Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

Design and Implementation of an Autonomous-Ready Electric Motorcycle Using Drive PX2

Kleanthis Kyriazakis

Submitted in partial fulfillment of the requirements for the Diploma in Electrical and Computer Engineering of the Technical University of Crete

July 2023

Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Σχεδίαση και Ανάπτυξη μιας Autonomous-Ready Ηλεκτρικής Μοτοσυκλέτας με χρήση του Drive PX2

Κλεάνθης Κυριαζάκης

Η εργασία εκπονήθηκε στο πλαίσιο των απαιτήσεων για την απόκτηση Διπλώματος Μηχανικού από την Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πολυτεχνείου Κρήτης

Ιούλιος 2023

Thesis Committee

Michail G. Lagoudakis, Ph.D. *Professor*	School of Electrical and Computer Engineering, Technical University of Crete
Eftychios Koutroulis, Ph.D. *Professor*	School of Electrical and Computer Engineering, Technical University of Crete
Eleftherios Doitsidis, Ph.D. *Assistant Professor*	School of Production Engineering and Management, Technical University of Crete

Abstract

The history of mass-production vehicles has been moving in two directions in recent years: electrification and autonomous driving. In fact, the combination of these two features looks set to dominate in the coming decades, as a large amount of resources is devoted to the development of these technologies. Responding to this trend, the motorcycle industry has recently been moving in the same directions, already presenting the first prototypes with impressive safety features in terms of balance and ride. The present diploma thesis proposes a low-cost, autonomous navigation system, designed for a prototype, autonomous-ready electric motorcycle that is being developed at the Technical University of Crete. The proposed system is based on Nvidia's Drive PX2 computing platform, which acts as the main brain of the vehicle, as well as two high-resolution cameras, which act as the main perception sensors. Using the Python programming language and the TensorFlow library, a real-time visual object detection system, based on a convolutional neural network, was developed, which specializes in vehicle and road traffic signal detection. At the same time, an Adaptive Cruise Control system was developed for the autonomous adjustment of the speed and the steering of the motorcycle, based on the perception of the current situation on the road. Information and data are displayed to the motorcycle rider via a touchscreen display, which offers various functions through a friendly graphical user interface, as well as analysis of specific scenarios. The interconnection of all subsystems, including the power supply from the motorcycle batteries, is achieved through a specially designed circuit and appropriate wiring. The proposed system is ready to be installed on-board in the prototype, autonomous-ready electric motorcycle, once its mechanical part is designed and tested for conventional driving.

Περίληψη

Η ιστορία των οχημάτων μαζικής παραγωγής κινείται τα τελευταία χρόνια προς δύο κατευθύνσεις: ηλεκτροκίνηση και αυτόνομη οδήγηση. Μάλιστα, ο συνδυασμός αυτών των δύο χαρακτηριστικών φαίνεται ότι θα κυριαρχήσει στις επόμενες δεκαετίες, καθώς αφιερώνεται μεγάλος όγκος πόρων στην ανάπτυξη αυτών των τεχνολογιών. Ανταποκρινόμενη σ' αυτήν την τάση, στις ίδιες κατευθύνσεις κινείται πρόσφατα και η βιομηχανία μοτοσυκλετών, παρουσιάζοντας ήδη τα πρώτα πρωτότυπα με εντυπωσιακά χαρακτηριστικά ασφάλειας ως προς την ισορροπία και την οδήγηση. Η παρούσα διπλωματική εργασία προτείνει ένα σύστημα αυτόνομης πλοήγησης χαμηλού κόστους, σχεδιασμένο για μια πρωτότυπη, autonomous-ready ηλεκτρική μοτοσυκλέτα που αναπτύσσεται στο Πολυτεχνείο Κρήτης. Το προτεινόμενο σύστημα βασίζεται στην υπολογιστική πλατφόρμα Drive PX2 της Nvidia, η οποία λειτουργεί ως ο κύριος εγκέφαλος του οχήματος, καθώς και σε δύο κάμερες υψηλής ανάλυσης, οι οποίοι λειτουργούν ως κύριοι αισθητήρες αντίληψης. Με τη χρήση της γλώσσας προγραμματισμού Python και της βιβλιοθήκης TensorFlow, αναπτύχθηκε ένα σύστημα οπτικής ανίχνευσης αντικειμένων σε πραγματικό χρόνο, βασισμένο σε ένα συνελικτικό νευρωνικό δίκτυο, το οποίο ειδικεύεται στην ανίχνευση οχημάτων και σημάτων οδικής κυκλοφορίας. Παράλληλα, αναπτύχθηκε και ένα σύστημα Adaptive Cruise Control για την αυτόνομη ρύθμιση της ταχύτητας και της διεύθυνσης της μοτοσυκλέτας, βάσει της αντίληψης της τρέχουσας κατάστασης στον δρόμο. Πληροφορίες και δεδομένα προβάλλονται στον αναβάτη της μοτοσυκλέτας μέσω μιας οθόνης αφής, η οποία προσφέρει διάφορες λειτουργίες μέσα από ένα φιλικό γραφικό περιβάλλον χρήστη, καθώς και ανάλυση συγκεκριμένων σεναρίων. Η διασύνδεση όλων των υποσυστημάτων, συμπεριλαμβανομένης και της τροφοδοσίας από τις μπαταρίες της μοτοσυκλέτας, επιτυγχάνεται μέσω ενός ειδικά σχεδιασμένου κυκλώματος και κατάλληλης συνδεσμολογίας. Το προτεινόμενο σύστημα είναι έτοιμο για τοποθέτηση on-board στην πρωτότυπη, autonomous-ready ηλεκτρική μοτοσυκλέτα, μόλις ολοκληρωθεί η σχεδίαση και η δοκιμή του μηχανολογικού μέρους της για συμβατική οδήγηση.

Acknowledgements

I would like to express my deepest appreciation to my thesis supervisor, Professor Michail G. Lagoudakis for the extended assistance and constructive advice he provided on the current thesis. I cannot begin to express my thanks to Professor Eleftherios Doitsidis for playing a decisive role in the completion of this work and his invaluable encouragement and suggestions throughout the duration of this project. I would also like to thank Professor Eftychios Koutroulis, a member of my thesis committee for helpful advice reviewing the manuscript. I would also like to extend my deepest gratitude to my family for their wise counsel and sympathetic ear. You are always there for me. I am extremely grateful to all the new friends I made, Nicholas, John, George, Sifis, Vasilis, Nikos, who have always been a major source of support and for all their help in the conceptualization and implementation of the work presented here. Finally, I could not have completed this dissertation without the help of Dr. Polychronis Spanoudakis and members of the TUCer team. This thesis would not be possible without your support.

Contents

List of Figures

Acronyms and Abbreviations

ACC	Adaptive Cruise Control
ADC	Analog to Digital Converter
API	Application Programming Interface
ARM	Advanced RISC Machine
AV	Autonomous Vehicle
BB	Bounding Box
CAN	Controller Area Network
CES	Consumer Electronics Show
CV	Computer Vision
DC	Direct current
dGPU	Dedicated Graphics Processing Unit
DNN	Deep Neural Network
FMCW	frequency-modulated continuous-wave
FPS	Frames Per Second
GMSL	Gigabit Multimedia Serial Link
GPU	Graphics Processing Unit
GTC	GPU Technology Conference
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
iGPU	Integrated Graphics Processing Unit
INT8	8-Bit Integer
IO	Input and Output
IoT	Internet of Things
IR	Infrared
ISP	Image Signal Processor
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LiDAR	Light Detection And Ranging
ML	Machine Learning
NAS	Smart Network Attached Storage
NASA	National Aeronautics and Space Administration
OBD	On-board diagnostics
OPS	Operations Per Second
OTA	Over-The-Air
PCAP	Projected Capacitive
PPS	Points Per Second
PVA	Programmable Vision Accelerator

Radar	Radio Detection And Ranging
RCCB	Red Clear Clear Blue
RISC	Reduced Instruction Set Computer
SAE	Society of Auto-motive Engineers
SDK	Software Development Kit
SNR	Signal to Noise Ratio
SoC	System on a Chip
Sonar	Sound Navigation And Ranging
TOT	Time of Transmission
WVGA	Wide Video Graphics Array

Chapter 1

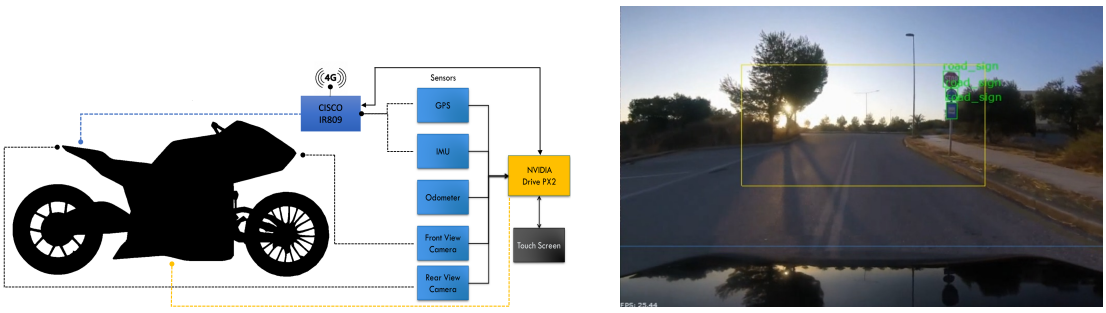
Introduction

1.1 Thesis Motivation

In recent years, advances in driving technology as well as challenges in urban transportation created opportunities for development of fully autonomous vehicles and have attracted numerous manufacturers that invest immense amounts of resources and time [1]. Currently, it has evolved into a race with the ultimate goal of being the first to make this dream come true. It is not certain when fully autonomous vehicles will enter mass production, and some would say that it is a question of "if", rather than "when". A rational scenario would be for most people to simply order a driverless vehicle in order to travel to a location they want, while having the opportunity to read a book, watch TV, answer e-mails or even take a nap. In urban areas with driverless cars, the human error while driving is eliminated which makes our transportation a lot safer and the number of vehicles is reduced, positively impacting the environment. With the rise of electric vehicles with autonomous capabilities, the electric motorcycle industry also attracted a lot of interest that continues to grow on a daily basis.

1.2 Thesis Contribution

This thesis presents an autonomous-ready system (Figure 1.1) designed for a prototype three-wheeled electric motorcycle using Nvidia's Drive PX2 as the main processing unit and Grayhill's 3D70 display for human-computer interaction.



(a) System concept configuration (b) Onboard view during test run

Figure 1.1: Autonomous system and onboard footage

The development workflow of this work is presented in Figure 1.2 and its main goals are the following:

- A hardware and software configuration using Nvidia Drive PX2, that utilizes two high resolution cameras, for object detection and recognition applications.
- The development of an object detection and recognition system, that can analyze the camera feed and perceive the environment surrounding the motorcycle.
- An Adaptive Cruise Control system, which is simulated and tested in the CARLA virtual environment in order to verify and improve its behavior, with the intention being its deployment to the Drive PX2 system, as a smart driver assistance system.
- Implementation of a traffic sign recognition model that leverages the object detection system mentioned earlier. Its role is important on traffic assistance driving systems and automatic driving systems.
- The design and implementation of a Graphical User Interface with multiple functionalities intended for the Grayhill 3D70 touchscreen display. The display provides the driver with necessary information the Drive PX2 system yields, as well as grant situational awareness by displaying the real-time camera feed to the user.
- A method to link the above sub-systems into a single autonomous-ready system, to be deployed on the electric motorcycle.

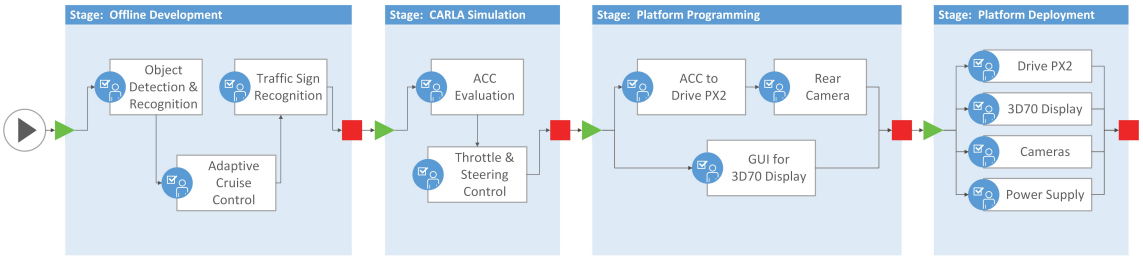


Figure 1.2: Development Workflow of this work

1.3 Thesis Outline

The rest of this thesis is organised as follows.

In Chapter 2 we explain self-driving vehicles and the way they are categorized, as well as some of the characteristics featured. Afterwards, various sensors widely used in autonomous vehicles are discussed and their operation logic is described. Some of those sensors are Odometry, Inertial Measurement Units, Cameras, GNSS, Radar and so on. Succeeding this, the workings of some communication protocols adopted in modern vehicles, such as CAN, LIN and FlexRay busses are explained. Next, Nvidia's DRIVE platform, along with its most compelling features offered, is discussed.

Chapter 3 concerns similar autonomous applications on motorcycles and autonomous vehicles that utilize Nvidia's DRIVE platform.

Chapter 4 describes the platform used in this work, including specifications of the electric motorcycle, Drive PX2 system, cameras and Grayhill 3D70 display.

Chapter 5 includes details concerning software development of systems for offline object detection, traffic sign recognition and adaptive cruise control, and offline experiments conducted in the Carla Simulator in order to test the algorithms in a virtual urban area.

Chapter 6 includes details about deploying the above algorithms on the DRIVE PX2 platform, and also design and testing of the Graphical User Interface for the 3D70 display.

Chapter 7 specifies the proposed key components for the final deployment of our system on the motorcycle including schematics and PCB designs, and details on how they connect and interact with each other.

In the final chapter we discuss future work to extend this system's capabilities and features.

Chapter 2

Background

A self-driving, also known as an autonomous vehicle (AV), is capable of sensing its surrounding environment and operating without requiring human interaction. By fusing the information gathered from different sensors, such as Cameras, Radars, LiDARs and odometry measurement units, it is able to perceive its surroundings. AVs can go anywhere a traditional car goes and do everything that an experienced human driver does. The Society of Automotive Engineers (SAE) currently categorizes autonomy in vehicles in six Levels [2] ranging from Level 0 (no automation) to Level 5, meaning fully autonomous requiring no human interaction.

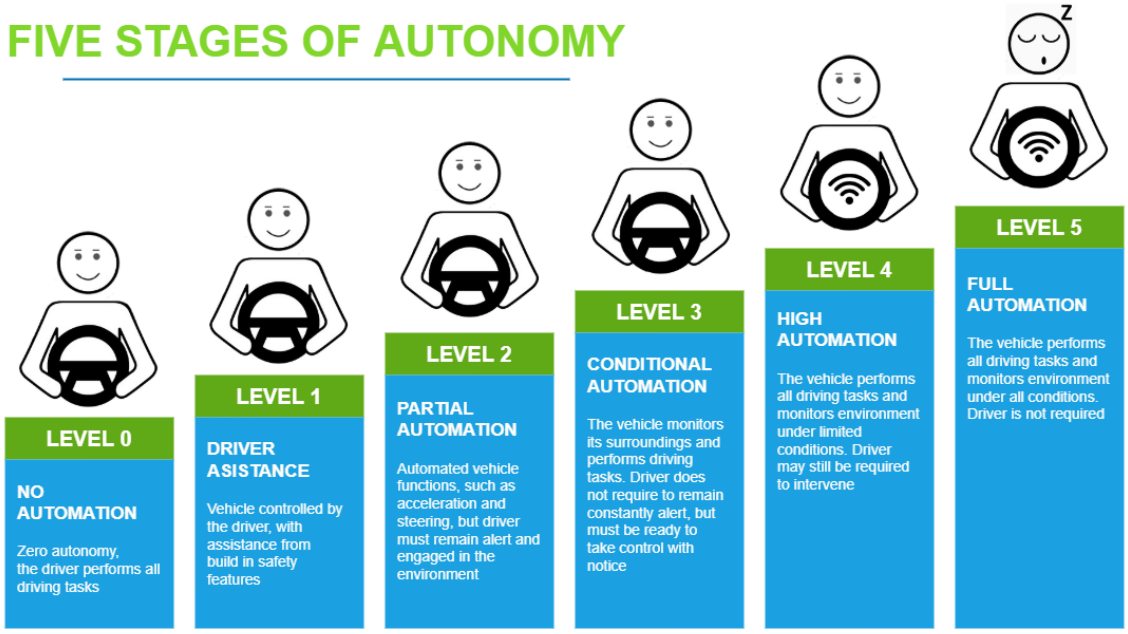


Figure 2.1: The six stages of autonomy

Autonomous vehicles rely on sensors, actuators, complex algorithms, machine learning models, and powerful processors to execute software. Modern self-driving cars generally use localization and mapping algorithms [3], which fuse data from multiple sensors to calculate current location estimates. Waymo, formerly Google's self-driving car project and now an autonomous driving technology development company, has developed a variant of those algorithms that include detection and

tracking of other moving objects (DATMO), which also handles obstacles, such as cars and pedestrians [4].

Modern vehicles provide features that help keeping the car within its lane on a highway and parking assistance that helps takes over the steering function while parking, as the driver is responsible for the throttle control. Braking and steering assistance, or any other feature that require the driver to have full responsibility for monitoring the road and taking over, if the assistance system fails, are qualified as Level 1 automation. Level 2 driving automation is considered "hands off", meaning the automated system is capable of taking full control of the vehicle's basic functions (throttle and steering control). Despite its name, the term "hands off" should not be taken literally, as it is necessary for the driver to constantly monitor the driving situation and be prepared to intervene immediately at any point if the system does not respond properly. The 3rd Level is the "eyes off" category, that allows the driver to safely turn their mind away from driving and into something else. At this point, the system is capable of resolving every situation that might arise and call for immediate action. This system, when the time comes, will alert the driver in an orderly fashion, when it is time the driver must take over control, and the driver has to be prepared to intervene within a limited time, usually specified by the system's manufacturer. The high automation 4th Level is considered "mind off" and is very similar to the previous one, with the difference being that the driver's attention will never be required for safety, and they are free to even leave the driver's seat. This kind of automation is only allowed in certain spatial areas at the time. Finally, Level 5 automation is called a "steering wheel optional" and is exactly what its name implies. A Level 5 vehicle could be a car designed to work on any kind of surface, in any weather condition, and not bound to specific locales.

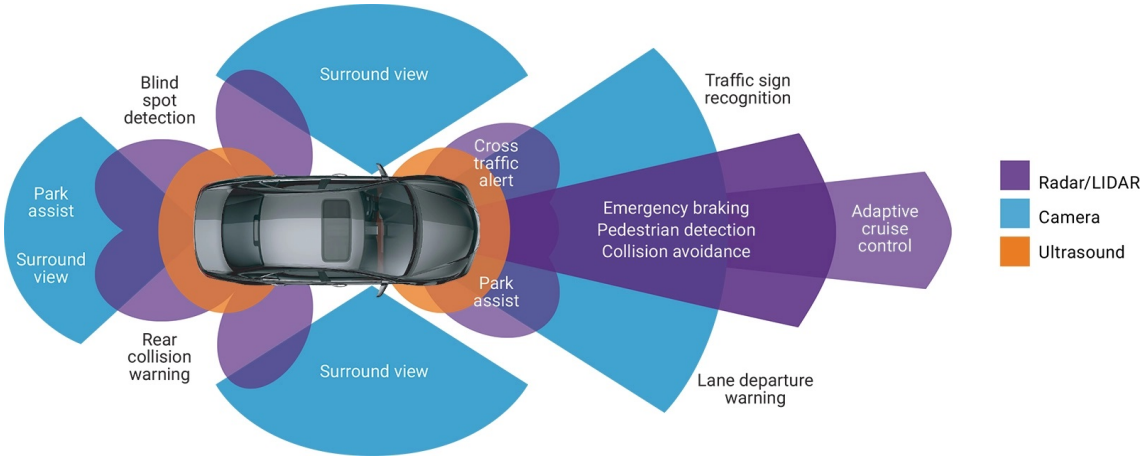


Figure 2.2: Sensor configuration of an autonomous car [5]

In Figure 2.2, a possible sensor configuration on an autonomous vehicle is presented. Cameras are used to provide a 360-degree view of the vehicle, parking assistance and object detection and recognition features. Ultrasound sensors placed in the front and rear of the car are responsible for close-range object detection mainly in tight spaces and while parking. A combination of Radar and LiDAR sensors provide high accuracy perception of distant objects with high frequency sampling. Finally, the

vehicle utilizes Global Navigation Satellite Systems (GNSS), Inertial Measurement Units (IMUs) and odometry sensors for localization in real-time.

2.1 Sensors

2.1.1 Odometry

Odometry is the estimation of relative location based on data gathered by sensors, such as Cameras or Rotation Optical Encoders, that measure wheel rotation. The most common types of sensors used is wheel odometry that calculates the vehicle's speed and location using the wheel's rotational speed. Although this method is susceptible to errors due to the integration of velocity over time, high frequency and precise measurements in addition to a calibrated sensor can make odometry measurements effective and accurate. Other sensors that can be used for odometry measurements are the Inertial Navigation System (INS), Optical Cameras, Laser Sensors, Sonar/Ultrasonic sensors and Global Positioning Systems (GPS). In Figure 2.3 we can see the visual pipeline described in [6], that determines odometry information by using sequential frames from a stereo camera setup. Acquired camera frames, undergo a feature tracking process, which include feature detection and descriptor extraction. These descriptors are used to track the correspondent features in both temporal and stereo domain. After rejecting possible outliers generated by the feature matching process, the odometry information is estimated.

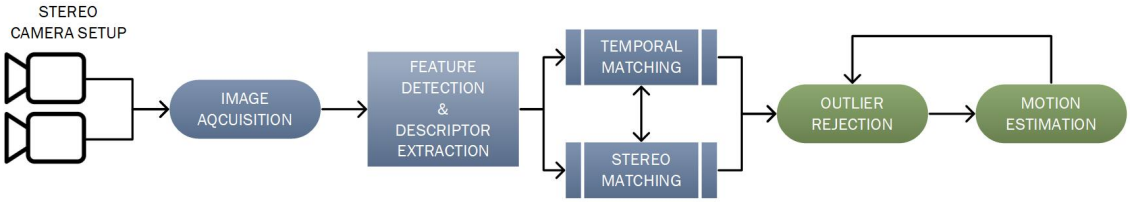
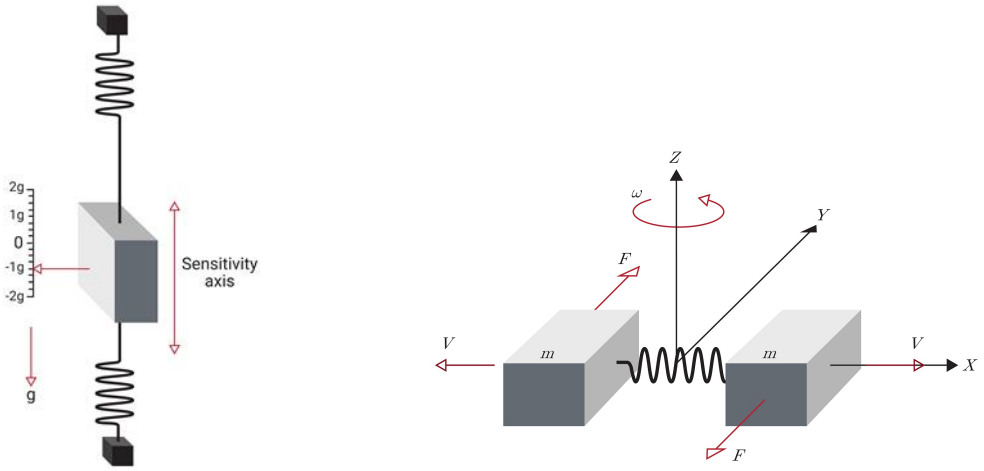


Figure 2.3: Visual Odometry Pipeline [6]

2.1.2 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is an assembly of at least three gyroscopes and three accelerometers, used to obtain inertial parameters of a moving object along the three axis. This includes its attitude parameters (yaw, pitch, roll), position, speed and usually measure the magnetic field strength along the three-dimensional axis. In aircrafts, these systems are integrated into Inertial Navigation Systems (INS) that can utilize and process measurements to constantly calculate the state of the vehicle. The IMU-equipped INS sets the foundation for the navigation and control of all commercial or military vehicles, such as aircrafts, ships, missiles, satellites etc. In land vehicles, IMUs can be found inside navigation systems and enable them to gather as much accurate data as possible about the vehicle's speed, acceleration, heading, steering rate and much more. Besides navigation, they are used for many orientation purposes in various consumer products, such as smartphones, laptops, fitness trackers and other wearables.



(a) IMU basic accelerometer model (b) IMU basic gyroscope model

Figure 2.4: IMU's sensor models [7]

An IMU's primary sensor is the accelerometer, responsible for measuring the change of velocity over time, or acceleration. They can be found in wide variety of types, including mechanical, capacitive, quartz or MEMS accelerometers with capacitive being the most common. Its basic principle, as seen in Figure 2.4a, is essentially a mass connected to two springs that react to acceleration along its sensitivity axis and shifts to either side.

Gyroscopes are the sensors responsible for angular rate measurements. There are numerous types of gyroscope architectures, with various levels of performance according to their application and include mechanical gyroscopes, fiber-optic gyroscopes (FOGs), ring laser gyroscopes (RLGs), and quartz/MEMS gyroscopes. In Figure 2.4b, we can see a simply gyroscope model, called Tuning Fork configuration, that consists of two objects of known mass connected by a spring, and can measure angular rate on along a specific axis.

2.1.3 Camera

One of the primary sensors of every autonomous vehicle is the camera. As one of the main sensors used in AVs, they are very good at high resolution tasks, like classification, scene understanding or tasks that require color perception, like traffic light or sign recognition. Nowadays, vehicles rely on multiple cameras placed on every side to stitch together a 360°degree view of their environment that provide visuals of the surroundings and help detect the speed and distance of nearby objects, as well as their three-dimensional shape. In order to acquire the best view possible, they require an unobstructed line of sight, and thus they can only be placed behind a translucent surface such as glass, that also protect them from different types of weather conditions, such as humidity, rain or temperature.

Camera sensors work on the principle of the light entering an enclosed box through a converging or convex lens and an image is recorded on a light-sensitive medium. This medium, also called the image plane, is where the light is stored forming an image. CCD and CMOS [8] are the most common type of camera sensors, but depending on the manufacturer the camera and sensor layout may differ. After

photons pass through the lens, the light sensitive surface converts those rays into electrons, those electrons are then converted to voltage, amplified, passed through an Analog-Digital Converter (ADC) and converted to a number in a metric unit or pixels (Figure 2.5). A camera is a passive type of sensor designed to detect and measure reflected natural emissions produced by constituents of the Earth's surface and its atmosphere.

