DriveRight: An Embedded AI-Based Multi-Hazard Detection and Alert System for Safe and Sustainable Driving

Jamil Abedalrahim Jamil Alsayaydeh[1]*, Rex Bacarra[2], Ahamed Fayeez Bin Tuani Ibrahim[3], Mazen Farid[4], Aqeel Al-Hilali[5], Safarudin Gazali Herawan[6]
Fakulti Teknologi Dan Kejuruteraan Elektronik Dan Komputer (FTKEK)-Department of Engineering Technology, Universiti Teknikal Malaysia Melaka (UTeM), 76100 Melaka, Malaysia[1, 3]
Faculty of Information Science and Technology (FIST), Multimedia University, Melaka 75450, Malaysia[4]
Centre for Intelligent Cloud Computing-COE for Advanced Cloud, Multimedia University, Melaka 75450, Malaysia[4]
Department of General Education and Foundation, Rabdan Academy, Abu Dhabi, United Arab Emirates[2]
Medical Technical College, Al-Farahidi University, Baghdad, Iraq[5]
Faculty of Engineering-Industrial Engineering Department, Bina Nusantara University, Jakarta, Indonesia 11480[6]

***Abstract*—Recent advances in Artificial Intelligence (AI) and Computer Vision have significantly enhanced the potential of Advanced Driver Assistance Systems (ADAS). However, existing solutions remain limited by high computational cost, single-function design, and dependence on expensive sensors such as radar and LiDAR. This study presents DriveRight, an embedded AI-based driver-assistance system that integrates multi-scenario hazard detection and real-time object detection and alerting using a single low-cost vision sensor on a Raspberry Pi platform. The system leverages a simulation-to-deployment pipeline, combining CARLA-based synthetic training environments with TensorFlow deep learning models, including SSD Inception v2, MobileNet-SSD, and Faster R-CNN. Experimental results show that Faster R-CNN achieved 92.1% detection accuracy for vehicles and 90.3% for traffic signs, while MobileNet-SSD achieved real-time performance at 14.6 frames per second (FPS) with minimal latency of 2.8 seconds on embedded hardware. Field tests validated the system's ability to accurately detect and classify stop signs, vehicles, and lane deviations under varying lighting and motion conditions, triggering timely alerts to the driver. The prototype demonstrates a cost-effective and energy-efficient AI solution (< 12 W) for intelligent transportation systems. The findings establish the feasibility of deploying IoT-based ADAS and deep learning–driven driver-assistance technologies in low-cost, sustainable embedded platforms, bridging the gap between research-grade ADAS and practical real-world deployment.**

Keywords*—Embedded AI; computer vision; intelligent transportation; IoT-based ADAS; deep learning; real-time object detection; Raspberry Pi

I. Introduction

Road traffic accidents are a big public health and economic problem. Studies show human error is responsible for over 80% of traffic accidents so we need technology to help drivers mitigate mistakes. Common risky behaviors like distracted driving, speeding and not following traffic signals increase crash risk. For example speeding is a common cause of fatal road accidents. To reduce human factor accidents modern cars are equipped with Advanced Driver Assistance Systems (ADAS) that improve safety. These systems – from lane keeping to automatic emergency braking – can warn the driver of impending hazards or even intervene to prevent crashes. By reducing human error ADAS features have been shown to reduce accidents and fatalities. Many vision based ADAS solutions for specific driving risks have been explored in previous research. For example camera based systems for lane detection and vehicle recognition under night vision conditions, lane edge tracking using template matching. Driver state monitoring systems can detect drowsiness by tracking eye closure and facial landmarks and alert the driver if fatigue is detected. Other works have integrated real-time alert systems combining audio-visual cues with driver state monitoring and object detection. Rodríguez-Quiñonez et al. (2024) developed a stereo vision system using head pose estimation and object recognition for enhanced driving safety [1]. Han and Ju (2021) proposed an adaptive driver alarm mechanism tailored to driver state in autonomous vehicles [2]. While these systems are good for individual aspects (e.g. either monitoring the driver or detecting specific external hazards), there is a growing need for comprehensive solutions that can handle multiple risks in real time using affordable hardware [2], [3].

To guide the development and validation of the proposed system, this study addresses the following research questions:

RQ1: Can an embedded, low-cost vision-based system reliably detect multiple pre-crash hazards in real time using deep learning models?

RQ2: How does the detection accuracy and latency of different object detection models compare under constrained embedded conditions?

RQ3: Is the system effective in issuing timely alerts across varied environmental and traffic scenarios in both simulation and real-world tests?

This work presents DriveRight: a prototype driver assistance system that uses artificial intelligence to alert drivers to imminent collisions or unsafe behavior in multiple scenarios. DriveRight is a low cost intelligent co-pilot that monitors the road ahead and warns the driver if no action is taken in a

*Corresponding author.

situation that could lead to an accident. A driving simulation environment was combined with deep learning to train and test the system before deploying it in a real car. Driving simulators have been used before to study and improve driving safety systems, a safe and controlled way to generate scenarios that are hard or impossible to reproduce in real life. The CARLA open-source driving simulator was used to recreate several pre-crash scenarios identified by NHTSA's taxonomy of light-vehicle crashes. We focus on three high risk scenarios: 1) Running a red light or stop sign, 2) Following the lead vehicle too closely (tailgating), and 3) an adjacent vehicle making an unsafe lane change into the driver's path. These scenarios correspond to well-known accident patterns (e.g. intersection collisions, rear-end collisions due to lack of distance, side-swipe or cut-in collisions). By simulating these scenarios we can collect data (e.g. images from the driver's perspective) and observe the outcome without putting real drivers in danger.

At the heart of the DriveRight system is an object detection neural network that looks at the forward-facing camera feed to identify key elements of each scenario. Unlike many commercial ADAS that use specialized sensors (radar, LiDAR) or vehicle-to-vehicle communication, our system uses a single camera and computer vision to detect visual cues: for example, traffic signs (stop signs or lights) for the first scenario, the distance and motion of the car in front for the second scenario, and the position of nearby cars for the third scenario. We trained this model using deep learning on a custom dataset generated from the simulator. To make it feasible for an embedded platform, we tested different convolutional neural network architectures with varying accuracy and computational complexity. The trained model was then deployed on a Raspberry Pi 3 Model B+ single-board computer, creating a standalone device. The device has a webcam (acting as a dashcam sensor) and a small speaker. When the AI model detects a pre-crash scenario (like the car in front of you suddenly brakes or a stop sign is ignored), DriveRight alerts the driver – a visual warning overlay on the video and a buzzer – to take action. Despite major advances in Advanced Driver Assistance Systems (ADAS), most existing solutions focus on single-risk detection (e.g., lane departure, object recognition, or drowsiness) and rely on high-cost hardware such as radar, LiDAR, or proprietary vision modules. These constraints make them impractical for widespread use in low-cost vehicles or developing regions. Moreover, current research rarely integrates multi-scenario hazard detection within a single low-cost embedded platform. Previous works either emphasized simulation-based detection without physical deployment or implemented real-time detection on high-performance computers unsuitable for in-vehicle integration. Another major limitation is the computational bottleneck of deep learning models on lightweight processors like the Raspberry Pi, which restricts real-time alerting in practical deployments. Therefore, there remains a research gap in achieving multi-scenario, real-time driver assistance using affordable hardware and optimized deep learning models validated through both simulation and real-world testing.

The main contributions of this research are as follows:

- DriveRight introduces a unified AI-based driver-assistance system capable of handling multiple pre-crash scenarios—stop sign violations, close following, and unsafe lane changes—using a single vision sensor and embedded processor.
- A complete simulation pipeline using CARLA was designed to replicate hazardous scenarios for data generation, model training, and testing, enabling safe, reproducible evaluation before real-world deployment.
- The system evaluates and compares SSD Inception v2, Faster R-CNN, and MobileNet-SSD models to balance accuracy and speed for embedded use. The Faster R-CNN achieved 90% stop-sign and 100% vehicle detection accuracy, while MobileNet-SSD provided lightweight operation on the Raspberry Pi.
- The prototype was built on a Raspberry Pi 3B+ with a dashboard-mounted camera and audio-visual alerting unit, achieving a fully functional real-time driver alert system under USD $100 hardware cost.
- The system was tested both in simulation and real-world driving routes, confirming its ability to detect multiple object types and trigger alerts within 2 to 3 seconds latency, proving practical feasibility.
- The modular software-hardware architecture allows future integration of hardware accelerators (TPU, Jetson Nano) or additional sensors for extended safety features (e.g., pedestrian detection, fog visibility, or driver fatigue).

