PROJECT TOPIC :-

Real-time Object Detection, Tracking

SAYAN ROY CHOWDHURY

INDEX

INTRODUCTION

- PURPOSE
  The basic idea of **RODTSAS (Real-time Object Detection, Tracking, and Spatial Analysis System)** is to provide a comprehensive framework for detecting, tracking, and analyzing objects in real time. It aims to enhance situational awareness by identifying objects, defining region-of-interest (ROI) dimensions, estimating source distances, and predicting future movement paths. The system integrates cutting-edge computer vision and predictive analytics to offer a robust solution for various applications, including surveillance, traffic monitoring, and autonomous navigation. We strongly aim to replace traditional object detection systems, which focus only on identification, with a more advanced, **context-aware** system that enables real-time decision-making and spatial intelligence.

- SCOPE
  - **Real-Time Object Detection**
    - The system uses YOLOv8 to detect and identify objects in real time through a webcam.
    - Efficiently tracks objects within the camera's field of view.
  - **Object Measurement**
    - Calculates the distance and dimensions of detected objects using depth estimation techniques.
    - Useful for applications in surveillance, robotics, and autonomous vehicles.
  - **Autonomous Decision-Making**
    - The system is designed to mimic an artificial brain for autonomous vehicles or robots.
    - Makes real-time decisions to avoid obstacles and adjust movement direction.
  - **Multi-ROI (Region of Interest) Monitoring**
    - Monitors multiple regions using customizable ROI areas for targeted object detection.

- Provides region-specific object count data for analytics and surveillance.

Definitions, Acronyms, and Abbreviations

Definitions

1. Object Detection – The process of identifying and classifying objects in an image or video stream using computer vision techniques.
2. Object Tracking – Continuously following detected objects across multiple frames to analyze movement patterns.
3. Region of Interest (ROI) – A specific area in an image or video where the system focuses its detection and analysis.
4. Depth Estimation – A technique used to calculate the distance of objects from the camera using stereo vision or depth sensors.
5. Autonomous Decision-Making – The ability of an AI-driven system to make real-time choices based on object detection and tracking data.
6. Surveillance System – A security-focused application that monitors activities in a specific area using cameras and AI-based analysis.
7. Predictive Analytics – The use of AI and machine learning to anticipate future object positions and movements.
8. Multi-ROI Monitoring – A feature allowing multiple predefined regions in a video feed to be analyzed separately for different objects.
9. Collision Avoidance – A technique used in robotics and autonomous vehicles to prevent crashes by analyzing detected objects' movement patterns.
10. Machine Learning Model – An AI-based algorithm that learns from data and improves decision-making over time.

Acronyms and Abbreviations

- RODTSAS – Real-time Object Detection, Tracking, and Spatial Analysis System
- YOLO – You Only Look Once (Object detection model)
- ROI – Region of Interest

- LiDAR – Light Detection and Ranging (a depth-sensing technology)
- CV – Computer Vision
- FPS – Frames Per Second (measures video processing speed)
- RGB – Red, Green, Blue (color channels used in image processing)
- CNN – Convolutional Neural Network (used for image analysis)
- HOG – Histogram of Oriented Gradients (a feature descriptor for object detection)
- SSD – Single Shot Detector (another object detection model)

Technologies used in RODTSAS

Object Detection and AI

- *YOLOv8 (You Only Look Once):* State-of-the-art object detection model for real-time object detection and classification.
- *Ultralytics YOLO Library:* Provides a simple and efficient interface to train, validate, and deploy YOLO models.

Programming Languages

- *Python:* Primary language for implementing object detection, data analysis, and decision-making algorithms.

Computer Vision

- *OpenCV:* For video processing, image manipulation, and real-time visualization of detected objects.

Mathematics and Data Processing

- *NumPy:* For efficient mathematical operations and matrix manipulations.
- *Pandas:* Used to store and manipulate prediction data for further analysis.

Visualization and Debugging

- *Matplotlib and Seaborn:* For visualizing object detection results, frame analysis, and debugging.

Hardware Interface

- *Webcam or Camera:* Captures live video feed for object detection.
- *GPU (NVIDIA CUDA):* Accelerates model inference using GPU computing for real-time performance.

Machine Learning Model Management

- *Joblib:* For saving and loading trained models efficiently.

Automation and Video Processing

- *TQDM:* For tracking progress in video processing tasks.