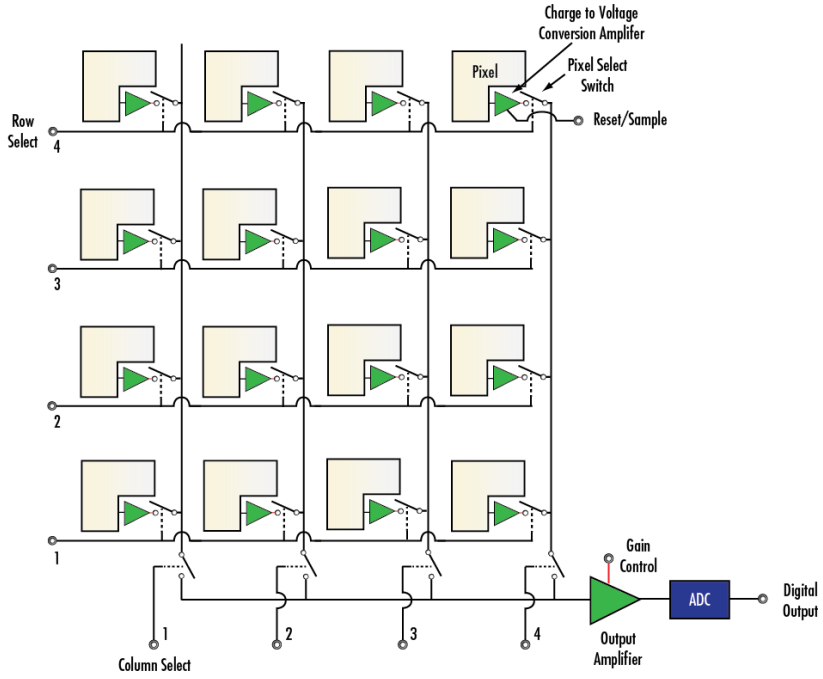
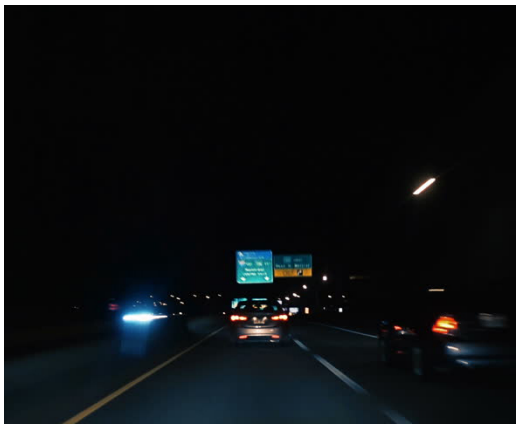


Figure 2.5: Simplified configuration of a Camera's sensor [9]

Depending on the lens type, some have a wide field of view, as much as 120°degrees, and a shorter range, while others focus on a narrower view to provide long-range visuals to a specific direction. Although they are relatively cheap compared to other sensors, like Radar or LiDAR, and provide accurate visuals, cameras have their limitations. Their working principle is similar to the human eye, that acquires light rays bouncing around and uses glass to redirect them to a single point, creating a sharp image. Thus, in cases where it is hard for humans to observe their environment, cameras also struggle. For instance, in Figure 2.6 we can see the influenced camera feed in low visibility conditions, like fog and nighttime, that render the object detection harder for camera-based sensors.

A vision sensor requires an unobstructed line of sight, which means it has to be placed in open air or behind a translucent surface (i.e. the wind shield). A vision system relies on image processing algorithms to detect and classify objects. Image processing is a demanding process, but the information that can be extracted from images is very useful and can be used for many tasks, like mapping and navigation. Camera sensors are able to capture colors, vividness, minutiae of scenes with very high resolution compared to other sensors, such as lasers, radars or ultrasound. But their biggest drawback is their lack of depth perception. Humans and animals have two eyes, that transmit information to the brain. These data are almost identical, but using the slight displacements observed, the brain can translate them to depth information. This logic can also be adapted in camera systems, in situations that

(a) Camera view during fog



(b) Camera view during night

Figure 2.6: Examples of camera limitations

depth perception is essential. In Figure 2.7 we have an example of a stereoscopic camera system connected on a Raspberry Pi system that handles all the necessary calculations.

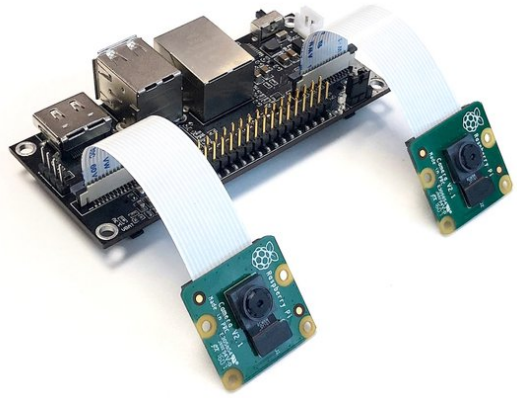


Figure 2.7: Stereoscopic camera system with Raspberry Pi [10]

The use of two or more camera sensors a set distance apart and data triangulation of similar pixels from both 2D frames, provide three dimensional measurements throughout the full field of view. Three dimensional imaging can be achieved through many different ways, with the main categories being passive and active [11]. Passive stereo imaging is dependent on the environment lighting and does not apply external light. It is a cost-effective solution with high efficiency in well-lit areas, but suffers in low visibility and non-textured scenes. Active stereo imaging overcomes these challenges by employing light using a laser or a projector. In well lit areas or in long ranges, it works similarly to passive imaging, but offers high efficiency in low-light or non-textured scenes.

2.1.4 GNSS

One of the global navigation satellite systems (GNSS) is the Global Positioning System (GPS), a satellite navigation or satnav system owned by the United States government. Global Navigation Satellite System (GLONASS) is also a Russian

space-based satnav system. Other systems include European Union's Galileo and the Chinese navigation system BeiDou. These systems use satellites to provide accurate geo-spatial positioning, by allowing GPS receivers to determine longitude, latitude and elevation data with a few centimeters precision. The original motivation for navigation using satellites was many military applications that allow precise delivery of weapons. Nowadays, such systems are used to determine the user's location or track an object at any given moment and provide numerous sectors, such as science, aviation or agriculture, with the means to explore, map and analyse regions across the entire globe.

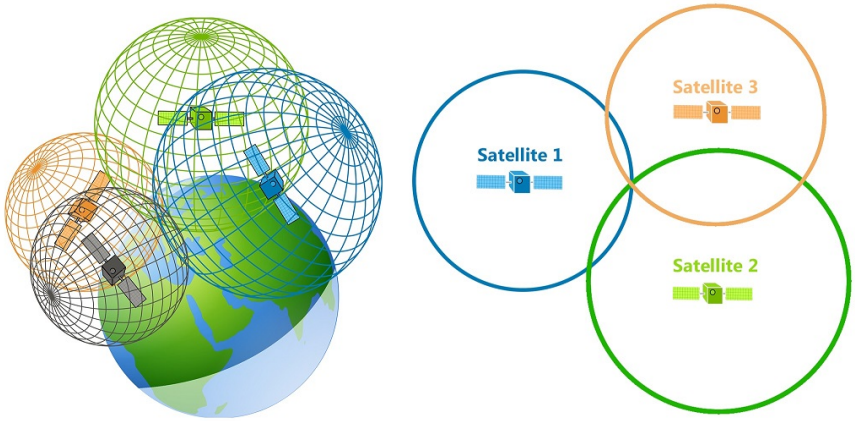


Figure 2.8: Geo-spacial Positioning using Trilateration [12]

GPS satellites circle the earth every twelve hours in a precise orbit. The Unites States currently have 31 active satellites in orbit, making it impossible for someone with a GPS receiver to get lost, no matter where they are. Every satellite is equipped with a very stable atomic clock synchronized with one another and with ground clocks and acts as a transmitter that constantly transmits a unique signal that contains its orbital parameters and a message that includes the time of transmission (TOT). A GPS receiver device, such as a smartphone, collects the signal and using the included TOT data, calculates the distance from each satellite. As demonstrated in Figure 2.8, using three distances, trilateration can pinpoint a precise location at the point where all circles intersect. With a minimum of 3 satellites, the GPS receiver can get a two-dimensional (horizontal) fix by making the assumption that the current altitude is at mean sea level. For three-dimensional fix, a minimum of 4 satellites are required, though typically many more than that are used in order to increase accuracy.

Autonomous vehicles use Radars, LiDARs and Cameras to read and understand the environment around them and sense movements with Inertial Measurement Units. Using these sensors they can navigate through towns and highways with safety, while following the traffic laws. However, in order to geolocate themselves around the entire globe and perform long range path-planning, they need a high precision Global Navigation Satellite System (GNSS). This technology provides the high accuracy and reliability an autonomous vehicle needs. GNSS system data is usually fused with data from other sensors, such as IMUs [13], using a processing unit, in order to

further increase the data accuracy. They can also be used in the development and assessment of new maps, as well as smart path finding, in order to get around traffic jams, accidents or environment obstacles from data gathered by other AVs.

2.1.5 Radar

Radio detection and ranging (Radar) is a detection system that actively transmits and receives electromagnetic waves in the microwave range (1 to 1000 GHz). A radar system usually consists of an antenna used both for transmitting and receiving and a processor used to analyse the collected signal and determine the properties of the object. Radar sensors collect the waves reflected by an object that return to the receiver, and can detect, track and position an object along with its speed.

Radar was first developed for military application before the second World War, and initially was used to aim searchlights and later to aim anti-aircraft guns. The end of the war precipitated researchers to improve resolution and portability. In modern aircrafts, as well as in air traffic control, pulse-doppler radars are used in order to detect detect aircrafts, ships, weather formations, and terrain. Automotive radars are of smaller size and needed for collision avoidance, pedestrian and cyclist detection systems. The technology generally used for these applications is frequency-modulated continuous-wave (FMCW), which is quite different from the pulse-Doppler radar. In the following figure (Figure 2.9), a simplified Radar architecture is presented that illustrates the main components of a Radar sensor.

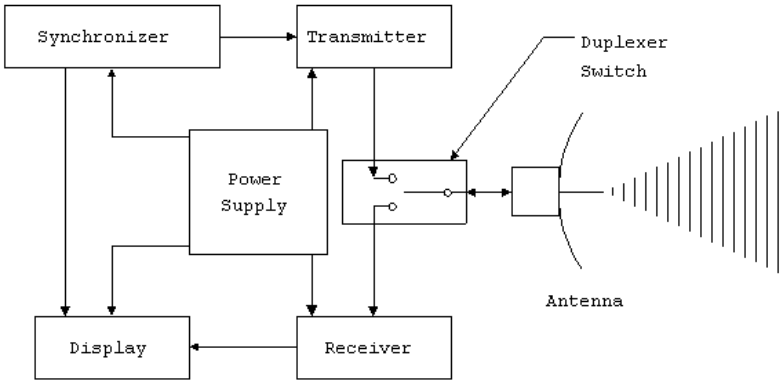


Figure 2.9: Simplified Radar architecture [14]

Many high-tech radar systems are associated with digital signal processing, machine learning and are able to perceive useful information from very high noise levels. In the automotive industry, radar systems are the primary sensors used in driving assistance systems, such as adaptive cruise control, collision avoidance or pedestrian detection. As one of the main sensors used in modern vehicles, it plays an important role in long distance detection. With a typical range of 300 to 500 meters, it can reliably measure an object's distance and relative speed. Due to radio wave properties, it remains effective through rain, fog, smoke and of course nighttime. Finally, Radar is one of the few sensors that provide the flexibility to function even when placed behind opaque surfaces, that systems like cameras, LiDAR or ultrasonic sensors disallow.

Automotive Radar sensors are categorized based on their detection range into Short and Long Range radars [15]. Short-Range radars are commonly used in obstacle avoidance or blind-spot detection systems with a maximum detection range of 30 meters. They use the 24GHz frequency band, can be placed around the vehicle to increase the detection region and are typically inexpensive compared to their counterparts. Long-Range Radar systems use the higher frequency band of 77GHz and provide higher resolution with greater range. Typical ranges can reach 200 meters, and can calculate distance to other vehicles as well as their speed. Additionally, they can offer long range obstacle detection in cases where an obstruction is indistinguishable by a Camera sensor or the human eye.

2.1.6 LiDAR

Light Detection and Ranging (LiDAR) is a remote sensing technology for measuring ranges by using the pulse of a laser. Also known as 3D scanning or laser scanning, it can be used to create digital 3-D representations and map of various environments and ocean surfaces. The concept of LiDAR was conceived in 1930 and involved the use of powerful searchlights to probe the atmosphere. Since then, LiDAR's major purpose was atmospheric research and meteorology. A LiDAR system mounted on an aircraft or satellite, can carry out different surveying and mapping tasks. Recently, NASA described LiDAR as the key technology for enabling developing autonomous systems to maneuver and perform precision landing with future robotic and crewed lunar-landing vehicles. A complete LiDAR system used in mapping includes the laser scanning system and is integrated with an Inertial Measurement Unit (IMU) and a GPS receiver. Apart from mapping and environment applications, today it can be used in Architecture, real estate and construction to survey buildings, road and railway networks, scan and produce floor-plans and accurate 3D models for constant monitoring of structures, for architects and structural engineers.

Autonomous vehicles use all kinds of sensors to "see" around them such as cameras, ultrasound sensors, radars, etc, but are limited in terms of range, depth and coverage area. LiDAR technology can provide a 360-degree view of the environment (Figure 2.10) with further depth and detail than other solutions, high efficiency at night and low light conditions, and thanks to recent developments, the ability to detect objects with a 250 to to 400 metres maximum range. Although considered a game-changer equipment for driving assistance system developers, LiDAR comes with a few downsides. The data captured are recreations of reality rather than photos, which means a vehicle system can be vunerable to tampering or manipulation, and thus have weaker security compared to images captured from a camera sensor. At the moment, Artificial Intelligence and Machine Learning algorithms with LiDAR data input that gradually learn over time are only at research stage. Finally, LiDAR requires moving parts working in high precision that is easy to malfunction or brake, thus its cost of acquisition and maintenance is high.

The main component of a LiDAR system is the laser transmitting and receiving systems along with their common or individual optical lens. The laser sends out light pulses of near-infrared wavelength of usually 905 or 1550 nanometers and measure how long it takes for the receiver to detect the reflected pulse. The data processing system combines the direction and calculated distance of each pulse to create a

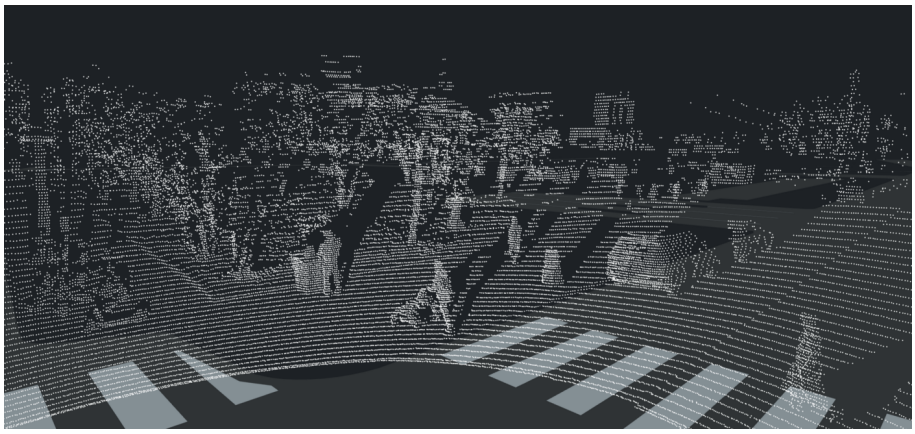


Figure 2.10: LiDAR captured point-cloud view [16]

point-cloud representation. The LiDAR laser system emits pulses with a speed of thousands to millions pulses per second, in a circle covering a 360 degree area, while also moving up and down. This way, it generates a real-time 3D representation of the environment to be used by the vehicle's computer for safe navigation.

Different LiDAR sensors exist for various applications, with different vertical and horizontal field of view, depending on the area we need to cover. The number of channels can also vary, with more channels producing a denser point-cloud and providing a more detailed view, but increasing the cost. Other features include higher Points Per Second and safety certification for systems that operate amongst humans. In Figure 2.11, the simplified LiDAR architecture is presented, with the basic components, as well as the data processing systems required.

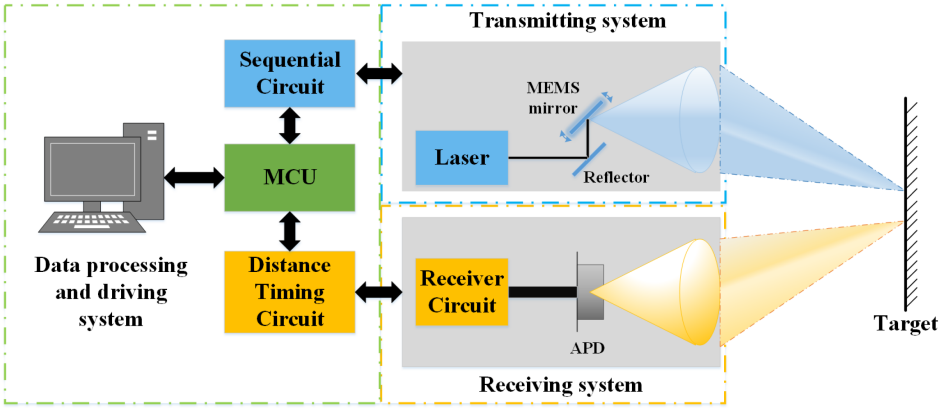


Figure 2.11: Simplified LiDAR sensor diagram [17]

Evidently, the LiDAR technology is very similar to radar's, but it is essential to identify their differences. While their purpose is the same, LiDAR uses light waves and RADAR systems use radio waves. The main advantage of radio waves is their capacity to easily reflect on surfaces and are not as easily absorbed, and ergo can work on long distances unaffected by fog or clouds. On the other hand, LiDar sensors use shorter wavelength of light waves, they are very effective in shorter distances and offer very high accuracy. But the light beam can be easily absorbed and reflected by small particles, thus LiDARs cannot see through fog, dust, rain or snow. Figure

2.12 illustrates the differences between a LiDAR and a high resolution Radar in clear-day conditions.

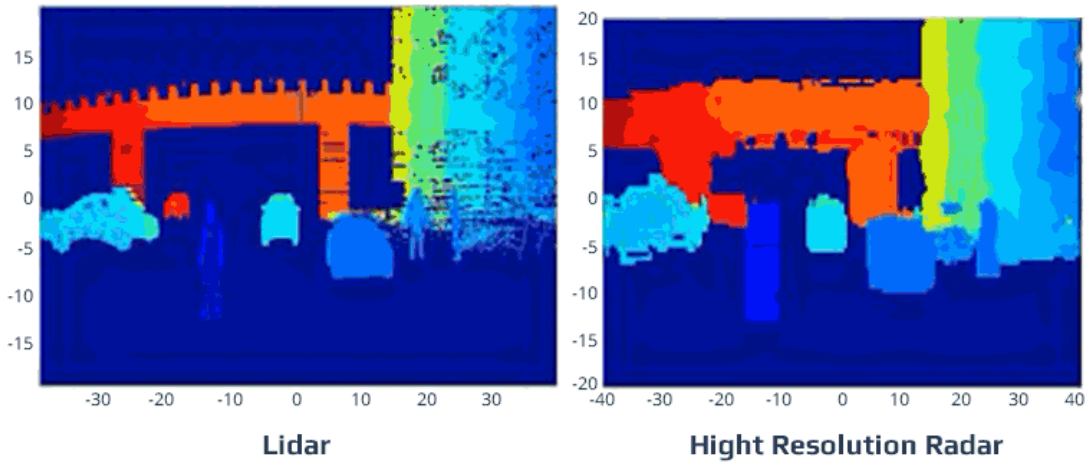


Figure 2.12: LiDAR and Radar accuracy [18]

2.1.7 Ultrasound

Sound Navigation and Ranging (Sonar) technology is usually adopted for underwater applications, such as navigation, distance measuring, object detection or even communication with other vessels. It is similar to Radar and LiDAR techniques, but is based on sound propagation instead of emitting electromagnetic waves. Although a great number of animals use sound for communication and detection, it was first used by humans in 1490 in an experiment designed and recorded by Leonardo da Vinci. The first listening device based on Sonar was invented in 1906 and was used to detect icebergs. A few years later, the design was revised and a Sonar based system to detect submarines was invented. The wave frequencies used in these systems vary from low (infrasonic) to very high (ultrasonic). Ultrasonic sensors work by transmitting a sound pulse at a specific frequency, which propagates through the air and will eventually bounce of an object back to the sensor. Figure 2.13 demonstrates modern ultrasonic sensors used in automotive industry.

Nowadays, vehicles are equipped with ultrasonic sensors primarily used to cover driver's blind spots as well as for parking assistance. With the rise of research and development of autonomous vehicles, ultrasonic sensors are used alongside other sensors to create a full picture of the vehicle's surroundings. Due to the way sound waves travel, even with directional transmitters, all we can do is limit the spread but not eliminate it completely, and thus their resolution is limited. They are mainly used for close proximity object detection and usually placed around the vehicle to provide 360 degree coverage. Tesla uses twelve ultrasonic sensors in their vehicles that allow detection of objects up to five meters. Data from these sensors, fused with information gathered from other available sensors, such as Cameras and Radar, as illustrated in Figure 2.14, is fed to autopilot and driver assistance systems.

Figure 2.13: Automotive Ultrasonic Sensors
Source: BOSCH

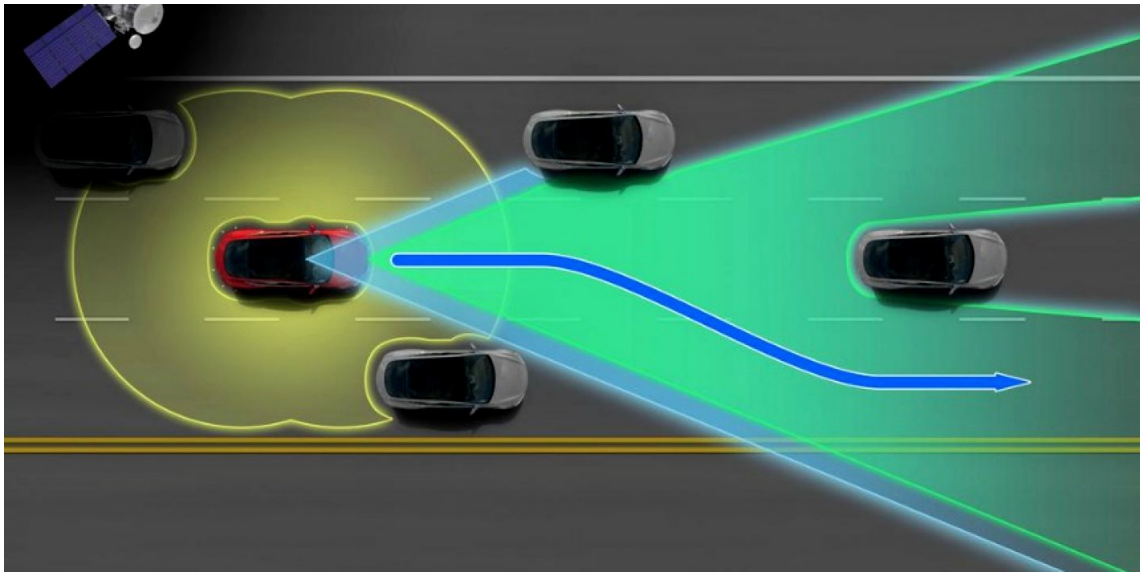


Figure 2.14: Tesla configuration: Radar(Green), Camera(Blue), Ultrasonic sensors(Yellow) [19]

A significant drawback with this technology is that sound does not propagate in the air as good as in water, especially when a vehicle travels at high-speed generating vortexes. This, combined with noise pollution found in urban areas, reduces the system's accuracy and range. On the contrary, sonars are a low-cost option that almost all modern vehicles use and are not effected by color or transparency of nearby objects, or by environmental conditions, such as dust, moisture or low visibility.

2.2 Communication Protocols

In the earlier days, the car radio was the only electronic equipment present in most vehicles. As technology advanced and safety requirements roll out, vehicles call for multiple and complex electronic modules often designed and manufactured by different companies. Modules, like Engine Control Unit, Transmission Control Unit, Anti-Lock Braking System and much more, are constantly monitoring the vehicle's state and communicate with each other. The Automotive Industry, needed a way to avoid connecting each sensor to every other module, with a low cost and reliable

solution. Thus, a local network was developed, with high and low speed bus protocols, such as CAN bus, LIN bus and FlexRay. This allows multiple modules to be plugged in the network, using the standardized connections without additional planning and development.

2.2.1 CAN

Controller Area Network (CAN) [20] is a powerful bus standard, mainly used in automotive applications, designed to allow micro-controllers and devices of any kind to interact with each other, without the need of a host computer. Over the years, with the exponential increase of sensor and control units in a vehicle, the need to keep wiring cost low lead to the invention of this message-based protocol, designed for multiplex electrical wiring. Every device is connected to a mutual bus, data is transmitted sequentially in a way that, when multiple devices transmit at the same time, the one with the highest priority will continue and the rest will back off. Additionally, this standard can aid the fault diagnosis throughout the vehicle through an On-board diagnostics (OBD) port. A typical CAN configuration connecting all the important nodes in a car, is illustrated in Figure 2.15.

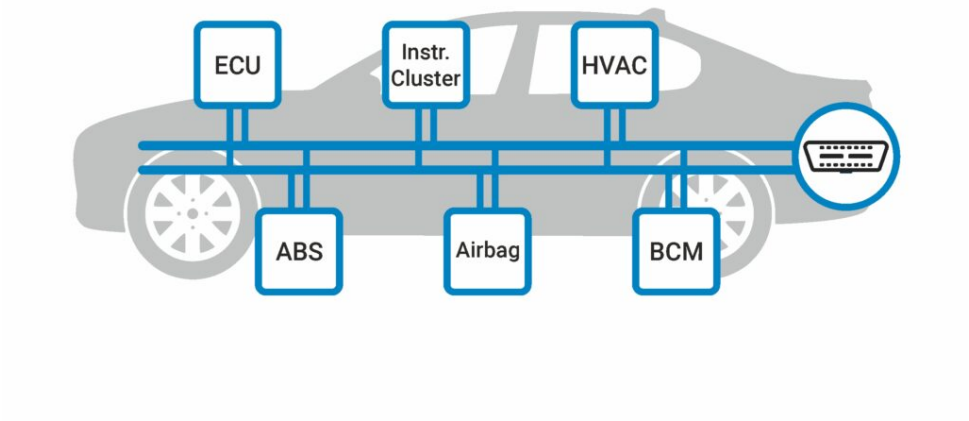


Figure 2.15: Simplified CAN configuration in a modern vehicle [21]

With as many as 70 electronic control units for numerous devices, forming independent subsystems, most of the time communication between them is essential. The biggest processor usually is the engine control unit, while others, used for autonomy driving, airbags, anti-lock braking, cruise control, transmission etc, are also present. A subsystem may require information from a sensor typically used by another system, or take over a control unit, and thus, the CAN standard can fill this need. The interconnection between all subsystems of a vehicle contributes to safety and economy, as well as the advantage of upgrading/updating features using software alone.

CAN is a serial bus standard in which multiple master nodes can be present. These nodes are usually electronic control units, requiring at least two for a CAN network

to operate. A node type may vary from a simple digital logic chip up to a powerful embedded computer and can also provide a gateway allowing a computer to communicate with the CAN network through a USB port. The bus consists of a two-wire (CAN high and CAN low) twisted pair with characteristic impedance of 120 Ohms. When the bus is in idle mode, both lines convey 2.5 Volts. During data transmission, CAN high line rises to 3.75V and CAN low line dips to 1.25V to create a 2.5V voltage difference between the two lines. Communication, with a maximum rate of 1 Mbit/s, is established by measuring voltage differential, so electrical fields and noise do not affect CAN bus' reliability and performance.