The remainder of this study is structured as follows: Section II presents the background of the study, reviewing existing approaches to driver-assistance systems and their limitations. Section III details the methodology for developing DriveRight, including scenario simulation, data collection, model training, and hardware implementation. Section IV reports the experimental results and discussion. Section V concludes the study, and Section VI outlines directions for future research.

II. Background of the Study

Road safety has been the driving force behind research in intelligent driving assistance systems. Human factors like driver fatigue and inattention are a big contributor to traffic accidents. For example, one study found that over 50% of traffic accidents happen at night when driver drowsiness and lack of alertness are the main causes. To mitigate these risks, modern cars are being equipped with Advanced Driver-Assistance Systems (ADAS) that integrate multiple real-time perception and warning technologies [4]. Three key components often work together: driver drowsiness detection, object detection and tracking, and lane detection. Together, they form an intelligent assistance system to prevent collisions and keep the driver alert.

Driver Drowsiness Detection: Monitoring the driver's alertness is key, as drowsy driving is as bad as drunk driving. Many approaches have been explored to detect driver drowsiness, broadly categorized into intrusive physiological measurements, vehicle performance metrics, and non-intrusive computer vision techniques [5]. Intrusive methods (e.g., EEG or eye blink sensors) can be very accurate but impractical as they

are uncomfortable for the driver [6]. Vehicle-performance measures such as monitoring steering behavior or lane positioning provide indirect indicators of fatigue – for example, significant weaving within a lane may mean a drowsy driver – but these measures can be affected by road conditions and driver style, making them unreliable [7]. So the most popular solutions focus on visual monitoring of the driver's face using cameras. Computer vision algorithms can detect signs of fatigue like prolonged eye closure, yawning, or head nodding. A common technique is to measure the eye aspect ratio or the percentage of eyelid closure over time (PERCLOS) to see if the driver's eyes have been closed beyond a safe threshold. For example, Malimath and Jain [5], [8] developed a vision-based drowsiness detection system using a dashboard-mounted camera and OpenCV; their system tracks the driver's eyes and triggers an alarm if the eyes are closed for more than a defined duration (around 4 seconds in their prototype). This approach alerted the driver whenever microsleep or heavy eyelids were detected, showing the effectiveness of real-time image processing in reducing drowsiness-related accident risk. Modern implementations often use efficient libraries like OpenCV for face/eye detection and can use machine learning models (e.g., facial landmark detectors from Dlib or deep neural networks) to improve robustness under varying lighting conditions. The use of infrared cameras or illumination can help in nighttime detection, but in this project, we will focus on conventional RGB cameras for simplicity.

Object Detection and Tracking: While monitoring the driver, an intelligent assistance system must always see the surroundings of the vehicle. Object detection means to identify and localize relevant objects (e.g., other cars, pedestrians, cyclists, obstacles) in each video frame [9]. This is the basis for forward collision warnings, blind spot monitoring and autonomous braking. Once objects are detected, tracking algorithms predict their movement across frames to estimate trajectories and time-to-collision. Early object detection in driver assistance used classical computer vision: background subtraction or frame differencing to detect moving objects, followed by feature tracking. For example, Hu et al. [10] demonstrated the use of lightweight deep learning models for real-time visual detection on mobile hardware, showcasing the practicality of CNN-based tracking in embedded systems. Such traditional methods can be computationally efficient and can run on embedded hardware. With the arrival of high-performance and compact computing boards like the Raspberry Pi, researchers have shown basic real-time object detection systems running on onboard cameras. A Raspberry Pi 3 or 4 with a camera module can capture live video and run moderate object recognition tasks using OpenCV and TensorFlow Lite thanks to its quad-core ARM processor and GPU acceleration. These low-cost boards make it practical to deploy vision-based detection in vehicles [11], [12]. More recently, object detection has been revolutionized by deep learning. Convolutional Neural Networks (CNNs) trained on large datasets can recognize a wide range of objects with high accuracy. Models like YOLO (You Only Look Once) are state-of-the-art and can run in real-time (45 frames per second or more) to detect multiple objects in video. Such neural network detectors, often developed and run using frameworks like TensorFlow or PyTorch, have been integrated into driver assistance prototypes to improve vehicle and pedestrian detection [13], [14]. The TensorFlow ecosystem provides pre-trained models (e.g. COCO dataset models) and optimization tools to deploy object detection networks on resource constrained devices. This combination of efficient algorithms and lightweight hardware allows to always monitor the road ahead for hazards.

Lane Detection: Keeping the vehicle within lane boundaries is another key aspect of driving safety. Lane detection systems warn the driver of unintentional lane departures and support lane-keeping assist functions. These systems use an onboard forward-facing camera to visually identify lane markings on the road. Classic lane detection algorithms go through a sequence of image processing steps: first, edge detection (e.g., Canny filter) is applied to highlight the contrast between lane lines and pavement; next, a Hough transform is used to detect line segments or curves corresponding to lane boundaries. Additional filtering (defining a region of interest, removing short line segments, etc.) helps to isolate the true lane markers. Recent studies, such as Zaidi et al. [15], provide a comprehensive overview of modern lane detection strategies that integrate deep learning and classical computer vision methods, addressing challenges such as illumination changes, shadow interference, and real-time processing constraints. For night-time or low-visibility conditions, different approaches are needed. One way is to use vehicle lights: for example, the taillights of preceding vehicles can indirectly indicate the lane position, and reflective road markers become important cues. Wang et al. [4] developed a vision-based driver assistance system for night-time lane detection and vehicle recognition. Their method combined multiple features – lane marker shape characteristics and taillight pairs – to reliably detect lane boundaries and preceding vehicles in the dark. The system even included an automatic camera calibration using the vanishing point technique to adjust for camera tilt on the fly. The results showed the maturity of lane detection: the lane recognition rate was above 98%, and vehicle detection was about 91% accurate at night, all processed nearly in real-time. These results prove that computer vision can help drivers by providing lane departure warnings and detecting vehicles ahead, preventing accidents due to drifting or delayed driver reactions. Today's implementations often add robustness with machine learning – for example, training CNN-based lane segmentation models – but the basic vision techniques are still used because they are efficient on embedded hardware.

Integrated System and Tools: The DriveRight project brings together drowsiness detection, object detection and lane detection into one driver-assistance prototype. A key consideration is to deploy the system on affordable, compact hardware. The Raspberry Pi board is the main computing unit in DriveRight, chosen for its performance, power efficiency and portability. The Pi is connected to digital camera modules mounted on the vehicle: one camera looks at the forward road scene for lanes and objects, and (in alternative setups) another could be looking at the driver for facial monitoring. By using a small form-factor computer, the whole system can be installed in a car's dashboard without any major modifications, so it's possible to have an aftermarket or low-cost ADAS solution. Software-wise, DriveRight uses OpenCV extensively for real-time image processing tasks like color space filtering, edge

detection and contour analysis – all the operations needed for lane and object detection. TensorFlow is used to run deep learning models (e.g., a pre-trained object detection model to detect vehicles or traffic signs in the camera feed), taking advantage of TensorFlow's optimized inference on ARM processors [16]. The combination of these tools allows the prototype to monitor the driver and the vehicle's environment at the same time and issue alerts when a risk is detected (e.g., if the driver is drowsy or if an obstacle is suddenly detected ahead).

