Productivity and Vocational Training Service

Misr International Computer & AI

Department of Artificial Intelligence and Embedded Systems

# *Auto Zone*


## **Team Members**:

Ahmed Mohamed Elshrkawy

Adham Mohamed Elshrkawy

Belal Mohamed Salah

Fady Maged Ezkeil

## **Under Supervision Of:**

Dr. Michael Nassif Michael

Eng. Hassan Talal Hetta


2024/2025

Productivity and Vocational Training Service

Misr International Computer & AI

Department of Artificial Intelligence and Embedded Systems

# *Auto Zone*


## Team Members:

Ahmed Mohamed Elshrkawy

Adham Mohamed Elshrkawy

Belal Mohamed Salah

Fady Maged Ezkeil

## Under Supervision Of:

Dr. Michael Nassif Michael

Eng. Hassan Talal Hetta


2024/2025

# In Memory and Gratitude to Our Late School Owner

With hearts heavy with sorrow and eyes filled with tears, we mourn the passing of our esteemed school Owner, **MR./Ahmed Hamdy**, who has left this world, leaving behind a legacy of wisdom, education, and leadership.

He was more than just a principal; he was a mentor, a guide, and a father figure to us all. His dedication to shaping young minds, his unwavering commitment to excellence, and his kind and generous spirit will forever be remembered. He left his mark in every corner of our school, in every student he inspired, and in every teacher, who worked alongside him.

With his passing, we have lost a true leader—one who gave selflessly and worked tirelessly for the success and growth of those under his care. But in our grief, we find comfort knowing that his legacy lives on in the knowledge he imparted and the values he instilled in us.

In this moment of profound loss, we extend our deepest gratitude to everyone who has shared in our sorrow. We pray that God grants him eternal peace, embraces him in His mercy, and grants his family, loved ones, and all who cherished him patience and strength.

**Rest in peace, dear principal. You will always remain in our hearts, and your impact on our school will never be forgotten.**

# Acknowledgment

We extend our deepest gratitude and appreciation to **Dr. Michael Nassif Michael** for his invaluable guidance, unwavering support, and insightful advice throughout the development of this project. Their expertise, patience, and encouragement have been instrumental in shaping the direction and quality of this work.

Throughout this journey **Dr. Michael Nassif Michael** has provided not only academic supervision but also inspiration and motivation, fostering an environment of learning and growth. His constructive feedback and thoughtful suggestions have greatly enriched this project, and for that, we are truly grateful.

We would also like to express our sincere thanks to **Eng. Hassan Talal Hetta** for his unwavering assistance, technical insights, and constructive feedback. His dedication and willingness to share his knowledge significantly contributed to the quality and success of this work.

Throughout this journey, **Eng. Hassan Talal Hetta** has not only provided exceptional supervision but has also offered invaluable guidance, encouragement, and support. His mentorship extended beyond the academic realm, as he treated us with kindness and sincerity, offering advice and motivation at every stage. His unwavering dedication played a crucial role in the successful completion of this project, and for that, we express our deepest gratitude.

It has been an honor to work under their supervision, and We sincerely appreciate their dedication and generosity in sharing their knowledge. May their efforts be rewarded with continued success and fulfillment.

Furthermore, we are immensely grateful to our professors, colleagues, and everyone who has offered their support, whether through mentorship, valuable discussions, or motivation. Their encouragement has been a driving force behind our perseverance.

Last but not least, we extend our heartfelt appreciation to our families and friends, whose constant encouragement, patience, and unwavering belief in us have been a source of strength throughout this endeavor.

# Abstract

With hearts filled with gratitude, we begin by praising Allah, the Best tower of knowledge and wisdom, whose guidance has illuminated our path. It is by His grace that we have completed this endeavor, and we pray that He grants it acceptance, blesses its impact, and rewards all those who contributed to its completion.

The **Auto Zone** presents a self-driving car model that operates without human intervention, relying on advanced artificial intelligence models that have been specifically developed for this purpose. The system is built upon a network of communication protocols that facilitate seamless interaction between the car model, the AI system, and the designated destination. Throughout its journey, the vehicle dynamically interprets and adapts to traffic regulations, ensuring compliance while efficiently navigating various road conditions.

# Table of Contents

# Table of Figures

# Chapter [1]:

# Introduction

## 1-1: Project Idea:

- We endeavored to create a miniature model that closely resembles the most advanced self-driving cars available today. To achieve this, we analyzed the features offered by leading companies in their artificial intelligence models and developed our own models tailored to suit the specific conditions of our local road environment.

## 1-2: Project Objective:

One of the primary objectives of this project is to develop self-driving cars capable of operating and adapting to the unique environmental conditions in Egypt. These autonomous vehicles are not solely reliant on technologies and components imported

from China, the United States, or other countries. Instead, this project aims to innovate and create self-driving cars and artificial intelligence models that are specifically designed to function effectively within the local environment, demonstrating the capability to develop advanced autonomous systems domestic.

## What Did We Try to Solve?

- We aim to develop local alternative solutions to replace similar foreign products while contributing to environmental sustainability in the future. Our goal is to help reduce pollution by minimizing the reliance on fuel and oil combustion, which significantly contributes to air contamination through vehicle emissions.

## What Did We Learn?

- **Python Programming Language and Algorithms**
- **Embedded and IOT**
- **Communication Protocols**
- **AutoCAD and Designing**

## Implemented With:

- **Laptop**
- **Car Chassis**
- **Customized Map**

## Economy:

- We have demonstrated that Egyptian industrial engineering possesses the capability to manufacture self-driving vehicles and develop advanced artificial intelligence models that can compete with the global industry. Moreover, these models are designed to adapt efficiently to both local and diverse environmental conditions with a high degree of effectiveness.

## 1-3: Project Flow Diagram:

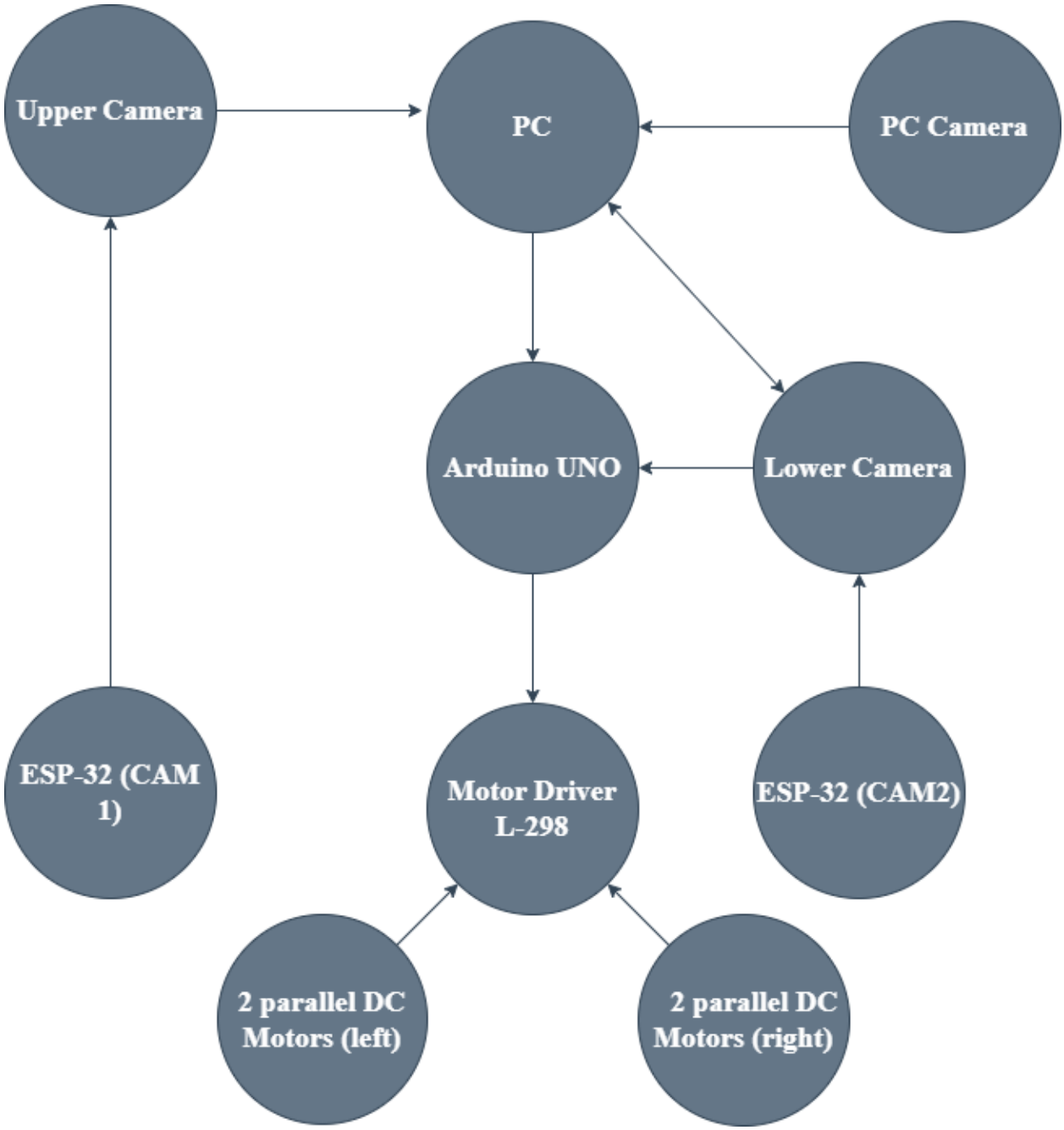



**Figure 1-1: Project Flow Diagram**

# Chapter [2]:
# Scientific Background

## Self -Driving Cars Scientific History:

The development of the self-driving car, more formally known as an **autonomous vehicle** (AV), is a complex and multifaceted endeavor spanning over a century of research and development. Its history can be broadly categorized into distinct phases, each marked by significant technological advancements.

**Phase 1**: Conceptualization and Early Experiments (Early 20th Century - 1970s)

The initial seeds of the autonomous vehicle concept were sown in the early 20th century, with science fiction writers and inventors envisioning vehicles capable of navigating without human intervention. However, true scientific exploration began in the mid-20th century. Early experiments focused on rudimentary forms of automation, such as guided vehicles following tracks or wires embedded in the road. Key milestones during this period include:

1920s: Early concepts of radio-controlled cars and automated guidance systems.

1950s: Research into automated highway systems and the development of the first experimental autonomous car by General Motors.

1960s: Continued exploration of automated highway systems and the development of early computer vision systems.

**Phase 2**: Prototyping and Technological Foundations (1980s - 2000s)

This phase witnessed the development of the fundamental technologies that underpin modern AVs. Researchers began integrating sensors, computers, and control systems into vehicles, enabling them to perceive their environment and make basic driving decisions. Key advancements during this period include:

1980s: Development of the first self-sufficient and truly autonomous cars by Carnegie Mellon University (Navlab and ALV projects) and Mercedes-Benz. These vehicles demonstrated the feasibility of autonomous navigation, albeit in controlled environments.

1990s: Focus on improving sensor technology, particularly computer vision, and the development of sophisticated algorithms for path planning and decision-making.

2000s: Increased computational power and the emergence of machine learning techniques, laying the groundwork for more advanced autonomous capabilities.

**Phase 3**: Rapid Advancement and Real-World Testing (2010s - Present)

The past decade has seen an exponential acceleration in AV development, driven by advancements in artificial intelligence, sensor technology, and computing power. Companies like Google, Tesla, and numerous automotive manufacturers have invested heavily in AV research and development, leading to the deployment of prototype vehicles on public roads for testing and data collection. Key characteristics of this phase include:

Emphasis on Deep Learning: Deep learning algorithms have revolutionized computer vision and perception, enabling AVs to accurately interpret their surroundings.

Sensor Fusion: AVs utilize a suite of sensors, including cameras, lidar, radar, and ultrasonic sensors, to create a comprehensive understanding of their environment. Sensor fusion algorithms combine data from these diverse sources to enhance accuracy and robustness.

## 2-1: Introduction:

In the field of embedded systems and artificial intelligence, programming languages and algorithms play a crucial role in enabling automation, object detection, and real-time decision-making. Among the widely used programming languages, **Python** and **Arduino C** stand out due to their versatility and efficiency in handling complex computations and hardware interactions.

## 2-2: Algorithms:

### 1. Python Programming Language

Python is a high-level, general-purpose programming language known for its simplicity and readability. It is extensively used in artificial intelligence, machine learning, and computer vision due to its vast ecosystem of libraries. One of the most powerful libraries for image processing and computer vision in Python is **OpenCV (cv2)**, which enables real-time image analysis, object recognition, and feature extraction. Python's flexibility and its ability to integrate with deep learning frameworks make it a preferred choice for developing self-driving systems and AI-based applications.

### 2. Arduino C Programming Language

Arduino C is a variant of the C/C++ programming language specifically designed for microcontroller programming. It is widely used in embedded systems and robotics due to its efficiency in managing hardware components such as **sensors, motors, and communication modules**. In self-driving car models, Arduino C is used to program microcontrollers such as the **Arduino Uno**, enabling them to process sensor data, control motor drivers, and communicate with AI-based modules for autonomous navigation.

## 2-2-0: Computer Vision and AI Algorithms:

To enhance autonomous decision-making, various computer vision algorithms are integrated into self-driving models:

- **2.2.1: OpenCV (cv2):** A powerful open-source computer vision library that provides tools for image and video processing, object detection, and facial recognition. OpenCV enables real-time analysis of the surrounding environment, which is crucial for self-driving vehicles.
- **2.2.2: Dlib:** A machine learning library specialized in **face detection, facial landmark recognition, and object tracking**. It is widely used in driver monitoring systems to detect **eye movements, drowsiness, and facial expressions**, ensuring safety in autonomous vehicles.
- **2.2.3: YOLO (You Only Look Once):** An advanced deep learning-based object detection algorithm known for its real-time performance and high accuracy. YOLO enables the vehicle to detect pedestrians, traffic signs, and obstacles efficiently, contributing to improved situational awareness in self-driving applications.

- **2.2.4: Image Moment Algorithm:** A mathematical approach used in **object shape analysis and feature extraction**. In self-driving vehicles, Image Moment helps in lane detection and road boundary identification by analyzing the geometric properties of objects in an image.

