AI-Based Smart Traffic Police Constable

Prof.Ganesh Sarmokaddam , Mangesh Upadhyay ,Jayesh Vyankatkar ,Shrawan Waghmare

Department of Electronics and Communication
Priyadarshini College of Engineering Nagpur, Maharashtra , India Sarmokaddam.ganesh@gmail.com

• ***Abstract*** Road traffic enforcement depends heavily on human officers who cannot be present at every junction at every hour, and manual monitoring suffers from fatigue, uneven coverage and inconsistent record keeping. This paper presents an AI-based smart traffic police constable, a desktop prototype that assists traffic officers, rather than replacing them, by automatically screening images and live camera frames for selected two-wheeler safety violations. The system pairs a YOLO-family object detector with a Tkinter dashboard built on OpenCV and Pillow. The implementation performs detection only. It does not track vehicles, read number plates, keep a database or establish legal liability. Intended applications include assisted monitoring at campus gates, checkpoints and traffic-control rooms, with every enforcement decision verified by authorised personnel.

Keywords: traffic enforcement; YOLO; helmet detection; computer vision; road safety; intelligent transportation systems

1. Introduction

2.1 Road Safety and Intelligent Traffic Management Road traffic is a shared system in which the behaviour of each rider and driver affects everyone nearby. When the rules that govern it, such as wearing a helmet, limiting the number of riders on a motorcycle, stopping at a signal and parking in designated places, are followed inconsistently, the cost is paid in injuries, delays and enforcement workload. Helmets illustrate the point well. They are the main protective equipment for a motorcycle rider, yet enforcing their use is described in the literature as a time-consuming and labour-intensive task [3]. Automated video-based detection is therefore attractive, because it could support education and enforcement campaigns at a scale that manual observation cannot reach [8]. Intelligent traffic management, in the broad sense used here, applies sensing and computation to observe road conditions, flag unsafe behaviour and give decision-makers timely information.

2.2 Common Urban Traffic Problems Cities face a cluster of related problems: congestion at busy junctions, signal and stop-line violations, improper or obstructive parking, riding.

without a helmet, carrying more than the permitted number of riders on a two-wheeler (commonly called triple riding), and wrong-side driving. These problems reinforce one another. Congestion raises the temptation to jump a signal, illegal parking narrows lanes and worsens congestion, and a thinly stretched enforcement presence lets all of them continue. This project addresses the subset that can be screened visually from a single camera view with an object detector: helmet non-use and triple riding on two-wheelers, together with supporting object categories.

2.4 Artificial Intelligence in Modern Transportation Machine learning has become a standard tool in transportation research, from motorcycle detection in urban video [1] to licence plate recognition [10]. Learning-based systems can generalise across variations in appearance that hand-written rules cannot anticipate, provided they are trained on representative data.

2.5 Computer Vision and Machine Learning in Traffic Monitoring Early helmet-detection systems combined motion segmentation with hand-crafted features and classical classifiers [2], [3]. Convolutional neural networks later took over feature extraction, and single-stage detectors such as YOLO reframed detection as a regression problem solved by one network in a single pass, which makes near-real-time operation feasible on modest hardware [5]. A detector of this kind receives an image and returns, for every object it finds, a bounding box, a class label and a confidence score. Those three outputs are the raw material from which the dashboard in this paper builds counts, status indicators and alerts.

2.6 Scope and SignificanceThe scope is single-camera, single-frame detection on a desktop computer, with image upload and webcam input. Vehicle tracking, number plate recognition, database storage, signal-violation or parking - violation detection and integration with official systems are outside the implemented scope and are discussed as future work. The significance of the project lies in showing a complete detection-to-alert pipeline built from freely available tools, and in providing a base that can be extended in the directions described in Section 6.

2.7 Main Contributions1. An integrated pipeline that takes an image or webcam frame through detection, on-screen presentation, evidence saving and e-mail notification.2. A code-level analysis of the prototype, including limitations found during review, such as the class-name mismatch described in Section 5.5.3. A clear separation of vehicle detection, tracking, number plate recognition and violation determination, together with the additional logic and data each would need.4. A proposed evaluation protocol that distinguishes measured results from expectations.

2. Related Work

- This review is organised around the capabilities the prototype touches. Only sources whose bibliographic details could be checked are cited. Where a method is described here on the strength of another paper's summary, the text says so.

