INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 4, April 2026 | ISSN: 2456-4184 | IJNRD.ORG

SARAYU – DRONE-BASED AI TRAFFIC SIGNAL AUTOMATION SYSTEM

[1]Sravanthi Dubanisi, [2]Aileni Varshith Reddy, [3]Kunde Badrinath, [4]Kyatam Nihal Reddy, [5]Mitta Sudheer Reddy

[12345]B.Tech Student, Department of Information Technology,
Vidya Jyothi Institute of Technology, Hyderabad, Telangana, India

Abstract: Most traffic lights in India still run on set timers, ignoring the chaos and constant changes at busy intersections. That means more traffic jams, wasted fuel, frustrated drivers, and emergency vehicles stuck waiting—plus, some directions get way more green time than others, no matter how crowded they are. We introduce SARAYU, a drone-powered smart traffic signal system created for the Smart India Hackathon 2025. A drone flies over a four-way intersection and streams live video. With YOLOv8-based object detection, SARAYU spots and counts vehicles in each direction and figures out if they're moving or stuck. The system tracks every vehicle across frames, assigns higher priority to heavy vehicles like buses and trucks, and uses a custom algorithm to decide which direction gets the green light and for how long. Signal orders are relayed to a microcontroller which directly controls the traffic lights. Tests show the system detects vehicles accurately, sorts all kinds of traffic, cuts down waiting time, and saves fuel compared to old-school fixed-timer lights.

Index Terms - YOLOv8, Drone-Based Traffic Monitoring, Adaptive Signal Control, Region of Interest, Moving/Stationary Classification, Arduino, Smart India Hackathon, Indian Traffic Management, Real-Time Object Detection.

I. INTRODUCTION

Traditional traffic signals in India stick to set timing cycles. They don't react to real, changing traffic at intersections. So, you end up with lanes packed full of frustrated drivers getting stuck for ages, while empty lanes sit at green lights, wasting time and fuel. The whole system feels out of sync. On top of that, these fixed signals ignore the mix of cars, bikes, buses, and trucks — each with its own speed and size — all converging at chaotic four-way crossings. This rigid setup just doesn't cut it. We need smarter, responsive systems that actually pay attention to what's happening on the roads.

This paper introduces SARAYU, a drone-powered traffic signal system developed for the Smart India Hackathon 2025. SARAYU blends aerial surveillance with real-time computer vision to fine-tune signal timing, using live traffic data instead of stale assumptions. A drone hovers above intersections, streaming video that captures every approach lane from above. Compared to ground cameras, this bird's-eye view avoids blind spots and catches the whole picture, even with vehicles constantly weaving and changing lanes.

That video feeds into a YOLOv8 object detection model, which sorts vehicles into categories — cars, motorcycles, buses, trucks — and counts them direction by direction. The system tracks every vehicle frame by frame, so it knows who's moving and who's waiting. SARAYU uses a priority-based algorithm to share out green light durations dynamically, weighing vehicles by size and impact. Signal timings are sent straight to a microcontroller, which updates the lights in real time.

Tests show SARAYU nails vehicle detection and sorts moving from stationary traffic reliably, even during heavy congestion. By adjusting green times smartly, it cuts average waiting times and lifts overall throughput compared to old fixed-cycle signals. Fewer delays and smoother flow also mean less fuel wasted and lower emissions.

II. LITERATURE REVIEW

A. Conventional and Adaptive Systems

Fixed-time signal controllers have been around since Webster's formula back in the late '50s. They work okay if traffic is always predictable, but that's rarely the case—especially in India. Vehicle-actuated signals take things a step further by using loop detectors under the road, but even they just react to whatever happened in the last moment. And because Indian roads are a mix of everything—cars, rickshaws, buses and sometimes animals—these sensors often have no clue what's going on in each lane.

When it comes to adaptive signal systems, two names show up everywhere: SCOOT and SCATS. Running in cities worldwide, these systems get praise for cutting average delays by up to 20% over fixed-timers, as Robertson and Bretherton pointed out. But there's a catch: they need lots of fancy hardware and strict lane discipline—a tough ask in Indian cities.

