



COLLEGE OF ENGINEERING, DESIGN, ART AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

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

BY

TUSUUBIRA LATEEFA SHIBAH

17/U/10659/PS

**Main Supervisor**

Ms. Agatha Amara Turyagyenda

**Co-supervisor**

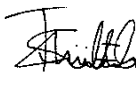
Dr. Maximus B. Byamukama

*Submitted in partial fulfilment of the requirement for the award of the degree of Bachelor of  
Science in Telecommunications Engineering*

## 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.

Signed 

Date .....8/02/2022.....

## Approval

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

**Main supervisor:** Ms. Agatha Amara Turyagyenda

Official Designation: Lecturer

Department of Electrical and Computer Engineering, Makerere University

Signature: 

Date: .....10/02/2022.....

**Co-supervisor:** Dr. Maximus B. Byamukama

Official Designation: Lecturer

Department of Electrical and Computer Engineering, Makerere University


Signature:  23/02/2022

Date: .....

**Field Supervisor:** Mr. Simon Peter Miyingo

Official Designation: Information Systems Manager

Department of Product Development, Kiira Motors Corporation

Signature: 

Date: .....09/02/2022.....

## **Dedication**

To my loving parents, family, academic supervisors, field supervisors at Kiira Motors Corporation and friends for continuous words of encouragement, guidance and support during the course of this project. May the Almighty reward them abundantly.

## **Acknowledgments**

I would like to extend my gratitude to the Almighty God for all the bounties He has bestowed onto me and enabling me to successfully complete not only my Final Year Project, but also four years of Engineering amidst the global pandemic.

I thank my parents, Mr. Musinguzi Dauda and Ms. Asimwe Sarah as well as my family for the continuous words of encouragement, guidance and support during my time at Makerere University.

Special thanks to the Kiira Motors Corporation (KMC) team; Mr. Simon Peter Miyingo, Mr. Kaweesa Brian, Mr. Fred Matovu, Ms. Thatcher Nakimuli, Mr. Ian John Kavuma, Mr. Paul Isaac Musasizi, Mr. Ali Ziryawulawo, Mr. Bimark Kyabangi and other members of staff that shared their experience, expertise and knowledge with me.

In a special way I would like to thank my Academic Supervisors, Ms. Agatha Amara Turyagenda and Dr. Maximus B. Byamukama for all the advice, guidance and help they rendered for the duration of this project and during the compilation of this report.

## **Abstract**

Blind Spots are regions around vehicle that cannot be viewed by a driver while using rear view and side mirrors. They are a major contributing factor to road traffic incidents around the world. 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 public transport operators operate on increasingly crowded roads, drivers need to help in eliminating blind spots and highlight potential collisions before they occur. The Kayoola Buses, developed by KMC potentially falls in this category. It is important that such locally developed transport solutions integrate navigation aids for object recognition in blind spots so as to reduce the likelihoods of RTIs

This project is focused on the Design and Implementation of a Blind Spot Detection and Monitoring System for Kayoola buses. The system is characterized by a hardware sub-system that measures gathers information such as motion of a vehicle, range of object in blind spot area to vehicle as well as video feed of the object. All this information is processed within the software sub-system so as to provide a driver with information of the object in the blind spot region.

# Table of Contents

Declaration .....	i
Approval .....	ii
Dedication .....	iii
Acknowledgments .....	iv
Abstract .....	v
List of Figures .....	ix
List of Tables .....	x
List of Abbreviations and Acronyms .....	xi
1. Introduction .....	1
1.1 Background .....	1
1.2 Problem Statement .....	2
1.3 Objectives .....	2
1.3.1 Main objective .....	2
1.3.2. Specific objectives .....	2
1.4 Justification .....	2
1.5 Scope .....	2
1.5.1 Object detection model .....	3
1.5.2 Test scope .....	3
2 Literature Review .....	4
2.1 Introduction .....	4
2.1.1 Adaptive Cruise Control .....	4
2.1.2 Active Park Assist .....	4
2.1.3 Automated Emergency Braking .....	4
2.1.4 Blind-Spot Monitor .....	4
2.1.5 Parking Sensors .....	4
2.2 Blind Spot Detection Systems .....	5
2.2.1 Vision-based solution .....	6
2.2.2 Non-vision based solution .....	7
2.2.3 Hybrid solution .....	8
2.3 Sensor Technologies .....	9
2.3.1 Ultrasonic sensors .....	9

2.3.1	Camera sensors .....	9
2.3.3	Radar sensors .....	10
2.3.4	Lidar sensors .....	10
2.4	Graphical User Interface .....	10
2.5	Deep Learning .....	10
2.5.1	Computer Vision.....	11
2.5.2	Object Detection.....	12
2.6	Microcontrollers .....	12
3	Methodology.....	13
3.1	Introduction .....	13
3.2	System Requirements Analysis.....	14
3.3	System Modelling and System Architecture.....	16
3.3.1	The Context View.....	17
3.3.2	The Functional View .....	17
3.3.3	The Information View .....	19
3.3.4	The Concurrency View.....	19
3.3.5	Interaction scenarios .....	20
3.4	System Design.....	21
3.4.1	Circuit design .....	21
3.4.2	The GUI Wireframes .....	22
3.5	System Implementation.....	24
3.5.1	System Core Algorithm .....	24
3.5.2	Coordinate Mapping.....	25
3.5.3	GUI implementation .....	26
3.5.3.1	Splash screen .....	26
3.5.3.2	Standby screen.....	26
3.5.3.3	Monitor mode screen.....	26
3.5.4	Firmware Implementation .....	27
3.5.5	The Object Detection Model .....	30
3.6	Test specification.....	32
4	Results .....	33
4.1	Introduction.....	33



4.2 Results for Unit Tests .....	33
4.4.1 The Hardware unit .....	33
4.3 Integration and System test results .....	42
5. Conclusions, Challenges and Recommendations .....	47
5.1 Conclusions .....	47
5.2 Challenges .....	47
5.3 Recommendations .....	47
6. Further Work .....	i
6.1 Identification of Blind Spot Regions on the Bus .....	i
6.2 Integration of CAN Communication Protocol .....	i
6.3 Packaging for Deployment .....	i
6.4 Sensor Selections .....	i
6.5 Software licensing .....	ii
6.6 In-vehicle testing .....	ii
References .....	iii

## List of Figures

Figure 1: Illustration of the Different sensors in a vehicle .....	9
Figure 2: Neural Networks organized in layers consisting of interconnected nodes.....	11
Figure 3: BSDS Context View.....	17
Figure 4: BSDS Functional View .....	18
Figure 5: BSDS Information View .....	19
Figure 6: BSDS Concurrency View.....	20
Figure 7: BSDS Sequence Diagram.....	21
Figure 8: Circuit Diagram for BSDS .....	22
Figure 9: Splash Screen Wireframe .....	23
Figure 10: Standby Screen Wireframe.....	23
Figure 11: Monitoring Screen Wireframe.....	24
Figure 12: System Flow Diagram for the Core Algorithm.....	24
Figure 13:Left[Virtual coordinates]: Right[Order of Operations during mapping of nodes to edges] .....	25
Figure 14: GUI Monitoring Screen mode .....	27
Figure 15: BSDS Peripherals .....	27
Figure 16: Results from Custom Object Detection Model .....	31
Figure 17: Splash Screen .....	34
Figure 18: Standby Mode Screen.....	34
Figure 19: Scenarios from the Monitoring Mode .....	35
Figure 20: Code Compilation for the Accelerometer Unit Test .....	36
Figure 21: Results for Accelerometer Unit Test.....	37
Figure 22: Code Compilation for the Ultrasonic Sensor Unit Test .....	38
Figure 23: Results for the Ultrasonic Sensor Unit Test .....	39
Figure 24: Code Compilation for the LED Unit Test .....	39
Figure 25: Result for the LED Unit Test .....	40
Figure 26: Result for the Speaker Unit Test .....	40
Figure 27: Result for the Object Detection Model Unit Test, Using the Pi Camera .....	41
Figure 28: Code Compilation for the Object Detection Processing Code .....	41
Figure 29: Results from the Object Detector post processing .....	42
Figure 30: Fully Integrated BSDS .....	43
Figure 31: Blind Spot Regions.....	i

## List of Tables

Table 1: Renault Koleos Blind Spot Warning System.....	5
Table 2: Volvo Blind Spot Information System .....	6
Table 3: Intervention Logic .....	13
Table 4: System Requirements .....	14
Table 5:Functional Elements .....	18
Table 6: Connections of Peripherals to the Raspberry Pi .....	29
Table 7: Annotation of the Images.....	30
Table 8: Training of a Custom Object Detection Model .....	31
Table 9: BSDS System Integration .....	42
Table 11:Requirement Traceability Matrix .....	43

## List of Abbreviations and Acronyms

3D	3-Dimension
ABS	Anti-lock Braking System
Adobe XD	Adobe Experience Design
AI	Artificial Intelligence
BLE	Bluetooth Low Energy
BSDS	Blind Spot Detection System
CAN	Controller Area Network
CLI	Command Line Interface
CNN	Convolution Neural Network
DNN	Deep Neural Network
GPIO	General Purpose Input Output
GUI	Graphical User Interface
IoT	Internet of Things
KMC	Kiira Motors Corporation
LED	Light Emitting Diode
LIDAR	Light Detection and Ranging
RADAR	Radar Detection and Ranging
RAM	Random Access Memory
RTI	Road Traffic Incident
UN	United Nations
USB	Universal Serial Bus

# 1. Introduction

## 1.1 Background

Uganda has one of the highest rates of Road Traffic Incidents (RTIs) globally. Over the last decade, the road crash fatalities recorded rose from 2,597 to 3,503 representing a growth of 25.9%. The accident severity index is 24 people killed per 100 road crashes [1]. On average, Uganda loses 10 people per day in road traffic crashes, which is the highest level in East Africa [2]. 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) [3].

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) [3]. 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, 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 [4]. 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 [5]. 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 [6]. This system can only detect cars and possesses a high processing load due to the 3-D visual involved [6].

Another instance is Bosc-Mobility Solutions which offers a sensor-based blind-spot detection system with two ultrasonic sensors on each side of the vehicle that 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 through a warning sign in the side mirror. If the driver fails to see or ignores the warning and later 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 provides feedback through the blinking of the LED and auditory feedback [7].

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 Blind Spot 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**

Available solutions (blind spot detection systems) have varying features such as 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 extremely expensive rates (typically between 10 - 20 million Uganda Shillings). There is therefore a need for a locally developed, low-cost and customized blind spot detection system for the Kayoola buses.

## **1.3 Objectives**

### **1.3.1 Main objective**

To develop a system that detects objects in blind spot areas of the Kayoola Buses and alerts the driver of their proximity.

### **1.3.2. Specific objectives**

1. To develop the hardware and software requirements specifications for the Blind Spot Detection and Monitoring System.
2. To develop logical and physical design models for the System.
3. To implement the design specification into a functional prototype.
4. To implement an object recognition algorithm onto the Raspberry Pi.

## **1.4 Justification**

As public transport operators operate on increasingly crowded roads in Uganda, it is hard for drivers of long vehicles to view everything around the vehicle, yet the consequences of missing an obstruction could be catastrophic. The Kayoola Bus, developed by KMC potentially falls in this category. It is important that such locally developed transport solutions integrate navigation aids for object recognition in blind spots so as to reduce the likelihoods of RTIs.

## **1.5 Scope**

This project targeted the Kayoola buses, therefore our prototyping, deployment in future, and data collection 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 will first restrict our study to the Kayoola EVS (The Electric bus).

#### 1.5.1 Object detection model

The object detection being simply a feature of the system, the focus of the project will be in identifying an object detection model and fine tuning it other than developing one from scratch. This model was limited to the identification of cars, motorcycles, bicycles and people. Objects other than those listed above were labelled and unknown by our deep learning model.

#### 1.5.2 Test scope

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

## 2 Literature Review

### 2.1 Introduction

This chapter contains an overview of vehicle safety features including different blind spot detection systems. This chapter also provides information on concepts like deep learning, sensor technologies, Graphical User Interfaces (GUIs) as well as microcontrollers.

Vehicle safety features have evolved a great bunch the years. Features like crumple zones, seat belts and airbags all provide protection if a crash occurs, however active safety assist technologies which can prevent a crash from occurring are now a significant point of differentiation. These include Blind Spot Monitoring (BSM), Autonomous Emergency Braking (AEB), active Lane Keep Assist (LKA) and Intelligent Speed Adaptation (ISA) [8].

