Tishreen University

Graduation project report
Design and implementation of a mobile robot controlled using computer vision

College of Mechanical and electrical Engineering
Mechatronics Engineering

Student implementation:

Haidara Samer Ibrahim Hussein Abdul Hakim Turab
Firas Ayman Tamer Ali Emil Ibrahim

Supervised by: Dr. Ali Zaher

Latakia, Syria
2024/8

Index:

1. Abstract---------- 6
2. Introduction---------- 8
2.1 Background on Self-Driving Cars---------- 8
2.2 Project objectives----------8
3. Theoretical framework----------10
3.1 Self-driving car concepts---------- 10
3.2 Computer vision and artificial intelligence technologies in self-driving cars---------- 11
3.3 Self-driving car control systems---------- 12
3.4 Kinematic study of the robot---------- 13
4. System design---------- 18
4.1 Main components of the system----------18
4.2 System architecture and component integration---------- 19
4.3 Control and processing algorithms----------20
5. System implementation----------27
5.1 Hardware Setup---------- 27
5.2 Arduino Programming---------- 28
5.3 Programming the ESP8266---------- 28
5.4 Training and Implementing the YOLO V8 Model---------- 28
5.5 System Integration and UDP Communication Protocol----------41
6. System testing and performance evaluation---------- 44
6.1 Test Methodology---------- 44
6.2 Test results---------- 45
6.3 Performance Analysis----------46
6.4 Analysis of results---------- 48
6.5 Challenges and Limitations---------- 48
6.6 Comparison with existing systems---------- 49
7. Conclusion and future work----------51
7.1 Summary of Achievements----------51

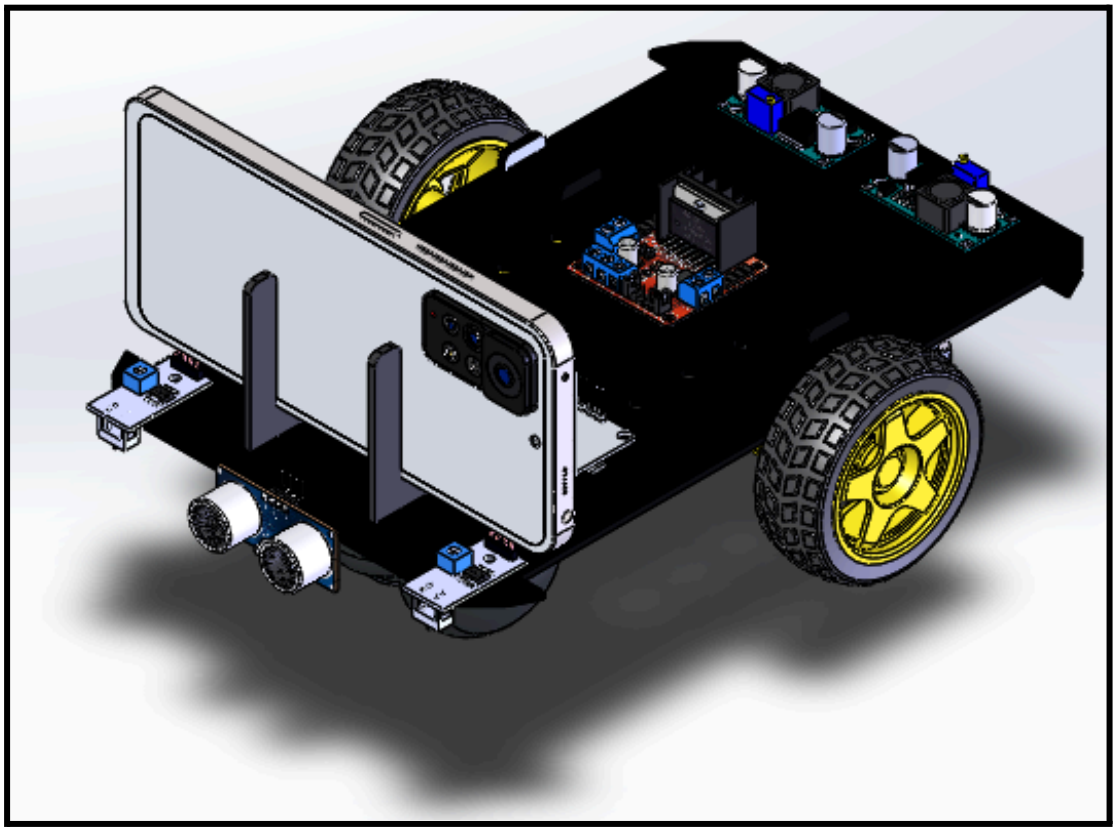
7.2 Conclusions----------51
7.3 Suggestions for future improvement and development---------- 51
8. Appendices and References---------- 54
8.1 System schematics---------- 54
8.2 Source Code---------- 55
8.3 References:----------61
Thanks and appreciations---------- 62

Chapter One
Summary

1. Summary

This project aims to build a Differential mobile robot able to respond to a limited number of speed signals, stopping at obstacles similar to self-driving cars.
The system uses an advanced Computer Vision model "YOLOv8".
We trained this model from scratch to recognize speed signals with high accuracy. It sends data to the main controller (Arduino) via the ESP8266.
The Arduino processes the data and makes the necessary decisions to adjust the motor speed using the L298N driver circuit.
This project achieved good results in signal recognition and speed adjustment, representing a successful integration between the fields of computer vision, artificial intelligence, wireless communications, control, the robot, mechanical design and electronic circuits.
The result is an integrated and efficient system capable of autonomous navigation and dynamic response to traffic signals in a realistic environment and it can also be used for educational purposes.

- Car body design using SOLIDWORKS



Chapter Two
Introduction

2. Introduction

2.1 Background on Self-Driving Cars

Self-driving cars are considered one of the most prominent technological revolutions of the 21st century, seeking to reshape the transportation sector and improve people's lives. This project aims to build a scale model of a self-driving car capable of navigating a specific environment and addressing some of the fundamental challenges in this field.

This will be achieved by integrating a variety of technologies, including computer vision, machine learning, and control systems.

2.2 Project objectives

This project can be considered as having educational objectives as it aims to:

- Students can learn to apply programming and control skills to a mobile robot.
- They learn concepts in digital image processing.
- Learn how to build and train a neural network for image analysis and classification.
- They learn how to program and implement an integrated robotic system using microcontrollers.

Scientific objectives:

- Development of an autonomous robot control system.
- Improve communication between robot and computer.
- Improve image analysis and classification algorithms.
- Implement solutions in real-world environments.
- Improving the results of training neural networks to recognize traffic signs.

Chapter Three

Theoretical framework

3. Theoretical framework

3.1 Self-driving car concepts

A self-driving car is defined as a vehicle capable of navigating and driving from one point to another without the need for constant human intervention. These cars rely on an integrated suite of advanced technologies, including sensors, computers, and software, to determine their location, analyze their surroundings, and make the decisions necessary to navigate safely and efficiently.

Self-driving cars can be divided into several levels of automation, which are:

- **Level 1:** Driver assistance system, where the system assists the driver with certain tasks such as controlling speed or maintaining a safe distance.
- **Level 2:** Semi-automated driving system, where the car can perform some driving tasks such as changing lanes or parking, but requires the driver to be ready to intervene at any time.
- **Level 3:** Conditional autonomous driving, where the car can perform most driving tasks under specific conditions, but requires the driver to be ready to take over control in an emergency.
- **Level 4:** A highly automated driving system, where the car can drive autonomously in most conditions, but human intervention is still required in certain situations.
- **Level 5:** A fully automated driving system, where the car can drive completely autonomously in all conditions without the need for any human intervention.

3.2 Computer vision and artificial intelligence technologies in self-driving cars

Computer vision and artificial intelligence technologies are the backbone of self-driving cars, enabling them to "see" the world around them and make appropriate decisions. These technologies rely on a set of complex algorithms and neural networks that enable the car to:

- **Object recognition:** Identify and classify different objects in the surrounding environment, such as other cars, pedestrians,Traffic lights and road signs. Algorithms such as YOLO and CNN are used to achieve this.
- **Object tracking:** Tracks the movement of detected objects to estimate their paths and speeds. This technology helps the vehicle predict the movement of other objects and avoid collisions.
- **Create 3D maps:** Building a 3D model of the vehicle's surroundings, helping it understand the geometry of the space and pinpoint its location.
- **Pattern recognition:** Recognize recurring patterns in visual data, such as road lines, traffic lights, and road signs.

Deep neural networks play a crucial role in these techniques. These networks are trained on massive amounts of visual data to learn to recognize complex patterns and make informed decisions.

In our project We have used the YOLOv8 algorithm to detect objects and road signs, this algorithm is characterized by its high speed and accuracy, making it suitable for applications that require real-time response.

Thanks to these technologies, self-driving cars can understand their surroundings and make the decisions necessary to drive safely and efficiently.

Types of sensors used:
Cameras, infrared sensors, ultrasonic sensors, etc.
Challenges facing computer vision technologies:
Such as bad weather conditions, low light, and fast moving objects.
Other applications of computer vision and artificial intelligence technologies:
Such as parking assistance, a safe distance keeping system, and adaptive cruise control.

3.3 Self-driving car control systems

Control systems serve as the backbone of a self-driving car's operation, responsible for translating data collected by sensors into precise control commands that guide the vehicle to achieve its goals.
These systems rely primarily on the principle of automatic control, using mathematical models to describe the vehicle's behavior and design control systems that ensure the system's stability and track-following.

Self-driving car control systems can be divided into several levels:

- **Low control level:** It deals with the direct control of vehicle components such as the engine, brakes, and steering.

  PID (Proportional-Integral-Derivative) controllers are commonly used at this level to achieve fast and accurate response.