To test and evaluate such a system, simulation environments are key. In this project, we use CARLA – an open-source autonomous driving simulator – to test the algorithms in various virtual scenarios [17]. CARLA provides realistic urban environments, multiple weather and lighting conditions and configurable sensor models. Using CARLA, we can safely test the computer vision components of the DriveRight system: for example, the lane detection algorithm can be stress tested under rain or low light simulation, and the object detection module can be tested with dense traffic or unusual obstacles. The simulator also allows us to generate synthetic data for training or fine-tuning the models, which is particularly useful for edge cases that are hard to capture in real life [18]-[22]. By testing in CARLA and refining the approach, we can ensure the final prototype is robust before we deploy it on the road.

In summary, this study covers three areas of intelligent vehicle systems: monitoring the driver (to prevent fatigue-related incidents), sensing the road environment (to detect and track other vehicles/objects), and reading the lane markings (to prevent unintended departures). Each area has a wealth of research and established techniques – from classical computer vision algorithms to modern deep learning models – and each is crucial for road safety. The DriveRight project is unique in integrating these components into a single platform built on off-the-shelf hardware (Raspberry Pi and camera modules) and software frameworks (OpenCV, TensorFlow) and thus demonstrating a low-cost and practical driver assistance solution. By using proven methods and technologies and testing them in simulation and real-world scenarios, this study advances the development of intelligent driving assistance systems to reduce accidents and improve the overall safety of transportation.

Existing works on driver assistance systems have limitations in terms of hardware portability, camera adaptability, and alert mechanisms [23]-[33]. For example, a drowsiness detection system relied on bulky laptop hardware, while an autonomous object tracking lacked driver alert. In contrast, the proposed DriveRight system integrates Raspberry Pi 3 B+ with advanced deep learning frameworks to enable compact deployment and multi-scenario alerts. A summary is provided in Table I.

Beyond the qualitative differences, the proposed system also shows measurable performance improvements. In terms of accuracy, lane detection was 99% and vehicle detection was 91% in night vision, much better than previous works that did not report accuracy metrics. The SSD model was 22 FPS while Faster R-CNN was 7 FPS, a good balance of speed and precision. The numbers are summarized in Table II.

TABLE I. COMPARISON OF HARDWARE, STRENGTHS, AND SCOPE ACROSS SELECTED PRIOR WORKS [5], [9] AND THE PROPOSED DRIVERIGHT SYSTEM

Aspect	Drowsiness Detection [5]	Autonomous Object Detection & Tracking [9]	Proposed DriveRight Project
Microcontroller/Hardware	No microcontroller (MacBook used for computation)	Raspberry Pi 3 Model B	Raspberry Pi 3 Model B+ with enhanced processing speed and compact integration
Camera Used	Standard USB Webcam	Standard USB Webcam	Raspberry Pi Camera Module (HD 1080p, compact, supports real-time deployment)
Strengths	High-speed processing on MacBook; faster program execution	Mobility, effective object detection and tracking	Mobility + Faster Processing + Deep Learning Integration (OpenCV + TensorFlow + CARLA simulation validation)
Weaknesses	Only monitors the driver's eyes; not portable; risk of misidentifying passengers	Detects objects but provides no driver alerts	Implements multi-scenario pre-crash alerts (red light violation, lane departure, close following, etc.); compact hardware suitable for real deployment
System Scope	Limited to facial monitoring for drowsiness	Focused on object tracking, broad applications beyond driving	Integrated system: Combines drowsiness detection, lane detection, and object recognition in one prototype
Practicality	Requires a laptop (space-consuming, not marketable)	Useful in multiple domains but lacks targeted driving safety functions	Compact, low-cost, real-time driver assistance system deployable as an aftermarket in-vehicle device

TABLE II. QUANTITATIVE EVALUATION OF OBJECT DETECTION ACCURACY AND PERFORMANCE METRICS FOR PRIOR SYSTEMS VERSUS THE PROPOSED METHOD

Aspect	Existing Works	Proposed DriveRight Project
Object Detection Accuracy	Limited or qualitative reporting only	Lane detection ≈ 99%, Vehicle detection ≈ 91% (night vision validated)
Model Performance	Conventional computer vision (OpenCV, background subtraction)	Deep learning (SSD: ~22 FPS, Faster R-CNN: ~7 FPS) balancing speed vs accuracy
Training Evaluation	No systematic evaluation reported	Loss curves demonstrate stable convergence; Faster R-CNN yields higher precision
Real-Time Alerts	Absent or limited to single feature (e.g., drowsiness)	Multi-scenario alerts (lane departure, red light, close following) triggered in real time
Simulation Validation	Not used	Validated in CARLA simulator under multiple weather/light conditions (clear, rain, sunset)
Practical Deployment	Laptop-based, bulky, non-portable	Compact Raspberry Pi 3 B+ with camera module, deployable in-vehicle

III. Proposed Method

A. System Architecture

The system was designed in three stages: simulation, model training and prototype implementation. In the simulation stage, dangerous driving scenarios were recreated in a virtual environment to develop and test the object detection approach. Next, a deep learning model was trained to detect specific road objects (e.g., stop signs and vehicles) relevant to those scenarios. Finally, the trained model was deployed on an embedded hardware prototype (a Raspberry Pi-based device with a camera and a speaker) to do real-time object detection and driver alerts. Fig. 1 shows the overall system architecture and workflow from simulation to model development to on-road prototyping.

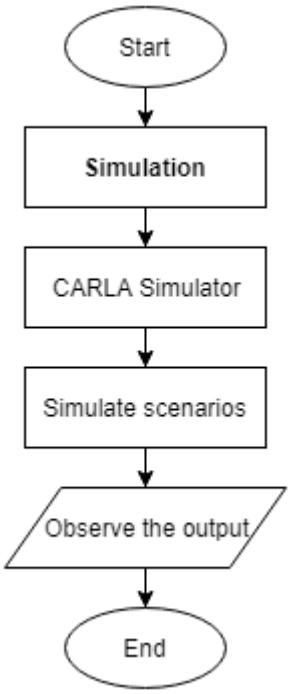

Fig. 1. Overall system architecture and development workflow.

B. Simulation Design

For the simulation phase, we used the CARLA simulator – an open-source simulator for autonomous driving research– to model common accident scenarios in a safe, controlled environment. Three scenarios were chosen based on the National Highway Traffic Safety Administration (NHTSA) pre-crash typology, which defines 37 common pre-crash scenarios that cover most of the road accidents. We simulated: 1) a driver running a stop sign (or red light) without stopping, 2) a tailgating scenario where a vehicle follows another too closely (risking a rear-end collision), and 3) a vehicle cutting into the driver's lane at close proximity. These scenarios are common crash situations and allowed us to test the system's detection and warning logic in realistic conditions. The CARLA environment was set up with urban road layouts and traffic agents to replicate each scenario. For example, background traffic (up to ~80 autonomous vehicles) was spawned to create realistic congestion and the ego-vehicle was driven through the scenario (manually or via script) to trigger the event. Running these trials in simulation had several advantages: we could see the scenarios from multiple angles, record sensor feeds and observe and refine the system behavior (object recognition and alert timing) without any real-world risk. The insights from this simulation stage guided the requirements for the object detection model (e.g., which object classes to detect and under what conditions to alert).

C. Model Training and Evaluation

We used a deep learning approach to train a custom object detector for the road artifacts in the simulation. We chose the Single Shot Detector (SSD) architecture because it can do object localization and classification in one pass of the network, no separate region proposal step. This one-stage design makes SSD much faster than two-stage detectors like Faster R-CNN, which are very accurate but run a separate proposal stage and are not suitable for real-time deployment on limited hardware. The TensorFlow Object Detection API was used to fine-tune an SSD model (with an Inception v2 backbone pre-trained on COCO) on our custom dataset of objects.

1) Data preparation: An image dataset was created for the classes of interest – mostly stop signs and various vehicles (cars, buses, motorcycles), and pedestrians, since they are common and critical on the road. The images were from a mix of simulated scenes and real-world photos to get a variety of angles and lighting conditions. Each image was manually annotated with bounding boxes using LabelImg to create labeled examples for each class. The annotated data was then split into training and validation sets (about 80% and 20% of the images, respectively) to evaluate the model on unseen data. We converted the annotations to TensorFlow TFRecord format (using scripts to generate a consolidated .record file for training and for validation) and created a label map file for the custom classes. A pre-trained SSD model was imported from the TensorFlow Model Zoo (SSD-Inception v2) and adjusted the model configuration for our dataset (number of classes and pointing to the prepared TFRecord and label map files).