#### 2.1.1 Adaptive Cruise Control

Adaptive Cruise Control uses the car's radar and camera modules to change the set cruising speed if it detects a slower vehicle ahead. When adaptive cruise control is engaged, the car will maintain a specific distance from the car in front[1].

#### 2.1.2 Active Park Assist

Using sonar and radar, vehicles equipped with *Active Park Assist* will look out for and measure empty parking spots and then actively steer the vehicle into them while the driver works the accelerator and brake.

#### 2.1.3 Automated Emergency Braking

Using forward-facing cameras and radar, vehicles with Automated Emergency Braking will warn the driver of an imminent forward collision with another vehicle, pedestrian, or any other object and then brake (stop) the vehicle on behalf of the driver if they do not take any action.

#### 2.1.4 Blind-Spot Monitor

Using sonar sensors attached to the rear bumpers or sometimes cameras fixed in the exterior mirrors, ***blind-spot monitoring systems*** watch adjacent lanes and alert the driver to other vehicles that might be in the driver's blind spot or hidden by the vehicle's profile (roof pillars). Most cars with this feature have warning lights in the exterior mirrors that flash or blink when a vehicle is detected close by and one lane over[1].

#### 2.1.5 Parking Sensors

Parking sensors; also called proximity sensors aid the driver during parking maneuvers by using ultrasonic transducers to locate obstacles such as parked cars, or curbs and alert the driver with a series of beeps that increase in intensity as the vehicle nears the object. Sensors are usually located on the front and rear bumpers[1].



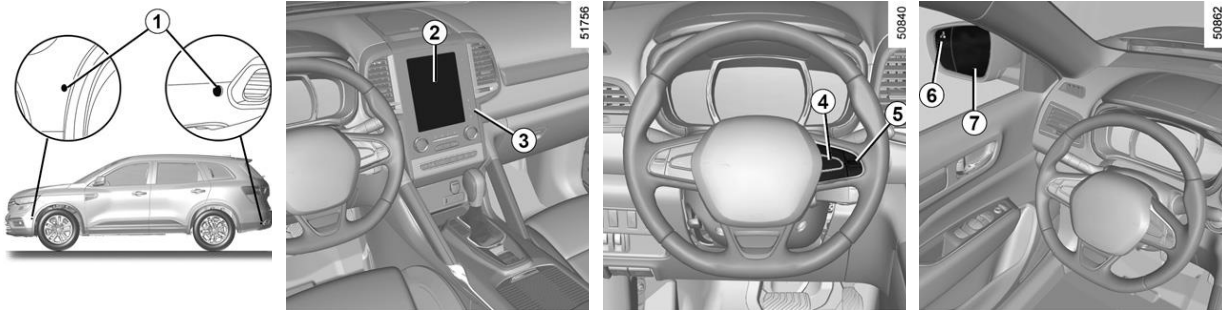
## 2.2 Blind Spot Detection Systems

Many car manufacturers and private companies such as Bosch, Renault, Volvo, Toyota, and Ford have developed Blind Spot Detection System (BSDS) using different methods and techniques from each other but still harboring the same approach which is to detect object presence in blind spot areas and alert the driver.

The Bosch blind spot detection system observes the surroundings of the vehicle when changing lanes and warns drivers of dangers. Two ultrasonic sensors are situated on each side of the vehicle and monitor the space in the adjacent lane, allowing the system to cover the dangerous blind spots. If another vehicle is situated in the monitored area, the driver is alerted to the potential danger by means of a warning sign located in the side mirror. If the driver fails to spot or ignores this warning and activates the turn signal to change lanes, the system is also able to trigger an audible warning.

The Renault Koleos Blind Spot Warning System alerts the driver about other vehicles in the detection zone. The system is activated when the vehicle is in motion with its speed between approximately 30 km/h (19 mph) and 140 km/h (87 mph). This function uses sensors installed in the front and rear bumper of both sides.

*Table 1: Renault Koleos Blind Spot Warning System*



The Volvo BLIS (Blind Spot Information System) is a function designed by Volvo for providing support for the driver when driving in dense traffic on roads with several lanes in the same direction. It is activated when the engine is started. This is confirmed by the indicator lamps located in the door panels blinking once. The BLIS function can also be deactivated/activated by pressing the BLIS button on the center console.

Table 2: Volvo Blind Spot Information System



Studies on Blind Spot Detection Systems have been focused on two kinds of Blind Spot Detection Systems that is; Vision-based and Non-vision based. Vision based systems use camera sensors with computer vision as well as deep learning techniques while Non-vision based systems use radar, infra-red, Bluetooth, and ultrasound as sensors for blind spot detection.

### 2.2.1 Vision-based solution

Yiming Zhao, Lin Bai, Lecheng Lyu, and Kinming Huang presented a design of 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 depth wise 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, in this case the tradeoff was between accuracy and cost. The limitation to this research was that only a few road and weather conditions were considered which is practically not sufficient for such systems to be deployed in the real World [2].

Huei-Yung, Jyun-Min, Lu-Ting, and Li-Qui proposed a vision-based driver assistance system for highway, urban, and city environments. Their system consisted of three subsystems which are lane change detection, forward collision warning, and overtaking vehicle identification. During the implementation, they used two monocular car digital video recorders to capture the front and rear views of the traffic scenes [3]. 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 demonstrated the effectiveness of the proposed techniques [3]. However, on the downside, they did not adopt stereo vision for the cameras making depth estimation for the front vehicles more difficult.

D. Kwon, R. Malaiya, G. Yoon, J. Ryu, and S. Pi, developed a camera-based vehicle blind spot detection system through the FCN (Fully Connected Networks) model. Their main research goal was the development of a very safe and lightweight camera-based blind spot detection system for the application in future autonomous vehicles [4]. The established research framework had five stages: data preprocessing, feature extraction, FCN model learning, vehicle blind spot setting, and false positive reduction. Overall, 99.45% training accuracy and 98.99% testing accuracy of the FCN model were achieved, respectively. After deploying the software on the embedded board for

actual testing on a real road, they confirmed 93.75% average blind spot detection accuracy with three false positives [4].

### 2.2.2 Non-vision based solution

N. De Raeye, M. De Schepper, J. Verhaevert, and P. Van Torre, 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 [5]. The vulnerable road user has a wearable device that scans these advertisement packets. The algorithm inside the wearable interprets these messages and applies filtering on their RSSI levels [5]. 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  $\pm 8$  m distance. The test with multiple vulnerable road users at the same time led to the same results [5]. 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 in order to receive an alert, the complete group will be alerted due to the light and sound effect of the others [5]. 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 design of two standalone sub-system, one for the car and the other is a wearable. Besides, developing the two subsystems may not be the main issue but making sure every other road user wears is another issue requiring attention.

Liu, Wang, and Zou [6], 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 [6]. The transmitted signal from this millimeter radar system was defined in the form,

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

Where,  $f$  is the operation frequency of the FMCW radar,  $B$  is the bandwidth of modulation frequency,  $T$  is the time of modulation frequency [6].

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 [6]. The IF signals captured from the front-end of the radar follows 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 false detection rate is lower at 2.63% [6]. For the system control, the coordinates of the radar were mapped into the 2D coordinate of the vehicle. Based on the calibrated blind spot area coordinate system, the developed system determines if the target is in the blind spot area or not. The system was implemented on 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 build using computer vision [6]. 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 solution

J. Katarzyna, K. Maciej, and S. Wojciech, proposed safety support systems which were designed for the needs of the race Shell Eco-marathon. Shell Eco-marathon is the world's largest race for energy efficient vehicles [7]. Among the concepts of safety support systems, they presented three proposed solutions for blind spot detection and selected only one of those as explained below;

- i. The first concept involved the use of 9 photoelectric sensors with a range of 5m [7]. This solution offered merits of small dimensions and low price, but had issues due to sunlight interference, and low accuracy.
- ii. The second concept involved the use of Microsoft KINECT 4 devices [7]. With the built infrared scanner, it was possible to obtain high-resolution scanning. However, dimensions of the device may disrupt the aerodynamics of the vehicle and direct sunlight can disrupt the infrared scanner. In addition, devices were characterized by Kinect dead center to the distance of 0.4 m from the device.
- iii. The third concept was to use the Hokuyo laser scanner with a first class safety [7]. In this concept, one laser scanner was to be used to get the desired effect. The device is characterized by a wide angle and a high frequency operation of the scan. Besides the high equipment cost, its compact design and high frequency scanning makes it the optimum choice for blind spot detection.

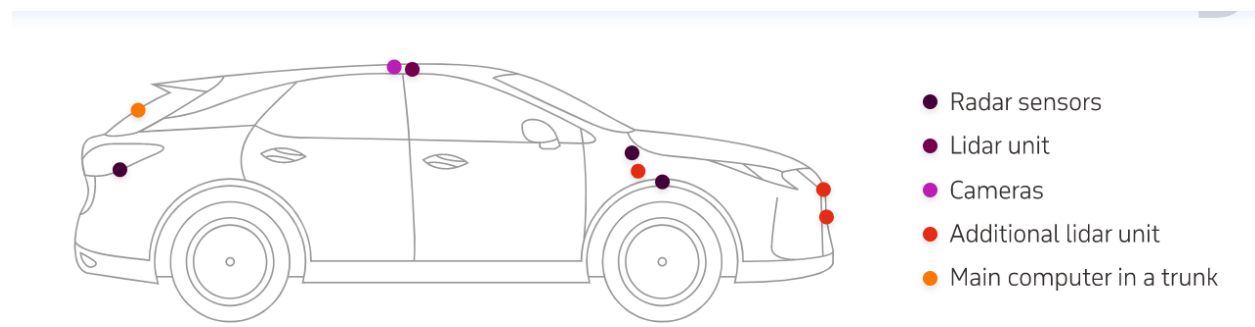
When they simulated the performance of their system, it was possible to alert the driver only when there was need [7]. The final data presentation to the driver was done in two ways but based on driver's preference, these methods include; showing the angle of the approaching vehicle and its corresponding distance to the driver or toggling between three LEDs (left, center, and right) to show the driver that their attention is required. The main downside of this solution was its cost being very high.

As part of the DESERVE (Development platform for Sales and Efficient Drive) project funded by the European Commission under ECSEL joint undertaking program, Pyykonen, Virtanen, and Kyytinen developed an intelligent blind spot detection system for long vehicles carrying heavy goods [8]. 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 under study to identify the optimal solution of best performance, cost and reliability [8]. 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 straight forward [8]. 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 sets had one Vislab 3DV-E camera and several ultrasonic range finders installed under the cargo bed [8]. 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 Sensor Technologies

The vehicles today have been integrated with a wide range of sensors providing critical data for performance, safety, convenience and comfort functionality. With the significant improvement in sensor, communication and information technology and the reliable application of obstacle detection techniques and algorithms, automated driving fast tracking to becoming a pivotal technology that will revolutionize the future of transportation and mobility. Sensors play a key role to the perception of vehicle surroundings in the automated driving systems, and the use and performance of the different integrated sensors can directly determine the safety and feasibility of automated driving vehicles [9].



*Figure 1: Illustration of the Different sensors in a vehicle*

Most of the automotive manufacturers today commonly use three types of sensors in autonomous vehicles: cameras, ultrasonic sensors, radars, and lidars.

### 2.3.1 Ultrasonic sensors

Ultrasonic sensors are usually mounted onto the vehicle bumpers for Assisted Parking Systems. So far, these sensors are only expected to function when the vehicle is in motion at a speed of less than 10 km/hour therefore, they are not able to measure small distances with 100% accuracy. In autonomous cars, however, these sensors could potentially be used along with radar, cameras and other sensor technologies to provide the distance measuring functionality [10].

### 2.3.1 Camera sensors

Autonomous cars usually have video cameras and sensors so as to observe and interpret the objects in the road the same way human drivers do with their eyes. By equipping these vehicles with

cameras at every angle, the vehicles are capable of maintaining a 360° view of their external environment, therefore providing a broader picture of the traffic conditions and objects around them.

Cameras are relatively inexpensive and with their appropriate software, can detect both obstacles in motion or static obstacles within their field of view and provide high-resolution images of the external surroundings. Today, 3D cameras are available and being used for displaying highly detailed and realistic images. These cameras automatically detect objects, classify them, and determine the distances between them and the vehicle. For example, the cameras can easily identify other cars, pedestrians, cyclists, traffic signs and signals, road markings, and curb [11].

### 2.3.3 Radar sensors

Radar (Radio Detection and Ranging) sensors perform a crucial role to the overall function of autonomous driving as they send out radio waves that detect objects and gauge their distance and speed relative to the vehicle in real time.

