Adaptive Traffic Signal Timing: Leveraging YOLOv10 and Computer Vision for Real-Time Optimization

Suhas Papanashi
Dept. of Electronics and Telecomm.
RV College of Engineering,
Bengaluru
suhaspapanashi.et22@rvce.edu.in

Manya Chadaga
Dept. of Computer Science & Eng.
RV College of Engineering,
Bengaluru
manyachadaga.cs22@rvce.edu.in

Kshithi R
Dept. of Computer Science & Eng.
RV College of Engineering,
Bengaluru
kshithir.cs22@rvce.edu.in

Santosh S Huddar
Dept. of Electronics and Telecomm.
RV College of Engineering,
Bengaluru
santoshsh.et22@rvce.edu.in

Dr. K Sreelakshmi
Dept. of Electronics and Telecomm.
RV College of Engineering,
Bengaluru
sreelakshmik@rvce.edu.in

Dr. Ramakanth Kumar P
Dept. of Computer Science & Eng.
RV College of Engineering,
Bengaluru
ramakanthkp@rvce.edu.in

***Abstract*—Traffic congestion poses a significant challenge in modern urban environments, particularly in densely populated cities. Traditional traffic signals, which operate on fixed timings, often require manual adjustments to respond to varying traffic densities, leading to inefficient traffic flow. This issue arises because fixed intervals do not account for the dynamic nature of traffic patterns throughout the day. While developing intelligent systems that adjust signal durations dynamically offers a potential solution, such systems involve substantial financial investment and extended implementation timelines. This paper proposes a method to optimize green light durations by analyzing footage collected from a traffic junction in Bengaluru. This analysis can be performed using existing CCTV footage or manually gathered data. Even limited sample sizes can provide valuable insights for preliminary adjustments. The approach utilizes YOLOv10, a state-of-the-art computer vision algorithm, to extract relevant traffic data. The data is processed by an algorithm designed to reallocate green light durations, which are then compared with the original timings. An experimental setup involved recording one-minute videos over three days at the selected junction and applying the proposed algorithm to assess and refine signal durations. The results demonstrate the feasibility of using such algorithms to improve traffic signal management effectively.**

Keywords—Computer Vision, Data-Driven Traffic Management, Urban Mobility, Smart Cities, Video analytics Introduction (Heading 1)

I. Introduction

Traffic congestion remains a major issue in today's world. With around 1.3 billion vehicles globally, a significant amount of time is wasted idling at intersections, causing delays, higher fuel consumption, and increased emissions, which worsen urban pollution. This issue is particularly severe in India's densely populated cities, where outdated traffic systems persist. Many signals rely on fixed timers, requiring manual adjustments. This becomes critical during accidents or congestion, where delayed reporting hinders prompt response. Technological advancements provide solutions. This paper uses the YOLOv10 model for zone-based vehicle counting in traffic footage. YOLOv10's capabilities enable accurate, efficient data analysis. The data informs an algorithm that optimizes green light durations, improving current strategies and setting the stage for real-time intelligent traffic systems.

II. Related Work

Previous research has explored various technologies to address traffic management challenges at junctions. Many studies focus on advanced computer vision algorithms for traffic flow management. Algorithms like YOLOv2 [1], YOLOv7 [2], and Faster R-CNN [3] have been used for vehicle counting, while MATLAB's digital image processing tools [4] analyze traffic patterns. Wireless technologies such as RFID [5] and IR sensors [6] have also been used to detect vehicle densities and manage traffic accordingly. A recent implementation, Vehicle Actuated Control (VAC) [13], uses magnetic loops embedded in roads to detect zero-vehicle counts at specific lanes, adjusting traffic signal timings to optimize flow. Additionally, other researchers have developed algorithms that adjust traffic signal timings based on real-time data, using techniques like Convolutional Neural Networks (CNNs) [7] and Reinforcement Learning [9]. Open-source software, such as SUMO, is also used to simulate and test these algorithms [10] [11]. This paper introduces a novel approach, developing an algorithm that predicts optimized traffic light timings using recorded video data. This method generates improved fixed-time signal combinations, useful in areas lacking advanced infrastructure. The data from this algorithm can also train reinforcement learning or machine learning models, advancing traffic management.