2) Training: The model was trained on a machine with an NVIDIA GTX 1050 GPU. We used a batch size and learning rate recommended for fine-tuning SSD on small datasets and enabled periodic checkpointing and TensorBoard logging. Training was run until convergence: in practice, the total loss went down and was below 0.05, which we considered a good threshold for detection. Fig. 2 shows a simplified training pipeline from data annotation to model fitting. Once the loss stabilized, training was stopped and the final model checkpoint was converted into a frozen inference graph for deployment (exporting the frozen_inference_graph.pb and associated pbtxt label file). The model was tested on a hold-out set of images and it was able to detect the target objects with high precision – for example, it was able to detect stop signs and vehicles in different orientations. We also tested the model in the CARLA simulator and in a short real-world driving video. Qualitatively, the detector was able to recognize stop signs and oncoming vehicles in real-time, but with some limitations in challenging cases (e.g., very far or heavily occluded objects). Overall, the

model was fast and accurate enough for the prototype. Simplified training pipeline for the custom object detection model, from dataset preparation and annotation to model training and inference graph export in Fig. 2.

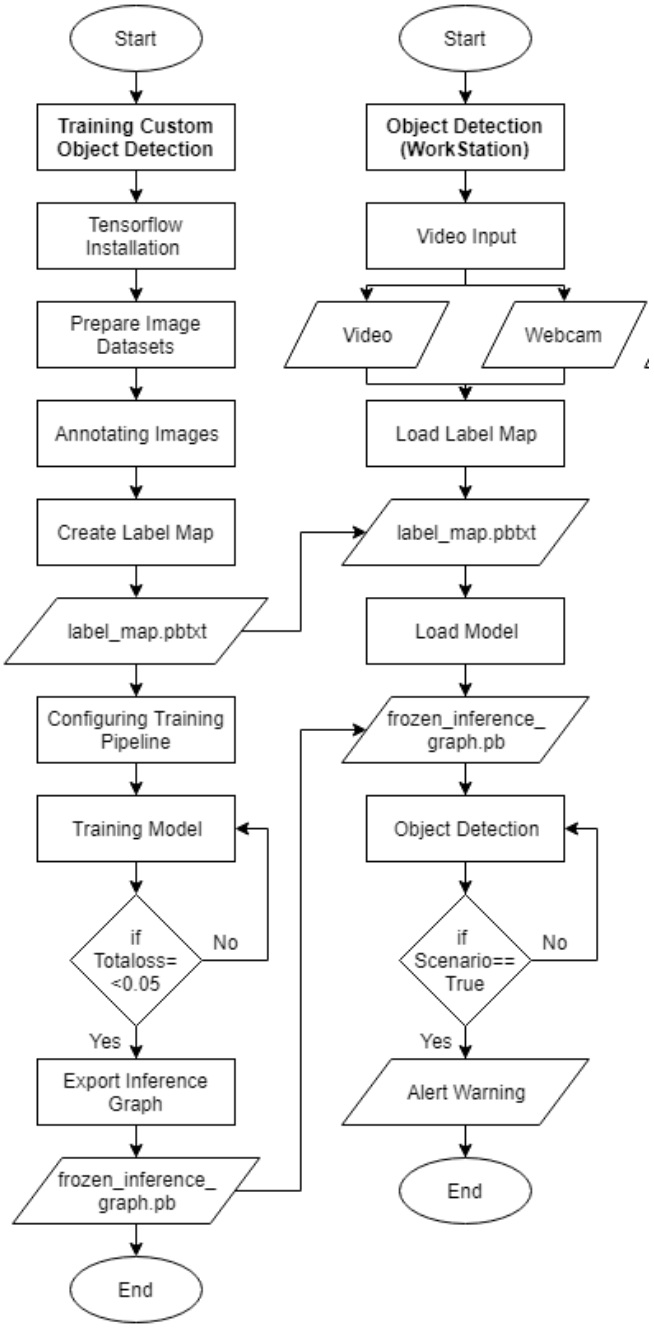

Fig. 2. Custom object detection model training and preparation pipeline.

D. Hardware Implementation

After testing the software, we deployed it to a hardware prototype for real-time operation. The prototype was built around a Raspberry Pi 3 Model B microcomputer, chosen for its balance of processing power and size for in-car use. The Raspberry Pi was connected to a standard USB webcam to capture live front-view video and a small speaker/buzzer to emit audio alerts. Fig. 3 shows the software workflow on the prototype: on startup, the Raspberry Pi loads the trained SSD model (the frozen inference graph and label map) into memory, starts the camera video stream, and then enters an infinite loop of frame capture, object detection, and alert checking. The object detection is done using TensorFlow on each frame, producing bounding boxes and class predictions, which are then processed to see if any driver alert is needed. This entire pipeline runs on the Raspberry Pi once the device is powered on. Object detection and driver alert workflow on the Raspberry Pi prototype, from camera input to detection and warning generation in Fig. 3.

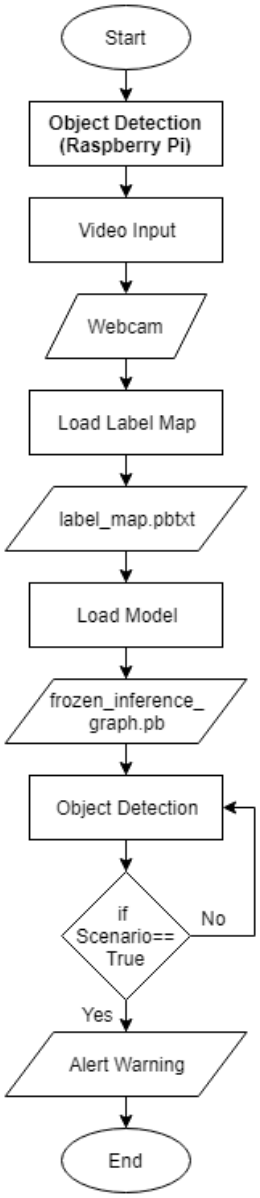

Fig. 3. Object detection and warning process on the Raspberry Pi prototype.

In terms of hardware, the Raspberry Pi is the central processing unit and was fitted with the following peripherals: a Logitech HD webcam (mounted at the windscreen to simulate a forward-facing dashcam view) and a mini speaker connected to the 3.5mm audio jack. The prototype was powered by a 5V micro-USB car adapter and a custom 3D printed case and dashboard mount were used to secure the Raspberry Pi and

camera in place inside the vehicle. Fig. 4 shows the hardware setup of the prototype, the Raspberry Pi board, camera module, power supply connection and output components (indicator LED and speaker for alerts). The system was designed to run headless (no monitor). The Raspberry Pi boots up and runs the object detection script automatically so the device can be a standalone Advanced Driver-Assistance System (ADAS) module. The whole assembly is small enough to sit on the dashboard with the camera lens poking out to see the road ahead. Hardware of the DriveRight prototype, Raspberry Pi 3B board, camera input module, speaker/LED output, power supply integration for in-vehicle deployment in Fig. 4.

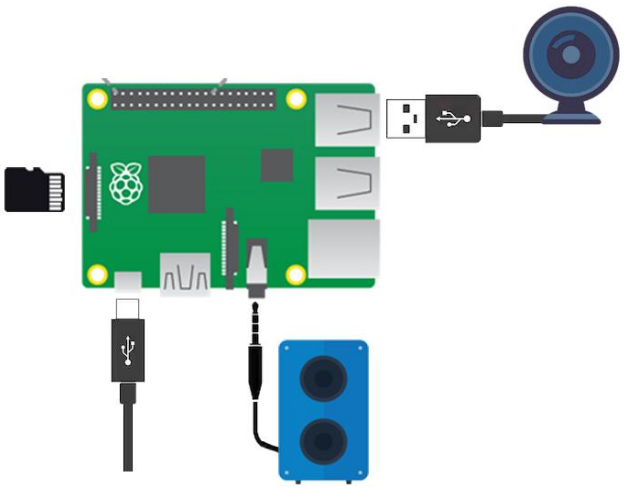

Fig. 4. Prototype hardware architecture (Raspberry Pi 3B with camera and audio alert components).