## Algorithms Diagram:

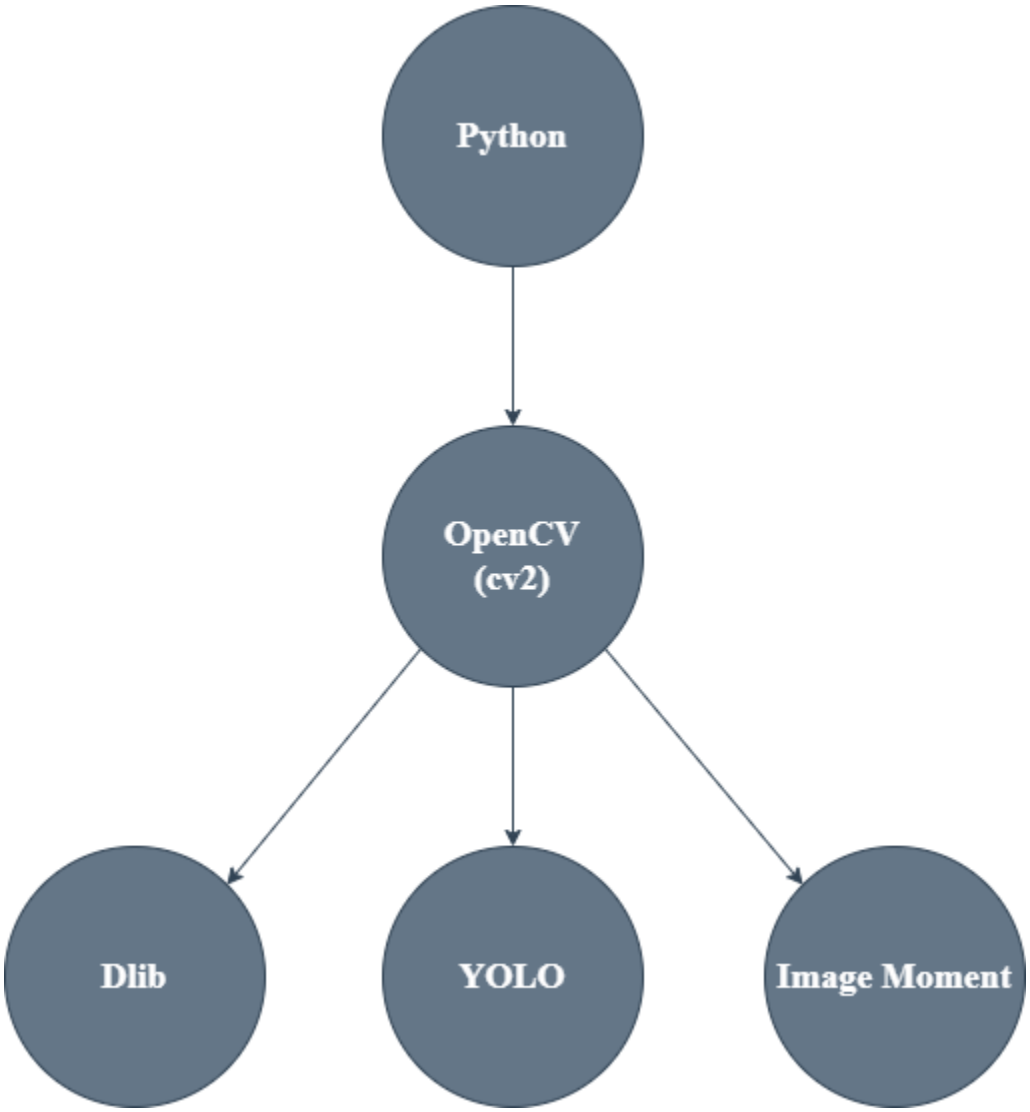


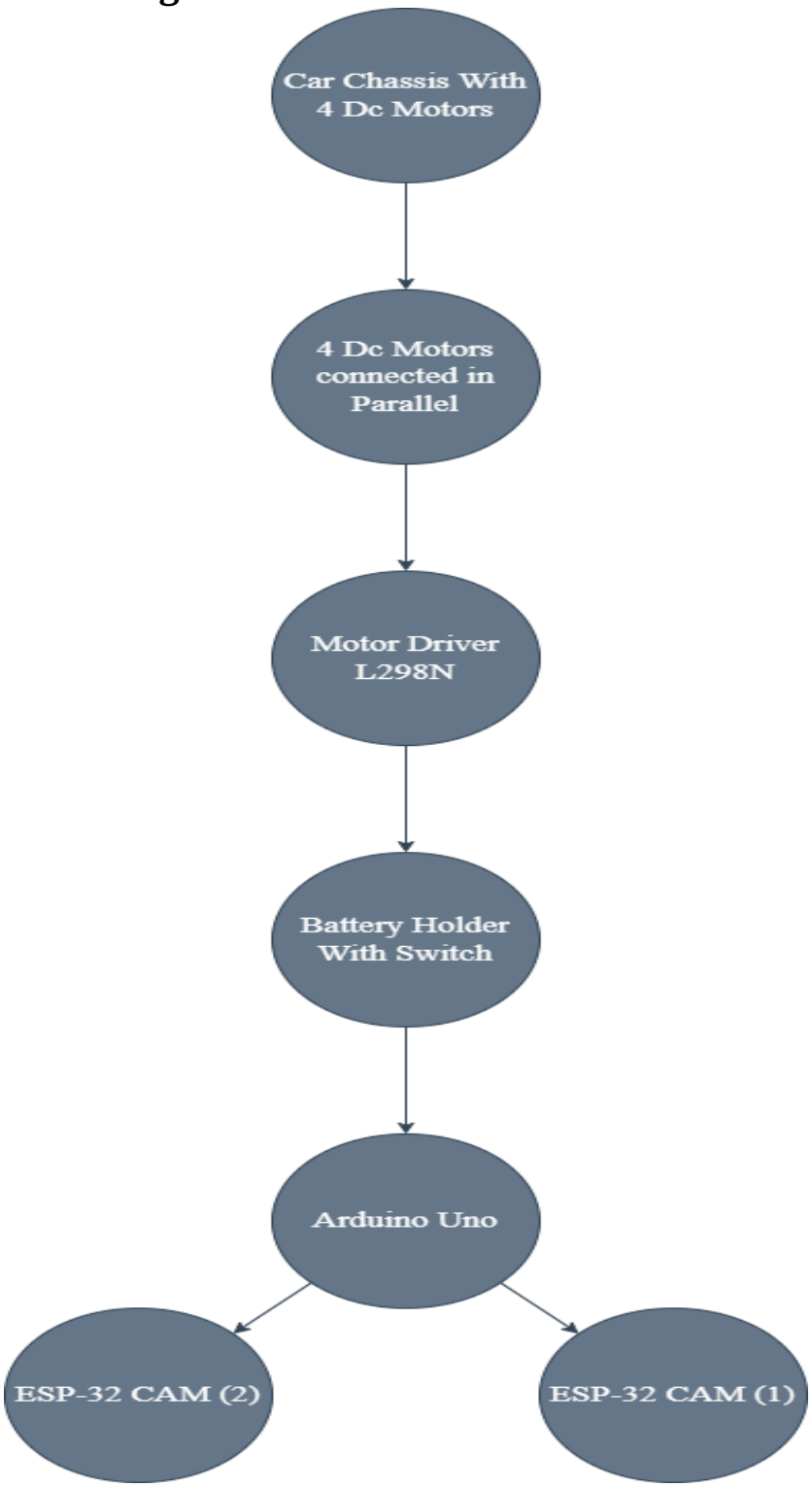

**Figure 2-1: Algorithms Diagram**

By integrating **Python for AI processing, Arduino C for hardware control, and advanced computer vision algorithms**, self-driving vehicle models can achieve high levels of automation and adaptability. These technologies collectively enable vehicles to perceive their environment, make intelligent decisions, and operate efficiently under various conditions, paving the way for advancements in autonomous transportation.

# Chapter [3]:

# System Hard-Ware

## 3-1: Hard-Ware Diagram:



## 3-2: Hard-Ware Subsystems:

- **3.2.1:** Car Chassis:

| Component | Specification |
|---|---|
| Chassis Type | Four-Wheel Car Chassis for Robots |
| Size of Each Board | 179cm x 109cm x 3mm |
| Distance Between Boards | 24 mm |

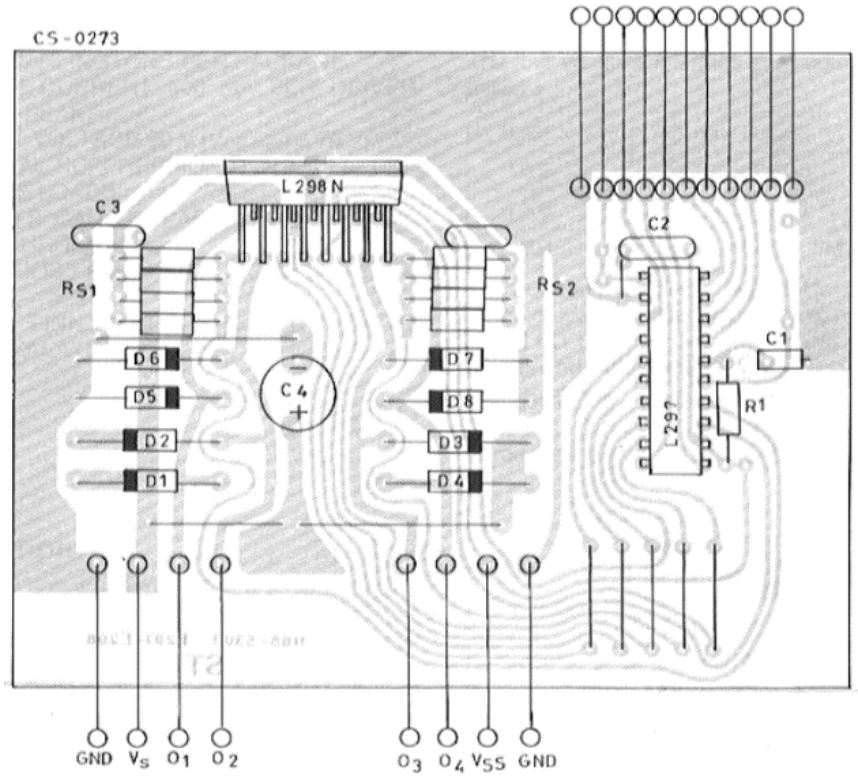
- **3.2.2: DC-Motors:**

| Component | Specification |
|---|---|
| Motor Type | 130-Size Brushed DC-Motor |
| Operating Voltage | 3~12 volts |
| Size | 25mm x 15mm x 20mm |
| Weight | 18g |
| Shaft Diameter | 2mm |
| No-Load Speed at 6 volts | 11500 RPM |
| No-Load Current at 6 volts | 70 mA |
| Stall Current at 6 volts | 800 mA |

- **3.2.3: Motor Driver L298:**

The L298 is an integrated monolithic circuit, it is a high-voltage, high-current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as: Relays, Solenoids, Dc-Motor, Stepper-Motor, it contains two H-bridges, allowing to control two motors independently or one step motor.

| Specification | Description |
|---|---|
| Operating Voltage | Up to 46 volts |
| Logic Supply Voltage | Up to 7 volts |
| Logic Input Voltage | -0.3V to 7V |
| Peak Output Current | 2 A (maximum/channel) |
| Operating Temperature | 0°C to 70°C |



- ### 3.2.4: Rocker Switch:

The rocker switch is a type of electrical switch that's activated by rocking a lever or button back and forth, it is a small seesaw-like mechanism, when you press one side it tilts the mechanism to close the electrical circuit.

| Specification | Description |
|---|---|
| Rated Voltage | 125 volts AC, 250 volts AC |
| Rated Current | 10 A, 6 A |
| Insulated Resistance | >= 100 MΩ |
| Contact Resistance | <= 50 mΩ |
| Endurance | >= 10,000 cycles |

- ### 3.2.5: Battery Holder:

This battery holder is designed to hold securely the lithium-ion batteries connected in series, connecting the batteries in series increases the voltage.

| Specification | Description |
|---|---|
| Battery Type | 18650 |
| Number of Cells | 3 cells |
| Connection | Series |
| Dimensions | 75mm x 61mm x 21mm |
| Weight | 21.5g |

- ### 3.2.6: Batteries:

It is a high discharged performance battery; it is designed for applications demanding bursts of high current.

| Specification | Description |
|---|---|
| Battery Type | 18650 |
| Battery Voltage | 4.2 volts |
| Battery Current | 8800 mAh |
| Weight | 160g |
| Size | 32mm x 65mm |
| Cycle Life | More than 2000 times |

- ### 3.2.7: Arduino-Uno:

Arduino-Uno is a microcontroller board based on the ATmega328P; it has a 14-digital input/output pins (6 of them 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.

| Specification | Description |
|---|---|
| **Microcontroller** | ATmega328P |
| **Operating Voltage** | 5 volts |
| **Input Voltage** | 7~12 volts |
| **DC Current** | 20mA |
| **Memory** | 2KB SRAM, 32KB FLASH, 1KB EEPROM |
| **Clock Speed** | 16 MHz |

| | |
|---|---|
| **Dimensions** | 6.86mm x 53.4mm |
| **Weight** | 25g |

- **Circuit:**

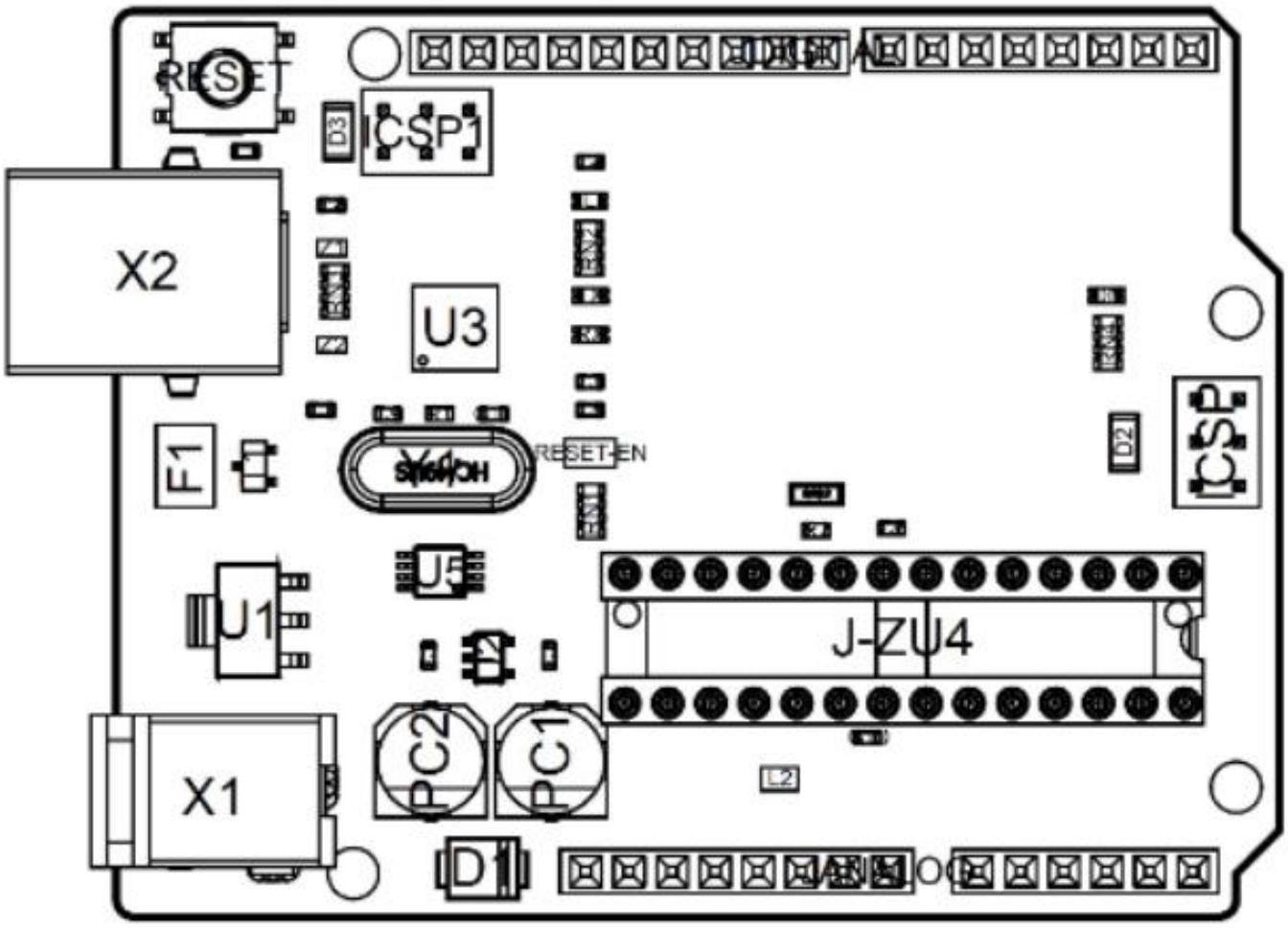



**Figure 3-3: Arduino Uno**

- **3.2.8: Esp-32 Cam:**

The Esp-32 Cam is a mighty module that combines a powerful microcontroller with a camera, making it perfect for a wide range project

| Specification | Description |
|---|---|
| Microcontroller | ESP32 (Dual-Core, up to 240 MHz) |
| Camera | 2 MP OV2640 |
| Memory | 520 KB RAM, 4 MB PSRAM |
| Storage | Micro SD-Card (up to 4 GB) |
| Connectivity | Wi-Fi 802.11 b/g/n, Bluetooth 4.2 |
| Power | 5 volts |
| Size | 27mm x 40mm |

