# ABSTRACT

The automotive industry is transforming as vehicles integrate advanced digital technologies, enhancing usability and real-time data access. Among these advancements, self-driving vehicles have emerged as a key focus. Defined by the NHTSA as vehicles capable of operating without human intervention, autonomous cars are set to reshape transportation. The levels of autonomous driving range to 6 levels, from level 0 (no automation) to level 5 (full automation). Our project contributes by developing basic autonomous functions, such as driving in straight lines and following geometric patterns. This is done by using some components that determine the direction and control the direction of the vehicle through some codes from the C/C++ language.

# INTRODUCTION

Nowadays, the majority of vehicles have been converted to digital form. Offer the driver improved ease of use and enhanced data, including up-to-the-minute traffic updates, performance stats evaluation of information similar to velocity, and playing audio online. Cloud computing, among other advancements, has made cars today highly technologically advanced, and there is a lot more yet to happen. Tomorrow's car will transform the automotive industry. Be a significant improvement from what is currently available. Self-driving or autonomous vehicles have become prevalent. a long-cherished aspiration—a dream that has consistently been unsuccessful—come into existence. Self-driving vehicles are now a reality.  A highly specific and specialized In the market, the autonomous car sector is progressing quickly. Progress in incorporating numerous technologies from various sources Developing a self-driving car requires a complex ecosystem. To begin with, what exactly are autonomous vehicles? Per the information provided According to the National Highway Safety Administration (NHTSA), self-driving vehicles are cars capable of driving on their own without human assistance. No human intervention is necessary to steer. We are increasing speed and reducing speed. The definition above suggests self-driving vehicles equipped with autonomous technologies. Allow the vehicle to travel from Point A to Point B by executing all the necessary features needed for a vehicle to operate safely without any passengers inside. Despite the prevailing notion, driverless vehicles are considered a futuristic idea. Competition has commenced to introduce these vehicles onto our streets. These vehicles are causing a disturbance of unprecedented scale and reach. As we know that there are many levels of self-driving, we aspire to reach the highest level of these levels with new technologies. To reach all of this, we will learn about our first steps so that we can work on this. We have used some of the components that help us in controlling and obtaining the correct directions so that the vehicle can drive correctly. We will learn about each of them separately. We have also created a basic form for the application to control the vehicle's movement and direction remotely.

# 1. AUTONOMOUS DRIVING TECHNOLOGY

Early research and development of autonomous driving systems began decades ago. Significant milestones include the development of driver assistance systems in the 1980s and the DARPA Grand Challenges in the early 2000s. Commercial ADAS have been introduced in recent years.

### ➢ Levels of Autonomy:

The Society of Automotive Engineers (SAE) has defined six levels of autonomy for self-driving cars, as shown in Figure 1 :

- ***Level 0: No Automation***

  The driver controls all aspects of driving.

- ***Level 1: Driver Assistance***

  The vehicle can assist with steering or acceleration/deceleration, but the driver is still in control.

- ***Level2: Partial Automation***

  The vehicle can control both steering and acceleration/deceleration under certain conditions, but the driver must remain vigilant.

- ***Level 3: Conditional Automation***

  The vehicle can handle most driving tasks under certain conditions, but the driver must be ready to take over.

- ***Level 4: High Automation***

  The vehicle can drive itself under specific conditions, but a human driver may be needed in some situations.

- ***Level 5: Full Automation***

  The vehicle can drive itself under all conditions, without any human input.

