Smart Traffic Light System for Curry Avenue and Arteche Boulevard in Catbalogan City: A Raspberry Pi-Based Approach

Catbalogan City faces significant traffic issues due to its growing population and increased economic activity. This problem is especially acute at the Curry Avenue and Arteche Boulevard intersections, where congestion peaks during busy hours. With no traditional traffic management systems in place, delays are frequent at these critical junctions (Allequer, 2017).

To alleviate congestion and enhance safety in these busy areas, the Smart Traffic Light System for Curry Avenue and Arteche Boulevard proposes an innovative solution. By leveraging Raspberry Pi technology and AI, the system dynamically controls traffic signals based on lane congestion and the timing of previous signal changes. This approach aims to maximize efficiency and reduce vehicle wait times by prioritising lanes with the heaviest traffic flow. Equipped with cameras and artificial intelligence, the system identifies vehicle presence and adjusts signal timing to promote smoother traffic flow.

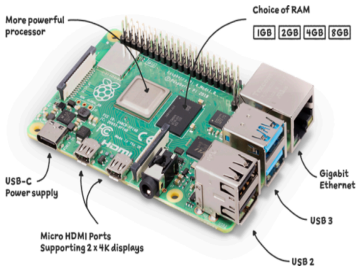
TensorFlow Lite is a lightweight, open-source framework for deploying machine learning models on mobile, embedded, and IoT devices. It is a part of the TensorFlow ecosystem, designed to enable the execution of TensorFlow models on devices with limited computational resources. TensorFlow Lite provides tools for model conversion, optimization, and efficient on-device inference. Implementing a smart traffic light system using TensorFlow Lite on a Raspberry Pi involves several steps including data collection, model training, model conversion, deployment, and integration with the traffic light system.This system can help optimise traffic flow at Curry Avenue and Arteche Boulevard in Catbalogan City, improving traffic efficiency and reducing congestion.

II. OBJECTIVES

- Develop a Smart Traffic Light System tailored for Curry Avenue and Arteche Boulevard in Catbalogan City, utilising Raspberry Pi-based technology and machine learning algorithms.
- Conduct testing through simulations to determine how many vehicles cause congestions in the intersection.
- Implement the system through prototyping/simulations or Miniature of the intersection.

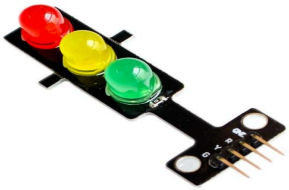
III. SYSTEMS DESIGN

MATERIALS:



Raspberry Pi4

Web Camera



LED Lights

Jumper Wires

Raspberry Pi4

➢ The Raspberry Pi 4 can play a crucial role in a smart traffic monitoring system due to its versatility, computational power, and affordability. It serves as an effective, low-cost solution for implementing a smart traffic monitoring system. Its ability to integrate with various sensors, process data locally, and control traffic lights in real-time makes it a powerful tool for improving traffic management and reducing congestion.

Web Camera

➢ A webcam is a digital camera that captures video and audio data and transmits it in real-time over the internet. It is commonly used for video conferencing, live streaming, online meetings, and recording videos.

LED Traffic Lights

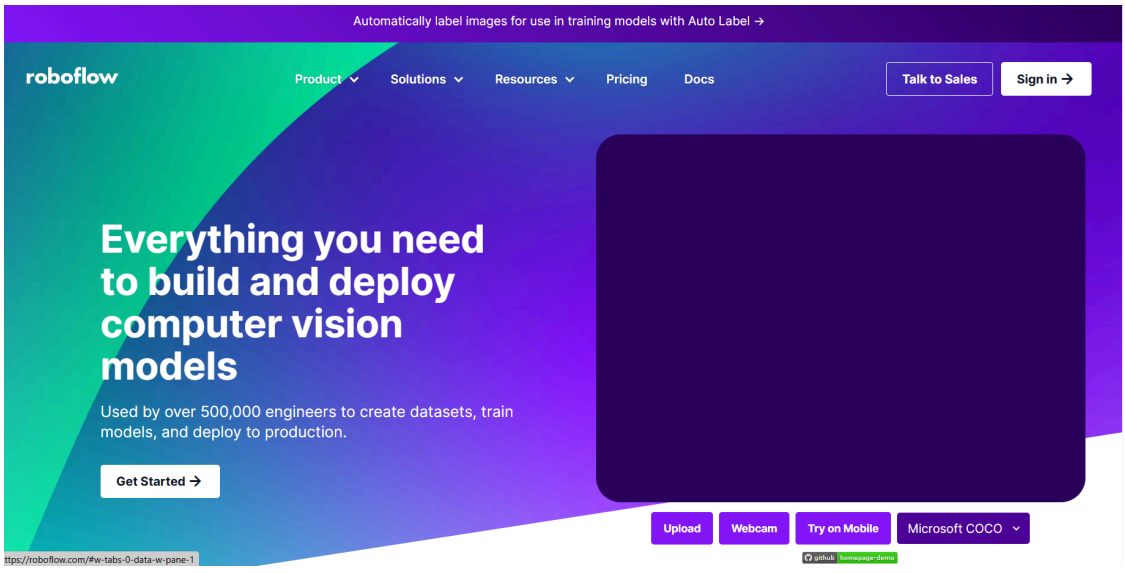
➢ Smart traffic systems utilize LED traffic lights for dynamic signal timing, adjusting based on real-time traffic conditions to reduce congestion and improve flow. These lights can be integrated with sensors to detect vehicles and pedestrians, enabling responsive control.

