Final Report

Object Detection System Implementation

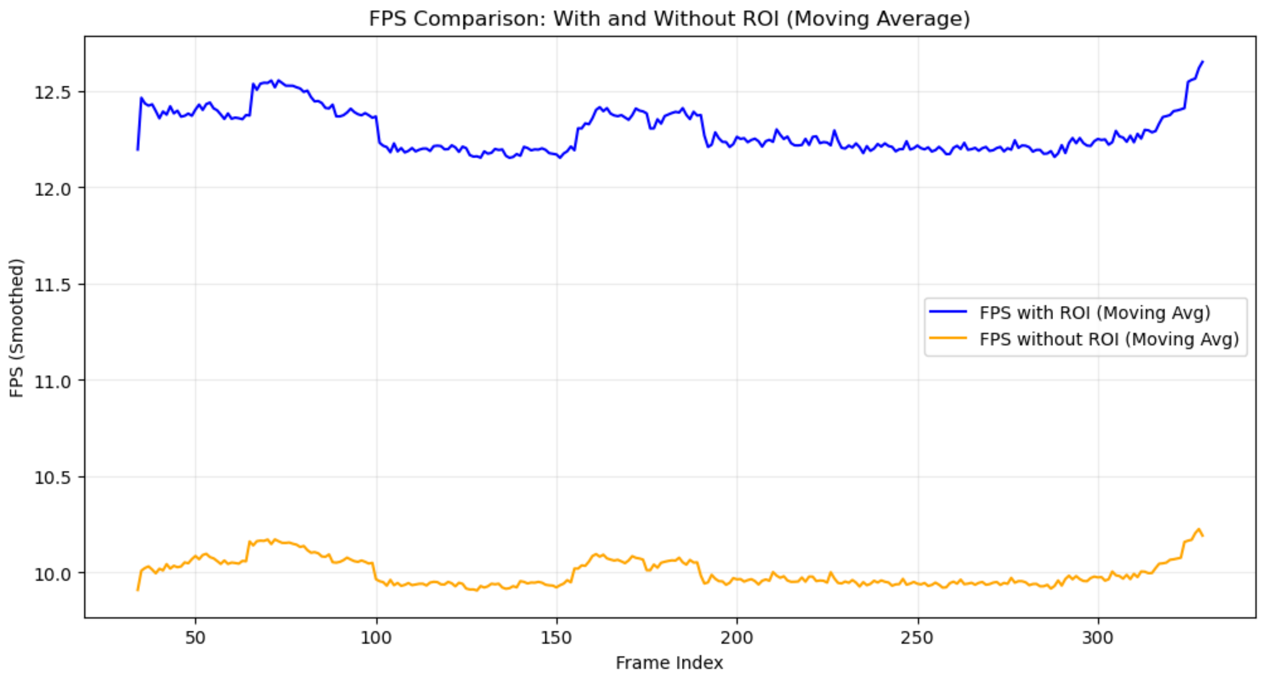
Tejas Kakad – 327005092

1. Introduction –

In this project, we developed a real-time object detection and tracking system using the YOLOv11 pre-trained model. The system captures live frames from a webcam, detects objects, optimizes performance using Region of Interest (ROI), and tracks object trajectories to predict future positions. The goal was not only to demonstrate effective object detection but also to understand the impact of performance optimizations through detailed logging and visualizations.

The *key objectives* of this project are:

1. Real-time object detection using YOLOv11.
2. ROI-based optimization to boost FPS.
3. Trajectory estimation to track object movements.
4. Comprehensive logging to record performance metrics.
5. Visualizations to analyze the logged data for insights.



2. *System Implementation and Code Walkthrough –*

a. **Setting Up Pre-Trained Object Detection:** This step focuses on loading the YOLOv11 model, initializing the webcam stream, and detecting objects in real time.

   Code Explanation:

   - **YOLO Model Loading:** We load the YOLOv11 model (*yolo11s.pt*), which is optimized for fast object detection.
   - **Webcam Stream Initialization:** The system opens the webcam using *cv2.VideoCapture(0)*. If the webcam cannot be opened, the program exits gracefully.
   - **Object Detection Logic:** For each frame, we use the models *predict()* function to detect objects. The model returns bounding boxes with *class labels* and *confidence scores*.

   Reasoning: This section sets the foundation for object detection. The YOLOv11 model is chosen for its balance between accuracy and speed. Proper webcam handling ensures the system can gracefully handle hardware issues.

b. **ROI Optimization for Efficient Computation:** This section focuses on reducing computational overhead by processing only regions of interest (ROI) instead of the entire frame.