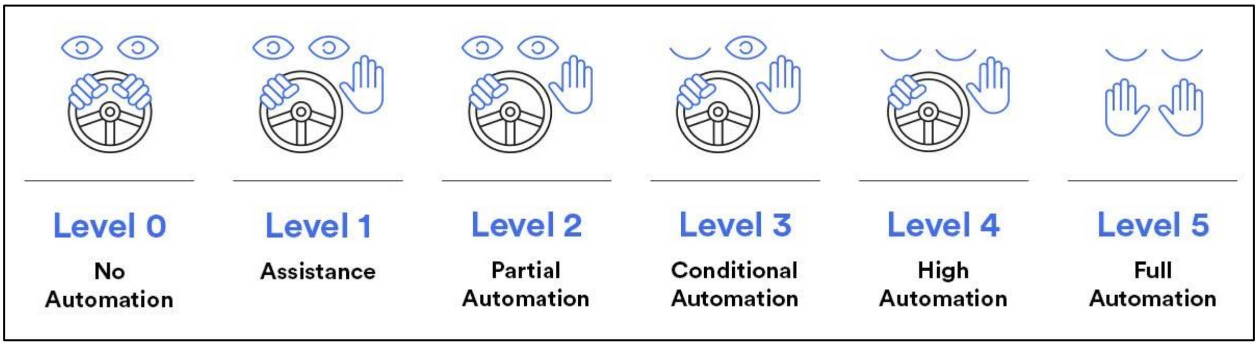


Figure 1 LEVELS OF AUTONOMY

## 2. COMPONENTS OF PROJECT :

## 2.1. Arduino UNO

*Arduino UNO* is a microcontroller board based on the ATmega328P, as shown in Figure 2. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.

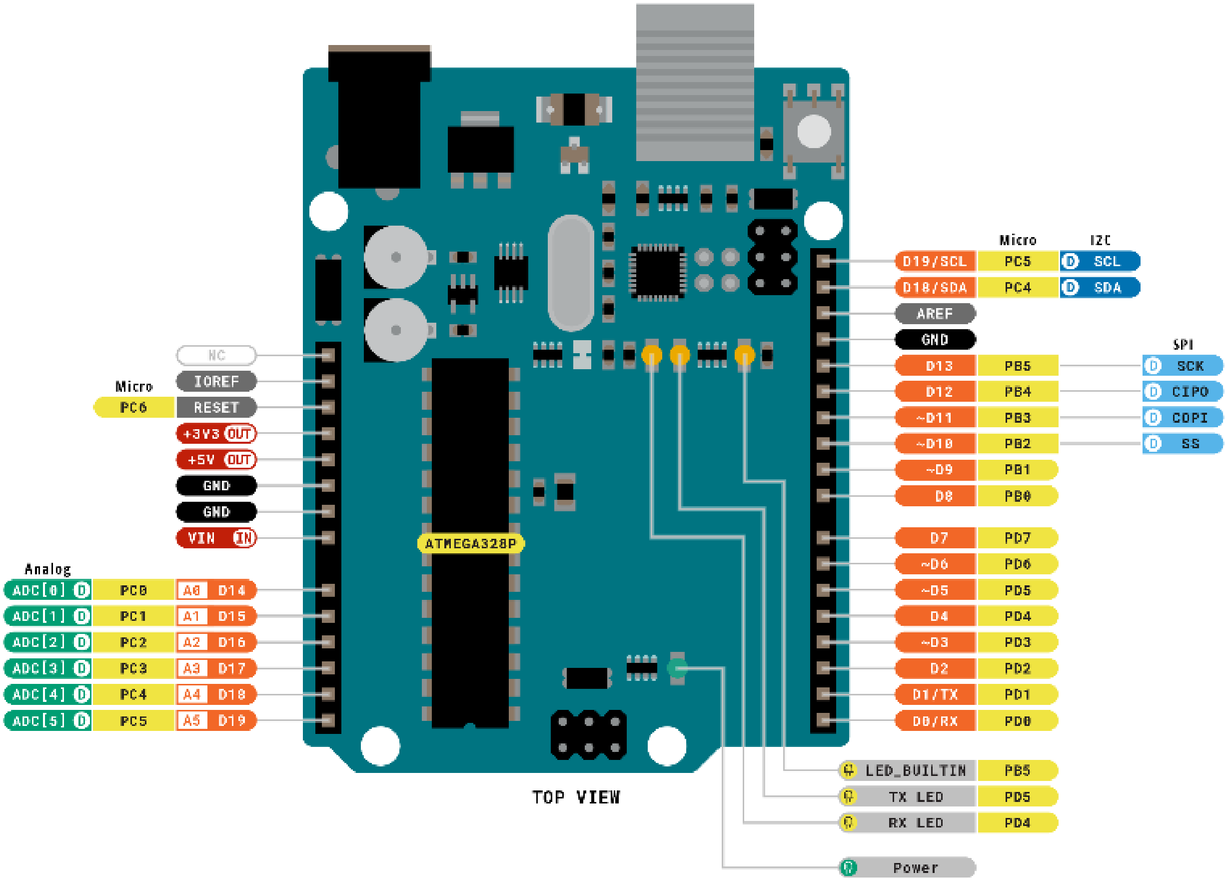


Figure 2 ARDUINO UNO PINOUT

## 2.2. L298 motor driver

*The L298 motor driver* is a popular dual H-bridge motor driver integrated circuit (IC) as shown in Figure 3, commonly used in projects requiring motor control :