Jumper Wires

➢ Jumper wires are essential components in smart traffic monitoring systems, facilitating reliable and efficient connections between various electronic components and sensors. These wires are used to establish connections on circuit boards, ensuring communication between microcontrollers, sensors, LEDs, and other crucial elements of the system.

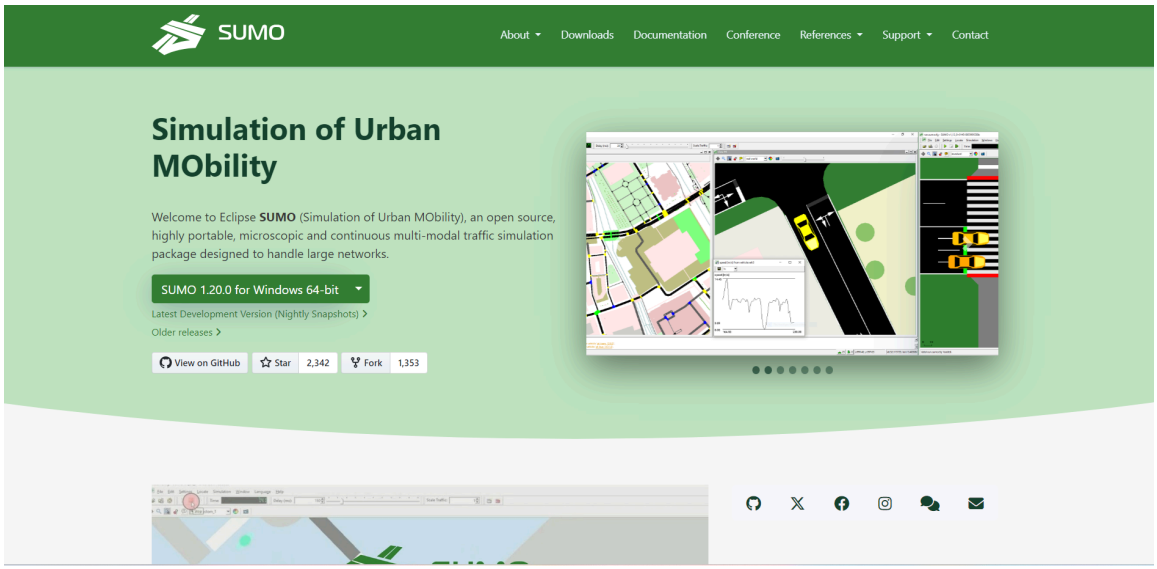
METHODS USED:

ROBOFLOW

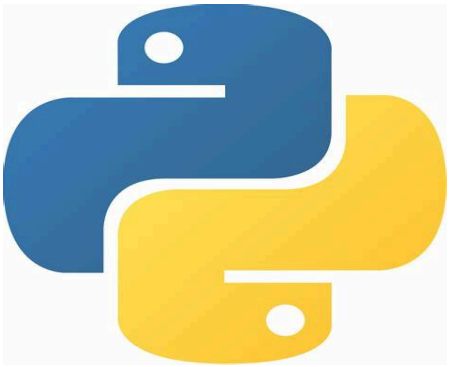


❖ Roboflow is a comprehensive platform that streamlines the workflow for creating and deploying computer vision models. It offers robust tools for dataset management, including annotation and versioning, to track changes and improvements. The platform provides image preprocessing and augmentation capabilities, enhancing datasets with transformations such as rotations and flips. Roboflow supports major machine learning frameworks like TensorFlow, PyTorch, and Keras, and includes AutoML features for easy model building.