III. Using Computer Vision

A. Object Detection and Tracking Model

The YOLO series is renowned for its accuracy and efficiency in computer vision applications, with YOLOv10 [12] being the latest iteration. YOLOv10 offers both object detection and tracking capabilities.

Trained on the MS COCO (Common Objects in Context) dataset, YOLOv10 learns to predict annotations by adjusting its internal parameters to minimize errors in bounding box positions, object classifications, and objectness scores. The

model processes an image through its network to generate predictions for bounding boxes, class labels, and objectness scores. For object tracking, YOLOv10 uses algorithms such as Kalman filtering to link detections across video frames, allowing it to track objects as they move, even if they briefly disappear or overlap with other objects. The major improvement of the v10 model over YOLOv9 are:

- YOLOv10 uses a new backbone network, EfficientNet-B4, which enhances feature extraction.
- YOLOv10 refines the traditional NMS technique to better handle overlapping detections, by filtering out duplicate bounding boxes, keeping the one with the highest confidence score and removing others that overlap.
- YOLOv10 introduces a consistent matching metric to improve object tracking across frames. This metric helps maintain consistent identities for objects over time, even through occlusions and overlapping, as shown in Figure 1.

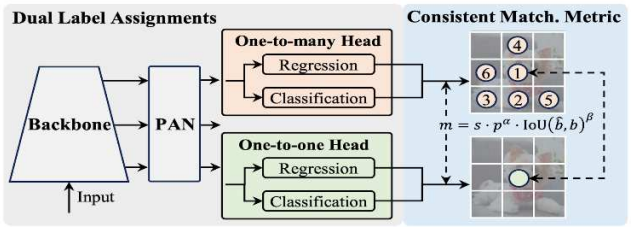


Fig 1: YOLOv10 Architecture [12]

YOLOv10 employs a multi-layered architecture [12] optimized for both speed and accuracy in object detection. The model begins with convolutional layers that apply filters to the input image to detect fundamental features such as edges and textures. Batch normalization follows to stabilize these features, ensuring consistent and efficient training. Activation layers, usually Leaky ReLU, introduce non-linearity, enabling the model to learn complex patterns. Down sampling layers, including max-pooling and strided convolutions, reduce the spatial dimensions of feature maps, improving computational efficiency while capturing higher-level abstractions.

The backbone processes these features, which are then refined by the neck layer that integrates information across different scales. The Path Aggregation Network (PAN) layers combine low-level detailed features with high-level abstract ones to enhance detection accuracy. YOLOv10 employs a dual-head system: during training, one head generates multiple predictions per object, while during inference, the other head provides the most accurate prediction, eliminating the need for additional post-processing steps like Non-Maximum Suppression. The final output layers predict bounding boxes, class probabilities, and confidence scores, concluding the detection process.

YOLOv10 offers 6 different models [12] :

- YOLOv10-N: Nano version for extremely resource-constrained environments.
- YOLOv10-S: Small version balancing speed and accuracy.
- YOLOv10-M: Medium version for general-purpose use.
- YOLOv10-B: Balanced version with increased width for higher accuracy.
- YOLOv10-L: Large version for higher accuracy at the cost of increased computational resources.
- YOLOv10-X: Extra-large version for maximum accuracy and performance.

All these models have improved latency, performance and mAP values over the previous versions.

B. Experimental Setup

1) Data Preparation (Videos): A significant challenge encountered during manual footage collection is ensuring comprehensive coverage of all lanes at an intersection. Achieving this coverage is particularly difficult, as it typically requires an elevated vantage point, which can be facilitated either by installing CCTVs at the junction or deploying drones. An alternative solution involves utilizing multiple cameras to record simultaneously and then merging the footage. For this purpose, a suitable 4-way junction near RV College of Engineering in Bengaluru, India, was identified, where access was available to the top floor of a corner building. Two videos were recorded simultaneously, each capturing two lanes of the junction, using iPhone 14 cameras at 30 fps with a resolution of 1920x1080. To integrate the footage, a Python script utilizing OpenCV was developed to stack the two videos. The resulting combined video maintained a resolution of 1920x2160 to prevent any loss of detail. Over several days, two distinct instances of traffic at the junction were recorded, resulting in a total of six combined videos.