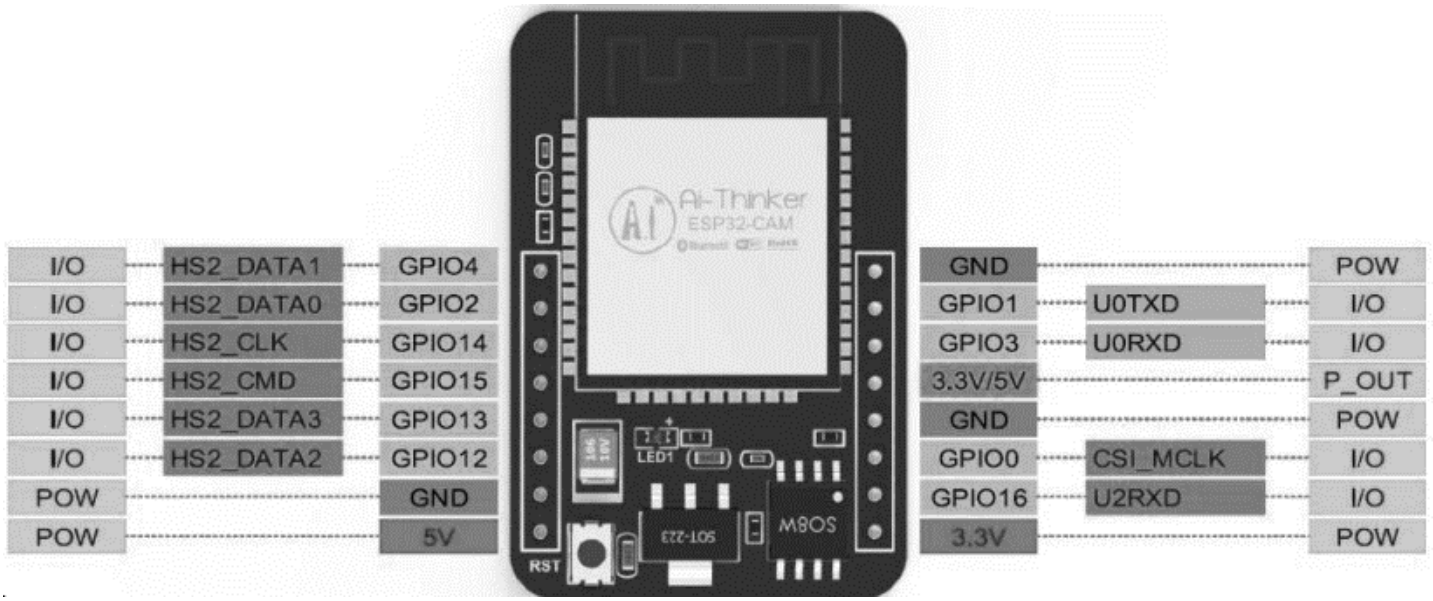
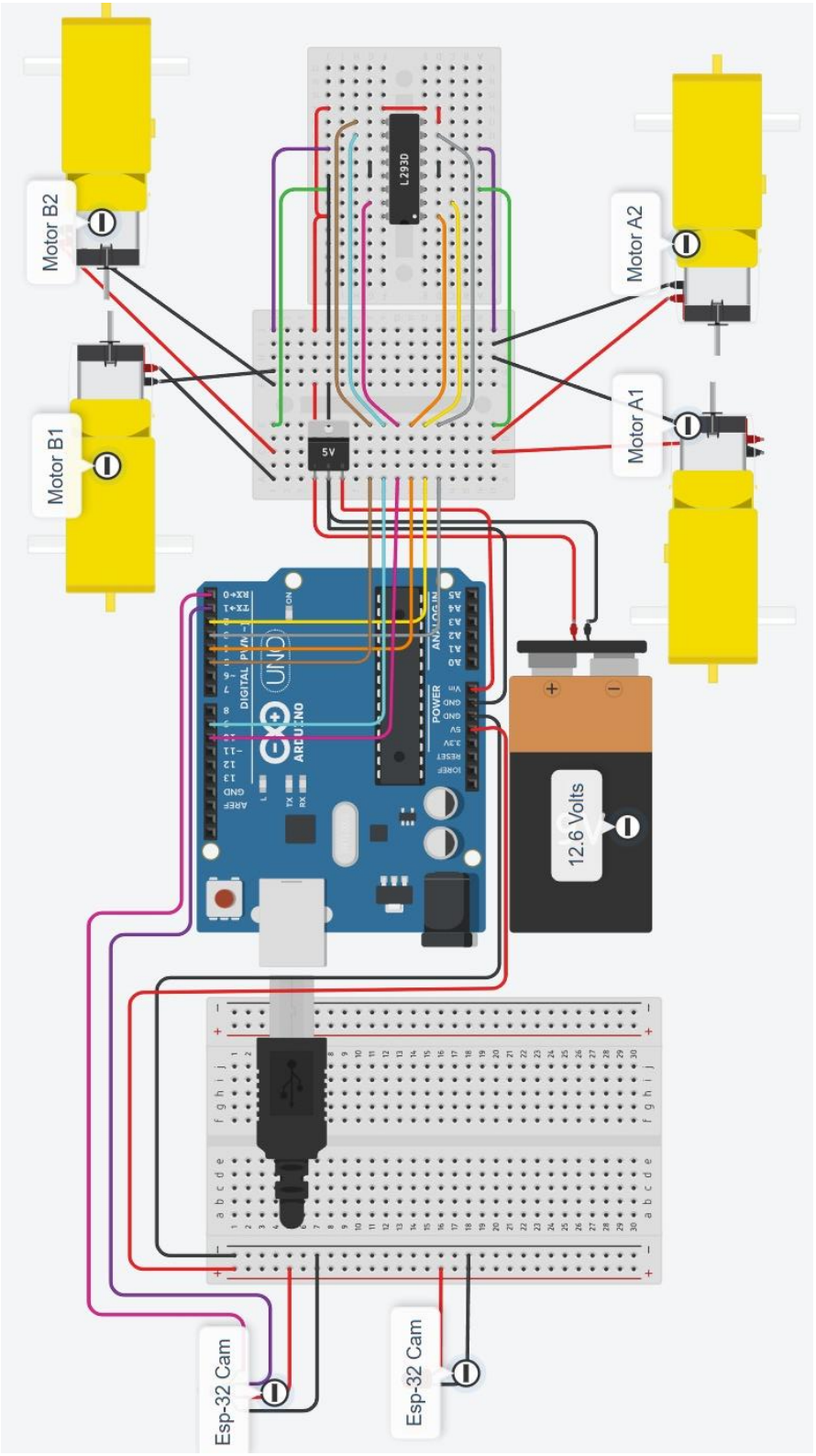


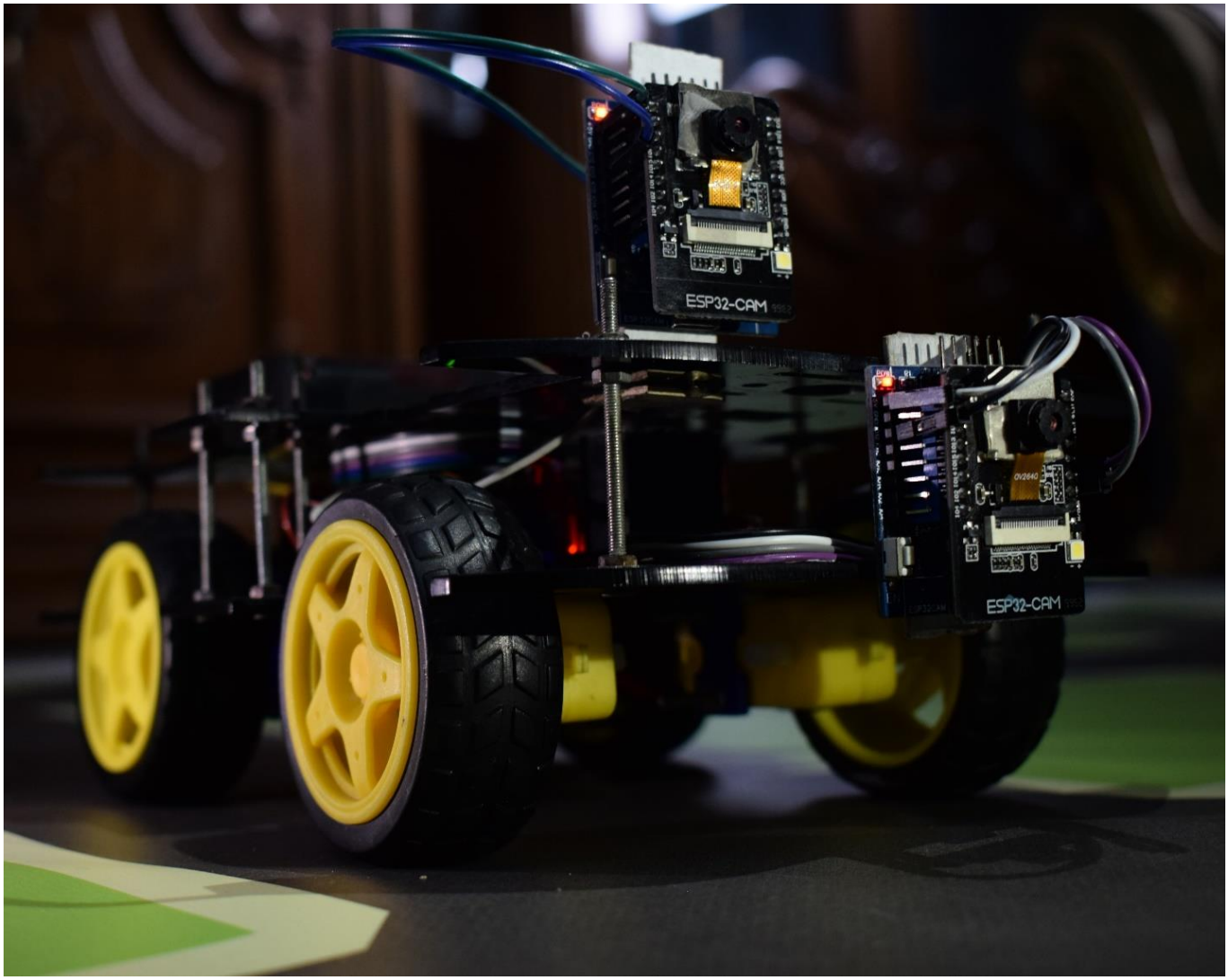

**Figure 3-4: Esp-32 Cam**

# 3-3: Complete System:

- ### 3.3.1: Wiring Diagram:



- **3.3.2: Final System Photos:**



## 3-4: Map Design:

The **self-driving car navigation map** serves as a foundational component in autonomous vehicle systems, providing a structured visualization of the designated routes and destinations. This map outlines the three predefined destinations, ensuring that the vehicle can navigate efficiently while adapting to the surrounding environment inspired from iconic race tracks.

To achieve a precise and professional representation, the design was meticulously crafted using **Adobe Illustrator** and **AutoCAD**—two most powerful tools for digital design and engineering visualization.

- **AutoCAD:** a leading software in computer-aided design (CAD), was utilized for drafting and technical detailing. Its precision-oriented features enabled the creation of an accurate layout, ensuring that the self-driving system operates within well-defined parameters.

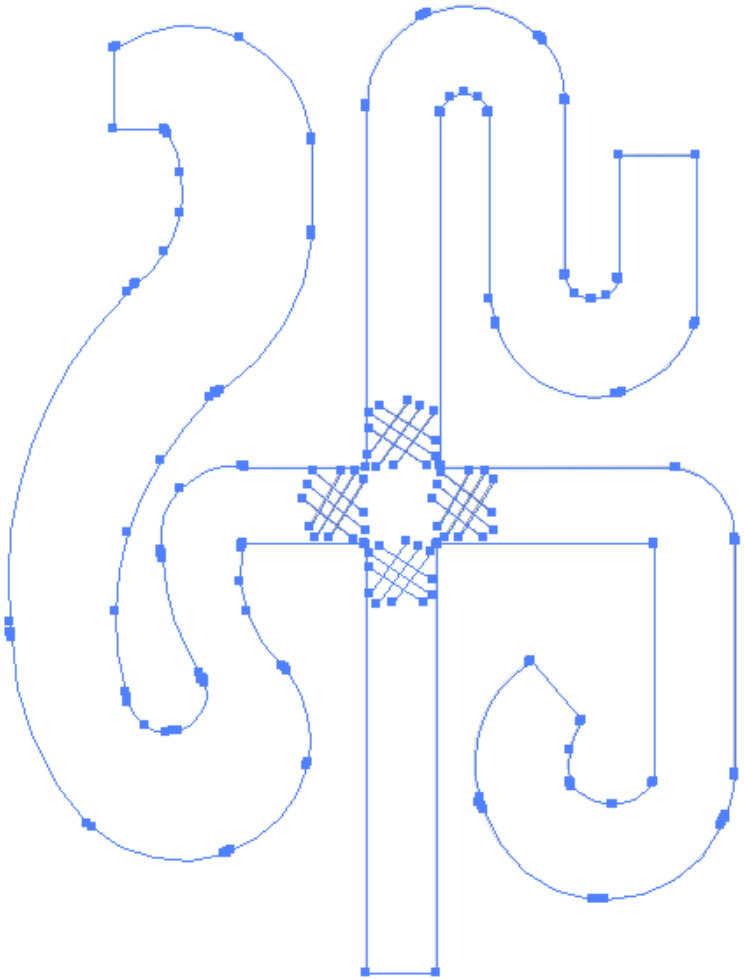


**Figure 3-6: AutoCADMasking**

**Adobe Illustrator:** was employed for creating a high-resolution, scalable vector representation of the map, ensuring clarity, accuracy, and a visually polished layout. This tool allowed for detailed graphical elements, such as road markings, waypoints, and structural components, to be seamlessly integrated into the design.

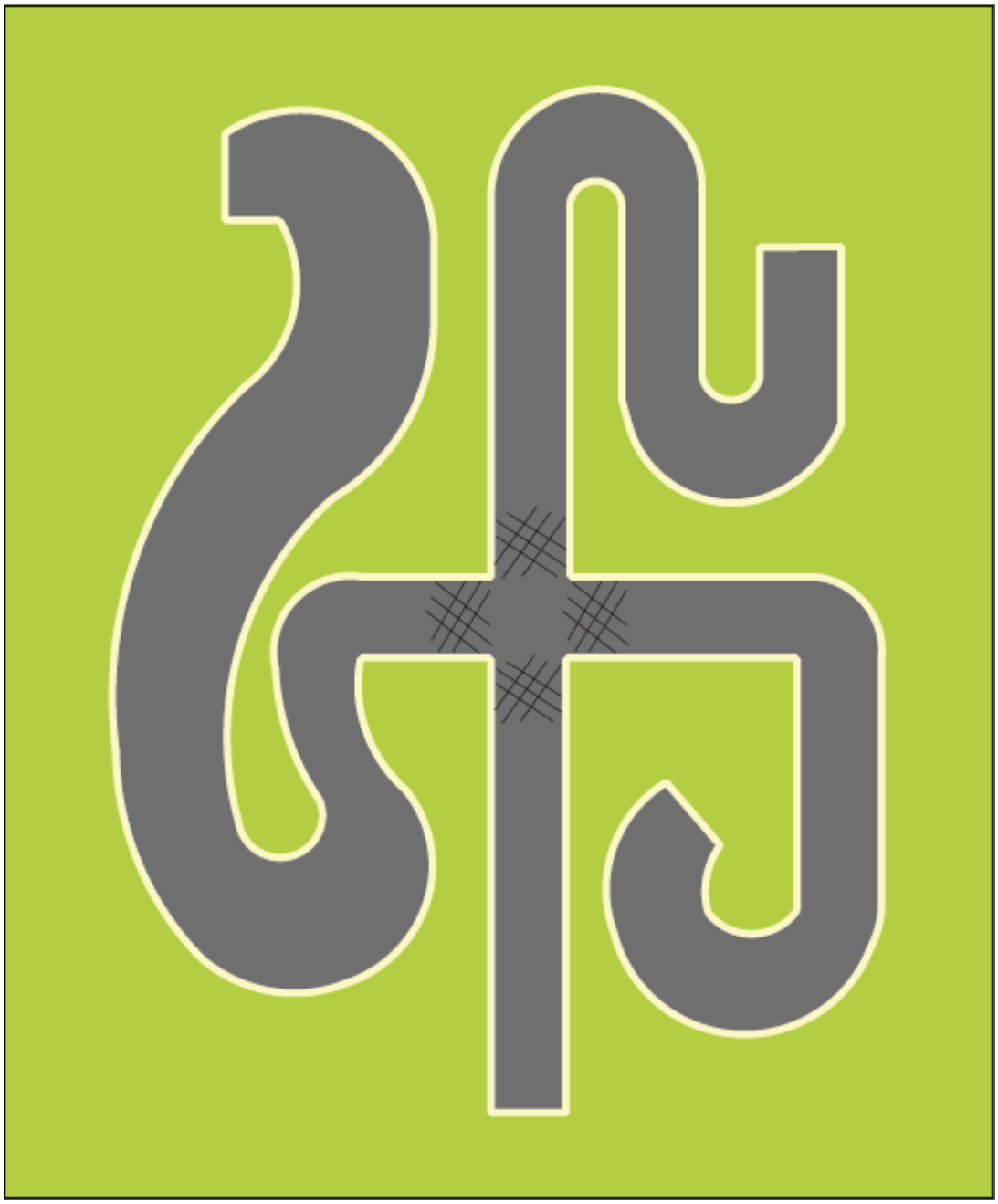


**Figure 3-7: Adobe Illustrator Final Design**

By leveraging these tools, the **navigation map provides an essential guide for the self-driving system**, allowing it to interpret routes, make autonomous decisions, and navigate through the designated areas efficiently.