- **Dual H-Bridge**
- **Maximum Voltage:** up to 46V
- **Current Handling:** The L298 can handle peak currents of up to 2A per channel and 4A if a heat sink is used .
- **Enable Pins:** Each H-bridge has an enable pin that allows for PWM (pulse-width modulation) control of motor speed.

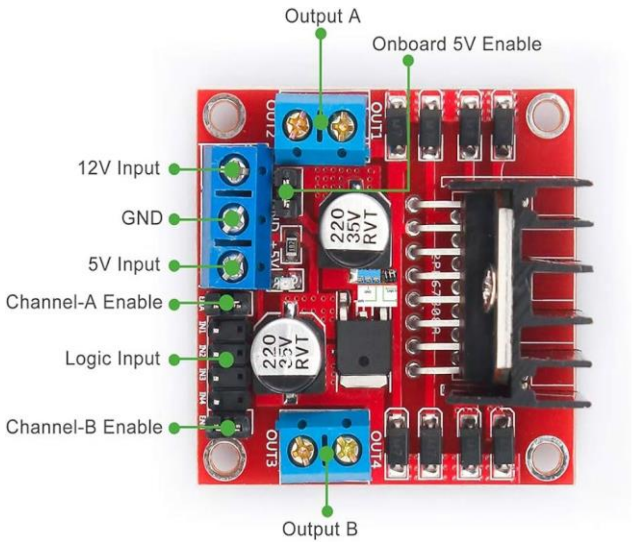


Figure 3 L298 PINOUT

## 2.3. DC Motors

*A DC motor* is an electrical device that converts electrical energy into mechanical motion. It operates on direct current (DC) power and is used in autonomous vehicles, for functions like propulsion and steering. These motors enable the car to move, accelerate, decelerate, and navigate autonomously, playing a key role in the vehicle's overall functionality and movement capabilities. Powered by 3-12 volts, draws an average of 190 mA of current (max 250 mA), as shown in Figure 4 .

Figure 4 DC MOTORS + TIRES

## 2.4. MPU6050

*The MPU-6050* is a Motion Tracking Device that combines a 3-axis gyroscope and a 3-axis accelerometer in a single chip. It is commonly used in various electronic applications for measuring motion and orientation, as shown in Figure 5 . Here are the specifications of the MPU-6050 :

- **Operating voltage :** 2.375 V to 3.46 V
- **Gyroscope :**
  - **Angular Velocity:** Measures rotational motion in degrees per second (dps).
  - **Sensitivity:** The sensitivity can be configured based on the selected range (e.g., ±250, ±500, ±1000, or ±2000 dps).
- **Temperature Sensor :** The MPU-6050 has an integrated temperature sensor that measures the device's temperature.

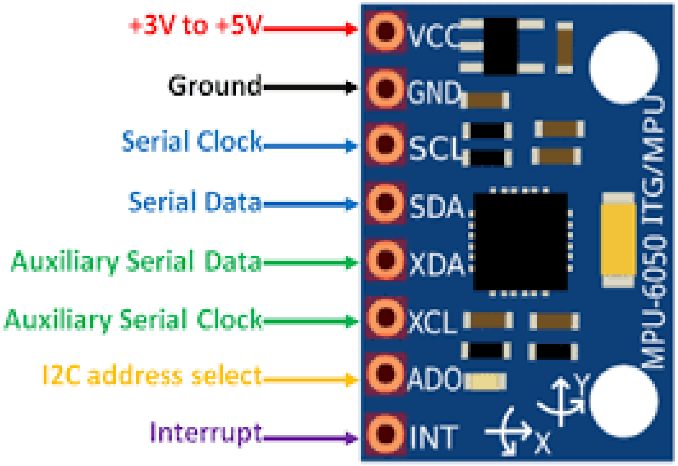

Figure 5 MPU6050

- **Accelerometer :**
  - **Acceleration:** Measures linear acceleration in three axes (X, Y, Z).
  - **Sensitivity:** The sensitivity can be configured based on the selected range (e.g., ±2g, ±4g, ±8g, or ±16g).
- **The Digital Low Pass Filter** can be configured to filter out high-frequency noise from sensor data.