Both short and long-range radar sensors can be deployed all around the car and each one has different functions. While short range (typically 24 GHz) radar applications enable blind spot monitoring, the ideal lane-keeping assistance, and parking aids, the roles of the long range (typically 77 GHz) radar sensors include automatic distance control and brake assistance [11].

### 2.3.4 Lidar sensors

Lidar (Light Detection and Ranging) sensors work in a way similar to radar systems, differing only with the use of lasers instead of radio waves. LiDAR is a remote sensing technology that works on the principle of emitting pulses of infrared beams (laser light) which reflects off target objects. These reflections are then detected by the instrument and the time taken between emission and receiving of the light pulse enables the estimation of distance between the object and the vehicle. As the LiDAR scans the external surroundings of the vehicle, it generates a 3D representation of the scene in the form of a point cloud [10].

## 2.4 Graphical User Interface

A graphical user interface (GUI) is a type of user interface where users interact with electronic devices through visual indicator representations. There are different visual programming languages each with its unique advantages for the development of a graphical user interface design over the other. Examples include python, C# or Java.

## 2.5 Deep Learning

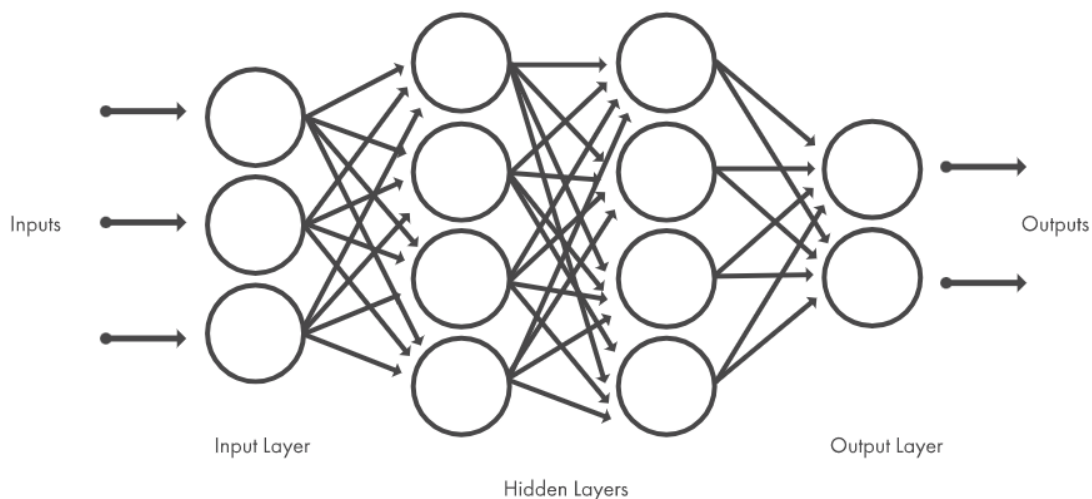
Deep learning is a machine learning technique that teaches computers to accomplish what comes naturally to humans that is; learn by example. Deep learning is a key technology behind

autonomous vehicles; enabling them to recognize a stop sign, or to distinguish a pedestrian from a poster sign along roads. [12].

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound provided by the programmer. Deep learning models can achieve high levels of accuracy, and sometimes even exceeding human-level performance. Models are trained by using a very large set of labeled data and neural network architectures that contain multiple layers [12].

Most deep learning methods use **neural network** architectures, hence the reason why deep learning models are usually referred to as **deep neural networks**. “Deep” in this instance usually refers to the number of hidden layers within the neural network. Traditional neural networks normally contain only 2-3 hidden layers, while deep networks could have as many as 150 [12].

Deep learning models are trained using large sets of labeled data and neural network architectures that learn features directly from the data in the dataset without the need for manual feature extraction [12].



*Figure 2: Neural Networks organized in layers consisting of interconnected nodes*

### 2.5.1 Computer Vision

Computer vision is a set of techniques used for extracting information from images, videos, or point clouds provided by the programmer. Computer vision includes image recognition, activity recognition, motion estimation, video tracking, and object detection. Examples of real-world applications include; face recognition as a security feature for logging into smartphones, pedestrian and vehicle avoidance in autonomous vehicles, and tumor detection in medical sector using MRIs. Software tools such as MATLAB<sup>®</sup> and Simulink<sup>®</sup> are usually used to develop these computer vision techniques [13].

Deep learning approaches to computer vision are useful during object detection, object recognition, image deblurring as well as scene segmentation. Deep learning approaches used for computer vision involve training Convolutional Neural Networks (CNNs), which learn directly from labeled data using patterns at different scales. CNN training requires a large set of labeled training images, videos or point clouds. Transfer learning uses pre-trained networks can accelerate this process with less training data [13].

### 2.5.2 Object Detection

Object detection is a computer vision technique for identifying instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results [14].

Object detection plays a crucial role in development of Advanced Driver Assistance Systems (ADAS) that enable cars to detect driving lanes and also perform pedestrian detection to improve on road safety. There are two key approaches to object detection using deep learning:

- **Create and train a custom object detector.** When training a custom object detector from scratch, one needs to design a network architecture for the computer to learn the features for the objects of interest (in the images or videos). One may also need to compile a very large set of labeled data in order to train the CNN. The results from a custom object detector can be impressive however the programmer needs to manually set up the layers and weights in the CNN, which requires a lot of time and training data [14].
- **Use a pre-trained object detector.** The majority object detection workflows using deep learning leverage transfer learning. This approach enables one to start with a pre-trained network and then fine-tune it for your application they are working on. This method can provide faster results because the object detectors have already been trained on thousands, or even millions, of images as well as videos[14].

## 2.6 Microcontrollers

A microcontroller is a small and low-cost microcomputer, that is designed to perform the specific roles of embedded systems for example displaying a microwave's information, receiving remote signals, and so on. Basically, a microcontroller gathers inputs, processes this information, and outputs a certain action based on the information gathered.

A microcontroller can be viewed as a small computer because of the essential components inside of it that is; the Central Processing Unit (CPU), the Flash Memory, the Serial Bus Interface, the Input/Output Ports (I/O Ports), the Random-Access Memory (RAM), and in many cases, the Electrical Erasable Programmable Read-Only Memory (EEPROM).



### 3 Methodology

#### 3.1 Introduction

This chapter details the steps taken to design and implement the blind spot detection and monitoring system. The Systems Engineering Methodology was followed; where systems requirements for the software and hardware subsystems were developed, system modelling and architecture, design models were developed, the system was developed and implemented into a prototype that was tested in conformance to the system requirements as illustrated in the table below.

Table 3: Intervention Logic

SN	Milestone	Key Questions	Instruments, Tools, Methods & Data Sources
1.	Requirements Specification	(1) Who are th 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, Surveys  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	<ol style="list-style-type: none"> <li>(1) What user interfaces does the system have?</li> <li>(2) In what environment will the system be used?</li> <li>(3) What are the inputs and the outputs of the system?</li> <li>(4) How are the inputs processed?</li> <li>(5) How much power is needed for the different components of the system to operate?</li> </ol>	<p>Data Sources: Internet, SRS</p> <p>Methods: Conceptual Data Modelling</p> <p>Tools: Adobe XD, Fritzing software</p> <p>Output: SDD (System Design Document)</p>
4.	Implementation	<ol style="list-style-type: none"> <li>1) What are the competences of the team members?</li> <li>2) What hardware will be used?</li> <li>3) What software will be used for development?</li> <li>4) What development methodologies will be employed to come up with the various components of the system?</li> <li>5) What Programming Language(s) will be used to develop the system?</li> </ol>	<p>Data Sources: Internet, SRS, SDD, Books</p> <p>Methods: A mix of Prototyping</p> <p>Tools: Equipment Datasheets, Micro Controller Units, Sensors, Actuators</p> <p>Output: Prototype</p>
5.	Test Specification	<ol style="list-style-type: none"> <li>(1) What is the scope of the testing? (Components that will be tested)</li> <li>(2) What type of testing will be performed?</li> <li>(3) What are the objectives of testing the system?</li> <li>(4) What is the test environment?</li> </ol>	<p>Data Sources: Internet, SRS, SDD</p> <p>Methods: Unit Testing, Integration Testing, System Testing</p> <p>Tools: Configuration management tools</p>

### 3.2 System Requirements Analysis

Requirements Analysis is the process of defining the expectations of the users for a system that is to be built. It involves the tasks that are conducted to identify the needs of the stakeholders. The requirements of the BSDS were categorized into two based on the sub-system; the software and hardware sub-system. These were presented in the classes of Functional Requirements (**REQF**) and Non-functional Requirements (**REQNF**).

*Table 4: System Requirements*

SN	Sub-System	ID	Requirements Description
----	------------	----	--------------------------

1.	Software	<b>Functional Requirements</b>	
		<b>REQF001</b>	Shall provide visual display of the location and distance of the objects in the blind spots of the bus in real-time.
		<b>REQF002</b>	Shall be capable of reading raw sensor values from the accelerometer, ultrasonic sensors and camera.
		<b>REQF003</b>	Shall process raw sensor values into formats suitable for decision making as well as formats that can be interpreted by the user.
		<b>REQF004</b>	Shall initiate the blinking of LEDs when an object in the blind spot surpasses the defined threshold distance value [1m].
		<b>REQF005</b>	Shall initiate auditory feedback when the distance between the bus and object gets smaller than the threshold [1m].
		<b>REQF006</b>	Shall start on system start-up.
		<b>REQF007</b>	Shall be activated when motion of the bus has been detected.
		<b>REQF008</b>	Shall seamlessly interact with the hardware sub-systems which include; - the Raspberry Pi and the peripheral devices.
		<b>Non-functional Requirements</b>	
		<b>REQNF001</b>	Shall not to fail due to inability to read sensor outputs
		<b>REQNF002</b>	Shall withstand component and environmental failures.
		<b>REQNF003:</b>	The functions of the software shall be easily understood by the user (the driver).
		<b>REQNF004</b>	Worst case sensor response time shall be 1s.
		<b>REQNF005</b>	Shall use relatively optimum system resources, such as memory, CPU and disk.
		<b>REQNF006</b>	Shall identify the root cause of failure when it occurs.
		<b>REQNF007</b>	Shall be easily tested for any desired features.
		<b>REQNF008</b>	Shall be readily installable on the Raspberry Pi
		<b>REQNF009</b>	Shall conform to the Raspbian OS.
		<b>REQNF010</b>	Shall be easy to replace the different software components at any desired time.
		<b>REQNF011</b>	Shall require minimum attention of the user (i.e., driver does not need to continuously glance at the display) so they can focus driving.

		<b>REQNF012</b>	Shall present a user interface which is slick, intuitive and attractive.
		<b>REQNF013</b>	Shall notify user in the event that the system fails.
2.	Hardware	<b>Functional Requirements</b>	
		<b>REQF009</b>	Shall have ultrasonic sensors for range measurements of target objects
		<b>REQF010</b>	The accelerometer shall be able to measure the presence or absence of motion to provide system power on, off or sleep mode regardless of the different environment variations (for example temperature and background noise).
		<b>REQF011</b>	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
		<b>REQF012</b>	The Camera shall capture frames from the blind spot areas that shall be fed to the object detection model.
		<b>REQF013</b>	The LEDs shall illuminate at the start of the bus to show that they are in proper working conditions. They shall blink when there a body at close proximity with the body of the bus.
		<b>REQF014</b>	The Speaker shall produce an alarm when an object or vehicle is in close proximity to the body of the bus.
		<b>Non-functional Requirements</b>	
		<b>REQNF014</b>	When an unpredictable failure occurs in reading values from either the accelerometer or the ultrasonic sensor, system shall recover briefly to full capacity or to safe mode respectively.
		<b>REQNF015</b>	The system shall be able to handle many inputs from its environment.
		<b>REQNF016</b>	The different components shall be enclosed in a casing in order to keep the connections firm and protect them from mechanical damage.

### 3.3 System Modelling and System Architecture

The purpose of this system modelling is to provide a comprehensive architectural overview of the Blind Spot Detection System (BSDS), using different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made about the system. The description and development of the architecture of the BSDS is modelled basing on the approach of multiple viewpoints and perspectives of the system stakeholders.

### 3.3.1 The Context View

This describes the relationships, dependencies and interactions between the BSDS and its environment (i.e. the people and external entities that it interacts with). It also demonstrates the interaction scenarios and sequences.

