location-based AR.
5. **EasyAR:** EasyAR is a markerless AR SDK that allows developers to create AR applications that can track and map the environment without the need for markers or special sensors. EasyAR supports a wide range of features, including image recognition, object recognition, and facial recognition.

#### 6.2.4 Benefits of Markerless AR

1. **No need for physical markers:** Markerless AR does not require the use of physical markers or codes, which makes it more accessible and convenient for users.
2. **Real-time tracking:** Markerless AR uses computer vision algorithms to track objects and features in real-time, allowing for more accurate and precise overlay of digital content onto the real world.
3. **More immersive and interactive:** Markerless AR enables users to interact with virtual objects and information seamlessly in their natural environment, creating a more immersive and interactive experience. This can be particularly useful for training, simulation, and visualization applications.
4. **Cost-effective:** Markerless AR can be more cost-effective than marker-based AR, as it does not require the production and distribution of physical markers or codes. This makes it more accessible for businesses and individuals who want to create AR experiences.

### 6.2.5 Limitations of Markerless AR

1. **Limited accuracy:** Markerless AR relies on computer vision algorithms to track objects and features in real-time, which can result in limited accuracy and precision. This can lead to issues such as misalignment of virtual objects, lagging or stuttering in movement, and occlusion of virtual objects by real-world objects.
2. **Limited compatibility:** Markerless AR requires devices with advanced cameras, sensors, and processing power, which may not be available or accessible to all users. This can limit the compatibility and reach of markerless AR applications.
3. **Lighting and environmental conditions:** Markerless AR can be affected by lighting and environmental conditions, such as shadows, reflections, and changes in lighting. This can result in tracking issues and inaccuracies.
4. **Processing power:** Markerless AR requires significant processing power and memory, which can strain the hardware of mobile devices and computers. This can result in slower performance and reduced battery life.
5. **Complex development:** Markerless AR development can be more complex and challenging than marker-based AR development. It requires advanced computer vision algorithms and machine learning techniques, which can be difficult to implement and optimize.

In this project, we have used Marker-based AR and developed an application using Unity to implement the same.

## 6.3 Code Used to Implement Marker-Based AR

### 6.3.1 Libraries Used

1. **cv2:** The cv2 library in Python provides bindings for the OpenCV (Open Source Computer Vision Library), which is widely used for real-time computer vision applications. Key features include:

**Image Processing:** Reading, writing, transforming (resize, rotate, crop), filtering, and converting color spaces.

**Video Processing:** Capturing and writing video, processing video frames in real-time.

**Feature Detection and Matching:** Detecting keypoints (e.g., SIFT, ORB), computing descriptors, and matching keypoints between images.

**Object Detection and Recognition:** Using pre-trained classifiers (e.g., Haar cascades) and deep learning models (e.g., YOLO, SSD).

**Geometric Transformations:** Affine and perspective transformations, homography.

**Image Segmentation:** Thresholding, contour detection, and advanced segmentation techniques like the watershed algorithm.

**Camera Calibration and 3D Reconstruction:** Calibrating cameras to remove distortions and performing stereo vision for 3D reconstruction.

**Machine Learning:** Implementing and using various machine learning algorithms (e.g., k-NN, SVM).

**GUI Features:** Displaying images and videos, handling user interactions, and drawing shapes.

2. **argparse:** The argparse library in Python is a module for parsing command-line arguments. It allows you to write user-friendly command-line interfaces by defining the arguments your program requires, parsing those arguments, and automatically generating help and usage messages. Key features include:

**Argument Definition:** Easily define positional and optional arguments.

**Automatic Help Generation:** Generates help and usage messages for users.

**Type Conversion:** Automatically converts argument values to the appropriate types.

**Validation:** Checks the validity of argument values.



**Custom Actions:** Define custom actions to be taken when specific arguments are encountered.

3. **math:** The math library in Python provides a wide range of mathematical functions and constants. It is part of Python's standard library, which means it is always available without needing to install additional packages.
4. **numpy:** numpy is a powerful numerical computing library in Python, widely used for array and matrix operations. It provides a high-performance multidimensional array object and tools for working with these arrays. Key features include:

**Multidimensional Arrays:** Efficient and flexible array handling with support for various data types.

**Mathematical Functions:** A wide range of mathematical operations on arrays (e.g., linear algebra, statistics, trigonometry).

**Random Number Generation:** Tools for generating random numbers and sampling.

**Broadcasting:** Allows operations on arrays of different shapes.

**Integration with Other Libraries:** Works seamlessly with libraries like pandas, scipy, and matplotlib.

5. **os:** The os library in Python provides a way to interact with the operating system.