# Chapter [4]:
# Embedded Soft-Ware

## 4-1: Soft-Ware Data Flow Diagram:

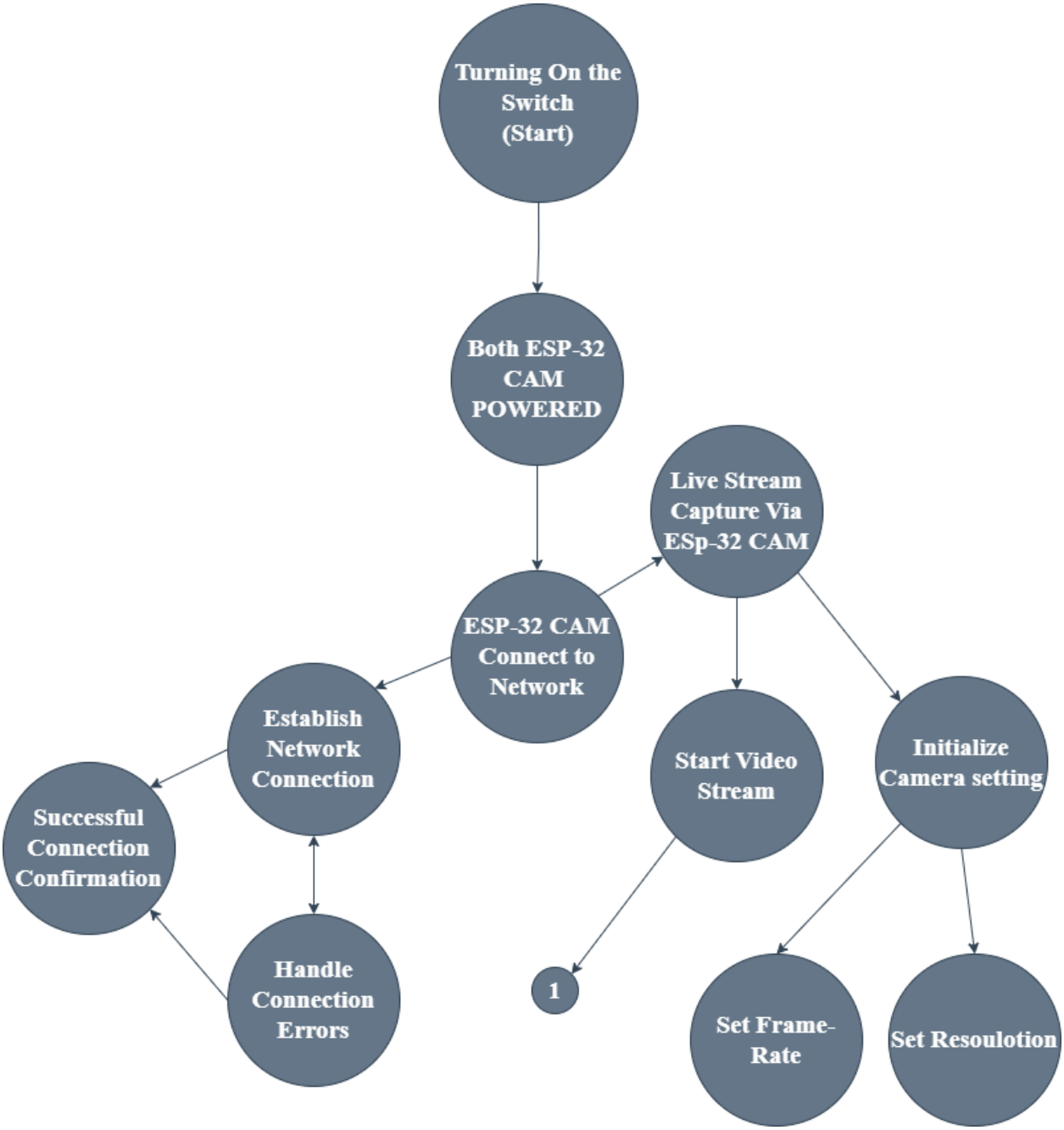



**Figure 4-1: Embedded Flow Diagram part [1]**

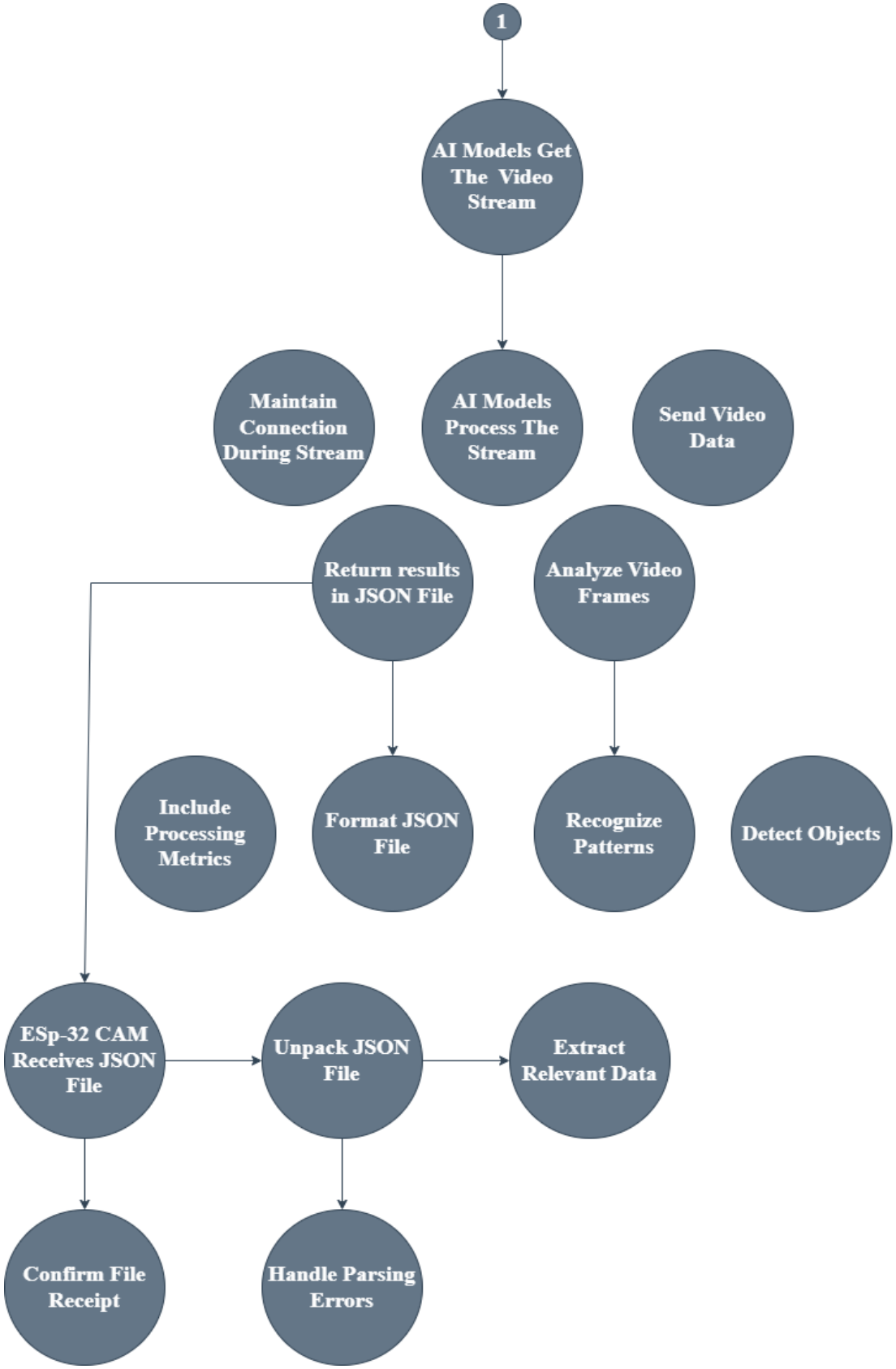



**Figure 4-2: Embedded Flow Diagram part [2]**

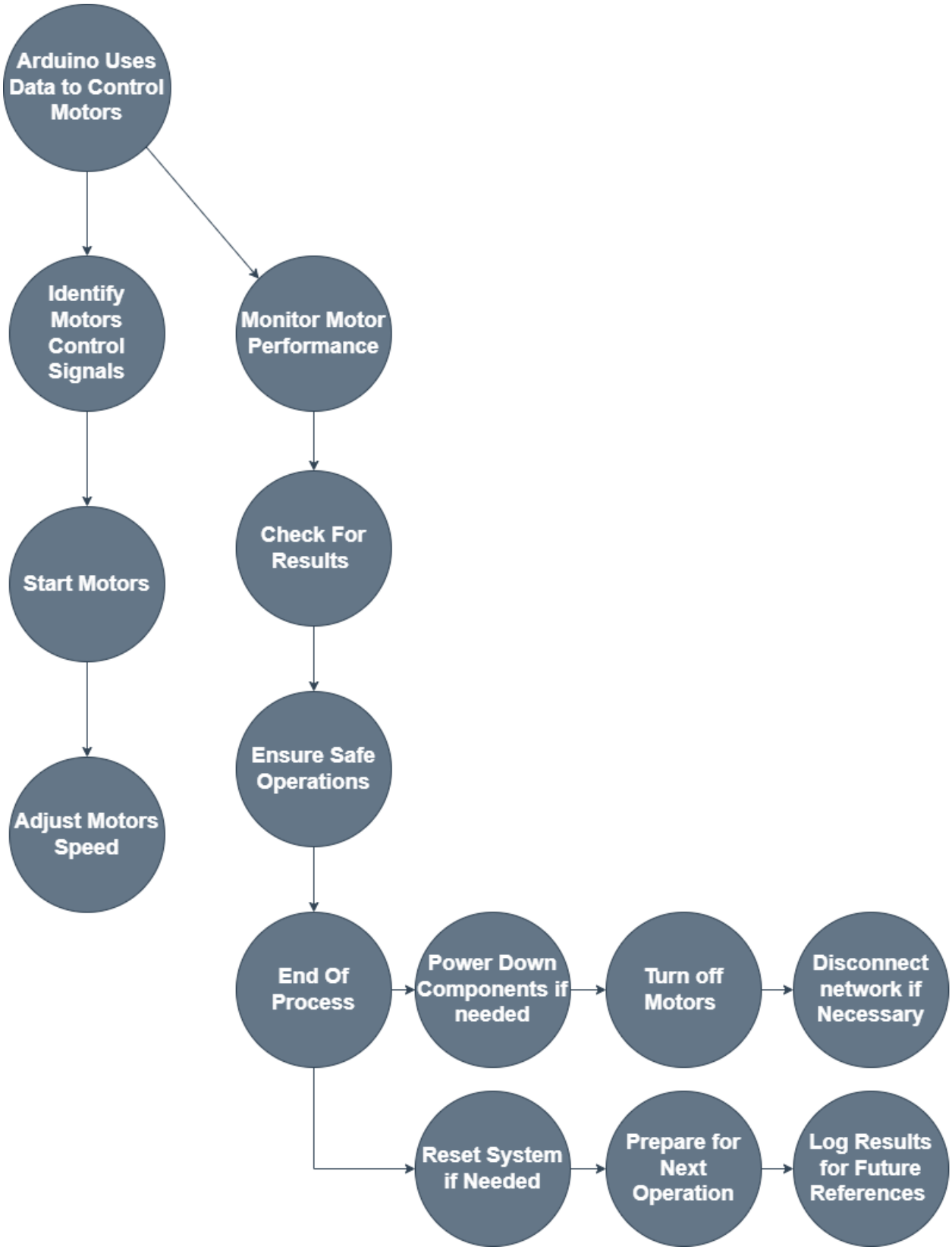



**Figure 4-3: Embedded Flow Diagram Phase [2]**

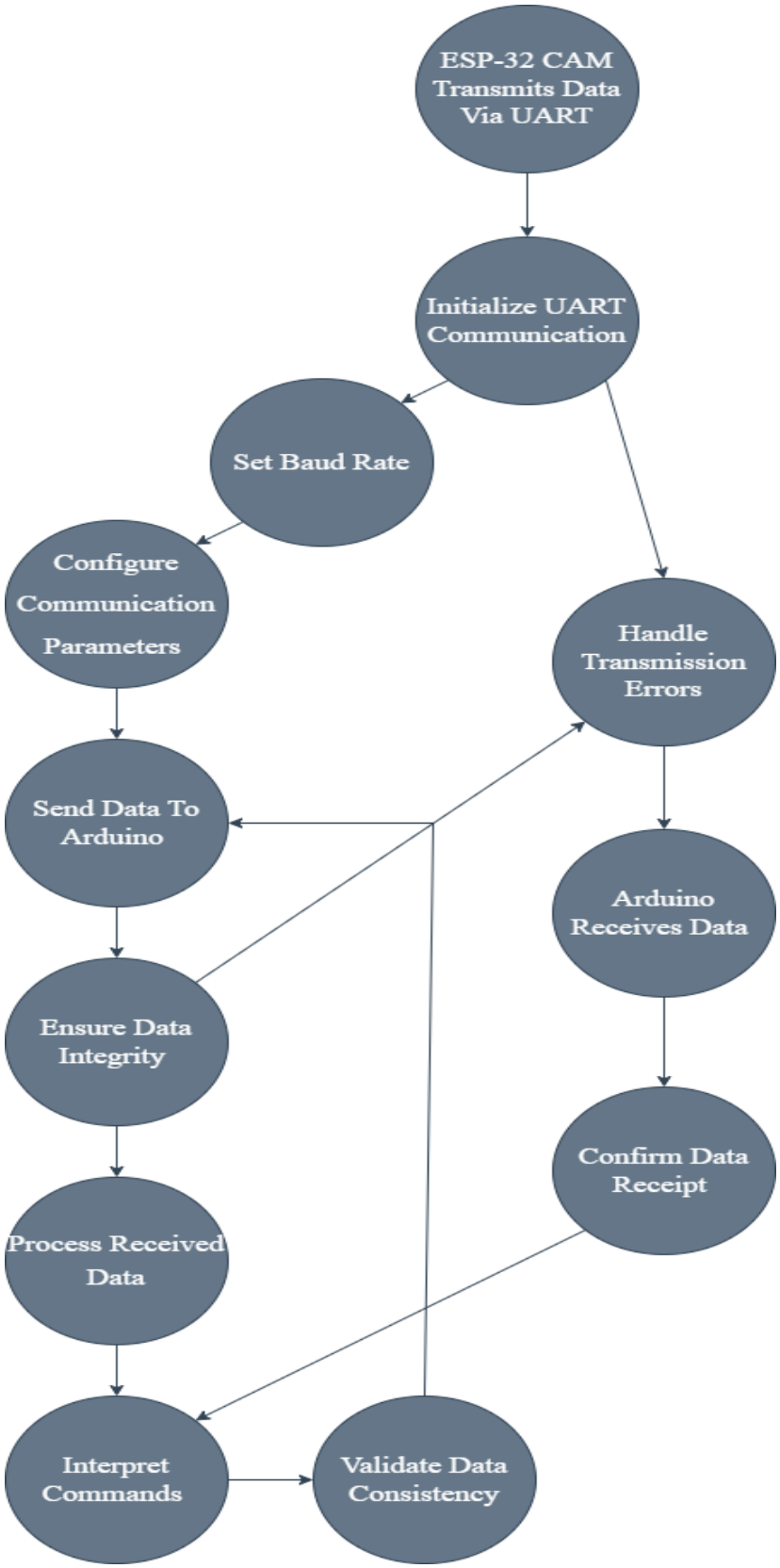
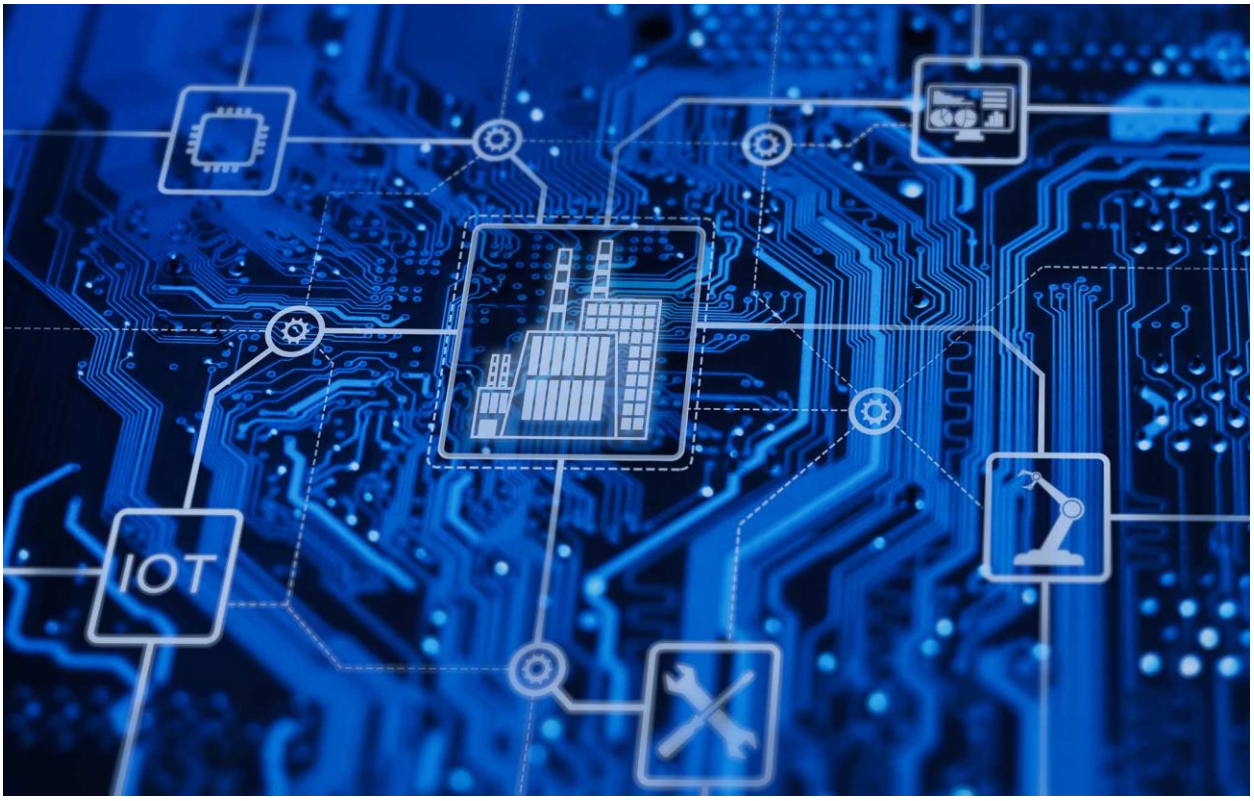



**Figure 4-2: Embedded Flow Diagram Phase [3]**

- **4-2: IOT Definition:**

It is a physical device, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and network connectivity, which enables these objects to collect and exchange data without human-to-human or human-to-computer interaction.




- 4-2-0: Embedded Definition:

  It is a system of interconnected, physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and network connectivity, which enables these objects to collect and exchange data without human-to-human or human-to-computer interaction.

