
IoT Application System

# Smart lights

**Smart traffic controller project**

# Smart lights

**Team contribution**

| | |
|---|---|
| SHOKHZODBEK | 12200295 (Team leader, Presenter) |
| NOZIMJON | 12204507 (Facilitator and recorder) |
| SHAKHZODBEK | 12200293 (Survayer) |
| MIRJAVKHARBEK | 12214737 (Developer) |
| MARC | 12235365 (Developer |
| BAKHTIYORJON | 12200288 (Developer) |

# Project background

## MAIN CONCEPT

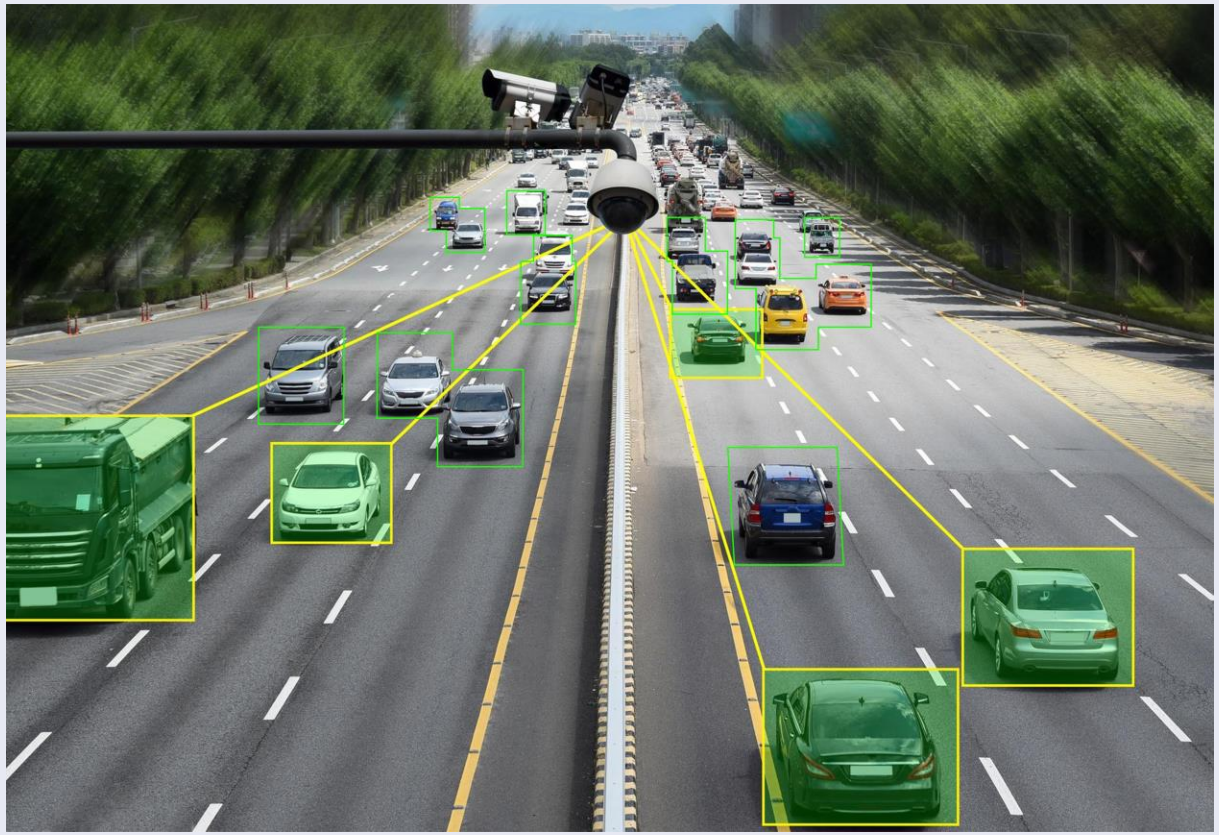
The Smart Traffic Light integrates **Raspberry Pi, OpenCV, and YOLO camera modules** to revolutionize traffic management. Using real-time object detection, it dynamically adjusts signal timings for both cars and pedestrians, optimizing traffic flow.

This innovative system harnesses cutting-edge technology to create a responsive, efficient, and adaptive traffic control solution for modern urban environments.

# Identified Challenge

Traditional traffic lights suffer from **static timings** that fail to adjust to fluctuating traffic loads. This rigidity leads to congestion during peak hours and underutilization during low traffic times, **causing delays, frustration, and heightened accident risks**.

An adaptive and intelligent traffic control system is crucial to address these inefficiencies.