## 2.5. ESP8266

*The ESP8266* is a low-cost, highly integrated Wi-Fi microcontroller chip designed by **Espressif Systems**. It's widely used in IoT (Internet of Things) projects due to its ease of use, low power consumption, and robust wireless connectivity, as shown in Figure 6. Below are the key details about the ESP8266 :

- **Memory :**
  - **RAM:** 160 KB of on-chip SRAM.
  - **ROM:** 32 KB boot ROM.
  - **Flash:** The chip supports external QSPI flash memory.
- **Power :**
  - **Operating Voltage :** 3.0 V to 3.6 V.
  - **Power Consumption :** Low power consumption with sleep modes.

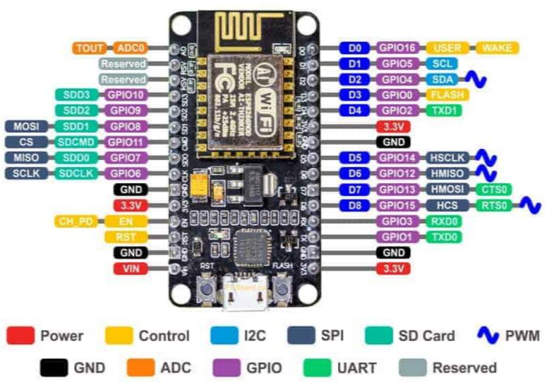

Figure 6 ESP8266 PINOUT

- **Programming Languages :** Typically programmed in C/C++.

# 3. CIRCUIT DIGRAM

We will show how to connect the components to each other as shown in Figure 7 .

