inventions

MDPI

Article

A Self-Adaptive Traffic Signal System Integrating Real-Time Vehicle Detection and License Plate Recognition for Enhanced Traffic Management

Manar Ashkanani, Alanoud AlAjmi, Aeshah Alhayyan, Zahraa Esmael, Mariam AlBedaiwi and Muhammad Nadeem *

College of Engineering and Technology, American University of the Middle East, Egaila 54200, Kuwait
* Correspondence: muhammad.nadeem@aum.edu.kw

Abstract: Traffic management systems play a crucial role in smart cities, especially because increasing urban populations lead to higher traffic volumes on roads. This results in increased congestion at intersections, causing delays and traffic violations. This paper proposes an adaptive traffic control and optimization system that dynamically adjusts signal timings in response to real-time traffic situations and volumes by applying machine learning algorithms to images captured through video surveillance cameras. This system is also able to capture the details of vehicles violating signals, which would be helpful for enforcing traffic rules. Benefiting from advancements in computer vision techniques, we deployed a novel real-time object detection model called YOLOv11 in order to detect vehicles and adjust the duration of green signals. Our system used Tesseract OCR for extracting license plate information, thus ensuring robust traffic monitoring and enforcement. A web-based real-time digital twin complemented the system by visualizing traffic volume and signal timings for the monitoring and optimization of traffic flow. Experimental results demonstrated that YOLOv11 achieved a better overall accuracy, namely 95.1%, and efficiency compared to previous models. The proposed solution reduces congestion and improves traffic flow across intersections while offering a scalable and cost-effective approach for smart traffic and lowering greenhouse gas emissions at the same time.

Keywords: smart cities; machine learning; adaptive traffic light control; optical character recognition; digital twin; intelligent transport systems

Academic Editors: Kwok Tai Chui and Brij B. Gupta

Received: 2 December 2024
Revised: 14 January 2025
Accepted: 22 January 2025
Published: 5 February 2025

Citation: Ashkanani, M.; AlAjmi, A.; Alhayyan, A.; Esmael, Z.; AlBedaiwi, M.; Nadeem, M.. A Self-Adaptive Traffic Signal System Integrating Real-Time Vehicle Detection and License Plate Recognition for Enhanced Traffic Management. *Inventions* **2025**, *10*, 14. https://doi.org/10.3390/inventions10010014

1. Introduction

Urbanization is on the rise across the globe, and it is expected that 70% of the world's inhabitants will settle in urban areas by 2050 [1]. This puts extra pressure on traffic infrastructure, which is unable to handle this inflow, leading to excessive congestion on the roads as more and more people use private cars instead of public transport. For example, Kuwait is a country where 83.72% of individuals commute by automobile, and the situation is expected to be exacerbated as the population is projected to reach 7 million people by 2030. The excessive number of vehicles leads to congestion and intersections are the main areas affected. Intersections are normally controlled using fixed-time controllers, where the duration of the green signal for each segment of the road remains the same, irrespective of the traffic density. This poor traffic light prioritization, especially under varied traffic density conditions, frequently turns intersections into hotspots of traffic congestion [2]. The length of time that vehicles must stop in order to move through an intersection increases considerably, resulting in excessive waiting times at intersections, as well as increased stress

and greenhouse emissions. The frequent occurrence of such unwanted delays may lead to rash driving and increase the chance of traffic rule violations. Another option is to alternate the phase timing in each cycle, as proposed in [3]. But this option suffers from the drawback of reduced sensitivity to traffic conditions due to its simplified phase timings scheme.

If traffic volumes at an intersection at different times are predictable, i.e., peak hours often occur at the opening and closing times of schools or offices, then this type of congestion is referred to as recurring congestion, which can be avoided and its effects mitigated by adopting a time-of-day approach, which uses preset sequences derived from past data [4]. But this method does not adapt well under low-density traffic or non-recurring congestion conditions, which occur due to random events on the roads, such as accidents, vehicle breakdowns, etc. We need to adopt an approach in which both the phase and cycle timings are not fixed, but both adapt based on traffic requirements. This requires the detection of random or non-recurring congestion or traffic conditions and the adaptation of signal timings in an effective way to reduce congestion and mitigate its effects.

Various types of sensor technologies have been deployed by researchers to detect traffic congestion and optimize traffic flow. For example, congestion patterns can be identified using GPS data gathered from moving vehicles [5], or through the collection of content generated by users on social media platforms [6]. One interesting approach adopted by Joyo et al. [7] used a smart antenna to reduce waiting times at intersections by exploiting the locations of phones; however, this method was not accurate as pedestrians also carry phones [7]. Other sensor technologies integrated into current signaling systems which adapt traffic signals include the use of RFID tags on automobiles [8], inductive loops [9], radar sensors [10], wireless sensor networks [11], ultrasonic sensors [12,13], and IR sensors [14]. But such systems often require the installation of multiple sensors that are too near the roadside, making them vulnerable and prone to vandalism and damage. Recently, traffic management systems have increasingly begun using video surveillance and monitoring systems for improving traffic safety at intersections. Road intersections are often equipped with cameras for surveillance purposes and the images acquired by them are rich in information, which can be used to perform different tasks, such as detecting, tracking, and obtaining information about vehicles, including license plates. A detailed comparison of different sensing technologies is provided in Table 1. This information can be used to assess traffic conditions for optimizing the traffic flow and reducing red light violations.

Table 1. Comparison of sensing technologies for vehicle detection and monitoring.

Sensing Technology	Classification	Multiple Detection	NPR	Cost	Small Dimensions	Range	Power	Easy to Install
Inductive loops	✗	✗	✗	Low	No	High	High	No
Magnetometer	✗	✗	✗	Low	Yes	Medium	Low	Yes
Magnetic induction coil	✗	✗	✗	Low	Yes	Low	Low	No
Microwave radar	✓	✓	✗	High	Yes	Long	High	Yes
Active infrared	✓	✓	✗	Medium	Yes	Medium	Medium	Yes
Passive infrared	✗	✗	✗	Medium	Yes	Medium	Low	Yes
Acoustic array	✗	✓	✗	Medium	Yes	Short	Low	Yes
Radar	✗	✗	✗	High	Yes	Long	High	Yes
Cameras	✓	✓	✓	High	No	Long	High	Yes
Infrared	✗	✓	✗	Medium	Yes	Medium	High	Yes
Ultrasonic	✗	✗	✗	Low	No	Short	Medium	Yes
Wireless Communication	✗	✓	✗	Low	Yes	Medium	Medium	Yes

Vehicle and license plate detection is a computer vision task focused on identifying specific visual objects in images or video frames. Traditionally, unsupervised image processing methods have been used for these tasks, which require manual labeling of images for model training [14]. OpenCV is commonly used for this purpose, although the accuracy can be affected by challenges such as occlusion, lighting, shadows, and clutter [15]. HOG and DPM came after groundbreaking work in the realm of objects performed by the Viola–Jones algorithm. Deep learning methods have become popular for object detection owing to their robustness in handling occlusions, lighting, and complex scenarios. These methods require large labeled datasets, although public datasets can facilitate the training process. Detection algorithms are classified into two- and one-shot types [16]. Two-shot algorithms, such as RCNN and Faster RCNN, first propose object regions, offering higher accuracy but slower processing. One-stage algorithms such as YOLO, SSD, and RetinaNet directly predict bounding boxes, making them faster and more suitable for real-time applications [16]. YOLO has gained popularity because of its ease of use and fast response, making it suitable for real-time applications. This led to the use of YOLO for vehicle detection.

In addition, the number of automobiles on the road encourages more criminal activities and traffic violations. The daily volume of vehicles implies that automatic number plate recognition is becoming increasingly important. It was first recognized by British law enforcement agencies in 1976, and has recently gained popularity in a diverse range of applications, including effective traffic administration, law enforcement, and security applications. License plate recognition (LPR) is gaining popularity owing to its extensive use in various industries, including traffic monitoring, toll collection, and criminal searches [17]. Conventional techniques identify the LP by processing a small number of characteristics and using manually created features such as edges, colors, and shapes. Deep learning-based methods, which have recently shown encouraging results, automatically extract reliable characteristics from data. Deep learning-based methods approach this as an optical character recognition (OCR) process. Character recognition is regarded as an object detection procedure because these approaches can only process a certain number of characters, allowing LP detection and recognition to be performed at the same time [18].

Digital twin technology has recently gained popularity across various industries. It represents a physical object, person, system, or process in a virtual fashion modeled within a digital version of its environment. The digital twin serves as a virtual representation of the actual road network in a traffic control system, enabling real-time traffic pattern analysis, congestion prediction, and signal timing optimization based on real-time sensor data. This in turn enables proactive traffic management and speedier reactions to events such as accidents or road closures [19].

Despite the development of numerous intelligent transportation systems (ITSs) and object detection techniques, many face challenges such as low adaptability to diverse traffic conditions, limited scalability, and insufficient integration of real-time data into decision-making processes. Additionally, current solutions often neglect the integration of license plate recognition systems, which can substantially improve traffic law enforcement. This study bridges these gaps by introducing a comprehensive system that integrates advanced YOLOv11-based object detection with license plate recognition and digital twin technology for adaptive traffic control. In contrast to conventional methods that depend on fixed signal timings or single-purpose algorithms, our approach optimizes traffic light timings dynamically based on real-time traffic, detects red light violations, and captures the details of the license plate. This study addresses the critical need for a scalable, data-driven solution that can operate under diverse conditions and significantly improve urban traffic management systems. This represents a step forward in addressing traffic congestion and rule enforcement comprehensively.

This research offers several key contributions:

- Development of an adaptive traffic light control system that dynamically adjusts signal timings based on real-time traffic conditions. By utilizing machine learning algorithms, the system optimizes traffic flow and reduces congestion at intersections, thereby enhancing urban traffic management.
- Integration of YOLOv11-based vehicle detection with license plate recognition within a single adaptive framework.
- Performance evaluation of state-of-the-art object detection models YOLOv11 and YOLOv8.
- Development of lightweight, cost-effective signal controller capable of operating in real-time.
- Validation of the system in real-world scenarios, showing its potential for traffic congestion reduction and rule enforcement.
- Implementation of a web-based digital twin that offers a virtual representation of intersection by visually displaying real-time traffic information, signal timings, and red light violations.

2. Related Work

Many algorithms have been employed to facilitate real-time decision making and optimization of traffic flow. Some commonly employed algorithms include YOLO, reinforcement learning, and convolution neural network (CNN). These algorithms can be employed individually or in combination to develop adaptive traffic control systems capable of dynamically adjusting signal timings and optimizing traffic flow in real time. The choice of a particular algorithm depends on factors such as the specific objectives of the traffic control system, complexity of the traffic environment, and available computational resources.

The use of fuzzy control for urban traffic signals represents an effective and nuanced approach for addressing urban traffic challenges. Fuzzy logic-based controllers leverage expert knowledge to make real-time decisions under uncertainty, offering resilience against noisy or incomplete data. On the other hand, self-organizing systems focus on local optimization at intersections, enabling decentralized and scalable solutions. Previous studies have highlighted their effectiveness in reducing congestion with minimal infrastructure. Wath et al. [20] evaluated the problem of traffic congestion and the challenges that are common in urban areas using fuzzy logic, which allows for handling imprecise and uncertain input data. The algorithm was implemented in MATLAB, which uses image processing libraries to perform vehicle detection. The images were converted to binary frames, gray codes, enhanced through a Wiener filter, and Gaussian density was utilized to filter the noise. The researchers in [21] used images captured during daylight to calculate traffic signal timings based on queue length and flow rate. These parameters were then fed into the Fuzzy Logic system to calculate the signal timings with an accuracy ranging from 40% to 100%. The system was simulated, and it was shown that 18% more vehicles were able to cross the intersection in a single phase, and the wait time was reduced by 21%. The research in [22] modified the reward function of a system using a Deep Q-network with fuzzy logic to consider the waiting time of a vehicle in every lane. The simulation results obtained from SUMO exhibited a reduction of 18.46% in the waiting time of the vehicle compared to the conventional Deep Q-network with fuzzy logic. However, fuzzy logic-based solutions are not optimal because they rely on predefined rules set by experts. These rules are static and do not adapt to new traffic patterns, resulting in suboptimal performance under changing conditions. The performance of fuzzy logic systems is affected by the quality of the designed rules. This dependence on rules also limits the scalability, as coordinating traffic across multiple intersections requires a significantly larger rule base,

making computations heavy [23]. These limitations make it less suitable for modern data-driven traffic management scenarios, where advanced techniques such as reinforcement learning, deep learning, or hybrid systems offer greater potential.

