Intel® Unnati Industrial Training Program

PROJECT REPORT

Vehicle Movement Analysis and Insight Generation in a College Campus using Edge AI

Coimbatore Institute of Technology

Department of Electronics and Communication Engineering

Guided :

Dr. M. Poongothai (Mentor),

Professor,

Dept. of ECE.

CIT

Team Members:

Sanjay V	- 71762104041
Sabarivasan RT	- 71762208013
Karthik Venkatesh T	- 71762208015
Edwin Nova A	- 71762208023

Table of Contents

PROBLEM STATEMENT

Managing vehicle movement and parking on a college campus presents numerous challenges. To enhance campus security and management, an intelligent system is needed to analyze vehicle movement, monitor parking occupancy, and match vehicles to an approved database.Our project aims to develop an Edge AI-based solution capable of analyzing vehicle movement using data from cameras that capture vehicle photos and number plates. The solution will process image data in real-time and analyse various parameters such as:

Vehicle Movement Patterns: To analyze the timing of vehicle movements in and out of the campus, identify peak times and patterns.

Parking Occupancy: To monitor parking lot occupancy in real-time, determining which lots are frequently occupied and at what times.

Vehicle Matching: To match captured vehicle images and license plates to an approved database, identify unauthorized vehicles

.EXISTING PARKING SLOT SYSTEM:

Video Analytics Systems: These systems use cameras equipped with computer vision algorithms to monitor parking lots in real-time. They can detect the presence of vehicles, estimate occupancy levels, and even identify specific vehicles based on license plate recognition.

Sensor-based Systems: Utilizing sensors embedded in parking spaces or at entry/exit points, these systems can provide data on occupancy rates and monitor the flow of vehicles entering and leaving the campus. Sensors can detect when a vehicle enters or leaves a parking space, providing accurate real-time data.

RFID or NFC Technology: RFID (Radio Frequency Identification) or NFC (Near Field Communication) tags can be used on vehicles or parking permits to automatically track vehicles entering and exiting the campus. This allows for seamless monitoring without requiring visual recognition.

INNOVATION FROM OUR SOLUTION COMPARED TO THE EXISTING SYSTEM:

Edge AI Deployment: Unlike traditional systems that often rely on centralized servers or cloud computing for data processing, an Edge AI system performs computations locally on the edge devices (like cameras or edge servers) near where the data is generated (e.g., at campus entry points or parking lots). This reduces latency and dependency on continuous internet connectivity, making it suitable for real-time analysis.

Real-Time Processing: Edge AI systems excel in real-time processing of data. They can analyze vehicle movements and parking occupancy instantly, providing immediate insights without relying on data transmission to a remote server. This capability is crucial for time-sensitive applications such as managing parking congestion during peak hours.

Efficiency: Edge AI systems are designed to be efficient in resource utilization. They optimize computational tasks locally, minimizing the need for large-scale infrastructure and reducing operational costs compared to cloud-based solutions that require significant bandwidth and server capacities.

Privacy and Security: By processing data locally, Edge AI systems can enhance privacy and security. They reduce the risk of exposing sensitive data to external networks and mitigate concerns about data breaches or unauthorized access that may arise with cloud-based solutions.

Scalability: Edge AI systems are scalable in deployment across different locations within the campus without extensive infrastructure changes. They can easily integrate with existing camera networks and sensors, leveraging their local processing power to expand capabilities as needed.

Customization and Adaptability: Edge AI systems can be tailored to specific campus requirements and operational needs. They offer flexibility in adapting algorithms and models based on local data patterns, optimizing the system's performance over time.

OVERVIEW:

Our project utilizes Edge AI technology to streamline vehicle management and parking on college campuses, leveraging the YOLOv8 machine learning model at its core. First, through OCR algorithms, we accurately detect and register vehicles entering the campus by reading their license plates. This automation facilitates efficient access control and enhances campus security measures.

YOLOv8 enables real-time monitoring of parking slot availability and congestion analysis across campus. By continuously assessing parking occupancy and movement patterns, our system provides up-to-date information on available spaces, reducing congestion and optimizing parking utilization. Additionally, the model identifies unauthorized vehicles, swiftly alerting security personnel to potential breaches. Implemented as an Edge AI solution, our system processes data locally on campus servers or edge devices, ensuring rapid response times and enhancing data privacy. This approach minimizes reliance on external networks

PROPOSED SOLUTION:

Our innovative solution aims to revolutionize traffic management and congestion control within college campuses through advanced technology integration. Central to our system are five camera modules, each employing cutting-edge machine learning techniques, particularly utilizing the YOLOv8 model for enhanced accuracy and efficiency.

The first camera module (Cam 1) focuses on vehicle identification by detecting and reading number plates using optical character recognition (OCR) technology. This capability enables us to efficiently track vehicles entering and exiting the campus, ensuring comprehensive monitoring of traffic flow.

Cam 2 is strategically positioned to detect vehicles parked in unauthorized areas. This feature serves as a deterrent by alerting vehicle owners promptly, promoting adherence to designated parking zones and enhancing overall campus security and orderliness.