2.1 Helmet Detection with Classical Computer Vision Chiverton [3] trained support vector machines on histograms computed from head-region images of motorcycle riders. In the video pipeline, riders are segmented by background subtraction, their heads are isolated, and each rider's track is labelled by averaging the per-frame classifier outputs, which reduces sensitivity to a single poor frame. The study reports accurate helmet classification on static photographs and describes tracking tests that support the approach; its abstract gives no numerical results, so none are repeated here. Dahiya et al. [2] also began with background subtraction to find riders and then applied visual features and a binary classifier, as summarised in [9]. The weaknesses of this family, noted in [9], are that multi-stage pipelines struggle to run in real time, that motion segmentation degrades with congestion, camera shake and moving foliage, and that classifiers struggle when an unhelmeted passenger is partly hidden by a helmeted rider. The present project avoids background subtraction by detecting objects directly in each frame. That suits still images and webcam frames, but it gives up the temporal evidence that tracking provides.

2.2 Deep-Learning Helmet and Rider Analysis Vishnu et al. [4], as summarised in [9], used adaptive background subtraction to extract moving objects, a convolutional network to identify motorcyclists among them, and a further network on the upper part of each motorcyclist to find riders without helmets. Chairat et al. [6] and Saumya et al. [7] moved to learned object detectors. According to [9], approaches of the first kind detect the motorcycle region and classify its upper part, and approaches of the second kind detect motorcycles and people, infer riders from bounding-box overlap and then detect helmet use. The title of [7] shows that it also targets triple riding, the second violation type handled by our prototype. Lin et al. [8] addressed a harder problem: tracking individual motorcycles across frames and registering helmet use for each rider. They released the HELMET dataset of 91,000 annotated frames of 10,006 motorcycles from 12 observation sites in Myanmar, reached more than 8 frames per second on consumer hardware, and obtained a weighted F-measure of 67.3% for counting riders and helmet use of tracked motorcycles. That moderate figure is a useful reminder that rider-level helmet analysis in real footage is far from solved. Tran et al. [9] built a two-stage framework in which a YOLOv8 detector localises motorcycles with their riders and a second YOLOv8 identifier classifies each rider and helmet state, and ranked detections by the product of the two confidence scores.

2.3 They scored 0.7754 on the public leaderboard of the AI City 2023 Challenge Track 5, and they reported that illumination changes and haze produce false detections and that the best confidence threshold varied between datasets. Their finding that locating the whole motorcycle with its riders first works better than detecting every part at once is directly relevant to the design limitations discussed in Section 5.5.

2.4 Deep-Learning Object Detection Redmon et al. [5] introduced YOLO, which frames detection as regression from a full image to spatially separated bounding boxes and class probabilities using one network. Their base model ran at 45 frames per second; the authors acknowledged more localisation errors than the leading detectors of that time but fewer false positives on background. Ultralytics' YOLO26 [11] continues this lineage with an end-to-end design that removes the non-maximum-suppression post-processing step and the distribution focal loss module, aiming at faster CPU

inference on edge devices. The vendor reports 40.9 mAP on COCO validation at 640 pixels, 38.9 ms CPU inference with ONNX and 2.4 million parameters for the nano model. These figures are vendor-reported on COCO, not on traffic data, and no formal YOLO26 research paper has been published, so they indicate scale rather than expected accuracy for helmet detection. The prototype uses the Ultralytics Python package [12] to load and run its model.

2.5 Motorcycle Detection Reviews and Broader Intelligent Transportation Work Espinosa et al. [1] reviewed motorcycle detection in urban traffic video, covering both traditional and deep-learning methods, as summarised in [9]. We did not verify peer-reviewed sources on signal-violation detection, illegal-parking detection or traffic-control-room dashboards, so this paper makes no literature-based claims in those areas. The corresponding project features are treated only as proposals, and the authors should add verified references there before submission if they wish to expand those themes.