# Solutions

**Smart Traffic Lights:**
Implement an intelligent traffic light system using sensors and machine learning algorithms to dynamically adjust signal timings based on real-time traffic conditions.

**Integrated Transportation Systems:**
Create a cohesive infrastructure connecting traffic lights, public transportation, and emergency services. Utilize data analytics to optimize traffic flow and enhance overall urban mobility.

# Web(Socket) based interface

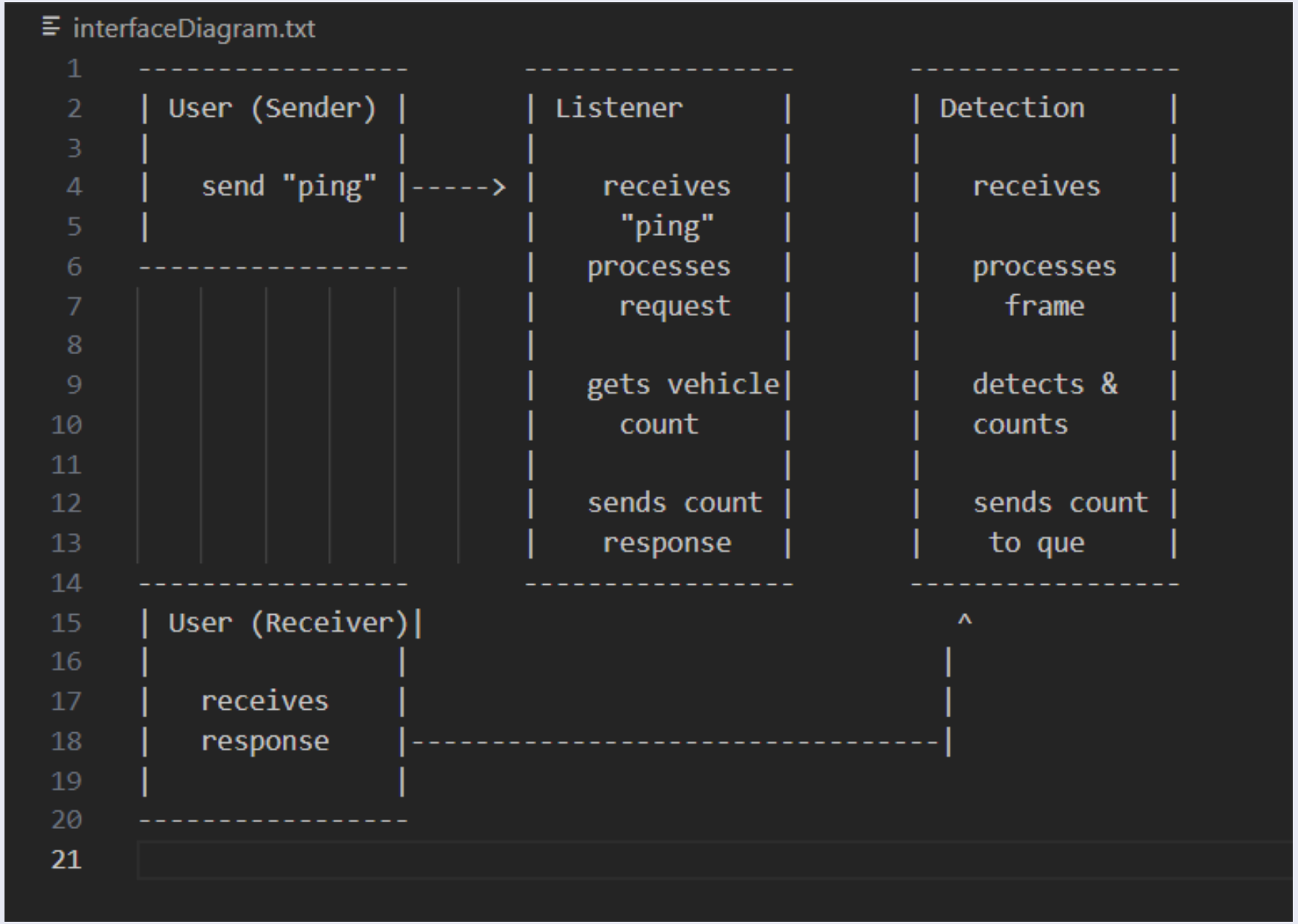
**Listener:**

Creates a listener using sockets, awaiting 'ping' messages from connected clients. Upon receiving a 'ping', it retrieves the current time and a vehicle count (placeholder logic provided), crafts a response message, and sends it back to the client. It listens continuously for incoming messages, processing them accordingly