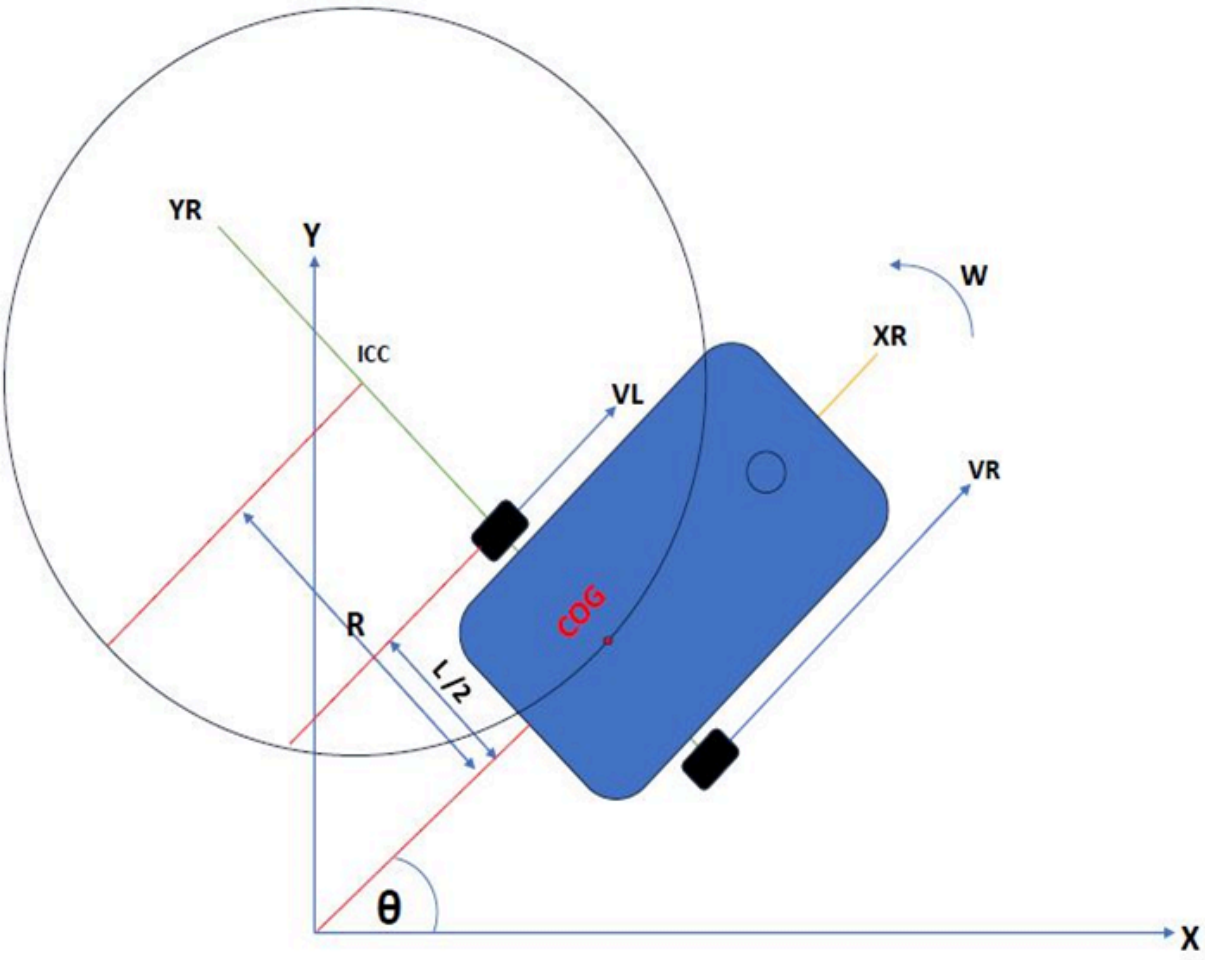
- **Medium control level:** It deals with path planning and obstacle avoidance. Planning algorithms such as ***A**** and ***Dijkstra's*** algorithm are used to determine the optimal path to reach the goal.
- **High level of control:** It handles strategic decision-making, such as choosing the optimal route, handling emergencies, and communicating with other cars and infrastructure. Artificial intelligence techniques such as machine learning and neural networks are used to teach the car to make the right decisions.

Factors that influence the design of a self-driving car's control system include:

- Vehicle dynamics: Vehicle properties such as mass, moment of inertia, and coefficient of friction.
- Environmental conditions: road conditions, weather, and lighting.
- Performance requirements: responsiveness, accuracy, and security.

Control systems are an integral part of any self-driving car, responsible for ensuring the safety and comfort of passengers.

3.4 Kinematic study of the robot



$V_L = w \times (R - \frac{L}{2})$ ..... (1)

$V_R = w \times (R + \frac{L}{2})$ ..... (2)

$\Rightarrow w = \frac{V_L}{R - \frac{L}{2}} = \frac{V_R}{R + \frac{L}{2}}$ ..... (3)

$\Rightarrow R = \frac{L \times (V_R + V_L)}{2 \times (V_R - V_L)}$ ..... (4)

If $V_R = V_L$, $then\ R = \infty \Rightarrow$ Rectilinear motion.

If $V_R = A$, $V_L = 0$, $then\ R = \frac{L}{2} \Rightarrow$ Circular motion centered at the left wheel and its radius is $\frac{L}{2}$.

If $V_R = 0$, $V_L = A$, $then\ R = \frac{L}{2} \Rightarrow$ Circular motion centered at the right wheel and its radius is $\frac{L}{2}$.

If $V_R = -V_L$, $then\ R = 0 \Rightarrow$ The robot rotates around a point located at the middle of the wheel axis.

By substituting R in (1) and (2):

$w = \frac{V_R - V_L}{L}$

$V = \frac{V_R + V_L}{2}$

$V_R = r \cdot \theta'_R$ $\qquad$ $w = \frac{r}{L} \times (\theta'_R - \theta'_L)$

$\Rightarrow$

$V_L = r \cdot \theta'_L$ $\qquad$ $V = \frac{r}{2} \times (\theta'_R + \theta'_L)$

$X' = V \times \cos\theta = \frac{r}{2} \times \cos\theta \times (\theta'_R + \theta'_L)$

$Y' = V \times \sin\theta = \frac{r}{2} \times \sin\theta \times (\theta'_R + \theta'_L)$

$$\theta' = w = \frac{r}{L} \times (\theta'_R - \theta'_L)$$

$$\begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \frac{r}{2} \times \cos\theta & \frac{r}{2} \times \cos\theta \\ \frac{r}{2} \times \sin\theta & \frac{r}{2} \times \sin\theta \\ \frac{r}{L} & \frac{-r}{L} \end{bmatrix} \begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix}$$

Inverse kinematics model:

$$\begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix} = \begin{bmatrix} \frac{r}{2} \times \cos\theta & \frac{r}{2} \times \cos\theta \\ \frac{r}{2} \times \sin\theta & \frac{r}{2} \times \sin\theta \\ \frac{r}{L} & \frac{-r}{L} \end{bmatrix}^{-1} \begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix}$$

Here, the matrix is non-square, so we resort to the pseudoinverse (Moore-Penrose inverse): $J^+ = (J^T \cdot J)^{-1} \cdot J^T$

$$\begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix} = \begin{bmatrix} \frac{\cos\theta}{r} & \frac{\sin\theta}{r} & \frac{L}{2r} \\ \frac{\cos\theta}{r} & \frac{\sin\theta}{r} & \frac{-L}{2r} \end{bmatrix}^{-1} \begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix}$$

$$\theta'_R = \frac{\cos\theta}{r} + \frac{\sin\theta}{r} + \frac{L}{2r}\theta'$$

$$\theta'_L = \frac{\cos\theta}{r} + \frac{\sin\theta}{r} - \frac{L}{2r}\theta'$$

- **Substitution in the direct kinematics matrix:**

$$\begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \frac{32.5}{2} cos\,\theta & \frac{32.5}{2} \times cos\,\theta \\ \frac{32.5}{2} \times sin\,\theta & \frac{32.5}{2} \times sin\,\theta \\ \frac{32.5}{170} & \frac{-32.5}{170} \end{bmatrix} \begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix}$$

$$\begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix} = \begin{bmatrix} 16.25\,cos\,\theta & 16.25\,cos\,\theta \\ 16.25\,sin\,\theta & 16.25\,sin\,\theta \\ 0.1912 & -\ 0.1912 \end{bmatrix} \begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix}$$

- **Substitution in the inverse kinematics matrix:**

$$\begin{bmatrix} \theta'_R \\ \theta'_L \end{bmatrix} = \begin{bmatrix} \frac{cos\,\theta}{32.5} & \frac{sin\,\theta}{32.5} & 0.002615 \\ \frac{cos\,\theta}{32.5} & \frac{sin\,\theta}{32.5} & -\ 0.002615 \end{bmatrix}^{-1} \begin{bmatrix} X' \\ Y' \\ \theta' \end{bmatrix}$$

Chapter Four

System design

4. System design

4.1 Main components of the system

Each component of the system forms a cornerstone in achieving the project objectives:

Main control unit (Arduino):

- **Role:**

  The Arduino acts as the brain of the system, receiving data from sensors and controlling actuators.

- **Jobs:**
  a. Receive data from the ESP8266 module regarding the set speed value.
  b. Calculating the motor speeds based on the set speed value.
  c. Sending commands to the L293N driver module to control the speed and the direction of the motors motion.
  d. Monitor system status and diagnose errors.

Wireless communication module (ESP8266):

- **Role:** Provides wireless communication between the traffic signal detection system (computer with AI)
  and the main control unit (Arduino).
- **Jobs:**
  a. Receive the specified speed value from the computer via UDP protocol.
  b. Send speed value to Arduino via serial connection.

Traffic Signal Detection System (YOLOv8):

- **Role:** Responsible for locating traffic signals and setting the speed limit.

- **Jobs:**
    a. Processing images and videos captured by the camera.
    b. Detecting the presence of traffic signals and determining their type and choosing the speed limit.
    c. Sending speed value to ESP8266 module.

Tracking system (IR sensors):

- **Role:** Responsible for tracking the black line drawn on the ground.

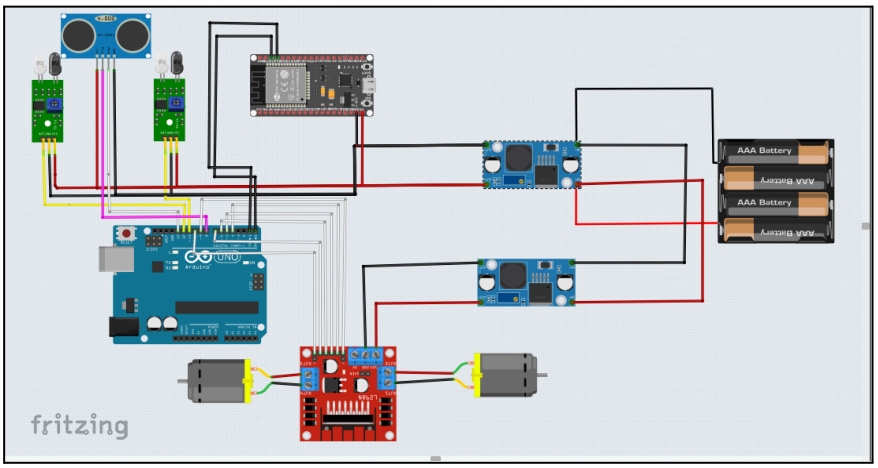
- **Jobs:**
    a. Detecting deviation from the black line.
    b. Sending signals to Arduino to correct the car's path.