3.6 Comparison of Reviewed Studies

Author and Year	Research Focus	Method Used	Main Findings	Limitations
Espinosa et al., 2021 [1]	Motorcycle detection in urban traffic	Review of traditional and deep-learning video methods (as summarised in [9])	Organises the field into traditional and learning-based groups	A survey; no deployable system
Chiverton, 2012 [3]	Helmet classification with tracking	Background subtraction, head-region histograms, SVM, track-level averaging	Accurate on static photographs; tracking tests support the approach	Depends on background subtraction; abstract gives no numbers
Dahiya et al., 2016 [2]	Bike riders without helmets in surveillance video	Background subtraction, features, binary classifier (per [9])	Pipeline aimed at real-time operation	Multi-stage; sensitive to congestion and camera shake (per [9])
Vishnu et al., 2017 [4]	Motorcyclists without helmets in video	Adaptive background subtraction, CNN classification of motorcyclists and upper region (per [9])	Learned features for the helmet decision	Foreground extraction remains classical; weak in crowded scenes (per [9])
Chairat et al., 2020 [6]	Low-cost helmet violation detection	Motorcycle region detection, then upper-part classification (grouping per [9])	Details not independently verified	Weak when several riders share a motorcycle (per [9])
Saumya et al., 2020 [7]	Helmet and triple-ride detection	YOLOv3 rider inference by box overlap, then helmet detection (per [9])	Targets both helmet use and triple riding	Details not independently verified
Lin et al., 2020 [8]	Helmet use of tracked motorcycles	CNN multi-task learning; HELMET dataset (91,000 frames, 10,006 motorcycles)	Over 8 FPS; weighted F-measure 67.3%	Moderate accuracy; data from sites in one country

3. *Proposed System*

3.1 System Overview

The proposed system is a desktop application that acts as a digital assistant to a traffic police officer. It examines an image or a live camera frame, locates road users, and flags two rider-safety conditions: a rider without a helmet and a motorcycle carrying three people (triple riding). When either condition is detected, the officer-facing dashboard
turns to a violation status and an e-mail alert with the annotated picture is sent to a configured recipient. Based on the uploaded project, the system is best described as a prototype desktop application with image-upload and webcam modes. It is not a networked, multi-camera, or production enforcement platform. It runs on a single computer (the README gives Windows instructions) with Python, a trained detection model file, and optionally a webcam and internet access for the e-mail alert.

3.2 System Architecture

The architecture is a single-process pipeline in which a graphical front end calls a finetuned detector and then applies a simple rule to its output. Figure 1 shows the data flow;
every block corresponds to code in dashboard.py .

3.3 Working Principle

1. The operator starts the application. It loads models/best.pt and shows MODEL READY or MODEL ERROR.
2. The operator either uploads an image or starts the camera.

AI-Based Smart Traffic Police Constable - Proposed System, Implementation, Conclusion and References
Page 2 of 13

3. The image or frame is passed to the YOLO detector with the chosen confidence threshold (default 0.40).
4. The program draws the boxes on the image, counts detections per class and computes the mean confidence.
5. A rule checks whether any detected label corresponds to a no-helmet or triple-seat violation.
6. If so, the status banner shows VIOLATION DETECTED and a background thread emails the annotated image, subject to a cooldown. Otherwise the banner shows SAFE.
7. The annotated result is written to the results folder with a timestamped file name.

Component	Status	Remarks
Computer (Windows PC or laptop)	Confirmed	README gives Windows instructions; nano-scale model is small (about 5.4 MB), so a CPU may suffice for images
Webcam or USB camera	Optional	Needed only for live mode (camera index selectable)
GPU	Optional, not required by code	Would improve live-frame rate; no GPU-specific code is present
Internet connection	Optional	Required only for e-mail alerts
Roadside IP/CCTV camera, edge board	Proposed	Not implemented

Software	Status	Role
Python 3	Confirmed	Implementation language
`ultralytics`	Confirmed (requirements.txt)	YOLO model loading and inference; also installs PyTorch
`opencv-python`	Confirmed (requirements.txt)	Camera access, frame handling, image writing
`Pillow`	Confirmed (requirements.txt)	Frame conversion for display
Tkinter, `smtplib`, `threading`, `pathlib`	Confirmed (standard library)	GUI [4], e-mail, background work, file paths
SMTP account (Gmail on port 587 in code)	Optional	Alert delivery
Trained weights `models/best.pt`	Confirmed	The detector

4.4 AI and Computer Vision Techniques

Object detection with YOLO. The core technique is single-stage object detection in the YOLO family, which predicts boxes and class probabilities in one forward pass of a convolutional network and is therefore fast enough for near-real-time use [1]. The project uses the Ultralytics implementation [2].

