AN INTRA-VEHICULAR WIRELESS MULTIMEDIA SENSOR NETWORK FOR SMARTPHONE-BASED LOW-COST ADVANCED DRIVER-ASSISTANCE SYSTEMS

by

Christiaan Müller Fourie

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

October 2021

SUMMARY

AN INTRA-VEHICULAR WIRELESS MULTIMEDIA SENSOR NETWORK FOR SMARTPHONE-BASED LOW-COST ADVANCED DRIVER-ASSISTANCE SYSTEMS

by

Christiaan Müller Fourie

Supervisor:	Prof H.C. Myburgh
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Computer Engineering)
Keywords:	ADAS, ADAS and smartphones, IVWSN, object detection, WMSN

Advanced driver-assistance systems (ADAS) are more prevalent in high-end vehicles than in low-end vehicles. The research proposes an alternative for drivers without having to wait years to gain access to the safety ADAS offers. Wireless Multimedia Sensor Networks (WMSN) for ADAS applications in collaboration with smartphones is non-existent. Intra-vehicle environments cause difficulties in data transfer for wireless networks where performance of such networks in an intra-vehicle network is investigated.

A low-cost alternative was proposed that extends a smartphone's sensor perception, using a camera-based wireless sensor network. This dissertation presents the design of a low-cost ADAS alternative that uses an intra-vehicle wireless sensor network structured by a Wi-Fi Direct topology, using a smartphone as the processing platform. In addition, to expand on the smartphone's other commonly available wireless protocols, the Bluetooth protocol was used to collect blind spot sensory data, being processed by the smartphone. Both protocols form part of the Intra-Vehicular Wireless Sensor Network (IVWSN).

Essential ADAS features developed on the smartphone ADAS application carried out both lane detection and collision detection on a vehicle. A smartphone's processing power was harnessed and

used as a generic object detector through a convolution neural network, using the sensory network's video streams. Blind spot sensors on the lateral sides of the vehicle provided sensory data transmitted to the smartphone through Bluetooth.

IVWSNs are complex environments with many reflective materials that may impede communication. The network in a vehicle environment should be reliable. The network's performance was analysed to ensure that the network could carry out detection in real-time, which would be essential for the driver's safety. General ADAS systems use wired harnessing for communication and, therefore, the practicality of a novel wireless ADAS solution was tested.

It was found that a low-cost advanced driver-assistance system alternative can be conceptualised by using object detection techniques being processed on a smartphone from multiple streams, sourced from an IVWSN, composed of camera sensors. A low-cost CMOS camera sensors network with a smartphone found an application, using Wi-Fi Direct to create an intra-vehicle wireless network as a low-cost advanced driver-assistance system.

LIST OF ABBREVIATIONS

ADAS	Advanced Driver-Assistance System
ACC	Autonomous Cruise Control
BLE	Bluetooth Low Energy
CAN	Controller Area Networks
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
DNN	Deep Neural Network
ECU	Electronic Control Unit
FCA	Forward Collision Avoidance
FPS	Frames Per Second
IOU	Intersection Over Union
IVWSN	Intra-Vehicular Wireless Sensor Network
LIN	Local Interconnected Networks
MOST	Media-Oriented System Transport
OF	Optical Flow
RoI	Region of Interest
SAD	Sum of Absolute Difference
SLIC	Simple Linear Iterative Clustering
SoC	System on a Chip
SBZA	Side Blind Zone Alert
SVM	Support Vector Machine
TPMS	Tire Pressure Monitoring System
TSD	Traffic Sign Detection
UWB	Ultra-wideband
WMSN	Wireless Multimedia Sensor Network
WSN	Wireless Sensor Network
YOLO	You Only Look Once

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PROBLEM STATEMENT

1.1.1 Context of the problem

Vision-based vehicle detection has improved over the last couple of years. However, a better understanding of the vehicles on the road will continue to remain an active research topic in the years to come. There is a need to develop a system that would act as a more affordable advanced driver-assistance system (ADAS) alternative for drivers without having to wait years to gain access to the safety it offers. A Wireless Multimedia Sensor Networks (WMSN) network approach is a less expensive alternative to the industry standard of using a wired Controller Area Network (CAN). CAN is a robust protocol, but requires expensive equipment aimed at automobile manufactures. CAN bus lines also require physical wires to be set up in the vehicle, preventing or complicating it to equip a vehicle with such a device after being manufactured. Image processing techniques to be investigated will best suit the limited resources of a smartphone. Wired solutions of vision sensors in ADAS already exist, but are costly and do not cater for low-end vehicles. Wired off-the-shelf blind spot detection sensors primarily exist to assist in parking assistance kits, but would not be a good fit for the requirement of implementing a real-time feed of information while driving. The design and implementation of such a blind spot device would also need investigation.

1.1.2 Research gap

There is no active research on WMSN for ADAS applications, and its collaboration with smartphones is non-existent. Complementary Metal-Oxide Semiconductor (CMOS) cameras sensors have become more affordable and have shown to be helpful in ADAS systems but only exist as wired solutions; a wireless network of vision sensors has research potential [1]. Ultra-wideband (UWB) and Wi-Fi have high transmission rates that can be used for video transfer in smartphones, making them excellent additions to vision sensor networks. UWB would be the ideal choice because of UWBs low power

usage, high transmission rate and low cost. However, UWB modules in smartphones have only recently become commercially available in high-end models, such as the iPhone 11, subsequently leading to new research avenues. An Intra-Vehicular Wireless Sensor Network (IVWSN) can expand a smartphone's limited sensor perception to a cost-efficient camera-based ADAS system that uses real-time object detection to detect potential hazards around the vehicle's rear, blind spots and front spatial areas.

1.2 RESEARCH OBJECTIVE AND QUESTIONS

- Multiple video streams: Will a smartphone be able to handle multiple video streams from an IVWSN network and carry out image processing on multiple video sources?
- Low-cost network: Can a WMSN be implemented, using only a smartphone and wireless CMOS vision sensor nodes?
- Communication: Can the wireless video be transferred in real-time, using technologies, such as UWB or Wi-Fi, in a challenging intra-vehicle WSN environment?
- Feasible ADAS system: Can a wireless low-cost camera-only network ADAS system operate on a smartphone and benefit a driver as a low-cost option?

1.3 APPROACH

The approach which was followed to investigate the research questions, consisted of a comprehensive literature study on ADAS systems by focusing on their use within smartphones. Various techniques used by ADAS systems were studied, considering lane assistance, collision detection, blind spot detection and traffic sign detection. Furthermore, the literature study considers different network topologies being used by smartphones and their use within vehicles. Wireless vehicle networks propagation and data transfer were explored by considering IVWSN and available mobile network topologies that fall within a smartphone's capability.

Smartphones have inherent hardware restrictions. Traditional image processing techniques and more recent machine learning techniques, such as Convolutional Neural Networks (CNN), supported design choices. Wi-Fi and Bluetooth stacks, commonly available to smartphones, were studied to harness their wireless capabilities for an IVWSN. Sensors capable of communicating wirelessly with a smartphone, were considered. A low-cost network approach that does not require additional routers, was used. Due to Wi-Fi's higher data transfer rates, Wi-Fi Direct was the communication protocol for streaming video streams through the network. Wi-Fi Direct allowed the smartphone to initiate a self-hosted IP network, capable of multiple connections from external camera nodes. It was possible to use off-the-

shelf components, such as CMOS cameras, Bluetooth modules and Wi-Fi antennae to implement the electronic hardware for a low-cost vision-sensor node.

CMOS camera nodes were built, using low-cost off-the-shelf components. Different sensors were investigated to be used as vision sensors, provided that they were cost and power-efficient. The developed camera nodes had to connect to the network seamlessly to form part of the IVWSN network. Bluetooth communication was another wireless communication available to smartphones, being used for sensors that allow lower data transfer. By using Bluetooth and a proximity sensor, a blind spot detector will form part of the IVWSN network.

Finally, an Android application was built that would offer the driver ADAS capabilities by giving the driver visual feedback. The Android application development used a constructed video simulator, intended to stream multiple video streams. The detection algorithms were developed and tested, using the simulator. The algorithms were then ported to the Android application to investigate the system's performance in a real-time application. The network's performance was measured, using multiple simulated and actual camera nodes. All the nodes sent their data to a centralised smartphone that carried out object detection and image processing.

The fabricated camera and blind spot node prototypes were measured for practical performance and were used to validate simulated results from the simulator. Real-time video from multiple vision sensors was validated to acquire real-time performance results. Using the IVWSN and the ADAS system on the smartphone, the smartphone's visual feedback screen assisted the driver. Chapter 4 explains the designed and developed methodology.

1.4 RESEARCH GOALS

The research goal of this dissertation is to design, implement and investigate an IVWSN network for a low-cost ADAS system: (i) The vision sensors should wirelessly communicate with a smartphone, (ii) the network should allow for multiple video streams, (iii) multiple communication protocols generally used by smartphones should be used to form part of the IVWSN, (iv) the ADAS system should be low in cost and (v) the smartphone ADAS system should be able to carry out lane, collision and blind spot detection.

1.5 RESEARCH CONTRIBUTION

Using an IVWSN structured by a Wi-Fi Direct topology, a low-cost alternative was proposed that would extend a smartphone's perception, using a camera-based wireless sensor network. A smartphone's processing power was harnessed as a generic object detector through a convolution neural network to carry out ADAS functions. It was found that a more affordable advanced driver-assistance system alternative can be conceptualised by using object detection techniques, being processed on a smartphone from multiple streams sourced from an intra-vehicle wireless sensor network, composed of camera sensors. A low-cost CMOS camera sensors network with a smartphone found an application by using Wi-Fi Direct to form an intra-vehicle wireless network as a low-cost advanced driver-assistance system.

1.6 OVERVIEW OF STUDY

The dissertation is divided into five main sections, commencing with a literature study and background and leading to an implemented solution. In Chapter 2, the literature study investigates advances that ADAS systems have made, including different driver-assistance techniques and methods. Smartphones' applications in ADAS systems and how they have improved low-cost alternatives for drivers, are explored. By keeping the focus on smartphones and their compatible wireless technologies, wireless communication is considered for ADAS. Wireless networks are not standard in ADAS solutions, and alternatives are studied where vision and proximity sensors are used for monocular vision and blind spot detection.

Chapter 3 discusses all relevant techniques to be considered for an ADAS system that uses a wireless network. The chapter includes an in-depth investigation into object detection, from traditional image processing techniques to machine learning networks, using neural networks. Improvements of these techniques are studied, to be applied to smartphones. The research investigates different ADAS techniques, namely collision, lane and blind spot detection, which are the primary detection methods of this paper, even though different detection types are included throughout the paper.

The ADAS system is wholly dependent on the data received by the IVWSN. Wireless technologies available on smartphones are considered, namely Wi-Fi and Bluetooth. In Chapter 4, these two wireless technologies are explored to use their capabilities and technology stacks that sensors will then implement in the IVWSN. Different network topologies for multiple sensor use are examined, followed by detailed communication methods and protocols to communicate with different sensors

in the network. Detailed designs of the proposed camera node that uses video streaming through the network, presented. Additional sensory data is also provided by a detailed design of blind spot nodes. A comprehensive explanation of a developed video streaming simulator and smartphone ADAS application follows the hardware development, realising the complete solution.

In the concluding sections, Chapter 5 and Chapter 6, the system being designed and developed, are discussed. The network's results are divided into: propagation, performance and power consumption. Simulated and actual environments are shown, which are compared with controlled environments and actual intra-vehicle counterparts. The performance results of the ADAS techniques being implemented on the smartphone are illustrated and discussed for lane, collision and blind spot detection.

CHAPTER 2 LITERATURE STUDY

This chapter explains the basis of ADAS systems by investigating the main approaches and the advancements made in these fields of technology. Firstly, ADAS is reviewed, followed by a shift towards smartphones and their use within ADAS systems. Smartphones' relevance within this field is discussed and the possible scope for the technology to be applied as a cheaper alternative by replacing costly equipment, is inspected. Image processing techniques are compared by considering the manner in which image processing and machine learning assist in solving problems in the automotive industry.

Vision from inside vehicles is becoming more common, especially in autonomous vehicles. Sensory networks used within vehicles and how their application improves the awareness of a vehicle on the road are investigated. IVWSNs, focusing on different sensory types that cater to proximity and vision sensory, are inspected. The possibility of using vision in a sensory wireless network within a vehicle environment and how the signal's data transfer rate and propagation are affected by the vehicle's interior, such as bodywork, passengers, and car seats, are investigated.

2.1 ADVANCED DRIVER-ASSISTANCE SYSTEMS, USING SMARTPHONES

Smartphones are being used in forward, lateral and inside assistance ADAS application in object and lane detection, and tracking and traffic sign detection. The following sections discuss some of the main contributions. Many different fields of ADAS exists, namely: forward assistance, lateral assistance and inside assistance [2]. Fig. 2.1 shows an overview of the different areas that the ADAS systems consider.

Forward assistance includes Autonomous Cruise Control (ACC) that assists drivers in automatically keeping a safe driving distance and Forward Collision Avoidance (FCA), which provides a warning to the driver in the event of a potential accident. Forward assistant methods use radar and LiDAR. FCA

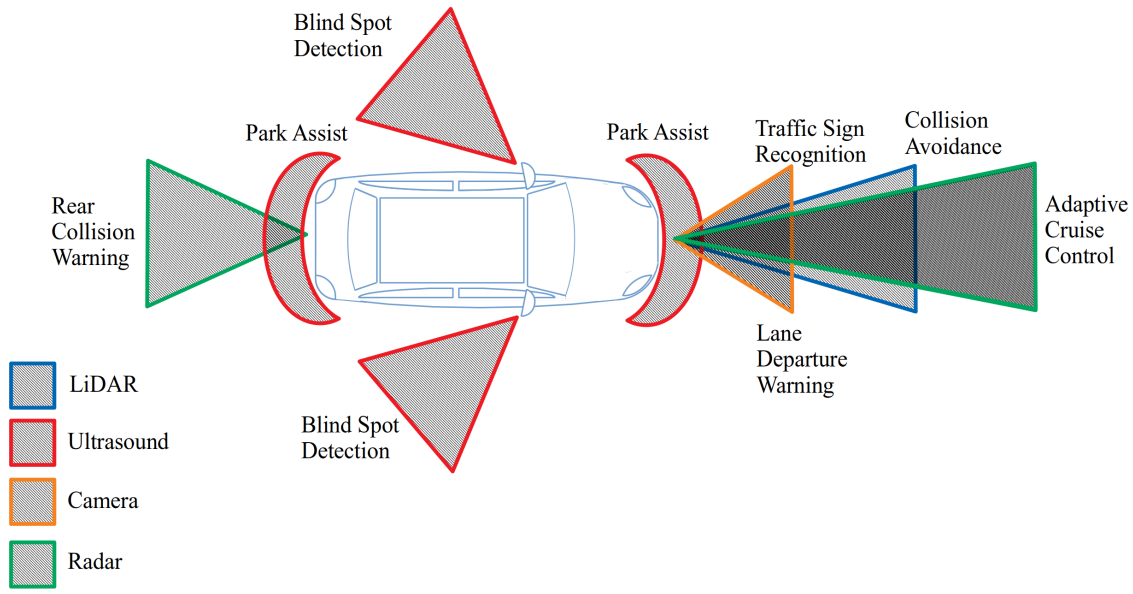


Figure 2.1. Overview of an Advanced Driver-Assistance System with areas showing different ADAS technologies.

also uses sensors, such as radar and LiDAR, but car manufacturers are using video in conjunction with radar, which has opened up a wide range of topics in research, using image processing in object detection, while sensor fusion techniques can be used to complement sensor devices for vehicle detection [3, 4, 5, 6]. ACC requires a high level of reliability in recognition of vehicles in dense traffic situations, whereby the use of multi-sensor tracking has shown to have positive outcomes, improving manoeuvring in traffic [3]. Sensor fusion allows the independent detection of objects of interest between sensors but, for example, if radar should fail, vision sensors will compensate with supporting evidence, which will prevent false positives.

Lane Change Assistance (LCA) deals with the areas exposed on the sides of the vehicle. ADAS supports the driver where the driver does not always have a clear view of the blindspots and when lane changing occurs, but LCA can also be included in forwarding detection where vision sensors detect lanes. ADAS alerts the driver when the vehicle seems to deviate from the lane or when the vehicle enters an unsafe driving distance.

2.1.1 Forward assist

2.1.1.1 Object detection and distance estimation

Smartphones are cheaper alternatives for forward assistance ADAS. Many methods of monitoring road and traffic conditions, using smartphones, have been proposed, where the first approaches used sensors, three-axis accelerometers and GPS [7, 8, 9, 10]. However, advances have been made to improve forward collision by detecting road hazards, such as speed bumps, with the help of image processing techniques [11]. Due to smartphone shortcomings, such as vibration patterns of the sensor data and GPS error, cameras facing the front of the road were used for road surface monitoring to detect speed bumps by using morphological image processing [12] and then, in a later study, used Gaussian filtering [13]. However, these two approaches were not implemented on a smartphone that could have harnessed the smartphone's camera and internal processing. Vehicle detection has been done using image processing, alternatives to radar and LiDAR by using Local Binary Patterns (LBP) and Haar-like features to train AdaBoost classifiers [14]. The image frames are searched and pruned to use different search windows instead of scanning the entire computationally intensive image. Different search windows are used, such as a low-resolution large image area, to detect vehicles close by and high resolution for the area closer to the horizon for vehicles further away. Fig. 2.2 shows the different detection windows mentioned to help determine vehicles being near or far.

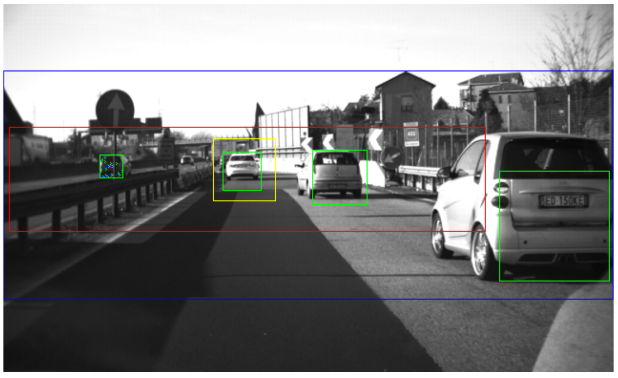


Figure 2.2. Three detection stages. Blue is the near area, yellow is the far window and red represents the intermediate window. Green boxes show potential vehicles being detected. Taken from [14], © 2015 IEEE.

Other monocular vision avenues have been investigated that detect pedestrians and vehicles, using Haar-like features and cascaded weak classifiers with distance estimation, which use a pinhole model described as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_u \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.1}$$

that assumes that the camera sensor is a point in space with focal lengths f_u and f_v to the 2D image coordinates (u, v) with a flat ground assumption of fixed phone height of the phone z, with the assumption that vehicles are on a flat plane [15]. In [16], the pinhole model was also used, but used the image pixel distance instead, d_1 and d_2, with the focal length f of the camera lens to the image sensor and the real-world ground projection distance Z to the detected object is determined, using

$$Z_2 - Z_1 = (\frac{1}{d_2} - \frac{1}{d_1} \times f \times H), \tag{2.2}$$

where H is the smartphone's mounting height. The related graphical representation is shown in Fig. 2.3.

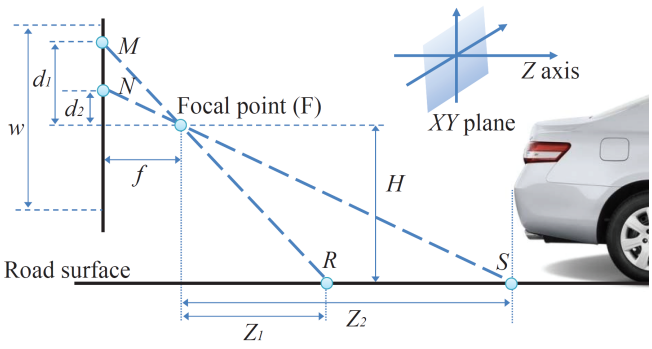


Figure 2.3. Pinhole model of monocular vision used to predict distance by using the camera image and estimated projected ground distance. Taken from [16], © 2013 ACM.

This method used monocular vision only, but smartphones have been used in object detection and tracking, using stereo vision. In [17], stereo vision uses two cameras to carry out 3D reconstruction in such a way that the cameras are calibrated to compute extrinsic parameters to calculate the necessary coordinate system in front of the vehicle. Edges in the images of the cameras pointing to the same area are then used as features to match and find similarities of the two images to perform the Sum of Absolute Difference (SAD) metric. Object detection is then carried out by finding superpixels (pixels with the same intensity and location), each with a 3D coordinate and height from the pitch angle, within the Region of Interest (RoI), using Simple Linear Iterative Clustering (SLIC), where points with heights higher than a set expected height are taken as potential obstacles. This approach is restricted for dual-camera smartphones and was limited to low speed in urban traffic, but can expand to use multiple cameras in a network that could interface with a smartphone to process 3D reconstruction.

A technique used extensively by ADAS systems in smartphones is the Viola-Jones algorithm that uses a sliding window over the whole image, looks for Haar-like features selected by AdaBoost and uses weak classifiers to detect objects [18]. Object detection using Convolutional Neural Networks (CNN), is used in smartphones that use pre-trained CNNs [19]. Object detection, using CNNs has been improving since R-CNN's inception, which uses a region proposal to extract potential bounding-boxes to run classifiers on these boxes instead of scanning the whole image to You Only Look Once (YOLO), that uses the entire image. It divides the image, predicts bounding-boxes and their confidence in being objects, and then predicts class probabilities. The images are analysed, and predictions are informed by the global context of the image [20, 21, 22, 23]. YOLO was shown to be the fastest real-time detector on PASCAL VOC at speeds of up to 155 FPS for Fast YOLO, currently being the best general detector that detects a variety of objects simultaneously and performs well to detect vehicles and other objects with its superior precision to recall ratio, as shown in Fig. 2.4 [23].

Smartphones are limited to computing power, local storage and excessive power consumption. The mentioned CNNs-trained models are too large for portable mobile devices and are improved by using MobileNets that uses depth-wise separable convolutions to build lightweight deep neural networks [24]. A trade-off between latency and accuracy exist but this can be adjusted, depending on the constraints of the problem.

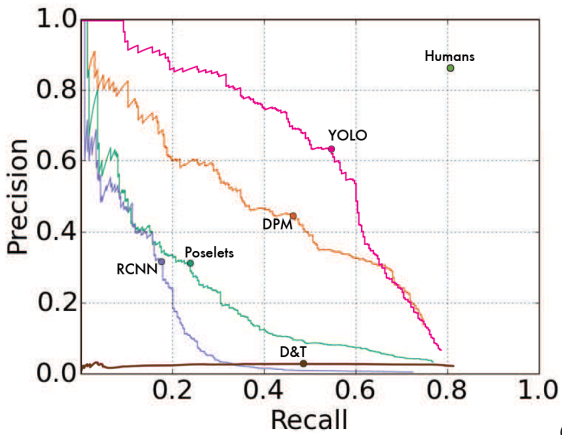


Figure 2.4. Precision-recall curve results for different object detectors, using the Picasso Dataset. Taken from [23], © 2016 IEEE.

2.1.2 Lateral assist

2.1.2.1 Lane detection

Lane detection ensures that the vehicle orientates between lanes and, if required, keeps the vehicle within the middle of the lanes by warning the driver if the vehicle is diverging. Lane detection comprises different steps, starting by finding the RoI, which assists in finding the area where the lanes are generally present, removing unnecessary processing. More preprocessing is then done on the image where noise has been eliminated, for example, grey-scaling, which makes road lines more prominent. After the preprocessing, edge detection techniques are commonly implemented on the image, one popular technique being Canny Edge Detection.

Different approaches to lane detection by using smartphones exist, such as LaneQuest, which uses the smartphone's GPS and inertial sensors [25]. DeepLane uses GPS and camera-assisted vision from the camera at the back of the smartphone to estimate the vehicle's lane position [19].

Lane detection technologies such as SmartLDWS showed successful lane tracking, consisting of a lane detection module that uses image processing to do edge detection through Hough transformation, where the processing was done on a smartphone attached to a vehicle's windscreen, thereby facing the front of the vehicle [26]. The real-time lane detection was also done in [27], with the addition of rear-end collision warning by using the camera of the smartphone to detect vehicles and their distance, using Canny Edge Detection and triangulation, respectively. Other vehicle detection techniques implemented on the DriveSafe application, initially developed for scoring driver behaviours [28], were expanded by using different detection windows to detect vehicles with the help of AdaBoost classifiers and tracking, using Kalman filters [14].

The lane detection methods being described, assume that roads are straight lines, where Hough transformation can predict lanes. A shortcoming in Hough transformation has been identified, where performance on curved roads diminishes. Algorithms do exist that cater for the curvature of the road, such as detecting painted lines on the ground captured by cameras, implementing LiDAR to reduce false positives and determining each lane's geometric information by calculating features from local maxima to determine Hermite splines to form continuous curves [29].

With the emergence of multi-core architecture in mobile devices, detecting road lane boundaries

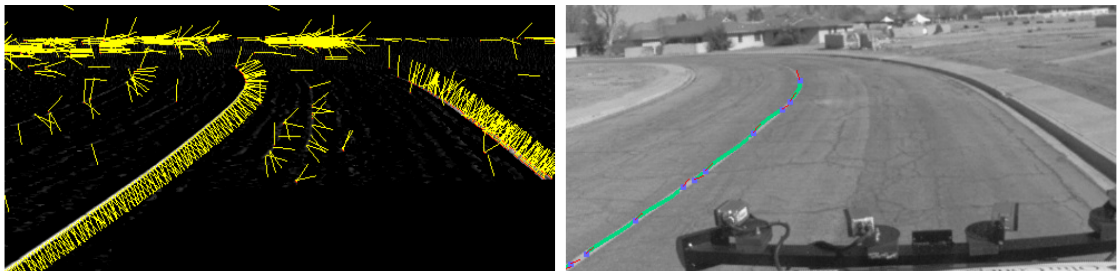


Figure 2.5. Direction of curvature attached to each feature from calculated local maxima (left) and a continuous curve, formed by connecting features by using cubic Hermit splines. Taken from [29], © 2009 Springer.

with curvature in real-time was made possible with more processing-intensive curve fitting by using second-order polynomials to approximate curves [30]. The best fitting curve is formulated by using the least-square error

$$LSE = \sum_{i=1}^{n} [y_i - (ax_i^2 + bx_i + c)]^2, \tag{2.3}$$

where (xi, yi) are the coordinates of an edge pixel and by using partial derivatives set to 0, yields coefficients (a, b and c) of the polynomial of the smallest error to find the best curve fit. DeepLane ignores road lanes entirely, which is beneficial in rural areas or crowded traffic lanes where lines are not visible. DeepLane does not track the lanes but uses optical flow (OF) to detect the flow of pixels and the flow of objects where the objects are detected, using YOLO [23]. By using DNN and deep-learning SegNet image segmentation removes objects that are not part of the road that produces noise [19]. Fig. 2.6 illustrates the two processes carried out by DeepLane to find road and vehicle information.

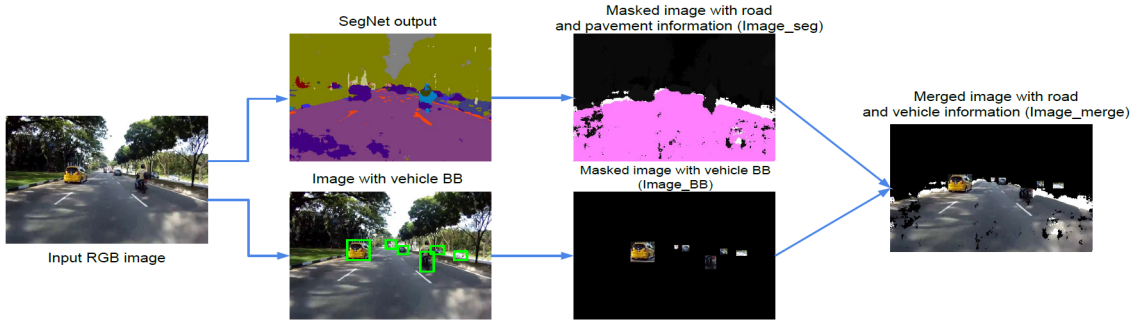


Figure 2.6. Two processes split of SegNet image segmentation and YOLO object detection with boundary boxes merged into an image with only road and vehicle information. Taken from [19], © 2018 ACM.

2.1.2.2 Traffic sign detection

ADAS uses traffic sign detection (TSD) to inform the driver of potential hazards or speed regulations on the road. Smartphone-powered low-cost approaches exist in smartphones. Real-time detection and classification are requirements for a constantly changing driving environment. Detection was limited

to traffic light detection of colour regions that were extracted, using histograms of oriented gradients (HOG) and LBP to obtain candidate regions, and for detection of traffic lights, a learning machine (K-ELM) was used, employing the smartphone's CPU and GPU to speed up computation for real-time application [31].

Contributions of TSD and Traffic Sign Recognition (TSR) have also been made that extend beyond only traffic lights only [32, 33, 34]. The SmartRoad mobile application employs crowd GPS sensing to detect generic stop signs and traffic lights by using statistical classification techniques [34]. This approach is practical only after collection of large amounts of crowd data. Improvement can be made by using crowdsensing in conjunction with real-time detectors, using vision-based sensings, such as in [32] where road signs being detected, are recognised with the actual speed of the smartphone's GPS to issue warnings to the driver. Another approach that is not limited to speed sign detection is a vision-based TSD and recognition (TSDR) system that uses feature vectors extracted via HOG and recognition by a Linear Support Vector Machine (LSVM) for real-time TSD [33].

2.1.3 Lateral assistance

This kind of ADAS approach tries to avoid collision with vehicles in the driver's blind spot or when the driver is unable to anticipate the speed of and the possible collision with another vehicle outside the driver's view.

IVWSNs have been used to develop low-cost substitutes of Side Blind Zone Alert (SBZA) systems that monitor blind spots of vehicles and alert users. An IVWSN uses Bluetooth Low Energy (BLE), which is a low-cost and low-power Bluetooth technology [35]. BLEs act as broadcast beacons from the wheels of the targets' cars to the detector car fitted with the receiver. The experiment shown in Fig. 2.7 conceptualises Tire Pressure Monitoring System (TPMS) sensors in cars that in principle could be used to do the broadcasting of beacon packets where a received signal strength indicator (RSSI) signal strength can be measured.

These IVWSN networks create potential opportunities for smartphones to access the network. However, very few smartphones can communicate to Zigbee communication. Nevertheless, they would be able to access these wireless sensor networks through a network topology, consisting of a base station [36].

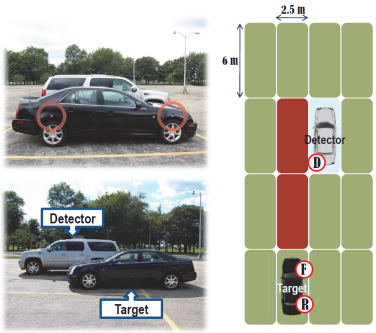


Figure 2.7. Experimental setup of BLE nodes on a target vehicle with the detection vehicle. Taken from [35], © 2015 IEEE.

Even though BLE node beacons use low-cost technology, the problem arises when vehicles without beacons, such as pedestrians, fall within the driver's blind spot. High-end commercial vehicles equipped with expensive radar and LiDAR made progress in mitigating these problems but come at high costs. Blindspot detection, using cameras pointing in the area of a vehicle's blind spots, has been studied, which shows valid detection results, using image processing techniques from the cameras mounted on side mirrors of vehicles [37, 38, 39]. Vision-based detection is primarily affected by weather conditions, as well as night versus day, which requires different image processing techniques. The weather is addressed by extracting regions, using image processing methods to detect targets, whereas, at night-time, the headlights of target vehicles are detected [37].

The application of smartphones' involvement with blind-spot detection is limited: SideEye attempted blind-spot detection on the driver's side of the car, using the front camera of the smartphone, as shown in Fig. 2.8. The front camera captures some of the blind-spot areas where image processing is used to detect other vehicles [40].

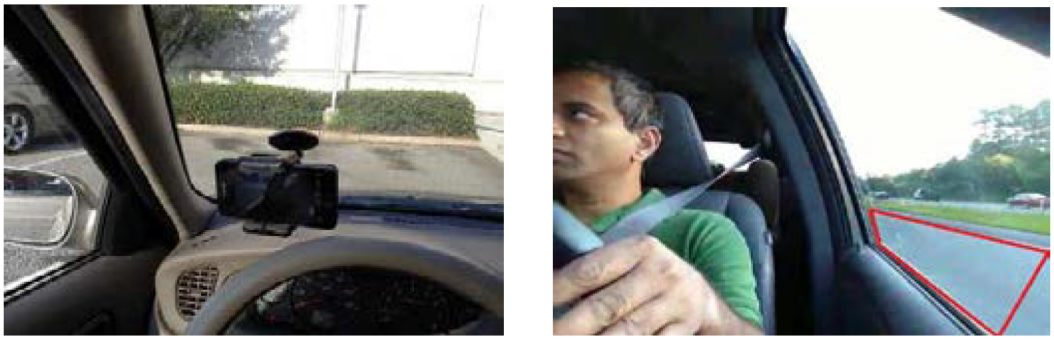


Figure 2.8. Mounted position of the smartphone and the smartphone's front camera, showing the region of interest. Taken from [40], © 2014 IEEE.

2.1.4 Inside assistance

Inside assistance includes analysing driving behaviour such as drowsiness, sleepiness and fatigue that could be detrimental to the driver [2]. A seminal paper on driver monitoring for intelligent vehicles categorises causes of inattention as a distraction and fatigue [41]. Smartphones' embedded hardware and cost-efficiency benefits have been used to address these driving behaviours [15], [16], [42, 43, 44].

Monitoring drivers' facial expressions help to detect driving behaviours such as sleepiness (e.g. the drivers might fall asleep at the wheel). Drivers also get distracted (e.g. taking their eyes off the road), irritability and careless lane changing. Algorithms are used to detect driving events while using the front camera of the smartphone to monitor the driver's facial expressions and gestures; intervention can be triggered before an accident would occur [45]. CarSafe was the first driver safety smartphone application on smartphones to use dual cameras where the front camera was used for monitoring the driver and the rear camera to detect following distance and lane drifting [16]. A graphical representation of the concept is shown in Fig. 2.9. CarSafe's implementation required switching between the rear and front cameras because smartphones at the time could not process both video streams; however, these constraints improved as smartphones became more powerful, and pedestrian and vehicle detection, as well as vision-based drowsiness detection, were shown to work simultaneously [15].