Development Environment

- *Visual Studio Code:* Preferred IDEs for project development.

Overview

Existing System

Current object detection and tracking systems face multiple challenges, such as:

1. Limited Spatial Awareness – Many object detection systems fail to measure object distance and dimensions accurately.
2. No Future Path Correction and Prediction – Conventional tracking systems do not predict the movement trajectory of detected objects.
3. Restricted ROI Monitoring – Existing systems do not provide multi-ROI analysis for different regions in a video feed.
4. Lack of Autonomous Decision-Making – Most tracking systems only detect and track objects without making intelligent decisions for navigation or alerting users.

Drawbacks

1. Many real-time object detection models do not support accurate depth estimation and source distance measurement.
2. Existing object tracking solutions fail to predict the future movement of detected objects.
3. Lack of secure and adaptable mechanisms to control information access based on user roles.

Our Plan

To overcome these challenges, RODTSAS provides the following:

Real-Time Object Detection & Tracking

- Utilize **YOLOv8** for detecting and tracking multiple objects in real time.
- Implement **Filters** for smooth tracking and predicting the object's future path.

Region-Based Object Detection & Tracking

- Define **customizable Region of Interest (ROI)** to track objects in specific areas.
- Monitor object movement within different ROIs for surveillance and analytics.

Spatial Analysis (Distance & Dimension Estimation)

- Use depth estimation techniques to measure **object dimensions and distances**.
- Assist in applications like **autonomous navigation, smart surveillance, and industrial automation**.

Predictive Analytics & Future Path Prediction

- Apply **Filters** and trajectory analysis to predict future object movements.
- Helps in **autonomous decision-making**, preventing collisions, and optimizing object tracking.

Multi-Object Categorization & Counting

- Categorize detected objects based on **class labels** (people, vehicles, etc.).
- Generate **real-time statistical reports** on object count and movement trends.

Autonomous Decision-Making System

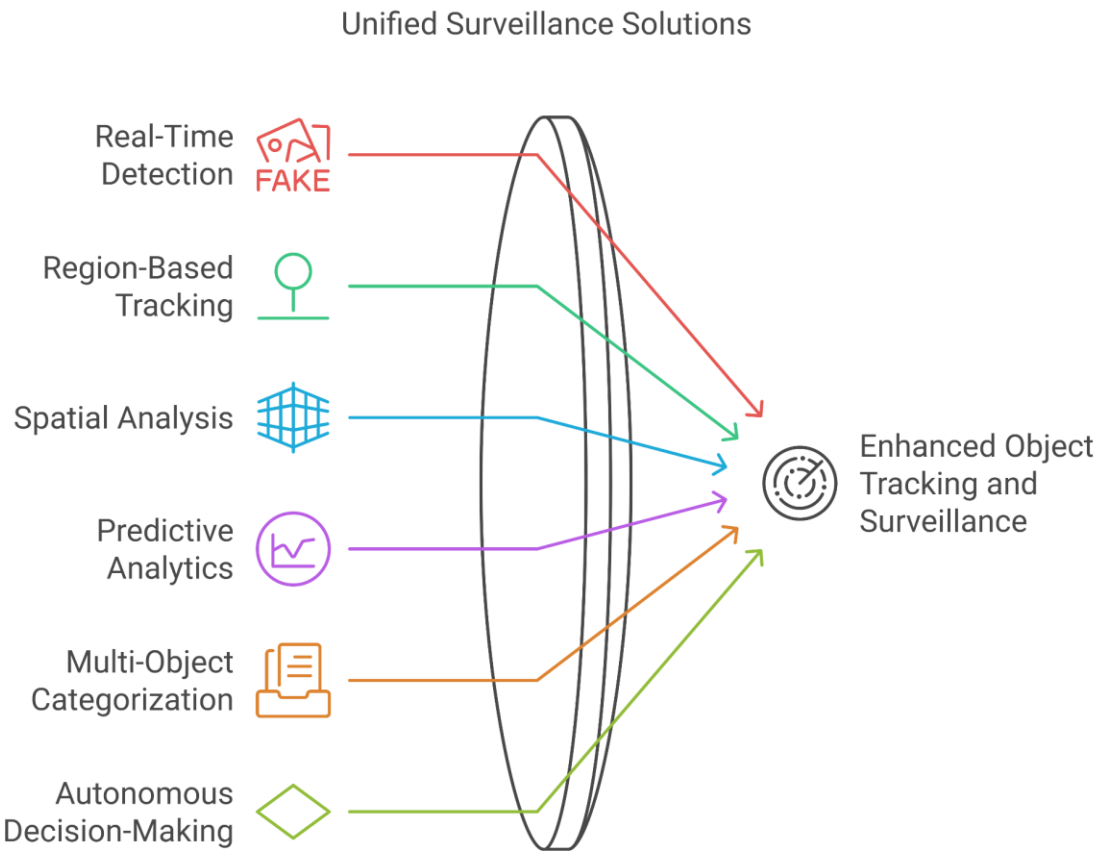
- Integrate AI-based decision-making for **robotics and autonomous vehicles**.
- Adjust navigation or send alerts based on detected objects and their predicted paths.