B. YOLO Family of Models

Redmon and his team kicked off something big with YOLO in 2016. Instead of slowly scanning each image for objects, YOLO breaks the image into a grid and checks everything at once—it finds, classifies, and draws boxes around objects in a single pass. YOLOv8, released by Ultralytics in 2023, introduced an anchor-free architecture with improved performance—88% mAP on VisDrone at 15–25 FPS (Bakirci, 2025). Patel and Deshmukh (2024) reported over 90% accuracy for standard vehicles on Indian roads in daylight, recommending a confidence threshold of 0.1–0.25.

C. Drone-Based Surveillance Systems

Drones beat fixed road cameras hands down when it comes to watching traffic. From the sky, you see every road leading up to an intersection. Vehicles don't get hidden behind each other, making counting and tracking way easier. Zhu et al. (2022) noted challenges including scale variation as drones move, vehicle clustering, tricky camera angles, and small vehicle detection. Ke et al. (2020) and Xu et al. (2021) showed that drones can pull reliable traffic data, but stopped short of connecting that data to actually changing traffic signals — that's exactly where SARAYU comes in.

D. Research Gaps Identified

A few gaps jump out from the research. Nobody has really connected live drone vehicle detection straight to controlling traffic lights, especially not with the unpredictable, mixed traffic found in India. Most adaptive systems lean on Western data or run on simulators. There also isn't any work that links aerial detection with figuring out if a line of vehicles is moving or stuck. Finally, deep reinforcement learning like DQN needs a lot of GPU power, making rapid field deployment difficult.

III. METHODOLOGY

A. System Architecture

This system follows a modular, five-layer design. At the base, the Input Layer grabs raw video from a drone-mounted camera using OpenCV's VideoCapture. Up next, the Detection and Tracking Layer runs YOLOv8 Nano on every frame, assigns unique IDs to each vehicle with an IoU-based tracker, and sorts them as Moving or Stationary by checking centroid displacement. The Decision Layer maps vehicle centroids to specific ROI zones and works out green signal times based on priority. The Hardware Control Layer sends timing commands straight to an Arduino using PySerial. On top, the Monitoring Layer runs a Flask server that streams real-time vehicle counts and signal states to a dashboard.