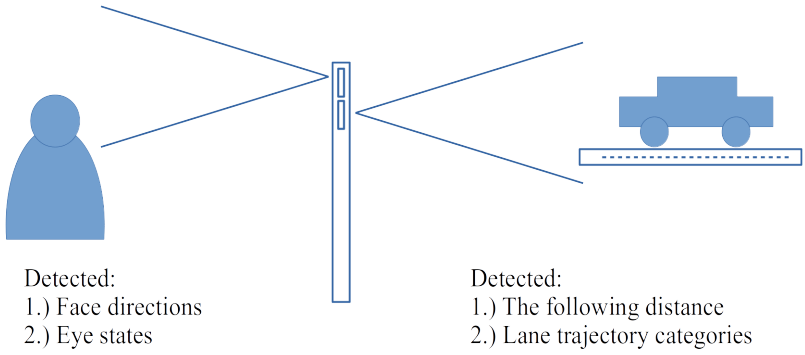


Figure 2.9. CarSafe uses dual-camera sensing, harnessing the sensing capability of the front and rear cameras of the smartphone. Adapted from [16], © 2013 ACM.

2.1.5 Smartphone requirements in ADAS

ADAS applications on smartphones have been implemented, using iOS, as well as Android operating systems, including CarSafe, SideEye and DeepLane [16, 14, 40, 19]. Vehicle detection was carried out in an iOS operating system on an iPhone 6, which allowed for real-time vehicle detection, tracking and lane detection [14]. iPhone 5's frame rates were operating at a slow 7.6 fps, which would not be successful in practice, but newer models with better GPUs and RAM perform at usable real-time speeds. The Android smartphones have an extensive range of different hardware combinations, but as shown by DeepLane, smartphones with better GPUs run at higher frame rates, making real-time applications reachable [19].

Development for iOS is exclusively limited to iPhones, and development is restricted to the macOS operating system. To reach a low-cost audience, development for an Android device is more appealing, and development can be done on different operating systems such as Windows, macOS and Linux. Benchmark testing exists that separate mobiles in low-end, mid-range and high-end devices. All devices can be placed through a benchmark test, which can indirectly indicate the possibility of using the device for an ADAS system application.

2.2 WIRELESS SENSOR NETWORKS FOR ADAS

Most commercial vehicles have wired internal networks that communicate information to the electronic control unit. Controller Area Networks (CAN) and Local Interconnect Network (LIN) are the most common automotive industry-standard communications between sensors and peripherals because of CAN protocol's reliability and robustness. CAN have shown to have bandwidth limitations on the CAN bus with the introduction of camera-based ADAS systems [46]. Other wired networks, also used in vehicles, exists with varying bitrates, as shown in Table 2.1, that have higher bandwidth, capable of transmitting vision information, such as Media-Oriented System Transport (MOST) [47].

Table 2.1. Current automotive phyical layer technologies. Taken from [46], © 2014 IEEE.

Protocol	Bitrate	Medium	Communication
LIN	19.2 Kbps	Single Wire	Serial
CAN	1 Mbps	Twisted Pair	CSMA/CR
FlexRay	20 Mbps	Twisted Pair/Optical Fibre	TDMA
MOST	150 Mbps	Optical Fibre	TDMA
LVDS	655 Mbps	Twisted Pair	Serial/Parallel

2.2.1 Wireless networks in vehicles

IEEE 802 and Ultra-wideband (UWB) solutions are currently being investigated, but because they still require an electrical power source from the vehicle, the advantage is mitigated [46]. However, low-end cars that do not have factory-built ADAS systems could still use such a wireless system, using power sourced from around the vehicle, such as USB powered outlets.

Wired architectures in vehicles are the most traditional, but new wireless network alternatives exist, such as IVWSN, which are being considered by the automotive industry. With the help of wireless technologies, the vehicle's harnessing is reduced, thereby lowering costs and fuel consumption [35]. However, as shown in Fig. 2.10, the inner vehicle is a challenging environment, for radio propagation because of metal objects and passengers inside the vehicle [48]. As a result of this complex environment, design considerations should be carefully considered when developing a wireless sensor network inside a vehicle where the reliability of the network diminishes as traffic on the network increases, causing delays in information being transferred [36], [49].

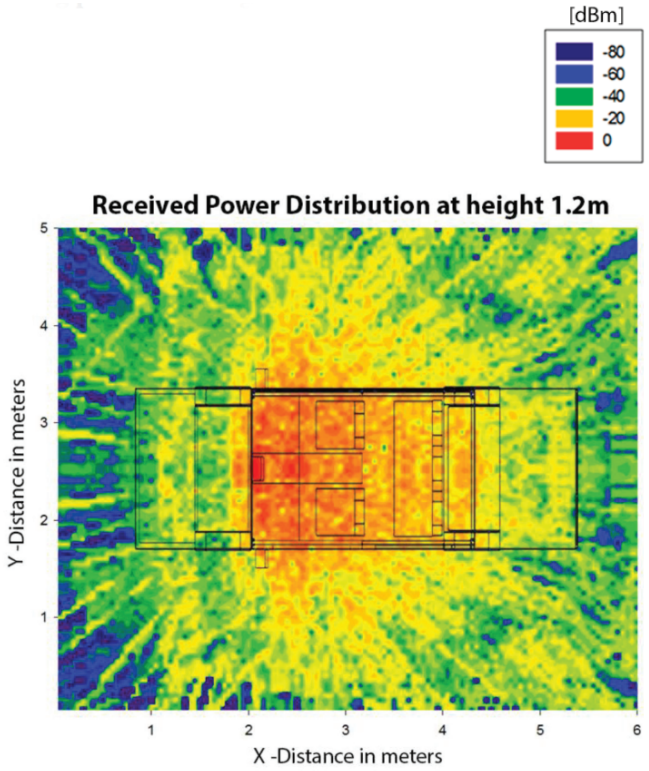


Figure 2.10. Power coverage of Zigbee in an intra-vehicle environment. Taken from [48], © 2014 IEEE.

Currently, different network compatible devices are available, e.g. Wi-Fi, Bluetooth, UWB and Zigbee. Bluetooth transmission at high speeds requires high power usage, which is not ideal for battery-enabled sensors in vehicles. Zigbee, having a more prominent communication protocol, allows for real-time monitoring at a low cost. An IoT-IVWSN network consisting of end-devices, control unit and display that has a large number of end-device sensors, was shown to be a good alternative in vehicles to control and manage sensors [49]. Zigbee technology was shown to be useful for sensors that do not require high data transmission rates. However, when transferring multimedia and real-time video streaming, it would not be a practical solution. There is no built-in support for Zigbee that can split up large amounts of data into smaller packets. As shown by Table 2.2, Wi-Fi and UWB technologies are better suited for faster data transmission, where UWB would be a cheaper design solution. Sensors that are capable of collecting multimedia data require computation-intensive processing, which may require processing to be done on the sensor nodes or computational hubs where Wireless Multimedia Sensor Networks (WMSN) enable a new approach to perform distributed computations on nodes [50].

Table 2.2. Comparison of existing wireless technologies for IVWSNs. Taken from [49], © CC BY.

Standard	Bluetooth	Wi-Fi	UWB	Zigbee
IEEE spec	802.15.1	802.11 a/b/g	802.15.3a	802.15.4
Frequency band	2.4 GHz	2.4 GHz; 5 GHz	3.1-10.6 GHz	868/915 GHz; 2.4 GHz
Max signal rate	1 Mb/s	54 Mb/s	110 Mb/s	250 Kb/s
Nominal range	10 m	100 m	10 m	10-100m
Max cell nodes	8	2007	8	> 65000
Chip price	$5	$20-25	$1	$2
Data protection	16-bit	32-bit CRC	32-bit CRC	16-bit CRC
Topology	Star	ESS, BSS	Peer-to-peer	Tree, star, mesh

UWB has a transfer speed of over 100 Mbps and benefits the transmission of multimedia applications such as video. Fig. 2.11 shows how transmission time is affected by the increase of payload size. The data size for monitoring and controlling sensors within a vehicle would function better through Bluetooth and Zigbee protocols, showing better data coding efficiency. However, UWB and Wi-FI are better solutions for high data rate implementations, such as multimedia video, because of its high data rate and low power [51].

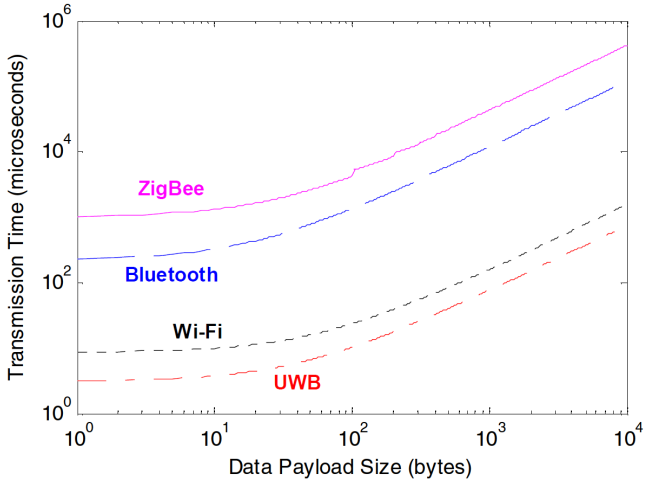


Figure 2.11. Different protocols and how the transmission time is affected by the increase of payload size. Taken from [51], © 2007 IEEE.

2.3 VISION SENSORS FOR ADAS

Vision-based sensors have become very popular in ADAS systems because of their low cost, compared to radar and LiDAR, but vision-based sensors perform poorly in unfavourable weather conditions. Measuring the distance of an object, using vision sensors, is also less accurate than LiDAR and radar, whereby using stereo vision (dual cameras) needs a wide base length between cameras for triangulation purposes; whereas radars' robustness against weather and the ability to determine distance more accurately can be harnessed instead and have shown to work well in multi-sensor scenarios that lower false positives in detection [4], [52], [53]. Sensor fusion was used to combine both individual detection systems, such as the camera's object detection and LiDAR detection, for more robust detection.

Off-the-shelf System on a Chip (SoC) solutions, such as the Texas Instrument's TDA2x product range [54]. Unfortunately, this is not a low-cost solution, but it does give a good overview of a high-level structure at designing cheaper alternatives. SoC uses multiple cameras as sensor inputs being fed to the vehicle where the processing units process the incoming video. These devices are expensive and are manufactured for commercial ADAS systems, whereas smartphones contain GPUs and CPUs, capable of image processing. This processing advantage with low-cost off-the-shelf CMOS cameras could be integrated within an intra-vehicle network.

Two primary image sensors, namely Charge-Coupled Device (CCD) and Complementary Metal-Oxide Semiconductor (CMOS) convert optical images to electrical signals. When comparing the two, it was

shown that CCD has better image quality, light sensitivity and resolution, while CMOS sensors are faster, smaller, cheaper and more power-efficient, with a higher image rate [1].

A network of wirelessly interconnected devices that retrieve video and audio streams is known as Wireless Multimedia Sensor Networks (WMSN), where low-cost CMOS cameras, supplying multiple media streams, can provide a multi-resolution description of scenes. The networks can be branched out by interconnecting “islands” of sensors, but this would not be important with a small network for vision sensors within a car, but could be advantageous for Vehicle-to-Vehicle (V2V) communication. Show in Fig. 2.12, are three sensor networks with different characteristics. The topology on the far left shows processing hubs where each sensor is responsible for its processing, forming a distributed processing architecture where in the middle, a topology video is being relayed to a central processing cluster head [50].

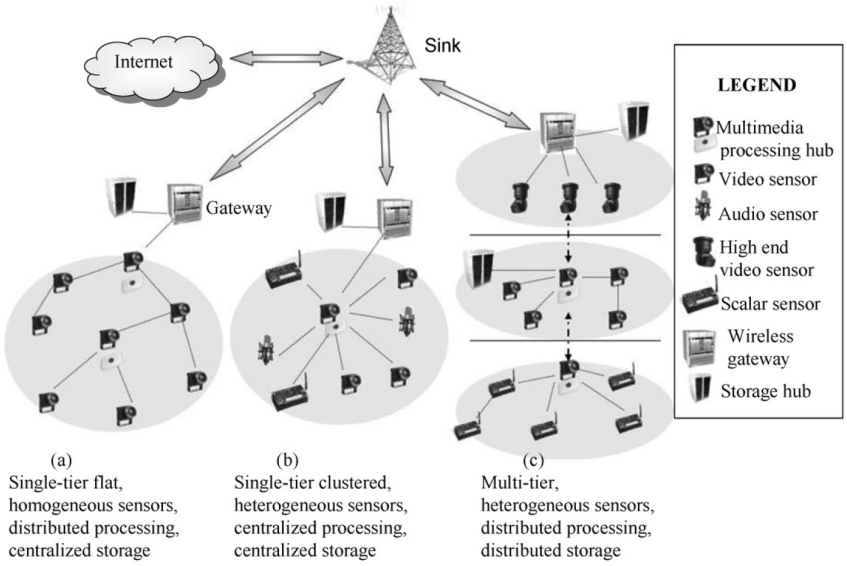


Figure 2.12. Proposed architectures for wireless multimedia sensor networks with different processing and storage methods for different needs and scenarios. Taken from [50], © 2007 Elsevier.

2.4 PROXIMITY SENSORS FOR ADAS

ADAS systems depend on sensor information from the surrounding environment, processed by the Electronic Control Unit (ECU) in high-end vehicles. These sensors include radar, LiDAR, ultrasonic sensors and IR sensors, and accommodate ADAS features, such as lane departure warning, parking assistance and blind spot monitoring [55], [56].

Ultrasonic sensors are among the cheapest sensors in ADAS systems, which are primarily used to find distances of static objects at a short-range and at slower speeds [55]. Blind spot detectors in high-end vehicles use radar sensors placed at strategic positions on the body of the vehicle. Ultrasonic sensors are used to assist drivers with parking, but could be used in other features such as blind spot detection. Ultrasonic sensors have a shorter range than radar, limited to a range of 2 metres, but could serve as a cheaper, less power consuming alternative for low-cost ADAS systems.

2.5 CONCLUSION

In future, modern vehicles will be equipped with more wireless sensors, forming part of an IVWSN. Wired networks in vehicles are the standard form of communication, but wireless alternatives have been investigated for an alternative cost-efficient solution. A lower-cost ADAS system allows low-end vehicle drivers to be equipped with added safety that is produced by ADAS. Different wireless technologies were considered by taking transfer speeds, payload sizes and costs into consideration. By using vision and proximity sensors, a wireless network can be developed that uses a smartphone's processing, which was shown to be capable of carrying out different driving assistance methods. Advancements in CNNs have improved object detection in embedded and mobile devices, allowing for improvements in object detection and distance estimation methods on a smartphone. A wireless alternative is considered to transfer video streams and other information, such as proximity sensor data, through a wireless sensory network. An IVWSN is a complex environment with many reflective objects within the driving cabin where data transfer and performance of a wireless network are unknown. This paper aims to investigate these problems.

CHAPTER 3 METHODS AND BACKGROUND TECHNIQUES

Object detection consists of a wide range of specialised topics. In the last decade, a considerable amount of progress has been made in machine learning and image processing, which is under constant improvement. This paper studies the most commonly used methods in the object detection field while considering that the focus is structured around vehicles by applying the discussed methods to ADAS systems.

Vehicle detection and tracking, using Viola-Jones algorithms, are summarised, followed by Convolutional Neural Networks (CNN), examined to form a more holistic object detection approach. CNNs allow for the detection of a collection of objects such as humans, vehicles and animals. CNNs have been shown to perform well in image processing and have become the most common image detection techniques. Well-known CNN methods will be discussed, including R-CNN, Fast CNN, Faster CNN, and the latest, better performing YOLO. CNN methods being mentioned, expand on one other, resulting in improvements on the next. Ultimately, this progress allows real-time computation of object detection. Real-time object detection is a required attribute of ADAS systems of vehicles. CNN models can become large and consume memory and processing that is not always available on mobile devices where MobileNet optimise CNNs for smaller embedded devices.

ADAS collision detection uses detection of objects. By using collision detection, the driver is warned visually of any potential collision. The system supplies essential ADAS features that determine whether the object is within proximity of the driver's vehicle or diverging on a driving lane. Distant estimation techniques and lane detection methods are studied. The most commonly-used traditional image processing techniques are explained, including the Canny and Hough transformations.

Knowledge about the vehicle's surroundings is required to allow for computation to occur on an ADAS system. Wireless sensors capture various information around the vehicle. Different sensors are discussed, including camera sensors to capture image frames around the vehicle. Different kinds of blind spot sensors and techniques are investigated. Lastly, a Wireless Sensor Network (WSN) is discussed to transfer the captured data for processing. Wi-Fi and Bluetooth are used as the wireless protocols for the network to form the Intra-Vehicular Wireless Sensor Network (IVWSN).

3.1 IMAGE PROCESSING AND MACHINE LEARNING TECHNIQUES

3.1.1 Viola-Jones object detection

The Viola-Jones algorithm is based on a sequential classifier that uses Haar-like features being used in face detection problems. Researchers have proposed several approaches to classification algorithms. There are too many to be discussed in this paper, and the most popular well-performing classifier, known as AdaBoost, is studied. Even though Viola-Jones was originally used for face detection, it has been used in other object detection settings, such as vehicle detection [14], [57].

Viola-Jones provides good object detection ratings where the training is slow, but the detection is fast, making it favourable for real-time applications. Features of Viola-Jones have a very high detection rate and low levels of false positives. The approach is seen as a detection method and not recognition, since the algorithm only determines if, for example, the object is a face or not but cannot discern between distinct faces. The method consists of different steps whereby alterations are common under different researchers. The main topics of the Viola-Jones algorithm consist of Haar-like features selection, summed-area or integral table, AdaBoost, and finally, cascading classifiers.

3.1.1.1 Haar-like feature selection

Haar-like features is a way to categorise subsections of images by using adjacent rectangular regions, that uses pixel intensities of each region, to find the difference between these sums. The Viola-Jones algorithm uses four key features and four rectangle features, as seen in Fig. 3.1.

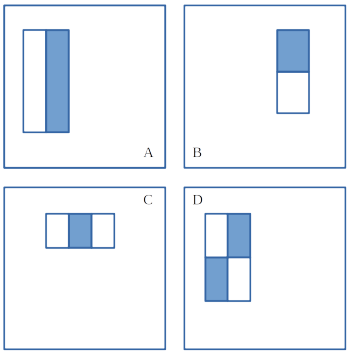


Figure 3.1. Example of rectangle features, shown relative to the enclosing detection window. Taken from [18], © 2001 IEEE.

If face detection is taken as an example, the features can indicate standard features unique to a face. The Haar-like feature is selected and applied to the detection window by taking the sum of the pixels in the grey area and subtracting the result from the sum of the pixels in the white area relative to the enclosing detection window. Viola and Jones came up with an intuitive approach to allow fast computation of Haar-like features by using summed-area tables, by calculating the integral image as

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y'). \tag{3.1}$$

Using the graphical representation of Fig. 3.2 and Equation 3.2, the computation of an integral image can be explained. By using an integral image, any rectangular sum can be computed in four array references [18]

$$I(x,y) = i(x,y) + I(x,y-1) + I(x-1,y) - I(x-1,y-1). \tag{3.2}$$

For example, using Fig. 3.2, it can be assumed that $\sum i(x,y) = I(4) + I(1) - I(2) - I(3)$.

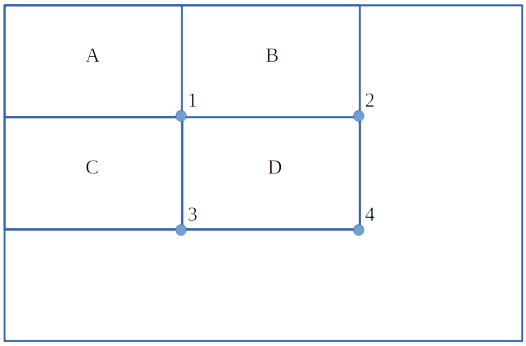


Figure 3.2. Graphical representation of Equation 3.2.

By using the integral image, any rectangular sum can be computed with four array references. The difference between two rectangles can be calculated by using the same logic as when using eight references.

3.1.1.2 AdaBoost

Ensemble methods use a pool of learning algorithms to obtain better predictive performance than what could have been obtained from any of the elementary learning algorithms alone. AdaBoost is favourable because it helps to form a robust classifier with the help of many weak classifiers. Exercising face-detection, using Viola-Jones Haar-like features, would require the processing of 160 000+ features. Using AdaBoost, helps to inspect which of the features will be a relevant feature for classification. In training the AdaBoost, the features are checked for error rates, and if the error is low in training, then the classifier is added to AdaBoost to form part of the classifier [58].

AdaBoost combines a lot of "weak" learners to make classifications. The "weak" learners are mostly stumps. Some stumps are regarded higher than others in the classification. Many Viola-Jones Haar-like features, using complex classifiers, will cause over-fitting, but AdaBoost uses "weak" classifier stumps, which will prevent this. AdaBoost can be added to a classifier as it is a technique that builds on top of other classifiers, as opposed to being a classifier itself. By using integral image processing, there will be features with an object and features that do not contain an object. There will be many features where AdaBoost will help, by using weak classifiers and cascading them. AdaBoost assists to set weights on both classifiers and samples to enable the classifiers to focus on observations that are difficult to classify. Being an ensemble method, AdaBoost allows learning to combine with several models to allow for final predictive outcomes and an improvement of performance.

The boosting algorithm can be explained by using the following training samples

$$(x_1, y_1), ..., (x_m, y_m), \tag{3.3}$$

where x_* is the inputs and y_* is the outputs limited to the elements $y_i \in \{-1, +1\}$ with weights defined, initialised initially

$$D_1(i) = \frac{1}{m} \text{ for i=1 ,... ,m .} \tag{3.4}$$

D is the weights of the samples, where Equation 3.4 indicates the weights in the first iteration of AdaBoost that should be initialised for all weights by 1 divided by the number of samples. The "weak" classifiers are picked in the next step to formulate a hypothesis, such that $h_t \in \{-1, +1\}$. For each round T, the distribution D_t for $1, ..., T$ a "weak" classifier/hypothesis is selected relative to D_t with a

low-weighted error fulfilled by

$$\varepsilon_t = Pr_{i\sim D_t}[h_t(x_i) \neq y_i], \tag{3.5}$$

also intuitively explained as

$$\varepsilon_t = \frac{\text{misclassification rate of t}}{\text{N total weights of distribution of t}} = \frac{\sum_{i=1}^{N} w_i I(y_i \neq h_t(x_i))}{\sum_{i=1}^{N} w_i}. \tag{3.6}$$

Correctly classified cases have a decrease in weight to prevent the classifier from adding attention to the classifier. In contrast, misclassified cases require the weight to be increased so that the classifier can emphasise correcting the misclassification. For the round $t+1$ new weights need to be calculated, using the factor

$$\alpha_t = \frac{1}{2} ln(\frac{1-\varepsilon_t}{\varepsilon_t}). \tag{3.7}$$

Then, update the new weight values for each sample $i = 1,...,m$ where Z_t is the normalisation factor

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}. \tag{3.8}$$

Once enough "weak" classifiers have been accumulated, a stronger hypothesis can be constructed with a summation

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)). \tag{3.9}$$

The final step in the Viola-Jones method is cascading the classifiers. Each stage in the cascade consists of bundled classifiers, as shown in Fig. 3.3. As the detection window moves from the top left to the right, gradually making its way down the image, the detection window can quickly move on to a new position without wasting computation time if the first few stages in the cascade have rejected the windows as having a definite object. If, however, the detection is positive in the first few stages, the testing moves deeper into the cascading classifiers to further inspect the detection of a potential search for an object.

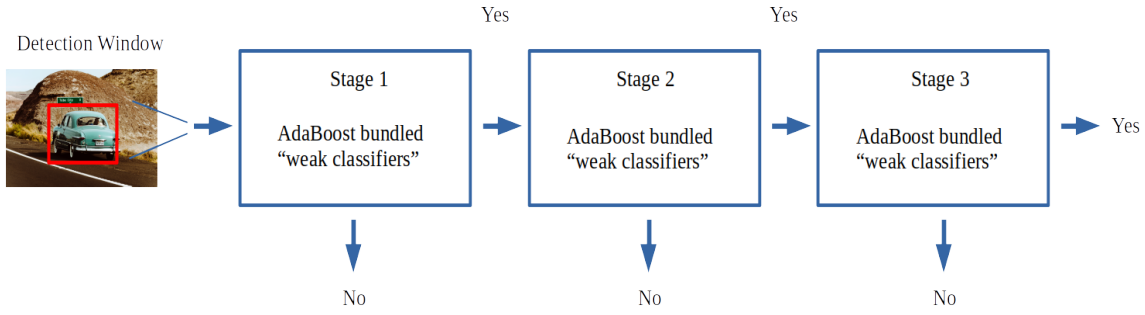


Figure 3.3. Cascading classifiers, showing the stages of a potential object in a detection window, being passed.

3.1.2 Convolutional Neural Networks

A convolutional neural network consists of deep neural networks and is extensively used within image processing. The network consists of hidden layers in-between the input and output layers. The hidden layers consist of computation steps, such as activation functions and filtering, which use kernels that derive activation maps and pooling, e.g. max pooling and fully connected layers. A CNN helps to reduce the image to a pattern that is processed easier, resulting in improved predictions.

CNNs provides the ability, with enough learning, to allow the network to acquire filters and characteristics of the network where other classification methods require several pre-processing and manual adjustments of filtering. The network allows adjustment of filters and reduces processing where training can assist in increased understanding of the image. Fig. 3.4 shows a basic structure of a typical CNN architecture. A detailed discussion of the different components that accounts for the building blocks of a CNN will follow. CNNs have different architectures and patterns, and can be trained on different datasets. For example, CNNs can be pre-trained on a large dataset such as ImageNet or AlexNet [20] and then tuned for much smaller class amounts, such as for the PASCAL VOC challenge with minor modifications.

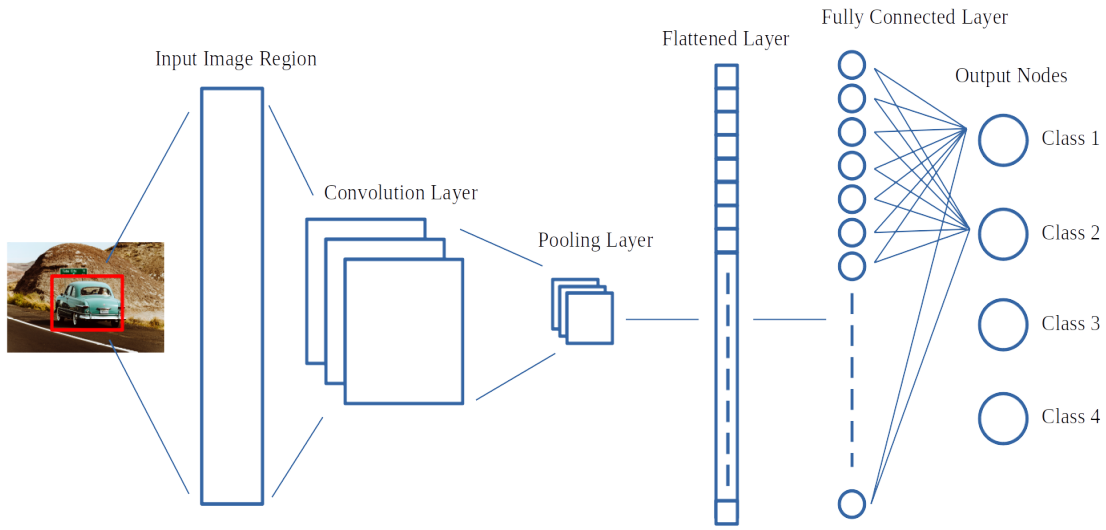


Figure 3.4. Common building blocks of a CNN.

3.1.2.1 Convolutional layer

The first hidden layer of a CNN is a convolutional layer, which consists of a kernel for filtering. A kernel, also known as a convolutional matrix, is convolved around the image to get a filtered outcome. Different kernels for different applications of filtering are known. The process of convolution consists of an input feature map that convolves with the kernel to produce an output feature map. In a graphical illustration, as depicted in Fig. 3.5, the kernel matrix is moved from left to right over the input feature map with a set step size, called the stride. At each incremental location, the product is calculated between each value of the kernel and the input element that it overlaps. The result is summed to obtain the output of the current location.

$$g(x,y) = w * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t), \tag{3.10}$$

where $g(x,y)$ and $f(x,y)$ are the input and output feature maps of the CNN, respectively. w represents the kernel with coordinates as s and t. The example given in Fig. 3.5 is a 2-D convolution, but can be extended to N-D convolutions.

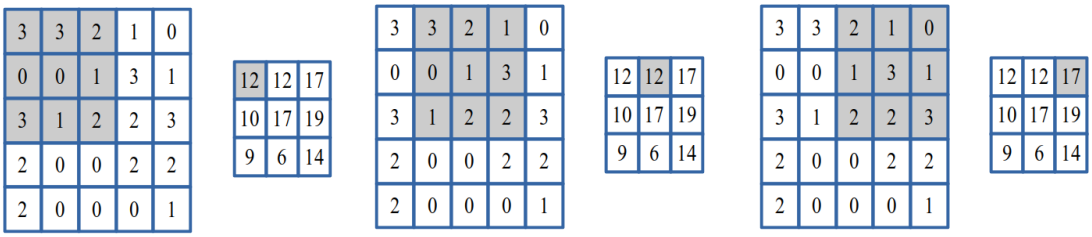


Figure 3.5. The first three iterations in computing the output values of a discrete convolution.

A collection of kernels can be used where the collection of N-depths is moved in the same fashion across the image, with strides from left to right, and then hops down to the beginning, moving from left to right once again until the entire image has been traversed. If an image has multiple channels of colours, e.g. Red, Blue and Green (RGB), the kernel of the same channels will then go through matrix multiplication where the results are summed, giving a one-depth channel convoluted output. If padding is added to the bounds of the input image and is convoluted with the kernel, the output image has the exact dimensions compared to the input, known as Same Padding. If, however, no padding is added, then the output feature's dimension decreases to that of the kernel's after convolution; this is known as Valid Padding [59]. The output feature can be the same or smaller than the input feature, depending on the application of either Valid Padding or Same Padding.

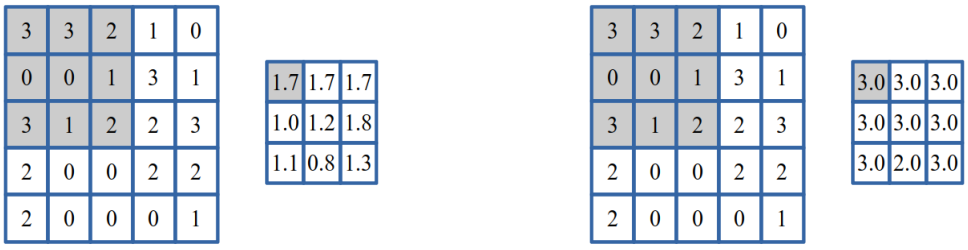
3.1.2.2 Pooling Layer

Another essential part of a CNN is pooling. Pooling reduces the size of a feature map. Reducing the feature map is necessary because this helps to reduce computational processing power with the reduction of dimensions. More dominant features of the image are emphasised and extracted more vigorously. Pooling uses a sliding window that moves in the same way as the convolutional layer but performs a relevant arithmetic pooling function instead. Two general types of pooling functions can be found namely Max Pooling and Average Pooling, as shown in Fig. 3.6 and respectively expressed as

$$f_{max}(X) = max_i \boldsymbol{x}_i \tag{3.11}$$

$$f_{avg}(X) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i. \tag{3.12}$$

The Max Pooling function takes the maximum value in the vector X in each window stride step as its output element. Average Pooling, however, sums up all the elements in the window vector and returns the average [60].



(a) Average pooling **(b)** Max pooling

Figure 3.6. Graphical representation of a pooling iteration.

The layers mentioned are not limited to a specific number of layers. However, they can be adjusted depending on the application, which increases the depth of the deep-learning network. The pooling layer outcome is not in a format that the neural network can interpret. The layers are converted to flattened layers to transpose the processed layers to a comprehensible form to be used directly into a Fully Connected (FC) layer.

3.1.2.3 ReLU Layer

ReLU is an element-wise operation applied to every pixel whereby each pixel is replaced with a negative value with a zero. ReLU can be seen as a layer since it can be applied to the CNN architecture as permitted or required. ReLU is usually applied after a convolution layer

$$f(x) = max(0,x). \tag{3.13}$$

ReLU helps in speeding up the processes of training. The computation of ReLU is also not a process-intensive step to add to the CNN. The primary purpose of ReLU is to introduce non-linearity into the CNN network, since real-world data is non-linear. Convolution layers and pooling still constitute linear operations (such as matrix multiplication and addition), where adding a ReLU activation function promotes the non-linearity within the network.

3.1.2.4 Fully Connected layer

The Fully Connected (FC) classification layer aims to use the features being extracted from the previous convolutional and pooling layers, for classifying the input image into different classes, based on the training data being used. After the features have been extracted through other layers also mentioned, the data needs to be classified where the FCs assist in making the model end-to-end trainable. The CNN can be seen as a feature extractor and is then fed into a classifier. It is also more commonly known to use classifiers, such as a Support Vector Machine (SVM), to classify objects. An FC helps to learn possible non-linear functions that are present in the image space. FCs are also known as Multi-level Perceptrons that use a softmax activation function in the output layer. The FCs will then interpret the image converted to a single dimension column vector and fed into a neural network. At the output of a CNN, the CNN converts to a fully connected layer where the last fully connected layer holds the output.

3.1.3 R-CNN

Over the last few years, region-based Convolutional Neural Networks (R-CNN) have been fine-tuned, tweaked, and evolved but remain a basis for object detection models. An R-CNN is slow and cannot be used for real-time object detection, but provides an excellent fundamental understanding of image processing within the CNN realm. The R-CNN approach uses pre-trained CNNs. These CNN architectures, such as AlexNet, ResNet, GoogLeNet and ImageNet, to name a few, can be used as the pre-trained CNN for the R-CNN. The next step uses the selective search algorithm to find regions of interest (RoI) where, from an input image, around 2 000 region proposals are extracted [61]. As shown in Fig. 3.7, each proposed region is then warped to compute a fixed-sized CNN input regardless of the image region's shape. By using a CNN, features are computed for each of these region proposals. Afterwards, it produces a 4 096-dimension feature vector used in a classification section that classifies each region, using SVMs to classify objects. Offset values are also predicted to increase the precision of the bounding boxes. Localisation errors are minimised by using a regression model where the offset values help by adjusting the bounding boxes of the region proposed. Bounding-box regression should not be done on proposed regions that are classified; it serves no purpose to waste computational power

on localising objects that have no ground truth.

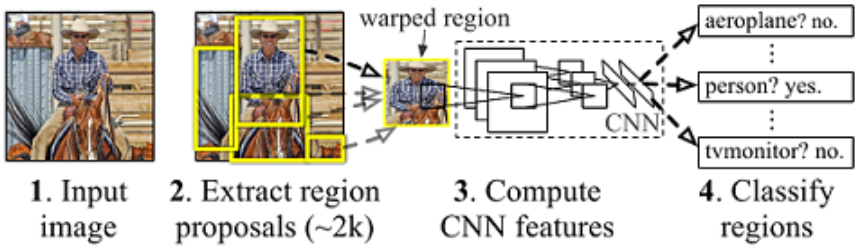


Figure 3.7. The architecture of R-CNN. Taken from [20], © 2014 IEEE.

