MAKERERE UNIVERSITY

College Of Engineering, Design, Art, and Technology

Department Of Electrical and Computer Engineering

Design and Implementation of a Blind Spot Detection and Monitoring System for The Kayoola Buses

Case Study: Kiira Motors Corporation

Submitted By

Stephen Tipa Augustine

17/U/1149

Main Supervisor

Ms. Amara Agatha Turyagyenda

Co-Supervisor

Dr. Byamukama Maximus Baguma

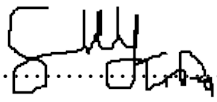
A final project report submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Science in Telecommunications Engineering

Monday, February 21, 2022

Declaration

No portion of the work in this document has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

I have abided by the Makerere University academic integrity policy on this assignment.

Signature: ……  ……...

Date: 23/02/2022……...

Approval

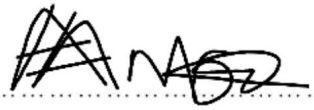
This report has been submitted with the approval of the following supervisors,

Main Supervisor

Ms. Amara Agatha Turyagyenda

Lecturer

Department of Electrical and Computer Engineering, Makerere University

Signature: 

Date: 3/3/22

Co-supervisor

Dr. Byamukama Maximus Baguma

Lecturer

Department of Electrical and Computer Engineering, Makerere University

Signature: 

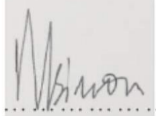
Date: 23/02/2022

Company Supervisor

Mr. Simon Peter Miyingo

Information System Manager, Kiira Motors Corporation

Department of Product Development

Signature: 

Date: 23/02/2022

Dedication

My dear Uncle, Mr. Okuma Augustine,

I dedicate this report to you, time is an independent variable that no human has control over. A few years back in high school, you tailored a way for me to further my studies, but you did not stop there, you still went ahead and inspired me to become a Telecommunication Engineer. I greatly feel honored by you. Though you do not work in this field I know you like science and technology as much as I do, and therefore, this report shall please as much as my excitement in preparing it.

Your affectionate Nephew,

Stephen Tipa Augustine

Secondly, I dedicate this report to my elder brother, Inyani Francis, and mother, Mike kide. Even in this hard period of the COVID 19 pandemic, my studies were your priority, and you were so supportive to me in different capacities. I am pretty sure that when you look at this report you will realize that, the trust you put in me was not in vain.

Thirdly, I will also like to dedicate this report to my fellow coursemates. You have been supportive in different capacities, and more importantly, you made me strive effortlessly.

Preface

Joining the College of Engineering, Design, Art, and Technology has been my dream since childhood, as a result, I have been tailored to work hard in the past academic levels to get me in the Department of Electrical and Computer Engineering.

In 2017 enrolling in Makerere for Telecommunication Engineering many of the people around discouraged me saying that Civil engineering would be the best course for me, but I insisted and followed my quest. After finding out of the many interesting careers a Telecommunication engineer can undertake, I told myself, you had eyes that many never had. My quest for hands-on experience in the Telecommunication field began when I attended the recess term held within the university in 2018, then followed by the Internship Training with Nokia Solutions and Networks in 2019 and finally my second internship training at Kiira Motors Corporation in 2020 where I am up to date implementing this project. I strongly believed that this final year project has consolidated my hands-on experience making me appreciate the principles and theories that I have learned from early school days till to date.

This report has been well-integrated and concise, in a manner intended to be brief but large enough to cover all aspects of the final year project. The text is up to date, covering the broad fields of software application development, Artificial Intelligence, and consolidating the aspect of System Engineering in practice.

This report has been organized perfectly, thus it will touch the heart of everyone who chances on it. However, I deliberately addressed it to my academic supervisors, and the other parties stated in the declaration and approval sections of this report.

The report has been organized in this manner;

Chapter 1, covers the introductory part, explaining what IT means, its objectives, and information about the Company of placement.

Chapter 2, Discusses the literature review, starting with Web application development to Requirement Engineering, each section of this chapter explains the literature of the respective fields tackled during the internship training.

Chapter 3, covers the practical aspect performed by the author, starting with Web application development to Requirement Engineering, each section of this chapter explains the steps undertaken for these practical tasks of the respective fields tackled during the internship training.

Chapter 4, Gives conclusive remarks, challenges encountered during the training, and achievements attained from IT.

After this list of reference have been presented, using the IEEE referencing style.

The documents end with the Appendices in which other important data has been presented.

Acknowledgments

I would like to offer my sincere appreciation to my Academic Supervisors, both the main and the co-supervisors for the advice they gave me during the implementation of this project. Their valuable and constructive suggestions and ideas helped tailor this project towards its objectives. Special thanks still go to my company supervisor the Information Systems Manager, Mr. Simon Peter Miyingo for the guidance he offered me throughout this project.

I would like to acknowledge the help given to me by the employees of Kiira Motors Corporation. They have been all helpful in their different capacities.

Not forgetting the Chief Executive Officer, Mr. Paul Isaac Musasizi for his continuous guidance throughout the entire period of the project. Thanks, to the undergraduate research assistances for the inspiration they instilled in me. I know that the list is long and thus I cannot mention all these important people, but I appreciate all the help offered to me by them.

I am particularly grateful to Mr. Kenneth Kahuma for being informative throughout this period of the project and the Electrical and Computer department for embedding this module as a degree requirement in the Telecommunication Engineering program.

My special thanks are extended to my family members especially, my mother Ms. Mikelina Kide, my brother Mr. Inyani Francis, my uncle Mr. Okuma Augustine and my sponsors (the Government of Uganda) for all the financial support they offered me throughout my studies up to date.

Abstract

Changing lanes or negotiating a turn in a congested area while having no information about the objects in the blind spot area can be dangerous. It is particularly hard for drivers of the largest vehicles to see everything around them but the consequences of missing an obstruction could be catastrophic.

As buses operate on increasingly crowded roads, drivers need to help in eliminating blind spots and highlight potential collisions before they occur. In this project, a Sensor-Based Blindspot Detection System for the Kayoola Buses to detect objects was proposed.

This concept is implemented using a Raspberry Pi and two kinds of sensors; The Raspberry gathers input parameters from the two types of sensors (an ultrasonic sensor for object detection along with how far they are from the bus as well as an accelerometer for detecting motion in the bus). In addition, camera feeds are also fed to the Raspberry to allow for object detection. The microcontroller has predefined thresholds to decide if the measured value is off the range. If so, the driver is first alerted on LCD and LED, then later when the distance between the ego vehicle and the target gets smaller, auditory feedback is initiated as well.

With the above setup, a hybrid blindspot detection system was feasible. In-vehicle tests were however not conducted, all tests were alpha tests performed on the bench. The project findings have shown that the HC-SR04 ultrasonic sensors and the Raspbian cameras are not suitable for deployment. For a complete deployable system, there is a need for identifying superior sensors, cameras, and microcontrollers that can easily be integrated with the CAN communication protocol.

Contents

List of Tables

List of Figures

List of acronyms

2D – 2 Dimensional

3D – 3 Dimensional

ABS – Anti-lock Braking System

Adobe-XD – Adobe-Experience Design

AEB – Autonomous Emergency Braking

AI – Artificial Intelligence

BIWS – Blindspot Information Warning System

BLE – Bluetooth Low Energy

BSD – Blind Spot Detection

BSDS – Blind Spot Detection System

CAN – Controller Area Network

CDF – Cumulative Distribution Function

CGSA-CFAR – Cell Greatest, Smallest, and Averaging Constant False-Alarm Rate

CISC-CISC – Complex Instruction Set Computers

CLI – Command Line Interface

CNN – Convolutional Neural Network

DL – Deep Learning

DSP – Digital Signal Processing

ESC – Electronic Stability Control

FCN – Fully Convolutional Network

FMCW – Frequency Modulated Continuous Wave

GDP – Gross Domestic Product

GPS – Global Positioning System

GUI – Graphical User Interface

HMD – Human Machine Interface

IDE – Integrated Development Environment

IEEE – Institute of Electrical and Electronics Engineers

IoT – Internet of Things

ISA – Intelligent Speed Assistant

LCD – Liquid Crystal Display

LDW – Lane Departure Warning

LED – Light Emitting Diode

LFMCW – Line Frequency Modulated Continuous Wave

Lidar – Light detection and ranging

LKA – Lane Keeping Assistant

LSTMS – Long Short-Term Memory

ML – Machine Learning

NN – Neural Network

NTV – National Television

Radar – Radio detection and ranging

RISC-RISC – Reduced Instruction Set Computers

RSSI – Received Signal Strength Indicator

RTIs – Road Traffic Incidents

SDS – System Design Specification

SLI – Speed Limit Information

SRS – System Requirement Specification

SVM – Support Vector Machine

TI – Texas Instrument

TI-DSP – Texas Instrument Digital Signal Processing

TPMS – Tyre Pressure Monitoring System

UML – Unified Modeling Language

UN – United Nation

VGG – Visual Geometry Group

VLSL – Very Large-Scale Integration

VST – Vehicle Safety Technology

List of Principle Symbols

Seconds (s) is the unit for time

Meters (m) is the unit for distance

Degrees Celsius (º C) is the unit for temperature

Hertz (Hz) is the unit for frequency

1. Introduction

1.1. Background

Uganda has one of the highest rates of Road Traffic Incidents (RTIs) globally. In the last decade alone, recorded road crash fatalities rose from 2,597 to 3,503 in 2016 representing a growth of 25.9%. The accident severity index is 24 people killed per 100 road crashes [1], [2]. On average, Uganda loses 10 people per day in road traffic crashes, which is the highest level in East Africa [3]. The overall annual cost incurred due to road crashes is currently estimated at approximately UGX 4.4 trillion ($1.2 billion), representing 5% of Uganda's gross domestic product (GDP) [4].

In an attempt to curb the rampant RTIs, the Government of Uganda developed a comprehensive road safety road map as one of the ways to achieve a 50% reduction in road traffic accident deaths by 2020, as recommended by the UN resolution on Decade of Action for Road Safety (2011-2020) [4]. It focused on road safety management through establishing infrastructure for the protection of vulnerable road users in urban areas, driver training and testing, enforcement of traffic rules, a road crash database, and a post-crash care response and coordination system. Although these solutions were very good, they have a great limitation: Human error. A police report in the first week of July 2017 stated that out of all the 3000 plus deaths that occurred in 2016 due to accidents, over 80% of them were caused by human error [1].

Blind spots are one of the most common sources of "Human Error." In the United States, over 800,000 blind spot accidents occur each year with approximately 300 fatalities [5]. In Europe, blind spots are among the main contributing factors to road accidents in that European Union Law requires lorries to be fitted with blind-spot mirrors to give drivers a wider field of vision [6]. In Uganda, most of the RTIs are caused by reckless and careless driving that is rooted in a lack of focus and general unawareness of other road users.

To minimize the causative effect of blind spots on RTIs and fatalities, automotive industries have implemented blind spot detection systems. Typical systems employ various sensors and computer vision methods for obstacle detection and driver alerts. An example is a vision-based blind-spot warning system that provides a driver assistance interface for visualizing the cars around them on a 3D platform, powered by neural networks for car detection and depth estimation [7]. This system can only detect cars and possesses a high processing load due to the 3-D visual involved.

Another instance is Bosc-Mobility Solutions which offers a sensor-based blind spot detection system. In one radar is employed, while in another set ultrasonic sensors are used [8]. These sensors monitor the space in the adjacent lane, allowing the system to cover the blind spots. If another vehicle is situated in the monitored area, the driver is alerted to the potential danger using a warning sign in the side mirror. If the driver fails to spot or ignores this warning and activates the turn signal to change lanes, the system can also trigger an audible warning. The system recognizes stationary objects on or alongside the road, such as guardrails or parked

vehicles. This system however does not provide visual views of the precise location of the objects in the blind spot, but rather the blinking of the LED and auditory feedback.

Traffic accidents on roads and highways represent one of the most serious problems worldwide leading to loss of lives and damage of property. Long vehicles like the Kayoola buses that have a length of 12.19m have multiple blind spots and yet have no Blindspot Detection System, while other road users are typically unaware of the extent of these blind spots leading to many accidents occurring when cyclists or pedestrians disappear from the driver's view.

1.2. Problem Statement

Traffic accidents on roads and highways represent one of the most serious problems worldwide leading to loss of lives and damage of property. Firstly, the Kayoola buses have multiple blind spots yet other road users are typically unaware of the extent of these blind spots, leading to many accidents occurring when cyclists or pedestrians disappear from a driver's view [9].

Following the above argument, Kiira Motors Corporation (KMC) has already five buses on the road by the time of writing this report and is planning to deploy a total of 1030 buses in the financial year 2021-2022. To reduce blind spot-related accidents, all these busses will require a blind spot detection and monitoring system.

Secondly, based on our research, the available blind-spot detection systems have varying features like GUI display, LED alerts, adaptive alert level based on driver reluctance, and auditory feedback, which are equally important and yet these are only provided for luxury vehicles at expensive rates.

Thirdly, the existing blind spot detection and monitoring systems developed by automotive companies like Bosch Mobility, Toyota, and others do not pay attention to the traffic conditions in Uganda. Very important attention has to be paid to the behavior of pedestrians, motorcycle riders, bicycle riders, taxi drivers, government vehicles as well as private cars. A system that would effectively take care of the different behavior of road users in Uganda as well as meet the engineering system requirements will have to be having both vision and non-vision-based features. A vision-based system uses cameras and a deep learning model to recognize objects, whereas a non-vision-based system uses radar or ultrasonic sensors to map objects in its surrounding.

All the above arguments called for a customized system for the Kayoola buses that is affordable, locally built and that fits into Uganda's Vision 2040 with industrialization with all the above features incorporated. The Uganda Vision 2040 identifies the Government of Uganda's development paths and strategies to operationalize the country's vision statement of "A Transformed Ugandan Society from a Peasant to a Modern and Prosperous Country within 30 years" [10].

1.3. Proposed solution

The stated problem was addressed using the vision-based and non-vision-based approaches that will be elaborated more on in the literature review (Next chapter). The system used three types of sensors, ultrasonic sensors for distance measurement, an accelerometer for motion detection, and a camera for object identification. The sensor data and camera feeds are sent to the Raspberry Pi (Master Module). Based on the measured parameters the master module decides to alert visually using LEDs and LCD or auditory feedback. The system has an adaptive alert system using visual and auditory feedback. The high-level view of the system is shown in **Figure 1-1**. For simplicity, only one sensor element of each sensor type, as well as camera and LED, were shown, however, this does not imply the system will employ only say one sensor. The clear functional architecture of the system is presented in Chapter three.

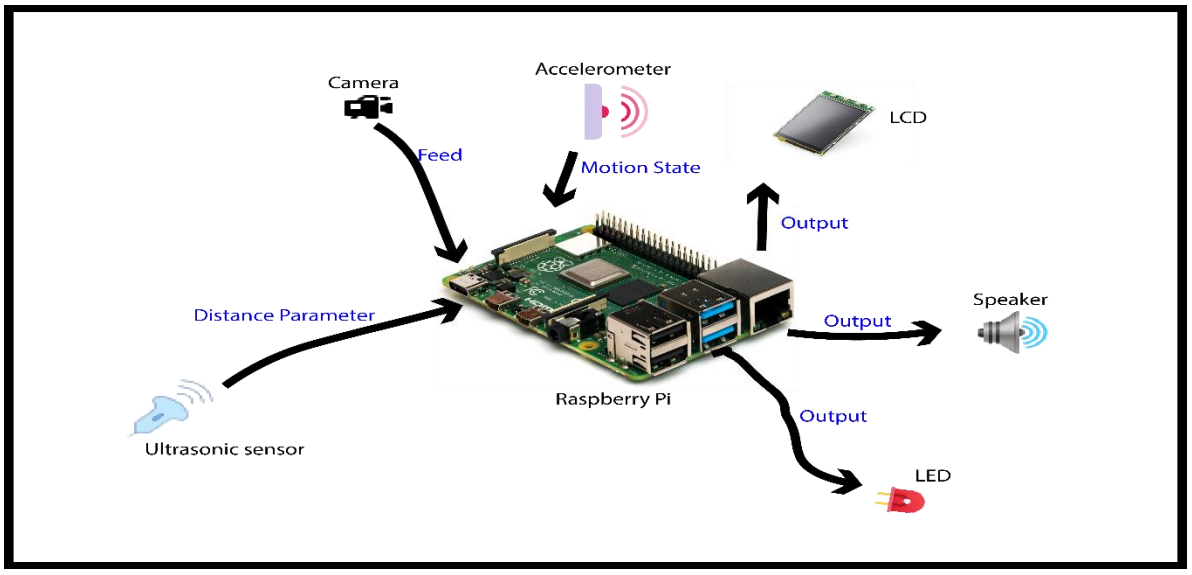


Figure 1-1: System Pictorial View

1.4. Rationale

Firstly, the existing systems are not only expensive but also do not incorporate all of the desired features required in the project in question, these features include

i. Graphical display of the position of objects based on sensor data
ii. Adaptive enough to identify high way and cities as well as traffic congestion.
iii. Ability to detect if the bus is in motion or parked and activate only when the bus is in the motion.
iv. Ability to increase alert level based on driver reluctance to take proper action

Based on the studies performed on the existing systems, none have all those qualities thus, calling for the need to customize a unique system for the Kayoola buses.

Secondly, both vision-based and non-vision-based solutions have their inherent limitations. To therefore achieve the best performance, it is required to harness the strength of both approaches. As will be discussed in the next chapter, very few works have been done in this line, and indeed calls for further studies.

Finally, the systems in the automotive industry are biased towards traffic conditions of other nations. Uganda has unique traffic situations that need great attention when developing a driver assistant system. With the proposed Blindspot Detection System installed in the bus, safety would be significantly improved as the driver is notified of nearby cars, cyclists as well as pedestrians, thereby reducing the chances of collisions.

1.5. Objectives

1.5.1. Main Objectives

The main objective of this project was to develop a system that detects objects in blind spot areas of the Kayoola Buses and alerts the driver of their proximity to prevent road accidents.

1.5.2. Specific Objectives

The key objectives driving this project were as follows,

1. To develop the hardware and software requirements specifications for the Blindspot Detection System
2. To derive logical and physical design models for the System
3. To implement the design specification into a practical prototype

1.6. Value Proposition

The implementation of the proposed system for Kiira Motors Corporation entails the following key benefits;

i. Increase comfort in driving the Kayoola buses since drivers would not have to strain to see objects in the blind spot
ii. Reduces incidents of road accidents related to blind spots for the Kayoola buses
iii. Increases market appeal and value of the Kayoola buses as safety increases

1.7. Project Scope

This project targeted the Kayoola buses, therefore our prototyping, deployment in the future, and data collection are already done or those that will be collected in due course will be in association with the Kayoola buses. And more specifically, since new versions of the Kayoola buses are arising we restricted our study to the Kayoola EVS (The Electric bus).

1.7.1. Object detection model

The object detection being simply a feature of the system, the focus of the project was in identifying an object detection model and fine-tuning it other than developing one from scratch. This model was only limited to the identification of cars, motorcycles, bicycles, and humans. Objects other than those listed above may not be detected by our deep learning model.

1.7.2. Test scope

This project employed a systematic testing paradigm that involve unit testing, integration testing, and system testing. In-vehicle or road testing was not conducted.

2. Literature Review

2.1. Introduction

This chapter is aimed at giving background information on the topic at hand. In this literature review, the author intended to identify existing work in line with the topic in the automotive industry. The literature is also aimed at providing the required justification to ascertain the problem statement discussed in chapter one.

For completeness and to be more concrete firstly, the traffic conditions of Ugandan roads are unique in a way, secondly, the Kayoola buses have a length of about 12.2m with no onboard blind spot warning system yet these buses will soon take up public transport in the country. Thirdly, the existing systems built by Bosch Mobility, Ford, GM, Toyota, etc. are designed for luxurious vehicles at expensive prices. This project, therefore, seeks a way to reduce blind spot-related accidents in Uganda by locally designing a system for this purpose while paying close attention to the road conditions, road user behavior, and product costs.

Of recent, there has been a comprehensive development in the automotive industry. Auto-manufacturers are working hard to improve the designs of their vehicles to achieve competitive advantages over their peers. This on the other hand has placed a strain on these companies as they have to meet the growing and changing customer demands as well as work within a small budget so that, the end products are not overpriced. The main developmental features have been in exploiting sensor technologies. In vehicles, sensors are being exploited in many tasks and the key concern for this project is sensor application in the development of an active safety feature in the vehicle. Not only sensors but there has also been an increase in the adaptation of artificial intelligence (AI) in the automotive industry as well. The main application of AI in automobiles so far is in the famous computer vision (CV) field as well as the concept of autonomous vehicles. The former extract features from images or videos to give meaningful information while the latter applies AI to enable self-driven vehicles.

This chapter has been organized as follows; section 2.2 about related work gives a scholarly review of the work done concerning the aspects of this topic. The review has been broken down into three themes and these are work-related to vision-based systems, non-vision-based systems, and hybrid systems. The insights of these terms are explained in the next section. In each theme, the author identified relevant projects and investigated their approaches to the problem while paying attention to the strength and weaknesses of each approach.