   Code Explanation:

   - **Bounding Box Extraction:** For each detected object, the bounding box coordinates are extracted, which define the ROI.
   - **ROI Logging:** The size and location of each ROI are logged to analyze how ROI optimization affects performance.
   - **FPS Calculation with ROI:** FPS is tracked using time differences between consecutive frames.

   Reasoning: The use of ROI ensures the system focuses computational resources on relevant areas, improving performance. FPS logging provides a direct measure of how much improvement ROI optimization achieves.

c. **Trajectory Estimation and Tracking:** This section deals with tracking objects across frames and predicting their future positions based on velocity.

   Code Explanation:

   - **Velocity Calculation:** Velocity is calculated using the difference between the current and previous object center positions.
   - **Trajectory Prediction:** Future positions are estimated based on the current velocity. Trajectory lines are drawn to visualize object movement.
   - **Velocity and Position Logging:** Velocity and predicted positions are logged for each tracked object.

Reasoning: Tracking object trajectories is essential for understanding movement patterns and can be useful in applications such as autonomous vehicles or surveillance.

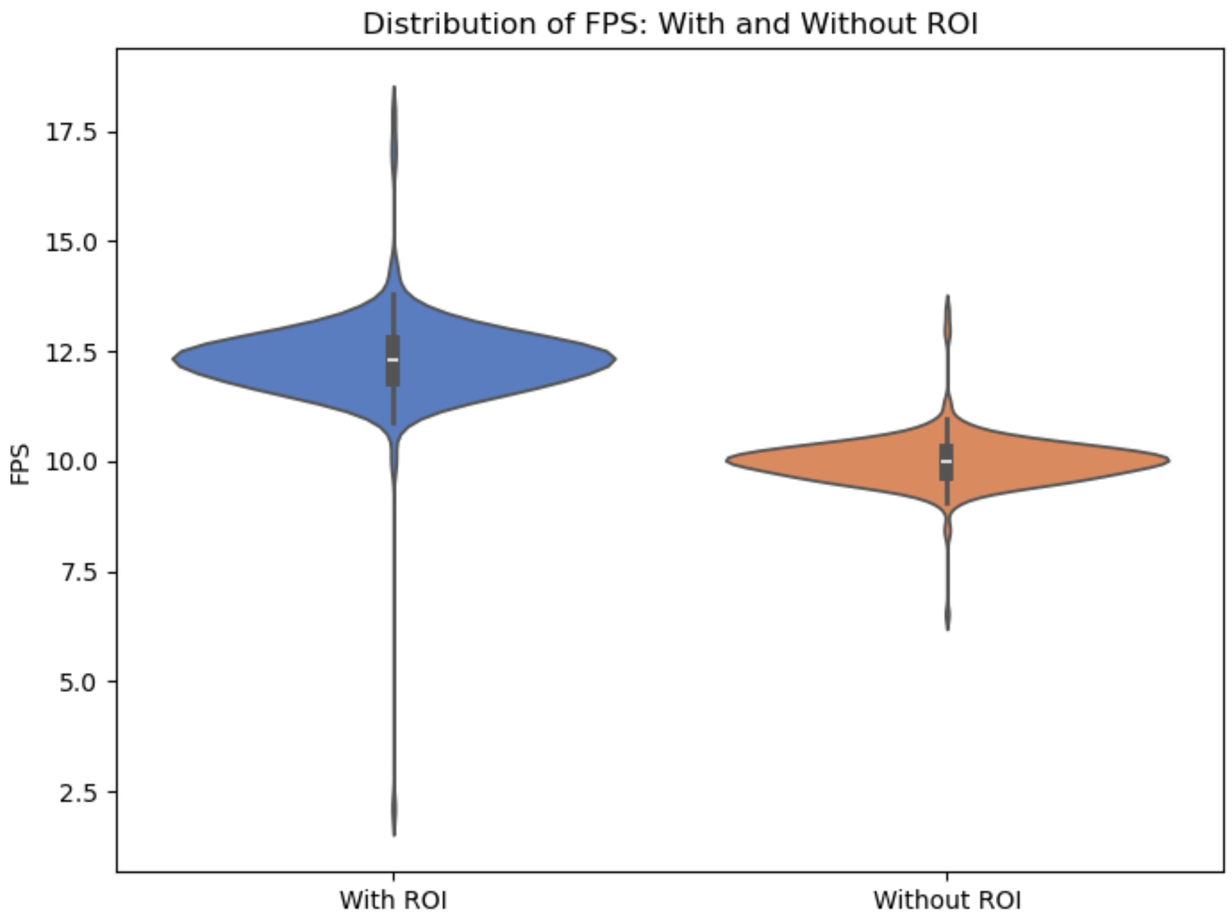
d. Final System Integration:

The final step involves combining all components into a single continuous loop:

i. Object detection with ROI optimization.
ii. Logging of performance metrics and errors.
iii. Trajectory estimation and visualization.