3.1.3.1 Selective Search

Selective search is a method to be used in object recognition. Selective search combines the power of exhaustive search and segmentation. The aim is to capture all possible object locations, using an exhaustive search to avoid missed objects. An image is hierarchical in, for example, that wheels of the car are sub-categorised as part of a car, but cars are on the same level as animals in a hierarchical object level. Exhaustive search is then done to prevent any potential objects being missed, but this is computationally expensive. Image segmentation is used to improve these drawbacks to try and capture all possible object locations. All object scales need to be considered in the selective search, where some objects have less clear boundaries than others. Regions form an object because of colour, texture or enclosed parts. Lighting conditions also play a role in the region formation.

The selective search uses bottom-up grouping segmentation, generating locations at all scales until a single region has been formed where regions give more information than pixels. Using Felzenszwalb and Huttenlocher's algorithm [62], selective search creates initial regions. An example of an image segmentation algorithm outcome is shown in Fig. 3.8. Then, by using the greedy algorithm, regions are

Figure 3.8. Segmentation results produced by Felzenszwalb and Huttenlocher's algorithm. Taken from [62], © 2004 Springer.

grouped by comparing similarities between neighbouring regions. Regions that are the most similar are grouped, and then new similarities are calculated and compared with resulting regions and neighbours.

The process continues until the whole image becomes one region, forming the Hierarchical Grouping Algorithm[61].

3.1.3.2 Bounding-box regression

The R-CNN approach uses offset values passed on from the CNNs flattened features to find the bounding boxes around the detected object. An object is correctly localised if the bounding box encapsulates the target object sufficiently with ground truth. Ground truth refers to the original image before regions are proposed. The accuracy of such overlapping intersection is called Intersection Over Union (IOU), which is measured between two bounding boxes [63]. Successful localisation is generally set at above 0.5. Bounding-box regression aims to map a proposal box P to a ground truth box G by training a model and learning a predicted bounding box $\hat{G}$. Each box is specified by a top-left coordinate, width and height, with the proposal seen as $P = (P_x, P_y, P_w, P_h)$. Learnable functions transform P into

$$\hat{G}_x = P_w d_x(P) + P_x \tag{3.14}$$

$$\hat{G}_y = P_h d_y(P) + P_y \tag{3.15}$$

$$\hat{G}_w = P_w \exp d_w(P) \tag{3.16}$$

$$\hat{G}_h = P_h \exp d_h(P), \tag{3.17}$$

where $d_x(P)$ and $d_y(P)$ are scale-invariant translations of the coordinates (x, y). $d_w(P)$ and $d_h(P)$ are log-space translations of width and height. In R-CNN the feature vector of $pool_5$ is used for proposal P and the learning functions $d_*(P)$ with weights w_* for training, using ridged regression. Y_* is then a matrix of regression targets t_*. Training pairs are as follows

$$t_x = \frac{(G_x - P_x)}{P_w} \tag{3.18}$$

$$t_y = \frac{(G_y - P_y)}{P_h} \tag{3.19}$$

$$t_w = \log(\frac{G_w}{P_w}) \tag{3.20}$$

$$t_h = \log(\frac{G_h}{P_h}). \tag{3.21}$$

Different regression models can be used in training by using the learning functions but go beyond the scope of this paper. It is important to note that the bounding-box regression model uses a regularisation term. A High regularisation value is used for more dependable results [20].

3.1.4 Fast R-CNN

R-CNNs have notable drawbacks with training, being a multi-stage pipeline, and training is expensive in space and time because both the SVM classifier and bounding-box regressor features need to be

extracted for each object proposal. R-CNNs' object detection is also too slow for real-time applications; detection with a VGG16 CNN takes 47 seconds per image [21]. The slow performance of R-CNN, by not sharing computational power being caused by the forward passing of each object proposal, is improved in Fast R-CNN.

Fast R-CNN takes the whole image to be processed and object proposals are being fed through several convolutional and max-pooling layers to produce a feature map. Each object proposal is then fed through a region of interest (RoI) pooling layer to extract a feature vector. The feature vector is passed through fully connected layers, which split into two output layers. The softmax estimators estimate K classes and a layer that outputs four values that point to the bounding-box positions for the K classes. The Fast R-CNN architecture is depicted in Fig. 3.9, showing the final outputs to the softmax layer, and bounding-box regression stages [21].

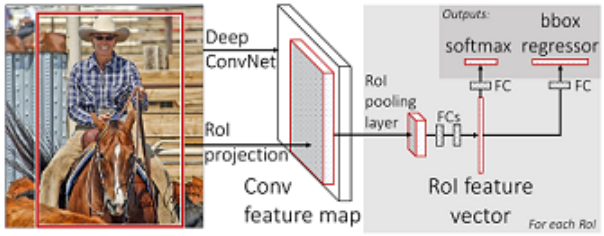


Figure 3.9. The architecture of Fast R-CNN. Taken from [21], © 2015 IEEE.

3.1.4.1 RoI pooling layer

RoI pooling uses max pooling to convert features inside a region of interest into a smaller feature map. The projected region of interest is divided into a $H \times W$ grid of smaller windows of approximate size that equates to $h/H \times w/W$, where h and w are the RoI's parameters. The smaller windows are then max-pooled. Each RoI uses a multi-task loss L to train for classification and bounding-box regression, using Equation 3.22, where p is the discrete probability distribution per RoI computed by softmax and v is the bounding-box values with t^u being set as the predicted bounding box for each class label u, so that $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$.

$$L(p,u,t^u,v) = L_{cls}(p,u) + \lambda[u \geq 1]L_{loc}(t^u,v) \tag{3.22}$$

With a log loss for true class u, the equation is formulated as

$$L_{cls}(p,u) = -logp_u. \tag{3.23}$$

For bounding-box regression the loss equates to form a localisation loss as

$$L_{loc}(t^u,v) = \sum_{i\in\{x,y,w,h\}} \text{smooth}_{L_1}(t^u_i - v_i), \tag{3.24}$$

where the smoothing function's bounds are set, defined as

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| \geq 0 \\ |x| - 0.5 & \text{otherwise} \end{cases}, \tag{3.25}$$

Lastly, the λ is a hyper-parameter that controls the balance between two localisation and classification losses.

3.1.5 Faster R-CNN

Selective search being discussed in earlier sections is used in R-CNN and Fast-CNN but is slow and time-consuming at 2 seconds per image in a CPU implementation. An intuitive way was introduced to find region proposals without using selective search but instead allowed the CNN to learn the region proposals through sharing full-image convolutional features with the detection network [22]. This method also uses an RoI pooling layer, similar to Fast-CNN. However, the predicted region proposals formulated from the CNN have been shaped and used to find the bounding-box values for localisation and then to classify the image. This network being used in the Faster R-CNN paper, that replaces selective search, is called the Region Proposal Network (RPN). RPN shares convolutions at test time with computing proposals at 10 ms per image, making it favourable for real-time detection.

3.1.5.1 Region Proposal Networks

The RPN uses pyramids of filters and "anchor" boxes that cater as references at different scales and ratios. RPNs use a training scheme between fine-tuning the region proposal task and object detection task whereby the network's convolutional features are shared. The RPN takes an image of any size as input and returns rectangular object proposals with an object score whereby computation can be shared with the detection network. The CNN's convolutional feature map is used by sliding a small network over it. This small network takes an $n \times n$ window from the convolutional feature as an input. This window is then mapped to a lower-dimensional feature where this lower-dimensional feature is fed into two fully connected layers, namely a box-regression layer and a box-classification layer. Fig. 3.10 shows the small network, constructed by the RPN. RPN uses "anchors" that refer to k proposals that are parameterised relative to k reference boxes in each sliding window shown in Fig. 3.10. The anchor is centred at the centre of the sliding window. The maximum possible proposals for each location is represented as k. The regression layer has 4 k, pointing to the coordinates of the k boxes and a classification layer of 2 k scores, estimating the presence of an object.

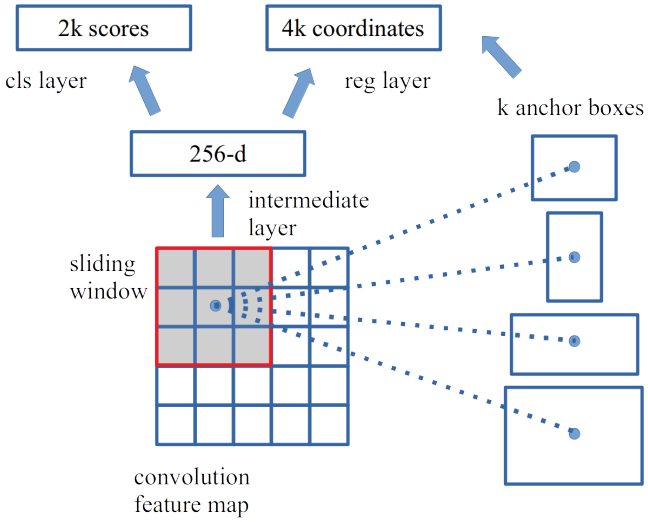


Figure 3.10. Region Proposal Network (RPN). Adapted from [22], © 2016 IEEE.

3.1.6 YOLO

Fast R-CNN and Faster R-CNN provide better speeds and accuracies than R-CNN but still lag to be used within real-time applications. YOLO does not use a Selective Search or shared computation mechanism, such as mentioned previously for Fast and Faster R-CNN networks. Instead, YOLO uses a unified single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation at real-time speeds of 45 frames per second [23]. YOLO does not use a sliding window and region proposal techniques but instead uses the entire image during training and testing. YOLO divides an input image up into an $S \times S$ grid. Each grid cell is responsible for detecting objects that fall within it and predicts the bounding boxes and confidence scores for that box. All bounding boxes for all classes of an image are simultaneously predicted. The confidence scores point to the confidence of the presence of an object. This is defined by

$$confidence = Pr(Object) * IOU^{groundtruth}_{predictedbox}, \tag{3.26}$$

The confidence prediction is the IOU between the predicted box and the ground truth. Each bounding box contains predictions: centre coordinates, width, height and the object's confidence. Grid cells predict class probabilities given the presence of an object, where conditional scores are calculated for probabilities of a class appearing in the box and how well it fits using

$$Pr(Class_i|Object) * Pr(Object) * IOU^{truth}_{pred} = Pr(Class_i) * IOU^{truth}_{pred}. \tag{3.27}$$

As mentioned in [23] and shown in Fig. 3.11, evaluating the CNN on the PASCAL VOC detection dataset and using grid cells of 7×7, set at two boundary-boxes and 20 different classes, C would equate to predictions equal to $S \times S \times (B * 5 + C)$ and a predictions tensor of $7 \times 7 \times 30$. Using this tensor being formed by the YOLO model, the CNN can be used for the predictions. YOLO performs linear regression, using two fully connected layers for all the grid cells and keeps those with high confidence scores as the final predictions.

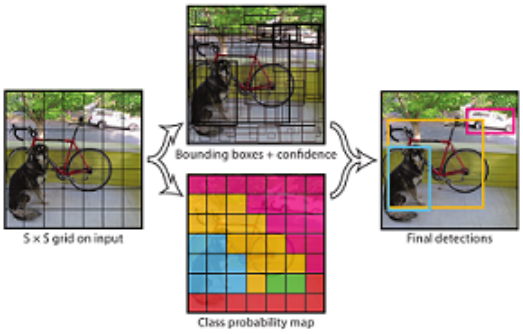


Figure 3.11. YOLO's model with $S \times S$ grid predicting B bounding boxes and confidence with C class probabilites. Taken from [23], © 2016 IEEE.

3.1.7 MobileNets

More complicated neural networks result in better accuracy but also causes models to increase in size. Real-world applications such as mobile and embedded vision applications have limited resources that can use MobileNets for more efficient models. MobileNets have trade-offs, but a suitable-sized model can be chosen while considering speed for a specific application that will perform better than larger CNNs. MobileNet architectures include a set of two hyper-parameters, namely width multiplier (α) that controls the number of channels and resolution multiplier (ρ), which controls the input image resolution [24]. The architecture is built up from depthwise separable convolutions. As discussed in previous sections, standard CNNs house a large number of parameters, where MobileNets reduce the number of parameters, resulting in a lightweight deep neural network, shown in Fig. 3.12.

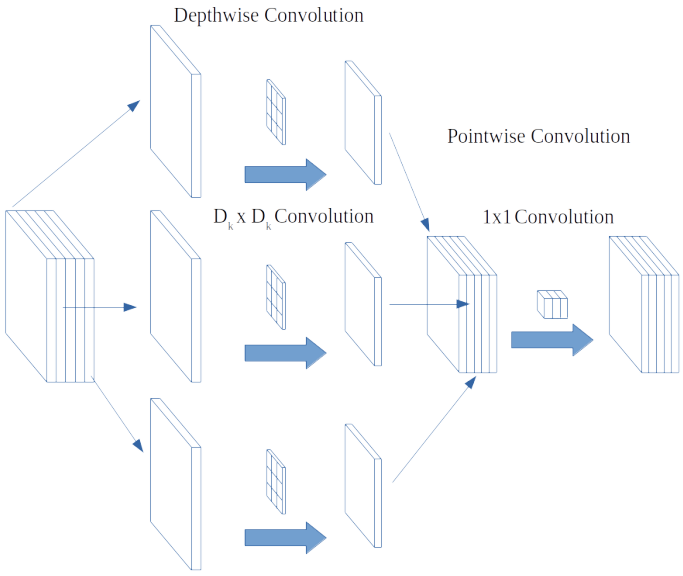


Figure 3.12. MobileNet architecture, expressing the Depthwise convolution and Pointwise convolution graphically. Adapted from [24], © 2016 IEEE.

MobileNets split convolutions into 3x3 Depthwise convolutions, followed by 1x1 Pointwise convolution, where convolution is followed by batch normalisation and ReLU when normal CNNs is used, as shown in Fig. 3.13.

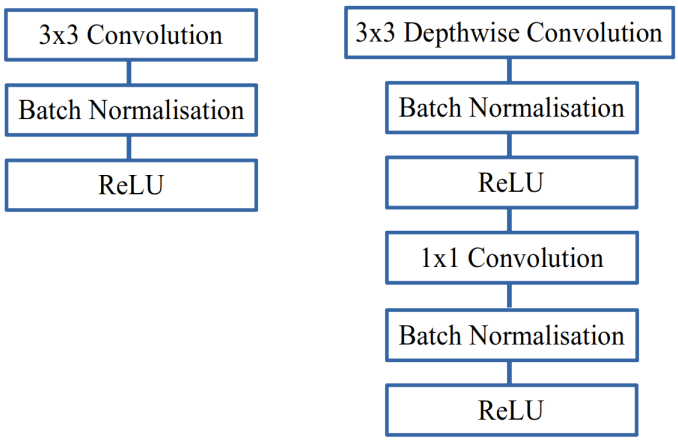


Figure 3.13. The left side expresses standard convolutions and the right side expresses Depthwise Separable convolutions, followed by Pointwise convolutions. Adapted from [24], © 2016 IEEE.

The reduction in parameters reduces the Multiply-Accumulates (MACs), which measure the number of multiplication and addition operations. Depthwise separable convolution consists of two opera-

tions, namely: Depthwise convolution and Pointwise convolution. The Depthwise convolution is the channel-wise spatial convolution. If there were N channels, then there would be N of $D_k \times D_k$ spacial convolutions, resulting in the cost being expressed as

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F, \tag{3.28}$$

where D_F represents the feature map, D_k the kernel size and N as the number of output channels, compared to standard convolution, which is

$$D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F, \tag{3.29}$$

resulting in a computation reduction of

$$\frac{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_k^2}. \tag{3.30}$$

3.2 ADAS DETECTION TECHNIQUES

An ADAS system consists of sensory input through different sensors. A general ADAS system is shown in Fig. 3.14, where FU 1 and FU 2 represent the shared vision and proximity sensors, respectively. FU 1.2 and FU 2.2 supply data transmission to the ECU in FU 3, using a wired connected network. The ADAS processing (FU3.3) and image processing (FU 3.1) are carried out on the ECU, using prepared data from sensors in the wired network. The result is then transferred to the display unit of the vehicle (FU 4.2) to inform the driver of potential warnings, which can also be emphasised through the vehicle's audio output (FU 4.4).

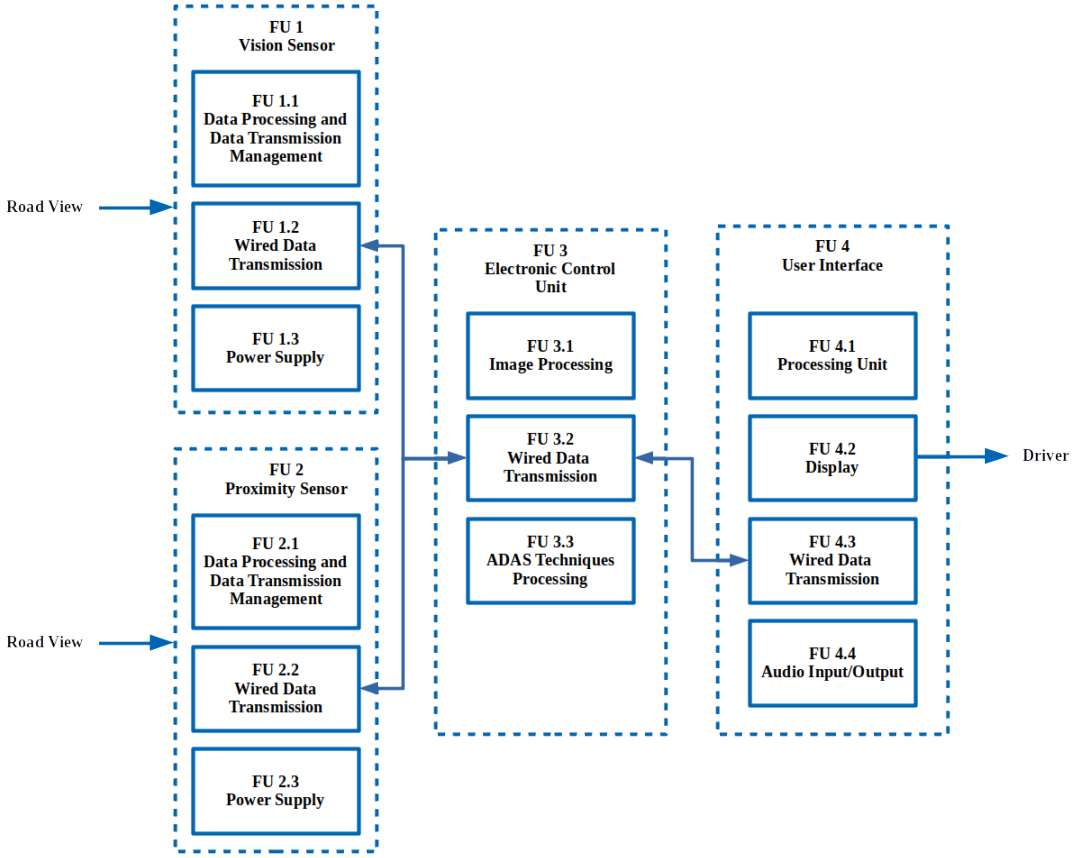


Figure 3.14. Functional diagram of a general ADAS system.

Many different ADAS processing techniques are used in FU 3.3, such as collision avoidance, lane assistance, traffic light detection and blind spot detection. Some of these ADAS techniques are discussed in the following sections.

3.2.1 Collision detection

The most basic form of collision detection is forward detection, which ensures that a safe distance is kept between the driver's vehicle and other vehicles on the road. Collision systems monitor a vehicle's speed, compared to the speed and distance of the other vehicles on the road, in order to prevent a collision. By preventing vehicles from entering unsafe zones, potential crashes can be avoided. Forward Collision Warning (FCW) does not intervene on behalf of the driver but instead warns the driver by utilising sound or visual warnings.

Most common FCW systems being implemented on high-end vehicles use vision and radar sensors where the collision warning is displayed on the driver's information panel, accompanied by sound. Radar sensors can become costly, but distance estimation can be used as an alternative by using low-cost vision sensors and a smartphone. A pinhole method can estimate distances by using a smartphone that is capable of running a CNN MobileNet to detect objects. Suppose that the smartphone is able to detect vehicles at a reasonable frame rate while ensuring a stable video stream, the smartphone can then be used as the information panel to warn the driver visually and audibly.

A streaming system consists of ingestion, event and processing times. The ingestion time is the time that it would take the system to take the event through the CMOS image capturing. The event time is when the event occurs, while the processing time is the time that it would take the system to process the incoming data, which will result in frame rates being dropped. These types of time delay mentioned, need to be considered when working on real-time streaming systems. The data flow of a conceptual visual sensor ADAS component is shown in Fig. 3.15.

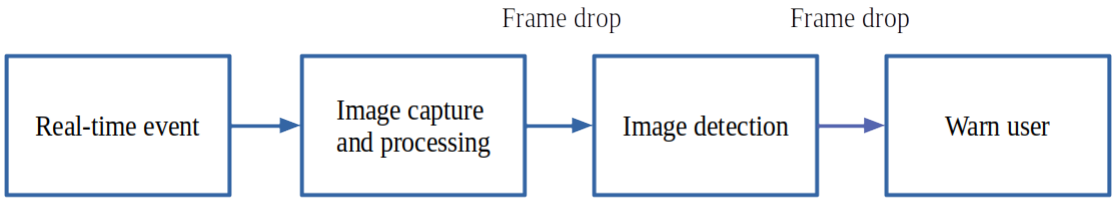


Figure 3.15. Frame drops occur from the time that the event occurred, until the user is being warned.

These frame drops are inevitable but should not drop below a frame rate worthless to the driver. The systems response time will need to be increased as the speed of the vehicle is increasing.

3.2.1.1 Monocular distance estimation

By using image processing techniques, the 3-D world projected to a 2-D space being captured by the camera feed, can be used to calculate the distance of objects. It was shown that accurate distance estimations of objects could be made by using monocular vision and by using a pinhole method [64]. This method uses triangle similarity to calculate the distance to an object after a focal length has been

calculated [16]. Fig. 3.16 shows the pinhole model representation of the manner in which the sensor captures the object through the lens of the camera.

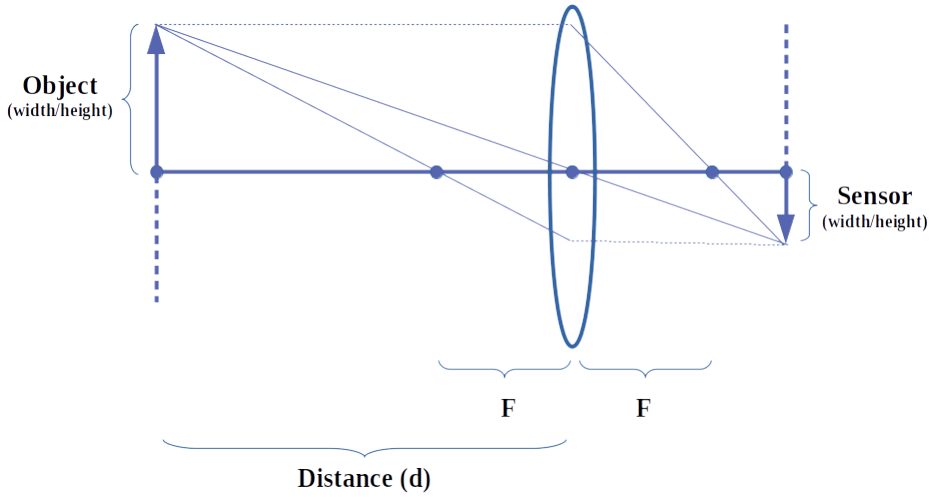


Figure 3.16. Representation of a pinhole model, showing the optical operation of rays through a lens.

The focal length can be calculated by using the pinhole model that uses a predetermined object's height or width and distance from the lens. The focal length can then be calculated by using

$$f = \frac{width_{pixels} \times distance_{object}}{width_{object}}, \tag{3.31}$$

where the detected width in pixels is used with the actual distance, as well as the width of the object. By simplifying the model to describe the relationship between the real-world points in Cartesian coordinates $(x;y;z)$ and the sensor image $(u;v)$, the following relation can be equated as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{3.32}$$

where $(u_0;v_0)$ represent the pixels in the centre of the image, f_u and f_v are the focal lengths of every coordinate in the image, and λ represents the scaling factor. The limitation of the pinhole model is that it assumes that the vehicle is located within the ground plane, and the camera is fixed at a set height [15].

3.2.2 Lane detection

Speed and accuracy are crucial for real-time detection of lanes on a road. False-positive detection could be detrimental, and slow detection would defeat the purpose of a real-time strategy to support vehicles. Using Hough transforms in lane detection, is an approach used in many research investigations,

such as [65],[66] and [67]. Hough transform is a widely used method in lane detection, but because of the complexity of Hough transformation, the computation speeds are prolonged, and the false detection rate is significant. Hough transform is not a process that is done directly on an image but rather a process that is induced onto an image as an end-stage after an image has gone through pre-processing. Pre-processing consist of stages, such as grayscaling, Gaussian blurring and Canny transformation.

Neural networks have made their way into lane detection, using a spiking neural network to extract better edge lines, with most recent studies using Spatial CNNs in lane detection [68]. Due to Hough transforms architecture, it has flaws in lane detection, such as braking in curved lanes or sharp turns, where CNNs introduce new possibilities for improvement.

3.2.2.1 Canny detector

Canny is an edge detector that consumes a grayscale image and outputs an image that shows tracked intensity discontinuities. A 2-D Gaussian convolution first smoothes the image with the form

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{3.33}$$

Using a convolution kernel that approximates a Gaussian of σ of 1.0, an example of a 2-D matrix is approximated as

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}. \tag{3.34}$$

Next, a 2-D first derivative operator is applied to highlight edges in the image with high first spatial derivatives. Derivate operators such as Sobel and Roberts are then used along the x- and y-axis to find edges in the image, using kernels such as being used in Roberts Cross edge detecting kernels

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.35}$$

$$G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{3.36}$$

These kernels are applied to the image and combined to find the absolute magnitudes

$$|G| = \sqrt{Gx^2 + Gy^2}. \tag{3.37}$$

In Fig. 3.17 the output of a Sobel derivative operator of a gradient magnitude image is shown.

Figure 3.17. Edge detection based on Sobel operator. Taken from [65], © 2015 IEEE.

The Canny operator then cleverly uses a tracking process hysteresis, controlled by two thresholds, $T1 > T2$, where tracking can only begin at a pixel gradient higher than $T1$. Tracking then continues in both directions until the value is lower than $T2$. This process ensures that noise does not interfere in breaking up lines into multiple edge pieces. Hough transformation is needed for further processing because the result from Canny alone does not indicate roads lanes from a vehicle's point of view, but only detects available edges in images. Hough transformation is required to detect lanes.

3.2.2.2 Hough transformations

Hough transformation is the transformation from the Cartesian space to Hough space. A straight line $y = mx + c$ in a Cartesian coordinate system is a point in Hough space. Polar coordinates are used instead of Cartesian coordinates to prevent vertical line infinity gradients in the Cartesian planes. Hough transform allows for converting the coordinate image space to the parameter space to find a fitting line curve. A line can be expressed as a standard Polar radius calculation

$$r = xcos\theta + ysin\theta. \tag{3.38}$$

If an image with a familiar lane is used as an example, then the scattered points in a Cartesian plane are indicated by points which some line will be able to connect. This line can be found by mapping the points from the Cartesian system to Hough space and find the intersection, as shown in Fig. 3.18.

The intersection point in the Hough references a straight line detection, in the case of a road, which is most likely a road with a lane.

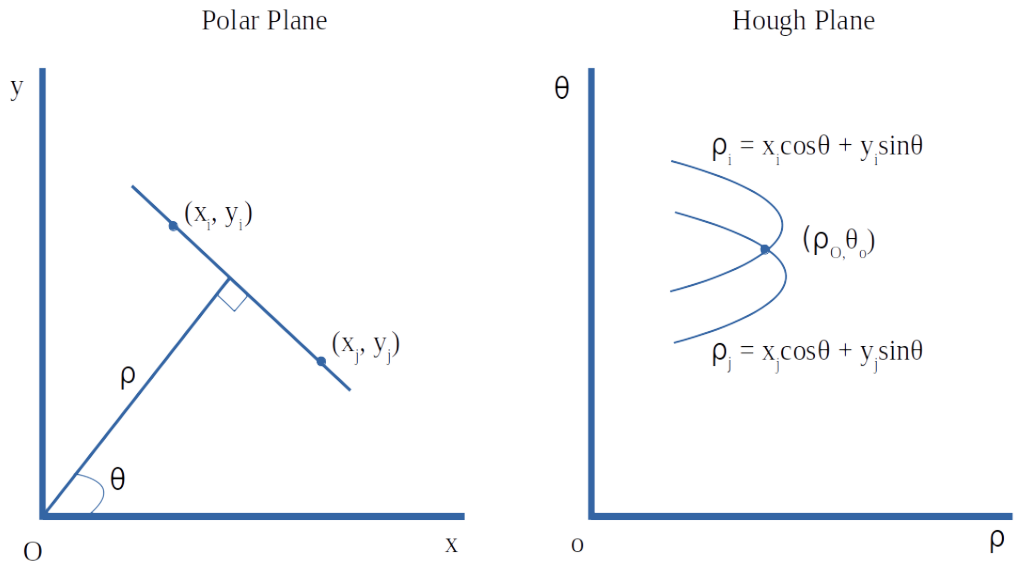


Figure 3.18. Polar coordinates being mapped to Hough space. Adapted from [65], © 2015 IEEE.

3.3 INTRA-VEHICULAR WIRELESS SENSOR NETWORKS

An ADAS system communicates with sensors and receives information through a network. The leading group of sensors for ADAS systems are radar, LiDAR and vision. Radar sensors can either be long-range or short-range, with ranges within centimetres or a few hundred metres being placed mainly on the front or rear of the vehicle [69]. The increase in the number of sensors being placed on vehicles has increased attention to wireless networks within vehicles known as IVWSNs.

This system removes the need for wires, by removing weight on the vehicle and by not adding heavy electrical harnessing. Most sensors operate within vehicles by using Controller Area Networks (CAN) and Local Interconnected Networks (LIN) connected to an Electronic Controller Unit (ECU). IVWSNs are appealing because wires limit the locations where sensors can be stored and get worn down over time, which will cause the replacement of expensive harnessing. IVWSNs replace the communication between sensors and ECUs. IVWSNs originally stem from classical WSNs, but have characteristics unique to the challenging vehicle's internal environment for radio propagation, caused by metal objects and passengers inside. Another essential requirement of IVWSNs is their strict reliability and low communication latency, which is vital for vehicle applications. Reliability is a fundamental requirement of a network being run in a vehicle, and a proper network design is required. One such architecture is the one shown in Fig. 3.19.

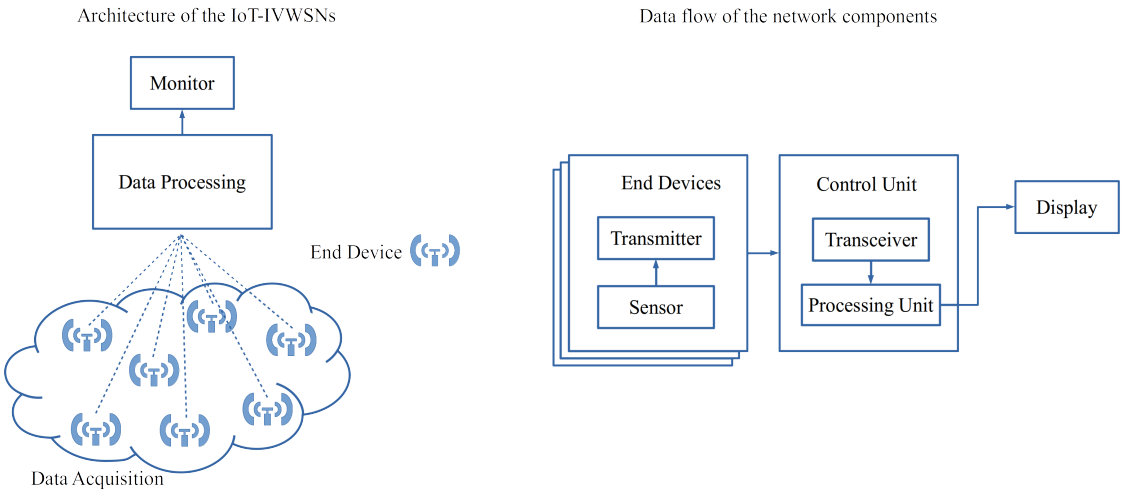


Figure 3.19. Network architecture of an IVWSN network. Adapted from [49], © CC BY.

In a smartphone environment, an ADAS system is limited to Bluetooth and Wi-Fi wireless communication because these are the general means of communication between peripherals for mobile devices. Bluetooth is a cheaper option and would make compatibility between sensors and mobile devices a good option, but has slower transfer rates for data-intensive sensors.

Various sensors inside the vehicle collect the data being transmitted and received by the control unit and processing unit. The architecture mentioned uses a single-hop star topology, which allows each end device to communicate with the same signal strength, allowing the network to skip broken nodes and choose the shortest paths to areas in the network. These two wireless communication options are discussed as reasonable solutions by considering cost, complexity and performance.

By using a wireless ADAS system as shown in Fig. 3.20, the ECU and display unit in a high-end ADAS system are merged into FU 2, consisting of a smartphone. The sensors, FU 1 and FU 3, communicate with the smartphone through a wireless network where the ADAS techniques and processing are carried out (FU 2.4). By using the standard connection capabilities, the network can consist of Wi-Fi and Bluetooth protocols. Wi-Fi is a better-suited protocol for transferring information from camera sensors where an IP network is required to establish communication. The smartphone in FU 2.2 becomes an Access Point (AP), and the camera sensor's hosted video server allows the smartphone to access video through the Wi-Fi network (FU 1.2).

The network also consists of blind spot sensors that are not as data-intensive as the camera sensors.

The blind spot sensors only require the distance to be sent to the smartphone and require minimum data throughput. FU 3.2 is used by the blind spot sensor to transfer data to the smartphone. Finally, the display and audio of the smartphone are used to interact with the driver, warning about potential risks.