Researchers have also successfully employed reinforcement learning (RL) [24] to optimize traffic signal control policies, where RL agents learn to make decisions based on rewards received from the environment, allowing for adaptive signal timing adjustments to minimize congestion and maximize traffic flow efficiency. Ma et al. [25] conducted a simulation study comparing the suggested method (SDAC) with classic adaptive signal timing (AST), Deep Q-Network (DQN), and simplified variants of the actor–critic model (RAC and DAC). The results unequivocally show that SDAC is superior in lowering queue lengths and average delay times and enhancing throughput under different traffic demand situations. Tan et al. [26] proposed a Deep Reinforcement Learning (DRL)-based adaptive traffic signal control framework to address the complex and dynamic nature of traffic scenarios by explicitly considering factors such as realistic traffic conditions, sensor data, and physical constraints inherent in the traffic environment. The framework was validated using the VISSIM platform that replicates real-world traffic scenarios and sensor data streams. Deep Q-Network (DQN) algorithm was used to evaluate the effectiveness of the framework. The results indicate that the fully actuated controller demonstrates superior management of queue lengths in both directions compared with the pre-timed controller. Damadam et al. [27] also used Deep Reinforcement Learning (DRL) to enhance traffic control in the urban area of Shiraz City. Real traffic data from the transportation and municipality were used, and a distributed Multi-Agent Reinforcement Learning (MARL) algorithm was employed at each intersection for traffic signal control. Each intersection was equipped with an RL agent tasked with managing local traffic lights to regulate vehicle flow in all directions. The efficacy of the approach was evaluated by performing numerical simulations on an open-source simulation platform, SUMO, at two synthetic intersections. The findings revealed a significant decrease in vehicle waiting time compared to the conventional traffic signal control system. Despite its superior performance and efficacy, it suffers from the issue of generalizability. It focused on a specific area of Shiraz City with six signalized junctions, which may limit the generalizability of the findings, lack real-life validation, and does not explicitly address external factors. In general, for problems with many states and actions, such as traffic management, testing all possible scenarios is challenging owing to the vast range of situations and variables, making it impractical and time-consuming to conduct sufficient experiments. One issue is that RL models trained in simulations often struggle to generalize to real-world traffic conditions. In addition, the success of RL systems relies primarily on the design of the reward function. Therefore, poorly designed reward functions can lead to inadvertent behaviors such as prioritizing short-term gains over long-term efficiency or unfairly favoring certain traffic flows. However, these limitations can be overcome by combining them with a rule-based approach to enhance their capability in real-world traffic management [28].

Researchers are becoming increasingly interested in using machine learning techniques, such as neural networks, for traffic light control because of their demonstrated performance in various areas. CNNs provide fine-grained control over individual aspects of traffic light control and are tailored specifically for this application, allowing for the incorporation of domain-specific knowledge and optimizations to improve performance and reliability. Furthermore, it can achieve high levels of accuracy, which is crucial for ensuring safe and efficient traffic flows at intersections. Rao et al. [29] used a CNN to enhance the accuracy of detecting vehicles and their speed using a diverse dataset comprising images collected from different weather conditions and environments. Bidwe et al. [30] deployed a five-layered CNN with different input feature maps and filter sizes to estimate

the traffic density and classify them into High, Medium, and Low traffic categories. The system was trained using a dataset that consisted of traffic video footage captured by highway CCTVs in Seattle and Washington with a total of 13,326 images, achieving an accuracy of 99.6%. Although the model demonstrates high accuracy, the use of deeper networks makes it prone to overfitting, requires more computational resources for training, and may be difficult to optimize. Methiane et al. [31] used the 2D-CNN algorithm for detecting and counting vehicles to calculate the traffic density and change the traffic signal lights at an intersection. A comparison was made between Simulation of the Urban Mobility (SUMO) and Pygame implementation, and the results showed that Pygame has 90% more accuracy than SUMO. Chaudhari et al. [32] deployed a Faster R-CNN for traffic management, particularly for vehicle segment problems. This problem is very challenging because of occlusion, background clutter, and variations in traffic density. The Faster R-CNN incorporates adaptive background modeling techniques to address issues such as noise, inconsistency, and unbalanced classes in images, enhancing the robustness of the segmentation process. The custom dataset used comprises 6000 images of multiclass vehicle objects and incorporates challenging factors, such as traffic jams and overlapping vehicles. The proposed system shows promising results on benchmark datasets, such as COCO and DAWN, which contain challenging images with different environmental conditions. Although Faster R-CNN is robust and highly accurate, it is computationally intensive and requires significant processing power, particularly during the training and inference stages. This can result in longer processing times and higher resource requirements. Faster R-CNN requires fine-tuning to adapt the model to the characteristics of the target dataset, which requires careful consideration of the hyperparameters. The choice of anchor box size and aspect ratio during region proposal generation can affect the model's ability to detect objects of different scales and shapes.

The field of adaptive traffic control systems has seen significant advancements with the emergence of deep learning techniques, particularly the You Only Look Once (YOLO) algorithm, which excels in speed owing to single regression. Asha et al. [33] utilized a combination of YOLOv2 together with the Correlation Filter technique to enhance the accuracy and efficiency of vehicle counting on highways by analyzing a video captured using a portable camera. The correlation filter is trained by utilizing a k-dimensional Histogram of Oriented Gradient (HOG) features extracted from an object sourced from online data with a Gaussian template as the desired output. They achieved an accuracy of 92–100% when detecting different classes such as cars, bikes, and buses. Shinde et al. [34] employed the YOLO algorithm to detect vehicles on the road and perform traffic flow analysis. The collected data are analyzed on queue density and waiting time per vehicle, which is then used to adapt the traffic control signals, enabling more vehicles to pass through intersections safely with minimal waiting time. The study conducted by Oltean et al. [35] deployed a pretrained Tiny YOLOv3 for detection, motion estimation, and tracking vehicles in both one-way and two-way traffic scenarios, ensuring that each vehicle was counted only once. The author claims to achieve an accuracy of 100% and a lower inference time of 33.5 FPS on real-life traffic videos. Mahmood et al. [36] deployed an infrared camera and the YOLOv3 algorithm to detect, track, and count vehicles in traffic scenarios to estimate the traffic density for enhanced visibility, anti-interference ability, all-weather observability, high precision, and long detection distances for vehicle tracking in traffic. The proposed technique achieved a processing rate of 18.1 FPS. Similarly, Khan et al. [37] used the YOLOv3 framework and considered various time parameters, including the number of vehicles (such as two-wheelers and four-wheelers), road width, and junction crossing time, to estimate 'on' time for the green light, achieving an average accuracy of 81.1% in detecting vehicles. Anil et al. [38] deployed a pretrained YOLOv3 and SORT

algorithms to detect and count vehicles, respectively. The system was tested using the UA-DETRAC Dataset, consisting of 10 h of videos recorded in 24 places. Sakharae et al. [39] applied YOLOv3 in conjunction with the OpenCV library and Darknet neural network framework, achieving real-time accuracy exceeding 90% for the detection of three classes. The successor in the form of YOLOv4 incorporates several architectural enhancements and training strategies to improve the detection accuracy compared to YOLOv3. It achieved higher precision and recall rates, particularly for small objects and challenging scenarios. Sentosa et al. [40] employed a pretrained Yolov4 model to detect and count vehicles. Yolov4 predicts the class and bounding box of each object using the head section. The author limited the number of classes to five (cars, buses, trucks, bicycles, and motorcycles) to increase the speed of object detection. The system required approximately 18 s to detect all classes and 15 s for five vehicles.

YOLOv5 offers speed and accuracy, making it more suitable for traffic control. Dirir et al. [41] used YOLO v5 to detect objects and CSRT to count and track objects. It was evaluated using a dataset comprising 16 videos covering diverse traffic densities, weather conditions, and image qualities and yielded promising results. Hu et al. [42] used camera calibration techniques and YOLOv5 to solve this problem. The calibration element focuses on unknown camera parameters and the complex geometries of the road. The developed model was trained on a large dataset, which enabled it to adhere to accuracy requirements. The author in [43] detected and classified vehicular objects in densely crowded images using YOLOv5 and Non-Maximum Suppression Ensembling. The proposed model addresses the limitations of YOLO in detecting small objects by combining the predictions of four YOLOv5 models. The performance of the model was evaluated on the Dhaka AI dataset, achieving an mAP@0.5 of 0.458 with an inference time of 0.75 s. However, ensembling multiple YOLOv5 models using techniques such as Non-Maximum Suppression can introduce complexity into the system. Shirazi et al. [44] deployed YOLOv5 in conjunction with the discrimination correlation filter for detecting and tracking cars. The traffic data provided by the system includes the speed of the vehicles as well as the turning movement count. The analysis was performed using the turning movement count with SUMO. The dataset used was the pedestrian dataset from the University of Nevada, Las Vegas. The efficiency of the system was improved by using this system by 0.62 s and an accuracy of 90% was achieved. Kunekar et al. [45] used YOLOv7 to optimize the traffic flow by adapting the green lighting duration so that the maximum number of vehicles could pass, and waiting was reduced. It performs object detection in a single forward pass of the neural network, thus enabling it to accurately recognize objects in images, streaming broadcasts, and videos. Srikantswera et al. [29] deployed YOLOv8 to detect vehicles, and by applying correlation filters, they were able to detect and count the vehicles. A total of 1044 images of vehicles and 1213 images of emergency vehicles were used to train the model. The test results revealed that the system had an accuracy of 90.25%. A similar study was conducted by [30] employed YOLOv8 on the Jakarta-Cikampek Toll Road for traffic counting, bringing notable enhancements to the process. It divides the detected objects into three categories: cars, buses, and trucks. The outcomes of the traffic volume computations were stored in Excel (xls) using the OpenXL function. The system achieved accuracies of 98.74%, 92.69 %, and 96.85% for cars, buses, and trucks, respectively. Although YOLOv8 offers significant advantages in traffic counting and object detection tasks, there are some potential challenges and limitations associated with its use. A large amount of annotated training data is required to achieve high accuracy. Moreover, being a deep learning model may require powerful hardware resources such as GPUs for efficient training and inference. Many researchers have conducted only evaluation research showing the efficiency of the methodology without deploying it in real life [36,41,42]. YOLOv11 is an improved version of YOLOv9 and YOLOv10 with better

training techniques, feature extraction algorithms, and architectural designs. It stands out because of its exceptional combination of speed, accuracy, and efficiency, making it one of the most potent ultralytic models to date. Therefore, YOLOv11 was preferred in this study to provide better speed and accuracy. In addition, it requires minimal training data and can be easily ported to cost-effective hardware, such as Raspberry Pi.

Many strategies have been proposed to improve ANPR system performance. The introduction of deep learning approaches has revolutionized the field of ANPR. For instance, the AOLP dataset, which includes three picture categories, was utilized in [46] to test the CNN, RNN, and LSTM models. While RNN with LSTM was utilized for character identification, the CNN model was trained with four layers to categorize the photos of license plates. In object recognition, the model used in this study yielded an accuracy of 86.22%, precision of 97.18%, and recall of 97.19%. However, the ability of this method to handle various climatic circumstances and license plate forms is limited. Another study [47] proposed a hybrid CNN and SVM technique. The segmented characters were then scaled to 28 × 28 images for further processing. An additional study conducted by [48] also used a CNN for license plate classification. Three well-known benchmarks, the Stanford Cars dataset, the Indian License Plates dataset, and the Car License Plate Detection dataset, were utilized to assess the suggested architecture, and the model achieved an accuracy of 90%. Recently, models that can accurately recognize objects in real time, such as YOLO, have attracted considerable attention. The research in [49] used the YOLOv3 algorithm for object detection on roads and OpenCV for traffic rule violation detection. It identifies vehicles violating red light signals by drawing lines on the video feed to represent traffic signals and detecting when a vehicle crosses the red light boundary. The system achieved high accuracy for both object detection and number plate recognition. The accuracies of car detection and number plate recognition were approximately 94% and 98%, respectively. The processing time for detecting red light violations and number plate recognition is 2.84 and 3.85 s, respectively. The research in [50] employed YOLOv8 for ANPR and achieved an accuracy of 93%.

Despite these advancements, the integration of real-time vehicle detection and license plate recognition into adaptive traffic signal systems remain underexplored. Most RL and fuzzy logic approaches assume simplified sensing models, neglecting the potential of real-time, high-resolution data. This study addresses this gap by combining state-of-the-art YOLO-based detection with a robust adaptive signal controller, thereby advancing the field by demonstrating how real-time detection improves traffic flow efficiency. Moreover, a web-based digital twin is provided to monitor traffic conditions in real time without raising concerns about privacy.

3. Preliminaries

In this section, we introduce and explain some of the key terms and methodologies used in this study to ensure clarity for the audience.