Smart Surveillance & Industrial Monitoring

- Detect unauthorized access in restricted areas.
- Track employee movement in industrial settings for safety compliance.

Statistical Report Generator & User Alerts

- Provide **graphical reports** on detected objects, movement history, and system efficiency.
- Implement **email/SMS alerts** for specific events like object intrusion or threshold breaches.



Software interface

Client Side

Web Browser, Operating System (any)

Web Server

WASCE, Operating System (any)

Development End

Python, Ultralytics, YOLOv8, TensorFlow, Matplotlib, Numpy, Seaborn, Pandas, WebCam, OpenCV, TQDM,

Hardware Interface

Requirements: -

Hardware Equipments			
Browser	Processor	RAM	WebCam Resolution
Google Chrome 11	Intel i7 13th Gen or AMD Ryzen 7000 series	16 GB	1080p

Development Side			
YOLOv8	All Intel or AMD - 1 GHZ	Version 8	3 GB
OpenCV			500 MB (Excluding Data Size)

Product Functions

1. Initialization Phase

- Hardware Setup: Connect a webcam or external camera for real-time video capture.
- Model Loading: Load the YOLOv8 pre-trained model using the Ultralytics library.
- Define Parameters: Set parameters like confidence threshold, frame size, and Region of Interest (ROI) areas.

2. Video Capture and Pre-Processing

- Video Stream Capture:
  - Use OpenCV to capture frames from the webcam in real time.
- Resize Frames:
  - Resize the captured frames to optimize detection speed and accuracy.
- Define ROI (Optional):
  - Specify regions in the video where object detection will be focused.
  - Useful for monitoring specific areas (e.g., restricted zones or traffic lanes).

3. Object Detection using YOLOv8

- Apply YOLOv8 Model:
  - Perform object detection using the YOLOv8 model.
  - Extract bounding box coordinates, class labels, and confidence scores.
- Filter Detections:
  - Apply a confidence threshold to filter out low-confidence detections.
- Class Identification:
  - Recognize the object type using YOLO's class labels.

4. Tracking and Measurement

- Object Tracking:
  - Track objects across frames using a unique ID for each detected object.
- Distance Calculation:
  - Calculate the distance between the camera and the detected object using size estimation and focal length methods.
- Dimension Estimation:
  - Estimate object dimensions based on bounding box data and known object size ratios.