Section 2.3 intuitively elaborated on the current state of vehicle safety features, giving the different safety features that exist in the industry. Section 2.4 discussed sensor technology concerning the automotive industry. It further elaborated on the common sensor types currently being used. Section 2.5 introduced the aspect of graphical user interface design in the context of vehicle information systems. Still in this section, the author discussed the different languages and frameworks used for graphical user interface designs. Section 2.6 discussed AI in the field of the automotive industry, and finally, section 2.7 gives intuition on microcontrollers and microprocessors. Section 2.7 also gives the stand of microcontrollers in the current state of

vehicle design and the programming languages vastly used for the development of embedded systems.

This chapter ends with a conclusion in section 2.8, where the author highlighted a few of the key concerns of the chapter. Further, the key choices for the tools and technologies to be employed in this project are made clear. These choices are backed up by prominent arguments.

2.2. Related work

Over the recent years, many studies focused on the development of two kinds of blind spot detection systems, either vision-based or non-vision-based. For vision-based systems, camera sensors are used alongside computer vision techniques as well as deep learning or ordinary machine learning. On the other hand, in non-vision-based systems, radar, infra-red, Bluetooth, and ultrasound are the main underlying technologies for blind spot detection. In this literature review, the author adopted the thematic approach to address scholarly findings in this subject. The three kinds of blind spot detection systems are dissected in-depth; however, this project adopted the hybrid system. The hybrid system harnesses the benefits of both the vision and non-vision-based systems.

2.2.1. Vision-based solutions

Yiming Zhao, Lin Bai, Lecheng Lyu, and Kinming Huang according to [11] presented a design of a neural network with only a few layers for real-time embedded systems of which one of the applications was blind-spot detection. Usually, better accuracy requires deeper models and better computational costs. However, according to them by using depthwise separable convolution, they were able to dramatically reduce the model parameters and operations. The key focus of their research was the transfer of blind spot detection into an image classification task. Like any engineering task, a gain in a parameter leads to compromise in another. For this case, the tradeoff was between accuracy and cost. According to the researchers in [11], compared with the VGG block, the Sep-Res-SE block, combining depthwise separable convolution, residual learning, and squeeze-and excitation achieved similar detection accuracy with far fewer parameters and operations. As a limitation of this research, only a few road and weather conditions were considered which is practically not sufficient for such systems to be deployed in the real world.

Huei-Yung, Jyun-Min, Lu-Ting, and Li-Qui [12], propose a vision-based driver assistance system for highway, urban, and city environments. Their system consists of three subsystems which are lane change detection, forward collision warning, and overtaking vehicle identification. In the implementation, they used two monocular car digital video recorders to capture the front and rear views of the traffic scenes [12]. The front vehicles were identified by a new CDF-based symmetry detection technique. For overtaking detection, the motion cue obtained from optical flow was combined with convolutional neural networks for vehicle identification with repetitive patterns removal. Their experiments and evaluation carried out on various real traffic scenarios have demonstrated the effectiveness of the proposed techniques [12]. However, on the downside, firstly, they did not adopt the stereo vision for the cameras making depth estimation for the front vehicles more difficult. Secondly, the overtaking technique

did not perform quite well when the vehicle was making turns. Lastly, the proposed solution was not implemented on an embedded platform or evaluated in a real-time scenario.

In the LaRASideCam project [13], Blanc, Nicolas Steux, Bruno Hinz, and Thomas proposed a fast and robust vision-based blind spot detection system. In their approach, they used video data obtained from two cameras mounted near the left and right-side mirrors of the car. This data was then subjected to two algorithms, the one hand edge detection and support vector machine (SVM) and on the other hand template matching [13]. The former algorithm detects the characteristical elements on the front of the car, while the latter detects the wheels of the car when the front is not visible. For the performance and validation of the project, 9 minutes and 10 minutes video at a rate of 30fps was used for training data and testing data respectively [13]. Since there were two algorithms, the test conditions were first applied to the algorithms individually, and then when both were also working together at the same time. In achieving this, a recall situation was defined by a negative response of the algorithm when there was a very visible car front or visible car wheels. A false detection situation was defined by the positive response of the algorithm when there was completely no visible part of the car on the screen [13]. Using this approach as a strength, the researchers were able to achieve increased reliability, the program was also fast, simple and most of all adaptive. However, they only concentrated on the identification of other cars and paid no attention to other vulnerable road users like cyclists and pedestrians.

In [14], Kwon, Park, Baek, Malaiya, K Yoon, and Ryu in the 2018 IEEE International Conference on consumer electronics developed a vision-based BSD system. Their main drive was the smart connectivity in future autonomous IoT-based vehicles. The main focus of their research was hardware design and development as well as interface architecture and design and the introduction of the idea for object detection in the blindspot based on deep learning methodology. In the architectural layout, they used three cameras one on each of the side mirrors and one on the rear, the design also had a vision board (which is composed of power and camera boards) as well as a PC (personal computer) for visualization of the output. In their approach, a detection area and ground detection area were first identified and then, the ground detection area is further subdivided into four areas [14]. They adopted four stages in the overall system. First, they estimated weather-adaptive threshold values to reduce false alarms and eliminate lane marks and traffic signals in the ground detection area. Second, the optical flow was used as the main feature for object detection and static feature detection was used to avoid false alarms of similar speeds vehicles. The third stage involved the motion-based detection of candidate vehicles. And, the last stage involves decision detection based on **Equation 2.1**;

$$w(j) = \frac{1}{length(j)} \tag{2.1}$$

Where the denominator is the length of the line segment in the ground detection area at j vertical image coordinate. This methodology showed an accuracy of 95.67% in eight weather conditions [14]. On the downside, this accuracy is quite below that obtained by other existing methodologies. On this account, the researchers investigated different deep learning models to improve the accuracy of their system. Among those they considered were Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTMs) networks, and Fully Connected Networks (FCNs). At last, they chose to work with FCN because it can be applied to dynamic

image data, unlike the first two models. With this, accuracy > 95% was obtained [14]. They were not the only ones to have proposed FCN models for this application, D. Kwon, R. Malaiya, G. Yoon, J. Ryu, and S. Pi in [15] proposed a vision-based blindspot detection system using the FCN model as well. For their case, they were able to attain an overall, 99.45% training accuracy and 98.99% testing accuracy of the FCN model were achieved, respectively.

2.2.2. Non-vision-based solutions

N. De Raeve, M. De Schepper, J. Verhaevert, and P. Van Torre [16], proposed a blind spot detection and warning system in which the system warns both the driver and the vulnerable road user. Unlike most non-vision-based systems, their solution was based on BLE (Bluetooth Low Energy) wireless communication and relying on RSSI (Received Signal Strength Indicator) measurements. The system consisted of five detection nodes around the truck which advertise their presence [16]. The vulnerable road user uses a wearable that scans these advertisement packets. The algorithm inside the wearable interprets these messages and applies filtering on their RSSI levels [16]. During a real-life measurement, their system performed reliably well. The first alert for a vulnerable road user starting from the back of the truck was received at a ±8 m distance. The test with multiple vulnerable road users at the same time led to the same results [16]. When the wearable was surrounded by many people, the system alert came at a little later time. In a group of people, only a few needed to wear the wearable to receive an alert, the complete group will be alerted due to the light and sound effect of the others [16]. The outstanding feature of this system is the fact that both parties (the driver and vulnerable road user) are warned. However, the overall system context seems complex, as it requires the design of two standalone subsystems, one for the car and the other, which is a wearable. Besides, developing the two subsystems may not be the main issue but making sure every other road user wear is another issue requiring attention.

Liu, Wang, and Zou [17] proposed a blindspot information system. This system detects and warns in both daytime and nighttime conditions. Their research focused on generally five (5) concepts, and these were; - system architecture, radar system structure and algorithms, IF (Intermediate Frequency) signal processor, motive target detector and blindspot calibration method, and system control strategy. They used the Line Frequency Modulated Continuous Wave (LFMCW) radar system to monitor the moving targets which are in the blindspot areas [17]. The transmitted signal from this millimeter radar system was defined by **Equation 2.2**,

$$T(t) = \theta \cos\left[2\pi\left(f + \frac{Bt}{2T}\right)t\right] \tag{2.2}$$

Where f is the operating frequency of the FMCW radar, B is the bandwidth of modulation frequency, T is the time of modulation frequency [17].

Using the Doppler shift in the range of the transmitted signal, the target can be identified as stationary or moving, and if moving they were also able to deduce its range from the ego vehicle as well as its relative velocity. They then based their choice on the clutter distribution model to select the “cell greatest, smallest and averaging constant false-alarm rate” (CGSA-CFAR)

detection algorithm [17]. The IF signals captured from the front-end of the radar follow Rayleigh distribution, they are first filtered by digital filter banks that can suppress noise effectively. The filtered signal is then fed to the detector. When the researchers experimented with this detector alongside several others, they found out that it outperforms them with a detection rate up to 97.78% and the false detection rate is lower at 2.63% [17]. For the system control, the coordinates of the radar were mapped into the 2D coordinate of the vehicle. Based on the calibrated blindspot area coordinate system, the developed system determines if the target is in the blindspot area or not. The system was implemented on a TI DSP-embedded platform and installed on Chery Arrizo7. Then tests were conducted in real urban environments and considering both daytime and nighttime conditions. Their tests showed that for daytime and nighttime the achieved early warning rates were 98.38% and 98.34% respectively as compared to any system built using computer vision [17]. On the downside, radar-based systems can achieve high accuracy, but rather than being expensive they can also interfere with other wireless systems using the same frequency band.

2.2.3. Hybrid solutions

Very little research has been done while considering the strength of both vision-based and non-vision-based approaches. In the course of this review, the author came across two articles written in this line. In both cases, the researchers used cameras alongside radar or ultrasonic or similar sensor technology to harness the power of both approaches as discussed below;

J. Katarzyna, K. Maciej, and S. Wojciech [18], proposed safety support systems that were designed for the needs of the race Shell Eco-marathon. Shell Eco-marathon is the world's largest race for energy-efficient vehicles [18]. For the blindspot detection they presented three options; First the use of 9 photoelectric sensors with a range of 5m. Second, the use of Microsoft KINECT 4 devices. Lastly, the use of Hokuyo laser scanner with first-class safety. Besides the high equipment cost, they opted for the third solution (the Hokuyo laser scanner) because of its compact design and high-frequency scanning. This makes it optimum for the application. When they simulated the performance of their system, it was possible to alert the driver only when there was a need [18]. The final data presentation to the driver was done in two ways but based on the driver’s preference, these methods include; showing the angle of the approaching vehicle and its corresponding distance to the driver or toggling between three LEDs (left, center, and right) to show the driver that their attention is required.

Still in this line, Pyykonen, Virtanen, and Kyytinen developed an intelligent blindspot detection system for heavy goods vehicles [19]. Even though their choice of the methodology was biased by the fact that sensor installations are limited by the regulation which says protrusions of over 50mm from the vehicle are not allowed, the researchers had up to three sets of options understudy to identify the optimal solution of best performance, cost, and reliability [19]. The first set consists of three Vislab 3D-E cameras, one on the front and then on both sides of the vehicle. All cameras were installed near the top of the vehicle with them facing downwards, thus, any object is elevated from the ground level and detection is straightforward [19]. The second set had a single Vislab 3DV-E stereo camera at the front, and additional three Continental SRR 20X radars installed under the cargo bed. One radar at the right side, one at the left, and the third on

the rear. Finally, the third sensor set had one Vislab 3DV-E camera and several ultrasonic range finders installed under the cargo bed [19]. Their design did not give the driver the camera feeds, but audio and visual feedback as follows. When an obstacle comes in range, say on the right, an audible warning is given on the right side of the vehicle which is also heard by the vulnerable user, on the other hand, the right side of the visual display is also stressed to let the driver know the location of the candidate object. After carrying out several tests, they were able to find out that, stereo cameras can be used to identify small objects. The main downside of this research is that they failed to propose the best sensor combination from the three sets defined, and also some of their tests were only done in simulation software and hence did not consider real World experience.

2.3. Vehicle Safety Features

Vehicle safety technology (VST) in the automotive industry refers to the special technology developed to ensure the safety and security of automobiles and their passengers. Vehicle safety is a very crucial requirement in the design of modern vehicle manufacturers. Safety feature plays an important role when it comes to making a decision buying decision. Manufacturers are continually developing vehicle safety technology to not only mitigate the effects of a collision but, in some cases, avoid one altogether. Vehicle safety kits can be classified in two ways: ‘active’, which intervenes before an accident; ‘passive’, which operates to protect passengers once a crash has happened [20]. An active safety system can help prevent an accident from happening, or reduce the impact of the accident once it happens. **Figure 2-1** below shows generally what safety features and how they work.

Vehicle active safety features can further be divided into two categories namely; the first wave and the second wave active safety features. The former category is already widely introduced in the industry, fitted in most commercial and passenger cars. The latter category is being introduced using new cutting-edge technologies some of these are onboard sensors, radars, lasers, GPS, and Cameras. The second wave is not yet widely distributed but exits in a few luxury vehicles [21].

The two outstanding safety features under the first wave category are the Anti-lock Braking System (ABS), which prevents the wheel from locking when braking heavily and also keeps the driver steering; and the Electronic Stability Control (ESC), which helps prevent the vehicle from skidding while negotiating corners.

The second wave of active safety features include but is not limited to Adaptive Cruise Control; Autonomous Emergency Braking (AEB); Lane Departure Warning (LDW); Lane Keeping Assistant (LKA), unlike LDW which simply warns the driver, the LKA applies a torque to the steering wheel or pressure to the brakes on lane departure; Drowsiness and attention detection; Speed Limit Information (SLI), Tyre Pressure Monitoring Systems (TPMS), Blindspot Information Warning Systems (BIWS) and Intelligent Speed Assistant (ISA).

Some of the common passive safety features employed by vehicle manufacturers are a strong body shell, this resist and dissipate crash forces and offers protection for those in the cabinet;

Dual-stage airbags, these have sensors that trigger them differently depending on the accident severity; knee airbags; side curtain airbags; Isofix child seat mounts.

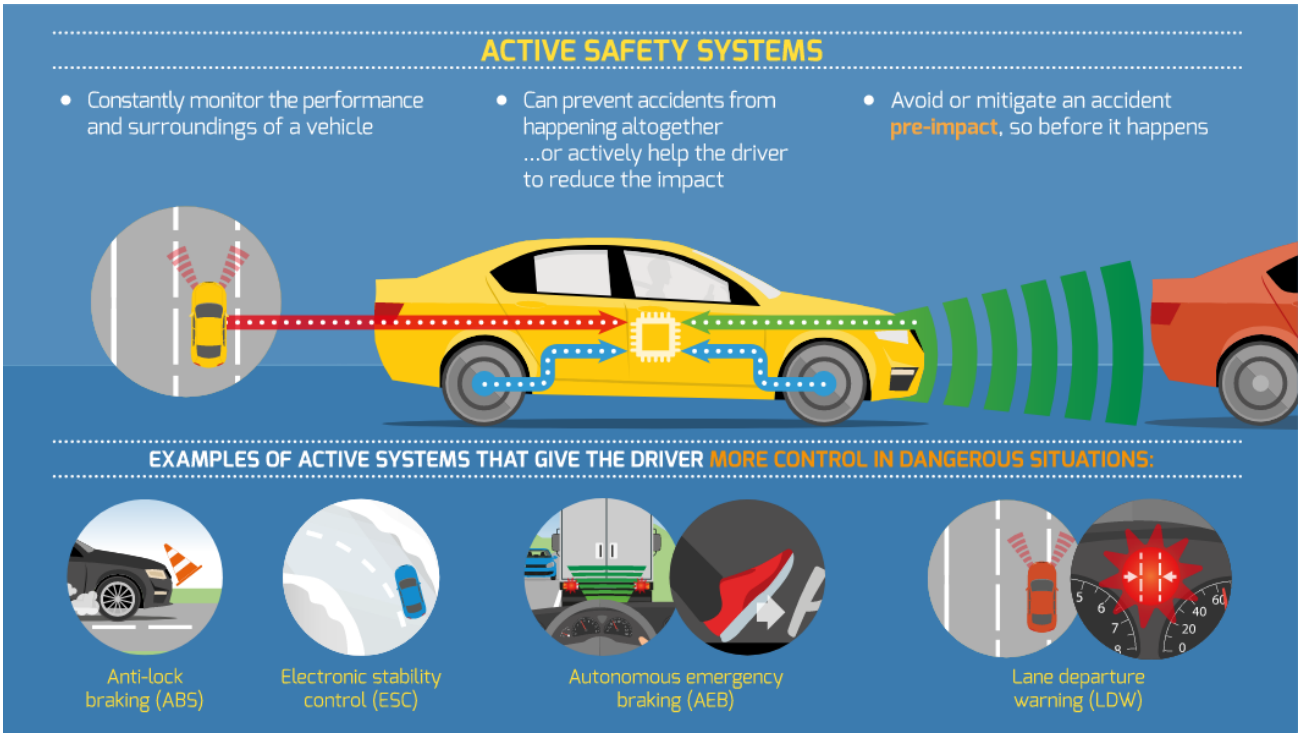


Figure 2-1: How vehicle safety features work, adopted from [21]

2.4. Sensor technologies

A sensor is a device that converts signals from one energy domain to an electrical domain for example humidity sensor, alcohol sensor, pressure sensor, accelerometer, etc. Sensors are now a vital part of any modern automobile design, serving many different purposes. Some of the quantities that require sensors are speed, current, pressure, temperature, positioning, proximity detection, gesture recognition. They are instrumental in helping car manufacturers to bring models to market that are safer, more fuel-efficient, and more comfortable to drive [22]. This is very vital as explained in the preceding sub-section that safety is one key consideration in buying decisions by customers. In the below discussion, the most common sensors used in active safety features are explained.

2.4.1. Ultrasonic sensors

At the moment, ultrasonic sensors are typically mounted into vehicle bumpers for assisted parking systems. So far, such sensors are only expected to function at a driving speed of less than 10 km/hour and they are not able to measure small distances with 100% accuracy. In an autonomous car, however, such sensors could potentially be used in combination with radar, cameras, and other sensing technologies to provide distance measuring functionality. In its basic principle of operation, the ultrasonic sensor transmits an ultrasonic sound and intercepts the sound reflected by the target. The range of the target is obtained using the time taken for the

sound to travel to and back from the target. **Figure 2-2** pictorially demonstrates the mode of operation of this sensor.

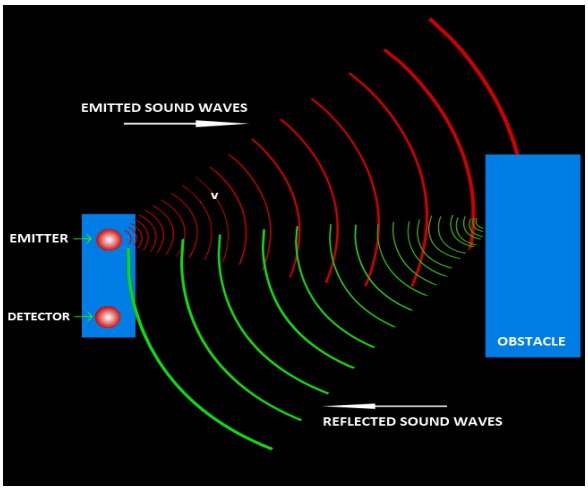


Figure 2-2: Mode of operation of an ultrasonic sensor, adopted from [23]

2.4.2. Radar

The radar unit operates using a specific frequency for either long-range or short-range scanning of the area in front of the vehicle as shown in **Figure 2-3** which is analyzed by a processor to determine the closing distance and time to objects in front of the vehicle. Based upon the information, the vehicle takes the preprogrammed corrective actions ranging from alerting the driver, prefilling the brake reservoir, or applying the brakes. Based on the mechanism of operation of the radar, the radar can measure the range, position, and velocity of the target object.

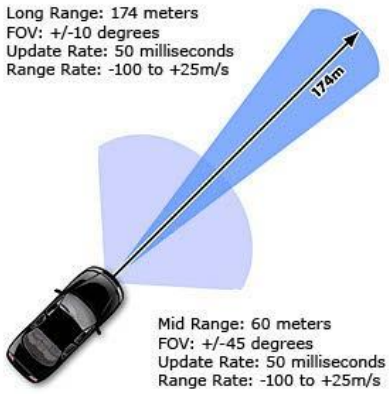


Figure 2-3: Front radar mount, adapted from [24]

2.4.3. Lidar

The limitation of hardware image processing power makes usage of cameras difficult, and this gives another perspective view, thus, Lidar. Lidar seems destined to prove advantageous over cameras in this aspect. Lidar works according to the same principle as radar and is based on the measurement of the reflection of a transmitted signal. While radar relies on radio waves, lidar makes use of light beams (e.g., laser). The distance to the object or surface is calculated by

measuring the time that elapses between the transmission of a pulse and when a reflection of that pulse is received. The big advantage of lidar is that the technology enables much smaller objects to be detected than is possible with radar. In contrast to a camera, which views its environment in focal planes, lidar delivers an accurate, relatively detailed 3D rendering. Through this, it is easy to isolate objects from what is in front of them or behind them, regardless of the lighting conditions (day or night) [22].