2) Performing Object Detection and Tracking on the Data: The Ultralytics library offers a region-counting example, which was used to develop a script to run the YOLOv10 model on the Traffic junction videos, with some additional features. First, To deploy 4 zones, (one to count at each of the lanes), a list holding 4 dictionaries, each having 7 key attributes is initialized at the beginning of the script, as described in Table 1.

Table 1: Attribute features of every zone

Attribute	Description
name	Name of the region
polygon coordinates	Define the region in the video where the zone is supposed to be placed
counts	Count of the current number of vehicle in the zone

dragging	a Boolean to check if user is using mouse currently to move the zone
region color	color of the region (BGR value)
text color	color of the number displaying the count in every zone (BGR value)
Detected ids	a set to to keep track of the unique IDs of vehicles that have been detected within the specific region

To determine zone coordinates, the process starts by using FFmpeg to capture a screenshot, ensuring consistent resolution. A free online tool is then used to draw polygon regions and extract coordinates, which are unique to each video due to recording angle variations. These coordinates must be recalculated to cover the required areas for all four lanes. A `run()` function is defined for frame-by-frame processing. It takes key arguments such as `weights` (model file), `source` (video path), and `classes` (object classes). YOLO runs on either CPU or GPU. Here, an M1 MacBook Air with an ARM-based processor and 8-core GPU was used, with PyTorch MPS providing speeds 20 times faster than the CPU alone. YOLOv10 is trained on the COCO dataset with 80 object classes. For vehicle detection, specific classes like Bicycle, Car, and Truck are chosen. OpenCV creates a video window, and within a loop, YOLO performs detection and tracking for each frame, incrementing the respective count when the bounding box center falls within a defined zone.

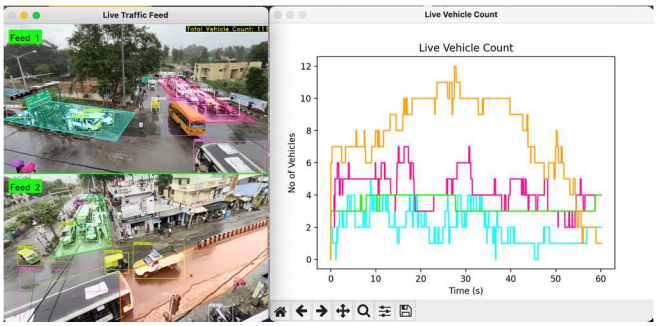


Fig 2: Windows displaying the vehicle counting process on the videos

3) Additional Features: In order to extract and store data from the video efficiently, and to improve functionality of the script, the following additional features were integrated:

- save_to_csv() : This function creates a csv file in the same directory as where the output video gets saved and prints the values of the current time (No of frames // 30) and the number of vehicles in each of the zones, for every frame of the video.

- Graph Plot : Using the matplotlib library, a graph window is created, and it runs on the side with the video window, and simultaneously plots 4 graphs, each for one of the zones, for better visualization. The final graph at the end of the video is also saved in the same directory.

- Mouse Callback : In case there is slight change in the angle of the video, and the zones get misaligned, they can be dragged back into place, using the mouse. This functionality is created using OpenCV.

- argparse : Using this library, we can pass several arguments when executing the file, in the terminal, without having to search for and change them every time in the script. We can easily select the video to be run, whether to save the results, bounding box thickness, weights and class numbers.

When the script is executed on a selected video, as illustrated in Figure 2, two windows are displayed: one for object detection and counting, and the other for simultaneous graph plots.

C. Results and Data:

The collected video samples indicated that the average minimum duration for which a lane was set to green was approximately 30 seconds. It was occasionally observed that a lane remained green despite having no vehicles, causing other lanes to experience delays. Additionally, in some of the videos, there was rain, but it did not impact the accuracy of vehicle detection and counting.