5. Decision-Making and Obstacle Avoidance (Optional)

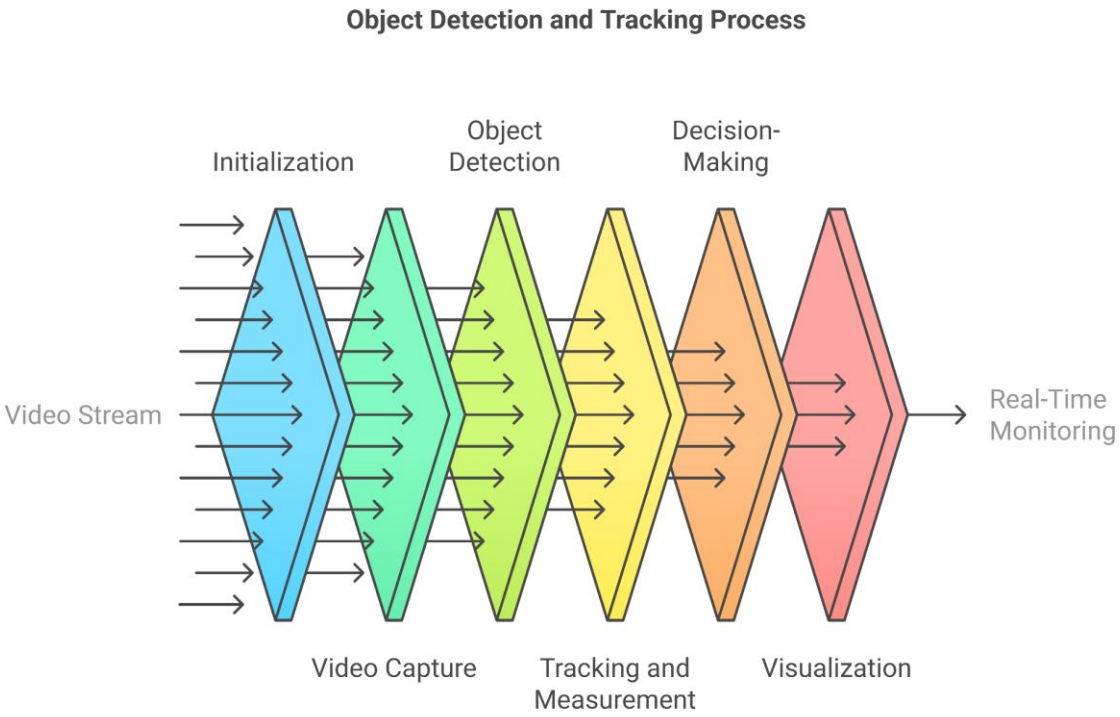
- Path Planning:
  - If an object is detected as an obstacle, apply path-planning algorithms to determine a safer path.
- Autonomous Movement:
  - Send real-time commands to a robot or vehicle for obstacle avoidance.
- AI Decision-Making:
  - Implement reinforcement learning models to improve movement strategies based on environmental data.

6. Visualization and Logging

- Display Results:
  - Display the video feed with bounding boxes, object labels, and confidence scores.
  - Overlay the object distance and dimensions on the frame.
- Logging Data:
  - Save detected object data (e.g., class, position, distance) for analysis using Pandas.
- Generate Reports:
  - Create visual reports using Matplotlib or Seaborn for further insights.

7. Final Output

- Real-Time Monitoring:
  - View detected objects live through a window using OpenCV.



Code And Output

```
pip install ultralytics
```

```
Requirement already satisfied: ultralytics in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (8.3.87)Note:

Requirement already satisfied: numpy<=2.1.1,>=1.23.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (fr
Requirement already satisfied: matplotlib>=3.3.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from u
Requirement already satisfied: opencv-python>=4.6.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (fro
Requirement already satisfied: pillow>=7.1.2 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultra
Requirement already satisfied: pyyaml>=5.3.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultra
Requirement already satisfied: requests>=2.23.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ul
Requirement already satisfied: scipy>=1.4.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultral
Requirement already satisfied: torch>=1.8.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultral
Requirement already satisfied: torchvision>=0.9.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from
Requirement already satisfied: tqdm>=4.64.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultral
Requirement already satisfied: psutil in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultralytics)
Requirement already satisfied: py-cpuinfo in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultralyt
Requirement already satisfied: pandas>=1.1.4 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ultra
Requirement already satisfied: seaborn>=0.11.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ult
Requirement already satisfied: ultralytics-thop>=2.0.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ma
Requirement already satisfied: cycler>=0.10 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from matplo
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from m
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from m
Requirement already satisfied: packaging>=20.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from mat
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from ma
Requirement already satisfied: python-dateutil>=2.7 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (fro
Requirement already satisfied: pytz>=2020.1 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from pandas
...
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from
Requirement already satisfied: colorama in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from tqdm>=4.64
Requirement already satisfied: six>=1.5 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from python-dat
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sayan roy chowdhury\appdata\local\programs\python\python312\lib\site-packages (from jin
```

```python
import cv2
from ultralytics import YOLO
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import os
import subprocess
from tqdm.notebook import tqdm
import IPython
from IPython.display import Video, display
%matplotlib inline
```

[2] Python

```
c:\Users\SAYAN ROY CHOWDHURY\AppData\Local\Programs\Python\Python312\Lib\site-packages\torch\utils\_pytree.py:185: FutureWarning: optree is installe
  warnings.warn(
```

```python
model = YOLO('yolov8n.pt')
dict_classes = model.model.names
```