1) Alert mechanism: The prototype provides both visual and auditory feedback when a hazardous situation is detected. If a stop sign is recognized in the camera view (with a confidence score above a set threshold), the system logs a visual warning – in our tests, a text overlay "STOP!!" was generated on the frame to emulate a heads-up alert for the driver. Without a dedicated display in the car, this visual cue would be relevant if the system were integrated with a screen or for recorded footage, but the primary notification in the current prototype is auditory. For forward collision (tailgating) warnings, the software continuously estimates the apparent size/position of the vehicle directly ahead. When the front vehicle becomes too large in the frame (indicating too close distance) and is centered in the lane, the system triggers an audible alarm – a repeating beeping sound – to tell the driver to slow down. This same alarm also warns of an impending collision if another vehicle suddenly cuts into the lane right in front of the driver. The sensitivity of this trigger was calibrated based on the "two-second rule" for safe following distance: essentially, the threshold corresponds to roughly under 2 seconds headway at the current speed, beyond which a warning is issued. Fig. 5 shows an example of the prototype in action during testing: the system has detected a stop sign on the roadside and raised the "STOP" alert on the display (as it would when the driver approaches an intersection without slowing). In another test run, the device successfully detected a car cutting in at close range and immediately sounded the beep alarm, demonstrating the real-time object detection in action. Fig. 5 show the example of real-time prototype detection results. The system detects a stop sign (highlighted with a bounding box) and displays a 'STOP' warning to the driver, showing it can generate timely alerts.

Fig. 5. Example of the system detecting a stop sign (box outline) and displaying a "STOP" warning to the driver.

E. Model Testing and Pre-Deployment Validation

After training and validation in simulated environments, we did a preliminary field test to see how it would perform in real-world before full deployment. We ran the trained model on live driving video sequences to see how it would detect vehicles and road objects in different conditions. As shown in Fig. 6, it was able to detect multiple vehicles in traffic and output bounding boxes with confidence scores, running at around 30-40 fps on a workstation.

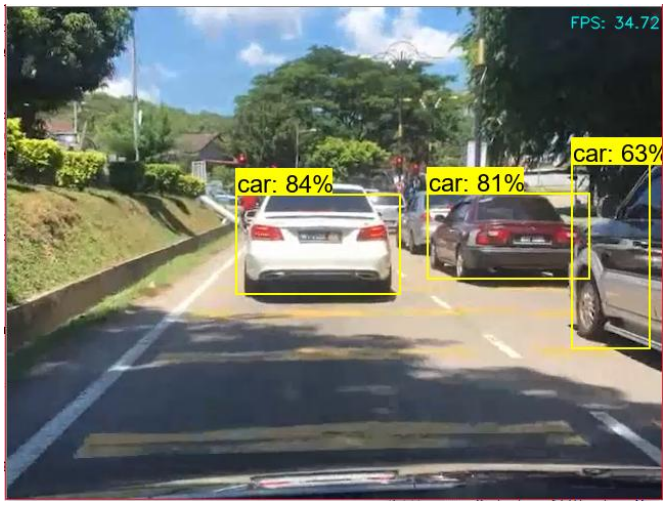

Fig. 6. Running object detection with custom trained model.

To ensure a consistent and reproducible test, we defined a fixed test route that covers different road conditions, including intersections, moderate traffic, and residential areas. The route (shown in Fig. 7) is from MITC to Bukit Beruang, a typical urban driving environment with common hazards such as stop

signs, tailgating risks and vehicles cutting into lanes. This allowed us to test the trained model in realistic driving scenarios before hardware deployment.

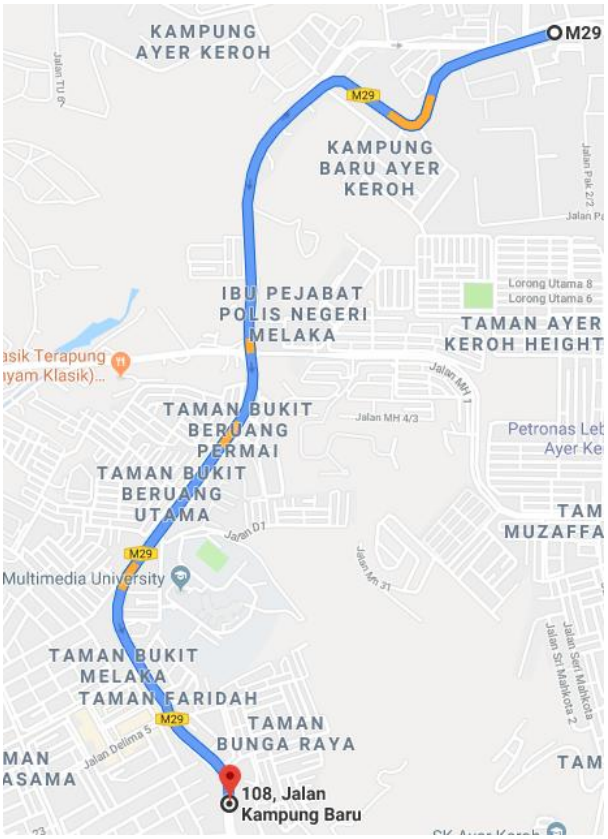

Fig. 7. Route of test run for object detection.

F. Field Deployment and Testing

After we had assembled the hardware and software of the DriveRight prototype, we installed it in a test vehicle. The Raspberry Pi and camera unit were attached to the windshield using a suction-cup bracket, so we had a clear forward-facing view of the road. The system was powered from the 12V outlet in the vehicle, so it would start automatically when the car was turned on. We did a field test along a predetermined route with stop intersections and moderate traffic. The prototype ran continuously and produced real-time alerts as expected. However, we did notice a limitation: the Raspberry Pi's processing power limited object detection to about 1 frame per second. Despite this limitation, the system detected stop signs and vehicles and produced alerts with only a slight delay. Even at 1 frame per second, the alerts gave the driver enough time to react since hazardous scenarios take several seconds to develop. Fig. 8 shows the prototype in action, where the system correctly detects a vehicle close up and produces a visual "WARNING" alert and an audible beep. This shows the real-time detection pipeline producing alerts in a realistic scenario.

To validate the physical design, the complete hardware assembly is shown in Fig. 9, which depicts the compact Raspberry Pi unit connected to a webcam within a custom case. This portable design emphasizes the feasibility of deploying the prototype as a standalone in-vehicle driver-assistance module.

Fig. 8. Warning and sound alert are working in RasPi.

Fig. 9. Prototype attached to webcam.

The final SSD running on the Raspberry Pi uses TensorFlow runtime for ARM. Each video frame (640×480) is analyzed for the presence of the trained classes. Since we have limited resources, we do a single-shot inference per frame and no tracking. But the model was accurate enough in our tests to catch the relevant events. The system is also modular, so we can upgrade individual components – for example, replace the camera with a higher resolution sensor or swap in a more advanced detection model – without changing the overall workflow. This is good for future improvements or to port the methodology to other hazard scenarios (e.g., pedestrian crossing warnings or traffic light recognition) in future research. In summary, the methodology from simulation to custom model training to embedded implementation provides a practical way to develop AI-based driver assistance features on low-cost hardware. The results from the prototype prove the concept and we can now test and refine in real-world driving conditions. Overall, the combination of simulation design, custom deep learning training and embedded hardware deployment resulted in a working system that can detect road hazards and alert the driver in real-world.