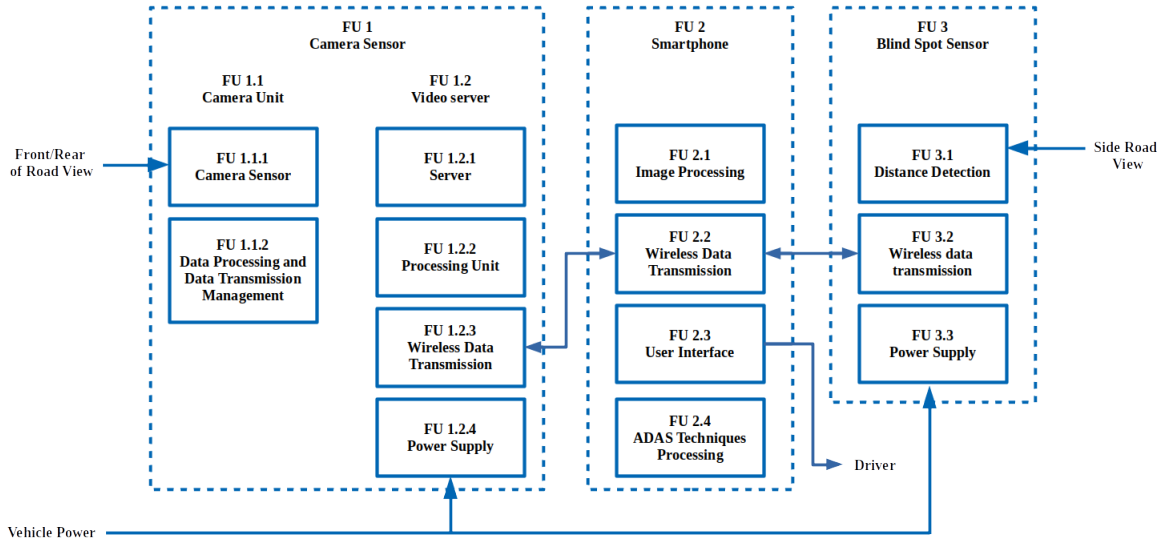


Figure 3.20. Functional diagram of a proposed wireless ADAS system, using a smartphone.

3.3.1 Bluetooth

Bluetooth is cost-efficient and power-efficient but is not well-suited for real-time video due to the limited bandwidth, high degree of error rates and time-varying nature of radio links. Bluetooth has three kinds of connection schemes: Point-to-point, piconet and scatternet. Piconet is a point-to-multipoint topology with one device being a master and the rest of the devices becoming slaves. In a scatternet, a slave in a piconet can become a master in another piconet system. Video streaming, using Bluetooth technology, has shown to have empirical working outcomes where streaming of video was streamed back and forth from a mobile phone to a desktop, using a Bluetooth network as last-hop network [70]. Different packetisation was investigated with different protocol layers, such as L2CAP and RFCOMM on Bluetooth V1.1 and v2.0 being compared.

Bluetooth v2.0 uses two primary modulation schemes, which are the Basic Rate, Gaussian Frequency Shift (GFSK) for backwards compatibility with older Bluetooth versions, as well as Enhanced Data Rate, which uses two modulation types of PSK; $\frac{\pi}{4}$ - DQPSK and 8DPSK. Basic Rate gives a data rate of 1 Mbps, and 2 to 3 Mbps for Enhanced Rates. Two link types are used in Bluetooth: Synchronous

Connection Oriented (SCO) and Asynchronous Connectionless Link (ACL). ACL was used for greater flexibility and increased data throughput [70], SCO is a point-to-point circuit-switched connection with 64 Kbps and ACL is a point-to-multipoint with up to 732 Kbps downlink and 128 Kbps uplink.

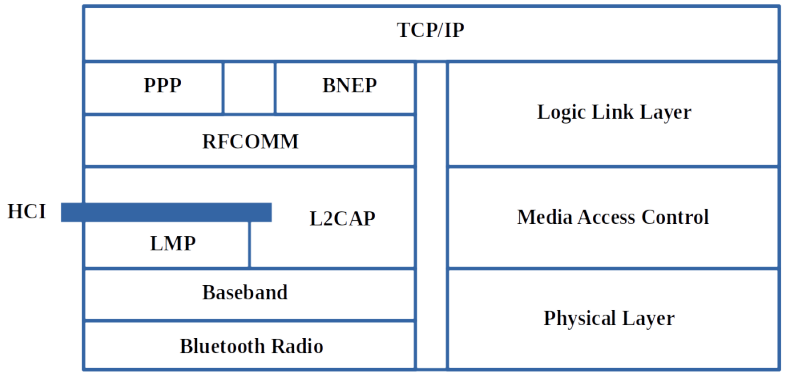


Figure 3.21. The protocol stack of Bluetooth. Adapted from [71], © CC BY.

There are three different layers in the Bluetooth stack: physical layer, link layer and application layer. In Fig. 3.21 the different protocols and layers are presented. The TCP/IP network can be accessed through Point-to-Point Protocol (PPP) or Bluetooth Network Encapsulation Protocol (BNEP). Different Bluetooth video streaming options were considered in [70], namely HCI, L2CAP and IP. HCI is device-dependent with devices having different buffer sizes, and if an L2CAP header is not present, HCI packets are dropped by some devices. L2CAP is a lower-layer protocol, and more effort is involved in HCI.

Some form of video streaming, using L2CAP, and RFCOMM protocols [71], was shown to be effective. An architectural design is represented for video streaming Bluetooth, as show in Fig. 3.22. The raw data is compressed, since bandwidth is limited to 732 Kbps. The device that serves the client device uses a QoS control module that adapts the media stream and adjusts transmission parameters. The stream is then processed and packetised, using the already mentioned L2CAP, HCI or IP from the intermediate layer. The packets are then transmitted to the Bluetooth module for transmission and received by the client's Bluetooth module. The intermediate protocols use them for reassembling and the packets are decoded further for decompression. Other implementations exist, namely MPEG4BT via HCI, IP and L2CAP, respectively, where comparisons were made by considering the size of overheads, segmentation efficiency, reassembly and hardware compatibility [71].

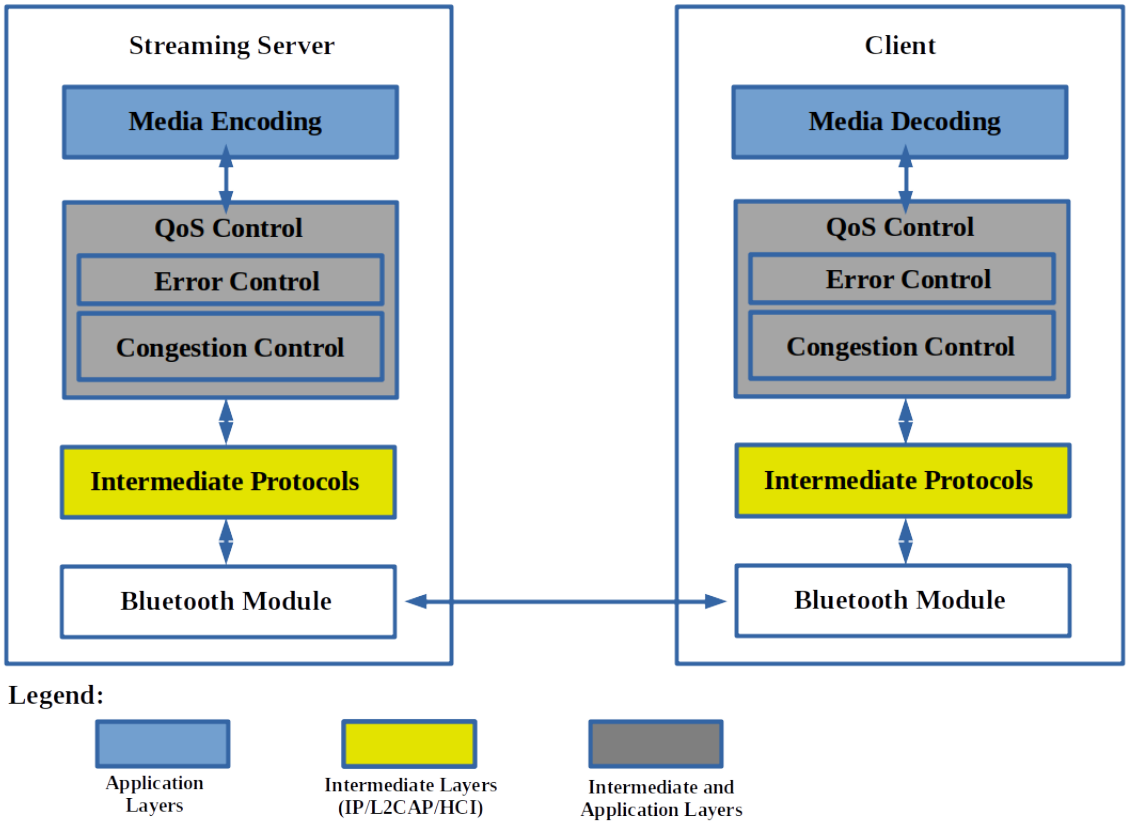


Figure 3.22. Architecture for streaming over Bluetooth. Adapted from [71], © CC BY.

Video streaming via HCI requires an application-layer software implementation because of its lack of segmentation and reassembly (SAR), which loads the hosts' resources. Streaming, using L2CAP, hides Bluetooth's lower layers, where L2CAP allows SAR of higher-layer packets with larger sizes between the protocol and the smaller baseband packets. The disadvantage of using L2CAP over HCI is the more significant overheads needed for packet encapsulation. Different Bluetooth profiles are used by different devices, depending on the desired service. The profiles provide standards that can be used for an intended purpose and use only certain Bluetooth stack parts. In terms of looking for a profile to cater for video streaming, profiles such as Generic Audio/Video Distribution Profile (GAVDP) can be used via L2CAP protocol. The Audio/Video Distribution Transport Protocol (AVDTP) consists of a signalling entity for negotiation and, based on the result, the Audio/Video applications transport video content. Streaming video over IP can be done by using the Bluetooth profile LAP or the host stack's Bluetooth Network Encapsulation Protocol (BNEP), which delivers network packets on top of L2CAP; the profile Personal Area Networking (PAN) profile is used. Bluetooth's LAP profile uses the Point-to-Point Protocol (PPP) over the host stack RFCOMM, which allows packets to flow over

the link, where the LAP profile uses a Bluetooth access point. BNEP passes the packets through the TCP/IP stack layer where a header is added to the packet, and then being passed through an Ethernet Frame layer and finally, to L2CAP. On the receiving side, the application converts the packet to a video stream. Streaming via IP relies on the bridging of TCP/IP and Bluetooth, which is a duplication of IP-based video streaming but comes at the cost of having overhead added by upper layers, thereby degrading performance for limited bandwidth Bluetooth.

3.3.2 Wi-Fi

Most Wi-Fi solutions require a network or router for communication, but Wi-Fi Direct allows terminals to establish a Wi-Fi network without a real Access Point (AP). Devices that are Wi-Fi Direct enabled, also known as Wi-Fi P2P, allow for direct connections between devices wirelessly in the same way that many users may accomplish by using cables. Devices can make a one-to-one connection, or a group of devices can connect simultaneously.

Wi-Fi Direct is used with P2PSIP protocol, which enables real-time communication without the need for any server [72]. Wi-Fi Direct allows the connection of devices without a hotspot where the main goal is to create a P2P network by using traditional Wi-Fi signals. By forming Groups, Wi-Fi Direct can allocate a device (Group Owner) in charge of the Group that decides which devices are allowed to join. The Group Owner acts as a "virtual" AP. A "virtual" AP is a good route to take when being forced to use Wi-Fi and when no real AP is available or is not architecturally allowed. Different communication protocols exist, where in the case of Wi-Fi Direct, P2P technology plays a role when a direct connection is required. P2PSIP is an application-layer signalling protocol that improves conventional SIP, which forms part of a network, consisting of nodes with different peers-issued roles. In P2PSIP, there are normal peers and joining peers where they connect directly to one another by joining and notifying one another of their presence while sending data back and forth between the network. In Fig. 3.23 the P2PSIP network is expressed graphically. As seen in the figure, two kinds of signalling are presented in a green line for presence-related information, whereas the dotted blue line is used for media-related information. The data represented by the red line is exchanged directly between two peers.

A P2PSIP network could be used for an ADAS implementation, but the network of an ADAS system would not be as dynamic as the architecture in Fig. 3.23, which caters for the joining and leaving of peers. P2PSIP could be used as an initial setup on the vehicle but remains static afterwards. P2PSIP is

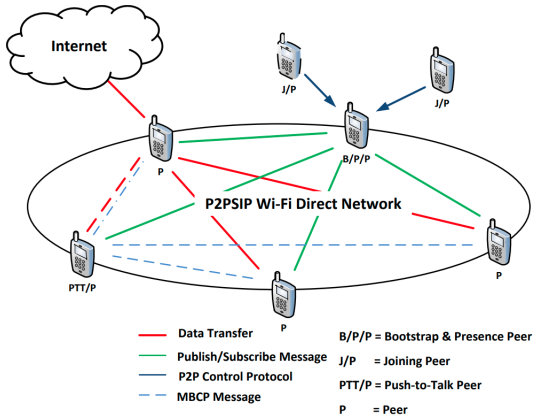


Figure 3.23. A sharing-service P2P overlay in Wi-Fi Direct. Taken from [72], © 2012 IEEE.

an improvement on SIP, which was not spontaneous and as easy to set up in emergency scenarios. The architecture can be used in environments that require high data transmitting speeds by using Wi-Fi Direct on an Android platform, which would be advantageous to a wireless ADAS system [72].

In an ADAS system, the smartphone can be the Group Owner, the discoverable device where the sensor/camera nodes would attempt to join the Group. Standard Wi-Fi authentication is included within Wi-Fi Direct's environment, which includes obtaining credentials and authenticating searched devices, and by adding a device by invitation, the P2P invitational procedure enables a device to become part of an existing P2P group [73]. Accordingly, Wi-Fi becomes more like a Bluetooth connection, but can communicate with more devices simultaneously at faster Wi-Fi transfer rates and at an increased range. Only one of the devices needs to be Wi-Fi Direct to establish P2P connections, where most smartphones are Wi-Fi Direct enabled. Wi-Fi communication would require nodes to support Wi-Fi to interact and transfer information to the Wi-Fi Direct Group Owner. From Table 3.1, the comparison between Wi-Fi and Bluetooth is shown. Wi-Fi's advantage over Bluetooth is the data transfer speed, considering video streaming feed for several camera sensors.

Table 3.1. Performance comparison of wireless technologies. Taken from [74], © 2015 IEEE.

Factor	Bluetooth	Wi-Fi	Wi-Fi Direct
Frequency	2.4 GHz	2.4, 3.6 and 5 GHz	2.4 and 5 GHz
Transmission rate	25 Mbps	250 Mbps	250 Mbps
Distance range	30 m	100 m	100 m
Operating system	Android / iOS	Android / iOS	Android only
Topology used	P2P	Star	P2P
Maximum devices	7	Router-dependent	+3
Security	Matching password	WEP, WPA, WPA2	WPA2
Need for AP	No	Yes	No

Wi-Fi Alliance's Wi-Fi Display is used to stream real-time video to devices [74]. These devices are sorted between a sink and source, the sender and the receiver, respectively. The source device captures the video, encodes and packetises the data, and then transfers the data while the sink device is receiving the data, depacketises and decodes it, while rendering the video. Using Wi-Fi Display still uses Wi-Fi Direct by using a P2P topology with the same Group Owner setup as mentioned earlier but uses the RTSP protocol to transfer video. The device involved in the Wi-Fi Display session contains basic information, called WFD IE (Information Element), that optimises the connection between the devices. Then the source device selects the sink device to interact with local policies that are being configured. In order for the connection between the two devices to be successful, an exchange of WFD IE occurs where upon success, a TCP connection setup is established, causing the source device to become a TCP server and the sink device to become the TCP client. During this WFD session, the RTSP protocol starts the interaction between the two parties.

Negotiations by using RTSP protocol is done by a handshake process, which consists of responses named M1 to M7, where M8 represents termination. Following a successful negotiation of up to M7 and status of RTSP OK, a session is established, which will start playing the video stream.

3.3.3 Camera sensors

Vision-based sensors have become very popular in ADAS systems because of their low costs. Even though camera-based systems are considered inaccurate in poor conditions with poor visibility, many efficient techniques are now available to improve accuracies for poor weather conditions. Two primary

camera sensors exist, namely Charge-Coupled Device (CCD) and Complementary Metal-Oxide Semiconductor (CMOS). An in-depth comparison between CCD and CMOS concluded that even though CCD allows for better quality, light sensitivity and resolution, CMOSs are faster, smaller, cheaper and more power-efficient. CMOS image rates are higher, dominating the automotive industry market of ADAS [1].

Different techniques are used in camera sensors to find distance measurements between the camera sensors and objects. Techniques include optical flow, which is commonly used to measure distances and the direction of an object in a mono camera. Radar capabilities are being promoted within CMOS camera chips, which also automates the process of finding the distances of objects, which has been proven to work well in all weather conditions [1]. These aforementioned radar-enhanced CMOS cameras also improve on calculated speed and direction of objects where techniques, such as motion blur and smearing effect, can also be used with standard CMOS cameras to calculate distance and speed.

Two-dimensional cameras are widely available where superimposing of additional information has found a commonplace in commercial ADAS systems. These cameras need a high dynamic range, which is important in delivering high-quality imaging with direct sunlight that may shine into the lens. Current commercial ADAS architectures are built around the ECU where the processing of the video streams from cameras around the vehicle occurs, also known as a centralised image processing approach. The processing of the images requires high performing image processors, and additional FPGA's are required for hardware acceleration. This architecture mentioned, is shown in Fig. 3.24 from [69], which shows the current restrictions and bottlenecks that exist in ADAS camera systems.

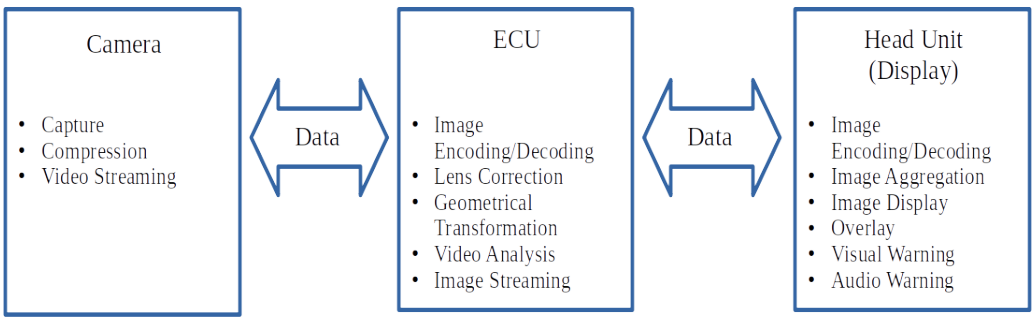


Figure 3.24. Centralised image processing.

A new approach is a decentralised method that focuses on the idea of "smart" cameras. This allows the ECU to be entirely excluded where only the head unit remains. Each camera then performs its own image processing while doing most of the heavy lifting, allowing the data to be distributed to the head unit where the images are aggregated and presented. Using 360-degree view cameras in ADAS systems, allows drivers to see a top-down view of the surrounding vehicles. These kinds of systems consist of four fish-eye lens cameras, all of them facing different directions. Off-the-shelf chipsets exist that process and manage the 360-degree views, such as TI's Automotive ADAS SoC processors [75]. The 360, view solution consists of two algorithms: geometric alignment and composite view synthesis. The geometric alignment converts the fish-eye camera's distortion and converts them to a standard top bird's eye perspective. The synthesis algorithm ensures a composite view following the geometric correction. In [75] another algorithm is used to "stitch" together a surround-view, called "photometric alignment", which adjusts colour and brightness mismatch. The overlapping areas from the different cameras are then blended, or a binary decision is made. Then, lastly, a photometric alignment algorithm deals with the colour and brightness mismatch.

Front camera ADAS solutions, such as [76], explores Texas Instrument's family of SoC for ADAS that integrates GPP, DSP, SIMD and HWA. Texas Instruments Driver Assist 3x focuses on showing the capabilities of a typical camera system [76]. The benefits of using an SoC, such as the TI solution are that various algorithms can be developed across multiple cores by using the dedicated Image Signal Processor (ISP), Embedded Vision Engine (EVE) for vector processing and a Digital Signal Processor. Fig. 3.25 shows the block diagram of the aforementioned architecture of SoC.

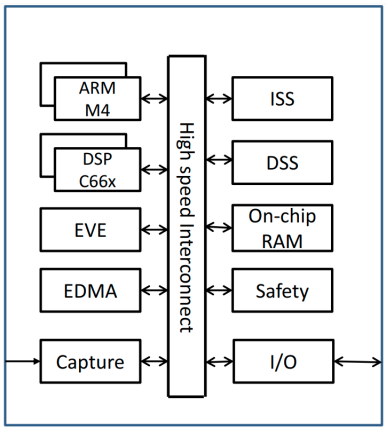


Figure 3.25. Block diagram of TDA3X SoC. Taken from [76], © 2015 IEEE.

The Software Development Kit (SDK) can be used for the implementation of the image processing algorithms by using the framework's concept of "Links and Chain". Different cores can be accessed

for different needs, by using the SDK, with an example shown in Fig. 3.26. This type of architecture results in processing to be carried out on the SoC, which prevents running image recognition and processing algorithms at the head unit. By using a distributed method that includes external "smart" cameras that use an SoC solution, a system can execute multiple ADAS algorithms, namely pedestrian, vehicle detection and traffic sign detection, which are scalable, using the software framework provided. In the case of a low-cost smartphone approach, the processing would use the smartphone's processors. More cameras require more streams to be processed, thereby putting strain on a smartphone's central processing.

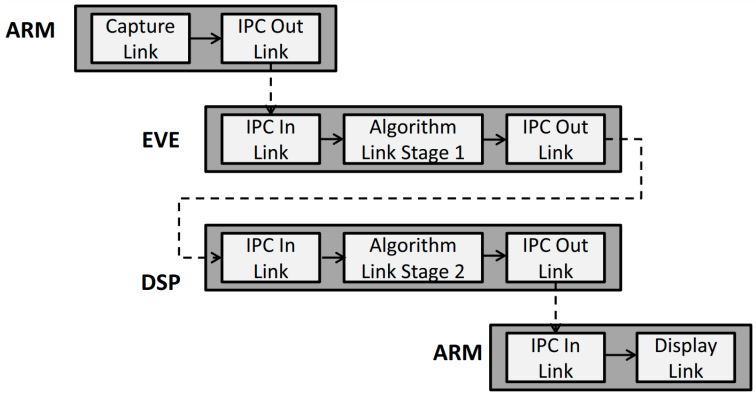


Figure 3.26. Vision SDK example across cores. Taken from [76], © 2015 IEEE.

3.3.4 Blind spot sensors

Many approaches can be taken when detecting blind spots in vehicles; for this paper, however, wireless implementations are favoured, where the data is collected and sent to the smartphone in real-time to alert the driver about possible danger. Detection of blind spots can be divided into two groups: passive and active systems. A passive system does not warn the driver, but gives the driver the ability to see a blind spot by using mirrors or camera feed on a screen that is visible to the driver. Active systems automatically warn the driver about any danger. Active systems currently in the market consist of radar, LiDAR, proximity sensors and camera sensors that use image processing. Fig. 3.27 shows a generalised scenario of a blind spot area. The driver is only aware of Area 1 and Area 2 when driving. Without information about the blind spot area, a collision can occur if the driver has not been warned beforehand.

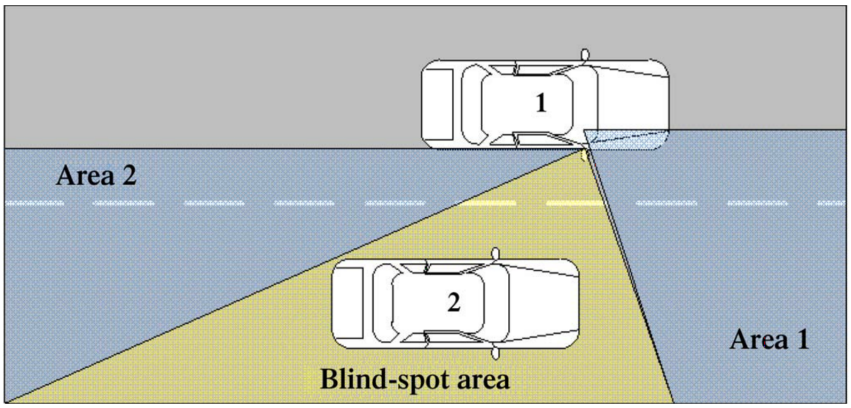


Figure 3.27. Blind spot area. Area 1 and 2 represent the visual field of the driver. Vehicle 2 is in the blind spot zone. Taken from [38], 2012 IEEE.

3.3.4.1 BLE

A blind zone alert system designed for an IVWSN environment can be built by using a platform that is based on BLE technology [35]. The IVWSN-based SBZA system (Side Blind Zone Alert system) is a detection method, using an IVWSN-based SBZA system that shows 95% to 99% detection rate with less than 15% false alarm rate. The benefit of using such an approach is the costs, compared to an expensive camera and radar equipment. The system uses BLE beacons that send out packets to indicate the presence of the beacon. These BLE beacons do not collect or distribute data but rather broadcast their presence. In the aforementioned implementation, the beacons are placed on the target vehicle. The detection system is placed in the ADAS vehicle, which forms part of the IVWSN network where the detection device monitors the Received Signal Strength Indicator (RSSI). Tire Pressure Monitoring System (TPMS) sensors can be used to send out packets, periodically acting as beacons of the target objects. In the US, all vehicles manufactured after 2006 and beyond have had to have TPMS installed.

These sensors could be used as beacons, sending out packets to detect possible collision threats in blind spots. Using Neyman-Peason's classifier as the detection algorithm, threshold parameters can be adjusted to set detection and false alarm rates. The data being collected by the wireless sensors is transferred to the ECU. BLE can also be used to detect vulnerable road users in blind spots by assuming that they have smartphones with them, which will act as the BLE beacons [77]. When a small program is installed on the user's mobile device, the programmed BLE-enabled device can constantly advertise its presence. The receiver node can detect these advertised packets, thereby contributing to the RSSI. When using BLE technology, it is assumed that the target is equipped with a BLE-enabled device. The BLE device should be in the form of a TPMS sensor, installed in the target's wheels, with the assumption that the driver in the blind spot of the vehicle has a smartphone that accompanies him/her. These are rare cases and do not cater for everyday scenarios.

3.3.4.2 Proximity sensors

Proximity sensors can detect objects close to the vehicle. Proximity sensors in the automotive industry include but are not limited to radar, LiDAR and ultrasonic sensors. Due to the cost of radar and LiDAR, other alternatives are favoured. Blind Spot Accident Prevention Systems (BSAPS), using ultrasonic and infrared (IR) sensors, either individually or in combination, use ultrasonic transducer pairs where low-level hardware is used to trigger alarms to notify users [56]. The ultrasonic sensors are placed close to the corners of the rear end of the vehicle where a data bus, accompanied by harnessing, is used to relay information to the ECU, as shown in Fig. 3.28.

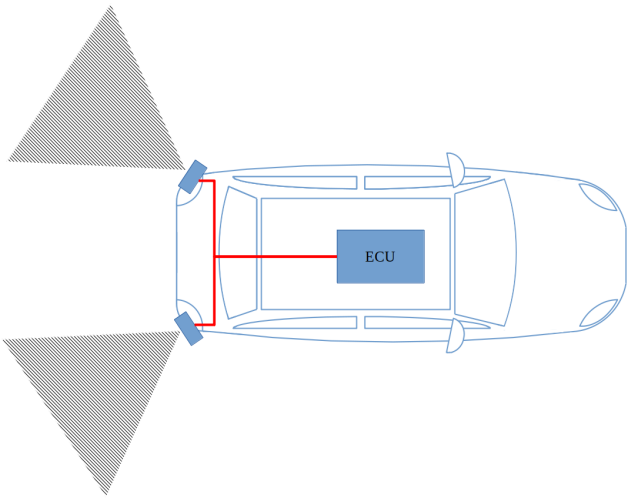


Figure 3.28. Ultrasonic sensor placement on a vehicle to capture blind spot information. Adapted from [56], © 2008 IEEE.

The blind spot detection can be incorporated within an IVWSN by capturing the data being collected from the ultrasonic transducers, and transmitting the data over wireless communication, Bluetooth or WiFi, to a smartphone or ECU. Unfortunately, these range sensors, such as ultrasonic and radar, provide no information about the type of object; therefore, their detection cannot assure a vehicle's presence but will provide some indication of a potential collision.

3.3.4.3 Camera-based blind spot detection sensors

Camera technology, using image processing, is taking the lead in blind spot detection, where CNNs are being used to detect objects. Camera sensors provide a larger field of view and provide more information to carry out processing and detection. Camera sensors are also cheaper, which make them suitable for blind spot detection. A vehicle detection system for the blind spot area of a car has been developed where the video from the cameras in the blind spots is analysed and warns the driver if a vehicle has been detected [38]. Many other techniques can be used where blind spot training data is used to train a detector with the help of AdaBoost. Fig. 3.29 shows an example of mounting a practical implementation of the physical camera, where the camera is mounted inside the car's right front window.

Figure 3.29. Camera is mounted on inside of right side window. Taken from [38], 2012 IEEE.

3.4 CONCLUSION

Different object detection techniques were investigated to develop an object detection feature for an ADAS system's collision avoidance. The evolution of CNNs was explored, as well as their building blocks. Many advancements have made CNNs accurate and faster to the extent that they can be used in real-time applications. CNNs make trade-offs between accuracy and speed where the model's size also limits usage when working with hardware, such as embedded devices and smartphones. MobileNets were investigated and were proven to be a better architecture when used in smartphones. Monocular distance estimators were investigated to add distance parameters to the collision avoidance system.

Traditional image processing techniques were explored to create a lane detection feature of the ADAS system. The most common techniques, such as Canny and Hough transformations, have shown to be the better approach to traditional lane detection. By systematically cascading techniques together, the smartphone should be able to carry out essential lane detection successfully.

Intra-vehicular wireless sensor networks have shown to be a complex environment. Creating a low-cost wireless ADAS system comes with challenges, such as using less hardware, but as shown, a smartphone can host a network with which sensors can interact over TCP/IP. By creating an IP network and using Wi-Fi, vision sensors can stream data to the smartphone for further use by the ADAS system. Bluetooth was also investigated as a means of communication between the sensors and the smartphone. Lower bandwidth communication, such as serial communication, was proposed as being useful in blind spot devices.

CHAPTER 4 INTRA-VEHICULAR WIRELESS SENSORY NETWORK FOR ADAS

Using an intra-vehicle wireless sensor network, structured by a Wi-Fi Direct and Bluetooth topology, a low-cost ADAS alternative extends a smartphone's sensory perception by using a camera-based wireless sensor network. As previously shown in Fig. 3.20, the Bluetooth sensors and Wi-Fi sensors communicate directly with the smartphone. A smartphone's processing power is harnessed, where the sensors are placed around the vehicle at areas of interest shown, as in Fig. 4.1. The figure contains vision sensors with a focus on the front and the rear of the vehicle. The sensors on either lateral side of the vehicle ensure that the vehicle's blind spots are covered.

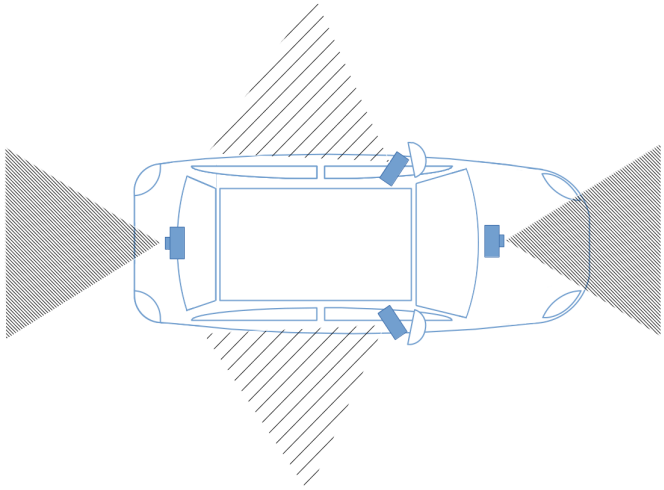


Figure 4.1. Illustration of the wireless sensors being placed in areas of interest for the ADAS system.

4.1 NETWORK TOPOLOGY

The network's main objective is to supply video feed to the processing unit, which is the Android device in the topology previously described. Wi-Fi, which is part of the IEEE 802.11 standard, is used to transfer the data to the mobile device. Wi-Fi requires a router or Access Point (AP) to manage routing

tables, policies, and local networks, usually done through an external router. Wi-Fi Peer-to-Peer (P2P), also known as Wi-Fi Direct, is used in the design architecture to avoid having to use a physical router in the topology. A P2P 1:n topology is used where multiple clients are connected to one Group Owner with a single SSID, with a single security domain [78]. Not all devices support Wi-Fi Direct, but Group Owners support client legacy devices that fall outside the Directional Multi-Gigabit (DMG) support. The transfer will still operate at IEEE 802.11g or newer 2.4GHz, supporting maximum physical bit rates of 54 Mbit/s. The legacy device supporting Wi-Fi, identifies the Group Owner as a standard AP, as long as the smartphone supports Wi-Fi Direct. Most off-the-shelf Wi-Fi modules do not support Wi-Fi Direct, and using a legacy approach is more attainable. The basic topology is shown graphically in Fig. 4.2.

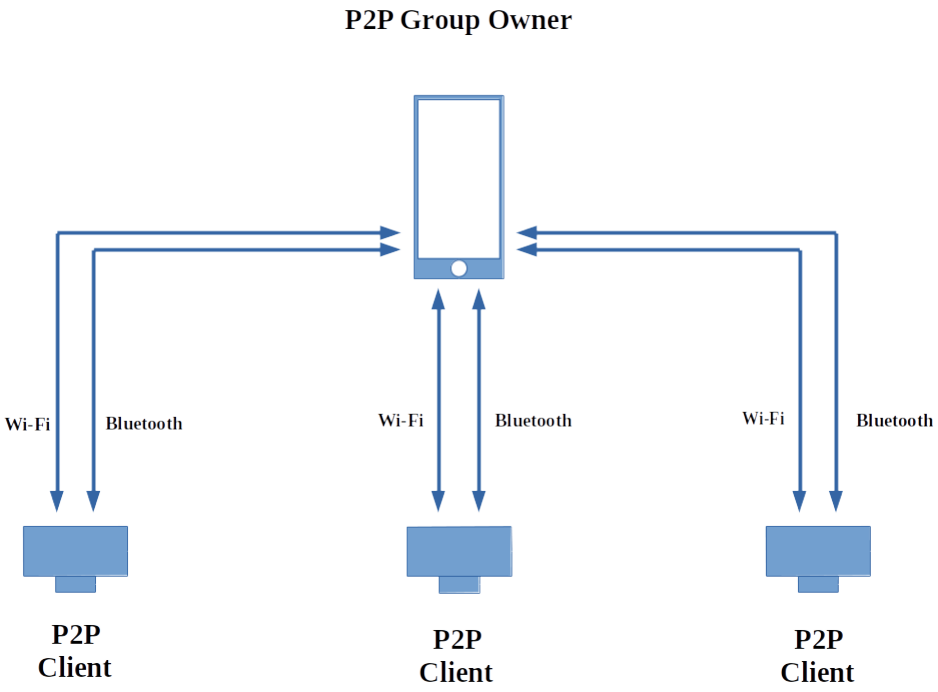


Figure 4.2. P2P toplogy 1:n, used by the camera nodes and smartphone.