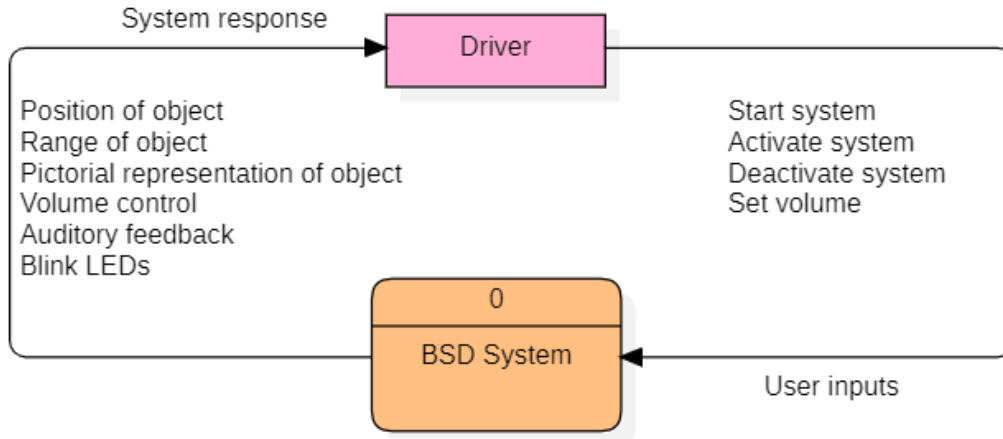


Figure 3: BSDS Context View

Generally, there is one external entity that interacts with the system and this is the driver. The driver supplies some inputs to the system labelled by the “User inputs” and the system in question gives the driver feedback labelled as “System response”.

### 3.3.2 The Functional View

This defines the significant functional elements, the responsibilities of each, the interfaces they offer and the dependencies between elements. Functional elements, scenarios and system-wide processing.

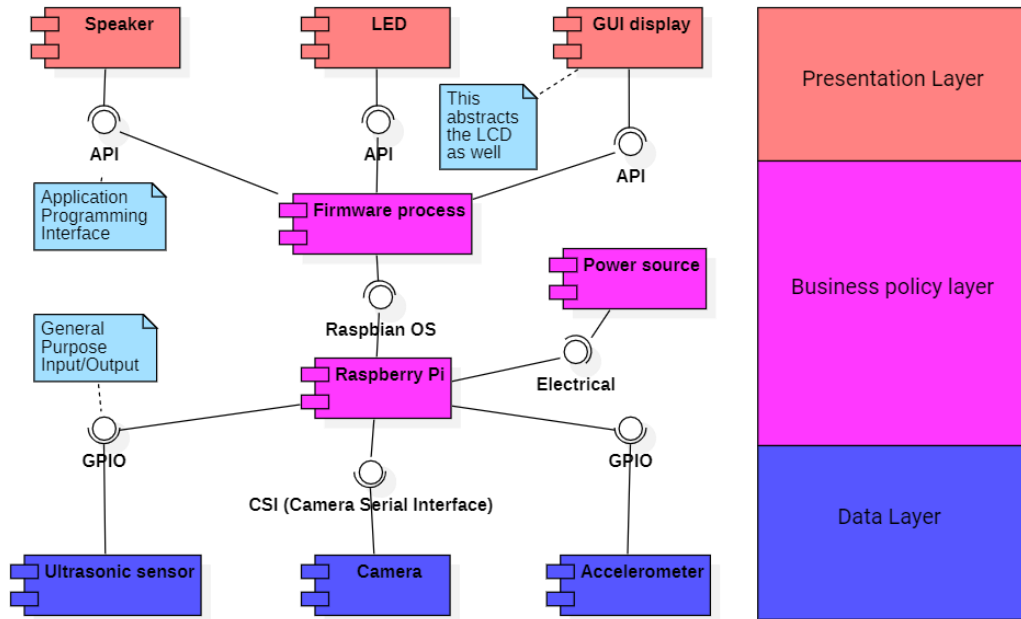


Figure 4: BSDS Functional View

Table 5: Functional Elements

Element Name	Responsibilities	Interfaces
Raspberry Pi	It receives data from the sensors. It processes this data, and make 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 its distance.	GPIO, Air interface
Accelerometer	This sensor detects if the bus is in motion or stationery.	GPIO
GUI display	This is the visual display; the LCD which is part of this components 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

### 3.3.3 The Information View

This defines the structure of the system's stored and transient information (e.g. databases and message schemas) and how related aspects such as information ownership, flow, currency, latency and retention will be addressed.

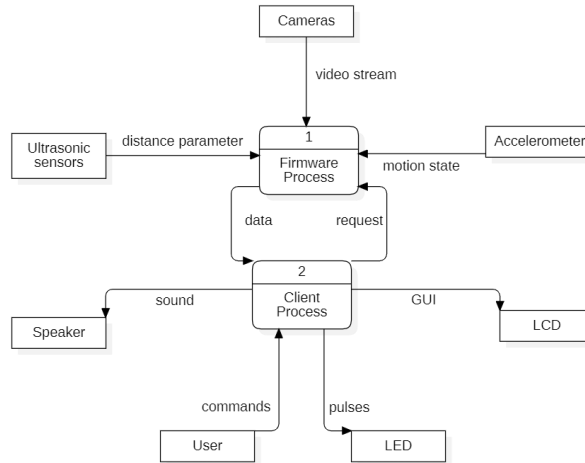


Figure 5: BSDS Information View

The figure shows the data flow diagram for the BSDS. The system has two key processes. The firmware process reads inputs from three sensors, the cameras, ultrasonic sensors and accelerometer. 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.

### 3.3.4 The Concurrency View

This defines the set of runtime system elements (such as operating system processes) into which the system's functional elements are packaged.

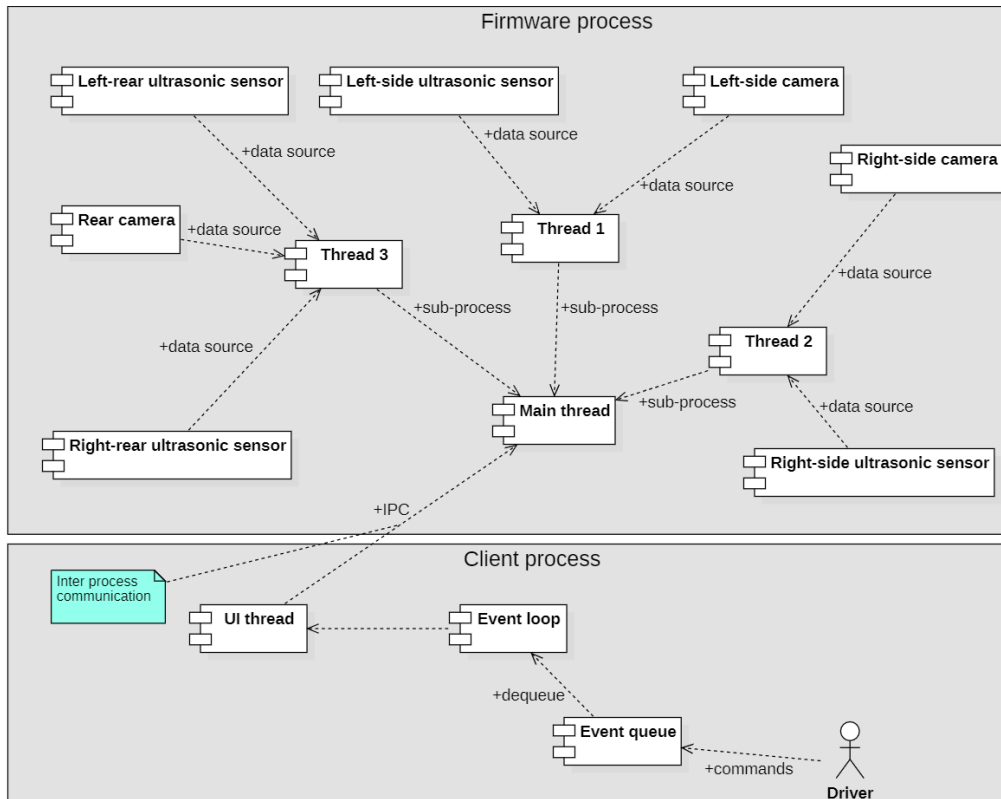


Figure 6: BSDS Concurrency View

### 3.3.5 Interaction scenarios

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



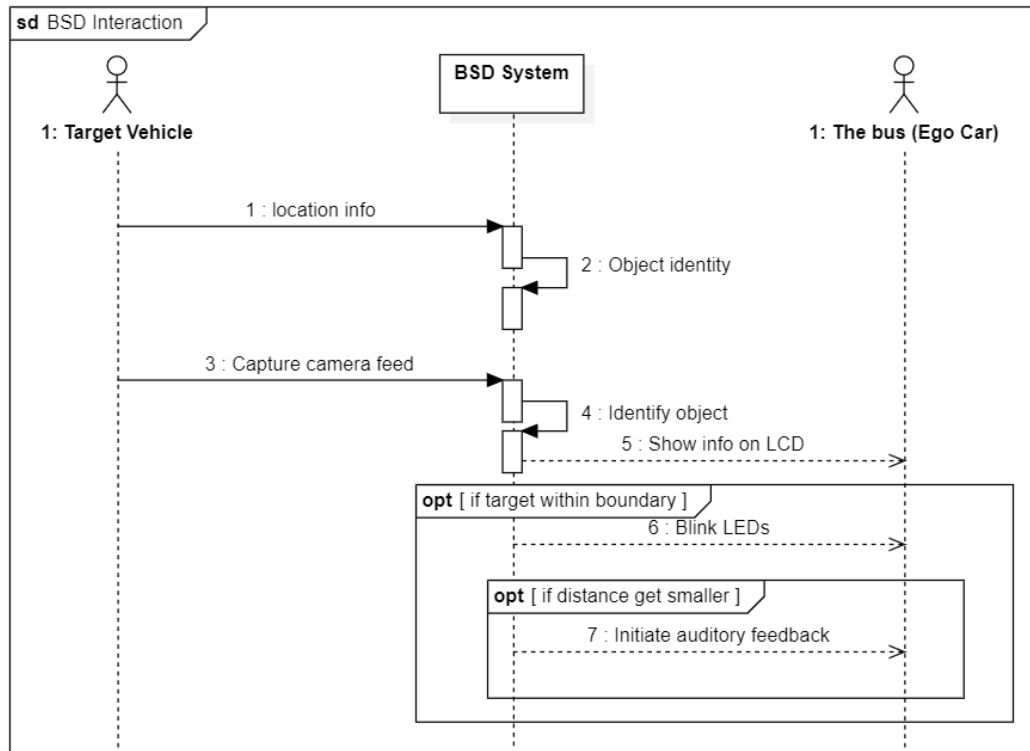


Figure 7: BSDS Sequence Diagram

The sequence diagram above models the interaction between the involved entities and the BSDS. 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 the blinking of the LEDs. When the distance between the target and the ego vehicle gets smaller, auditory feedback is initiated and the blinking rate of the LEDs increases.

### 3.4 System Design

System design is the process of defining the architecture, data structures, interfaces and modules for a given system. The primary work product of this stage is a blueprint for the coding of individual modules, programs, and ultimately the entire system.

#### 3.4.1 Circuit design

The figure below shows the basic circuit design of the core components. The Raspberry Pi model 4 has been used for this implementation. Two ultrasonic sensors were employed corresponding to the left and right side of the target vehicle. 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.