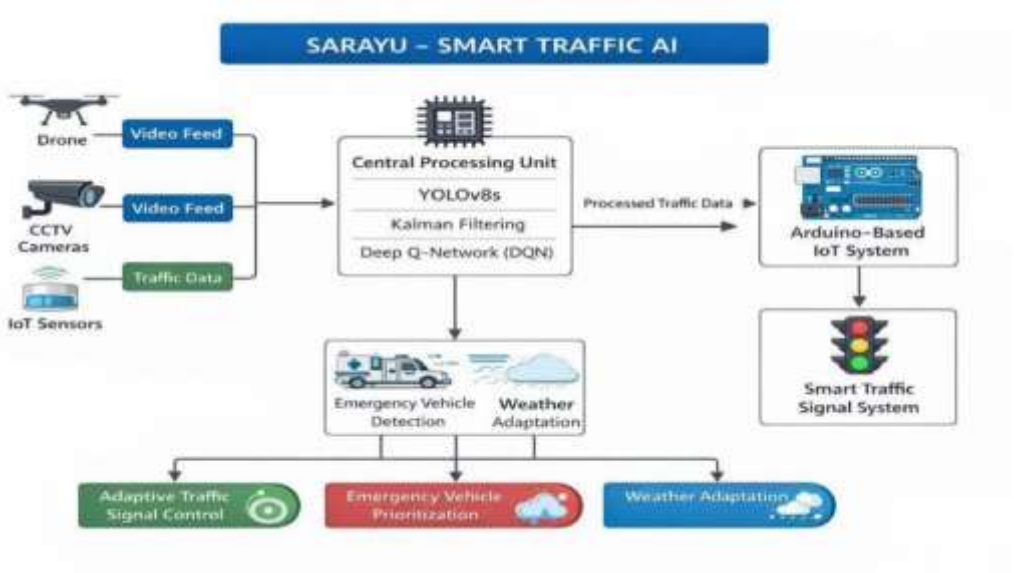

Fig. 1 — System Architecture

B. YOLOv8 Architecture

YOLOv8 Nano (yolov8n.pt) is the model of choice for its mix of speed and decent accuracy. It's lightweight — about 3.2 million parameters, 6.2 MB. The model has three key pieces: the CSPDarknet Backbone, which grabs detailed feature maps from each 640×640 input frame; the PANet Neck, which blends these features together for strong detection at any altitude or vehicle size; and an Anchor-Free Detection Head, which spits out bounding boxes and class predictions all in one go. The system uses a low confidence threshold of 0.10 to 0.20 to catch as many vehicles as possible in dense Indian traffic. On basic CPUs, this setup runs at about 6–15 FPS, and jumps to 20–30 FPS with a GPU.

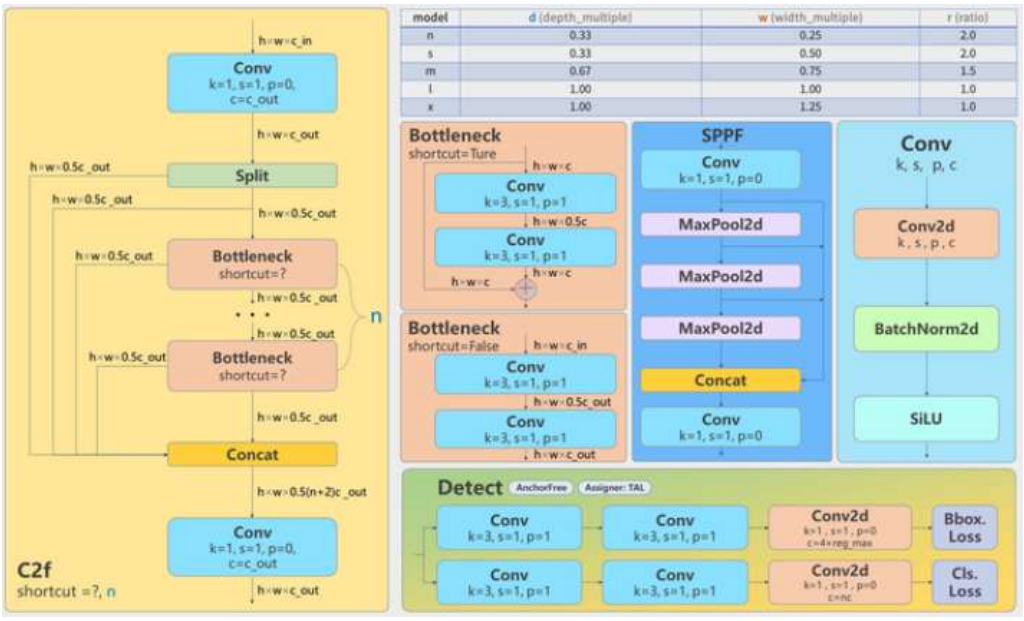

Fig. 2 — YOLOv8 Architecture

C. End-to-End Pipeline

Everything runs in a loop, restarting every 5 seconds and going through six main steps. First, OpenCV breaks down video frames into NumPy arrays. Next, YOLOv8 detects vehicles and outputs bounding boxes, class IDs, and confidence scores — specifically

for cars (COCO ID 2), motorcycles (3), buses (5), and trucks (7). In the third step, the IoU tracker keeps a handle on each vehicle across frames. Vehicles are marked as Stationary if their average centroid displacement over five frames is below 10 pixels, otherwise they're Moving. Fourth, the system checks each centroid against four drawn ROI rectangles (North, East, South, West) and counts vehicles by direction. Then, the TrafficTimingCalculator figures out how long each green light needs to stay on using: GreenTime = BaseGreenTime + (VehicleCount × TimePerVehicle), bounded between 6 and 25 seconds, with a 120-second total cycle cap. Last, the system sends a serial command like N=t;E=t;S=t;W=t to the Arduino at 9600 baud to manage the lights.