The network consists of another Bluetooth network used by both the blind spot sensors and vision sensors, also shown in Fig. 4.2. The primary purpose of the Bluetooth network is discussed in following sections.

4.2 DATA COMMUNICATION

In the presented network topology, the network needs to be established by the Android device, which becomes the Group Owner on application startup. The camera nodes wait for the AP to start and then connect to the Group Owner. Depending on whether the network has been created before in the Android application's life-cycle, the network can be re-established with a created SSID and passcode, but when the Android API has created a new network, a randomly generated SSID passcode is created. This caveat of Wi-Fi Direct prevents hard coding of the AP credentials to the camera node. A means of communication is needed to update the credentials on the camera nodes, in order to update network security details. Bluetooth pairing between the sensors and the smartphone is used for this communication. Bluetooth has shown unsatisfactory results when used for video transfer, but many low-cost Wi-Fi modules include Bluetooth communication as well [51]. Bluetooth protocol has been used as a serial communicator between devices, leaving the Wi-Fi protocol exclusively for video streaming. Using the Bluetooth serial communicator, information such as the mentioned random generated passcode AP credentials can be passed securely to the Bluetooth-paired camera nodes from the Android device. Additional information (excluding video) can be shared, that will be used in future, such as restarting or communicating events in the network. Fig. 4.3 provides the overview of the communication between the camera node and Android smartphone. Even though the network consists of multiple camera nodes, the figures are focused on two-way communication between one client and Group Owner.

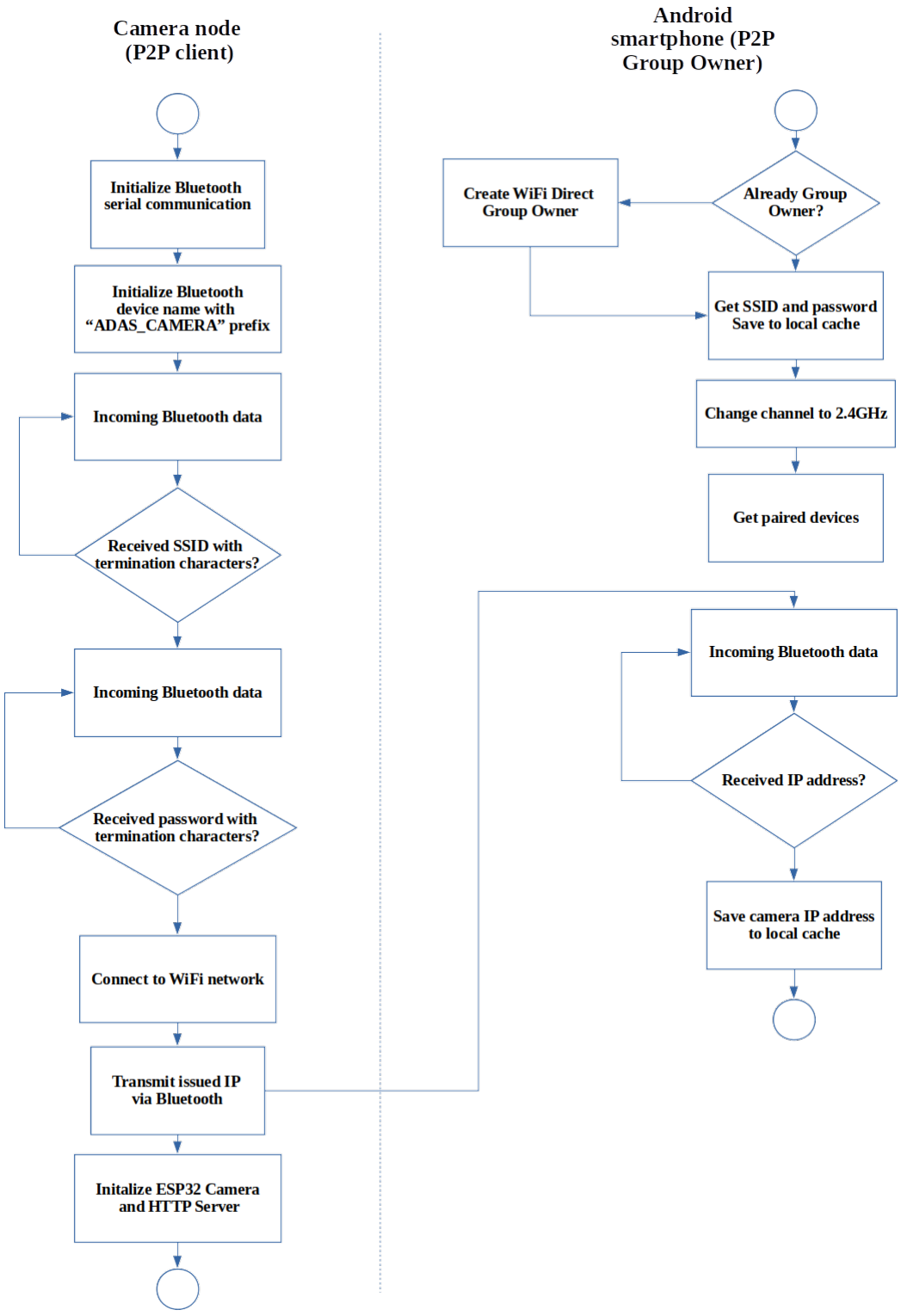


Figure 4.3. Network communication overview.

4.3 BLUETOOTH COMMUNICATION

The mobile device is Bluetooth-paired to the camera nodes for added security before the Android application is being engaged. The Bluetooth module does not require constant communication with the Android device but only on the node's initialisation. The initial Bluetooth connection does not mean that the devices will be connected through the application's life cycle, using Bluetooth. However, the devices will be known as paired devices when communication is needed. Only the SSID, passcode, and IP address are being communicated through Bluetooth between the camera node and Android device in the current implementation.

The Bluetooth protocol RFCOMM, in addition to L2CAP protocol, emulates an RS-232 serial port. The serial communication between the Android device and the camera node is used with a simple data stream and registered serial port service Universally Unique Identifier (UUID) to allow communication and use of this service [79]. After the connection to the camera node through Bluetooth protocol has been established, a socket is registered as an asynchronous task with the following pseudocode on the Android device:

Algorithm 1 Camera node Bluetooth connection algorithm

Require: $CN_1 \ldots CN_N$
  for $i \leftarrow 1$ to CN_N **do**
    - Register Bluetooth serial with Serial Port Service Class UUID
    - Connect to socket and create output stream
    - Write output stream with SSID from Group Owner
    - Write output stream with passcode from Group Owner
    - Read input stream sent from CN_i, containing IP address
  end for
  {CN = Camera nodes}

On each return of an IP address from a camera node, successful communication has been established. The IP addresses are saved in the local cache of the Android application to be retrieved when streaming has been initiated.

4.4 WI-FI COMMUNICATION

The network is being managed and hosted by the Android application as the Group Owner. The camera nodes are seen as legacy devices as they do not support Wi-Fi Direct, but this does not negatively affect the architecture, since other intricate Wi-Fi Direct features are not needed. The camera node works within 2.4 GHz networks only. Such a 2.4 GHz network creation will cause miscommunication if the Group Owner establishes a network set at newer 5 GHz networks, as seen on high-end Android devices,

which will cause the scan phase that has been initiated by the camera node to fail in connecting to the network. For this reason, the Android application is set to change the bandwidth on the creation of the network at 2.4 GHz channel 1 between the frequency range 2401-2423 MHz. Other channels, 1 to 14, can be explored if the need arises. The Wi-Fi network is hosted in a background thread of the Android application.

4.5 VIDEO STREAMING

The camera nodes host an HTTP server that serves JPEG images from the CMOS camera. Once the camera node has been connected to the network, the Android application can request data from the camera node, using the IP address acquired from the, previously mentioned Bluetooth engagement. The images taken by the CMOS camera are streamed through TCP on the Wi-Fi network that is hosted by the Group Owner. A port and HTTP end-point are opened on the camera node HTTP server, allowing GET requests with multipart content-type "multipart/x-mixed-replace" to stream the image frames to the mobile phone. A boundary parameter is passed to the content-type as a delimiter to separate body parts of data, coming from frames [80]. The connection is kept open as long as the client is requesting packets, allowing the Motion-JPEG (M-JPEG) stream to be processed by the Android device. The M-JPEG is not handled by the Android device as a video file but is received as an image bitmap instead. A raw bitmap supports future object detection implementation without stripping frames from a video file, thereby requiring extra processing on the Android application.

An ADAS system requires real-time capabilities where delays can be hazardous to the driver. The safety of the driver is an uncompromising factor that needs to take priority. Acceleration and deceleration time (ADRT) is used when the driver adjusts his/her speed. The time difference between the driver having received the visual signal and his/her reaction, needs to be slower than the video feed, which alternatively would serve as no use to the driver [51]. For the video streams to be helpful in a real-time application, the frame rate experienced by the driver on the ADAS assistance display should be at a minimal delay. Previous testing calling tasks, such as Bluetooth communication has caused the Android video output view experience to be staggered, due to shared processing. The camera capturing an HTTP server was placed on a dedicated core on the camera node's microcontroller to prioritise the video stream, in order to improve upon this problem. Another interaction, such as Bluetooth communication, was placed on another core with lower prioritisation. Prioritisation and core tainting were done by using FreeRTOS, a real-time operating system kernel for embedded devices and the previously used microcontroller's dual-core capability. Each camera node consists of the state machine,

as shown in Fig. 4.4.

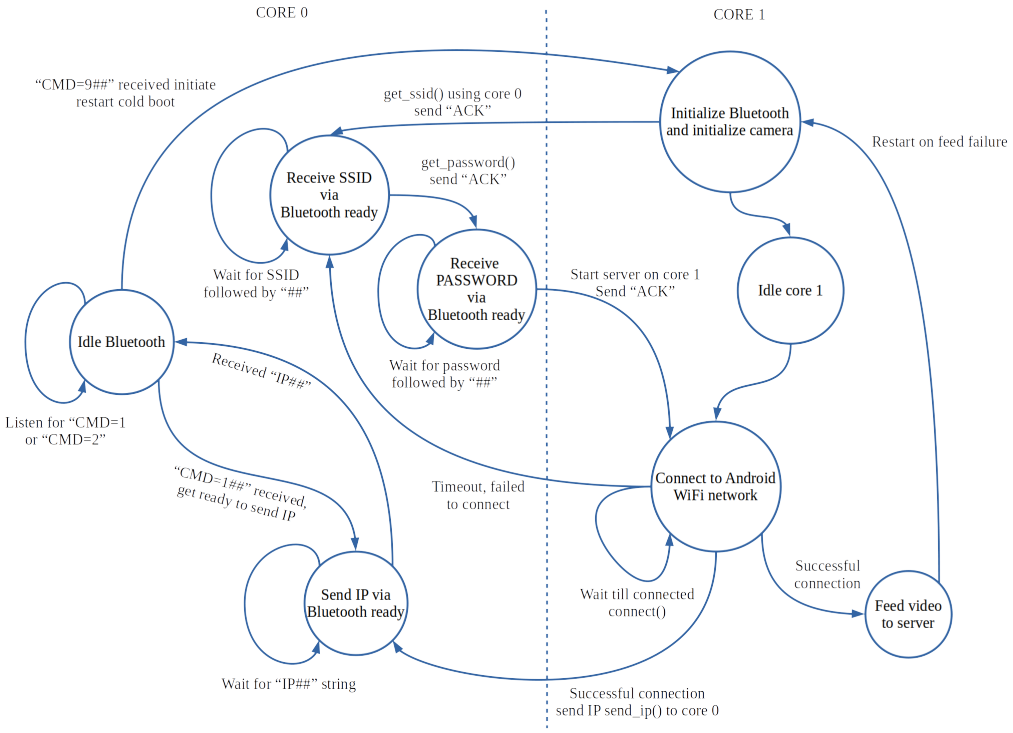


Figure 4.4. State machine of two cores for communication and dedicated video feed.

4.6 CAMERA NODE

4.6.1 Functional design

The functional diagram shown in Fig. 4.5 illustrates the design of the camera node. A low-cost OV2640 CMOS image sensor (FU1.1) captures images with an image array capable of maximum image transfer rates of 30 fps at SVGA and up to 60 fps in CIF [5]. The OV2640 camera chip also gives users control over image quality and formatting for future results optimisations (FU1.2). The off-the-shelf component ESP32-CAM development board that is used, contains the aforementioned image sensor , as well as the ESP32 32 bit microcontroller (FU2.1), capable of running the HTTP server housed with the Wi-Fi MAC and Bluetooth controller (FU3.1 and FU3.2)

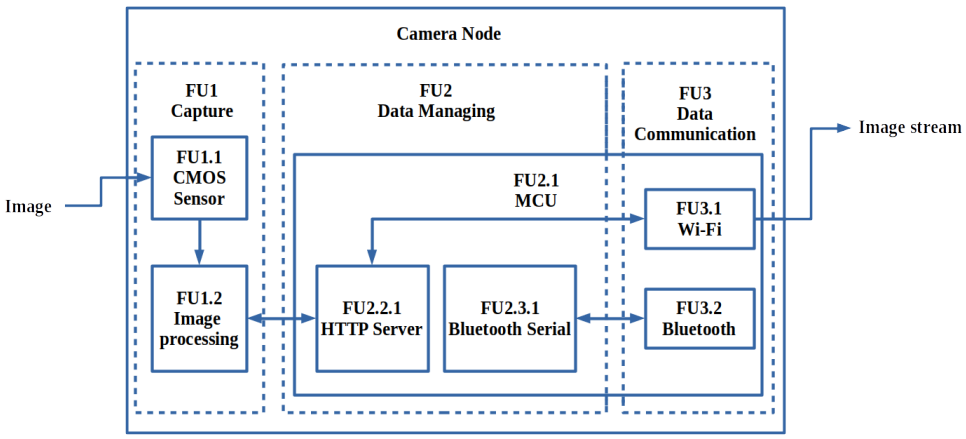


Figure 4.5. Functional diagram of the camera node, transferring image stream over Wi-Fi.

4.6.2 Design schematics and PCB layouts

A breakout board for the ESP32-CAM board has been designed that allows the development board to mount the camera node to a vehicle PCB, shown in Fig. 4.6. The camera node should be able to withstand outdoor elements to be able to take proper field readings. The PCB will be concealed in a protective container with ingress protection (IP), protected from dust and water. Fig. 4.7 shows the designed schematic, where FT232RL IC is a TTL/RS232 converter that was used to program the ESP32 through a USB connection. The camera node consumes power through the USB connection, but jumpers have been placed on consuming battery power for future field testing.

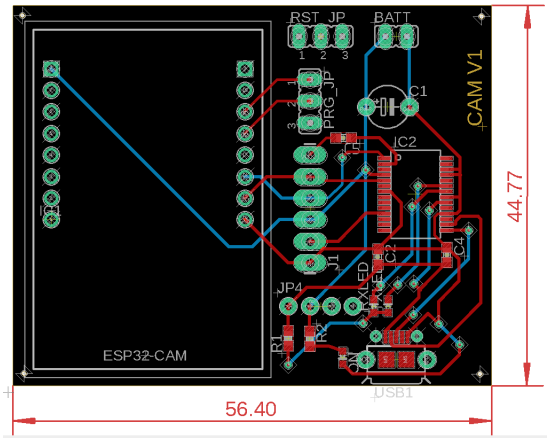


Figure 4.6. Designed camera node PCB board for ESP32-CAM breakout.

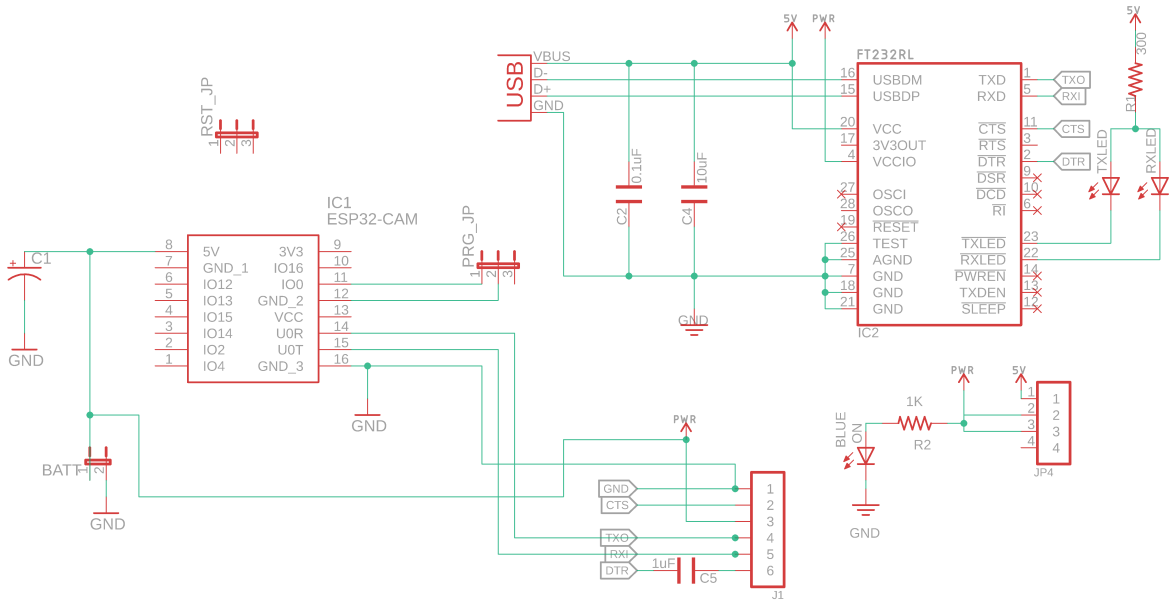


Figure 4.7. Schematic of the camera node.

4.6.3 Camera node embedded logic

The camera node's software was written in C to communicate with the ESP32 microcontroller. Before the camera node hosts the HTTP server, the Bluetooth controller is engaged by starting the Bluetooth serial service by broadcasting the device name. The device name is prefixed with "*ADAS_CAMERA*", which the Android application uses to filter paired devices for the ADAS application. The camera nodes remain in this state until the Android application sends the aforementioned SSID and passcode that has been terminated by consecutive 0x23 hex values to indicate successful transfer for both the SSID and passcode. The pseudocode for the camera node initialisation is as follows:

Algorithm 2 Android and camera node connection

Require: Initialise Bluetooth
- Device name prefixed "*ADAS_CAMERA*"
- Start serial communication service

while No terminating characters of hex 0x23 **do**
- Wait for SSID

end while
while No terminating characters of hex 0x23 **do**
- Wait for passcode

end while
Require: Initialise Wi-Fi
- Connect to Android hosted network

Require: Initialise Camera hardware

After credentials have been received successfully, the camera is initialised, and the Wi-Fi module

attempts to connect to the Android-hosted network. If a successful connection to the network has been completed, the local IP of the camera node is transmitted to the Android application, and the node's HTTP server is started. The lightweight HTTP server creates a listening socket on TCP for HTTP traffic, where user-registered handlers are invoked, and sends back HTTP response packets to the Android application. The server has one purpose, which is to feed the image feed from the camera where the server's port socket is set at 80, and the user-registered handler is an HTTP GET method at the root "/". The Android application then requests the feed from the URL "http://<camera node ip>:80". When the user-registered handler is invoked, an image is requested from the camera and converted to JPEG compression, which is then sent to the Android application. The invoked handler continues this process indefinitely until the connection is closed by the smartphone client.

Upon booting of the camera node, the Bluetooth communication is initialised with Wi-Fi communication. This only happens once during the camera node's life cycle, but this can fall over to a restart if a failure occurs, such as brownouts or system locks. Added handshaking has been implemented to force the video streaming to restart on the Android device command "CMD=9". This mechanism of using "CMD" with a number, is reserved to carry out other functionalities in future. After initialisation, FreeRTOS is used to initiate a task pinned to core 0 of the microcontroller. This core deals with Bluetooth communication only. The state machine on core 0's side receives the group owner's (GO) SSID and password from the Android application by having implemented the handshaking mechanism shown in Fig. 4.8.

If no errors occur and the camera node connects to the Android's GO successfully, the IP of the node's server is sent to the Android device being stored in the program's storage. By the time that the IP has been received and stored on the mobile device, the video capturing would have started on core 1. The image quality would have been downgraded, including smaller resolution improving frame rates over 25 fps for real-time detection. Three camera node prototypes and an Android phone are shown in Fig. 4.9.

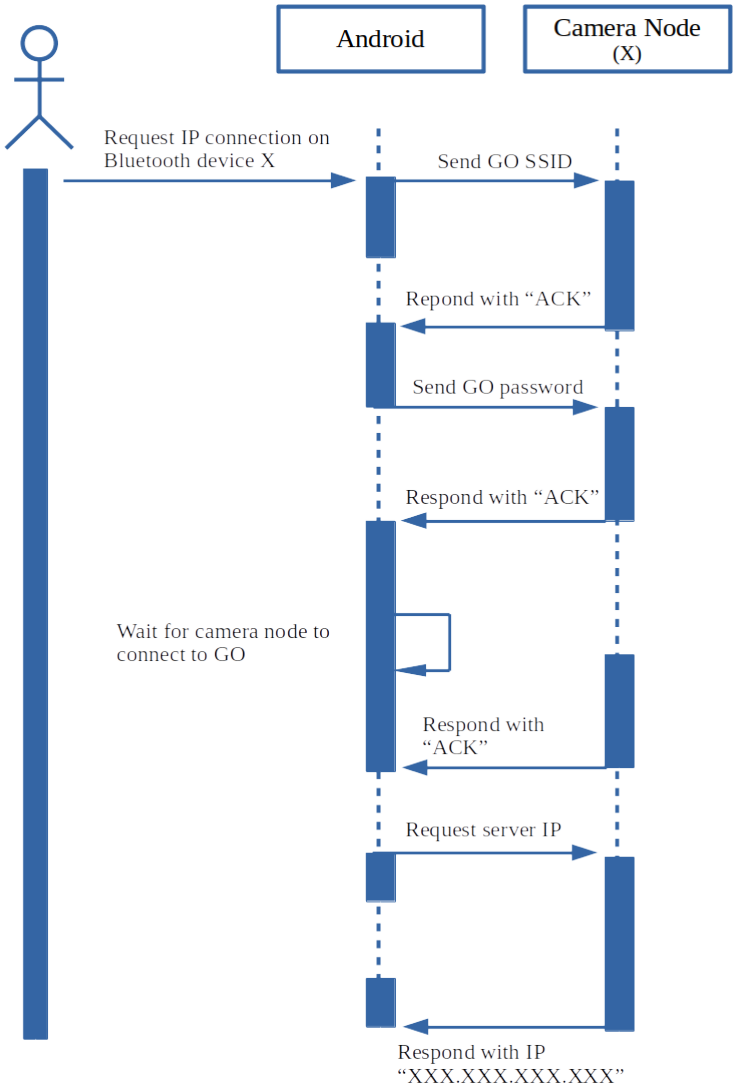


Figure 4.8. Bluetooth communication between Android application and camera node.

Figure 4.9. Android device and the three camera node sensors.

4.7 BLIND SPOT NODE

4.7.1 Functional design

The functional diagram shown in Fig. 4.10 illustrates the design of the blind spot node. The node consists of three main components, namely the proximity sensor (FU1), microcontroller (FU2) and Bluetooth communication (FU3). Focusing on keeping the entire ADAS system at a low cost, an ultrasonic sensor is used as the proximity sensor, which provides a 2 cm to 400 cm non-contact measurement with a ranging accuracy of 3 mm [81]. The ultrasonic sensor uses an echo and trigger mechanism that has been acquired by the microcontroller (MCU), where the distance is calculated by using the speed of sound. The same ESP32 development board being used for the camera node from previous investigations, was used as the MCU of the blind spot node. The ESP32 32 bit microcontroller can run the Bluetooth controller (FU3.1), which is used to send through the calculated distance that has been received from the ESP32 microcontroller and ultrasonic transceivers, using an RFCOMM Bluetooth serial.

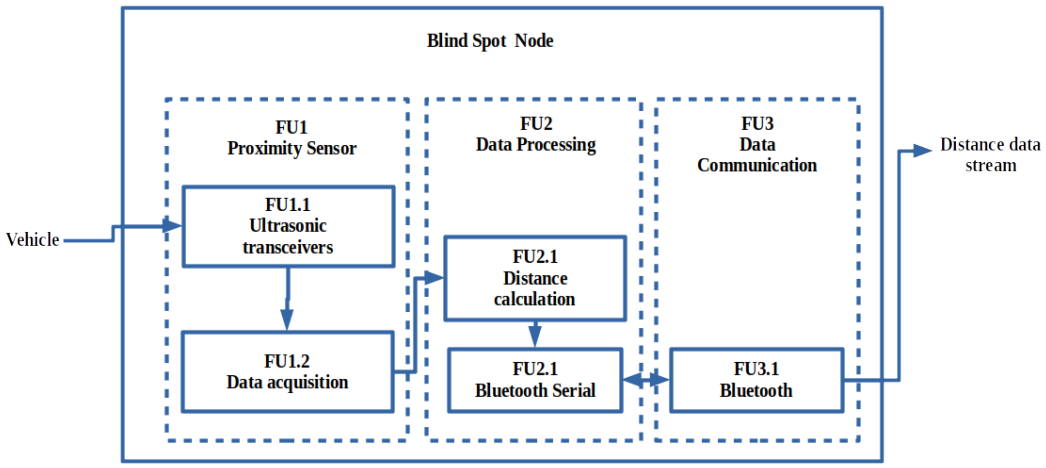


Figure 4.10. Functional diagram of the blind spot node, expressing the manner in which the distance measurement is streamed through Bluetooth.

4.7.2 Hardware design

A breakout board for the ESP32 development board, as shown in Fig. 4.11, was designed to mount the blind spot node to a vehicle's side mirrors. In order to take proper field readings, the blind spot node should be able to withstand outdoor elements. The PCB will be concealed in a protective container with ingress protection (IP), being protected from dust and water. Fig 4.11 shows a CAD design, harnessed on a vehicle's side mirror.

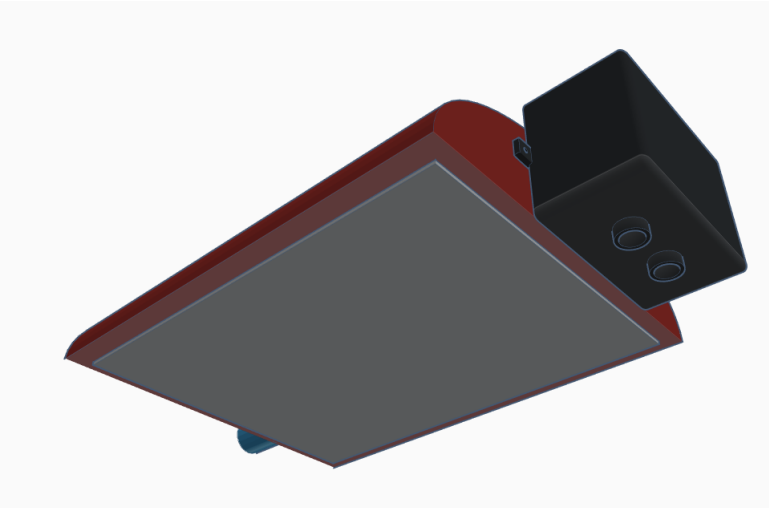


Figure 4.11. A CAD design of the blind spot node connected to a vehicle's rear-side mirror.

Fig. 4.12 and Fig. 4.13 shows the designed schematic where the FT232RL IC is a TTL/RS232 converter, being used to program the ESP32 through a USB connection. For this prototype, the USB connection is used to power the device from the vehicle. The USB can be replaced at a later stage with a battery in future field testing. The ultrasonic sensor is powered from the same source with GPIO pins from the ESP32 being used to fire the echo and trigger pins to calculate the distance between a potential vehicle and the device.

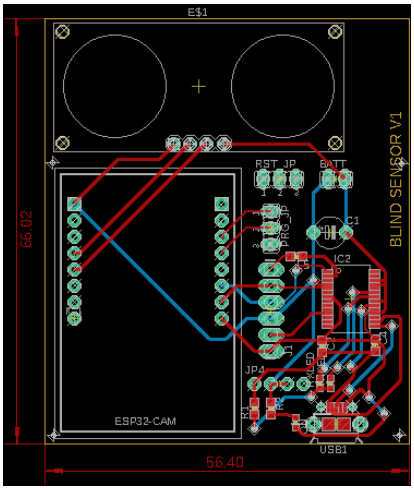


Figure 4.12. Designed blind spot node PCB board, including ultrasonic transceiver.

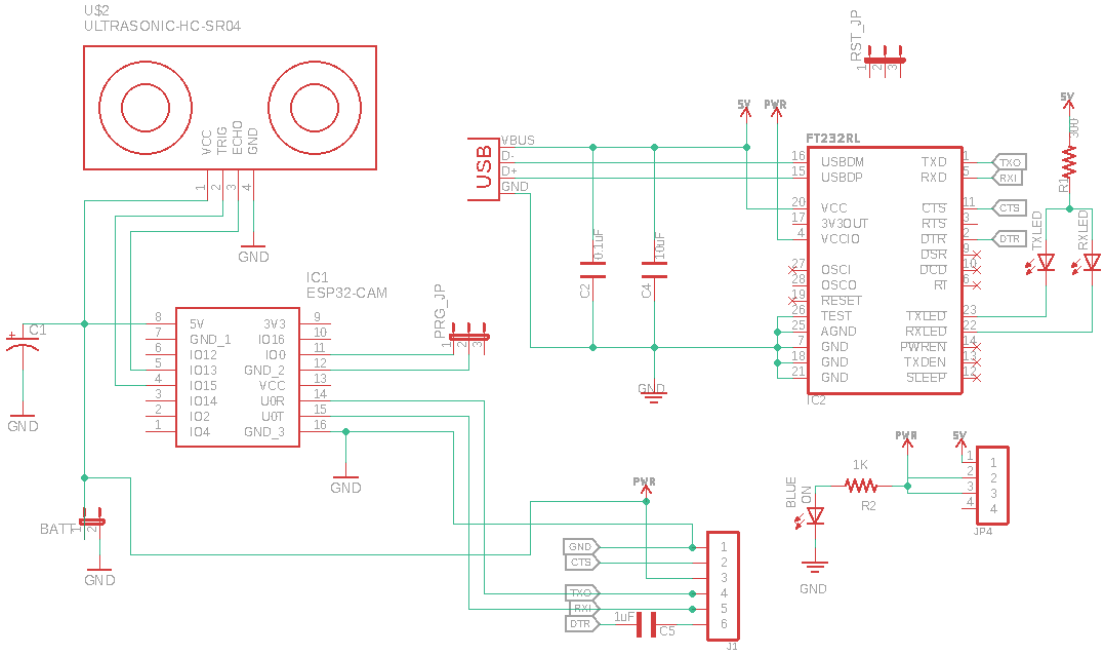


Figure 4.13. Designed schematic of the blind spot node.

4.7.3 Embedded software implementation

C programming language is used to write the firmware to communicate with the ESP32 microcontroller. The Bluetooth controller is engaged by starting the Bluetooth Serial service on the ESP32, where the device name is labelled as "*ADAS_LEFT_BLIND*" and "*ADAS_RIGHT_BLIND*" for blind spot nodes on both the left and right. According to the timing diagram in Fig. 4.14, a short 10 µS pulse needs to be sent to the ultrasonic sensor, which will then send out an 8 cycle burst at 40 kHz. The echo being received can then be used to calculate the distance, by using the speed of sound, as follows

$$x_{distance}(cm) = \frac{t_{echo}(s) \times 0.0343}{2} \tag{4.1}$$

Two pins of the ESP32 are directly coupled to the echo and trigger pins of the ultrasonic sensor, where the microcontroller can send and receive the echo pulse. The device is placed in an infinite loop with the single function of sampling distance values, which are directly passed to the Bluetooth Serial as a string value appended with a terminating character, which will eventually be used by the Android application.

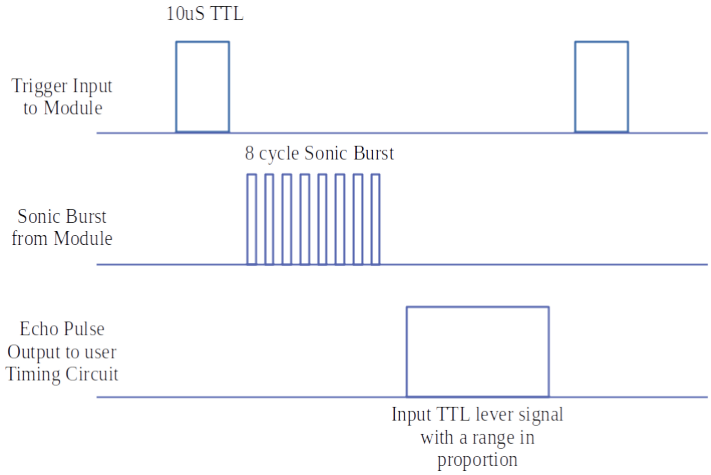


Figure 4.14. Timing diagram of the ultrasonic sensor.

The state machine of the node is illustrated in Fig. 4.15, showing a constant stream of distant values to the listener of the Bluetooth device. The receiver of the data will receive multiple values, of which the average can be taken on a selected amount of samples.

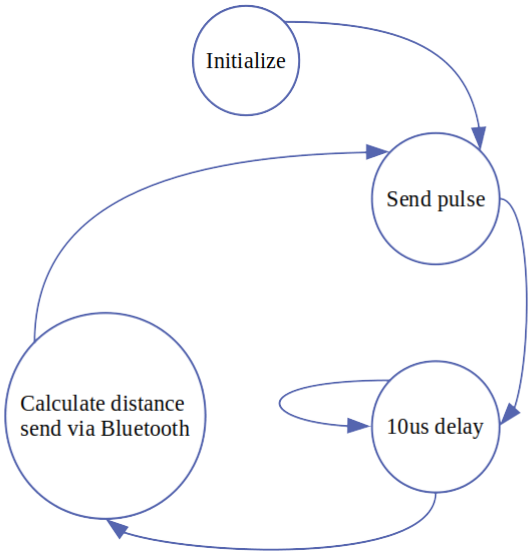


Figure 4.15. State machine of the embedded firmware.

The final nodes being built, were placed in casings with USB powered connectors, as shown in Fig. 4.16. The nodes are marked "LEFT" and "RIGHT" because they have been uniquely programmed in the firmware and have been made known to the Android device. The ultrasonic sensors pointing out from the side of the casing are directed to the blind spot area of the vehicle. The final prototype is shown in Fig. 4.16.