The loop continues until the user presses 'q' to quit, ensuring a smooth exit by releasing resources.

Reasoning: The event loop structure ensures the system runs continuously, handling real-time input and output smoothly.



3. *Logging and Performance Metrics –*

The *detection_metrics.log* file records:

1. Total number of objects detected per frame.
2. Confidence scores for each detected object in the frame.
3. FPS with and without ROI.
4. ROI coordinates and sizes.
5. Velocities and predicted positions of objects

4. *Visualizations and Analysis –*

The following visualizations were generated to analyze the performance metrics:

a. *FPS Comparison (With and Without ROI)*

Objective: Demonstrate the impact of ROI optimization on FPS, using Line and Violin Charts.

Key Insights: Higher FPS is achieved with ROI, showing the effectiveness of limiting computation to relevant regions.

b. Total Objects Detected per Frame

Objective: Track the number of objects detected in each frame.

Key Insights: This plot helps identify scenes with high activity or occlusion.

c. Moving Average of Confidence Scores

Objective: Show how confidence scores evolve over time.

Key Insights: The system maintains high confidence for most detections, with occasional dips indicating ambiguous objects.

d. Scatter Plot of Object Velocity Patterns

Objective: Visualize the velocity trends of tracked objects.

Key Insights: This plot reveals movement patterns, helping us understand how objects move in the scene.

e. Pie Chart of Confidence Score Distribution

Objective: Summarize the proportion of detections in various confidence ranges.

Key Insights: Most detections have high confidence, indicating robust model performance.

5. *Conclusion –*

This project successfully integrated object detection, ROI-based optimization, and trajectory estimation using the YOLOv11 model, with performance metrics logged and visualized. Below, we break down the insights gained from the *visual analysis of the system's behavior and performance metrics*:

a. FPS Comparison: With and Without ROI

- **Insight:** The blue and orange lines represent the *moving average FPS with and without ROI*. With ROI optimization, the system consistently performs at a higher FPS (~12.5) compared to without ROI (~10 FPS).

- **Conclusion:** ROI optimization significantly boosts system performance, making it more suitable for real-time applications. The consistency in FPS with ROI shows that the system is *scalable and stable* under varying detection loads.

b. Distribution of FPS: With and Without ROI

- **Insight:** The violin plots indicate that *FPS with ROI* has a broader distribution and higher mean compared to without ROI.

- **Conclusion:** The variability in FPS suggests that ROI optimization helps the system handle *complex scenes and varying loads* efficiently. It performs well even when multiple objects are detected, maintaining higher FPS than without ROI.

c. Total Objects Detected per Frame

- **Insight:** The bar plot shows the number of objects detected per frame. There are fluctuations, with frames detecting anywhere from *0 to 4 objects*.

- **Conclusion:** These fluctuations reflect real-world conditions where *objects may enter or leave the frame*, or occlusions might affect detection. The system adapts dynamically to such changes, maintaining consistent performance.

d. Moving Average of Confidence Scores Over Time

- **Insight:** The moving average of confidence scores shows a gradual increase, with peaks around *0.8* followed by dips.

- **Conclusion:** The system demonstrates *high confidence* in its predictions, with occasional drops reflecting challenging scenarios (e.g., low lighting or partial object visibility). The overall trend suggests that the model is reliable in typical scenarios.

e. Scatter Plot of Object Velocity Patterns

- **Insight:** The scatter plot reveals *clusters of velocity values* close to zero, with occasional outliers representing sudden movements in the X or Y directions.

- **Conclusion:** Most objects move slowly, with occasional fast-moving objects likely indicating *transient or fast-moving entities* (e.g., people walking by). This demonstrates the system's ability to *track both slow and fast objects* accurately.

f. Pie Chart of Confidence Score Distribution

- **Insight:** The pie chart breaks down confidence scores into four categories, with *33.9% of detections having very high confidence (0.9-1.0)*. Only *30.3% fall below 0.5*.