The YOLOv10-L model was utilized to ensure precise detections. Initially, when the script was executed on the CPU, the average processing time per frame was 2000 ms. Upon transitioning to the GPU, the processing time decreased to 90 ms per frame, resulting in a 20-fold improvement in speed, as shown in Figure 3.

Run on CPU (default)

```
0: 640x576 15 cars, 6 motorcycles, 1 bus, 8 trucks, 1715.1ms
Speed: 2.5ms preprocess, 1715.1ms inference, 0.3ms postproces

0: 640x576 14 cars, 8 motorcycles, 1 bus, 8 trucks, 2265.5ms
Speed: 2.4ms preprocess, 2265.5ms inference, 0.3ms postproces

0: 640x576 16 cars, 7 motorcycles, 1 bus, 7 trucks, 1884.4ms
Speed: 2.4ms preprocess, 1884.4ms inference, 0.4ms postproces
```

GPU Utilized (M1)

```
0: 640x576 12 cars, 4 motorcycles, 2 buss, 7 trucks, 95.1ms
Speed: 3.0ms preprocess, 95.1ms inference, 21.1ms postproces

0: 640x576 15 cars, 4 motorcycles, 2 buss, 6 trucks, 90.0ms
Speed: 3.2ms preprocess, 90.0ms inference, 44.6ms postproces

0: 640x576 14 cars, 4 motorcycles, 2 buss, 7 trucks, 93.9ms
Speed: 3.0ms preprocess, 93.9ms inference, 16.8ms postproces
```

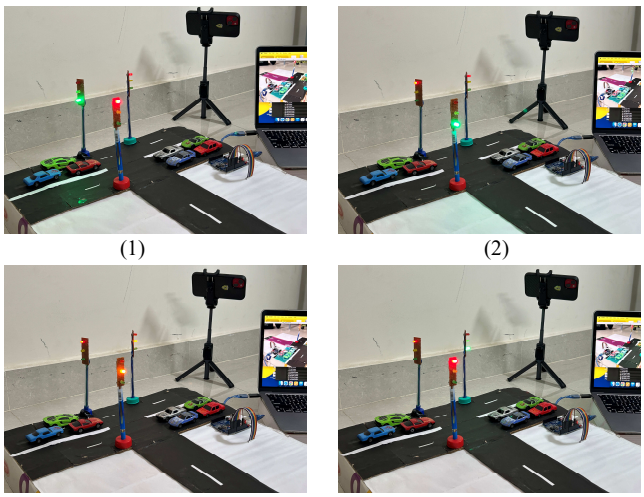
Fig 3: Terminal outputs during runtime when same video was run on a) CPU b) GPU

IV. Arduino Prototype

The primary issue at traffic junctions is when a lane remains green despite having no vehicles, while other lanes with waiting vehicles experience delays, due to outdated fixed-timing methods. This problem can be addressed by implementing a system that continuously monitors vehicle

counts across all lanes and adjusts signal timings if a lane remains empty for more than 3 seconds. To demonstrate this solution, we developed a prototype using an Arduino Uno microcontroller, 3V LEDs, and a phone camera to simulate a T-shaped, three-lane junction. Toy cars were placed on two lanes, leaving the third lane empty. We modified our existing Python script (described in Section III) for this setup. The live camera feed was captured using OpenCV's webcam feature, while the Arduino was programmed with a fixed signal timing of 20 seconds per lane, as shown in Figure 4.

The Python script, utilizing the pyserial library, communicated with the Arduino to skip the wait time if a lane's vehicle count was zero for more than 3 seconds while that lane was green. We achieved a response time of less than 3 seconds, demonstrating that this approach could be effectively implemented in real traffic junctions. A similar concept is being explored through VAC [13], but leveraging computer vision for vehicle counting offers the potential for greater accuracy.