Figure 4.16. Left and right blind spot nodes.

4.8 SIMULATOR

4.8.1 Object detection

The camera simulation mimics the streams that would be received from the wireless video network and should be similar with regard to data being received from the implemented network. The wireless video from the hardware devices in the network stream continuous with JPEG images, where the simulated streams are also being sent frame by frame. The pre-recorded videos being hosted on the simulator are read from the file where frames have been sent through an HTTP multipart request. A multipart HTTP connection allows larger files to be passed through HTTP by keeping the connection open and by sending the video frame by frame; this will also be used in the wireless network, to be able to stream from the video network. The HTTP request is served from a local Python Flask server that provides an API end-point for each video stream. The host machine runs the already mentioned multi-stream server that exposes sockets that the object detection device can consume. An outline of the simulator is shown in Fig. 4.17.

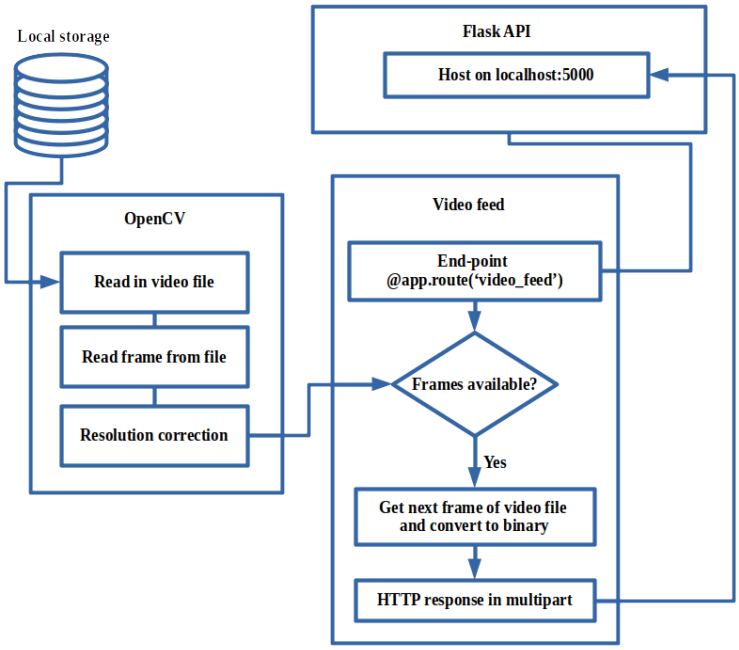


Figure 4.17. Basic overview of the multi-stream simulator.

The video files that are being streamed, are stored in the host machine's local storage where it is fed to a web interface, as shown in Fig. 4.18. In Fig. 4.18 only three streams are used as an example, as three live streams are being streamed from the API end-points to the front-end web browser. These individual streams can be consumed from an end device where object detection will take place. The end device can open an HTTP connection by using a URL, such as "http://<host-IP>:<port>/videofeed".

Figure 4.18. Screenshot of localhost web interface showing, multiple streams being served on host machine.

The simulator will be used in testing the object detection, as well as optimising detection. Other adjustments and optimisation of future smart video network switching algorithms will require the

designed simulator for performance testing. For future development, the simulated streams will be provided from recorded streams that are synchronously captured to optimise real-time video network scenarios, but will require data collection when all streams are being run and placed on the vehicle.

4.8.2 Lane detection

As an experimental implementation, the lane detection was written in Python, which was then written in JAVA on Android. OpenCV is a well-known library that focuses on real-time computer vision and has been used in this implementation [82]. The same library is ported and available to the Android environment, where the same implementation logic is used for the Android development after simulation experimentation had been successful. The implemented Python streamer uses a previously developed simulator to simulate the on coming road video stream. Each frame is captured and passed through the process shown by Fig 4.19.

The captured frame is copied so that the image processing can be carried out on the copied image, where the output results are then superimposed over the original image. The width and height are also extracted from the original image being used in a later process within the lane detection. The implementation consists of two main sections, namely the preprocessing subprocess, followed by the lane detection subprocess, as shown in Fig 4.19.

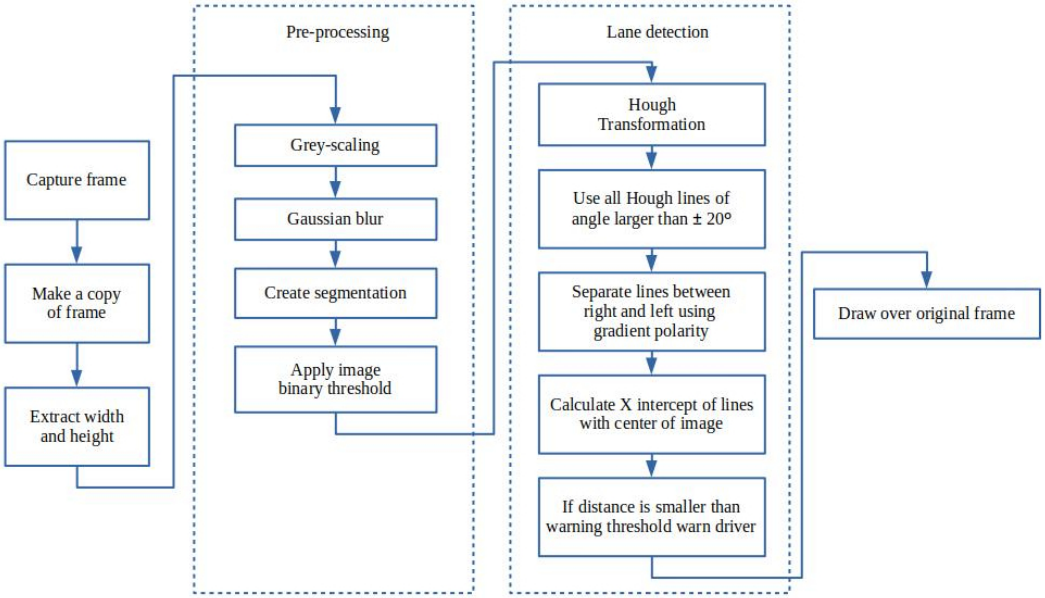


Figure 4.19. Functional diagram of the designed lane detection for lane assistance.

4.8.2.1 Image preprocessing

The captured frame in Fig. 4.20 shows the segmented image that has been grey-scaled, and that has gone through noise reduction, namely Gaussian blur where the segmentation has been extracted from passing the mask of zero matrix over the cropped-out polygon area. The segmentation allows the region of interest to be focused on the road, removing distractions above the lanes.

Figure 4.20. Captured frame that has gone through noise reduction and segmentation.

After segmentation, the processed image is passed through a threshold. The threshold is a binary output where the value on the image matrix is one or zero, depending on the intensity of the pixels. Most urban roads and highways have prominently painted white or yellow lines that contrast with the darker asphalt and will be distinctly noticeable above the threshold that is required for the lane output, as shown in Fig. 4.21.

Figure 4.21. Image threshold carried out on the segmentation.

Once the lanes have been amplified through the image threshold, the Hough transformation can be carried out for lane detection. The top section does not impede lane detection as no lines are being

detected due to zero padding, and white patches that may arise on the horizontal cut-off will also be dealt with in subsequent sections.

4.8.2.2 Lane detection

An edge detection mechanism is used to detect lanes. Hough transform assists in detecting imperfect straight lines, which will provide further assistance when dotted lanes are at play.

In this implementation, the preprocessed frame is passed through a Hough transformation that returns an array of lines. A gradient is then calculated for each line in the array, where a slope smaller than a set acute angle is rejected. This slope is expressed by

$$\theta = \left| \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \right| \tag{4.2}$$

By rejecting angles that are not within the set bounds, errors are avoided, such as horizontal lines that do not serve as possible road lanes.

Furthermore, to determine an estimate of where the vehicle is positioned relative to the lanes, a simple approach was taken by using the x-intercept of the detected lane and the centre of the width of the frame. The intercept of the lane was calculated by using the following

$$x_{intercept} = \frac{x_1 \nabla - y_1 - I}{\nabla}; \nabla = \frac{y_2 - y_1}{x_2 - x_1}, \tag{4.3}$$

where I is the image height in pixels, assuming a Cartesian plane of the image where a $(0,0)$ point starts at the top left corner, and the x-intercept is then used to calculate the distance from the centre by using half the width of the image. A graphical illustration shown by Fig. 4.22 contains two x-intercept dots outside the image.

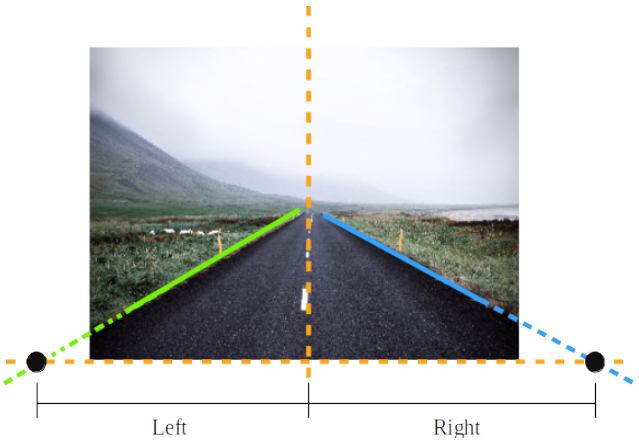


Figure 4.22. Illustration showing x-intercepts of detected lines to calculate length to centre of image.

By using the polarity of the gradients of the detected lanes, left and right lanes are distinguished by two different colours, as shown in Fig. 4.23. The figure also shows "Left" and "Right", followed by a unit-discussed x-intercept distance from the centre. The unit is the pixel count and will vary relative to different video feeds. The unit was used in the Python experimental implemented environment and can be set as a percentage of the distance.

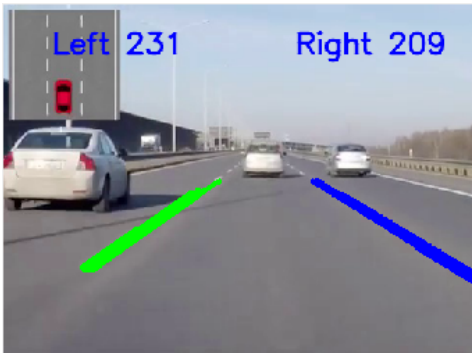


Figure 4.23. Final output, overlaying the Hough transform detected lines on original image and left/right distance from X-intercepts and centre.

Essential lane assistance has then been added, which will trigger a warning when the driver exceeds a certain tolerance of the left or right side lanes. The tolerance can be set as a percentage of half the width. Once the driver has exited the safe zone, the driver will be warned that the vehicle is diverging lanes. In Fig. 4.24 the danger zone has been entered on the lane to the right, and the driver is warned.

Figure 4.24. Example of lane assist, warning the driver that the vehicle has passed the crossing threshold.

4.8.3 Collision detection

A multi-scale approach to optimize detection discovery by using different detection windows to detect objects in proximity to the driver's vehicle was shown to deliver good results [14]. In this paper,

however, detection does not use these windows as detection windows. Instead, it uses them as collision potentials because the detection has already been done through a convolution neural network. The different regions are used as collision levels, namely near, intermediate and far, to warn drivers that a collision could occur. The far region wraps the vanishing point area in the horizon, where the closer regions wrap a larger area of the images, as shown in the Python simulation in Fig. 4.25.

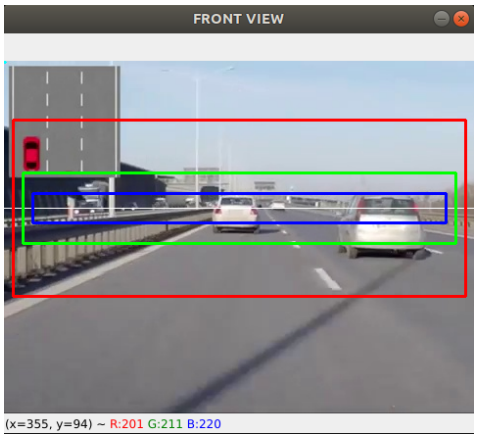


Figure 4.25. Different collision windows being centre-positioned on the vanishing point.

The overlapping region is calculated for each detection to determine whether a detected box falls within a particular region. As shown in Fig. 4.26, the overlapping area shows the manner in which the overlapping area is referenced where one of the bounding boxes is a collision region and the other a detection box.

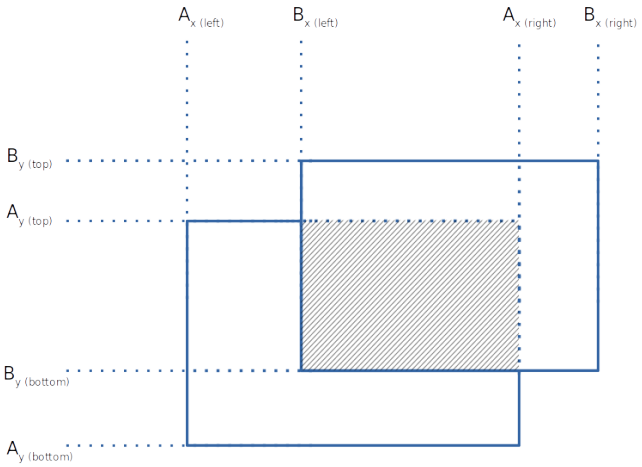


Figure 4.26. Overlapping boxes *A* and *B* with intersection area being etched.

The two metrics $Overlap_x$ and $Overlap_y$ are used to compute the $OverlapArea$, that can be calculated as the product by

$$Overlap_x = max(min(A_{x(right)}; B_{x(right)}) - max(A_{x(left)}; B_{x(left)})) \tag{4.4}$$

$$Overlap_y = max(min(A_{y(bottom)}; B_{y(bottom)}) - max(A_{y(top)}; B_{y(top)})) \tag{4.5}$$

$$OverlapArea = Overlap_x \times Overlap_y. \tag{4.6}$$

4.8.3.1 Calibration

A manual calibration process is required to determine the focal length, which is required before distance estimation can be carried out. By using the implemented Python simulator, the image is streamed through by only allowing one frame pass where the vehicle's distance and width are known. The simulator then returns the focal length, which can then be used in real-time detection. An estimate of the average vehicle width of 1.5 m is assumed, and the distance to the vehicle is then measured manually [83]. Fig. 4.27 shows the way in which the simulator allows the vehicle to be traced with a bounding box where the resulting value is the focal length.

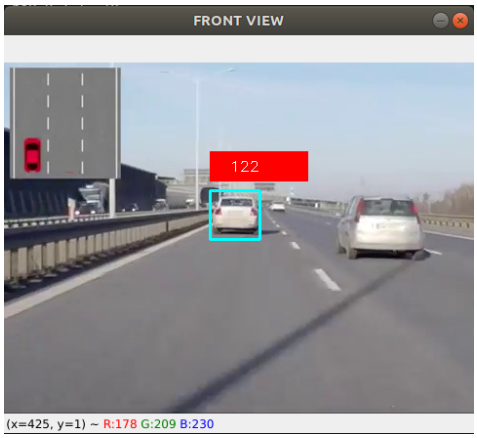


Figure 4.27. Positioning a detection box over the vehicle to calibrate and find the focal length.

After the focal length has been calculated through a calibration process, the distance of objects can be estimated by using the same equation, with the extension of adding a right angle calculation to determine the deviation from the centre of the image, which is better illustrated by Fig. 4.28.

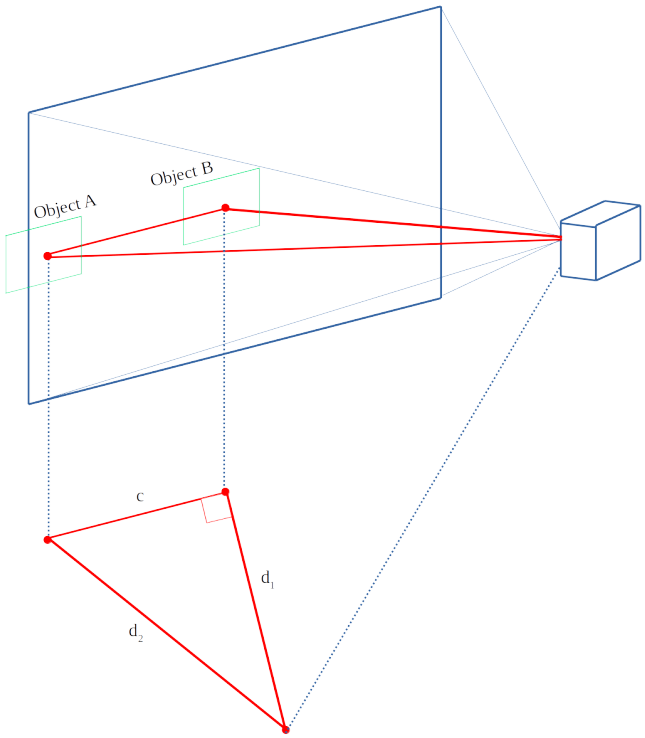


Figure 4.28. Projection of the camera, capturing the incoming scene.

The figure shows how the distance is recalculated when an object has been detected and deviates from the centroid of the image. The recalculated object is calculated by using the adjusted equation

$$d_2 = \sqrt{(\frac{width_{object} \times f}{width_{pixels}})^2 + (\left|CentroidX_{image} - CentroidX_{object}\right|)^2}. \tag{4.7}$$

4.8.3.2 Distance simulation

The distance to the centre of a detect box is calculated by using the aforementioned pinhole method. If the detected box deviates from the centre, the distance is adjusted by using Equation (4.7). The distance is processed, using the following algorithm:

Algorithm 3 Estimate distance

Require: $image, f, object, detectbox$
Ensure: $detectboxCentroidX \leftarrow (detectbox_{x2} - detectbox_{x1}) + detectbox_{x1}$
Ensure: $imageCentroidX \leftarrow \frac{image_{x2} - image_{x1}}{2}$
  if $detectboxCentroidX != imageCentroidX$ **then**
    $distance \leftarrow \sqrt{(\frac{width_{object} \times f}{width_{pixels}})^2 + (\left|CentroidX_{image} - CentroidX_{object}\right|)^2}$
  else
    $distance \leftarrow \frac{width_{object} \times f}{width_{pixels}}$
  end if
  return $distance$

Fig. 4.29 shows a screenshot of the simulator, determining the distance by using the aforementioned method. The vehicle detection occurred on the right and deviated from the centre, where the new distance was recalculated.

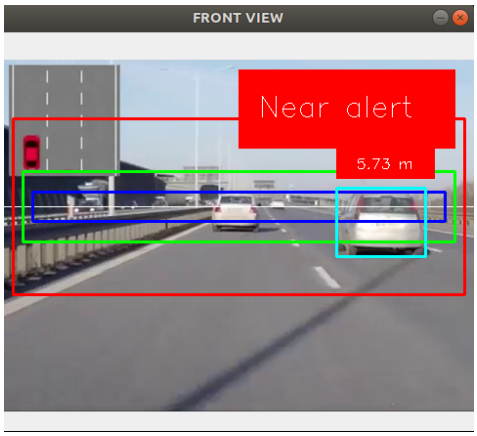


Figure 4.29. Distance estimation by using the focal length and pinhole method.

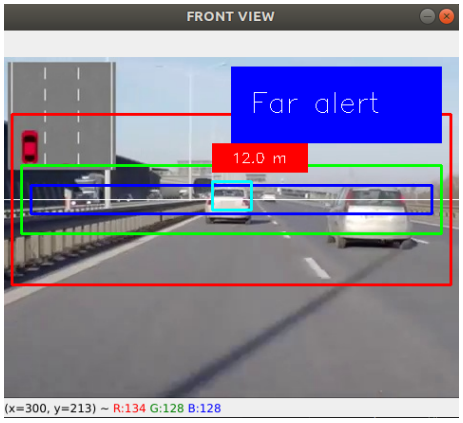
4.8.3.3 Collision warning simulation

The overlap of each detection box is computed by using the aforementioned collision estimation. The three regions, namely "near", "intermediate" and "far" overlay are calculated with the detection box area where each overlay metric, namely $OverlayNear$, $OverlayIntermediate$ and $OverlayFar$ is used to determine the collision risk. The "near" region is a higher risk and is given preference that gives regions "intermediate" and "far" a medium and low risk, respectively. The following algorithm was used:

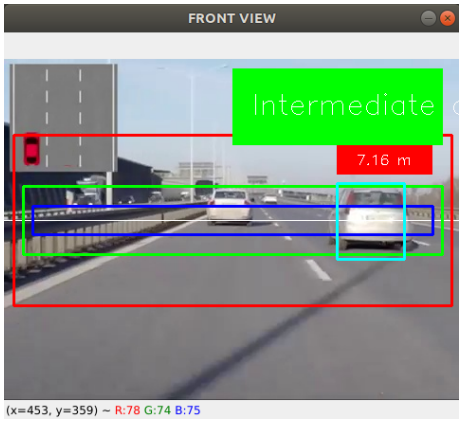
Algorithm 4 Calculate collision risk

Require: $OverlayNear, OverlayIntermediate, OverlayFar, DetectionArea$
Ensure: $oN = OverlayNear, oI = OverlayIntermediate, oF = OverlayFar, dA = DetectionArea$
  $DetectionRegion \leftarrow$ "$None$"
  if $oN == dA \ \& \ oI == dA \ \& \ oF > dA \times 0.8$ **then**
    $DetectionRegion \leftarrow$ "Far"
  else if $oN == dA \ \& \ oI > dA \times 0.8$ **then**
    $DetectionRegion \leftarrow$ "$Intermediate$"
  else if $oN > dA \times 0.8$ **then**
    $DetectionRegion \leftarrow$ "$Near$"
  end if

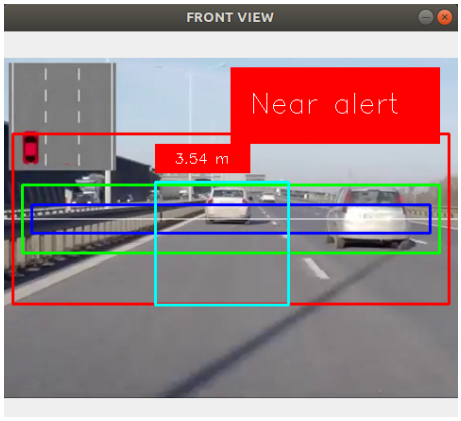
The collision regions have been determined by using different percentages of the image height. The regions are then drawn over the image where the aforementioned algorithm is then implemented in Python, as shown in Fig. 4.30. A detection area requires an 80% overlay to fall within a region where the "near" region is given preference. As shown with the intermediate detection box, preference is



(a) Far detection box



(b) Intermediate detection box



(c) Near detection box

Figure 4.30. Three different collision estimations.

given to the "intermediate" region over the "far" region. The collision risk is reassessed as the simulator is moving the detection box to a different size and detection area.

4.9 SMARTPHONE IMPLEMENTATION

The simulator was used to simulate the camera nodes, which assisted with development. The implemented methods that had used the simulator was rewritten in JAVA to run on an Android smartphone. Each module was built apart from the other to allow for further additions of ADAS methods, if it should be required. The Android implementation consists of a section that captures the image from the stream on the network where the rest of the modules use it to carry out algorithms. Fig. 4.31 shows the UML diagram, which will be described in the following sections.

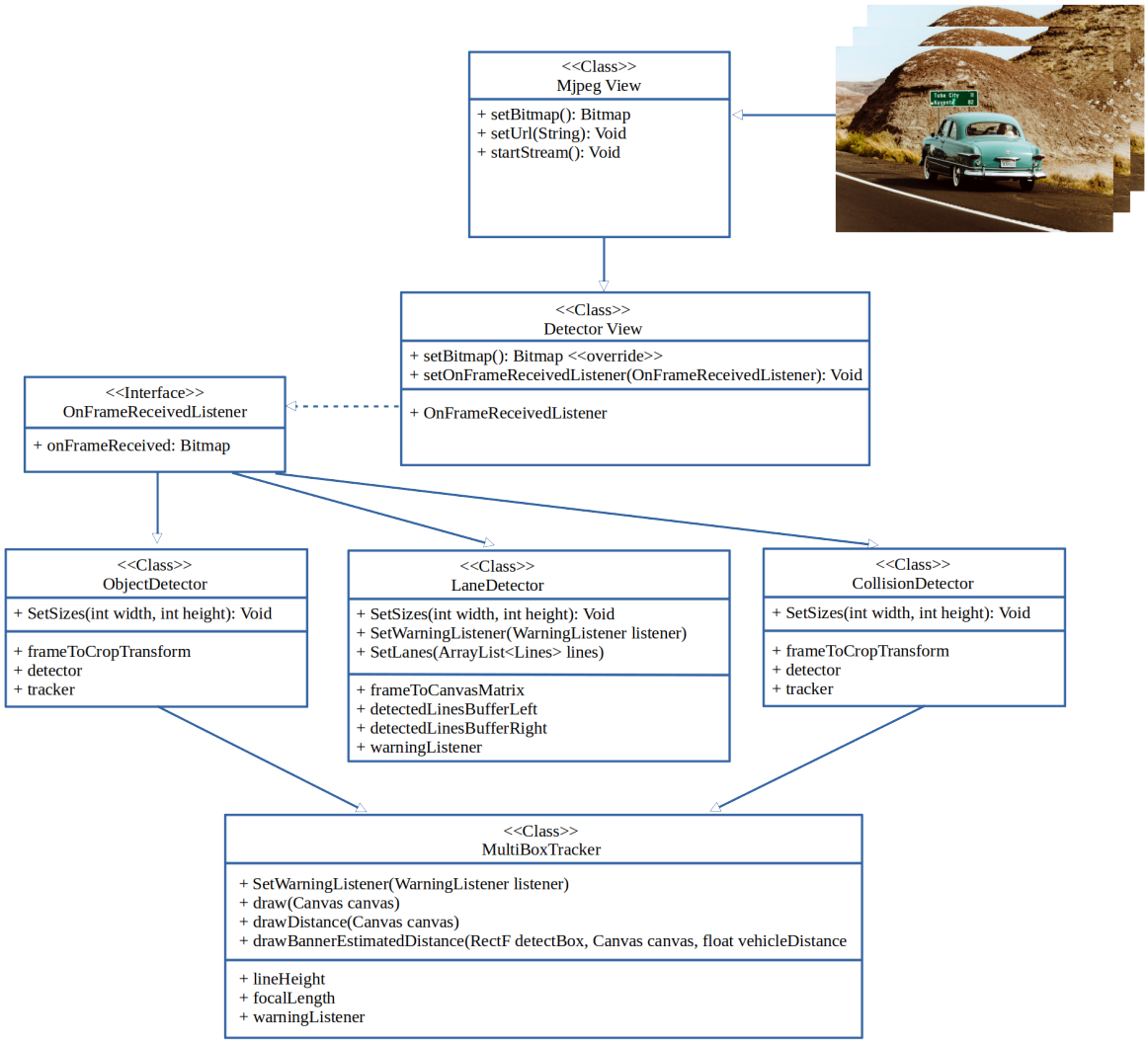


Figure 4.31. UML diagram, illustrating the capturing, processing and detection of a single M-JPEG image.

4.9.1 Object detection

The Android device can consume multiple streams because each microcontroller hosts its private server. After the IP addresses have been revealed to the Android device through Bluetooth communication,

the smartphone can access the video streams by accessing the camera node's server socket to stream images being captured through the HTTP protocol. Fig. 4.32 shows the three camera nodes that have completed the state machine processes being described earlier and have successfully supplied their IPs. At closer inspection, the IPs all fall within the same subnet that has been hosted by the Android device, which was created without the Android device having required a router.

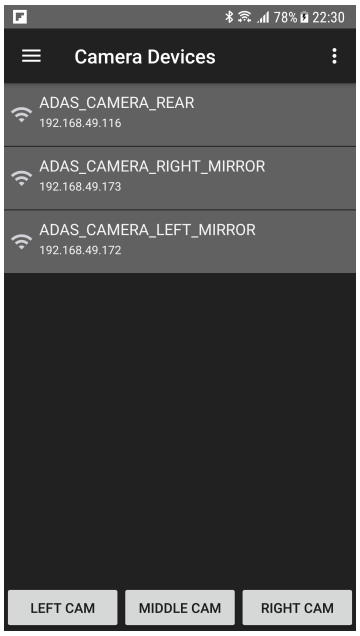


Figure 4.32. Three camera nodes connected to the Android's Wi-Fi Direct network.

The main fragment of the Android application consists of a DetectorView that inherits functionality from an MjpegView to get access to streaming Motion-JPEG. The IPs are stored in the application's program memory, using SharedPreferences. SharedPreferences are used to point to server URLs of the three camera node servers by using a setUrl method in the DetectorView. Once the startStream() method has been called, the streaming of the camera node server from the set URL is invoked. The DetectorView overrides the setBitmap() method of the MjpegView, where the bitmap from the camera node enters the application to continue processing. Fig. 4.31 shows a single image entering the process where the incoming bitmap is being passed through the callback listener OnFrameReceivedListener that forms part of an interface. OnFrameReceivedListener allows the bitmap to be shown on the main capturing view. The bitmap size is used to determine the detection window in the Detector class, using a method setSizes(). The detection window resolution is required to keep the ratio between incoming stream at bitmap size and the view area on the Android application by a transform matrix to transpose an image used by the bounding boxes being mapped to the canvas size in the Android view. In using

this approach, video streams are being fed directly through the detection model from the Detector class, where inference can be done and will allow bounding boxes to be superimposed over the incoming frame [84]. The detection is displayed on the main home window where only one stream is shown, but all three streams are sent through the same detection model and view. As explained in earlier sections, object detection has been carried out on the image frames as bitmaps that pass through to the detection view. These images are passed through a pre-trained model that returns a class, score and location. The class is the object type where 80 different objects have been pre-trained for the COCO dataset. The score is the prominence of the detected object, and location refers to the bounding-box locations around the detected object. The optimisation of the object detector was not investigated, since the focus is on an end-to-end system in this paper, and for this reason, a pre-trained SSD MobileNet model, being trained with the COCO dataset, has used. SSD MobileNet model is a well-known model that has shown to have competitive inference times with high accuracy, as compared to mobile benchmarks [85]. YOLOv3 turned out to be a large model that gave slow inference and performance times on low hardware devices. MobileNet has an inference time of 23 ms on 4-threaded CPU processing. This allows more than 30 frames to be processed per second, making the frame rate of the video stream feasible for real-time detection.

4.9.2 Lane detection

A lane detection module has been added to the previously implemented object detection. Both the object detector and newly implemented lane detector module use the DetectorView module that inherits functionality from an MjpegView that accesses the video stream. Once the startStream() method has been called and streaming starts, frames are passed to an onFrameReceived() method that allows bitmaps to be processed by either the object or lane detection modules. The lane detector keeps the ratio between incoming stream at bitmap size and view on Android by using a transform matrix frameToCanvasMatrix. This matrix is calculated so that the incoming bitmap and canvas size on the Android application will allow the detected lanes to be transformed to a scaled size on the Android application's view.

The same image processing steps being described in the Python implementation of OpenCV computer vision library has been used in the Android implementation. OpenCV's Android library has been referenced within the Android source code, where the incoming bitmaps from the DetectorView are converted to OpenCV's Matrix type. Gaussian blur, segmentation, thresholding and Hough transformation are then carried out on this matrix to find a matrix of lines.

The same sorting logic that rejects slope angles at a certain threshold is carried out on the Android implementation. Two buffers are used for the left and right lanes of captured lines. These buffers are used to paint lines over the canvas. A matrix of lines is then drawn on the canvas, shown visually to the UI for the user. The method SetLanes() consumes an ArrayList of Lines object that contains gradients and slope angles. The average distance to the centre for each frame is calculated using the total detected lanes that pass the set angle threshold, which forms part of the lane assistance.

In the same way, the experimental Python implementation has been carried out, where the user is warned about lane divergence when the detected lanes have exceeded the danger zone. A warning is activated by using a listener and SetWarningListener() method. This allows the UI to flash a hazard sign to the user when the user is drifting outside the boundaries allowed. The UML diagram with the LaneDetector module is shown in Fig. 4.31. The UI allows the user to toggle between object detection and lane detection where the different modules are instantiated and will start processing incoming bitmaps.

4.9.3 Collision detection

Object detection has already been implemented on Android from incoming video streams on a video network. A collision avoidance module has been added to the previously implemented object detection. Both the object detector and newly implemented collision avoidance module use the DetectorView module that inherits functionality from an MjpegView that accesses the video stream. Once the startStream() method has been called and streaming starts, frames are passed to an onFrameReceived() method that allows bitmaps to be processed by either the object detection, lane detection or collision avoidance modules. The collision avoidance module keeps the ratio between incoming stream at bitmap size and view on Android by using a transform matrix frameToCanvasMatrix. This matrix is calculated so that the incoming bitmap and canvas size on the Android application will allow the detected boxes to be transformed to a scaled size on the Android application's view.

The same algorithms being described in the Python implementation of the collision avoidance have been used in the Android implementation. The tracker being used in the object detection module has been refactored to accommodate the addition of a distance and collision risk area detector. The MultiBoxTracker, shown in Fig. 4.31, that is used in object detection has already had an implemented draw() method from previous development, that drew boxes over detected objects. A drawDistance() method has been added to draw the collision areas and carry out the collision risk and distance

estimation by using the aforementioned pinhole approach.

In the same way, the experimental Python implementation has been carried out. The user is warned about a collision risk when objects are entering the collision regions. A warning is activated by using a listener and SetWarningListener() method in the MultiBoxTracker. The listener allows the UI to flash a hazard sign to the user if the driver is at risk. The UI allows the user to toggle between object detection, lane detection and collision avoidance where the different modules are instantiated and will start processing incoming bitmaps.

4.9.4 Blind spot detection

The Android application harnesses the telemetry from the blind spot node being passed via Bluetooth by using the distance determined by the detector. The driver needs to be warned if a car is entering the vicinity of the vehicle's blind spot. A new blind spot module has been added to the already existing Android application. The BlindspotDetector module runs independently on its thread, updating the user interface (UI) and forming two new threads for both the left and right blind spot detectors. The previously implemented BluetoothHelper, which has been used to send credentials in the camera nodes' WiFi network, is extended to connect to "*ADAS_LEFT_BLIND*" and "*ADAS_RIGHT_BLIND*" blind spot nodes upon start-up of the application. As the application is starting, two RFCOMM serial sockets are initiated to receive byte packets. The average of the samples being received at an instance is taken and used to determine the danger zones 1, 2 or 3. The processed distance is then passed to an onDistanceReceived() method to invoke the UI update. The BlindspotDetector contains an onWarningDetected() method, which is overridden on the UI, which is then invoked when the detector threads have detected a blind spot distance that is dangerous. Three different zones can be sent to the onWarningDetected() method, depending on the distance being detected.