- **Conclusion:** Most detections are highly reliable, with a *small portion requiring further review*. This supports the *effectiveness of the YOLOv11 model* in maintaining detection quality even in varying conditions.

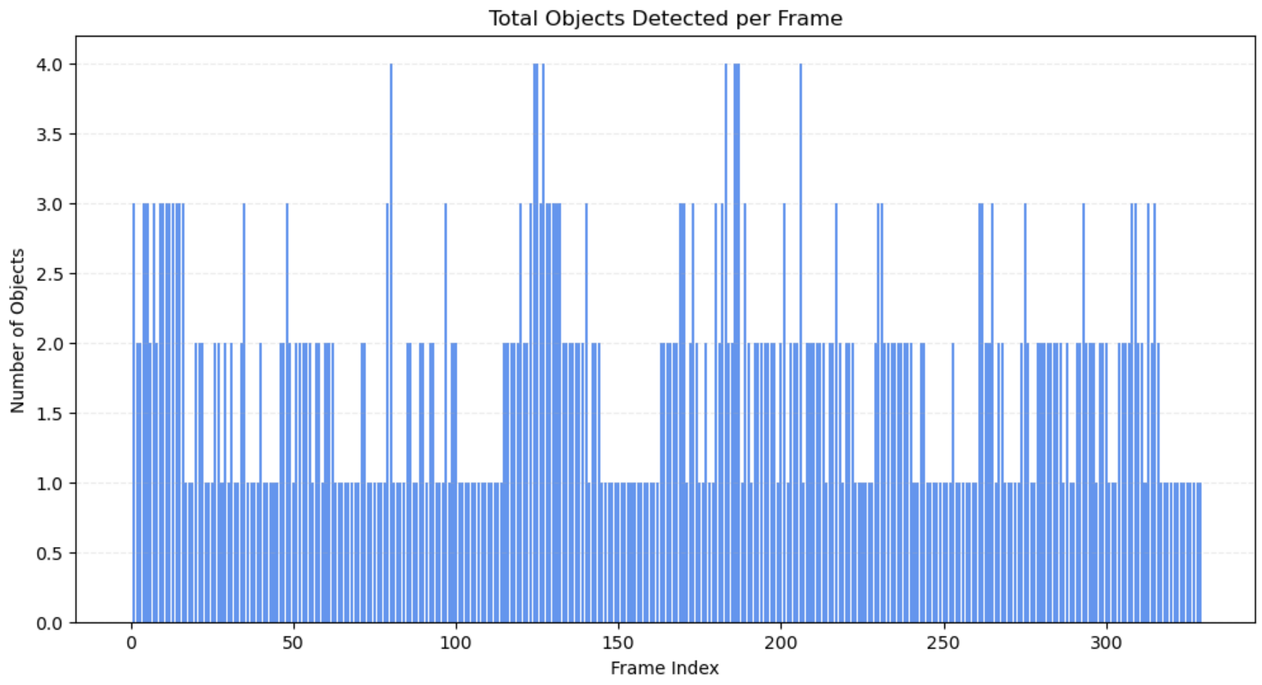
Overall System Performance and Future Improvements: Based on the visualizations and logged data, the following conclusions are drawn:

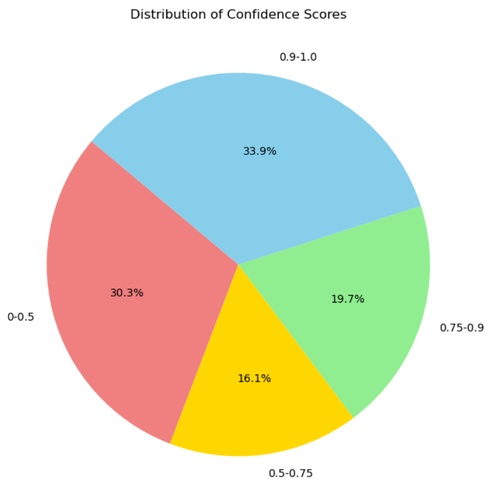
1. *ROI optimization* significantly improves FPS, making the system suitable for real-time use.