3.1. Road Intersection Scenario

The typical structure of a signalized four-way road intersection has two roads crossing each other. This configuration is one of the most common intersection designs used in urban and suburban road networks. Figure 1 shows a four-way intersection of the adjacent roads on each side. Each road features a median strip that divides it into two carriageways, each with three lanes in both directions. Four connecting roads, R_1, R_2, R_3, and R_4, meet at central point I_1, and each road is divided into two carriageways (lanes) for opposing traffic directions. Vehicles traveling straight continue across the intersection to the opposite side. Vehicles turning left yield to oncoming traffic. Right-turning vehicles merge with traffic on adjacent roads. Traffic signals S_1, S_2, S_3, and S_4 are installed on roads (overhead

on signal poles) to regulate the flow of traffic by providing instructions in the form of red, yellow, and green lights. Traffic signals are electrically operated control devices that guide roadway users by assigning a right-of-way to different approaches and movements. They facilitate shared road spaces by managing conflicting movements, allocating delays, and enhancing the safety and mobility of various traffic actions. The video camera is installed over the traffic light facing down to detect and count the vehicles waiting at the intersection when the signal is red and inform the controller that it can initiate actions such as calling for additional green time and allowing for the passage or extension of the green interval to accommodate the movement of waiting vehicles. During the morning when the university starts, the number of vehicles approaching S_2 and S_3 is higher than that of S_1, thus causing congestion at these two signals, and S_1 is more congested at the end of the university day. For a fixed cycle time, the maximum waiting time for a vehicle approaching the signal when it turns red is between 120 s and 160 s with no congestion. Suppose S_1 has medium-level congestion and the car approaching the signal is unable to pass during the first green phase; the maximum waiting is between 280 and 320 s. In the case of higher congestion, a vehicle might need to wait for another cycle of lights, increasing the maximum waiting time to 440–480 s. This is a scenario which most university students come across when leaving the university, and they must wait for this much time at the intersection. The timings are summarized in Table 2.

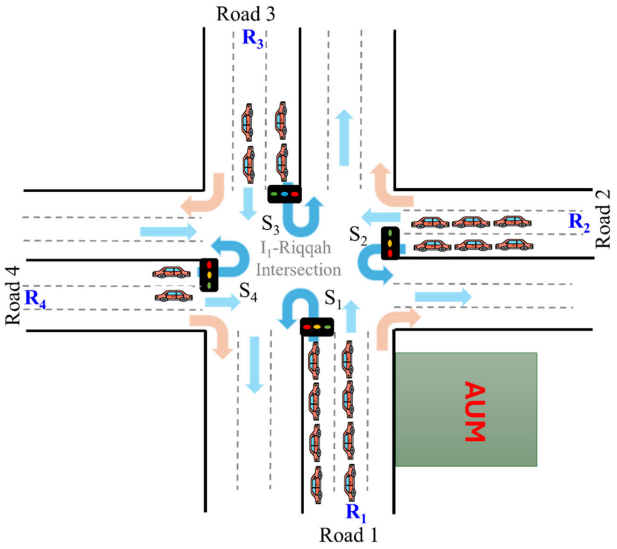


Figure 1. A 4-way road intersection scenario.

Table 2. Waiting time in intersection—baseline scenario.

Congestion Level	Maximum Waiting Time
With no congestion	120–160 s
Medium congestion	280–320 s
High congestion	440–480 s

3.2. YOLOv11

We used YOLOv11 for vehicle and number plate detection, which is the outcome of many improvements made to the original YOLO. Its architecture has been crafted to enhance both speed and accuracy, building upon the progress made in earlier versions, such as YOLOv8-10. The major improvements include the C3K2 block, SPFF module, and C2PSA block, which improve the model's capability to process spatial information while preserving the inference speed. C3K2 blocks were used for feature extraction using

YOLOv11 throughout its backbone. The C3K2 block is a result of the CSP bottleneck and improves information flow by splitting the feature map and applying several small convolutions of 3×3, resulting in improved feature representation with fewer parameters compared to previous YOLO versions, such as YOLOv8. C3K2 utilizes the C3K block to process the data, which is very similar to the C2F block, and processes the input through a convolutional block and multiple bottleneck layers with concatenations. In C3K2, two convolutional blocks are placed at the start and end, with C3K blocks between them. The outputs are then concatenated to enhance both the speed and accuracy while maintaining the balance provided by the CSP structure. A comparison of the C2F and C3K blocks is shown in Figure 2.

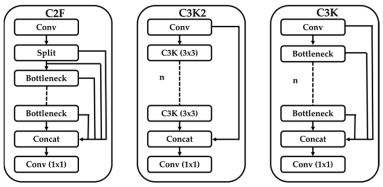


Figure 2. Comparison of C2F and C3K blocks.

The spatial pyramid pooling fast (SPFF) module used by YOLOv11, shown on the left in Figure 3, pools features from various areas of an image at multiple scales, enhancing the network's capability to detect objects of varying sizes, particularly tiny objects, which is a known issue in previous versions. SPFF performs this task by employing several max-pooling operations with different kernel sizes to gather contextual information, ensuring that small objects are effectively detected by combining information from different resolutions. By incorporating SPFF, YOLOv11 maintained real-time performance while improving its capability to detect objects of various scales.

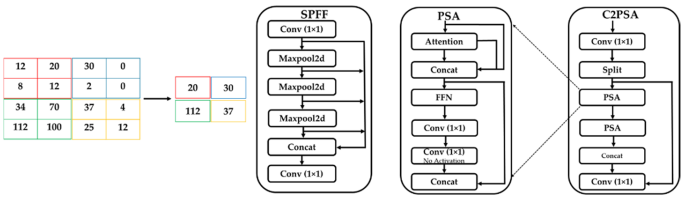


Figure 3. Spatial pyramid polling fast and C2-Position-Sensitive Attention Block (C2PSA).

The attention mechanism was introduced in YOLOv11 through the C2PSA block (cross-stage partial with spatial attention), which improves the emphasis on vital sections inside the image by accentuating spatial relatedness in the feature maps. PSA class provides the ability to apply position-sensitive attention and feed-forward networks to input tensors, thereby augmenting the feature extraction and processing capabilities of the model. The layer first processes the input through an attention layer, concatenates the input with the output of the attention layer, and propagates the outcomes through a Feed-Forward NN. This is followed by a convolutional block and another convolutional block without activation. Finally, the output from the second convolutional block was concatenated with the initial concatenation result. As shown in Figure 3, the C2PSA block incorporates two Partial Spatial Attention (PSA) components for processing isolated sections of the feature map, which are subsequently concatenated, similar to the C2F block structure. This design helps the model emphasize spatial information while balancing computational efficiency and detection accuracy. By applying spatial attention to the extracted features, the C2PSA block boosts the ability of the model to focus on the regions of interest, enabling YOLOv11 to surpass earlier versions, such as YOLOv8, especially in situations where precise object details are critical for accurate detection.

3.3. Digital Twin

There exist many definitions of digital twin, but no practitioners use the same definitions. The LMS researcher [51] described this as follows:

"While we see many definitions of" Digital Twin, "LMS Research keeps it simple: A Digital Twin is an executable virtual model of a physical thing or system".

One of the critical elements of a digital twin is that it must be linked to a real-world object that exists physically; otherwise, it will be considered a model only. It is undeniable that DT adds value to any industry by decreasing time-to-market, streamlining processes, cutting maintenance costs, boosting user engagement, and integrating information technologies. In 2020, the worldwide potential of DT is estimated at USD 37.9 billion annually [52], and it is anticipated to expand rapidly in the following years. The application of DT in traffic is not very common, but its integration with existing traffic control systems can provide effective solutions for tackling today's complex traffic challenges. The DT for such a system can be created using data from various sources such as cameras, GPS, and other monitoring systems. These DTs can be updated in real-time and have been used by many researchers to reduce congestion at intersections [53].

4. Methodology

The proposed system integrates a machine learning-based vehicle detection algorithm with optical character recognition (OCR). The former is used to assess traffic volume, and the latter is used to read the license plate of a vehicle violating a red light in real time. By employing YOLOv11 for object detection and Tesseract OCR for LPR, the system achieved a robust solution for dynamic traffic management. The controller adjusts green light durations proportionally to the vehicle volume across lanes and enforces traffic regulations as well.

4.1. System Architecture

Figure 4 represents the architecture of an adaptive traffic light control system that utilizes YOLOv11, Raspberry Pi, and Arduino to dynamically adjust signal timings based on real-time traffic congestion detected at an intersection and record any violation of a red light by reading the number plate. The camera was positioned at the intersection to monitor traffic flow. It captures real-time video feeds of vehicles approaching an intersection. The system detects the traffic volume by counting the number of vehicles in the region of interest. Raspberry Pi, which hosts a traffic light engine, runs an object detection model such as YOLOv11, the latest iteration in the YOLO series of models widely used for real-time object detection, to detect vehicles. Raspberry Pi adjusts green light timings for this section of the road to reduce jams and communicates the update signal timing information to the Arduino-based controller. The Arduino is used as a hardware interface to control the timing of red, yellow, and green lights based on the traffic data received. For example, if high congestion is detected in one direction, the green light duration is increased in that direction, whereas other directions may have shortened durations. In addition, the engine can detect red light violations. Minimum and maximum green light timings prevent starvation in low-congestion lanes. In addition, if any vehicle crosses the region marked by a line, its number plate is detected, and this region is cropped from the image. The license plate number of the vehicle was read using the OCR technique and stored in a database for further action. A web-based virtual model, known as a digital twin, was also provided to display the signal simulation of an intersection and countdown timer for each section of the road. Raspberry Pi updates this virtual model using real-time data on traffic conditions and signal timings.

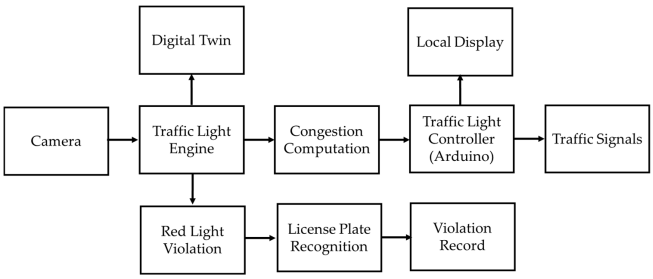


Figure 4. System architecture of the adaptive traffic light control system with integrated license plate recognition and digital twin visualization.

4.2. Camera Placement

In this study, cameras were deployed to monitor the upstream traffic, as shown in Figure 5. Upstream viewing offers the benefit of traffic queues not being blocked by bloomed headlights, glaring pathways, headlight beams, and obstructions from elevated vehicles. The generally rated surveillance range outlined by vendors is 10 times the height of the camera. Conventional design measures restrict the bounds to shorter distances owing to road outlines, congestion levels, and severe weather conditions. The distance d at which the system can distinguish between two closely spaced vehicles also depends on the gap between them, their elevation, and the pixel size. For roads without a grade, d is estimated by

$$d = h\frac{Veh_{gap}}{Veh_{height}}$$

where h denotes the camera mounting height, Veh_{gap} is the intervehicle gap, and Veh_{height} is the height of the vehicle.

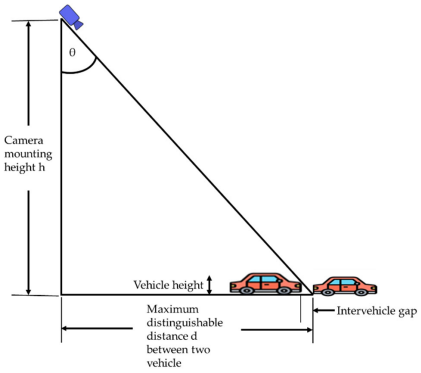


Figure 5. Calculation of camera mounting height.

4.3. Dataset for Model Training

Most studies rely on either pre-trained models or a custom dataset collected via recording devices to train ML models. For instance, the YOLO model is pre-trained on the MSCOCO dataset [54] and has been used in many research studies, such as [32,34,35,40,44,55]. Some other studies used different datasets, such as [26], which used datasets from the Iowa Department of Transportation that encompass traffic flow information during various periods, including morning peak, evening peak, and midday durations. The detection in adverse weather nature (DAWN) dataset [56] contains real-world images captured under various hostile weather conditions. The UA-DETRAC Dataset [38,42] comprises 10 h of videos that were recorded at 24 different locations. Other datasets used include Dhaka AI [43], Seattle and Washington [30], and the DSAT Website [57]. Other researchers used custom datasets for testing [31,37,38,41,58]. Similarly, for LPR, the dataset developed by [39] contains a diverse set of images, allowing the investi-

gation of LP-related features, and it has been tested under diverse conditions such as tilt, blur, rotation, or varying weather conditions [59]. The Global License Plate Dataset (GLPD) by Agarwal [60] contains over five million photos and includes a variety of samples taken from 74 nations, including careful notes, such as characters on the license plate, license plate corner vertices, plate segmentation masks, and car model, color, and manufacture.