[3] Python

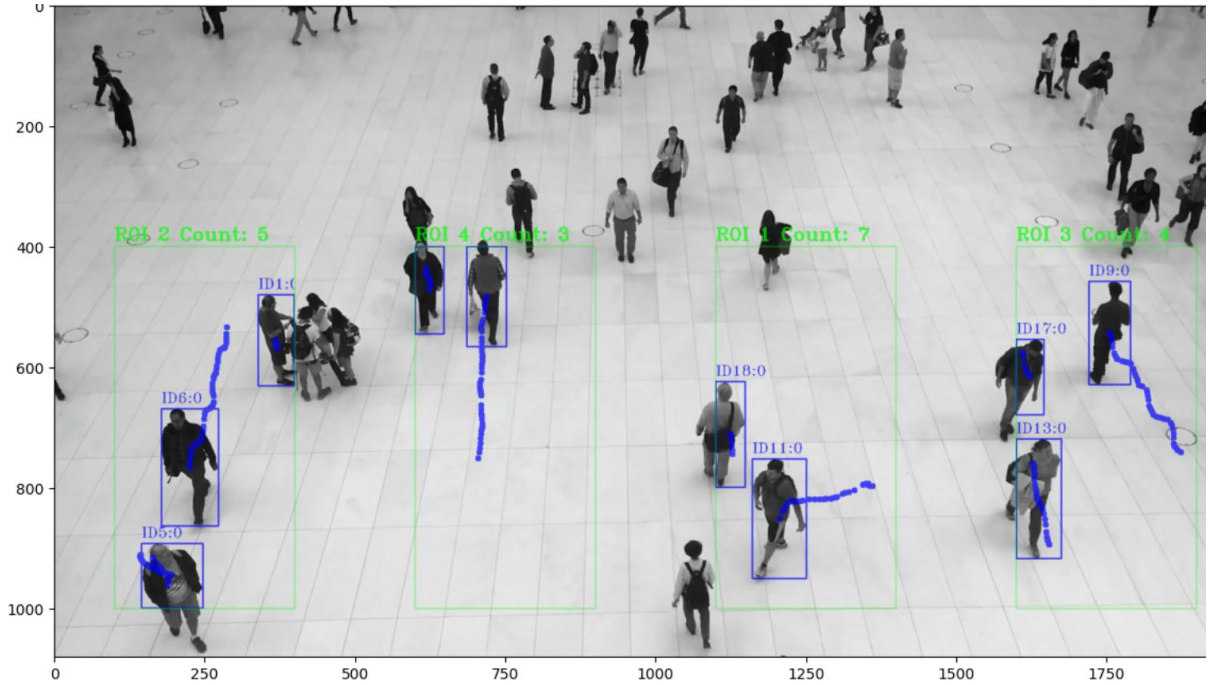
```python
def resize_frame(frame, scale_percent):
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    return cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)

def filter_tracks(centers, patience):
    return {k: dict(list(i.items())[-patience:]) for k, i in centers.items()}

def update_tracking(centers_old, obj_center, thr_centers, lastKey, frame, frame_max):
    is_new = 0
    lastpos = [(k, list(center.keys())[-1], list(center.values())[-1]) for k, center in centers_old.items()]
    lastpos = [(i[0], i[2]) for i in lastpos if abs(i[1] - frame) <= frame_max]
    previous_pos = [(k, obj_center) for k, centers in lastpos if (np.linalg.norm(np.array(centers) - np.array(obj_center)) < thr_centers)]
    if previous_pos:
        id_obj = previous_pos[0][0]
        centers_old[id_obj][frame] = obj_center
    else:
        id_obj = 'ID' + str(int(lastKey.split('D')[1]) + 1) if lastKey else 'ID0'
        is_new = 1
        centers_old[id_obj] = {frame: obj_center}
        lastKey = list(centers_old.keys())[-1]
    return centers_old, id_obj, is_new, lastKey
```