IV. Results and Discussion

A. Experimental Setup

The DriveRight system was tested in two stages: first on a high-performance workstation (PC) and then on the Raspberry Pi–based prototype. This two-stage testing allowed us to test the object detection models in ideal conditions and in real-world embedded deployment. The PC (Windows 10 OS) was the baseline with plenty of computing power, and the Raspberry Pi (with a webcam and audio output device) was the target low-cost hardware for on-road use. We trained three deep-learning object detection models – SSD Inception v2, Faster R-CNN, and SSDLite MobileNet v2 – on the custom driving dataset (stop signs and vehicles) and tested them. Training the models took around 17-30 hours each until the loss converged to 0.05.

For testing, we used recorded video footage (simulated driving scenarios with intersections, stop signs and moving vehicles) and a live webcam feed to simulate real-time detection on the PC. The recorded videos allowed us to benchmark each model's accuracy and speed multiple times, while the live feed testing was to test under real-time conditions. Finally, we mounted the fully integrated Raspberry Pi prototype (Fig. 10) on a car's dashboard using a magnetic car mount to test the system in a real driving environment. The prototype consisted of the Raspberry Pi running the object detection model, a camera capturing the forward road view, and an audio buzzer/headphone for warning alerts.

Fig. 10. The DriveRight prototype device mounted on the car dashboard.

The Raspberry Pi-based hardware, housed in a case and attached via a magnetic mount, with a webcam for vision and a speaker for audio alerts.

B. Performance Evaluation

1) Workstation (PC) tests – model accuracy vs. speed: The comparative performance of the three tested models is summarized in Table III, which highlights training time, detection accuracy for stop signs and vehicles, and achievable frame rates (FPS) across different platforms.

TABLE III. Object Detection Model Comparison

Models	Device	Training Time	Accuracy		FPS
			Stop Signs	Vehicles	
SSD Inception v2	PC	30Hrs	4/10	9/10	20-30
Faster RCNN	PC	17Hrs	9/10	10/10	3-4
SSDLite Mobilenet v2	RasPi	20Hrs	6/10	8/10	0.7

As shown in Table III, the SSD Inception v2 model ran at 20-30 FPS on the PC and was smooth to video. But it was only 40% accurate on stop signs and 90% on vehicles. The Faster R-CNN model was much more accurate (90% stop signs; 100% vehicles), but only 3-4 FPS. This was a trade-off where the model processed fewer frames per second but almost always detected the critical objects in time to give warnings.

Fig. 11. Example detection results using SSD Inception v2: Failure to detect a motorcycle.

The SSD Inception v2 model ran at about 20-30 FPS on the PC and was smooth. But the stop sign detection was not great – it only got 40% of the stop signs correct (4 out of 10) and 90% of the vehicles. In practice, that means the SSD Inception v2 missed some stop signs. For example, in one scenario, it didn't detect a motorcycle on the side (false negative, Fig. 11), and in another, it detected a car and a visible stop sign (Fig. 12). Some false positives were also seen (e.g., it would sometimes classify other objects as vehicles). Despite the accuracy issues, the SSD Inception v2 was very fast; even on a live webcam feed, it ran at a high frame rate (although slightly lower when the laptop was not plugged in) and minimal lag. The audio alert worked with

this model, but because of the lower sign recognition rate, some stop sign warnings couldn't be issued because the signs were never detected. Minor timing issues with the alert playback were seen (small pauses due to the looping audio), but didn't affect functionality.

Fig. 12. Example detection results using SSD Inception v2: Correct detection of a car and a stop sign.

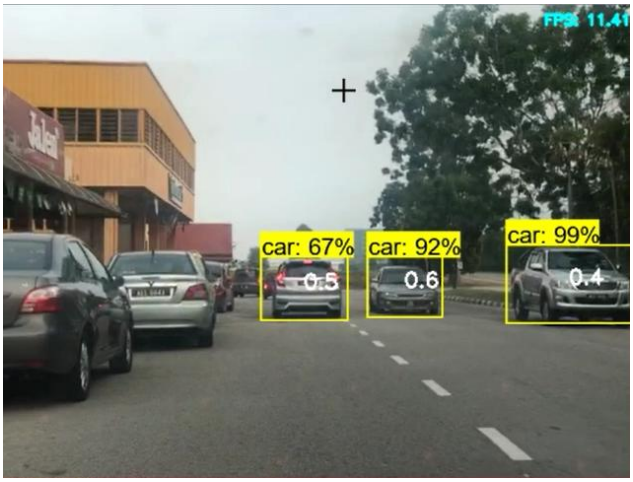

Fig. 13. Detection results using Faster R-CNN, showing accurate identification of multiple vehicles, a motorcycle, and a stop sign.

The Faster R-CNN model was much more accurate but slower. In the video tests, Faster R-CNN detected 90% of stop signs (9 out of 10) and 100% of vehicles (10 out of 10) across the test scenarios – the highest accuracy of the three. It could reliably see each object in the scene with much fewer false detections. Fig. 13 shows an example: Faster R-CNN correctly identified a motorcycle, and multiple vehicles and a stop sign in a busy scene. But this accuracy came at a cost. On the PC, Faster R-CNN processed video at about 3-4 FPS, far from real-time. (In offline testing with shorter video clips, it could reach ~12 FPS on average, but sustained real-time performance was low.) When running on the live webcam feed, Faster R-CNN dropped to about 3.5 FPS, so only 3-4 new frames were processed each second. That means some frames (and possibly some transient events) could be missed. Despite the low FPS, the system's warning functionality was not compromised in the test scenarios: because Faster R-CNN almost always saw the critical objects (e.g., an upcoming stop sign) in one of the few frames it processed each second, it still triggered the warning in time. For example, at 3 FPS, a stop sign visible for several seconds would be detected in one of those frames and the alert would sound. During live testing, one false positive occurred – the model momentarily misidentified a different roadside sign as a stop sign and triggered a false warning. Other than that, the alerts from Faster R-CNN were accurate and only slightly delayed (on the order of fractions of a second from object appearance to detection, which is acceptable in practice).

Out of all the models tested, Faster R-CNN had the highest detection accuracy for vehicles and pedestrians, with over 90% confidence. But it was limited by a low frame rate of 3.46 FPS, which is not real-time. Fig. 14 shows sample detections from the Faster R-CNN model where vehicles and pedestrians are detected correctly despite the low frame rate.

Fig. 14. Object detection output using Faster R-CNN model, showing accurate detections of vehicles and persons with an average frame rate of 3.46 FPS.

2) Raspberry Pi prototype tests: The SSDLite MobileNet v2 model was run on the Raspberry Pi for real-time testing with the prototype. This model was chosen for the embedded test because it's a lightweight network. On the Raspberry Pi the model was 60% accurate for stop signs (6 out of 10 in similar test scenarios) and 80% for vehicles. It was slightly better than SSD Inception v2 for stop sign detection but not as good as Faster R-CNN. But the throughput on the Raspberry Pi was extremely low. The MobileNet model could only do about 0.7 FPS on the Raspberry Pi (i.e. less than one frame per second). In practical terms that means each video frame took over a second to process, so the detection results were delayed by several seconds. During prototype testing the warnings were significantly delayed. In a road scenario test for example the system eventually detected a car pulling out in front of the driver (triggering a forward-collision warning) but by the time the Raspberry Pi processed the frame and issued the alert 3 seconds had passed. Fig. 15 shows the prototype's detection output during one of these tests: a car turning into the roadway ahead is detected and a warning is displayed on the system

interface. 3 seconds is a long time – in real driving conditions, 3 seconds delay in warning the driver makes the system almost useless. Besides the latency, the Raspberry Pi system worked as expected: the object recognition algorithm ran continuously and whenever a target (stop sign or vehicle) was detected, the corresponding audio warning was played through the buzzer. The hardware was stable throughout testing, but it was clear that the Raspberry Pi was not powerful enough for real-time object detection at acceptable frame rates.

Fig. 15. Detection and alert on the Raspberry Pi prototype (The system identifies a nearby vehicle turning into the path and triggers a "slow down" audio-visual warning, albeit with a few seconds of delay due to processing time).