- 4-2-1: Main Functions Description:

  - 4-2-1: First Esp-32:

    Includes the Necessary Libraries, Define Camera Model, Setting up Wi-Fi Credentials:

```
#include "esp_camera.h" // For camera functions
#include <WiFi.h>     // For WiFi connection

#define CAMERA_MODEL_AI_THINKER // For creating a web server
#include "camera_pins.h" // Include camera pin definitions

// WiFi Configuration
const char *ssid = "********"; // Your WiFi SSID
const char *password = "*******"; //Your WiFi password
```

**Esp-32Cam Configuration's**

  - 4.2.1.2: Configure Frame Size and Pixel Format:

This part is essential for streaming.

```
config.frame_size = FRAMESIZE_UXGA; // Frame size
config.pixel_format = PIXFORMAT_JPEG;  // JPEG format for streaming
```

**Esp-32Cam Window Format**

- 4.2.1.3: Configure Stream Quality:

This sets the JPEG compression quality.

```
config.jpeg_quality = 12; // JPEG quality
```

**Esp-32Cam Stream Quality**

- 4.2.1.4: Camera Error Handling:

It checks if the camera was initialized successfully and stops the program if it failed.

```
// camera initialize
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) { // Error handling
  Serial.printf("Camera init failed with error 0x%x", err);
  return;
}
```

**Esp-32Cam Error Handling**

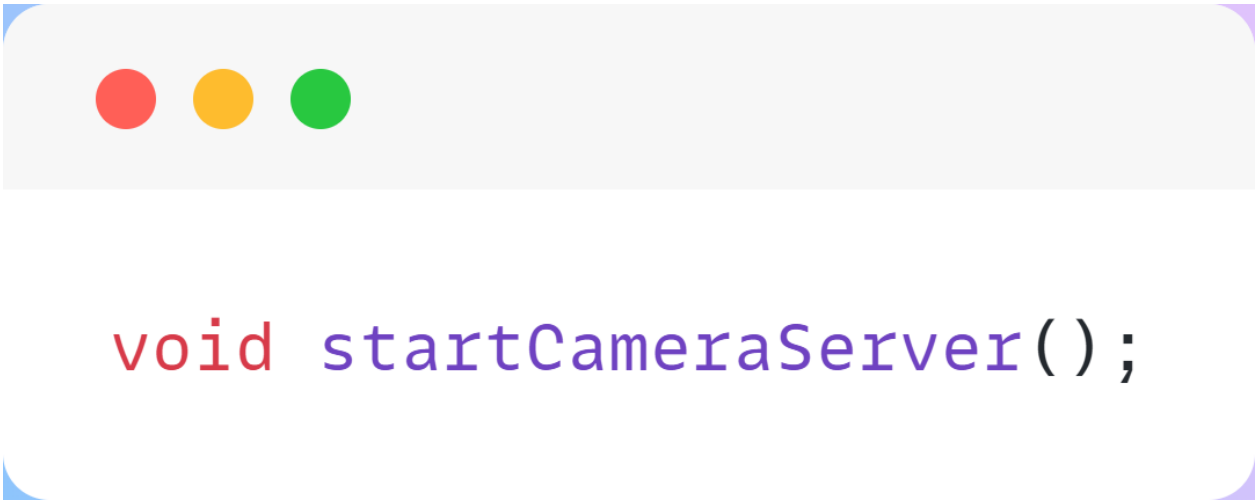
- 4.2.1.5: Network Connection Check:

  This loop waits until Esp-32 Cam successful connects to the Network.

```
while (WiFi.status() =≠ WL_CONNECTED) { // Wait for connection
  delay(500);
  Serial.print(".");
}
```

**Esp-32Cam Network Connection Check**

- 4.2.1.6: Start Camera Web Server:
  This function starts the web server that will allow you to access the camera stream from the web browser.




**Esp-32Cam Starting Server**

- 4.2.1.7: URL Printing:

This prints the URL you need to enter in your web browser to view the camera stream.

```
  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP()); //Prints the IP address
  Serial.println("' to connect");
}
```

**Esp-32Cam URL**

- **4.2.2: Second Esp-32:**

4.2.2.1: Includes the Necessary Libraries, Define Camera Model, Setting up Wi-Fi Credentials:

```
#include "esp_camera.h"
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

// WiFi Configuration
const char* ssid = "family";
const char* password = "a                                        .y";

// UART Configuration (Changed pins to avoid conflict with camera)
#define UART_BAUD 115200
#define UART_TX_PIN 1  // Changed from 17
#define UART_RX_PIN 3  // Changed from 16

WebServer server(80);
```

- 4.2.2.2: Setup Camera End Points:

This is the most important part of viewing the camera feed and its crucial for streaming.

```
void setupCameraEndpoints() {
  server.on("/capture", HTTP_GET, []() {
    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) {
      server.send(500, "text/plain", "Camera capture failed");
      return;
    }
    server.sendHeader("Content-Type", "image/jpeg");
    server.sendHeader("Content-Length", String(fb->len));
    server.send(200);
    server.sendContent((const char*)fb->buf, fb->len);
    esp_camera_fb_return(fb);
  });

  server.on("/stream", HTTP_GET, []() {
    WiFiClient client = server.client();
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: multipart/x-mixed-replace; boundary=frame");
    client.println();

    while (client.connected()) {
      camera_fb_t* fb = esp_camera_fb_get();
      if (!fb) {
        Serial.println("Camera capture failed");
        break;
      }

      client.println("--frame");
      client.println("Content-Type: image/jpeg");
      client.printf("Content-Length: %d\r\n\r\n", fb->len);
      client.write(fb->buf, fb->len);
      esp_camera_fb_return(fb);
      delay(33);
    }
  });
}
```

**Esp-32Cam Camera End Points**

- 4.2.2.3: Data Handling:

  This is how the model data is received, parse the JSON data received in POST request, extract the relevant information (Lane Data, Traffic Sign, Driver State), create a new JSON file to be sent to the Arduino, serialize the JSON file and send it to Arduino over UART.

```
void handleData() {
  if (server.method() == HTTP_POST) { // Handles POST requests to /data
    String body = server.arg("plain"); // Get data from request body

    StaticJsonDocument doc; // JSON document for incoming data
    DeserializationError error = deserializeJson(doc, body); // Parse JSON

    if (error) { // Error handling
      server.send(400, "text/plain", "Invalid JSON");
      return;
    }

    // Extract data from JSON
    float lane_left = doc["lane_left"];
    float lane_right = doc["lane_right"];
    String driver_state = doc["driver_state"];
    String traffic_sign = doc["traffic_sign"];

    // Create JSON for Arduino
    StaticJsonDocument arduinoDoc;
    arduinoDoc["ll"] = lane_left;
    arduinoDoc["lr"] = lane_right;
    arduinoDoc["ds"] = driver_state;
    arduinoDoc["ts"] = traffic_sign;

    String output;
    serializeJson(arduinoDoc, output); // Serialize JSON
    Serial2.println(output);           // Send over UART

    server.send(200, "text/plain", "Data received"); // Confirmation
  }
}
```

**Esp-32Cam Data Handling**

- 4.2.2.4: Configure Frame Size and Pixel Format:

  This part is essential for streaming.

```
config.frame_size = FRAMESIZE_UXGA; // Frame size
config.pixel_format = PIXFORMAT_JPEG;  // JPEG format for streaming
```

**Esp-32Cam Window Format**

- 4.2.2.5: Configure Stream Quality:

  This sets the JPEG compression quality.

```
config.jpeg_quality = 12; // JPEG quality
```

**Esp-32Cam Stream Quality**

- 4.2.2.6: Camera Error Handling:

It checks if the camera was initialized successfully and stops the program if it failed.

```
// camera initialize
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) { // Error handling
  Serial.printf("Camera init failed with error 0x%x", err);
  return;
}
```

**Esp-32Cam Error Handling**

- 4.2.2.7: Network Connection Check:

  This loop waits until Esp-32 Cam successful connects to the Network.

```
while (WiFi.status() != WL_CONNECTED) { // Wait for connection
  delay(500);
  Serial.print(".");
}
```

**Esp-32Cam Network Check**

- 4.2.2.8: Configure Stream Server:

This function starts the web server that will allow you to access the camera stream from the web browser.

```
  // Server Configuration
  server.on("/data", HTTP_POST, handleData);
  setupCameraEndpoints();
  server.begin();

  Serial.print("Stream URL: http://");
  Serial.print(WiFi.localIP());
  Serial.println("/stream");
}
```

**Esp-32Cam Stream Server Configuration**

- 4.2.2.9: Client Handling:

  This function handles incoming stream web requests.

```
server.handleClient();
```

**Esp-32Cam Client Handler**

- **4.3.3: Arduino-Uno:**

- 4.3.3.1: Include Library:
  This line is essential for parsing the JSON data.

```
#include <ArduinoJson.h>
```

**Arduino Library Include**

- 4.3.3.2: Define Pins:

This line defines the connection to the motor driver, it must be matching the actual wiring.

```
// Your motor pins configuration
#define ena 10
#define in1 2
#define in2 3
#define in3 4
#define in4 5
#define enb 11
```

**Arduino PINS Define**

- 4.3.3.3: PID Control:

These are PID tuning parameters. PID control is used to make the robot follow the lane smoothly.

```
// PID parameters
float Kp = 0.5;
float Ki = 0.01;
float Kd = 0.1;
float prev_error = 0;
float integral = 0;
float baseSpeed = 80;  // Base speed for straight movement
```

- 4.3.3.4: Setup:

This line is crucial as it configure the motor control pins as OUTPUTS, without this the Arduino can't control the motors.

```
void setup() {
  Serial.begin(115200);

  // Initialize motor control pins
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(ena, OUTPUT);
  pinMode(enb, OUTPUT);

  stop();  // Initial stop
}
```

**Arduino Motor Configuration**

- 4.3.3.5: Serial Monitor Checking:

Checks if the serial data form the Esp-32 Cam is available or not. And reads the incoming serial until a new line character.

```
    if (Serial.available()) {
```

```
void processData(String jsonData) {
  StaticJsonDocument<200> doc;
  DeserializationError error = deserializeJson(doc, jsonData);

  if (error) {
    Serial.print("JSON Error: ");
    Serial.println(error.c_str());
    return;
  }
```

- 4.3.3.6: Processing Data:

  This function parses the JSON data. there is an error handling line to check if the JSON file is invalid and to prevent crashes.

**Arduino Data Processing**

- 4.3.3.7: Extracting Values:

These lines extract the model's outputs from the JSON file. It is also very important in robustness: if the file is missing a value, it gives it a default value.

```
// Extract values
 float left = doc["l"] | 0.0;
 float right = doc["r"] | 0.0;
 String driverState = doc["d"] | "awake";
 String trafficSign = doc["s"] | "";
```

- 4.3.3.8: Emergency Stop:

This is a safety stop: if the driver is "Sleeping" or the traffic sign is "Stop"; the motors is stopped.

```
// Emergency stop conditions
 if (driverState == "sleeping" || trafficSign == "Stop") {
   stop();
   return;
 }
```

**Emergency Stop**

- 4.3.3.9: Adjust Motors:

This function translates the steering value into motor commands. The if structure determines the motor actions (Left, Right) based on the steering value

**Arduino Motor Adjustment**

```
void adjustMotors(float steering) {
  if (steering > 0.1) { // Right turn needed
    right_turn();
    speed(baseSpeed + abs(steering * 20));
  }
  else if (steering < -0.1) { // Left turn needed
    left_turn();
    speed(baseSpeed + abs(steering * 20));
  }
  else { // Go straight
    forward();
    speed(baseSpeed);
  }
}
```

# Chapter [5]:

# Artificial Intelligence Soft-Ware

## 5-1 Programming Language Used

The primary programming language utilized for this AI system is **Python**. Python is considered the best language for artificial intelligence due to its extensive libraries, ease of use, and strong community support.

1. **Image Moments (for shape detection and lane recognition)**
2. **Dlib (for driver drowsiness detection using facial landmarks)**
3. **YOLO (for real-time object detection, including license plate recognition)**

Each of these algorithms serves a specific purpose in **computer vision and AI-based automation**, but together, they can form a **comprehensive intelligent system** for autonomous vehicles, road safety, and AI-assisted monitoring. Let's break them down:

## 5.2: Image Moments (Lane Detection):

### 5.2.1: Purpose:

Image Moments are statistical measures used in **image processing and computer vision** to describe the shape, orientation, and distribution of objects in an image. They help in **lane detection**, **object tracking**, and **feature extraction** in an AI system.

### 5.2.2 :How It Works:

- The algorithm calculates **zero-order (M00)** and **first-order (M10, M01)** moments to find the **area** and **centroid** of a shape.
- Used for **lane recognition in autonomous vehicles** by analyzing **binary image transformations** (black & white pixel regions).

Detects contours of objects and extracts **geometric information**, such as **rotation** and **position** in the image.

$$M_{00} = \sum x \sum y \quad I(x, y)$$

**Figure 5-1: Image Moment Equation**

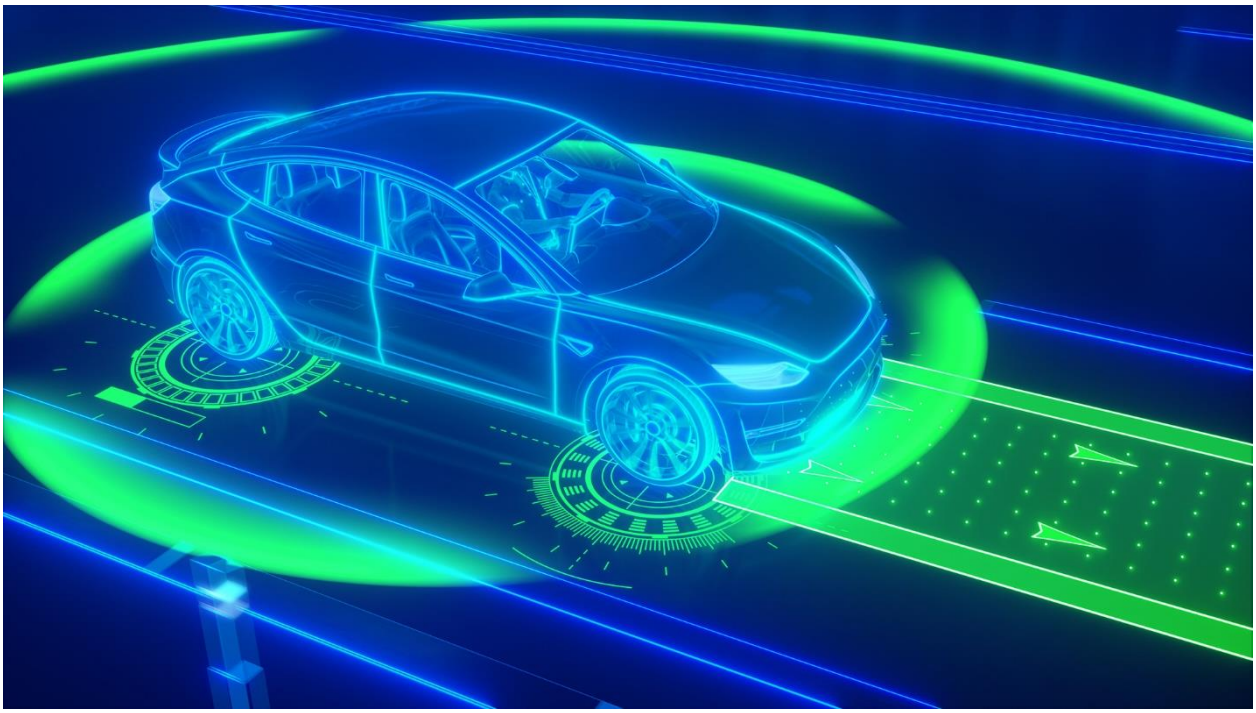
- **Area: In binary images, it is calculated as the zero-order moment M00, which equals the number of nonzero pixels.**
- **Centroid: It is computed using moments M10 and M01 divided by M00, giving the average position of nonzero pixels.**

$$\overline{X} = \frac{M_{01}}{M_{00}} = \frac{\sum\sum x \cdot I(x,y)}{\sum\sum I(x,y)}$$

$$\overline{Y} = \frac{\mathbf{M_{01}}}{\mathbf{M_{00}}} = \frac{\mathbf{1 + 2 + 3 + 4 + 1 + 2 + 3 + 4}}{\mathbf{8}} = \frac{\mathbf{20}}{\mathbf{8}}$$

**Figure 5-2: Image Moment Equation**

**Using these moments, we can extract important geometric information such as orientation and shape, aiding in improved shape recognition in image processing.**



## 5-2: Functions Description:

### 5.2.1: nothing(x):

- **Purpose: Placeholder function for trackbar callbacks.**

- **Why? OpenCV requires a function callback when creating a trackbar, even if it does nothing.**

**5.2.2: cv2.getTrackbarPos:(trackbar_name, window_name):**

- **Purpose: Retrieves the current value of a trackbar.**
- **Usage: Used to dynamically adjust HSV threshold values for image processing.**

**5.2.3: cv2.getPerspectiveTransform(pts1, pts2):**

- **Purpose: Computes the perspective transformation matrix.**
- **Usage: Used to warp the input frame to a top-down (bird's-eye) view for lane detection.**

**5.2.4: cv2.warpPerspective(image, matrix, (width, height)):**

- **Purpose: Applies a perspective transformation to an image.**
- **Usage: Used twice:**
    - **Once to transform the input image for lane detection.**
    - **Once to inverse-transform the detected lane back onto the original image.**

**5.2.5: cv2.inRange(hsv, lower, upper):**

- **Purpose: Creates a binary mask by filtering pixels within a specified HSV color range.**
- **Usage: Used to detect lanes by thresholding the transformed image.**

**5.2.6: cv2.findContours(image, mode, method):**

- **Purpose: Detects contours (boundaries) in a binary image.**

- **Usage: Used to locate lane positions in small sections of the frame.**

**5.2.7: cv2.moments(contour):**

- **Purpose: Computes spatial moments of a contour to find its centroid.**
- **Usage: Used to refine the lane center positions dynamically.**