Motion system (DC motors and L293N driver):

- **Role:** Responsible for moving the vehicle.

- **Jobs:**
    a. Convert electrical signals from the Arduino into mechanical energy to move motors.
    b. Control the speed and direction of rotation of motors.

4.2 System architecture and component integration



4.3 Control and processing algorithms

- **Line tracing algorithm:**
  A path-tracking algorithm is used in robots to control motion based on information from infrared (IR) sensors. In this project, the line is black and the background is white. The algorithm determines the robot's path based on readings from these sensors.

  Working principle:

  1. **Reading sensor data:**
     Two IR sensors are used: one on the left and one on the right. Values are read using digitalRead() function:
     a. **LOW**: when the sensor is above the white background.
     b. **HIGH**: when the sensor is above the black line.

2. **Determine the status of sensors:**

   The read values are analyzed to determine the robot's movement:

   - Both sensors on the back (LOW, LOW):
     The robot moves forward.
   - Left sensor above the line (HIGH) and right sensor on the background (LOW):
     The robot rotates to the right.
   - Left sensor on the back (LOW) and right sensor above the line (HIGH):
     The robot rotates to the left.
   - Both sensors above the line (HIGH, HIGH):
     The robot stops.

3. **Execute the movement:**

   The setMotorSpeeds() function is called to set the motor speeds based on the specified condition.

- **Speed control algorithm:**
  Traffic control systems require efficient algorithms to adjust vehicle speeds based on traffic signals. The system relies on values received from a surveillance camera, adjusting engine speeds according to the detected set speed.

  Algorithm steps:

1. **Receive speed value:**

   A UDP message containing the speed value is sent from the camera when a speed signal is detected.

2. **Speed value analysis:**
    a. The received message is analyzed to determine the required speed.
    b. If the message contains a value like "Speed 60", then the number 60 is extracted.
3. **Converting speed to a workable value:**

    We calculate the appropriate engine speed using a conversion formula.

    ```
    speed_value = int(70 + (detected_speed - 50) * (255 - 70) / (120 - 50))
    ```

    This equation adjusts the speed based on a specific range (50 to 120).

4. **Speed application to motors:**
    a. The calculated value is sent to the motors to adjust their speed.
    b. A protocol such as PWM (Pulse Width Modulation) is used to precisely adjust the speed of motors.
5. **Repeat the process:**

    The algorithm continues to receive new values and adjust the speed dynamically.

- **Decision making algorithm:**

  The robot operates according to a set of rules that determine its behavior in specific situations, such as detecting an obstacle or losing a line. This explanation will clarify how decisions are made in these situations.

  Decision-making situations:

  a. Obstacle detection

  b. Distance verification: The robot uses an ultrasonic sensor to measure the distance between its front and the obstacle.

Action upon detection of an obstacle:
If the distance is less than or equal to 10 cm:

- Stop motion: Stop the engines.
- Change status: Set "isMoving" to false.
- Send message: Print a message in the serial interface indicating that there is an obstacle at X distance.

Procedure when removing the obstacle:
If there is no obstacle (distance greater than 10 cm) and "isMoving" is false :
Movement resumes as it is adjusted. "isMoving" to true and returns the engines to the previous speed.

- **Technical details:**

Main wheels:

Two wheels, each wheel is mounted on a DC motor hub.

Engines:

The used motor is of the TT type. These motors operate at a voltage of (3-6) V, and consume a current ranging from (100-250) mA.

Typical torque for these motors ranges from (0.5 – 1) kgf/cm^3 and the rotation speed is around 200 rpm without load.

The motors were selected based on the required torque, where we distinguish two torques:

Starting Torque:

- It exceeds the driving torque.
- The force of static friction, which is greater than kinetic friction, must be overcome.

- Needs to accelerate the robot from a standstill.

Running Torque:

- It is less than the starting torque.
- It deals with kinetic friction, which is less than static friction.
- It only needs to maintain a constant speed (no acceleration).

Generally, the engine should be able to provide at least the take-off torque, as it is the highest.
Once motion begins, it will require less torque to maintain movement.
The coefficient of kinetic friction of the rubber with the surface used equals 0.3

Static friction:

Static friction is the force that prevents two objects in contact from moving relative to each other when they are at rest. This force is greater than kinetic friction.
It was measured experimentally by placing the robot on an inclined surface and increasing the incline angle until the robot starts to slide.

It was found by measurement that the value of the angle is equal to 31 degrees, and thus the coefficient of static friction is the tangent (Tan) of this angle which is 0.6, and the total weight of the robot with the motors and wheels is:

Torque	Equation	Value
While moving at a constant speed	$\tau_{run} = \mu_k * m * g * r$	0.098 N.m 0.98 kg.cm
Take-off torque	$\tau_{start} = \mu_s * m * g * r$	0.197 N.m 1.97 kg.cm

Whereas:

- τ_{run}: Kinetic torque N.m) and τ_{start}: Take-off torque (N.m).
- μ_k: Coefficient of kinetic friction and μ_s: Coefficient of static friction.
- m: Mass of robot (kg).
- g: Acceleration due to gravity (9.8 m/s^2).
- r: Wheel radius (m).

If we take the starting torque, this means that each engine must provide at least:

1.97/2 = 0.985 (kg.cm)

If each motor produces a torque capacity of 0.5 kg.cm (the minimum torque of the engine), the total torque equals 1 kg.cm, which is more than required. Therefore, the engines are sufficient with a good performance margin, so using higher-torque engines is considered a waste of resources.

Castor wheel:

The wheel used is a free-spinning wheel that allows movement in all directions.

In robotics, it provides stability and support without affecting the robot's steering.

It features 360-degree rotation, making it easy to maneuver the robot.

IR sensors:

It works by sending and receiving infrared rays.

L298N motor driver circuit:

- Operating voltage: 5V to 35V.
- DC current up to 2A per channel.
- Operating supply voltage up to 46V.
- Dual H-bridge design, allowing control of two DC motors at the same time.

Chapter Five

System implementation

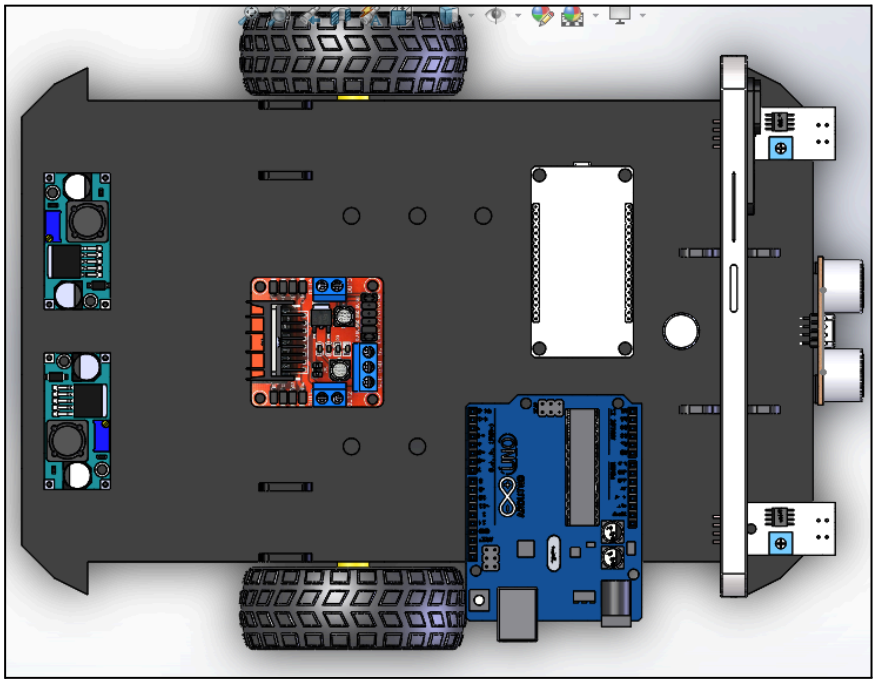
5. System implementation

5.1 Hardware Setup

At this stage, all hardware components are assembled and connected according to the design schematic. The basic steps include:

- **Vehicle body design and component assembly:**

  We have used the **SOLIDWORKS software.**



- **Vehicle body assembly:** Building the car body using the appropriate materials and installing the engines and steering wheels.
- **Installing electronic devices:** Installing the Arduino, ESP8266, camera, IR sensors, and L293N driver in their designated places.
- **Connecting the wires:** Connecting all components together according to the circuit diagram.
- **Energy saving:** Connecting the appropriate power supply to each component.

5.2 Arduino Programming

Arduino programming is the core of the system, where the code responsible for controlling all components is written. The main tasks include:

- **Reading data from sensors:** Reading the IR sensor signals to determine deviation from line.
- **Receiving commands from ESP8266:** Receiving the specified speed value from the ESP8266.
- **Engine speed calculation:** Calculating the required speed for each motor based on the set speed value and IR sensor readings.
- **Motor control:** Sending commands to the L293N driver circuit to control the speed and direction of rotation of the motors.
- **System monitoring:** Monitoring system status and diagnosing errors.

5.3 Programming the ESP8266

ESP8266 programming focuses on providing wireless communication between the computer and the Arduino module. Key tasks include:

- **Creating a contact point:** Establishing a wireless hotspot to allow your computer to connect to the robot.
- **Receiving data from computer:** Receiving the specified speed value from the computer via UDP protocol.
- **Sending data to Arduino:** Sending speed value to Arduino serially.

5.4 Training and Implementing the YOLOv8 Model

YOLOv8 Overview:

YOLOv8, or "You Only Look Once version 8," is the latest version of the family of real-time object detection algorithms. Building on the success of previous versions, it offers several improvements to enhance its performance and capabilities.

- **Key Features of YOLOv8**

1. Structure:

-BackBone: responsible for extracting features from the input image.

- Neck: Connects the **BackBone** layer with the **Head** layer. It is important for feature fusion. It helps combine features from different layers to improve detection accuracy.

- Head: Responsible for predicting bounding boxes, object classes, and confidence scores. The YOLOv8 header is designed to be more efficient and accurate.

2. Various models:

YOLOv8 presents different models, such as: YOLOv8l and YOLOv8s, which vary in size and computational complexity. This allows users to choose a model that best suits their specific needs, whether for resource-constrained environments or high-performance applications.

3. Training techniques:

YOLOv8 presents improvements in training strategies, including the optimization of Rectified Adam (RAdam) and the use of anchor-based or anchor-free object detection. These improvements contribute to faster convergence during training and improved performance on object detection tasks.