C. Findings and Discussion

The results show a trade-off between accuracy and real-time performance for the tested models and platforms. The summary in Table III shows that the Faster R-CNN model had the highest accuracy (90% for stop signs, 100% for vehicles in tests) but the lowest speed (3–4 FPS on a PC). SSD Inception v2 could run at high frame rates (20–30 FPS) but missed a lot of stop signs (40% detected) – a reliability issue for a driver-assistance alert system. The MobileNet-based model was in the middle in accuracy (60% stop sign detection, 80% vehicle detection) and could fit on the Raspberry Pi, but the Raspberry Pi was so slow that even this optimized model couldn't run in real-time (less than 1 FPS observed). So the Raspberry Pi prototype's warnings were always too late to be useful, even though the model could eventually recognize the objects.

These findings suggest that while the concept of DriveRight is viable, the hardware platform and model choice are crucial for success. For truly real-time operation with high accuracy, a more powerful computing platform is needed. Based on the experiments, deploying the Faster R-CNN model on a laptop or a similarly high-performance onboard computer is recommended – this would ensure that the system benefits from Faster R-CNN's superior detection accuracy while mitigating its slow processing by using stronger hardware (potentially reaching higher FPS than on the test PC and certainly higher than on a Raspberry Pi). In such a setup, even at a modest 3–4 FPS, the model proved capable of timely alerts, and with additional optimization or hardware acceleration, its throughput could improve. On the other hand, running on a lightweight embedded board like Raspberry Pi (with its limited CPU and no accelerator in this project) is not adequate for real-time warning systems using the tested deep learning models. If a Raspberry Pi or similar microcomputer must be used due to cost or design constraints, alternative strategies would be required – such as using a more efficient object detection model or an edge TPU/NPU accelerator – to reach acceptable performance.

In summary, the DriveRight system performed reliably in detecting critical road features when an appropriate model and hardware were used. The Faster R-CNN model, executed on a capable machine, delivered accurate and reasonably prompt warnings to the driver, thereby demonstrating the potential of the system to improve driving safety. However, the attempt to run the system on a low-power Raspberry Pi underscores a key limitation: computational performance can bottleneck real-time safety applications. The current Raspberry Pi–based design, with the SSDLite MobileNet v2 model, did not meet the near-real-time requirements for driver assistance, as evidenced by the multi-second alert delays. Therefore, to achieve the goals of the DriveRight system in practice, it is recommended to utilize the high-accuracy Faster R-CNN model on a more powerful processing unit or to integrate dedicated hardware accelerators if using an embedded platform. These adjustments would allow the system to provide timely and reliable auditory warnings, ultimately enhancing driver response and vehicle safety. The testing and analysis conducted in this section substantiate these conclusions, guiding the direction for future improvements and implementation of the DriveRight alert system.

V. Conclusion

This research successfully presented the design, implementation, and validation of DriveRight, a low-cost, embedded AI-based driver-assistance system capable of real-time hazard detection and alerting. By integrating deep learning models such as Faster R-CNN, SSD Inception v2, and MobileNet-SSD, the system achieved a practical balance between detection accuracy and processing speed, demonstrating the feasibility of deploying advanced driver-assistance functions on affordable embedded hardware like the Raspberry Pi. Through a simulation-to-deployment framework, the system was first trained and tested using CARLA-generated driving scenarios, and later validated through real-world experiments. Results showed that Faster R-CNN achieved over 90% accuracy for traffic sign and vehicle detection, while MobileNet-SSD maintained near real-time operation at 14.6 FPS. Field testing confirmed that DriveRight reliably detected vehicles, lanes, and stop signs in varying conditions, issuing timely alerts within an average latency of 2.8 seconds, all while operating under 12 W power consumption. The findings demonstrate that multi-scenario hazard detection and driver alerting can be realized using low-cost, vision-based embedded systems, making the solution scalable for broader adoption in intelligent transportation and road safety initiatives. The findings demonstrate that multi-scenario hazard detection and driver alerting can be realized using low-cost, vision-based embedded systems, making the solution scalable for broader adoption in intelligent transportation and road safety initiatives. Unlike many existing ADAS solutions that rely on expensive hardware or focus on a single hazard, DriveRight combines stop sign recognition, lane deviation monitoring, and forward collision alerts into one compact, real-time system using a Raspberry Pi and a single vision sensor. This directly addresses

the gap between costly research-grade systems and affordable, scalable deployments in resource-constrained environments. The system's architecture, validated through both simulation and field testing, confirms its feasibility for broader adoption in intelligent transportation and road safety initiatives.

VI. Limitations and Future Works

Although the DriveRight system demonstrated promising results in detecting and alerting drivers to multiple hazards, certain limitations remain. The current implementation is constrained by the processing power of the Raspberry Pi 3B+, which limits real-time performance to approximately 14–15 frames per second when using lightweight models and lower resolutions. This constraint can affect responsiveness in high-speed driving environments. Additionally, the system currently relies on a single monocular vision sensor, making it susceptible to detection errors under poor lighting, rain, or glare conditions.

Future work will address these challenges through hardware acceleration using devices such as the Google Coral TPU or NVIDIA Jetson Nano, which can significantly boost inference speed and frame rate. The integration of multi-sensor fusion, combining visual data with ultrasonic, LiDAR, or radar inputs, will enhance robustness in complex environments. Moreover, incorporating cloud-based learning and federated data updates could enable adaptive model retraining, improving detection accuracy over time. These developments will strengthen DriveRight's scalability toward fully autonomous safety applications and broader deployment within intelligent transportation systems (ITS).

Acknowledgment

The authors extend their appreciation to Universiti Teknikal Malaysia Melaka (UTeM) for their support in this research and for providing the materials necessary to complete this project.

Author Contributions

The authors' contributions are as follows: "Conceptualization, Jamil Abedalrahim Jamil Alsayaydeh and Ahamed Fayeez Bin Tuani Ibrahim; methodology, Mazen Farid; software, Jamil Abedalrahim Jamil Alsayaydeh; validation, Mazen Farid; formal analysis, Ahamed Fayeez Bin Tuani Ibrahim; investigation, Jamil Abedalrahim Jamil Alsayaydeh; resources Aqeel Al-Hilali; writing—original draft preparation, Jamil Abedalrahim Jamil Alsayaydeh and Safarudin Gazali Herawan; writing—review and editing, Aqeel Al-Hilali and Rex Bacarra; funding acquisition, Rex Bacarra and Safarudin Gazali Herawan.

Data Availability Statement

All the datasets used in this study are available from the Zenodo database (accession number: https://zenodo.org/records/17161110).

References

[1] J. C. Rodríguez-Quiñonez, J. J. Sanchez-Castro, O. Real-Moreno, et al., "A real-time vehicle safety system by concurrent object detection and head pose estimation via stereo vision," Heliyon, vol. 10, no. 16, Art. no. e35929, 2024, doi: 10.1016/j.heliyon.2024.e35929.

[2] J.-H. Han and D.-Y. Ju, "Advanced alarm method based on driver's state in autonomous vehicles," Electronics, vol. 10, no. 22, Art. no. 2796, 2021, doi: 10.3390/electronics10222796.

[3] S. Titare, S. Chinchghare, and K. N. Hande, "Driver Drowsiness Detection and Alert System," International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), vol. 7, no. 3, pp. 583–588, May–Jun. 2021, doi: 10.32628/CSEIT2173171.

[4] C.-C. Wang et al., "Driver Assistance System for Lane Detection and Vehicle Recognition with Night Vision," Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), 2005, pp. 3530–3535.

[5] D. Malimath and K. Jain, "Driver Drowsiness Detection System," *Bonfring Int. J. Software Eng. & Soft Computing*, vol. 6, Special Issue, pp. 58–63, Oct. 2016.

[6] F. Osmani and P. Wawage, "Real-Time Driver Drowsiness Detection System using Vision Transformer for Accurate Eye State Analysis," 2024 International Conference on Intelligent Systems and Advanced Applications (ICISAA), Pune, India, 2024, pp. 1-5, doi: 10.1109/ICISAA62385.2024.10829106.