## 6.3.2 Code to Implement 3D Marker-Based AR

```
1  import argparse
2
3  import cv2
4  import numpy as np
5  import math
6  import os
7
8
9  # Minimum number of matches that have to be found
10 # to consider the recognition valid
11 class OBJ:
12     def __init__(self, filename, swapyz=False):
13         """Loads a Wavefront OBJ file. """
14         self.vertices = []
15         self.normals = []
16         self.texcoords = []
17         self.faces = []
18         material = None
19         for line in open(filename, "r"):
20             if line.startswith('#'): continue
21             values = line.split()
22             if not values: continue
23             if values[0] == 'v':
24                 v = list(map(float, values[1:4]))
25                 if swapyz:
26                     v = v[0], v[2], v[1]
27                 self.vertices.append(v)
28             elif values[0] == 'vn':
29                 v = list(map(float, values[1:4]))
30                 if swapyz:
31                     v = v[0], v[2], v[1]
32                 self.normals.append(v)
33             elif values[0] == 'vt':
34                 self.texcoords.append(list(map(float, values[1:3])))
```

```

35         #elif values[0] in ('usemtl', 'usemat'):
36             #material = values[1]
37         #elif values[0] == 'mtllib':
38             #self.mtl = MTL(values[1])
39         elif values[0] == 'f':
40             face = []
41             texcoords = []
42             norms = []
43             for v in values[1:]:
44                 w = v.split('/')
45                 face.append(int(w[0]))
46                 if len(w) >= 2 and len(w[1]) > 0:
47                     texcoords.append(int(w[1]))
48                 else:
49                     texcoords.append(0)
50                 if len(w) >= 3 and len(w[2]) > 0:
51                     norms.append(int(w[2]))
52                 else:
53                     norms.append(0)
54             #self.faces.append((face, norms, texcoords, material))
55             self.faces.append((face, norms, texcoords))
56
57     MIN_MATCHES = 10
58     DEFAULT_COLOR = (0, 0, 0)
59
60
61     def main():
62         """
63         This functions loads the target surface image,
64         """
65         homography = None
66         # matrix of camera parameters (made up but works quite well for me)

```

```

67 camera_parameters = np.array([[800, 0, 320], [0, 800, 240], [0, 0, 1]])
68 # create ORB keypoint detector
69 orb = cv2.ORB_create()
70 # create BFMatcher object based on hamming distance
71 bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
72 # load the reference surface that will be searched in the video stream
73 dir_name = os.getcwd()
74 model = cv2.imread(os.path.join(dir_name, 'reference/dracula.jpg'), 0)
75 # Compute model keypoints and its descriptors
76 kp_model, des_model = orb.detectAndCompute(model, None)
77 # Load 3D model from OBJ file
78 # here, we can use any object which we want to augment on the model image
79 obj = OBJ(os.path.join(dir_name, 'reference/fox.obj'), swapyz=True)
80 # init video capture
81 cap = cv2.VideoCapture(0)
82
83 while True:
84     # read the current frame
85     ret, frame = cap.read()
86     if not ret:
87         print("Unable to capture video")
88         return
89     # find and draw the keypoints of the frame
90     kp_frame, des_frame = orb.detectAndCompute(frame, None)
91     # match frame descriptors with model descriptors
92     matches = bf.match(des_model, des_frame)
93     # sort them in the order of their distance
94     # the lower the distance, the better the match
95     matches = sorted(matches, key=lambda x: x.distance)
96
97     # compute Homography if enough matches are found
98     if len(matches) > MIN_MATCHES:
99         # differentiate between source points and destination points

```

```

100 src_pts = np.float32([kp_model[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
101 dst_pts = np.float32([kp_frame[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)
102 # compute Homography
103 homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
104 if args.rectangle:
105     # Draw a rectangle that marks the found model in the frame
106     h, w = model.shape
107     pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
108     # project corners into frame
109     dst = cv2.perspectiveTransform(pts, homography)
110     # connect them with lines
111     frame = cv2.polylines(frame, [np.int32(dst)], True, 255, 3, cv2.LINE_AA)
112 # if a valid homography matrix was found render cube on model plane
113 if homography is not None:
114     try:
115         # obtain 3D projection matrix from homography matrix and camera parameters
116         projection = projection_matrix(camera_parameters, homography)
117         # project cube or model
118         frame = render(frame, obj, projection, model, False)
119         #frame = render(frame, model, projection)
120     except:
121         pass
122 # draw first 10 matches.
123 if args.matches:
124     frame = cv2.drawMatches(model, kp_model, frame, kp_frame, matches[:10], 0, flags=2)
125 # show result
126 cv2.imshow('frame', frame)
127 if cv2.waitKey(1) & 0xFF == ord('q'):
128     break
129
130 else:
131     print("Not enough matches found - %d/%d" % (len(matches), MIN_MATCHES))
132