4. Performance:

YOLOv8 is known for its efficiency and accuracy in real-time object detection tasks, it achieves excellent object detection results while maintaining a balance between speed and accuracy.

It is widely used in various computer vision applications, including autonomous vehicles, surveillance systems, and image and video analysis.

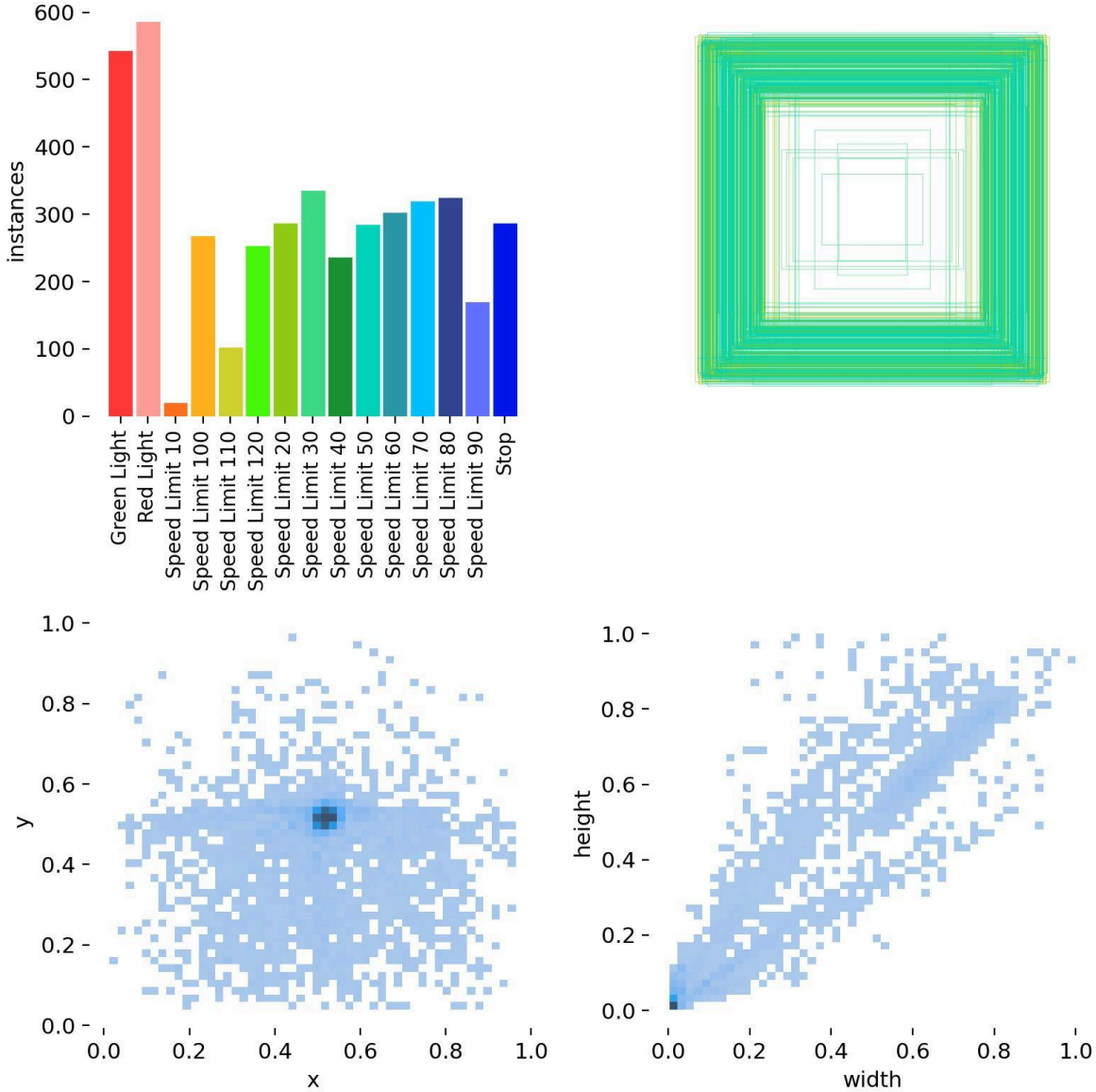
❖ **Training and implementing the YOLO V8 model requires the following steps:**

I. Data collection:

Collect a large collection of photos and videos containing different traffic signs in various environmental conditions.

In this project we have used the German Traffic Sign Detection Benchmark database.

The GTSDB (German Traffic Sign Detection Benchmark) database is used to train traffic sign detection systems. This dataset contains images of various traffic signs with annotated locations.



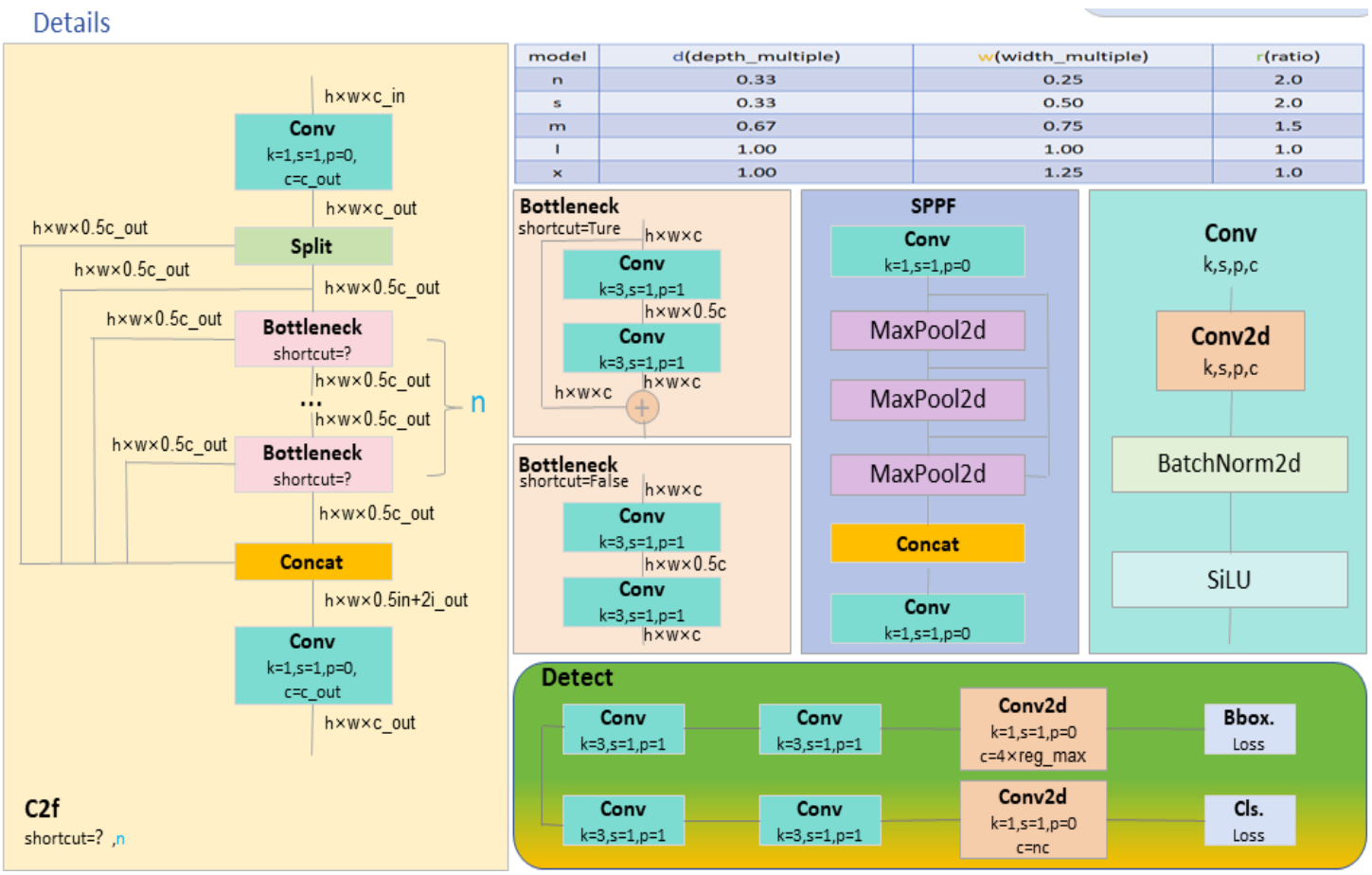
Database format in YOLOv8:

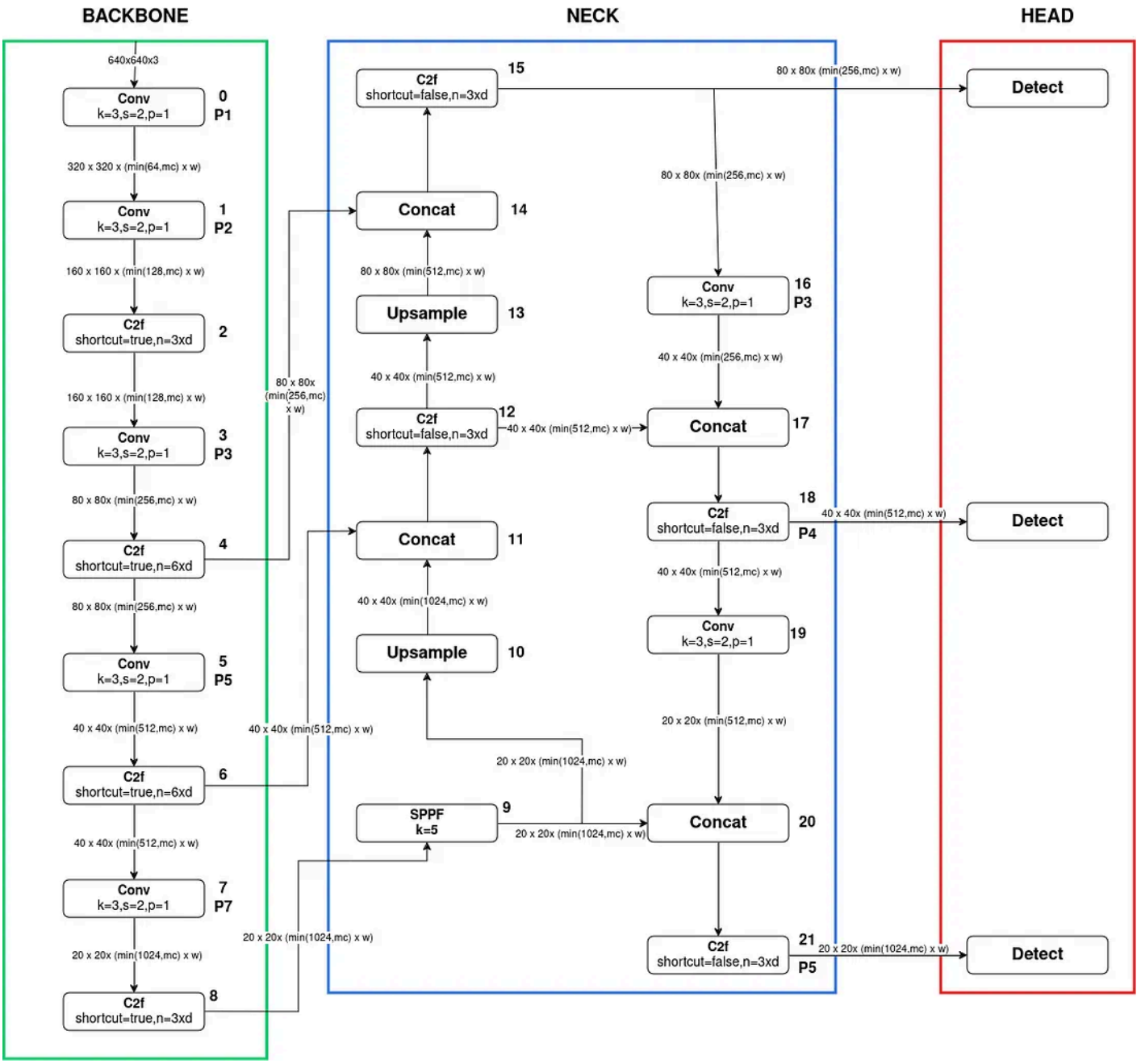
Pictures: multiple images of traffic signs.