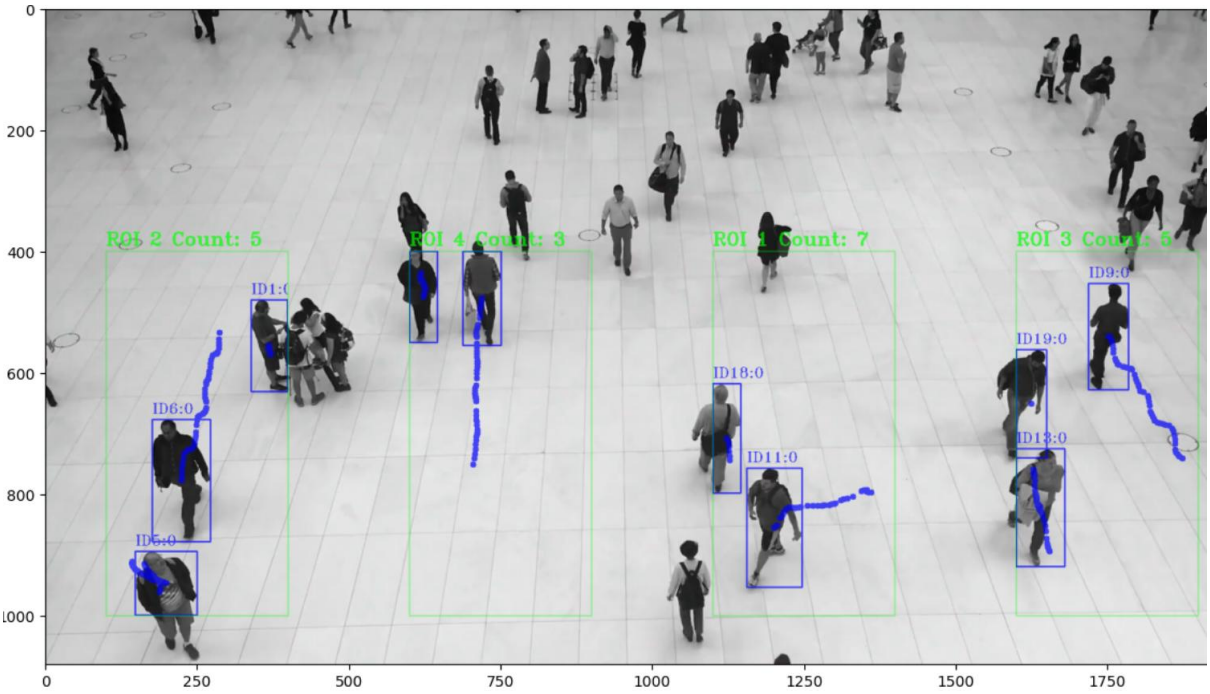
[4] Python

```python
path = 'my-data-private/vid1.mp4'
verbose = False
scale_percent = 100
conf_level = 0.8
thr_centers = 20
frame_max = 5
patience = 100
alpha = 0.3

# Read video
video = cv2.VideoCapture(path)
height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
fps = video.get(cv2.CAP_PROP_FPS)

# Define four ROI areas
roi_areas = [
    np.array([(1400, 400), (1100, 400), (1100, 1000), (1400, 1000)], np.int32),
    np.array([(400, 400), (100, 400), (100, 1000), (400, 1000)], np.int32),
    np.array([(1900, 400), (1600, 400), (1600, 1000), (1900, 1000)], np.int32),
    np.array([(900, 400), (600, 400), (600, 1000), (900, 1000)], np.int32)
]

output_path = "result.mp4"
tmp_output_path ="tmp_result.mp4"
VIDEO_CODEC = "MP4V"
output_video = cv2.VideoWriter(tmp_output_path, cv2.VideoWriter_fourcc(*VIDEO_CODEC), fps, (width, height))

centers_old = {}
lastKey = ''
roi_counts = {i: 0 for i in range(len(roi_areas))}  # Track people count for each ROI
frames_list = []

for i in tqdm(range(int(video.get(cv2.CAP_PROP_FRAME_COUNT)))):
    _, frame = video.read()
    if frame is None:
        break
    frame = resize_frame(frame, scale_percent)

    overlay = frame.copy()
```