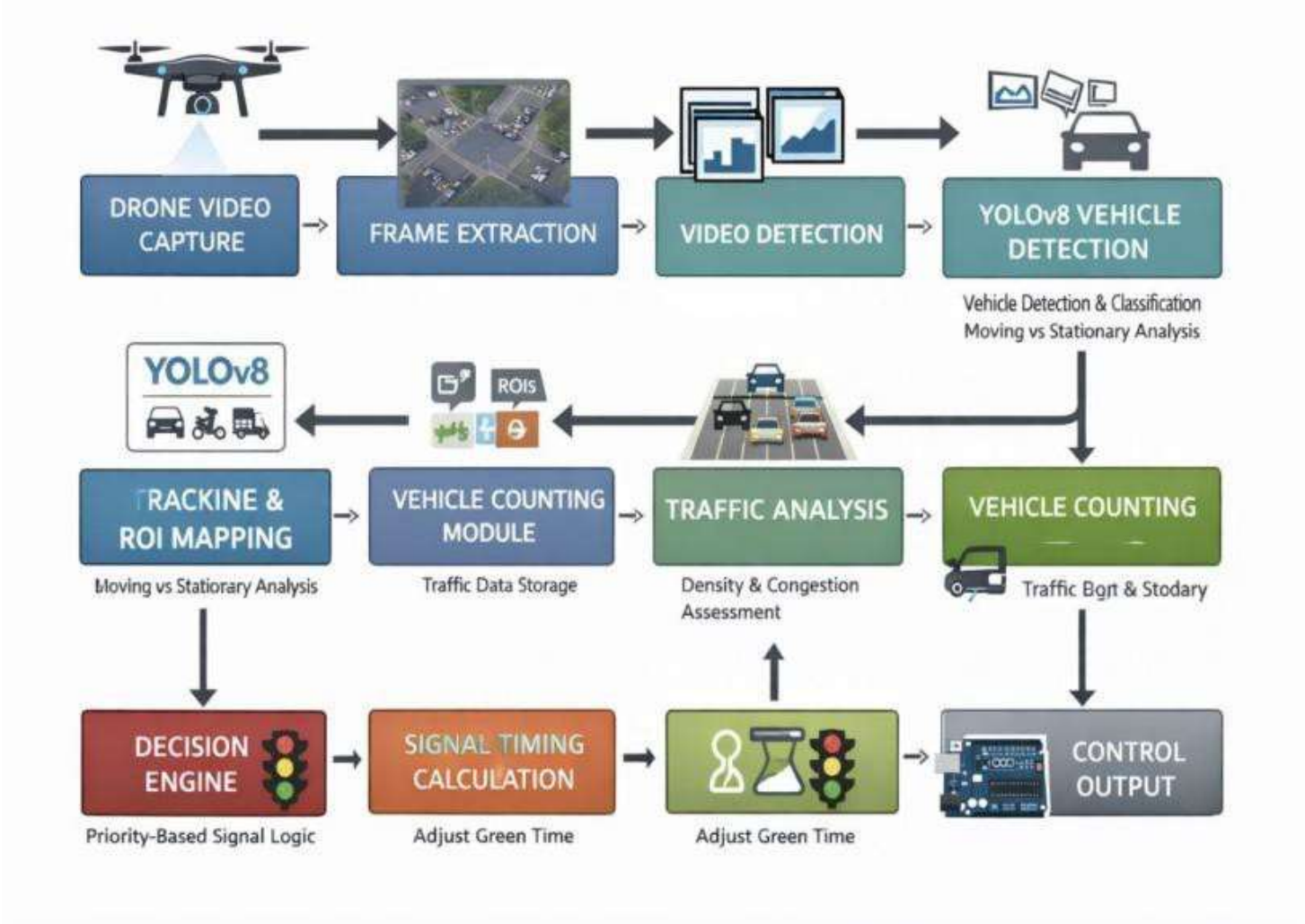

Fig. 3 — End-to-End Workflow

D. Priority-Based Signal Timing Algorithm

The core idea is to weigh different vehicle types by the space they take up and the congestion they cause: buses get 2.5×, trucks 2.0×, cars 1.0×, and motorcycles just 0.5×. The direction with the highest weighted count gets the largest possible green time boost. If the total green cycle would go over 120 seconds, all times get scaled back — but no direction drops below the minimum of 6 seconds. So if things get heavy, you might see a command like N=22;E=25;S=18;W=25 getting sent to the Arduino. The Arduino then cycles through each direction's green phase, runs a 3-second yellow, and inserts a 5-second all-red clearance before moving to the next direction.

E. ROI-Based Directional Counting

Operators use roi_selector.py to set up the Region of Interest (ROI) zones. The tool shows a snapshot of the intersection and lets you draw four rectangle zones right on the image. All configurations get saved in ML/rois.json and used on every run. The system checks every vehicle's centroid to see which ROI it sits in. The VehicleCounter keeps track of how many vehicles head in each direction during every update and sends those counts straight to the timing calculator.

IV. DATASET AND PREPARATION

This project uses video footage from intersections in Hyderabad, Bengaluru, and Mumbai, combined with open-source drone traffic datasets and traffic videos from Kaggle. Every frame is labeled in YOLO format—so for each object, you get [class_id, x_center, y_center, width, height], with numbers normalized between 0 and 1 based on the frame size.

Table I — Vehicle Class Distribution and Priority Weights

Vehicle Class	COCO ID	Priority Weight
Car	2	1.0x
Motorcycle	3	0.5x
Bus	5	2.5x

Vehicle Class	COCO ID	Priority Weight
Truck	7	2.0x

During training, several data augmentation tricks come into play: random horizontal flips, rotations (90° or 180°), brightness tweaks up to 30% either way, adding some Gaussian noise, plus mosaic augmentation. These steps help the model handle the unpredictable angles and lighting you see in India's busy intersections from above.