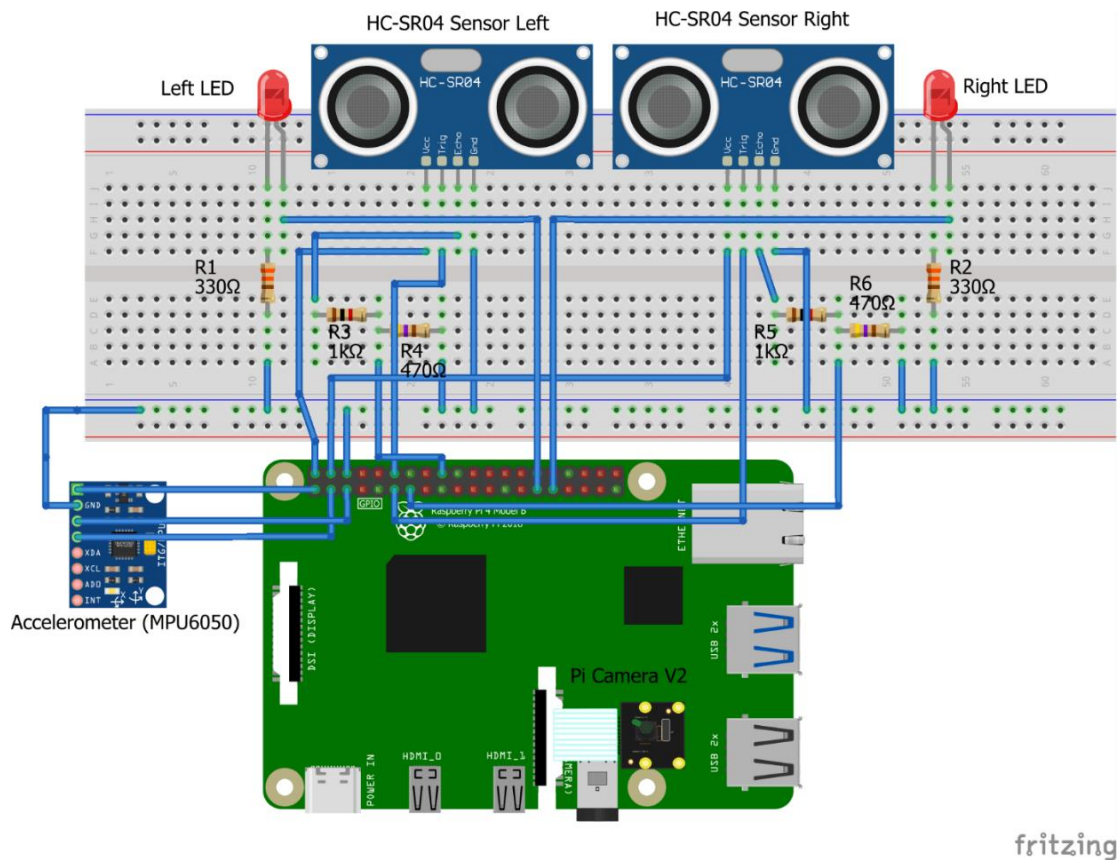


Figure 8: Circuit Diagram for BSDS

Resistors R1 and R2 were connected to the respective cathodes of the LEDs so as to reduce 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 were 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 requires 3.3V. These resistor pairs therefore form a voltage divider network that reduces the 5V to a safe level.

### 3.4.2 The GUI Wireframes

Three screens were designed for the GUI display using Adobe XD software. These include;

- The Splash Screen  
This can also be referred to as the Launch screen. It is the first screen shown as the program loads.

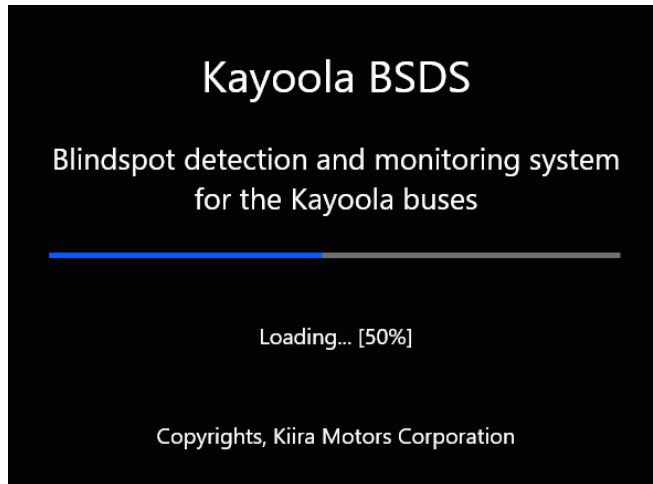


Figure 9: Splash Screen Wireframe

- The Standby Screen  
 This is the screen that is shown when the program is started but not yet activated by the motion sensor or with manual activation.

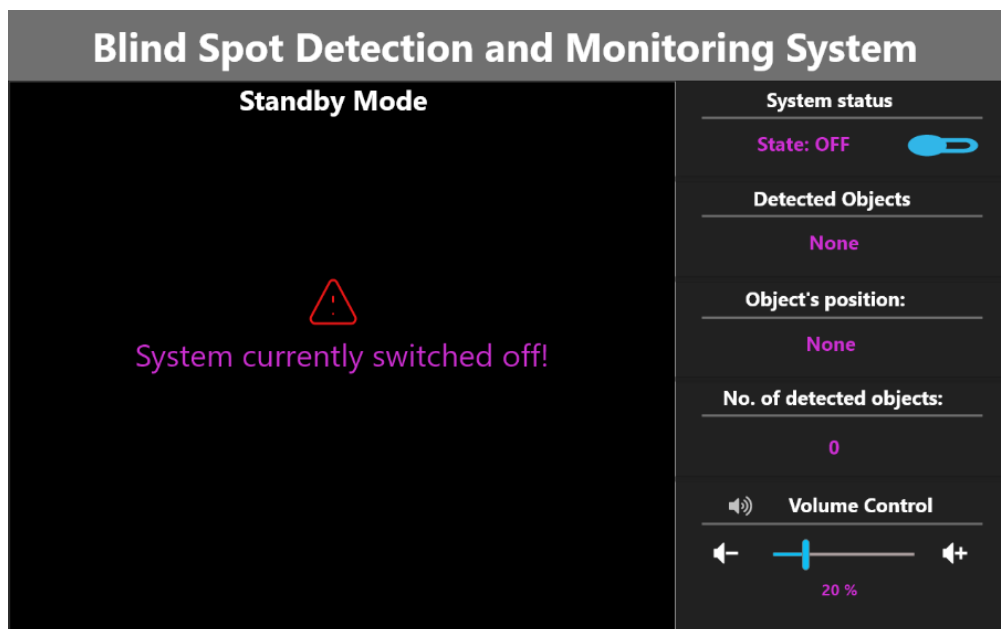


Figure 10: Standby Screen Wireframe

- The Monitor Screen  
 This is shown when the program has started and is activated. In this mode, the system constantly scans for objects around the target vehicle and provide valuable information to the driver.

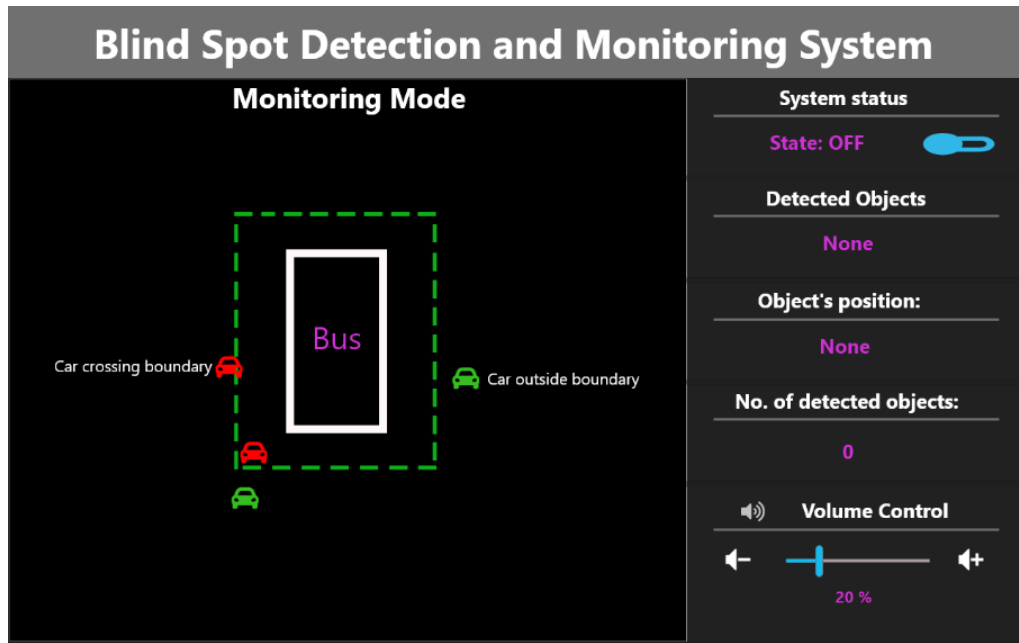


Figure 11: Monitoring Screen Wireframe

### 3.5 System Implementation

#### 3.5.1 System Core Algorithm

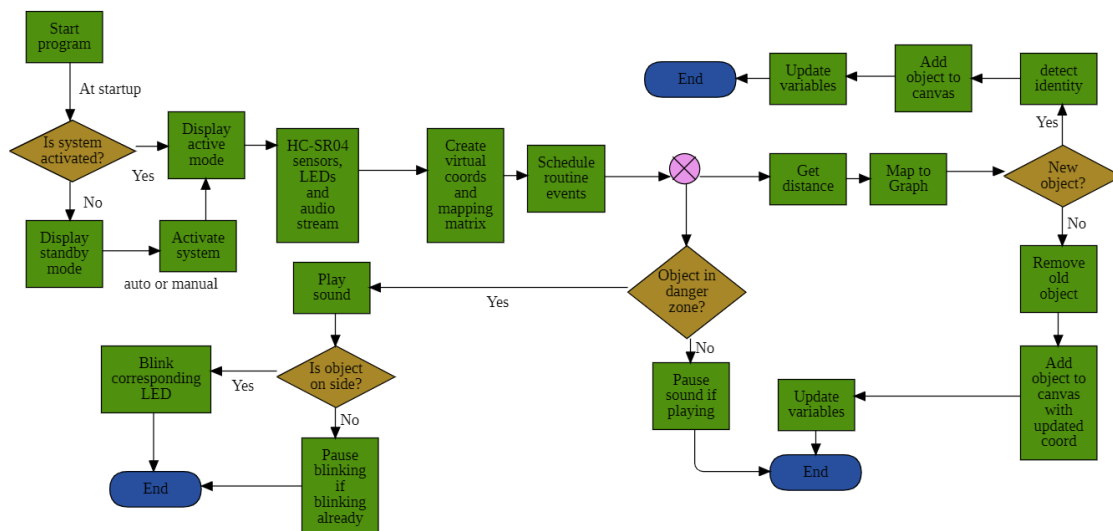


Figure 12: System Flow Diagram for the Core Algorithm

The figure above shows the core algorithm of the system at a high level. It describes how the system was implemented how the operation flows;

- 1) The program starts at 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 displayed. The user can also manually turn the system ON or OFF 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 ultrasonic sensor then routinely monitors nearby objects. Once an object is detected, it identifies it using the object detection model. It then either adds it to the canvas or update the already existing object if had been detected before.
- 5) If object is in danger zone, the system sounds the speaker and if the object is on the side of the bus, the LED corresponding to that side of the bus blinks.

### 3.5.2 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 represents the nodes while their equivalents in pixels which specifies the location of the object on the canvas represents the edges.

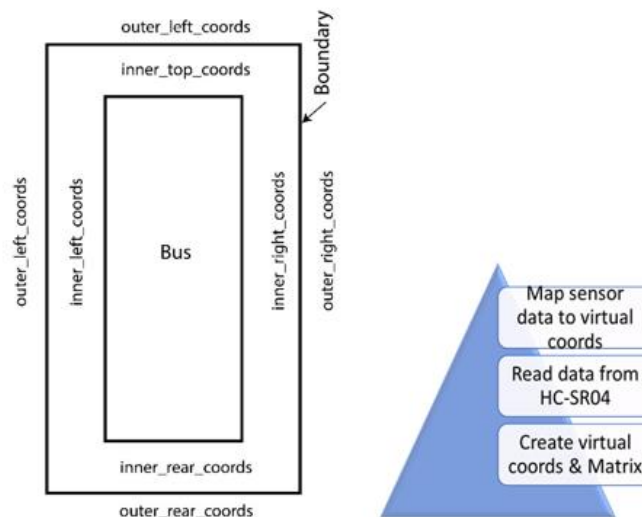


Figure 13: Left [Virtual coordinates]: Right [Order of Operations during mapping of nodes to edges]

The steps involved in coordinate mapping between the measured distance and pixel coordinates include;

- Creating of the possible pixel coordinates (virtual coordinates) and their access matrices. This is done as the program is loading after creating the canvas widget on which the objects will be placed after detection.
- After the virtual coordinate creation process is successfully completed, the system reads distance using the ultrasonic sensor and then maps this value to its corresponding pixel coordinate.

### 3.5.3 GUI implementation

For implementation of the GUI, Python programming language 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.

The view has a large canvas on which a 2D-map of the target vehicle is placed with a boundary line in green. The boundary line is 1m away from the target vehicle (this limit is due to the fact that 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 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.

#### 3.5.3.1 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.

#### 3.5.3.2 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.

#### 3.5.3.3 Monitor mode screen

In this mode as shown in the figure below 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 due to the fact that, 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 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 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 information on the category of the detected objects, alongside this categories, number of 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 sound by toggling the speaker icon. The user can also increase and decrease the volume of the auditory feedback.