**User:**

script establishes a socket connection with a listener at a specified address and port. It sends a 'ping' message to the listener, waits for a response, and prints the received response. Finally, it closes the connection with the listener.

interfaceDiagram.txt



Container.py

**Fixed Size Stack:**
This Python code creates a fixed-size stack using ***deque*** from ***collections***. It limits the stack to a maximum size and allows adding elements ***(put)*** while keeping track of the most recent element. The ***get*** function retrieves the latest element or returns None if the stack is empty.

container.py > ...

```python
from collections import deque

class FixedSizeStack:
    def __init__(self, max_size):
        self.max_size = max_size
        self.container = deque(maxlen=max_size)

    def put(self, data):
        self.container.append(data)  # Append data

    def get(self):
        if not self.container:
            return None  # Return None if the cont
        return self.container[-1]  # Get the newes

```

# Implementation

Our main Python script uses Raspberry Pi's GPIO for traffic light control. It employs multiprocessing to handle object detection, listening for data, and simulating traffic changes. The code demonstrates remote priority adjustments for the traffic controller and ensures GPIO cleanup before exiting.

myMain.py > ...

```
1  > import RPi.GPIO as GPIO ···
14
15 > def simulate_traffic(controller): ···
23
24 > if __name__ == "__main__": ···
61
62   # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
63   #
64   #  create -> TrafficLightController (controller)
65   #  create -> FixedSizeStack (data queue)
66   #  get source
67   #
68   #  # # # Multriproccessing # # # # #
69   #  process -> Detection(*ergs=data_queue) | pushes count to queue
70   #
71   #  process -> Listener(*ergs=data_queue)  | gets count from queue
72   #
73   #  process -> Simulate_trafficlight(*ergs=controller)
74   #
75   #  cleanup
```

# Object detection

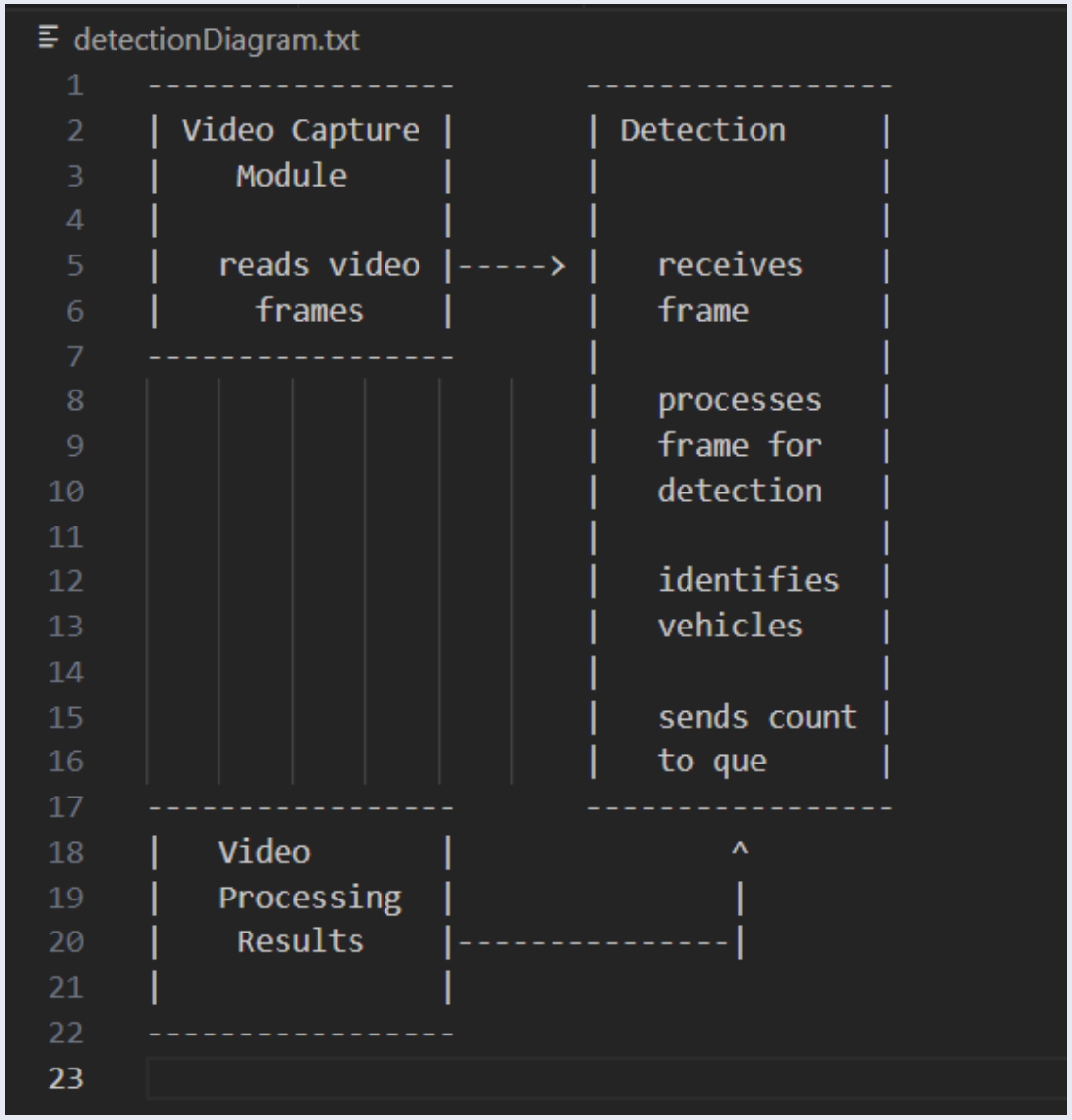
Implementing a YOLO-based system for real-time vehicle detection and counting in video streams.