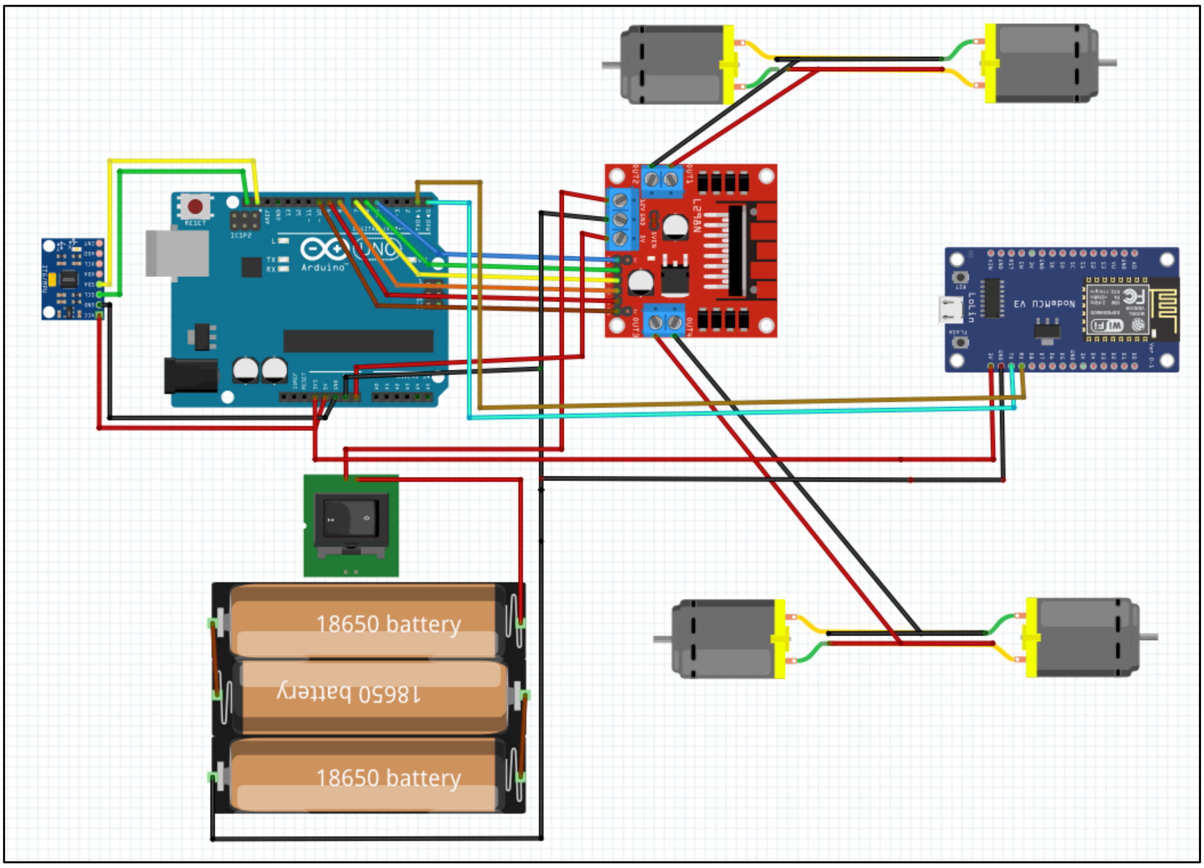


Figure 7 CIRCUIT DIGRAM

# 4. ARDUINO CODE FLOWCHART

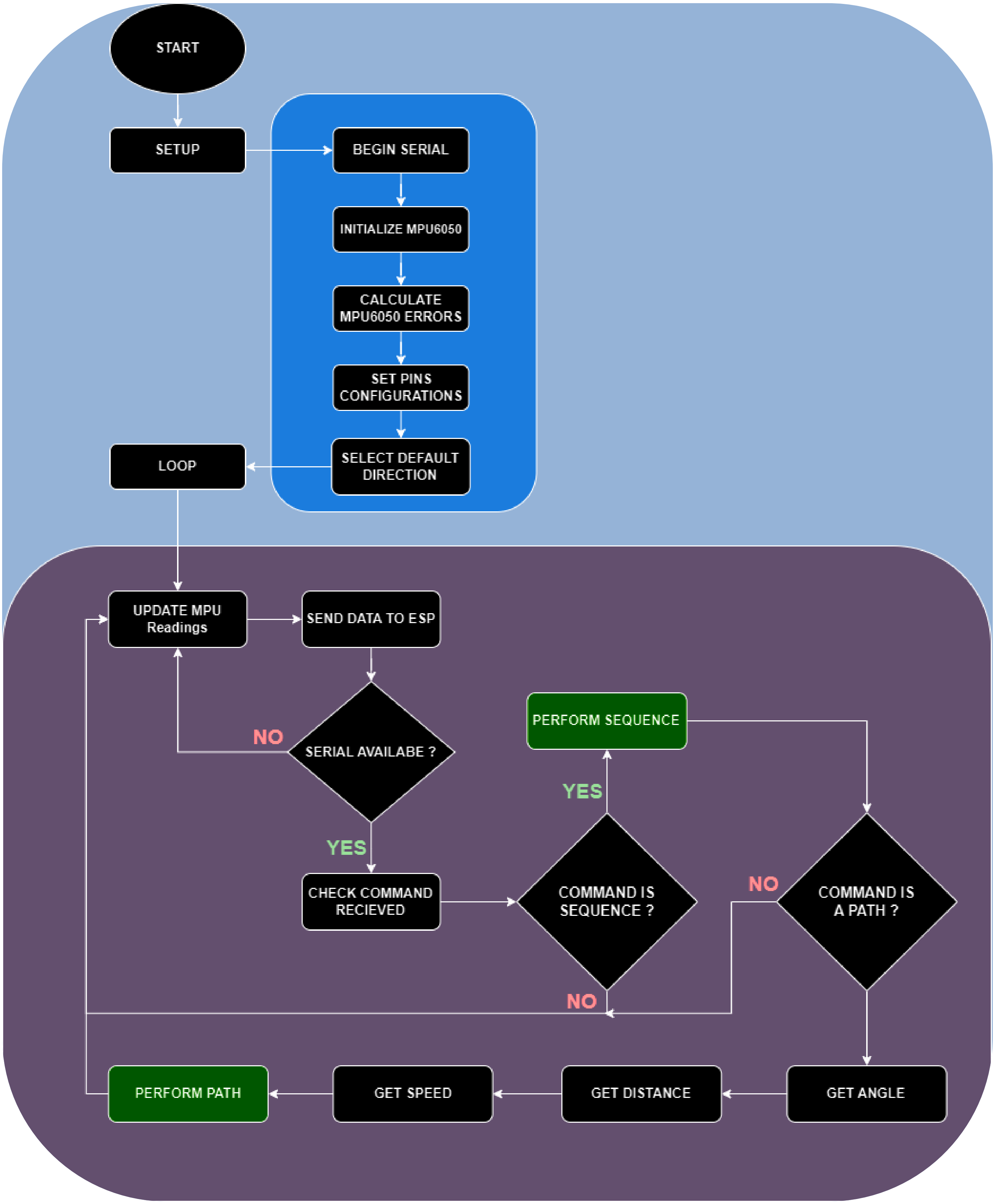


Figure 8 FLOW CHART OF ARDUINO CODE

# 5. EXPLAIN MPU6050 EQUATIONS WITH COMPLEMENTARY FILTER

*MPU6050* has two main functions in Arduino code as follows :

- ***Calculate_IMU_error :***

This function calculate_IMU_error(), is designed to compute the error values for both accelerometer and gyroscope sensors of the Inertial Measurement Unit (IMU) *MPU6050*.

## 1. Accelerometer Data Error Calculation :

```
while (c < 200) {
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true);
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
  AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
  AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
  c++;
}
AccErrorX = AccErrorX / 200;
AccErrorY = AccErrorY / 200;
```

- The function uses an I2C protocol (Wire library) to communicate with the IMU sensor (MPU).
- It sends a command to read the accelerometer data (starting from register 0x3B for the *MPU6050*).
- It then requests 6 bytes of data from the IMU. The accelerometer provides three 16-bit values (X, Y, and Z axes).
- The 6 bytes are split into three 16-bit values for the X, Y, and Z axes.
- The raw accelerometer data is divided by 16384.0 to convert it into units of gravitational force (g), assuming the sensitivity scale is set to ±2g (for the *MPU6050*).
- The function calculates the pitch (AccErrorX) and roll (AccErrorY) angles from the accelerometer data using trigonometric formulas.
- The error is accumulated over 200 readings. The formula uses the atan() (arctangent) function to derive the angle in degrees from the accelerometer's raw data.