**5.2.8: cv2.fillPoly(image, [points], color):**

- **Purpose: Draws and fills a polygon on an image.**
- **Usage: Used to overlay detected lane markings on the transformed frame.**

**5.2.9: cv2.addWeighted(src1, alpha, src2, beta, gamma):**

- **Purpose: Blends two images together using weighted transparency.**
- **Usage: Used to overlay lane detection results onto the original frame.**

**5.2.10: cv2.imshow(window_name, image):**

- **Purpose: Displays an image in a window.**
- **Usage: Shows various stages of processing (e.g., transformed view, lane mask, final result).**

**5.3.11: cv2.waitKey(delay):**

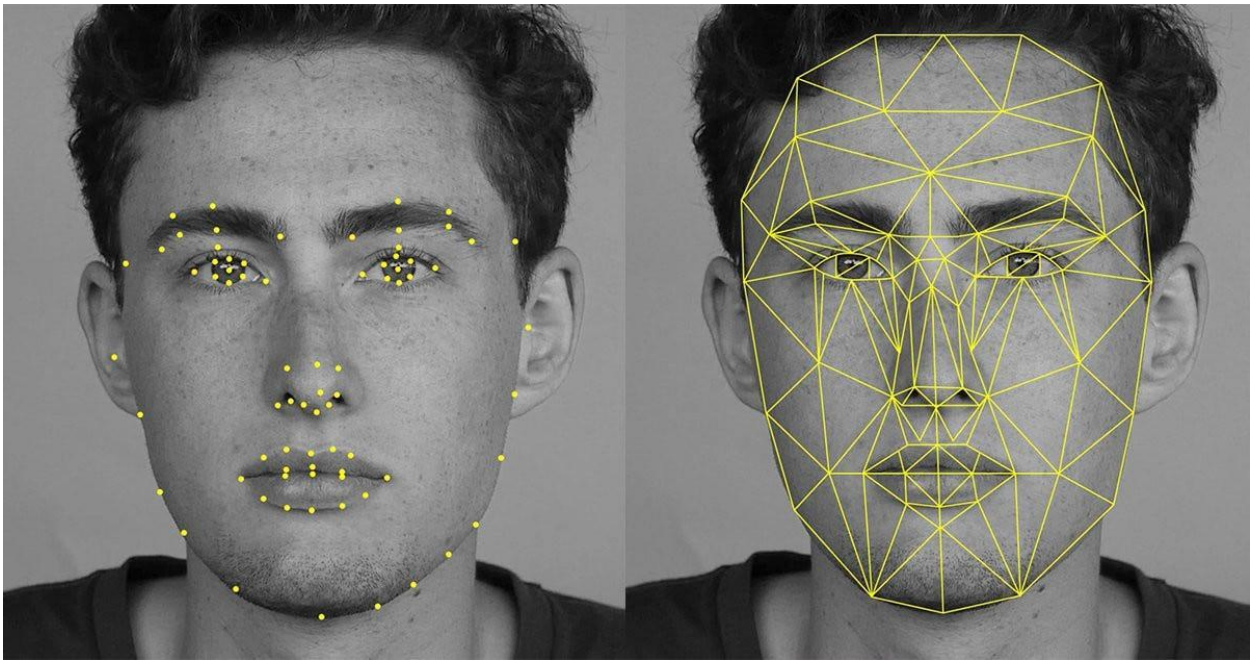
- **Purpose: Waits for a key event for a specified delay.**
- **Usage: Exits the loop if the Esc key is pressed.**

**5.2.12: time.sleep(seconds):**

- **Purpose: Pauses execution for a fixed duration.**

- **Usage: Used to maintain a consistent frame rate for real-time processing.**

## 5-3: Dlib: Facial Landmark Detection and Machine Learning:



Dlib is an advanced open-source library that provides powerful machine learning algorithms for **computer vision**, **facial recognition**, and **object tracking**. Developed in **C++** with **Python bindings**, it is widely used in **artificial intelligence (AI)** applications, including **biometric security, medical imaging, and driver monitoring systems**.

Among its many capabilities, one of Dlib's most popular features is **facial landmark detection**, which plays a crucial role in **driver drowsiness detection, facial expression analysis, and emotion recognition.**

### 5.3.2: How Dlib Works?

Dlib uses **pre-trained deep learning models** to extract **68 facial landmarks** from an image or video frame. These landmarks represent key facial features, such as:

- Eyes
- Eyebrows
- Nose
- Lips
- Jawline

The process of facial landmark detection involves:

1. **Face Detection:** Identifying the presence of a face in an image.
2. **Feature Extraction:** Detecting 68 key points on the face.
3. **Analysis:** Using these landmarks to monitor expressions, blinks, or drowsiness.

One of the most useful applications of Dlib's facial landmarks is the calculation of the **Eye Aspect Ratio (EAR)**, which determines whether a person's eyes are open or closed. EAR is used in driver monitoring systems to detect signs of drowsiness and issue alerts.

### 5.3.3: Applications of Dlib

- **Driver Drowsiness Detection:** Identifies when a driver is falling asleep and triggers an alarm.
- **Biometric Authentication:** Used in facial recognition for security purposes.
- **Health Monitoring:** Helps track facial muscle movement for medical research.

- **Virtual Try-On & AR:** Used in augmented reality applications to overlay digital elements on the user's face.

5.3.4: Advantages of Dlib

- **Highly Accurate:** Uses deep learning models for precise landmark detection.
- **Lightweight & Efficient:** Can run on low-power devices like **Raspberry Pi**.
- **Easy Integration:** Supports **Python and C++**, making it easy to implement in various AI systems.
- **Works with Pre-Trained Models:** Eliminates the need for manual training.

Dlib is widely used in **autonomous vehicles, surveillance systems, and smart assistants**, making it an essential tool for AI-driven facial analysis

---

## YOLO (You Only Look Once): Real-Time Object Detection:

YOLO (**You Only Look Once**) is a state-of-the-art object detection algorithm that **balances speed and accuracy**, making it one of the most widely used AI models for **real-time image recogn ition**. Unlike traditional object detection models that scan an image multiple times, YOLO processes the entire image in **a single forward pass**, dramatically improving efficiency.

How YOLO Works?

YOLO operates using a **deep neural network (DNN)** to divide an image into multiple **grid cells**. Each cell predicts bounding boxes and class probabilities to determine what objects are present and where they are located.

**Key Steps in YOLO's Object Detection Process:**

1. **Image Input:** The algorithm receives an image and resizes it to a fixed dimension (e.g., 416x416 pixels).
2. **Grid Division:** The image is divided into an **S x S** grid.
3. **Bounding Box Prediction:** Each grid cell predicts multiple bounding boxes
4. **Object Classification:** The algorithm assigns labels and confidence scores.
5. **Non-Maximum Suppression (NMS):** Overlapping boxes are filtered to keep the most relevant detections.

Applications of YOLO

- **Autonomous Vehicles:** Detects pedestrians, road signs, and other vehicles in real-time.
- **Surveillance Systems:** Used in security cameras for object and face detection.
- **Retail & Inventory Management:** Tracks product placement and stock levels.
- **Medical Imaging:** Detects tumors or abnormalities in X-rays and MRIs.
- **License Plate Recognition:** Identifies and extracts text from vehicle plates.

## Advantages of YOLO

- **Real-Time Performance:** Processes images at speeds exceeding **30 FPS (frames per second).**

- **High Accuracy:** Uses deep learning models to precisely detect multiple objects in an image.
- **Versatile:** Can be trained on **custom datasets** for specific tasks.
- **Efficient for Embedded Systems:** Runs on devices like **NVIDIA Jetson, Raspberry Pi, and mobile phones**.

YOLO's ability to **process images in real-time** makes it ideal for applications requiring **instant decision-making**, such as **self-driving cars, security surveillance, and industrial automation**.

---

## Dlib vs. YOLO: Key Differences

| Feature | Dlib (Facial Landmark Detection) | YOLO (Object Detection) |
|---|---|---|
| Primary Use | Facial landmark detection | General object detection |
| Model Type | Pre-trained landmark models | Deep neural networks |
| Processing | Detects facial features only | Detects multiple objects in one pass |
| Applications | Driver monitoring, biometrics | Surveillance, autonomous driving, license plate recognition |
| Speed | Fast but depends on CPU/GPU | Optimized for real-time applications |

Dlib specializes in **detailed face tracking**, while YOLO excels in **detecting multiple objects** in a single frame. Together, they can be used in advanced AI-driven applications, such as **automated traffic monitoring and smart security systems.**