Label files: Each image has a text file that specifies:

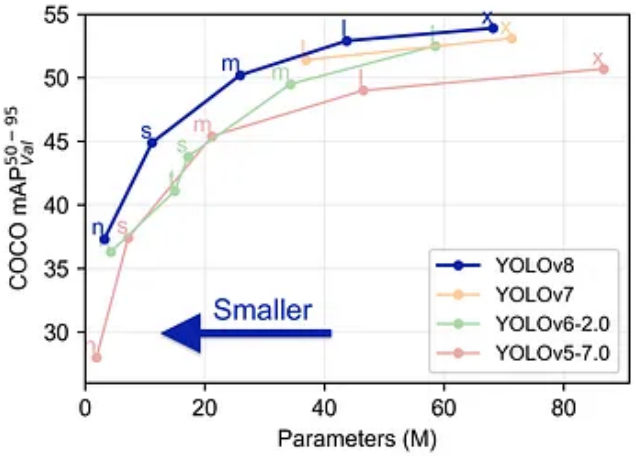
- **Category**: Type of traffic signal (e.g., stop, yield).
- **Coordinates:** YOLO format uses relative coordinates: (x_{center}, y_{center}, width, height) for the image size.

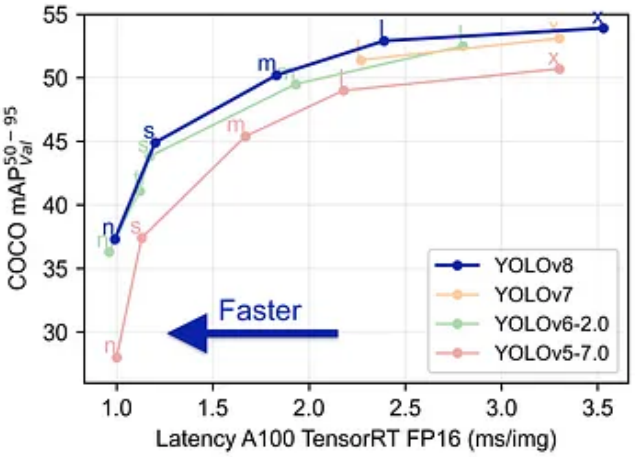
II. Structure and specifications of the model used:




BACKBONE
NECK
HEAD
640x640x3
Conv k=3,s=2,p=1
0 P1
320 x 320 x (min(64,mc) x w)
Conv k=3,s=2,p=1
1 P2
160 x 160 x (min(128,mc) x w)
C2f shortcut=true,n=3xd
2
160 x 160 x (min(128,mc) x w)
Conv k=3,s=2,p=1
3 P3
80 x 80x (min(256,mc) x w)
C2f shortcut=true,n=6xd
4
80 x 80x (min(256,mc) x w)
Conv k=3,s=2,p=1
5 P5
40 x 40x (min(512,mc) x w)
C2f shortcut=true,n=6xd
6
40 x 40x (min(512,mc) x w)
Conv k=3,s=2,p=1
7 P7
20 x 20x (min(1024,mc) x w)
C2f shortcut=true,n=3xd
8
80 x 80x (min(256,mc) x w)
40 x 40x (min(512,mc) x w)
20 x 20x (min(1024,mc) x w)
C2f shortcut=false,n=3xd
15
Concat
14
80 x 80x (min(512,mc) x w)
Upsample
13
40 x 40x (min(512,mc) x w)
C2f shortcut=false,n=3xd
12
Concat
11
40 x 40x (min(1024,mc) x w)
Upsample
10
SPPF k=5
9
20 x 20x (min(1024,mc) x w)
80 x 80x (min(256,mc) x w)
Conv k=3,s=2,p=1
16 P3
40 x 40x (min(256,mc) x w)
40 x 40x (min(512,mc) x w)
Concat
17
C2f shortcut=false,n=3xd
18 P4
40 x 40x (min(512,mc) x w)
Conv k=3,s=2,p=1
19
20 x 20x (min(512,mc) x w)
Concat
20
C2f shortcut=false,n=3xd
21 P5
20 x 20x (min(1024,mc) x w)
Detect
Detect
Detect

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8





III. Model training: YOLOv8 model training on a dataset using deep learning libraries like ***PyTorch*** and ***ultralytics*** .
Google Collab was used in the training process due to the lack of capabilities of our computers.
The model was trained according to the following training parameters:

```
task: detect
mode: train
epochs: 100
time: null
patience: 100
batch: 16
imgs: 416
save: true
save_period: -1
optimizer: auto
dropout: 0.0
val: true
split:val
lr0: 0.01
lrf: 0.01
auto_augment: randaugment
```

IV. Performance evaluation: Evaluating the performance of the trained model on a test dataset.

1. Loss Metrics

- **train/box_loss**:
  - The box loss appears to decrease rapidly as the number of epochs increases, indicating that the model is learning well. Values start at around 0.8 and go up to around 0.4.
- **train/cls_loss**:
  - The classification loss also decreases, but at a slower rate. This loss reflects how accurately the model classifies objects. Here, values range from about 2.5 to about 0.5.
- **train/dfl_loss**:
  - The DFL (Distribution Focal Loss) loss shows a continuous decrease, which means that the model becomes more accurate in identifying objects.
- **val/box_loss**:
  - The box loss in the validation set follows the same trend, indicating no overfitting yet. Values range from 0.75 to 0.55.
- **val/cls_loss**:
  - The classification loss on the validation set also decreases, meaning the model still performs well on unseen data.
- **val/dfl_loss**:
  - The DFL loss on the validation set also decreases, indicating the stability of the model.

2. Performance Metrics

- **metrics/precisionB**:
  - The model's precision increases significantly, meaning that the model makes a greater number of correct predictions across all predictions. The values converge towards 1.

- **metrics/recallB**:
    - Recall also improves, indicating that the model is able to detect most real objects. The values also converge towards 1.
- **metrics/mAP50B**:
    - The average precision at a threshold of 50 (mAP@50) shows good results, meaning that the model performs strongly in object identification.
- **metrics/mAP50-95B**:
    - The average accuracy at different ranges of IOU mAP@50-95 also shows significant improvement, demonstrating that the model is not only accurate but also robust in various scenarios.

3. Confusion Matrix

Original confusion matrix

The original confusion matrix shows the model's accuracy in classifying different types of traffic signs. The numbers on the main diagonal represent the number of correct predictions for each category, while the other numbers indicate classification errors. For example:

Green signal: 90 signals were correctly classified and 2 were incorrectly classified.

Speed Limit 100: 51 were correctly classified, but there were errors repeated in several speed limits.

Natural confusion matrix

The normalized confusion matrix shows the percentages of correct and incorrect classifications. For example:

Red light: 70% were correctly classified, while 30% were incorrectly classified.

4. Precision-Confidence Curve

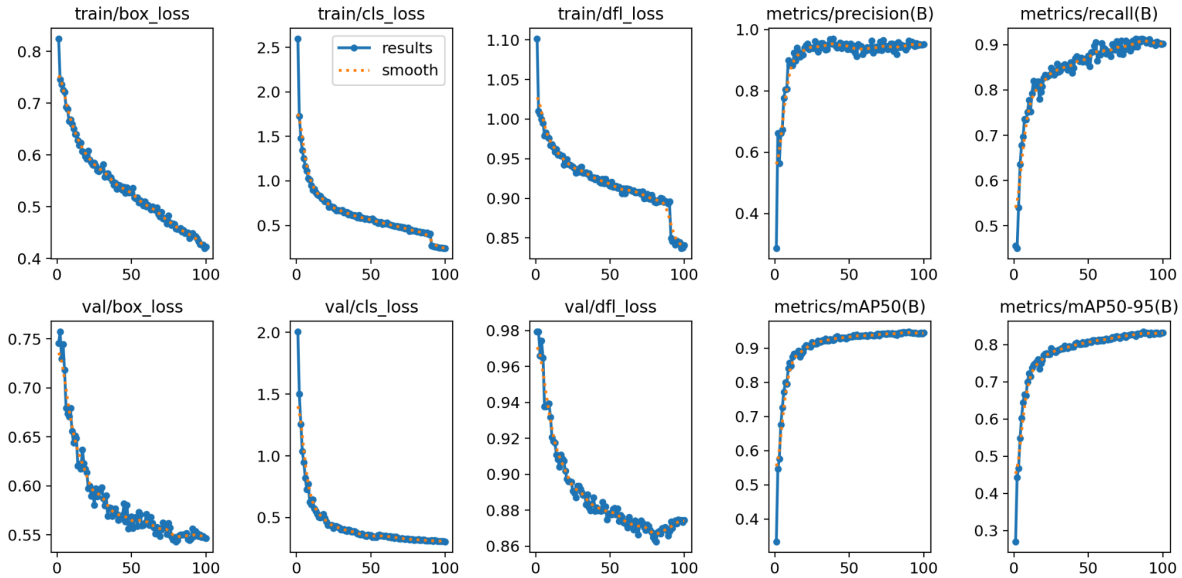
The accuracy and confidence curves illustrate the relationship between accuracy and different confidence levels. All classes achieve high accuracy, indicating the model's ability to accurately recognize various traffic signs.