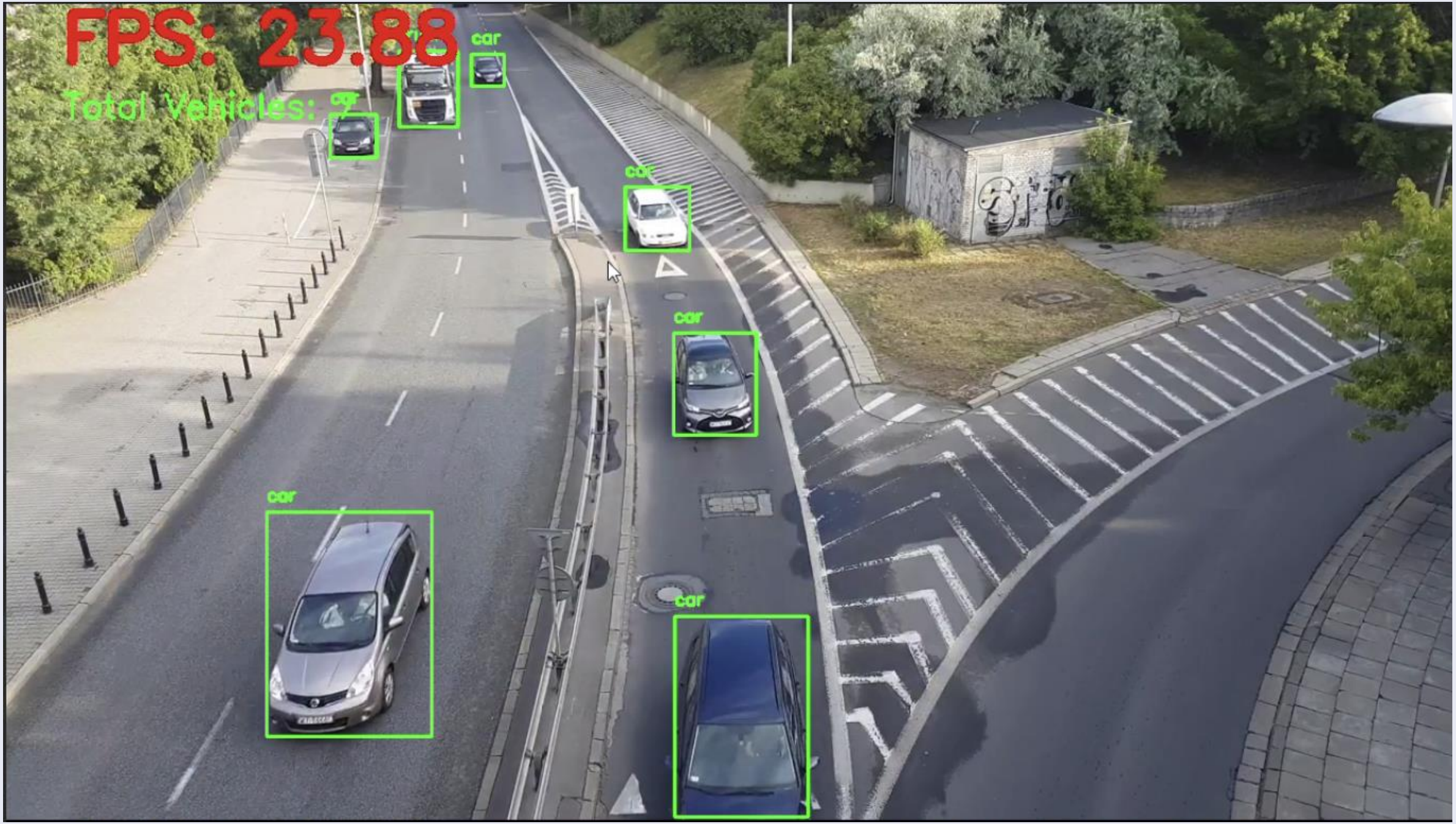
Detects vehicles in every Nth frame, annotates with bounding boxes, and counts the identified vehicles.