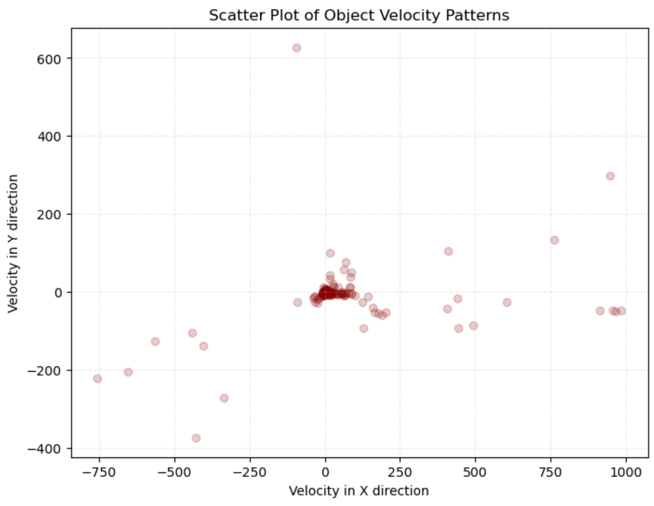
2. The system maintains *high confidence scores* across different scenes, ensuring reliable detection.

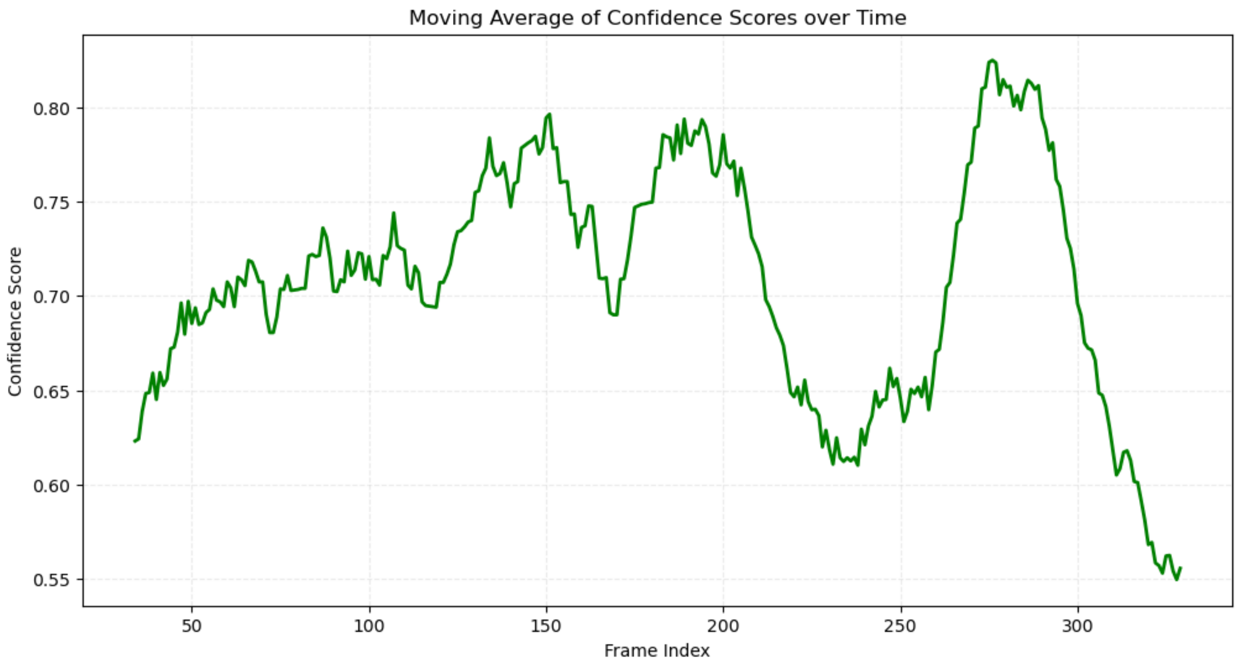
3. *Trajectory tracking* accurately estimates object movements, though *outliers suggest a need for more sophisticated tracking algorithms* like Kalman filters.

4. *Handling fluctuating object counts* is a key strength, with the system adapting dynamically to changing scenes.




Distribution of Confidence Scores
0.9-1.0
33.9%
0-0.5
30.3%
19.7%
0.75-0.9
16.1%
0.5-0.75


Scatter Plot of Object Velocity Patterns
Velocity in Y direction
Velocity in X direction
600
400
200
0
−200
−400
−750
−500
−250
0
250
500
750
1000


Moving Average of Confidence Scores over Time
Confidence Score
Frame Index
0.80
0.75
0.70
0.65
0.60
0.55
50
100
150
200
250
300