4.10 CONCLUSION

It has been shown that an IVWMSN network for an intra-vehicle environment that uses low-cost camera nodes, blind spot nodes and a smartphone device, could be developed. By using smartphone-implemented Wi-Fi technologies such as Wi-Fi Direct, the smartphone can act as the network host onto which video sensors can directly connect without adding a physical router as part of the architecture. By using the Wi-Fi protocol, higher transfer rates can send sensory data from a video sensor to enable advanced object detection around a vehicle. Video streaming from the network can then be used for real-time object detection applications for an ADAS system on a smartphone device. Detection has successfully been carried out on video being sourced from a simulated video stream and network-

sourced video streams. Due to large models not being able to run on smaller embedded devices, such as smartphones, they have used a faster and lighter CNN, such as MobileNet, instead.

Lane detection and collision avoidance have been implemented for the ADAS system, running on Android. It has been shown that a low-cost ADAS system, using a smartphone, can carry out image processing techniques capable of detecting lanes on a real-time video stream. It has also been shown that a low-cost ADAS system, using a smartphone, is able to carry out object detection techniques where distance estimation and collision risk can be calculated on a real-time video stream. The real-time video stream being sourced from the intra-vehicular wireless video network, is used to assist the driver by detecting lanes and warning the driver when he/she is moving closer to the centre of that lane. The implemented system helps the driver by warning him/her visually on the screen about lane divergence to the left or right of the vehicle. The real-time video stream also assists the driver by warning the driver when his/her vehicle is moving closer to hazardous objects. The implemented system helps the driver by warning him/her visually on the screen about a possible collision in the front or rear of the vehicle.

Blind spot detection has been added to the implemented ADAS system running on Android. It has been shown that a low-cost ADAS system, using a smartphone, can carry out blind spot detection where ultrasonic sensors can detect the distance of a vehicle in the driver's blind spot, in order to prevent a collision. The Bluetooth devices form an addition to the already developed network that provides a real-time video stream being sourced from the intra-vehicular wireless video network. The blind spot addition will assist the driver with the help of visual warnings on a screen if a possible collision in the blind spot of the vehicle is going to occur.

CHAPTER 5 RESULTS AND DISCUSSION

5.1 NETWORK PROPAGATION

5.1.1 PyLayers coverage simulation

As previously explained, the smartphone is the Group Owner in the Wi-Fi Direct network and receives packets from the vision and blind spot sensors. The network hosted by the smartphone uses Wi-Fi IEEE 802.11b with a transmitting power of 17±2 dBm, which is used in the coverage simulation with a spectrum range of 2412 - 2484MHz. A vehicle environment is a complex environment that includes loss mechanisms, such as seats and dashboards, and multipath components, such as metallic bodywork. Network coverage simulations were carried out to ensure optimal sensor placement. PyLayers is an open-source site-specific radio channel simulator being used as the simulating software. A conceptual 3D model, as shown in Fig. 5.1, illustrates the cutting planes at 1 m where the mobile device was placed in the simulation.

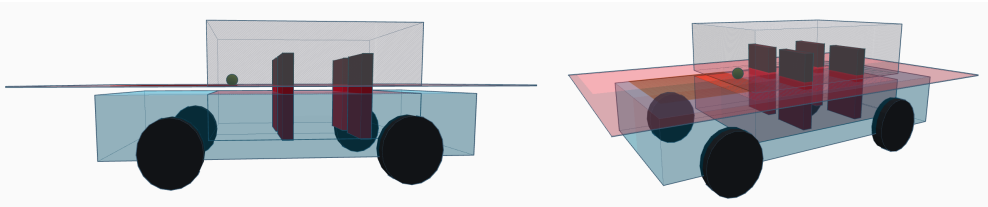


Figure 5.1. The vehicle model used, showing the yellow sphere and the red cutting planes at 1 m height, where simulations results are measured.

Suppose that the smartphone is placed inside the vehicle at a lower height. In that case, the vehicle's metal body causes most of the power to remain inside the vehicle because of the reflective metal properties. This results in unwanted multipath propagation, which will result in latencies. For this reason, placing the mobile device higher, at window height, will allow power to be transmitted to sensors outside the vehicle. Fig. 5.2 shows the simulation where optimal placement would be directly in front of the rear of the vehicle.

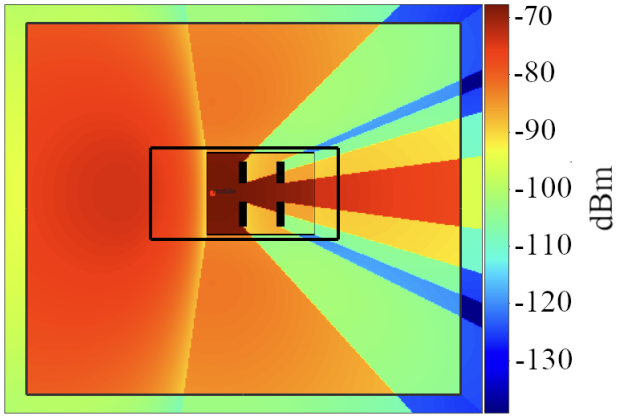


Figure 5.2. An IEEE 802.11b with $f_c = 2.412GHz$ polar plot simulation estimate of the received power at cutting plane set at a height of 1 m.

5.1.2 Actual network coverage

The camera nodes are then placed around the vehicle's intra-vehicle environment. One camera node is placed in the front of the vehicle and another at the rear. The signal strength is measured inside the vehicle where the smartphone will be placed for the driver, when connecting to sensors. The signal strengths are plotted in Fig. 5.3.

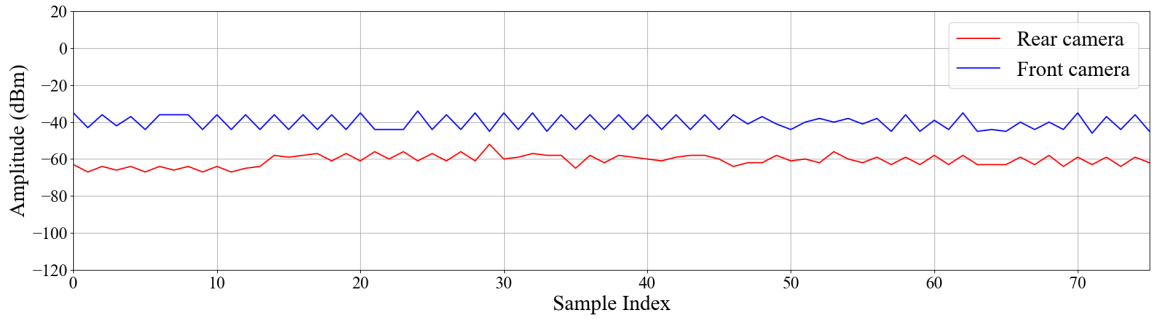


Figure 5.3. Signal strength of the intra-vehicular network, measured from sensor placement at front and rear of vehicle.

Given that the smartphone is mounted onto the centre of the dashboard, the simulated environment shows that the Wi-Fi signal that is/has been propagated, is best-suited to the front and the rear of the vehicle, where the cornered rears of the vehicles show evident signal attenuation. The front lateral side is where the left and right mirrors are also more suitable for sensor placement above the metal work.

5.2 NETWORK PERFORMANCE

Tests have been carried out, using different environments. Firstly, the nodes are simulated by using simulator nodes, hosting streams at different resolutions. The different streams consist of low, medium, high and ultra-high resolutions, tested at concurrent connection combinations of 1 to 6 streams. The simulator ensures that each stream acts as a camera node by looping the same video at different resolutions. Network performance is then shown for the actual hardware camera nodes in a controlled environment to illustrate the manner in which the network and camera nodes perform without an intra-vehicle environment. Lastly, an intra-vehicle environment is used where camera nodes are placed inside the vehicle's front and rear to take the same readings as the controlled environment.

5.2.1 Simulated environment

The simulator mimics the same transfer method that the hardware camera node is carrying out. As mentioned, the camera node transfers images, in JPEG and over HTTP, which are captured from the CMOS sensor. The simulator uses a sample video, originally in ultra-high 4K, which has been converted into four lower resolutions, set at 120 p, 320 p, 720 p and 1080 p. The streams on the simulators have not been limited to 30 frame rate streams as already discussed in previous sections, which have been used for simulating real-time video over JPEG multi-path requests on HTTP. By not limiting the rate of the simulator, the client can reach the maximum frame transfer being allowed. Table 5.1 shows the simulated results of a two-node camera network, as it would be used on a vehicle.

Table 5.1. Results for different resolutions, using two simulated nodes.

Resolution		Node 1	Node 2
120p	Frame rate (FPS)	257	219
	Throughput (bit/s)	24 992 826	21 463 908
320p	Frame rate (FPS)	38	34
	Throughput(bit/s)	24 444 562	22 116 785
720p	Frame rate (FPS)	2.5	3
	Throughput(bit/s)	9 114 936	9 209 112
1080p	Frame rate (FPS)	1	1.2
	Throughput(bit/s)	5 893 721	5 843 405

Table 5.1 shows the manner in which the frame rate drops as the resolution is being increased. The throughput also decreases, indicating that the data does not obstruct the network's speed, but latency is

caused by the post-processing of the application that has requested the data.

Six streams are being run from one to six concurrent streams at different resolutions, and then the averages of the frame rates per second were calculated, as shown in Table. 5.2. At the lowest resolution, frame rates are the highest, with decreasing frame rates as the streams are increasing. As the resolutions are being increased, frame rates drop to rates that do not accommodate real-time applications.

Table 5.2. Average frames per second of different resolutions at different streamed nodes.

Resolution	1 stream	2 streams	3 streams	4 streams	5 streams	6 streams
120p	522.22	265.91	175.64	131.91	107.31	87.73
320p	47.02	38.90	22.76	20.35	14.64	13.47
720p	3.57	2.65	2.43	2.02	1.71	1.52
1080p	1.04	1.00	1.00	1.00	1.00	1.00

At high resolutions, the current solution cannot support an ADAS system. Improved transmission methods, such as HTTP Live Streaming (HLS) and Real Time Streaming Protocol (RTSP), would improve this by incorporating improved encoding and decoding strategies, but would require more demanding hardware and increased cost.

5.2.2 Controlled environment

The Wi-Fi network performance has first been tested without the vehicle involved. Doing this, the network's performance can be investigated before the complex intra-vehicle environment may cause interference. Only a rear and front camera node is required, but an extra node is added to obtain different results to test the network's performance. The cameras are then placed next to each other, with the smartphone close to the camera nodes. A TCP dump is carried directly onto the Android device for 10 minutes while all three streams are running on the Android device. Fig. 5.5 shows the throughput of the network, averaging around 400 kbit/s for the three camera nodes.

Furthermore, the frame rate is being recorded during the same 10 minutes. The frame rate is a crucial determining factor for streaming in real-time. The real-time frame rates of all three camera nodes being fed to the Android device is shown in Fig. 5.6. The frame rates are captured by counting and logging the number of frames received by the Android device every second.

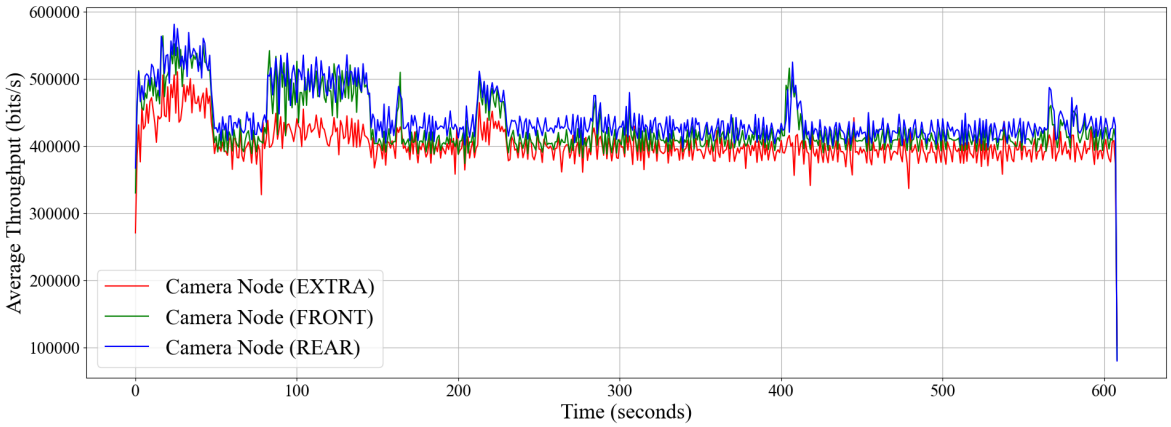


Figure 5.4. Throughput of three camera nodes over 10 minutes.

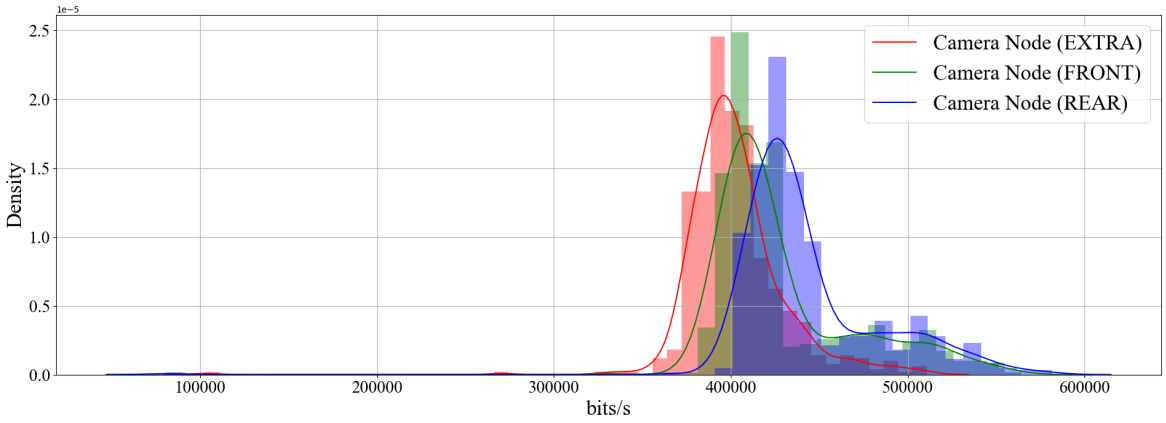


Figure 5.5. Density plot of throughput for three camera nodes over 10 minutes.

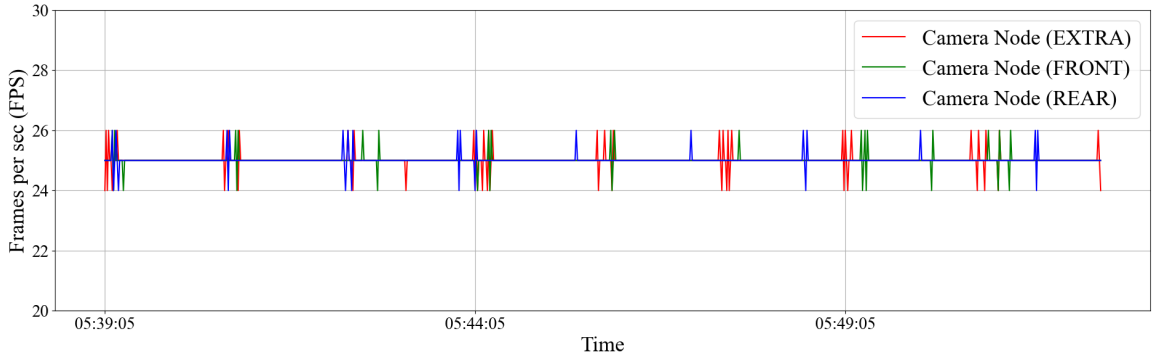


Figure 5.6. Frame rate of all three camera nodes being fed to the Android device over a 10 minute period.

By expressing the collected frame rates of each camera, using Kernel Density Estimation, the plot shown in Fig. 5.7 reveals an average of 25 frames per second (FPS).

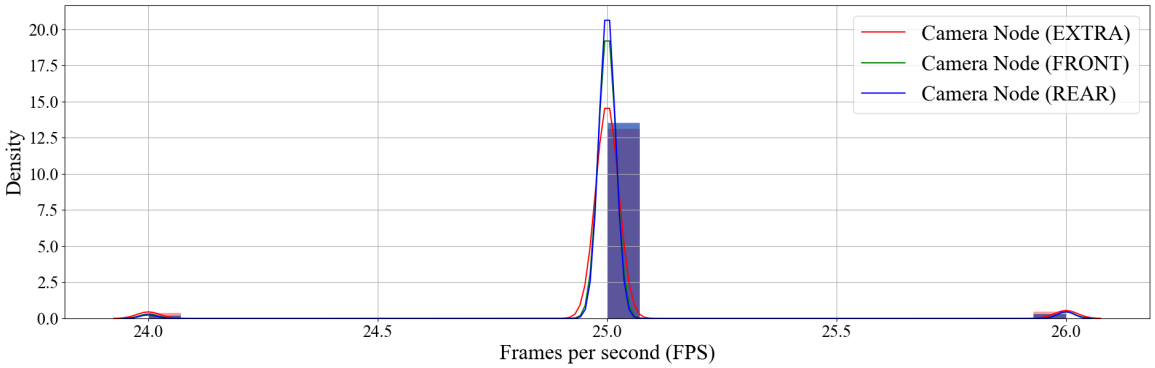


Figure 5.7. Frames per second expressed through a Kernel Density Estimation plot in a controlled environment.

At a constant average video feed rate of 25 frames per second, the ADAS system will operate in real-time. In normal driving conditions, the frame rate allows the ADAS system ample time to give adequate feedback to the driver, in order for the driver to react.

5.2.3 Intra-vehicle environment

The intra-vehicle environment has been tested by placing the sensors externally at the front and at the rear of the vehicle. An extra camera node is also placed on the passenger seat to be compared to the controlled environment. Readings are then taken during the 10 minutes. The smartphone is mounted inside the vehicle at the driver's seat near the vehicle's dashboard. Fig. 5.8 shows the throughput of all three sensors and Fig. 5.9 reveals the density plot. The throughput of the network hovers at under 300 kbit/s.

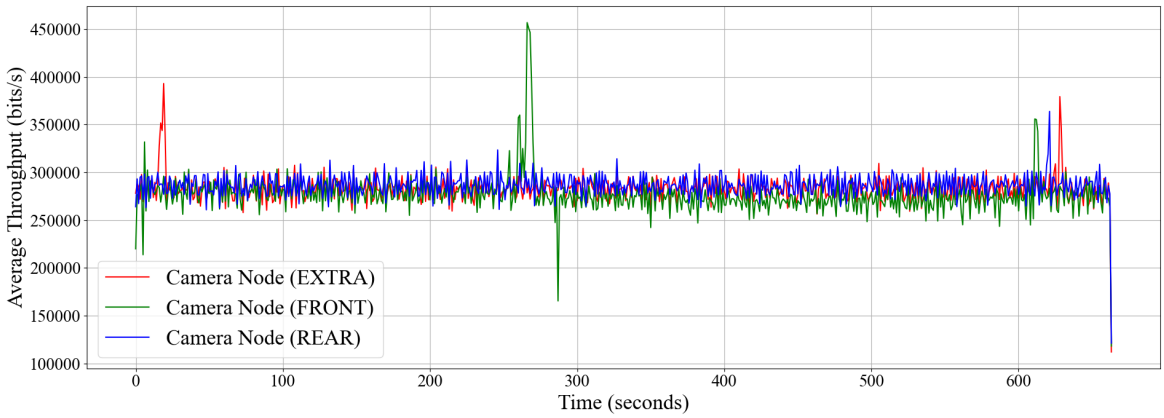


Figure 5.8. Throughput of three camera nodes in an intra-vehicle environment where readings were taken over 10 minutes.

Frame rates are being captured for 10 minutes through the Android network, as shown in Fig. 5.10.

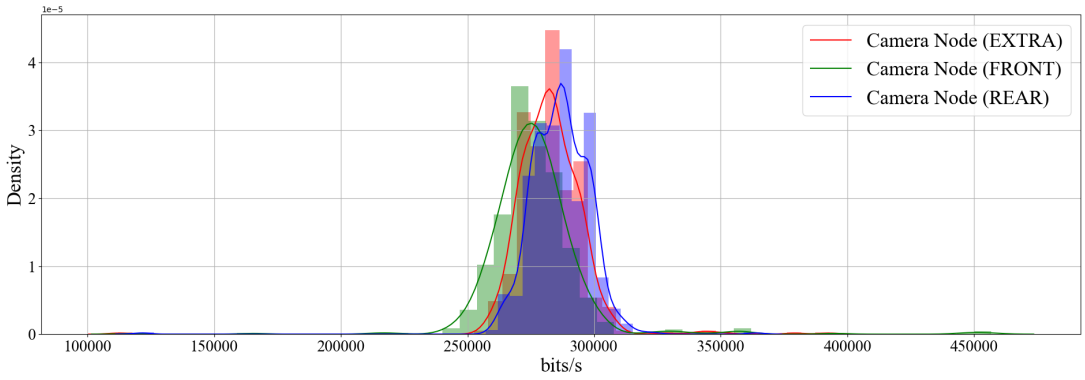


Figure 5.9. Density plot of the throughput of three camera nodes in an intra-vehicle environment over 10 minutes.

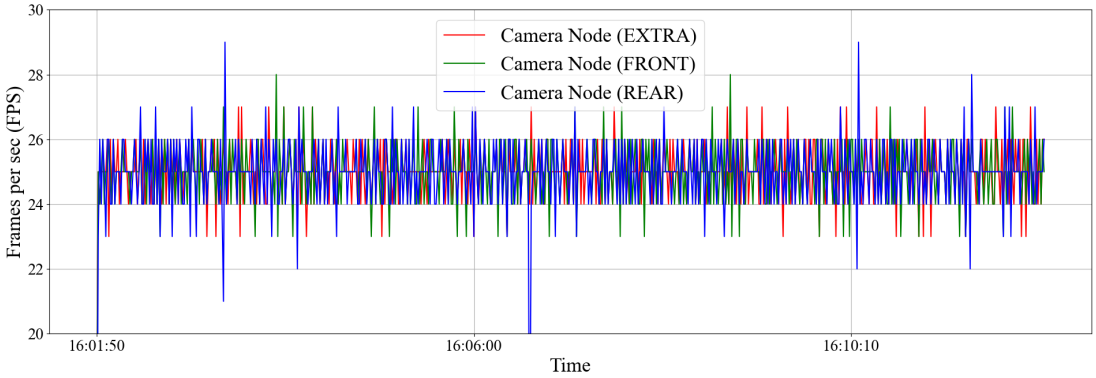


Figure 5.10. Frame rate of all three camera nodes mounted on the vehicle, being fed to the Android device over a 10 minute period.

The frame rate density plot of the camera nodes in the intra-vehicle environment is shown in Fig. 5.11. On average, the frame rate remains in the 25 frames per second region, and the variance of the frame feed shown on the Android device is not visually noticeable.

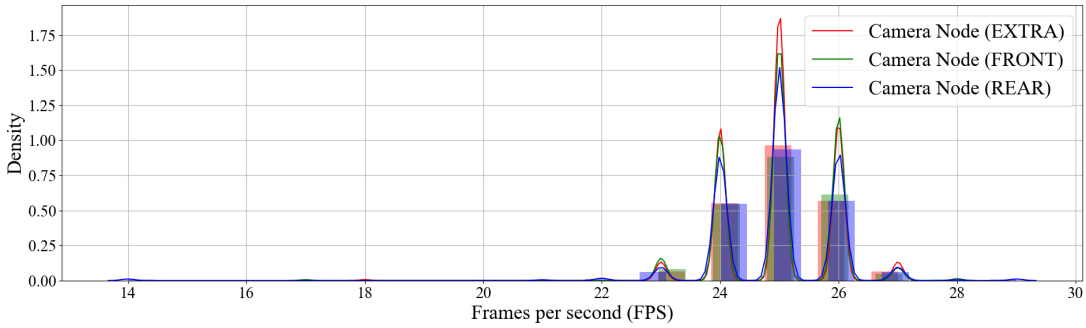


Figure 5.11. Frames per second expressed through a Kernel Density Estimation plot in an intra-vehicle environment.

5.3 NETWORK POWER CONSUMPTION

5.3.1 Camera node power usage

Focusing on the general commuter, the average South African vehicle driver daily spends approximately 90 minutes travelling to work and back [86]. The sensor draws 180 mA at 5 V when fully operational. The camera sensor will require 10.5 hours of an operational power supply when considering a whole work-week, including the weekend. With a power consumption calculated as

$$P_{sensor} = (180mA)(5V) \tag{5.1}$$

$$P_{sensor} = 0.9W, \tag{5.2}$$

the battery size in Watt-hours is calculated as

$$B_{Li-ion} = \frac{100 \cdot I \cdot t}{100 - Q}, \tag{5.3}$$

where I represents the load current, and t is the duration in hours. Q is the required remaining charge where a lithium-ion battery should not go below 20% for prolonged battery life. This results in

$$B_{Li-ion} = \frac{(100)(100mA)(10.5hours)}{100 - 20} \approx 2500mAh. \tag{5.4}$$

The same size of batteries being used in mobile phones would be a good fit if used as a power supply for the camera nodes.

5.3.2 Android battery consumption

Assuming that the driver has not connected the phone to a charger to determine the power drain that the network has on the mobile phone, the current will be measured over some time. The net current being discharged from the battery is measured by using the Android operating system's available API to access the battery metrics. All the nodes are connected to the network and are left to stream for a few minutes. The Wi-Fi Direct network is then disconnected, and readings are continued for a few minutes. The results are shown in Fig. 5.12.

Halfway through the plot of Fig. 5.12 a significant drop in power usage is detected when the network has been turned off. The network increases to an average current usage of 0.15 Ampere when the network is running. The power draw is less than that of the power consumption of streaming HD video over a Wi-Fi network on low-end and high-end smartphones [87].

Multiple streams, as well as different resolutions, have been tested to record current draws at different combinations. Fig 5.13 shows the matrix of current draw averages at different stream to resolution

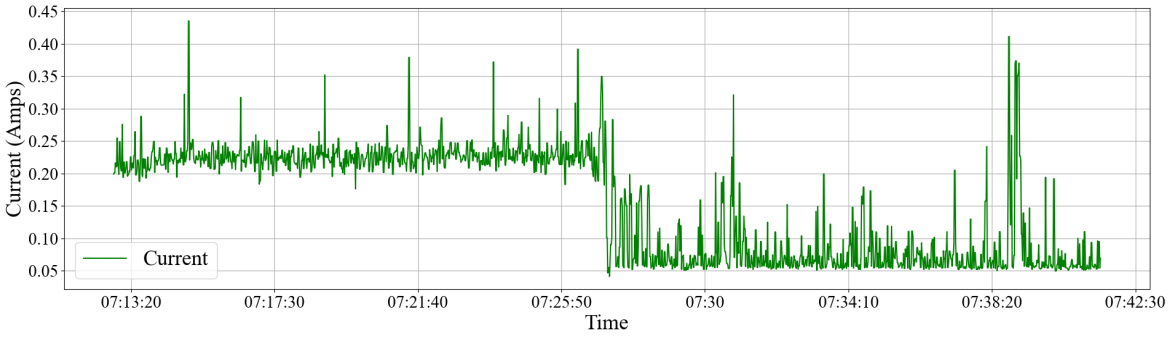


Figure 5.12. Android battery current draw while network is running, followed by turning the network off.

combinations. As expected, as streams and resolutions are increasing, the current draw increases, due to more processing on the smartphone.

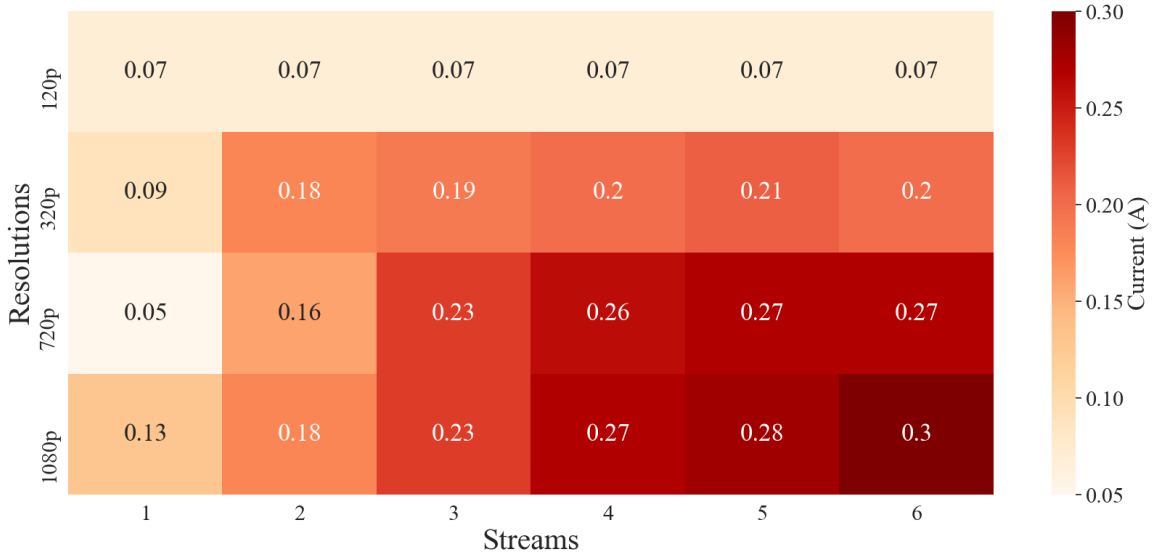


Figure 5.13. A heat map of average current draws on the Android device while running a number of streams.

5.4 LANE DETECTION

As discussed, lane detection, using traditional image processing techniques, do not perform well in urban areas and require more powerful machine learning models. Using a dataset, such as TuSimple, the lane detection method was tested for clearly indicated lane highways. Public roads without heavy traffic were considered. Fig. 5.14 shows an example of the lane detection method being carried out on one image next to the ground truth from the dataset.

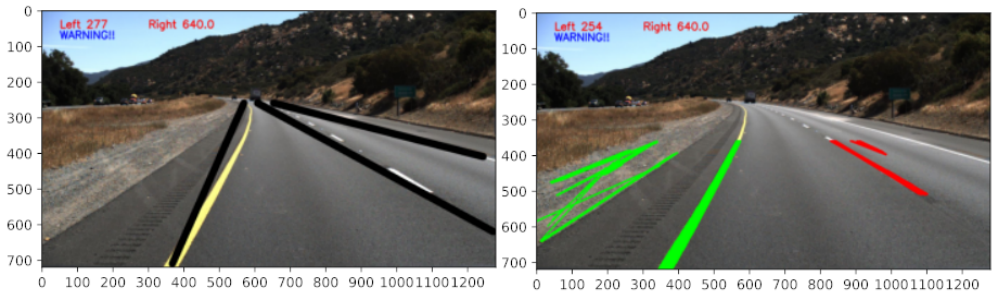


Figure 5.14. A labelled ground truth image on the left and the lane detection method shown on the right image.

Forming part of the ADAS system, the driver is warned when potentially entering a lane on the left or right. If the warning tolerance is too high, then every lane being detected will fire a warning to the driver. Too high a tolerance is not ideal because only lanes close to the vehicle, which are assumed as drifting of the vehicle, should be used to warn the driver. On the other hand, if the tolerance is too low, the detected lanes will never enter the tolerance area that should trigger a warning. The tolerance is set at an estimate of the width of the ADAS-equipped vehicle. Fig. 5.15 shows the area where warnings to the driver will be fired if the lane's x-intercepts would fall within the hatched areas.

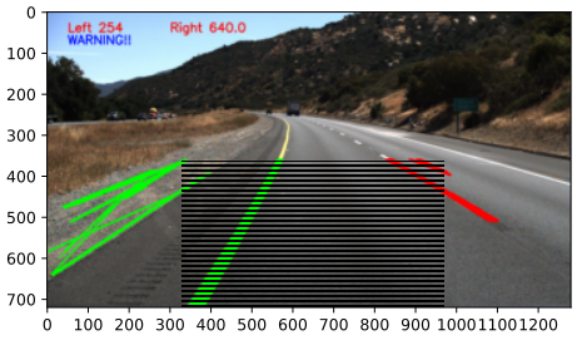


Figure 5.15. An example of a left lane warning where the hatched area represents the tolerance of the position where lanes can be detected.

Accuracy is the number of correctly classified lanes, divided by the total number of examples in the test set. Accuracy is helpful but does not cater for subtleties of imbalances or weighting of false negatives and false positives. The F-score helps to solve this by placing more emphasis on recall or precision. By setting $\beta = 2$, recall is twice as important as precision, considering that it is much worse to miss a lane than to give a false alert for a non-existing lane.

$$F_{score} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP+\beta^2 FN+FP} = 0.94 \tag{5.5}$$

$$Accuracy = \frac{TP+TN}{samples} = 96\% \tag{5.6}$$

Each image can have either a left side or right side warning. Each assessment frame is treated individually for the left and the right, resulting in two individual outcome inferences. Fig. 5.16 shows the different outcome combinations when using TuSimple's test dataset to form the confusion matrix.

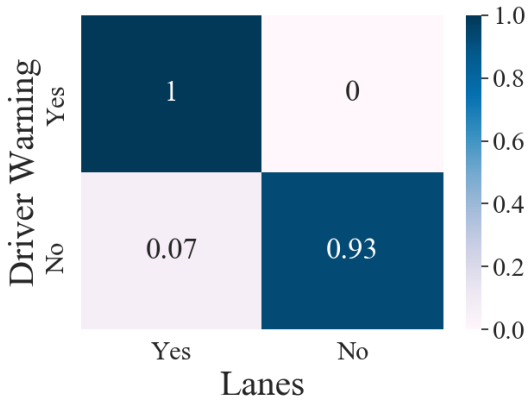


Figure 5.16. Confusion matrix of the lane warning, using the TuSimple dataset.

Fig. 5.17 shows a screenshot of the implemented lane detection on Android. The vehicle's top view is added to the bottom of the video feed view to indicate hazardous signals around the vehicle.

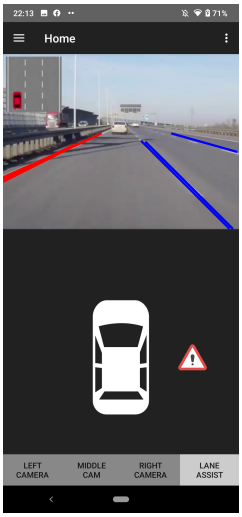


Figure 5.17. A screenshot of the running lane detection addition to the ADAS system on Android with lane assistance to warn the driver that the vehicle is exiting the lane.

The screenshot also shows the lanes' detected overlay on the real-time image stream. The left lanes are coloured in red, and the right lanes are blue. As the vehicle is moving to the right lane, the danger-zone threshold is being entered, and a warning signal is shown to the driver. The hazard warning discontinues once the driver has rectified the vehicle's position back in-between the lanes.