2.2.2 LIN

Local Interconnect Network (LIN) [22] is an automotive specific serial network protocol used in communication between various components. This protocol supports data transmission of 19.2 Kbit/s, a maximum bus length of 40 meters and operates using a single wire technology. As new technologies were equipped in vehicles, the cost to implement CAN bus on every component was too high. Thus, automakers needed a cheap alternative and founded LIN Consortium in the late 1990s, with the first LIN version released in mid-2002.

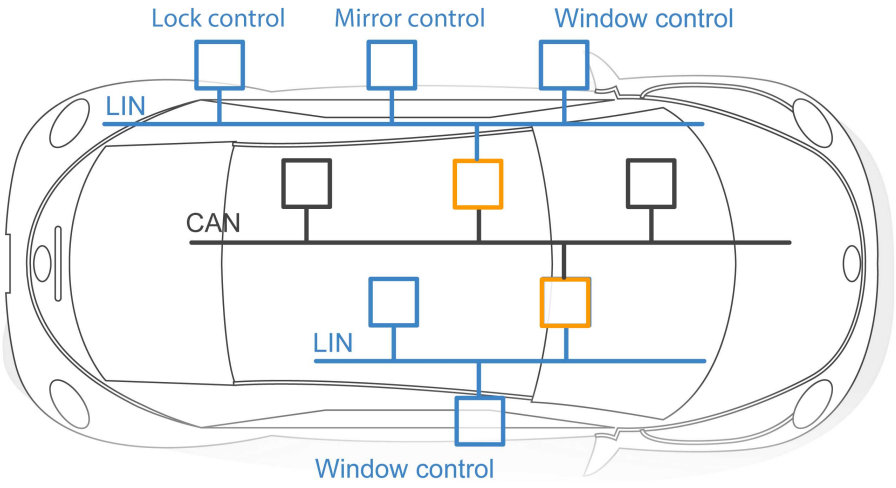


Figure 2.16: Simplified LIN configuration for component control in car door [23]

LIN is a network consisting of 16 nodes one of which acts as the master node and typically up to 15 slave nodes. Only the master can initiate messages, hence a collision detection system is not required. It is adopted by combining multiple LIN-based sensors and components to create small subsystems, that ultimately communicate with other subsystems using another network, such as CAN (Figure 2.16).

It is widely used in applications where low cost is essential and bandwidth is not important, such as climate or illumination control, door, seat and roof component control and much more. It reduces the number of required harnesses throughout the vehicle and offers reliable, easy to use and most importantly cheap alternative to other communication busses.

2.2.3 FlexRay

FlexRay is another network communications protocol for automotive applications, developed by the FlexRay Consortium. It is a communications bus designed to deliver top performance with data rates of up to 10 Mbit/s in a rugged environment, while it can continue to operate properly in the presence of a fault in one or more of its components. FlexRay supports single and dual channel configurations consisting of one or two unshielded twisted pair cables respectively to connect nodes with each other. It is used to reliably connect multiple electronic control units (ECU) in a vehicle, mainly in highly demanding applications, such as anti-lock braking, electronic power steering, advanced driver assistance systems and more. In Figure 2.17, we can see a car configuration example utilizing CAN, LIN and FlexRay communication protocols.

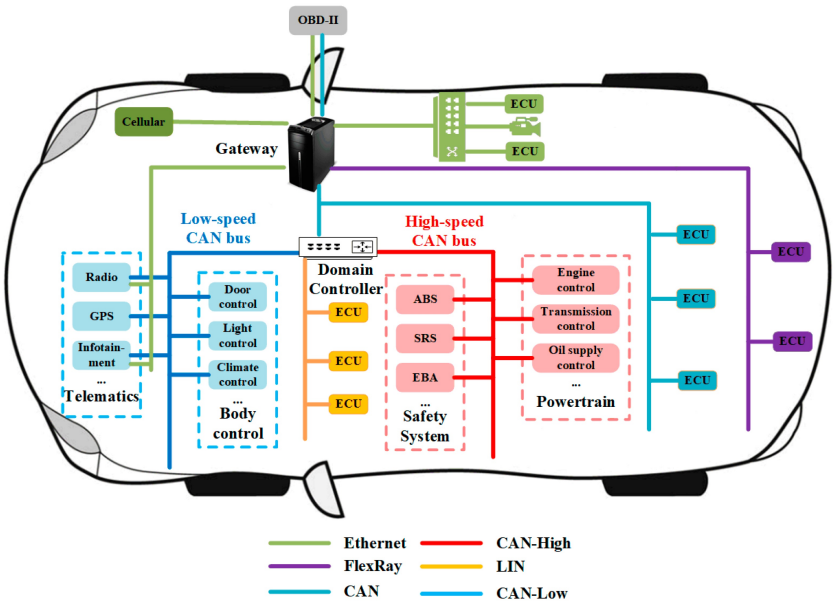


Figure 2.17: Car configuration utilizing CAN, LIN and FlexRay [24]

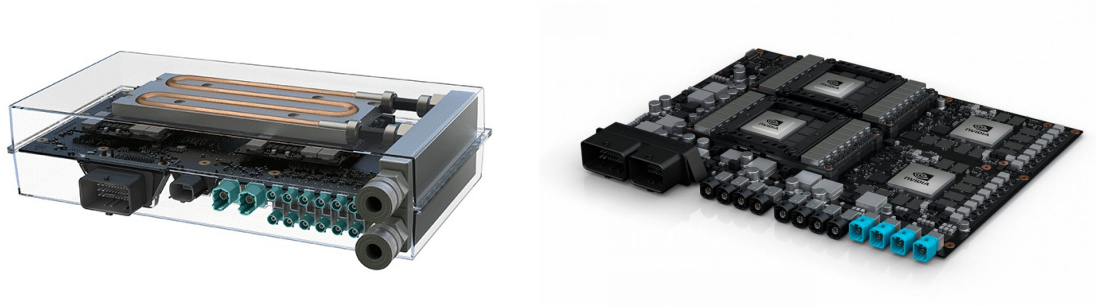
2.3 Nvidia Drive Platform

The DRIVE family of Nvidia products [25] is an open autonomous vehicle development platform available to software developers. It includes various DRIVE software and libraries that allow developers to join their innovations with a DRIVE system. Aimed at providing the means to power deep learning and artificial intelligence applications, some of its use cases include active safety, automated driving and parking, AI cockpit capabilities that can enhance autonomy characteristics of an AV to at least Level 2, all the way to Level 5. Drive platforms are designed to provide high processing power to utilize all possible sensors a vehicle could use. They offer network connectivity features and can acquire and process data from GPS, Radar, LiDAR, Cameras, ultrasound sensors. Additionally, with powerful dedicated GPUs, many autonomy features can be developed and efficiently executed. Finally, they provide means to connect a variety of sensors, controllers or displays together through communication protocols such as CAN, LIN and FlexRay busses.

2.3.1 Hardware for Self-Driving Cars

Nvidia's first autonomous systems, Drive CX and Drive PX, were announced at CES 2015. Drive CX is powered by a single Tegra X1 System on a Chip (SoC), while Drive PX is equipped with two chips in order to increase performance. Tegra X1 is a chip developed by Nvidia that features four 64-bit ARM cores, a Maxwell based 256 core GPU, various video encoding/decoding capabilities and a maximum power consumption of 15 Watts. They were Nvidia's first platforms destined for semi-autonomous vehicles.

A year later, a new version of the platform, the Drive PX2 was announced at CES 2016. It came in two possible configurations, the Drive PX2 AutoCruise and Drive PX2 AutoChauffeur, with one or two Tegra X2 SoCs respectively and discrete Graphics Processing Units (GPU) based on the Pascal micro-architecture. Since its release, all Tesla Motors vehicles manufactured from October 2016 were equipped with a Drive PX2 AutoCruise [26] system with a modified chip, with liquid cooling capabilities. The platform's processing power was used in Tesla's autopilot and self-driving functionalities.



(a) Tesla Modified Drive PX2 [26] (b) Nvidia Drive PX Pegasus

Figure 2.18: Drive PX2 and AGX Platforms

Xavier AI Car Supercomputer was the new Drive system powered by Volta micro-architecture that was announced at CES 2017 and was re-branded as Drive PX Xavier at CES 2018. The performance boost compared to its predecessor, the Drive PX2 AutoChauffeur, was approximately 50% greater, with a theoretical limit of 30 INT8 TOPS and a power consumption of only 30 Watts. Nvidia's Drive PX Pegasus system (Figure 2.18b), was announced at GTC Europe in October 2017, and was developed to power fully-autonomous robo-taxis. Based on two Xavier CPU/GPU devices and two Turing generation GPUs able to deliver over 320 trillion operations per second, with ten times higher computing capabilities than the Drive PX2, it would be able to supervise and operate Level 5 autonomous vehicles.

The DRIVE Orin SoC, a result of four years of R&D investment, is Nvidia's next generation chip for automotive applications announced in late 2019. Due to current demand for lower-end systems targeted at level 2+ vehicles Nvidia recognized that fully autonomous level 5 robo-taxis are years away from achieving significant volume. This will enable scalable deployment of software for self-driving vehicles as well as the advantage of smooth transition to newest platforms of the DRIVE family. The Orin family of SOCs production will start from late-2022 or early 2023 and will be utilizing the new Ampere GPU micro-architecture.

2.3.2 Software for Self-Driving Cars

In order to efficiently harvest a powerful hardware's true potential, an equally powerful and sophisticated Operating System is needed. As a consequence, the need for a reliable in-vehicle Operating System, that provide developers with the necessary tools, support for real-time data processing and multiple sensor data fusion is raised. Nvidia offers a complete software suite with all the building blocks required for autonomous application development, assisting developers to efficiently design and deploy perception, planning, mapping algorithms. These tools are explained in the following paragraphs and are illustrated in Figure 2.19.

DRIVE OS Nvidia DRIVE OS [27] is an operating system designed for Nvidia DRIVE systems with support for GPU accelerated computing. It includes NvMedia API for hardware accelerated sensor data processing, CUDA and cuDNN libraries for efficient parallel computing implementations, Nvidia TensorRT for real-time deep learning applications with a hardware-dependant optimizer for low latency inference, graphics APIs (OpenGL, OpenGL ES, EGL), and many other developer tools and modules to access hardware engines. This operating system was specifically developed for creating and deploying Autonomous Vehicle applications for DRIVE-based systems. With features like secure boot, firewall and over-the-air (OTA) updates, it becomes a secure execution environment ideal for safety-critical applications. Its Software Development Kit (SDK) equips developers with all necessary libraries, software and tools to design, build, debug and ultimately deploy parallel computing and deep learning applications for self-driving, autonomous vehicles.

DriveWorks Nvidia's DriveWorks [28] is the Software Development Kit acting as the groundwork for autonomous vehicle software development providing all processing modules, frameworks or tools required. Its Sensor Abstraction Layer (SAL) aids the introduction of various sensor configurations, including radars, LiDARs,

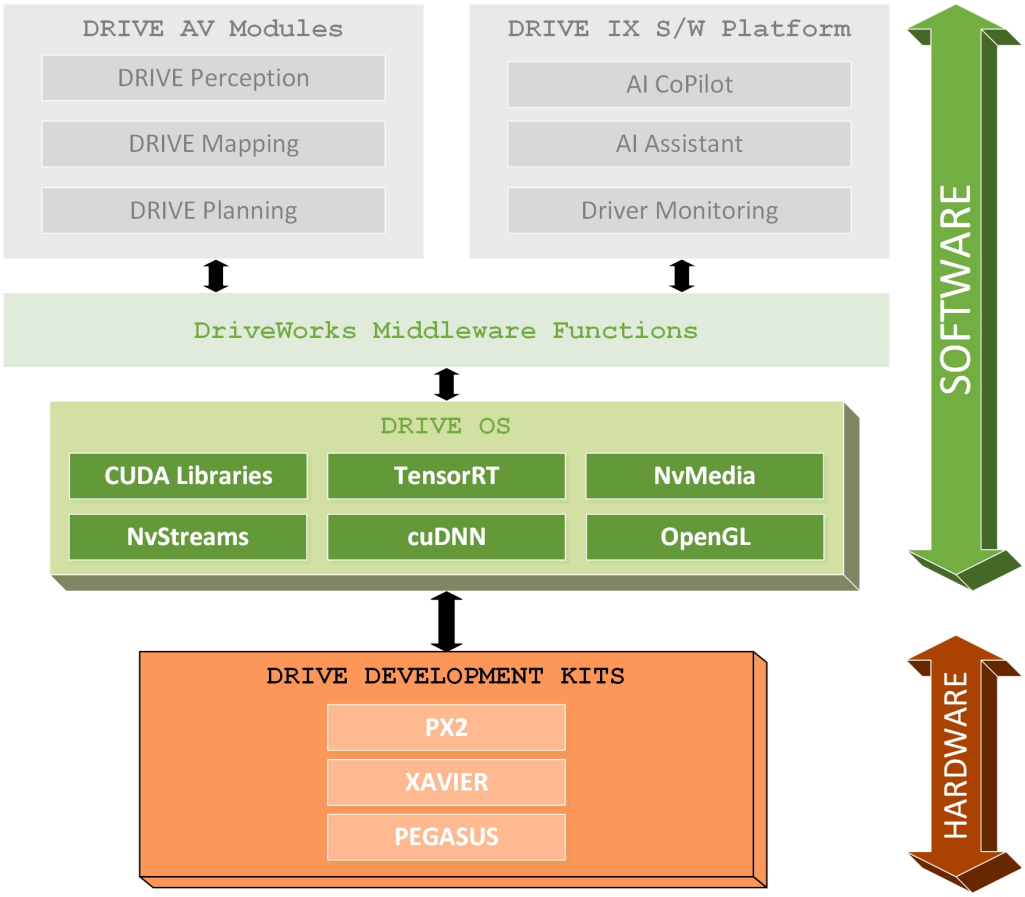


Figure 2.19: Nvidia DRIVE Software Stack
Source: Nvidia DRIVE OS Documentation [27]

cameras, GPS and IMU devices, and the tools for data processing acceleration. Developers can integrate automotive sensors withing their software, accelerate data processing or neural network inference for AV perception, efficiently track and predict a vehicle position and orientation.

DRIVE AV DRIVE AV [29] is a software stack that offers perception, mapping and planing capabilities, as well as diverse DNNs trained on real world data. The DRIVE Perception layer can detect, track and estimate distances from objects, using data gathered from various sensors. DRIVE Mapping localizes the vehicle to a continuously updated high-precision map. DRIVE Planning is responsible of controlling the vehicle's motion and planning its path and behavior through the environment.

DRIVE IX DRIVE IX [30] is an open software platform that delivers interior sensing and driver monitoring for innovative cockpit solutions with advanced artificial intelligence features. The AI CoPilot module uses driver-facing camera, infrared LEDs and deep learning algorithms to ensure the driver is alert at all times or even take action, if the driver is distracted or drowsy.

With advanced face tracking technology, monitors and analyzes blink frequency to estimate fatigue or drowsiness levels. The Visualization module receives data from

multiple sensors through DRIVE AV and provides real-time information on the instrument cluster for the driver and represents what the sensors or driver monitoring systems are viewing using visual information. This representation is achieved through Confidence View, a virtualized view for the instrument cluster. Finally, a powerful interface for human-machine interaction is the AI Assistant that utilizes voice recognition through natural language understanding models, face identification and emotions detection.

Chapter 3

Related Work

3.0.1 Nvidia Drive Hyperion

One of the Autonomous Vehicle platform currently leveraging Nvidia Drive technology is the Drive Hyperion. Consisting of a completely optimized and tuned sensor suite, along with high performance computed platform powered by AGX Orin, AGX Pegasus and Hyperion 8.1 Developer kits, fused together into a single Drive Orin system-on-a-chip (SoC), it functions as the reference architecture aiming to aid the development of Level 2+ and Level 3 autonomous vehicle systems.

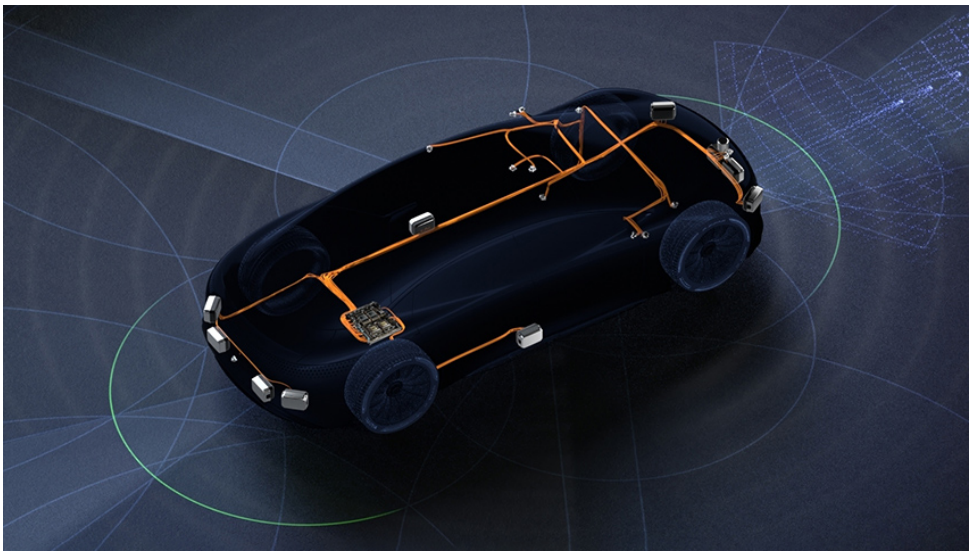


Figure 3.1: Nvidia DRIVE Hyperion [31]

The Orin SoC is the main processor paired with an Ampere architecture integrated GPU, capable of delivering up to 254 TOPS of processing power, also combined with other high performance components, such as Programmable Vision Accelerators (PVA), Image Signal Processor (ISP), video encoder and decoder. The DRIVE Hyperion 8.1 Development Kit solution includes three AGX Orin Kits with interfaces to connect multiple sensors, such as cameras, LiDAR and Radar modules. It is also paired with two DRIVE Recorder modules and two Smart Network Attached Storage (NAS). The recorder modules offer very high speed connectivity with multiple

interfaces, such as Mellanox ConnectX6-DX adapters, 100 Gbit Ethernet networking and vehicle IO harness for CAN, FlexRay, GPIO protocols. Additionally, up to 4 M.2 NVMe SSDs can be connected to each NAS module, offering 2GB/s lossless data recording.

Nvidia's Level 2+/Level 3 sensor set include 12 exterior cameras, six short-range and three long-range radars, one LiDAR sensor and 12 ultrasonic modules. For vehicle dymanics and location detection, two IMUs and one GPS module, as well as three interior cameras, are included for driver and passenger monitoring systems. Finally, Nvidia DRIVE Sim is an end-to-end simulation platform that offers accurate ray tracing virtual sensor and vehicle simulation, a tool designed to test and profile autonomous systems using synthetic data before real world deployment.

3.0.2 Yamaha Motobot

Yamaha's completely autonomous racing motorcycle project is the Motobot. It is a humanoid robot, that from a distance is indistinguishable from a human rider, designed to ride a vehicle unmodified for autonomous use around a racetrack. All processing systems and sensors are integrated inside the robot's body with simple connections to the motorcycle in order to acquire data about vehicle's speed, engine RPM, attitude etc. The Motobot is placed on the Yamaha YZF-R1M super-bike capable of more than 200 km/h of top speed. This super-bike is usually seen in motorcycle racing, as well as the top choice for bike enthusiasts. To perfect the bike's design and ensure the rider's safety, motorcycles need to be thoroughly tested, but during extreme conditions, using a human driver can be dangerous and risky. This autonomous system was initially developed as a smart 'crash test dummy' in order for Yamaha to test its high-performance super-bikes.

Figure 3.2: Yamaha Motobot YZF-R1M [32]

Motobot receives data from the motorcycle and controls six actuators operating steering throttle, front and rear brake, clutch and gearshift pedal, and fully operates the vehicle autonomously. It uses GPS and IMU sensors in order to navigate, but does not require Cameras, LiDARs or Radars, since it is designed to run on an empty racetrack and not for public roads with other vehicles and traffic. The first challenge their engineers encountered was the balance controller and to teach the Motobot how to balance the motorcycle at lean angles of more than 50 degrees at various speeds reaching 200 km/h. It should be capable of rapidly and reliably make precise

bank angle changes and identify the motorcycle limits without crashing. Another challenge was the process of analyzing vehicle and sensor data, and using a path-following algorithm to take advantage of the circuit's characteristics, including high speed-straights, a variety of turn types, and when to accelerate or decelerate.

The Motobot was put to the test against Valentino Rossi, a former professional motorcycle road racer and nine-time Grand Prix motorcycle racing World Champion. Rossi managed to lap 32 seconds faster than his autonomous rival in a 3.2 km circuit with a 1:25.740 lap, compared to a 1:57.501 lap of the Motobot. Although the difference might seem colossal, a 1:57 lap is considered an average trackday pace, with the exceptional 1:25 achieved by one of the most successful racers that only a handful of people are able to compete with. With future time and money investing, Yamaha could ultimately create a system that could post decent lap times and compete with a few of the best drivers.

3.0.3 BMW R 1200 GS

At CES 2019, BMW demonstrated its self-driving motorcycle, able to balance itself, start or stop to a standstill and turning without requiring a driver. It was developed under a secret, five year project carried out by BMW Motorrad. Considering the evolution of autonomous cars, and internet of things (IoT), they are aiming to involve motorcycles in the conversation of the Autonomous Future. This motorcycle is a stock-looking R 1200 GS model, with three hard pack cases equipped on the back of the bike, that contain all the necessary electronics for autonomous applications and radio communications. A rear-mounted antenna receives commands from a human operated controller about when to start or stop, and instructions about wide or tight turns. Its system, translates these commands and using its road-reading and path finding capabilities, actively controls the throttle, brake and steering inputs, just like a human driver would. The motorcycle, actively balances itself while moving using steering, but does not use a gyroscopic system to keep the bike upright while stationary.

Figure 3.3: BMW R1200GS self-driving motorcycle

At the moment, BMW does not have a consumer plan for this project, as it is not designed to replace the classic motorcycle's in our roads. The current focus is

designing a platform for testing and evaluating new innovations. Technology that can help accelerate the training of new drivers on how to approach various types of corners, or support riders that do not pay attention to their surroundings.

3.0.4 Waymo Urban Driver

Waymo Urban Driver is am autonomous vehicle solution available in Phoenix, Arizona and now expanding to San Francisco. Numerous of these driverless vehicles can be found around the city twenty-four hours a day, seven days a week, available to anyone that can simply book a ride using an app on their phone. It is acknowledged as the world's most experienced driver by Alphabet AI, with over 32 billion kilometers of distance traveled and a mission of making driverless and autonomous traveling saver and more reliable in the future. The fleet currently consists of thousands of these vehicles equipped with multiple LiDAR and camera sensors and Radars in order to sense a 360 degree view around the vehicle in any weather condition, day or night.

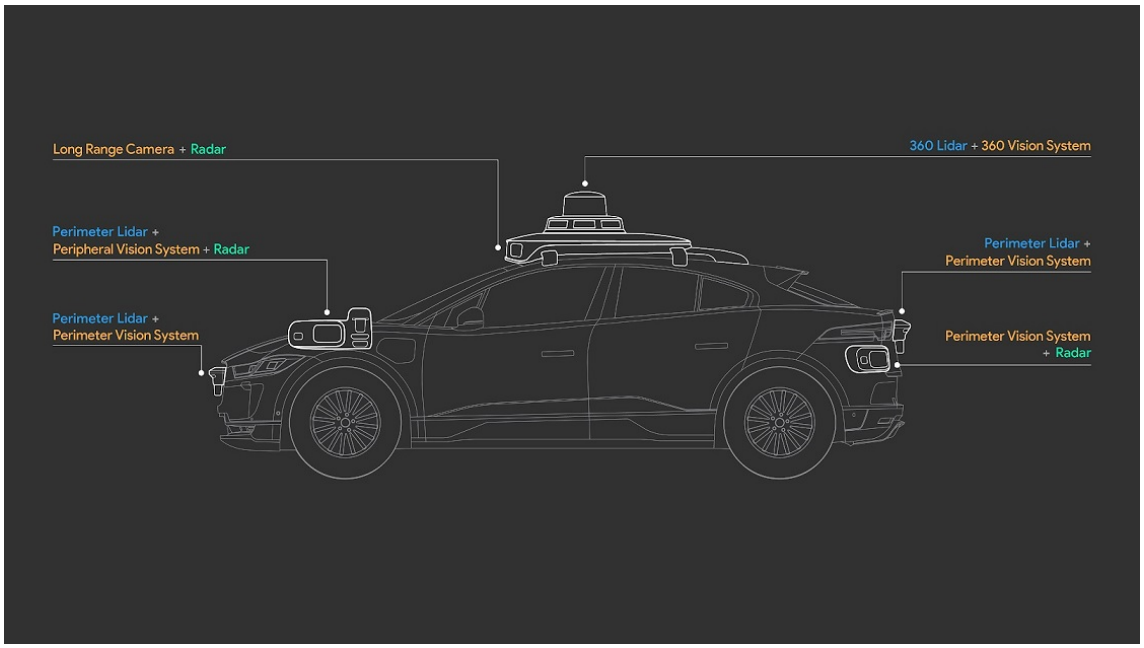


Figure 3.4: Waymo autonomous Vehicle sensor configuration [33]

After years of research on AI and machine learning, Waymo created a strong training and evaluation infrastructure aimed towards their ML-basted driving system. Every aspect of the software, including perception, semantic understanding, path planning and behavior prediction both adapt to the city's characteristics and complexities while driving, while also contributing to the research community with the Waymo Open Dataset [34]. The aim is to develop fully-autonomous vehicles that do not rely on human interaction, with the engineering team focusing on the hardest parts of this challenge. Meaning, that in order to achieve true autonomy, the system must be able to handle extremely rare events, while putting the safety of its passengers, as well its surroundings, at the highest priority.

Chapter 4

Platform Description

4.1 Electric Motorcycle

The proposed system is intended for a prototype three-wheeled electric motorcycle designed and under development at Technical University of Crete. It features a lightweight aluminum chassis, which offers high strength to weight ratio, while still being reasonably affordable and corrosion resistant. The total weight without a driver is 240 kilograms and its dimensions measure 2.1 × 0.8 × 1.2m (LxWxH). The motorcycle is powered by a 100 KW (134.1 HP) brushless electric motor with a maximum torque of 230 Nm and 5200 maximum RPM.

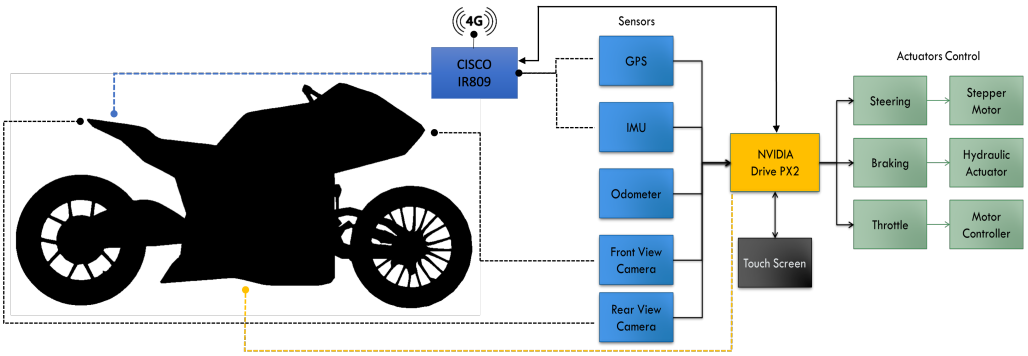


Figure 4.1: Bike concept

The proposed final version of this autonomous motorcycle features actuators for throttle, break and steering control handled by the Drive PX2 system for advanced driver assistance technologies. One forward facing and one rearward facing camera, as well as Odometry, IMU, GPS and Radar sensors. Additionally, all the necessary information will be displayed through the touch screen display to the user, and data can be transmitted to the web through a CISCO 4G modem.