SUMO Simulator and Pygame:



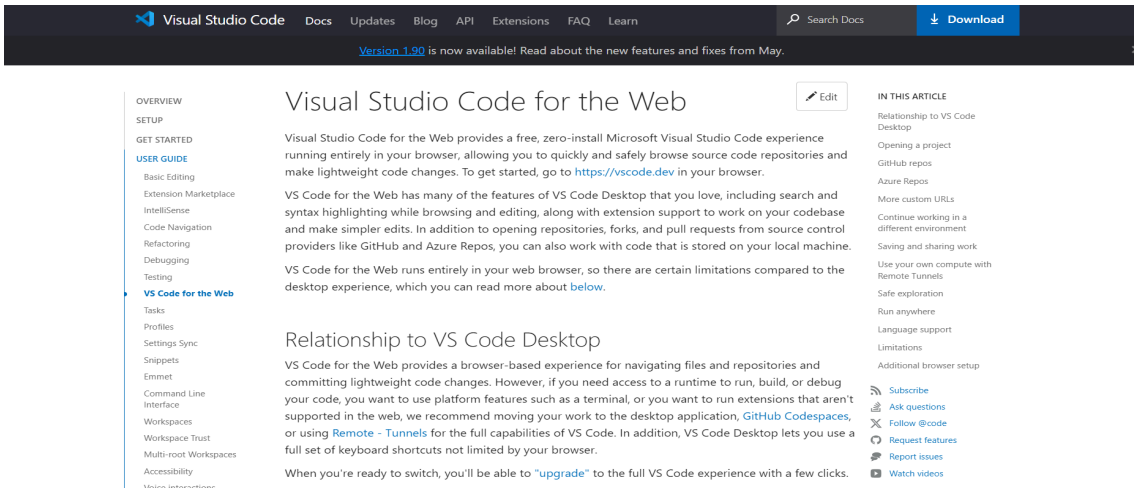
SUMO (Simulation of Urban Mobility) - is an open-source traffic simulation software used to simulate road traffic dynamics in urban areas. It allows for the modeling of various aspects of traffic systems, including vehicles, traffic signals, and infrastructure.



Pygame - is a Python library used for game development and multimedia applications. In the context of SUMO, Pygame may be used for visualization and interaction with the

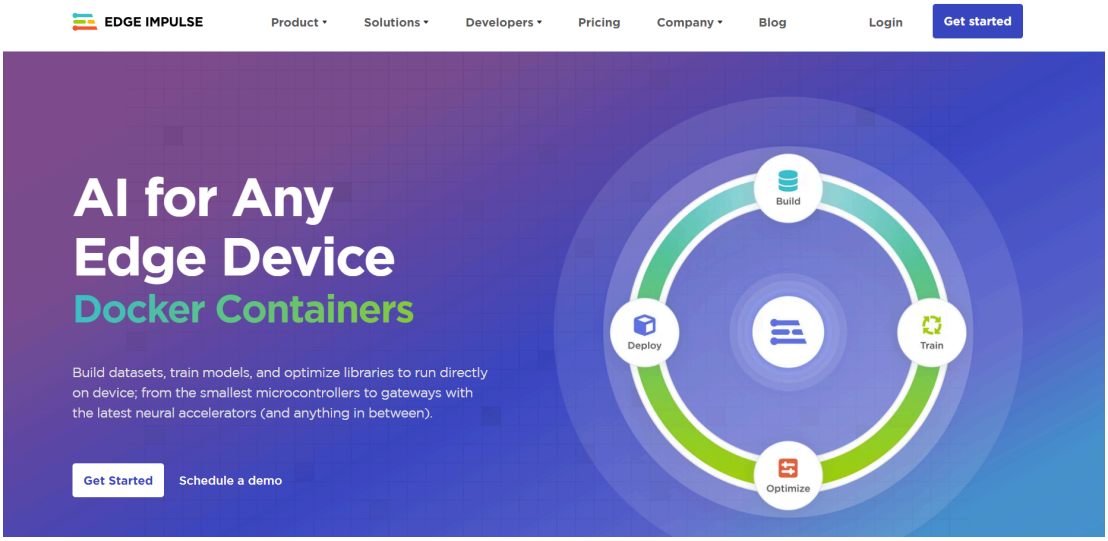
simulated traffic environment, providing a graphical interface for users to observe and control the simulation.