Inspection of the metadata stored inside best.pt shows a detection-task checkpoint built from the yolo26n.yaml architecture file and initialised from yolo26n.pt , that is, a nano-scale model. The architectural details of that model should be taken from the official Ultralytics documentation [2] rather than assumed. Confidence thresholding and duplicate suppression.

Detections below a user-adjustable confidence threshold are removed. Ultralytics also applies its own post-processing to handle overlapping boxes; the classical form of this step is non-maximum suppression [5]. The dataset itself is not part of the ZIP, so its size, annotation source and class balance cannot be reported Techniques that are not used. The project contains no object tracking, no number-plate detection or OCR, no signal-state recognition, no speed estimation and no face recognition. These are discussed only as future work.

5. *Implimantation and software required*



5.1 Development Environment

The application is a single Python file (dashboard.py , about 1,700 lines) organised as one class, Traffic Safety Dashboard . The folder layout is: dashboard.py , requirements.txt , README.txt , models/best.pt , input/ (one sample image), and results/ (ten saved annotated images from testing).

Dependencies are installed with pip install -r requirements.txt and the program is started with python dashboard.py . The interface is built with Tkinter [4], with a dark theme and a 1450 x 850 pixel default window.

5.2 System Workflow

1. Define paths, e-mail settings, alert classes and a 60-second e-mail cooldown.
2. Create the main window, load the model, and build the header, statistics cards, detection view, control bar and status bar.
3. Wait for user action: Upload Image, Start Camera, Save Result, Reset, Stop, or Test SMTP.
4. On detection, run inference, update the display, evaluate the violation rule, and optionally send an alert. 5. On closing the window, release the camera and exit.

5.3 Input Acquisition and Processing Image upload. A file dialog accepts JPG, JPEG, PNG and WEBP files. Detection runs in a separate thread so that the window does not freeze during inference. Results are returned to the interface thread through Tkinter's after mechanism, which is the AI-Based Smart Traffic Police Constable - Proposed System,

Implementation, Conclusion and References Page 6 of 13 correct way to update widgets from a worker thread. Camera. The user enters a camera index and presses Start Camera. OpenCV opens the device and the program reads one frame, runs inference, updates the display, and schedules itself again after 10 ms. No video-file input is provided; only still images and a live camera are supported. Inference in this mode runs in the interface thread, so slow inference reduces interface responsiveness. The live-mode function also writes an annotated JPEG for every processed frame; because file names are timestamped to the second, this produces roughly one file per second and will consume disk space over long sessions.

5.4 AI Model Integration The model is loaded once with YOLO(str(MODEL_PATH)) , and class names are read from the model rather than hard-coded. Inference uses model . predict(source, conf=threshold, save=False, verbose=False) . For each result, the program iterates over result . boxes , reading the class index, the confidence score and the label name; the annotated image is produced by result . plot() , which draws bounding boxes with class labels and confidences. Class names are normalised (lower case; underscores and hyphens replaced by spaces) before the dashboard logic uses them, so no-helmet becomes no helmet and Person and person are both counted as person.

5. Dashboard and User Interface

The window has five regions. The header shows the title and a model status indicator. The statistics row has six cards (Person, Bike, Sunroof, Helmet, No Helmet, Triple Seat). The detection view shows the annotated image or live video. The live detections panel lists each detection with its class and confidence percentage and shows total detections, mean confidence, and a SAFE or VIOLATION status. The control bar provides Upload Image, Start Camera, Stop, Save Result, Reset, a camera-index box, a confidence slider from 10% to 90% (default 40%), and a Test SMTP button. A status bar reports progress and errors. Note that the dashboard has no counters for the model's car and scooty classes, which are detected and drawn but not summarised.

5.6 Testing andEvaluation Evidence available.

The project contains ten annotated outputs from informal trial runs and no quantitative evaluation: no labelled test set, no precision or recall, no frame-rate measurement. Therefore no accuracy figures are reported in this paper. Those results should be treated as qualitative demonstrations only. Proposed methodology.

(1) Build a held-out test set of annotated images that was not used for training, covering day, night, glare, crowded scenes and several camera angles.