- After collecting 200 readings, the accumulated error is divided by 200 to compute the average error, which represents the systematic bias in the sensor readings.

## 2. Gyroscope Data Error Calculation :

```
c = 0;
 while (c < 200) {
   Wire.beginTransmission(MPU);
   Wire.write(0x43);
   Wire.endTransmission(false);
   Wire.requestFrom(MPU, 6, true);
   GyroX = Wire.read() << 8 | Wire.read();
   GyroY = Wire.read() << 8 | Wire.read();
   GyroZ = Wire.read() << 8 | Wire.read();
   GyroErrorX = GyroErrorX + (GyroX / 131.0);
   GyroErrorY = GyroErrorY + (GyroY / 131.0);
   GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
   c++;
 }
```

- The gyroscope error calculation begins in the same way as the accelerometer, except it reads from a different register (0x43 for the *MPU6050*).
- The function requests 6 bytes of data corresponding to the gyroscope's X, Y, and Z axis readings.
- Like the accelerometer, the 6 bytes are split into three 16-bit values for the X, Y, and Z axes of the gyroscope.
- The raw gyroscope data is scaled by dividing it by 131.0. This value assumes the gyroscope is set to a sensitivity of ±250 degrees per second (dps) for the *MPU6050*, where each unit represents 1/131.0 degrees per second.
- The error for each axis is accumulated over 200 readings.

## 3. Averaging the Gyroscope Errors :

- After collecting 200 readings, the accumulated gyroscope error is divided by 200 to compute the average error, representing the bias in the sensor.

```
GyroErrorX = GyroErrorX / 200;
GyroErrorY = GyroErrorY / 200;
GyroErrorZ = GyroErrorZ / 200;
```

## ➢ Update_mpu_readings :

This function update_mpu_readings() reads data from an *MPU6050* sensor and calculates roll, pitch, and yaw using a complementary filter.