In this study, we used a dataset with two primary components: a vehicle detection dataset and a license plate recognition dataset. We used open-source datasets, such as the Vehicle_Detection_YOLOv11 Image Dataset [61] and vehicle_license_plate [62] published on Roboflow in 2023 and 2021, respectively. The first dataset comprises 779 images that were split into training, validation, and test sets: 681, 65, and 33, respectively. The original dataset had 325 images that were augmented, producing three augmenting versions of an image that effectively tripled the size of the dataset. Augmenting includes horizontal flipping of the vehicles, that is, a vehicle facing left in the original image will face right in the augmented image, helping the model to learn left–right invariance. Random cropping of 0–20% was also performed, which helped the model focus on smaller portions. The images are also rotated by 10 degrees (both clockwise and counterclockwise), the brightness of images is adjusted, ranging from −15% to +15%, and noise of up to 0.5% of pixels is added. The second dataset used was the vehicle and license plate dataset, which comprised 5555 images, with 5102 images used for training, 432 for validation, and 21 for testing [62]. The dataset contained 2167 original images, which were augmented by flipping the images horizontally and rotating them by 10° in both directions. To ensure diversity, data were collected under various weather conditions, lighting variations, and traffic contexts, as shown in Figure 6. Both datasets consisted of a diverse set of images showcasing the various types of vehicle license plates used in Indonesia. It encompasses a range of lighting conditions, weather scenarios, and plate orientations to enhance the robustness of the model training and testing. These datasets also included images that were unrelated to the project and were removed. However, the dataset was enhanced with images of the vehicles used in Kuwait under different conditions, such as during daylight and evening. The images were acquired using mobile phone cameras, and some images were obtained from the Internet. These images were annotated and labelled manually using the LableImg tool with two classes: license_plate and vehicle.

Figure 6. Dataset images represent various traffic scenarios and environmental conditions.

We then processed the dataset to satisfy the requirements of the algorithm used. First, we cleaned the dataset but removed unwanted and related images from the dataset. Both the datasets used in this project had multiple classes. We relabeled these datasets into two classes similar to our custom dataset. We fused these three datasets into a single dataset with only two classes. To expedite this otherwise tedious task, we used a Python script. The fused dataset comprised approximately 7000 images, of which 6394 were used for training, 594 for validation, and 101 for testing. It contains images from different regions and under different environmental conditions, including Kuwait.

4.4. Signal Controller

The signal controller is the cornerstone of the proposed adaptive traffic management system and performs the following tasks.

- Detects the vehicles in the image.
- Calculates the congestions level by counting the number of cars.
- Computes the green time duration for each phase before it is about to turn green during every cycle.
- Communicates the timings to the physical controller controlling the signal lights as well as the digital twin.
- Controller ensures that there is no starvation but restricts minimum and maximum green light time.
- Detects the red light violations and records license plate number of the car.
- The controller operates in a feedback loop, continuously updating the signal timing based on real-time data.

Object detection of the vehicles in the image involves searching the entire image for areas that geometrically or photometrically match the target item. If there is a substantial degree of resemblance between the template and the image, detection is announced. Vehicle detection accurately classifies the types of vehicles and locates them by identifying vehicles in a designated monitoring zone. The YOLOv11 detection module serves as the primary processing engine of our system, building on the foundation of earlier YOLO architectures. Figure 7 shows the iterative nature of model training, evaluation, and deployment in object detection systems. This ensures that the YOLO model is robust, accurate, and ready for practical applications, such as traffic monitoring, safety compliance, and industrial automation.

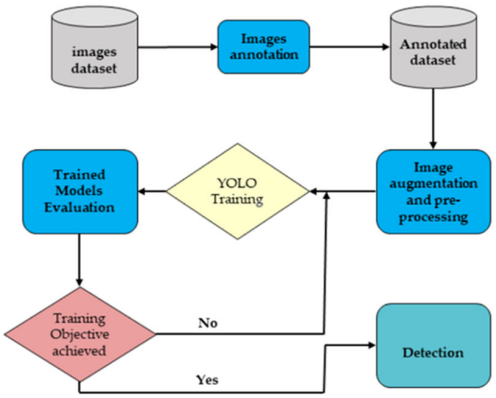


Figure 7. Vehicle detection design flow.

Different methods are used to optimize traffic signal time based on cars waiting at the junction, including traffic density or traffic volume. The former uses the concentration of vehicles, and the latter uses the number of cars accumulated at an intersection over a specific period, that is, during a red light. We used a technique based on traffic volume, as

it does not require calculation of the distance which is needed for computing density. The traffic volume at an intersection with multiple lanes can be calculated as follows:

$$V_{total} = \sum_{i=1}^{L} N_i$$

N_i is the number of vehicles counted in lane i, and L is the maximum number of lanes.

The proposed system determines the congestion level by computing the number of cars in a frame and the traffic light timings, as shown in the flowchart in Figure 8. Upon starting the system, a video frame is received from the camera installed at the intersection. We define the fixed region of interest (ROI) in the frame representing the intersection area where traffic congestion is monitored, which essentially reduces the number of lanes to one only. We then define and initialize the variables to keep track of the vehicle count in the ROI, and subsequently compute the congestion level. Each frame was then pre-processed, vehicles in the ROI were detected using YOLOv11, and their bounding box coordinates were extracted and stored in the list. If the bounding box is within the ROI, the vehicle count is incremented, and the congestion level is computed again. Based on this congestion level, new values for the traffic light timings were calculated, computed, and communicated to the traffic light controller, Arduino Uno. This process can be repeated for each frame after a specific period. However, it is recommended to acquire the frame a couple of seconds before the end of the red light and update the timing.

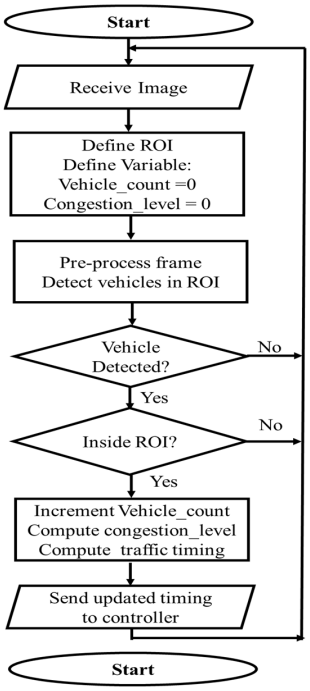


Figure 8. Flowchart for congestion calculation.

The hardware platform used for traffic light control was an Arduino Uno. This versatile microcontroller board is built around the ATmega328P clocking at 16 MHz and is widely used for collecting sensor data in diverse applications. It operates at 5 V, although the recommended range is 7–12 V (with limits of 6–20 V). It could deliver a current of up to 40 mA per pin at 5 V. Figure 9 outlines the entire sequence and flow of the system. First, all signals were pre-timed at default values of 1 s, and all signals were red. Starting with R_1, the system sets the green light on for 30 s (S_1_G_T = 30s) while the red light is turned on in all other lanes at the intersection. The green light for R_2 is turned on. While the red

light of R_2 is red, its ROI is monitored constantly by the ML engine, and depending on the congestion level, the period of the green light for R_2 is specified and communicated to the Arduino. If a high level of congestion is detected, the green light duration is configured as 30 s (S_2_G_T = 30s), 20 s for medium-level congestion, and 10 s for low-level congestion. Note that the red light of the subsequent signal (S_3_R_T) is configured with the same duration as the red light duration of the signal under consideration. The same procedure was iterated for setting the green light duration for S_3, S_4, and S_1 in the same sequence. All the signal timings were scaled down by a factor of 10 for testing purposes.

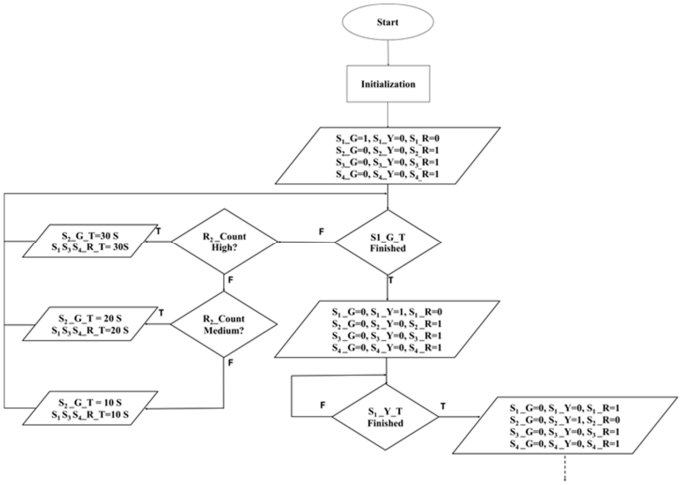


Figure 9. Flowchart explaining traffic light control.

The algorithm used for violation detection is described below. The bounding box of each detected vehicle was defined by the top-left and bottom-right corner coordinates. The algorithm first detects all vehicles in the frame, extracts their bounding box coordinates, and stores them in a list. It then compares the bottom of the coordinates of the bounding box with that of the line marking the end of the authorized region or the start of the prohibited region. If the bottom of the bounding box is ahead of the line, a violation is registered, and the bounding box coordinates are added to the list of violating vehicles.

Algorithm 1: Violation Detection

1. Input: Video frame from a traffic camera
2. Initialize an empty list 'detected_vehicles' to store the coordinates of detected vehicles
3. Initialize an empty list 'violating_vehicles' to store the coordinates of vehicles violating traffic rules
4. Define the line coordinate 'prohibited_line' marking the boundary between the authorized and prohibited regions
5. For each frame in the video:
   a. Detect all vehicles in the frame using the object detection model
   b. For each detected vehicle:
      i. Extract the bounding box coordinates '(x_min, y_min, x_max, y_max)'
      ii. Add the bounding box coordinates to 'detected_vehicles' list
      iii. Compare the bottom of the bounding box 'y_max' with the 'prohibited_line':
         -If 'y_max' > 'prohibited_line', then:
            -Register a violation
            -Add the bounding box coordinates of the vehicle to 'violating_vehicles' list
6. Output the list 'violating_vehicles' with the coordinates of violating vehicles
7. Repeat for each frame in the video

4.5. License Plate Recognition

LPR is performed using OCR, which can extract machine-printed or handwritten data text from a 2-D image and convert it into machine-readable text. This is performed by analyzing the pixels in an image to detect patterns that correspond to letters, numbers, and other characters. This is achieved by performing multiple operations, including preprocessing the image to locate the text inside, after which the characters in the image are segmented and recognized. We used an open-source text recognition engine called Tesseract, developed by Google, to perform this task. It deploys a neural network system configured to recognize text in lines. The flow of the OCR process using the Tesseract engine is illustrated in Figure 10.

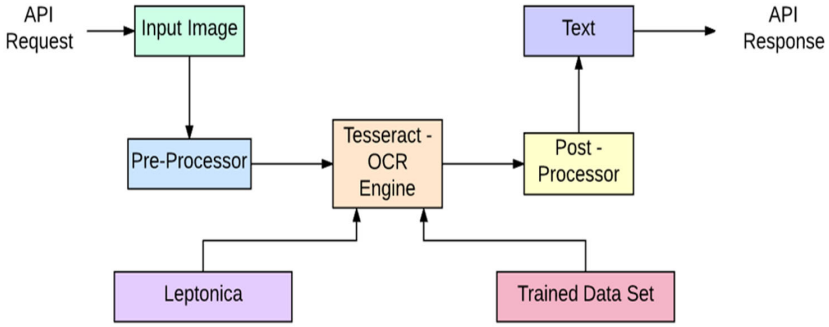


Figure 10. OCR process flow.

The processing is performed using Leptonica, which is an open-source image processing and analysis library that provides support for various image manipulation tasks, such as filtering, thresholding, scaling, rotation, and image format conversion. The input image was fed to a preprocessor module that performed normalization, scaling, noise removal, and deskewing. The normalization operation modifies the range of values of the pixel intensities, bringing them in a range that can be sensed easily. In the next step, if the images were skewed, the level of skewness was determined and corrected. For better performance, the image resolution should be more than 300 pixels per inch, and it should be increased using the Pillow library if this criterion is not met. Noise removal steps remove any distortions in the image to further enhance the accuracy of the image. Any dots or patches in the image that are found to have an intensity higher than that of other parts of the image are removed, resulting in a smoother image. This operation can be performed using fast NI Means Denoising Colored function in OpenCV. Next, the width of the strokes in the writing is made uniform through thinning and skeletonization. This is particularly useful for handwritten texts, which typically have different stroke widths for different writers. The image was then converted to a grayscale image, where the pixel colors varied between white and black. Finally, the image was converted to a binary image with only black and white colors. This was achieved by selecting a suitable threshold using Adaptive Binarization.