4.2 Platform and Sensors

4.2.1 Nvidia Drive PX2

The Nvidia Drive PX2 platform has two variants based on one or two Tegra X2 SoCs, where each SoC contains 2 Denver cores, 4 ARM A57 cores and a dedicated GPU from the Pascal generation. The configuration we have available is the AutoChauffeur, that features two Tegra X2 SoCs and two dedicated Pascal GPUs. It supports up to 12 cameras, plus connectivity for perception modules such as LiDAR, Radar or Ultrasonic sensors. Its general purpose IO includes separate Ethernet, USB and HDMI output ports for each Tegra sub-system, connectivity that can be very usefull in our proposal. Finally, designed for automotive use, interfaces such as CAN, LIN, FlexRay are available through the vehicle harness connector and can be used to wire sensors all around the motorcycle.

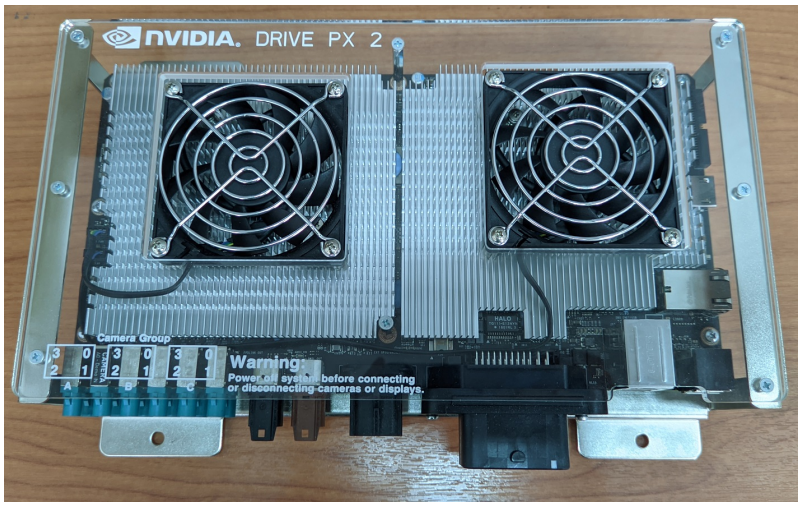


Figure 4.2: Drive PX2 Platform - top view

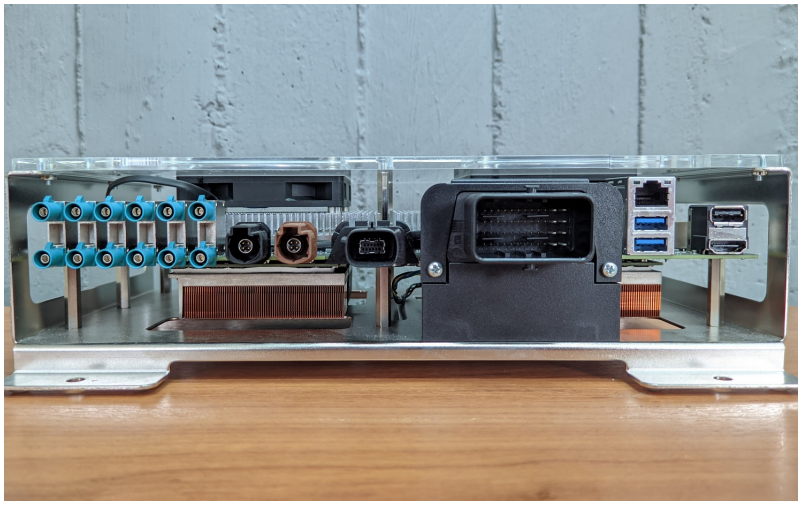


Figure 4.3: Drive PX2 Platform - side view

Nvidia's mechanical installation guide for the Drive PX2 specifies all the dimensions required to safely mount the device to a vehicle as well as the pinout of every connector and harness. This way, we can develop our custom harness and design

how the system will be connected to every part of the motorcycle. The device requires 12 Volts to operate, with a max power consumption of 300 Watts. As the included power supply cannot be used on a vehicle, it is necessary to implement an adapter that allows us to power the device from the vehicle's battery pack.

4.2.2 SEKONIX SF3324-100 Cameras

This camera model is a GMSL standard (Gigabit Multimedia Serial Link), meaning that the video signal is digitally transmitted without compression via a coaxial cable. Thanks to this higher resolution, the image is sharper than with an analog system. The cameras we have available are SEKONIX's SF3324-100 that feature a 2 Megapixel Red Clear Clear Blue (RCCB) sensor. It consists of a 14 bit parallel pixel interface with a 27MHz input clock and effective pixel resolution of 1928×1208 and a framerate of 30 FPS. This sensor needs a color balance and color correction matrix in order to compensate for the lack of green filtered pixels and obtain an ordinary RGB image. RCCB sensors are usually designed for automotive use and offer better signal to noise ratios (SNR) and high contrast on both daytime and nighttime.

Figure 4.4: SEKONIX SF3324-100 GMSL Camera

The SF3324-100 model is equipped with the NA1262 lens that meets automotive qualification and provide 120 degree horizontal and 75 vertical viewing angles. With a very compact size of $26 \times 26 \times 30.7$ mm (including the lens) it can be easily mounted in tight spaces. Additionally, it provides protection against ingress of dust and high temperature as well as high pressure water with an IP rating of IP69K. All the products of this family are designed and natively supported by Nvidia DRIVE systems.

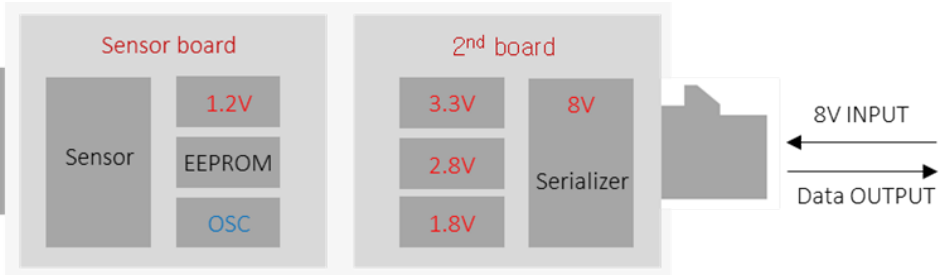


Figure 4.5: SEKONIX SF3324-100 block diagram [35]

4.2.3 Grayhill 3D70 Display

The electric motorcycle's will be equipped with the Grayhill series 3D70 display [36] to be used as its main dashboard. Below, the hardware architecture including its available features and ports will be described in detail, as well as the development kit and the way it will be deployed on the motorcycle. The display will provide the rider all necessary information about the motorcycle's state, include a front and rear camera feed, and basically serve as an advanced dashboard.

(a) 3D70 Display Front Side (b) 3D70 Display Back Side

Figure 4.6: Grayhill 3D70 Display

This 7.0-inch backlit WVGA LCD with a resolution of 800 × 480, 16 bit color and a peak brightness of 1000 nits provides excellent readability during daylight. Additionally, the LED backlighting can be software controlled and adjusted during nighttime. It features a responsive projected capacitive (PCAP) touchscreen that can recognize both bare and gloved fingers even if the screen display is wet. A cover glass is bonded to the LCD display offering scratch resistance and anti-glare properties, making it ideal for extreme environments. Provides easy GUI Design and application creation and integration with VUI Builder, Qt and more. The display module comes equipped with an ARM based embedded computer running Linux operating system that can monitor and display many events simultaneously. The system's boot up time is three seconds and can be set up to automatically load the intended program requiring no user input. It can function in both 12 and 24 Volts making it compatible with every vehicle type. A numerous options of IO are present, such as CAN-bus ports, camera input ports, digital and analog input, digital output, USB, Ethernet and audio ports. This Grayhill display series is intended for many agriculture and construction vehicle applications, such as virtual gauges, diagnostics, fault indicators and many more, since it is fully programmable.

Development Kit

Grayhill offers the 3D70Dev-100 Kit which apart from the screen, includes a development stand with a breakout board, two 18-pin cables to connect the display to the board, USB to Can adapter, 12V power supply and more. This kit enables us to experiment with the display and the features it provide, as well as design, test and optimise a functional Graphical User Interface for our end product. The breakout board design is illustrated in Figure 4.7, with additional information about the display and its development kit available in the quick-start guide [37] by Gray-Hill.

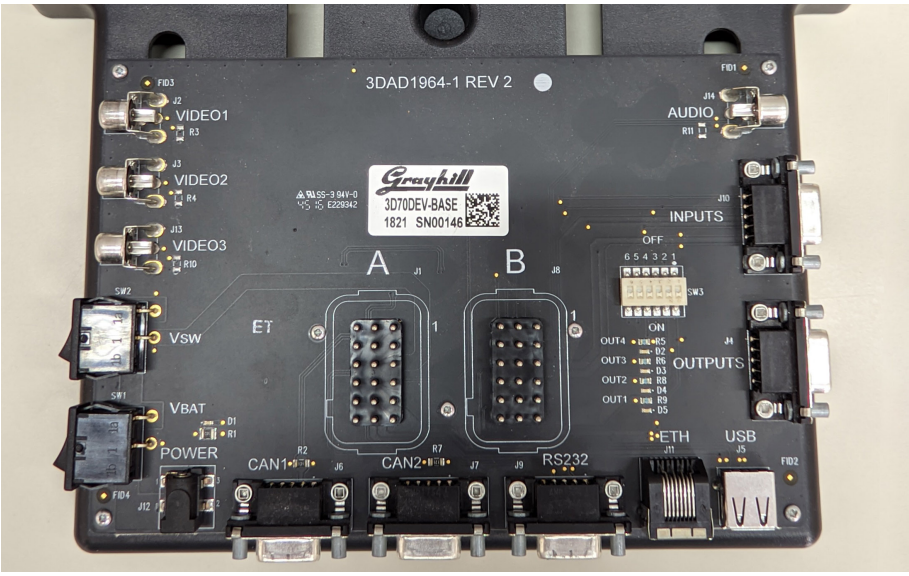


Figure 4.7: Grayhill 3D70 Development breakout board

This breakout board is designed with the necessary connections to make all ports available to the developer. In the middle of the board, we can see two 18-pin harness connectors that link to the display. On its sides, power, video, CAN, GPIO connectors are available, in addition to various switches and LED indicators used in some conditions. Although connection between the display and breakout board is achieved through two 18-pin cables, the end user is free to design an application specific harness for their needs using Grayhill's wiring diagram included in the quick start guide. This board will act as the foundation for software development and will aid in the testing of the various inputs this display supports.

Chapter 5

Offline Programming

This work focuses on directions and tactics commercial vehicles follow, in order to design and implement an autonomous-ready system for our electric motorcycle. The recognition of traffic signs using a camera in order to assist the driver, as well as the ability to safely follow a vehicle through urban areas with a fixed distance are two of the goals of this project. Our platform of choice is the Drive PX2 system along with two high-resolution cameras. Thus, our solutions are based on data input from camera sensors and data processing on an ARM based portable system.

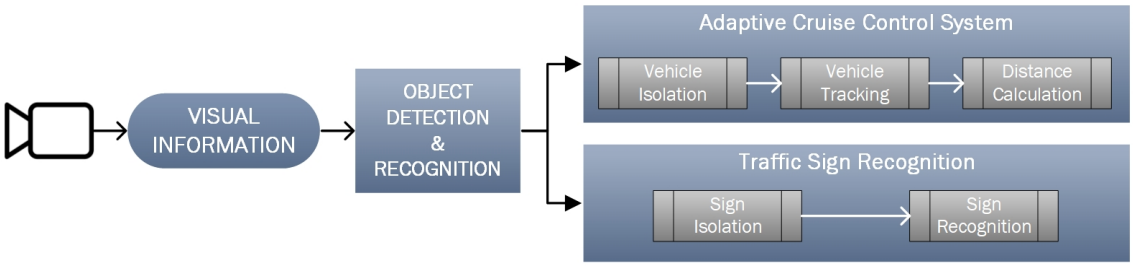


Figure 5.1: Flowchart of the proposed network

The aim of this section is to design a network that can offer autonomous capabilities to our system using real-time visual information gathered by the vehicle cameras. Figure 5.1, analyzes the data workflow of our proposed network. The object detection and recognition procedure is explained in Section 5.1, that uses the visual information from the camera sensor to produce a list of detected objects of various types. Following that, in Section 5.3, we demonstrate the Adaptive Cruise Control system that isolates the desired vehicle we wish to track, and constantly calculates its distance based on its bounding box. The data and calculations yielded from the Adaptive Cruise Control system will be later used as input to the control unit that will handle the motorcycle. Section 5.2 explains the way we take advantage of the detected traffic sign during the previous step, in order to analyze its type and figure out its meaning, in order to provide the driver additional visual information through the display. Finally, in Section 5.4, we use the CARLA simulator in order to test and evaluate our ACC system on real world scenarios.

5.1 Object Detection and Recognition

For the development of an object detection system we started by adopting a Haar Cascade Classifier with an existing model, pre-trained on car detection (as suggested in [38]) using only the OpenCV libraries. The working principle of a Cascade Classifier Pipeline is illustrated in Figure 5.2; it uses edge or line detection algorithms across the frame in order to detect the desired object (face detection in Figure 5.2). The Frame images are divided to sub-windows of 24×24 resolution that get fed into multiple cascade stages. The first few stages are trained to reject negative sub-windows, and keep most candidates with face-like content. A stage is only activated for a sub-window, only if the previous stages yields a positive result. The total number of stages can vary from as low as 7 to more than 20. A sub-window that passes all the stages of the cascade classifier, is labeled as a face detection. Haar Cascade Classifiers are commonly used in object detection [39] [40], with high speed being the result of their simplicity. The primary advantage of Haar cascades is that they can be faster than many modern algorithms. Our classifier analyzes the camera footage frame by frame and creates an array with the pixel coordinates of all the detected vehicles and later draws rectangles around them as a visual aid for the user.

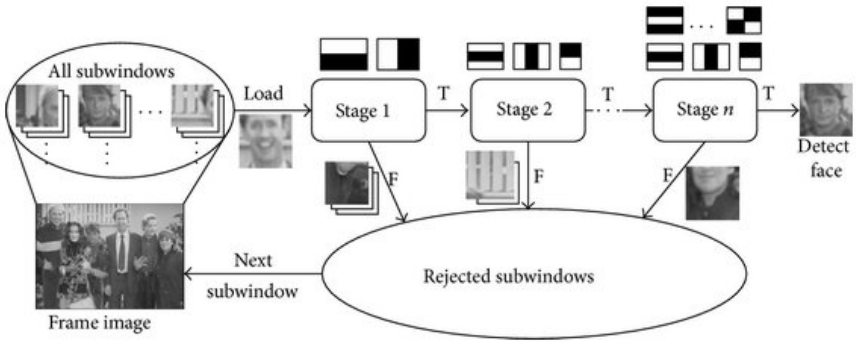


Figure 5.2: Cascade Classifier Pipeline [41]

This method of vehicle detection is very fast and does not require GPU acceleration. Such low-processing power requirements make it ideal for low-end or mobile systems, surveillance or traffic cameras, its main disadvantage being their tendency towards false-positive detections. Thus, the overall accuracy and consistency is reduced, raising a significant problem. For the ACC to work effectively in our use case, we must always be aware of where the vehicle being followed is located. This classifier is highly sensitive to lighting and vehicle angle, and as a result it often lost track of the vehicle we were trying to follow. Subsequently, the most significant factor of the ML model we need to adopt for image clustering, object detection and classification is its accuracy and not its speed. The training of a new ML model can be completed on the premise that a large enough dataset suitable for our use case can be found. Unfortunately, this process, especially on the automotive industry can cost a few days or even weeks, in order to deliver a model with high accuracy. Having said that, TensorFlow's object detection API is an open source framework that aids the construction, training and deployment of object detection models.

TensorFlow Model Garden [42] provides a wide selection of models pre-trained on

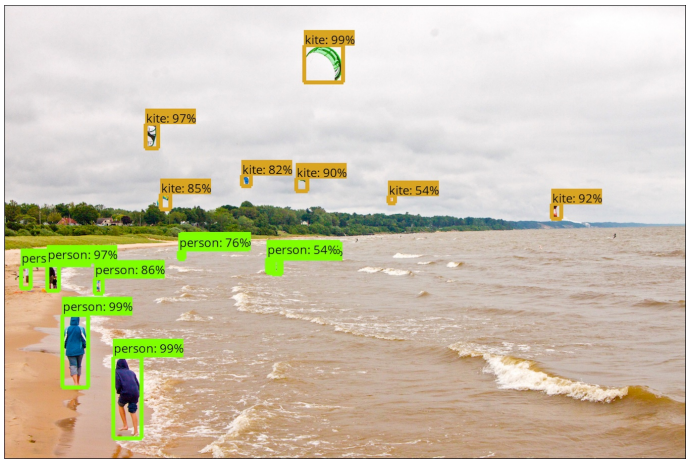


Figure 5.3: Tensorflow Model Garden Example [42]

the COCO dataset, Kitti dataset, and more. There are multiple models to chose from, depending on the use case. In our case, we need a model that offers high accuracy without taking a big hit on speed. The average frame processing time of the available models varies from 26ms to 2 seconds with a mean Average Precision (mAP) metric in the range (16, 43) ms. Our model of choice is the Region Proposal Network "Faster R-CNN resnet50 coco"[43] trained on Microsoft's Comon Objects in Context[44] (COCO) dataset, a large-scale object detection, segmentation, key-point detection, and captioning dataset. This deep learning based object detection system with a decent performance of 5-17 frames per second (average frame time of 89ms), provides high quality region proposal (mAP metric of 30) and enhances the overall object detection accuracy. During testing with the recorded video, we noticed substantial improvement, and a consistent detection of pedestrians, traffic signs, and multiple vehicles throughout a crowded area. It is important to note that, as this model is trained to classify a plethora of objects, we only chose to make use only the ones that are of interest to our use case. Additionally, we can use this model in order to detect traffic signs and their location on the frame, but not interpet their meaning. This procedure will be explained in Section 5.2.

5.2 Traffic Sign Recognition

A practical feature for every autonomous vehicle is their ability to recognise traffic signs, interpret their meaning and present them to the driver if necessary. A typical traffic sign recognition system [45] [46] reprocesses an image frame in order to extract regions of interest. Then, a feature descriptor algorithm, such as the histogram of oriented gradients (HOG), is performed, a variety of machine learning algorithms, such as SVM, KNN or Random Forest [47], followed by a context-aware filter in order to localise and recognize the traffic sign meaning. This approach includes a primitive model training based on [48], using TensorFlow and Keras libraries.

Advanced traffic sign recognition models [49], perform both object recognition, in order to localize traffic signs in camera frames, and analysis, to detect the sign's meaning. In our proposed network, the detection process is undertaken by the previous stage discussed in Section 5.1. Thus, we implemented a simplified yet functional sequential Convolutional Neural Network model, whose input is not the

entire camera frame. Instead, we use the bounding box for every detected traffic sign in the previous stage, to crop the part of the frame and use it as the input. It consists of the convolution layer as its major building block that performs convolution operation on the input image down-sampled to a 32×32 resolution with a bi-linear interpolation method, as illustrated in Figure 5.4. The resolution of the bounding box of the detected traffic sign can vary, depending on distance and angle from our camera sensor.

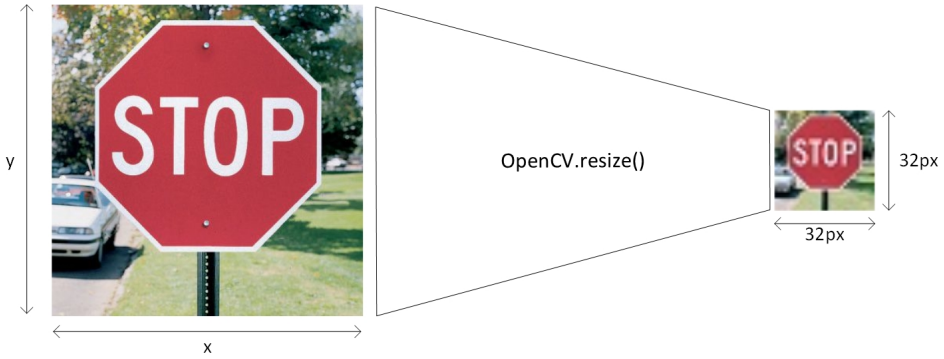


Figure 5.4: Convolutional Neural Network input resizing

This is succeeded by a Pooling Layer responsible for reducing dimensionality of the image, but preserving important features. We can chose Max Pooling or Average Pooling depending on our use case. For this instance, the selected layer was Max Pooling, that selects the maximum value from a feature map in contrast to Average Pooling that calculates the average for each patch of the feature map. The layer that helps the decision process about the information that needs to be processed further and those who do not is the Activation Function. Essentially, it defines how the weighted sum of the input is transformed into an output and introduces non linear properties to the network. Different types of activation functions include Sigmoid, Tan H, ReLU, Binary Step function and more. The most popular, and the one we adopted, is the ReLU function with a range of $[0,+\infty]$. Finally, in the output layer, the softmax function is used as the activation function that predicts a multinomial probability distribution and essentially classifies the input image over 43 different classes. In Figure 5.5, the overall stucture of the sequential neural network is presented. In Figure 5.6, a sample of the data used in the training process is illustrated which was was split for train and test with a 80% train and 20% data ratio.

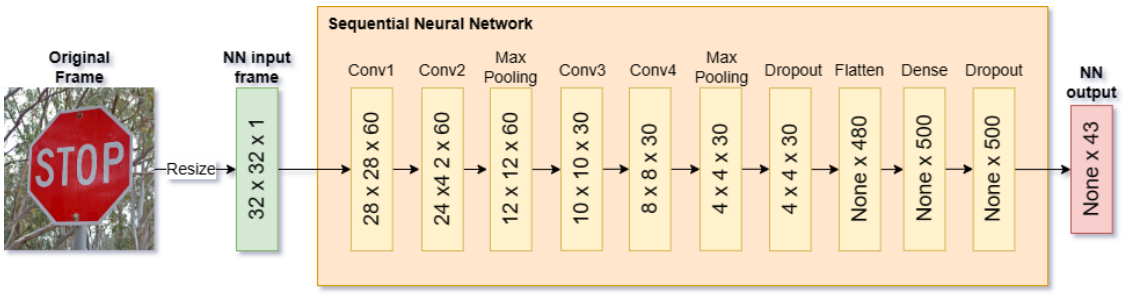
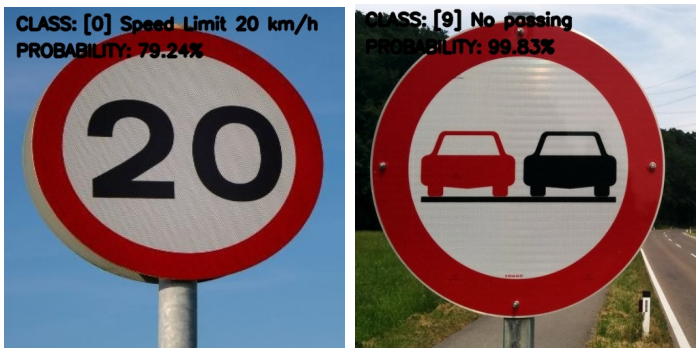


Figure 5.5: Neural Network Structure

Figure 5.6: Convolutional Neural Network training data sample

For testing purposes, in order to evaluate the model's training result, we gathered multiple traffic signs photos using Google Maps and Google Street View of various angles and resolutions, with environment typically present behind traffic signs, and introduced them as input to our model. For outlier rejection, a confidence threshold of 75% was set, and the recognition results are depicted in Figure 5.7.



(a) 20km/h Limit (b) No Overtaking Sign

(c) Stop Sign (d) Work Ahead Sign

Figure 5.7: Traffic Sign Recognition Test Results

This model was developed using Python 3.9 along with Machine Learning libraries, such as TensorFlow, Keras, OpenCV, and although they are supported in the latest releases of the DRIVE platform, Drive PX2 has no official support for python program execution. In order to deploy our project to this platform, our saved model, which is stored in Keras format, must be converted and serialized into a common UFF MetaGraph format using Nvidia's official converter. This process also optimises

the model for Nvidia's TensorRT engine for higher efficiency and speed and can be executed on the PX2 platform using the included C++ Application Programming Interface.

5.3 Adaptive Cruise Control

One of the autonomy features we aim to develop, is the adaptive cruise control (ACC), that is an enhancement of conventional cruise control. An ACC system regularly calculates the distance to the vehicle in front and automatically adjusts the speed in order to maintain a fixed distance. Regular cruise control systems are useful on highways for maintaining constant speed. ACC systems provide more flexibility, as they can also be deployed on urban areas with relatively slow speeds, as they can adapt to the traffic environment.

Before we develop the ACC module for the Drive PX2 platform, a proof of concept had to be designed to be executed and tested offline using pre-recorded camera footage obtained using a GoPro mounted on a car. This was developed using the Python programming language along with its Computer Vision (CV) and Machine Learning (ML) libraries, such as OpenCV, TensorFlow, Keras and more.

In order to test the working of this proof of concept, we need data as realistic as possible. In order to gather meaningful data, a GoPro camera was mounted on top of a car and recorded a high resolution video. This type of camera was chosen thanks to its very similar characteristics with the Drive PX2 cameras we were planning to use, such as 120°degree Field of View, high frame-rate and similar resolution. The experiment took place at the Technical University of Crete, with a car normally driving, braking and steering around the campus and the second car with the GoPro mounted on it, following closely behind and recording the whole process.