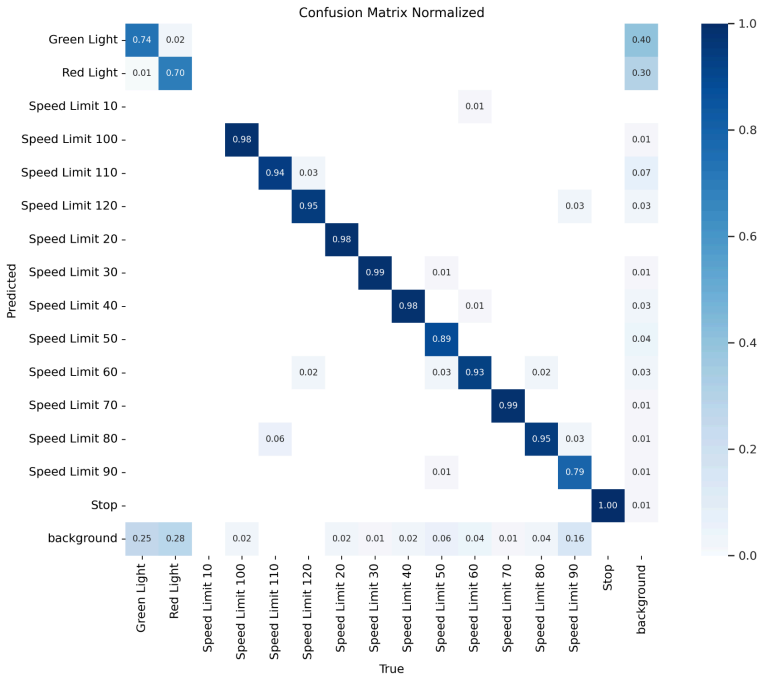
5. Precision-Recall Curve

The precision-recall curves illustrate the relationship between precision and recall for different classes. It can be seen that most classes achieve high precision (>0.9) and good recall, indicating that the model performs very efficiently.

6. Recall-Confidence Curve

These curves show how recall changes with varying confidence levels. The model exhibits good recall in most classes, demonstrating that it extracts objects effectively.





V. Model implementation:

a. Environment preparation:

Installing the required libraries:

- *Ultralytics* for *YOLO*.
- *cv2* for *OpenCV*.
- *cvzone* to display texts on images.

b. Connecting the camera

c. Program configuration

Specifying the IP address and port to which the data will be sent. In this code, we used:

```
ip_address = '192.168.137.72'
```

```
port = 4210
```

d. Downloading the form

```
model = YOLO(model_path)
```

Then we specified the path of the trained model.

e. Starting video capture

The camera opens and starts taking pictures in an endless loop:

```
success, img = cap.read()
```

If the image capture is unsuccessful, the episode is terminated.

f. Processing the results

The image is passed to the model to get the results:

```
results = model(img, stream=True)
```

The results are analyzed and information about bounding boxes, confidence, and class ID is extracted.

g. Drawing boxes on the image

If the confidence is greater than 0.8, a box is drawn around the signal and text is added explaining the class name and confidence:

```
cv2.rectangle(img, (x1, and1), (x2, and2), (255, 0, 255), 3)
```

h. Sending data

If the flag is "speed", the message is modified and sent over UDP.

```
send_udp_message(message, ip_address, port)
```

i. Frame rate calculation

Frame rate (FPS) is calculated to measure program performance:

```
fps = 1 / (new_frame_time - prev_frame_time)
```

j. Showing the pictures

The captured image is displayed with boxes and text using:

```
cv2.imshow("Image", img)
```

5.5 System Integration and UDP Communication Protocol

System integration involves ensuring that all components function properly and communicate effectively with each other. This includes:
Specify the communication protocol: Our use of the **UDP** to ensure fast and reliable communication between your computer and the **ESP8266** module .

UDP communication protocol

UDP (User Datagram Protocol) is one of the core protocols in the Internet Protocol (IP) suite. It is connectionless and does not require establishing a connection before data transmission, making it an ideal choice for this project, which requires high-speed, continuous data transmission.

UDP Protocol Features

1. Connectionless: **UDP** does not require a connection to be established before data can be transmitted, which reduces latency and increases connection speed.
2. **Simplicity: UDP** has a simple structure, with a fixed data header of only 8 bytes, which reduces the size of the data transmitted..
3. **No delivery guarantee: UDP** does not guarantee data delivery, ordering, or duplication protection, making it less complex and faster to perform..
4. **Safety Check: UDP:** Provides data integrity verification using checksum method.

Using UDP protocol

The project requires high speed and continuity of data transfer, so **UDP** is the optimal choice for the following reasons:

1. **Low delay time:** Since no connection is required, latency is greatly reduced, ensuring high-speed data transfer.
2. **High efficiency:** Thanks to its simple structure, the amount of data transmitted is reduced, which increases communication efficiency.
3. **Temporal sensitivity:** UDP is ideal for sending control data, which is a string of characters that represents the traffic signal recognized by the computer to the ESP8266.

In short, UDP provides an efficient and fast solution for data transmission in applications that require speed and continuity of performance.

- **Requirements:**

1. Wireless connection via computer hotspot.
2. **UDP** library: ***WiFiUdp.h*** for **ESP**.
3. Defining the data structure: The type of data Structure used is **strings**.
4. **Socket** library to send data from **Python IDE** to **ESP.**

Chapter Six

System testing and performance evaluation

6. System testing and performance evaluation

6.1 Test Methodology

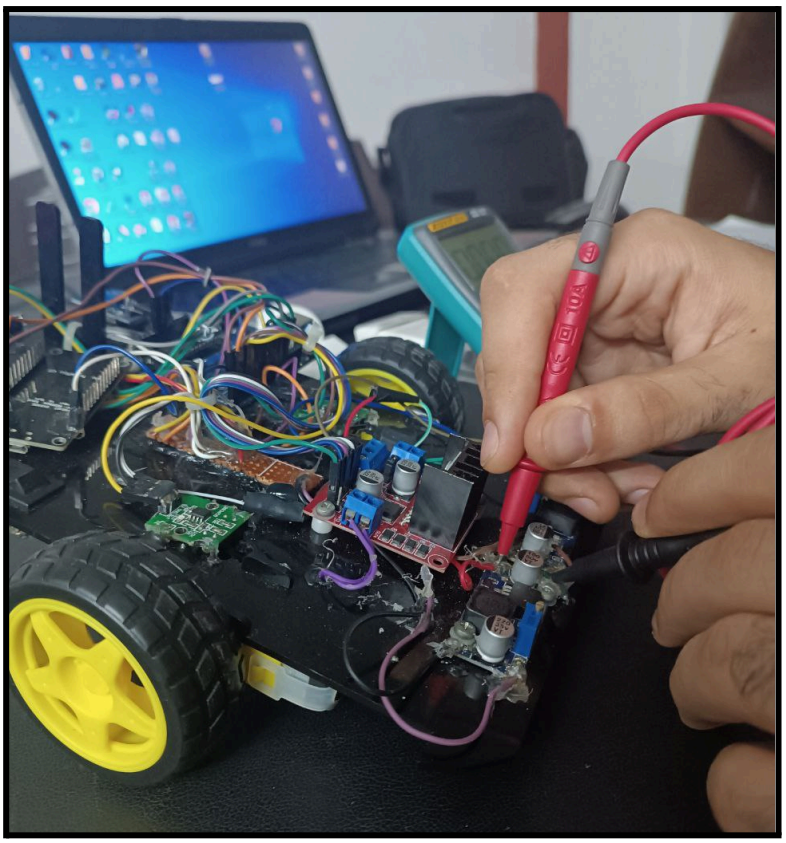
To comprehensively evaluate the system's performance, a series of tests covering various aspects of the system were conducted. These tests include:

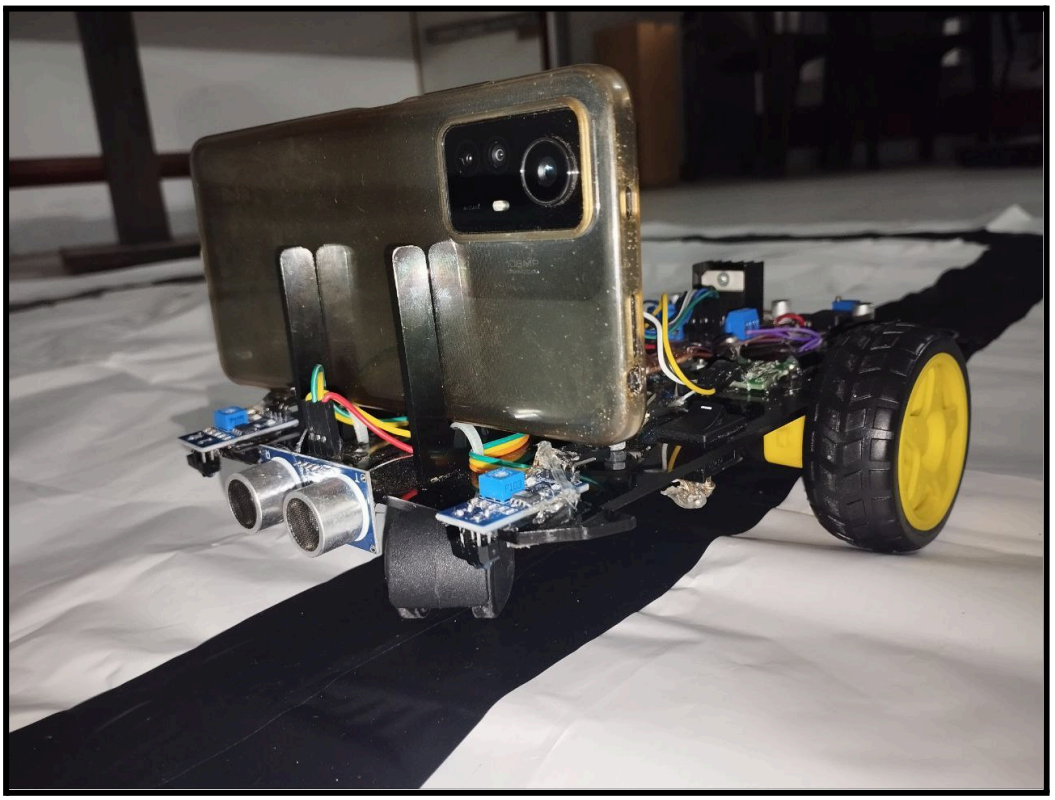
- **Unit tests:** Testing each component of the system separately to ensure it is working properly.
  Such as testing sensors separately..
- **Integration tests:** Testing how components work together to ensure the system works as a whole.
- **Performance tests:** Measuring system response speed, traffic signal detection accuracy, and lane tracking effectiveness.
- **Endurance tests:** Testing the system's ability to operate under different conditions, such as changes in lighting, the presence of obstacles, or high speeds.