After preprocessing, the image is fed to the Tesseract engine, which segments it into smaller components, such as blocks, paragraphs, lines, words, and characters. This hierarchical segmentation facilitates the isolation of individual characters for recognition. However, it first detects individual lines of text in the image and then separates these lines into words and individual characters. After segmenting each character, it analyzes features such as the curves, edges, and intersections of each character. The features extracted for each character were compared to known character patterns stored in an internal database to identify the characters. The latest version of Tesseract uses a Long Short-Term Memory (LSTM) neural network model that learns from large amounts of labeled text data. This

improves the accuracy of character recognition, particularly for more complex fonts or handwriting. Once characters are recognized, a large language model is used to interpret words that help in correction by comparing recognized words with a dictionary of known words, thus improving the accuracy of the system. Finally, recognized text in machine-readable formats, such as plain text, PDF, or other structured data formats, is produced.

4.6. Digital Twin Implementation

The developed DT model for road intersections includes key structural components such as four-way intersections with adjacent roads on each side. Each road features a median strip that divides it into two carriageways, each with two lanes in both directions. The model includes four blocks of vehicle movement logic, one for each direction, and model light signalling elements. The model features a system dynamics module that simulates the operation of the local intersection controller. The technologies used to implement DT include HTML5, which defines the structure of the webpage, including the layout for the intersection and traffic signals. CSS3 provides styling for the page, including traffic signal lights, road layout, and responsiveness to various screen sizes. JavaScript is used to handle client-side functionality such as updating traffic signal lights and timers dynamically based on server updates. Socket.IO enables real-time communication between the client and the server. The client receives updates on the traffic light states and timers through WebSocket connections. The server-side framework is Flask, which serves static files and provides the API endpoints. Media queries in CSS ensure that the layout adapts to various screen sizes, making it suitable for both desktop and mobile devices. Flask web application with Flask-SocketIO is used to manage traffic signals based on car counts detected by a YOLOv11 model. The Flask app and SocketIO loaded the YOLOv11 model for car detection, defined paths to road intersection images, initialized states for four signals (S_1, S_2, S_3, S_4), and set up serial communication with an Arduino on COMX. After this, it uses YOLO to count vehicles in each image, updates the state and timer of a signal, notifies clients via SocketIO, and sends these updates to Arduino. The Arduino controller manages the transitions between signals based on the congestion level sent and sets the current signal to green for a duration based on the car count, yellow, and red iteratively. The loop continuously updates the signal states and emits updates to the clients, ensuring that they receive real-time information about the signal states. Finally, the server is started with SocketIO and runs the Flask-SocketIO app on 0.0.0.0 at port 5000.

5. Experimental Design

This section describes the experimental setup and evaluation procedures used to assess the performance of the system. Experiments were conducted using a combination of real-time video feeds and pre-labeled datasets to test the system's accuracy, processing speed, and adaptability under varying traffic conditions.

5.1. Training and Validation

The development of an adaptive traffic controller can be divided into two parts: training of the model and deployment of the model in the field. The training part is computationally intensive and, therefore, is performed on powerful machines. Some researchers, such as [33,37], trained models on local machines, which may take a longer time. One option is to use resources offered by Google Colab [32,58], which offers high-speed computational resources hosted on the cloud that can be accessed by simply creating an account and logging into it. For example, Ref [35] used a GPU on a GeForce GTX 950M, 650 CUDA core, Ref [25] used two RTX-2080Ti, Ref [30] deployed an NVIDIA GPU, and

Ref [32] employed a T4GPU. Others have used simulations to demonstrate the effectiveness of their approach using platforms such as SUMO [27,31,44] or VISSIM [26].

For the training and evaluation of our models, we used the Ultralytics module in Python [63]. We trained all models on Google Colab because of the limited processing power of desktop computers, and it supports both GPU and TPU instances. The models were trained for 125 epochs with an image size of 640. A detailed comparison of the training parameters and configuration values used for YOLOv8 and YOLOv11, which is essential for fine-tuning the performance of each model, is provided in Table 3. The Epochs parameter indicates the total number of complete datasets passed during training, while Warmup Epochs and Warmup Momentum help stabilize the learning process by gradually increasing the learning rate in the initial stages. The batch size specifies the number of samples processed simultaneously, thus influencing computational efficiency and model convergence. The Initial and Final Learning Rates determine how quickly the model updates weights; starting with a higher rate allows for broad adjustments, whereas a lower final rate enables fine-tuning. Patience controls the waiting period before early stopping if progress stalls. The Optimizer type, such as SGD or Adam, affects how the model adjusts the weights based on gradients. NMS IoU applies a threshold to minimize duplicate detections, which is essential for accurate bounding box predictions. Momentum helps in convergence speed by adjusting the influence of past gradients, while the Mask Ratio is applied to image segmentation by masking part of the input image during augmentation. Weight Decay is a regularization method to prevent overfitting by penalizing large weights, and Learning Rate Decay gradually reduces the learning rate to stabilize training as it progresses. Finally, the Scheduler determines the method for learning rate reduction, such as Cosine or Step, which helps maintain effective learning throughout training. Together, these parameters define the training behavior and help optimize the performance of the YOLOv8 and YOLOv11 models for various detection tasks.

Table 3. YOLOv8 and YOLOv11 training parameters and configuration values.

Symbol	Parameters	Values YOLOv8	Values YOLOv11
1	Epochs	125	125
2	Warmup epochs	3.0	3.0
3	Warmup momentum	0.8	0.8
4	Batch size	40	304
5	Image size	640	604
6	Initial learning rate	0.1	0.1
7	Final learning rate	0.001	0.001
8	Patience	100	10
9	Optimizer	SGD	SGD
10	NMSIoU	0.45	0.45
11	Momentum	0.937	0.937
12	Mask ratio	4	0.1
13	Weight decay	0.0005	0.0005

Table 4 provides a comparison between the YOLOv8 and YOLOv11 models, detailing their architectures, performance metrics, and computational efficiencies. YOLOv8 consists of 168 layers and over 25 million parameters, giving it a higher model capacity for complex feature extraction, whereas YOLOv11 is more lightweight with 238 layers but only approximately 2.6 million parameters. This difference is reflected in their GFLOPs (giga-floating point operations per second), with YOLOv8 requiring significantly more computations (78.7 GFLOPs) than YOLOv11 (6.3 GFLOPs), suggesting that YOLOv11 is more computationally efficient. Performance metrics show that YOLOv11 has a higher precision (0.951) than YOLOv8 (0.929), indicating that it makes more accurate positive detections, while YOLOv8 has a slightly higher recall (0.924), suggesting that it identifies

a higher proportion of actual positives. In terms of mAP50, both models perform well, with YOLOv11 achieving a slight edge (0.951) over YOLOv8 (0.942). The mAP50-95 metric, which assesses precision across stricter IoU thresholds, shows YOLOv11's significant advantage (0.854 compared to YOLOv8's 0.662), indicating a stronger performance in more challenging detection scenarios. The computational efficiency varies between the models; YOLOv8 has shorter preprocessing (0.1 ms) and post-processing times (0.8 ms), but a longer inference time (5.0 ms), whereas YOLOv11 excels in inference speed (1.5 ms) and training time but takes longer in preprocessing and postprocessing.

Table 4. Comparison of YOLOv8 and YOLOv11 models.

Metric	YOLOv8	YOLOv11
Layers	168	238
Parameters	25,840,339	2,582,542
GFLOPs	78.7	6.3
Training Time	4.61 h	0.794 h
Preprocessing Time	0.1 ms	0.5 ms
Inference Time	5.0 ms	1.5 ms
Postprocessing Time	0.8 ms	1.8 ms

Validation also plays a pivotal role in the machine learning process, enabling the evaluation of the quality of the trained models. Ultralytics YOLOV11's validation mode offers a comprehensive set of tools and metrics to assess the performance of the object detection model. Figure 11 displays the training and validation results of the machine learning model. The *train/box_loss* graph shows the loss related to the bounding box regression during training. The steady decline indicates that the model learns to better predict the bounding boxes for detected objects as training progresses. The initial box loss is approximately 1.17 and drops to approximately 0.367, which signifies a good improvement throughout the training. The *train/cls_loss* (Classification Loss) graph represents the classification loss during the training. A similar trend is observed, where the loss decreases rapidly from a high initial value of 2.801 to a much lower value of 0.263, indicating that the model is improving its ability to classify objects correctly. The *train/dfl_loss* (Distribution Focal Loss) represents the distribution loss (probably related to the Focal Loss) during training. The decrease from 1.486 to 0.946 indicates that the model is becoming better at focusing on harder-to-classify examples as it trains. Precision measures the proportion of correct identification. The precision increased rapidly and stabilized at approximately 0.951, indicating that the model reduced false positives and became more precise in its predictions. The recall measures the proportion of actual positives that are correctly identified. It started lower but climbed to approximately 0.852, indicating that the model also improved its ability to detect most actual objects. Val/box_*loss* is the validation loss for the bounding box predictions. Similar to the training box loss, it steadily declined from ~1.75 to ~0.5, which suggests that the model generalizes well on unseen data when predicting bounding boxes. Val/cls_*loss* is the validation classification loss, which decreases from 3.0 to 0.5, indicating that the model is improving its classification on the validation set and is not overfitting. The *val/dfl_loss* is the validation distribution focal loss, which decreases from ~2.0 to ~1.0, indicating that the model is learning to focus on difficult examples in the validation set. The mAP@50 metric steadily increased to approximately 0.951, showing a strong model performance at a relatively lenient threshold (50% IoU overlap). The mAP@50-95 value of 0.854 indicated that the model performed well across a range of stricter IoU thresholds.

The graph in Figure 12a depicts the F1 score as a function of confidence for different classes (licenses, vehicles, and all classes combined). As the confidence increases from 0 to approximately 0.5, the F1 score improves for all classes, reflecting better model performance

as it becomes more confident in its predictions. The highest F1 score was achieved for the license class, approaching a value of near 0.95 at peak confidence. This suggests that the model is highly effective in correctly identifying license plates. The vehicle class shows a lower F1 score, which can be due to overlapping features or more complexity in vehicle objects. The overall F1 score for all classes combined was 0.92 at a confidence level of approximately 0.479. This indicates that the model performed well across both license plates and vehicles at this confidence level. The optimal F1 score for all the classes was reached at a confidence level of approximately 0.479. The graph in Figure 12b is a Precision–Confidence Curve that shows the relationship between precision and confidence. The precision is consistently higher across all confidence thresholds compared to the vehicle starting above 0.8 at low confidence thresholds, and increases as the threshold increases, approaching 1 as the confidence level reaches 1. The aggregated performance across both classes exhibited a precision of 0.951 at a confidence threshold of 1. This means that at the strictest confidence level, approximately 95% of the predictions are correct, resulting in very few false positives. The Precision–Confidence Curve in Figure 12c shows the relationship between the precision and confidence threshold. Recall measures the ability of a model to correctly identify all actual positive instances. The recall value for license plate detection was high across almost all confidence thresholds, achieving a maximum value of 0.976 at a threshold of 0.8, and starts dropping off as the model became stricter in its prediction. This means that the model is very effective in detecting licenses, even at higher confidence levels. The final recall value for the vehicle was 0.876 and the overall recall value was 0.812, indicating that the model maintained a high recall even at a moderately high confidence level. The graph shown in Figure 12d is a Precision–Confidence Curve for the object classes license plate and vehicles, with an average precision of 0.5. License plate recognition performed better than vehicle detection. The overall model achieves an impressive mAP of 0.951 at 0.5 IoU, reflecting high precision and recall across all classes. However, the model performs better on license plates than on vehicles.

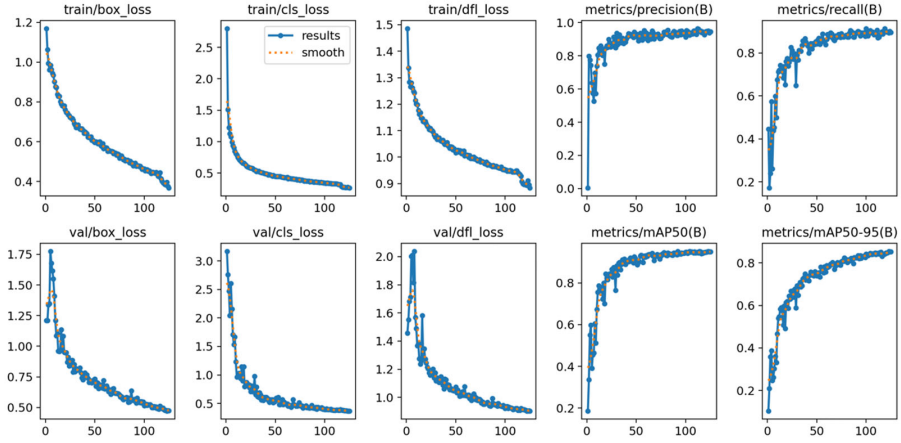


Figure 11. Training and validation results.