Our Adaptive Cruise Control software is designed and built upon the Object detection and recognition system described in the previous section. It starts by reading a video file located in its path, the same recorded video described earlier. Then, the model's name is declared along with its URL address. Afterwards, the frozen TensorFlow model's parameters and metadata are loaded into memory, if it already exists in our work-space, or gets downloaded based on the variables mentioned above. A few additional elements must also be imported, such as model specific label maps that match indices to category names, as well as other object detection libraries and helpful visualization utilities. After that, the program sequentially prepares every video frame and parses it through our model for the detection step. Its output is a number of various detections along with their confidence level (score), their bounding box coordinates and size, and lastly the class index that indicates what each prediction corresponds to. It is worth mentioning that for each detection, a confidence level threshold is applied, to ensure only high-score predictions are visualized and handled by the ACC system. In our case, we selected a confidence threshold with a value of 0.86 (86%) that is highly dependant on the object detection model used. This number was heuristically chosen in order to reject as many false positive detections as possible. By observing the detection confidence level of our vehicle of interest during both normal and extreme lighting and angles, we ensured that our threshold would only reject outliers and detection out of our interest. Two other

fundamental variables that the system needs are the distance and horizontal angle of the vehicle we aim to follow. In our use case, the only perception sensor we have in our disposal is a camera, ideally pointing straight towards our vehicle's heading. In order to accurately calculate the distance to the leading vehicle, a depth perception sensor, such as a stereo camera setup, a Radar or LiDAR sensor, is necessary. In single camera setups, the distance calculation can only be done using the bounding box size and will be measured in pixels. Thus, the bounding box diagonal was chosen, as it includes both the width and height of the detected object's bounding box. Thus, we estimate a detected vehicle's distance by using the following equation: $size = \sqrt{width^2 + height^2}$. Each vehicle's angle is dependent on the horizontal center of the camera frame (Equation 5.1), meaning that when an object is dead ahead, the angle will be zero, on the right edge the angle will be +1, and on the left edge it will be -1 as visualised in figure 5.8.

$$ACC_{angle} = \frac{box_x - frame_{width}/2}{frame_{width}/2} \tag{5.1}$$

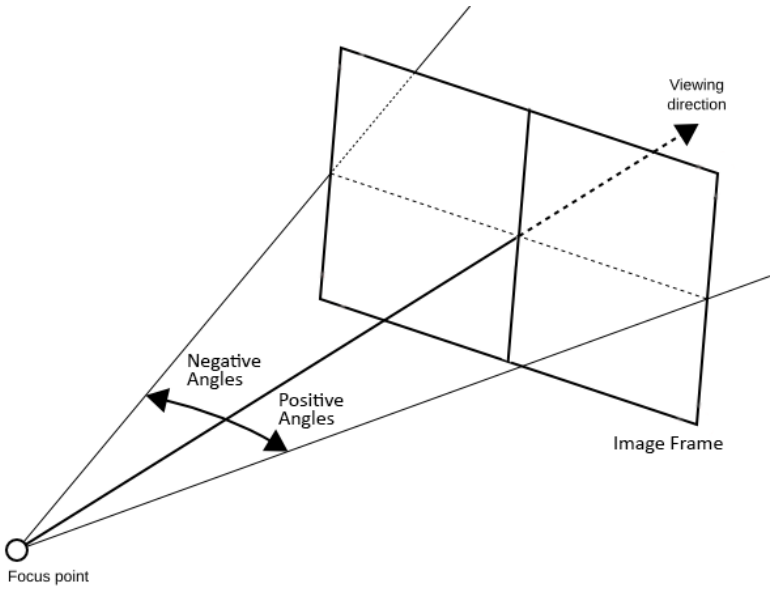


Figure 5.8: Visualisation of how angle is calculated

where box_x is the horizontal coordinate of a vehicle's bounding box, and $frame_{width}$ is the camera's frame maximum horizontal size, measured in pixels. Eventually, the moment we chose to initialize the Adaptive Cruise Control system, the algorithm analyzes the camera feed and detect all possible vehicles. It then isolates the detected objects inside a pre-defined threshold described in equation 5.2, and finally selects and stores the calculated distance and angle of the vehicle that is closest to the center of the video frame as the vehicle the system is now tracking.

$$-2 * angle_{thresh} < ACC_{angle} < 2 * angle_{thresh} \tag{5.2}$$

where $angle_{thresh}$ in our case has a value of 0.08. For the ACC system to work efficiently, it needs to maintain contact with the specific vehicle that has started

tracking. After it "locks" on the desired vehicle, it continuously analyzes the camera frames in a stricter manner. The vehicle's bounding box location in the previous frame is now considered the frame center, and is distinguished at the current frame using a similar procedure but with a smaller threshold (Equation 5.3). The one vehicle with an angle delta ($\Delta Angle_n$) that satisfies the inequality in Equation 5.3, is the one we continue to track.

$$-angle_{thresh} < \Delta Angle_n < angle_{thresh} \tag{5.3}$$

$$\Delta Angle_n = Angle_{n-1} - Angle_n \tag{5.4}$$

where n is the frame number. The angle delta between frames is expected to be reasonably small and for that reason, each boundary's multiplication by a factor of 2, featured in Equation 5.2 is not present in Equation 5.3. In the event that more than one vehicles are detected in the window, our algorithm will calculate their bounding box angle and continue to track the one with the smallest angle deviation. This ensures that our motorcycle will follow a vehicle smoothly and abrupt maneuvers caused by suddenly tracking another vehicle, will be avoided. During the process of tracking the specific vehicle of interest, its bounding box color is altered to white (as seen in Figure 5.9), as a visual indicator of our system's behaviour. In rare occasions that our object recognition model fails to detect the vehicle we are following, the last known position and distance is stored, and after a reasonable number of frames, the algorithm can still resume tracking if the car's angle has not changed drastically.

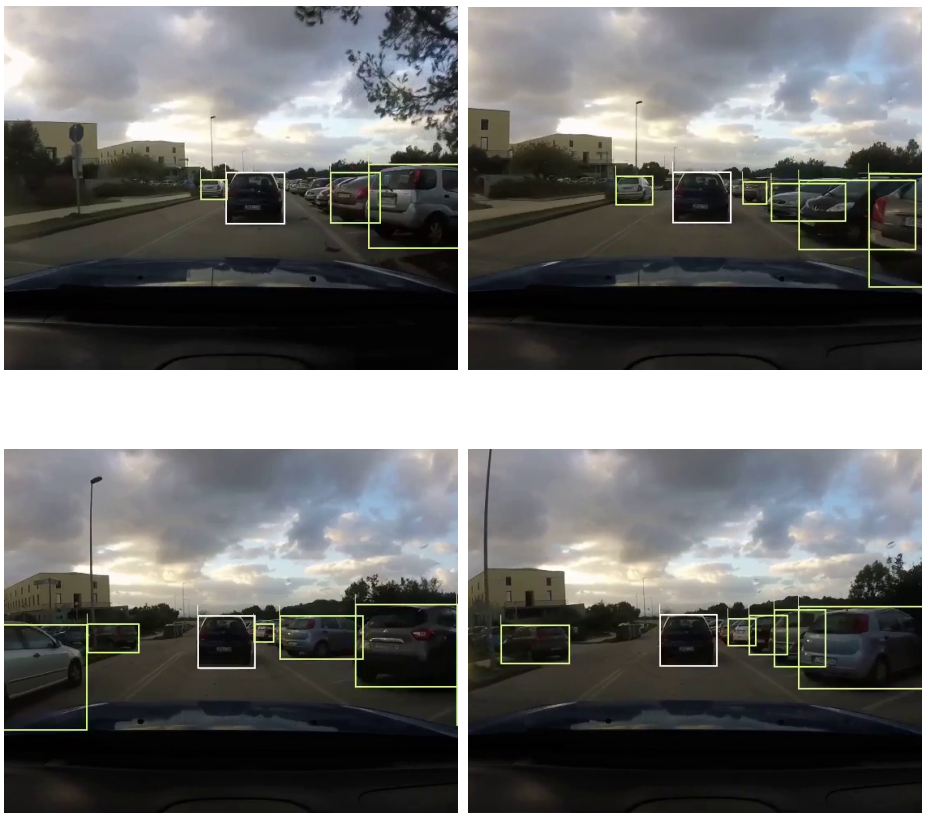


Figure 5.9: Sequence of frames during testing of Vehicle Tracking in a crowded area

In order to validate the systems performance, the GoPro footage was used that contained slow and high speed turns, areas with multiple vehicles present, and both low and high lighting conditions. In Figure 5.9, a sequence of frames are presented, that show how our system keeps track of the desired vehicle (white bounding box), while ignoring the other detected vehicles on the side of the road. Calculated vehicle angle and distance data are also printed on the system console, to verify the correct operation of the ACC system. During the frames in Figure 5.9, the bounding box size was in the range $[0.2, 0.29]$ that translates to $[6, 10]$ meters. Our solution proved to be able to successfully track the leading vehicle throughout the experiment.

5.4 Simulation

An advanced driver-assistance system must be thoroughly tested, before it is released and fitted on an actual vehicle for safety reasons. To validate our approach we decided to employ a state-of-the-art simulator used by many automotive companies. This way, parameters that control the way it operates, can be fine tuned to guarantee its stability and performance. Additionally, the electric motorcycle we plan to deploy this system to, is not mechanically ready at the time of writing and under these circumstances, the Adaptive Cruise Control system will be independently simulated, tested and fine tuned. This way, high testing cost and high risks present in real life testing will be substantially reduced and the system will later be adjusted and fitted into the vehicle easier. The proposed network is illustrated in Figure 5.10 below and will be discussed in the following sections.

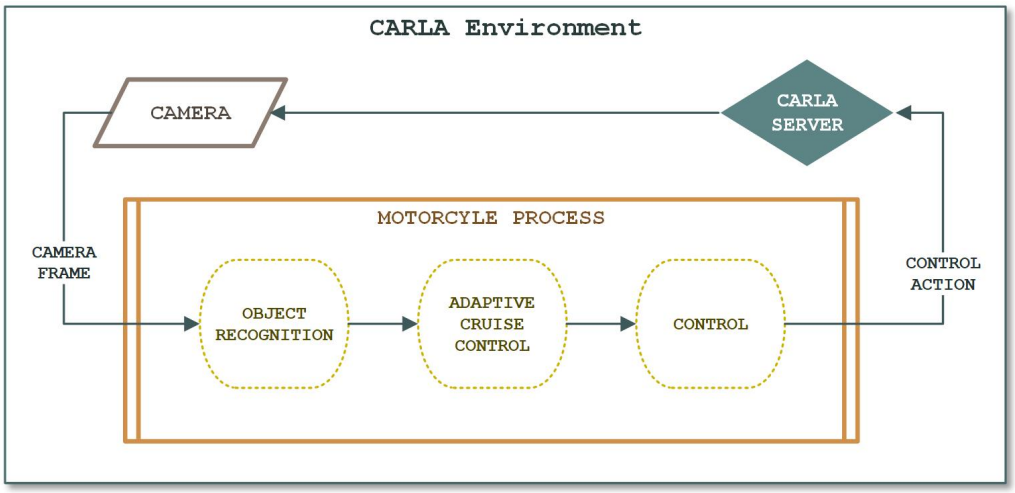


Figure 5.10: Proposed network for CARLA simulation

5.4.1 CARLA Simulator

We decided to simulate our object detection algorithm and the adaptive cruise control system described in Section 5.3 in a safe and controllable environment. The simulator of our choice is CARLA [50], an open source simulator that runs on Unreal Engine 4 and is developed with the aim to support implementation, training and testing of various autonomous driving systems. This simulator is widely used to evaluate sensor fusion, such as LiDARs [51], Radars, Cameras, etc, for object detection and recognition applications, custom dataset generation [52] and neural network

training. It includes a variety of digital assets, such as urban layouts (Towns), buildings, allowing us to design and simulate any desired scenario with multiple vehicles and high-quality graphics and physics. The time of simulation, including sun azimuth and altitude, as well as weather, cloud, fog or rain are highly customizable by the developer in order to simulate any scenario.

Finally, the built-in vehicles that vary from trucks to sedans and motorcycles, include all common sensors, such as depth and RGB cameras, GNSS, IMU, LiDAR and Radar sensors, along with collision, lane invasion, obstacle detectors. Customizability concerning every aspect of the simulation is achieved through the provided Python API. CARLA's construction has a double-head form. To begin with, the server side is in charge of everything related to the simulation itself, such as graphics rendering, lighting and physics computation. On the other hand, the client-side is responsible for the actor's logic, path planning, sensor utilization, as well as setting world conditions. As the number of simultaneous clients connected to the CARLA server is not limited to one, one can simulate traffic and realistic conditions, in parallel to an autonomous system.

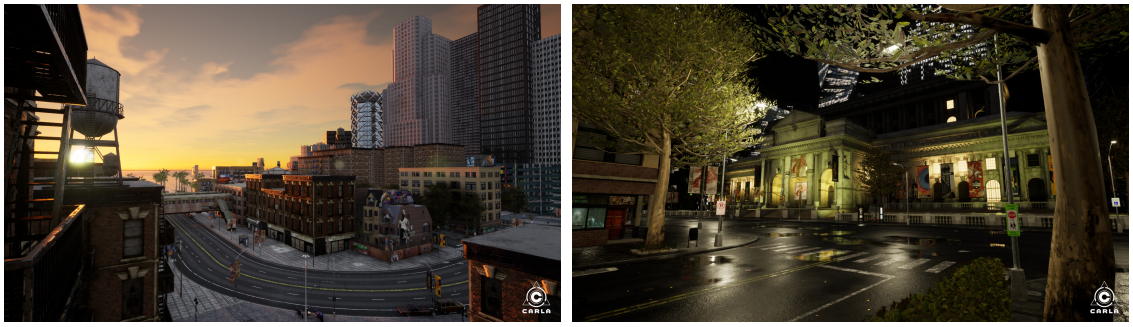


Figure 5.11: Shots of CARLA Town 10

When spawning a vehicle through the Python API, every localization and perception sensor is initialized automatically. This means that the developer has access to sensor data, as well as to systems, such as collision and lane invasion detection, along with semantic segmentation and 3D bounding box display. These systems are useful for some applications, but in our case we need to manually perform the object recognition in the manner that our system will work when deployed on a real life vehicle. Therefore, only the real-time forward facing camera feed is accessed in our implementation and our vehicle is being controlled only using this data.

5.4.2 Proposed Approach

In this section, we propose an autonomous control system capable of leveraging high level software based on Artificial Intelligence and Deep Neural Networks to analyze the motorcycle's surroundings, in combination with a low level control system that takes over throttle and steering input. The goal is to simulate a motorcycle equipped with a front facing RGB camera, and perform the object detection process on its feed in order to analyze the environment. Then, to test and evaluate the adaptive cruise control by setting up a scenario in which our motorcycle follows a vehicle and adapts its speed in an urban environment. CARLA consists of multiple towns of

different sizes to experiment in, and so, "Town10" was selected for this experiment as illustrated in Figure 5.11. As the 3D motorcycle model of the actual vehicle this project is intended for, was not available at that time, we opted for Yamaha's YZF R6 motorcycle with similar characteristics, such as weight and dimensions, as its model is already included in the simulator. The way CARLA simulates two-wheeled vehicles makes them impossible to tumble. In conditions, such as extreme leaning angles, the motorcycle will just spin around. The overall performance concerning steering and power behavior does not have considerable differences and can be adopted to the our intended platform with minor changes.

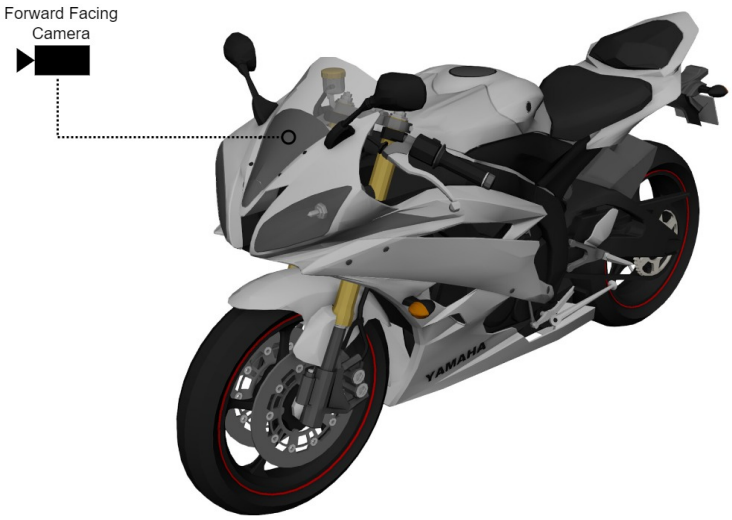


Figure 5.12: Yamaha YZF model with camera placement

In Figure 5.12, the 3D motorcycle model is presented. The camera's placement corresponds to the actual position of the camera in the prototype motorcycle. It is centered between the two headlights (y axis), and elevated to the middle of the windshield by adjusting its x and z axis. In order to simulate this configuration, after the motorcycle is spawned into the world in one of the available spawn points, the sensor has to be initialized.

Figure 5.13: Yamaha YZF in Carla simulation

An RGB camera with a resolution of 960 $\times$ 540 and a 120 degree field-of-view is set-up offsetted on the x and z axis by 0.7 and 1.6 meters respectively, is used.

This resolution and field of view were selected based on the SF3324-100 Camera specifications, which is selected for the real prototype. The camera sensor yields an image for every server tick - an update between the server and the connected clients. The frame is then resized, according to the model's input shape and the detection process is next. This simulated sensor slightly differs from a real life camera, with a minor blur in distant scenes, where we expect the object detection algorithm to have reduced efficiency. As illustrated in Figure 5.14, its effectiveness on nearby objects, such as cars and traffic lights, is at the desired level. As described in Section 5.3, it produces a list of bounding boxes (BBs) containing all the detected objects, therefore we can initiate the Adaptive Cruise Control system.

Figure 5.14: Object Detection in Carla

In Figures 5.15 we can see two frames with the results of object detection while tracking the white vehicle with two cars behind it. The two cars eventually get in the tracking window limits in Equation 5.3, but by analyzing the angle deviation of each car between frames, the system does not lose track.

Figure 5.15: Object tracking with cars surrounding the desired vehicle

In order to debug and study the system's behavior, a data logging tool was implemented using Python's logging libraries [53]. A sample table taken from one of our experiments is displayed in the Figure 5.16.

1	BB Size	BB Angle	X coord	Y coord	Z coord	Speed (km/h)
2	0.026468269668564304	-0.04507508873939514	-114.58673858642578	70.059814453125	-0.05485814809799194	1.057761786454739e-09
3	0.026134966450528374	-0.046200960874557495	-114.58673858642578	70.059814453125	-0.05485814809799194	1.1699229207594827e-06
4	0.025086591203029585	-0.04438796639442444	-114.58673858642578	70.059814453125	-0.05485814809799194	1.4445794960188633e-06
5	0.024534077439331448	-0.041731297969818115	-114.58573150634766	70.115478515625	-0.05480626970529556	3.7713078701179925
6	0.02512315109124419	-0.04322195053100586	-114.5837173461914	70.26360321044922	-0.05477168783545494	6.492301318722043
7	0.023535270807518494	-0.03805568814277649	-114.58023834228516	70.47344970703125	-0.05477147921919823	8.310509708723854
8	0.02495599162287565	-0.03498375415802002	-114.57549285888672	70.72537994384766	-0.05478305742144585	9.653734033280053
9	0.02483281716266106	-0.0282975435256958	-114.56926727294922	71.01033782958984	-0.054794978350400925	10.667576254369573
10	0.026645394765506758	-0.021079838275909424	-114.5619125366211	71.31771850585938	-0.05480766296386719	11.35823871985742
11	0.026315643881610562	-0.021262764930725098	-114.5538558959961	71.64051055908203	-0.054819390177726746	11.766244884982891
12	0.027401000733708436	-0.012702196836471558	-114.54469299316406	71.97052764892578	-0.054830797016620636	11.9682467539427
13	0.027175724798163436	-0.009232133626937866	-114.53533172607422	72.30482482910156	-0.054839495569467545	12.056814098493758
14	0.02865470693363381	-0.013397634029388428	-114.52587890625	72.63971710205078	-0.054846782237291336	12.059324317985203
15	0.027281381218782208	-0.0118979811668396	-114.51554107666016	72.97351837158203	-0.054852791130542755	11.951648819830206
16	0.028574286313261865	-0.007594883441925049	-114.50474548339844	73.30299377441406	-0.0548577681183815	11.826177175310145
17	0.029738838400627188	-0.009618550539016724	-114.49392700195312	73.63018035888672	-0.05485990270972252	11.721502937750506
18	0.027176375120522067	-0.018748939037322998	-114.48277282714844	73.95182037353516	-0.05486356467008591	11.448007610373928
19	0.02751113181187481	-0.011496394872665405	-114.47032165527344	74.26580047607422	-0.054866235703229904	11.256591896697111
20	0.0280699204167556	-0.011747479438781738	-114.45767211914062	74.57821655273438	-0.054863736033439636	11.253963292369264
21	0.028485433285840855	-0.016543447971343994	-114.44462585449219	74.88985443115234	-0.054862212389707565	11.199586835484292
22	0.028946833527258242	-0.019031286239624023	-114.43058013916016	75.19905090332031	-0.054862212389707565	11.089100641070738
23	0.0261039859176595	-0.02147945761680603	-114.41558074951172	75.50444793701172	-0.05486331880092621	10.931865938484734
24	0.024371586158516756	-0.024785876274108887	-114.3995361328125	75.80667114257812	-0.054862458258867264	10.930388143699814
25	0.023224391624435903	-0.022445499897003174	-114.38179779052734	76.11449432373047	-0.054855555295944214	11.287075756213685
26	0.022026789490597665	-0.01930096745491028	-114.36228942871094	76.4355239868164	-0.05484645813703537	11.848921577103292
27	0.020493312654384255	-0.028672635555267334	-114.34097290039062	76.77437591552734	-0.05483798682689667	12.561074508541447
28	0.0200065371618372	-0.02125096321105957	-114.31607818603516	77.13542938232422	-0.054829806089401245	13.450386103566112
29	0.01973620922348296	-0.02617672085762024	-114.28831481933594	77.52310180664062	-0.05482255667448044	14.445990390187916
30	0.019481695000111543	-0.025483936071395874	-114.25666809082031	77.93836212158203	-0.05481778830289841	15.442062582671522

Figure 5.16: Data log sample during experiment in CARLA

The calculated bounding box size and angle are being recorded on every server tick, in the first and second columns respectively. The following three columns contain our motorcycle's X,Y,Z coordinates that CARLA returns. As we can see, the Z coordinate which is the altitude, remains constant because the town we are using is flat, but will be useful for the path drawing in order to track the motorcycle's steps. CARLA also provides three dimensional velocity vectors of the vehicle's current speed in meters per second. We calculate and log its magnitude in km/h using the following expression.

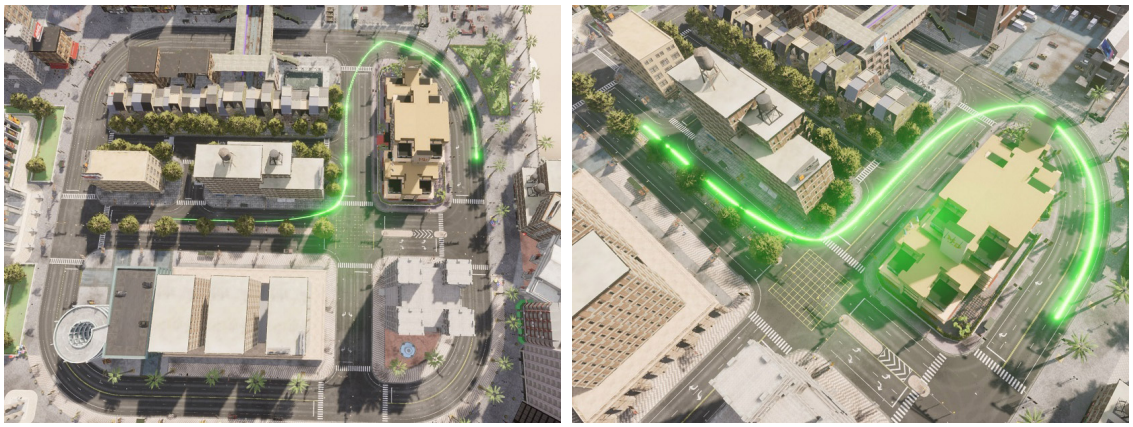
$$Vel = 3.6 * \sqrt{{Vel_x}^2 + {Vel_y}^2 + {Vel_z}^2} \tag{5.5}$$

Where Vel_x, Vel_y, Vel_z are the velocities of the motorcycle in each one of the 3D axis, measured in m/s.

5.4.3 Simple P Controller for Throttle Control

The goal of this experiment is to analyze the system's behavior and the way it translates camera feed data and using bounding box information to actively produce throttle and steering commands. Our motorcycle, as well as every CARLA "actor" can be controlled using the three control functions: *control.throttle(th)*, *control.brake(br)* and *control.steer(st)* with $th, br \in [0, 1]$ and $st \in [-1, +1]$. First, we spawned the leading car, and then, a few meters behind, the motorcycle. The ACC system is initialized the moment the motorcycle spawns, therefore it immediately starts tracking and following the car with a desired distance of about 6 meters. The car was manually controlled using the keyboard arrow keys through a script included

in CARLA's Python API, while the motorcycle's control was completely taken over by the Adaptive Cruise Control system. The path illustrated in Figures 5.17a and 5.17b, was the one that the motorcycle traversed and logged throughout the experiment. It included a slow speed sharp turn and two complete stops, one in the middle of the run and one at the end. These extreme maneuvers where performed in order to check the stability of the system and rule out a possible vehicle collision.



(a) Top view of motorcycle's path (b) Side view of motorcycle's path

Figure 5.17: Motorcycle's path during ACC test

The ACC system calculates the angle of the bounding box in relation to the camera frame center in the range of $[-1, +1]$ from left to right respectively. The motorcycle's steering in CARLA is handled using a floating number in the range of $[-1, +1]$ as a command. As our proposed ACC system in its final form is only in control of throttle and brake, the calculated angle is directly used as a steering command, in order to keep the car straight in front. TensorFlow and ObjectDetection APIs use image frame coordinates to refer to bounding boxes coordinates in form of percentages, in the range $[0, 1]$ for both x and y. This means that the coordinates (0,0) are the start of an image (at the top left side of the frame) and (1,1) coordinates point to the bottom right side of the frame, no matter what the resolution or aspect ratio is. Therefore, a typical bounding box size is in the $[0, 0.45]$ range and during this test, its deviation to the desired box size was multiplied with a weight in order to transform it to the $[0, 1]$ range and produce the throttle command, thus acting as a simple P controller.

In the Figure 5.18, we can view the bounding box size of the detected leading car that was logged during the test along with the desired size, portrayed by a dashed line. The bounding box size represented in the y axis is inversely proportional to the distance, therefore the bigger the value, the smaller the distance to the car in front. In Figure 5.19 we observe the bounding box angle values during the test, with positive values indicating right turns and negative values left turns.

The results from the distance graph look promising considering the use of the bounding box as a distance reference. The motorcycle traversed the path illustrated in Figure 5.17 and followed the vehicle in front through the urban area. The oscillations visible in the distance plot were also noticeable through the on-board camera and were the result of abrupt acceleration and braking. One large spike along with a few smaller ones are spotted that indicate the motorcycle came very close to colliding

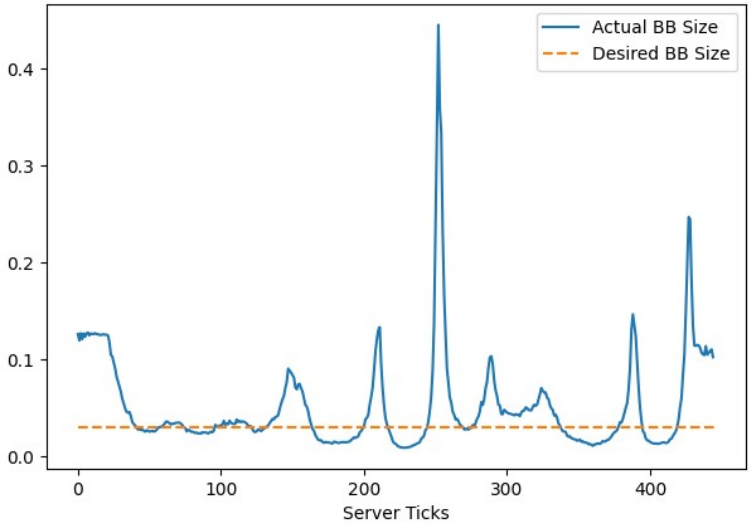


Figure 5.18: Distance plot during ACC test

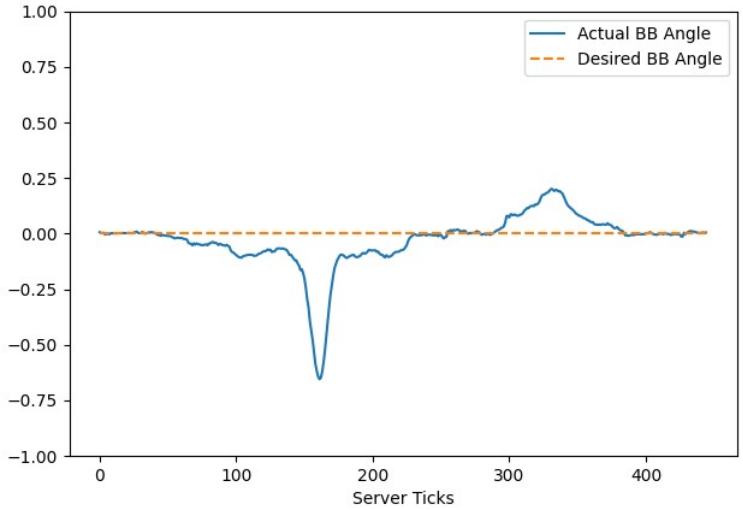


Figure 5.19: Angle plot during ACC test

with the car it was trying to follow, but it successfully braked and avoided contact. A bounding box size of 0.4 is equivalent to around 1 meter gap between the two vehicles. In the angle graph we can notice the sharp left turn at the beginning of the test and the blunt right turn later. This way of translating the bounding box angle to a steering input yielded acceptable results by consistently keeping the following car to the center of the frame. It is important to note that during steering, the motorcycle's velocity plays an important factor. Our low speed during the first turn let the leading vehicle gain some distance throughout the turn with the motorcycle having no time to react and accelerate, thus producing high negative angle values in the first left hand turn.