To perform these tests, the following steps were followed:

- **Design test scenarios:** Designing a set of scenarios that cover various possible use cases for the system.
- **Setting up the test environment:** Simulating real-world conditions as closely as possible.
- **Test execution:** Executing each test scenario and recording the results.
- **Analysis of results:**Analyze the results to identify the strengths and weaknesses of the system.

6.2 Test results





These results include:

- **Traffic sign detection accuracy:** The percentage of signals correctly recognized.
- **Response time:** The time it takes for the system to detect a traffic signal and make an appropriate decision.
- **Line tracking error rate:** The percentage of time periods the vehicle deviated from the drawn line.
- **Energy consumption:** The amount of energy consumed by the system.
- **Qualitative notes:** Any comments on system performance under specific conditions.

6.3 Performance Analysis

Based on the test results, we can draw the following conclusions:

- **Strengths:**

  1. Responding with trust:

    - Most classes, such as "Green Light" and "Speed Limit 100," appear to achieve high F1 values (over 0.9) at confidence levels between 0.3 and 0.8.
    - Other categories, such as "Red Light" and "Speed Limit 20," also perform well, although they may require a higher confidence level to achieve the same F1 level.

2. Performance of different categories:

- Green Light: High F1 indicates the model's ability to accurately recognize this class.
- Speed Limit 100: It also shows excellent performance, indicating that the model handles well in this class.
- Other cues: Some classes may need improvement, as the F1 for some of them is less than 0.8 at low confidence levels.

3. High accuracy:

The results demonstrate high accuracy in traffic sign detection, making the model suitable for practical applications.

4. Continuous improvement:

Decreasing loss over the course of training indicates continued performance improvement.

5. Good response to new data:

Generalization ability demonstrates the model's ability to handle unfamiliar data.

6. The robot's response speed when turning even at high speeds

7. High accuracy in detecting traffic signals: The system demonstrated a high ability to recognize different traffic signs under various lighting conditions and viewing angles.

8. Quick response: The system responded quickly to changes in the surrounding environment, such as the appearance of an obstacle or a change in traffic lights.

9. System flexibility: The system has proven its flexibility and ability to operate in various environmental conditions.

- **Weaknesses:**

  1. Error rate in some categories:
  The model may need further training to improve performance on some high difficulty categories.

  2. Need for a larger dataset:
  Improving performance may require expanding the dataset to include more traffic signal diversity.

  3. The system was affected under certain low light circumstances:
  The system had some difficulty recognizing traffic signs in very low light conditions.
  4. Sensitivity to interference:
  The system is affected by external interference such as noise or unexpected obstacles.

  5. Limits in dealing with complex cases: The system may have difficulty handling complex situations that require quick, multiple decisions.

In general, The system works well and achieves its objectives.

However, there are some aspects that could be improved in the future.

6.4 Analysis of results

After conducting comprehensive tests and evaluating the system's performance, we can draw several important conclusions. The results demonstrated that the system is capable of achieving its primary design objectives: accurately tracking black lines, recognizing traffic signals, and making appropriate decisions based on this information.

6.5 Challenges and limitations

The project faced many challenges and constraints during the design and implementation phases, including:

- **Developing a deep learning model: YOLOv8** model development required to learn traffic signs took a lot of effort and required a lot of data.
- **Set parameters:** Tuning various system parameters, such as motor control parameters and deep learning model parameters, was a complex process that required a lot of trial and error.
- **Energy saving:** There was a challenge in achieving a balance between system performance and power consumption.
- **Limited technical capabilities.**

6.6 Comparison with existing systems

Compared to existing educational systems, the proposed system is still in its initial development stage, but it has some features that distinguish it from other systems, such as:

- **Low cost:** Compared to other systems this one has a much lower cost.
- **Ease of modification:** The system can be easily modified to add new features or improvePerformance or even switch some systems.
- **Focus on specific tasks:**The system focuses on specific tasks, such as lane tracking and traffic sign recognition, making it simpler and easier to implement.

However, the proposed system faces some challenges compared to the previous ones, The system still lacks many of the advanced capabilities found in other systems.

Chapter Seven

Conclusion and future work

7.Conclusion and future work.

7.1 Summary of Achievements

This project aims to design and implement a two-wheeled differential robot capable of tracing a path, recognizing traffic signals, and making appropriate decisions. Several achievements were made during this project, including:

- **Design and implementation of an integrated system:** An integrated system combining hardware and software components is designed and implemented to perform the required tasks.
- **Developing a model for traffic sign recognition:** A **YOLOv8** model has been developed to recognize traffic signs with high accuracy.
- **Development of control algorithms:** Algorithms have been developed to control robot's speed following the black line.
- **System performance evaluation:** The system performance was comprehensively evaluated through a range of tests.

7.2 Conclusions

The results of this project demonstrated that it is possible to design and implement a simple system for a self-driving car using available components. The system demonstrated its ability to perform the basic tasks assigned to it accurately and efficiently. However, several technologies must be added to complete the system.

73. Suggestions for future improvement and development

There are many aspects that can be improved and developed in the future, including:

- **Improve traffic sign recognition accuracy:** The accuracy of the model can be improved by increasing the size of the training data and experimenting with different neural network architectures.
- **The project can be improved and brought closer to real self-driving cars by using an artificial intelligence model to detect paths, such as convolutional neural networks, where the output of the network is the orientation angle of the robot.**

  In fact, we have already trained a model that works in this way, but the results were not accurate enough to be applied to our current project, and also due to the weakness of technical capabilities.

- **Increase the system's ability to handle changing environmental conditions:** The system must be equipped with better capabilities to handle changing environmental conditions.Other, and moving obstacles.
- **Developing more complex decision-making algorithms**
- **Error rate in some classes:** The model may need more training to improve performance in some high-difficulty classes.
- **Changing the current controller to one capable of running AI models.**
  Such as ***RASPBERRY PI 5***
- **Adding high-level control algorithms**
- **Adding additional sensors:** Additional sensors, such as **LiDAR (Light Detection and Ranging)**, can be added to improve the system's ability to understand the surrounding environment.