2.4.4. Motion sensors

In many situations, we may want to know the motion state of the vehicle to make certain decisions and for these purposes, the motion sensors come in handy. The common sensors are shown in **Figures 2-4, 2-5 and 5-6** employed in motion detection are the accelerometers, an accelerometer is an electronic sensor that measures the acceleration forces acting on an object, to determine the object's position in space and monitor the object's movement; Wheel speed sensor, it detects the rotational wheel speed of vehicles using a non-contacting measurement principle; Steering angle sensor, it determines the steering wheel angle as well as the steering velocity.

Figure 2-4: Wheel speed sensor

Figure 2-5: Steering angle sensor

Figure 2-6: Accelerometer

2.4.5. Camera

This sensor is used to detect the position of the vehicle relative to its intended lane of travel, the presence of pedestrians, and other uses. The data is fed to a processor that allows it to actions ranging from alerting the driver, nudging the vehicle back into its lane if drifting, and steering the car if necessary.

One limitation of this sensor is the current hardware image processing capabilities, it is not near good to yield the best perspective view.

2.5. GUI (Graphical User Interface) design

A graphical user interface (GUI) is a type of user interface through which users interact with electronic devices via visual indicator representations [25]. In-vehicle information systems, the GUI has become a user-centered design. It gives drivers the ability to intuitively operate embedded processors in the vehicle. Some of the common graphical user interfaces in today's vehicles include but are not limited to the instrumentation panel, infotainment systems, navigation panel, and maps.

GUI design principles conform to the model–view–controller software pattern, which separates internal representations of information from how information is presented to the user, resulting in a platform where users are shown which functions are possible rather than requiring the input of command codes [25]. Users interact with information by manipulating visual widgets, which are designed to respond by the type of data they hold and support the actions necessary to complete the user's task.

2.5.1. Advantage of GUI design

There is a stark improvement in usability for the average person. The features of a GUI leverage familiar metaphors, such as drag-and-drop for transferring files, and use familiar icons, such as a trash bin for deleted files, creating an environment in which computer operations are intuitive and easily mastered without any prior practice or knowledge of computing machinery or languages. Graphical user interface applications are self-descriptive, feedback is typically immediate, and visual cues encourage and steer discoverability [25].

2.5.2. Best programming languages for GUI design

Of recent, there are many programming languages at the disposal of a GUI designer. Some of the common languages from which a designer can pick are C#, Java, C/C++, Python, and HTML5/JavaScript. All these languages have their strengths and weaknesses, the choice for a particular task may depend on familiarity, performance considerations, etc.

2.5.3. Best frameworks for GUI design

Like the languages, there are so many frameworks from which a GUI designer can pick a design tool. These frameworks speed up development, as they have prewritten codes and large libraries that save programmers from writing these lines of code by themselves. Some of the common frameworks are;

1. Python frameworks; kivy, Tkinter, QTK, PyQt
2. C/C++ frameworks; TK, GTK, FLTK, CEGUI
3. Java frameworks; Swing, Apache pivot, JavaFX, Standard Widget Toolkit, and Google Web Toolkit (GWT).

Some of these frameworks are cross-platform while others are platform-specific.

2.6. Artificial Intelligence

Artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals. AI refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving [26].

AI is finding use increasingly in the fast-growing world of technology such as automobiles, medicine, space, etc. Today in the fast-growing world, where the computer is playing an essential role in data storage. This artificial intelligence forms the basis for all computer learning and its use. The future for all types of complex decision-making is on artificial intelligence

usage. Chess-playing application in smartphones today is the biggest example of AI. It allows the user to play with a fully automated chess game. The present world of automobiles is seeing the version of self-driving cars or driverless cars which use Al as the major source of satisfying the customer [27].

There are three kinds of AI, and these are Narrow or Weak AI, General or Strong AI, and Artificial Super Intelligence.

Narrow AI is what we see all around us in computers today: intelligent systems that have been taught or learned how to carry out specific tasks without being explicitly programmed how to do so [28]. This type of machine intelligence is evident in the speech and language recognition of the Siri virtual assistant on the Apple iPhone, in the vision-recognition systems on self-driving cars, in the recommendation engines that suggest products you might like based on what you bought in the past. Unlike humans, these systems can only learn or be taught how to do specific tasks, which is why they are called narrow AI.

Artificial general intelligence is very different and is the type of adaptable intellect found in humans, a flexible form of intelligence capable of learning how to carry out vastly different tasks, anything from haircutting to building spreadsheets, or reason about a wide variety of topics based on its accumulated experience. This is the sort of AI more commonly seen in movies, the likes of HAL in 2001 or Skynet in The Terminator, but which doesn't exist today, and AI experts are fiercely divided over how soon it will become a reality.

The artificial superintelligence category performs tasks better than humans due to its superior data processing, memory, and decision-making abilities [29]. No real-world examples exist today.

There is a broad body of research in AI, much of which feeds into and complements each other. The following discussion explores some of the familiar concepts.

2.6.1. Machine Learning (ML)

Currently enjoying something of a resurgence, machine learning is where a computer system is fed large amounts of data, which it then uses to learn how to carry out a specific task, such as understanding speech or captioning a photograph.

2.6.2. Neural Networks and Deep Learning

The key to the process of machine learning is neural networks. These are brain-inspired networks of interconnected layers of algorithms, called neurons, that feed data into each other, and which can be trained to carry out specific tasks by modifying the importance attributed to input data as it passes between the layers. During the training of these neural networks, the weights attached to different inputs will continue to be varied until the output from the neural network is very close to what is desired, at which point the network will have 'learned' how to carry out a particular task.

A subset of machine learning is deep learning, where neural networks are expanded into sprawling networks with a huge number of layers that are trained using massive amounts of data. It is these deep neural networks that have fueled the current leap forward in the ability of computers to carry out tasks like speech recognition and computer vision.

There are various types of neural networks, with different strengths and weaknesses. Recurrent neural networks are a type of neural net particularly well suited to language processing and speech recognition, while convolutional neural networks are more commonly used in image recognition. The design of neural networks is also evolving. Researchers recently refined a more effective form of deep neural network called long short-term memory or LSTM, allowing it to operate fast enough to be used in on-demand systems like Google Translate.

2.6.3. Elements of Machine Learning

There are generally three elements of machine learning and these are; supervised learning, data sets are labeled so that patterns can be detected and used to label new data sets; secondly, unsupervised learning, data sets aren't labeled and are sorted according to similarities or differences. Finally, in reinforcement learning, data sets aren't labeled but, after performing an action or several actions, the AI system is given feedback.

2.6.4. Modern applications of AI

AI has the unique ability to extract meaning from data when you can define what the answer looks like but not how to get there. AI can amplify human capabilities and turn exponentially growing data into insight, action, and value. Today, AI is used in a variety of applications across industries, including automotive, healthcare, manufacturing, and government [29]. Here are a few specific use cases:

i. **Prescriptive maintenance and quality control** improve production, manufacturing, and retail through an open framework for IT/OT. Integrated solutions prescribe the best maintenance decisions, automate actions and enhance quality control processes by implementing enterprise AI-based computer vision techniques.
ii. **Speech and language processing** transforms unstructured audio data into insight and intelligence. It automates the understanding of spoken and written language with machines using natural language processing, speech-to-text analytics, biometric search, or live call monitoring.
iii. **Video analytics and surveillance** automatically analyze video to detect events, uncover identity, environment, and people, and obtain operational insights. It uses edge-to-core video analytics systems for a wide variety of workload and operating conditions.
iv. **Highly autonomous driving** is built on a scale-out data ingestion platform to enable developers to build the optimum highly autonomous driving solution tuned for open-source services, machine learning, and deep learning neural networks.
v. **Machine vision:** This technology gives a machine the ability to see. Machine vision captures and analyzes visual information using a camera, analog-to-digital conversion, and digital signal processing. It is one highly used AI technology in vehicle safety features.
vi. **Robotics:** Researchers are building robots that use AI to interact in social settings.

2.7. Microcontrollers and microprocessors

A microcontroller is a solitary chip microcomputer fabricated from VLSI fabrication. A microcontroller is also known as an embedded controller [30]. Today various types of microcontrollers are available in the market with different word lengths such as 4bit, 8bit, 64bit, and 128bit microcontrollers. A microcontroller is a compressed microcomputer manufactured to control the functions of embedded systems in office machines, robots, home appliances, motor vehicles, and several other gadgets [30]. A microcontroller comprises components like - memory, peripherals, and most importantly a processor. Microcontrollers are employed in devices that need a degree of control to be applied by the user of the device.

The key distinction between a microcontroller and a microprocessor is the fact that a microprocessor has only the central processing unit (CPU) while a microcontroller has peripheral components shipped with it alongside the core processor. It is also important to note that, microcontrollers are intended for embedded applications, unlike processors which are being used in PCs and other all-purpose devices. The basic structure of a microcontroller consists of the following; -

i. **CPU:** This is the brain of the microcontroller which is employed to fetch data, decode it and at the end complete the assigned task successfully.
ii. **Memory:** This is used to store programs and data. Inherently, microcontrollers are built with a certain amount of ROM or RAM or Flash memory.
iii. **Input/Output ports:** These are used to interface with devices like LEDs, printers, etc.
iv. **Serial ports:** This allows the microcontroller to interact with its peripheral devices via serial communication protocols.
v. **Timers:** A microcontroller might be inbuilt with one or more timers. These control all timing and counting operations in the microcontroller.
vi. **ADC (Analog to digital converter):** This converts analog signals to digital format.
vii. **DAC (Digital to analog converter):** This converts digital signals to analog format.
viii. **Interpret control:** This is employed for giving a delayed control for a working program, and this device can be internal (On the Circuit board) or external.
ix. **Special functioning block:** In some special microcontrollers such as those employed in space systems or robots, there are some special units employed to perform special functions.

Microcontrollers can be categorized based on memory, bits, architecture, and instruction sets. Based on bit categories, we can have 8, 16, 32, or even 64-bit microcontrollers. The greater the word length in terms of the bits, the greater the performance and the accuracy. Therefore, for precision systems, microcontrollers with higher word lengths are preferred. Memory-wise, microcontrollers can be categorized as external memory or embedded memory microcontroller. The former doesn't have a working memory on-the chip with it, while the latter has a functional memory block in the chip with the microcontroller. The external and embedded memory features each have their strengths and weaknesses, and thus, a particular choice will depend on many design considerations.

Based on instruction sets, microcontrollers can be categorized as CISC-CISC or RISC-RISC. CISC- CISC means complex instruction set computer, it allows the user to apply 1 instruction as an alternative to many simple instructions. RISC- RISC means Reduced Instruction Set Computers. RISC reduces the operation time by shortening the clock cycle per instruction. Lastly, microcontrollers can be categorized based on memory architecture as Harvard or Princeton memory architecture microcontrollers.

2.7.1. Embedded systems in automobiles

According to Nicholas and Francoise [31], The automotive industry is today the sixth-largest economy in the world, producing around 70 million cars every year and making an important contribution to government revenues all around the world. In these cars, significant improvements in functionalities, performance, comfort, safety, etc. are provided by electronic and software technologies. Since 1990, the field of embedded electronics and more precisely embedded software has been increasing at an annual rate of about 10%, and in 2006, the cost of an electronic-embedded system represented at least 25% of the total cost of a car and more than 35% for a high-end model [31]. This stated cost is equally shared between the electronics and the embedded software. It should be noted, the development of an embedded system comprises both software and hardware, and thus, the total cost is the contribution of both development domains.

The above developmental trend has led to currently embedding up to 500MB or more on over 70 microprocessors connected via communication in the vehicle, the CAN network. **Figure 2-9** which is adapted from [31] demonstrates the various embedded components consisting of various microcomputers interconnected to control the engine and other functionalities in the vehicle.

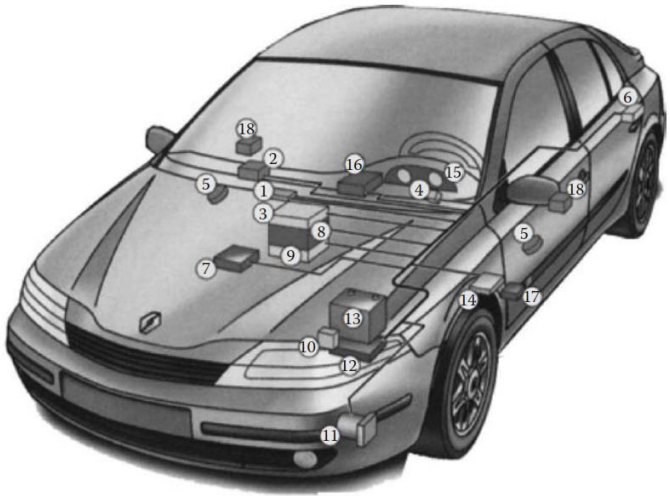


Figure 2-7: Renault Laguna showing various embedded electronics, adopted from [31]

In 2004, the embedded electronic system of a Volkswagen Phaeton was composed of more than 10, 000 electrical devices, 61 microprocessors, three controller area networks (CAN) that support the exchanges of 2, 500 pieces of data, several subnetworks, and one multimedia bus.

The automotive industry has evolved rapidly and will evolve even more rapidly under the influence of several factors such as pressure from state legislation, pressure from customers, and technological progress (hardware and software aspects). Indeed, a great surge for the development of electronic control systems came through the regulation concerning air pollution.

In-vehicle embedded systems are usually classified according to domains that correspond to different functionalities, constraints, and models. They can be divided among “vehicle-centric” functional domains, such as power train control, chassis control, and active or passive safety systems, and “passenger-centric” functional domains where multimedia/telematics, body/comfort, and human-machine interface (HMI) can be identified. This project falls under the “vehicle-centric”, and more precisely active safety domain.

2.7.2. Programming languages

As far as programming languages are concerned, in the embedded systems the normal programming languages that programmers employ when designing the normal software programs are used. The only attention that designers have to pay attention to are a few syntax changes to suit embedded systems. For example, in Python, a programmer has to learn the syntax of micro-python. These rules are however not that complex or far from the usual programming rules. As stated in the beginning, many languages are being employed but Python, C, and C++ have topped the list of recent for their outstanding strengths and familiarities. Many lesser-known languages such as Elixir and Ada are also used for programming embedded devices [32].

2.8. Conclusion

In this chapter, the author identified three themes in the scholarly reviews; - the vision-based system, the non-vision-based system, and the hybrid system. The vision-based system utilizes cameras which then employ computer vision to identify meaningful patterns in images or videos. The non-vision-based system employs sensors mainly ultrasonic sensors, lidar, or radar. And, the hybrid is the combination of vision and non-vision-based features.

It has also been pointed out that, vehicle safety features are very important considerations in buying decisions. Vehicle safety features can be broadly divided into passive and active features. The latter is associated with safety features that are aimed at preventing accidents while the former is aimed at protecting passengers and the vehicle when an accident occurs. In the design of these safety features and other embedded systems in a car, many sensor technologies are employed. The most common sensor technologies being employed in safety features are ultrasonic sensors, radar, lidar, motion sensor, and camera.

To improve user experience most of the software systems currently employ the GUI design as opposed to the Command Line Interface (CLI). The GUI uses the pictorial representation to allow users to execute certain commands on their computers while the CLI requires the user to specify a command in a terminal. The common languages used for the design of modern GUIs are C, C++, Java, and Python. While these are common, many other languages are also employed. To shorten development time, designers most times use frameworks other than writing codes from scratch. Some of the common frameworks are Kivy, PyQt, GWT, Swing, etc.

Artificial intelligence is an interesting field that is finding great acceptance in the automotive industry. In machine learning, a huge amount of data is fed to a machine from which it identifies patterns to make future predictions. Direction is shifting to Neural networks and deep learning as these give better performance and accuracy concerning image and natural language processing. The main applications of AI in the automotive industry are in object detection and autonomous vehicles. One important limitation of AI is the computing capabilities of the hardware especially when it comes to computer vision. Attention has to be paid to optimization algorithms to enable deployability on small-sized microcontrollers.

There has also been some development in the field of microcontroller and microprocessor technology concerning vehicle manufacturing. Embedded systems are the core of the vehicle structure, and are employed in every electronic control system. Embedded system development is a complex problem as it involves both hardware and software development and thus, an error can occur in both domains. The top languages being used in embedded system development are C, C++, and Python. The choice of the language may be influenced by many factors such as ease of use, readability, familiarity, etc.

As stated at the beginning of this literature review, most of the work done was either targeted vision-based or non-vision-based blind spot detection systems. Both systems have strengths and weaknesses, for instance, the vision-based does not perform well on rainy days or nighttime conditions. Also, the non-vision-based systems besides being expensive cannot at most times detect humans or smaller objects like bikes. Our project aims at harnessing the strength of both, the vision and non-vision-based systems. For the non-vision, we intend to use ultrasonic sensors over radar fact that they are cheaper, and also being best for object presence detection and profiling. More importantly, if the target is in a dirty and wet environment where it is moving quite slowly, ultrasonic sensors flourish. For the vision functionality, we will adopt computer vision techniques using an open-source Python library called OpenCV for preprocessing and feature extraction and then feed these features to a Fully Connected Network (FCN) model for object identification. This project not being explicitly an AI project, we will adopt a trained or pre-trained model that will meet our design requirements as will be outlined in the following chapter. Our design shall be implemented using the Raspberry Pi processor and Python programming language for the reasons of suitability, cost, and familiarity.

3. Methodology

3.1. Introduction

This chapter will discuss the approaches adopted to implement this project. The main objective of this project was to develop a system that detects objects in blind spot areas of the Kayoola Buses and alerts the driver of their proximity. To achieve this main objective, the author executed the following tasks; - developed the hardware and software requirements for the system, designed physical and logical design models for the system, implemented an object detection model, and finally came up with a functional prototype that conforms to the system requirements. The roadmap towards the project objective is best illustrated by the intervention logic shown in **Table 3-1**.

Table 3-1: Intervention logic

SN	Milestone	Key Questions	Instruments, Tools, Methods & Data Sources
1.	Requirements Engineering	(1) Who are the users of the system? (2) What are the system features? (3) What are the external interface requirements? (4) What are the functional requirements of the system? (5) What are the non-functional requirements of the system?	Data Sources: Papers, Books Methods: Desk Research, Benchmarking, Interviews Tools and Instruments: Internet Output: SRS (System Requirement Specification)
2.	System Architecture and System Modelling	(1) What are the components of the system? (2) How will the components of the system interact? (3) What are the boundaries of the system? (4) What other systems will the system interact with? (5) What are the views, models, behavior, and structure of the system?	Data Sources: Internet, Books, Methods: Desk Research, Drawing/ Modelling Tools: Star UML Output: SAD (System Architecture Description)
3.	Design Specification	(1) What does the system do and how quickly does it do it? (2) What user interfaces does the system have? (3) In what environment will the system be used? (4) What are the inputs, input data types, and outputs of the system? (5) How are the inputs processed? (6) How much power is needed for the different components of the system to operate?	Data Sources: Internet, SRS Methods: Conceptual Data Modelling Tools: IEEE Standards, Star UML, Fritzing software Output: SDD (System Design Document)

4.	Test Specification	(1) What is the scope of the testing? (Components that will be tested) (2) What type of testing will be performed? (3) What are the objectives of testing the system? (4) What is the test environment?	Data Sources: Internet, SRS, SDD Methods: Unit Testing, Integration Testing, System Testing Tools: Configuration management tools
5.	Implementation	(1) What are the competencies of the team members? (2) What hardware will be used? (3) What software will be used for development? (4) What development methodologies will be employed to come up with the various components of the system? (5) What Programming Language(s) will be used to develop the system?	Data Sources: Internet, SRS, SDD, Books Methods: A mix of Prototyping Tools: Equipment Datasheets, Micro Controller Units, Sensors, Actuators Output: Prototype

The rest of this chapter has been organized as follows; the system requirements are presented, followed by the system architectural views. This is then followed by an intuitive design of the system design using UML (Unified Modelling Language). The test specification is presented and then the details of the software and hardware implementation. This chapter is concluded by giving the rationale for choosing these methods of implementation, and the challenges met.

3.2. System requirement analysis

Table 3-2 shows the requirements for the system. The requirements have been divided into two based on the sub-systems, which are the software and hardware subsystems. Under each subsystem, the requirements have been further categorized as functional and non-functional requirements. The **REQF** and **REQNF** prefixes in the **ID** column corresponds to the functional and non-functional requirements respectively.

It should be understood that the threshold distance of 1m utilized in the requirements is a limitation of the ultrasonic sensor type used. This sensor is cheap making it better for prototyping and yet has a range of about 4m.