Data_queue for storing vehicle counts.

Creates an output video file with annotated vehicle counts and real-time FPS display.

detectionDiagram.txt

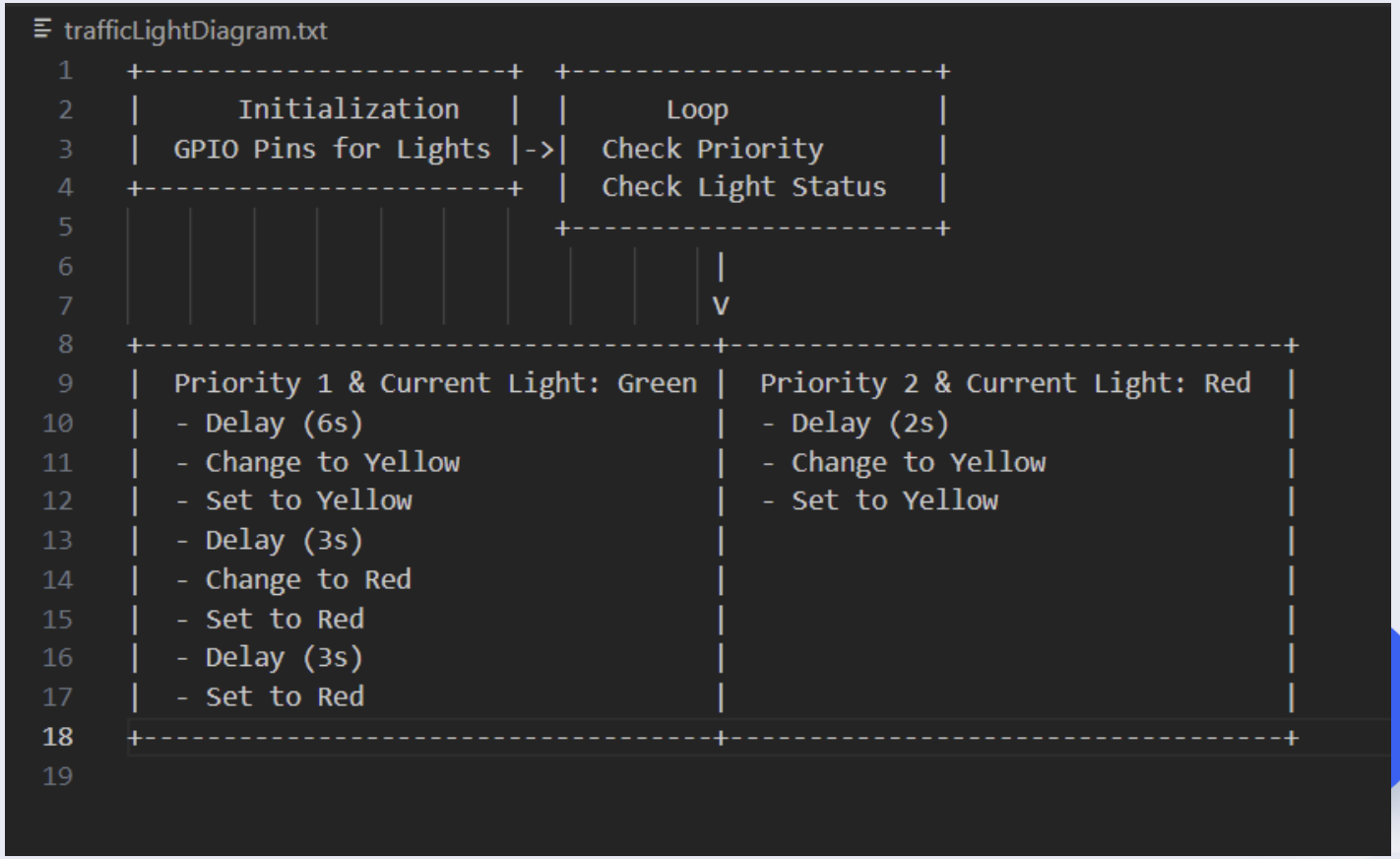



FPS: 23.88
Total Vehicles: 7
car