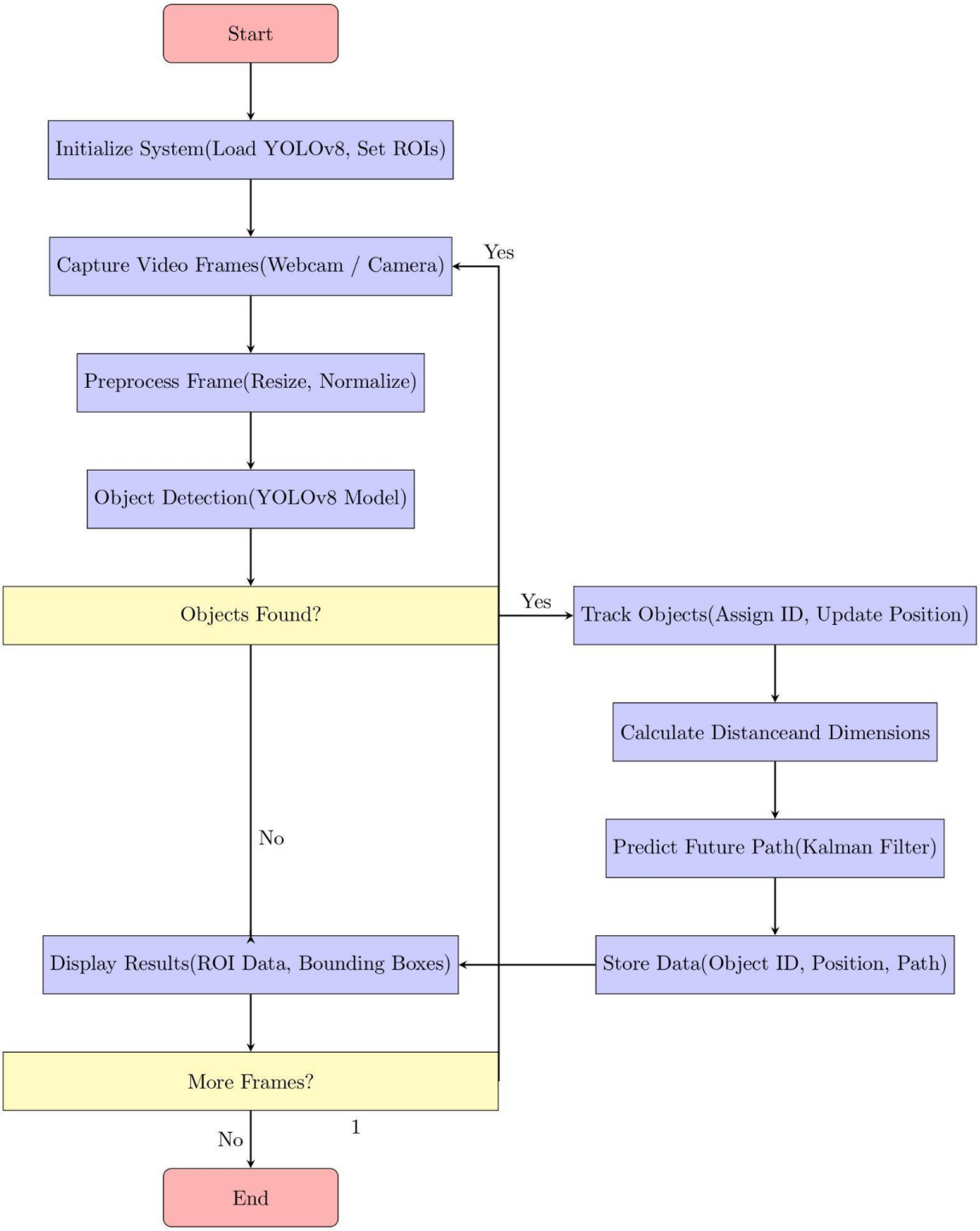
```
    for roi_idx, roi_area in enumerate(roi_areas):
        x_min, y_min = np.min(roi_area, axis=0)
        x_max, y_max = np.max(roi_area, axis=0)
        ROI = frame[y_min:y_max, x_min:x_max]

        y_hat = model.predict(ROI, conf=conf_level, classes=[0], device=0, verbose=False)
        positions_frame = pd.DataFrame(
            np.column_stack([y_hat[0].boxes.xyxy.cpu().numpy(), y_hat[0].boxes.conf.cpu().numpy(),y_hat[0].boxes.cls.cpu().numpy()]),
            columns=['xmin', 'ymin', 'xmax', 'ymax', 'conf', 'class']
        )

        for ix, row in positions_frame.iterrows():
            xmin, ymin, xmax, ymax, confidence, category = row.astype(int)
            center_x, center_y = int((xmax + xmin) / 2), int((ymax + ymin) / 2)

            centers_old, id_obj, is_new, lastKey = update_tracking(
                centers_old, (center_x, center_y), thr_centers, lastKey, i, frame_max
            )

            if is_new:
                roi_counts[roi_idx] += 1  # Update count for the specific ROI

            cv2.rectangle(ROI, (xmin, ymin), (xmax, ymax), (0, 0, 255), 2)
            for cx, cy in centers_old[id_obj].values():
                cv2.circle(ROI, (cx, cy), 5, (0, 0, 255), -1)
            cv2.putText(ROI, f'{id_obj}:{str(np.round(confidence, 2))}',
                        (xmin, ymin - 10), cv2.FONT_HERSHEY_TRIPLEX, 0.8, (0, 0, 255), 1)

        # Draw ROI on overlay
        cv2.polylines(overlay, [roi_area], isClosed=True, color=(0, 255, 0), thickness=2)

    # Display ROI counts on the frame
    for idx, roi_area in enumerate(roi_areas):
        x_min, y_min = np.min(roi_area, axis=0)
        cv2.putText(frame, f'ROI {idx+1} Count: {roi_counts[idx]}',
                    (x_min, y_min - 10), cv2.FONT_HERSHEY_TRIPLEX, 1, (0, 255, 0), 2)

    frame = cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0)
    centers_old = filter_tracks(centers_old, patience)

    frames_list.append(frame)
    output_video.write(frame)

output_video.release()
if os.path.exists(output_path):
    os.remove(output_path)
subprocess.run(["ffmpeg", "-i", tmp_output_path, "-crf", "18", "-preset", "veryfast", "-hide_banner", "-loglevel", "error", "-vcodec", "libx264", output_path])
os.remove(tmp_output_path)
```

```
for i in [62,63, 64, 65, 81]:
    plt.figure(figsize =( 14, 10))
    plt.imshow(frames_list[i])
    plt.show()
```