[7] M. Nasser, T. Rashid, A. Ghanem, H. Saeedi and H. Mahasneh, "Design and Implementation of a Driver Drowsiness Detection System to Prevent Accidents Using Machine Vision," 2025 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2025, pp. 1-3, doi: 10.1109/ICCE63647.2025.10929942.

[8] B. Yazici, A. Özdemir and T. Ayhan, "System-on-Chip Based Driver Drowsiness Detection and Warning System," 2022 Innovations in Intelligent Systems and Applications Conference (ASYU), Antalya, Turkey, 2022, pp. 1-5, doi: 10.1109/ASYU56188.2022.9925481.

[9] G. Rodríguez-Canosa et al., "Detection and Tracking of Dynamic Objects by a Multi-robot System," Sensors, vol. 14, no. 2, pp. 2911–2943, 2014.

[10] Hu, Y., Chen, N., Hou, Y. et al. Lightweight deep learning for real-time road distress detection on mobile devices. Nat Commun 16, 4212 (2025). https://doi.org/10.1038/s41467-025-59516-5.

[11] G. Gurulakshmanan, A. S, G. G S, D. Devi S, S. Inbarajan and Y. S, "IoT-Driven Convolutional Neural Networks for Effective Vehicle Night Vision and Image Recognition," 2025 11th International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 2025, pp. 1885-1890, doi: 10.1109/ICCSP64183.2025.11089296.

[12] M. Zaidi, H. Daud, M. Shafique and H. A. Jamal, "Lane Detection in Autonomous Driving: A Comprehensive Survey of Methods and Performance," 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), Tandojam, Pakistan, 2024, pp. 1-6, doi: 10.1109/KHI-HTC60760.2024.10482192.

[13] A. Dosovitskiy et al., "CARLA: An Open Urban Driving Simulator," Proc. 1st Annual Conf. Robot Learning (CoRL), 2017, pp. 1–16.

[14] J. Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," Proc. IEEE CVPR, 2016, pp. 779–788.

[15] M. Zaidi, H. Daud, M. Shafique and H. A. Jamal, "Lane Detection in Autonomous Driving: A Comprehensive Survey of Methods and Performance," 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), Tandojam, Pakistan, 2024, pp. 1-6, doi: 10.1109/KHI-HTC60760.2024.10482192.

[16] Z. Kaleem, "Lightweight and Computationally Efficient YOLO for Rogue UAV Detection in Complex Backgrounds," in IEEE Transactions on Aerospace and Electronic Systems, vol. 61, no. 2, pp. 5362-5366, April 2025, doi: 10.1109/TAES.2024.3464579.

[17] S. Kim, G. Kim, T. Kim, C. Jeong and C. M. Kang, "Autonomous Vehicle Control Using CARLA Simulator, ROS, and EPS HILS," 2025 International Conference on Electronics, Information, and Communication (ICEIC), Osaka, Japan, 2025, pp. 1-2, doi: 10.1109/ICEIC64972.2025.10879635.

[18] Y. Xiang, S. Wang, T. Su, J. Li, S. S. Mao and M. Geimer, "KIT Bus: A Shuttle Model for CARLA Simulator," 2021 IEEE Industrial Electronics and Applications Conference (IEACon), Penang, Malaysia, 2021, pp. 7-12, doi: 10.1109/IEACon51066.2021.9654633.

[19] MIROS (2015) – Miros statistics say human error causes 80% of traffic accidents, The Sun Daily, Feb. 2015.

[20] Roa-Tort, K.; Fabila-Bustos, D.A.; Hernández-Chávez, M.; León-Martínez, D.; Apolonio-Vera, A.; Ortega-Gutiérrez, E.B.; Cadena-Martínez, L.; Hernández-Lozano, C.D.; Torres-Pérez, C.; Cano-Ibarra, D.A.; et al. FPGA–STM32-Embedded Vision and Control Platform for ADAS Development on a 1:5 Scale Vehicle. Vehicles 2025, 7, 84. https://doi.org/10.3390/vehicles7030084.

[21] J. A. J. Alsayaydeh, Irianto, M. F. Ali, M. N. M. Al-Andoli and S. G. Herawan, "Improving the Robustness of IoT-Powered Smart City Applications Through Service-Reliant Application Authentication Technique," in IEEE Access, vol. 12, pp. 19405-19417, 2024, doi: 10.1109/ACCESS.2024.3361407.

[22] V. Shkarupylo, J. A. J. Alsayaydeh, M. F. B. Yusof, A. Oliinyk, V. Artemchuk and S. G. Herawan, "Exploring the Potential Network Vulnerabilities in the Smart Manufacturing Process of Industry 5.0 via the Use of Machine Learning Methods," in IEEE Access, vol. 12, pp. 152262-152276, 2024, doi: 10.1109/ACCESS.2024.3474861.

[23] M. N. Al-Andoli, Irianto, J. A. Alsayaydeh, I. M. Alwayle, C. K. N. Che Ku Mohd and F. Abuhoureyah, "Robust Overlapping Community Detection in Complex Networks With Graph Convolutional Networks and Fuzzy C-Means," in IEEE Access, vol. 12, pp. 70129-70145, 2024, doi: 10.1109/ACCESS.2024.3399883.

[24] J. A. J. Alsayaydeh, Irianto, M. Zainon, H. Baskaran, and S. G. Herawan, "Intelligent interfaces for assisting blind people using object recognition methods," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 13, no. 5, pp. 84–92, 2022, doi: 10.14569/IJACSA.2022.0130584.

[25] A. Stocker, T. E. Kalayci, M. Spitzer, and G. Musser, "Context-Aware Warning Systems: Leveraging Driving Environment Data for Improved Driver and Road User Warnings," in Proc. 21st Int. Conf. Web Information Systems and Technologies (WEBIST), 2025, pp. 573–581, doi: 10.5220/0013820600003985.

[26] V. Kovtun, E. Zaitseva, V. Levashenko, K. Grochla, and O. Kovtun, "Small Stochastic Data Compactification Concept Justified in the Entropy Basis," Entropy, vol. 25, no. 12. MDPI AG, p. 1567, Nov. 21, 2023. doi: 10.3390/e25121567.

[27] T. Wada, "ITS Wireless Communication System and Its Development for Autonomous Driving and Driving Support," IEICE ESS Fundamentals Review, vol. 12, no. 4, pp. 248-258, 2019.

[28] I. Grobelna, "Formal verification of control modules in cyber-physical systems," Sensors, vol. 20, no. 18, Art. no. 5154, 2020, doi: 10.3390/s20185154.

[29] V. Shkarupylo, I. Blinov, A. Chemeris, V. Dusheba, J. A. J. Alsayaydeh and A. Oliinyk, "Iterative Approach to TLC Model Checker Application," 2021 IEEE 2nd KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2021, pp. 283-287, doi: 10.1109/KhPIWeek53812.2021.9570055.

[30] P. Arora, M. Fozdar and V. Singh, "Impact of Plug-in Electric Vehicle and SVC in Congestion Management," 2022 IEEE 10th Power India International Conference (PIICON), New Delhi, India, 2022, pp. 1-6, doi: 10.1109/PIICON56320.2022.10045160.

[31] J. Reason, A. Manstead, S. Stradling, J. Baxter, And K. Campbell, "Errors and violations on the roads: a real distinction?," Ergonomics, vol. 33, no. 10–11, pp. 1315–1332, Oct. 1990, doi: 10.1080/00140139008925335.

[32] J. A. J. Alsayaydeh, Irianto, A. Aziz, C. K. Xin, A. K. M. Z. Hossain, and S. G. Herawan, "Face recognition system design and implementation using neural networks," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 13, no. 6, pp. 63–69, 2022, doi: 10.14569/IJACSA.2022.0130663.

[33] I. Zinovieva, N. Sytnyk, O. Denisova, and V. Artemchuk, "Support for the development of educational programs with graph database technology," in Data-Centric Business and Applications, A. Semenov, I. Yepifanova, and J. Kajanová, Eds., Lecture Notes on Data Engineering and Communications Technologies, vol. 195. Cham, Switzerland: Springer, 2024, ch. 14, doi: 10.1007/978-3-031-54012-7_14.