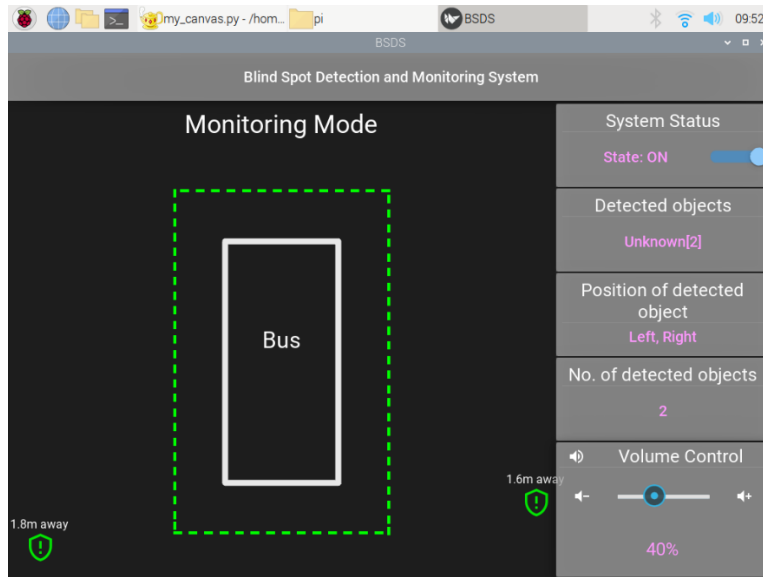


Figure 14: GUI Monitoring Screen mode

### 3.5.4 Firmware Implementation

The system generally has five categories of peripherals and these are shown in below.

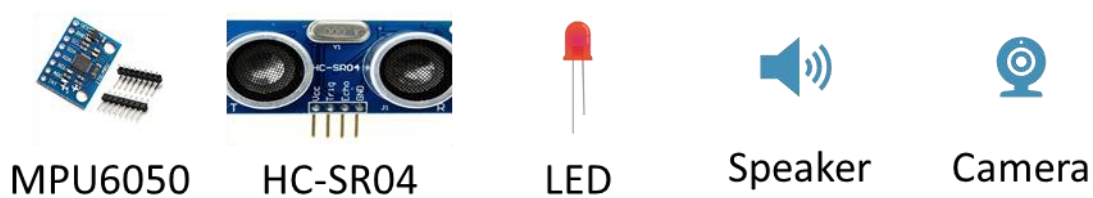


Figure 15: BSDS Peripherals

#### A) The MPU6050

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

This module uses the I2C module for interfacing with Raspberry Pi. 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 the rotational and linear acceleration.

#### B) The HC-SR04

The HC-SR04 Ultrasonic distance sensor consists of two ultrasonic transducers. The one acts as a transmitter which converts 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 travelled.

When a pulse of at least 10  $\mu$ S in duration is applied to the Trigger pin, the sensor transmits a sonic burst of eight pulses at 40 KHz. This 8-pulse pattern makes the “ultrasonic signature” from the device which is 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. If those pulses are not reflected back, then the Echo signal will timeout after 38 ms and return low. If those pulses are reflected back, the Echo pin goes low as soon as the signal is received. This produces a pulse whose width varies between 150  $\mu$ S 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 reflected object using the equation below;

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

The computation of distance and reading the sensor data happens in a thread separate from the main loop. This operation is a blocking one; 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.

### **C) The LED**

A light-emitting diode is a semiconductor light source that emits light when current flows through it. This is used for visual alert. Two scenarios were implemented; one turns OFF the LED while the other turns it ON. Using the blinking rate of 1s, this task was scheduled to turn the LED ON and OFF. 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.

### **D) The 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 the system was implemented using the Pygame module of the Python programming language. A short piece of music that plays for 32s was created and once an object is in the danger zone this music plays. If the object moves out of the danger zone, the music stops to play.

### **E) Camera**



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

### Device Connection

Two ultrasonic sensors were used for during implementation one was to detect objects from the left side of the target vehicle and the other detect objects from the right side of the target vehicle. Similarly, two LEDs were used during implementation, one for the left side of the target vehicle and the other for the right side of the target vehicle. The peripherals were connected to the Raspberry Pi as shown below;

*Table 6: Connections of Peripherals to the Raspberry Pi*

<b>Raspberry Pi</b>	<b>MPU 6050</b>
Pin 1 (3.3V)	VCC
Pin 3 (SDA)	SDA
Pin 5 (SCL)	SCL
Pin 6 (GND)	GND
<b>Raspberry Pi</b>	<b>HC-SR04 (Left)</b>
Pin 2 (5 V)	VCC
Pin 12 (GPIO 18)	TRIG
Pin 18 (GPIO 24)	ECHO (5 V)
Pin 14 (GND)	GND
<b>Raspberry Pi</b>	<b>HC-SR04 (Right)</b>
Pin 4 (5 V)	VCC
Pin 11 (GPIO 17)	TRIG
Pin 13 (GPIO 27)	ECHO (5 V)
Pin 9 (GND)	GND
<b>Raspberry Pi</b>	<b>LED (Left)</b>
Pin 25 (GND)	CATHODE
Pin 29 (GPIO 5)	ANODE
<b>Raspberry Pi</b>	<b>LED(Right)</b>

Pin 34 (GND)	CATHODE
Pin 31 (GPIO 6)	ANODE

### 3.5.5 The Object Detection Model

The purpose of the object detection model in this system is to identify the object in the blind spot so that, the driver can make decisions accordingly. The model developed was been limited to only people, vehicles, motorcycles and bicycles.

The steps taken in developing the object detection model included;

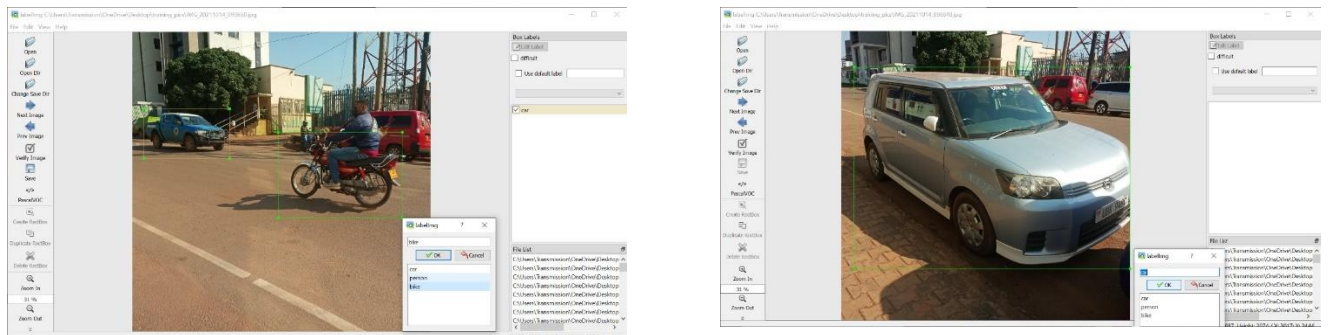
#### 1) Data Collection

A total of 504 different images were collected. The comprised of images of different cars, motorcycles and people. These were taken from a phone camera with a 64MP resolution.

#### 2) Data Preprocessing

The images were resized to a height and width of 640 pixels. They were also annotated with the labels corresponding to the object in the image using the graphical image annotation tool LabelImg.

Table 7: Annotation of the Images

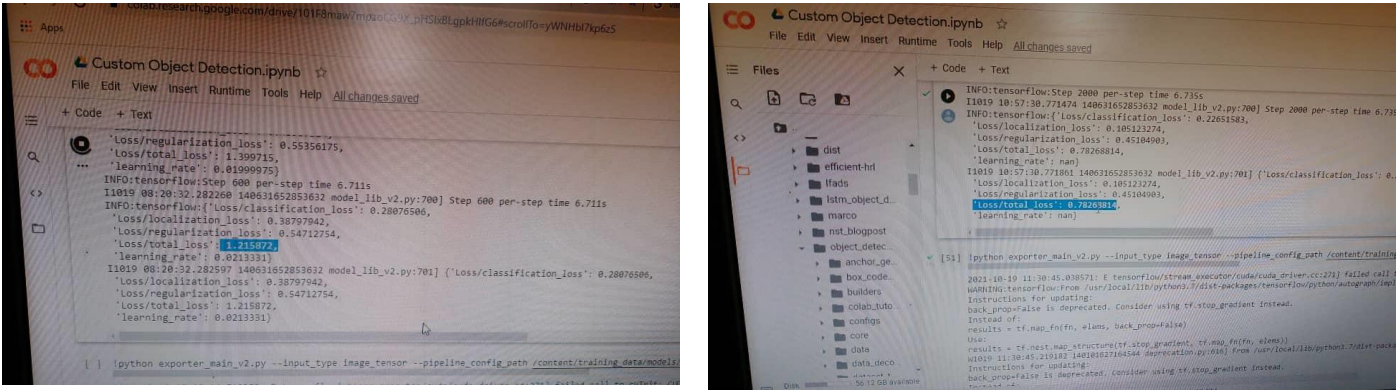


#### 3) Training of the Model

The dataset was split into two with a 9:1 ratio for training and testing respectively and a label map, which namely maps each of the used labels to integer values was created. We downloaded a pre-trained model; the *SSD ResNet50 V1 FPN 640x640* model, since it provides a relatively good trade-off between performance and speed.

We began training our custom model using our dataset from the model downloaded using Google Colab notebooks.

Table 8: Training of a Custom Object Detection Model



We tested the performance and realized that its accuracy was good (with a 81% confidence level) however the speed for detection would be longer than the duration stated during Requirements Engineering (realizing speeds between 20-30 seconds).

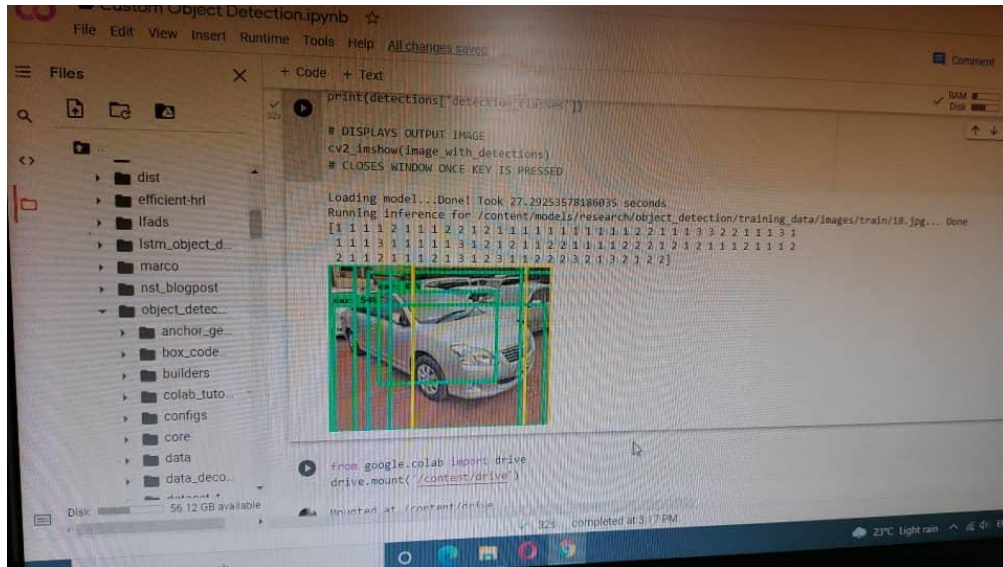


Figure 16: Results from Custom Object Detection Model

Since scope of the project was not to design a unique deep learning AI model, a huge repository of object detection models was explored so as to obtain a model with optimal accuracy and speed characteristics that could run for the Raspbian Pi 4 model B platform.

We chose the *SSD Mobilenet V1* model for this system. This is an object detection model trained on the COCO (Common Objects in Context) dataset. COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features some of which are: Object segmentation, Recognition in context, Super pixel stuff segmentation, with 330,000 images (>200,000 labelled). This model can detect an object from a frame with a latency of order of

~500ms. The Tensor Flow lite model was integrated into the system using the “tflite\_runtime” library for edge devices.

#### 4) Camera configuration

The Raspberry Pi camera was configured 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 object detection feature was implemented to be an independent daemon thread. The first task as the program loads is initializing the Pi camera, 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 suite the input features and finally feeds it to the model. The output of the model is then processed based on the labels, the 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 Test specification**

The BSDS is averagely complex and therefore involves many test activities being conducted at different levels of development. In order to structure the test processes and facilitate testing of the system, 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 fit into the larger system. Finally, the system testing checks to see how all units fits together to meet the system mission statement.

## **4 Results**

### **4.1 Introduction**

This chapter gives the results obtained after evaluating the test specifications. It presents the results obtained from training and testing the models used for the object detection model for implementation.

### **4.2 Results for Unit Tests**

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.

#### **4.4.1 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. The hardware comprises of; - a Raspberry Pi 4 model B that controls all other hardware components; two HC-SR04 sensors (ultrasonic sensors), used for range measurements of objects in the surrounding; an accelerometer (MPU6050), used for motion detection; two LEDs for visual alert; the speaker for auditory feedback, and a 5V DC power source for powering the raspberry Pi and the speaker.

The test objective here was to validate the ability of the hardware to support the embedded software.

#### **A) The GUI unit**

The display of this system has three views; the splash screen, the standby mode and monitoring mode displays.

#### **The Splash Screen**

The splash screen shown below is the first view of the GUI. It is what the user sees when the system starts as initialization occurs in the background. The figure shows the result of running the GUI program and thus the outcome of the implementation using Kivy Python GUI framework. This code was running on the Raspbian OS.

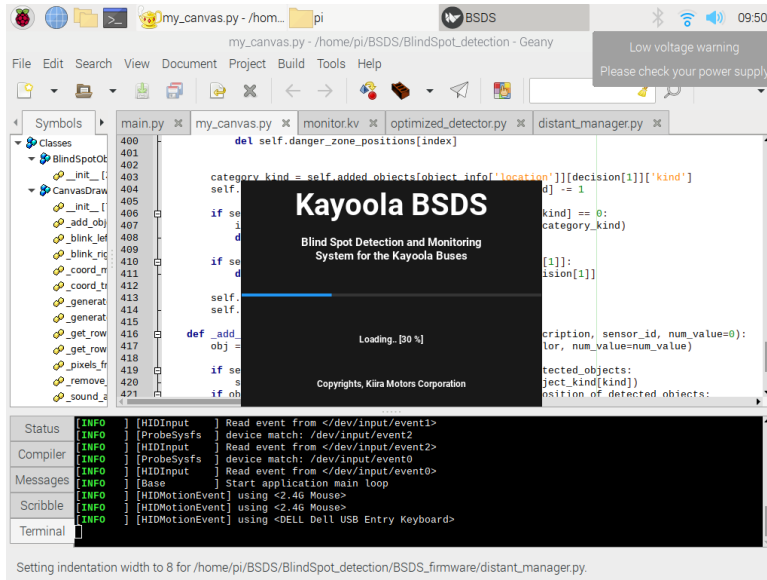


Figure 17: Splash Screen

## The Standby Screen

After the program loads fully, the splash screen switches to the standby mode. The view is shown below. 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.

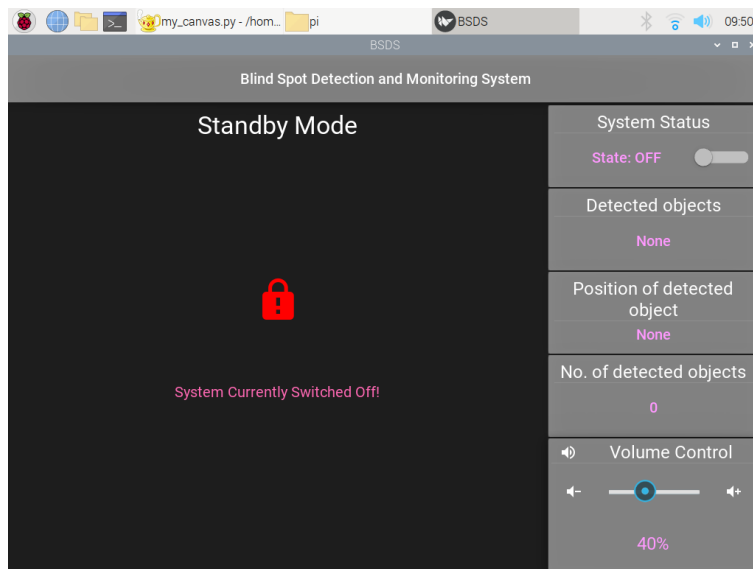


Figure 18: Standby Mode Screen

## The Monitoring Mode Screen

Once the system is activated manually or automatically by the motion sensor, the view shifts to the monitoring mode. The view has a large canvas on which a 2D-map of the target vehicle is placed with a boundary line in green. The boundary line is 1m away from the target vehicle (this limit is due to the fact that 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 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.

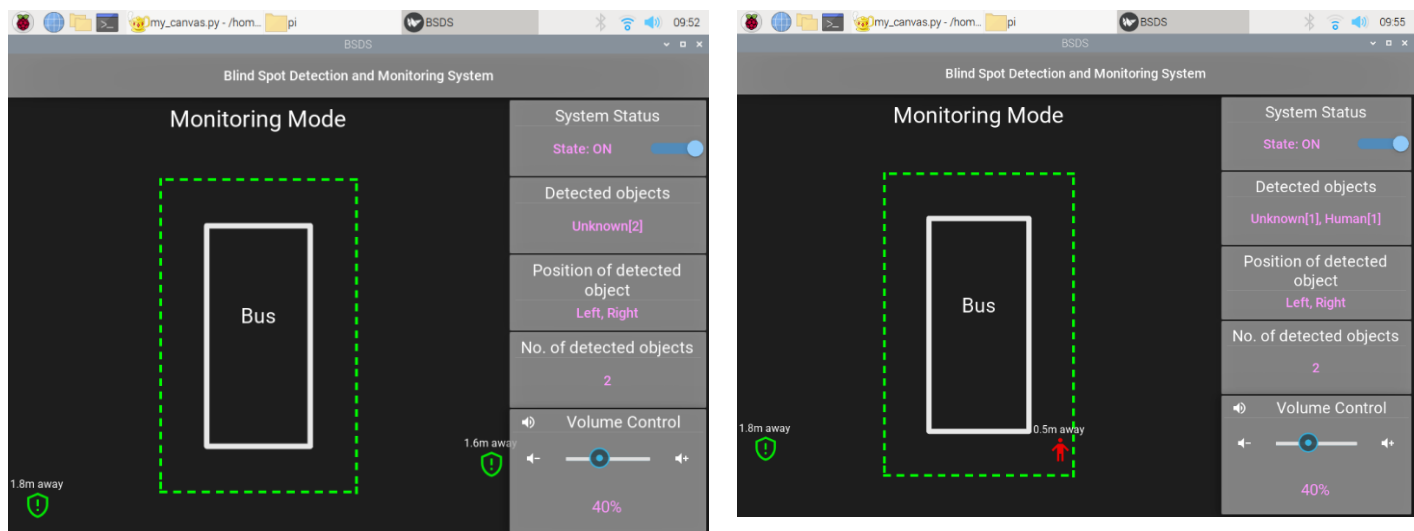


Figure 19: Scenarios from the Monitoring Mode

On the right side of the view, there is a list of cards stacked vertically that allows for interaction with system or to provide information to the user.

- The first card indicates the System Status. The user can manually switch the system ON and OFF.
- The second card indicates the Category of the Detected Objects, as well as the number of objects detected altogether from both sides.
- The third card indicates the Position of the Detected Objects that is Left or Right.
- The fourth card shows the total number of detected objects at the current instant.
- The fifth card is for volume control from the speaker. From this card the user can disable sound by toggling the speaker icon. The user can also increase and decrease the volume of the auditory feedback.

### B) 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 below.

```

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 20: Code Compilation for the Accelerometer Unit Test*

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; whether it’s in motion or not is defined by the “detect\_state” method.

When the above code was executed, the result in the figure below was obtained. The objective of this test scenario, was to check for the sensor functionality.





```

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 22: Code Compilation for the Ultrasonic Sensor Unit Test

The sensor is initialized during instantiation of the class. The “compute\_distance” method deduce 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 the figure below was obtained. The objective of this test was to find out if the sensor can determine range of targets in its vicinity.



```
Measured Distance = 2106.1 cm
Measured Distance = 2106.1 cm
Measured Distance = 401.4 cm
Measured Distance = 2104.8 cm
Measured Distance = 401.4 cm
Measured Distance = 2106.5 cm
Measured Distance = 249.3 cm
Measured Distance = 2104.5 cm
Measured Distance = 2104.6 cm
Measured Distance = 249.7 cm
Measured Distance = 251.4 cm
Measured Distance = 2104.9 cm
Measured Distance = 251.5 cm
Measured Distance = 399.7 cm
Measured Distance = 399.7 cm

Status 10:30:35: File /home/pi/BSDS/BlindSpot_detection/BSDS_firmware/distant_manager.py opened (/).
10:30:50: File /home/pi/BSDS/BlindSpot_detection/my_canvas.py closed.
Compiler 10:30:52: File /home/pi/BSDS/BlindSpot_detection/BSDS_firmware/distant_manager.py closed.
```

Figure 23: Results for the Ultrasonic Sensor Unit Test

## D) The LED firmware

For this implementation, the Raspberry Pi GPIO library was used. The LED has two pins with the longer being the anode. Implementation followed the design and pin configuration in chapter 3. The figure below shows the firmware implementation for the LED.

```
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()

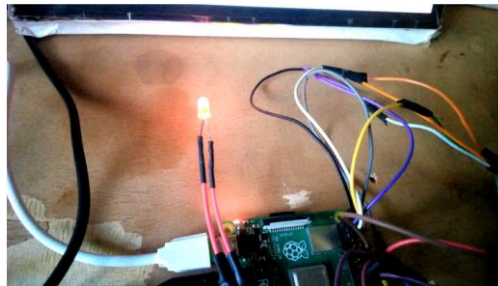
    def run(self, turn_off=False):
        if self.state or turn_off:
            self.turn_off()
        else:
            self.turn_on()

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

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

Figure 24: Code Compilation for the LED Unit Test

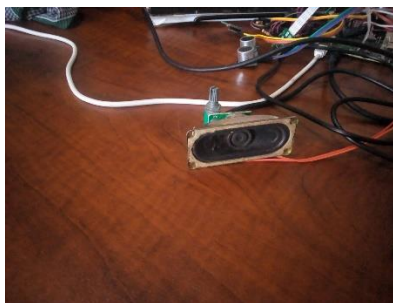
Instantiation of the class initializes the LED. During this step, the class is invoked with the pin that represents the anode. The “turn\_on” and “turn\_off” methods are used to turn the LED on and off respectively. The “cleanup” methods clean the GPIO pins. The “run” methods alternately turns the LED ON and OFF. When the code in the figure above was executed, the two LEDs employed were able to blink as required.



*Figure 25: Result for the LED Unit Test*

### **E) The Auditory feedback**

The Auditory feedback was implemented using the pygame module of 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 to play. The objective of the test was to show that the system gives auditory feedback and it performed as required.



*Figure 26: Result for the Speaker Unit Test*

### **F) The Object detection model**

The model was set to can identify objects from the camera feed limited to identifying people, vehicles, bicycles and motorcycles. The model is a Tensor Flow lite model when deployed on the Raspberry pi and was executed, it was able to identify the required objects.



Figure 27: Result for the Object Detection Model Unit Test, Using the Pi Camera

It was required that the video not to be but rather the information about the kind of the object. This information was presented by different icons shown on the GUI display. For this purpose, the code in the figure below was used producing the results shown on the GUI. To achieve the objective of only changing the icon used to represent the target, we processed the detector output as follows; we looped through all detections and looked up the name of the detected objects. When the detected object is not among the required detectable objects discard 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 28: Code Compilation for the Object Detection Processing Code

The output of the code is the name of the icon corresponding to the detected target objects. In the figure below '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.

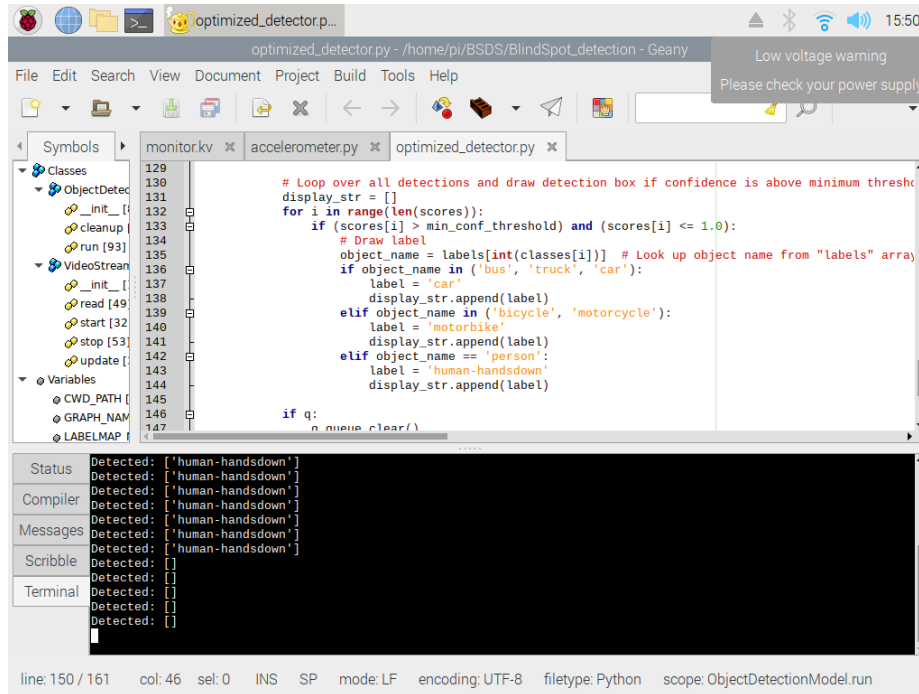


Figure 29: Results from the Object Detector post processing

### 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 fits together to meet the system mission statement.

The figure below shows the fully integrated unit. The components are labelled as shown in te figure below.

Table 9: BSDS System Integration

1	Speaker
2	Ultrasonic sensor (Right Hand Side)
3	Pi Camera (Right Hand Side)
4	LED (Right Hand Side)
5	3D Casing

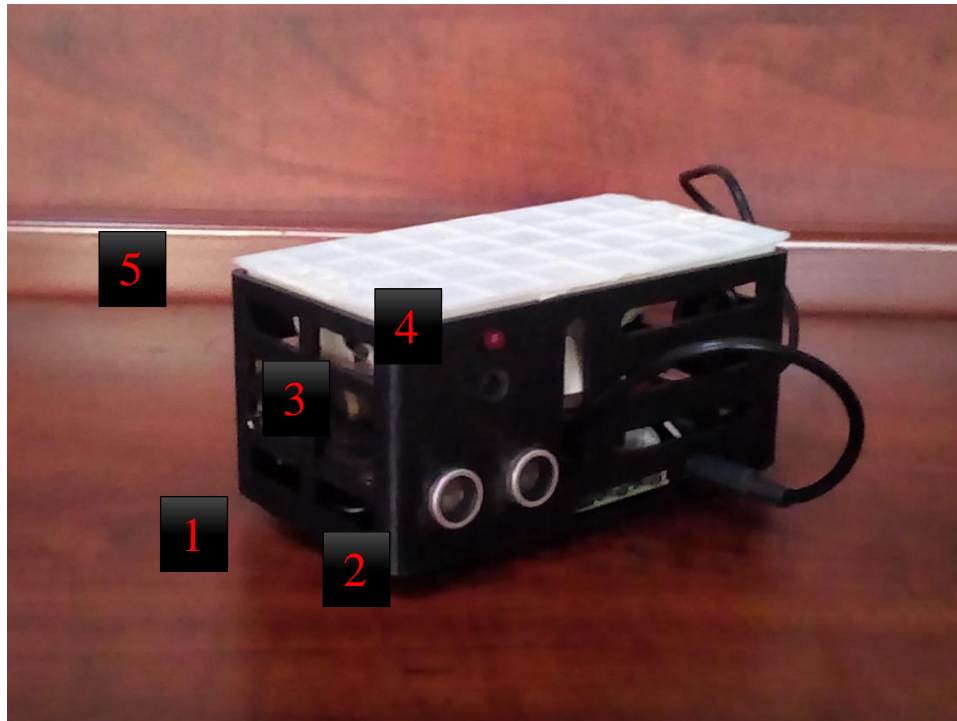


Figure 30: Fully Integrated BSDS

The system test was accomplished using a Requirement Traceability Matrix in the table below. 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. Each requirement has a unique ID; “REQF” and “REQN” representing the functional and nonfunctional requirements respectively. All requirements were 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.

Table 10: Requirement Traceability Matrix

SN	Sub-System	ID	Requirements Description	Ranking	Status
1.	Software	REQF001	Shall provide 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	Essential	Done

			well as formats that can be interpreted by the user.		
		<b>REQF004</b>	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
		<b>REQF005</b>	Shall be capable of initiating auditory feedback when the distance between the bus and object gets smaller than the threshold [1m].	Essential	Done
		<b>REQF006</b>	The sound intensity of the auditory feedback shall increase when the target object gets closer to the body of the bus.	Essential	Done
		<b>REQF007</b>	Shall start on system start-up.	Essential	Done
		<b>REQF008</b>	Shall be activated when motion of the bus has been detected.	Conditional	Done
		<b>REQF009</b>	Shall seamlessly interact with the hardware sub-systems which include; - the Raspberry Pi and the peripheral devices.	Essential	Done
		<b>REQT001</b>	Shall not to fail on regular purposes (due to failure to input values from the sensors), only at extreme bugs.	Conditional	Done
		<b>REQT002</b>	Shall withstand component and environmental failures.	Conditional	Done
		<b>REQT003:</b>	The functions of the software shall be easily understood by the user (the driver).	Conditional	Done
		<b>REQT004</b>	The response for sensor inputs shall be relatively low [1s].	Essential	Done
		<b>REQT005</b>	Shall use relatively optimum system resources, such as memory, CPU and disk.	Conditional	Done
		<b>REQT006</b>	Shall identify the root cause of failure when it occurs.	Conditional	Done
		<b>REQT007</b>	Shall be easily tested for any desired features.	Essential	Done



		<b>REQT008</b>	Shall be readily installable on the Raspberry Pi	Essential	Done
		<b>REQT009</b>	Shall conform to the Linux OS of the Raspberry Pi.	Essential	Done
		<b>REQT010</b>	There shall be ease in replacement of the different software components at any desired time.	Conditional	Done
		<b>REQT011</b>	Shall require minimum attention of the user (i.e., driver does not need to continuously glance at the display) so they can focus on other tasks.	Conditional	Done
		<b>REQT012</b>	Shall present a user interface which is slick, intuitive and attractive.	Conditional	Done
		<b>REQT013</b>	Shall notify user in the event that the system fails.	Essential	Done
<b>2.</b>	<b>Hardware</b>	<b>REQF010</b>	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
		<b>REQF012</b>	The accelerometer shall be able to measure the presence or absence of motion to provide system power on, off or sleep mode regardless of the different environment variations (for example temperature and background noise).	Essential	Done
		<b>REQF013</b>	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
		<b>REQF014</b>	The Camera shall capture frames from the blind spot areas that shall be fed to the object detection model.	Essential	Done

		<b>REQF015</b>	The LEDs shall illuminate at the start of the bus to show that they are in proper working conditions. They shall blink when there a body at close proximity with the body of the bus.	Essential	Done
		<b>REQF016</b>	The Speaker shall produce an alarm when an object or vehicle is in close proximity to the body of the bus.	Essential	Done
		<b>REQT014</b>	When an unpredictable failure occurs in reading values from either the accelerometer or the ultrasonic sensor, system shall recover briefly to full capacity or to safe mode respectively.	Essential	Done
		<b>REQT015</b>	The system shall be able to handle many inputs from its environment.	Essential	Done
		<b>REQT016</b>	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. Conclusions, Challenges and Recommendations**

This chapter contains concluding remarks of the project that is; the conclusions challenges met and how they were handled.

### **5.1 Conclusions**

From the Results presented in chapter 4, a Blind Spot Detection and Monitoring System was developed with a functional prototype using a Raspberry Pi, Ultrasonic sensors, and Accelerometer, a Camera and a Screen.

The system realized automation by having the functionality of system startup with the Accelerometer detecting motion. Using the object detection functionality, the user is able to identify the object in a blind spot area. And with the Ultrasonic sensors, the user is able to map out the distance the object is from the vehicle.

In order to minimize on the complexity of the project, we developed the system with the peripheral mentioned. These however cannot be used in deployment of the project further to the bus. The functionality that the system had at the end of this project can be used for low velocity applications for example Park Assist and Lane Maneuvering.

### **5.2 Challenges**

Training a large data set during the development of the Object Detection Model was a long process and the resultant models had to be scaled down so as to run on the Raspberry Pi. Because of this, we decided to explore other models and optimize them so as to achieve object detection as was a requirement for the system.

The Raspberry Pi has one slot for the camera module and so we were not able to realize object identification on both sides of the vehicle (in the prototype). We resigned to using one camera as a proof of concept to the functionality of the project.

### **5.3 Recommendations**

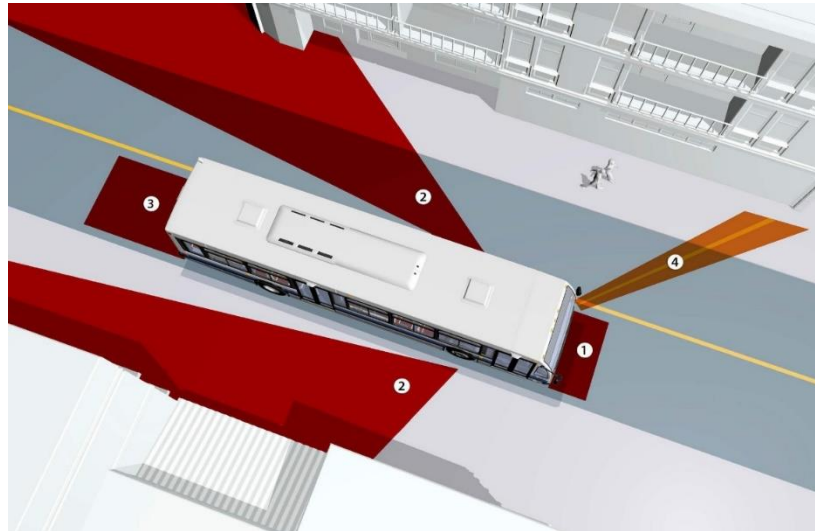
For future advancements to the project, USB Camera can be used with a corresponding Ultrasonic sensor so as to demonstrate the functionality of identifying the detected object of the Ultrasonic sensor.

## 6. Further Work

There are a number of tasks that need to be completed in order to have the system ready for deployment on the buses. The output of this project is a proof of concept that such a system can be realized. In this section, aspects that are required for the project to be deployed are discussed.

### 6.1 Identification of Blind Spot Regions on the Bus

Depending on the length of the vehicle, the exact blind spot areas may differ. 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.



*Figure 31: Blind Spot Regions*

### 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. 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 as final phase of embedded system development. The sensors need proper packaging for them to function as desired in the operational environment. This would require a throughout study of the operational environment. The main goals of packaging are offering mechanical protection, cooling features, safety and capabilities for mobility.

### 6.4 Sensor Selections

The ultrasonic sensors and the camera used in this project were well suited for demonstrating the concept but cannot be realized for deployment onto the bus due to ranging issues with the

ultrasonic sensor and the resolution issues with the camera. This is not practical, and calls for sensor of longer range or even a sensor of different technology such as Radar sensors.

### **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 concerns discussed in the chapter, the system will have to be tested in the bus and on the road.

## References

- [1] “Everything You Need to Know about Car Safety Features.” <https://www.caranddriver.com/features/g27612164/car-safety-features/> (accessed Feb. 01, 2022).
- [2] P. Lightweight and N. Network, “Camera-Based Blind Spot Detection with a General Purpose Lightweight Neural Network,” 2019, doi: 10.3390/electronics8020233.
- [3] F. Collision and O. Detection, “Forward Collision and Overtaking Detection †,” pp. 1–19, 2020, doi: 10.3390/s20185139.
- [4] 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.
- [5] N. De Raeve, M. De Schepper, J. Verhaever, 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.
- [6] 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.
- [7] 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.
- [8] P. Pyykonen, A. Virtanen, and A. Kyytinen, “Developing intelligent blind spot detection system for Heavy Goods Vehicles,” pp. 293–298, 2015.
- [9] “Sensor Technology in Autonomous Vehicles | Encyclopedia.” <https://encyclopedia.pub/9236> (accessed Feb. 04, 2022).
- [10] “3 types of autonomous vehicle sensors | Itransition.” <https://www.itransition.com/blog/autonomous-vehicle-sensors> (accessed Feb. 04, 2022).
- [11] “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 Feb. 04, 2022).
- [12] “What Is Deep Learning? | How It Works, Techniques & Applications - MATLAB & Simulink.” <https://www.mathworks.com/discovery/deep-learning.html> (accessed Feb. 04, 2022).
- [13] “Computer Vision - MATLAB & Simulink.” [https://www.mathworks.com/discovery/computer-vision.html?s\\_tid=srchtitle\\_computer\\_vision\\_3](https://www.mathworks.com/discovery/computer-vision.html?s_tid=srchtitle_computer_vision_3) (accessed Feb. 04, 2022).
- [14] “What Is Object Detection? - MATLAB & Simulink.” <https://www.mathworks.com/discovery/object-detection.html> (accessed Feb. 04, 2022).