ROI 2 Count: 5
ROI 4 Count: 3
ROI 1 Count: 7
ROI 3 Count: 4
ID1:0
ID6:0
ID5:0
ID18:0
ID11:0
ID9:0
ID17:0
ID13:0
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


ROI 2 Count: 5
ROI 4 Count: 3
ROI 1 Count: 7
ROI 3 Count: 5
ID1:0
ID6:0
ID5:0
ID18:0
ID11:0
ID9:0
ID19:0
ID13:0
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

Workflow Diagram



Constraints

- *Predefined ROI Areas* – Users must manually set Region of Interest (ROI) areas for object tracking.
- *Model Training Constraints* – The YOLOv8 model needs periodic retraining for improved accuracy with new datasets.
- *Limited Sensor Integration* – Currently supports camera-based detection; future versions may integrate LiDAR or depth sensors.

Conclusion

The Real-time Object Detection, Tracking, and Spatial Analysis System (RODTSAS) successfully integrates YOLOv8, Kalman Filters, and OpenCV to detect, track, and analyze objects in real-time. The system accurately identifies objects, measures distances, and predicts future movement paths, making it valuable for applications in surveillance, autonomous navigation, and traffic monitoring.

By leveraging multi-ROI monitoring and predictive analytics, RODTSAS enhances situational awareness and decision-making. Future improvements may include LiDAR integration, advanced deep learning models, and cloud-based processing for greater accuracy and efficiency.This project demonstrates the potential of AI-driven spatial analysis in transforming real-world applications.

References

- Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement.* arXiv preprint arXiv:1804.02767. https://arxiv.org/abs/1804.02767
- Jocher, G., et al. (2023). *Ultralytics YOLOv8: Real-Time Object Detection Model.* GitHub Repository. https://github.com/ultralytics/ultralytics
- Bradski, G. (2000). *The OpenCV Library.* Dr. Dobb's Journal of Software Tools. https://opencv.org/
- Welch, G., & Bishop, G. (1995). *An Introduction to the Kalman Filter.* University of North Carolina at https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

Acknowledgement

We sincerely thank Sir Subhabrata Sengupta for the invaluable guidance and support throughout this project, Real-time Object Detection, Tracking, and Spatial Analysis System (RODTSAS).

We appreciate our College for providing a learning environment that enabled our research. Special thanks to the open-source community for developing tools like YOLOv8, OpenCV, and Kalman Filters, which were essential to our system.

Lastly, we thank our peers, and family for their encouragement and motivation in completing this project.