5.5 COLLISION AVOIDANCE

5.5.1 Region-based detection

Collision detection uses a MobileSSDv1 model, pre-trained on the COCO dataset to detect vehicles from the video network stream. The detector uses three different areas to inform the driver about a potential collision. The performance of the detector has been tested by implementing the Bosch Boxy data set, which consists of annotated vehicles for training and evaluating object detection methods for self-driving cars on freeways [88]. The validation set is used as ground truth to compare the collision detector's performance. Fig. 5.18 shows an annotated frame of the Bosch Boxy data set next to this study's detector, both including region detection boxes. Fig. 5.18 shows vehicles being detected in all three regions, namely far (blue), intermediate (green) and near (red).

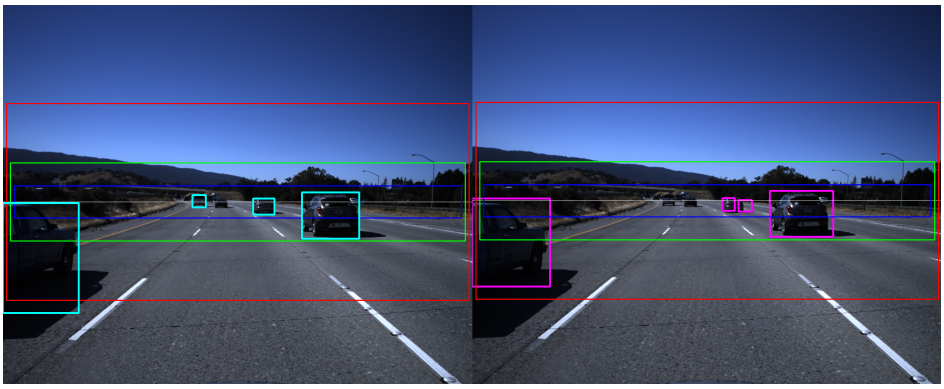


Figure 5.18. A bounding box annotated frame from Bosch Boxy data set next to collision detector frame (right).

By comparing the annotated frames with the detection carried out by the collision detector, a confusion matrix, as shown in Fig. 5.19, has been constructed. The matrix is divided into three regions to illustrate different binary outcomes. Fig. 5.20 shows the accumulative matrix.

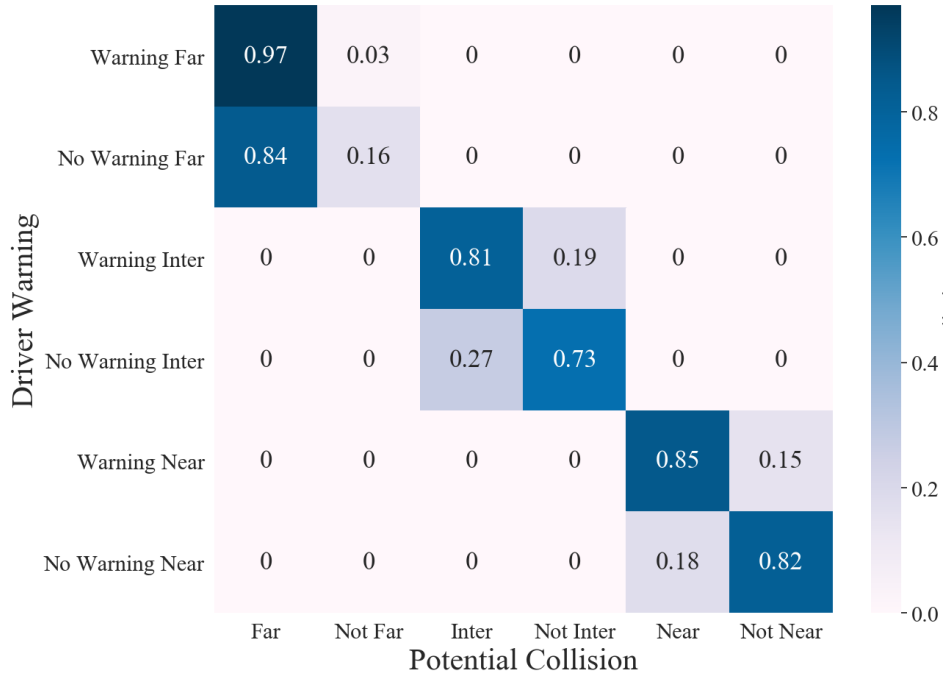


Figure 5.19. Confusion matrix of three detection regions and their binary outcomes.

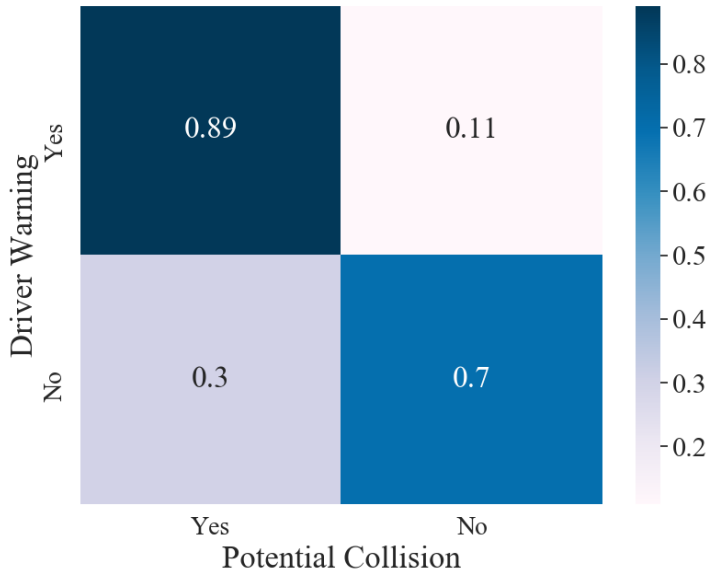


Figure 5.20. Confusion matrix of the accumulative potential collisions for the collision detector.

Due to the size of the Bosch dataset, the validation dataset has been batched into segments where the F-scores are calculated for each, using Equation 5.5, as shown in Table 5.3. Improvements can be made by adding average times that vehicles are in detection regions that would require vehicle tracking.

Using improved object detection models would also increase accuracy by detecting correct objects in regions for frames where no detection or incorrect detection has been made.

Table 5.3. A summary of the batched Bosch Boxy validation set F-scores.

Batch	F-score
1	0.6895
2	0.8103
3	0.8596
4	0.9149
5	0.8633
6	0.8622
Average	0.8333

The first batch has a lower F-score, and upon inspection of the frames, it is found to be due to the section of the dataset that has had poor lighting, with vehicles far in the distance being annotated in the validation set but were not easily detected by the object detector being used.

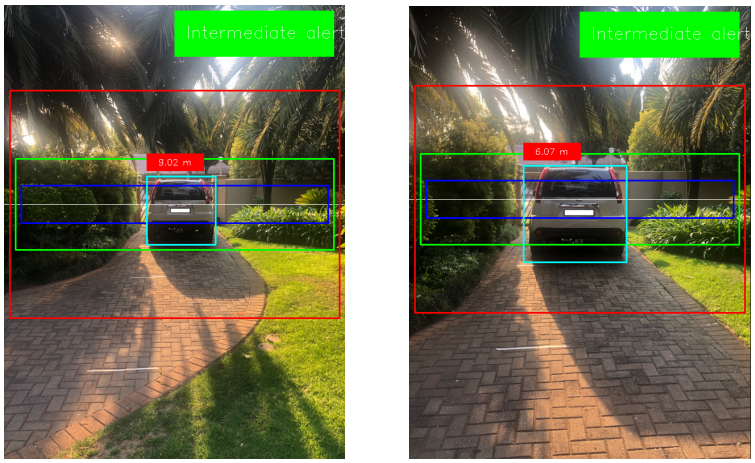
5.5.2 Distance detection

A test vehicle is used to carry out the distance estimation algorithm. Three different intervals are marked on the road at 3 metre intervals. The focal length has been calibrated at a measured 9 metres, and then the distance is estimated at ground truth measurements of 3, 6 and 9 metres, respectively. Table 5.4 shows the different intervals, and the accuracy of the readings after distance estimation has been carried out. Distance errors aggravate as the distance from the original 9 metre calibration point is changing.

Table 5.4. Results for a test vehicle with a width of 155 cm.

Test interval	Actual distance (cm)	Calculated distance (cm)	Error (cm)	Accuracy %
3 metre mark	300	362	62	83%
6 metre mark	600	604	4	99%
9 metre mark	900	902	2	100%

Fig. 5.21 shows the test setup with markers on the floor taken at different intervals at the height of 1.5 metres.



(a) Distance estimated at measured 9 metres.

(b) Distance estimated at measured 6 metres.

Figure 5.21. A screenshot of the Python application, carrying out the distant estimation at different distances.

(a) Near vehicle detected.

(b) Intermediate vehicle detected.

(c) Far vehicle detected.

Figure 5.22. A screenshot of the Android application, running the collision avoidance module.

Fig. 5.22 shows a screenshot of the implemented collision avoidance module on an Android smartphone. After the user has selected the collision avoidance option, the detector starts capturing the video feed from the front camera feed and processes the detection on each frame. The figure also shows the different collision risk areas coloured as red, green and blue as each new frame detects a vehicle. The pinhole method distance estimation is labelled in the top left corner of the bounding box.

5.6 BLIND SPOT DETECTION

A distance experiment has been set up where intervals separated by a metre, starting from the vehicle, are tested to test the accuracy and performance of the developed blind spot nodes. Raw Bluetooth distance sample outputs are taken at every metre interval, using the blind spot node and a Bluetooth Android terminal application. Fig. 5.23 shows the distribution plots of two intervals at 1, 2 and 2.5 metres, respectively. The samples decrease at distances larger than 2.5 metres, due to the ultrasonic sensors diminishing echo signal returns. The accuracy also decreases as the distance is increasing. The distance accuracy degradation has been expected, as the sensor's specifications only support 2 metres.

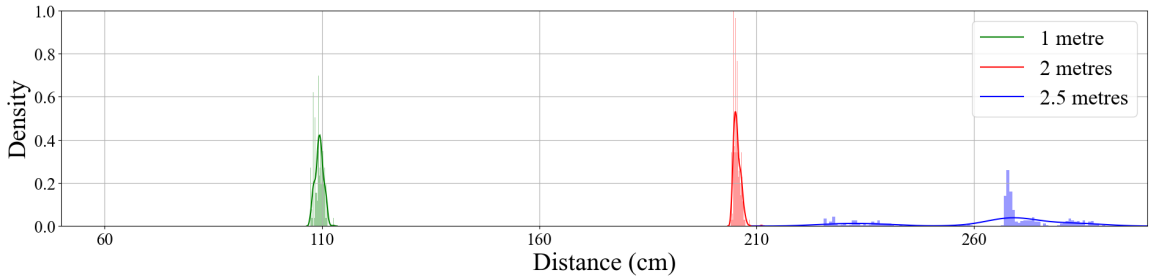


Figure 5.23. Distribution plot for samples measured at two different distance intervals to the car.

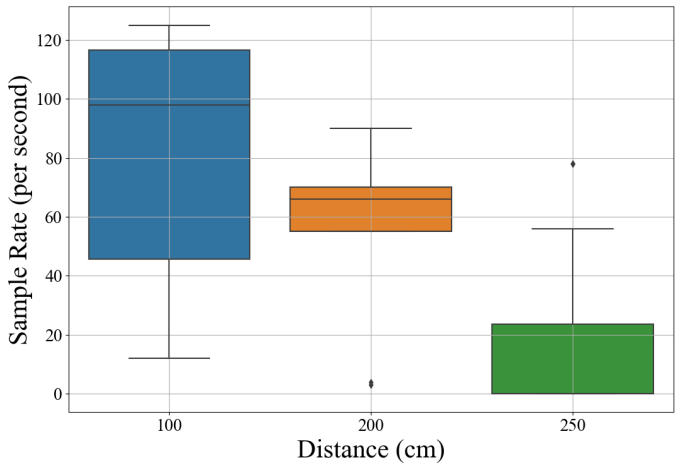


Figure 5.24. Box and whisker plot of the sample rates of the data received by the Android device at different distances measured.

The distance measured by the blind spot nodes is sent to the Android application. Different-coloured tiers indicate distance and danger severity by visually warning the driver about potential danger in the blind spot area. Fig. 5.25 shows an example of positive detection of a vehicle on the right side of the

driver's blind spot. The added feature forms an addition to the collision avoidance system that has been previously implemented, and works together concurrently.

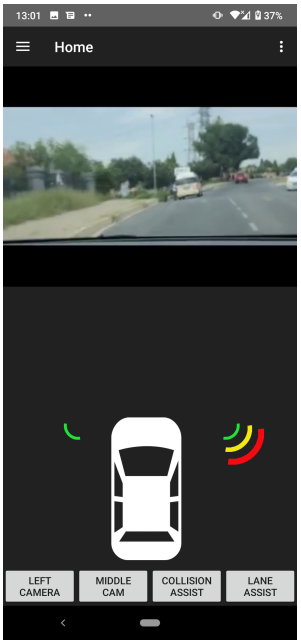


Figure 5.25. Screenshot of the Android application running, the blind spot module. The right side blind spot shows detection of a vehicle to the right of the driver.

5.7 CONCLUSION

The collision and lane detection processing speeds are measured for each frame before and after detection. The sample rate of the blind spot detector is calculated by the number of packets received per second. Fig. 5.26 shows the density plot of lane, collision and blind spot inference times for the ADAS system, being run on a middle-range Android smartphone.

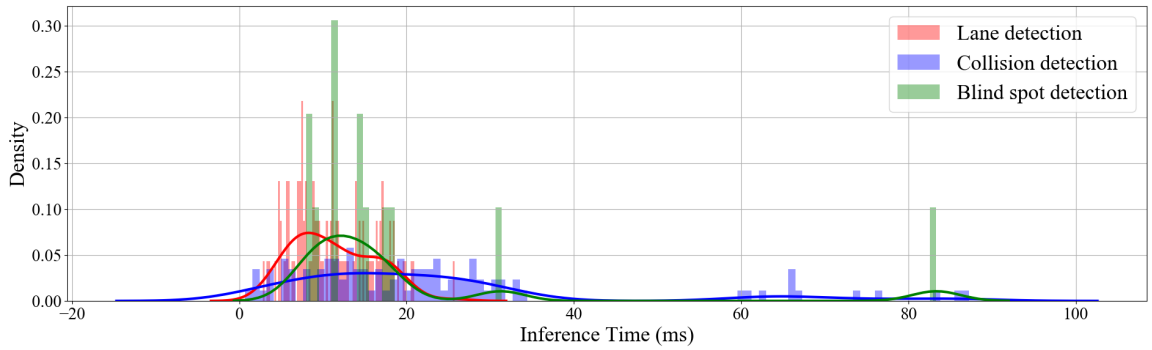


Figure 5.26. Density distribution plot, showing the processing times for collision, lane and blind spot detection.

Collision detection has the most significant average delay; Fig. 5.27 shows the density distribution plot of the frames per second, processed on a middle-range Android smartphone, running the collision avoidance module. The network delay and the collision detection delay have caused the drop in frames but are still usable in a real-time application.

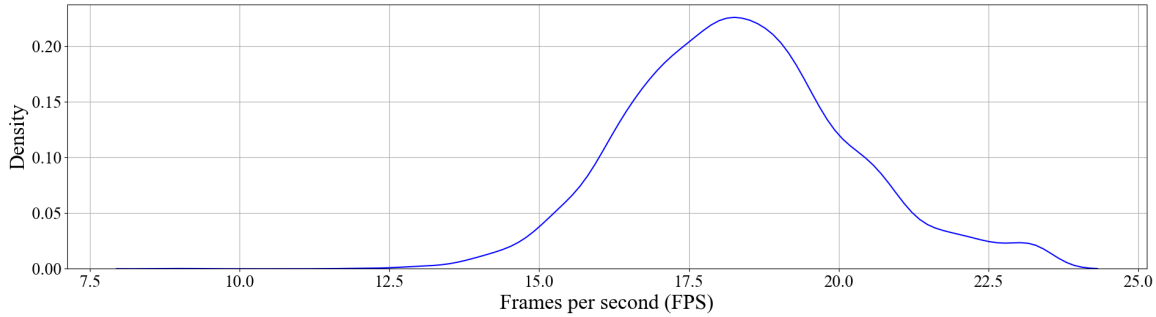


Figure 5.27. Density distribution plot, showing the frame per second performance of the collision detector through the network, utilising it in real-time applications.

A summary of the proposed system's processing times is shown in Table 5.5, which includes their corresponding frame rates. The network delay is the most significant contributing factor to frames being dropped.

Table 5.5. Processing times for proposed system.

Delay type	FPS	Average (ms)	Standard deviation (ms)
Collision detection	44.1	22.7	20.1
Lane detection	82.6	12.1	4.8
Blind spot detection	65.7	15.2	25.4
Camera node (including network)	25.6	39	1.1

CHAPTER 6 CONCLUSION

ADAS systems are prevalent in high-end vehicles, and low-cost ADAS systems have not been widely available. Most drivers have smartphones that are capable of carrying out complex processing, and can perform network communications. It has been shown that smartphones are helpful in ADAS systems, but sensors are limited to the peripherals on the smartphone device. It has been found that the smartphone's sensors can be extended by using the smartphone in conjunction with an IVWSN.

It has been demonstrated that camera nodes in an intra-vehicle network can be robust object detector sensors, using low-cost CMOS camera sensors and a smartphone, capable of carrying out image processing techniques. Additionally, low-cost proximity sensors can form part of the smartphone network and are included in the multiple communication protocol network to execute lateral vehicle detection, such as blind spot detection. Wi-Fi communication can transfer the necessary video streams from the camera nodes to the smartphone, where the smartphone can prioritise the processing of multiple videos feeds accordingly. It has been shown that by using the Wi-Fi protocol of a smartphone, higher transfer rates can send sensory data from a video sensor to enable advanced object detection around a vehicle. Additional routing hardware to host a Wi-Fi network is prevented by using Wi-Fi Direct on the smartphone to host a network over IP protocol.

High data transmission needed for camera-based sensors in WMSNs has not been used in an ADAS environment, and the performance and speed of the system are analysed. Additionally, using multiple communication protocols available to the smartphone, Bluetooth is used for lower data-intense sensors, such as proximity sensors. It has been shown that a low-cost ADAS system, being implemented on a smartphone can then consume the sensory data received from an IVWSN and carry out lane, collision and blind spot detection in real-time.

6.1 CONTRIBUTIONS

It has been found that an inexpensive advanced driver-assistance system alternative can be conceptualised by using object detection techniques being processed on a smartphone from multiple streams, sourced from an intra-vehicle wireless sensor network, composed of camera sensors. To allow the smartphone application to be used by a driver in real-time, the frame rate is required to be high enough to accommodate the user to react to an ADAS warning. Table 5.5 shows a summary of the average processing times of the proposed system, which uses the IVWSN network.

6.2 FUTURE WORK

The Wi-Fi network can reach high throughput rates, but real-time processing is a trade-off that can be improved by using encoding and compression techniques for transferring video streams at higher resolutions. Even though the detectors prefer lower resolution images, other WSN applications might benefit from transferring video streams at higher resolutions. More efficient management of video streams can be implemented to alleviate processing strain and power usage by the smartphone. A switching algorithm can accommodate multiple camera streams by focusing on detection areas of higher priority, depending on driving scenarios determined by the object detection. As mentioned, UWB is uncommon in smartphones and restricted in certain countries [89]. Nonetheless, the latest iPhones are manufactured with UWB technology being included, and other smartphone manufacturers will likely follow suit. UWB implementation of a smartphone ADAS system, using a UWB powered IVWSN, would be a more power-efficient solution but will be accompanied by challenges.

The ADAS techniques used in this paper can be optimised to improve the feedback response time for drivers. The traditional lane detection technique used can be replaced by a CNN to allow detection in urban areas and allow for lower detection errors when lanes are visually obstructed or faded. Different CNN models can also be used to improved the collision detection where ADAS specific models can be trained to optimise inference.

REFERENCES

[1] M. Akhlaq, T. R. Sheltami, and Helgeson, "Designing an integrated driver assistance system using image sensors," *Journal of Intelligent Manufacturing*, vol. 23, no. 6, pp. 2109–2132, June 2012.

[2] A. M. López, A. Imiya, T. Pajdla, and J. M. Álvarez, "Computer vision in vehicle technology: land, sea, and air," in *Computer vision in vehicle technology: Land, sea, and air*, 1st ed. United Kingdom: John Wiley & Sons, 2017, pp. 100–117.

[3] F. Liu, J. Sparbert, and C. Stiller, "IMMPDA vehicle tracking system using asynchronous sensor fusion of radar and vision," in *2008 IEEE Intelligent Vehicles Symposium*. IEEE, 2008, pp. 168–173.

[4] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford, "LiDAR and camera detection fusion in a real-time industrial multi-sensor collision avoidance system," *Electronics*, vol. 7, no. 6, p. 84, 2018.

[5] A. Sole, O. Mano, G. P. Stein, H. Kumon, Y. Tamatsu, and A. Shashua, "Solid or not solid: Vision for radar target validation," in *IEEE Intelligent Vehicles Symposium, 2004*. IEEE, 2004, pp. 819–824.

[6] C. Coue, T. Fraichard, P. Bessiere, and E. Mazer, "Multi-sensor data fusion using Bayesian programming: An automotive application," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2, 2002, pp. 442–447 vol.2.

[7] R. Bhoraskar, N. Vankadhara, B. Raman, and P. Kulkarni, "Wolverine: Traffic and road condition estimation using smartphone sensors," in *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*. IEEE, 2012, pp. 1–6.

[8] M. Fazeen, B. Gozick, R. Dantu, and M. Bhukhiya, "Safe driving using mobile phones," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1462–1468, March 2012.

[9] P. Chaovalit, C. Saiprasert, and T. Pholprasit, "A method for driving event detection using SAX on smartphone sensors," in *2013 13th International Conference on ITS Telecommunications (ITST)*. IEEE, 2013, pp. 450–455.

[10] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: using mobile smartphones for rich monitoring of road and traffic conditions," in *Proceedings of the 6th ACM Conference on Embedded Network sensor Systems*. ACM, 2008, pp. 357–358.

[11] F. Orhan and P. E. Eren, "Road hazard detection and sharing with multimodal sensor analysis on smartphones," in *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2013, pp. 56–61.

[12] W. Devapriya, C. N. K. Babu, and T. Srihari, "Advance driver assistance system (ADAS)-speed bump detection," in *2015 IEEE international conference on computational intelligence and computing research (ICCIC)*. IEEE, 2015, pp. 1–6.

[13] W. Devapriya and C. N. K. Babu, "Real time speed bump detection using Gaussian filtering and connected component approach," in *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*. IEEE, 2016, pp. 1–5.

[14] E. Romera, L. M. Bergasa, and R. Arroyo, "A Real-Time Multi-scale Vehicle Detection and Tracking Approach for Smartphones," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1298–1303.

[15] A. Allamehzadeh, J. U. de la Parra, A. Hussein, F. Garcia, and C. Olaverri-Monreal, "Cost-efficient driver state and road conditions monitoring system for conditional automation," in *2017*

IEEE intelligent vehicles symposium (IV). IEEE, 2017, pp. 1497–1502.

[16] C.-W. You, N. D. Lane, F. Chen, R. Wang, Z. Chen, T. J. Bao, M. Montes-de Oca, Y. Cheng, M. Lin, and L. Torresani, "Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2013, pp. 13–26.

[17] A. Petrovai, R. Danescu, and S. Nedevschi, "A stereovision based approach for detecting and tracking lane and forward obstacles on mobile devices," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 634–641.

[18] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition.*, vol. 1. IEEE, 2001, pp. I–I.

[19] R. Bhandari, A. U. Nambi, V. N. Padmanabhan, and B. Raman, "DeepLane: camera-assisted GPS for driving lane detection," in *Proceedings of the 5th Conference on Systems for Built Environments*, 2018, pp. 73–82.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[21] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[25] H. Aly, A. Basalamah, and M. Youssef, "Lanequest: An accurate and energy-efficient lane detection system," in *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2015, pp. 163–171.

[26] M. Lan, M. Rofouei, S. Soatto, and M. Sarrafzadeh, "SmartLDWS: A robust and scalable lane departure warning system for the smartphones," in *2009 12th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2009, pp. 1–6.

[27] S. J. W. Tang, K. Y. Ng, B. H. Khoo, and J. Parkkinen, "Real-time lane detection and rear-end collision warning system on a mobile computing platform," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 563–568.

[28] L. M. Bergasa, D. Almería, J. Almazán, J. J. Yebes, and R. Arroyo, "Drivesafe: An app for alerting inattentive drivers and scoring driving behaviors," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 240–245.

[29] A. S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller, "Finding multiple lanes in urban road networks with vision and LiDAR," *Autonomous Robots*, vol. 26, no. 2, pp. 103–122, February 2009.

[30] P. Chanawangsa and C. W. Chen, "A new smartphone lane detection system: realizing true potential of multi-core mobile devices," in *Proceedings of the 4th Workshop on Mobile Video*, 2012, pp. 19–24.

[31] W. Liu, S. Li, J. Lv, B. Yu, T. Zhou, H. Yuan, and H. Zhao, "Real-time traffic light recognition based on smartphone platforms," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 5, pp. 1118–1131, May 2016.

[32] C. Pritt, "Road sign detection on a smartphone for traffic safety," in *2014 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 2014, pp. 1–6.

[33] P.-C. Shih, C.-Y. Tsai, and C.-F. Hsu, "An efficient automatic traffic sign detection and recognition method for smartphones," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2017, pp. 1–5.

[34] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: Smartphone-based crowd sensing for traffic regulator detection and identification," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 4, pp. 1–27, April 2015.

[35] J.-R. Lin, T. Talty, and O. K. Tonguz, "A blind zone alert system based on intra-vehicular wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 476–484, 2015.

[36] M. A. Rahman, "Design of wireless sensor network for intra-vehicular communications," in *International Conference on Wired/wireless Internet Communications*. Springer, 2014, pp. 29–40.

[37] B.-F. Wu, H.-Y. Huang, C.-J. Chen, Y.-H. Chen, C.-W. Chang, and Y.-L. Chen, "A vision-based blind spot warning system for daytime and nighttime driver assistance," *Computers & Electrical Engineering*, vol. 39, no. 3, pp. 846–862, 2013.

[38] B.-F. Lin, Y.-M. Chan, L.-C. Fu, P.-Y. Hsiao, L.-A. Chuang, S.-S. Huang, and M.-F. Lo, "Integrating appearance and edge features for sedan vehicle detection in the blind-spot area," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 737–747, 2012.

[39] C. Chen and Y. Chen, "Real-time approaching vehicle detection in blind-spot area," in *2009 12th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2009, pp. 1–6.

[40] S. Singh, R. Meng, S. Nelakuditi, Y. Tong, and S. Wang, "SideEye: Mobile assistant for blind spot monitoring," in *2014 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2014, pp. 408–412.

[41] Y. Dong, Z. Hu, K. Uchimura, and N. Murayama, "Driver inattention monitoring system for intelligent vehicles: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 596–614, February 2010.

[42] Y. Qiao, K. Zeng, L. Xu, and X. Yin, "A smartphone-based driver fatigue detection using fusion of multiple real-time facial features," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016, pp. 230–235.

[43] J. He, S. Roberson, B. Fields, J. Peng, S. Cielocha, and J. Coltea, "Fatigue detection using smartphones," *Journal of Ergonomics*, vol. 3, no. 3, pp. 1–7, 2013.

[44] B.-G. Lee and W.-Y. Chung, "A smartphone-based driver safety monitoring system using data fusion," *Sensors*, vol. 12, no. 12, pp. 17 536–17 552, 2012.

[45] A. Smirnov, A. Kashevnik, I. Lashkov, O. Baraniuc, and V. Parfenov, "Smartphone-based identification of dangerous driving situations: algorithms and implementation," in *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*. IEEE, 2016, pp. 306–313.

[46] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2014.

[47] R. J. Green, Z. Rihawi, Z. A. Mutalip, M. S. Leeson, and M. D. Higgins, "Networks in automotive systems: The potential for optical wireless integration," in *2012 14th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2012, pp. 1–4.

[48] P. L. Iturri, E. Aguirre, L. Azpilicueta, U. Garate, and F. Falcone, "ZigBee radio channel analysis in a complex vehicular environment [wireless corner]," *IEEE Antennas and Propagation Magazine*, vol. 56, no. 4, pp. 232–245, 2014.

[49] M. A. Rahman, J. Ali, M. N. Kabir, and S. Azad, "A performance investigation on IoT enabled intra-vehicular wireless sensor networks," *International Journal of Automotive and Mechanical*

Engineering, vol. 14, pp. 3970–3984, 2017.

[50] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer networks*, vol. 51, no. 4, pp. 921–960, 2007.

[51] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi," in *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2007, pp. 46–51.

[52] T. Kato, Y. Ninomiya, and I. Masaki, "An obstacle detection method by fusion of radar and motion stereo," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, no. 3, pp. 182–188, 2002.

[53] F. Zhang, D. Clarke, and A. Knoll, "Vehicle detection based on LiDAR and camera fusion," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014, pp. 1620–1625.

[54] K. Chitnis, R. Staszewski, and G. Agarwal, "TI vision SDK, optimized vision libraries for ADAS systems," *White Paper: SPRY260, Texas Instrument Inc*, 2014.

[55] J. Khan, "Using ADAS sensors in implementation of novel automotive features for increased safety and guidance," in *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 2016, pp. 753–758.

[56] R. Mahapatra, K. V. Kumar, G. Khurana, and R. Mahajan, "Ultra sonic sensor based blind spot accident prevention system," in *2008 International Conference on Advanced Computer Theory and Engineering*. IEEE, 2008, pp. 992–995.

[57] A. Haselhoff and A. Kummert, "A vehicle detection system based on haar and triangle features," in *2009 IEEE Intelligent Vehicles Symposium*. IEEE, 2009, pp. 261–266.

[58] R. E. Schapire, "Explaining AdaBoost," in *Empirical Inference*. Springer, 2013, pp. 37–52.

[59] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[60] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 464–472.

[61] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.

[62] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[63] N. L. Dickerson, "Refining bounding-box regression for object localization," Ph.D. dissertation, Portland State University, 2017.

[64] R. K. Megalingam, V. Shriram, B. Likhith, G. Rajesh, and S. Ghanta, "Monocular distance estimation using pinhole camera approximation to avoid vehicle crash and back-over accidents," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE, 2016, pp. 1–5.

[65] X. Li, Q. Wu, Y. Kou, L. Hou, and H. Yang, "Lane detection based on spiking neural network and hough transform," in *2015 8th International Congress on Image and Signal Processing (CISP)*. IEEE, 2015, pp. 626–630.

[66] G. A. P. Coronado, M. R. Muñoz, J. M. Armingol, A. de la Escalera, J. J. Muñoz, W. van Bijsterveld, and J. A. Bolaño, "Detection and classification of road signs for automatic inventory systems using computer vision," *Integrated Computer-aided Engineering*, vol. 19, no. 3, pp. 285–298, 2012.

[67] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 20–37, 2006.

[68] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial CNN for traffic scene understanding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[69] G. Rudolph and U. Voelzke. "Three Sensor Types Drive Autonomous Vehicles." Fierce Electronics - Components. https://www.fierceelectronics.com/components/three-sensor-types-drive-autonomous-vehicles (accessed Oct. 26, 2021) .

[70] D. Catania and S. Zammit, "Video streaming over Bluetooth." University of Malta. Faculty of ICT, 2008.

[71] W. Xiaohang *et al.*, "Video streaming over Bluetooth: A survey," *Institute for Infocomm Research, School of Computing, NUS*, vol. 6, p. 52, 2011.

[72] T. N. Duong, N.-T. Dinh, and Y. Kim, "Content sharing using P2PSIP protocol in Wi-Fi Direct networks," in *2012 Fourth International Conference on Communications and Electronics (ICCE)*. IEEE, 2012, pp. 114–118.

[73] Wi-Fi Alliance, "Wi-Fi certified Wi-Fi direct," *White paper*, p. 7, 2010.

[74] Y. W. Sitorus, G. Frans, P. Prasetiyo, T. Adiono, A. H. Salman, and N. Ahmadi, "Establishment of Wi-Fi display session between source and sink device in wireless Android screencasting," in *2015 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. IEEE, 2015, pp. 663–668.

[75] V. Appia, H. Hariyani, S. Sivasankaran, S. Liu, K. Chitnis, M. Mueller, U. Batur, and G. Agarwa, "Surround view camera system for ADAS on TI's TDAx SoCs," *White paper*, pp. 2–8, 2015. Accessed: Oct. 26, 2021. [Online]. Available: https://www.ti.com/lit/wp/spry270a/spry270a.pdf

[76] M. Mody, P. Swami, K. Chitnis, S. Jagannathan, K. Desappan, A. Jain, D. Poddar, Z. Nikolic, P. Viswanath, M. Mathew *et al.*, "High performance front camera ADAS applications on TI's TDA3x platform," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. IEEE, 2015, pp. 456–463.

[77] J. Verhaevert, "Detection of vulnerable road users in blind spots through bluetooth low energy," in *2017 Progress In Electromagnetics Research Symposium-Spring (PIERS)*. IEEE, 2017, pp. 227–231.

[78] Wi-Fi Alliance, "Wi-Fi Peer-to-Peer (P2P) Technical Specification Version 1.7," 2016. Accessed: Oct. 26, 2021. [Online]. Available: https://www.wi-fi.org

[79] "Bluetooth Overview - Key classes and interfaces." Android Developers documentation guides. https://developer.android.com/guide/topics/connectivity/bluetooth (accessed Oct. 26, 2021).

[80] "HTTP - Hypertext Transfer Protocol." W3C architecture domain. https://www.w3.org/Protocols/ (accessed Oct. 26, 2021).

[81] "Ultrasonic Ranging Module HC - SR04," 2013. Accessed: Oct. 26, 2021. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[82] "OpenCV Releases." OpenCV.org. https://opencv.org/releases (accessed Oct. 26, 2021).

[83] "How Long Is A Car? (Average car length according to types)." Average Car Length. https://www.smartmotorist.com/average-car-length (accessed Oct. 26, 2021).

[84] "Object detection | TensorFlow Lite." TensorFlow Lite example applications for Mobile and IoT. https://www.tensorflow.org/lite/models (accessed Oct. 26, 2021).

[85] "Performance benchmarks | TensorFlow Lite." Performance measurement benchmark tools. https://www.tensorflow.org/lite/performance/benchmarks (accessed Oct. 26, 2021).

[86] Stats SA, "National Households Travel Survey 2020," 2020. Accessed: Oct. 26, 2021. [Online]. Available: http://www.statssa.gov.za/publications/P0320/P03202020.pdf

[87] P. Spachos and S. Gregori, "Wi-Fi throughput and power consumption tradeoffs in smartphones," in *2017 24th International Conference on Telecommunications (ICT)*. IEEE, 2017, pp. 1–5.

[88] K. Behrendt, “Boxy vehicle detection in large images,” in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019, pp. 840–846.

[89] “Ultra Wideband availability.” Apple Support. https://support.apple.com/en-za/HT212274 (accessed Oct. 26, 2021).