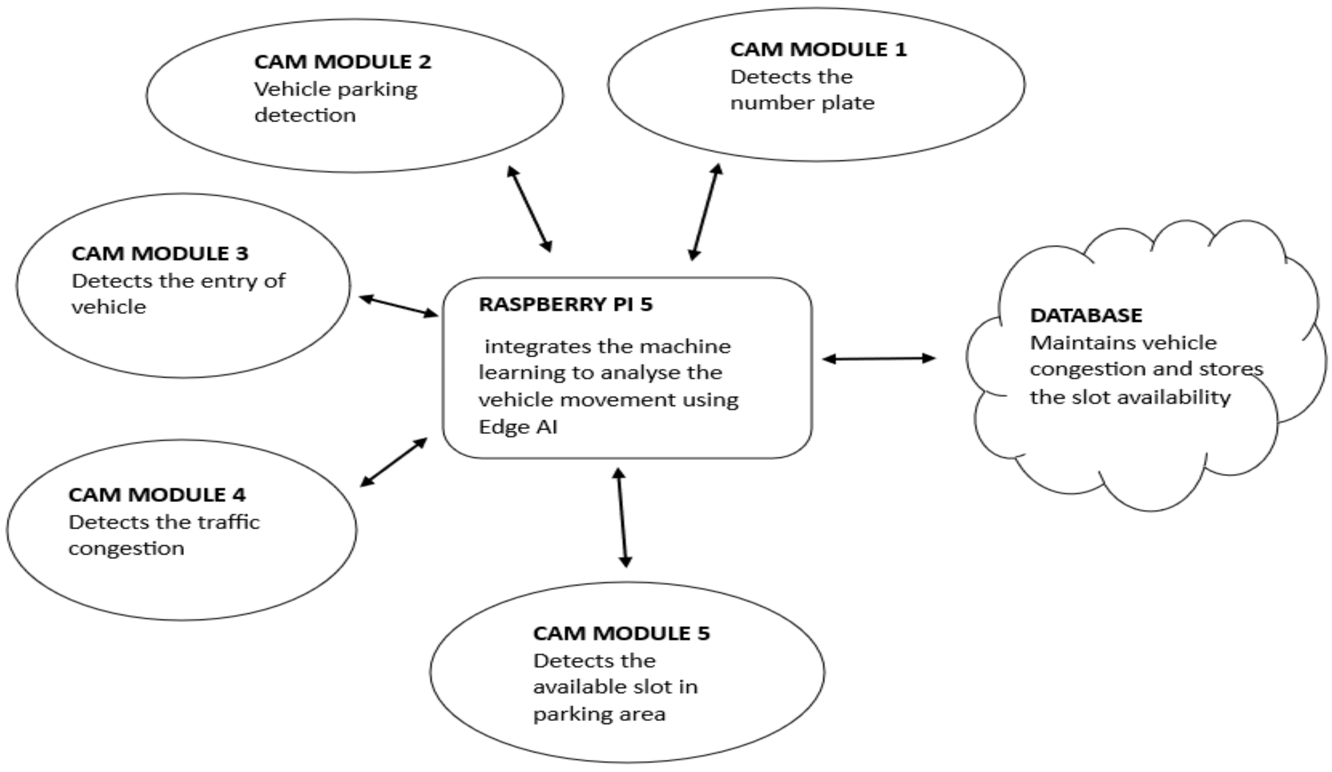
To manage and optimize parking availability, Cam 3 monitors vehicle entries at the main entrances. By accurately counting the frequency of vehicles entering, our system provides real-time insights into traffic patterns, facilitating proactive adjustments to traffic management strategies.

Cam 4 is dedicated to monitoring traffic congestion within the campus parking areas. Through sophisticated analysis, it identifies areas where congestion occurs, enabling prompt interventions such as redirecting vehicles to less congested zones or notifying authorities to manage the situation effectively.

The final camera module, Cam 5, focuses on analyzing available parking slots. By continuously assessing occupancy levels in parking areas, it provides up-to-date information on available spaces. This data is crucial for campus administrators and drivers alike, facilitating efficient parking allocation and reducing the time spent searching for parking.

All data collected by these camera modules are integrated using Edge AI which stored and managed in a centralized database. This database serves as a comprehensive repository of vehicle movement, parking availability, and traffic congestion details. It enables administrators to analyze historical trends, make informed decisions, and implement targeted improvements to enhance traffic flow and optimize parking resources effectively.

BLOCK DIAGRAM:


CAM MODULE 2
Vehicle parking detection
CAM MODULE 1
Detects the number plate
CAM MODULE 3
Detects the entry of vehicle
RASPBERRY PI 5
integrates the machine learning to analyse the vehicle movement using Edge AI
DATABASE
Maintains vehicle congestion and stores the slot availability
CAM MODULE 4
Detects the traffic congestion
CAM MODULE 5
Detects the available slot in parking area

1. COMPONENTS & FEATURES OFFERED

COMPONENTS:

The key components and functionalities of the proposed solution are outlined below:

- **CAMERA MODULES**:
  - **Purpose**: Camera modules with edge AI enable real-time vehicle detection, tracking, and behavior analysis for efficient parking management and traffic control. They enhance safety by monitoring traffic flow, detecting incidents, and optimizing parking space usage locally, minimizing reliance on cloud computing.
  - **Uses**: They enable real-time detection and recognition of vehicles, number plates (OCR), and parking slots. Each module serves a specific purpose:
    - Cam 1: Identifies and reads vehicle number plates for entry and exit tracking.
    - Cam 2: Detects vehicles parked in unauthorized areas, ensuring compliance with parking regulations.
    - Cam 3: Monitors vehicle entries at campus entrances to analyze traffic patterns and frequency.
    - Cam 4: Identifies and monitors traffic congestion within parking areas, guiding parking management decisions.
    - Cam 5: Analyzes parking slot availability, providing real-time updates to optimize parking allocation.

- **RASPBERRY PI 5**:
  - **Purpose**: Acts as an edge AI (Artificial Intelligence) device, processing data locally from the camera modules.
  - **Uses**:
    - Runs the YOLOv8(You Only Look Once version 8) model for real-time object detection and analysis.
    - Collects and preprocesses data before sending it to the cloud for storage and further analysis.
    - Facilitates edge computing to reduce latency and enhance system responsiveness.

- **AWS CLOUD (DATABASE MANAGEMENT)**:
  - **Purpose**: Provides scalable and reliable cloud infrastructure for storing and managing data from the traffic management system.
  - **Uses**:
    - Stores vehicle movement data, including entry/exit timestamps and parking slot availability.
    - Maintains databases for vehicle records, parking slot status, and historical traffic patterns.
    - Enables real-time analytics and reporting for administrators to make informed decisions on traffic flow and parking management.

FEATURES:

- **Real-time Vehicle Detection and Tracking**:
  - Utilizes YOLOv8 model for accurate and fast detection of vehicles entering and exiting campus premises.
  - Tracks vehicle movements in real-time, providing up-to-date data on traffic flow.
- **Number Plate Recognition (OCR)**:
  - Cam 1 identifies and reads vehicle number plates, enabling automated entry and exit logging.
  - Facilitates efficient management of vehicle access and monitoring.
- **Unauthorized Parking Detection**:
  - Cam 2 detects vehicles parked in unauthorized areas, triggering alerts to vehicle owners or security personnel.
  - Enhances campus security and ensures compliance with parking regulations.

- **Traffic Pattern Analysis**:
  - Cam 3 monitors vehicle entries at campus entrances to analyze traffic patterns and frequency.
  - Provides insights into peak traffic times and helps optimize traffic management strategies.
- **Parking Slot Availability Monitoring**:

  Cam 4 analyzes parking areas to detect congestion and occupancy levels.