```

```

133     cap.release()
134     cv2.destroyAllWindows()
135     return 0
136
137 def render(img, obj, projection, model, color=False):
138     """
139     Render a loaded obj model into the current video frame
140     """
141     vertices = obj.vertices
142     scale_matrix = np.eye(3) * 3
143     h, w = model.shape
144
145     for face in obj.faces:
146         face_vertices = face[0]
147         points = np.array([vertices[vertex - 1] for vertex in face_vertices])
148         points = np.dot(points, scale_matrix)
149         # render model in the middle of the reference surface. To do so,
150         # model points must be displaced
151         points = np.array([[p[0] + w / 2, p[1] + h / 2, p[2]] for p in points])
152         dst = cv2.perspectiveTransform(points.reshape(-1, 1, 3), projection)
153         imgpts = np.int32(dst)
154         if color is False:
155             cv2.fillConvexPoly(img, imgpts, DEFAULT_COLOR)
156         else:
157             color = hex_to_rgb(face[-1])
158             color = color[::-1] # reverse
159             cv2.fillConvexPoly(img, imgpts, color)
160
161     return img
162
163 def projection_matrix(camera_parameters, homography):
164     """
165     From the camera calibration matrix and the estimated homography

```

```

166     compute the 3D projection matrix
167     """
168     # Compute rotation along the x and y axis as well as the translation
169     homography = homography * (-1)
170     rot_and_transl = np.dot(np.linalg.inv(camera_parameters), homography)
171     col_1 = rot_and_transl[:, 0]
172     col_2 = rot_and_transl[:, 1]
173     col_3 = rot_and_transl[:, 2]
174     # normalise vectors
175     l = math.sqrt(np.linalg.norm(col_1, 2) * np.linalg.norm(col_2, 2))
176     rot_1 = col_1 / l
177     rot_2 = col_2 / l
178     translation = col_3 / l
179     # compute the orthonormal basis
180     c = rot_1 + rot_2
181     p = np.cross(rot_1, rot_2)
182     d = np.cross(c, p)
183     rot_1 = np.dot(c / np.linalg.norm(c, 2) + d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
184     rot_2 = np.dot(c / np.linalg.norm(c, 2) - d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
185     rot_3 = np.cross(rot_1, rot_2)
186     # finally, compute the 3D projection matrix from the model to the current frame
187     projection = np.stack((rot_1, rot_2, rot_3, translation)).T
188     return np.dot(camera_parameters, projection)
189
190 def hex_to_rgb(hex_color):
191     """
192     Helper function to convert hex strings to RGB
193     """
194     hex_color = hex_color.lstrip('#')
195     h_len = len(hex_color)
196     return tuple(int(hex_color[i:i + h_len // 3], 16) for i in range(0, h_len, h_len // 3))
197
198
199 # Command line argument parsing
200 # NOT ALL OF THEM ARE SUPPORTED YET
201 parser = argparse.ArgumentParser(description='Augmented reality application')
202
203 parser.add_argument('-r', '--rectangle', help='draw rectangle delimiting target surface on frame', action='store_true')
204 parser.add_argument('-mk', '--model_keypoints', help='draw model keypoints', action='store_true')
205 parser.add_argument('-fk', '--frame_keypoints', help='draw frame keypoints', action='store_true')
206 parser.add_argument('-ma', '--matches', help='draw matches between keypoints', action='store_true')
207 # TODO jgallostraa -> add support for model specification
208 #parser.add_argument('-mo', '--model', help='Specify model to be projected', action='store_true')
209
210 args = parser.parse_args()
211
212 if __name__ == '__main__':
213     main()

```

## 6.3.2 Explanation of the Code

This code is for an augmented reality application that overlays a 3D model onto a video feed. It uses OpenCV and NumPy for computer vision tasks and mathematical computations, respectively, and argparse for command-line argument parsing.

The **OBJ** class is responsible for loading a 3D model from an OBJ file. It parses the vertices, normals, and texture coordinates from the file, handling different types of data lines such as vertex positions (v), vertex normals (vn), and texture coordinates (vt). It also processes the face definitions (f) to form a complete representation of the 3D model.