## 1. Reading Accelerometer Data :

- The function communicates with the *MPU6050* via I2C and sends a request to start reading accelerometer data from register 0x3B.
- The function requests 6 bytes of data, corresponding to the accelerometer values for the X, Y, and Z axes (each value is 16-bit, hence 2 bytes per axis).

```
Wire.beginTransmission(MPU);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true);
```

## 2. Extracting and Normalizing Accelerometer Data :

```
AccX = (Wire.read() << 8 | Wire.read()) / 16384.0;
AccY = (Wire.read() << 8 | Wire.read()) / 16384.0;
AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0;
```

- The 6 bytes of data are split into three 16-bit integers for the X, Y, and Z axes using bitwise operations (<< 8 |).
- The raw accelerometer values are divided by 16384.0 to convert them to units of gravitational acceleration (g). This scaling factor corresponds to the *MPU6050* being set to a ±2g range (as per the sensor's datasheet).

## 3. Calculating Roll and Pitch from Accelerometer Data :

```
accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorX;
accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) - AccErrorY;
```

- The function uses trigonometric formulas to compute roll (accAngleX) and pitch (accAngleY) angles based on the accelerometer data.

- atan() calculates the arctangent of the given ratio, and then the result is converted from radians to degrees (* 180 / PI).

- The calculated accelerometer errors (AccErrorX, AccErrorY) are subtracted to compensate for the inherent bias of the sensor, which was determined using the calculate_IMU_error() function.

- Formulas used :

    - **Roll** (rotation about the X-axis):

$$Roll = \text{atan}(\frac{A_y}{\sqrt{A_x^2 + A_z^2}})$$

    - **Pitch** (rotation about the Y-axis):

$$Pitch = \text{atan}(\frac{-A_x}{\sqrt{A_y^2 + A_z^2}})$$

## 4. Reading Gyroscope Data :

```
previousTime = currentTime;
currentTime  = millis();
elapsedTime  = (currentTime - previousTime) / 1000.0;
Wire.beginTransmission(MPU);
Wire.write(0x43);
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true);
```

- The function records the time between the current and previous readings (elapsedTime), which is used to compute the angular displacement from the angular velocity.
- elapsedTime is measured in seconds by dividing the time difference in milliseconds by 1000.
- The function starts reading from register 0x43 to get gyroscope data.
- 6 bytes are requested to get the X, Y, and Z angular velocities.

## 5. Extracting and Scaling Gyroscope Data :

```
GyroX = (Wire.read() << 8 | Wire.read()) / 131.0;
GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
```

- Like the accelerometer, the 6 bytes are split into three 16-bit integers for the X, Y, and Z axes.
- The raw gyroscope values are divided by 131.0 to convert them into degrees per second (dps). This scaling factor assumes the gyroscope is set to a sensitivity of ±250 degrees per second (as per the sensor's datasheet).

## 6. Correcting Gyroscope Data Using Error Values:

- The gyroscope readings are corrected by subtracting pre-calculated error values (GyroErrorX, GyroErrorY, GyroErrorZ),

```
GyroX = GyroX - GyroErrorX;
GyroY = GyroY - GyroErrorY;
GyroZ = GyroZ - GyroErrorZ;
```

which were determined using the calculate_IMU_error() function.

- This corrects the gyroscope bias for more accurate readings.

### 7. Calculating Gyroscope Angles:

```
gyroAngleX = gyroAngleX + GyroX * elapsedTime;
gyroAngleY = gyroAngleY + GyroY * elapsedTime;
yaw = yaw + GyroZ * elapsedTime;
```

- The angular velocities from the gyroscope are integrated over time to compute the angular displacement.
- Since gyroscope values are in degrees per second, multiplying by elapsedTime (in seconds) gives the angle in degrees.
- gyroAngleX, gyroAngleY, and yaw represent the angles calculated solely from the gyroscope data.

### 8. Applying Complementary Filter:

```
roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;
```

- The complementary filter combines the gyroscope and accelerometer angles to get more stable and accurate values for roll and pitch.
- The gyroscope provides quick response to changes, but it tends to drift over time. The accelerometer gives more stable values but can be noisy.
- The complementary filter formula weights the gyroscope data more heavily (0.96) but still uses a small portion of accelerometer data (0.04) to counteract drift.
- The final roll and pitch angles are a blend of both gyroscope and accelerometer data, where the filter helps in maintaining both responsiveness and long-term stability.

## 6. HOW TO CHANGE DIRECTION OF THE CAR

The direction of the car is controlled by adjusting the motor pin signals based on yaw readings from the *MPU6050* sensor.
There are three main functions to change direction of the car:

## 1. Forward Movement :