(1) (2) (3) (4)

Fig 4: Arduino Prototype of a 3-Lane Junction, depicting four situations: (1) Lane 1 is green for entire 20sec, (2) Lane 2 turns green (3) Lane 2 immediately skips wait time due to no vehicle presence (4) Cycle continues, and Lane 3 shows green for 20sec

Arduino pins, as shown in Figure 5, are categorized into digital (D0-D13) and analog (A0-A5) types. Digital pins can perform both input and output operations, with specific pins (D3, D5, D6, D9, D10, D11) also supporting Pulse Width Modulation (PWM). Analog pins are used for Analog-to-Digital Conversion (ADC) to measure varying voltage levels. The board also features power pins (3.3V, 5V, GND) and specialized communication pins for I2C (A4, A5) and SPI (D10-D13) protocols.

For this experiment, nine digital pins were used to control each of the LEDs, and all grounds were connected to a common ground terminal. Additionally, an external battery power source was required to ensure that all LEDs were brightly illuminated.

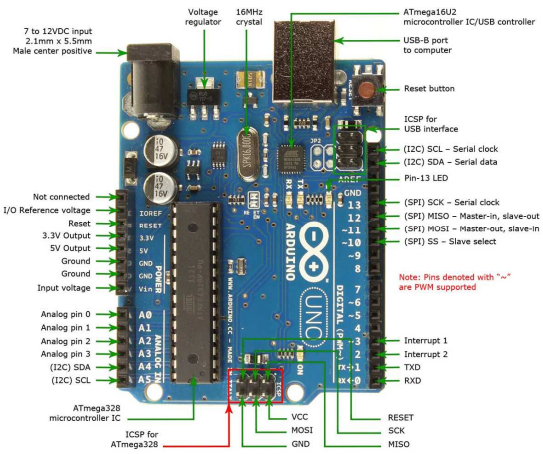


Fig 5: Arduino Pin Diagram

V. Proposed Algorithm

After encountering the zero-vehicle issue, the next significant challenge is to create an algorithm that dynamically adjusts the timings of every lane based on the number of vehicles. This is particularly crucial in dense urban areas where long traffic jams are common. While some researchers have explored Reinforcement Learning and Machine Learning techniques, their implementation faces challenges such as lack of computational resources and data. This paper takes a different approach to this problem: performing an analysis on recorded video data and estimating what could have been the optimal combination of signal timings for that video, compared to the actual combination that operated during the recording. Algorithm 1 defines this concept. The variable T_min sets the minimum allocation duration. A smaller value implies faster switching at the signal; therefore, using at least 8 seconds would be ideal. The variable R represents the rate at which vehicles move out of or into the counting zone once the green signal is applied. This can be approximated after observing the video footage. An example value would be 1 vehicle/2sec, i.e., R= 0.5. The algorithm takes as input the vehicle count data for every zone at each second of the video. The input format consists of a timestamp column followed by columns representing the count at each zone. Based on this data, the algorithm assigns optimal green signal times for the entire duration of the video. The process works as follows: For each second, the algorithm first checks if any lane currently has a green light. If so, it continues that green light. If not, it selects a new lane for the green light based on current vehicle counts and fairness constraints. A rotation counter for each lane keeps track of how many times a lane has received the green light. If a lane has had the green light for the maximum allowed cycles, it won't be selected again until other lanes have had a chance. This ensures a fairer rotation. The algorithm prioritizes lanes that haven't reached the maximum cycles. Among these eligible lanes, the one with the highest vehicle count is selected. If all lanes have been selected equally, the counters reset, and the lane with the highest count gets the green light. Each selected lane is guaranteed a minimum green light duration of T_min seconds. After each green light assignment, the algorithm adjusts the vehicle counts based on the R value, simulating traffic flow. The count for the lane

with the green light decreases, while the counts for other lanes increase. This algorithm effectively reallocates green signal times and adjusts the original values based on the chosen parameters. It theoretically recalculates what could have been a better combination of green signal timings that would have moved traffic at the junction more efficiently, with every vehicle experiencing the least possible wait time.