Chapter Eight

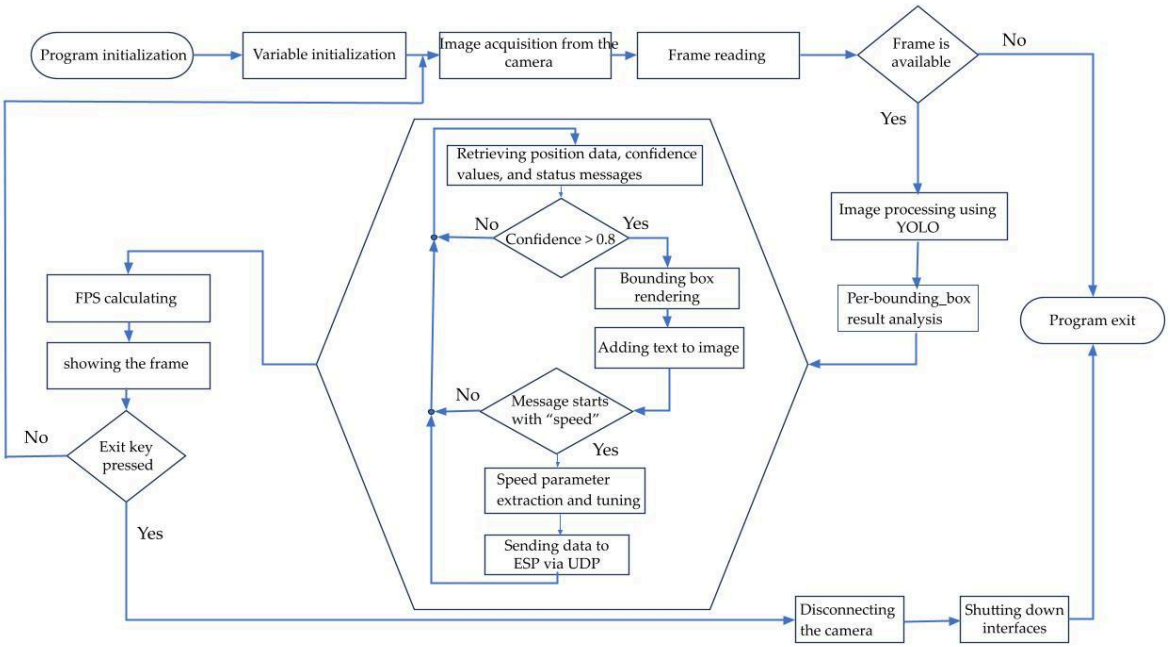
Appendices and References

8.Appendices and References

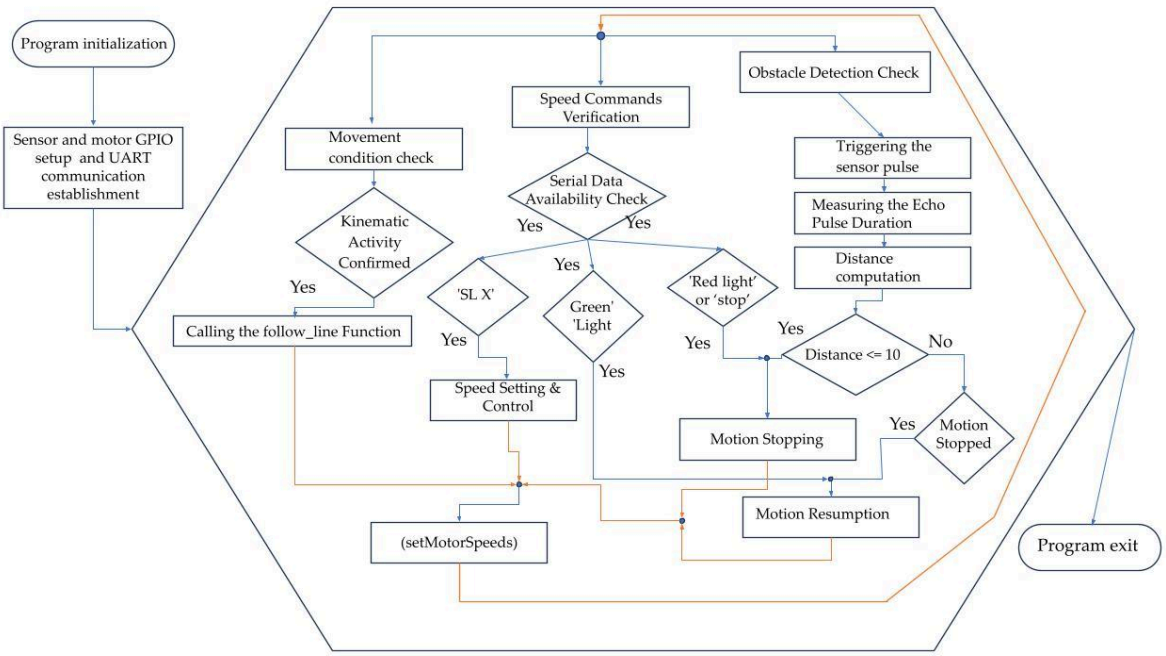
8.1 System schematics

- **Flowcharts:**

  1. Computer vision system flowchart



2. Operating system flowchart:



8.2 Source code

- **Arduino Code: The complete code for programming the Arduino board.**

```
// Pin definitions
#define TRIG_PIN 8
#define ECHO_PIN 13
#define LEFT_MOTOR_EN 10
#define LEFT_MOTOR_IN1 6
#define LEFT_MOTOR_IN2 7
#define LEFT_IR_PIN 12
#define RIGHT_MOTOR_EN 9
#define RIGHT_MOTOR_IN1 4
#define RIGHT_MOTOR_IN2 5
#define RIGHT_IR_PIN 11
float clipration = 1.19;
int currentSpeed = 0;
bool isMoving = true;
String receivedString = "";
void setup() {
  pinMode(LEFT_IR_PIN, INPUT);
  pinMode(RIGHT_IR_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN,INPUT);
  pinMode(LEFT_MOTOR_EN, OUTPUT);
  pinMode(LEFT_MOTOR_IN1, OUTPUT);
  pinMode(LEFT_MOTOR_IN2, OUTPUT);
  pinMode(RIGHT_MOTOR_EN, OUTPUT);
  pinMode(RIGHT_MOTOR_IN1, OUTPUT);
  pinMode(RIGHT_MOTOR_IN2, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  receiveSpeedLimit();  // Check for commands first
  checkObstacle();  // Check for obstacles
  if (isMoving) {
    followLine(); }}
```

```
void followLine() {
  int leftIR = digitalRead(LEFT_IR_PIN);
  int rightIR = digitalRead(RIGHT_IR_PIN);

  if (leftIR == LOW && rightIR == LOW) {
    // Move forward
    setMotorSpeeds(currentSpeed, currentSpeed);
  } else if (leftIR == HIGH && rightIR == LOW) {
    // Turn right
    setMotorSpeeds(0, currentSpeed);
  } else if (leftIR == LOW && rightIR == HIGH) {
    // Turn left
    setMotorSpeeds(currentSpeed, 0);
  } else {
    // Stop (both sensors on white)
    setMotorSpeeds(0, 0);}}

void checkObstacle() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN,HIGH);
  float distance = duration * 0.034 / 2;
  if (distance <= 10) {
    Serial.println("There is an obstacle");
    setMotorSpeeds(0, 0);
    isMoving = false;
  } else if (!isMoving) {
    isMoving = true;
    Serial.println("Obstacle removed, resuming movement"); }}

void receiveSpeedLimit() {
  while (Serial.available()) {
    char c = Serial.read();
    if (c == '\n') {
      processReceivedData(receivedString);
      receivedString = "";
    } else {
```

```
    receivedString += c; }}}
void processReceivedData(String data) {
  data.trim();
  if (data.startsWith("SL ")) {
    currentSpeed = data.substring(3).toInt();
    isMoving = true;
    Serial.print("Speed set to: ");
    Serial.println(currentSpeed);
  } else if (data == "Stop" || data == "Red Light") {
    isMoving = false;
    setMotorSpeeds(0, 0);
    Serial.println("Stopped");
  } else if (data == "Green Light") {
    isMoving = true;
    Serial.println("Moving"); }}
void setMotorSpeeds(int leftSpeed, int rightSpeed) {
  if (!isMoving) {
    leftSpeed = 0;
    rightSpeed = 0; }

  leftSpeed = leftSpeed * clipration;
  leftSpeed = map(leftSpeed, 0, 304, 0, 255);
  analogWrite(LEFT_MOTOR_EN, leftSpeed);
  analogWrite(RIGHT_MOTOR_EN, rightSpeed);

  digitalWrite(LEFT_MOTOR_IN1, leftSpeed > 0 ? HIGH : LOW);
  digitalWrite(LEFT_MOTOR_IN2, leftSpeed > 0 ? LOW : HIGH);
  digitalWrite(RIGHT_MOTOR_IN1, rightSpeed > 0 ? HIGH : LOW);
  digitalWrite(RIGHT_MOTOR_IN2, rightSpeed > 0 ? LOW : HIGH);

  Serial.print("Motors set to - Left: ");
  Serial.print(leftSpeed);
  Serial.print(", Right: ");
  Serial.println(rightSpeed); }
```

- **ESP8266 Code: The complete code for programming the ESP8266 module.**

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "HUSSAIN";
const char* password = "123456789";

WiFiUDP Udp;
unsigned int localUdpPort = 4210;  // local port to listen on

char incomingPacket[255];  // buffer for incoming packets

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

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");

  // Start UDP
  Udp.begin(localUdpPort);
  Serial.printf("Now listening at IP %s, UDP port %d\n", WiFi.localIP().toString().c_str(),
localUdpPort);
}

void loop() {
  int packetSize = Udp.pressPacket();
  if (packetSize) {
    // Receive incoming UDP packet
    int only = Udp.read(incomingPacket, 255);
    if (only > 0) {
      incomingPacket[len] = 0;  // Null-terminate the string
    }
    Serial.printf("UDP packet contents: %s\n", incomingPacket);
```

```
// Forward the received string to Arduino Uno via Serial
Serial.println(incomingPacket); }}
```

- **Python code: The code used to train and run the modelYOLOv8.**

```
from webcam_capture import webcam_capture
from detect_traffic_signs import detect_traffic_signs
from send_data import send_udp_message
importcv2
if __name__== "__main__":
  class_names = [
     'Green Light', 'Red Light', 'Speed Limit 10', 'Speed Limit 100', 'Speed Limit 110', 'Speed Limit 120',
     'Speed Limit 20', 'Speed Limit 30', 'Speed Limit 40', 'Speed Limit 50', 'Speed Limit 60', 'Speed Limit
70',
     'Speed Limit 80', 'Speed Limit 90', 'Stop' ]
    model_path = "yolov8s smaller data set last resutls/yolov8s smaller data set last
resutls/416traininglast/weights/best.pt"

  cap = webcam_capture()
  detect_traffic_signs(cap, model_path, class_names)
```

- **Traffic Signs Detection Code detect_traffic_signs.py**

```
ip_address='192.168.137.72'
port =4210

def detect_traffic_signs(cap, model_path, class_names):
  model = YOLO(model_path)
  prev_frame_time = 0
  new_frame_time = 0
  while True:
     new_frame_time = time.time()
     success, img = cap.read()
     if not success:
        break
     results = model(img, stream=True)
     for r in results:
```

```
        boxes = r.boxes
        for box in boxes:
            # Bounding Box
        x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
w, h = x2 - x1, y2 - y1
            # Confidence
            conf = math.ceil((box.conf[0] * 100)) / 100
            # Class id
            cls = int(box.cls[0])
            if conf > 0.8:
                cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
                # Class name
                class_name = class_names[cls] if cls < len(class_names) else "Unknown"
                cvzone.putTextRect(img, f'{class_name} {conf}', (max(0, x1), max(35, y1)), scale=1,
thickness=1)
                message=class_name
                if message.startswith('Speed'):
l= message.split()
                    num=int(l[2])
                    num = int(70+ (num -50) * (255 - 70) / (120 - 50))
                    message= 'SL '+str(number)
                send_udp_message(message, ip_address, port)
    fps = 1 / (new_frame_time - prev_frame_time)
    prev_frame_time = new_frame_time
    print(fps)
    cv2.imshow("Image", img)
    if cv2.waitKey(1) == word('q'):
        break
  cap.release()
  cv2.destroyAllWindows()
```

- **Image reception and configuration code webcam_capture.py**

```
importcv2
def webcam_capture():
  cap = cv2.VideoCapture(1)  # For Webcam
  cap.set(3, 1280)
  cap.set(4, 720)
```

8.3 References:

R. Kumar, A. Gupta and R. D, "Traffic Sign Detection Using YOLOv8," *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDC)*, Bengaluru, India, 2024, pp. 756-761, doi: 10.1109/IDCIoT59759.2024.10467849.

Du, Songjie & Pan, Weiguo & Li, Nuoya & Dai, Songyin & Xu, Bingxin & Liu, Hongzhe & Xu, Cheng & Li, Xuewei. (2024). TSD-YOLO: Small traffic sign detection based on improved YOLO v8. IET Image Processing. n/a-n/a. 10.1049/ipr2.13141.

Ultralytics. 2023. "YOLOv8." GitHub. https://github.com/ultralytics/ultralytics
Ultralytics. 2023. " YOLOv8." https://docs.ultralytics.com/

Thanks and appreciation

With pride and gratitude we would like to extend our sincere gratitude and appreciation to **Dr. Ali Zaher**, our graduation project supervisor, who spared no effort in guiding and supporting us throughout the project. We also extend our deep appreciation to the professors and engineers in the **Mechatronics Department**, who were always ready to provide assistance and advice, which greatly contributed to this achievement and graduation. Your continued support and belief in our abilities have had a significant impact on our success.**Thank you all.**

We would like to thank **Dr. Nael Daoud**, who was dedicated to teaching, guiding and supporting us.
And **Dr. Muhammad Khnisi**, who has been a great asset to us since the third year until this moment.
And **Dr. Issa Al-Ghannam** for his valuable support and guidance, which was a fundamental pillar in the success of our project.
Dr. Fadi Mutawaj has a unique style of explaining and conveying information.
And **Dr. Heba Halawa**, who helped us understand the science of image processing and automatic control, and we applied many of the principles she taught us.
And the professors:
Eng. Karam Ghazal .
Eng. Ahmed Ahmed
Eng. Ali Aswad.
Eng. Zeina Ali
Eng. Mohammed Moeen Ibrahim
Professor and wonderful friend **Eng. Adam Al-Duwairi.**
You were true teachers and friends.