  - Guides drivers to available parking slots and optimizes parking space utilization.
- **Real-time Parking Slot Updates**:
  - Cam 5 continuously assesses and updates the availability of parking slots.
  - Provides real-time information to drivers and campus administrators, reducing search time for parking.
- **Edge AI Processing**:
  - Raspberry Pi 5 serves as an edge AI device, processing data locally from camera modules.
  - Reduces latency in data processing and enhances system responsiveness.
- **Cloud-based Data Storage and Management**:
  - AWS Cloud infrastructure stores and manages data from camera modules securely.
  - Supports scalable storage for vehicle records, parking slot status, and traffic analytics.
- **Historical Data Analysis**:
  - • Maintains databases for historical traffic patterns, aiding in long-term traffic management planning.
  - Enables administrators to analyze trends and make data-driven decisions for campus infrastructure improvements.

- **Integration and Scalability**:
  - Integrates with other campus systems for enhanced functionalities such as notifications/alerts and predictive analytics.
  - Scalable architecture allows for future expansion and integration of additional features or improvements.

- **Enhanced Campus Security and Efficiency**:
  - Improves overall campus security by monitoring and managing vehicle movements effectively.
  - Enhances operational efficiency by optimizing traffic flow and parking space allocation.

PARKING SLOT AVAILABILITY DETECTION CODE

```
import cv2

import pandas as pd

from ultralytics import YOLO

import cvzone

import urllib.request

import numpy as np

url = 'http://192.168.79.97/cam-hi.jpg'

cv2.namedWindow("Live Cam Testing", cv2.WINDOW_NORMAL)

cv2.resizeWindow('Live Cam Testing', 1280, 720)

cv2.moveWindow('Live Cam Testing', 0, 0)

picam2 = cv2.VideoCapture(url)

if not picam2.isOpened():

    print("Failed to open the IP camera stream")

    exit()

    model = YOLO('/home/raspberry/Intel_Project/ML_yolov8/Parking_slot_availability_detection/Main
Models/Parking_slot_detection_yolov8s/weights/best.pt')

my_file = open("/home/raspberry/Intel_Project/ML_yolov8/Parking_slot_availability_detection/Main
Models/Parking_slot_detection_yolov8s/weights/coco4.txt", "r")

data = my_file.read()

class_list = data.split("\n")

count = 0

while True:

    img_resp = urllib.request.urlopen(url)
    imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint
    im = cv2.imdecode(imgnp, cv2.IMREAD_COLOR)
    results = model.predict(im)
    a = results[0].boxes.data
    px = pd.DataFrame(a).astype("float")
    for index, row in px.iterrows():
        x1 = int(row[0])
        y1 = int(row[1])
        x2 = int(row[2])
        y2 = int(row[3])
        d = int(row[5])
        if d < len(class_list):
            c = class_list[d]
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
        else:
            c = 'Unknown'
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
     cv2.imshow("Live Cam Testing", im)
    if cv2.waitKey(1) == ord('q'):
        break
cv2.destroyAllWindows()
```

VEHICLE DETECTION CODE

```
import cv2

import pandas as pd

from ultralytics import YOLO

import cvzone

import urllib.request

import numpy as np

url = 'http://192.168.79.97/cam-hi.jpg'

cv2.namedWindow("Live Cam Testing", cv2.WINDOW_NORMAL)

cv2.resizeWindow('Live Cam Testing', 1280, 720)

cv2.moveWindow('Live Cam Testing', 0, 0)

picam2 = cv2.VideoCapture(url)

if not picam2.isOpened():

   print("Failed to open the IP camera stream")

   exit()

model = YOLO('/home/raspberry/Intel_Project/ML_yolov8/Vehicle_Detection/weight/vehicle_detecton_best.pt')

my_file = open("/home/raspberry/Intel_Project/ML_yolov8/Vehicle_Detection/weight/coco1.txt", "r")

data = my_file.read()

class_list = data.split("\n")

count = 0

while True:

   img_resp = urllib.request.urlopen(url)
```

```
    imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
    im = cv2.imdecode(imgnp, cv2.IMREAD_COLOR)
    results = model.predict(im)
    a = results[0].boxes.data
    px = pd.DataFrame(a).astype("float")
    for index, row in px.iterrows():
        x1 = int(row[0])
        y1 = int(row[1])
        x2 = int(row[2])
        y2 = int(row[3])
        d = int(row[5])
        if d < len(class_list):
            c = class_list[d]
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
        else:
            c = 'Unknown'
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
    cv2.imshow("Live Cam Testing", im)
    if cv2.waitKey(1) == ord('q'):
        break
cv2.destroyAllWindows()
```