car
car
car
car

# Traffic light

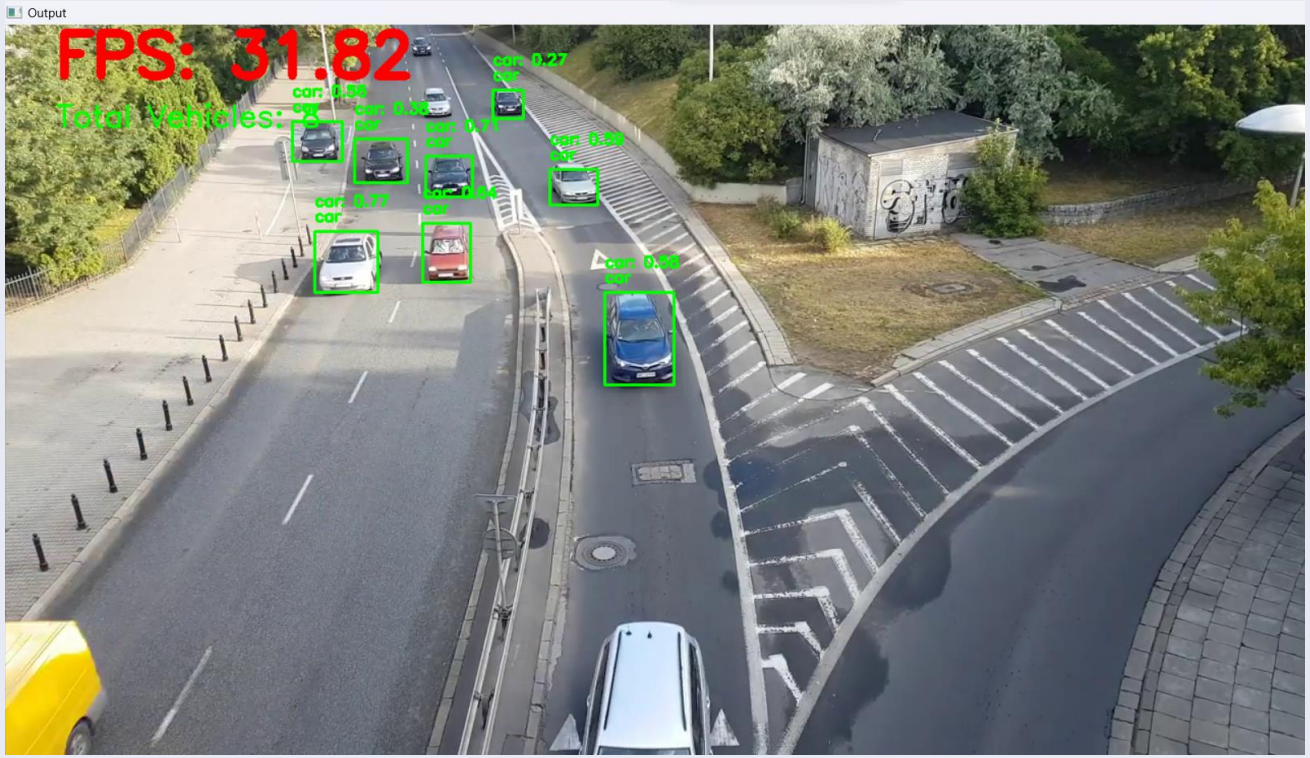
TrafficLightController manages traffic light colors based on vehicle count and priority settings, altering lights via GPIO pins to optimize traffic flow.