An important factor that lowers the ACC system's reaction time, is the low frame-rate of the simulation (2-3fps) derived from the CPU execution of the object detection and recognition model. Finally, we conclude that the disadvantages of this

approach, such as the oscillation featured in all P controllers, will produce instability to our system. A controller that acts only based on the distance variation is not suitable for this use case. In Subsection 5.4.4, we redesigned the experiment path and propose a PID controller as a solution, that is not only proportional to the distance error, but also sums up the errors over time and tries to predict the future trend of the error with the Integral (I) and Derivative (D) term respectively.

5.4.4 PID Controller for throttle control

In order to control the throttle efficiently, we investigated the use of a simple, yet efficient, methodology compared to the simple P controller. Several approaches were found in the literature [54], that include self-tuning fuzzy controllers, model reference adaptive systems, etc. We decided to use the basic, but highly effective, concept of a PID controller that constantly checks and corrects the bounding box size of the leading vehicle using throttle and brake commands. Using Python's "simple-pid" libraries [55], the throttle control was replaced by a fully customizable PID controller. Its input (error) is the difference between the current bounding box size calculated by the detection model and the desired size that is set by the user. This setpoint as well as the P, I and D variables are initialized at the ACC system starting point. CARLA vehicle control is achieved with floatin numbers in the $[0, 1]$ range for both throttle and brake. Therefore, the PID controller's output limits are also set to our acceptable range $[-1, +1]$ with positive values corresponding to throttle control and negative values to brake control. Finally, during each server tick, after object detection and recognition, the calculated distance (bounding box size) acts as the negative feedback signal, so the controller corrects its output based on the new conditions and the yielded throttle/brake command is applied to our vehicle. The block diagram of our configuration for the PID controller is presented in Figure 5.20.

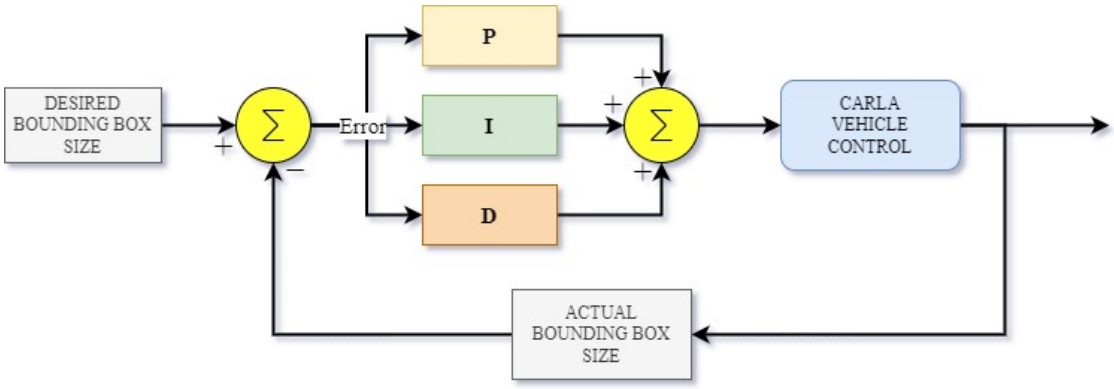


Figure 5.20: Throttle control PID diagram

The PID testing procedure included a path consisting of one wide turn and then mostly of a straight line. The traversed path during this test is presented in Figure 5.21. A constant velocity of 22 km/h was applied to the vehicle in front in order to prevent fluctuations between tests. Next, the system's performance with different PID values was evaluated on the exact same conditions. Four test are completed with the respective distance plots presented in the Figure 5.22. Although not optimal, the PID controller values were selected heuristically, with a goal of producing a

stable system that can be easily re-configured based on the actual motorcycle model, throttle and brake actuators, as well as object detection and tracking efficiency of the Drive PX2 platform.

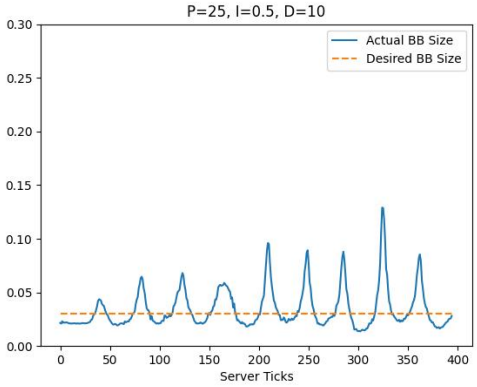
Figure 5.21: Motorcycle path for the second experiment

The first test with a PID configuration ($K_P = 20, K_I = 0.1, K_D = 10$) yielded improved results over the previous version of throttle control. Sharp and slow-speed turns that could shift our system's response are avoided in these tests. In the first distance graph (Figure 5.22a) we observe oscillations around the desired point, that are derived from high acceleration during a negative error value and braking during positive error. During that test, the motorcycle was spotted coming to a complete stop in order to avoid a collision. This was due to a low K_D value, responsible for monitoring the rate of change in the error process that can be quite useful in situations that we are near the desired set-point but still accelerating. Additionally, the integral part of the controller, that sums the instantaneous error over time and gives the accumulated offset that should have been corrected previously would yield negative results in cases with prolonged stops (e.g. during traffic or red traffic light). Thus, it would be best for the K_I value to be assigned a low value, if not zero.

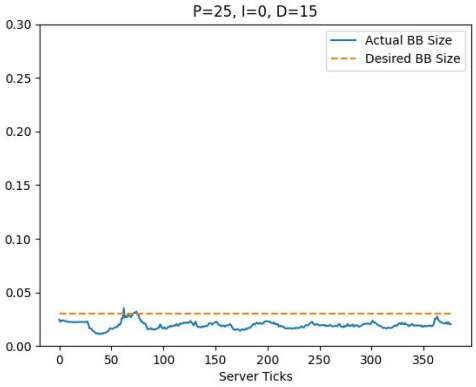
PID Controller Fine Tuning

For the next experiment presented in Figure 5.22b, with values $K_P = 25, K_I = 0, K_D = 15$, the Integral part of the controller was eliminated and the Derivative parameter was increased to counterbalance the oscillations presented in the previous figure. Additionally, it helped make our system more responsive to sudden speed changes of the leading vehicle, offering a more pleasant ride. As illustrated in Figure 5.22b, this decision greatly improved our results with the motorcycle falling slightly behind the desired distance and the fluctuations were entirely eliminated. The next experiment with slightly higher K_P and K_D values of ($K_P = 28, K_I = 0, K_D = 17$) returned similar results with the deviation of the set-point further reduced. On the 4th attempt, illustrated in Figure 5.22d, the K_P variable's raise was higher than

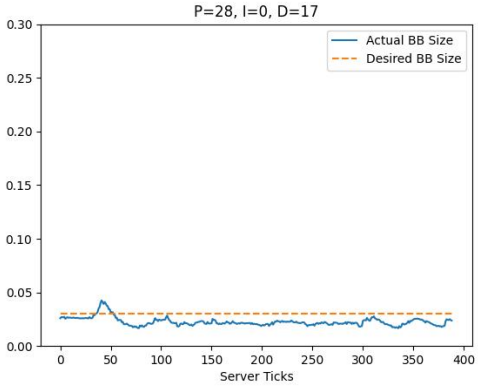
the raise of K_I, which 'broke' the balance we had earlier between these two values and resulted in the reintroduction of oscillations late in the experiment, although improving the error at the start. This, confirms that the two variables K_P, K_D need to be counterbalanced and generate a stable system that offers safety and a comfortable trip to the rider.



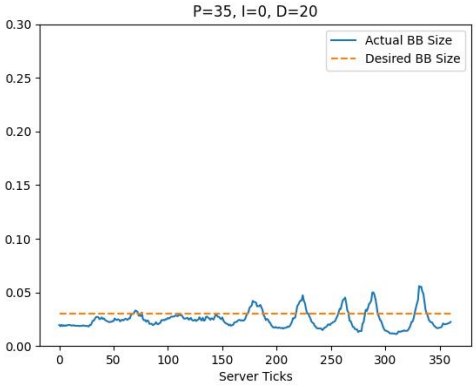
(a) Throttle PID test (25, 0.5, 10)



(b) Throttle PID test (25, 0, 15)



(c) Throttle PID test (28, 0, 17)



(d) Throttle PID test (35, 0, 20)

Figure 5.22: Throttle PID control test results

The data gathered by these experiments, helps fine tune the PID parameters and understand their behavior. In cases where the car we are following is stationary, due to a red traffic light, or a "STOP" sign, and our motorcycle is closer to it than desired, the controller is unable to reverse in order to fix that error, and all it can do is brake. In this, or similar cases, a positive K_I value, that serves as the memory of the controller, is going to result in either very sudden and hasty or delayed and slow response to a change of speed of the car in front. It also has a big impact in situations when the vehicle in front is further away than desired, and thus we accelerate more than the optimal, and abrupt oscillations appear. For this reason we decided to set the K_I value of the controller to zero.

The parameter K_D defines the system's ability to predict a change of state. It proved to be very effective in situations where the vehicle in front make sudden speed changes, and a high value manages to prevent possible collision by keeping the motorcycle in a safe distance. For our configuration the PID parameters that were

found to perform quite well are $K_P = 37, K_I = 0, K_D = 25$, and our motorcycle consistently and smoothly follows the car, in conditions that it keeps a constant speed, such as highways. The experiment results using these parameters and the same path shown earlier in Figure 5.21, are illustrated below in Figure 5.23.

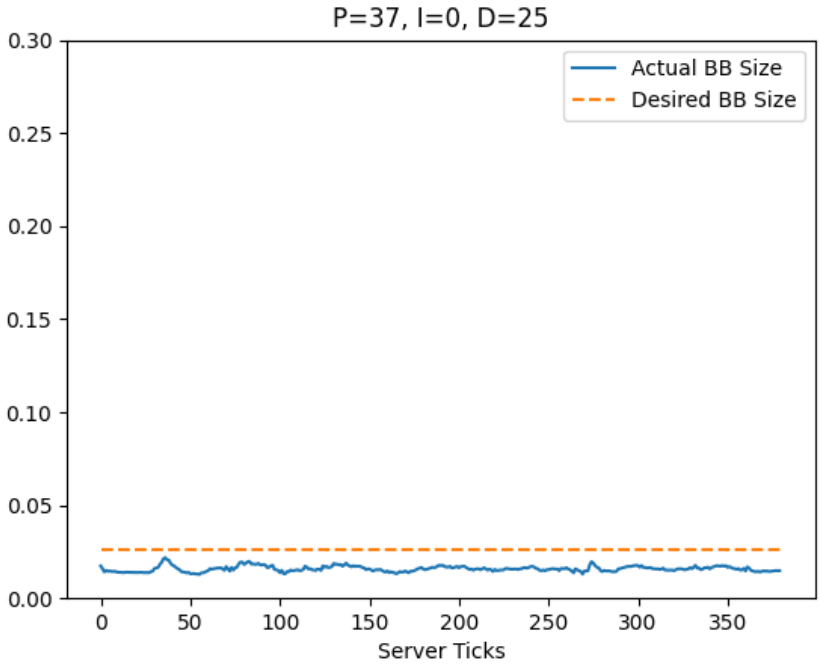


Figure 5.23: Distance plot during ACC test with the fittest PID parameters

The actual vehicle distance throughout this test was slightly higher than desired, as a result of the low frame-rate of the simulation and the bounding box size value being in the compact range of [0, 0.05] and thus, small-scale size differences are overlooked by the system. This is one of the reasons, further fine-tuning of the PID controller parameters is troublesome. A more effective object-detection model, able to increase the processed frames per second to at least 10, as well as a more efficient way to calculate the distance to the leading vehicle, such as Radar or LiDAR sensor, and of course, the model of the actual motorcycle it will be deployed on, are expected to yield better results.

5.4.5 Extensive PID Experiments

In order to assess the system through various conditions, we conducted an extensive experiment that covers a great portion of the town and included random stops of the leading vehicle to simulate traffic or red traffic lights. Its speed was manually controlled varying from 0 to more than 60 km/h. The motorcycle was automatically following the car using the ACC system and its traversed path is shown in Figure 5.24. As in the previous experiments, the motorcycle does not have a lane following algorithm and acts only as commanded by the Adaptive Cruise Control system, that has a target of staying at a pre-defined distance from the leading vehicle. Thus, rules concerning lane following and compliance to traffic laws are not followed.

The results are displayed in Figure 5.25, and include the bounding box size and angle

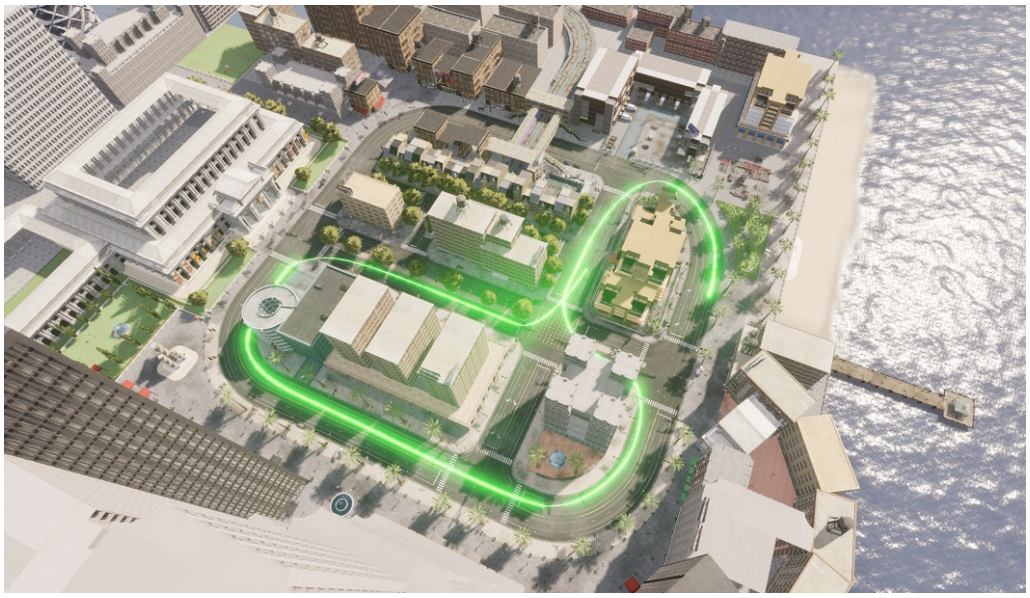
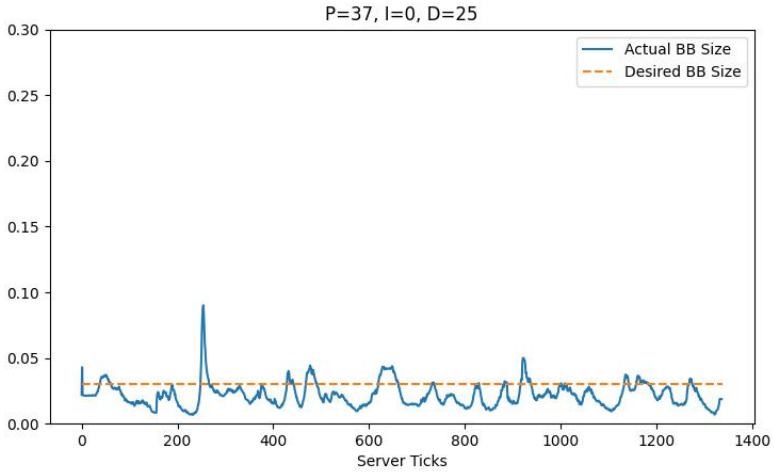


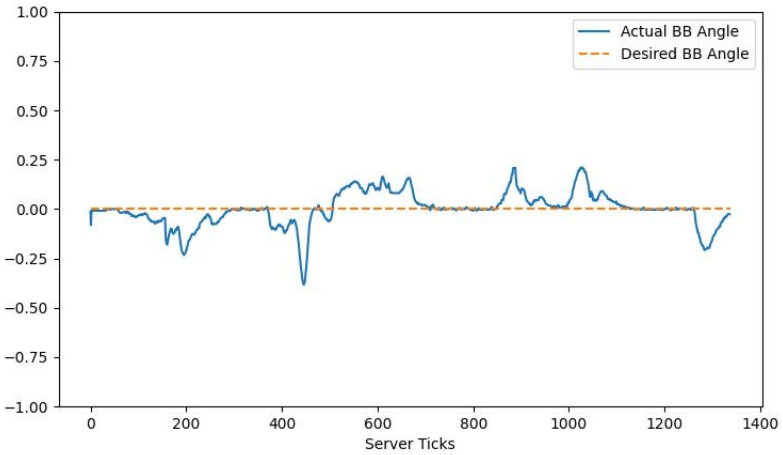
Figure 5.24: Motorcycle path during the extensive experiment

as well as motorcycle speed that were recorded during the experiment. As expected, the angle was kept in the tight range of $[-0.4, 0.4]$ which means the leading car did not maneuver away from the motorcycle's field of view. Combining the BB size with the speed graph, we can deduct the reason of oscillation reappearance in the distance graph. At around 250 server ticks, where the motorcycle's speed is at the maximum recorded, the leading car performs a sudden stop, that leads the motorcycle to brake, and results in the spike seen in the BB size plot. It is noteworthy that the 0.10 value of size is translated to about 4 meters of distance between the two vehicles. Furthermore, in the speed plot, we can distinguish the high speed portions of the experiment, as well as the parts where the leading car completely stopped.

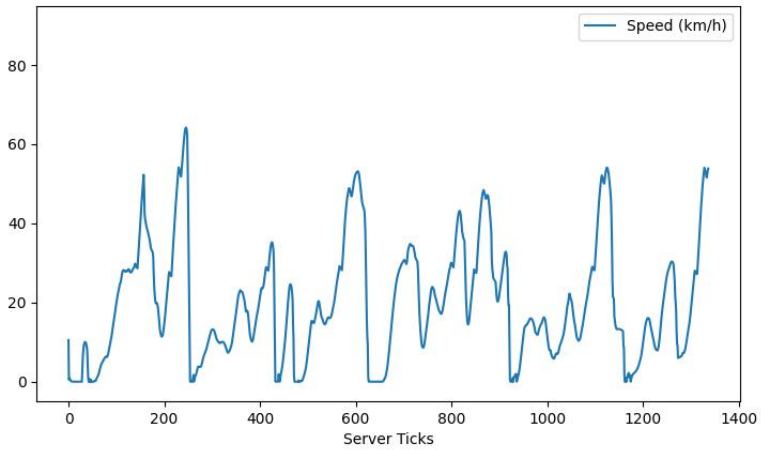
For the entire experiment duration, the results indicate a satisfactory system behavior. Looking at the BB size graph, despite the reappearance of mild oscillations, the distance between the two vehicles is constantly at a safe level, that excluding the exception at 250 server ticks, its maximum value is at 0.05. This occurs, while the motorcycle's speed is constantly changing in order to adapt to the leading car. Finally, using the bounding box angle data, the motorcycle consistently corrects its path in order to follow the car during a turn.



(a) Extensive test, BB size graph



(b) Extensive test, BB angle graph



(c) Extensive test, motorcycle speed graph (km/h)

Figure 5.25: Extensive PID test results

Chapter 6

Platform Programming

6.1 Nvidia Drive PX2 Software

6.1.1 Software Development

The software development for the Drive PX2 platform requires a cross-compilation workflow, meaning a 3rd party Linux host system equipped with an Nvidia graphics card is needed. Although, each of the Tegra sub-systems of the platform can act as a separate ARM-based computer, alongside its peripherals (monitor, Ethernet, keyboard, mouse), it lacks the capability to compile source code intended for itself. For this reason, in order to test and debug programs that use Drive PX2 features and its connected sensors, a host system with all Nvidia DRIVE and CUDA SDKs installed, is needed. Using arm64 packages, Nvidia libraries and the DRIVE PX 2 toolchain, we cross-compile our software on the host system and make the binaries available to the target system (Drive PX2) by copying them to its working directory.

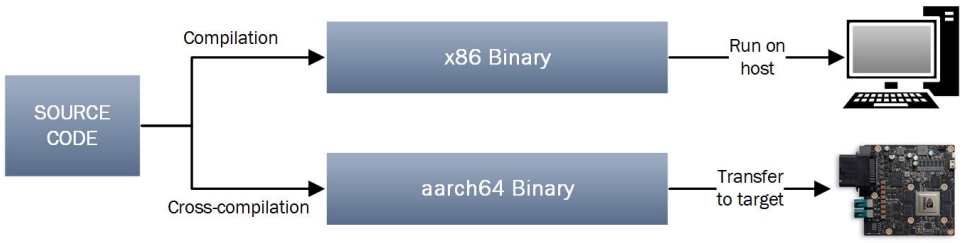


Figure 6.1: Source compilation workflow
Source: Nvidia Developer Blog

6.1.2 Object Detection

The ability of an autonomous vehicle to detect and track surrounding objects is essential to its safety. In this work, it is achieved using a single forward-facing, high field-of-view camera. Nvidia's DriveWorks SDK includes object detection and object tracking models trained for DRIVE platforms. DriveNet is a sophisticated, multi-class, higher-resolution sampling classifier that uses a trained proprietary deep neural network (DNN) to perform object detection. It is designed to perform inference on

either RAW camera stream or pre-recorded RAW video, making it ideal for testing before its actual deployment. Other models included in DriveWorks SDK are lane detection sample (LaneNet) and free-space detection (FreeSpaceNet). It is worth noting that the source code for these samples is available to every DRIVE platform developer.

The motorcycle's driver should have the option to view the real-time analysed camera feed through the display. The DriveNet version that is provided to us by the Drive system, is used as groundwork to detect and localise cars, bicycles, pedestrians, traffic signs and traffic lights. By default, all object detection models are executed on the system's dedicated GPUs in real-time with a constant performance of 23-29 frames per second. Detections coming from DriveNet are associated with a confidence value in the (0,1) range, an urgency value (1/s), which is the inverse of time to collision, and a distance value (m). The urgency value and distance estimation, are not released by Nvidia for Drive PX2, and are destined for future models. For this reason, each detection's distance is estimated by the size of its bounding box similarly to Section 5.3, as the square root of the sum of its dimensions squared ($size = \sqrt{width^2 + height^2}$). The only difference with our ACC system described in Section 5.3, is that the bounding box position and size in the camera frame is interpreted by pixel coordinates, and not via a normalized vector in the range $[0, 1] \times [0, 1]$ regardless of the frame size, featured in section 5.3. After modifying DriveNet's source code, we isolate the low-confidence and demonstrate the high-certainty detections, so the system and the user will not see ambiguous predictions. Lastly, the user is given the option to turn on or off the drawing of bounding boxes for each detection category (traffic sign, vehicles, pedestrians) separately to match their preferences.

DriveNet is trained and optimised for daytime and clear weather, and its performance may differ under dark or rainy conditions. Additionally, we have to consider that the training data collected came from the United States and might have different performance in other locales.

(a) Car detection example (b) Traffic sign detection example

Figure 6.2: Modified DriveNet running on Drive PX2 with pre-recorded video

The way this algorithm works is by cropping and down-scaling each image by half so the resulting frames have a resolution of 960×540. Additionally, we have the option to enable foveal detection mode, which increases detection rate and works by further cropping the center region of the original image, while maintaining its

aspect ratio. Then, a follow-up algorithm clusters detections from both images to compute a more stable response. The foveal detection region is shown by the yellow box in Figures 6.2.

6.1.3 Rear Facing Camera

The proposed configuration of this vehicle includes one forward and one rearward facing camera. As the motorcycle features no rear view mirrors, the rear camera will assist the driver's situational awareness through the on-board display. The camera feed will not get analyzed by an object detection algorithm during this work and will only be used to provide a view of what is happening on the back of the vehicle. An included program along with its source code, leveraging DriveWorks SDK libraries to provide real-time footage of cameras, is adopted for this use case. It is modified to select the camera connected to the correct port of the Drive PX2 system, (in our case, port A1) and displays the 30 frames per second feed in full screen.

6.1.4 Autonomy and Adaptive Cruise Control

The object detection system described in Section 6.1.2 serves as the groundwork of the motorcycle's autonomous capabilities. It is used to observe the environment in front of the vehicle through the camera, and serves as a driver assistance system. The source code is modified to provide the option for the drivers to personalise what they want to view through the display. They can chose to enable the drawing of regions of interest (ROIs) of the different types of detected objects or disable them selectively. They can specifically choose the ROI display of detected cars, motorcycles, traffic signs or pedestrians. Additionally, for each one of those detected types, a different type of functionality is implemented for a different use case.

When traffic signs are detected, their bounding box is highlighted to the driver if the respective setting is enabled, and their coordinates in the picture frame are stored. This is done, because the detection algorithm is trained to detect where a traffic sign is located, but not its meaning. Therefore, we use the information about the location and size of the bounding box to provide a cropped image containing only the traffic sign and not the rest of the camera frame to the traffic sign recognition algorithm. This way, we avoid additional image segmentation, clustering and detection that a typical traffic sign detection and recognition algorithm would normally do, thus conserving computation power and resources.

On pedestrian detection, apart from the display of the ROI for every spotted pedestrian, a safety functionality was also implemented. Using the coordinates of the bounding box, we can determine if the pedestrian is in the vehicle's path and warn the driver. This is achieved by calculating the bounding box's angle in relation to the horizontal center of the camera frame, similarly to the angle calculation in Section 5.3 using the following equation.