VEHICLE TRAFFIC ESTIMATION CODE

```
import cv2

import numpy as np

from ultralytics import YOLO

# Load the best fine-tuned YOLOv8 model

best_model                                                                                         =
    YOLO('/home/raspberry/Intel_Project/ML_yolov8/Vehicle_Traffic_Estimation/Weigt
    h/models/best.pt')

# Define the threshold for considering traffic as heavy

heavy_traffic_threshold = 4

# Define the vertices for the quadrilaterals

vertices1 = np.array([(465, 350), (609, 350), (510, 630), (2, 630)], dtype=np.int32)

vertices2 = np.array([(678, 350), (815, 350), (1203, 630), (743, 630)], dtype=np.int32)

# Define the vertical range for the slice and lane threshold

x1, x2 = 325, 635

lane_threshold = 609

# Define the positions for the text annotations on the image

text_position_left_lane = (10, 50)

text_position_right_lane = (820, 50)

intensity_position_left_lane = (10, 100)

intensity_position_right_lane = (820, 100)

# Define font, scale, and colors for the annotations
```

```
font = cv2.FONT_HERSHEY_SIMPLEX

font_scale = 1

font_color = (255, 255, 255)    # White color for text

background_color = (0, 0, 255)  # Red background for text

# Open the video

cap                                                                                    =
    cv2.VideoCapture('/home/raspberry/Intel_Project/ML_yolov8/Vehicle_Traffic_Estim
    ation/Weigth/sample_video_01.mp4')

# Define the codec and create VideoWriter object

fourcc = cv2.VideoWriter_fourcc(*'XVID')

out                                                                                    =
    cv2.VideoWriter('/home/raspberry/Intel_Project/ML_yolov8/Vehicle_Traffic_Estimat
    ion/Weigth//processed_sample_video_1.mp4',    fourcc,    20.0,    (int(cap.get(3)),
    int(cap.get(4))))

# Read until video is completed

while cap.isOpened():

    # Capture frame-by-frame

    ret, frame = cap.read()

    if ret:

        # Create a copy of the original frame to modify

        detection_frame = frame.copy()

    # Black out the regions outside the specified vertical range

        detection_frame[:x1, :] = 0  # Black out from top to x1
        detection_frame[x2:, :] = 0  # Black out from x2 to the bottom of the frame

        # Perform inference on the modified frame

        results = best_model.predict(detection_frame, imgsz=640, conf=0.4)

        processed_frame = results[0].plot(line_width=1)

        # Restore the original top and bottom parts of the frame

        processed_frame[:x1, :] = frame[:x1, :].copy()

        processed_frame[x2:, :] = frame[x2:, :].copy()

        # Draw the quadrilaterals on the processed frame

        cv2.polylines(processed_frame, [vertices1], isClosed=True, color=(0, 255, 0), thickness=2)

        cv2.polylines(processed_frame, [vertices2], isClosed=True, color=(255, 0, 0), thickness=2)

        # Retrieve the bounding boxes from the results

        bounding_boxes = results[0].boxes

# Initialize counters for vehicles in each lane

        vehicles_in_left_lane = 0

        vehicles_in_right_lane = 0

     # Loop through each bounding box to count vehicles in each lane

        for box in bounding_boxes.xyxy:

            # Check if the vehicle is in the left lane based on the x-coordinate of the bounding box

if box[0] < lane_threshold:
```

```
            vehicles_in_left_lane += 1
        else:
            vehicles_in_right_lane += 1
            # Determine the traffic intensity for the left lane
    traffic_intensity_left = "Heavy" if vehicles_in_left_lane > heavy_traffic_threshold else "Smooth"
    # Determine the traffic intensity for the right lane
    traffic_intensity_right = "Heavy" if vehicles_in_right_lane > heavy_traffic_threshold else "Smooth"
# Add a background rectangle for the left lane vehicle count
    cv2.rectangle(processed_frame, (text_position_left_lane[0]-10, text_position_left_lane[1] - 25),
                  (text_position_left_lane[0] + 460, text_position_left_lane[1] + 10), background_color, -1)
# Add the vehicle count text on top of the rectangle for the left lane
    cv2.putText(processed_frame, f'Vehicles in Left Lane: {vehicles_in_left_lane}', text_position_left_lane,
                font, font_scale, font_color, 2, cv2.LINE_AA)
# Add a background rectangle for the left lane traffic intensity
    cv2.rectangle(processed_frame, (intensity_position_left_lane[0]-10, intensity_position_left_lane[1] - 25),
   (intensity_position_left_lane[0] + 460, intensity_position_left_lane[1] + 10), background_color, -1)
```

```
 # Add the traffic intensity text on top of the rectangle for the left lane

    cv2.putText(processed_frame, f'Traffic Intensity: {traffic_intensity_left}',
  intensity_position_left_lane,

               font, font_scale, font_color, 2, cv2.LINE_AA)

# Add a background rectangle for the right lane vehicle count

    cv2.rectangle(processed_frame, (text_position_right_lane[0]-10,
  text_position_right_lane[1] - 25),

                (text_position_right_lane[0] + 460, text_position_right_lane[1] + 10),
  background_color, -1)

# Add the vehicle count text on top of the rectangle for the right lane

    cv2.putText(processed_frame, f'Vehicles in Right Lane: {vehicles_in_right_lane}',
  text_position_right_lane,

               font, font_scale, font_color, 2, cv2.LINE_AA)

# Add a background rectangle for the right lane traffic intensity

    cv2.rectangle(processed_frame, (intensity_position_right_lane[0]-10,
  intensity_position_right_lane[1] - 25),

                (intensity_position_right_lane[0] + 460, intensity_position_right_lane[1] +
  10), background_color, -1)

# Add the traffic intensity text on top of the rectangle for the right lane

    cv2.putText(processed_frame, f'Traffic Intensity: {traffic_intensity_right}',
  intensity_position_right_lane,

font, font_scale, font_color, 2, cv2.LINE_AA)

# Display the processed frame
```

```
        cv2.imshow('Real-time Traffic Analysis', processed_frame)
# Press Q on keyboard to exit the loop
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        Break
# Release the video capture and video write objects
cap.release()
out.release()
# Close all the frames
cv2.destroyAllWindows()
```

VEHICLE NUMBER PLATE DETECTION CODE

```
import cv2

import pandas as pd

from ultralytics import YOLO

import cvzone

import urllib.request

import numpy as np

url = 'http://192.168.79.97/cam-hi.jpg'

cv2.namedWindow("Live Cam Testing", cv2.WINDOW_NORMAL)

cv2.resizeWindow('Live Cam Testing', 1280, 720)

cv2.moveWindow('Live Cam Testing', 0, 0)

picam2 = cv2.VideoCapture(url)

if not picam2.isOpened():

    print("Failed to open the IP camera stream")

    exit()

model = YOLO('/home/raspberry/Intel_Project/ML_yolov8/Vehicle_number_plate_detection/weight/license_plate_detector_best_zoom.pt')

my_file = open("/home/raspberry/Intel_Project/ML_yolov8/Vehicle_number_plate_detection/weight/coco2.txt", "r")

data = my_file.read()

class_list = data.split("\n")
```

```
count = 0
while True:
    img_resp = urllib.request.urlopen(url)
    imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
    im = cv2.imdecode(imgnp, cv2.IMREAD_COLOR)
    results = model.predict(im)
    a = results[0].boxes.data
    px = pd.DataFrame(a).astype("float")
    for index, row in px.iterrows():
        x1 = int(row[0])
        y1 = int(row[1])
        x2 = int(row[2])
        y2 = int(row[3])
        d = int(row[5])
        if d < len(class_list):
            c = class_list[d]
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
        else:
            c = 'Unknown'
            cv2.rectangle(im, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cvzone.putTextRect(im, f'{c}', (x1, y1), 1, 1)
    cv2.imshow("Live Cam Testing", im)
    if cv2.waitKey(1) == ord('q'):
        break
cv2.destroyAllWindows()
```