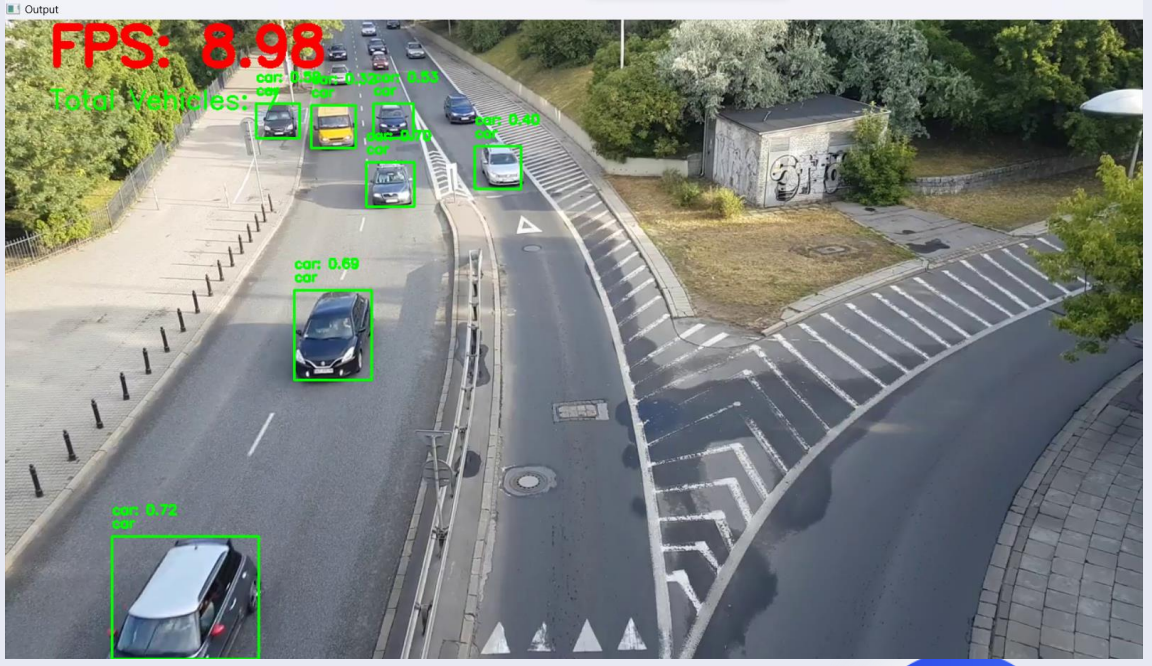
trafficLightDiagram.txt



# Performance

Object detection function is running on average 8 FPS




And can process 30FPS video stream after optimizations

# Performance

For precision testing we used comparative method.
Note: We are checking the precision of our system while using minimum, YOLOv3, and comparing to the lates YOLOv8 results.

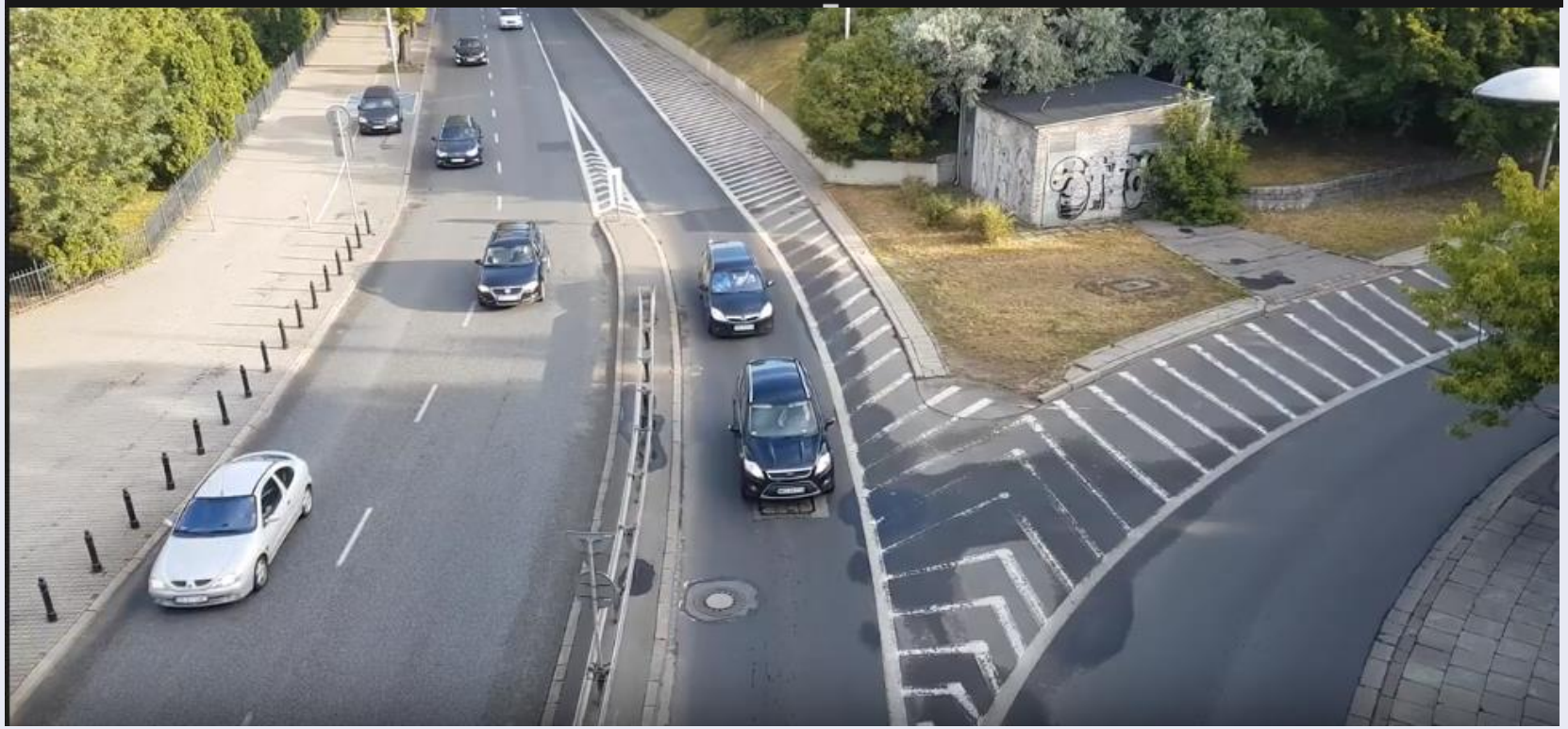
- On good visibility the Precision was : 75%
- On foggy condition the Precision was : 50%
- On foggy condition with very heavy traffic load the Precision was : 62%
- On night the Precision was : 35%
- On snow the Precision was : 58.8%

# Performance

On good visibility the Precision was : 75%

```
0: 320x640 6 cars, 1 frisbee, 112.1ms
Speed: 0.0ms preprocess, 112.1ms inference, 1.0ms postprocess per image at shape (1, 3, 320, 640)

0: 320x640 8 cars, 140.5ms
Speed: 0.0ms preprocess, 140.5ms inference, 0.0ms postprocess per image at shape (1, 3, 320, 640)
Precision: 0.7500
```



# Performance

On foggy condition the Precision was : 50%

```
0: 640x448 2 cars, 123.1ms
Speed: 0.0ms preprocess, 123.1ms inference, 0.0ms postprocess per image at shape (1, 3, 640, 448)

0: 640x448 3 cars, 1 truck, 193.3ms
Speed: 0.0ms preprocess, 193.3ms inference, 0.0ms postprocess per image at shape (1, 3, 640, 448)
Precision: 0.5000
```

# Performance

On foggy condition with very heavy traffic load the
Precision was : 62%

```
0: 416x640 14 cars, 131.8ms
Speed: 0.0ms preprocess, 131.8ms inference, 1.0ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 20 cars, 1 bus, 166.2ms
Speed: 0.6ms preprocess, 166.2ms inference, 0.0ms postprocess per image at shape (1, 3, 416, 640)
Precision: 0.6190
```

# Performance

On night the Precision was : 35%

```
0: 384x640 6 cars, 117.0ms
Speed: 4.0ms preprocess, 117.0ms inference, 0.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 14 cars, 147.4ms
Speed: 0.0ms preprocess, 147.4ms inference, 0.0ms postprocess per image at shape (1, 3, 384, 640)
Precision: 0.3571
```

# Performance

On snow the Precision was : 58.8%

```
0: 384x640 10 cars, 117.0ms
Speed: 0.0ms preprocess, 117.0ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 17 cars, 163.0ms
Speed: 0.0ms preprocess, 163.0ms inference, 0.0ms postprocess per image at shape (1, 3, 384, 640)
Precision: 0.5882
```

# Challenges

Adapting to the processing constraints of Raspberry Pi 3 posed a significant challenge. While maintaining accuracy, our solution started with YOLOv8 and progressively optimized down to YOLOv3-tiny to ensure real-time processing, striking a delicate balance between accuracy and speed.

# Challenges

Developing an efficient traffic light logic proved to be another hurdle in our project. Despite efforts, the current implementation remains suboptimal, requiring further refinement to strike the right balance between traffic flow optimization and real-time responsiveness.

# Challenges

Installing dependencies and required packages, such as ultralytics and pytorch

# future

- Develop a novel model aimed at directly detecting traffic density, shifting focus from individual vehicle detection to analyzing overall traffic flow and density.
- Test the system using simulated environments to assess its functionality under controlled conditions, allowing for iterative improvements and validations.
- Optimize the traffic logic within the system to enhance efficiency and responsiveness, ensuring it accurately adapts to varying traffic scenarios in real-time