ALGORITHM 1: TRAFFIC SIGNAL REASSIGNMENT

```
Input: df, T_min, R, max_cycles
Initialize:
green_intervals = zeros(num_seconds, num_lanes)
green_remaining = [0] * num_lanes
rotation_counter = [0] * num_lanes
for t in range(num_seconds) do
    if any(green_remaining) then
        max_lane = argmax(green_remaining)
        green_intervals[t, max_lane] = 1
        green_remaining[max_lane] -= 1
    else
        counts = df[t, 1:]
        eligible = [i for i, c in
        enumerate(rotation_counter) if c <max_cycles]
        if eligible is not empty then
            max_lane = argmax([counts[i] for i in
            eligible])
        else
            rotation_counter = [0] * num_lanes
            max_lane = argmax(counts)
        green_intervals[t, max_lane] = 1
        green_remaining[max_lane] = T_min
        rotation_counter[max_lane] += 1
        df[t+1, max_lane+1] = max(0, df[t+1,
        max_lane+1] - R)
        for lane in range(num_lanes) do
            if lane != max_lane then
                df[t+1, lane+1] += R
            end
        end
    end
end
Output:
generate_green_light_report(green_intervals)
```

Algorithm 1: Traffic Signal Optimization

A Python script was developed based on the algorithm, using the data outlined in Section III. Actual green signal timings were manually recorded from the video for comparison with the algorithm-generated timings. The parameter R was set to 0.5, with Tmin at 8 seconds. In most cases, the algorithm produced more efficient green signal timing combinations than the original, as shown in Table 2. Plotting the final vehicle counts against time, compared to the initial counts, demonstrates a clear reduction in variations and peak counts, as illustrated in Figure 6.

Table 2: Comparing the new and old signal times

Video	Original Times (in sec)	New calculated times (in sec)
1	0 - 23 L2 23 - 48 L3 48 - 60 L1	0 - 8 L4 8 - 16 L2 16 - 24 L4 24 - 32 L2 32 - 40 L3 40 - 48 L1 48 - 60 L3
2	0 - 20 L4 20 - 60 L2	0 - 8 L4 8 - 16 L3 16 - 24 L1 24 - 32 L3 32 - 40 L4 40 - 48 L2 48 - 60 L3
3	0 - 11 L2 11 - 43 L3 43 - 60 L4	0 - 8 L4 8 - 16 L3 16 - 24 L4 24 - 32 L1 32 - 40 L3 40 - 48 L2 48 - 60 L1
4	0 - 43 L1 43 - 60 L4	0 - 8 L4 8 - 16 L2 16 - 24 L4 24 - 32 L2 32 - 40 L1 40 - 48 L3 48 - 60 L1
5	0 - 12 L4 12 - 60 L2	0 - 8 L3 8 - 24 L4 24 - 32 L3 32 - 48 L2 48 - 60 L1
6	0 - 10 L3 10 - 60 L1	0 - 16 L4 16 - 24 L3 24 - 32 L2 32 - 40 L3 40- 48 L2 48- 61 L1

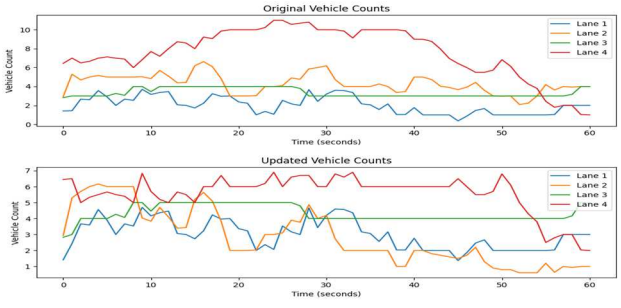


Fig 6: Comparing original and re-allocated times to observe the wait-time minimization

VI. Conclusion

This paper documents the efforts undertaken to advance traffic management at junctions through innovative methods. Initially, computer vision technology was employed to continuously and accurately count vehicles in each lane from the collected video data. Following a review of existing approaches in this domain, a novel algorithm was developed to analyze recorded footage and propose optimized green signal timings for the given video. The primary objective of this approach is to offer a cost-effective solution for enhancing traffic management. The experimental setup involved capturing one-minute videos of a traffic junction using multiple phone cameras to achieve comprehensive coverage from all angles. The recorded footage was then combined, and object detection and tracking algorithms were applied to extract the necessary data. Subsequently, this data was processed, through a designed algorithm to determine improved green signal timings. Future work involves collecting additional data at consistent times on specific days of the week to identify patterns and further refine the green signal timings. This approach could contribute to the development of an intelligent traffic management system capable of dynamically adjusting signal timings based on real-time data, aligning with the long-term goal of optimizing traffic flow.

Acknowledgment

This work was carried out at the Centre of Excellence in Computer Vision, RV College of Engineering, under the esteemed guidance and support of Dr. Hemavathy R, Associate Professor in the Department of Computer Science and Engineering, and Professor T. Shankar, Head of Projects at the Centre of Excellence in Computer Vision, RV College of Engineering.

References

[1] U. K. Sreekumar, R. Devaraj, Q. Li and K. Liu, "Real-Time Traffic Pattern Collection and Analysis Model for Intelligent Traffic Intersection," *2018 IEEE International Conference on Edge Computing (EDGE)*, San Francisco, CA, USA, 2018, pp. 140-143, doi: 10.1109/EDGE.2018.00028.

[2] A. P. Rangari, A. R. Chouthmol, C. Kadadas, P. Pal and S. Kumar Singh, "Deep Learning based smart traffic light system using Image Processing with YOLO v7," *2022 4th International Conference on Circuits, Control, Communication and Computing (I4C)*, Bangalore, India, 2022, pp. 129-132, doi: 10.1109/I4C57141.2022.10057696.

[3] M. T. Ubaid, T. Saba, H. U. Draz, A. Rehman, M. U. Ghani khan and H. Kolivand, "Intelligent Traffic Signal Automation Based on Computer Vision Techniques Using Deep Learning," in *IT Professional*, vol. 24, no. 1, pp. 27-33, 1 Jan.-Feb. 2022, doi: 10.1109/MITP.2021.3121804.

[4] V. Bhardwaj, Y. Rasamsetti and V. Valsan, "Image Processing Based Smart Traffic Control System for Smart City," *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2021, pp. 1-6, doi: 10.1109/ICCCNT51525.2021.9579787.

[5] A. Dutta et al., "Intelligent Traffic Control System: Towards Smart City," 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2019, pp. 1124-1129, doi: 10.1109/IEMCON.2019.8936188.

[6] A. Firdous, Indu and V. Niranjan, "Smart Density Based Traffic Light System," 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2020, pp. 497-500, doi: 10.1109/ICRITO48877.2020.9197940.

[7] A. K. Ikiriwatte, D. D. R. Perera, S. M. M. C. Samarakoon, D. M. W. C. B. Dissanayake and P. L. Rupasignhe, "Traffic Density Estimation and Traffic Control using Convolutional Neural Network," *2019 International Conference on Advancements in Computing (ICAC)*, Malabe, Sri Lanka, 2019, pp. 323-328, doi: 10.1109/ICAC49085.2019.9103369.

[8] D. Prasad, K. Kapadni, A. Gadpal, M. Visave and K. Sultanpure, "HOG, LBP and SVM based Traffic Density Estimation at Intersection," *2019 IEEE Pune Section International Conference (PuneCon)*, Pune, India, 2019, pp. 1-5, doi: 10.1109/PuneCon46936.2019.9105731.

[9] A. C. Egea, S. Howell, M. Knutins and C. Connaughton, "Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations," *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, ON, Canada, 2020, pp. 965-972, doi: 10.1109/SMC42975.2020.9283498.

[10] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 2018, pp. 2575-2582, doi: 10.1109/ITSC.2018.8569938.

[11] N. Kumar and S. S. Rahman, "Deep Reinforcement Learning with Vehicle Heterogeneity Based Traffic Light Control for Intelligent Transportation System," *2019 IEEE International Conference on Industrial Internet (ICII)*, Orlando, FL, USA, 2019, pp. 28-33, doi: 10.1109/ICII.2019.00016.

[12] Ao Wang, undefined., et al, "YOLOv10: Real-Time End-to-End Object Detection," 2024.

[13] N. Mishra, “Chandigarh: Futuristic vehicle detection system to cut wait times at traffic signals,” *Hindustan Times*, Jun. 24, 2024. [Online].