Visual Studio Code (VS Code)



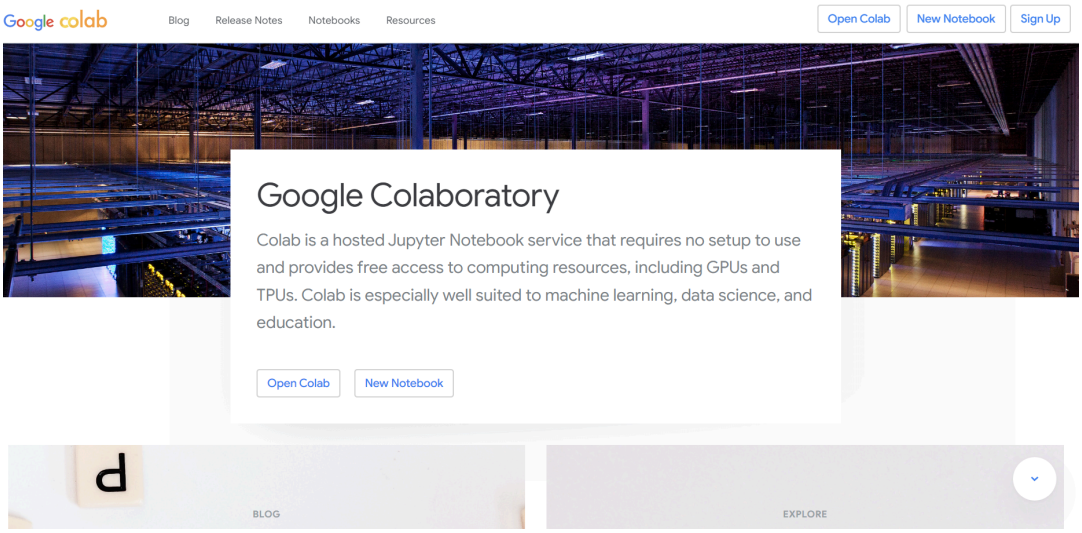
- Visual Studio Code is a free source-code editor developed by Microsoft. It supports various programming languages and features built-in debugging, syntax highlighting, code completion, and version control integration. VS Code is widely used by developers for writing and debugging code across different platforms.

Edge Impulse



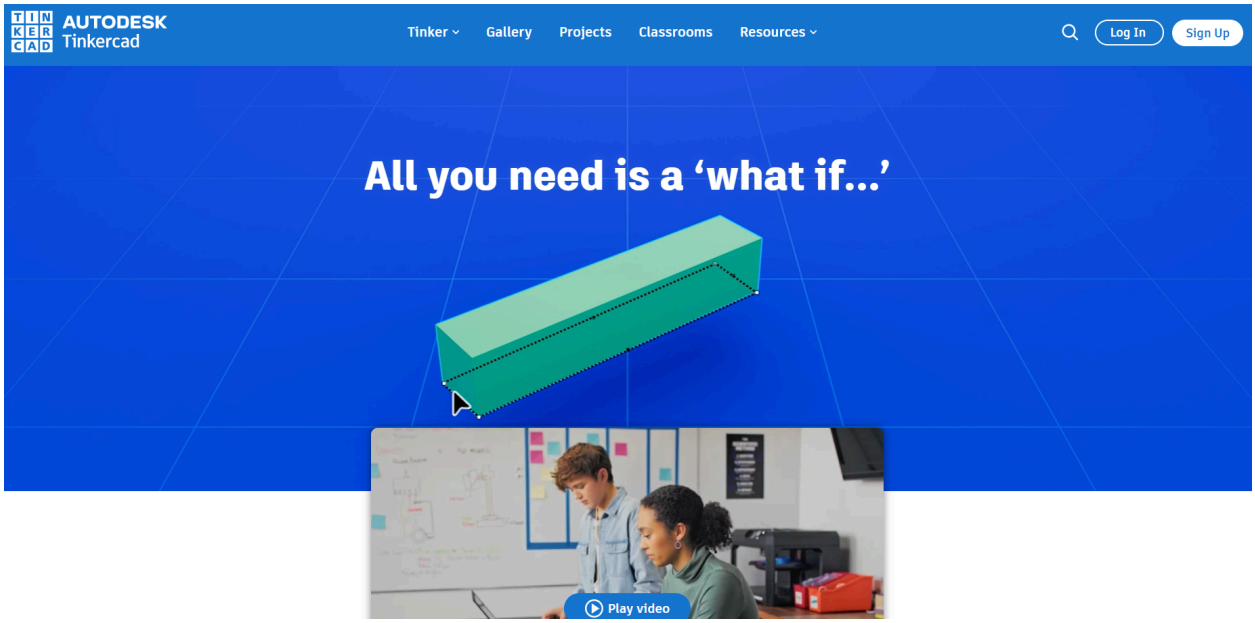
- is the leading edge AI platform for collecting data, training models, and deploying them to your edge computing devices. Its primary objective is to generate effective models for contexts with limited resources, such as microcontrollers, sensors, and mobile devices. The platform lets users gather and categorise sensor data from several sources by offering powerful capabilities for data management and collection. Edge Impulse's AutoML features make the process of creating models easier, especially for people who are not very experienced with machine learning.

Google Colab Notebook



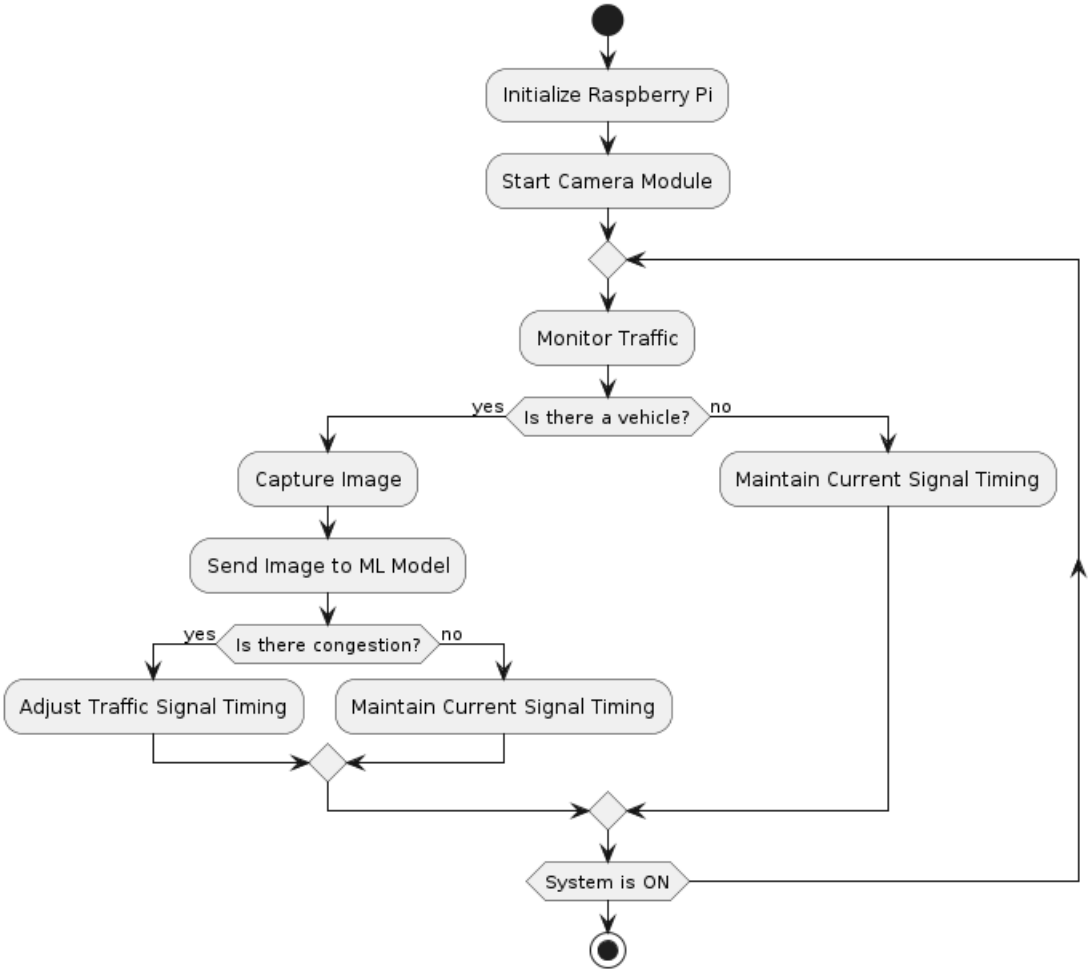
- Google Colab (short for Colaboratory) is a free, cloud-based Jupyter notebook environment provided by Google. It allows users to write and execute Python code through the browser, making it an ideal tool for machine learning, data analysis, and general Python programming.