The **main** function orchestrates the augmented reality process. It starts by initializing the camera parameters, creating an ORB keypoint detector, and a brute-force matcher for matching keypoints. It loads a reference image and computes its keypoints and descriptors. The function then loads the 3D model to be rendered.

The video capture begins, and for each frame captured from the webcam, the function detects and computes keypoints and descriptors. It matches these with the reference image's descriptors and sorts the matches by distance. If enough matches are found, it computes the homography matrix, which relates the positions of points in the reference image to their positions in the current video frame. Depending on command-line arguments, it might draw a rectangle around the detected reference surface or display keypoint matches.

The **render** function projects the 3D model onto the current video frame using the homography matrix and camera parameters. It adjusts the model's vertices according to the homography and fills the polygons on the video frame to create the augmented reality effect.

The **projection\_matrix** function computes the 3D projection matrix from the homography matrix and camera calibration parameters, which is crucial for accurately rendering the 3D model in the video frame.

The **hex\_to\_rgb** function is a utility to convert hexadecimal color strings to RGB tuples, though it's not utilized in the provided code.

Finally, the script sets up command-line argument parsing with argparse, allowing users to specify various options for the augmented reality application. The main function is called if the script is executed as the main module.



## 7.CONCLUSION

In this project we designed a car that can be controlled by simple hand gestures by placing an IMU sensor in a hand glove and monitoring its motion to relay appropriate instructions to the car. Moreover, we used Marker based AR that takes in the camera feed from a camera on the car and by recognizing certain markers, overlays specific objects into the camera feed which can be viewed on a computer. This results in a gamified experience for a user, where the user can see special objects in the camera feed of the car, and use hand gestures to control the car in presence of the virtual obstacles.

### 7.1 Key Achievements

**7.1.1 Hand Gesture controlled car :-** In this project we were successfully able to make a car that can be controlled entirely by intuitive hand gestures. We used the MPU6050's data and processed it to control the car according to the gestures performed by the user.

**7.1.2 Implementation of Marker-Based AR :-** In this project we used the camera feed from the car to not only give the perfect FPV to the user, but to enhance it further by adding virtual objects to the camera feed by implementing Marker-based Augmented Reality.

## 8. FUTURE ASPECTS

While this project has successfully achieved its primary objectives, there are several avenues for further improvement and expansion:

1. **Integration of flex sensors:-** Flex sensors can be added to monitor the motion of fingers, and then use specific hand gestures to do specific maneuvers for the car, or adding additional functionality to the gamified experience this project provides, like it can be used to change modes of the game or can give instructions to car other than controlling its motion.
2. **Implementation of Machine Learning:-** To recognise complex gestures accurately and distinguish them from other gestures, either by flex sensor data or by IMU or other sensors, Machine Learning models can be employed. These may be used for a variety of purposes, from controlling complex movements of car to controlling the video feed or to change modes.
3. **Implementation of Markerless AR:-** The current approach uses Markers to recognise certain features in the Camera feed and overlay virtual objects on them. This approach could be taken a step further by implementing Markerless AR, which eliminates the need of markers.
4. **Crash Prevention in Car:-** Infrared sensors can be added to the car to prevent it from crashing into nearby obstacles. Alternatively, camera feed from the onboard camera can also be used to detect objects and plan path accordingly to prevent crashing, although this approach can be a bit complicated and may require 2 onboard cameras to work.

## 9.REFERENCES

1. Arduino boards and Arduino IDE:-  
<https://docs.arduino.cc/learn/starting-guide/getting-started-arduino/>
2. NodeMCU:-  
<https://robocraze.com/blogs/post/what-is-nodemcu-esp8266>
3. Communication between 2 NodeMCUs:-  
<https://randomnerdtutorials.com/esp-now-two-way-communication-esp8266-node-mcu/>
4. MPU6050:-  
<https://www.flyrobo.in/blog/Exploring-Motion-with-MPU-6050-A-Quick-Guide>
5. Integration of L298 Motor Driver module with the Car:-  
<https://projecthub.arduino.cc/lakshyajhalani56/l298n-motor-driver-arduino-motors-motor-driver-l298n-7e1b3b>
6. Marker AR:-  
[https://youtu.be/FJAO6jDYljs?si=gnXa70x0ycK92R8\\_](https://youtu.be/FJAO6jDYljs?si=gnXa70x0ycK92R8_)