VEHICLE ENTERING COUNT CODE

```
# import the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os
import glob
import math
files = glob.glob('output/*.png')
for f in files:
    os.remove(f)
from sort import *
tracker = Sort()
memory = {}
line1 = [(400,638), (1250, 788)]
line2 = [(120,838), (1120, 1080)]
counter1 = 0
counter2 = 0
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
```

```
ap.add_argument("-i", "--input", required=True,
    help="path to input video")
ap.add_argument("-o", "--output", required=True,
    help="path to output video")
ap.add_argument("-y", "--yolo", required=True,
    help="base path to YOLO directory")
ap.add_argument("-c", "--confidence", type=float, default=0.35,
    help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.25,
    help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())
def adjust_gamma(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")
 # apply gamma correction using the lookup table
    return cv2.LUT(image, table)
# Return true if line segments AB and CD intersect
def intersect(A,B,C,D):
    return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)
def ccw(A,B,C):
    return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])
# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"] , "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")
# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(200, 3),
    dtype="uint8")
# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])
# load our YOLO object detector trained on COCO dataset (80 classes)
# and determine only the output layer names that we need from YOLO
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
# initialize the video stream, pointer to output video file, and
# frame dimensions
```

```
vs = cv2.VideoCapture(args["input"])
writer = None
(W, H) = (None, None)
frameIndex = 0
# try to determine the total number of frames in the video file
try:
    prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
        else cv2.CAP_PROP_FRAME_COUNT
    total = int(vs.get(prop))
    print("[INFO] {} total frames in video".format(total))
# an error occurred while trying to determine the total
# number of frames in the video file
except:
    print("[INFO] could not determine # of frames in video")
    print("[INFO] no approx. completion time can be provided")
    total = -1
# loop over frames from the video file stream
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()
    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break
# if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]
frame = adjust_gamma(frame, gamma=1.5)
    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (256, 256),
        swapRB=True, crop=False)
    net.setInput(blob)
    start = time.time()
    layerOutputs = net.forward(ln)
    end = time.time()
# initialize our lists of detected bounding boxes, confidences,
    # and class IDs, respectively
    boxes = []
    center = []
    confidences = []
    classIDs = []
# loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
       # extract the class ID and confidence (i.e., probability)
        # of the current object detection
        scores = detection[5:]
       classID = np.argmax(scores)
       confidence = scores[classID]
        # filter out weak predictions by ensuring the detected
        # probability is greater than the minimum probability
            if confidence > args["confidence"]:
                # scale the bounding box coordinates back relative to
                # the size of the image, keeping in mind that YOLO
                # actually returns the center (x, y)-coordinates of
                # the bounding box followed by the boxes' width and
                # height
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
                # use the center (x, y)-coordinates to derive the top
                # and and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))
                # update our list of bounding box coordinates,
                # confidences, and class IDs
                center.append(int(centerY))
                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                classIDs.append(classID)
                 # apply non-maxima suppression to suppress weak, overlapping
                # bounding boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"], args["threshold"])
    #print("idxs", idxs)
    #print("boxes", boxes[i][0])
    #print("boxes", boxes[i][1])
    dets = []
    if len(idxs) > 0:
        # loop over the indexes we are keeping
        for i in idxs.flatten():
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
            dets.append([x, y, x+w, y+h, confidences[i]])
            #print(confidences[i])
            #print(center[i])
    np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
    dets = np.asarray(dets)
    tracks = tracker.update(dets)
    boxes = []
    indexIDs = []
    c = []
     previous = memory.copy()
    #print("centerx",centerX)
    #  print("centery",centerY)
    memory = {}
for track in tracks:
        boxes.append([track[0], track[1], track[2], track[3]])
        indexIDs.append(int(track[4]))
        memory[indexIDs[-1]] = boxes[-1]

    if len(boxes) > 0:
```

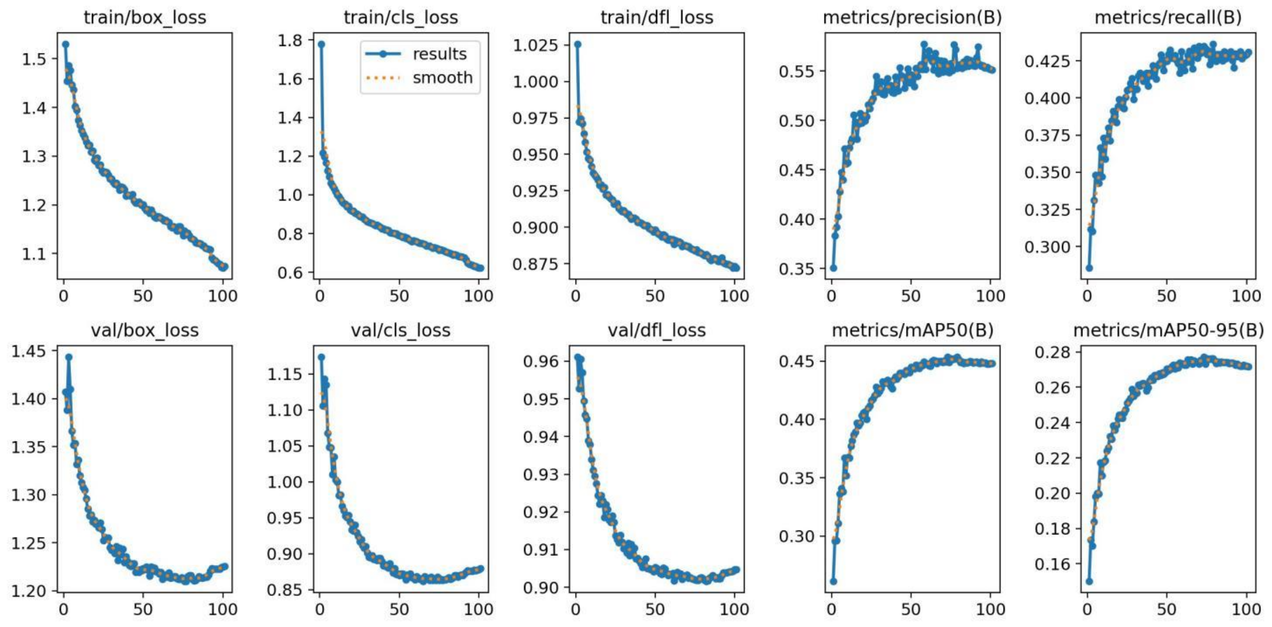
```
i = int(0)
for box in boxes:
    # extract the bounding box coordinates
    (x, y) = (int(box[0]), int(box[1]))
    (w, h) = (int(box[2]), int(box[3]))
    # draw a bounding box rectangle and label on the image
    # color = [int(c) for c in COLORS[classIDs[i]]]
    # cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
    color = [int(c) for c in COLORS[indexIDs[i] % len(COLORS)]]
    cv2.rectangle(frame, (x, y), (w, h), color, 2)
        if indexIDs[i] in previous:
        previous_box = previous[indexIDs[i]]
        (x2, y2) = (int(previous_box[0]), int(previous_box[1]))
        (w2, h2) = (int(previous_box[2]), int(previous_box[3]))
        p0 = (int(x + (w-x)/2), int(y + (h-y)/2))
        p1 = (int(x2 + (w2-x2)/2), int(y2 + (h2-y2)/2))
        cv2.line(frame, p0, p1, color, 3)
        #Speed Calculation
        y_pix_dist = int(y + (h-y)/2) - int(y2 + (h2-y2)/2)
        text_y = "{} y".format(y_pix_dist)
        x_pix_dist = int(x + (w-x)/2) - int(x2 + (w2-x2)/2)
```

```
                text_x = "{} x".format(x_pix_dist)
                #cv2.putText(frame, text_y, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 4)
                #cv2.putText(frame, text_x, (x, y + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 4)
                final_pix_dist = math.sqrt((y_pix_dist*y_pix_dist)+(x_pix_dist*x_pix_dist))
                speed = np.round(1.5 * y_pix_dist,2)
                text_speed = "{} km/h".format(speed)
                cv2.putText(frame, text_speed, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
               if intersect(p0, p1, line1[0], line1[1]):
                   counter1 += 1
#               if intersect(p0, p1, line2[0], line2[1]):
#                   counter2 += 1
# text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
            #text = "{}".format(indexIDs[i])
            #cv2.putText(frame, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color,
            i += 1

    # draw line
    cv2.line(frame, line1[0], line1[1], (0, 255, 255), 4)
#   cv2.line(frame, line2[0], line2[1], (255, 0, 255), 2)
```