Table 3-2: System requirements

SN	Sub-System	ID	Requirements Description
1.	Software	**REQF001**	Shall provide a visual display of the location and distance of the objects in the blind spots of the bus in real-time.
		REQF002	Shall be capable of reading raw sensor values from the accelerometer, ultrasonic sensors, and camera.
		REQF003	Shall process raw sensor values into formats suitable for decision making as well as formats that can be interpreted by the user.
		REQF004	Shall be capable of initiating the blinking of LEDs when an object in the blind spot

			surpasses the defined threshold distance value [1m].
		REQF005	Shall be capable of initiating auditory feedback when the distance between the bus and object gets smaller than the threshold [1m].
		REQF006	Shall start on system start-up.
		REQF007	Shall be activated when the motion of the bus has been detected.
		REQF008	Shall seamlessly interact with the hardware sub-systems which include; - the Raspberry Pi and the peripheral devices.
		REQNF001	Shall not fail due to inability to read sensor outputs
		REQNF002	Shall withstand component and environmental failures.
		REQNF003:	The functions of the software shall be easily understood by the user (the driver).
		REQNF004	Worst case sensor response time shall be 1s.
		REQNF005	Shall use relatively optimum system resources, such as memory, CPU, and disk.
		REQNF006	Shall identify the root cause of failure when it occurs.
		REQNF007	Shall be easily tested for any desired features.
		REQNF008	Shall be readily installable on the Raspberry Pi
		REQNF009	Shall conform to the Raspbian OS.
		REQNF010	There shall be ease in the replacement of the different software components at any desired time.
		REQNF011	Shall require minimum attention of the user (i.e., the driver does not need to continuously glance at the display) so they can focus on other tasks.
		REQNF012	Shall present a user interface that is slick, intuitive, and attractive.
		REQNF013	Shall notify the user if the system fails.
2.	Hardware	**REQF009**	It shall have ultrasonic sensors for range measurements of target objects
		REQF010	The accelerometer shall be able to measure the presence or absence of a motion to provide system power on, off, or sleep mode regardless of the different environment variations (for example temperature and background noise).
		REQF011	The control unit (i.e., the Raspberry Pi) shall handle fast calculations and computations from the sensors and deduce a given set of instructions corresponding to the sensor values
		REQF012	The Camera shall capture frames from the blind spot areas that shall be fed to the object detection model.
		REQF013	The LEDs shall illuminate at the start of the bus to show that they are in proper working conditions. They shall blink when there is a body near the body of the bus.
		REQF014	The Speaker shall produce an alarm when an object or vehicle is near the body of the bus.
		REQNF014	When an unpredictable failure occurs in reading values from either the accelerometer or the ultrasonic sensor, the system shall recover briefly to full capacity or safe mode respectively. This will depend on the intensity of the bug
		REQNF015	The system shall be able to handle many inputs from its environment.
		REQNF016	The different components shall be enclosed in a casing to keep the connections firm and protect them from mechanical damage.

3.3. System modeling and system architecture

The system architecture seeks to show how the various independent components of the BSD System built by various manufacturers using different environments and protocols can be brought together to function as a single comprehensive system. The main goal of the project is to improve the safety of passengers on the Kayoola EVS bus through; - simplifying lane change in heavy traffic by monitoring the blind spot areas of the Kayoola Buses.

Two architectural principles have been adopted for this activity, and these are UML (Unified Modelling Language) and the component-based architecture paradigm. The latter is adopted

because the system in question is composed of identifiable components. The former was chosen because it's the most used in the industry.

3.3.1. Functional scenarios

During this architectural description, the focus was on the following scenarios

Table 3-3: System functional scenarios

Scenario Reference	**SR1**
Overview	Blind Spot Monitoring
System State	Ultrasonic Sensors sending Sonic waves
External Stimulus	Environment
Required System Response	Receive reflected sonic waves and determine the distance of the obstacle
Scenario Reference	**SR2**
Overview	Object Detection
System State	The camera attached to the Ultrasonic sensor activated
External Stimulus	Object in Blindspot
Required System Response	Detect the object in the blind spot
Scenario Reference	**SR3**
Overview	Alerting Driver
System State	Display of camera feed on the LCD
System Environment	Linux
External Stimulus	Existence of Object in Blind Spot
Required System Response	Blinking of LEDs and Auditory feedback

3.3.2. Context diagram

The context view describes the relationships, dependencies, and interactions between the system and its environment; i.e., the people, systems, and external entities that it interacts with.

Figure 3-1 shows the context diagram of the proposed system. Generally, there is one external entity that interacts with the system and this is the driver. The driver supplies some inputs to the system labeled by the "User inputs" and the system in question gives the driver feedback labeled as "System response".

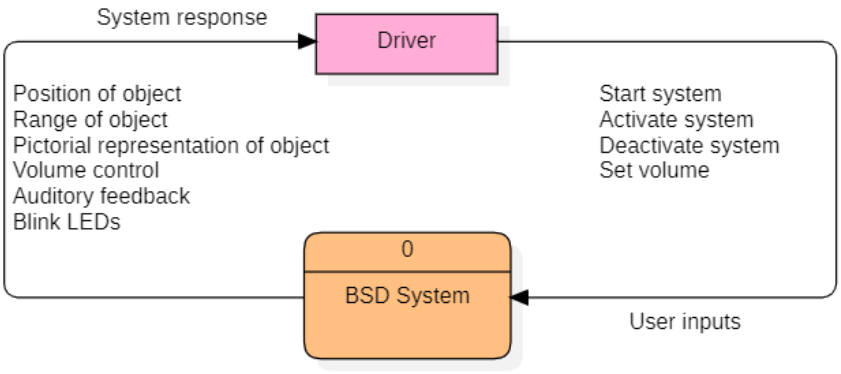


Figure 3-1: BSDS context diagram

3.3.3. Interaction scenarios

Some of the complex interaction sequences of the BSD system and its external entities are modeled and represented using UML sequence diagrams to help uncover implicit requirements and constraints and help to provide a further more detailed level of validation.

The sequence diagram in **Figure 3-2** modals the interaction between the involved entities and the Blindspot detection system. The system is activated when the car starts moving, and this is detected by the accelerometer. The location parameter of the target vehicle is picked up by the ultrasonic sensor, and the system initiates the object detection algorithm. To detect the object in the scene, the system identifies the camera associated with that particular ultrasonic sensor and uses its feed. This feed is applied to a deep learning modal. Once the object is identified, the system shows the information on the GUI display. The system then determines if the target is within the threshold region, if so, the system initiates auditory feedback. When the detected object is on the side of the bus and it is within the danger zone, the LED corresponding to that side starts to blink.

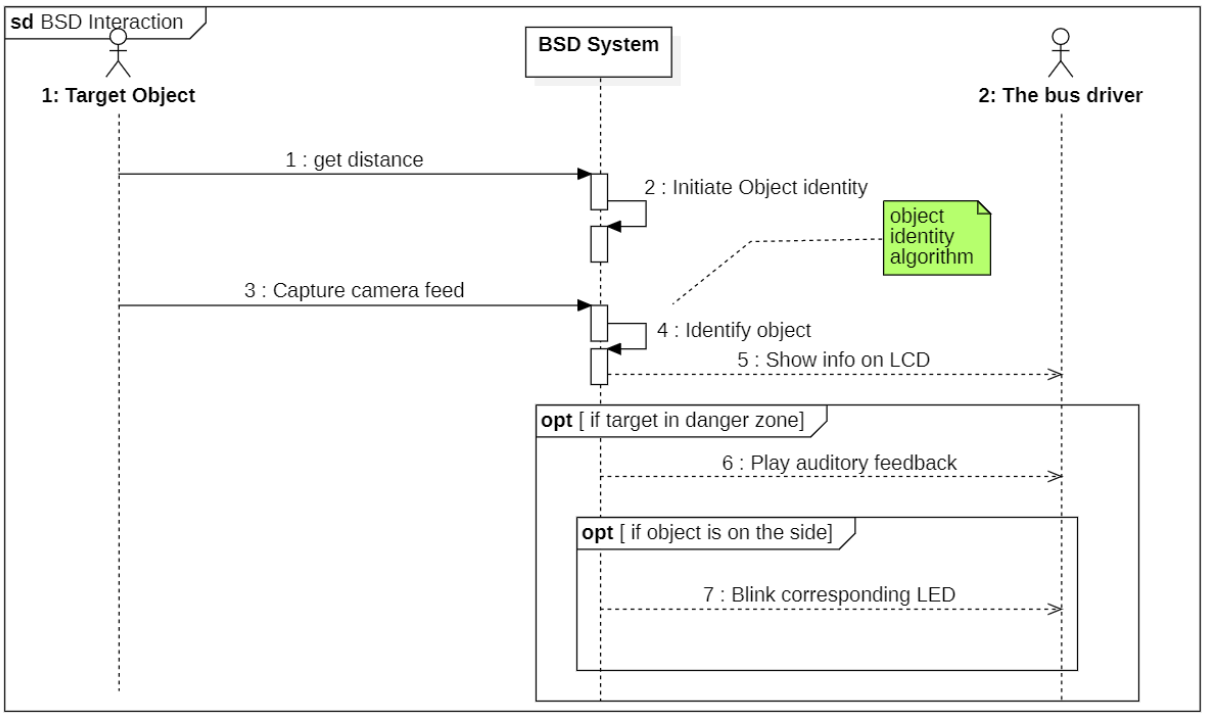


Figure 3-2: Sequence diagram for interaction between entities

3.3.4. Functional Architectural View

The functional view of the BSD system defines the system's architecturally significant functional elements, the responsibilities of each, the interfaces they offer, and the dependencies between these elements. **Figure 3-3** shows the functional architecture of the BSD system. Table 3-4 shows the decomposition of some elements into further sub-modules.

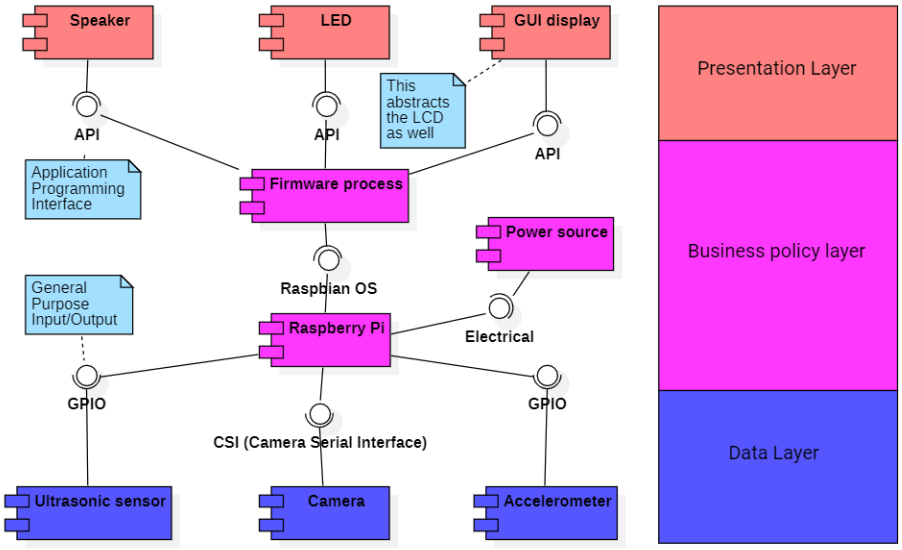


Figure 3-3: Functional Architecture

Table 3-4: Functional elements

Element Name	Responsibilities	Interfaces
Raspberry Pi	It receives data from the sensors. It processes this data and makes decisions to take necessary actions based on the result.	Raspbian OS, GPIO, and CSI
Ultrasonic sensor	This uses ultrasonic sound to detect the presence of objects and compute their distance.	GPIO, Air interface
Accelerometer	This sensor detects if the bus is in motion or stationary.	GPIO
GUI display	This is the visual display; the LCD which is part of this component's visual information will be displayed.	API, Touch Screen
Speaker	Use to give auditory feedback	Aux interface
LEDs	This component blinks when the target's distance from the bus becomes less than the predefined threshold.	GPIO
Camera	Captures live feed for object detection	CSI
Power source	This supplies DC power to the system	Electrical
Firmware	This process reads sensor data and presents them to the GUI, Speake, and LED	Raspbian OS and API

3.3.5. Other Architectural Views

Apart from the function scenarios, contextual view, interaction view, and functional view, there were many other architectural views that the author investigated to give a concrete picture of the working of the system. However, for brevity, the others have not been discussed deeply here but the information and concurrency views can be found in **Appendix A**. This is mainly because the above is more than sufficient to give the reader an intuitive understanding of the system operation.

These architectural views that have intentionally been omitted are the following; -

1) **Information view:** This defined the structure of the system's stored and transient information and how related aspects such as information ownership, flow, concurrency, latency, and retention were addressed.
2) **Concurrency view:** This defined the set of runtime system elements (such as operating system processes) into which the system's functional elements are packaged.
3) **Deployment view:** This defined the important characteristics of the system's operational deployment environment. It specifies the runtime platform and software dependencies.
4) **Development view:** This defined some of the key constraints on the software development process that are required by the architecture. This includes the system's module organization, common processing that all modules must implement, all required standardization of design, coding, and testing, and the organization of the system's code line.
5) **Operational View:** This defined how the system will be installed into its production environment, how data and users will be migrated to it, and how it will be configured,

managed, monitored, controlled, and supported once this is achieved. The aim of the information in this view is to show how the operational environment is to be created and maintained, rather than to define detailed instructions or procedures.

3.4. System design

3.4.1. Circuit design

Figure 3-4 shows the basic circuit design of the core components. It should be understood that some of the inherent components of this system have been omitted in this diagram for simplicity. These components that are missing are the power source, the speaker, and the display screen. One other important reason for omitting them is the fact that they require special connectors, hence making the entire diagram complex.

A raspberry pi model 4 has been used for this implementation. Two ultrasonic sensors have been employed corresponding to the left and right sides of the bus. In the actual deployment environment, more will be required to capture all the other blind spot regions. Similarly, two LEDs were used corresponding to the left and right. The motion sensor used was the accelerometer with model number MPU6050. The camera connects to the Camera Serial interface as shown in the figure.

Resistors R1 and R2 connected to the respective cathodes of the LEDs are aimed at reducing the peak current drawn by the LEDs, hence protecting the GPIO pins from being destroyed. In the same sense, resistor pairs R3, R4, and R5, R6 are used to protect the GPIO pins. The voltage output of the ECHO pin of the HC-SR04 sensor gives 5V which is high since the GPIO pins only require 3.3V. These resistor pairs, therefore, form a voltage divider network that reduces the 5V to a safe level. Though not shown for the reason stated above, the speaker connects to the aux port and is powered by 5V. the display screen connects to the HDM_0 port and is also powered by a 5V supply. The raspberry pi is powered via the port labeled POWER IN, in the diagram.

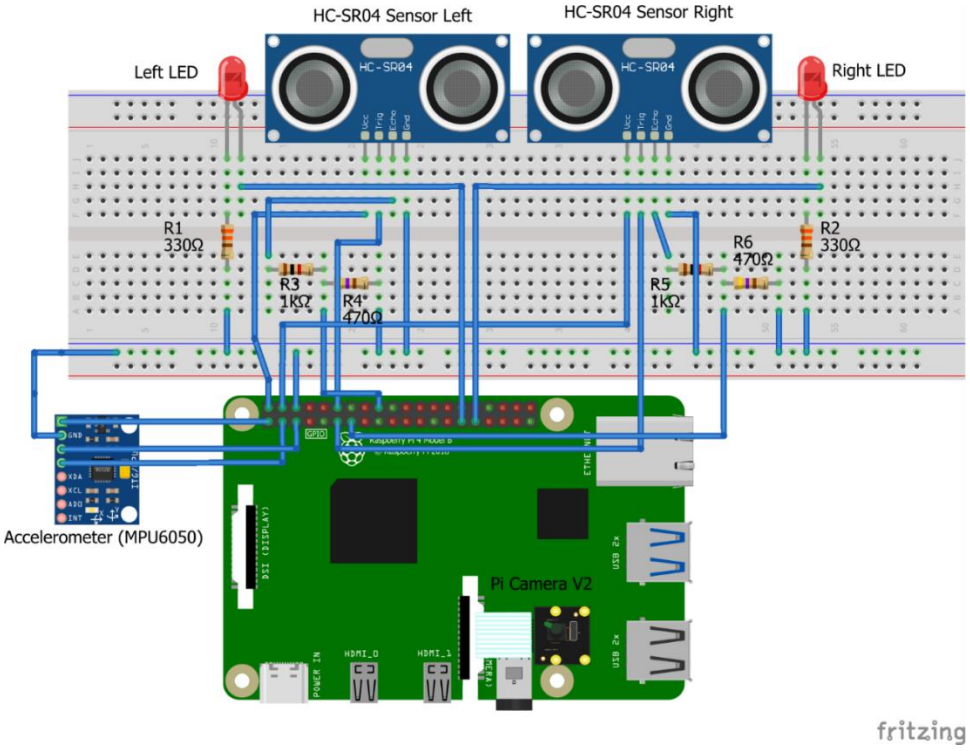


Figure 3-4: BSD Circuit design

3.4.2. Wireframes

Three screens have been designed for the GUI display using Adobe XD (Experience Design) software, these are the splash screen, standby screen, and monitor screen. **Figure 3-5** shows the splash screen, and this is the screen shown as the program loads. **Figure 3-6** shows the standby screen, and it is shown when the program is started but not yet activated by the motion sensor or manually. Finally**, Figure 3-7** shows the monitoring mode screen and it is shown when the program has started and is activated. In this mode, the system constantly scans for objects around the bus and provides valuable information to the driver.

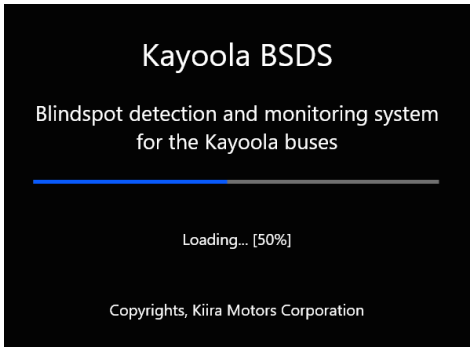


Figure 3-5: Splash screen

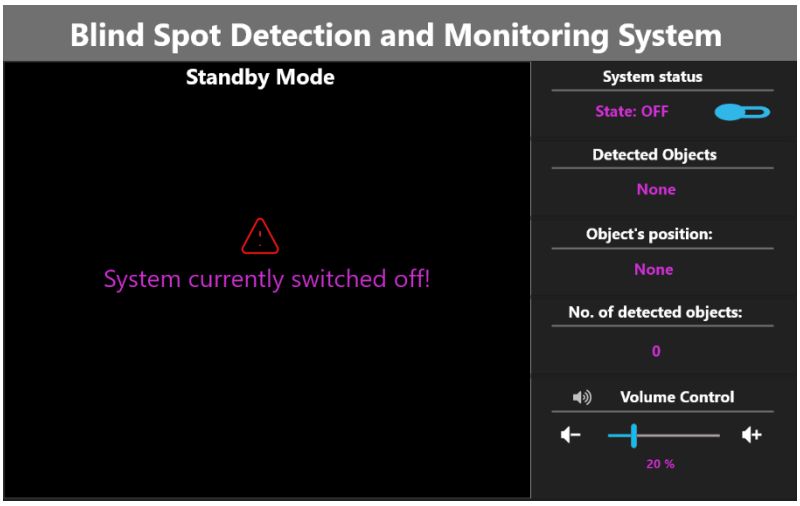


Figure 3-6: Standby screen

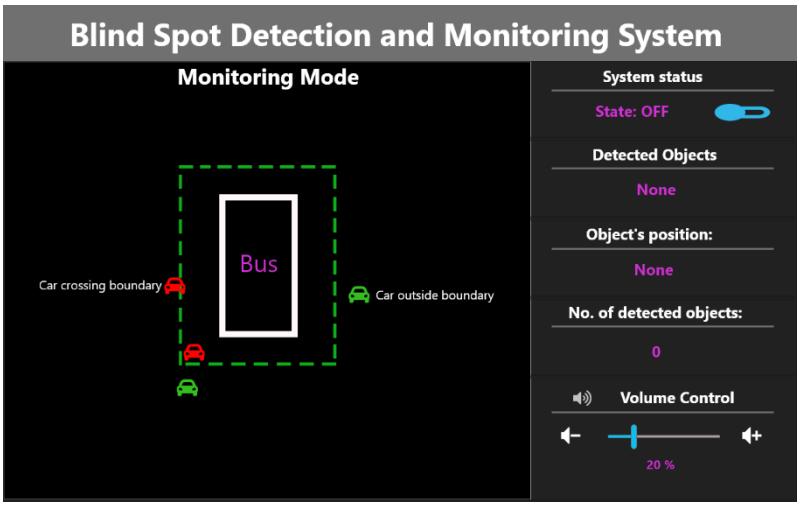


Figure 3-7: Monitoring mode screen

3.5. Test specification

"Dynamic testing" (in short, testing) is one of the most significant analytical quality assurance techniques for software, as it is the most elementary and certainly the most frequently used form of quality assurance [31]. In practice, it's the only method that allows for taking into consideration the actual development. It ensures that the dynamic properties of the system are checked. These properties are the runtime behavior and the computational accuracy. Apart from verifying conformance to system requirements and making release decisions, testing also minimizes risks. Systematic testing prevents projects from proceeding to the next step without passing the required test scenarios.

A core element in quality assurance of automotive-embedded system development is in-vehicle or road testing. This element is most suited for calibrating control parameters and validating the entire vehicle. However, it is often misused when it comes to identifying software-related specifications, design, and implementation errors. Compared to road testing, "systematic testing

of software" during its development enables earlier and more efficient detection of software-related errors. This concept works equally well for the hardware domain. However, this requires a practically oriented adaptation of the general testing techniques and approaches to the specifics of the application domain. To ensure high effectiveness, the tests should be carried out during the development process rather than just at the end of the development [31].

The system in question is averagely complex, and therefore involves many test activities being conducted at different levels of development. To structure the test processes and facilitate testing such complex systems, test phases have been defined as follows; - unit testing, integration testing, and system testing. Unit testing evaluates the performance of an independent unit of the system, such as a piece of code that performs a specific purpose. Integration testing evaluates the satisfaction of how a unit fits into the larger system. Finally, the system testing checks to see how all units fit together to meet the system mission statement. The test basis for these test phases will include; - system requirement specifications, use cases, and interaction scenarios. All these have been discussed at the beginning of this chapter.

It should be clearly understood that the in-vehicle test was not conducted. The test environment for code units was the development of Raspberry Pi 4 Model B.

3.6. System implementation

3.6.1.System algorithm

Figure 3-8 shows the core algorithm of the system, and this at a high level describes how the system was implemented to achieve its mission. The description below gives how the operation flows; -

1) The program starts at the system start-up of the Raspbian operating system
2) If the accelerometer detects motion, it activates the system and the active mode view is loaded on the display otherwise the standby mode view is loaded instead. The user can also manually turn OFF or ON the system using touch events on the touch screen.
3) Once the active mode view is loaded, the ultrasonic sensors and LEDs are initialized. At the same time, virtual coordinates using screen pixels are created alongside mapping matrices and are stored in memory.
4) The distance sensor then routinely monitors nearby objects, once detected it identifies it using the object detection model. It then either adds it to the canvas or updates the already existing object if it was detected before.
5) If the object is in the danger zone, the system sounds auditory feedback, and if the object is on the side of the bus, the LED corresponding to that side of the bus blinks.

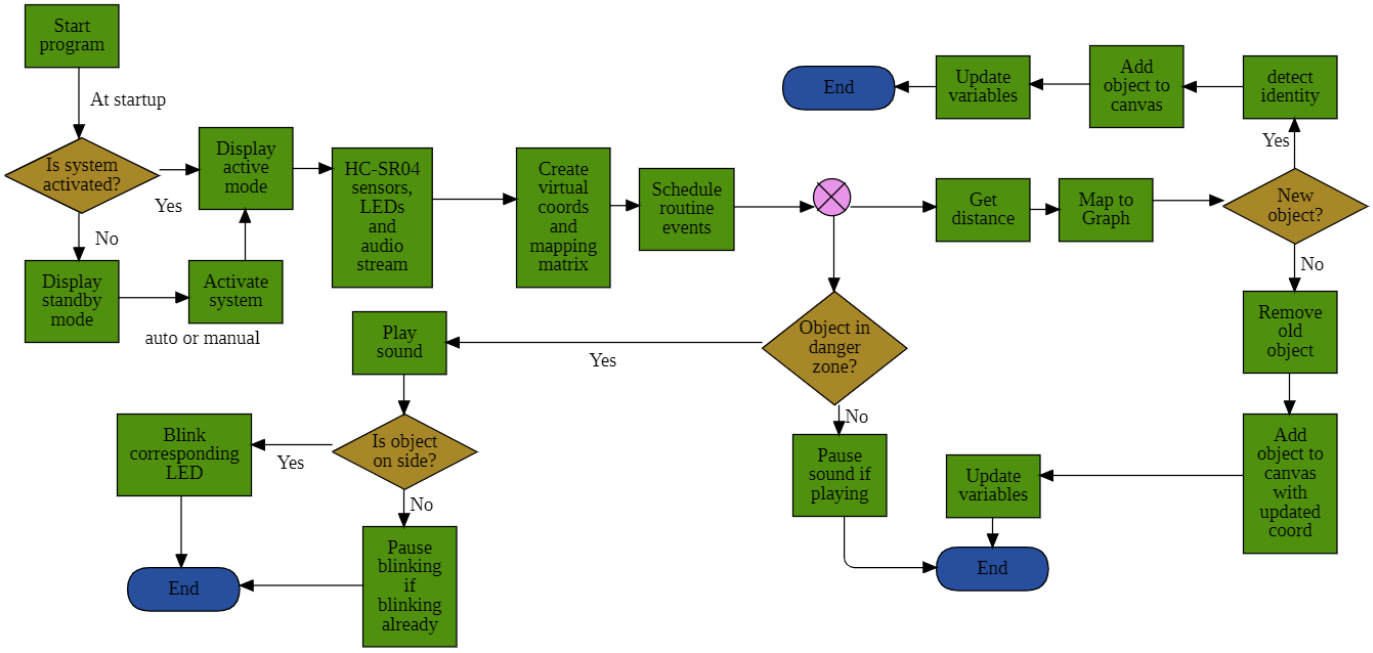


Figure 3-8: Core Algorithm

3.6.2.The concept of coordinate mapping

The graph data structure was adopted for this implementation; this is an abstract data structure consisting of nodes and edges. In this concept, the absolute distances measured using the ultrasonic sensor represent the nodes while their equivalents in pixels which specify the location of the object on the canvas represent the edges.

In **Figure 3-9**, the left diagram shows how the virtual coordinates have been divided while the right figure shows the order of operations during mapping of the nodes to the edges as explained below.

The first step in coordinate mapping between the measured distance and pixel coordinates is the creation of the possible pixel coordinates and their access matrices. We herein refer to the pixel coordinate as virtual coordinate. This is done as the program is loading after creating the canvas widget on which the objects will be placed after detection. A widget is a component of an application interface.

After the virtual coordinate creation process is completed, the system reads distance using the ultrasonic sensor and then maps this value to its corresponding pixel coordinate.

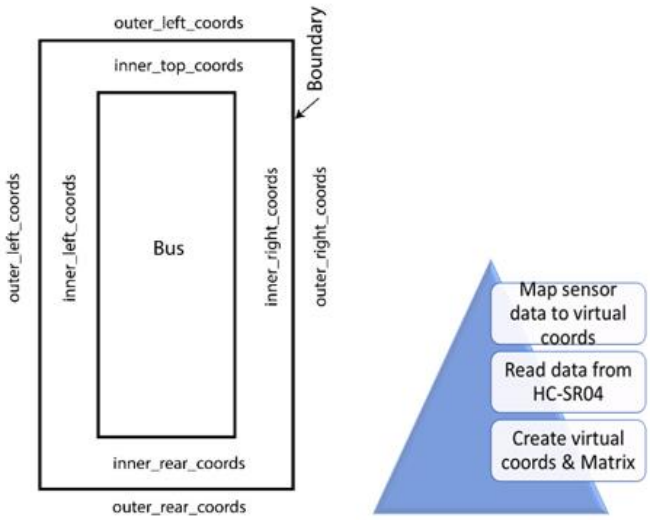


Figure 3-9: Mapping system

3.6.3.GUI implementation

for this purpose, the Python programming language was used, and more specifically the Kivy framework was used. Kivy - Open-source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps. Kivy runs on Linux, Windows, OS X, Android, iOS, and Raspberry Pi. Kivy is 100% free to use, under an MIT license. It is also GPU (Graphics Processing Unit) accelerated, since the graphics engine is built over OpenGL ES 2, using a modern and fast graphics pipeline.

For brevity, the code implementation of the GUI has not been discussed in this section. Instead, a descriptive discussion is given on the features implemented on each display screen. These screens are the splash screen, the standby mode screen, and the monitor screen as described in the design section in the preceding section. However, if the project code is required, please check it out on GitHub at the repo https://github.com/Stephen-Tipa-Augustine/BlindSpot_detection.

3.6.3.1. Splash screen

The splash screen is the first view of the GUI. It is what the user sees as the system is starting as many initializations occur and these can be time-consuming. This screen creates the impression that something is happening behind scenes. The screen shows the title of the application, “Kayoola BSDS” and the purpose. It shows the copyright information and the loading percentage of the program.

3.6.3.2. Standby screen

This is the view of the GUI shown to the user when the system is not yet activated. In this mode, the firmware running the ultrasonic sensors, LEDs are deactivated. The accelerometer firmware however constantly detects for motion and if motion is detected, the system enters the Monitoring mode. From this mode, the user can go to the monitoring mode manually by pressing the system status switch on the GUI.

3.6.3.3. Monitor mode screen

In this mode, as shown in **Figure 3-10**, the intended functionalities of the system occur. The view has a large canvas on which a 2D map of a bus is placed with a dashed boundary line in green. The boundary line is 1m away from the bus, this limit is because the maximum range of the ultrasonic sensor being used is only about 4m.

When an object is detected by the ultrasonic sensor, the identity of this object is obtained from the daemon thread that runs the object detection model. The distance of the object is converted to a pixel coordinate and finally, the object is placed on the canvas along with a label describing the position of the object. The icon of the object shown corresponds to the kind detected by the model.

On the right side of the view, there is a list of cards stacked vertically that allows for interaction with the system or to provide information to the user. The first card is for the system status, from which the user can turn on and off the system manually. The second card shows the information on the category of the detected objects, alongside this category, several objects in each of them are appended in a square bracket. The third card shows the position of the detected objects and these locations can be Left or Right or Bottom or Top. The fourth card shows the total number of detected objects at the current instant. And, finally, the last card is for volume control. From this card, the user can disable the sound by toggling the speaker icon. The user can also increase and decrease the volume of the auditory feedback.

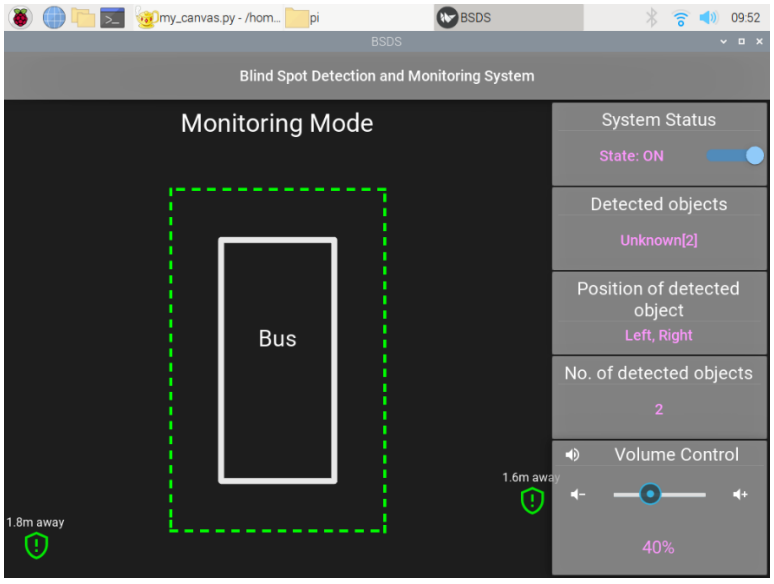


Figure 3-10: The monitoring mode screen

3.6.4.The firmware implementation

The system generally has five categories of peripherals and these are shown in **Figure 3-11.**

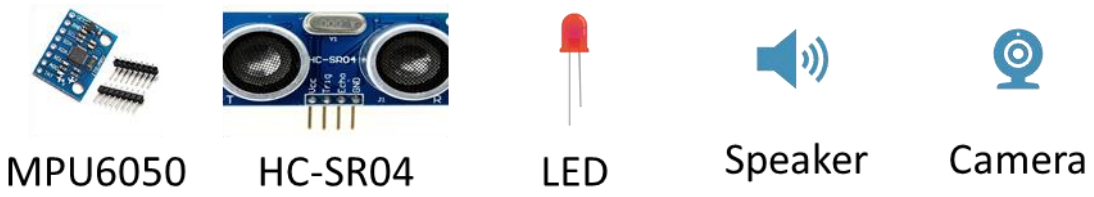


Figure 3-11: Peripherals

3.6.4.1. MPU6050

MPU6050 is a Micro Electro-mechanical system (MEMS), it consists of a three-axis accelerometer and a three-axis gyroscope. It measures velocity, orientation, acceleration, displacement, and other motion-like features. Structurally, it consists of Digital Motion Processor (DMP), which has the property to solve complex calculations. MPU6050 also consists of a 16-bit analog to digital converter hardware. Due to this feature, it captures three-dimension motion at the same time.

This module has some famous features which are easily accessible, due to its easy availability it can be used with a famous microcontroller like Arduino.

This module uses the I2C module for interfacing with Arduino and Raspberry Pi.

MPU6050 is less expensive, its main feature is that it can easily combine with an accelerometer and gyroscope.

3.6.4.1.1. Device connection

This sensor was connected to the Raspberry Pi as shown in **Table 3-5**; -

Table 3-5: MPU6050 device connections

Raspberry Pi	MPU 6050
Pin 1 (3.3V)	VCC
Pin 3 (SDA)	SDA
Pin 5 (SCL)	SCL
Pin 6 (GND)	GND

3.6.4.1.2. Business logic

For this sensor we implemented four methods, the first determines the accelerometer values in the three axes. The second determines the gyroscopes value still in the three axes. The third detect linear motion based on the accelerometer data, and the fourth detects rotational acceleration using the gyroscope data.

The accelerometer firmware runs on a daemon thread separate from the main program loop. It constantly detects to check if there's linear or rotational acceleration. It uses an "OR" operation to detect motion, using rotational and linear acceleration.

3.6.4.2. HC-SR04

At its core, the HC-SR04 Ultrasonic distance sensor consists of two ultrasonic transducers. One acts as a transmitter that converts an electrical signal into 40KHz ultrasonic sound pulses. The receiver listens for the transmitted pulses. If it receives them, it produces an output pulse whose width can be used to determine the distance the pulse traveled. As simple as pie!

The sensor is small, easy to use in any robotics project, and offers excellent non-contact range detection between 2 cm to 400 cm (that's about an inch to 13 feet) with an accuracy of 3mm. Since it operates on 5 volts, it can be hooked directly to an Arduino or any other 5V logic microcontrollers.

How it works

It all starts, when a pulse of at least 10 µS (10 microseconds) in duration is applied to the Trigger pin. In response to that, the sensor transmits a sonic burst of eight pulses at 40 kHz. This 8-pulse pattern makes the "ultrasonic signature" from the device unique, allowing the receiver to differentiate the transmitted pattern from the ambient ultrasonic noise.

The eight ultrasonic pulses travel through the air away from the transmitter. Meanwhile, the Echo pin goes HIGH to start forming the beginning of the echo-back signal.

In case, if those pulses are not reflected then the Echo signal will timeout after 38 mS (38 milliseconds) and return low. Thus a 38 mS pulse indicates no obstruction within the range of the sensor.

If those pulses are reflected the Echo pin goes low as soon as the signal is received. This produces a pulse whose width varies between 150 µS to 25 mS, depending upon the time it took for the signal to be received.

The width of the received pulse is then used to calculate the distance to the target object using the equation (3.1);

$$Distance = \frac{(speed \times time)}{2} \, , speed = 343ms^{-1} \tag{3.3}$$

3.6.4.2.1. Device connection

There were two ultrasonic sensors implemented for this application. One was to detect objects from the left side of the bus and the other detect objects from the right side of the bus. The sensors were connected to the Raspberry Pi as shown in **Table 3-6** and **Table 3-7**;

Table 3-6: HC-SR04 device connection (Left Ultrasonic sensor)

Raspberry Pi	HC-SR04
Pin 2 (5 V)	VCC
Pin 12 (GPIO 18)	TRIG

Pin 18 (GPIO 24)	ECHO (5 V)
Pin 14 (GND)	GND

Table 3-7: HC-SR04 device connection (Right Ultrasonic sensor)

Raspberry Pi	HC-SR04
Pin 4 (5 V)	VCC
Pin 11 (GPIO 17)	TRIG
Pin 13 (GPIO 27)	ECHO (5 V)
Pin 9 (GND)	GND

3.6.4.2.2. Business logic

The business logic of the ultrasonic sensor follows how the HC-SR04 sensor works, and it was implemented to use the formula stated in **Equation (3.1)**.

The computation of distance and reading the sensor data happens in a thread separate from the main loop, however, this operation is a blocking one. Blocking means that when getting the distance at a given instant of time the program waits till the thread returns a value. To avoid the user interface from freezing up, a mitigation technique was made in such a way that when the sensor fails to receive the echo sound after 25ms the system assumes no object was detected by the ultrasonic sensor.

When the distance value from the ultrasonic sensor is obtained, this value which is in centimeters is mapped to the corresponding virtual coordinate in the main loop, further detection is made to identify the object associated with that sensor and finally, the object is placed on the UI canvas using an appropriate icon (this icon corresponds to person, car, or bike).

3.6.4.3. LED

A light-emitting diode is a semiconductor light source that emits light when current flows through it. This is used for visual alerts.

3.6.4.3.1. Device connection

Two LEDs were implemented, one for the left side of the bus and the other for the right side of the bus. The LEDs were connected as shown in **Tables 3-8 and 3-9**;

Table 3-8: Left LED connection

Raspberry Pi	LED
Pin 25 (GND)	CATHODE
Pin 29 (GPIO 5)	ANODE

Table 3-9: Right LED connection

Raspberry Pi	LED
Pin 34 (GND)	CATHODE
Pin 31 (GPIO 6)	ANODE

3.6.4.3.2. Business logic

For the LED two main methods were implemented, one turns OFF the LED while the other turns it ON. Using a suitable blinking rate of 1s, this task was scheduled to turn the LED ON and OFF hence creating the feel of natural blinking. The LEDs blink only when the target is at distance less or equal to 1m and the target is on the side corresponding to the LED i.e., Left or Right.

3.6.4.4. Speaker

This is an output device used for auditory feedback, it is connected via the Aux interface and powered by a 5V DC source. Auditory feedback for this system has been implemented using the Pygame module of the Python programming language. We created a short piece of music that plays for 32s, once an object is in the danger zone this music plays. If the object moves out of the danger zone the music stops playing.

3.6.4.5. Camera

This is an input device used to capture live feeds that can then be used to detect the object's identity. The camera used here is the Raspberry Pi camera version 2.

3.6.5.Object detection model

These models recognize objects from an image, recorded video, or real-time video from a camera. The purpose of an object detection model in this system is to identify the object in the blind spot so that, the driver can make decisions accordingly. The scope of object detection has been limited to only humans, vehicles, and bikes (motorcycles and bicycles). Also, since the goal of this project was not to design a unique deep learning AI model, a huge repository of object detection models was explored. Therefore, instead of designing one, the focus of the project was to identify a suitable model that can perform the task and optimize it for the Raspbian Pi 4 Model B platform.

We chose the "SSD mobile net v1" model for this system. This is an object detection model trained on the COCO dataset. COCO stands for Common Objects in Context. COCO is large-scale object detection, segmentation, and captioning dataset. COCO has several features some of which are: Object segmentation, Recognition in context, Superpixel stuff segmentation, 330K images (>200K labeled), 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image, 250,000 people with keypoints.

This model can detect an object from a frame with a latency of order of ~500ms. The TensorFlow lite model was integrated into the system using the "tflite_runtime" library for edge devices.

3.6.5.1. Camera configuration

To be able to use the Picamera, it has to be first configured. To do this, the Raspbian OS has to be upgraded. This is done using the following commands;

- Sudo apt-get update
- Sudo apt-get dist-upgrade

The first command updates the repositories and the second command performs the upgrade.

The second step of the configuration is enabling the camera interface in the Raspberry Pi Configuration menu. This was done by clicking the Pi icon in the top left corner of the screen, selecting Preferences -> Raspberry Pi Configuration, and going to the Interfaces tab and verifying the Camera is set to Enabled. Finally, the system was rebooted for the new configurations to take effect.

3.6.5.2. Business logic

The object detection feature was implemented to be an independent daemon thread. The first task as the program loads is initializing the Picamera, loading the model into memory, and obtaining the output and input features of the model. The program then captures the most recent frame from the video stream and formats it to suit the input features and finally feeds it to the model. The output of the model is then processed based on the labels, scores, and classes to extract the labels of the detected objects. The program sleeps for 0.5s before proceeding to the next iteration.

3.6.6.Code compilation to an executable

Python compiles a Python source code (a .py file) into a byte-code with a .pyc extension. The Python interpreter automatically runs a .pyc file in preference to a .py file having the same name. it is important however to understand that, a program does not run any faster when it is read from a .pyc or .pyo file than from a .py file. The only benefit the compiled version has is the loading time which is pretty much shorter than that of the source code.

This being a complex program segmented into several modules, it would therefore be wiser to compile the project into an executable file. This primarily saves overhead incurred during compilation as the code executes. It also prevents the code from being easily altered by anyone. For this task, the Python "Pyinstaller" module was utilized. Figure 3-12 shows the configuration and command used to achieve the conversion of the .py files into an executable file. After this further configuration was made to make the program run at startup in conformance with the system requirement specifications.

3.7. Rationale and challenges

The methodology was kept code-free for brevity, however, the whole project code written in Python can be found in the git repo https://github.com/Stephen-Tipa-

Augustine/BlindSpot_detection. Generally, the project adopted the waterfall model. This breaks down the project into linear sequential activities just as was seen here. The activities are Requirement analysis, architecture description, system design, test specification, and implementation. There are other activities such as deployment and maintenance, but these have been omitted as they are out of the scope of the project.

We adopted a hybrid system since it is important to know the identity of the object. The driver can make different decisions based on the object category. The auditory feedback is to minimize driver attention and allows them to focus on other tasks, as their attention is only required objectively when the system warns. The GUI is aimed at giving the driver more details and also allowing for interactivity.

The main challenge faced during the implementation is the fact that the task was new. It is not that the author was unfamiliar with the technologies, but rather never used them for these tasks before. The automotive industry players tend to keep many of their technological principles private. This has limited the amount of data we gathered during the benchmark.

4. Results

4.1. Introduction

This chapter will give the results obtained after evaluating the test specifications given in the preceding chapter. The test specification adopted follows the systematic approach. The rationale for choosing this test paradigm is that, since embedded systems can be complex, then such a paradigm can be intuitive and save cost. This chapter is organized as follows, firstly, the unit test results are presented for the different components of the system. After that, the integration and system tests are discussed.

4.2. Unit test results

Unit testing evaluates the performance of an independent unit of the system, such as a piece of code or a hardware unit that performs a specific purpose. In this section of the results, the author dissects the entire system into functional units that are testable and then discusses the tests performed on them.

4.2.1. The Hardware unit

An embedded system is composed of both hardware and software domains. The hardware offers a platform on which the software runs, and in most cases, the hardware is controlled by the software. **Figure 3-4** shows the resultant circuit design of the hardware components. The hardware comprises of; - a Raspberry Pi 4 Model B, which is the master module and controls all other hardware components; two HC-SR04 sensors, these are ultrasonic sensors and are used for range measurements of objects in the surrounding; an accelerometer (MPU6050), this is used for motion detection; two LEDs for the visual alert.

Other components not shown in this figure are the speaker for auditory feedback and a 5V DC power source for powering the Raspberry Pi and the speaker. Resistors R1 and R2 are meant to reduce the peak current drawn by the LED to protect the GPIO pins of the Raspberry Pi from getting destroyed. Resistor pairs R3, R4, and (R5, R6) form a potential divider network that steps the voltage of 5V from the ECHO pin of the HC-SR04 sensor to a safe level to protect the GPIO pins.

The test objective here was to validate the ability of the hardware to support the embedded software. The embedded software was successfully deployed and executed on the hardware. The results of these are discussed in the following sections of this chapter.

4.2.1.1. Raspberry Pi performance

The project is using Raspberry Pi 4 Model B with the following description. It employs 8GM SDRAM and operates at a clock speed of 1.5GHz. It uses a Micro-SD card slot for loading operating systems and data storage. Operating temperature: 0 – 50 degrees C ambient. A detailed technical description of the specification of the Raspberry Pi is given in **Appendix B I**.

4.2.2. The GUI unit

The display of this system has two main views, and these are the standby mode and monitoring mode displays. In addition to these, we have the splash screen. The implementation of these

screens followed the design in the preceding chapter. The details of these designs are illustrated in the following sub-sections. Similarly, as in the preceding chapter, the author for brevity did not include codes except for the firmware core codes.

4.2.2.1. The Splash Screen

The splash screen shown in **Figure 4-2** is the first view of the GUI. It is what the user sees as the system is starting as many initializations occur and these can be time-consuming. This screen creates the impression that something is happening behind scenes. The figure shows the result of running the GUI program and thus the outcome of the implementation using the Kivy Python GUI framework. This code was running on the Raspbian OS.

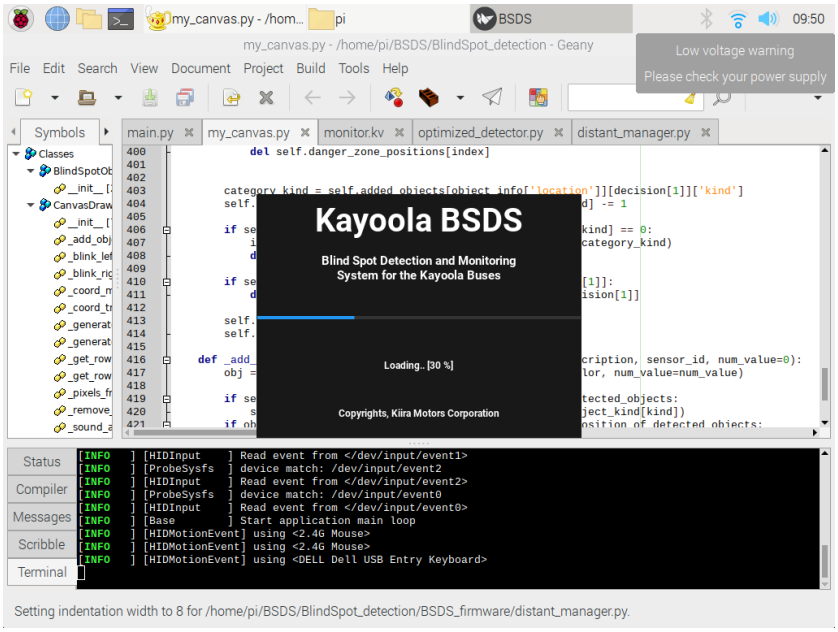


Figure 4-1: Splash screen

4.2.2.2. The Standby Screen

After the program loads fully, the splash screen switches to standby mode. The view is shown in **Figure 4-3**. This is the view of the GUI shown to the user when the system is not yet activated. In this mode, the firmware running the ultrasonic sensors, LEDs are deactivated. The accelerometer firmware however constantly detects for motion and if motion is detected, the system enters the Monitoring mode. From this mode, the user can go to the monitoring mode manually by pressing the system status switch on the GUI.

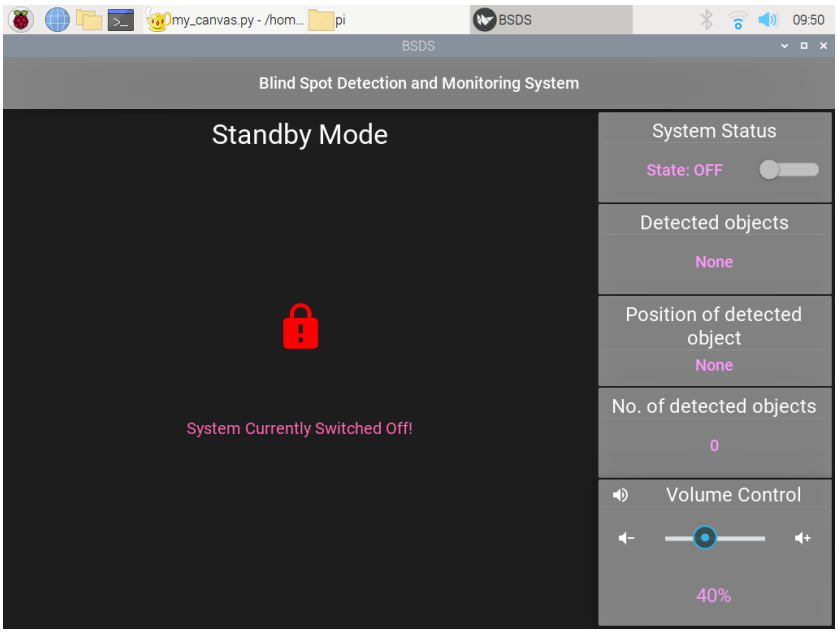


Figure 4-2: Standby mode

4.2.2.3. The Monitoring mode screen

Once the system is activated manually or automatically by the motion sensor, the view shifts to the monitoring mode. In this mode, the intended functionalities of the system occur. The view is shown in the following **Figures 4-4 and 4-5**.

The view has a large canvas on which a 2D map of a bus is placed with a dashed boundary line in green. The boundary line is 1m away from the bus, this limit is because the maximum range of the ultrasonic sensor being used is only about 4m.

When an object is detected by the ultrasonic sensor, the identity of this object is obtained from the daemon thread that runs the object detection model. The distance of the object is converted to a pixel coordinate and finally, the object is placed on the canvas along with a label describing the position of the object. This can be seen from the following figures. The icon of the object shown corresponds to the kind detected by the model.

On the right side of the view, there is a list of cards stacked vertically that allows for interaction with the system or to provide information to the user. The first card is for the system status, from which the user can turn on and off the system manually. The second card shows the information on the category of the detected objects, alongside this category, several objects in each of them is appended in a square bracket. The third card shows the position of the detected objects and these locations can be Left or Right or Bottom or Top. The fourth card shows the total number of detected objects at the current instant. And, finally, the last card is for volume control. From this card, the user can disable the sound by toggling the speaker icon. The user can also increase and decrease the volume of the auditory feedback.

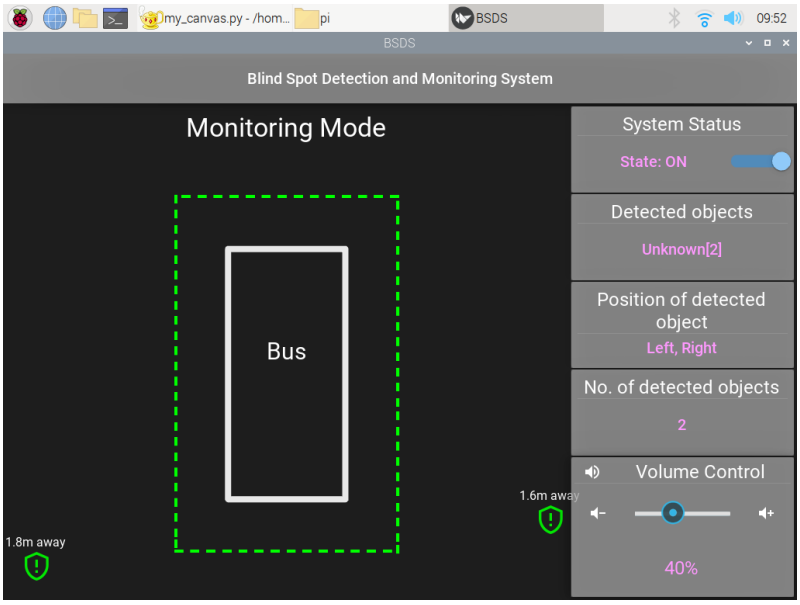


Figure 4-3: When the system is on and activated

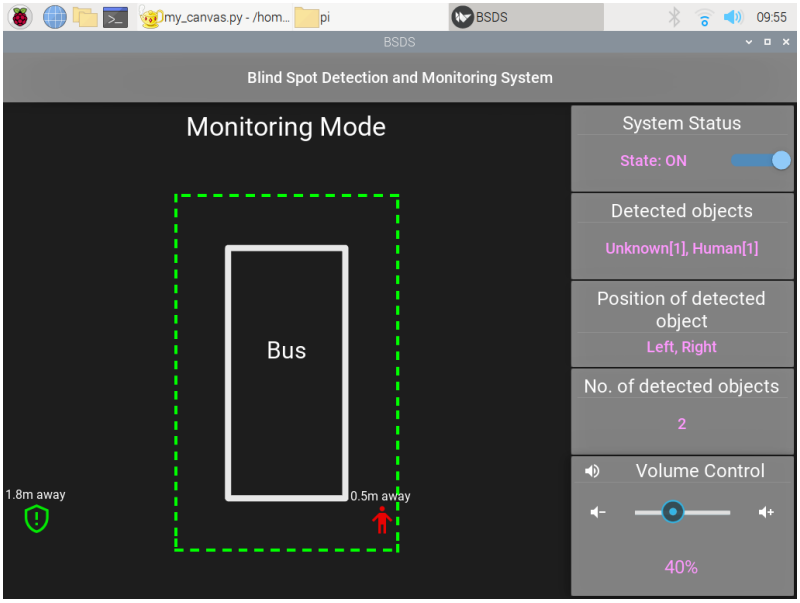


Figure 4-4: System detected human on the right and unknown object on the left

4.2.3. The Accelerometer firmware

This is the motion sensor, it consists of a 3-axis accelerometer, 3-axis gyroscope, and a temperature sensor. The firmware was implemented in Python using the “mpu6050” library as shown in **Figure 4-6**;

Different methods were implemented to obtain temperature, accelerometer, and gyroscope data. The “get_all_data” method returns the values of all these three quantities at once, unlike the “get_accelerometer_data” or “get_gyroscope_data” or “get_temperature”. The “__init__” method is an instance initialization method. The “vehicle_moving” and “vehicle_rotating”

methods detect whether the vehicle is moving or rotating based on the accelerometer or gyroscope respectively. The state of the vehicle as to whether it's in motion or not is defined by the "detect_state" method.

```
class Accelerometer:

    def __init__(self, **kwargs):
        self.sensor = mpu6050(0x68)
        self.running = True
        self.accel_data = self.get_accelerometer_data()
        self.gyro_data = self.get_gyroscope_data()
        self.moving = False
        self.rotating = False
        # threading.Thread(target=self.run()).start()

    def get_accelerometer_data(self):
        return self.sensor.get_accel_data()

    def get_gyroscope_data(self):
        return self.sensor.get_gyro_data()

    def get_temperature(self):
        return self.sensor.get_temp()

    def get_all_data(self):
        return self.sensor.get_all_data()

    def detect_state(self, data, state_kind='accel'):
        current_values = self.get_accelerometer_data() if state_kind == 'accel' else
self.get gyroscope data()
        difference = {'x': abs(current_values['x'] - data['x']),
                      'y': abs(current_values['y'] - data['y']),
                      'z': abs(current_values['z'] - data['z'])}
        changed_axis = 0
        for i in difference:
            if difference[i] > 2:
                changed_axis += 1
        return False if changed_axis < 1 else True

    def vehicle_moving(self):
        self.moving = self.detect_state(self.accel_data) if not self.moving else True
        self.accel_data = self.get_accelerometer_data()

    def vehicle_rotating(self):
        self.rotating = self.detect_state(self.gyro_data, state_kind='gyro') if not
self.rotating else True
        self.gyro_data = self.get_gyroscope_data()

    def run(self):
        while self.running:
            self.vehicle_moving()
            self.vehicle_rotating()
            time.sleep(5)

if __name__ == "__main__":
    obj = Accelerometer()
    obj.run()
```

Figure 4-5: MPU6050 firmware

When the above code was executed, the result in **Figure 4-7** was obtained. In this test scenario, the author was just interested in knowing if the sensor works. He was printing the values of the accelerometer data, gyroscope data, and temperature in the console. The outcome of the execution was as expected and can be seen from **Figure 4-7**.

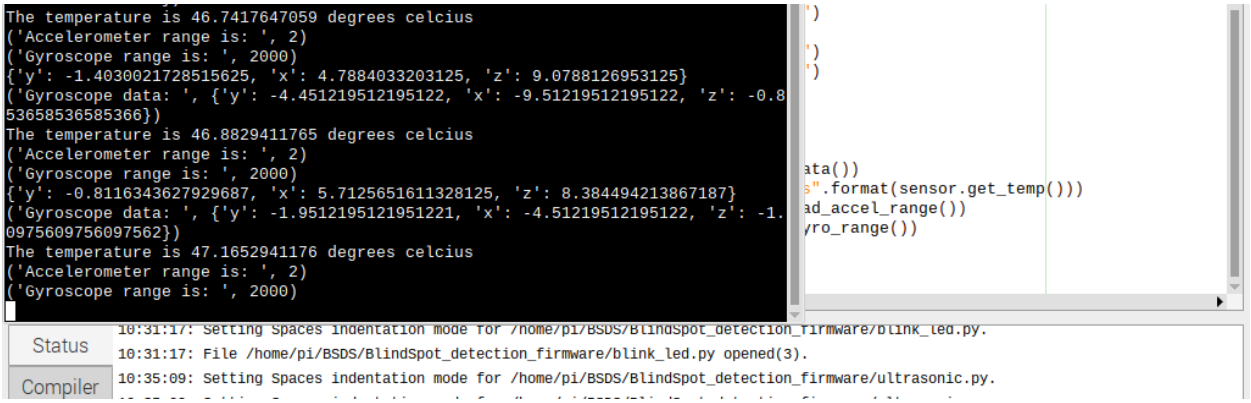


Figure 4-6: MPU6050 unit test result

4.2.4. The Ultrasonic sensor firmware

Here the HC-SR04 sensor was used, and its purpose was to measure range. With the base class in **Figure 4-8**, the user can measure the distance of objects in the vicinity of the sensor within a sweep angle of <15º. The maximum range is about 4m.

The sensor is initialized during the instantiation of the class. The “compute_distance” method deduces the distance by measuring the time it takes for the ultrasonic sound to travel to the target and back to the sensor. The “cleanup” method cleans up the associated GPIO pins once the process is terminated.

The “run” method invokes the method to compute the range and this invocation happens in a separate thread.

When the above code was executed on the Raspbian OS, the result in Figure 4-9 was obtained. This test aimed to find out if the sensor can determine the range of targets in its vicinity. It can be seen from the test result that; it prints the results of the detected object on the console as expected. The sensor, therefore, functions as expected.

```
class UltrasonicSensor:

    def __init__(self, trigger, echo, **kwargs):
        # GPIO Mode (BOARD / BCM)
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        # set GPIO Pins
        self.GPIO_TRIGGER = trigger
        self.GPIO_ECHO = echo
        # set GPIO direction (IN / OUT)
        GPIO.setup(self.GPIO_TRIGGER, GPIO.OUT)
        GPIO.setup(self.GPIO_ECHO, GPIO.IN)

        self.running = True

    def compute_distance(self):
        # set Trigger to HIGH
        GPIO.output(self.GPIO_TRIGGER, True)
        # set Trigger after 0.01ms to LOW
        time.sleep(0.00001)
        GPIO.output(self.GPIO_TRIGGER, False)
        start_time = time.time()
        stop_time = time.time()
        # save StartTime
        while GPIO.input(self.GPIO_ECHO) == 0:
            start_time = time.time()
        # save time of arrival
        while GPIO.input(self.GPIO_ECHO) == 1:
            stop_time = time.time()
            if stop_time - start_time > .025:
                return None
        # time difference between start and arrival
        t = stop_time - start_time
        return float("%.1f" % ((t * 34300) / 2))

    @staticmethod
    def clean_up():
        GPIO.cleanup()

    def run(self):
        m = ThreadManager()
        t = threading.Thread(target=lambda q:
q.put(self.compute_distance()), args=(m.que, ))
        t.start()
        m.add_thread(t)
        m.join_threads()
        distance = m.check_for_return_value()
        # print("Measured Distance = %f cm" % distance)
        return distance

    def stop(self):
        self.running = False
        self.clean_up()

if __name__ == '__main__':
    obj = UltrasonicSensor()
    obj.run()
```

Figure 4-7: HC-SR04 firmware

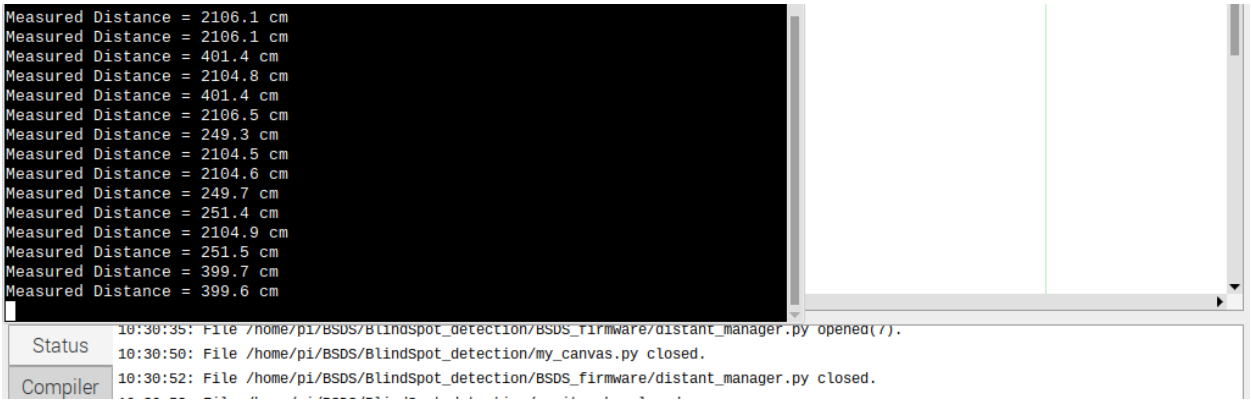


Figure 4-8: HC-SR04 unit test result

4.2.5. The LED firmware

For this implementation, the Raspberry Pi GPIO library was used. The LED has two pins with the longest being the anode. Implementation followed the design and pin configuration in the preceding chapter.

```
class BlinkLED:
    def __init__(self, anode, **kwargs):
        # GPIO Mode (BOARD / BCM)
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        # set GPIO Pins
        self.GPIO_LED = anode
        # set GPIO direction (IN / OUT)
        GPIO.setup(self.GPIO_LED, GPIO.OUT)
        self.state = False
        self.blinking = True

    def turn_off(self):
        GPIO.output(self.GPIO_LED, GPIO.LOW)
        self.state = False

    def turn_on(self):
        GPIO.output(self.GPIO_LED, GPIO.HIGH)
        self.state = True

    @staticmethod
    def clean_up():
        GPIO.cleanup()

    def run(self, turn_off=False):
        if self.state or turn_off:
            self.turn_off()
        else:
            self.turn_on()

    def stop(self):
        self.state = False
        self.blinking = False
        self.turn_off()
        self.clean_up()

if __name__ == '__main__':
    obj = BlinkLED()
    obj.run()
```

Figure 4-9: LED firmware

Figure 4-10 shows the firmware implementation for the LED. Instantiation of the class initializes the LED. During this step, the class is invoked with the pin that represents the anode. The "turn_on" and "turn_off" methods are used to turn the LED on and off respectively. The "cleanup" methods clean the GPIO pins. The "run" meth alternately turns on the LED ON and OFF.

When the code in **Figure 4-10** was executed, the two LEDs employed were able to blink as expected. This is shown in **Figure 4-11**.

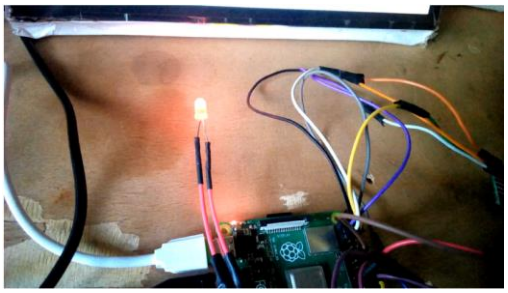


Figure 4-10: LED firmware unit test result

4.2.6. The Auditory feedback

The Auditory feedback was implemented using the pygame module of the Python programming language. A short piece of music that plays for 32s was created, once an object is in the danger zone this music plays. If the object moves out of the danger zone the music stops playing. The test objective here was to show that the system gives auditory feedback as in the requirement specification, and it performed as required. **Figure 4-12** shows the speaker employed for playing the sound.

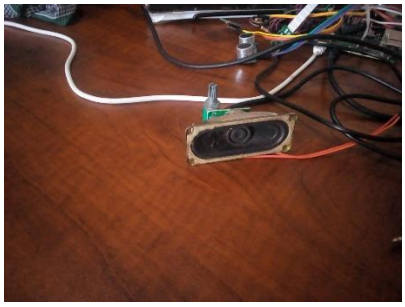


Figure 4-11: Speaker for auditory feedback

4.2.7. The Object detection model

As stated in chapter 3, the implementation of this model was not part of this project. The task the author performed was rather identifying a model that can identify objects and limiting it only to identifying humans, vehicles, bicycles, and motorcycles. The model is a TensorFlow lite model when deployed on the Raspberry pi and was executed, it was able to identify the required objects. As shown in Figure 4-13.

However, it was also required that the scene from the camera is not shown to the user, but rather the information about the kind of the object. This information should be presented by the icon

shown on the GUI display. For this purpose, the author altered the code giving the result in Figure 4-13 to one giving only the labels in text form.

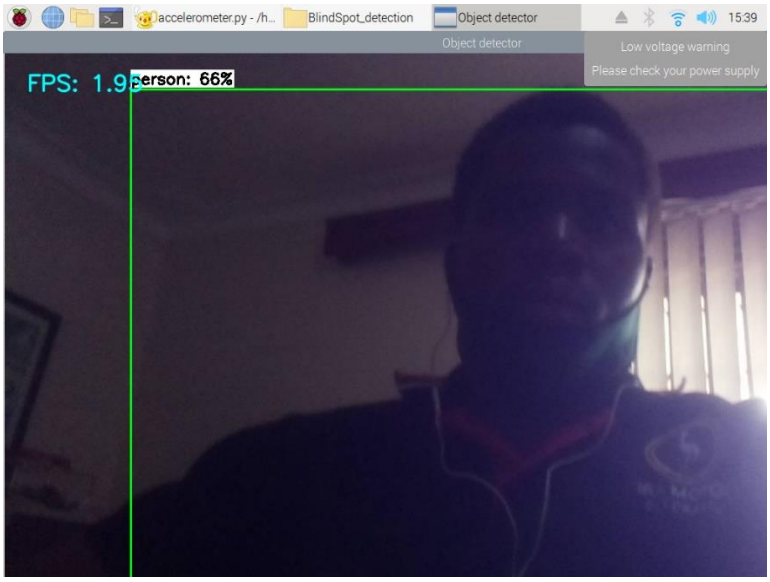


Figure 4-12: Test on the Object detection model

As stated, the model by default can detect several objects and the label is drawn on the frame. To achieve the objective of only changing the icon used to represent the target, we need to process the detector output based on the code in **Figure 4-14** as follows. We loop through all detections and look up the name of the detected objects. When the detected object is not among the required detectable objects discarded otherwise, the name is appended to the resultant list.

```
# Loop over all detections and retrieve label if confidence is above minimum threshold
display_str = []
for i in range(len(scores)):
    if (scores[i] > min_conf_threshold) and (scores[i] <= 1.0):
        # Draw label
        object_name = labels[int(classes[i])]  # Look up object name from "labels" array using
class index
        if object_name in ('bus', 'truck', 'car'):
            label = 'car'
            display_str.append(label)
        elif object_name in ('bicycle', 'motorcycle'):
            label = 'motorbike'
            display_str.append(label)
        elif object_name == 'person':
            label = 'human-handsdown'
            display_str.append(label)

if q:
    q.queue.clear()
    q.put(display_str)
time.sleep(.5)
```

Figure 4-13: Detection processing code

After the above modification, the output of the code is the name of the icon corresponding to the detected target objects. **Figure 4-15** shows the result of this modification. In **Figure 4-15**, 'human-handsdown' is the icon used to represent human objects. When there is no object in the

view of the camera or the objects are those not in the list of detectable objects, the detector returns an empty list.

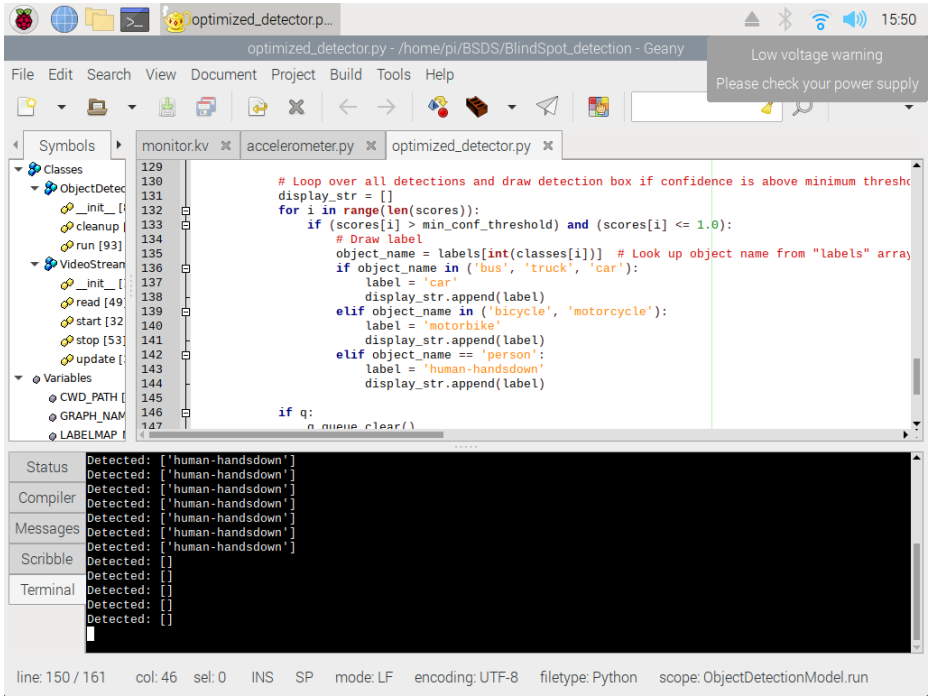


Figure 4-14: Detector output after post-processing

4.3. Integration and System test results

Integration testing evaluates the satisfaction of how a unit fits into the larger system, and the system testing checks to see how all units fit together to meet the system mission statement. In this section of the test results, the author inspects the system as a whole and discusses how it meets its functional requirements.

Figure 4-15 shows the fully integrated unit. The components are labeled as shown in the Figure.

1) This is the Display screen, it's a touch screen
2) This is the righthand side ultrasonic sensor
3) This is the righthand side camera
4) This is the righthand side LED
5) This is the casing that encloses the core components. It is designed with many passages for free movement of air to avoid overheating as no fan is mounted on the microcontroller.

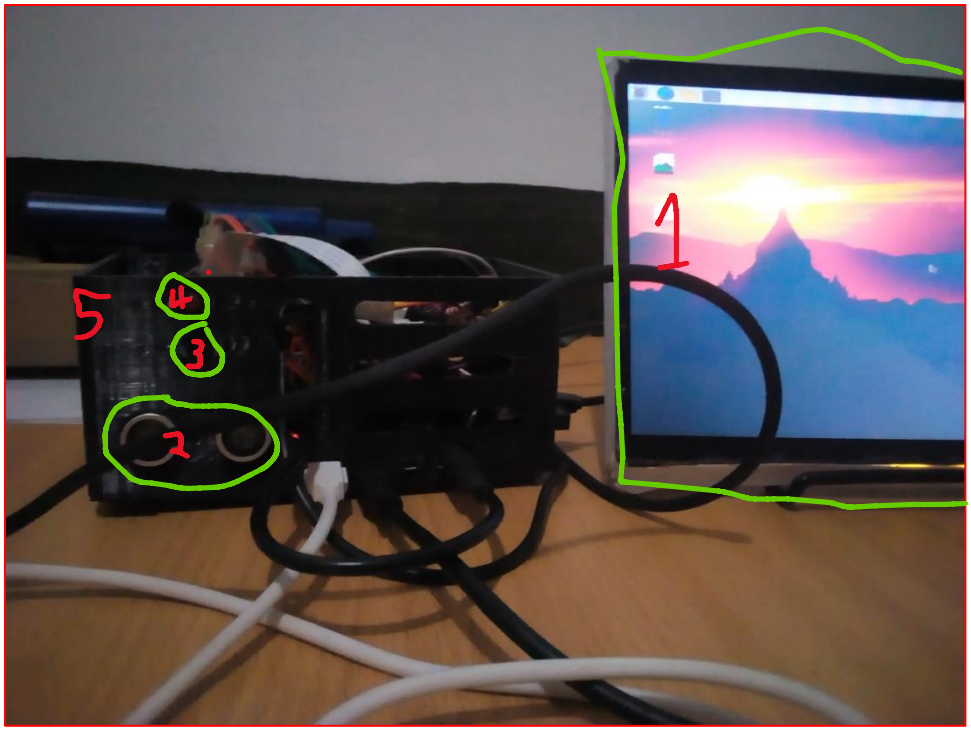


Figure 4-15: Fully integrated system

The system test was accomplished using a Requirement Traceability Matrix in **Table 4-1**. The table presents a summary of the test conducted on the system in conformance with the system requirements. This matrix typically relates the system requirements to the completed tasks. This, therefore, presents a check for the planned workload of the entire project. Each requirement has a unique ID, the "REQF" and "REQN" represents the functional and non-functional requirements respectively. All requirements have been ranked as either essential or conditional. The essential requirements are critical and must be fulfilled to realize the system in question. The Conditional requirements are not very critical but are required to be fulfilled for completeness of the system mission. Further, it can be seen that all requirements have been fulfilled based on the project scope defined in Chapter 1 of this document.

Table 4-1: Requirement Traceability Matrix

SN	Sub-System	ID	Requirements Description	Ranking	Status
1.	**Software**	**REQF001**	Shall provide a visual display of the location and distance of the objects in the blind spots of the bus in real-time.	Essential	Done
		REQF002	Shall be capable of reading raw sensor values.	Essential	Done
		REQF003	Shall process raw sensor values into formats suitable for decision making as well as formats that can be interpreted by the user.	Essential	Done
		REQF004	Shall be capable of initiating the blinking of LEDs when an object in the blind spot surpasses the defined threshold distance value [1m].	Essential	Done
		REQF005	Shall be capable of initiating auditory feedback when the distance between the bus and object gets smaller	Essential	Done

			than the threshold [1m].		
		REQF006	The intensity of the auditory feedback shall increase when the target object gets closer to the body of the bus.	Essential	Done
		REQF007	Shall start on system start-up.	Essential	Done
		REQF008	Shall be activated when the motion of the bus has been detected.	Conditional	Done
		REQF009	Shall seamlessly interact with the hardware sub-systems which include; - the Raspberry Pi and the peripheral devices.	Essential	Done
		REQT001	Shall not fail on regular purposes (due to failure to input values from the sensors), only at extreme bugs.	Conditional	Done
		REQT002	Shall withstand component and environmental failures.	Conditional	Done
		REQT003:	The functions of the software shall be easily understood by the user (the driver).	Conditional	Done
		REQT004	The response for sensor inputs shall be relatively low [1s].	Essential	Done
		REQT005	Shall use relatively optimum system resources, such as memory, CPU, and disk.	Conditional	Done
		REQT006	Shall identify the root cause of failure when it occurs.	Conditional	Done
		REQT007	Shall be easily tested for any desired features.	Essential	Done
		REQT008	Shall be readily installable on the Raspberry Pi	Essential	Done
		REQT009	Shall conform to the Linux OS of the Raspberry Pi.	Essential	Done
		REQT010	There shall be ease in the replacement of the different software components at any desired time.	Conditional	Done
		REQT011	Shall require minimum attention of the user (i.e., the driver does not need to continuously glance at the display) so they can focus on other tasks.	Conditional	Done
		REQT012	Shall present a user interface that is slick, intuitive, and attractive.	Conditional	Done
		REQT013	Shall notify the user in the event that the system fails.	Essential	Done
2.	**Hardware**	**REQF010**	The Ultrasonic Sensors shall provide an accurate measurement of the range distance to the target (object in the blind spot) within different environment variations (for example temperature, humidity, and background noise).	Essential	Done
		REQF012	The accelerometer shall be able to measure the presence or absence of a motion to provide system power on, off, or sleep mode regardless of the different environment variations (for example temperature and background noise).	Essential	Done
		REQF013	The control unit (i.e., the Raspberry Pi) shall handle fast calculations and computations from the sensors and deduce a given set of instructions corresponding to the sensor values	Essential	Done
		REQF014	The Camera shall capture frames from the blind spot areas that shall be fed to the object detection model.	Essential	Done
		REQF015	The LEDs shall illuminate at the start of the bus to show that they are in proper working conditions. They shall blink when there is a body in proximity with the body of the bus.	Essential	Done
		REQF016	The Speaker shall produce an alarm when an object or vehicle is near the body of the bus.	Essential	Done
		REQT014	When an unpredictable failure occurs in reading	Essential	Done

			values from either the accelerometer or the ultrasonic sensor, the system shall recover briefly to full capacity or safe mode respectively.		
		REQT015	The system shall be able to handle many inputs from its environment.	Essential	Done
		REQT016	The different components shall be enclosed in a plastic casing printed by a 3D printer to keep the connections firm as well as protect the electronic components from mechanical damage.	Essential	Done

5. Discussion

5.1. General discussion

Of late many automotive companies are focusing on designing vehicles that not only meet user demands and price but are also safe for usage. The safety of users has become one primary drive for innovation nowadays in this industry. Kiira Motors Corporation is in the course of building an indigenous automotive industry that's so focused on localization of the value chain. Companies like Toyota, Ford, etc. already have existing systems built for blindspot detection. However, these systems are mainly targeting luxurious vehicles and at expensive rates typically 10 to 40 million Ugandan Shillings, yet the goal currently in Uganda is to build 1030 Kayoola buses. These buses shall be deployed in the country, but one major problem is that they do not have Blindspot monitoring systems onboard. The traffic conditions, as well as the road conditions in Uganda, are a little bit different from that of the foreign countries, this demands a customized solution that has both vision and non-vision-based capabilities. This project, therefore, sought a solution that fits in the indigenous automotive industry's mission of localization, is cheaper, and takes into account the local road and traffic conditions of the country. The vision feature enables the system to identify the detected object so that, drivers can make an educated decision based on the kind of object identified.

In this project, an embedded system solution was proposed for the above argument. The core features of this system were a Graphical User Interface (GUI), a visual alert via LEDs (Light Emitting Diodes), and auditory feedback. Unlike most research conducted in this field which is either vision-based or non-vision-based, this project is a hybrid version that harnesses the benefit of both types. It employed ultrasonic sensors for range detection of objects, and typically the HC-SR04 sensor. Object identification was done with the aid of a Raspbian version 2 camera and a deep learning model trained on the COCO dataset. The system is activated manually by the user or automatically by an accelerometer (MPU6050) on detection of motion. As a fact, no particular work has been done with an arrangement and architecture like that of the project in question yet the concept proved feasible. The project has satisfactorily shown that a hybrid system can be built using ultrasonic sensors for range measurement and a camera for vision. Two forms of alerting systems were also realized, one being the visual via LEDs and the other being auditory via speakers.

Other than the feasibility of the project concept discussed in the above paragraph, the project also pointed out the HC-SR04 ultrasonic sensor is not feasible for blindspot detection due to poor precision and low range despite being cheap and simple to work with. The camera employed also proved not feasible to work in the operational environment of the vehicle. Both of these sensors are good for prototyping on the bench, but not even for in-vehicle testing. For instance, the maximum range of the ultrasonic sensor is 4m, which is not practical for blindspot detection systems since these systems work on highways where they are to detect objects a couple of meters away. This is basically because on highways vehicles move at a relatively high speed. A very important finding that this project pointed out was that the best performance of deep learning models can be achieved on the Raspberry Pi model 4 B by converting them to

TensorFlow lite format and invoking them using the “tflite_runtime” module. This way the model works faster compared to the ordinary TensorFlow format.

There have been a few limitations during the execution of this project, to some extent has affected the deductions made. Firstly, the Raspberry Pi has only one camera slot which makes it hard to utilize multiple cameras and hence draw a generic conclusion. Only one camera was utilized, and this was associated with one ultrasonic sensor. The microprocessor supports Network cameras that communicate via IP (Internet Protocol); however, this presents unacceptable latency which renders it unfit for the real-time flow of the system. Secondly, the ultrasonic sensor used had a limited range of 4m as stated before. This has limited the ability to deduce concrete conclusions such as the maximum speed for which a driver can be permitted to drive and still react to an alert within the acceptable delay.

5.2. Conclusions

A blind spot is an area of the road outside the driver's field of vision that cannot be seen in the rearview mirrors or through the windows [33]. Blindspot contributes to the major road accident causes. Nowadays automotive manufacturers are focusing on active as well passive safety features to curb accidents. Blindspot detection systems fall under the active safety feature category and when well-designed can reduce road accidents to a considerable level.

To reduce blind spot-related road accidents, a solution was proposed in this project where a Raspberry Pi model 4 B was used as the master module. This controls the entire operation of the system. The system also utilized two ultrasonic sensors for range measurements, a camera for object identity, a speaker for auditory feedback, and two LEDs for the visual alert. A deep learning model trained on the COCO dataset was employed for the object identification with its input being the feed from the camera. Below, the author highlights the key outcomes and findings of this project.

- This project has shown that the idea of a hybrid system that employs both vision-based and non-vision-based features is feasible.
- The project has also shown that it is possible to build a functional GUI for an automotive embedded system using the Kivy Python framework.
- This project realized three alert features, auditory feedback via the speaker and visual alerts via the LEDs and GUI.
- The HC-SR04 ultrasonic sensor is not well suited for range measurement when it comes to the application of a blind spot monitoring system.
- The Raspbian version 2 camera is not well suited to operate in the deployment environment of the vehicle.
- The “SSD mobile net v1” pre-trained object detection model can suffice for the task of object identification in the system, hence no need for retraining the model.
- The project has provided a tested functional architecture for a hybrid Blindspot detection and monitoring system for public transport that can be deployed on the Raspberry Pi and similar microprocessors.

5.3. Impact of the study

The project has satisfactorily shown that a hybrid blind spot detection system is feasible. This indeed has added on to the base of knowledge, because even though the literature has prior work that consolidates the aspect of vision-based and non-vision-based systems no clear conclusions have been made. Matter of fact, the hybrid system can be built to provide a robust system that will outperform the vision-based as well as non-vision-based systems individually.

5.4. Recommendations

In the future, if work is to be done in this area, or an improvement of this project is to be made then, the participants may have to choose superior sensor technologies like the corner radar and Houston radar. This provides the maximum range of about 36m and 100m respectively. Ultrasonic sensors would not be a good choice for high-speed applications due to their low range and being less robust. They will have to also identify better cameras and probably a better microcontroller other than the Raspberry pi. Since the Raspberry Pi has only one camera slot. A better camera choice would be the USB automotive camera of about 720p resolution, this does not only provide superior image quality but also allows for multiple cameras to be connected to the Raspberry Pi to cover most regions of the bus.

6. Further Work

To think that this project is ready and deployable on the Kayoola buses, would be quite an illusion. Many tasks need to be done for the commercialization of this project and hence, deployment onto the buses. The output of this project is pretty much a proof of concept from which a functional prototype can be built. Below, the author discussed some important aspects that require attention for the full accomplishment of this project.

6.1. Identification of blind spot regions

A blind spot is already defined in chapter 1 as an area of the road outside the driver's field of vision that cannot be seen in the rearview mirrors or through the windows [33]. The taller and longer the vehicle, the bigger the blind spots. All types of vehicles feature pillars that create blind spots. It is therefore very important to identify the blind spot region around the bus precisely so that, the sensors can be mounted at the exact locations. This shall minimize wastage as only the required number of sensors have to be employed.

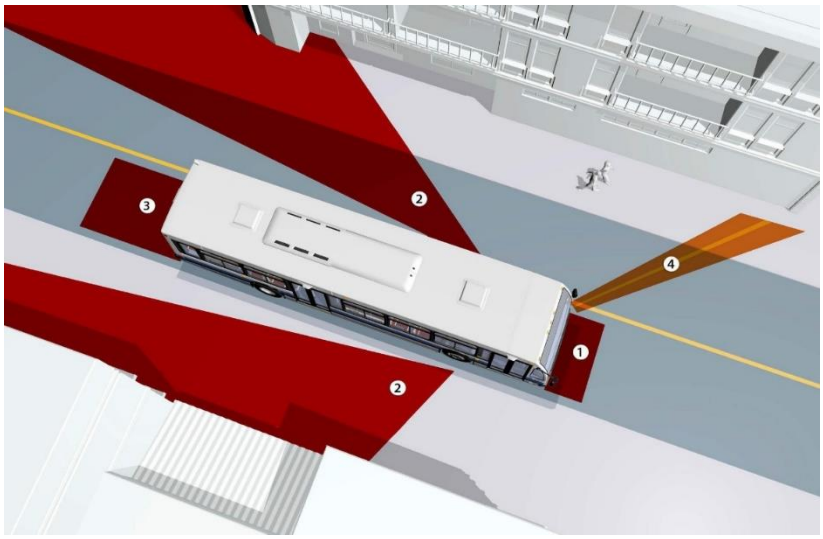


Figure 6-1: Blindspot regions around a city bus adopted from [33]

Figure 7-1 is a snapshot of the blind spot areas that need identification and these regions are; -

- ✓ In front: the front blind spot is small because city buses have flat fronts
- ✓ On the sides: because of the bus's large size, its side blind spots cover a significant area even with properly adjusted rearview mirrors.
- ✓ At the rear: even though the rear window makes the rear blind spot shorter, it still calls for attention.
- ✓ In front, on the left-hand side: the wider the pillar, the bigger the blind spot.

6.2. Integration of CAN communication protocol

Controller Area Network (CAN) is a serial network technology that was originally designed for the automotive industry, especially for European cars, but has also become a popular bus in industrial automation as well as other applications [34]. Today all embedded systems in

automobiles communicate via this protocol as it simplifies the network topology and is the industry standard. This project currently is a standalone system, but to integrate it into the bus we will need to configure it such that, the different components and sensors communicate to the master module via the CAN bus.

6.3. Packaging for deployment

This may be considered the final phase of embedded system development. An embedded system comprises processors, power supplies, I/O devices, connecting cables, and sensors of various categories. All these needs proper packaging for them to function as desired in the operational environment. This requires a throughout the study of the operational environment and the operational as well as deployment architectures. The main goals of packaging are offering mechanical protection, cooling features, safety, and capabilities for mobility.

6.4. Sensor selections

Typically, the ultrasonic sensors and the camera used in this project were well suited for demonstrating the concept but not good for deployment onto the bus. For instance, the ultrasonic sensor has a maximum range of 4m. This implies that, when the bus is moving at a speed of 4m/s and the system detects an object, then the driver's action has to be within 1s otherwise accident will occur. This is not practical and calls for the sensor of longer-range or even a sensor of another technology such as Radar systems. The rationale for the choice of the HC-SR04 was their simplicity and low cost. In the same manner, the camera employed has poor visibility and is not well suited for commercial deployment on the Kayoola buses.

6.5. Software licensing

A software license is a document that provides legally binding guidelines for the use and distribution of software. Kivy is a free open-source framework distributed under the MIT license. The source code written for this project may not require including licensing or copyright information. However, when binaries are created Kivy includes dependencies which may be the work of others. These dependencies may, therefore, need licensing. Extra effort is required in licensing the embedded software for commercialization.

6.6. In-vehicle testing

The main testing conducted for this system was the unit test, integration testing, and system testing which was primarily bench-test. After addressing the above concerns discussed in the chapter, the system will have to be tested on the bus and the road.

References

[1] K. E. Y. Words, "Road traffic incidents in Uganda: A systematic review study of five years trend," *J. Inj. Violence Res.*, vol. 9, no. 1, pp. 17–25, 2017, doi: 10.5249/jivr.v9i1.796.

[2] "Road Safety in Uganda | Traffic accidents, crash, fatalities & injury statistics | GRSF." https://www.roadsafetyfacility.org/country/uganda (accessed Dec. 14, 2021).

[3] C. N. Andrews, O. C. Kobusingye, and R. Lett, "Road traffic accident injuries in Kampala.," *East Afr. Med. J.*, vol. 76, no. 4, pp. 189–194, Apr. 1999.

[4] B. B. Smith, E. G. Pearson, and J. Leon, "Evaluation of normal triiodothyronine and tetraiodothyronine concentrations in llamas (Lama glama).," *Am. J. Vet. Res.*, vol. 50, no. 8, pp. 1215–1219, 1989.

[5] "How to Avoid Blind Spot Collisions | Berke Law Firm." https://www.nationwidedisabilityrepresentatives.com/avoid-blind-spot-collisions/ (accessed Dec. 14, 2021).

[6] "Road Safety." https://ec.europa.eu/transport/road_safety/index_en?wt-search=yes (accessed Dec. 14, 2021).

[7] V. R. Virgilio G., H. Sossa, and E. Zamora, "Vision-Based Blind Spot Warning System by Deep Neural Networks," in *Pattern Recognition*, 2020, pp. 185–194.

[8] "Blind spot detection for heavy commercial vehicles." https://www.bosch-mobility-solutions.com/en/solutions/assistance-systems/blind-spot-detection-cv/ (accessed Dec. 14, 2021).

[9] V. R. V. G. H. S. E. Zamora, "No Title," 2020.

[10] "Vision 2040 and National Development Plans – Local Development Partners' Group (LDPG)." https://www.ldpg.or.ug/vision-2040-and-national-development-plans/ (accessed Nov. 29, 2021).

[11] P. Lightweight and N. Network, "Camera-Based Blind Spot Detection with a General Purpose Lightweight Neural Network," 2019, doi: 10.3390/electronics8020233.

[12] F. Jiménez, N. José E., G. Oscar, and A. José J., "Forward Collision and Overtaking Detection †," pp. 1–19, 2020, doi: 10.3390/s20185139.

[13] N. Blanc, B. Steux, and T. Hinz, "LaRA SideCam: A fast and robust vision-based blindspot detection system," *IEEE Intell. Veh. Symp. Proc.*, pp. 480–485, 2007, doi: 10.1109/ivs.2007.4290161.

[14] D. Kwon, S. Park, S. Baek, R. K. Malaiya, G. Yoon, and J. Ryu, "A Study on Development of the Blind Spot Detection System for the IoT-Based Smart Connected Car," pp. 9–12, 2018.

[15] D. Kwon, R. Malaiya, G. Yoon, J. Ryu, and S. Pi, "applied sciences A Study on Development of the Camera-Based Blind Spot Detection System Using the Deep Learning

Methodology," no. October 2017, 2019, doi: 10.3390/app9142941.

[16] N. De Raeve, M. De Schepper, J. Verhaevert, and P. Van Torre, "A Bluetooth-Low-Energy-Based Detection and Warning System for Vulnerable Road Users in the Blind Spot of Vehicles," 2020, doi: 10.3390/s20092727.

[17] G. Liu, L. Wang, and S. Zou, "A radar-based blind spot detection and warning system for driver assistance," *Proc. 2017 IEEE 2nd Adv. Inf. Technol. Electron. Autom. Control Conf. IAEAC 2017*, pp. 2204–2208, 2017, doi: 10.1109/IAEAC.2017.8054409.

[18] J. Katarzyna, K. Maciej, and S. Wojciech, "ADVANCED DRIVER SAFETY SUPPORT SYSTEMS FOR THE URBAN," vol. 10, no. 4, 2015, doi: 10.21307/tp-2015-055.

[19] P. Pyykonen, A. Virtanen, and A. Kyytinen, "Developing intelligent blind spot detection system for Heavy Goods Vehicles," pp. 293–298, 2015.

[20] "Car Safety Features Explained - Which?" https://www.which.co.uk/reviews/new-and-used-cars/article/car-safety-features-explained-aUHSQ6N9iDKr (accessed Dec. 01, 2021).

[21] "Active safety systems: what are they and how do they work? | RoadSafetyFacts.eu." https://roadsafetyfacts.eu/active-safety-systems-what-are-they-and-how-do-they-work/ (accessed Dec. 01, 2021).

[22] "How Sensor Technology Will Shape the Cars of the Future #Melexis." https://www.melexis.com/en/tech-talks/how-sensor-technology-shape-cars-future (accessed Dec. 01, 2021).

[23] "What is an Ultrasonic Sensor? | FierceElectronics." https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor (accessed Dec. 01, 2021).

[24] "Active Safety Sensors : Automotive Safety Council." https://www.automotivesafetycouncil.org/safety-technologies/electronic-sensors/active-safety-sensors/ (accessed Dec. 01, 2021).

[25] "What is a Graphical User Interface? Definition and FAQs | OmniSci." https://www.omnisci.com/technical-glossary/graphical-user-interface (accessed Dec. 01, 2021).

[26] "Artificial Intelligence (AI) Definition." https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp (accessed Sep. 03, 2020).

[27] P. V Ajitha and A. Nagra, "An Overview of Artificial Intelligence in Automobile Industry–A Case Study on Tesla Cars," *Solid State Technol.*, vol. 64, no. 2, pp. 503–512, 2021.

[28] "What is Artificial Intelligence (AI)? - AI Definition and How it Works." https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence (accessed Dec. 02, 2021).

[29] "(1) New Messages!" https://www.hpe.com/in/en/what-is/artificial-intelligence.html?jumpid=ps_zsiufvnm5b_aid-

520042864&ef_id=Cj0KCQiA15yNBhDTARIsAGnwe0XLVVGyAZ_jcmdQu6ILGx8ywzogDEIlutChHtKN4f-vzuN2iuwRjRYaAnpkEALw_wcB:G:s&s_kwcid=AL!13472!3!558204152872!e!!g!!aidefinition!14386686693!128518517985& (accessed Dec. 02, 2021).

[30] "Learn about Types and Applications of Microcontrollers - EIT : EIT | Engineering Institute of Technology." https://www.eit.edu.au/resources/types-and-applications-of-microcontrollers/ (accessed Dec. 02, 2021).

[31] N. Nicolas and L. Francoise, Simonot, *Automotive embedded systems handbook*. 2009.

[32] "Top 17 programming languages for embedded systems | TechGig." https://content.techgig.com/top-17-programming-languages-for-embedded-systems/articleshow/77701805.cms (accessed Dec. 02, 2021).

[33] "Visibility Around Heavy Vehicles - SAAQ." https://saaq.gouv.qc.ca/en/road-safety/behaviours/blind-spots/visibility-around-heavy-vehicles (accessed Jan. 06, 2022).

[34] "A Brief Introduction to Controller Area Network." https://copperhilltech.com/a-brief-introduction-to-controller-area-network/ (accessed Jan. 06, 2022).

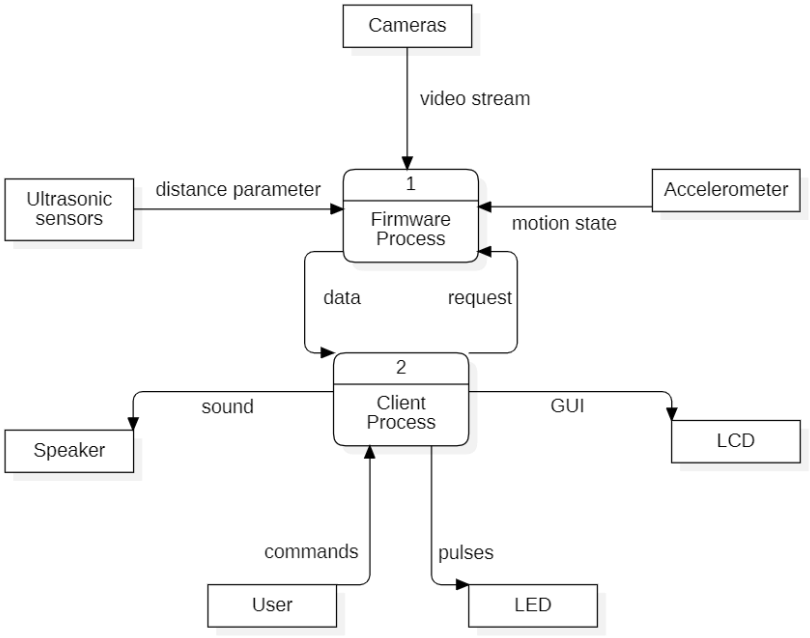
Appendices

Appendix A: Other architectural views

A 1. Data Flow

The figure below shows the typical UML data flow diagram for the system in question. The system has two key processes. This is a level 1 DFD (Data Flow Diagram), as the internal processes in the process 1 and 2 are abstracted. The firmware process reads inputs from three sensors, the cameras, ultrasonic sensors and accelerometer. It should be noted that, although one block entity was adopted for the cameras and ultrasonic sensors, the actual system will have more than one camera as well as ultrasonic sensors. This illustration was adopted for simplicity and brief clarity.

The firmware process communicates with the client process via an inter process communication protocol. The client process will receive touch events from the application user and also present three forms of outputs. These outputs will be via LEDs, LCD, and speakers as shown explicitly.



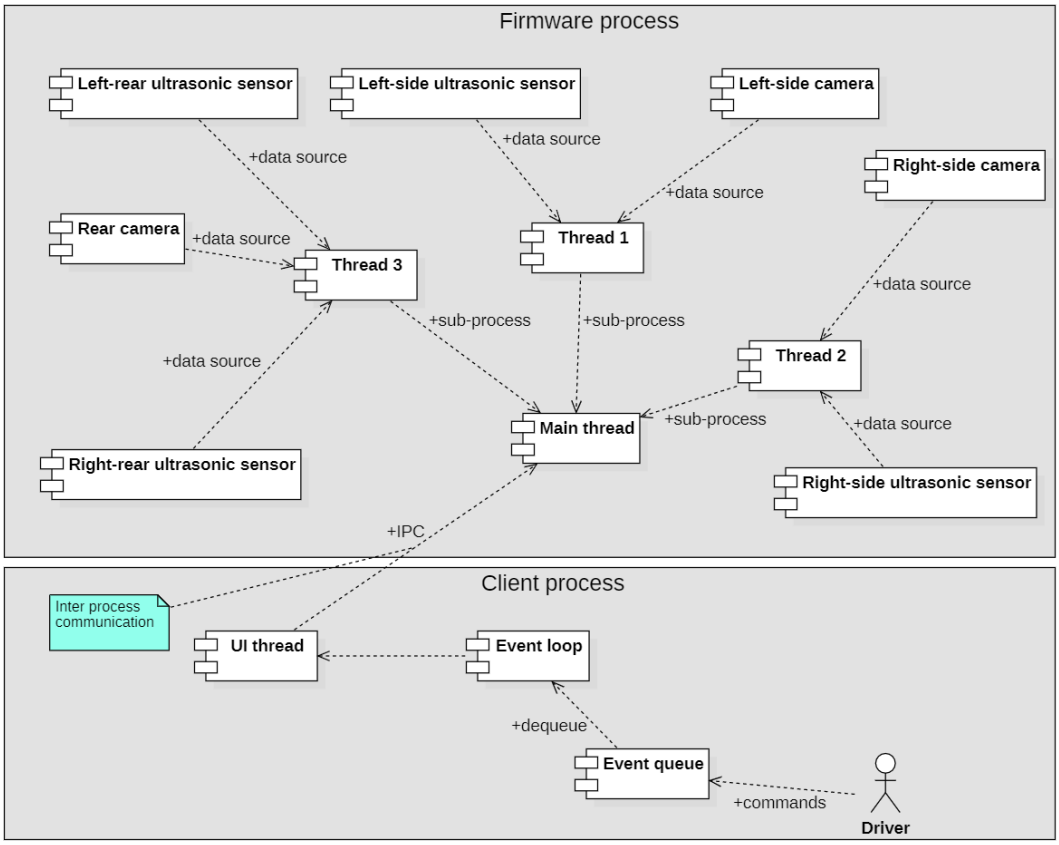
A 2. Concurrency Model

Data processing can be time-consuming when several data sources are in play, as is the case in the system at hand. To overcome this, we took advantage of powerful advanced programming techniques of multi-threading and multi-processing to reduce latency. We will have two independent processes for the entire system, one acting as the firmware process, and the other as the client process.

Further, within the firmware process as illustrated in the concurrency model below, three threads will be started to handle the three logical sub-divisions of the system. These sub-divisions are the left side, right side, and the rear of the bus. With the use of threading, there will be practically no

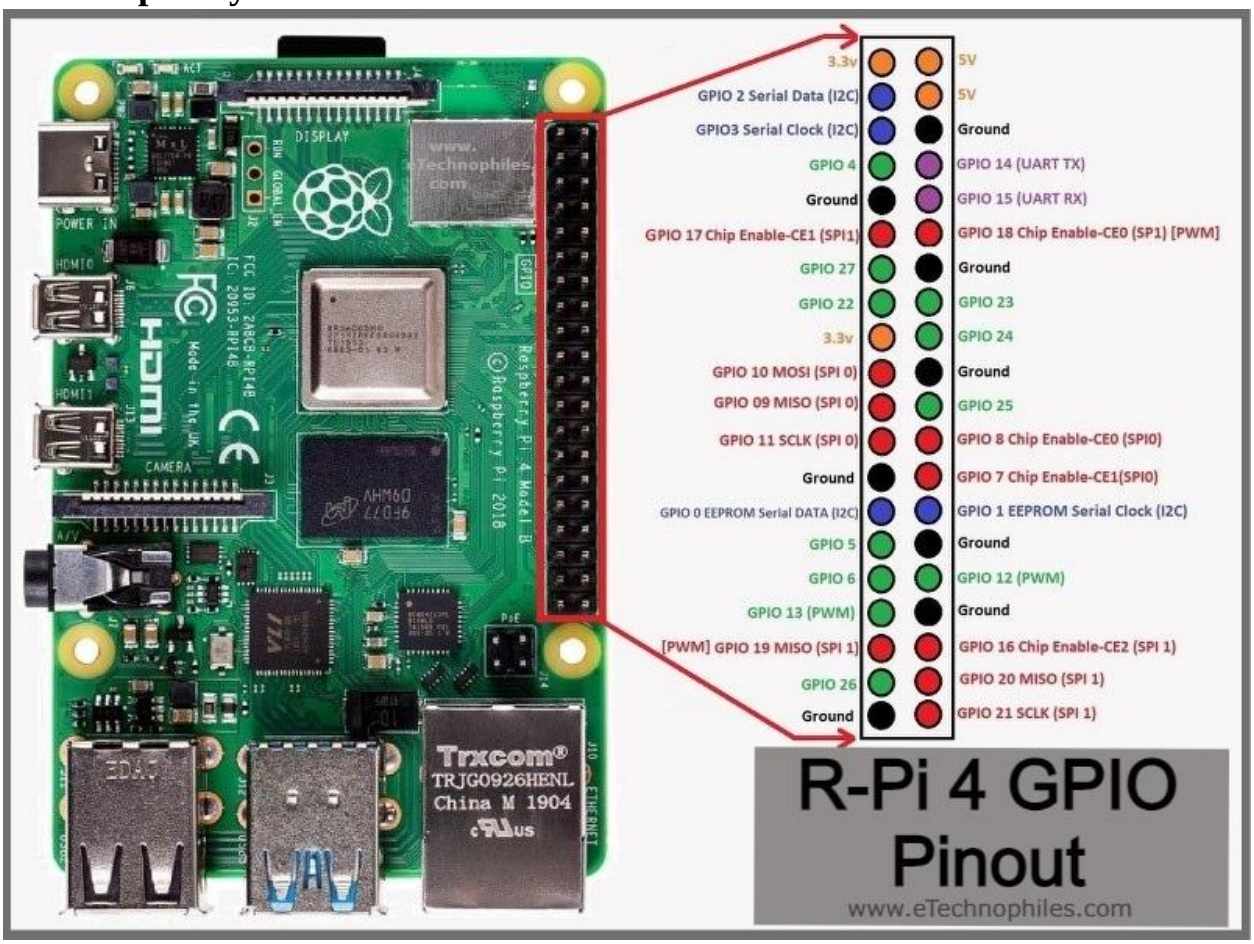
queuing since all thread runs in parallel. This means that object detection can occur in these three regions concurrently.

Inter-process communication protocols will be used to connect the two processes as shown below, and the client process will be accepting touch events from the driver in its event loop as well as UI (user interface) thread.



Appendix B: Raspberry Pi technical description

B 1: Raspberry Pi Pinout



B 2: Raspberry Pi technical description

Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)

2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE

Gigabit Ethernet

2 USB 3.0 ports; 2 USB 2.0 ports.

Raspberry Pi standard 40 pin GPIO header (fully backward compatible with previous boards)

2 × micro-HDMI ports (up to 4kp60 supported)

2-lane MIPI DSI display port

2-lane MIPI CSI camera port

4-pole stereo audio and composite video port

H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)

OpenGL ES 3.1, Vulkan 1.0

Micro-SD card slot for loading operating system and data storage

5V DC via USB-C connector (minimum 3A*)

5V DC via GPIO header (minimum 3A*)

Power over Ethernet (PoE) enabled (requires separate PoE HAT)

Operating temperature: 0 – 50 degrees C ambient