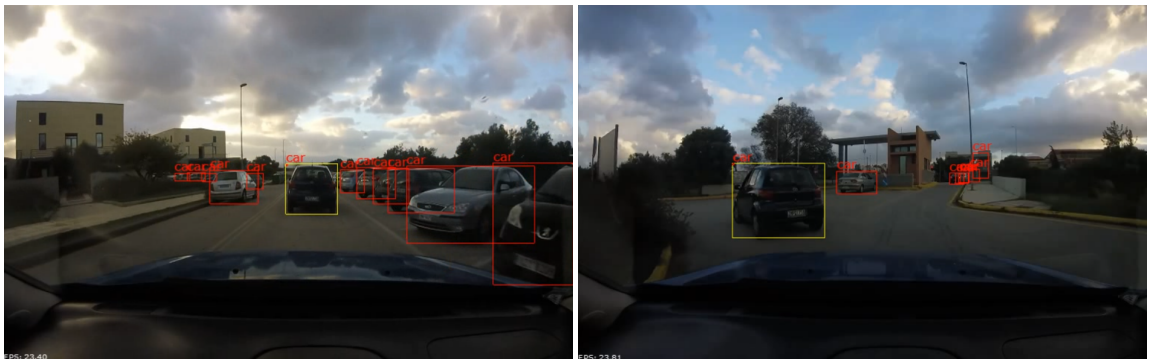
$$Pedestrian_{angle} = \frac{box_x - frame_{width}/2}{frame_{width}/2} \tag{6.1}$$

Then, we determine if the person is on our path using Equation 6.1, with a higher threshold of $angle_{thresh} = 0.12$ that widens the inspection area and thus improves

safety.

$$-2 * angle_{thresh} < Pedestrian_{angle} < 2 * angle_{thresh} \tag{6.2}$$

Finally, the bounding box display of detected vehicles can also be controlled by the user. Detected vehicles are taken into account in the adaptive cruise control process. When initialized, we start tracking the vehicle closest to the center of the camera frame, or wait until one moves near it and then track it, exactly like we do in the proof of concept in Section 5.3. The bounding box properties of the tracked vehicle are sent separately to Nvidia's render engine, so we draw it using a different color. Results from offline testing using the same pre-recorded video as in section 5.3 are illustrated in Figure 6.3.



(a) ACC tracking in crowded area (b) ACC tracking with high relative angle

Figure 6.3: ACC tracking on Drive PX2 using a pre-recorded video

The vehicle being tracked by the algorithm is in the yellow bounding box, while those in red box are the remaining detected vehicles. We can confirm the effectiveness of the designed system in a crowded area with vehicles on both sides of the road, as well as in an extreme condition that the tracked car deviates from the center of the camera frame.

Final step for the adaptive cruise control system is to adjust the motorcycle's speed in order to maintain a constant distance from the vehicle in front. At the moment, there is no implemented system fitted on the motorcycle that controls the throttle and braking systems, neither a sensor for the Drive PX2 system that accurately calculates distances like a LiDAR or Radar. Thus, the metric we use as a distance measurement is the size of the bounding box's diagonal measured in pixels, just like we do in Section 5.3. Then, a simple P controller continuously compares the current distance to the desired distance and yields an output that is destined for throttle and brake control.

$$Control = \frac{dist_{desired} - dist_{current}}{100} \tag{6.3}$$

where distances are currently the box diagonals, and can later be replaced by values measured from sensors. Future work can use this system as groundwork, as it can be later fine tuned based on data gathered from simulation of systems controlling

throttle and braking, as well as how the motorcycle responds to different kinds of inputs.

6.2 Grayhill 3D70 Display

Qt is widget toolkit widely used in automotive systems for designing and developing graphical user interfaces, as well as cross-platform software intended for various system configurations and operating systems such as Linux, Windows, Android and many more. Numerous car manufacturers have introduced Qt in digital instrument cluster design. Qt integrates C++ programming logic with QML applications. QML is versatile modeling language for designing user interface–centric applications and allows them to be described in terms of their visual components and how they interact and relate with one another. It allows complicated designs to be split down in to multiple components and offers complex animations and effects without compromising performance. Grayhill offers installation instructions, as well as libraries and support files required by Qt IDE, in order to develop, build and deploy applications for their 3Dxx display models.

6.2.1 Graphical User Interface

The designed Graphical User Interface consists of three different display arrangements, each for a different driving scenario. The "Main View", is illustrated in Figure 6.4, and consists of an analog speedometer and a battery level indicator, and a variety of other indicators in the form of the traditional "light bulb". Due to the fact that the motorcycle has no rear view mirrors, the driver must have a view of the situation behind the vehicle at all times, so the camera mounted on the rear is always shown at the top half of the screen. The driver can swap between display arrangements using the two touch buttons on the bottom left. All data displayed (speed, battery level, indicators, etc), are implemented to acquire information through the CAN bus connector in the future, requiring minimal development, except the camera feed that utilizes the display's analog video input.

The analog speedometer is a built-in QML component called "Circular Gauge" and can be customized according to our needs. For instance, the size and color of the needle can be redesigned, the maximum speed value and step between subdivisions can be adjusted according to the vehicle's specifications and many parameters more. The battery level indicator is a similar QML component called "Gauge" with similar customizable parameters as before. To create the two buttons used to select view modes, we used the QML component called "Button" that natively works on systems with either a touchscreen or a mouse cursor. Their text size, font and color, as well as the text content itself, can be personalized. When pressed, they change the state variable, which is responsible for shifting to a different view mode. Finally, the "light bulb" indicators are simulated using an "Image" QML component for each indicator that are contained in two "Row" components in order to keep them leveled and in order. Their active and inactive states are simulated by changing their opacity between 100 and 15 percent respectively.

The camera feed is set to be permanently displayed in the background in full-screen. This means that we have to keep a transparent portion of the rest of the design,

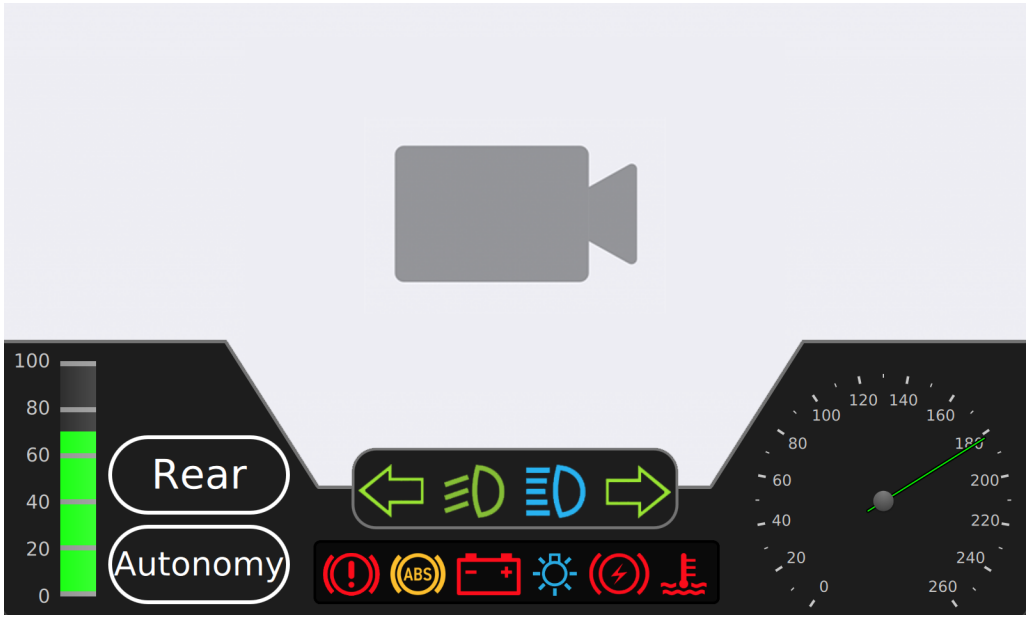


Figure 6.4: Main display view, designed on Qt Creator

the top half portion of the screen in the "Main View" instance, in order for the camera feed to be visible. Unfortunately, during development in the Qt IDE, it is impossible to simulate the display's inputs and that includes the camera, and hence we assigned a placeholder in its place. In the following chapter, that discusses the experiments and how the whole system is connected together, the "Main View" mode with camera feed directly from Drive PX2 is included.

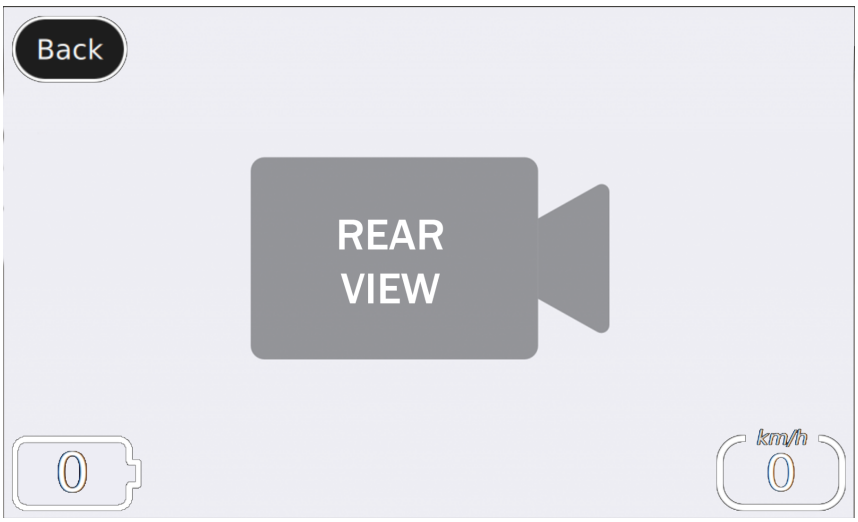


Figure 6.5: Rear View Screen Arrangement

The next two arrangements available, are called "Rear View" and "Autonomy Mode". The "Rear View" mode (illustrated in Figure 6.5) is essentially a full screen view of the rear camera feed and was designed to provide as much information as possible about the situation in the rear of the vehicle. In this mode, only essential information is displayed to the rider, such as the current speed and battery level, in digital representation rather than analog. This is accomplished using custom transparent images (a battery shape and a rounded rectangle) to surround the text

that contains battery and speed data. This way, we avoid the bigger and space-consuming gauges we have in "Main View" and still provide the same information. Naturally, the driver can exit this mode using the virtual "back" button and return to "Main View".

Autonomy Mode (illustrated in Figure 6.6) is similar to the previous mode. It contains a maximized view of the front camera feed. This feed also contains information from the autonomy software described in Section 6.1.4. All detected objects are highlighted with bounding boxes to the driver, along with the option to control the display for each type of detection (pedestrian, vehicle or traffic sign) giving them the freedom to personalise this mode however they want. This is achieved using a built-in "Toggle Switch" QML component for each option and the corresponding image next to it. Also, in Figure 6.6 an alert is shown that warns about pedestrian in our path, which will be raised if Drive PX2 software discussed in Section 6.1.4 yields it. Additionally, while the adaptive cruise control system is active, the rider can monitor the vehicle we are following as well. Finally, just like the previous mode, information concerning battery level and the motorcycle's speed is also digitally displayed.

Figure 6.6: Autonomy View Screen Arrangement

6.2.2 Connectivity

With this GUI design in mind, the connectivity ports that are required will be a single CAN port, two camera connectors, one for the front and one for the rear view camera, and of course the power supply port. USB and Ethernet ports are also used for software development, debugging and the upload of binaries to the display, but they are not needed while deployed on the motorcycle. Due to the size of the development board and the fact that it does not have any kind of weather or temperature protection, we must replace it with a more elegant solution that suits our needs. In order to replace the development board, a circuit that converts the single 18-pin input cable to the ports we need in our application is designed. The display will be powered through this cable, which is the official cable provided to us by the manufacturer, and it will also be used to transfer data for the rest of the ports discussed earlier.

6.3 Experiments

The target of this project is to provide a functional platform running driver-assistance features on the Drive PX2 platform, an interactive and customizable dashboard using the touchscreen 3D70 display, as well as ensure the communication between these two sub-systems. In Section 6.1.4 we discussed the use of a pre-recorded video for object detection and tracking testing purposes. Our software successfully tracked the desired vehicle throughout a lap around the Technical University of Crete campus and reported the calculated distance and angle of the car in the terminal. These values would be used in the PID controller responsible for throttle, brake and/or steering control when it is deployed on the actual motorcycle, similar to our simulated approach in Section 5.4.2. In Figure 6.7 we can see the proposed configuration featured in this experiment.

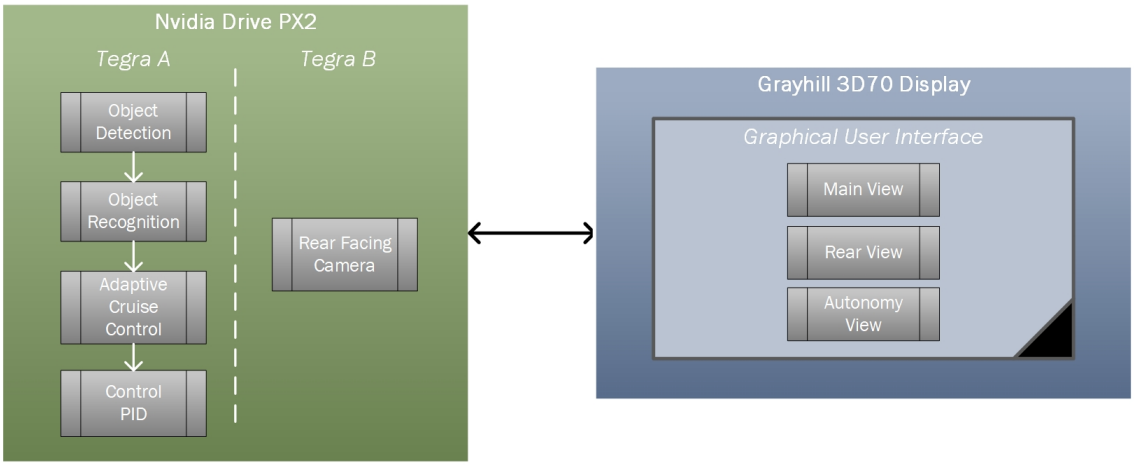


Figure 6.7: Main Display View experiment configuration

The designed dashboard GUI for our display was cross-compiled for the ARM processor used by the display. The binary files were transferred to the display system through an Ethernet connection to a Windows system using the WinSCP application that allows SSH file transfer. In order to execute commands like shutdown, restart or execute the GUI software in the display's operating system we connect to the device through SSH with the PuTTY program. Touch input data is natively supported in this display, and thus the GUI's functionality was exactly as simulated in the Qt IDE during development.

Similarly, the Object detection and recognition algorithms, as well as the Adaptive Cruise Control and PID systems, were as seen in Figure 6.7 cross-compiled for execution on the target device (Drive PX2) and transferred to it over the network. The experiment illustrated in Figure 6.8, was run in order to demonstrate how the HDMI display output of Drive PX2 Tegra A is portrayed on our display.

We observe how our system components connect to each other. Using an HDMI to AV adapter, the digital display signal is converted to the analog format our display requires. For the current testing purposes, the ACC system was running on the Drive PX2 with the pre-recorded front camera video and the dashboard GUI was running on the display, as seen in Figure 6.9. Normally, in the illustrated page of the dashboard GUI, the rear camera is shown, but since it was not simulated, we opted

Figure 6.8: Main Display View with front camera feed and ACC system

for the forward facing camera. Additionally, this experiment helped us verify that the GUI mostly obstructs viewing angles on the bottom corners and the important information present in the middle of the camera frame is still visible to the rider. Additionally, since the Drive platform is not set up to transmit data concerning the vehicle speed or battery level to the display, the dashboard functionality is limited.

Figure 6.9: Close up of the Main Display View

Chapter 7

Platform Deployment

Due to manufacturing constraints, our configuration, including the power supply, Drive PX2 and camera mounting and wiring as well as the display's deployment board, were not produced and tested on the actual motorcycle. This Chapter discusses our proposed subsystems for the final system, that are designed but not manufactured. A power supply that could handle both Drive PX2 and 3D70 display's power requirements while offering high efficiency and small form factor is selected and proposed in Section 7.1. In Section 7.2, we explain the Drive system powering and wiring, which is designed based on detailed guidelines offered by Nvidia to ensure the equipment's safety. Additionally, we designed a deployment board schematic with the corresponding Printed Circuit Board design, based on the display's development kit circuit board that was provided by Grayhill. Finally, in the final section of this chapter, the proposed system is illustrated in a diagram that provides both data and power flow information and shows how the sub-systems will communicate with each other.

7.1 Power Supply

Due to the motorcycle's compact body configuration, an auxiliary 12 Volt battery is not a viable solution. Our proposed solution utilizes the existing battery pack used by the electric motor in order to power other sub-systems, such as the display and Drive PX2 system. This solution comes with the complication of dealing with the variation in battery's voltage level (40 to 50Volts DC) depending on the battery's level percentage, as well as the motorcycle's motor different power requirements based on different driving conditions. Additionally, we must meet the theoretical maximum and typical power requirement of 300 Watts and 90 Watts respectively for the Drive PX2 system, and 7 Watts for the 3D70 Display. In practice, we will not reach the theoretical maximum power requirements of the Drive PX2 system as the system would need to operate constantly at 100% and that would cause a complication with its cooling and high-power draw that could pose a problem with the battery life and motorcycle's range. The proposed solution is the MEAN WELL RSD-300C-12 isolated DC/DC converter [56] with a maximum output power of 300 Watts.

Its input includes an EMI filter, isolation resistance of 100 MOhms / 500Volts DC,

Figure 7.1: MEAN WELL RSD-300C-12 Isolated DC/DC Converter

along with short-circuit, overload, over-voltage and over-temperature protections making it an excellent choice for protecting the valuable and costly systems it will power. With an operating input Voltage range of 33.6 to 62.4 Volts, it can deliver regardless of the battery's voltage level with 91% efficiency. Other important factors include its wide supported working temperature in the range of -40 to $+70$ °C with adequate cooling, 5% to 95% non-condensing humidity as well as a maximum operating altitude of 5000 meters. Finally, its small form factor of 216×96.5×40 mm and weight 1.19Kg will play a big part in deploying and mounting it to the motorcycle's chassis.

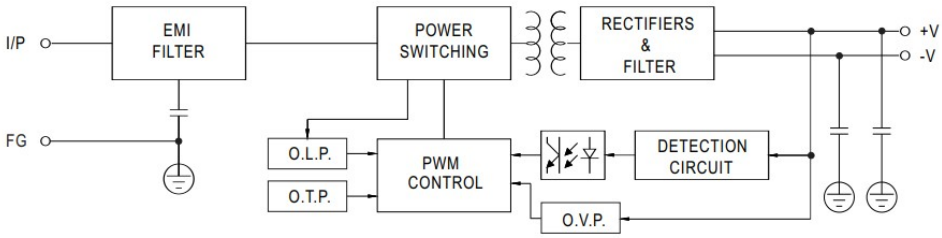


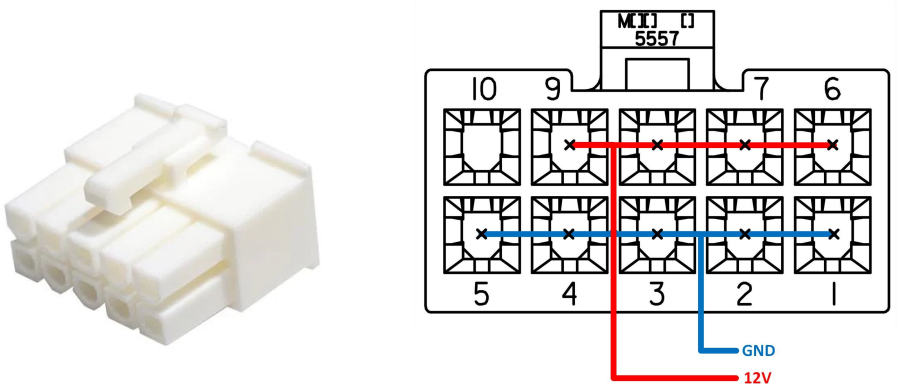
Figure 7.2: MEAN WELL RSD-300C-12 Isolated DC/DC Converter Block Diagram [56]

7.2 Drive PX2

The Drive PX2 system's bigger dimensions of 303×213×84.3 mm along with the recommended minimum keep-out zones of 35mm on each side and 63.6mm on the top to avoid obstructing the connectors and cooling, are challenge for its positioning. The system is destined to be mounted beneath the driver's seat. In order to offer protection against moisture and dust, as well as to provide adequate fresh air for effective cooling, an enclosure case is necessary. This enclosure case must be dependant on the Drive PX2 system's dimensions as well as the free space available beneath the driver's seat, and is not designed or developed in our work.

The included power supply delivers 12Volts to the system via a simple 10-pin Molex

connector with part number 39-01-2101 [57] shown in Figure 7.3a. Its wiring is as follows: the first five pins along with the pin number 10 are used for ground connection, with the remaining four pins (pins 6 to 9) supplying 12Volts. Pins 6 and 7 are used for the Tegra 1 and 2 subsystems respectively, and pins 8 and 9 are used to power the two dedicated GPUs (dGPU) installed. Pin 10 is not used. In cases where we have no dGPUs installed in the system, these two pins are optional and the whole system would require a maximum power of 150 Watts. In our case we are using both subsystems along with their dGPUs. This Molex connector has a current rating of 13Amps and 600Volts.



(a) Drive PX2 Power Supply Molex (b) Drive PX2 Molex wiring

Figure 7.3: Drive PX2 Power supply

For use in vehicles with internal combustion engine, the Drive System is powered by the 12Volt battery, and during cold crank cases, battery voltage could drop as low as 4.5Volts. For 300Watts of power Drive PX2 might draw, it would require 67Amps and thus Nvidia recommends an 80A fuse to be used. In electric vehicles were there is no case of cold engine crank, as well as in our case, the system is powered by the DC/DC converter with either 12Volts or 0Volts output in case its input conditions are not met. 300Watts at 12Volts require 25Amps and adding 20% variation, a 30Amp fuse will be used alongside the TP6KE33A [58] TVS diode placed in parallel to our circuit to protect the sensitive electronic equipment from voltage transients. The proposed circuit used to power the Drive system is portrayed below in Figure 7.4.

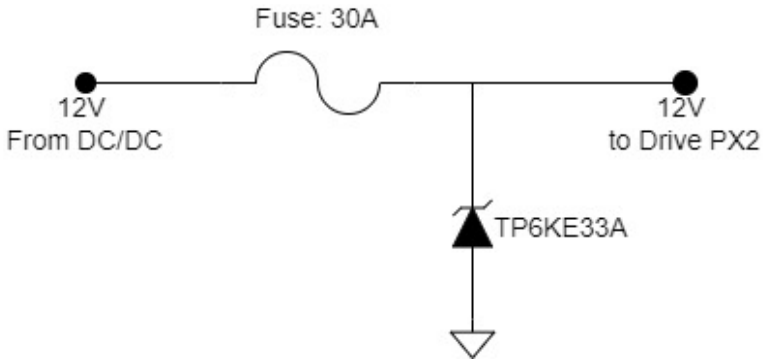


Figure 7.4: Drive PX2 Power Circuit

7.3 Camera Mounting and Wiring

The motorcycle is planned to feature two camera sensors in its configuration. Their small factor allows them to be easily mounted and concealed, while maintaining good view of the vehicle's surroundings. The front facing camera is mounted between the two headlights and the second one, on the rear section of the motorcycle's body to provide the rear view feed to the driver. As these camera models are intended for automotive use and feature IP69K protection, a sealed enclosure is not required.

Figure 7.5: Camera SMK CRS9001 Connector

Additionally, as illustrated in Figure 7.5, these sensors only require a two-wire connection, that acts both as a power input and data output. The cable connector SMK CRS9001 (FAKRA type) is all we need between the camera sensor and the Drive PX2 system and also supports IP69K protection. These qualities, featured in both the camera and its connection that eliminate the need for complex harnesses, provide an extra level of freedom for the mounting and wiring on the motorcycle.

7.4 Display

As previously mentioned, Grayhill's development kit for the 3D70 display is only utilized for laboratory use and will be replaced upon deployment. For the display mounting on the motorcycle, the included mounting frame is used. It can be permanently mounted on any surface using four screws and with minimal clearance behind it we can secure the display and access the two wire harness connectors. The mounting frame along with the display are presented in Figure 7.6.

As discussed earlier, a board that converts the two 18pin display connectors to the correct port for inputs or outputs that the display support is designed. The included GrayHill quick start guide includes a detailed schematic of the development board, which was used as foundation on designing our replacement board. It is designed using KiCad, an open source software suite, to be a small-factor alternative that consists of only the necessary connections for this project. That includes two analog video inputs for the two cameras, its 12Volt DC power source and a CAN bus connector. These specific connections happen to make use only of a single 18-pin Harness connection (DT Connector A) and the second one (DT Connector B) can be

Figure 7.6: 3D70 Display and its mounting frame

ignored. Naturally, if the need arises in the future, the board can be easily redesigned to additionally include an Ethernet, USB or general purpose IO port, although it might require the utilization of the second 18-pin harness. The proposed schematic is presented below in Figure 7.7. It is important to state that all connectors used in this design (CAN, Video input, power input etc) are recommended by GrayHill and are also present on the development board. This means that depending on the deployment scenario and the conditions the board might encounter (humidity, dust etc), some of these connections will be reconsidered, in order to ensure the correct operation of this system for all possible conditions.

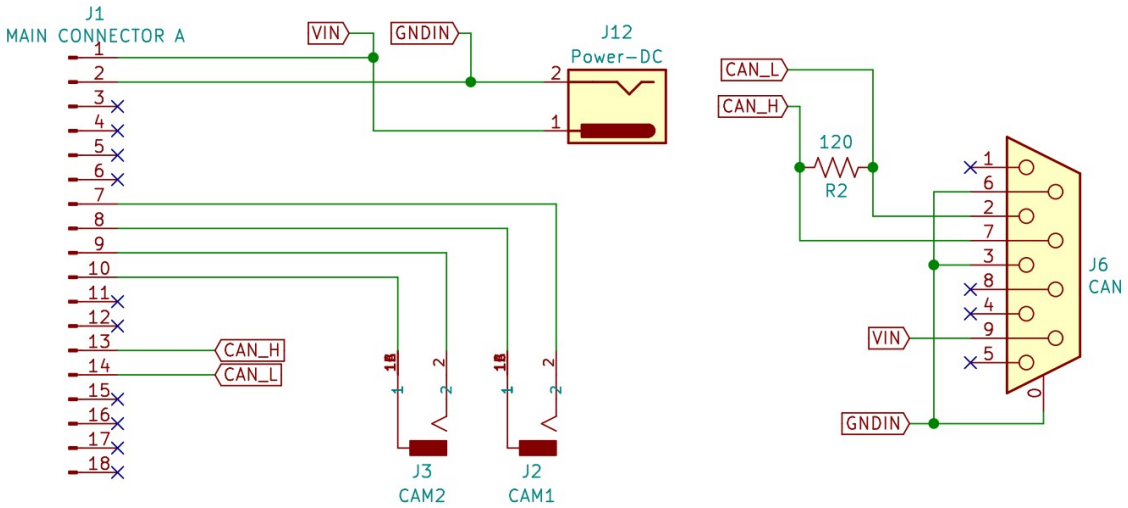


Figure 7.7: 3D70 Deployment board schematic

In the following Figures (7.8a and 7.8b), the connector types used in the developer kit for the 18-pin harness connection between display and board, as well as the CAN bus connector are illustrated.

(a) DEUTSCH DT16-18-SA-K004 18-Pin Harness

(b) TYCO 747844-6 CAN Connector

Figure 7.8: Display connectors used in the proposed board

The 18-pin connector is a Deutsch type connector manufactured by TE Connectivity. These types of connectors are environmentally sealed and designed for use in harsh conditions and built in a way that they can be disconnected and reconnected a multitude of times. They are ideal for numerous applications including automotive. The one Grayhill is using for its displays, has a 13 Amp current rating with an orientation of 3 rows of 6 pins each and provide IP6K9K protection. This protection standard ensures that dust does not enter the interior (6K-Dustproof) and protection against high-temperature, high-pressure water jet washing (9K-Waterproof). The CAN bus connection utilizes a four-wire system, that in our case occurs through a 9-pin D-Sub connector as we can see in Figure 7.8b, that offers a locking mechanism using screws to prevent disconnection, but does not offer high dust or water resistance.