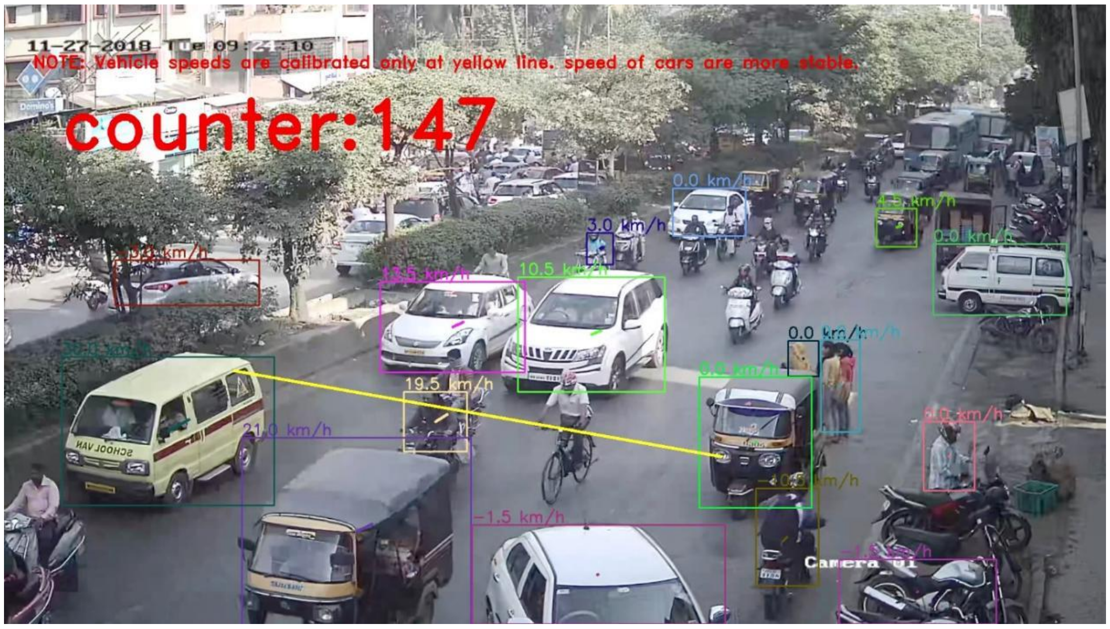
```
note_text = "NOTE: Vehicle speeds are calibrated only at yellow line. speed of cars are more stable."
    cv2.putText(frame, note_text, (50,110), cv2.FONT_HERSHEY_DUPLEX, 1.0, (0, 0, 255), 2)
    # draw counter
    counter_text = "counter:{}".format(counter1)
    cv2.putText(frame, counter_text, (100,250), cv2.FONT_HERSHEY_DUPLEX, 4.0, (0, 0, 255), 7)
#   cv2.putText(frame, "ctr2",str(counter2), (100,400), cv2.FONT_HERSHEY_DUPLEX, 5.0, (255, 0, 255), 10)
    # counter +=
    # saves image file
    #+cv2.imwrite("output/frame-{}.png".format(frameIndex), frame)
# check if the video writer is None
    if writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 15,
            (frame.shape[1], frame.shape[0]), True)

        # some information on processing single frame
        if total > 0:
            elap = (end - start)
            print("[INFO] single frame took {:.4f} seconds".format(elap))
            print("[INFO] estimated total time to finish: {:.4f}".format(
                elap * total))
# write the output frame to disk
    writer.write(frame)
     # increase frame index
    frameIndex += 1
#if frameIndex >= 4000: # limits the execution to the first 4000 frames
    #    print("[INFO] cleaning up...")
    #    writer.release()
    #    vs.release()
    #    exit()
    # release the file pointers
print("[INFO] cleaning up...")
writer.release()
vs.release()
```