The confusion matrix shown in Figure 13 provides an evaluation of the performance of our classification model by comparing the predicted labels with true labels. The diagonal values represent the true positive, indicating that the model correctly identified 98% of license-related objects, 88% of vehicle-related objects, and 86% of background objects. False Positives in the off-diagonal values indicate that 14% of the license-related objects were misclassified as vehicles, which suggests that the model sometimes struggles to distinguish between vehicles and license plates. A total of 12% of the background was incorrectly classified as vehicles, indicating false positives in vehicle classification. Only 2% of the vehicle-related objects were incorrectly classified as license plates, but this was

relatively low. Overall, the model performed very well in identifying license plates, with a high accuracy of 98%. It is difficult to distinguish between vehicles and license plates, as indicated by the 14% misclassification rate. The model also performs well for background detection but can mistakenly classify background objects as vehicles with a 12% error rate.

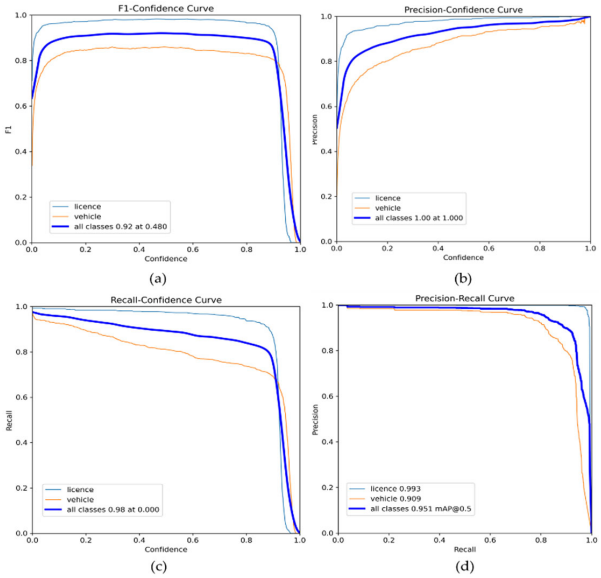


Figure 12. (**a**) F1-Confidence curve, (**b**) Precision–Confidence curve, (**c**) Recall–Confidence curve and (**d**) Precision–Recall curve.

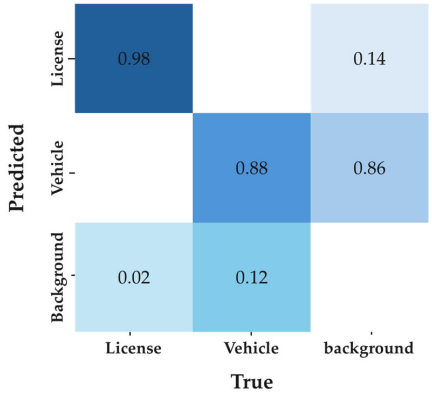


Figure 13. Confusion matrix.

5.2. Performance Comparison

Table 5 provides a comprehensive performance evaluation of YOLOv11 for all license plate and vehicle classes. For the overall detection performance (across all classes combined), YOLOv11 was tested on 594 images containing a total of 1450 instances. The model achieved a high precision of 0.951, indicating its ability to correctly identify true positives with minimal false positives across the dataset. The recall for all classes was also substantial at 0.894, suggesting that YOLOv11 detected nearly 89.4% of the actual objects present in the dataset. The mAP50 for all classes was 0.951, indicating an accurate bounding box localization for most detections. Under more stringent conditions that vary the IoU threshold (mAP50-95), YOLOv11 still achieved a strong score of 0.854, showing consistent performance even when more accuracy is required for bounding box overlap. For the license plate class, YOLOv11 achieved a remarkably high precision of 0.988, and the recall was outstanding at 0.976. The model's mAP50 for license plates reached 0.993, suggesting a near-perfect accuracy in detecting and localizing license plates at a 50% IoU threshold. Even at varying and more demanding IoU levels (mAP50-95), the model maintained a

high score of 0.876, demonstrating its reliability in accurately detecting license plates under stricter conditions. In the vehicle class, YOLOv11 performed well on 582 images, containing 784 instances. The precision of the model for vehicle detection was 0.913, and the recall for vehicles was 0.812. For localization accuracy, YOLOv11's mAP50 for vehicles was 0.91, indicating that the model performed well in accurately placing bounding boxes around vehicles at the 50% IoU threshold. The mAP50-95 score for vehicles was 0.833, demonstrating that the model's precision in detecting vehicles held up well across stricter IoU thresholds, although it was slightly lower than the license plate class. In summary, YOLOv11 exhibited outstanding performance for both license plates and vehicle detection. It achieves near-perfect detection and localization for license plates while also delivering high accuracy for vehicle detection.

Table 5. Performance comparison of YOLOv8 and YOLOv11 for vehicle and license plate detection.

Model	Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
YOLOv8	all	254	432	0.932	0.753	0.808	0.62
	license-plate	97	115	0.923	0.626	0.699	0.502
	vehicle	251	317	0.941	0.88	0.918	0.737
YOLOv11	all	594	1450	0.951	0.894	0.951	0.854
	license-plate	594	666	0.988	0.976	0.993	0.876
	vehicle	582	784	0.913	0.812	0.91	0.833

5.3. Output of the Detection System

The image collage provided in Figure 14 shows a collection of frames from a vehicle detection system based on the YOLOv11 object detection model. This illustrates the ability of the system to identify and localize vehicles in diverse scenarios. Red bounding boxes are drawn around detected vehicles, indicating the system's recognition of objects classified as "vehicles", and confidence scores (e.g., 0.78, 0.54) represent the model's certainty about its predictions, with higher values indicating stronger confidence. It also demonstrates the detection performance of the system under various real-world conditions. It detects vehicles in well-lit environments under low-light conditions at night, congested urban intersections, and open highways, showcasing the system's adaptability. Some of the images show adverse weather, such as sandstorms or fog, thereby testing the robustness of the model. Others capture traffic in both high-density and low-density scenarios, from heavily congested roads to sparsely populated highways, indicating the system's capability to function under varying vehicle densities. Bounding boxes are drawn across each detected image closely aligned with the vehicle edges, indicating precise localization. The system is also capable of detecting multiple vehicles in a single image as evident from some of the images. In some of the highly congested images, the bounding boxes overlap, indicating the difficulties in differentiating between closely packed vehicles. The model struggles to differentiate vehicles from the background due to insufficient contrast under low light and nighttime. The degradation in performance is also visible due to obscured visibility, which reduces the model's ability to detect key features like vehicle edges. This can be overcome by further optimization or using preprocessing techniques. Overall, the image effectively demonstrates the robustness and versatility of the vehicle detection system across a wide range of real-world conditions and scenarios.

Figure 14. Validation and testing batches.

5.4. System Interface and Output

Figure 15 shows the interface and outputs of the adaptive traffic signal control and violation detection system. The setup includes multiple windows displaying the detection results, code execution, and simulated intersections, highlighting the functionality of the system in real-time traffic management applications. The left side of the screen shows a code interface with logs from Python, which shows a preview of an image with detected vehicles on a highway or a multilane road. The red line across the lane acts as a virtual boundary for detecting traffic violations, where vehicles crossing this line in a prohibited manner are flagged. The right side includes the DT with four roads, representing a real-time adaptive control system for intersections.

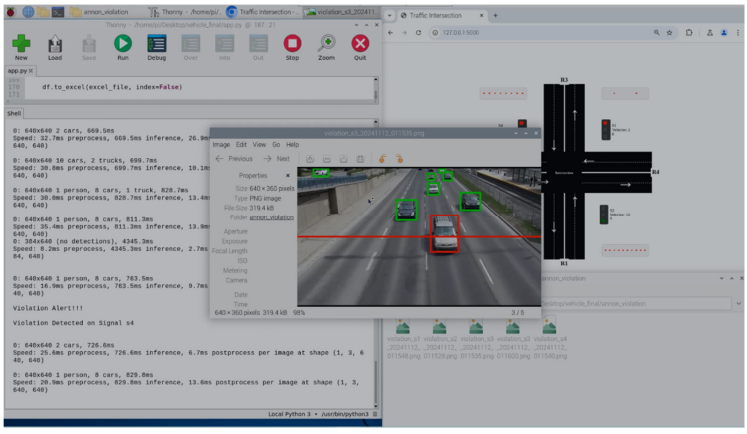
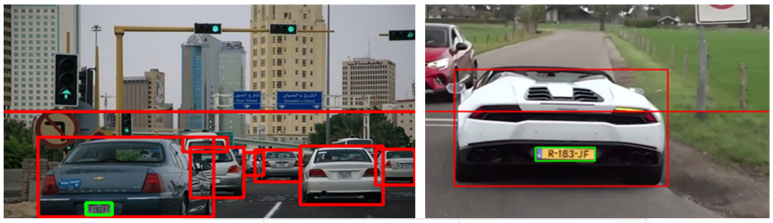


Figure 15. Red light violation detection.

5.5. System Output

The annotated photograph of the video generated after prediction by YOLOv11 provides a clear and visual depiction of the detected objects, as shown in Figure 16. It depicts a congested intersection scene showing multiple vehicles detected by the system, with red

bounding boxes highlighting each vehicle identified. The green box on the vehicles marks the detected license plate of cars violating the allowed region marked by the red line. This setup demonstrates the capability of the system to detect multiple vehicles in real time, enabling it to gather traffic density information at signalized intersections. The figure on the right shows a close-up view of the detected vehicle with a clear view of the license plate. The details of the vehicles committing a signal violation are recorded in an excel file, including the traffic signal location (e.g., "s_1" and "s_4"), the exact time each vehicle was detected, and the recognized license plate numbers for each vehicle. The accurate detection and timestamping of each vehicle makes the system effective for monitoring and managing traffic flow at intersections.



Signal #	Time Stamp	License Plat Number
s1	20241014_145006	R-183-JF
s4	20241014_145008	419297

Figure 16. Detection and OCR results.

We compared the maximum wait time of the vehicles at the intersection in our proposed system with that of a fixed cycle system. Again, we consider a fixed cycle time of 60 s to that of adaptable timings, as discussed in the previous section, given as follows:

- For low congestion, green duration time is 40 s.
- For medium congestion, green duration time is 60 s.
- For higher congestion, green duration time is 90 s.
- Now, consider S1 to be congested, and calculate the wait time for all three scenarios, as shown in Table 6. The wait time for the vehicle at the congested signal is reduced by 43.75% and 56.25% for the medium- and high-congestion cases, respectively. This can be reduced further by increasing the number of congestion levels. The adaptability of the traffic light control system is currently limited to scenarios with a specific number of congestion levels, which captures only a few highly dynamic and diverse traffic patterns.

Table 6. Waiting time comparison of the proposed system with the fixed-cycle system.

Congestion Level	Maximum Waiting Time Fixed Cycle System	Maximum Waiting Time Adaptive System	%Age Reduction in Waiting Time
With no congestion	120–160 s	120–160 s	0%
Medium congestion	280–320 s	120–180 s	43.75%
High congestion	440–480 s	120–210 s	56.25%

5.6. Output of Digital Twin

Figure 17 shows the digital twin of a cyclical traffic signal system, in which each direction is given a turn to proceed while the other directions are stopped. For each signal, two other parameters were displayed: vehicle count and green time. For example, in the current state of DT, signal s_2 detects 10 vehicles in the latest frame before turning green. This is indicated by the number of red dots in the box. This vehicle count at each signal highlights the number of waiting cars, which are used to adjust the timing dynamically

to reduce congestion. The vehicle count and time were scaled to 10 for testing. The actual vehicle count can be considered to be 100, indicating medium congestion, and the green duration time calculated for the system was 60 s. The timer counts until it reaches zero, signal S_3 turns green in the next phase, and its green duration is determined by counting the number of vehicles waiting. This approach allows each direction to periodically clear vehicles, ideally based on the traffic volume or vehicle count at each signal.

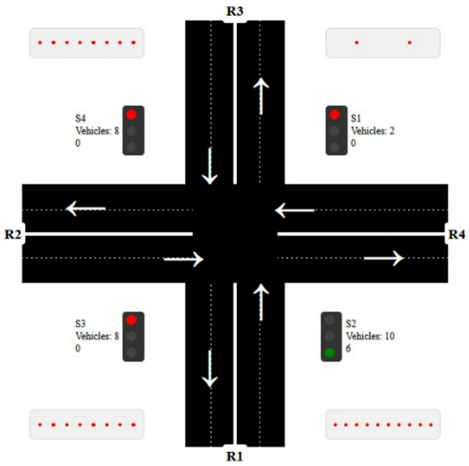


Figure 17. Digital twin of a signalized road intersection.