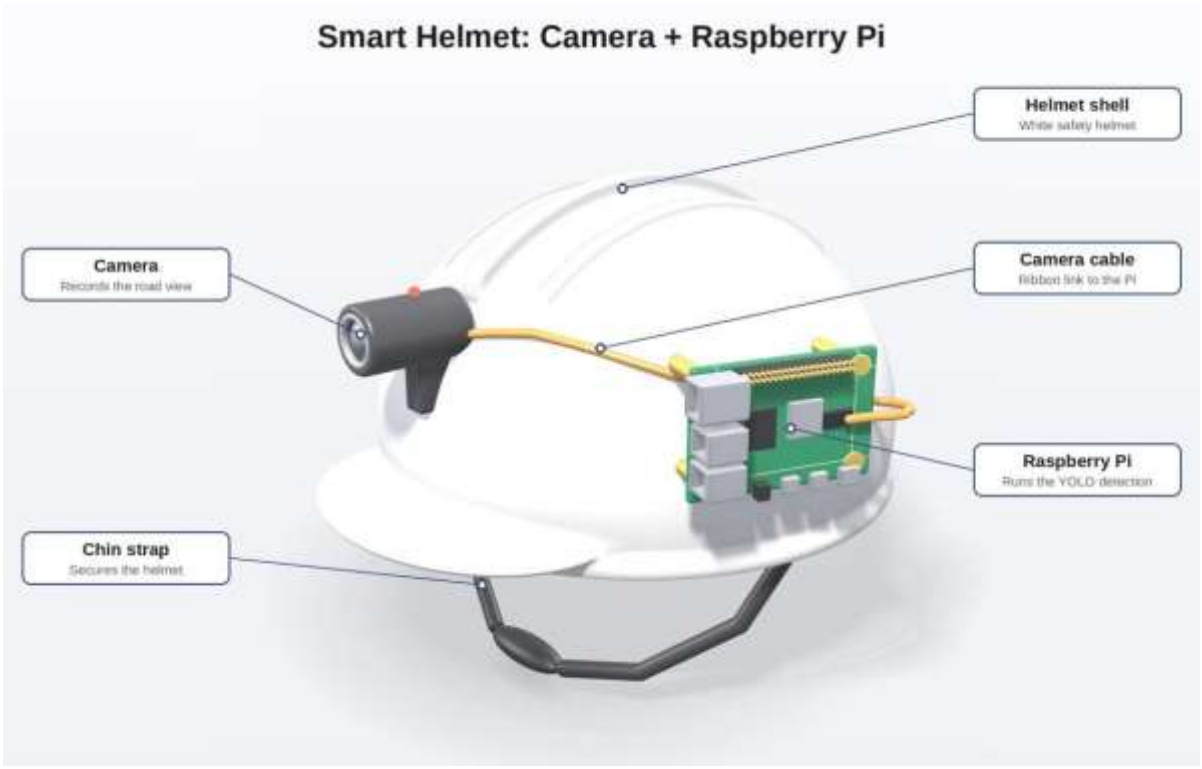
(2) Run the model at a fixed threshold and match predictions to ground truth with an Intersection-over-Union (IoU) criterion, commonly 0.5 [10], where B_p is the predicted box and B_g the ground-truth box:
\mathrm{IoU}=\frac{|B_p \cap B_g|}{|B_p \cup B_g|}

(3) Compute, per class, precision (P), recall (R) and F1-score from true positives (TP), false positives (FP) and false negatives (FN): P=\frac{TP}{TP+FP},\qquad R=\frac{TP}{TP+FN},\qquad F_1=\frac{2PR}{P+R} and the average precision (AP) as the area under the precision-recall curve, with mAP as its mean over classes [10], [11].

(4) Measure inference time per frame (mean and 95th percentile) on the intended hardware, and report the frames per second of the full pipeline including display.

(5) Evaluate at the event level: of the real no-helmet and tripleriding events, how many raised an alert, and how many alerts were false?

(6) Test the alert path (SMTP) separately, including the cooldown behaviour. Result classification. In the final version of the paper, results should be labelled as measured (from the test above), illustrative (the saved sample outputs) or expected (design intentions). The last category must not be presented as evidence.