7.5 Interconnection

For the final deployment, a few changes concerning the display deployment board are proposed. First, the board is directly powered by the DC/DC converter described in Section 7.1. The simple DC connector previously proposed is not suitable for outdoor use with conditions such as moisture. Additionally, vibrations during the vehicle's movement could cause it to disconnect and shut down the display. The proposed replacement, is the Molex 19429-0041 female connector (Figure 7.9a that can be directly mounted on the board enclosure. It is paired with the corresponding cable connecting it to the DC/DC converter 12Volt output and a Molex 19418-0008 male connector (Figure 7.9b. This industrial sealed connector pair offers IP67 protection and prevent disconnection with its locking mechanism.

(a) Display board female power connector
Molex 19429-0041

(b) Display board male power connector
Molex 19418-0008

Figure 7.9: Display board Molex power connectors

Due to the fact that DRIVE PX2 only outputs video through HDMI ports, HDMI to RCA composite video adapters will be used, as they have small form factor and efficiently convert HDMI's digital signal to analog required by the display. Thus, the two RCA connectors featured in our initial design and in the development board, are proposed in this design, to be used for the two camera feed inputs. Finally, 7Watts of power at 12Volts require 0.58 Amps, and considering a 20% variation, we added a 1Amp fuse after the power input connection to protect the components from possible electrical overloads caused by a faulty component in this board or the DC/DC converter. In Figures 7.10 and 7.11, the final version of the board schematic as well as the designed two-layer PCB are illustrated, both designed using the KiCad software suite.

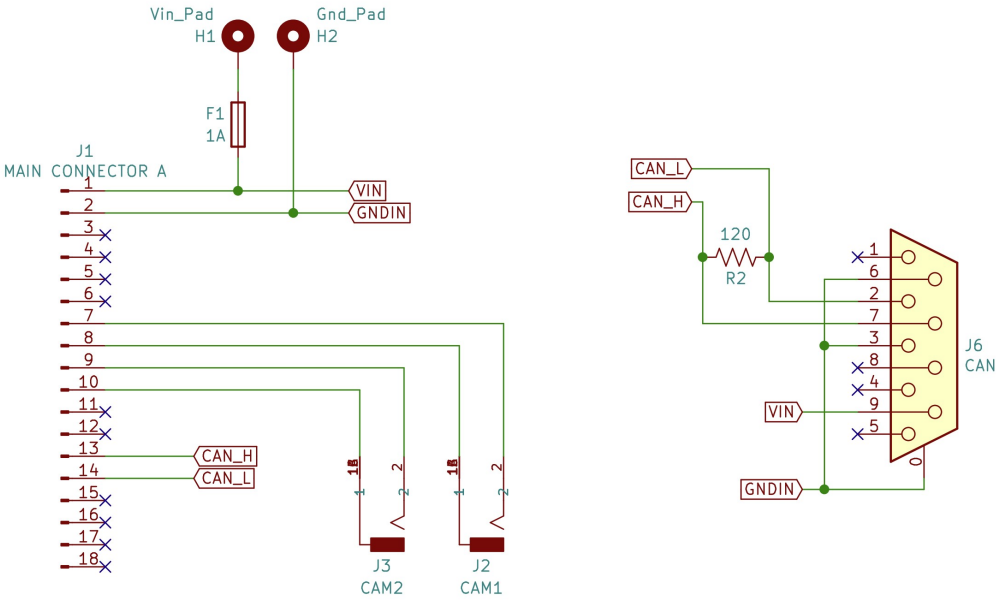
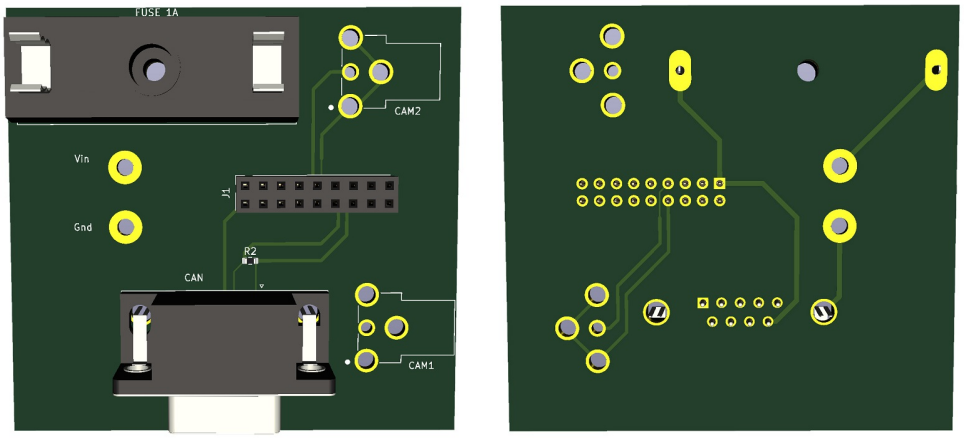


Figure 7.10: Final 3D70 Deployment board schematic

The socket in the top left part of the board in Figure 7.11a houses the safety fuse, and the two through-hole pads below are intended for the 12Volt and ground wires to be directly soldered. This removes the need for a separate connector for the power input. The 9-pin d-sub connector on the bottom side of the board is used for CAN bus link and the two placeholders on the right side labeled "CAM1" and "CAM2" are used for the two camera RCA socket inputs. They could alternatively be used to directly solder wires on them, in order to avoid using sockets. Finally, a 2×9 Pin grid is located in the middle of the board, intended for the display's harness to either connect to or directly soldered on.



(a) Deployment board Designed PCB front side

(b) Deployment board Designed PCB back side

Figure 7.11: Deployment board Designed PCB

The complete proposed pipeline concerning the component power and camera data transfer between devices, as well as how they are connected together, is illustrated in Figure 7.12. Both DRIVE PX2 and the 3D70 display are powered by the 12Volt output of the DC/DC converter, and both cameras are connected directly to the DRIVE system for real-time processing. Tegra A, the main sub-system of our platform, utilizes the forward facing camera's feed, passing it through object detection and localization algorithms, as well as other driver assistance modules, such as the Adaptive Cruise Control. Tegra B, has access to the rear facing camera, with the aim of providing the driver a view of the environment behind, whenever is needed. As the two Tegra sub-systems have the same processing power capabilities, the rear camera feed can also be utilized for early warning systems that could offer greater safety characteristics. The display outputs of both Tegra units, are directed to the display board through two HDMI to RCA converters, so both front and rear camera views can be available at the same time to the user. As the display's deployment board is specifically designed to our use case, camera feed, CAN bus and power is connected directly to it and the only connection between the board and the display is the 18-pin harness. The display GUI can be later redesigned or improved, without the need to revise the deployment board.

The proposed diagram shows how our systems communicate with each other and exchange data for the autonomous and rider assistance applications. Both Drive PX2 and 3D70 display have CAN bus capabilities that are not utilized in this project. In

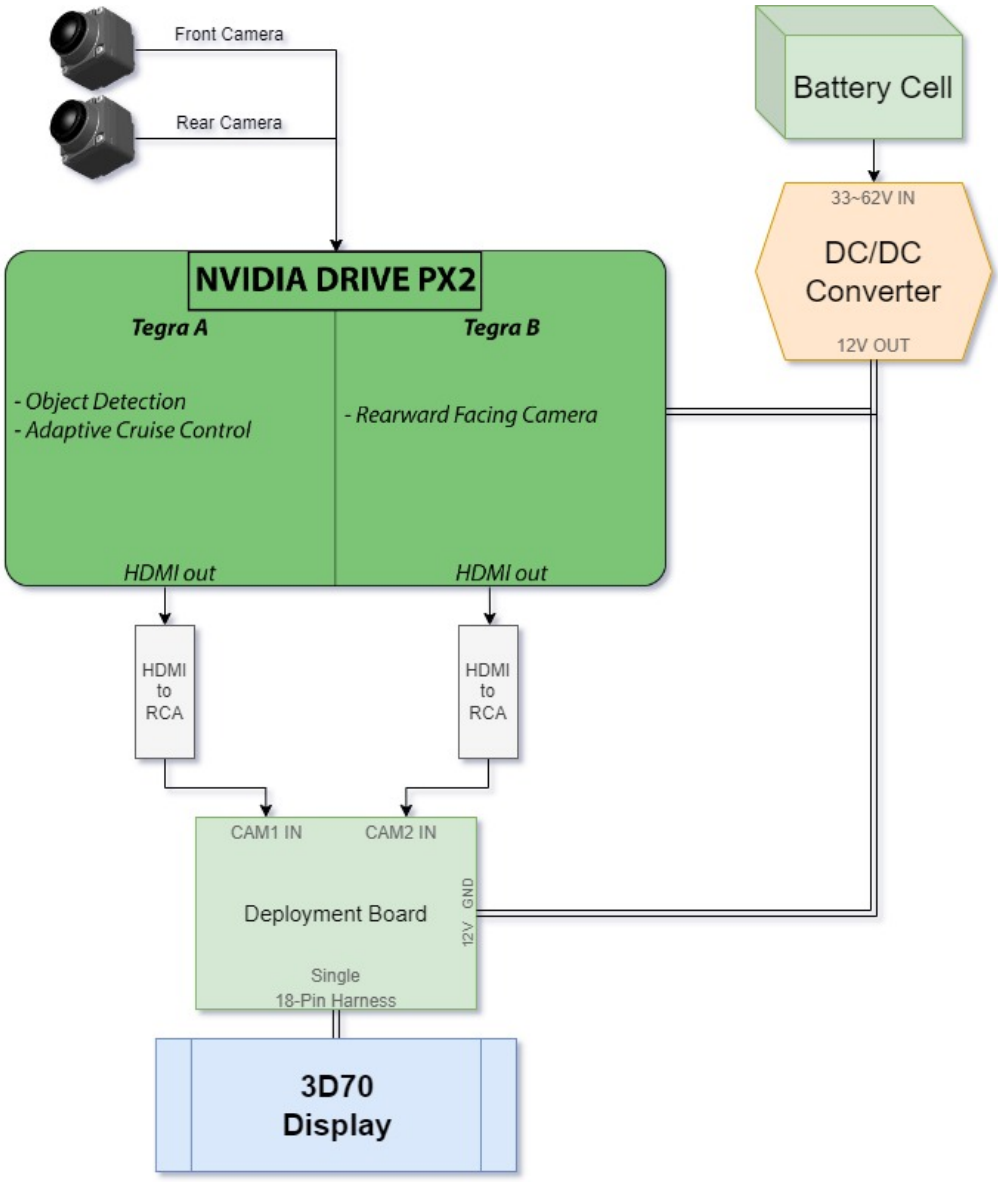


Figure 7.12: Proposed Pipeline of our system

future work, additional communication between the Drive platform and the display, using CAN bus protocol, is proposed, so the user can control the Drive's functionalities by enabling or disabling certain features. Additionally using the multiple CAN bus connectors Drive PX2 offers, it can draw information from sensors all around the motorcycle regarding battery level, DC/DC converter state, vehicle speed and many more.

Chapter 8

Conclusion and Future Work

This thesis describes the first step of implementing an autonomous ready electric motorcycle prototype. We discuss the simulation of the proposed systems such as object detection and adaptive cruise control in CARLA's virtual environment, as well as their adaptation on the Drive PX2 platform. The introduction of a PID controller for throttle control, will also act as a stepping stone upon which further development and fine tuning can take place using an accurate motorcycle model. Furthermore, the design and development of a Graphical User Interface of a digital dashboard intended for the 3D70 touchscreen display is analyzed. Finally, we proposed a deployment plan consisting of a DC/DC converter that powers Nvidia's Drive PX2 and the display, in addition to a PCB design that enables us to utilize the various inputs and outputs of the display.

As technology advances, more systems designed for similar applications are developed with higher processing power. The significantly smaller form factor of the newest DRIVE systems could make a big difference in our application considering the already low free space a motorcycle has. For example, Drive PX Xavier, a compact solution with nearly half the size of DRIVE PX2, and 50% greater performance, could be a beneficial upgrade in this case. Additionally, offering the same connectivity options, and a power consumption of 30Watts, it could make a compelling choice. Most importantly, and exactly as Nvidia intended, transitioning software from one Drive platform to another, requires insignificant changes to the source code.

In order to complete the autonomous motorcycle proposal, the ACC system must be simulated with a accurate motorcycle model in order to develop an ideal PID controller. This controller would directly regulate the motorcycle's speed through stepper motors for throttle and brake control via the high speed CAN bus interface, as proposed in Figure 8.1. As camera sensors are commonly used for semantic segmentation and object detection rather than distance calculations, the system's accuracy can be further increased by utilizing a second perception sensor such as a LiDAR or Radar, to continuously calculate distances between leading vehicles. An Adaptive Cruise Control system that utilizes LiDAR or Radar sensors can be further upgraded for lane detection using camera sensors. It can be used to help identify and track a vehicle in the same lane as the motorcycle or switch track to the appropriate vehicle after we perform a lane change. Additionally, by understanding and analyzing the behavior of two-wheeled vehicles [59], a steering system model

can be developed and integrated into the ACC system, that would ultimately take full control of the motorcycle. The CAN bus interface that offers the freedom to connect every electronic unit placed in the motorcycle can be convenient for this use case. Additionally, high priority data from the Electronic Control Unit (ECU), battery cell, bank angle sensors and any other safety equipment will be transferred through the common bus. It also allow us to share information between nodes such as the Drive PX2 and the display, or enable the ECU to actively communicate with Drive PX2.

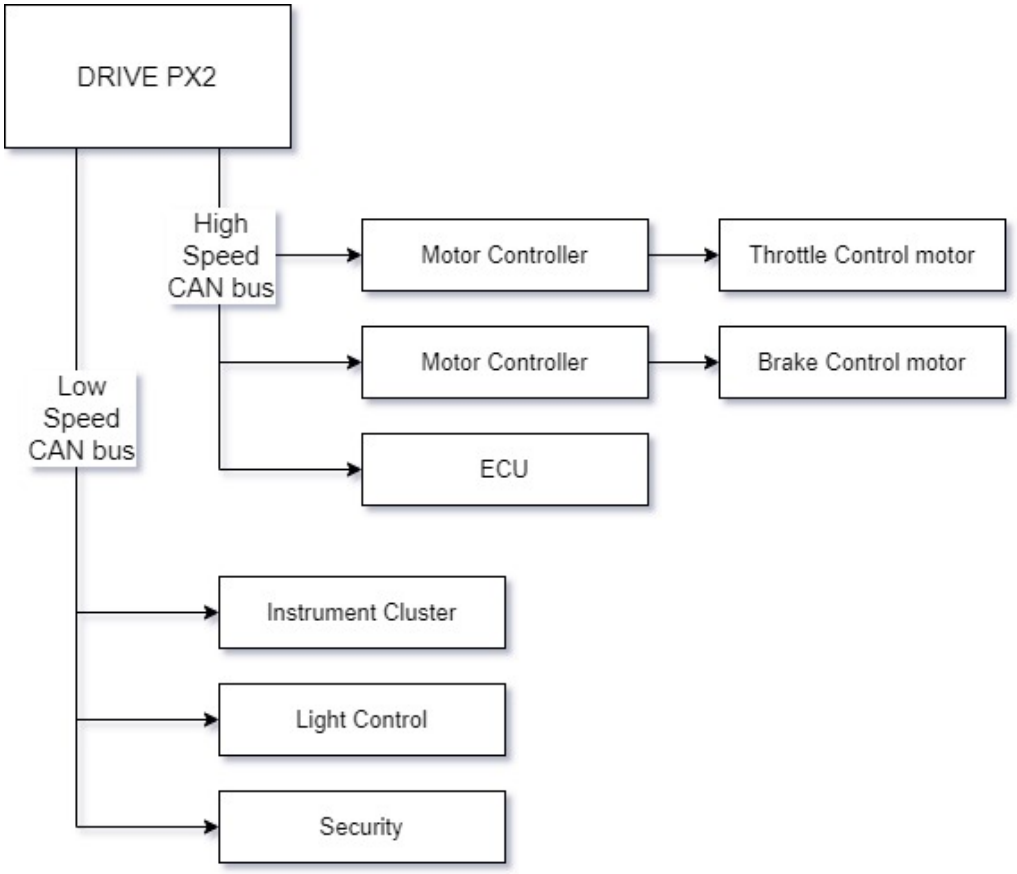


Figure 8.1: CAN bus wiring Proposal

Modern vehicles utilize camera feed in order to provide autonomous features, such as path finding and lane following in urban areas and highways. Deep neural networks can be trained and used in our project similar to [60], and could offer advanced vision system for navigation in crowded situations and urban areas. With the high processing power capabilities of Drive PX2 and newer platforms, machine learning and vision algorithms are easily deployable.

Bibliography

[1] Asif Iqbal Mohammad Faisal et al. "Understanding autonomous vehicles: A systematic literature review on capability, impact, planning and policy". In: *Journal of Transport and Land Use* 12 (Jan. 2019). DOI: 10.5198/jtlu.2019.1405.

[2] Society of Automotive Engineers. *Levels of Driving Automation (May 3, 2021).* URL: https://www.sae.org/blog/sae-j3016-update.

[3] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[4] Waymo. *Waymo Driver.* URL: https://waymo.com/waymo-driver/.

[5] Chris Clark. *Autonomous vehicles build on better sensor tech (August 12, 2021).* URL: https://www.embedded.com/autonomous-vehicles-build-on-better-sensor-tech/.

[6] Ivan Krešo and Sinisa Segvic. "Improving the Egomotion Estimation by Correcting the Calibration Bias". In: *VISAPP 2015 - 10th International Conference on Computer Vision Theory and Applications; VISIGRAPP, Proceedings* 3 (Jan. 2015), pp. 347–356. DOI: 10.5220/0005316103470356.

[7] Vectornav. *What is an intertial measurement unit?* URL: https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu.

[8] Thinklucid. *Understanding The Digital Image Sensor. Tech Brief (2020).* URL: https://thinklucid.com/tech-briefs/understanding-digital-image-sensors/.

[9] EdmundOptics. *Understanding Camera Sensors for Machine Vision Applications. Imaging Electronics 101.* URL: https://www.edmundoptics.eu/knowledge-center/application-notes/imaging/understanding-camera-sensors-for-machine-vision-applications.

[10] Realizator. *Stereo camera with CM3 inside for OpenCV learners drones and robotics. RaspberryPi Forum (June 2018).* URL: https://forums.raspberrypi.com/viewtopic.php?t=216940&sid=2fa548df6511298ce838dbdf067d5788.

[11] Vision team. *What is a stereo vision camera? Technology Deep Dive (October 2018).* URL: https://www.e-consystems.com/blog/camera/technology/what-is-a-stereo-vision-camera-2/.

[12] Stefan Norman and Liam Shelby-James. "Reliability and Trust in Global Navigation Satellite Systems". PhD thesis. University of Adelaide, Apr. 2020.

[13] Wan Rahiman and Zafariq Zainal. "An overview of development GPS navigation for autonomous car". In: *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*. 2013, pp. 1112–1118. DOI: 10.1109/ICIEA.2013.6566533.

[14] Masinde Muliro University. *Course: Itroduction To Radar Systems*. URL: http://mmust.elimu.net/BSC%28ELEC_COMM%29/Year_4/ECE%20451%20L_Radar_Eng_and_Facsimile/Introduction_to_Radar/Introduction_to_Radar.htm.

[15] Everything RF editorial team. *Automotive radar systems. Technical Articles (August 13 2019)*. URL: https://www.everythingrf.com/community/automotive-radar-basics.

[16] Wikipedia. *Yandex self-driving car*. URL: https://en.wikipedia.org/wiki/Yandex_self-driving_car.

[17] Yan Shiyu et al. "Distance–Intensity Image Strategy for Pulsed LiDAR Based on the Double-Scale Intensity-Weighted Centroid Algorithm". In: *Remote Sensing* 13 (Jan. 2021), p. 432. DOI: 10.3390/rs13030432.

[18] Intellias Automotive. *The Ultimate Sensor Battle: Lidar vs Radar. Medium Forum (August 9 2018)*. URL: https://medium.com/@intellias/the-ultimate-sensor-battle-lidar-vs-radar-2ee0fb9de5da.

[19] Tesla.com. *Future of Driving. Tesla Autopilot*. URL: https://www.tesla.com/autopilot.

[20] Wikipedia. *CAN bus*. URL: https://en.wikipedia.org/wiki/CAN_bus.

[21] DevCom Electronics Manufacturing. *CAN BUS Analysis. Automotive Systems (December 2, 2020)*. URL: https://www.devcom.cz/en/automotive-systems/can-bus-analysis/.

[22] Wikipedia. *LIN bus*. URL: https://en.wikipedia.org/wiki/Local_Interconnect_Network.

[23] Martin Falch. *LIN bus configuration. (April 2022)*. URL: https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics.

[24] Haichun Zhang et al. "CANsec: A Practical in-Vehicle Controller Area Network Security Evaluation Tool". In: *Sensors* 20.17 (2020). ISSN: 1424-8220. DOI: 10.3390/s20174900. URL: https://www.mdpi.com/1424-8220/20/17/4900.

[25] Nvidia. *Nvidia Drive Platform. NVIDIA DRIVE - Autonomous Vehicle Development Platforms*. URL: https://developer.nvidia.com/drive.

[26] Fred Lambert. *Drive PX2 on Tesla cars. Electrek News Articles (Oct 21 2016)*. URL: https://electrek.co/2016/10/21/all-new-teslas-are-equipped-with-nvidias-new-drive-px-2-ai-platform-for-self-driving/.

[27] NVIDIA. *Nvidia Drive SDK*. URL: https://developer.nvidia.com/drive/drive-sdk.

[28] Nvidia. *Nvidia DriveWorks*. URL: https://developer.nvidia.com/drive/driveworks.

[29] Nvidia. *Nvidia Drive AV*. URL: https://developer.nvidia.com/drive/drive-sdk#driveav.

[30] Nvidia. *Nvidia Drive IX*. URL: https://developer.nvidia.com/drive/drive-ix.

[31] NVIDIA. *Nvidia Drive Hyperion*. URL: https://developer.nvidia.com/drive/drive-hyperion.

[32] Wikipedia. *Yamaha Motobot*. URL: https://commons.wikimedia.org/wiki/File:Yamaha_Motobot_Ver.2_Tokyo_Motor_Show_2017_PB031807.jpg.

[33] Satish Jeyachandran. *Waymo EV sensor Configuration. Waymo Blog (March 4, 2020)*. URL: https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html.

[34] Pei Sun et al. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[35] SEKONIX Image Solution Team. *Sekonix SF332X-10X product family*. URL: http://sekolab.com/wp-content/uploads/2020/02/SF332X-10X_2Mega-LVDS-Automotive-Camera-Datasheet_Ver-2.2.5_190726.pdf.

[36] GrayHill. *3D70 Display, GrayHill Documentation*. URL: https://www.mouser.de/datasheet/2/626/3D70-datasheet-1138565.pdf.

[37] GrayHill. *3D70 Quickstart Guide. GreyHill Documentation*. URL: https://www.grayhill.com/documents/3D70-Quick-Start-Guide.

[38] Learning About Electronics. *How to Detect Cars in a Video in Python using OpenCV. Programming Articles*. URL: http://www.learningaboutelectronics.com/Articles/How-to-detect-cars-in-a-video-Python-OpenCV.php.

[39] Sander Soo. "Object detection using Haar-cascade Classifier". In: *Institute of Computer Science, University of Tartu.* (2014), pp. 1–12.

[40] Li Cuimei et al. "Human face detection algorithm via Haar cascade classifier combined with three additional classifiers". In: *2017 13th IEEE International Conference on Electronic Measurement and Instruments (ICEMI)*. 2017, pp. 483–487. DOI: 10.1109/ICEMI.2017.8265863.

[41] Mooseop Kim, Deok Gyu Lee, and Ki-Young Kim. "System Architecture for Real-Time Face Detection on Analog Video Camera". In: *International Journal of Distributed Sensor Networks* 2015 (May 2015), pp. 1–11. DOI: 10.1155/2015/251386.

[42] Hongkun Yu et al. *TensorFlow Model Garden*. https://github.com/tensorflow/models. 2020.

[43] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed3804Paper.pdf.

[44] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[45] Samuele Salti et al. “A traffic sign detection pipeline based on interest region extraction”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)* (2013), pp. 1–7.

[46] Vaibhavi Golgire. “Traffic Sign Recognition using Machine Learning: A Review”. In: *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT)*. Vol. 10. 2021.

[47] Degui Xiao and Liang Liu. “Super-Resolution-Based Traffic Prohibitory Sign Recognition”. In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2019, pp. 2383–2388. DOI: 10.1109/HPCC/SmartCity/DSS.2019.00332.

[48] Puneet Piyush Malhotra and Tanishq Chamola. *Traffic sign recognition using deep neural networks*. URL: https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70.

[49] A. de la Escalera, J.M Armingol, and M. Mata. “Traffic sign recognition and analysis for intelligent vehicles”. In: *Image and Vision Computing* 21.3 (2003), pp. 247–258. ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(02)00156-7. URL: https://www.sciencedirect.com/science/article/pii/S0262885602001567.

[50] Alexey Dosovitskiy et al. *CARLA: An Open Urban Driving Simulator. Proceedings of the 1st Annual Conference on Robot Learning*. 2017. arXiv: 1711.03938 [cs.LG].

[51] Daniel Dworak et al. “Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator”. In: *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2019, pp. 600–605. DOI: 10.1109/MMAR.2019.8864642.

[52] Jean-Emmanuel Deschaud. “KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator”. In: *CoRR* abs/2109.00892 (2021). arXiv: 2109.00892. URL: https://arxiv.org/abs/2109.00892.

[53] Python. *Python logging*. URL: https://docs.python.org/3/library/logging.html.

[54] Anil Yadav and Prerna Gaur. “Robust adaptive speed control of uncertain hybrid electric vehicle using electronic throttle control with varying road grade”. In: *Nonlinear Dynamics* 76 (Apr. 2013), pp. 305–321. DOI: 10.1007/s11071-013-1128-9.

[55] m-lundberg. *simple-PID (Oct 8, 2022)*. URL: https://github.com/m-lundberg/simple-pid.

[56] MEAN WELL. *RSD-300C-12 DC/DC Converter Datasheet (September 20, 2022)*. URL: https://www.meanwell-web.com/content/files/pdfs/productPdfs/MW/RSD-300/RSD-300-spec.pdf.

[57] Molex. *Molex 39-01-2100 Datasheet (October 11, 2021)*. URL: https://gr.mouser.com/datasheet/2/276/3/0039012100_CRIMP_HOUSINGS-2841684.pdf.

[58] Littelfuse. *TVS Diode TP6KE33A Datasheet (October 6, 2021)*. URL: https://gr.mouser.com/datasheet/2/240/Littelfuse_TVS_Diode_TP6KE_Datasheet_pdf-587274.pdf.

[59] Mohamed El Amine Khettat et al. "Autonomous motorcycles: Towards behavioral modeling of steering systems". In: *2012 International Conference on Multimedia Computing and Systems*. 2012, pp. 1103–1108. DOI: 10.1109/ICMCS.2012.6320211.

[60] Mohammad Ali Mohammadkhani, Babak Majidi, and Mohammad Taghi Manzuri. "Deep Vision for Navigation of Autonomous Motorcycle in Urban and Semi-Urban Environments". In: *2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*. 2019, pp. 1–5. DOI: 10.1109/ICSPIS48872.2019.9066130.