## 5-4: Artificial intelligence Soft-Ware Chart:

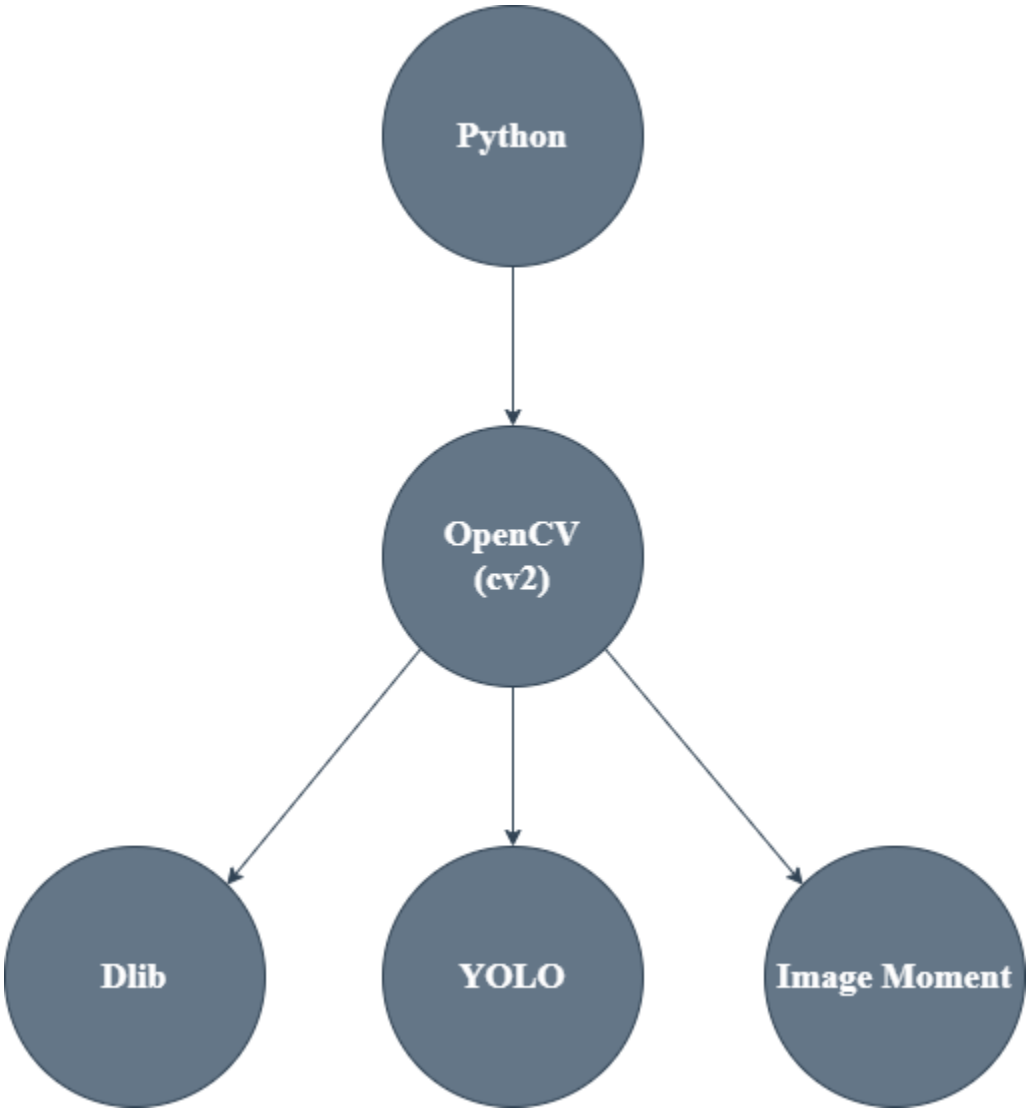



**Figure 5-4: SW chart**

## 5-6: Lane Flow Chart:

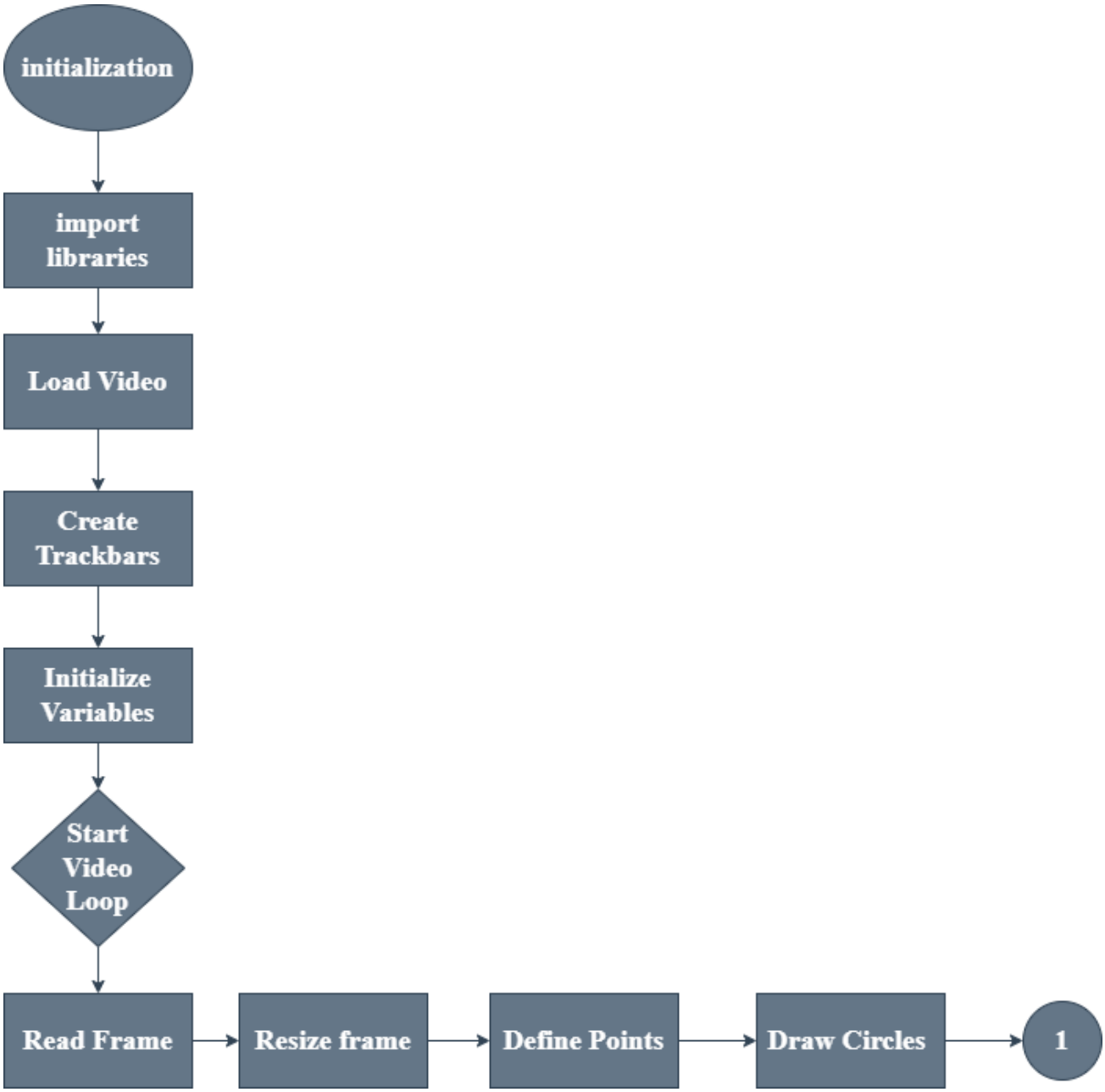



**Figure 5-4: Lane Model Flow chart part [1]**

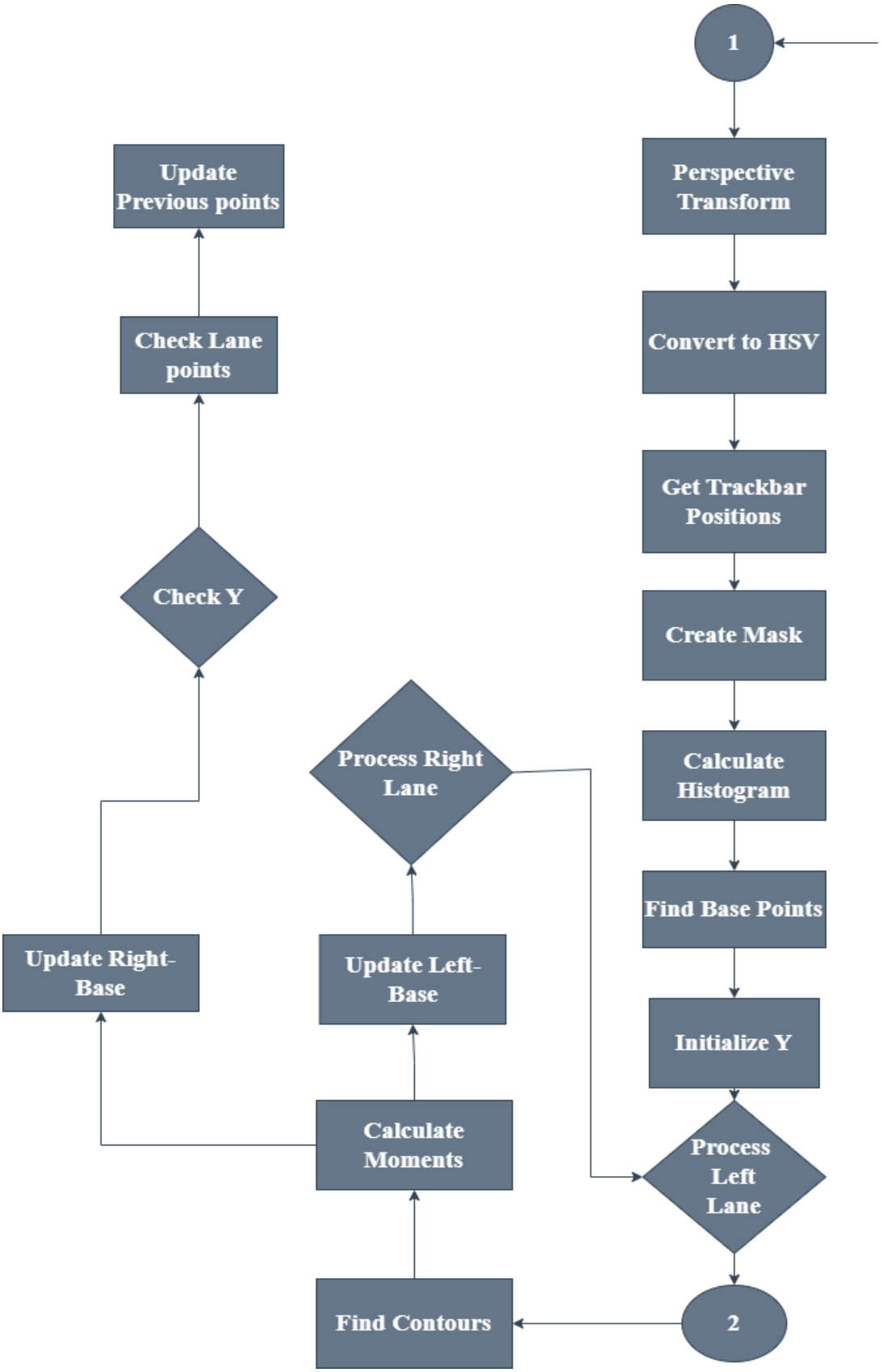



**Figure 5-5: Lane Model Flow chart part [2]**

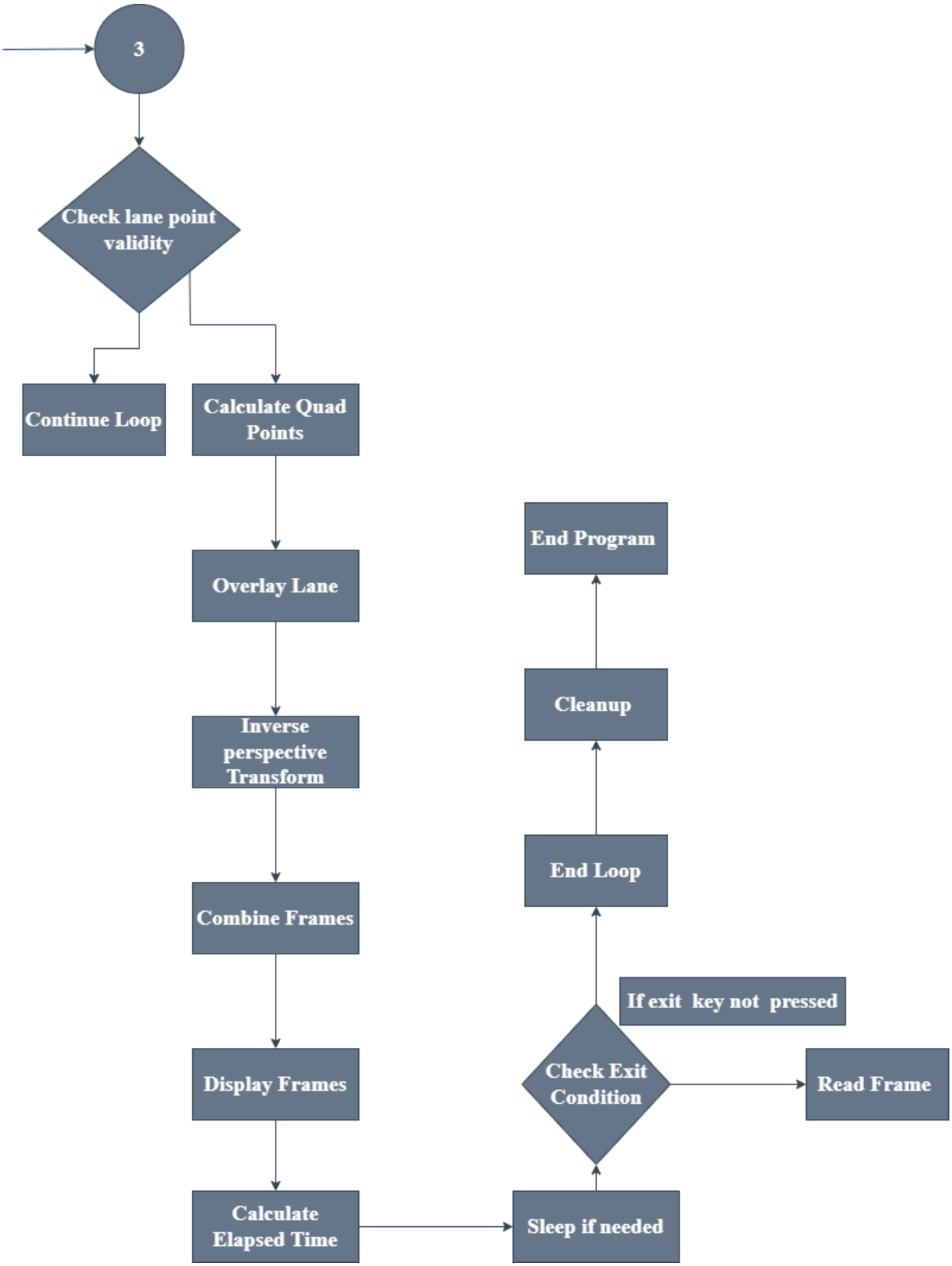



**Figure 5-6: Lane Model Flow chart part [3]**

## 5-7: Driver state Flow Chart:

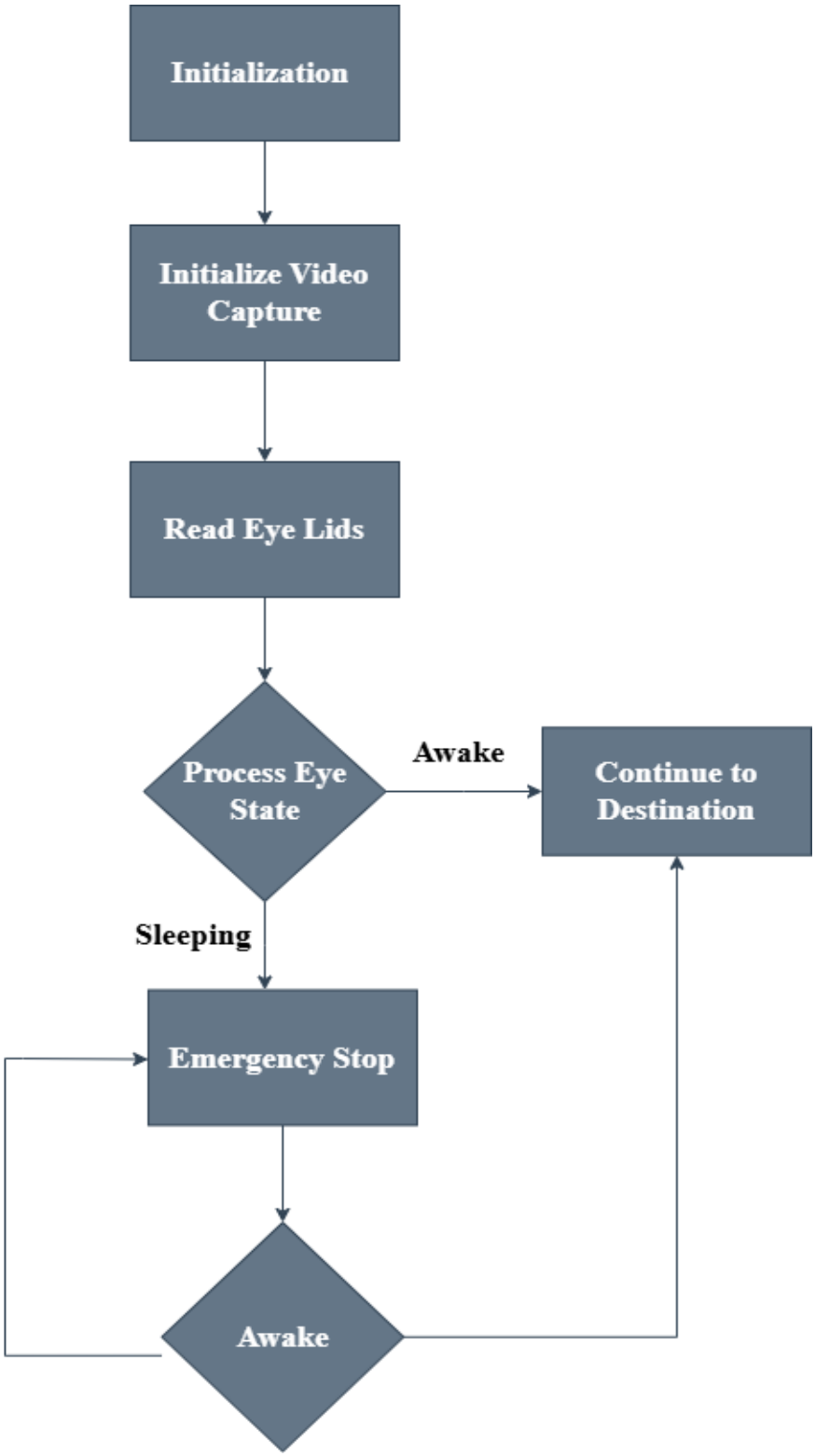



**Figure 5-7: Driver State Flow chart**

## 5-8: Sign Model Flow Chart:

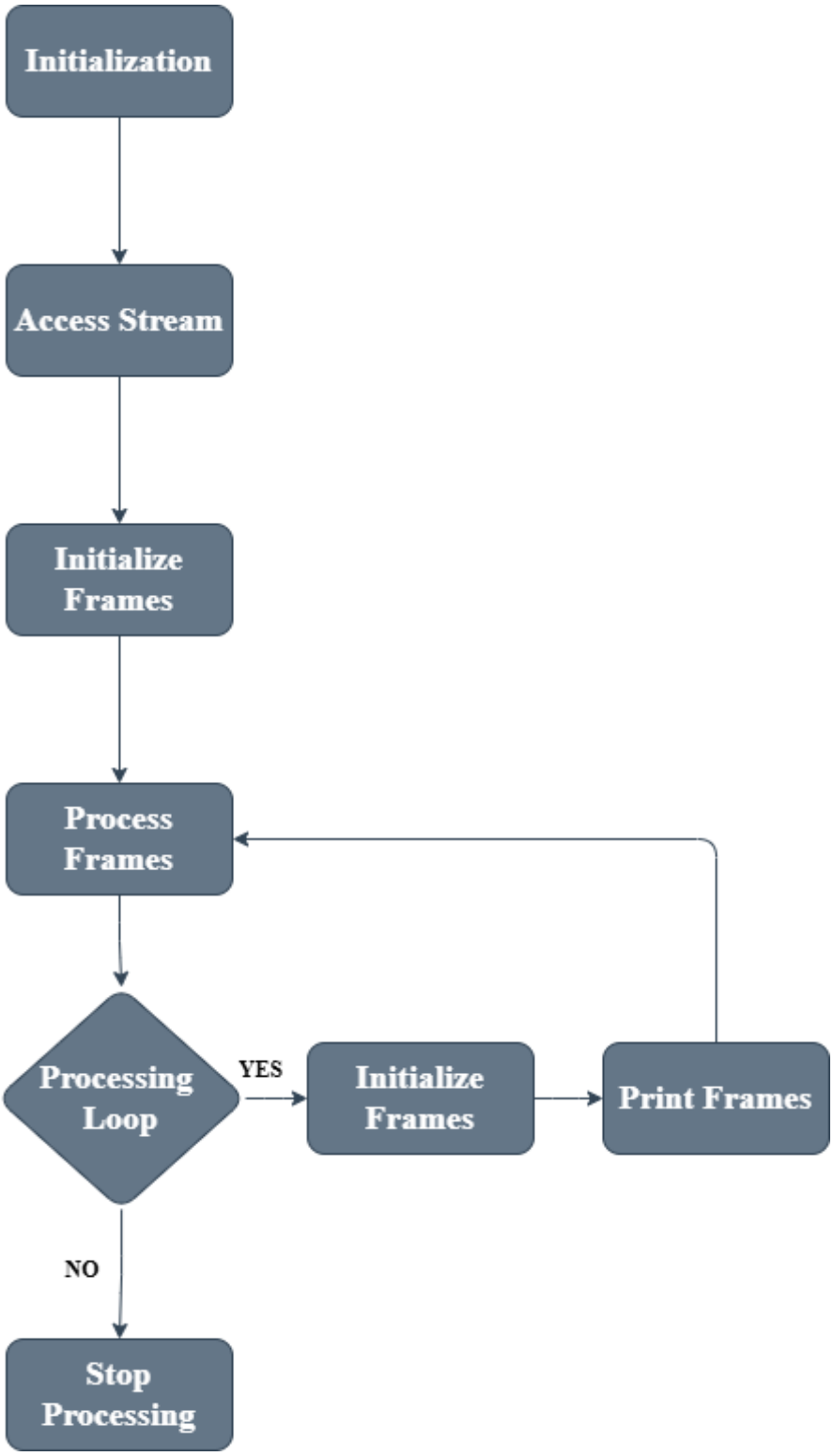



**Figure 5-8: Sign Model Flow chart**

# 5-5: Main Functions Description:

- 5.5.1: Sign Model Main Functions:

```
!pip install roboflow

from roboflow   import  Roboflow
rf = Roboflow(api_key=    "qlHLJivLDM8FrT8A7Qgu"    )
project = rf.workspace(    "testworkspace-hp4bn"    ).project(  "sign_detection-0ab1p"    )
version = project.version(5)
dataset = version.download(     "yolov11"  )
```

**Getting Data-Set of Sign Model**

```
from ultralytics    import  YOLO

model = YOLO(  "/content/sign_detection-5/yolo11n.pt"       )

results = model.train(data=     "/content/sign_detection-5/data.yaml"       , epochs=23, imgsz=640,batch=32)
```

**Training Data-Set of Sign Model**

- **5.5.2: Face-Landmarks Model Main Functions:**

```
from scipy.spatial    import  distance
from  imutils import face_utils
import imutils
import dlib
import cv2
import time

model_path =   "shape_predictor_68_face_landmarks.dat"

def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    ear = (A + B) / (2.0     * C)
    return ear
```

**Face Landmarks Model**

```
def  detect ():
    thresh   = 0.25
    frame_check    = 10
    detect   = dlib. get_frontal_face_detector    ()
    predict   = dlib. shape_predictor  (model_path)

    (lStart, lEnd)    = face_utils.  FACIAL_LANDMARKS_68_IDXS    ["left_eye"  ]
    (rStart, rEnd)    = face_utils.  FACIAL_LANDMARKS_68_IDXS    ["right_eye"  ]
    cap   = cv2. VideoCapture  (0)
    flag   = 0
    while   True :
        ret, frame    = cap. read ()
        frame    = imutils.  resize (frame, width  =640 , height =640 )
        gray    = cv2. cvtColor (frame, cv2.  COLOR_BGR2GRAY   )
        subjects    = detect (gray,  0)
        for subject    in subjects  :
            shape    = predict (gray, subject)
            shape    = face_utils.  shape_to_np  (shape)
            leftEye    = shape[lStart:lEnd]
            rightEye    = shape[rStart:rEnd]
            leftEAR    = eye_aspect_ratio   (leftEye)
            rightEAR    = eye_aspect_ratio   (rightEye)
            ear    = (leftEAR   + rightEAR)   / 2.0
            leftEyeHull     = cv2. convexHull  (leftEye)
            rightEyeHull     = cv2. convexHull  (rightEye)
            cv2.   drawContours  (frame, [leftEyeHull],     -1, (0,  255 , 0), 1)
            cv2.   drawContours  (frame, [rightEyeHull],     -1, (0,  255 , 0), 1)

            if    ear  < thresh :

                flag    += 1

                if    flag  >= frame_check  :
                    time.    sleep (0.01 )
                    print    ("sleeping"  )
                    cv2.    putText (frame,  "****************Drowsy!***************"       , (10 ,  30),
                                cv2.      FONT_HERSHEY_SIMPLEX    , 0.7 , (0,  0,  255 ),  2)
                    cv2.    putText (frame, 'sleeping time 00:'   +str (flag) +'sec' , (10 ,  60),
                                cv2.      FONT_HERSHEY_SIMPLEX    , 0.7 , (0,  0,  255 ),  2)

            else   :
                flag    = 0
                print    ("Awake" )
        cv2.  imshow ("Frame" , frame)
        key    = cv2. waitKey (1) & 0xFF
        if   key  == ord ("q" ):
            break
    cv2. destroyAllWindows   ()
    cap. release ()


detect ()
```