5.7. Comparison with Existing Work

Researchers have used various performance metrics to report the efficacy of their approach. Many different parameters have been reported for adaptive traffic light control, including the accuracy, precision, mAP, and F1-score. The factors affecting the performance include the algorithm deployed, size of the dataset used for training, training epochs, and class detected. Asha et al. [33] achieved an accuracy of 90–100% when detecting three classes such as bike, car, and bus using YOLO and tested it with a custom dataset. Using YOLOv3, Oltean et al. [35] claimed to achieve an accuracy of 100%, Khan et al. [37] 81.1% for two-wheelers and three-wheelers, Anil et. al [38] 84.45% for five classes. Similarly, Sentosa [40] reported an accuracy of 85.61% using YOLOv4 while detecting five classes. Both [49,51] reported an accuracy of 90% when detecting multiple vehicles using YOLOv5, whereas [48] was able to achieve higher accuracy by ensembling at the cost of more computations. Both [61,64] achieved an accuracy above 90% using YOLOv8 when detecting eight and three classes, respectively. Bidwe and Methaine [30,31] obtained high accuracies of 99.6% and 96, respectively, using CNN. Our proposed system achieved an overall accuracy of 95.1% in clear weather, with the performance dropping to 85% under adverse conditions, such as heavy rain, as shown in Table 7. However, in contrast to the existing solution, our system is also capable of detecting the number of plates of the vehicles.

The adaptive traffic light system operates in real time, making the inference time an important design criterion. This has become even more critical in large-scale traffic management systems that handle high volumes of traffic data from multiple intersections. Shinde et al. [34] deployed the YOLO algorithm and facilitated real-time vehicle counting at a speed of 24 FPS, which is sufficient for such an application. For Oltean [35], on the other hand, leveraging GPU processing significantly enhanced the speed of operation by achieving high accuracy at a speed of 33.5 FPS on real-life traffic videos. Mahmood yielded even more promising results using YOLOv3, boasting the most desired speed of evaluation at 18.1 FPS, and achieved the second-highest accuracy percentage among the compared techniques. Similarly, Rahman [43] achieved a speed of 1.33 FPS, Khan [37] of 0.5 FPS, and [40] that of 0.05 FPS. Our proposed solution achieved an overall speed of up to 5 FPS,

which is sufficient to meet the system requirements, as the volume information needs to be updated every second.

Table 7. Comparison with existing research.

Authors	Object Type	Dataset Name	Detection Algorithm	Accuracy
Oltean et al. [3]	Car, bus, truck	MS COCO	YOLOv3	100%
Darmadi et al. [14]	3 car bus truck	MS COCO	YOLOv8	92–98%
Bidwe et al. [30]	-	Seattle and Washington	CNN	99.60%
Mathiane et al. [31]	Cars, bus, truck, ambulance	Custom	2D-CNN	96%
Khan et al. [37]	2-wheeler, 4-wheeler	MS COCO + Custom	YOLOv3	81.10%
Anil et al. [38]	vehicles	UA-DETRAC	YOLOv3	85.45%
Sentosa et al. [40]	car, bus, truck, bicycle, and motorcycle	MS COCO	YOLOv4	85.6%
Dirir et al. [41]	Multiple	Custom	YOLOv5	96%
Hu et al. [42]	Multiple	6 datasets	YOLOv5	90%
Shirazi et al. [44]	Vehicle, pedestrian	MS COCO	YOLOv5	90%
Srikanteswara et al. [61]	8 different objects	custom	YOLOv8	90.20%
Asha et al. [33]	Bike, car, bus	Custom	YOLO	92–100%
Our approach	Vehicle and number plate	Fused dataset	YOLOv11	95.1%

Furthermore, the system is also ported to Raspberry Pi 4, which is a single-board computer launched by the Raspberry Pi Foundation, as shown in Figure 18. Its compact size and energy efficiency make it suitable for integration into embedded systems [34,57]. This will not only reduce the cost of the system but also make it possible to easily port and install it in any location. The integration of edge computing in the form of deployment of Raspberry Pi not only considerably reduces the latency but also improves the reliability of the system. The optimized use of resources, such as a single camera and edge computing, reduces the carbon footprint [65].

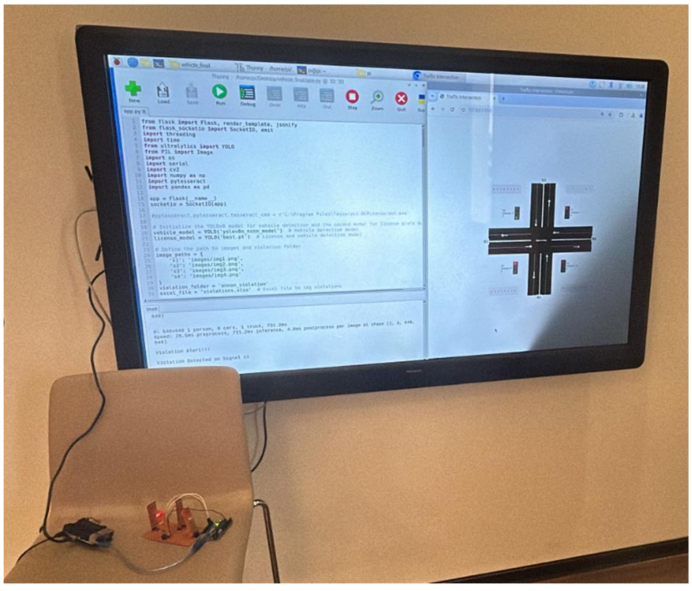


Figure 18. Raspberry Pi implementation.

6. Limitations and Future Scope

Despite promising outcomes, there are limitations to the current system that rely on a single camera for both counting vehicles at a traffic signal and reading license plates, offering several challenges, as they cannot capture both a wide scene for counting and close-up details for plate recognition, resulting in compromises in the field of view that affect accuracy. In addition, adverse weather conditions, such as heavy rain, fog, or sandstorms, may obscure camera vision and reduce detection accuracy. The performance of the system under adverse weather conditions, such as heavy rain or snow, has not been thoroughly validated owing to the limited data availability. Object detection accuracy may degrade when vehicles are occluded or in dense traffic conditions. Additionally, the integration of the digital twin is effective, but limited by the quality and frequency of real-time data input from sensors and cameras.

In the future, we intend to address these limitations by incorporating additional cameras for improved coverage and a higher resolution in license plate recognition. We also plan to enhance the robustness of the system under various environmental conditions using advanced preprocessing techniques and model training with diverse datasets. We intend to enhance the evaluation of performance by analyzing additional metrics, such as intersection throughput, vehicle delay, and fairness of green light allocation. The digital twin can be further expanded to include predictive analytics and simulations based on historical and real-time data to improve traffic management capabilities. In addition, the integration of IoT-enabled sensors with video feeds could provide a more comprehensive understanding of traffic scenarios, ensuring robust and scalable smart city applications. Multi-intersection coordination can be explored using connected vehicle technologies to optimize traffic flow across larger networks. The addition of pedestrian and cyclist detection capabilities can create a more inclusive and safer traffic management system.

7. Conclusions

This study introduces YOLOv11-based object detection with digital twin technology, adapting traffic light timings to optimize traffic flow at intersections. The integration of real-time vehicle detection and license plate recognition within a single framework addresses critical gaps in traditional ITS solutions. By dynamically adjusting timings based on the volume of traffic waiting at intersections and detecting red light violations, the system effectively reduced delays due to congestion as well as traffic violations. We also integrated an automatic license plate recognition system that reads and records the number of vehicles that committed signal violations. The incorporation of YOLOv11 for vehicle detection and Tesseract OCR for license plate recognition resulted in a better accuracy of 95.1% and lower processing latency, making it suitable for real-time applications. A digital twin of the system offers a virtual representation of intersections, providing valuable insights for monitoring, simulation, and optimization. This feature enhances transparency and facilitates decision making for city planners and traffic authorities.

This research lays a solid basis for developing adaptive traffic management systems that are scalable, inclusive, and robust to real-world difficulties by addressing the limitations and exploring the aforementioned future possibilities. The results highlight the importance of combining various datasets, reliable techniques, and state-of-the-art technologies to address the complexity of contemporary traffic systems.

Author Contributions: Conceptualization, M.N.; methodology, M.A. (Manar Ashkanani), A.A. (Alanoud AlAjmi), A.A. (Aeshah Alhayyan), Z.E., M.A. (Mariam AlBedaiwi), and M.N.; software, M.A. (Manar Ashkanani) and Z.E.; validation, M.A. (Manar Ashkanani), A.A. (Alanoud AlAjmi), A.A. (Aeshah Alhayyan), Z.E., and M.A. (Mariam AlBedaiwi); formal analysis, M.A. (Manar Ashkanani),

A.A. (Alanoud AlAjmi) M.A. (Mariam AlBedaiwi), and M.N.; investigation, M.A. (Manar Ashkanani), A.A. (Aeshah Alhayyan), Z.E.; resources, A.A. (Alanoud AlAjmi), A.A. (Aeshah Alhayyan), M.A. (Mariam AlBedaiwi); data curation, M.A. (Manar Ashkanani), A.A. (Alanoud AlAjmi), A.A. (Aeshah Alhayyan), Z.E., M.A. (Mariam AlBedaiwi); writing—original draft preparation, M.A. (Manar Ashkanani), A.A. (Aeshah Alhayyan), Z.E., and M.N.; writing—review and editing M.A. (Manar Ashkanani), A.A. (Alanoud AlAjmi), M.N., Z.E.; visualization, A.A. (Alanoud AlAjmi), A.A. (Aeshah Alhayyan), Z.E., M.A. (Mariam AlBedaiwi), and M.N.; supervision, M.N.; project administration, M.A. (Manar Ashkanani), Z.E., M.A. (Mariam AlBedaiwi), and M.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rashed, A.H. The impacts of unsustainable urbanization on the environment. In *Sustainable Regional Planning*; IntechOpen: London, UK, 2023.
2. AlRukaibi, F. The impact of congestion charging technique on traffic flow and atmospheric pollution in Kuwait city. *J. Eng. Res.* **2021**, *9*, 51–62. [CrossRef]
3. Lakhouili, A.; Essoufi, E.H. Cooperative Traffic Signal Control of n-intersections Using a Double Deep Q-Network Agent. *Int. J. Intell. Eng. Syst.* **2022**, *15*, 555.
4. Yousef, K.M.A.; Shatnawi, A.; Latayfeh, M. Intelligent traffic light scheduling technique using calendar-based history information. *Future Gener. Comput. Syst.* **2019**, *91*, 124–135. [CrossRef]
5. Kan, Z.; Tang, L.; Kwan, M.-P.; Ren, C.; Liu, D.; Li, Q. Traffic congestion analysis at the turn level using Taxis' GPS trajectory data. *Comput. Environ. Urban Syst.* **2019**, *74*, 229–243. [CrossRef]
6. Zulfikar, M.T. Detection traffic congestion based on Twitter data using machine learning. *Procedia Comput. Sci.* **2019**, *157*, 118–124. [CrossRef]
7. Joyo, A.; Yaqub, R.; Madamopoulos, N. Intelligent traffic-lights management by exploiting smart antenna technology (ITSAT). *IEEE Intell. Transp. Syst. Mag.* **2020**, *13*, 154–163. [CrossRef]
8. Yadav, S.; Singh, S.; Chaurasiya, V.K. Traffic light control using RFID and deep reinforcement learning. In *AI and IoT for Smart City Applications*; Springer Nature: Berlin/Heidelberg, Germany, 2022; pp. 47–64.
9. Singh, M.K.; Mishra, K.D.; Sahana, S. An intelligent real-time traffic control based on vehicle density. *Int. J. Eng. Technol. Manag. Sci. Website Ijetms.* **2021**, *5*, 24–29.
10. Yang, B.; Zhang, H.; Du, M.; Wang, A.; Xiong, K. Urban traffic congestion alleviation system based on millimeter wave radar and improved probabilistic neural network. *IET Radar Sonar Navig.* **2023**, *18*, 327–343. [CrossRef]
11. Rida, N.; Ouadoud, M.; Hasbi, A. Traffic signal control for a single intersection-based intelligent transportation system. In *Digital Transformation and Innovative Services for Business and Learning*; IGI Global: Hershey, PA, USA, 2020; pp. 159–180.
12. AlMulla, K.; Ashkanani, S.; Hassan, F.; AlMesbahi, H.; Alazmi, M.; Nadeem, M. IoT-based Adaptive Traffic Signal Controller to Optimize the Flow of Traffic and Reduce Congestion. In Proceedings of the 2024 Mediterranean Smart Cities Conference (MSCC), Tetuan, Morocco, 2–4 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1–6.
13. Chaware, S.; Chaware, T. Proposed algorithm for smart traffic control using ultrasonic sensors. *Int. J. Eng. Adv. Technol.* **2019**, *8*, 3912–3915. [CrossRef]
14. Zou, Z.; Chen, K.; Shi, Z.; Guo, Y.; Ye, J. Object detection in 20 years: A survey. *Proc. IEEE* **2023**, *111*, 257–276. [CrossRef]
15. Bradski, G. The opencv library. *Dr. Dobb's J. Softw. Tools Prof. Program.* **2000**, *25*, 120–123.
16. Rani, P.; Kumar, R.; Jain, A.; Lamba, R. Taxonomy of machine learning algorithms and its applications. *J. Comput. Theor. Nanosci.* **2020**, *17*, 2508–2513. [CrossRef]
17. Sultan, F.; Khan, K.; Shah, Y.A.; Shahzad, M.; Khan, U.; Mahmood, Z. Towards automatic license plate recognition in challenging conditions. *Appl. Sci.* **2023**, *13*, 3956. [CrossRef]
18. Mahmood, Z.; Khan, K.; Khan, U.; Adil, S.H.; Ali, S.S.A.; Shahzad, M. Towards automatic license plate detection. *Sensors* **2022**, *22*, 1245. [CrossRef]
19. Gao, J.; Shen, Y.; Liu, J.; Ito, M.; Shiratori, N. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv* **2017**, arXiv:1705.02755.