6. Conclusion

Road safety enforcement depends on officers observing large volumes of traffic, which is slow, inconsistent and difficult to extend to every junction. This work presented a prototype AI-based smart traffic constable that helps with one part of the task: automatically screening images and live camera frames for riders without helmets and for triple riding, and drawing an officer's attention to them. The implemented system is a Python and Tkinter dashboard built around a fine-tuned nano-scale Ultralytics YOLO detector with nine classes. OpenCV supplies camera access and image output. The dashboard displays annotated detections, per-class counters, a mean confidence value and a SAFE or VIOLATION status, saves annotated images, and sends an e- e-mail alert with an image attachment under a cooldown. The contribution of the project is a complete, working end-to-end baseline, from model to interface to notification, that is easy to run on ordinary hardware and easy to extend. Its limitations are equally clear. The project supplies no quantitative evaluation, so no claim of detection accuracy, accident reduction or enforcement benefit can be made. Violation detection is a label-presence rule, with no tracking, plate recognition, location or evidence database. The class named triple does not appear to match the rule that expects triple seat. Live-mode inference runs in the interface thread, every live frame is written to disk, and the e-mail configuration is hard-coded. In real deployment, further considerations include camera placement, night-time performance, dataset representativeness, data protection for captured images, legal admissibility of automated evidence, and regular re-validation of the model.

6.1 Future Scope

1. Correct and test the current logic: align class names with the rule, and produce the measured evaluation described in Section 5.9.

2. Live CCTV integration: support RTSP or IP camera streams and recorded-video input. Tracking and association: attach helmet status to the correct rider and count each vehicle once [7], [8].

3. Improved number plate recognition: add a dedicated plate detector and OCR stage, such as Tesseract [9] or a trained model, evaluated on local plate formats.

4. Edge computing: run a quantised model on an edge device near the camera to reduce bandwidth and latency.

5. Real-time notifications and cloud connectivity: add SMS or app push alerts and a secure server.

6. Analytics and storage: a database with time, place and violation-type statistics and a review interface. AI-Based Smart Traffic Police Constable - Proposed System, Implementation, Conclusion and References Page 11 of 13 .

7. Integration: connect with official traffic-management and challan systems through approved interfaces.

Any consequential enforcement action, such as issuing a fine, should require verification by authorized personnel who review the evidence. The system should be used to prioritise and support their work, not to make final decisions automatically.

7. Acknowledgment

We express our sincere gratitude to our project guide, [Guide name], [Designation], Department of [Department name], for valuable guidance, encouragement and constructive feedback throughout this work. We thank [Head of Department name], Head of the Department, and the faculty members of the Department of [Department name] for their support and for providing the facilities needed to complete this project. We are grateful to [Institution name] for providing the academic environment and resources for this research. We also thank our teammates, [Team member names], for their cooperation and contributions to the development and testing of the system, and [any other individuals or organizations that genuinely contributed] for their help. Finally, we thank our families and friends for their support and encouragement.

7. References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, realtime object detection," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[2] Ultralytics, "Ultralytics YOLO documentation and source code." [Online]. Available: https://docs.ultralytics.com and https://github.com/ultralytics/ultralytics (version-specific details for the YOLO26 model family to be verified against the documentation before submission).

[3] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[4] Python Software Foundation, "tkinter - Python interface to Tcl/Tk," Python 3 Documentation. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[5] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in Proc. 18th Int. Conf. Pattern Recognition (ICPR), Hong Kong, 2006, vol. 3, pp. 850-855, doi: 10.1109/ICPR.2006.479.

[6] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Computer Vision - ECCV 2014, Lecture Notes in Computer Science, vol. 8693, Springer, 2014, pp. 740-755, doi: 10.1007/978-3-319-10602-1_48.

[7] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and real time tracking," in Proc. IEEE Int. Conf. Image Processing (ICIP), Phoenix, AZ, USA, 2016, pp. 3464-3468, doi: 10.1109/ICIP.2016.7533003.

[8] N. Wojke, A. Bewley, and D. Paulus, "Simple online and real time tracking with a deep association metric," in

Proc. IEEE Int. Conf. Image Processing (ICIP), Beijing, China, 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.

[9] R. Smith, "An overview of the Tesseract OCR engine," in Proc. 9th Int. Conf. Document Analysis and Recognition (ICDAR), Curitiba, Brazil, 2007,
pp. 629-633, doi: 10.1109/ICDAR.2007.4376991.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," Int. J. Computer Vision, vol. 88, no. 2, pp. 303-338, 2010, doi: 10.1007/s11263-009-0275-4.

[11] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for objectd etection algorithms," in Proc. Int. Conf. Systems, Signals and Image Processing (IWSSIP), Niteroi, Brazil, 2020, pp. 237-242, doi: 10.1109/IWSSIP48289.2020.9145130.