**Face Landmarks Model**

- ## 5.5.3: Lane Model Main Functions:

```
import cv2
import numpy as np
import time

class HeadlessLaneTracker:
    def __init__(self, stream_url):
        self.STREAM_URL = stream_url
        self.cap = cv2.VideoCapture(self.STREAM_URL)
        self.cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

        # Verify stream connection
        if not self.cap.isOpened():
            raise ConnectionError("Failed to connect to camera stream")

        # Perspective transform parameters
        self.tl = (200, 338)
        self.bl = (70, 472)
        self.tr = (415, 338)
        self.br = (592, 462)
        self.pts1 = np.float32([self.tl, self.bl, self.tr, self.br])
        self.pts2 = np.float32([[0, 0], [0, 480], [640, 0], [640, 480]])
        self.matrix = cv2.getPerspectiveTransform(self.pts1, self.pts2)
```

**Lane Model Verify**

```
        # HSV thresholds
        self.lower = np.array([0, 0, 200])
        self.upper = np.array([255, 50, 255])

        # Tracking variables
        self.prev_left = 0.0
        self.prev_right = 0.0
        self.frame_width = 640
        self.frame_center = self.frame_width // 2

    def process_frame(self, frame):
        # Processing pipeline without visualization
        frame = cv2.resize(frame, (640, 480))
        transformed = cv2.warpPerspective(frame, self.matrix, (640, 480))
        hsv = cv2.cvtColor(transformed, cv2.COLOR_BGR2HSV)
        return cv2.inRange(hsv, self.lower, self.upper)

    def detect_lanes(self, mask):
        # Same detection logic without drawing
        histogram = np.sum(mask[mask.shape[0]//2:, :], axis=0)
        midpoint = histogram.shape[0] // 2
        left_base = np.argmax(histogram[:midpoint])
        right_base = np.argmax(histogram[midpoint:]) + midpoint

        y, lx, rx = 472, [], []
```

**Lane Model Points**

```
        while y > 40:
            # Left lane detection
            left_roi = mask[y-40:y, max(0, left_base-50):left_base+50]
            if left_roi.any():
                contours, _ = cv2.findContours(left_roi, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                for contour in contours:
                    M = cv2.moments(contour)
                    if M["m00"] > 100:
                        cx = int(M["m10"]/M["m00"])
                        lx.append(left_base - 50 + cx + max(0, left_base-50))
                        left_base = lx[-1]

            # Right lane detection
            right_roi = mask[y-40:y, right_base-50:min(right_base+50, mask.shape[1])]
            if right_roi.any():
                contours, _ = cv2.findContours(right_roi, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                for contour in contours:
                    M = cv2.moments(contour)
                    if M["m00"] > 100:
                        cx = int(M["m10"]/M["m00"])
                        rx.append(right_base - 50 + cx)
                        right_base = rx[-1]

            y -= 40

        return lx, rx
```

**Lane Model Points**

```
    def get_lane_positions(self):
        ret, frame = self.cap.read()
        if not ret:
            return self.prev_left, self.prev_right

        mask = self.process_frame(frame)
        lx, rx = self.detect_lanes(mask)

        # Calculate normalized positions
        left_pos = np.mean(lx) if lx else self.prev_left
        right_pos = np.mean(rx) if rx else self.prev_right

        left_norm = (left_pos - self.frame_center) / self.frame_center
        right_norm = (right_pos - self.frame_center) / self.frame_center

        self.prev_left = left_norm if lx else self.prev_left
        self.prev_right = right_norm if rx else self.prev_right

        return float(left_norm), float(right_norm)

    def release(self):
        self.cap.release()
```

**Lane Model Position**

```
if __name__ == "__main__":
    tracker = HeadlessLaneTracker("http://192.168.1.10/stream")

    try:
        while True:
            start_time = time.time()

            left, right = tracker.get_lane_positions()

            print(f"Left: {left:.2f} | Right: {right:.2f} | Center Offset: {(left + right)/2:.2f}")

            # Maintain processing rate
            elapsed = time.time() - start_time
            if elapsed < 0.07:  # ~14 FPS
                time.sleep(0.07 - elapsed)

    except KeyboardInterrupt:
        tracker.release()
        print("Stream processing stopped")
```

**Lane Model Visualization**

# Chapter [6]:
# Results and Future Work

## 6-1: Conclusion:

The Auto Zone project represents a significant step toward the development of locally designed and manufactured autonomous vehicle systems. By leveraging artificial intelligence, computer vision, and embedded systems, this project successfully demonstrated a self-driving car model that can operate without human intervention. The integration of key AI algorithms such as YOLO for object detection, Dlib for driver state monitoring, and Image Moments for lane detection enabled the vehicle to interpret its surroundings, make informed navigation decisions, and ensure compliance with traffic regulations.

One of the standout achievements of this project was the ability to create a self-driving model tailored to local environmental and infrastructural conditions. Unlike many imported autonomous vehicle technologies that may struggle with unique road conditions, traffic behaviors, and weather variations, this system was developed with a specific focus on adaptability. The use of ESP32-CAM for real-time vision processing, Arduino Uno for microcontroller-based automation, and L298 motor drivers for efficient vehicle control ensured that the hardware was both cost-effective and highly functional.

Through rigorous testing, the model demonstrated its capacity to autonomously detect and recognize objects, maintain lane positioning, and respond to various driving conditions. Additionally, the project successfully integrated IoT concepts, allowing real-time data transmission between different components. The decision-

making process of the vehicle was guided by deep learning algorithms, enabling efficient obstacle avoidance and adaptive driving responses.

Despite these successes, challenges were encountered, particularly in optimizing real-time processing speeds, refining object detection in dynamic environments, and enhancing the vehicle's response to sudden changes in traffic conditions. These challenges highlight areas for further improvement and serve as a roadmap for future research and development.

The Auto Zone project showcases the potential of local engineering expertise in developing competitive autonomous vehicle solutions. It demonstrates that with the right technological approach, hardware configurations, and software optimizations, it is possible to create self-driving models that meet international standards while being specifically optimized for local use. This research lays the foundation for future advancements in autonomous driving, positioning it as a viable solution for future smart transportation systems.

## Future Work:

While the Auto Zone project has achieved promising results, several areas of improvement and expansion remain to be explored. The following key aspects outline future enhancements that will contribute to making the system more robust, efficient, and scalable:

1. **Enhanced Object Detection and Tracking:**
    - The current object detection model, YOLO, performs well but can be further improved using more advanced deep learning models such as YOLOv8 or Transformer-based architectures.
    - Implementing multi-frame tracking techniques will allow the vehicle to maintain awareness of moving objects even when they temporarily leave the camera's field of view.
2. **Advanced Navigation and Route Planning:**
    - Currently, the model operates on predefined maps. Future developments should incorporate GPS-based navigation and real-time route planning.
    - Integration with real-time traffic data will allow the system to make smarter routing decisions, reducing travel time and optimizing fuel consumption in larger-scale applications.
3. **Integration with Cloud Computing and Edge AI:**
    - Cloud-based AI models can enhance computational capabilities, enabling the processing of complex deep learning models without overloading the vehicle's onboard system.

- Edge AI techniques can be explored to optimize data processing directly on the vehicle, reducing latency and enhancing real-time decision-making.

4. **Adaptation to Environmental Conditions:**
   - Implementing weather-adaptive AI models will improve the vehicle's ability to operate under different lighting, rain, or fog conditions.
   - Sensor fusion techniques, combining data from LiDAR, ultrasonic sensors, and thermal imaging, can enhance the vehicle's ability to detect obstacles in low-visibility scenarios.
5. **Improved Safety and Compliance Measures:**
   - Incorporating additional safety features, such as real-time collision avoidance, emergency braking, and pedestrian detection, will improve the reliability of the system.
   - Regulatory compliance with transportation authorities should be considered to ensure the system meets legal standards for autonomous vehicles.
6. **Scalability and Commercial Viability:**
   - The project can be expanded to full-scale autonomous vehicle prototypes that can be tested in real-world environments such as university campuses, industrial complexes, or urban pilot projects.
   - Collaborations with automotive manufacturers and government transportation agencies could pave the way for future commercialization and large-scale implementation of autonomous driving solutions.

7. **Energy Efficiency and Sustainability:**
    - Exploring renewable energy integration, such as solar-powered components, can make the system more energy-efficient.
    - Developing algorithms that optimize power consumption, particularly for battery-operated prototypes, will contribute to the longevity of autonomous vehicle deployment.

By addressing these areas, the Auto Zone project can evolve into a more advanced, commercially viable autonomous transportation system. These improvements will not only refine the vehicle's technical performance but also position it as a potential solution for smart cities and future mobility solutions.

## Final Results:

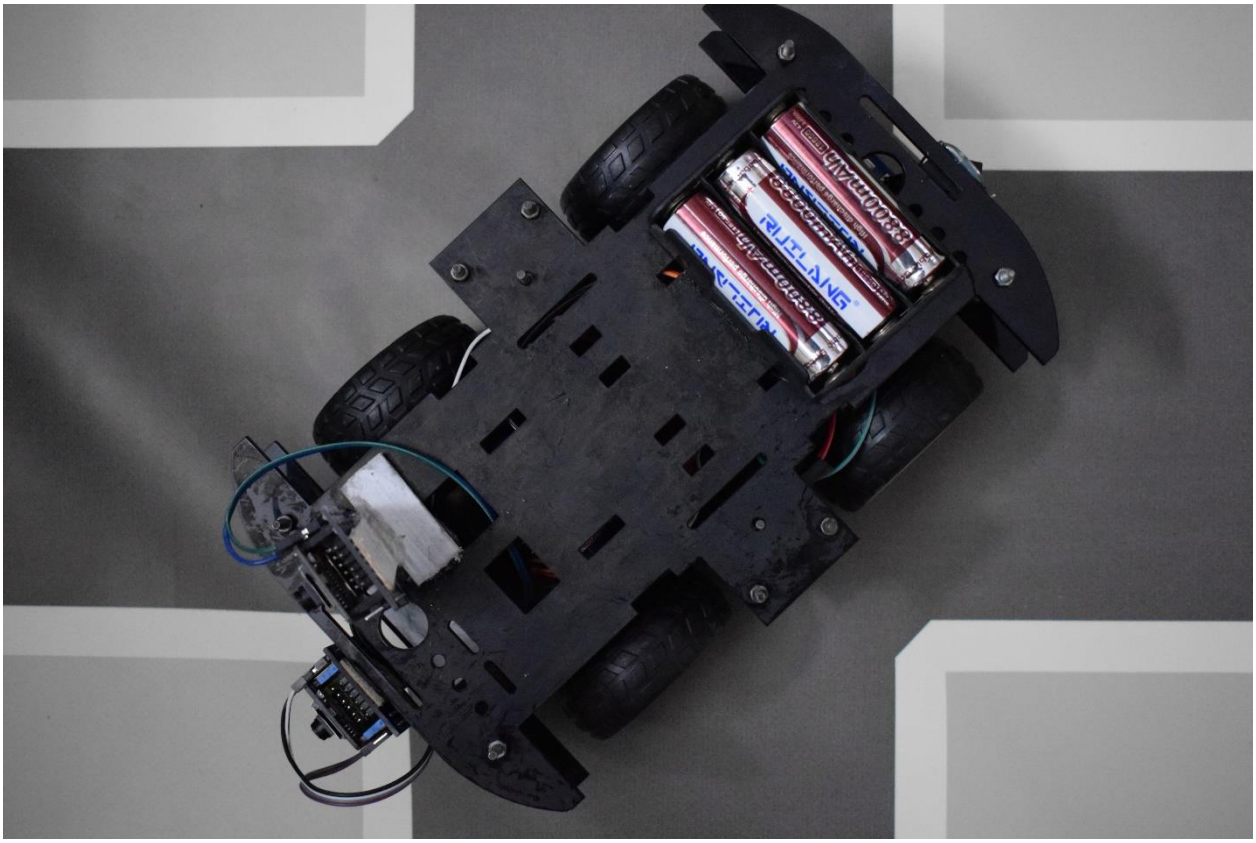


**Figure 6-1: Car Result Top View**

**“In every circuit, in every wheel turn, lies the power of innovation. This is more than just a robot—it's a testament to persistence, creativity, and the future in motion!”**

**Figure 6-1: Car Result Road View**

**“On the road of innovation, every turn is a challenge, every path a lesson. This is not just a robot—it’s a vision driving toward the future!”**

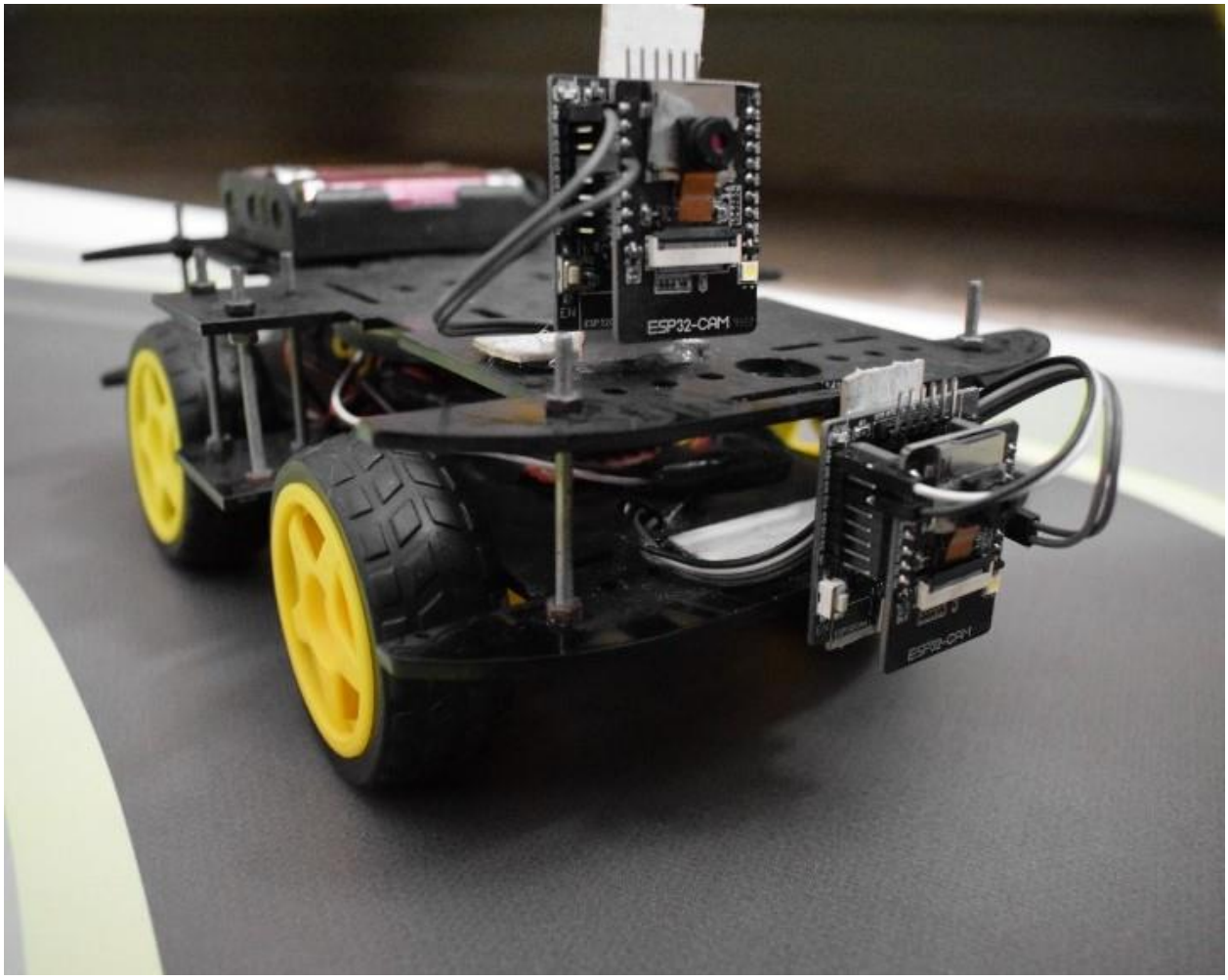



**Figure 6-3: Car Result Side View**

**"With eyes that see and wheels that move, this is more than a machine—it's the future rolling forward, one innovation at a time!"**

# Appendix:

**Arduino Code:**

**Driver State:**

**Lane Cam:**

**Main Code:**

**Lane Visualize:**

**Traffic Cam:**

**Traffic Visualize:**

## Reference:

**3.2.1:** https://lampatronics.com/product/robot-smart-car-4wd-chassis-without-motor-and-wheel/

**3.2.2:** https://www.pololu.com/product/1117/specs

**3.2.3:** https://www.st.com/resource/en/datasheet/l298.pdf

**3.2.4:** https://evelta.com/content/datasheets/408-KCD1-101.pdf

**3.2.5:**https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/106990288_Web.pdf

**3.2.6:** https://www.langruibattery.com/lithium-polymer-battery-list/62511975.html

**3.2.7:** https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

**3.2.8:**https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf

**Great Thanks and Appreciation from Auto Zone Team Members.**

# The End