20. Wath, A.K.; Daigavane, P.M.; Naidu, H.K. A Technique of Image and Fuzzy Logic to Solve Traffic Congestion at Signal. In Proceedings of the 2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON), Bangalore, India, 23–25 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6.
21. Fahrunnisa, Z.; Rahmadwati, R.; Setyawan, R.A. Adaptive Traffic Light Signal Control Using Fuzzy Logic Based on Real-Time Vehicle Detection from Video Surveillance. *J. Ilm. Tek. Elektro Komput. Dan Inform.* **2024**, *10*, 235–251. [CrossRef]
22. Meepokgit, T.; Wisayataksin, S. Traffic Signal Control with State-Optimizing Deep Reinforcement Learning and Fuzzy Logic. *Appl. Sci.* **2024**, *14*, 7908. [CrossRef]
23. Mohamed, N.E.; Radwan, I.I. Traffic light control design approaches: A systematic literature review. *Int. J. Electr. Comput. Eng.* **2022**, *12*, 5355–5363. [CrossRef]
24. Mannion, P.; Duggan, J.; Howley, E. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic Road Transport Support Systems*; Springer Nature: Berlin/Heidelberg, Germany, 2016; pp. 47–66.
25. Ma, D.; Zhou, B.; Song, X.; Dai, H. A deep reinforcement learning approach to traffic signal control with temporal traffic pattern mining. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 11789–11800. [CrossRef]
26. Tan, K.L.; Poddar, S.; Sarkar, S.; Sharma, A. Deep reinforcement learning for adaptive traffic signal control. In Proceedings of the Dynamic Systems and Control Conference, American Society of Mechanical Engineers, Park City, UT, USA, 8–11 October 2019; p. V003T18A006.
27. Damadam, S.; Zourbakhsh, M.; Javidan, R.; Faroughi, A. An Intelligent IoT Based Traffic Light Management System: Deep Reinforcement Learning. *Smart Cities* **2022**, *5*, 1293–1311. [CrossRef]
28. Tunc, I.; Soylemez, M.T. Fuzzy logic and deep Q learning based control for traffic lights. *Alex. Eng. J.* **2023**, *67*, 343–359. [CrossRef]
29. Rao, S.G.; RamBabu, R.; Kumar, B.S.A.; Srinivas, V.; Rao, P.V. Detection of Traffic Congestion from Surveillance Videos using Machine Learning Techniques. In Proceedings of the 2022 Sixth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Dharan, Nepal, 10–12 November 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 572–579.
30. Bidwe, S.; Kale, G.; Bidwe, R. Traffic monitoring system for smart city based on traffic density estimation. *Indian J. Comput. Sci. Eng.* **2022**, *13*, 1388–1400. [CrossRef]
31. Mathiane, M.J.; Tu, C.; Adewale, P.; Nawej, M. A vehicle density estimation traffic light control system using a two-dimensional convolution neural network. *Vehicles* **2023**, *5*, 1844–1862. [CrossRef]
32. Chaudhuri, A. Smart Traffic Management of Vehicles using Faster R-CNN based Deep Learning Method. *arXiv* **2023**, arXiv:2311.10099. [CrossRef]
33. Asha, C.S.; Narasimhadhan, A.V. Vehicle counting for traffic management system using YOLO and correlation filter. In Proceedings of the 2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 16–17 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
34. Shinde, P.; Yadav, S.; Rudrake, S.; Kumbhar, P. Smart traffic control system using YOLO. *Int. Res. J. Eng. Technol* **2019**, *6*, 169–172.
35. Oltean, G.; Florea, C.; Orghidan, R.; Oltean, V. Towards real time vehicle counting using yolo-tiny and fast motion estimation. In Proceedings of the 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME), Sibiu, Romania, 16–19 October 2024; IEEE: Piscataway, NJ, USA, 2019; pp. 240–243.
36. Mahmood, M.T.; Ahmed, S.R.A.; Ahmed, M.R.A. Detection of vehicle with Infrared images in Road Traffic using YOLO computational mechanism. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Sozopol, Bulgaria, 10–13 September 2020; IOP Publishing: Bristol, UK, 2020; p. 22027.
37. Khan, H.; Kushwah, K.K.; Maurya, M.R.; Singh, S.; Jha, P.; Mahobia, S.K.; Soni, S.; Sahu, S.; Sadasivuni, K.K. Machine learning driven intelligent and self adaptive system for traffic management in smart cities. *Computing* **2022**, *104*, 1203–1217. [CrossRef]
38. Anil, J.M.; Mathews, L.; Renji, R.; Jose, R.M.; Thomas, S. Vehicle Counting based on Convolution Neural Network. In Proceedings of the 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 17–19 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 695–699.
39. Sakhare, N.; Hedau, M.; Gokul, B.; Malpure, O.; Shah, T.; Ingle, A. Smart Traffic: Integrating Machine Learning, and YOLO for Adaptive Traffic Management System. *Int. J. Intell. Syst. Appl. Eng.* **2024**, *12*, 347–355.
40. Sentosa, E.P.; Gunawan, R.J.; Nurhasanah, N.; Irwansyah, E. Employing YOLOv4 in a Vehicle Counting System Using Random Dataset. In Proceedings of the 2022 5th International Conference on Information and Communications Technology (ICOIACT), Virtually, 24–25 August 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 462–467.
41. Dirir, A.; Ignatious, H.; Elsayed, H.; Khan, M.; Adib, M.; Mahmoud, A.; Al-Gunaid, M. An advanced deep learning approach for multi-object counting in urban vehicular environments. *Future Internet* **2021**, *13*, 306. [CrossRef]
42. Hu, Z.; Lam, W.H.K.; Wong, S.C.; Chow, A.H.F.; Ma, W. Turning Traffic Monitoring Cameras into Intelligent Sensors for Traffic Density Estimation. *arXiv* **2021**, arXiv:2111.00941.
43. Rahman, R.; Azad, Z.B.; Hasan, M.B. Densely-populated traffic detection using yolov5 and non-maximum suppression ensembling. In *Proceedings of the International Conference on Big Data, IoT, and Machine Learning: BIM 2021*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 567–578.

44. Shirazi, M.S.; Morris, B.T.; Zhang, S. Intersection analysis using computer vision techniques with SUMO. *Intell. Transp. Infrastruct.* **2023**, *2*, liad003. [CrossRef]
45. Kunekar, P.; Narule, Y.; Mahajan, R.; Mandlapure, S.; Mehendale, E.; Meshram, Y. Traffic Management System Using YOLO Algorithm. *Eng. Proc.* **2024**, *59*, 210. [CrossRef]
46. Li, H. Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs. *arXiv* **2016**, arXiv:1601.05610.
47. Sharma, N.; Haq, M.A.; Dahiya, P.K.; Marwah, B.R.; Lalit, R.; Mittal, N.; Keshta, I. Deep Learning and SVM-Based Approach for Indian Licence Plate Character Recognition. *Comput. Mater. Contin.* **2023**, *74*, 27899. [CrossRef]
48. Arora, S.; Mittal, R.; Arora, D.; Shrivastava, A.K. A Robust Approach for Licence Plate Detection Using Deep Learning. *Intel. Artif.* **2024**, *27*, 129–141. [CrossRef]
49. Shahrear, R.; Rahman, M.A.; Islam, A.; Dey, C.; Zishan, M.S.R. An automatic traffic rules violation detection and number plate recognition system for Bangladesh. *AIUB J. Sci. Eng. (AJSE)* **2020**, *19*, 87–98. [CrossRef]
50. Al-Hasan, T.M.; Bonnefille, V.; Bensaali, F. Enhanced YOLOv8-Based System for Automatic Number Plate Recognition. *Technologies* **2024**, *12*, 164. [CrossRef]
51. Hughes, A. Forging the digital twin in discrete manufacturing, a vision for unity in the virtual and real worlds. *LNS Research e-book*, 2018.
52. Making, E.D. *Economics of Digital Twins*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2024.
53. Dasgupta, S.; Rahman, M.; Jones, S. Harnessing Digital Twin Technology for Adaptive Traffic Signal Control: Improving Signalized Intersection Performance and User Satisfaction. *IEEE Internet Things J.* **2024**, *11*, 36596–36618. [CrossRef]
54. Romero-García, R.; Martínez-Tomás, R.; Pozo, P.; de la Paz, F.; Sarriá, E. Q-CHAT-NAO: A robotic approach to autism screening in toddlers. *J. Biomed. Inform.* **2021**, *118*, 103797. [CrossRef]
55. Darmadi, D.; Pratikso, P.; Rahmat, M. Traffic Counting using YOLO Version-8 (Case Study of Jakarta-Cikampek Toll Road). *ASTONJADRO* **2024**, *13*, 115–124. [CrossRef]
56. Kenk, M.A.; Hassaballah, M. Dawn: Vehicle detection in adverse weather nature dataset. *arXiv* **2020**, arXiv:2008.05402.
57. Farag, W.; Nadeem, M. Enhanced real-time road-vehicles' detection and tracking for driving assistance. *Int. J. Knowl.-Based Intell. Eng. Syst.* **2024**, *28*, 335–357. [CrossRef]
58. Srikanteswara, R.; Hegde, A.; Hegde, P.T. Emergency Vehicle Recognition Via Automated Smart Traffic Manager. In Proceedings of the 2023 International Conference on Evolutionary Algorithms and Soft Computing Techniques (EASCT), Bengaluru, India, 20–21 October 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–6.
59. Xu, Z.; Yang, W.; Meng, A.; Lu, N.; Huang, H.; Ying, C.; Huang, L. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 255–271.
60. Agrawal, S. Global License Plate Dataset. *arXiv* **2024**, arXiv:2405.10949.
61. Thilan. Vehicle Detection with YOLO V11 Dataset. Roboflow Universe. 2024. Available online: https://universe.roboflow.com/thilan/vehicle-detection-with-yolo-v11 (accessed on 28 December 2024).
62. Kendaraan, P. Vehicle and license plate Dataset. Roboflow Universe. 2023. Available online: https://universe.roboflow.com/plat-kendaraan/vehicle-and-license-plate (accessed on 28 December 2024).
63. Rai, K.M.; Vijayalakshmi, S. Density Based Traffic Light Control Using IR Sensors and Arduino. *Int. J. Res. Appl. Sci. Eng. Technol.* **2023**, *11*, 777–783. [CrossRef]
64. Liu, B.; Lam, C.-T.; Ng, B.K.; Yuan, X.; Im, S.K. A Graph-based Framework for Traffic Forecasting and Congestion Detection using Online Images from Multiple Cameras. *IEEE Access* **2024**, *12*, 3756–3767. [CrossRef]
65. Alhaila, S.; Almelhem, I.; Alsnafi, A.; Alameerah, M.; Alrumaidhi, K.; Nadeem, M. YOLO-Based Helmet Detection System for Safety Compliance in Oil and Gas Industry. In Proceedings of the 2024 5th International Conference on Industrial Engineering and Artificial Intelligence (IEAI), Bangkok, Thailand, 24–26 April 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 16–23.