RESULT

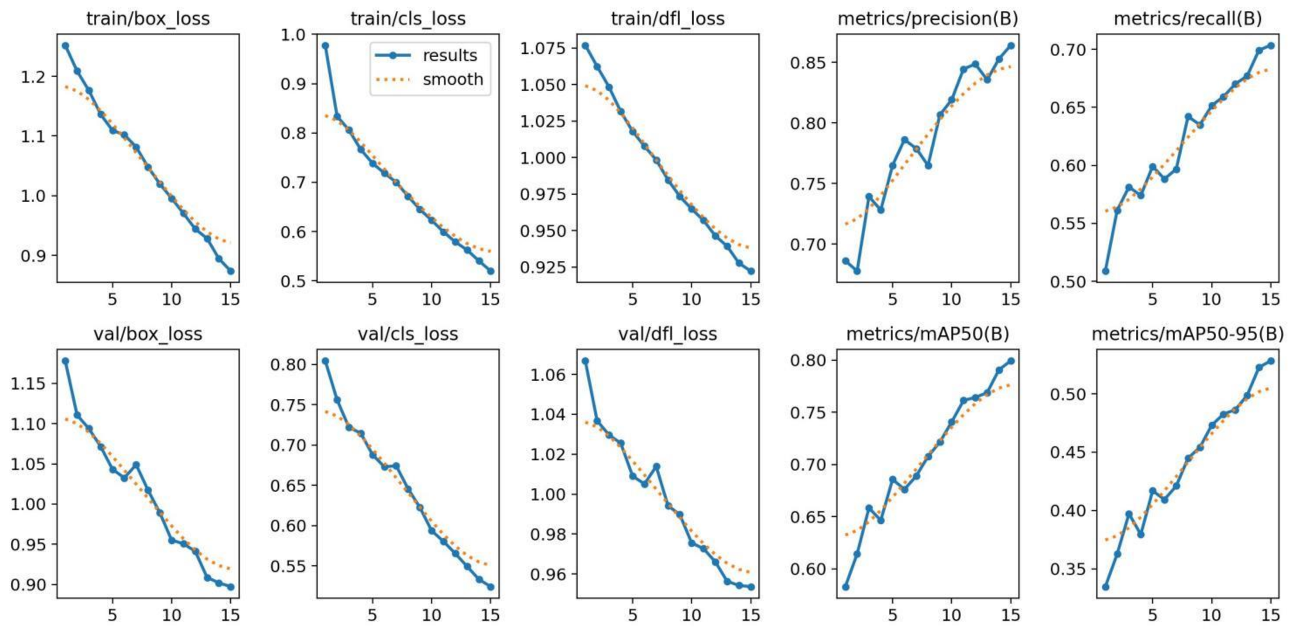
PARKING SLOT AVAILABILITY



VEHICLE COUNTING


11-27-2018 Tue 09:24:10
NOTE: Vehicle speeds are calibrated only at yellow line. speed of cars are more stable.
counter:147
Camera_01

VEHICLE DETECTION:



2. CHALLENGES & FUTURE ENHANCEMENTS

KEY CHALLENGES:

• **Limited Processing Power**: Edge devices typically have limited computational resources compared to cloud servers. This limits the complexity of AI models that can be deployed and the speed at which they can process data.

• **Real-time Processing**: Analyzing vehicle movement in real-time requires quick decision-making and low latency, which can be challenging for edge devices, especially when dealing with multiple concurrent inputs.

• **Data Storage and Management**: Edge devices may have limited storage capacity. Managing and storing data for analysis can be challenging, especially if continuous recording or extensive historical data is needed.

• **Power Constraints**: Edge devices are often battery-powered or have limited access to power. AI algorithms must be optimized to minimize power consumption while still performing necessary computations.

• **Environmental Variability**: Car parks can have diverse lighting conditions, weather, and varying vehicle types, which can affect the accuracy of edge AI algorithms trained in controlled environments.

• **Security Concerns**: Edge devices may be more vulnerable to physical tampering or data breaches compared to centralized cloud solutions. Ensuring data security and integrity is crucial. .

OVERCOMING THEM:

I• **Optimized AI Algorithms**: Develop lightweight and efficient AI models that can perform necessary computations within the constraints of edge devices. Techniques such as model compression, quantization, and pruning can reduce model size and computational complexity without sacrificing accuracy.

• **Edge Device Selection**: Choose edge devices with sufficient processing power and memory to handle real-time AI inference tasks. Consider devices equipped with GPUs or dedicated AI accelerators for enhanced performance.

• **Real-time Processing Optimization**: Implement optimizations such as batch processing, pipelining, and parallelization to minimize latency and improve real-time processing capabilities of edge AI algorithms.

- **Data Management Strategies**: Use efficient data storage and management techniques tailored for edge environments. This may include data compression, selective data sampling, and edge caching to optimize storage usage and retrieval.

- **Power Management**: Employ power-efficient AI algorithms and hardware configurations to extend battery life or reduce power consumption. Implement techniques like duty cycling and dynamic voltage frequency scaling (DVFS) to optimize power usage based on workload demands.

FUTURE DEVELOPMENTS:

- **Advancements in Edge Hardware**: Continued improvements in edge computing hardware, such as more powerful processors, AI-specific chips (like TPUs and NPUs), and increased memory capacity, will enable more complex AI models to be deployed at the edge.

- **Edge-Cloud Integration**: Enhanced integration between edge devices and cloud platforms will facilitate seamless data synchronization, model updates, and centralized management. This hybrid approach will leverage the strengths of both edge computing (low latency, local processing) and cloud computing (scalability, extensive storage).

- **Edge AI Model Optimization**: Ongoing research into model compression, quantization, and efficient algorithms will result in AI models that are even more lightweight, energy-efficient, and capable of real-time inference on edge devices.

- **AI-driven Automation**: Integration of AI with IoT sensors, cameras, and actuators in smart parking systems will enable automated monitoring, optimization of parking space utilization, and real-time adjustment of parking fees or availability notifications

OPERATING ENVIRONMENT & GUIDELINESS

HARDWARES:

RASPBERRY PI 5

- **CPU**: The Raspberry Pi 4 Model B is powered by the Broadcom BCM2711, a System-on-Chip (SoC) with a quad-core Cortex-A72 processor architecture. Each core can operate at up to 1.5GHz, providing substantial computing power for various tasks. This architecture is based on ARMv8-A, offering 64-bit support, which allows the Raspberry Pi 4 to run modern operating systems and applications efficiently.