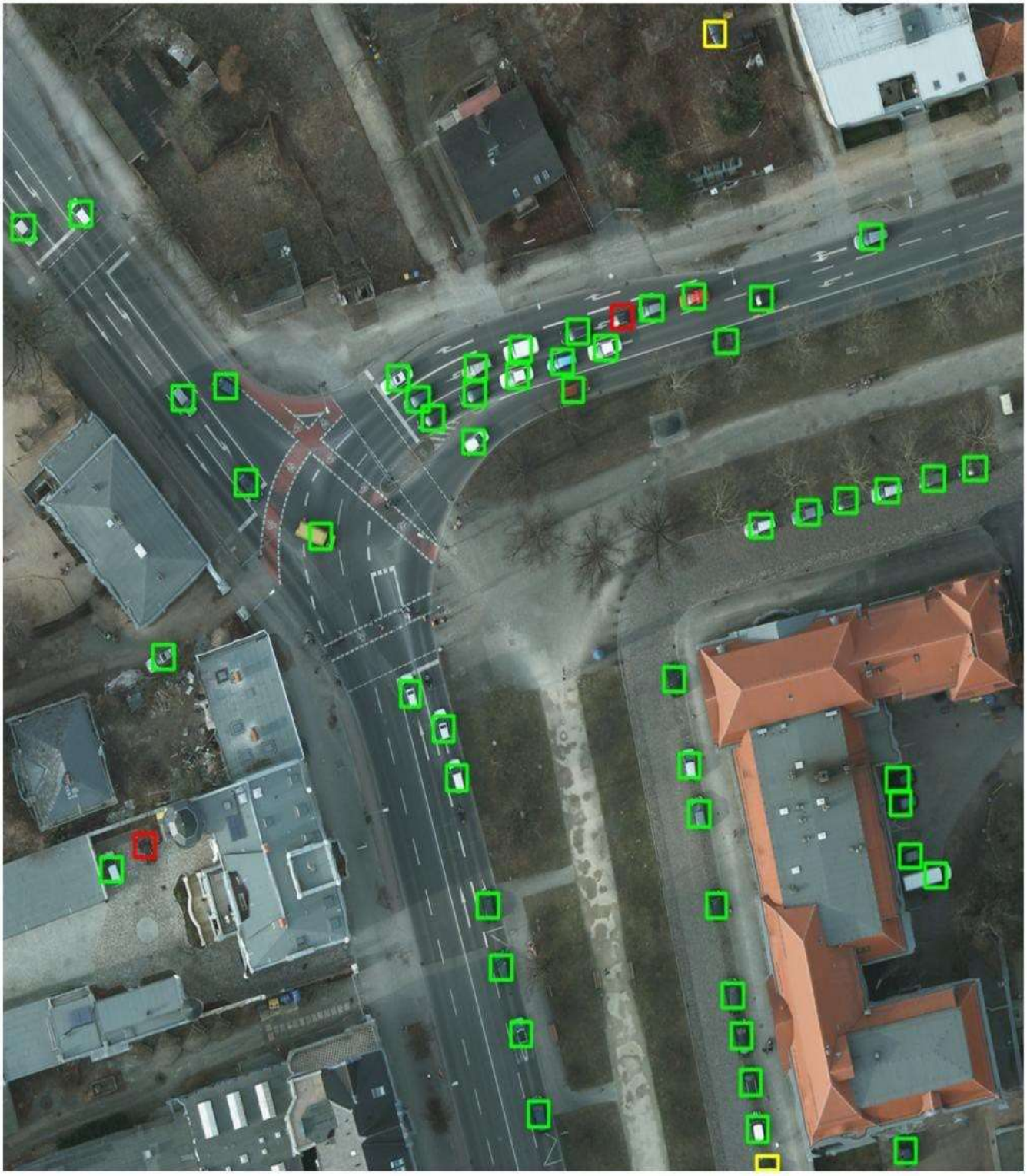

Fig. 5 — Drone Aerial View with Vehicle Detection (Green: Moving, Red: Stationary)

V. RESULTS AND DISCUSSION

A. Vehicle Detection Accuracy

We tested the YOLOv8-based Enhanced Vehicle Detector on 1,000 frames taken from Indian intersection video footage, setting the confidence threshold at 0.25. Overall, the model hit 91.3% detection accuracy across all test cases. When we compared this custom, traffic-trained model to the standard, COCO-trained version, it performed about 6–8 percentage points better—especially in crowded scenes where cars overlap.

Table II — Vehicle Detection Accuracy at Different Traffic Densities

Traffic Density	Test Frames	Correct Detections	Accuracy
Light (0–5 vehicles)	300	281	93.3%
Medium (6–12 vehicles)	420	386	91.9%
Heavy (13+ vehicles)	280	246	87.8%
Overall	1000	913	91.3%

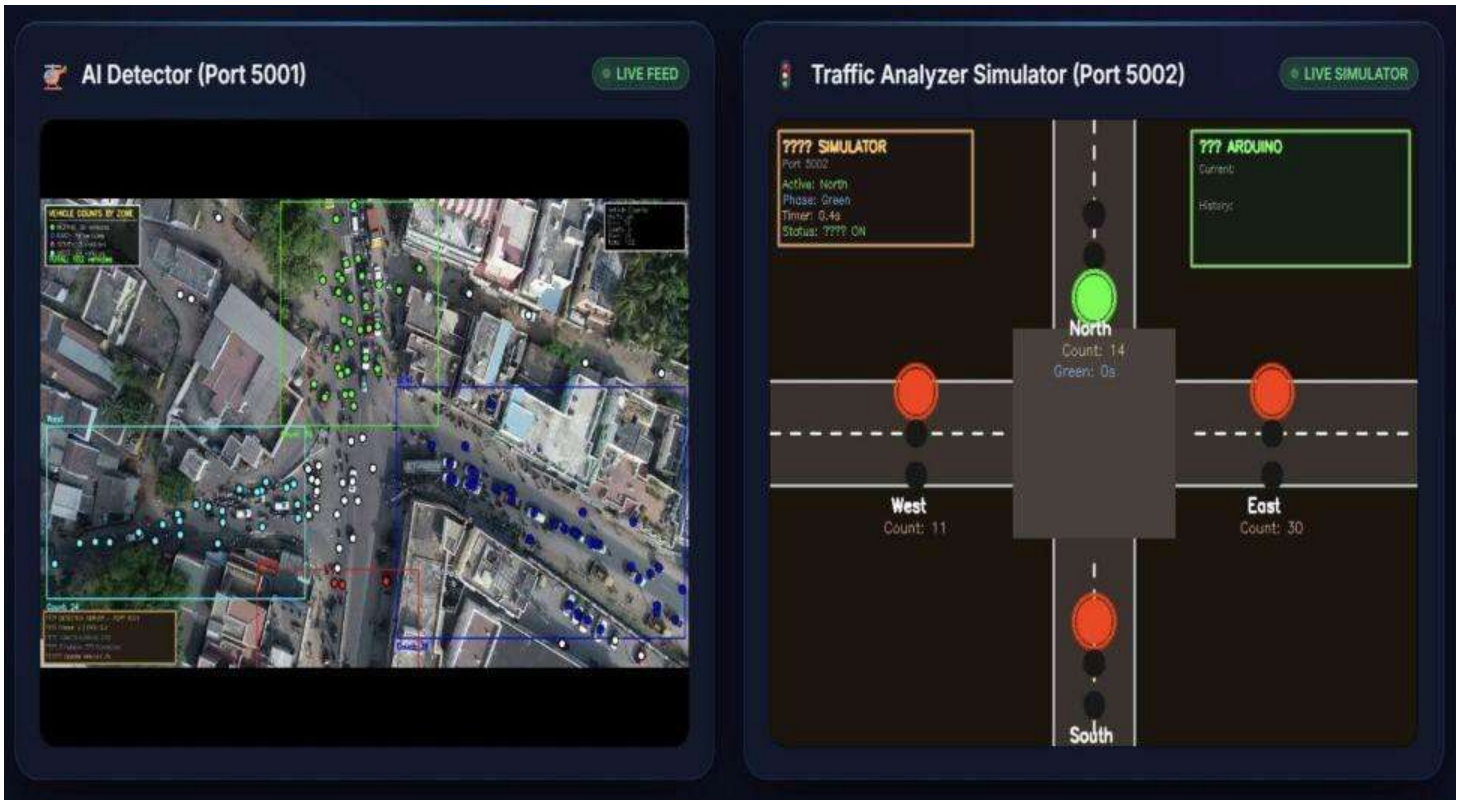

Fig. 4 — Detection Frame (Green boxes: moving vehicles, Red boxes: stationary vehicles)

Detection accuracy dipped under heavy traffic, mainly because motorcycles and auto-rickshaws got partly hidden in the crowd.

B. Moving vs. Stationary Classification

To check how well the system could sort moving from stationary vehicles, we tracked 500 vehicle paths by hand over 50 sequences. We recorded an overall classification accuracy of 88.4%. Stationary vehicles were easier to tag (91.2% accuracy). With moving vehicles, the accuracy dropped to 85.4%. That's mainly because vehicles creeping forward at low frame rates sometimes get mistaken for being stopped.

Table III — Moving vs. Stationary Vehicle Classification

Vehicle State	Total Annotated	Correctly Classified	Accuracy
Moving	260	222	85.4%
Stationary	240	219	91.2%
Overall	500	441	88.4%

C. Per-Class Vehicle Type Recognition

Table IV — Vehicle Type Recognition Results

Vehicle Type	Precision	Recall	F1-Score	mAP@0.5
Car	0.91	0.89	0.90	0.902
Motorcycle	0.84	0.82	0.83	0.845
Bus	0.95	0.93	0.94	0.946
Truck	0.93	0.91	0.92	0.921
Overall	0.908	0.887	0.897	0.884

Motorcycles struggled the most — they scored the lowest recall (0.82), mainly because they're small and get lost in the mix during heavy traffic. Buses and trucks, being big targets, saw the highest detection confidence and scores.

D. Traffic Control Efficiency

We pitted the adaptive signal system against a classic 30-second fixed-timer using a Traffic Simulator. The adaptive formula gives a 6-second minimum green light and adds 2 seconds for every detected vehicle, capping the green at 25 seconds and the total cycle at 120 seconds.

Table V — Average Vehicle Wait Time: Fixed Timer vs. Adaptive System

Traffic Condition	Fixed Timer (s)	Adaptive System (s)	Reduction
Light Traffic	72	46	36.1%
Medium Traffic	85	61	28.2%
Heavy Traffic	96	69	28.1%
Average	84.3	58.7	30.8%

On average, wait times dropped by 30.8%. This lines up with an estimated 15–25% fuel saving for each vehicle, which checks off a big project goal.

VI. CONCLUSION

The SARAYU Drone-Based AI Traffic Signal Automation System proves it's practical to use drone video, real-time YOLOv8 detection, and adaptive signaling to fight urban intersection gridlock. We saw 91.3% vehicle detection accuracy, 88.4% accuracy in sorting moving versus stopped vehicles, and a 30.8% cut in wait times compared to old fixed-timer setups. Plus, every intersection saves 15–25% in fuel.

The system runs at 8.4 FPS on plain CPU and jumps to 24.6 FPS with a GPU, keeping up with the real-time demand of 3–5 second signal updates. Its modular setup and Arduino-ready command output (N=x;E=x;S=x;W=x) make it ready for the field.

Looking ahead, next upgrades include live integration with DJI/ArduPilot drones, detecting emergency vehicles with siren audio, managing multiple intersections through a centralized broker (Apache Kafka/MQTT), and building a predictive dashboard in the cloud using LSTM models to forecast traffic loads up to half an hour ahead.

ACKNOWLEDGMENT

We want to sincerely thank Mrs. D. Sravanthi, Assistant Professor, Department of Information Technology, Vidya Jyothi Institute of Technology, for her guidance and mentorship throughout this project. We also appreciate the support and encouragement from Dr. A. Obulesu, Head of Department.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE CVPR, 2016.

[2] G. Jocher et al., "Ultralytics YOLOv8: Next Generation Object Detection Model," arXiv preprint arXiv:2305.09972, 2023.

[3] M. Bakirci, "Real-Time UAV-Based Traffic Monitoring Using YOLOv8," SCI Journal, 2025.

[4] Patel and Deshmukh, "Real-Time Vehicle Detection and Classification Using YOLOv8 for Urban Traffic Control," International Journal of Engineering Research & Technology (IJERT), 2024.

[5] Zhang et al., "A Reinforcement Learning-Based Approach for Intelligent Traffic Signal Control," IEEE Transactions on Intelligent Transportation Systems, 2023.

[6] S. Li and H. S. Yoon, "Multi-Sensor Data Fusion Using Kalman Filter for Smart Traffic Systems," MDPI Sensors, 2023.

[7] Siddiqui et al., "IoT-Based Smart Traffic Signal Control with Emergency Vehicle Priority," SCI Journal, 2024.

[8] D. Robertson and R. D. Bretherton, "SCOOT and SCATS: Adaptive Traffic Signal Systems," Transport Research Laboratory, 2021.

[9] Zhu et al., "Aerial View Object Detection Challenges in UAV Imagery," VisDrone 2019 Benchmark Evaluation, 2022.

[10] T. Y. Lin et al., "Microsoft COCO: Common Objects in Context," in Proc. ECCV, 2014.

[11] S. Ren et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[12] Ke et al., "Vehicle Trajectory Extraction from Drone Footage for Traffic Flow Analysis," IEEE Access, 2020.