TinkerCAD



- is a web-based computer-aided design (CAD) software developed by Autodesk. It is primarily used for creating 3D models, circuits, and simulations for educational and hobbyist purposes. TinkerCAD's intuitive interface and beginner-friendly tools make it suitable for users who are new to CAD and 3D modeling.

SYSTEM FLOWCHART

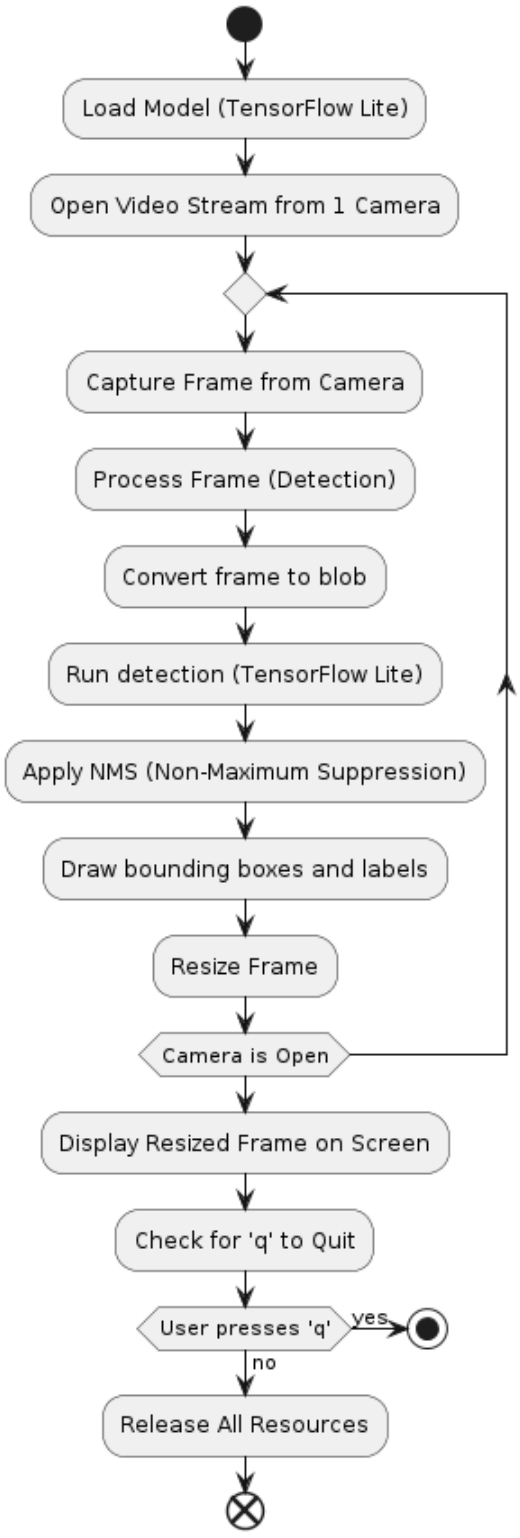


Flowchart fig 1.

This flowchart describes a traffic monitoring system that utilizes a Raspberry Pi and a camera module to manage traffic signal timing based on the level of congestion detected. Here's a step-by-step breakdown of the flowchart:

- **Initialize Raspberry Pi:** The system starts by powering up the Raspberry Pi, the main computing unit.
- **Start Camera Module:** Once the Raspberry Pi is operational, the camera module is activated to monitor traffic.

- **Monitor Traffic:** The camera continuously observes the traffic conditions.
- **Is there a vehicle?**: This decision point evaluates whether there is a vehicle in the camera's view. If no vehicle is detected, the system keeps the current traffic signal timing. If a vehicle is detected, it moves to the next step.
- **Capture Image:** The system captures an image of the traffic using the camera module.
- **Send Image to ML Model:** The captured image is sent to a machine learning model which analyses the image to assess the level of traffic congestion.
- **Is there congestion?:** The system decides whether there is congestion based on the machine learning model's analysis. If no congestion is detected, the traffic signal timing remains the same. If congestion is detected, the system adjusts the traffic signal timing.
- **Adjust Traffic Signal Timing:** The traffic signal timing is modified to accommodate the traffic flow, allowing more time for the congested route to clear.
- **Maintain Current Signal Timing:** If no changes are necessary, the traffic signal timing is maintained as is. The system operates in a continuous loop, monitoring traffic and adjusting signal timings as needed to manage traffic flow effectively.



Flowchart fig 2

Flowchart Operation of a Multi-Camera Surveillance System Using the TensorFlow Lite Model for Object Detection. Here's the step by step procedure;

1. **Load Model (TensorFlow Lite):**
   - ➢ The system starts by loading the pre-trained TensorFlow Lite model for object detection into memory.
2. **Open Video Stream from 1 Camera:**
   - ➢ The system initializes and opens a video stream from one of the connected cameras.
3. **Capture Frame from Camera:**
   - ➢ The system captures individual frames from the video stream in real-time.
4. **Process Frame (Detection):**
   - ➢ Each captured frame is processed for object detection.
5. **Convert Frame to Blob:**
   - ➢ The frame is converted into a blob format suitable for the TensorFlow Lite model.
6. **Run Detection (TensorFlow Lite)**:
   - ➢ The converted blob is fed into the TensorFlow Lite model to run the object detection algorithm.
7. **Apply NMS (Non-Maximum Suppression):**
   - ➢ Non-Maximum Suppression is applied to filter out duplicate or overlapping bounding boxes, ensuring each object is detected once.
8. **Draw Bounding Boxes and Labels:**
   - ➢ The system draws bounding boxes around detected objects and labels them with their respective classes.
9. **Resize Frame:**
   - ➢ The frame is resized if necessary to fit the display requirements.
10. **Camera is Open (Decision Point):**

- ➢ The system checks if the camera stream is still active. If yes, it continues to capture and process frames. If no, it moves to release resources.

11. **Display Resized Frame on Screen:**
    - ➢ The processed and resized frame with bounding boxes and labels is displayed on the screen.
12. **Check for 'q' to Quit:**
    - ➢ The system checks if the user has pressed the 'q' key to quit the application.
13. **User Presses 'q' (Decision Point):**
    - ➢ If the user presses 'q', the system proceeds to release all resources and close the application. If not, it continues capturing and processing frames.
14. **Release All Resources:**
    - ➢ The system releases all allocated resources, such as camera handles and memory, and shuts down the application

.

SOURCE CODE:

```
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util
from time import sleep
from gpiozero import LED

class VideoStream:
    def __init__(self, resolution=(640, 480), framerate=30):
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3, resolution[0])
```

```
        ret = self.stream.set(4, resolution[1])

        (self.grabbed, self.frame) = self.stream.read()

        self.stopped = False

    def start(self):
        Thread(target=self.update, args=()).start()
        return self

    def update(self):
        while True:
            if self.stopped:
              self.stream.release()
              return

            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
        return self.frame

    def stop(self):
        self.stopped = True


def isObjectInRect(rects, object_d):
    for i in range(len(rects)):
        rect = rects[i]
        l_x = rect[0]
        l_y = rect[1]
        u_x = rect[2]
        u_y = rect[3]
        if l_x <= object_d[0] <= u_x and l_y <= object_d[1] <= u_y:
            return True, i
    return False, None

parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in', required=True)
parser.add_argument('--graph', help='Name fo the .tflite file, if different than detect.tflite',
default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',
default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not
support the resolution enteredm, errors may occur.', default='1280x1000')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
```

```
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH =int(resW), int(resH)


pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
else:
    from tensorflow.lite.python.interpreter import Interpreter

CWD_PATH = os.getcwd()
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

if labels[0] == '???':
    del(labels[0])

else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean =127.5
input_std = 12.5

frame_rate_calc = 1
freq = cv2.getTickFrequency()
total_count = 0

videostream = VideoStream(resolution=(imW,imH), framerate=30).start()
time.sleep(1)

rect_areas0 = [[557,5,700,280]]
rect_areas1 = [[50,370,530,520]]
rect_areas2 = [[640,575,790,1000]]
rect_areas3 = [[800,330,1220,470]]

red0 = LED(17)
yellow0 = LED(27)
green0 = LED(22)

red1 = LED(26)
```

```
yellow1 = LED(19)
green1 = LED(13)

red2 = LED(16)
yellow2 = LED(20)
green2 = LED(21)

red3 = LED(14)
yellow3 = LED(15)
green3 = LED(18)

while True:
    current_count=0

    t1 = cv2.getTickCount()

    frame1 = videostream.read()

    frame = frame1.copy()
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    boxes = interpreter.get_tensor(output_details[0]['index'])[0]
    classes = interpreter.get_tensor(output_details[1]['index'])[0]
    scores = interpreter.get_tensor(output_details[2]['index'])[0]

    objects_in_rects0 = [0 for i in range(len(rect_areas0))]
    objects_in_rects1 = [0 for i in range(len(rect_areas1))]
    objects_in_rects2 = [0 for i in range(len(rect_areas2))]
    objects_in_rects3 = [0 for i in range(len(rect_areas3))]

    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
            center = (round(xmin / 2 + (xmax /2)), round(ymin / 2 + (ymax /2)))

            if int(classes[i]) < len(labels):
                object_name = labels[int(classes[i])]
            else:
```

```
            object_name = 'Unknown'

        label = '%s: %d%%' % (object_name, int(scores[i]*100))
        labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
        label_ymin = max(ymin, labelSize[1] + 10)
          cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10),
(255, 255, 255), cv2.FILLED)
        cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
        current_count+=1
        total_count=total_count+current_count

        if object_name == labels[int(classes[1])] :
            cv2.circle(frame, center, 5, (0, 0, 255), 5)
        else :
            cv2.circle(frame, center, 5, (34, 139, 34), 5)

        ret0, loc0 = isObjectInRect(rect_areas0, object_d=center)
        if ret0:
            if object_name == labels[int(classes[1])] :
                objects_in_rects0[loc0]+=1
            else :
                objects_in_rects0[loc0] += 1000

        ret1, loc1 = isObjectInRect(rect_areas1, object_d=center)
        if ret1:
            if object_name == labels[int(classes[1])] :
                objects_in_rects1[loc1]+=1
            else :
                objects_in_rects1[loc1] += 1000

        ret2, loc2 = isObjectInRect(rect_areas2, object_d=center)
        if ret2:
            if object_name == labels[int(classes[1])] :
                objects_in_rects2[loc2]+=1
            else :
                objects_in_rects2[loc2] += 1000

        ret3, loc3 = isObjectInRect(rect_areas3, object_d=center)
        if ret3:
            if object_name == labels[int(classes[1])] :
                objects_in_rects3[loc3]+=1
            else :
                objects_in_rects3[loc3] += 1000

        isim="+ ambulans"
        for i in range(len(objects_in_rects0)):

            print("Objects in rectangle0-", i, ": ", objects_in_rects0[i], "\n", end="")

            if objects_in_rects0[i] != 0:
                if objects_in_rects0[i] >= 1000:
                    yellow0.off()
```

```
            red0.on()
            green0.off()
        else:
            yellow0.off()
            red0.off()
            green0.on()
    else:
        yellow0.on()
        red0.off()
        green0.off()

for i in range(len(objects_in_rects1)):

    print("Objects in rectangle1-", i, ": ", objects_in_rects1[i], "\n", end="")

    if objects_in_rects1[i] != 0:
        if objects_in_rects1[i] >= 1000:
            yellow1.off()
            red1.on()
            green1.off()
        else:
            yellow1.off()
            red1.off()
            green1.on()
    else:
        yellow1.on()
        red1.off()
        green1.off()

for i in range(len(objects_in_rects2)):

    print("Objects in rectangle2-", i, ": ", objects_in_rects2[i], "\n", end="")

    if objects_in_rects2[i] != 0:
        if objects_in_rects2[i] >= 1000:
            yellow2.off()
            red2.on()
            green2.off()
        else:
            yellow2.off()
            red2.off()
            green2.on()
    else:
        yellow2.on()
        red2.off()
        green2.off()

for i in range(len(objects_in_rects3)):

    print("Objects in rectangle3-", i, ": ", objects_in_rects3[i], "\n", end="")

    if objects_in_rects3[i] != 0:
```

```
                if objects_in_rects3[i] >= 1000:
                    yellow3.off()
                    red3.on()
                    green3.off()
                else:
                    yellow3.off()
                    red3.off()
                    green3.on()
            else:
                yellow3.on()
                red3.off()
                green3.off()

                                                                cv2.putText(frame,'Total:                ' +
str(total_count),(10,25),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)

    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc = 1/time1

    cv2.imshow('Object detector', frame)

    if cv2.waitKey(1) == ord('q'):
        break

videostream.stop()
cv2.destroyAllWindows()
```