- **RAM**: The Raspberry Pi 4 Model B comes in three variants with different RAM configurations: 2GB, 4GB, and 8GB of LPDDR4-3200 SDRAM. This RAM type and speed provide improved performance compared to earlier Raspberry Pi models, making multitasking smoother and enabling the use of more memory-intensive applications.

- **Storage**: Unlike traditional computers that have built-in storage drives like SSDs or hard drives, the Raspberry Pi 4 Model B uses a microSD card for storage. This allows users to choose the size and speed of the storage medium based on their needs and budget. MicroSD cards are widely available and can be easily swapped for different operating systems or projects.

- **GPU**: The Raspberry Pi 4 Model B features the Broadcom VideoCore VI GPU, which supports OpenGL ES 3.x and provides hardware-accelerated video decoding. This GPU is capable of handling graphics-intensive tasks such as video playback, gaming, and graphical user interfaces smoothly, enhancing the overall multimedia experience on the Raspberry Pi.

- **Operating System**: The Raspberry Pi 4 Model B supports a variety of operating systems, with Raspberry Pi OS (formerly Raspbian) being the official and most widely used distribution. It is based on Debian and optimized for the Raspberry Pi hardware, offering a user-friendly environment for both beginners and advanced users. Additionally, other operating systems like Ubuntu, Arch Linux ARM, and specialized distributions for media centers or IoT projects are available, thanks to the ARM architecture compatibility.

ESP CAM MODULE:

- **Microcontroller**: The ESP-CAM module is built around the ESP32 microcontroller chip, which features dual-core processors operating at up to 240 MHz. This microcontroller is based on the Xtensa LX6 architecture and supports Wi-Fi and Bluetooth connectivity, making it suitable for IoT applications requiring wireless communication capabilities.

- **Camera Interface**: The key feature of the ESP-CAM module is its integrated camera interface. It typically supports OV2640 or OV7670 cameras, allowing the module to capture images and video streams directly. The OV2640 is a popular choice, offering resolutions up to 2 megapixels (1600x1200 pixels).

- **Wireless Connectivity**: Similar to other ESP32-based modules, the ESP-CAM includes built-in Wi-Fi (802.11 b/g/n) and Bluetooth (BLE) capabilities. This enables seamless integration into IoT networks and allows remote control and data transfer over the internet or local network.

- **Memory and Storage**: The module typically includes onboard flash memory for storing program firmware and configuration data. It also supports external microSD cards for additional storage, making it capable of storing captured images and videos locally.

- **Power Management**: The ESP-CAM module features efficient power management capabilities, allowing it to operate on low power when necessary. This is crucial for battery-powered applications, extending the module's usability in remote or mobile setups

SOFTWARES:

VS CODE :

Visual Studio Code (VS Code) is a popular, free, and open-source code editor developed by Microsoft. It has gained widespread adoption among developers for its versatility, extensive customization options, and rich ecosystem of extensions. Here are some key details about VS Code:

Cross-Platform: VS Code runs on Windows, macOS, and Linux, providing a consistent development experience across different operating systems.

Feature-Rich Editor: It includes a powerful editor with support for syntax highlighting, code snippets, IntelliSense (code completion and suggestions), and customizable key bindings.

Extensions: VS Code supports a wide range of extensions that enhance its functionality. These extensions cover areas such as language support (e.g., Python, JavaScript), debugging, version control (Git), themes, and more. Extensions can be easily installed from the Extensions Marketplace.

Integrated Terminal: It features an integrated terminal that allows developers to run shell commands, scripts, and other tools directly within the editor, enhancing productivity by eliminating the need to switch between applications.

Source Control Integration: VS Code integrates seamlessly with version control systems like Git. It provides visual indicators for file changes, supports staging, committing, and pushing changes to remote repositories—all from within the editor.

Debugging Support: Developers can debug their applications directly within VS Code using its built-in debugger. It supports various languages and frameworks, allowing for breakpoints, variable inspection, call stack navigation, and more.

JUPYTER NOTEBOOK :

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science, scientific computing, and education due to its interactive and exploratory nature. Here are some key details about Jupyter Notebook:

Interactive Environment: Jupyter Notebook provides an interactive computing environment where you can write and execute code in various programming languages, including Python and others. Each notebook runs a separate kernel that executes the code and sends the results back to the notebook interface.

Notebook Format: Notebooks are organized into cells that can contain code, Markdown text, equations (using LaTeX syntax), or raw text. This combination allows you to create rich, multimedia documents that integrate code with explanatory text, images, plots, and mathematical formulas.

Code Execution: You can execute code cells individually or all at once. Results, including output, errors, and plots, are displayed directly below the corresponding code cell. This immediate feedback facilitates iterative development and exploration of data and algorithms.

Rich Output: In addition to plain text output, Jupyter Notebook supports rich media representations, such as HTML, images, audio, and video. This capability enables the creation of interactive visualizations and multimedia presentations directly within the notebook.

Extensions and Widgets: Jupyter Notebook can be extended with plugins and widgets that add additional functionality and interactivity. For example, widgets allow you to create interactive GUI components (like sliders, checkboxes) to control and visualize data dynamically.

Sharing and Collaboration: Notebooks can be easily shared with others via email, Dropbox, GitHub, or the Jupyter Notebook Viewer (NBViewer). This makes it straightforward to disseminate research findings, educational materials, or data analysis workflows.

Integrated Development Environment (IDE): While primarily a web application, Jupyter Notebook provides a comprehensive IDE-like experience with features such as syntax highlighting, code completion, and integrated help/documentation.

Support for Various Languages: Originally developed for Python, Jupyter has expanded to support a variety of programming languages through different kernels. This versatility allows users to work in their preferred language while leveraging Jupyter's interactive features.

REFERENCES:

01.https://ieeexplore.ieee.org/abstract/document/9061155

02.https://www.sciencedirect.com/science/article/abs/pii/S095741742201171X

03.https://ieeexplore.ieee.org/abstract/document/8457691

04.https://ieeexplore.ieee.org/abstract/document/9598291

05.https://isprs-archives.copernicus.org/articles/XLIII-B1-2022/437/2022/isprs-archives-XLIII-B1-2022-437-2022.html

06. https://ieeexplore.ieee.org/abstract/document/10578218

07. https://ieeexplore.ieee.org/abstract/document/9431076

: