Design & implementation of a surround vision system for cooperative and autonomous driving

Citation for published version (APA):
Menezes, A. (2020). *Design & implementation of a surround vision system for cooperative and autonomous driving: Object-Lane-Freespace Detection & Pose Estimation of Targets*. Technische Universiteit Eindhoven.

Document status and date:
Published: 23/10/2020

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

TU/E WHERE INNOVATION STARTS TU/E

PDEng THESIS REPORT

Design and Implementation of a Surround Vision System for Cooperative and Autonomous Driving

Object-Lane-Free space Detection and Pose Estimation of Targets

Ashton Menezes
22 September 2020
Department of Mathematics & Computer Science

PDEng AUTOMOTIVE SYSTEMS DESIGN
Track AUTOMOTIVE SYSTEMS DESIGN

TU/e EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

Design & Implementation of a Surround Vision System for Cooperative and Autonomous Driving

Object-Lane-Free space Detection & Pose Estimation of Targets

Ashton Menezes

October 2020

Eindhoven University of Technology
Stan Ackermans Institute - Automotive/Mechatronic Systems Design

PDEng Report: 2020/058

Confidentiality status : Open access

Partners

Integrated Cooperative Automated Vehicles

Eindhoven University of Technology

Steering Group dr.ir. Tom van der Sande (I-Cave & TU/e)
dr.ir. Gijs Dubbelman (TU/e)

Date October 2020

Composition of the Thesis Evaluation Committee:

Chair: Dr. Gijs Dubbelman

Members: Dr. Tom van der Sande

Dr. Jos Elfring

Dr. René van de Molengraft

Dr. Peter S.C Heuberger

Dr. Narsimlu Kemsaram

The design that is described in this report has been carried out in accordance
with the rules of the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.072, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31 402743908

Partnership
This project was supported by Eindhoven University of Technology and I-Cave.

Published by
Eindhoven University of Technology
Stan Ackermans Institute

PDEng-report
2020/058

Preferred reference
Design & Implementation of a Surround Vision System for Cooperative & Autonomous Driving: Object-Lane-Free space Detection & Pose Estimation of Targets, Eindhoven University of Technology, PDEng Report 2020/058, October 2020

Abstract

Self-driving cars have the potential to change the landscape of urban mobility. However, the biggest roadblock for the success of such concepts is to perceive the environment robustly in real-time. This project provides a prototype solution to robustly estimate the surroundings around the vehicle using a set of monocular cameras.

Keywords
Surround Vision, Autonomous Driving, Platooning, Vehicle Pose Estimation

Disclaimer Endorsement
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or i-CAVE. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or i-CAVE, and shall not be used for advertising or product endorsement purposes.

Disclaimer Liability
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright © 2020. Eindhoven University of Technology. All rights reserved.
No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and i-CAVE.

Foreword

Ashton has been working in the i-CAVE project, which is a large NWO project, focused on automated and cooperative vehicles. One of the key challenges in achieving successful vehicle automation is environmental perception. In practice this means that a vehicle is equipped with various sensors such as RADAR, camera and LIDAR. In addition to fusion of these sensors to create a coherent image and navigate safely, cooperative driving requires contextual knowledge of the vehicles around it. The most important objective in cooperative driving is following the car in front of you, which in this case is a Renault Twizy. The main objective of Ashton's research was to develop a system that is able to identify this particular car. In addition to that, having knowledge of the location and orientation of the Twizy is of the utmost importance to achieve successful cooperative automated driving.

To that end, Ashton has created a framework for a set of cameras that he implemented on the research vehicle. He managed to create a detector that estimates the relative position of the Twizy and that also gives the heading angle with respect to that. Additionally, multiple modules, such as lane detection and free-space detection have also been implemented.

With the work of Ashton, another step has been set to achieve fully automated and cooperative driving vehicles. Together with the work of fellow PDEng researcher Varun Khattar, the environmental perception of the Renault Twizy has taken another step in the right direction.

Dr. Tom van der Sande
September 2020

Preface

This technical report describes my final assignment for the Professional Doctorate in Engineering (PDEng) program in Automotive Systems Design (ASD) at the Eindhoven University of Technology (TU/e). During this program the trainees work on several multidisciplinary projects with different automotive companies, where state-of-the-art system's engineering approach is followed. The program is divided into two phases, each having a one-year duration. During the first phase, the focus is on professional and personal development where the trainees work in teams by taking up different leadership roles while working with TU/e industrial partners. The second phase of the program consist of a twelve-month design experiment carried out at a company or the university.

In accordance with this, I carried out my final design assignment with the integrated cooperative automated vehicles (i-CAVE) project team at TU/e with the goal to design and implement a Surround Vision System (SVS) using the Nvidia Drive PX 2. The main task of the system is to provide coherent information of the environment around the ego-vehicle to enhance the sensor fusion and decision-making capabilities. The intent of this report is to enable a technical reader (engineers and researchers) to either understand, replicate or continue this project.

Ashton Menezes
September, 2020

Acknowledgements

I want to thank all the people who have supported me, especially during these nine-months and also during the entire program duration.

First of all, I would like to thank my supervisors, Gijs Dubbelman and Tom van der Sande, for giving me the opportunity to work on this project. Without their mentoring and invaluable support, it would have been impossible to tackle many technical and execution related issues.

Secondly, I would like to thank Narsimlu Kemsaram and Anweshan Das for their constant availability and support when it came to hardware execution on the Drive PX 2, camera calibration, validation and testing. They have also played a critical role in getting me up and running with all the necessary software setups during the initial phase of the project.

Furthermore, I would like to thank Peter Heuberger and Ellen van Hoof-Rompen, for guiding the project from an organizational point-of-view.

Finally, I would like to thank my fellow colleagues, Arash Arjmandi, for the extensive discussions we had concerning computer vision and software architecture and Salih Yousif for his valuable tips during stressful times.

Ashton Menezes
September, 2020

Executive Summary

Self-driving cars have the potential to change the landscape of urban mobility. However, the biggest roadblock for the success of such concepts is to robustly perceive the environment in real-time. Stereo Vision Camera sensors are widely used for extracting semantic information and estimating depth. However, they have a limited field of view, a high processing time and are economically not scalable.

The goal of this design experiment is to take a step forward in vehicle automation by investigating the possibilities of using monocular cameras to obtain a coherent image of the surrounding. To this purpose, a prototype of a Surround Vision System is designed and deployed on automotive-grade embedded platform, the NVidia Drive PX 2, to evaluate its real-time capabilities. The system is then demonstrated on a test vehicle, a Renault Twizy, for autonomous and cooperative applications.

The project will try to accomplish this goal by breaking it into three main parts: system architecture, design and implementation, and validation.

The first objective is to identify the problems in the existing stereo vision system, which is used for fusion with Radar measurements and address the key concerns of the stakeholders (Chapter 2). Using a system's engineering approach, different driving scenarios were analysed to determine the expected functionality and derive requirements. The Software Architecture was then derived by decomposing the system into functional modules and identifying the interfaces (Chapter 3 and 4).

The design and implementation (Chapter 5 and 6) involved evaluation of different design choices considering the concerns of stakeholders, project management constraints and the compatibility limitations with the NVidia Drive PX 2. The main challenge here was to robustly and efficiently detect the position of objects and the pose of potential platooning target vehicles using monocular vision.

To test and demonstrate the functionality in real-time, the designed system was deployed on a test vehicle, a Renault Twizy using the NVidia Drive PX 2. The system is capable of receiving information synchronously from multiple cameras mounted on the roof of the test vehicle. The system robustly detects and identifies the position of objects, pose of targets, the drivable free space and lanes around the vehicle. Each functionality is evaluated through visual verification, offline measurements and external sensors described in Chapter 7. The computed information is then provided to the vehicle control system at a desirable rate suitable for sensor fusion.

Table of Contents

Foreword i

Preface iii

Acknowledgements v

Executive Summary vii

Table of Contents ix

List of Figures xii

List of Tables xv

1. Introduction 1

1.1 Motivation *1*

1.2 Need for Surround Vision System *2*

1.3 Project Context *4*

1.4 Project Goals *4*

1.5 Interaction with other Projects *5*

1.6 Unique Contributions *5*

1.7 Overview of the Thesis *5*

2. Analysis 7

2.1 Problem Analysis *7*
2.1.1. Project Background 7
2.1.2. Problem Statement 8

2.2 Stakeholder Analysis *8*
2.2.1. I-Cave Program Partners (Eindhoven University of Technology) 8
2.2.2. Designers and Developers 9
2.2.3. PDEng Management (ASD Program) 9

2.3 CAFCR Analysis *9*

3. Requirements Elicitation 11

3.1 Elicitation process *11*

3.2 Real World Layer *11*

3.3 Vehicle Level *13*
3.3.1. System Context: 13
3.3.2. System Functions 13
3.3.3. Functional Requirements 15
3.3.4. Non-Functional Requirements 18

3.4 System External Interfaces *19*

4. System Architecture 21

4.1 Functional Viewpoint 21

4.2 Concurrency Viewpoint 22

4.3 Logical Viewpoint 24

4.4 Structural Views 25

5. Module Design 29

5.1 Camera Image Acquisition (CIA) 29

5.2 Object Perception 30

5.2.1. Object Detector 32
5.2.2. Vehicle Tracker 33
5.2.3. Tracked Object Manager 34

5.3 Lane Perception 39

5.4 Free Space Perception 40

6. Implementation 41

6.1 NVidia Drive Software 41

6.2 Mapping of SVS Architecture to DriveWorks APIs 41

6.2.1. Camera Image Acquisition 41
6.2.2 Deep Neural Networks 42
6.2.2.1 Object Detector 42
6.2.2.2 Object/Target Vehicle Tracker 42
6.2.2.3 Target Vehicle Classifier 42
6.2.3 Vehicle Position and Other Objects Estimation 43
6.2.4 Target Vehicle Pose Estimation 45
6.2.5 Lane Detection & Polynomial Fitting 46
6.2.6 Free Space Detection 46

7. Validation and Test Results 47

7.1 Experimental Setup 47

7.1.1. Carla Simulator 47
7.1.2. Test Vehicle 47

7.2 Unit Testing 48

7.2.1. Unit Testing Rendering Results: 49
7.2.2. Position Estimation Static Results 50
7.2.3. Position Estimation Dynamic Results 51
7.2.4. Target Pose (Heading) Estimation Results 53
7.2.5. Lane Polynomial Fitting Results 55

7.3 System-Level Testing 55

7.3.1. System Profiling: 56
7.3.2. Comparison with Stereo Vision System. 57

8. Project Management 59

8.1 Project Planning 59

8.2 Risk management 59

8.3 Project task execution 60

9. Conclusion and Recommendations 63

9.1 Conclusions *63*
9.2 Future Work and Recommendations *64*

Glossary 65

Bibliography 67

References *67*

Appendix A. 69

Appendix B. 74

Appendix C. 75

Appendix D. 75

About the Author 77

List of Figures

Figure 1: Platooning scenario. 1
Figure 2: a) i-CAVE work packages b) A Renault Twizy fitted with Sensors 2
Figure 3: Some highway scenarios of autonomous vehicle crashes 2
Figure 4: Sensors comparison in autonomous driving [3] 3
Figure 5: Proposed Surround Vision System 4
Figure 6: Existing Setup 7
Figure 7: CAFCR a Multi-view method for System Architecting [4] 9
Figure 8: CAFCR Framework Application 10
Figure 9: Real-World Layer: Context of the Ego-vehicle 11
Figure 10: Vehicle Level Layer: Context of the Surround Vision System 13
Figure 11: System Functions derived from the driving scenario 14
Figure 12: System External Interfaces 19
Figure 13: Functional Viewpoint - Functional Elements & their interactions 21
Figure 14: Functional Viewpoint - Each Functional Element & its decomposition 22
Figure 15: Concurrency View- SVS 23
Figure 16: Concurrency View - Object Detection 23
Figure 17: State Machine Diagram 24
Figure 18: Block Definition Diagram (BDD) representing the composition of SVS 25
Figure 19: Internal Block Diagram (IBD) of Surround Vision System. 26
Figure 20: Internal Block Diagram (IBD) of Object Perception Module 26
Figure 21: Camera Image Acquisition Flow-chart 29
Figure 22: Object Perception: Design Choice 1 30
Figure 23: Object Perception - Design Choice 2 31
Figure 24: Object Perception - Design Choice 3 31
Figure 25: Object Perception for Front Camera 32
Figure 26: Object Detector Flow Chart 33
Figure 27: Vehicle Tracking Algorithm 33
Figure 28: Vehicle Tracker (Front) Workflow 34
Figure 29: Tracked Object manager workflow 35
Figure 30: Width Based Methods: a) Intervehicle distance calculation based on bounding box height change b) Intervehicle distance calculation based on lane-markings 36
Figure 31: Position-Based Method 36
Figure 32: Position Estimator Workflow 37
Figure 33: Target Vehicle Classifier Workflow 38
Figure 34: Target Pose Estimator Workflow 38
Figure 35: Lane Perception Workflow 39
Figure 36: Free Space Perception Workflow 40
Figure 37:NVidia Drive Software Stack [10] 41

Figure 38: Image Processing Pipeline 42
Figure 39: Pipeline for Training and Deployment of Classifier 42
Figure 40: Position Estimation Schematic Diagram Front View... 43
Figure 41:Top-View Schematic for Heading and Orientation Measurements.. 45
Figure 42: 3D Box mesh manually aligned with the Twizy rear surface and feature points extracted................. 46
Figure 43: CARLA Simulator setup for unit testing of SVS functions 47
Figure 44: Ego-vehicle fitted with GSML cameras and connected to the DRIVE PX 2 47
Figure 45: Unit Testing of Object Perception Module 49
Figure 46: Unit Testing of Lane and Free space module.. 50
Figure 47: Static Test Setup with cones placed at steps of 10m. 50
Figure 48: Longitudinal position results for vehicle departing away from the ego-vehicle 51
Figure 49: Vehicle Height measurements & estimates of the detected preceding vehicle 52
Figure 50: Position estimation results for vehicle approaching the ego vehicle 52
Figure 51: Vehicle Height measurements and estimates of preceding vehicle 53
Figure 52: Test Setup for Target Pose Estimation 54
Figure 53: Correlation of heading angle measurements by camera with IMU measurements............................. 54
Figure 54: Lane Polynomial Fitted with measured data 55
Figure 55: On-Road testing with target and ego vehicle 56
Figure 56: Execution time for Front Camera when four new tracks are created 56
Figure 57: Execution time for Front Camera with existing tracks.. 57
Figure 58: Initial Project plan constructed at the beginning of the project. 59
Figure 59: Actual Executed Plan 60

List of Tables

Table 1: Driving Scenario - CACC 12
Table 2: Unit Testing Observations 48
Table 3: Position Estimation static testing results 50
Table 4: Comparison of the execution time of SVS and Stereo Vision systems on Host PC 57
Table 5: Risk Management Table 60

1.Introduction

This introduction chapter briefly describes the motivation, project context, the main stakeholders involved in the project, the high-level goals, and interaction with other projects. The purpose of designing a surround vision system is also discussed. A set of high-level goals for the project are also formulated, which will be explained in detail in the analysis of Chapter 2.

1.1 Motivation

In the ’50s and ’60s visions came up reaching from automated road traffic to “Flying Cars”. Some of these visions were inspired by the progress made in electronics and computer technology. Not all of these dreams have come true, but yes, we have indeed made tremendous progress in relation to self-driving cars, thanks to the explosion in Artificial Intelligence, especially when it comes to perception systems. The benefits are manifold from safety to convenience; self-driving vehicles have the potential to completely transform the transport infrastructure, which could lead to mobility as well as economic and sustainability gains.

Statistics from the National Highway Traffic Safety Administration (NHTSA) show that driver error is by far the most significant cause of road traffic accidents, due to factors like miscalculations, errors of judgment, speeding, drink-driving, and phone use. In fact, an alarming 94 percent of serious crashes are due to human error. [1]

Concerning the economic benefits, an NHTSA study proves that motor vehicle crashes in 2010 accounted for $242 billion in economic activity, including $57.6 billion in lost workplace productivity, and $594 billion due to loss of life and decreased quality of life due to injuries [1].

Fully autonomous vehicles would take human error out of the equation, thereby making our roads safer not just for drivers, but also passengers, cyclists, and pedestrians. Fully autonomous vehicles do not depend on communication with other traffic but rely on multiple onboard sensors to move and navigate independently. However, sensor failure or any other technical error may lead to disastrous consequences. Cooperative automated vehicles, on the other hand, can share system information with other vehicles making a significant contribution towards increasing road safety and improve mobility worldwide. Constructive information sharing will provide endless possibilities for safe driving, and multiple vehicles can collaborate to compensate for information scarcity. An example of such cooperative behaviour is platooning, where multiple vehicles drive together in formation. In a platooning scenario, the front target vehicle is controlled by the driver, while the ego vehicles autonomously follow the vehicle in front of it, is as shown in Figure 1.

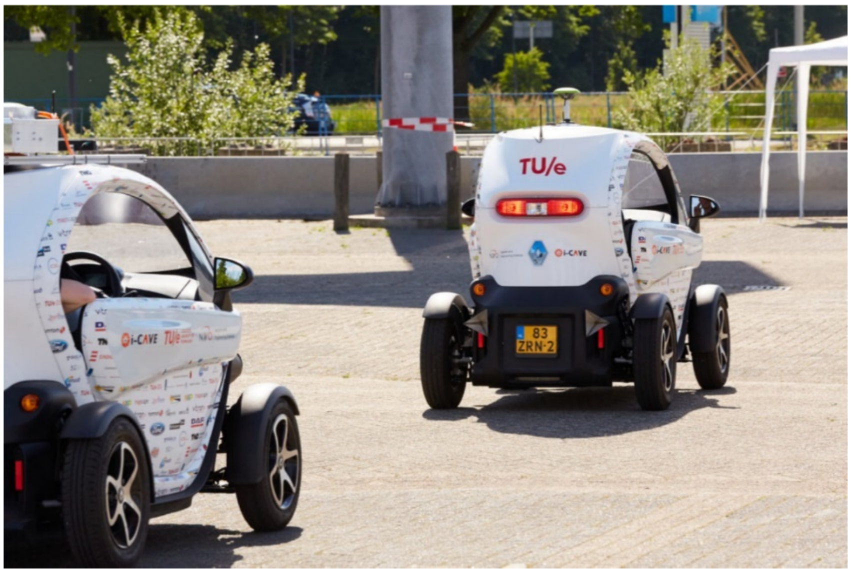


Figure 1: Platooning scenario.

In a platooning, the ego vehicle needs reliable and accurate perception capabilities to follow the target vehicle autonomously. Various perception algorithms are rapidly evolving, but the practical bottleneck is the amount of computational power available in these vehicles. Embedded platforms normally deployed in a car are simply not able to cope with the massive amount of data generated by different sensors such as cameras. NVidia Drive-PX 2, Intel-GO and NXP BlueBox are examples of such programmable platforms.

Inspired by the benefits and challenges mentioned above the i-Cave (integrated cooperative automated vehicles) Research Program [2] funded by the Netherlands Organization for Scientific Research (NWO) was started in the Netherlands to investigate the throughput and safety in relation to automated and cooperative driving with an ultimate goal towards SAE Level 5 Automation. This program is broken down into several work packages based on different disciplines, sensing, cooperative vehicle control, communication & human factors, to name a few as shown in Figure 2.a below.

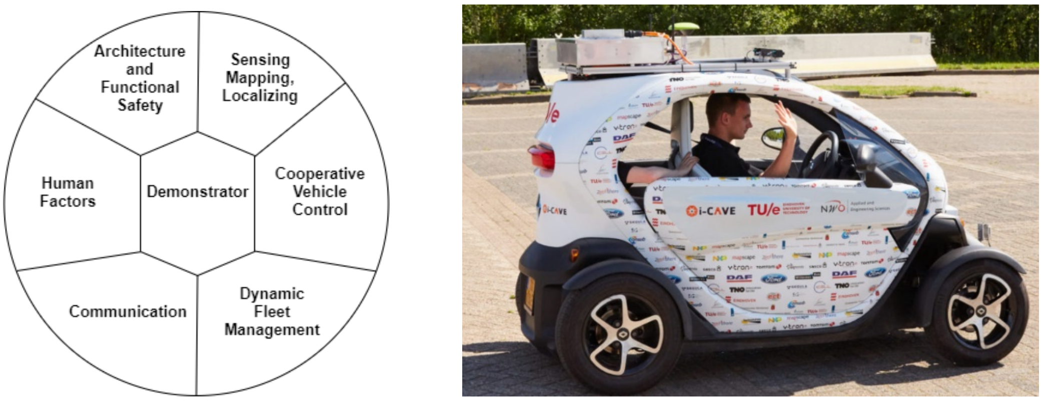

Architecture and Functional Safety
Sensing Mapping, Localizing
Human Factors
Demonstrator
Cooperative Vehicle Control
Communication
Dynamic Fleet Management
i-CAVE
TU/e

Figure 2: a) i-CAVE work packages b) A Renault Twizy fitted with Sensors

The primary motivation of this project is to make a step forward in vehicle automation by improving the sensing capabilities of the vehicle and evaluating the performance on a prototype vehicle. To that effect, demonstrator vehicles which are Renault Twizys (see Figure 2b) are equipped with various sensors, communication devices and controllers. This assignment mainly focused on the design and realization of the surround vision system using the NVidia Drive PX 2 embedded platform.

1.2 Need for Surround Vision System

An autonomous vehicle needs to accurately identify and localize dynamics objects in the surrounding environment of the vehicle. Having a full scan of the surrounding is crucial for many tasks, such as obstacle avoidance, path planning, and intent recognition. Figure 3, for example, shows some highways scenarios where a car accident is likely to occur due to the limited detection field of view of the vehicle. In these cases, the perception system would not trigger an alert since there is no vehicle in the system's field of view. If this happens in a highway situation, it is too late for the autonomous system to react due to high speeds. Therefore, comprehensive environment perception that detects, classifies and estimates the motion states of surrounding vehicles would have an enormous potential to enhance road safety by proactively reacting to traffic conditions or hazardous situations.

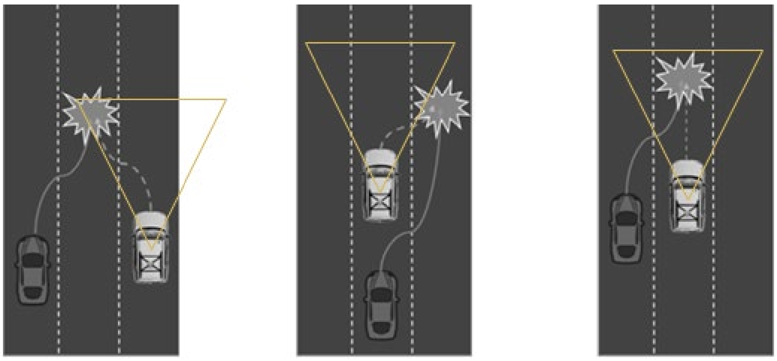
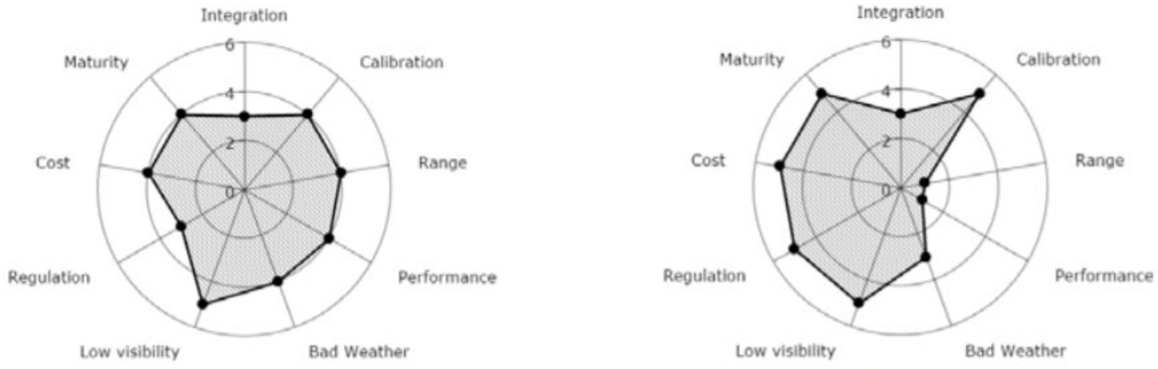


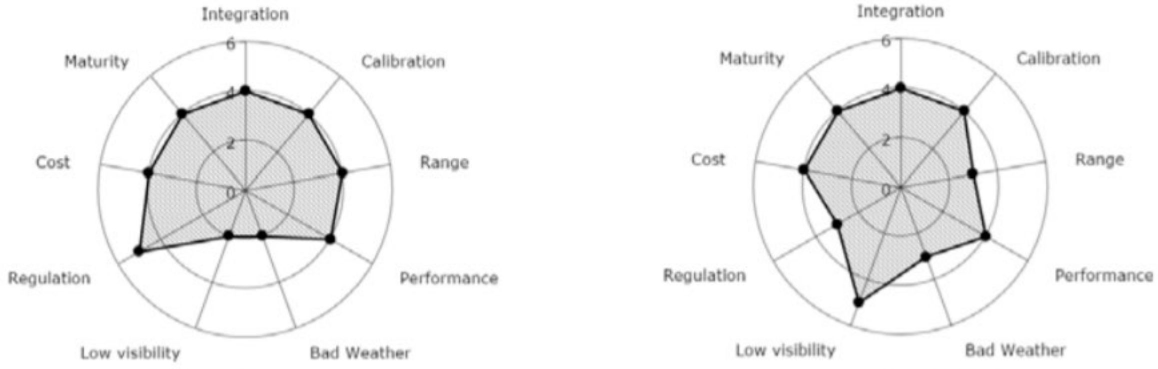
Figure 3: Some highway scenarios of autonomous vehicle crashes

In relation to the sensing capabilities, different types of sensors are used to provide external and immediate information of the vehicles surrounding. (see Figure 4)

- Radar – Small, inexpensive and good range, work in dark conditions and able to sense.
- Ultrasonic - Suitable only for near range 3D mapping.
- Lidar – Have excellent range and mapping capabilities but are too expensive for broad deployment.
- Vision-based sensors, i.e. Cameras, have become affordable, miniaturized, and with increasing resolution in recent years. Their colour, contrast, and optical character recognition capabilities give these sensors an edge compared to other sensors.



(A) Radar (B) Ultrasonic



(C) Camera (D) LiDAR

Figure 4: Sensors comparison in autonomous driving [3]

Stereo vision sensors are also used for tracking objects. However, the performance is limited by the quality of disparity estimates, longer execution time and the field of view (FoV) of the stereo pair. Furthermore, it is not economically viable to have multiple stereo cameras to cover the surrounding of the vehicle. Thus in this project, we investigate the possibility of a surround vision system using a set of monocular cameras to detect and estimate the position of objects, pose of potential platooning targets, lanes and the free space around the ego-vehicle as shown in Figure 5.

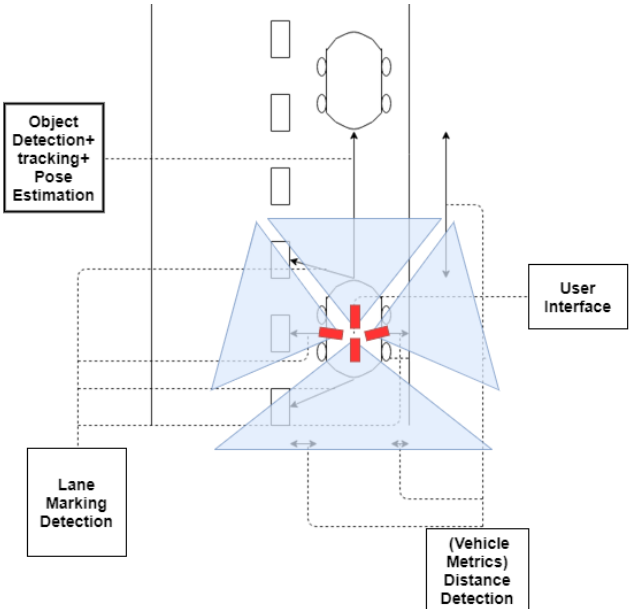

Object Detection+ tracking+ Pose Estimation
User Interface
Lane Marking Detection
(Vehicle Metrics) Distance Detection

Figure 5: Proposed Surround Vision System

1.3 Project Context

This project was performed in collaboration with the Mobile Perception Systems (MPS) group and the Dynamics & Control (D&C) group of the TU/e. The MPS group specializes in building sensor perception and prediction algorithms. The end goal of this department is more involved in improving the perception capabilities of the vehicle. They focus on methods that can sense the ever-changing environment around the car and perceive objects/obstacles around the vehicle.

The D&C group, on the other hand, is responsible for research into various control algorithms required for cooperative and self-driving applications. Their goal is to make sure that the vehicle can plan a path and make decisions in order to drive safely in the environment.

In relation to the i-CAVE program, the MPS group is responsible for the development of perception algorithms and D&C group is responsible for the development of controllers for vehicle platooning. These algorithms and controllers are then tested by deploying them in a demonstrator vehicle.

1.4 Project Goals

The primary customer of this project is the D&C group since all the information captured by the perception sensors will be utilized for sensor fusion and path planning. The customer wants to drive cooperatively in a platoon by tracking a target vehicle and drive autonomously in general driving conditions. The MPS team will play a role of a TIER1 supplier, responsible for providing all necessary software, hardware and technical support required for implementing this project. The primary purpose of this project is to design the surround vision software involved in the system, deploy it on an embedded device and demonstrate the functionality on a demonstrator vehicle.

The main project goals are as follows:

- Capture and analyze requirements from discussions with relevant stakeholders.
- Design the system so that it fulfils the requirements and accounts for design constraints caused by software APIs and embedded devices.
- Implement the system functionality using DriveWorks API's since they are optimized for implementation on Nvidia Drive PX 2.
- Verify and validate the final implemented system using the appropriate techniques.

Implementation in a prototype vehicle requires the installation of additional hardware, such as:

- GMSL Cameras - Mounted and calibrated in the desired positions.
- NVidia Drive PX 2 - An embedded device that realizes the designed functionality.
- HMI – Displaying the outputs of implemented functionality for testing and verification.

On the software side, the practical realization of the system requires:

- Implementation of an Object Detector to detect objects belonging to different class types.
- Training of a neural network that can detect and classify a Target Vehicle (Twizy)

- Setting up a software pipelines from capturing information from cameras to performing detections.
- Designing of a target vehicle pose estimator.
- Implementation of a Free space detector to estimate free space around the ego vehicle.
- Implementation of a Lane detector to estimate different lane boundaries. Providing Lane coefficients in vehicle world coordinates from the detected lanes in an image.
- Design of software pipelines to display outputs of what the vehicle perceives real-time to an HMI for testing and demonstration purposes.
- Unit Testing and validating the performance of each software module.
- Sending information of all detected objects and their positions in CAN format to vehicle controller.

Out of Scope:

- It was jointly agreed with all stakeholders that the task related to deciding the number of cameras and their optimal positions to prevent blind spots around the ego vehicle was kept out of scope.

1.5 Interaction with other Projects

The ego vehicle is fitted with different types of sensors like Radar, IMU, GPS, V2V and wheel-speed sensors besides the surround cameras. The information from these sensors is fused to get a robust estimate of the position of the ego vehicle and its surround objects. Since the camera information is also an input for this fusion, it involves working closely with the team involved with Radar and sensor fusion. Also, the lane information perceived by the system are inputs to the path planning algorithms. In the end, the surround vision system should couple well with the existing ongoing projects to be successful.

1.6 Unique Contributions

In this project, given the information obtained from multiple cameras mounted on the roof of a vehicle, a perception system is developed in order to get a meaningful representation of the environment. A systems engineering approach is used for identifying and decomposing the desired behaviour into modular functions. Although parts related to the implementation of these functions have been done before it does not appear that anyone has ever brought it all together into one single project and implemented it on a real-time system.

The first contribution of this project is the design of a novel framework that includes several functions: Object, Lane & Free space detection, Tracking, Target classification, Position estimation of vehicles and Pose estimation of targets. Each sub-function is addressed by using Deep-Neural networks and novel algorithms.

The second contribution is the implementation and evaluation of a vehicle-agnostic algorithm for accurately estimating the position of vehicles in real-time using monocular vision.

The third contribution is the implementation and evaluation of a pose estimation algorithm for evaluating the pose of a known target vehicle for vehicle platooning applications.

1.7 Overview of the Thesis

The following is a brief account of the contents of this thesis. Chapter 2 addresses the problem with the existing setup and the concerns of the stakeholders involved in this project. Chapter 3 describes the requirements elicitation process. Different driving scenarios are evaluated to identify the functional requirements and the system context. Chapter 4 follows the white-box modelling of the system by defining the architecture and subsystems. Chapter 5 considers different design choices and algorithms that fulfil the requirements and concerns. Chapter 7 and 8 explain how these algorithms are implemented and evaluated in conformance with the driving scenarios and requirements. Chapter 9 concludes with the conclusion and recommendations for future work.

■■■

2. Analysis

In this chapter, the main goal of the project is decomposed by analyzing the task required. First, a problem analysis is carried out to explain the existing hardware setup, identify the issues related to such a configuration and define a problem statement. A stakeholder analysis is carried out to explore the concerns and to identify the key drivers for this project. In the last section, the CAFCR methodology is explained to initiate the architecting process.

2.1 Problem Analysis

In a platooning scenario, cooperative automated vehicles need a robust and reliable perception system to perceive the environment accurately in real-time.

2.1.1. Project Background

Before this project, the demonstrator vehicle, i.e. a Renault Twizy, consisted of the following components, as shown in figure 6 below.

1) Front-facing stereo vision camera
2) NVidia Drive PX 2
3) Vehicle Control System: Simulink Real-time PC
4) Front-facing Radar

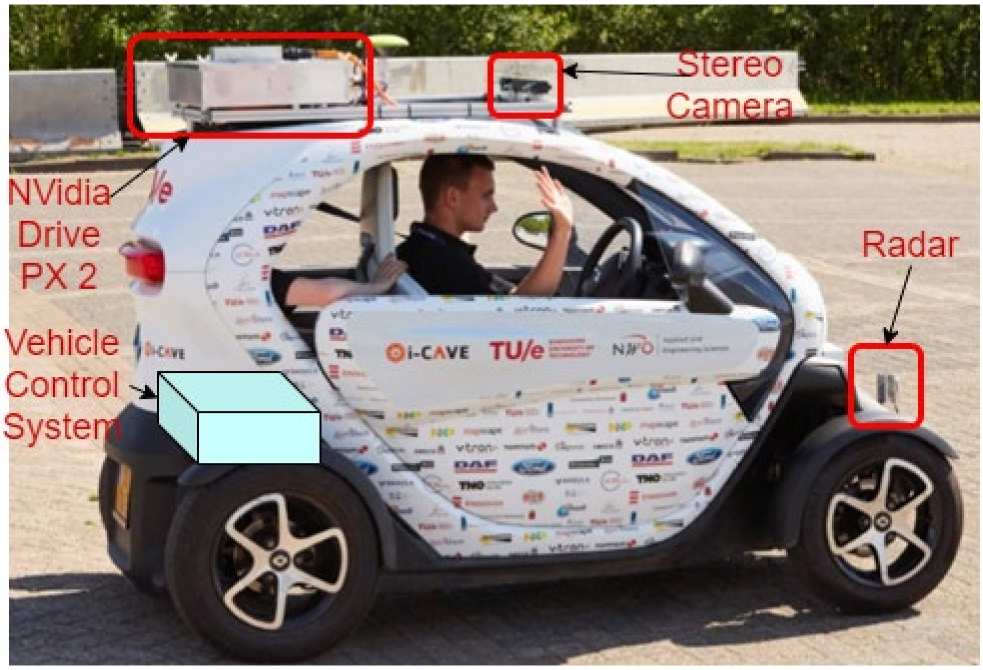


Figure 6: Existing Setup

The sensory information captured by the stereo vision camera was processed on the Drive PX 2 to detect and compute the position of objects in front of the vehicle. The computed information was provided to the vehicle control system for fusion with radar measurements.

The issues with such a configuration were as follows:

1) Since platooning is the main objective of the i-CAVE project it was required that the target vehicle, (another Renault Twizy) be placed in front of the ego-vehicle from the beginning of any test. The nearest vehicle was assumed as the target to be followed. In scenarios where multiple vehicles are in front of the ego vehicle, the control system would follow the nearest vehicle which may not be a target leading to undesired behavior.

2) For platooning, it was also necessary to have information related to the pose of the targets. This information was obtained only through V2V communication between the targets and the ego vehicle, which was unreliable. Hence, there was a need to measure the pose of the targets using extrinsic sensors, for.eg the vision system.

3) For safe autonomous driving, tracking of surrounding vehicles is essential for many tasks crucial to autonomous driving, such as obstacle avoidance, path planning, and intent recognition. The generated tracks should be accurate, long and robust to sensor noise to have good high-level spatial reasoning. In the existing setup, the detection and tracking of objects were limited to only the FOV of the front radar and the stereo camera pair, making it difficult for the ego-vehicle to pre-emptively take collision avoidance measures based on the knowledge of spatial positions of all objects around its surroundings.

4) In order to have better fusion, ideally, the radar and vision inputs must be synchronized. Due to the long execution time required for stereo depth calculation, the frequency at which vision measurements were provided, (7Hz) was lower compared to the radar measurements (14Hz) resulting in poor fusion.

2.1.2. Problem Statement

To address the problems cited above, a decision was made by the stakeholders to design a vision system that can have the following functions:

1) Identify a potential target vehicle (a specific type or brand, in this case, a Renault Twizy) among other vehicles
2) Investigate the possibility of estimating the position of objects using monocular vision with lesser execution time.
3) Estimate the pose of target vehicles using monocular vision.
4) Design a scalable system constituting of a variable number of monocular cameras and radars to capture the 360 degrees surrounding of the ego-vehicle.
5) Have additional functionalities to detect the free space boundary around the ego-vehicle and lanes in front and rear of the vehicle.

2.2 Stakeholder Analysis

In this section, a thorough analysis of different stakeholders, directly or indirectly involved in this project, is presented. This analysis was initially carried out during the project initiation phase and regularly revised to ensure that each stakeholder is constructively involved in contributing to the envisioned future.

The objective of this analysis was to identify concerns related to what are the financial and emotional interests each stakeholder has in the outcome of this doctorate assignment, What are various concerns and challenges that project stakeholders hope to solve, what expectations they have from the outcome of this project and how would it affect them in the long run. Another major objective of stakeholder analysis is to build consensus, between different parties, for project priorities and deliverables. Fortunately, there were no disagreements, and therefore no additional effort was required for alignment of the stakeholders. From a reader's viewpoint, it is imperative to get an insight into the circumstances and conditions guiding the strategic objectives and goals of this assignment. This insight will enable the reader to understand the various reasons behind the design choices made. The analysis is split into three different groups which are discussed in the following sections. The first group is the I-Cave program partners which address organizational and business-level objectives and interests. The second group, designers and developers, address their interests and concerns on a more personal level. Here, current developers and designers were considered, but also those who will be responsible for the continuation of the project. The final group is the PDEng Management, where the trainee needs to fulfil the requirements of the program.

2.2.1. I-Cave Program Partners (Eindhoven University of Technology)

Eindhoven University of Technology (TU/e) is committed to building solutions related to 'Smart Mobility'. TU/e holds a great deal of expertise in the fields of Intelligent Transport Systems, Automotive Technology, and ICT/ Embedded Systems.

The Mobile Perception group headed by assistant professor Dr. Dubbelman is involved in building perception algorithms related to autonomous applications. The main questions that Dr Dubbelman is trying to answer from this assignment are as follows:

1) Can we have a real-time implementation of an algorithm that can estimate the pose of a vehicle with prior knowledge of its shape?
2) Can we design a system that can efficiently, robustly detect & estimate the position of objects in real-time?

When referring to efficiency, it means the computational cost (i.e. Execution time and memory) is at or below some acceptable level.

From the Dynamics & Control group, Dr. Tom van der Sande is responsible for leading a team in the development of cooperative and automated driving controllers. The key research goals he is trying to solve are as follows:

1) Can we accurately identify and localize dynamic objects in the surrounding of the ego-vehicle?
2) Can we have a coherent image of the surrounding by having a wider FOV e.g.(surround vision) thus increasing the time to collision? This will lead to an improvement in the margin of safety and smoother controller action.
3) Can we have a robust estimate of surrounding objects by fusion of all sensor measurements?
4) Can we have a reliable pose estimate of target vehicles (Renault Twizys) to be followed in platooning, by fusion of measurements from multiple external sensors i.e. V2V Communication and cameras?

2.2.2. Designers and Developers

The group of designers and developers consists of people who are presently working on the project and those who will be working in the future. Dr. Narsimlu Kemsaram, a Post.Doc candidate at TU/e, is responsible for the implementation of different algorithms developed by the MPS group on the Drive PX 2. His main concerns are related to the non-functional aspects of the system. The developed software architecture should be easy to understand and maintainable. The code should be modifiable and extendable for the use of better algorithms in the future.

2.2.3. PDEng Management (ASD Program)

Supervised by Dr.ir. Peter Heuberger (Program Manager, PDEng MSD/ASD), the PDEng program is concerned with successful completion of the PDEng final assignment.

2.3 CAFCR Analysis

System design is a creative process and does not follow hard and fast rules, however, there are various guidelines and tools from system engineering are used to assist the process. One such tool is the CAFCR methodology which is an architectural reasoning framework that helps in making the design choices.

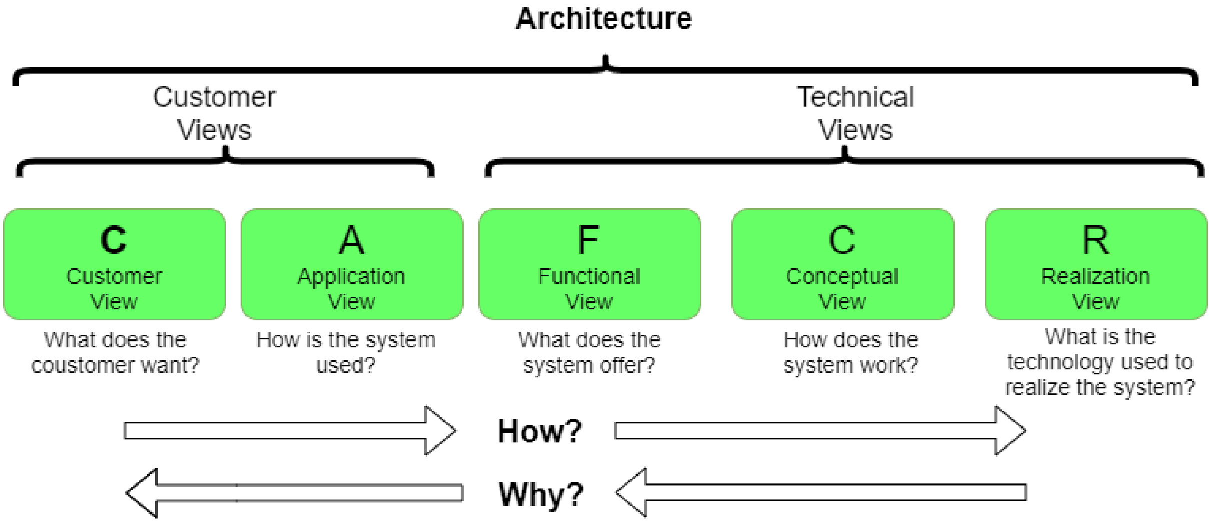


Figure 7: CAFCR a Multi-view method for System Architecting [4]

CAFCR provides various views depicted in Figure 7 in order to evaluate various design choices and analyze a design. These views enable the designer to consider the product from multiple perspectives.

The customer view provides answers to the ‘Why’ question from the customer. “Why is the customer interested in carrying out this design experiment?”. This is established in the 1st chapter. The customer’s view also includes defining the stakeholders and their concerns. An overview of the CAFCR Framework applied to this project is shown in Figure 8.

The application view focusses on the interaction of the system with the environment. The system must function robustly in different weather and driving conditions. An in-detail study is carried out in Chapter 3, considering the needs of the end-user in different possible driving scenarios.

The functional view addresses the question of “what the system offers”. This is described in detail in Chapter 1 and 3, highlighting the functional and non-functional requirements. Chapter 4 and 5 illustrate the conceptual view, describing the architecture and the possible concepts needed to build the functionality. Finally, the realization view is described in Chapter 6, emphasizing on how the system is implemented on the Drive PX 2.

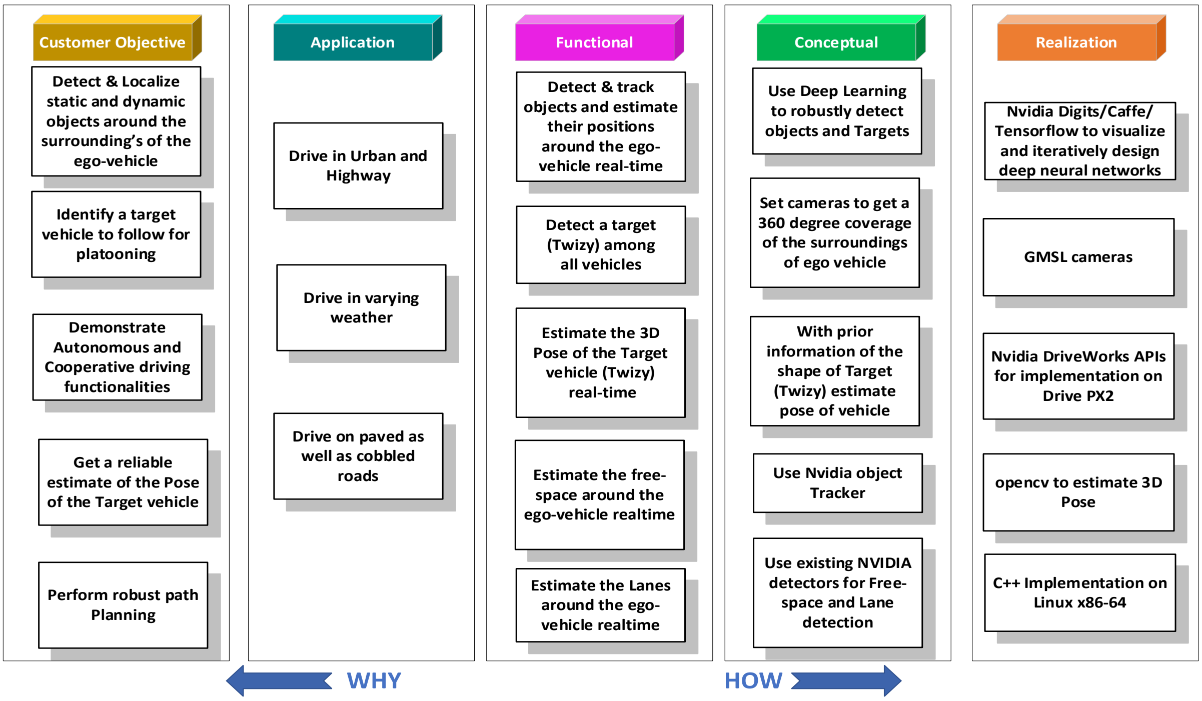

Customer Objective
Detect & Localize static and dynamic objects around the surrounding’s of the ego-vehicle
Identify a target vehicle to follow for platooning
Demonstrate Autonomous and Cooperative driving functionalities
Get a reliable estimate of the Pose of the Target vehicle
Perform robust path Planning
Application
Drive in Urban and Highway
Drive in varying weather
Drive on paved as well as cobbled roads
Functional
Detect & track objects and estimate their positions around the ego-vehicle real-time
Detect a target (Twizy) among all vehicles
Estimate the 3D Pose of the Target vehicle (Twizy) real-time
Estimate the free-space around the ego-vehicle realtime
Estimate the Lanes around the ego-vehicle realtime
Conceptual
Use Deep Learning to robustly detect objects and Targets
Set cameras to get a 360 degree coverage of the surroundings of ego vehicle
With prior information of the shape of Target (Twizy) estimate pose of vehicle
Use Nvidia object Tracker
Use existing NVIDIA detectors for Free-space and Lane detection
Realization
Nvidia Digits/Caffe/ Tensorflow to visualize and iteratively design deep neural networks
GMSL cameras
Nvidia DriveWorks APIs for implementation on Drive PX2
opencv to estimate 3D Pose
C++ Implementation on Linux x86-64
WHY
HOW

Figure 8: CAFCR Framework Application

■■■

3. Requirements Elicitation

This chapter describes the requirement elicitation process used for this project. As part of the Application and Functional views of the CAFCR methodology, different driving scenarios are analysed to derive the requirements and key performance indicators. Based on the actors identified and the expected behaviour system functions and requirements are determined for the system.

3.1 Elicitation process

To come up with a set of functional requirements for the system, the System Engineering Methodology for Automated systems (SEMAS) [5] described by Valeo was applied. SEMAS is a System Engineering Methodology that combines model-based system engineering with traditional requirements engineering. SEMAS is a layered approach where the analysis is performed at different levels of abstraction. The first and the highest layer is the Real-world Layer, where we analyze the needs of the end-user by specifying the expected behaviour of the ego-vehicle in interaction with its environment. In the second layer, we analyze the expected behaviour of the system under design (i.e. Surround Vision System) and the interaction with other interior systems in the vehicle. Finally, functional requirements are generated from the identified functions of the system.

3.2 Real World Layer

In this section, considering the ego-vehicle as a black-box, the behaviour and interactions with the environment are analyzed, as shown in Figure 9. Since the project focuses only on the perception aspect, the ego-vehicle's behaviour is scoped to only examining this aspect. For better analysis, it is assumed that the perception system consists of only the proposed system (i.e. SVS) and no other external sensors.

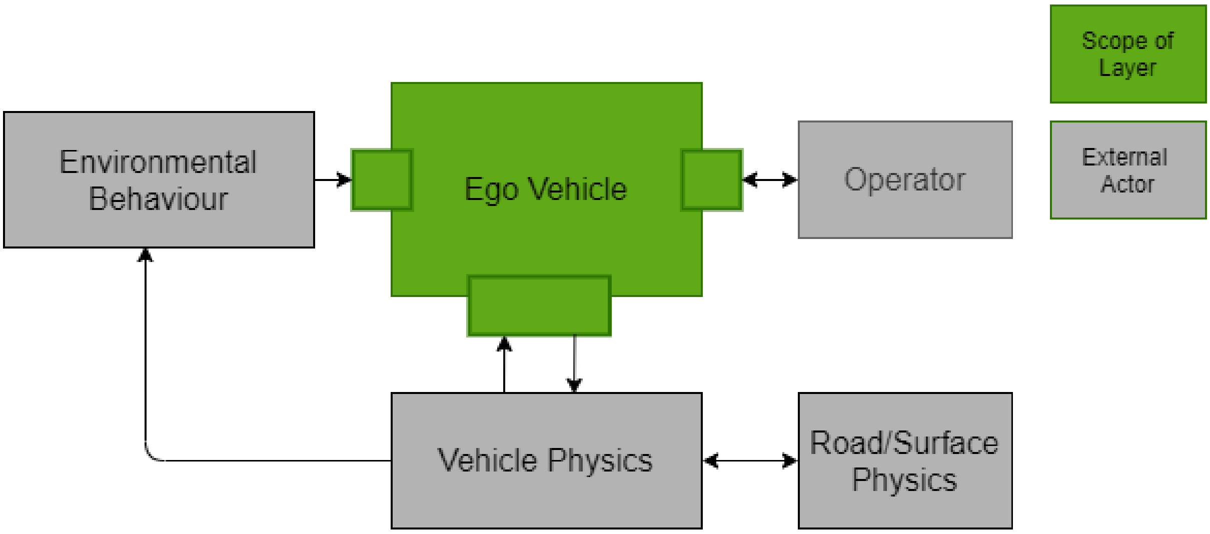
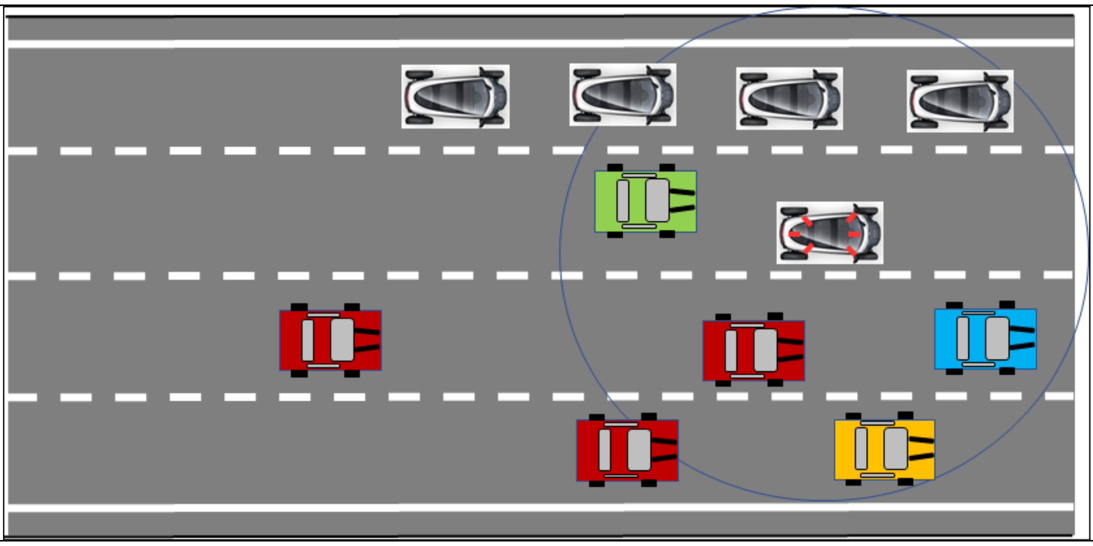


Figure 9: Real-World Layer: Context of the Ego-vehicle

Several scenarios were analyzed in varying levels of complexity, road and weather conditions, as described in appendix 1, for e.g. vehicle approaching and departing from the ego-vehicle, driving in a parking lot, preceding vehicle driving up a gradient, etc. One such scenario which describes CACC is described in Table 1 below.
By describing the driving scenarios, it is identified which external actors trigger activities. For instance, in the below scenario target & other vehicles, free space & Lane markings, need to be detected and their positions need to be estimated by the ego-vehicle. Secondly, which influences exist in the environment are also identified. For e.g. the road surface conditions influence vehicle physics (pitch, roll motion); similarly, the lighting conditions influences is the detection accuracy.

Table 1: Driving Scenario - CACC

DS 1 - CACC

Context
Ego-vehicle driving beside a platoon of target vehicles on a highway.
Actors
Target vehicles (Twizy's), other vehicles, Lane markings, Lighting conditions, Road conditions
Preconditions
1) The target vehicles are driving in a platoon 2) There are other vehicles around the ego-vehicle 3) The visibility is clear, and the weather is sunny 4) The SVS system is inactive
Trigger
1) SVS system is activated.
Expected ego-vehicle behaviour
1) The ego-vehicle detects and classifies the vehicles which are clearly or partially visible to the cameras as a "Target"(Twizy) or "other vehicle"(cars) around its surrounding 2) The ego-vehicle calculates the pose of the detected target vehicle (Twizy) with respect to itself. 3) The ego-vehicle calculates the position of other detected vehicles. 4) The ego-vehicle computes the curvature and position of the left and right lane markings belonging to the ego vehicle if present 5) The ego-vehicle detects the free-space around the vehicle 6) The ego-vehicle displays all perceived information to the operator
Postconditions
1) Detections are provided until the system is deactivated
Key Performance Indicators (KPIs)
1) The number of false negatives detected. (objects incorrectly classified have a higher risk) 2) The detection range of the system. (How far can an object be accurately detected?) 3) The accuracy at which the position of objects are estimated 4) The number of target vehicles the system can handle (detect as well as provide pose). 5) The time required to detect and estimate the position of vehicles/objects 6) The accuracy of the vehicle position estimate with respect to the ground truth 7) The execution time required to compute the pose of the target vehicle when detected 8) The accuracy of the computed pose of the target vehicle when detected 9) The rate at which the system updates the position of the targets and other vehicles on the CAN bus

3.3 Vehicle Level

In this level, the system under design, i.e. the surround vision system (SVS) is considered as a black box, and the interactions between vehicle internal components and functions are analyzed. Then system functions are derived which describe the primary functions of the system. After this, detailed functional requirements are derived. Non-functional requirements are also added based on the concerns of the stakeholders.

3.3.1. System Context:

The system under design, along with the actors directly interacting with the system, are shown in the below Figure 10.

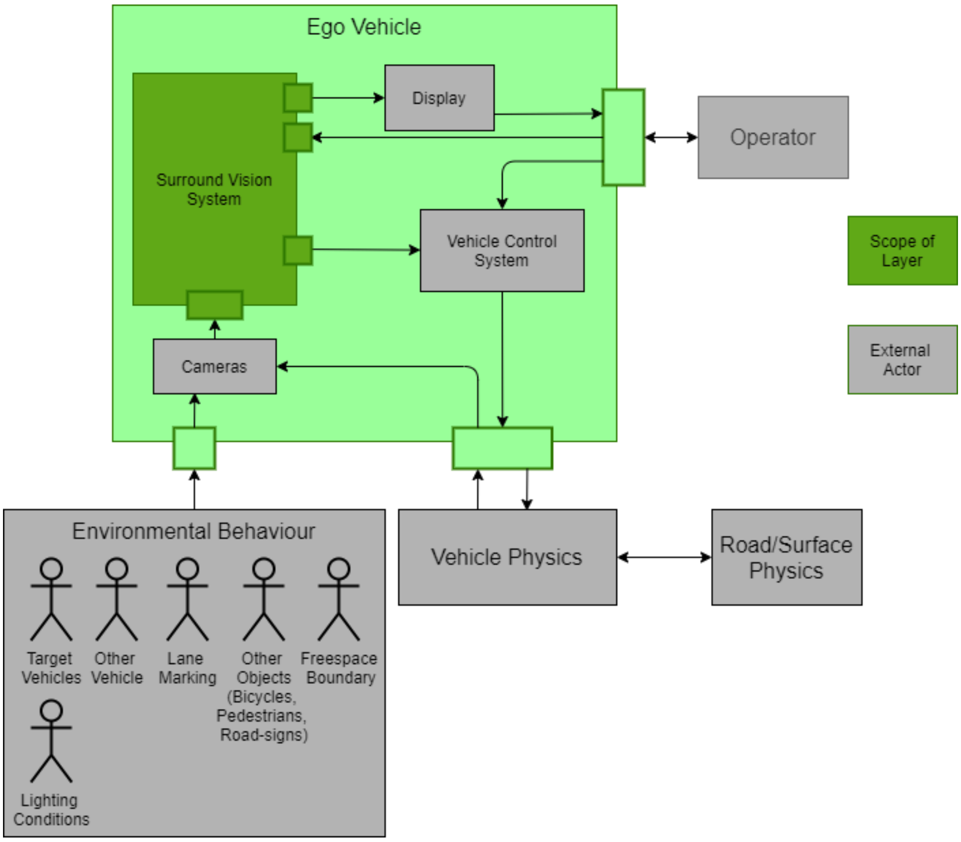

Ego Vehicle
Display
Operator
Surround Vision System
Vehicle Control System
Scope of Layer
External Actor
Cameras
Environmental Behaviour
Target Vehicles
Other Vehicle
Lane Marking
Other Objects (Bicycles, Pedestrians, Road-signs)
Freespace Boundary
Lighting Conditions
Vehicle Physics
Road/Surface Physics

Figure 10: Vehicle Level Layer: Context of the Surround Vision System

The surround vision system is a software application running on an embedded device (NVidia Drive PX 2) mounted on the ego vehicle. An operator interacts with the system for activation and deactivation. The presence of actors in the scene, identified in the Real-world level, i.e. target & other vehicles, bicycles, pedestrians, road sign, traffic sign, lane markings, and free space, are captured by the cameras mounted on the ego vehicle. The quality of the information captured by these cameras may be influenced by the environmental conditions like weather conditions (rainy/sunny), lighting conditions (day/night) or even glare from sunlight or shiny reflective surfaces. There is no upper bound on the number of actors present in the scene. There may be up to twelve cameras attached. Cameras having a different FOVs may be used and mounted on the ego-vehicle chassis at various locations/orientations. Vehicle motion may affect theses orientations. The cameras provide the information captured to the SVS system asynchronously. The SVS is responsible for performing various detections by analyzing the video feed received by the cameras. The results obtained are processed and communicated to the Vehicle control system via the controller area network (CAN). The format and type of information transferred is explained in Section 3.5 in detail. Besides this, for testing purposes, the results obtained are also rendered on a human-machine interface (HMI), which is realized as a display screen installed inside the ego vehicle.

3.3.2. System Functions

System functions are building blocks of end-user functions extracted from concrete scenarios. System functions are modelled in IBM Rational Rhapsody as operations and are represented in the system function diagram in figure 11. After analyzing each driving scenario, it was concluded that the behaviour of the SVS system does not change. It has the behaviour of a sensor (like a surround-view camera) having the same behaviour irrespective of the driving conditions. Thus, only one of the driving scenario, i.e. cooperative adaptive cruise control (CACC) having the most generalized behaviour was chosen for deriving the system functions and the requirements.

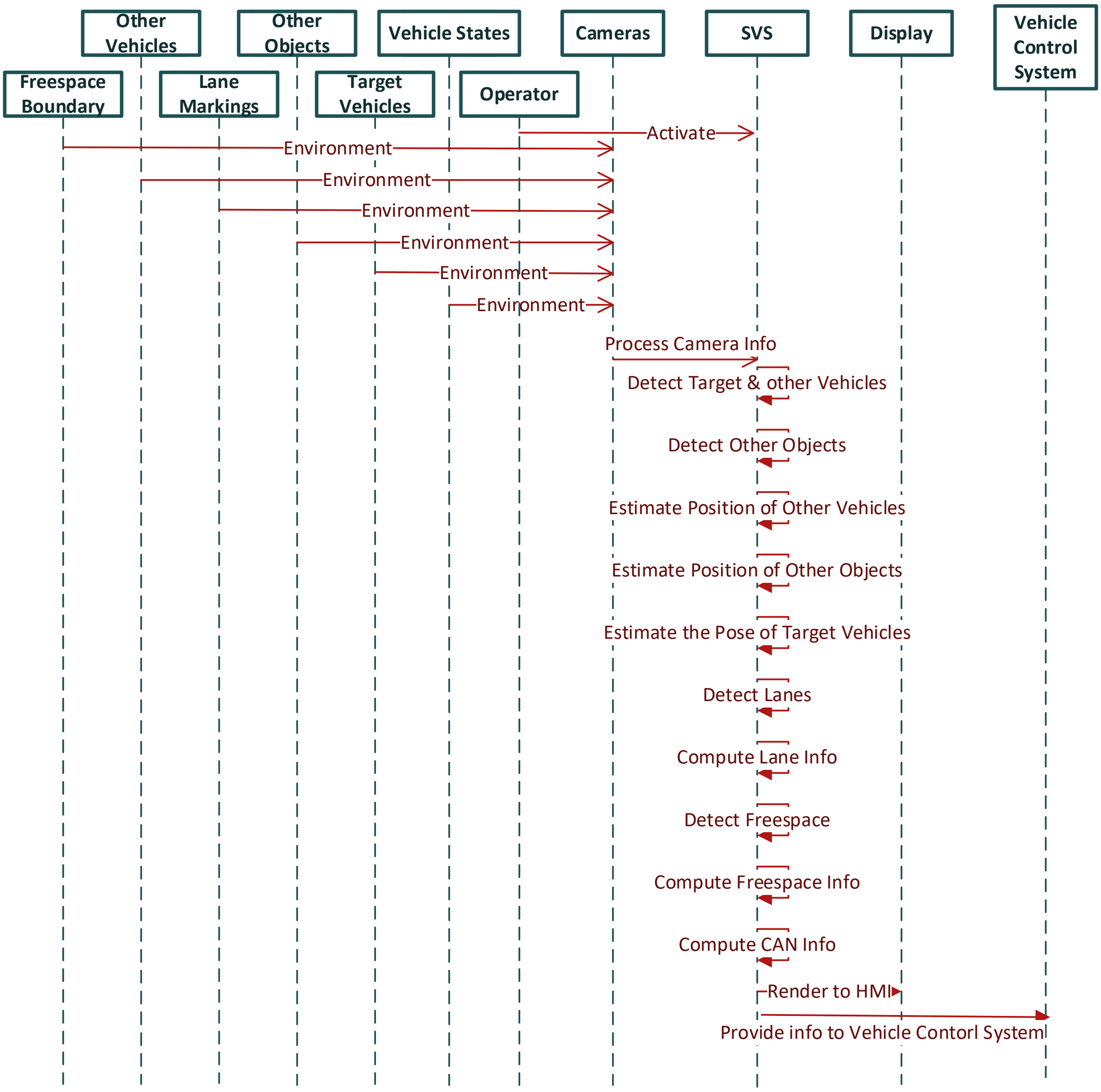

Other Vehicles
Other Objects
Vehicle States
Cameras
SVS
Display
Vehicle Control System
Freespace Boundary
Lane Markings
Target Vehicles
Operator
Activate
Environment
Environment
Environment
Environment
Environment
Environment
Process Camera Info
Detect Target & other Vehicles
Detect Other Objects
Estimate Position of Other Vehicles
Estimate Position of Other Objects
Estimate the Pose of Target Vehicles
Detect Lanes
Compute Lane Info
Detect Freespace
Compute Freespace Info
Compute CAN Info
Render to HMI
Provide info to Vehicle Contorl System

Figure 11: System Functions derived from the driving scenario

The system functions required for surround vision are as follows:

- FUNC01: System Activation/Deactivation
- FUNC02: Receive camera info
- FUNC03: Process camera info
- FUNC04: Detect vehicles
- FUNC05: Detect other objects
- FUNC06: Estimate target vehicle pose
- FUNC07: Estimate Position of Other Vehicles & objects
- FUNC08: Detect lanes
- FUNC09: Compute lane info
- FUNC10: Detect free Space

- FUNC11: Compute free space Info
- FUNC12: Compute CAN info
- FUNC13: Render to HMI
- FUNC14: Provide Info to Vehicle Control System

3.3.3. Functional Requirements

Detailed requirements are derived for each system function from the description of the Driving Scenario. Each system function should satisfy the corresponding requirements. Requirements are further categorized based on priority as low & high.

FUNC01: System Activation/Deactivation

REQ #	REQ Priority	REQ Title	REQ Description
REQ001	Low	Operator Activation/Deactivation	The system shall be activated or deactivated by the operator.

FUNC02: Receive camera info

REQ #	REQ Priority	REQ Title	REQ Description
REQ002	High	Camera Info	The system shall receive the following properties of the cameras mounted on the ego vehicle: Model: Pinhole resolution, distortion coefficients, camera location & orientation relative to vehicle rear axle (z=0, ground Plane). The camera type shall be “AR0231” camera with supported `RCCB` sensor.

FUNC03: Process camera info

REQ #	REQ Priority	REQ Title	REQ Description
REQ003	Low	Number of cameras	The system shall receive images from up to 12 cameras simultaneously.
REQ004	Low	Supported image properties	The system shall receive images from the cameras with the following properties: 1) Format: RCCB 2) Resolution: 1920X1208 3) Frames Per Second: 30 FPS

FUNC04: Detect vehicles

REQ #	REQ Priority	REQ Title	REQ Description
REQ005	High	Types of object classes	The system shall detect vehicles in the image and classify as a target vehicle or other vehicle.
REQ006	Low	Provide class confidence	The system shall provide a float value from 0 to 1 indicating how confident is the estimated class
REQ007	Low	Provide Timestamp	The system shall record and provide the timestamp for every vehicle detection.

FUNC05: Detect other objects

REQ #	REQ Priority	REQ Title	REQ Description
REQ008	Low	Other Class Types	The system shall detect other objects like pedestrians, road signs, traffic signs and bicycles.
REQ009	Low	Provide class confidence	The system shall provide a float value from 0 to 1 indicating how confident is the estimated class
REQ010	Low	Timestamp	The system shall record and provide the timestamp for every other object detected

FUNC06: Estimate target vehicle pose

REQ #	REQ Priority	REQ Title	REQ Description
REQ011	High	Target Position Information	The system shall provide the x, y coordinates for detected targets with respect to ego vehicle in meters.
REQ012	High	Heading Angle Information	The system shall provide the heading angle (yaw) of the target with respect to ego vehicle in deg.
REQ013	High	Timestamp	The system shall provide the Timestamp for every detected target object.
REQ014	High	Provide Instance & Camera ID	The system shall provide the camera id for each target detected along with a unique instance ID.
REQ015	High	Target Orientation Angle	The system shall provide the targets orientation angle.

FUNC07: Estimate the position of other vehicles & objects

REQ #	REQ Priority	REQ Title	REQ Description
REQ016	High	Position of other objects	The system shall provide the X, Y position of other vehicles with respect to ego vehicle in meters.
REQ017	High	Provide Instance & Camera ID	The system shall provide the camera ID for each vehicle detected along with a unique instance ID.

FUNC08: Detect lanes

REQ #	REQ Priority	REQ Title	REQ Description
REQ018	High	Perceive Lane Markings	The system shall perceive the visible lane markings in front and rear of the ego vehicle.
REQ019	High	Lane Positions	The system shall detect the following lane lines types- Ego-lane left line, Ego-lane right line, Left adjacent lane line, Right adjacent lane line
REQ020	High	Lane Types	The system shall detect lanes of the following type-- Solid lanes, Dashed lanes, Road Boundary, undefined.
REQ021	High	Camera ID and Timestamp	The system shall provide the timestamp and camera id from which camera the lanes are detected.

FUNC09: Compute Lane Info

REQ #	REQ Priority	REQ Title	REQ Description
REQ022	High	Calculate polyline	The system shall calculate the coefficients of a polyline fit over the lane markings.
REQ023	High	Format for Lane Info	The system shall provide the offset ($c0$), gradient ($c1$), curvature ($c2$) of the polyline, where polyline function is given as $y=c0+c1x+c2x^2+c3x^3$

FUNC10: Detect Free Space

REQ #	REQ Priority	REQ Title	REQ Description
REQ024	High	Free Space Boundary	The system shall identify and display the drivable free space from all the cameras.
REQ025	High	Free Space Boundary Labels	The boundary will be associated with four semantic labels: Vehicle, Pedestrian, Curb, Other

FUNC11: Compute Free Space

REQ #	REQ Priority	REQ Title	REQ Description
REQ026	High	Free Space Boundary Format	The system shall compute free space boundary point locations in car domain (meters).

FUNC12: Compute CAN Info

REQ #	REQ Priority	REQ Title	REQ Description
REQ027	High	Compute CAN messages of detected objects	The system shall compute CAN messages of the following - 1) Detected Vehicles 2) Detected other Objects 3) Detected Lanes 4) Free space.

FUNC13: Render to HMI

REQ #	REQ Priority	REQ Title	REQ Description
REQ028	High	Display Bounding Boxes	The system shall display bounding boxes over each object detected along with its label
REQ029	Low	Label Target vehicle Bounding Boxes	The system shall label detected target vehicles as “Target” and other vehicles as “Other vehicle”
REQ030	High	Camera feed and No. of Display Windows	The system shall display the video feed from each camera in a separate window along with its label.
REQ031	High	Display position and heading for targets, Other Vehicles & Objects	The system shall display the position X, Y for all vehicles. Also, additionally the heading & orientation values for target vehicles
REQ032	High	Display Lanes	The system shall display detected lanes on the HMI in the following semantic colours: - Red: Ego-lane left - Green: Ego-lane right - Cyan: Left adjacent lane - Blue: Right adjacent lane - Dark yellow: Undefined position
REQ033	High	Display Free Space	The system shall display the Free-space boundary on the HMI in the following semantic colours: Red: Vehicle, Green: Curb, Blue: Pedestrian, Yellow: Others

FUNC14: Provide info to Vehicle Control System

REQ #	REQ Priority	REQ Title	REQ Description
REQ034	High	Information Format	The system shall provide the information to Vehicle Control system in CAN format
REQ035	Low	Information Update Rate	The system shall update the information on the CAN bus at a baud rate of 500 kbps
REQ036	High	Transmitted data information Properties	The system shall transmit the following information: Struct Twizy: {Position: x,y meters Instance ID: val Heading: xx deg Orientation: xx deg Confidence: Value Timestamp: Value secs. Camera id: val }
REQ037	High	Transmitted Other Vehicle Information Properties	Struct Car: {Position: x, y meters Instance ID: val

			Camera id: val (camera that detected the object) Confidence: Value Timestamp: Value secs }
REQ038	High	Transmitted Lane Information Properties	Struct Lane_Boundary: {Poly Coef: *c1,c2,c3,c4* *(where* $y= c1+c2x+c3*x^2+c4x^4$) Where x, y are in world coordinates Camera id: val (camera that detected the object) Timestamp: Value secs }

REQ039	High	Transmitted Free Space Boundary Information Properties	Struct Free Space Boundary
			{
			Curb: (Boundary World Points: x,y,z)
			Vehicle: (Boundary World Points: x,y,z)
			Person: (Boundary World Points: x,y,z)
			(ego vehicle Coordinates)
			Camera id: Val (camera that detected the object)
			Timestamp: Value secs
			}

3.3.4. Non-Functional Requirements

Each module used for building the SVS system must execute at different stages of the program. These modules must be written in an object-oriented form so that their interfaces are clearly identifiable. Having an object-oriented design also helps in maintainability and testability of each module. The following are the requirements from the software design perspective.

REQ #	REQ Title	REQ Description
REQ040	Object-Oriented design	The software design must be hierarchical and completely object-oriented.
REQ041	Single entry, single exit	Software functions must have only one entry and exit point.
REQ042	Modular code	The individual class design must follow the principle of single responsibility and the principle of least knowledge from software engineering.

3.4 System External Interfaces

Based on the context diagram in Section 3.3.1 and the interface requirements described above in Section 3.3.3, a diagram highlighting the external interfaces of the system is shown below Figure 12. The diagram explains the direction of flow and the type of information flowing from the system.

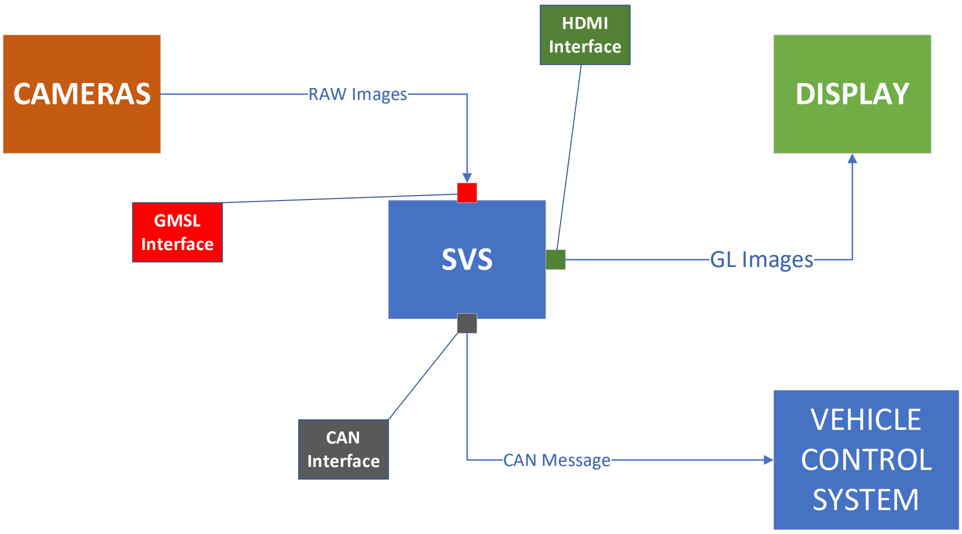


Figure 12: System External Interfaces

Information of the environment captured by the cameras is sent in the form of RAW camera frames to the system via the GMSL interface. The rate at which these frames are sent to the system depends on the FPS of the cameras. When the system is available, it picks the latest frames sent and processes it. The processed frames are converted into GL format, which are render ready. These frames are then sent via the HDMI interface to the display unit for rendering. Similarly, the CAN messages written by the system after performing all detections and computations are sent to the Vehicle Control system. In the architecture level described in the next chapter, only the interfaces of the SVS will be used for expressing any communication with the system's environment.

■■■

4. System Architecture

In the previous chapter, the system was considered as a black box, and its interfaces and functions were defined. In this chapter, the system is considered as the white box, the identified system functions are assigned to individual functional components. Furthermore, different architectural views are used to express when does each functional component process information, what are the interfaces of these components and what information is exchanged between these components.

Different viewpoints can be used to describe the architecture of the system. Each viewpoint tries to address specific concerns of the stakeholders. The Functional, Concurrency and Logical viewpoints are used to emphasize on the behaviour of the system. Furthermore, to explain how the software is developed, SysML structural diagrams are used to explain the software classes and their interfaces.

4.1 Functional Viewpoint

The functional view of a system defines the architectural elements that deliver the functionality of the system. This view describes the system's runtime functional elements, responsibilities and their primary interactions. It is a viewpoint which is understood by all stakeholders and hence explained first.

Figure 13 shows the interaction between various functional elements, and Figure 14 describes the list of functions provided by each functional element.

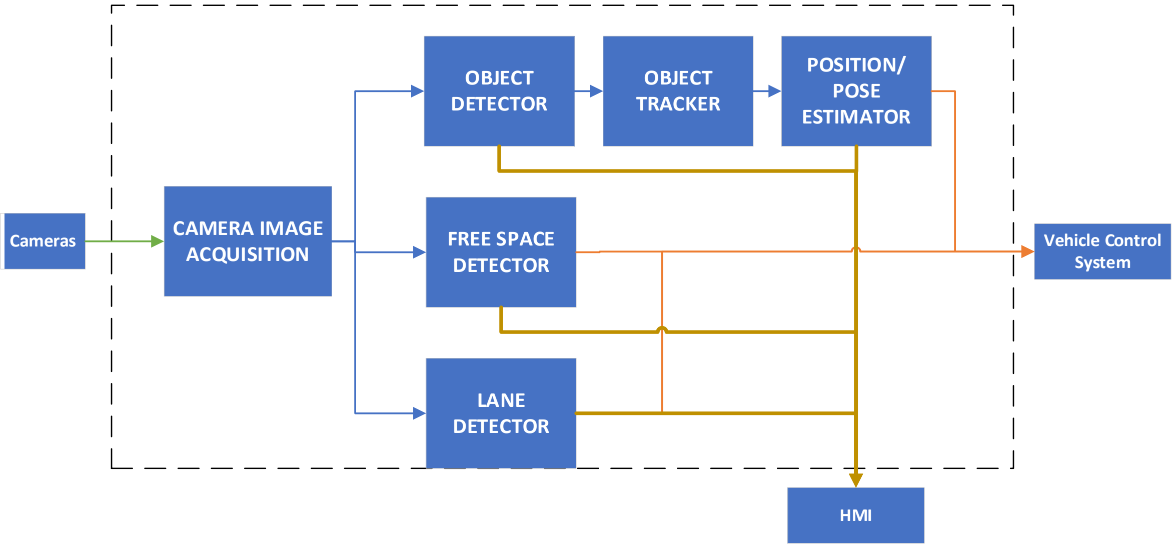


Figure 13: Functional Viewpoint - Functional Elements & their interactions

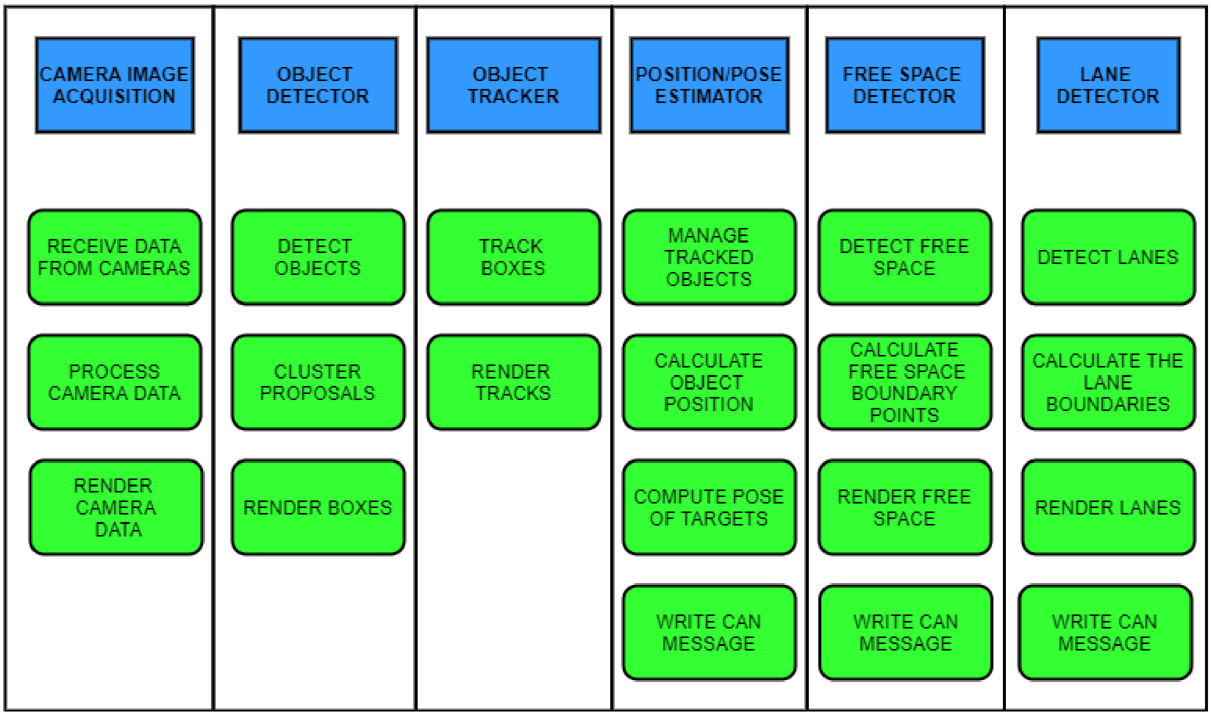

CAMERA IMAGE ACQUISITION
RECEIVE DATA FROM CAMERAS
PROCESS CAMERA DATA
RENDER CAMERA DATA
OBJECT DETECTOR
DETECT OBJECTS
CLUSTER PROPOSALS
RENDER BOXES
OBJECT TRACKER
TRACK BOXES
RENDER TRACKS
POSITION/POSE ESTIMATOR
MANAGE TRACKED OBJECTS
CALCULATE OBJECT POSITION
COMPUTE POSE OF TARGETS
WRITE CAN MESSAGE
FREE SPACE DETECTOR
DETECT FREE SPACE
CALCULATE FREE SPACE BOUNDARY POINTS
RENDER FREE SPACE
WRITE CAN MESSAGE
LANE DETECTOR
DETECT LANES
CALCULATE THE LANE BOUNDARIES
RENDER LANES
WRITE CAN MESSAGE

Figure 14: Functional Viewpoint - Each Functional Element & its decomposition

The cameras asynchronously provide image frames in RAW format to the system. Image processing operations are performed on these images to render and provide for other detectors.

The object detector performs the task of detecting objects in the image. It detects objects of desired classes and identifies the location of these objects in the image space by computing a bounding a box over each detection. Each object detected, has a label describing the class and bounding box coordinates with respect to the image plane.

While driving in traffic, there may be multiple objects present of different classes and thus in the image frames. To be conscious of how many objects are present in the scene, each object needs to be tracked and labelled by a unique instance identification number (instance ID). Moreover, it is also necessary to measure the lifetime of these objects around the ego-vehicle. Objects that have a longer lifetime are considered stable detections, worthy of communication to the vehicle control system. The object tracker is responsible for tracking these detected objects and providing stable tracks. The object tracker element accepts bounding boxes from the object detector and validates that each object detected, is present continuously in sequential frames. If the above condition is true, an object track is created having the following properties: a class label, box coordinates, an instance ID and Track-age. The tracker provides these object tracks to the pose/position estimator functional element.

The pose/position estimator accepts the object tracks and computes the positions of all tracked objects with respect to the ego vehicle. Furthermore, this element also estimates the pose (position+heading) for target vehicles. For further information on position estimation, refer to chapter 5. The pose/position information is then communicated to the vehicle control system via CAN-bus.

Besides object detection, lane detection and free-space detection functional elements are responsible for identifying the lane markings and the drivable collision-free space in the provided images respectively. The lane information and free space boundaries are communicated via the CAN-bus.

4.2 Concurrency Viewpoint

Concurrency view of a system is used to describe the system's concurrency state-related structure, and constraints. This involves defining the modules of the system that can run at the same time and how they can be controlled.

The system needs to accept inputs from multiple cameras and perform all inferencing operations simultaneously. As per the stakeholder's concerns described in Chapter 2, the total processing time for the system is critical and hence requires parallelization of those functions so that they can work independently. Another aspect that needs to be addressed in the architecture is the scalability of the system. The number of cameras that can be used is not predetermined, and hence the system must be scalable to work with an undefined number of cameras.

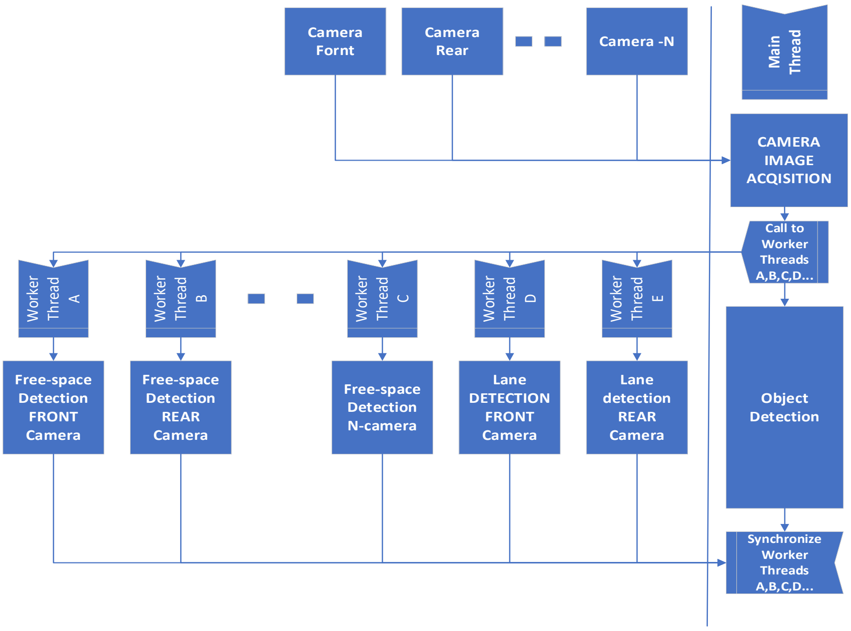

Camera Fornt
Camera Rear
Camera -N
Main Thread
CAMERA IMAGE ACQISITION
Call to Worker Threads A,B,C,D...
Worker Thread A
Worker Thread B
Worker Thread C
Worker Thread D
Worker Thread E
Free-space Detection FRONT Camera
Free-space Detection REAR Camera
Free-space Detection N-camera
Lane DETECTION FRONT Camera
Lane detection REAR Camera
Object Detection
Synchronize Worker Threads A,B,C,D...

Figure 15: Concurrency View- SVS

During runtime, the Camera Image Acquisition element is capable of handling multiple inputs from different cameras simultaneously. Once it reads from all cameras, it outputs an array of images. The object detector can accept this image array and perform inferencing on all the images simultaneously. However, the Lane and Free-space detector cannot handle an image array as input. Thus, each element of the array is fed to an instance of these detectors, as shown in Figure 15. These detectors work on independent threads and finally synchronize to the main thread after object detection.

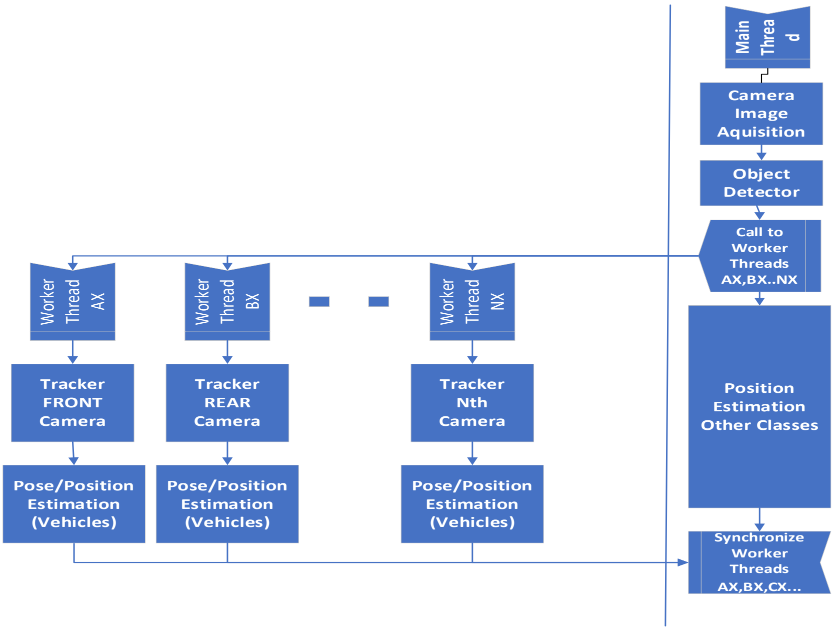

Main Thread
Camera Image Aquisition
Object Detector
Call to Worker Threads AX,BX..NX
Worker Thread AX
Worker Thread BX
Worker Thread NX
Tracker FRONT Camera
Tracker REAR Camera
Tracker Nth Camera
Pose/Position Estimation (Vehicles)
Pose/Position Estimation (Vehicles)
Pose/Position Estimation (Vehicles)
Position Estimation Other Classes
Synchronize Worker Threads AX,BX,CX...

Figure 16: Concurrency View - Object Detection

The concurrency view related to object detection is described in figure 16. The object detector outputs bounding boxes belonging to five object classes per camera which are inputs to the tracker module. The Tracker module can handle only bounding boxes related to a single class per camera. Ideally, if it is necessary to track objects of all five classes belonging to 6 cameras, 5x6=30 Trackers are required. This may have a significant load on the hardware/CPU. Since the main concern application in this project is in highway driving conditions, the scope is reduced to only tracking vehicles. The vehicle tracks outputted by these trackers will be fed to position/pose estimator of each camera and will finally synchronize with the main thread. While the worker threads perform the above operations, the main thread estimates the position for other objects using the clustered bounding from the object detector. The process of position estimation is different for the vehicle class and other classes. This is further elaborated in detail in Chapter 5.

4.3 Logical Viewpoint

For addressing the concern related to execution speed, it is necessary to separate activities which have high and low latency. Operations involving memory allocation & setting up inference pipelines are classified high latency operations and need to be performed only once during system initialization. In contrast, operations that execute iteratively on every image frame should be of low latency to achieve an overall reasonable execution rate. The functioning of the system at an abstract level is designed as a state machine having the following states: Standby, Initialization, Ready, Process & Release. Figure 17 shows the State-machine Diagram of the system.

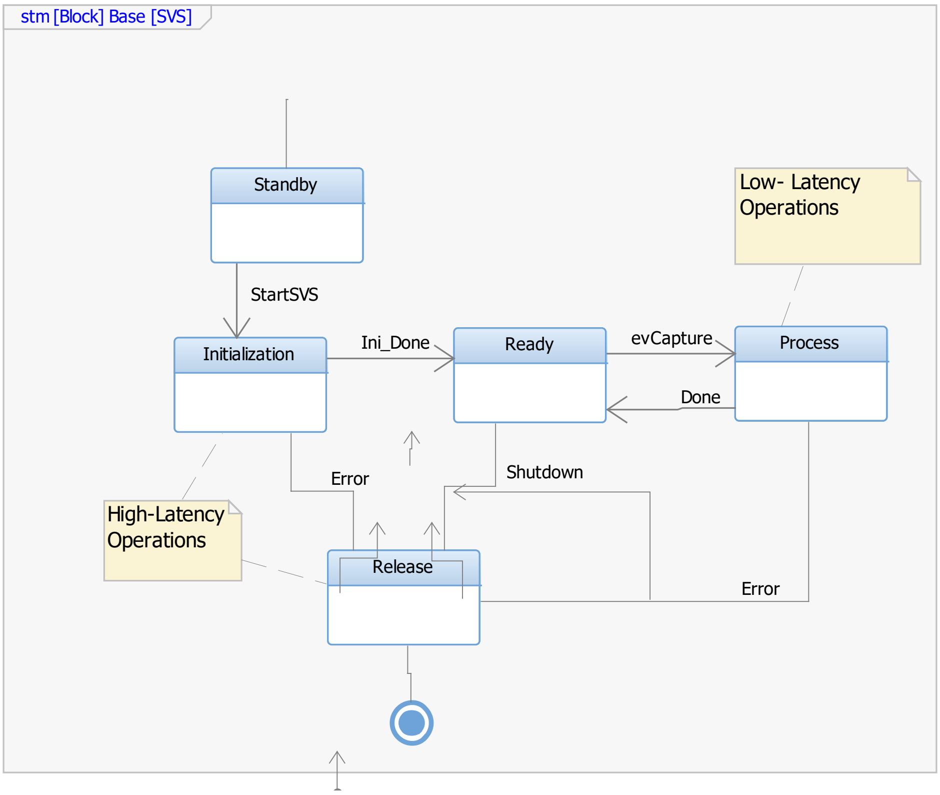


Figure 17: State Machine Diagram

The system is initially in the Standby state waiting for operator input. When the operator starts the system, it triggers a transition to the Initialization state where the main program execution starts.
During initialization the following functions are performed:

- Instantiation of functional components (depending on the number of cameras connected).
- Memory allocation of all data variables.
- Importing Deep Neural Network models and their memory allocation in the GPU.
- Creating Image Pipelines for processing images from raw input to a suitable format for each component.
- Setting the software handles of all sensors, detectors, trackers.

Once all the components have been successfully instantiated and initialized, the system is ready to accept inputs from the cameras and process the information iteratively. In the Ready state, the system waits for information from all the cameras. Once image frames are received, the system transitions to the process state.

The process state is where the low-latency critical operations of the system take place, as discussed in the concurrency view.
The following are the operations performed in the process state:

- Receive information from the cameras and process into a suitable format for detection & rendering
- Perform Lane, Object and Free space detections
- Tracking & pose estimation of vehicles.
- Render the image, detected Lanes, Free space Boundary, bounding boxes of detected objects and position of objects.
- Processing and sending the perceived obstacle information to the CAN-bus.
- Transition to the Ready state after successful completion of all computations.

The system transitions to the Release state in case of initialization and processing failures or when the operator terminates the system. In this state, the system releases all the variables and memory allocated during initialization. This state also is classified as high-latency.

4.4 Structural Views

The structural views of the system describe how the overall software is decomposed into classes. Based on the single responsibility principle, every module or class should have responsibility for a single part of the functionality provided by the software, and the class should entirely encapsulate that responsibility. As highlighted earlier in Section 4.1, each functional component is converted to modules such as Camera Image Acquisition, Lane Perception, Free space Perception and Object Perception.
Figure 18 describes that the SVS is composed of a single instance of Camera Image Acquisition Module, one to two instances of Lane Module for front and rear cameras, one or more instances of Free space Module depending on the number of cameras connected and a single Object Perception Subsystem. The Object Perception module encapsulates a single object detector, one or many vehicle trackers and Tracked object managers for each camera.

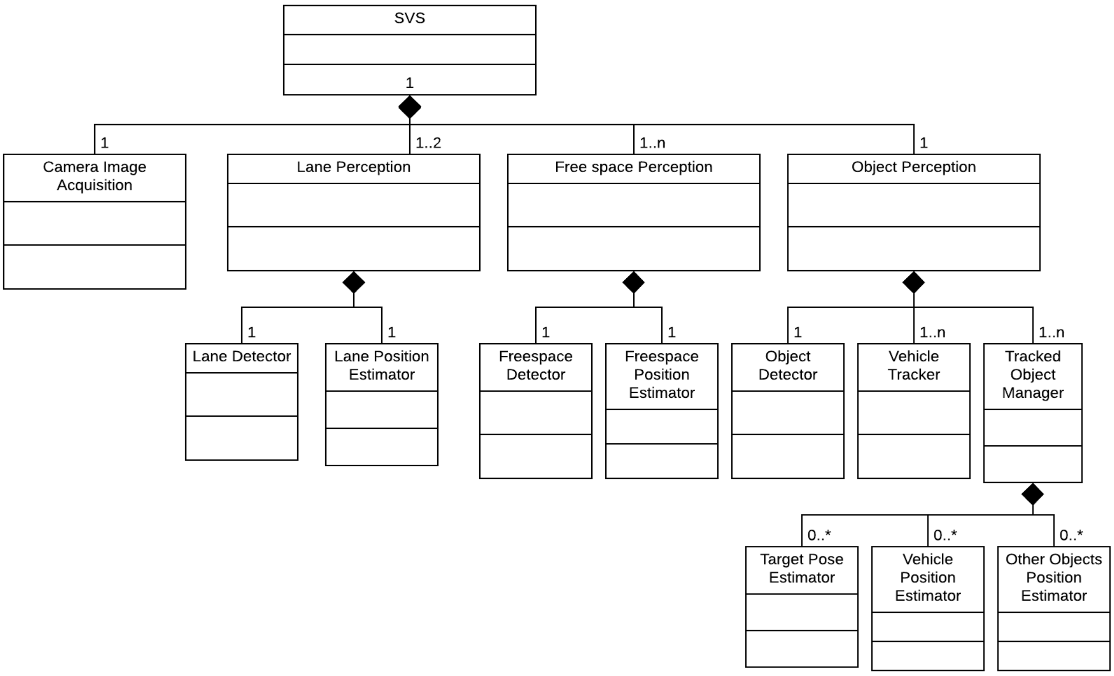

SVS
1
1
1..2
1..n
1
Camera Image Acquisition
Lane Perception
Free space Perception
Object Perception
1
1
1
1
1
1..n
1..n
Lane Detector
Lane Position Estimator
Freespace Detector
Freespace Position Estimator
Object Detector
Vehicle Tracker
Tracked Object Manager
0..*
0..*
0..*
Target Pose Estimator
Vehicle Position Estimator
Other Objects Position Estimator

Figure 18: Block Definition Diagram (BDD) representing the composition of SVS

In Section 3.5, the external interfaces of the system were described. In this section, the interfaces of each module are identified, along with the type of information exchanged. The following views will form a base for the design of these modules.

Figure 19 describes the internal block diagram of the SVS.

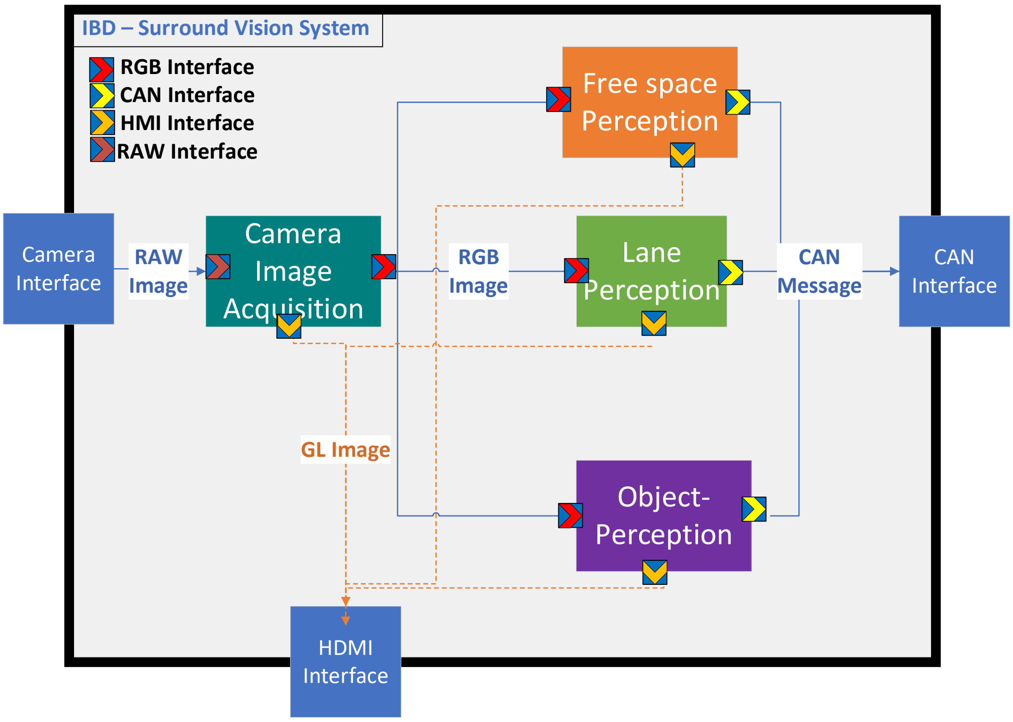

IBD – Surround Vision System
RGB Interface
CAN Interface
HMI Interface
RAW Interface
Free space Perception
Camera Interface
RAW Image
Camera Image Acquisition
RGB Image
Lane Perception
CAN Message
CAN Interface
GL Image
Object-Perception
HDMI Interface

Figure 19: Internal Block Diagram (IBD) of Surround Vision System.

The camera image acquisition module accepts RAW images, performs image processing and provides the information for the Lane, Free space & Object Perception modules. It also processes the images for rendering, in GL format (image format specific for rendering) and provides it at the HDMI interface. Every module renders its computed information and provides computed CAN messages to the CAN interface.

Figure 20 shows the IBD of Object Perception system along with its interactions.

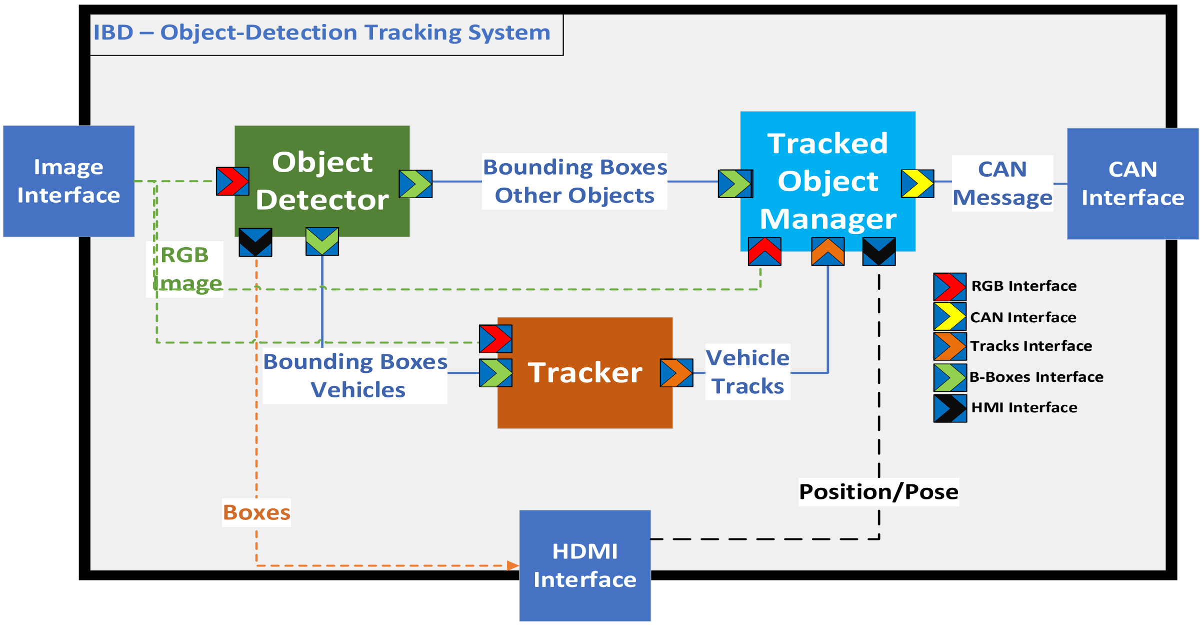

IBD – Object-Detection Tracking System
Image Interface
Object Detector
Bounding Boxes Other Objects
Tracked Object Manager
CAN Message
CAN Interface
RGB Image
RGB Interface
CAN Interface
Tracks Interface
B-Boxes Interface
HMI Interface
Bounding Boxes Vehicles
Tracker
Vehicle Tracks
Position/Pose
Boxes
HDMI Interface

Figure 20: Internal Block Diagram (IBD) of Object Perception Module

Each module has access to the image interface for its performing computations. The Object Detector communicates with the Tracker and the position Estimator by providing bounding box information of vehicles and other objects respectively. The Tracker provides the vehicle track information to the Tracked Object Manager to estimates the positions (other objects non-targets) & pose (for targets).

■■■

5. Module Design

In the previous chapter, the Surround Vision System (SVS) was decomposed into individual functional elements based on the functions desired in Chapter 3. In this chapter, the focus is laid on the detailed design of individual modules, i.e. How are the outputs computed based on the given inputs and what are the algorithms used to perform these functions? This chapter emphasizes the theory and working principles related to Deep Neural Networks and Computer Vision Techniques.

As discussed in the previous chapter, raw images from the cameras are processed and fed to different perception modules for perceiving the information of the surrounding. Section 5.1 elaborates on the image acquisition process and processing of these raw images. Section 5.2 is related to object perception where objects belonging to different classes are perceived and their positions computed. Similarly, Section 5.3 & 5.4 discusses on detecting lanes and free space, respectively.

5.1 Camera Image Acquisition (CIA)

The CIA module as derived from Section 4.1 is responsible for communicating with all the cameras connected to the system, processing and providing the processed images to other detector modules. As per the user, the number of cameras to be connected and their locations are not predefined. The user expects the system should be flexible such that the number of cameras connected and their positions be changed. Based on this requirement, a rig-configuration file is defined that describes the number of cameras connected along with the intrinsic and extrinsic parameters representing the six degrees of freedom (6 DoF) pose of each camera with respect to the vehicle. More information on the rig & camera coordinate systems is described in appendix B.

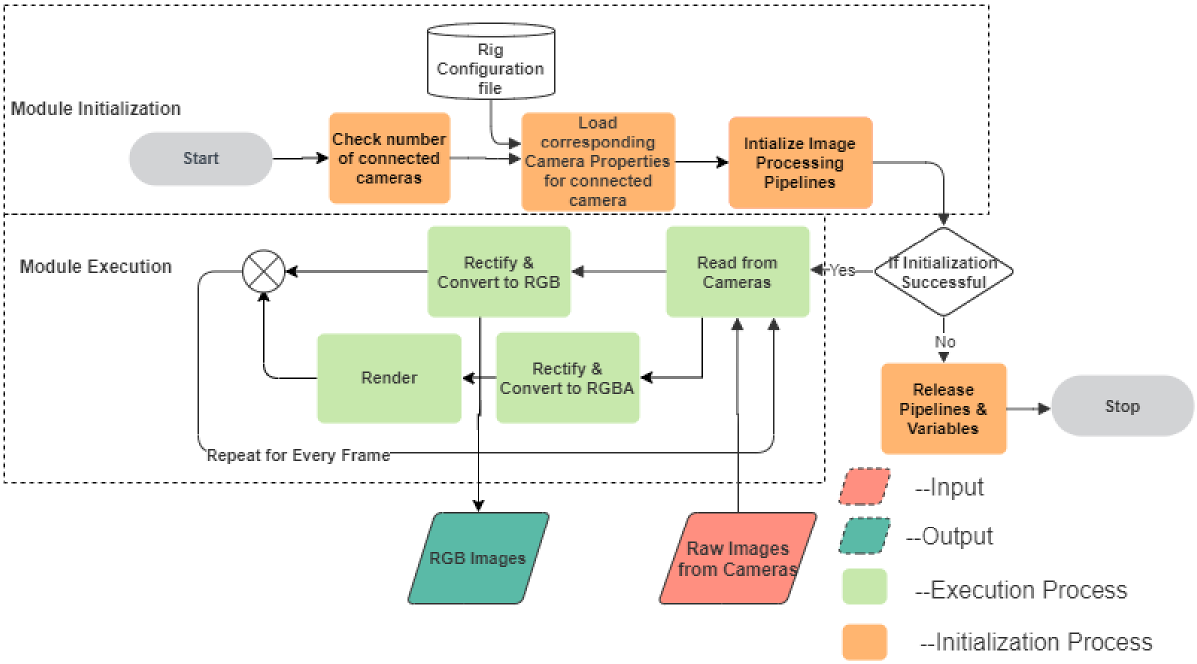

Module Initialization
Rig Configuration file
Start
Check number of connected cameras
Load corresponding Camera Properties for connected camera
Intialize Image Processing Pipelines
Module Execution
Rectify & Convert to RGB
Read from Cameras
Yes
If Initialization Successful
No
Render
Rectify & Convert to RGBA
Release Pipelines & Variables
Stop
Repeat for Every Frame
RGB Images
Raw Images from Cameras
--Input
--Output
--Execution Process
--Initialization Process

Figure 21: Camera Image Acquisition Flow-chart

Figure 21 describes the workflow of the CIA module. During initialization, the module loads the camera parameters from the rig configuration file. Based on the number of cameras connected, image pipelines are created for processing RAW image data to RGB & RGBA formats. Once the module is successfully initialized, images from all cameras are captured synchronously and converted to the above formats. Each image is then rectified to correct the lens distortion effects. The RGB format is compatible with all deep neural networks, while the RGBA format is suitable for rendering the output images. This module iteratively processes and provides the images for other modules. In case the initialization fails, all previously initialized variables are released, errors are displayed, and the system stops.

5.2 *Object Perception*

The object perception module is responsible for providing semantic information of the ego vehicle's surroundings. This section will focus on analyzing possible design solutions that can meet the customer's desired objectives within the given constraints.

The main objective of the customer is to operate the ego vehicle in autonomous as well as cooperative driving modes. Driving in the autonomous mode requires perceiving surrounding objects, while in the cooperative mode, in addition to perceiving, the ego vehicle also needs to identify potential target vehicles to be followed and estimate the pose of these target vehicles. These all functions need to be carried out with low latency and be robust to environmental conditions. Moreover, the solution needs to be universal so that it performs well in different possible scenarios. On the constraints side, the design choices need to be compatible with the existing hardware and are limited by the amount of hardware computational memory. From the project management point of view, the constraints are limited project time and human resources.
Based on the above requirements and constraints, the following possible design solutions were analyzed

Design Choice 1: Training a Custom Object Detector.

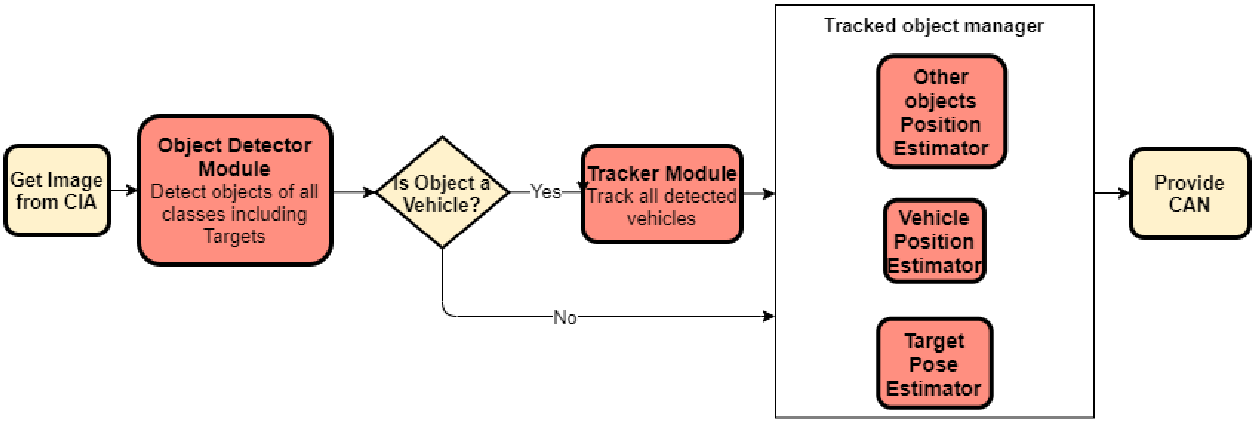


Figure 22: Object Perception: Design Choice 1

The first choice, as shown in Figure 22, involves training a custom object detector that can detect objects of all classes, including target vehicles in one single step given the input image frames from the CIA. The outputs from this step are bounding boxes for each object class. Tracking is performed only for vehicle objects due to memory limitations. The tracker accepts these bounding boxes and is responsible for providing a unique instance ID for each object in the real world. It performs this task by clustering bounding boxes over the next set of consecutive frames. Thus, in this way, only objects that are stable throughout their lifetime are processed further.

The Tracked Object Manager consists of additional submodules which estimate the pose of the target vehicles and position of other vehicles using the tracked bounding boxes. For other objects, the position is estimated only using the bounding boxes provided by the object Detector. The information computed by the Tracked Object Manager is finally provided to the CAN Interface.

Although this solution looks simple and straight forward, the task related to training a new object detector is quite cumbersome. Training involves capturing a large number (7000-10000 instances) of images for all object classes in different scenarios. Labelling an object detector involves a cumbersome process of drawing a bounding box over the object in the image as well as labelling its class. This activity requires additional human resources, training and installing the necessary software packages.

Another aspect is the object detector's network architecture. A deep neural network (DNN) is composed of several layers which perform different mathematical operations. For deployment, these layers are optimized to have low inferencing time using a tool called TensorRT specifically provided by NVidia for DriveWorks. These layers must be compatible with the provided TensorRT tool to integrate with the DriveWork's APIs. This constraint limits the number of possible network choices. A detailed study was performed to evaluate which networks are compatible with the Tensor RT 4.0 tool provided for DriveWorks 1.2 (Refer Appendix B for compatibility chart).

Design Choice 2: Existing Object Detector with a Target Vehicle Classifier

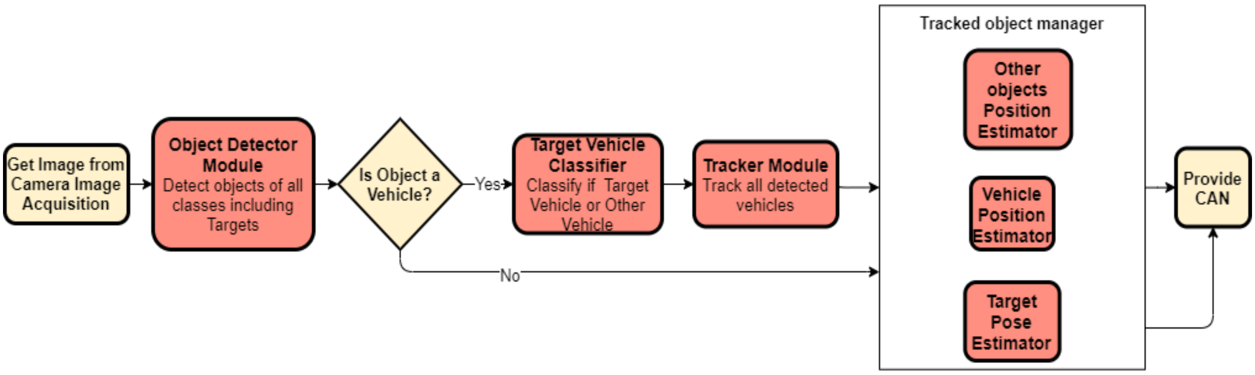

Get Image from Camera Image Acquisition
Object Detector Module
Detect objects of all classes including Targets
Is Object a Vehicle?
Yes
No
Target Vehicle Classifier
Classify if Target Vehicle or Other Vehicle
Tracker Module
Track all detected vehicles
Tracked object manager
Other objects Position Estimator
Vehicle Position Estimator
Target Pose Estimator
Provide CAN

Figure 23: Object Perception - Design Choice 2

Design choice 2 as shown in Figure 23, involves using an existing trained detector "DriveNet" provided by NVidia for detecting vehicles and a classifier for classifying the vehicle detections as targets or non-targets.
Classifier training is only a binary problem related to labelling only two classes (Target Vehicle & non-Target Vehicle) as compared to training an Object detector. Moreover, the performance of DriveNet is quite robust and used commercially. It provides detections for five object classes without much effort, i.e.(Cars, Pedestrians, Bicycles, Traffic-signs, Traffic-lights). It is quite challenging to reach the performance of DriveNet by training a new object detector.
In relation to execution speed, it is observed that during high traffic scenarios due to multiple detections in an image the entire pipeline is significantly slower than design choice 1 due to looping over each detected vehicle in every frame.
Scaling the number of classifiers (3-4 instances) to execute in parallel is one possibility. However, since the computational resources available on the GPU are limited, it is observed that each classifier does not execute independently, although they are implemented to run independently on separate threads and CUDA streams. Not all operations (kernels) belonging to each network run in parallel and independently.

Design Choice 3: Exiting Object Detector with a Tracked Target Vehicle Classifier

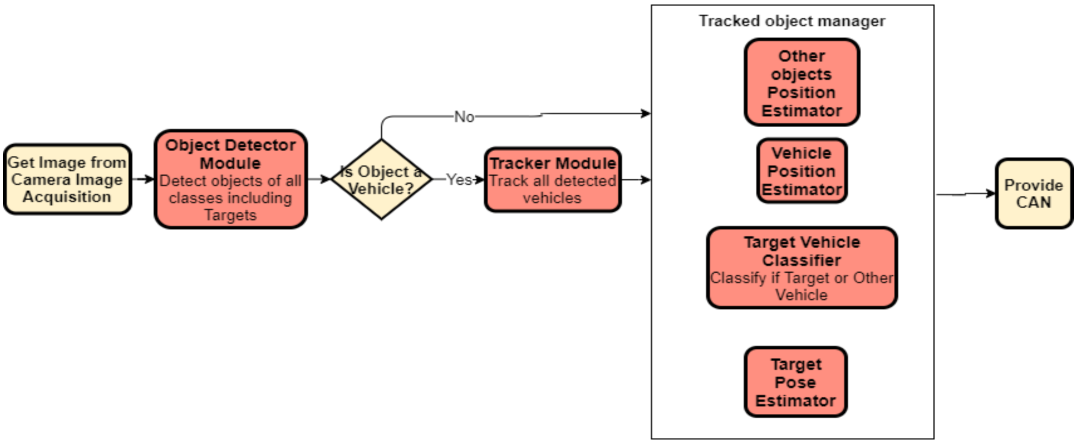

Get Image from Camera Image Acquisition
Object Detector Module
Detect objects of all classes including Targets
Is Object a Vehicle?
No
Yes
Tracker Module
Track all detected vehicles
Tracked object manager
Other objects Position Estimator
Vehicle Position Estimator
Target Vehicle Classifier
Classify if Target or Other Vehicle
Target Pose Estimator
Provide CAN

Figure 24: Object Perception - Design Choice 3

In this choice, as shown in Figure 24, the classifier is placed after the tracker. The objective is to perform classification on only tracked vehicle objects based on the distance or the age of the vehicle track. Thus, this design drastically reduces the number of classification operations required, especially in scenarios involving platooning where vehicles have a longer lifetime around the ego vehicle.

The sequence diagram shown below in Figure 25 describes the interactions between the modules for the object perception module for the front camera.

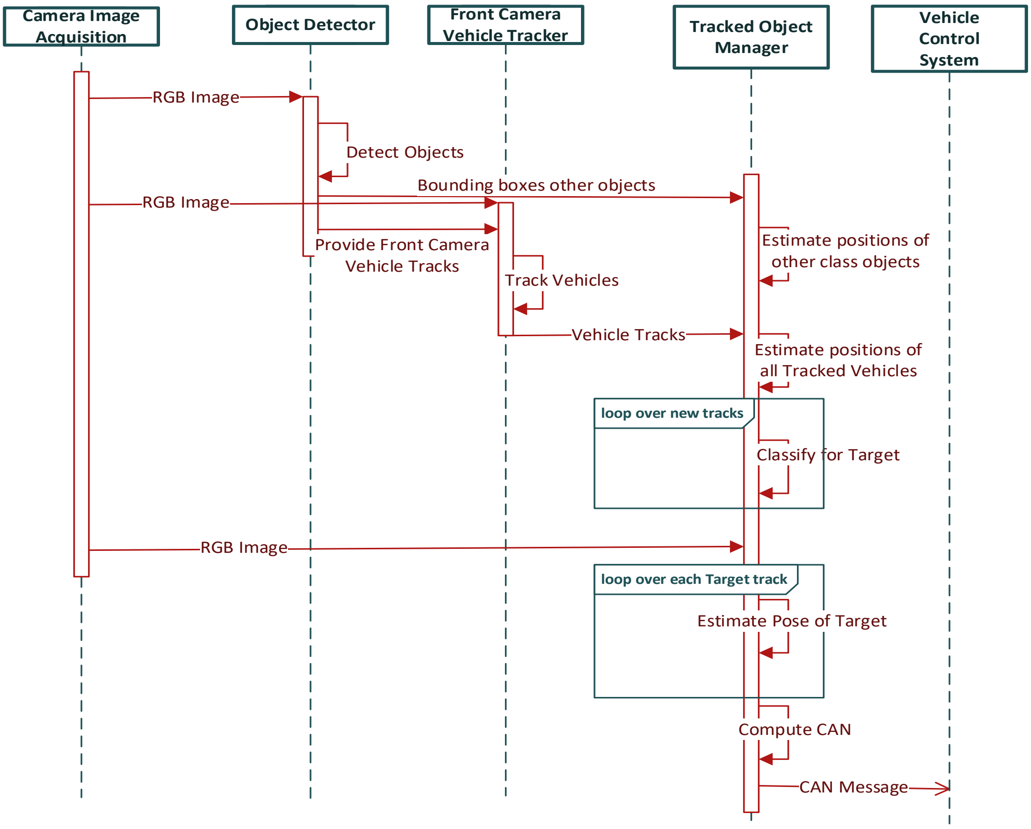

Camera Image Acquisition
Object Detector
Front Camera Vehicle Tracker
Tracked Object Manager
Vehicle Control System
RGB Image
Detect Objects
Bounding boxes other objects
RGB Image
Provide Front Camera Vehicle Tracks
Estimate positions of other class objects
Track Vehicles
Vehicle Tracks
Estimate positions of all Tracked Vehicles
loop over new tracks
Classify for Target
RGB Image
loop over each Target track
Estimate Pose of Target
Compute CAN
CAN Message

Figure 25: Object Perception for Front Camera

Object Detection is performed on each image to get bounding boxes for each object. The boxes are tracked for consecutive images, and vehicle tracks are provided to the Tracked Object Manager. The Tracked Object Manager first estimates the position for all vehicle tracks. All new vehicle tracks are first classified for targets and then pose estimation is carried out for only target vehicles. Similar behaviour is implemented for all cameras.

5.2.1. Object Detector

Object Detection involves the following steps:

- Creating regions of interest by dividing the input image into various regions.
- Passing all regions into a Convolutional Neural Network to obtain confidence values for each class and bounding box proposals for each region.
- Combining overlapping bounding box proposals from all regions to get a single bounding box describing the location of that object in the image.

The object detector module workflow is shown in Figure 26, the module accepts an array/batch of images, synchronously from the CIA, to infer and output detection lists of bounding boxes per camera per object class. Inputs and outputs to the system are represented in red and dark green colours, respectively. Inorder to separate high & low latency operations, the detector is broken into two phases, module initialization(in orange) and module execution(shown in light green).

The module loads the optimized TensorRT Model of the DNN in GPU memory. Data conditioning involves creating a software pipeline to resize the images to the input size of the network size. In this step, memory is also allocated for all bounding box lists. On successful initialization, the module accepts images from the CIA and executes iteratively. In case of failure, an error is displayed on the console describing the reason for failure, all initialized variables are released, and the program terminates.
During execution, the module iteratively accepts an RGB image array of size N(belonging to N-cameras) and performs inferencing in batch mode to output a list of bounding boxes belonging to "M" classes and "N" camera. As per the requirements in this project, five classes were decided, which were the following: i) vehicles, ii) pedestrians, iii) Bicycles, iv) Traffic Signs v) Traffic Lights.

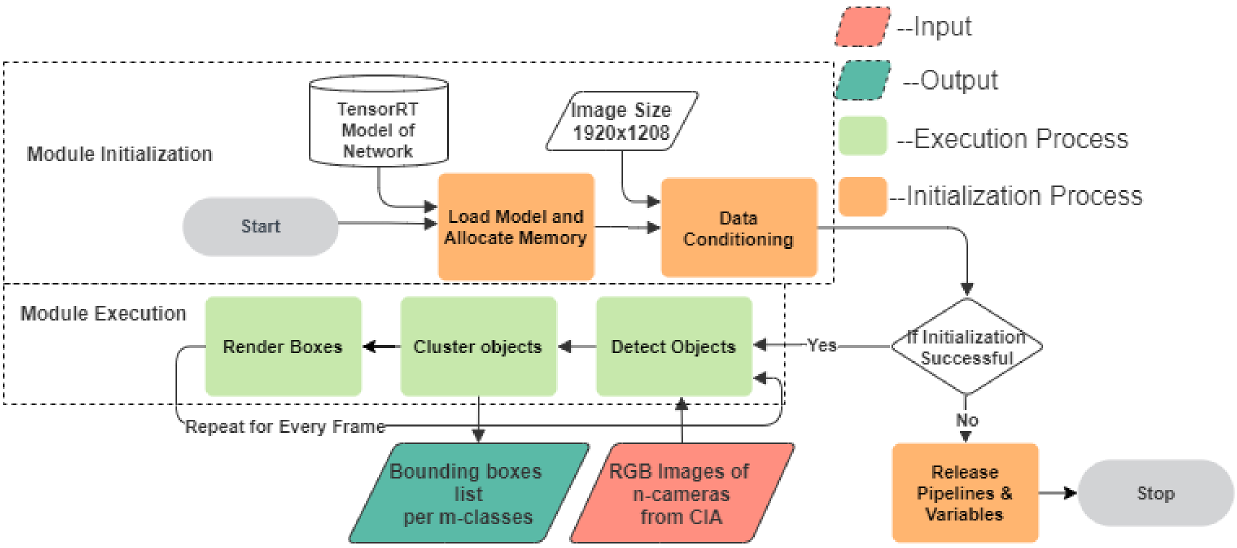

--Input
--Output
--Execution Process
--Initialization Process
Module Initialization
TensorRT Model of Network
Image Size 1920x1208
Start
Load Model and Allocate Memory
Data Conditioning
Module Execution
Render Boxes
Cluster objects
Detect Objects
Yes
If Initialization Successful
No
Repeat for Every Frame
Bounding boxes list per m-classes
RGB Images of n-cameras from CIA
Release Pipelines & Variables
Stop

Figure 26: Object Detector Flow Chart

5.2.2. Vehicle Tracker

The need for a vehicle tracker is to filter out false detections provided by the object detector. The concept of Multiple Object Tracking is used wherein the goal is to maintain the identities of individual objects across several video frames, yielding their individual trajectories. Feng Liu, [6] proposed an algorithm that uses the bounding box information and the features within the bounding box. By checking for the presence of the vehicle bounding boxes in consecutive frames, the system is more confident that the detection is an actual vehicle of interest.

The process involves two steps, as shown in Figure 27:

- Track Creation and Merging
- Track maintenance

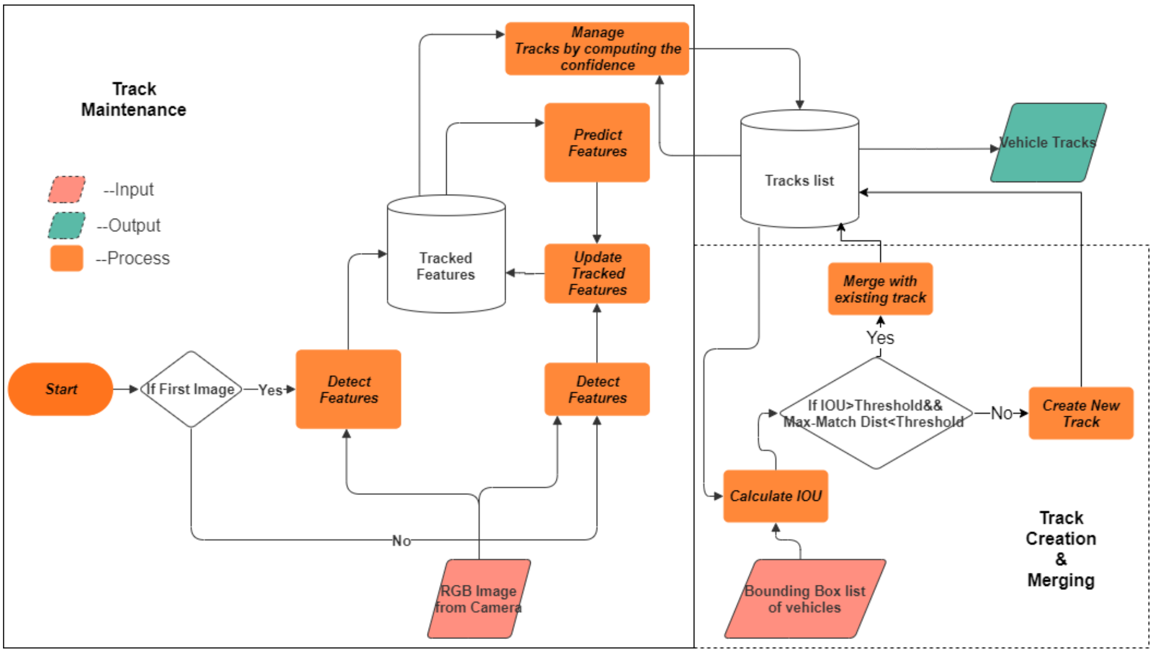

Track Maintenance
--Input
--Output
--Process
Manage Tracks by computing the confidence
Predict Features
Tracked Features
Update Tracked Features
Start
If First Image
Yes
Detect Features
Detect Features
No
RGB Image from Camera
Tracks list
Vehicle Tracks
Merge with existing track
Yes
If IOU>Threshold&& Max-Match Dist<Threshold
No
Create New Track
Calculate IOU
Bounding Box list of vehicles
Track Creation & Merging

Figure 27: Vehicle Tracking Algorithm

5.2.2.1 Track Creation and Merging

In this step, bounding box detections from the previous frame and current frame are merged so that a final list of detections (vehicle tracks) for the current frame are obtained. The merging takes place in such a way that if a new detection (bounding box) refers to an object that has been tracked from the previous frame, this new detection is merged into that object, and the object is then assumed to be "detected again". If a new detection refers to an entirely new object, it is added to a tracklist for future checks. Two criteria are followed for performing the merging to decide whether a new detection refers to the same object as one of the tracked objects:

- Intersection over Union Threshold: If the Intersection over Union (IOU) is higher than a threshold, the two bounding boxes (Tracked bounding box from previous frame and present detected box) belong to the same object.
- Maximum Match Distance Threshold: Maximum Match Distance is defined as 1-IOU. Within this threshold, the box with the most extended track history is preferred and is selected.

Track Maintenance

This step involves estimating the certainty or confidence value of the tracks. This is performed by detecting features (edges or corners of the tracked object in the image) present in bounding boxes of consecutive frames. A confidence decay rate is calculated based on the number of tracked features present for a track. A tracked feature is defined as an edge or corner of an object present in consecutive images. As the number of tracked features drop, the confidence decay rate increases. If the confidence of a track falls below a threshold, the track is discarded.

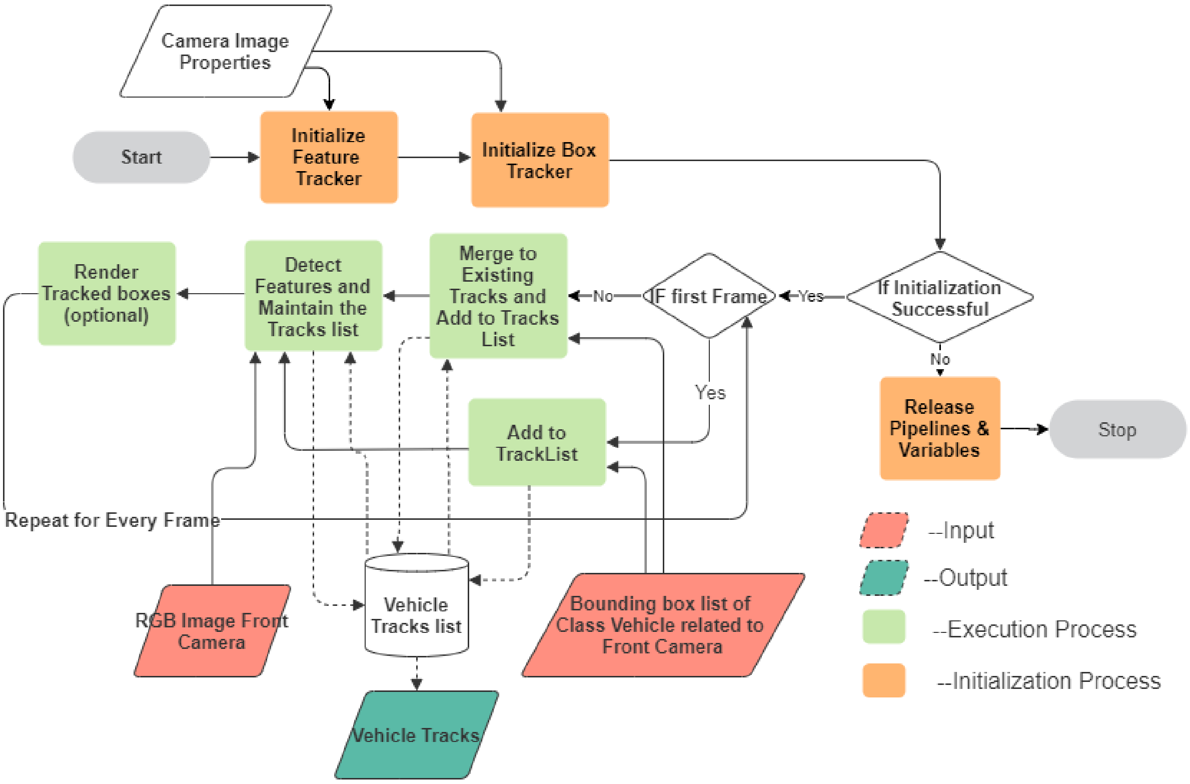

Camera Image Properties
Start
Initialize Feature Tracker
Initialize Box Tracker
If Initialization Successful
Yes
No
IF first Frame
No
Yes
Merge to Existing Tracks and Add to Tracks List
Detect Features and Maintain the Tracks list
Render Tracked boxes (optional)
Add to TrackList
Release Pipelines & Variables
Stop
Repeat for Every Frame
RGB Image Front Camera
Vehicle Tracks list
Bounding box list of Class Vehicle related to Front Camera
Vehicle Tracks
--Input
--Output
--Execution Process
--Initialization Process

Figure 28: Vehicle Tracker (Front) Workflow

The workflow of the front camera tracker is described in Figure 28 for illustration. During initialization, memory is allocated for storing the tracks and features. On successful initialization, bounding boxlist of the vehicle class and an RGB image belonging to the front camera is accepted. The boxes are merged with the existing tracks, and track maintenance is done by detecting and calculating the tracked features as explained in the algorithm. This process is repeated for every new frame.

5.2.3. Tracked Object Manager

As analyzed in requirements Section 3.4, the goal is to estimate the pose of target vehicles (Twizy), position of non-target vehicles and position of objects belonging to other classes (Pedestrians and Bicycles). For target vehicles, additional information about the dimensions of the vehicle and the shape are known, thus making it feasible to estimate its pose.

The Tracked Object Manager Module performs four functions which are as follows:

- Estimate the position of other classes (Pedestrian, Bicycles)
- Estimate the 2D position of all vehicles (Targets as well non-Target Vehicles)
- Classify whether a vehicle is a target vehicle or not.

- Estimate the pose of all target vehicles.

Figure 29 shows the flow-chart of the Tracked Object Manager Module, consisting of the functions in grey, the inputs described in red and outputs in green colours.

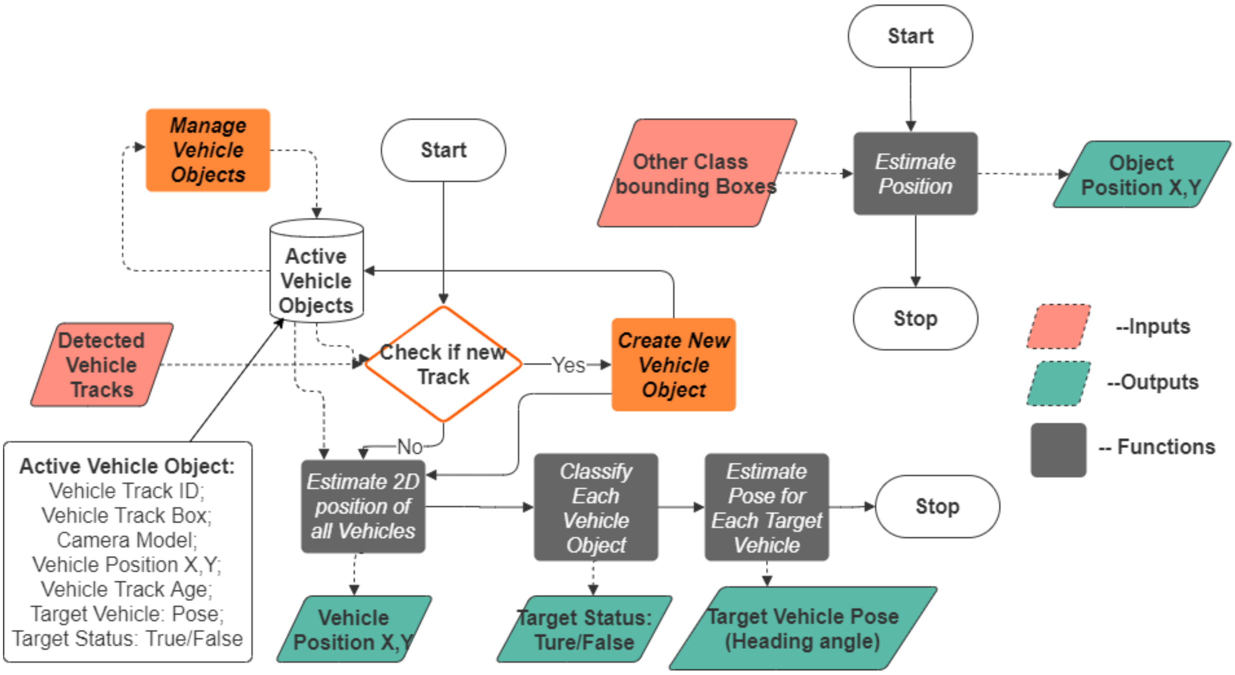

Start
Estimate Position
Other Class bounding Boxes
Object Position X,Y
Stop
Manage Vehicle Objects
Start
Active Vehicle Objects
Check if new Track
Yes
Create New Vehicle Object
No
Detected Vehicle Tracks
--Inputs
--Outputs
-- Functions
Active Vehicle Object:
Vehicle Track ID;
Vehicle Track Box;
Camera Model;
Vehicle Position X,Y;
Vehicle Track Age;
Target Vehicle: Pose;
Target Status: True/False
Estimate 2D position of all Vehicles
Classify Each Vehicle Object
Estimate Pose for Each Target Vehicle
Stop
Vehicle Position X,Y
Target Status: Ture/False
Target Vehicle Pose (Heading angle)

Figure 29: Tracked Object manager workflow

The module receives a list of bounding boxes belonging to other classes (bicycle, pedestrian) and vehicle tracks from the vehicle tracker. For other objects, the position is estimated in a single step, while for the vehicle tracks, the module creates, updates and stores the information as 'Active vehicle objects'. Active vehicle objects signify vehicles currently present in the real world around the ego-vehicle. Based on the Track ID, these objects are created and updated in the active vehicle objects list.

If the objects are not updated for n-consecutive frames, then the detected vehicle is assumed to have left the scene, and thus the vehicle object is destroyed. For each vehicle object, position estimation is carried out in every frame. Target classification need not be done frequently and may be based on specific rules for e.g. how much the position of the vehicle has changed based on the previous classification. Pose estimation is performed only for those vehicle objects that are classified as a target vehicle.

5.2.3.1 Position Estimation

Model-based estimation techniques and deep learning-based techniques are generally used for estimating the position of objects in images. In this project, the scope is limited to model-based techniques due to limited access to training data necessary for deep-learning and a tight project schedule.
Most of the literature related to model-based techniques use two standard algorithms:

- The width-based algorithm [7]
- The position-based algorithm [8]

The width-based method [7] estimates the absolute distance of a vehicle given the actual width (or height) of the vehicle, the width (or height) in the image plane and the focal length of the camera lens. As shown in Figure 30 (a), the inter-vehicle distance is inversely proportional to the height of the bounding box in the image. In practice, this method is useful if the size (height or width) for a specific vehicle is known beforehand. However, it is not suitable for estimating distances for different vehicles (having different heights or widths). For instance, a Truck will have a wider width and height compared to a car.

Lee, Jaemyoung [9] used another approach of the width-based method by assuming a specific scenario of vehicles driving on a straight highway with lanes. The distance between the lane markings is assumed constant (parallel lines). In Figure 30(b), L1 and L2 are the distances between two lane-markings. d1 and d2 are the distances from the front bumper of the ego vehicle to a predetermined position and the rear bumper of the front vehicle, respectively. Using the similarity of triangles d2 can be calculated. Though this method is robust to noise, it is suitable only for straight highway scenarios with lane marking.

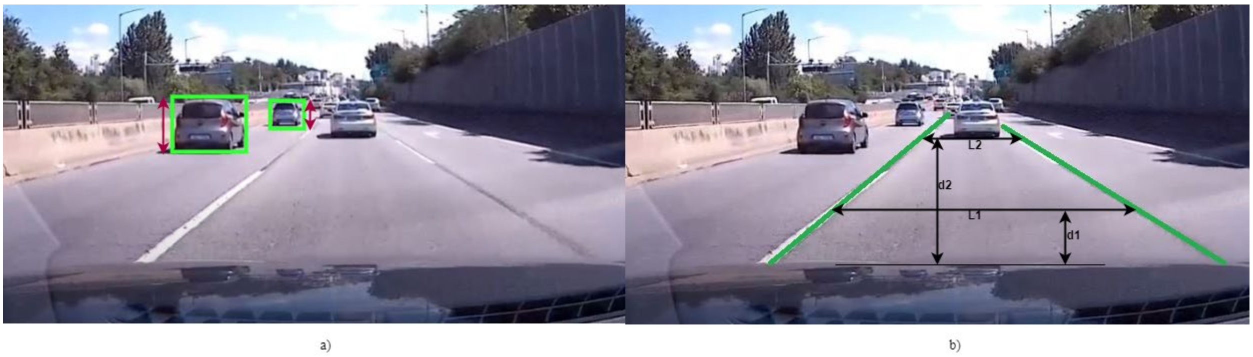


Figure 30: Width Based Methods: a) Intervehicle distance calculation based on bounding box height change b) Intervehicle distance calculation based on lane-markings

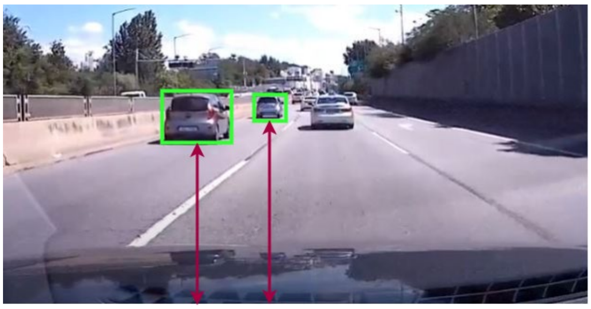


Figure 31: Position-Based Method

In the position-based algorithm, the vertical position of the bounding box from the bottom pixels of the image, as shown in figure 31, is used to calculate the inter-vehicle distance. The distance and vertical position are proportional. However, this method needs to assume the road is planar, even though the slope has nothing to do with it. If this assumption holds, this method is suitable for calculating the position of any type of object (Pedestrian, Bicycle) provided it is on the road. Despite this advantage, this method is susceptible to noise. A small pixel variation in the vertical position of the bounding box can cause significant distance errors, more specifically with objects that are further away than the closer ones. Secondly, for this method, minor deviations in the camera orientation (pitch angle) or height from the ground plane, may cause significant errors for objects that are further away. To overcome this disadvantage, Cho, G.Kim & J. [7] proposed a combination of the width and position-based methods along with a Kalman filter. The project demands a solution that is generalized and is applicable for highway as well as urban driving scenarios. Thus, using the above two methods, along with a Kalman filter, was a preferred choice.

Figure 32, describes the detailed algorithm designed for estimating the position of every vehicle object. A vehicle object contains a Track ID and its bounding box information. Based on this, vehicle coordinate positions X and Y, Kalman filter gains are computed. It is assumed that the bounding box fits the detected vehicle in the image correctly. The bottom pixels of the box signifies the contact points of the vehicle with the ground plane, which are the points of interest for estimating the absolute position.
The position of the vehicle is first calculated with respect to the camera coordinate system using the position estimation method based on the camera intrinsic parameters and the height of the camera from the ground plane. The lateral position(x-position) in camera coordinates is more stable compared to the longitudinal position (z-position). To compensate for measurement noise in the longitudinal position, the width-based method is used to compute the vehicle height given the bounding box height and the longitudinal distance. A Kalman filter, having a motion model as a constant height (to incorporate that the actual height of the vehicle is constant) is used for averaging out the vehicle height. The estimated height is again used to estimate the longitudinal distance. The last step involves the transformation of measurements from the camera to the ego-vehicle coordinate system. By following this process, the vehicle 2D position can be robustly estimated for any agnostic vehicle with low execution time.

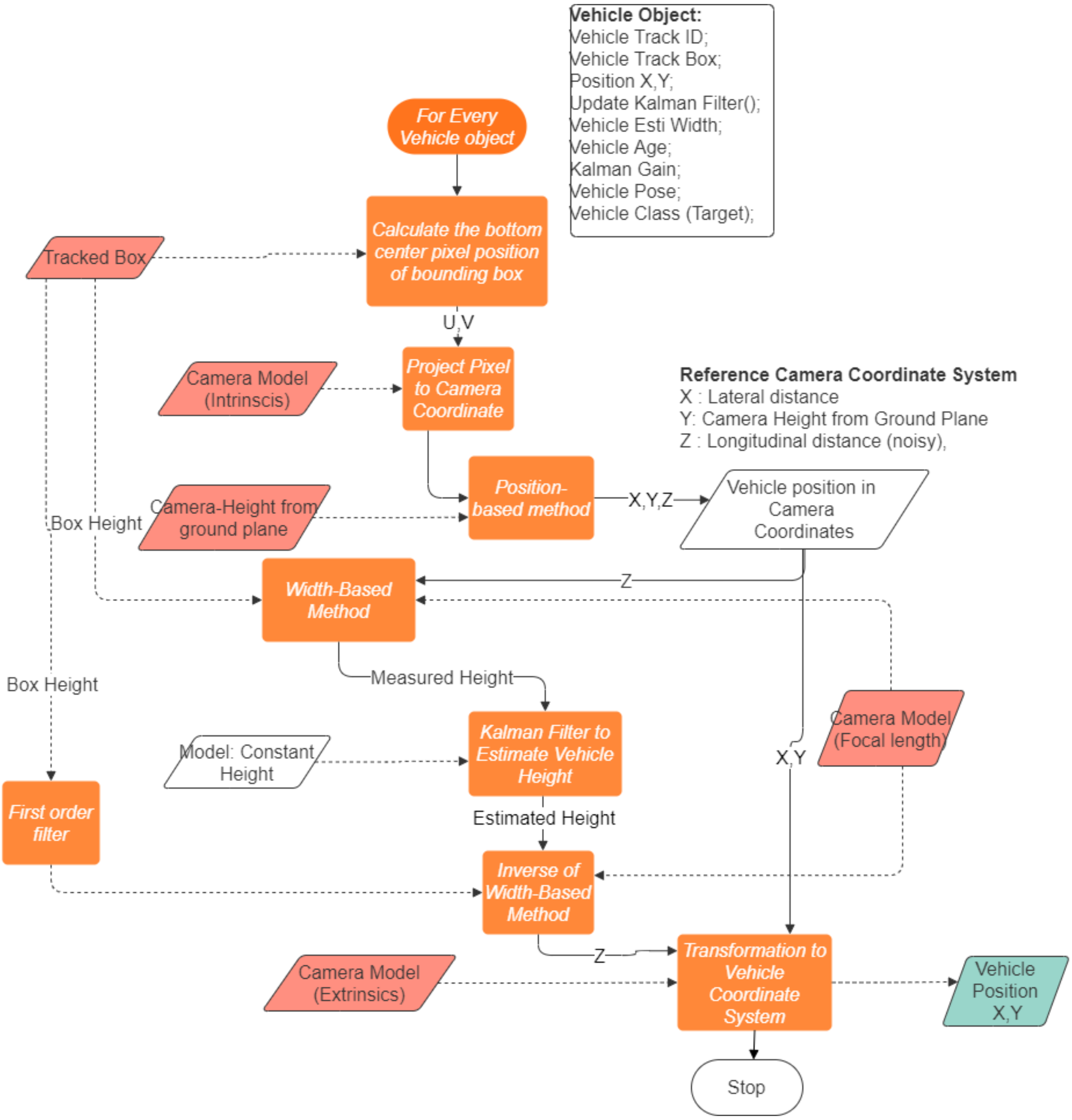

Vehicle Object:
Vehicle Track ID;
Vehicle Track Box;
Position X,Y;
Update Kalman Filter();
Vehicle Esti Width;
Vehicle Age;
Kalman Gain;
Vehicle Pose;
Vehicle Class (Target);
For Every Vehicle object
Calculate the bottom center pixel position of bounding box
Tracked Box
U,V
Camera Model (Intrinscis)
Project Pixel to Camera Coordinate
Reference Camera Coordinate System
X : Lateral distance
Y: Camera Height from Ground Plane
Z : Longitudinal distance (noisy),
Camera-Height from ground plane
Box Height
Position-based method
X,Y,Z
Vehicle position in Camera Coordinates
Z
Width-Based Method
Measured Height
Box Height
Camera Model (Focal length)
X,Y
Model: Constant Height
Kalman Filter to Estimate Vehicle Height
First order filter
Estimated Height
Inverse of Width-Based Method
Z
Camera Model (Extrinsics)
Transformation to Vehicle Coordinate System
Vehicle Position X,Y
Stop

Figure 32: Position Estimator Workflow

5.2.3.2 Target Vehicle Classifier

As classifying objects is computationally intensive, the whole process benefits from performing this task less often, therefore, to classify vehicles efficiently, we are suggesting an approach where the classification happens based on the distance between the target and ego vehicle.

The workflow of the classification module is similar to the object detector. Once the classification model is loaded and successfully initialized bounding boxes from every tracked vehicle object are fed to the classifier. The image is resized to the network dimensions, and inferencing is performed to output a binary result indicating whether the vehicle is a target or not.

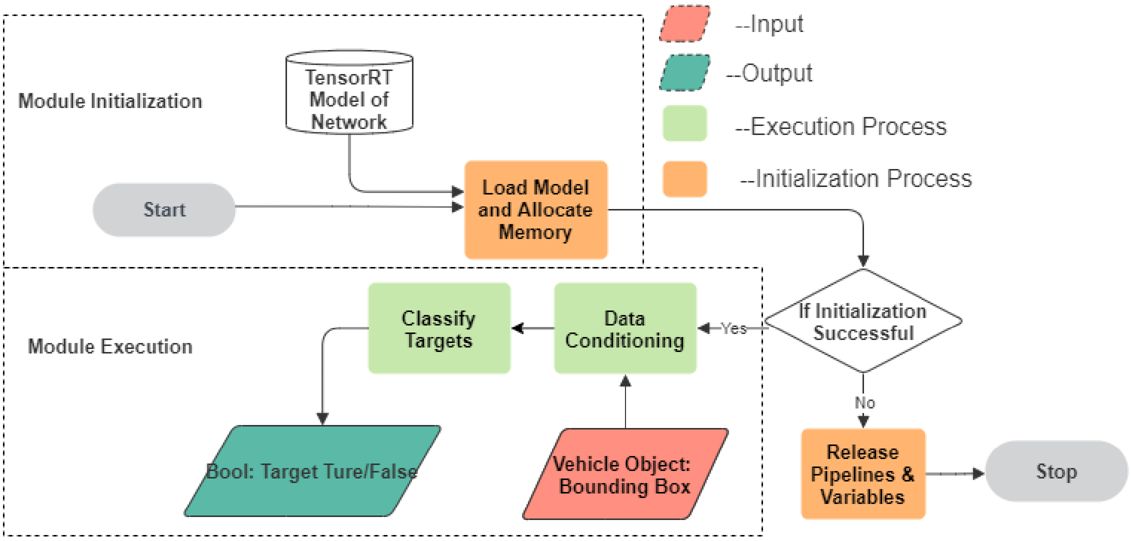

--Input
--Output
--Execution Process
--Initialization Process
Module Initialization
TensorRT Model of Network
Start
Load Model and Allocate Memory
If Initialization Successful
Yes
No
Module Execution
Classify Targets
Data Conditioning
Bool: Target Ture/False
Vehicle Object: Bounding Box
Release Pipelines & Variables
Stop

Figure 33: Target Vehicle Classifier Workflow

5.2.3.3 Target Vehicle Pose Estimation

Once a vehicle object is identified as a target, its pose needs to be estimated with respect to the ego-vehicle. Similar to position estimation described in Section 5.2.3.1, DNNs and conventional computer vision techniques are used depending on the requirements. In relation to DNNs, DeepIM [10], Keypoint detector localization [11] and PVNet [12]are examples of current cutting edge pose estimation methods. However, the challenge for the implementation of such methods requires large amount of training data. In this project, since the target vehicle is a textured object, sufficient features can be extracted, making computer vision techniques a preferred choice.

The process of pose estimation can be divided into two parts as shown in Figure 34, an offline process which involves registering the model of the target that the user wants to track and the second part involving an online process of estimating pose given the registered model.

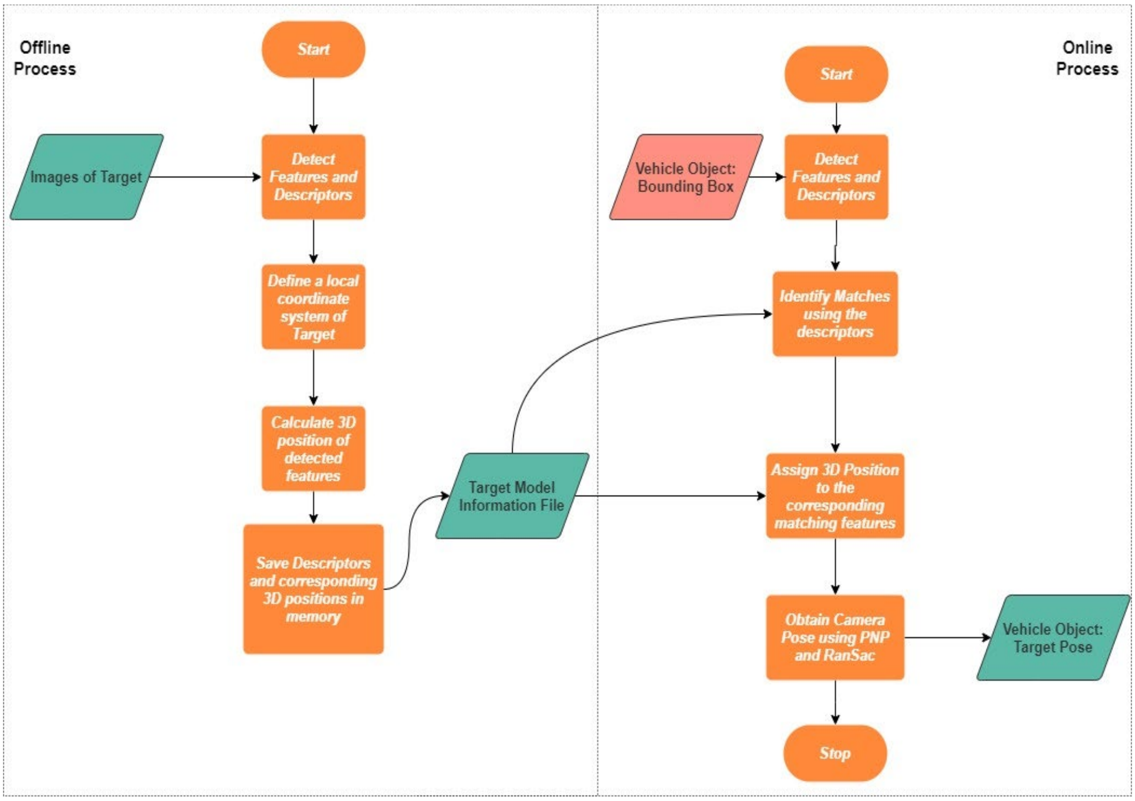

Offline Process
Start
Images of Target
Detect Features and Descriptors
Define a local coordinate system of Target
Calculate 3D position of detected features
Save Descriptors and corresponding 3D positions in memory
Target Model Information File
Online Process
Start
Vehicle Object: Bounding Box
Detect Features and Descriptors
Identify Matches using the descriptors
Assign 3D Position to the corresponding matching features
Obtain Camera Pose using PNP and RanSac
Vehicle Object: Target Pose
Stop

Figure 34: Target Pose Estimator Workflow

The offline model registration process involves the following steps:

- Detect features and descriptors for different images of the target.
- Compute the 3d position of the detected features with respect to a local coordinate system of the target.
- The descriptor information, along with its corresponding 3d position, are written to a file representing a model of the target.

Once a model registration is performed, the generated model is used for online pose estimation.
The following are the steps performed during online pose estimation:

- Features and descriptors are extracted within the bounding box for vehicle objects classified as targets.
- Feature matching is carried with the observed features in the above step and with those in the model.
- For the matched features, the corresponding 3d locations are assigned to the observed features.
- Given the 3d object points, the 2d image points of the features in the scene and the camera intrinsic properties, the pose of the camera can be computed relative to the detected object's local coordinate system.
- The Perspective-n-Point (PnP) function estimates an object pose by finding such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed image points and the projected object points. The use of RANSAC makes the function resistant to outliers.

5.3 Lane Perception

The lane perception module is responsible for perceiving the lanes present in front and rear of the ego vehicle. Figure 35 shows the design flow of the lane perception module. The lane perception module is responsible for the following functions:

- Identify and classify the lane markings such as left adjacent lane, left ego lane, right ego lane, and right adjacent ego lane if they are present on the road.
- Render the detected lane markings in-vehicle display.
- Project the image points to world points with respect to the ego-vehicle coordinate system.
- Fits a polynomial through the projected world points.

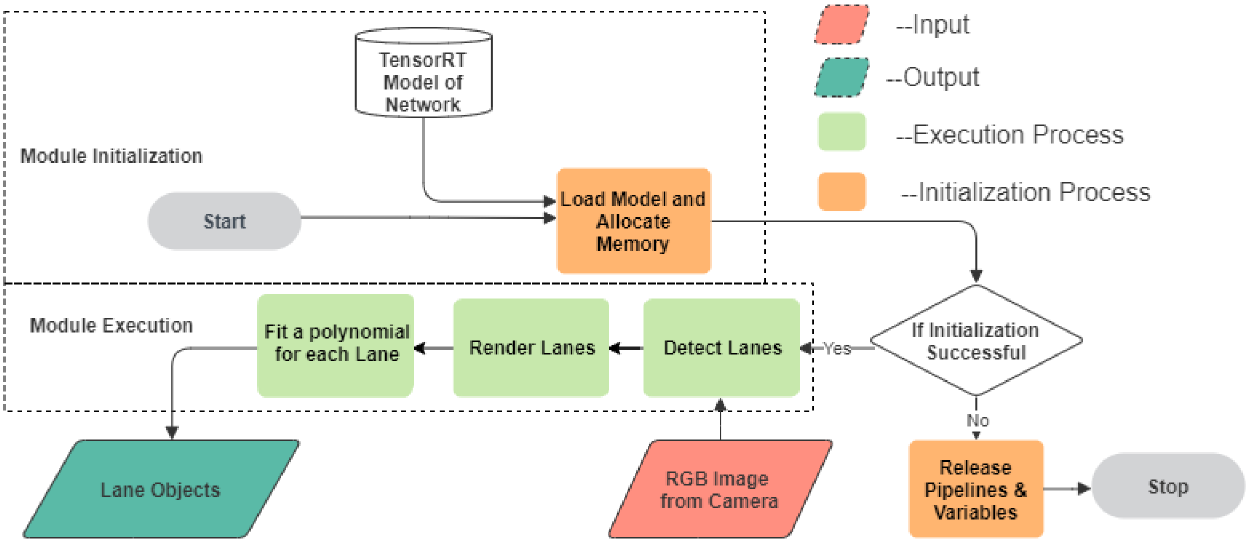

--Input
--Output
--Execution Process
--Initialization Process
TensorRT Model of Network
Module Initialization
Start
Load Model and Allocate Memory
Module Execution
Fit a polynomial for each Lane
Render Lanes
Detect Lanes
Yes
If Initialization Successful
No
Lane Objects
RGB Image from Camera
Release Pipelines & Variables
Stop

Figure 35: Lane Perception Workflow

The module outputs Lane Objects. Each Lane object has information related to the type of lane-marking (solid or dashed), type of lane (Ego lane or Ego adjacent lane) and coefficients of the polynomial. These objects are encoded as CAN messages and provided to the CAN interface.

5.4 *Free Space Perception*

The free-space perception module, as shown in figure 36, estimates the drivable space around the vehicle by computing the free-space boundary from images provided by every camera. The module uses a deep neural network to output image points representing the free-space boundary. These points are projected to world coordinates to calculate the actual free-space boundary around the ego vehicle. The coordinates of each point are provided to the vehicle control system over the CAN bus.

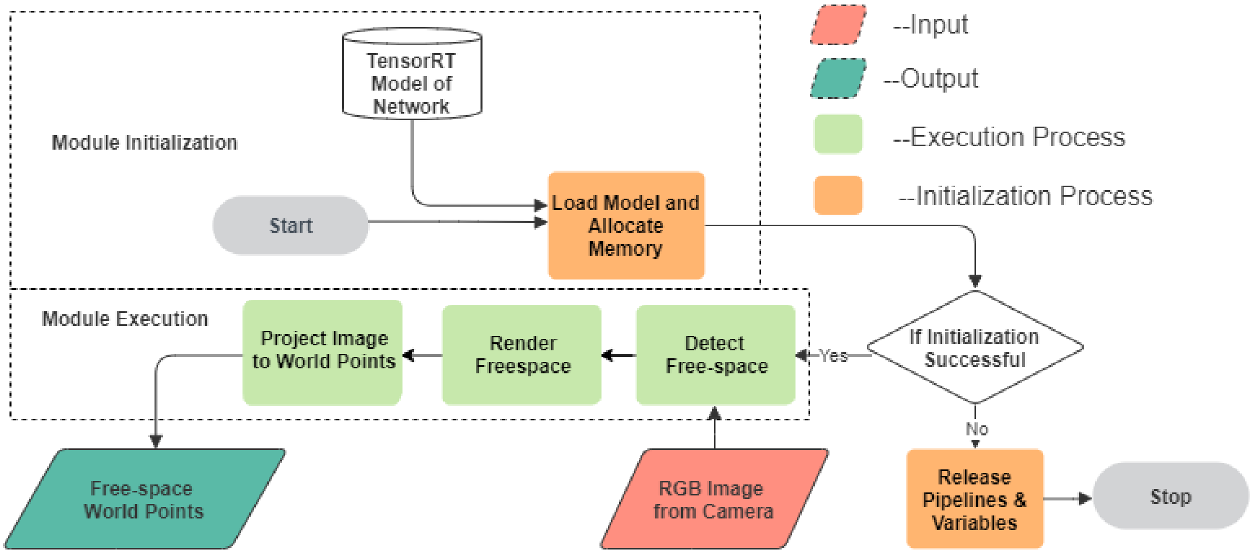

--Input
--Output
--Execution Process
--Initialization Process
Module Initialization
TensorRT Model of Network
Start
Load Model and Allocate Memory
If Initialization Successful
Yes
No
Module Execution
Detect Free-space
Render Freespace
Project Image to World Points
Free-space World Points
RGB Image from Camera
Release Pipelines & Variables
Stop

Figure 36: Free Space Perception Workflow

. ■ ■ ■

6.Implementation

The previous chapter reflected on different design methods that can be used to meet the desired functionality. The design of each module was explained by referring to the inputs of each module and what functions are performed to transform the inputs to desired outputs. This chapter refers to how these designs are finally realized and implemented in software code. In the first section, a brief description is provided about NVidia DriveWorks libraries reflecting on the reasons it is used. In the second section, the different methods used to implement the design are described.

6.1 NVidia Drive Software

The surround vision system (SVS) needs to run in real-time and be deployed on a cooperative automated vehicle. This requires an embedded device with high computational resources and low power consumption. To that effect, the NVidia drive PX 2, an embedded device specifically designed for autonomous driving functions is used. The main benefit this device offers is low power consumption along with desired graphics processing capabilities. The NVidia Drive software is used for developing applications on the Drive PX 2. Figure 37 describes the Drive software stack, which consists of the Drive OS (operating system), the DriveWorks Software Development Kit (SDK), and the high-level NVidia Drive Autonomous Vehicle (AV) library.

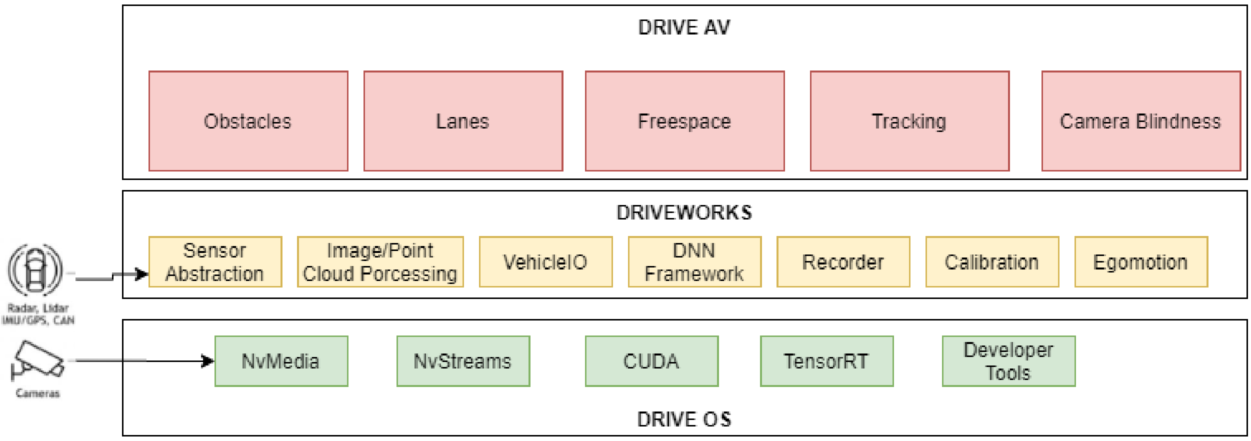


Figure 37:NVidia Drive Software Stack [10]

Drive OS is responsible for communication with the hardware components and connected sensors.
DriveWorks SDK enables developers to implement autonomous solutions by providing comprehensive libraries for sensor information acquisition, deep learning, and rendering.
Drive AV utilizes the DriveWorks SDK to provide high-level functions like Object Perception, Planning, and Mapping Modules. The Perception modules, for example, have deployment-ready neural networks for object detection (DriveNet), lane detection (LaneNet), and free space Detection (FreeSpaceNet).

6.2 Mapping of SVS Architecture to DriveWorks APIs

As discussed in the architecture Chapter 4 & 5, functionalities offered by the NVidia Drive software are mapped to individual modules. This mapping is explained in the following sub-sections:

6.2.1. Camera Image Acquisition

The camera image acquisition (CIA) module uses the NVmedia Interface (mapped as the GSML interface in the architecture) to communicate with the cameras. This interface reads the images from the connected cameras synchronously and provides the RAW images to the CIA module. Sensor abstraction & image processing APIs provided by DriveWorks are used to initialize the module and create an image processing pipeline to process the RAW images, respectively. Every image from each camera is then processed sequentially by passing through this pipeline to output RGB and RGBA formats as in figure 38. After processing, each image is rectified with its corresponding camera intrinsic parameters.

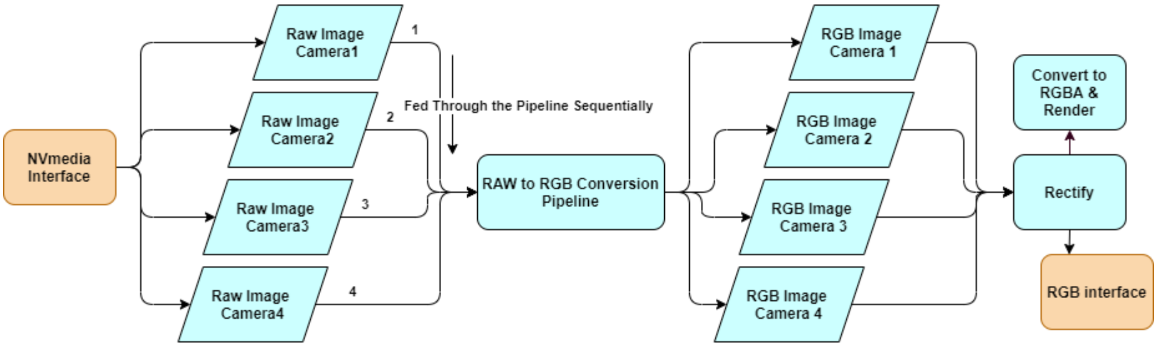

NVmedia Interface
Raw Image Camera1
Raw Image Camera2
Raw Image Camera3
Raw Image Camera4
1
2
3
4
Fed Through the Pipeline Sequentially
RAW to RGB Conversion Pipeline
RGB Image Camera 1
RGB Image Camera 2
RGB Image Camera 3
RGB Image Camera 4
Convert to RGBA & Render
Rectify
RGB interface

Figure 38: Image Processing Pipeline

6.2.2 Deep Neural Networks

NVidia Deep neural networks (DNN) are used for object detection, classification, lane detection, and free space detection. NVidia DriveWorks provides two options for implementation. The first option involves a DNN API which allows importing a TensorRT model of a custom neural network. The second option involves using NVidia proprietary neural networks like DriveNet, LaneNet & FreespaceNet for object, lane and free space detection, respectively. Since we are concerned with deployment and working with the outputs of the detector, we use the second option to perform all detection related activities.

6.2.2.1 Object Detector

As mentioned in the previous chapter, we use DriveNet object detector that operates in batch mode. It accepts an array of images and provides object proposals (in the form of bounding boxes) belonging to five classes. The objects detected per class per camera are saved in separate containers. For more details, please refer to NVidia DriveWorks Documentation [13].

6.2.2.2 Object/Target Vehicle Tracker

We use the DriveWorks Tracking API to perform vehicle Tracking. The API provides tunable parameters for setting the number of features for tracking and threshold parameters for clustering the bounding boxes between frames. Vehicles proposals detected by the object detector are fed to the trackers to output vehicle tracks. For more details refer NVidia DriveWorks Documentation [13].

6.2.2.3 Target Vehicle Classifier

A Renault Twizy is used as the target vehicle in this project. Figure 39 describes the approach taken for training the Target Vehicle Classifier.

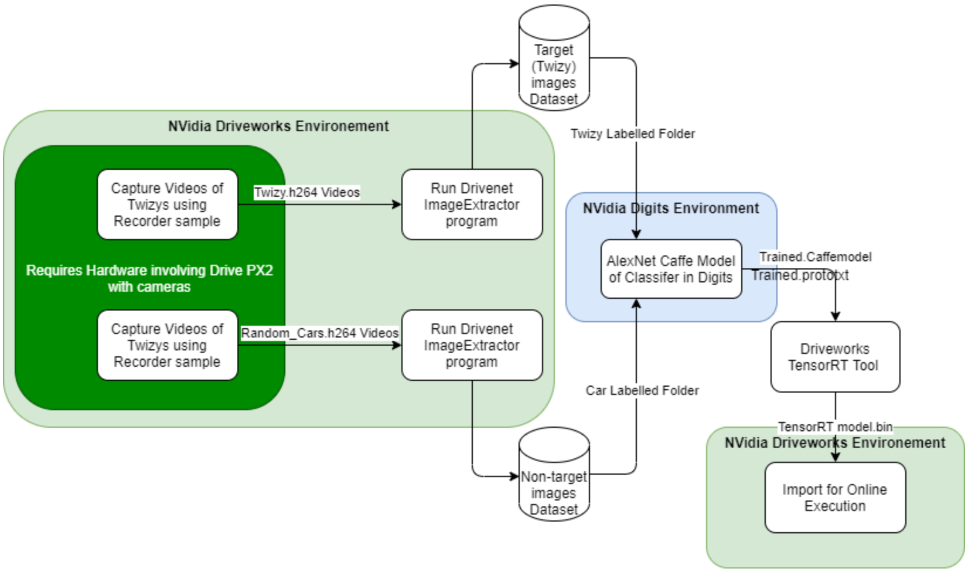

Target (Twizy) images Dataset
NVidia Driveworks Environement
Capture Videos of Twizys using Recorder sample
Twizy.h264 Videos
Run Drivenet ImageExtractor program
Requires Hardware involving Drive PX2 with cameras
Capture Videos of Twizys using Recorder sample
Random_Cars.h264 Videos
Run Drivenet ImageExtractor program
Twizy Labelled Folder
NVidia Digits Environment
AlexNet Caffe Model of Classifer in Digits
Trained.Caffemodel
Trained.prototxt
Driveworks TensorRT Tool
Car Labelled Folder
Non-target images Dataset
TensorRT model.bin
NVidia Driveworks Environement
Import for Online Execution

Figure 39: Pipeline for Training and Deployment of Classifier

For generating the target vehicle dataset, a Renault Twizy is driven around the ego vehicle at different distances from the cameras, and h264 videos are recorded. Inferencing is performed on these videos using the DriveNet Object Detector, and the detected vehicle images are saved to disk. A similar process is followed for generating the dataset for Non-target vehicles by driving on the highway and parking lots.

Objects that are further away from the ego vehicle have a smaller bounding box. These images, when saved and resized to network dimensions, appear pixelated and grainy. Using such images for training makes it difficult for the classifier to learn from such data and thus leads to poor classification performance. A solution to this problem is to classify boxes having a minimum size enough to incapsulate visibly distinct features for training. Thus, classifier training is performed only on vehicles within 30m distances where the size of the boxes is sufficiently large enough.

For training the classifier, a Docker image of NVidia Digits is used. Digits provide an interactive interface to choose from different existing network architectures and train for classification and object detection. We use a pre-trained model of GoogleNet (trained on the ImageNet dataset). The input image size of the network is 256x256, and the provided images are stretched and resized to the input dimensions for training. The two nodes at the output layers provide the probability to which class the object belongs. The output from Digits is the network model and a model description file describing the network layers. These two files are imported to the TensorRT tool of DriveWorks for inference optimization. The TensorRT model generated by the tool is then used for online inferencing.

During execution, the bounding boxes of vehicle class provided by object vehicle tracker are resized to input network dimension, the probabilities provided at the output nodes are compared, and the class to which the object belongs to is provided.

6.2.3 Vehicle Position and Other Objects Estimation

For estimating the position of all vehicles/other objects, we calculate the bottom centre pixel position of the bounding box $[u, v]$ with respect to image coordinates. As shown in figure 40, this point signifies the nearest distance of the object, based on the ground plane assumption.

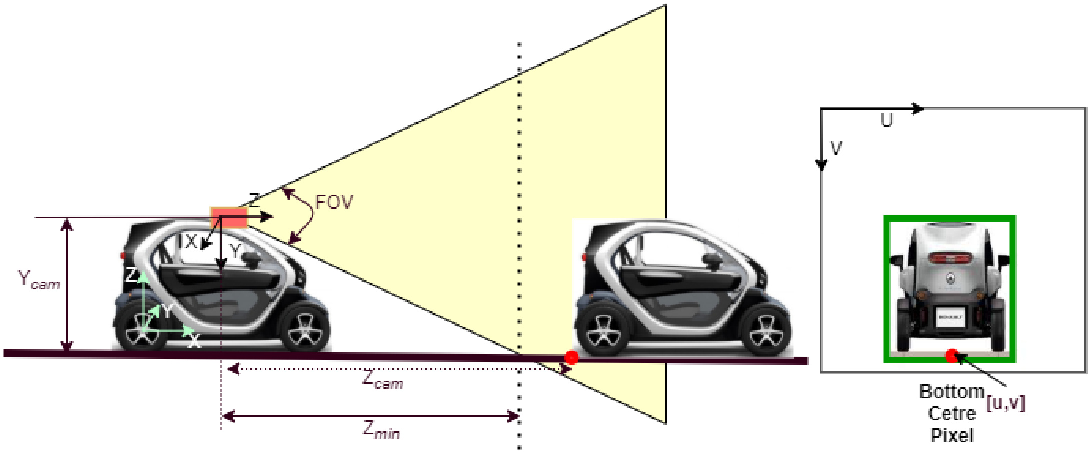


Figure 40: Position Estimation Schematic Diagram Front View

The transformation from pixel to ray for the camera can be expressed as

$$r = [K]^{-1} i \tag{6.1}$$

Where $i = [u\ v\ 1]^T$ is the position of the bottom-centre bounding box pixel in image coordinates, $K = \begin{bmatrix} f_x\ 0\ c_x \\ 0\ f_y\ c_y \\ 0\ 0\ 1 \end{bmatrix}$ is the Camera intrinsic Matrix wherein f_x, f_y, c_x, c_y are the focal lengths and camera principal points, r= $[dx\ dy\ dz]^T$ is the direction vector of the ray originating from the camera focal point and passing through the given pixel.

Given the height of the camera t_z (in meters) with respect to the ground plane, eq 6.2 calculates the scale factor s, required to obtain the intersection point of the ray with the ground plane. w_{cam} is the object's world position in meters with respect to camera coordinate system, x_{cam}, y_{cam} & z_{cam} being the lateral, vertical and longitudinal distance respectively.

$$w_{cam} = [x_{cam}\ y_{cam}\ z_{cam}]^T = [x_{cam}\ t_z\ z_{cam}]^T = sr \tag{6.2}$$

Solving and resubstituting for s gives

$$x_{cam} = t_z\, f_y/f_x \frac{(u-c_x)}{(v-c_y)},\quad y_{cam} = -t_z,\ z_{cam} = t_z f_y/(v-c_y) \tag{6.3}$$

These positions of detected vehicles/objects are then transformed into vehicle coordinates (coordinate system defined in Appendix D.2) using the camera extrinsics $[R], [t]$.

$$w_{veh} = [R]w_{cam} + [t] \tag{6.4}$$

Kalman Filter Calculation

As the preceding vehicle moves away from the ego-vehicle the bounding box translates from the bottom, to the center of the image. To analyze the sensitivity of the measured longitudinal position z_{cam} with respect to the bottom-center bounding box pixel height position v.

From eq 6.3

$$z_{cam} = t_z \frac{f_y}{p} \tag{6.5}$$

where $p = (c_y - v)$ is the vertical distance from the image center.

Sensitivity is defined as the change in longitudinal distance with respect to change in pixel vertical distance

$$|S| = \frac{dz_{cam}}{dp} = t_z\, f_y/p^2 \tag{6.6}$$

Thus, if the cameras optical axis is parallel to the ground plane, as the bottom of the bounding box moves to the centre of the image, the sensitivity increases parabolically.

To mitigate the effects of noise due to high sensitivity, a Kalman filter is implemented that uses the width-based method [7] by incorporating the bounding box height B_h information to calculate the vehicle height H_m given by eq 6.7.

$$H_m = B_h\, z_{cam}/f_y \tag{6.7}$$

The Kalman filter estimates the vehicle height state H, which is constant. The constant model is given below, and since we are confident that the vehicle's height is constant, the process noise is assumed small and positive definite.

$$H_k = H_{k-1} + w_k \qquad E(w_k w_k^T) = Q_k \delta_{i-j} = 0.001 \tag{6.8}$$

$$H_m = H_k + v_k \qquad E(v_k v_k^T) = R_k \delta_{i-j} = 0.1 \tag{6.9}$$

Here the state H which is the height of the detected vehicle.
The Kalman filter equations are as follows [11] :

The model priori covariance P_k^- is calculated as

$$P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + Q_{k-1} \tag{6.10}$$

$$= P_{k-1}^+$$

Kalman Gain K_k is given by

$$K_k = \frac{P_k^- C^T}{(CP_k^- C^T + R_k)} \tag{6.11}$$

$$= \frac{P_k^-}{(P_k^- + R_k)}$$

The model posteriori covariance P_k^+ is calculated as

$$P_k^+ = (1 - K_k C)P_k^- \tag{6.12}$$

$$= (1 - K_k)P_k^-$$

The model priori state estimate H_k^- is calculated as

$$H_k^- = A_{k-1}H_{k-1}^-$$

$$= H_{k-1}^+$$

The model posteriori state estimate H_k^+ is calculated as

$$H_k^+ = H_k^- + K_k(H_m - H_k^-) \tag{6.13}$$

Before calculating the estimated longitudinal distance, the noisy bounding box height measurement needs to be filtered so that the estimated longitudinal distance is not noisy. A first-order discrete filter is used to filter measurements below a time constant τ.

$$B_{h(i+1)} = \frac{\tau}{(1+\tau)}B_{h(i)} + \frac{1}{(1+\tau)}B_h \tag{6.14}$$

$$z_{cam\ est} = f_y * H_k^+/B_{h(i+1)} \tag{6.15}$$

The positions in the camera coordinate system are then transformed into the ego-vehicle coordinate system using equation 6.4.

6.2.4 Target Vehicle Pose Estimation

The Target vehicle pose estimator is responsible for estimating the pose, more specifically, the relative heading (α)and orientation of the target vehicle (θ) with respect to the ego-vehicle, as shown in figure 41.

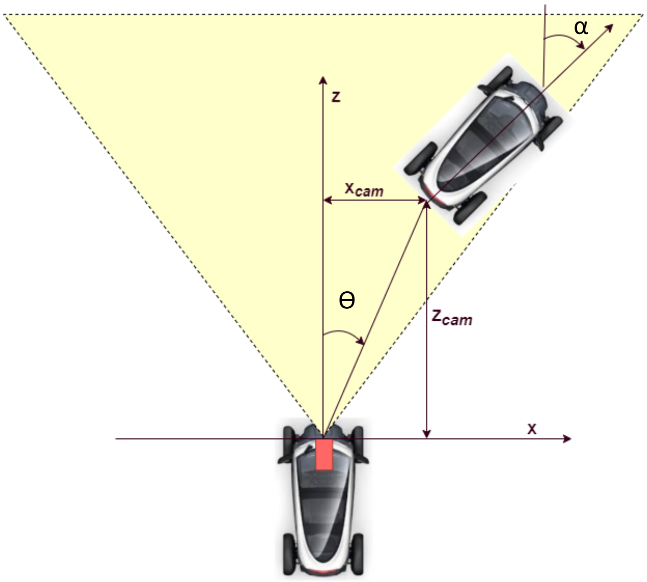


Figure 41:Top-View Schematic for Heading and Orientation Measurements

A model of the target vehicle is first registered offline and then used for online pose estimation. Since the camera sensor size & intrinsic properties are input parameters for both the above activities, the same camera must be used.

Model Registration

For the offline Model registration described in Section 5.2.2.3, a template image of the Twizy is captured using a Sekonix GMSL camera [12]. To extract features and descriptors from the template image, ORB features are used, since they are scale and illumination invariant. To extract the 3D points of the corresponding features, it is assumed that the rear shape of the Twizy is similar to a 3D box. By manually aligning a 3D mesh and using the Moller-Trumbore ray-tracing algorithm [16], the 3d points on the surface of the Twizy are computed as shown in Figure 42.

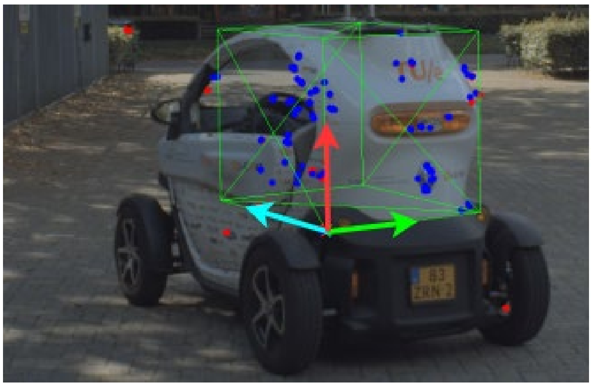


Figure 42: 3D Box mesh manually aligned with the Twizy rear surface and feature points extracted

Another approach can be to estimate the relative 3d position of all features from a stereo image. However, the effects of noise need to be investigated.

Online Pose Estimation

Online pose estimation is performed in OpenCV environment. ORB features & descriptors from the bounding box of a detected target are matched with the features in the model using a matching algorithm (e.g. Flann matcher algorithm) to output a set of matches. The corresponding 3d points of the model are assigned to the matched features in the scene. There may be conditions in which not all the matched 3d points lie on the vehicle. To eliminate these points (outliers), Random Sample Consensus (RanSac) [14] along with PnP algorithms are used to estimate an optimal pose that best fits the given set of points. A 3d mesh box is then fitted to visualize the estimated 6D pose of the target vehicle. Refer OpenCV Real-time Pose Estimation documentation for more details [15].

6.2.5 Lane Detection & Polynomial Fitting

The Lane detection API from DriveWorks provides the image points of the detected lanes. These image points are projected to world points in the vehicle coordinate system using the transformations explained in eq 6.2 & eq 6.4. A 3rd order polynomial is fit on these world points. The polynomial regression model [16] is used for obtaining the polynomial coefficients. The polynomial regression model is given as

$$y_i = C_o + C_1 x_i + C_2 x_i^2 + C_3 x_i^3 + \epsilon \tag{6.15}$$

$$y = XC + \in$$

Where $\in$ is an unobserved random error with mean zero conditioned on a scaler variable x.
C is the vector of estimated polynomial regression coefficients (using ordinary least squares estimation) is

$$C = (X^T X)^{-1} X^T y$$

6.2.6 Free Space Detection

We use the Free Space detection API provided by DriveWorks. The API provides a set of image points which need to be projected to world points similar to Section 6.2.5. Since the free space image points provided by the detector are large, communication of all these points requires too many CAN message IDs which is not feasible. If an object is detected in the image, only free space boundary points in the vicinity of the detected object are provided.

■■■

7. Validation and Test Results

After completion of the implementation of all modules, the final step is the testing and validating the performance of the system. The performance is evaluated using two experimental setups. The first, involves a vehicle simulator for evaluating different possible scenarios. The second involves the vehicle setup involving the Drive PX 2 and the GMSL cameras. In both setups, unit testing is first performed to check whether each function satisfies the requirements and provides necessary information at its interfaces. We then perform system-level testing to evaluate the performance in conformance with the driving scenarios discussed in Chapter 3.

7.1 Experimental Setup

7.1.1. Carla Simulator

A vehicle simulator is necessary for quickly & safely evaluating the performance of each module in different scenarios involving urban layouts (freeways, cities), different vehicle types, buildings and weather conditions. To that effect, the Carla vehicle simulator [18] is used on Ubuntu 16.04 LTS (x86_64 architecture). A BMW sedan is chosen as the ego vehicle and fitted with four cameras placed horizontally, 1.5m from the ground plane. Each cameras have a 60deg FOV with a resolution of 1080x720. Different scenarios are simulated in Carla and fed to the SVS software to validate each functional module, as shown in Figure 43.

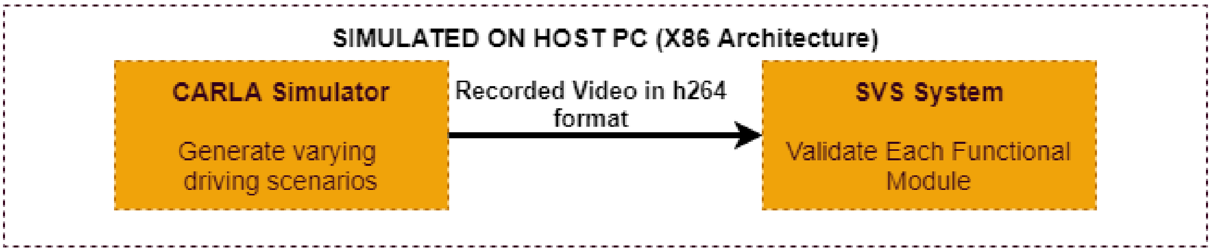

SIMULATED ON HOST PC (X86 Architecture)
CARLA Simulator
Generate varying driving scenarios
Recorded Video in h264 format
SVS System
Validate Each Functional Module

Figure 43: CARLA Simulator setup for unit testing of SVS functions

7.1.2. Test Vehicle

We conducted the experiments on a Renault Twizy cooperative automated driving research platform. It consists of one lead vehicle and one ego vehicle. The ego vehicle, Renault Twizy, which is equipped with four GMSL Sekonix cameras [12] mounted on the roof covering the front, rear, front-left and front-right views as shown in figure 44. Each camera has a 60deg FOV with a resolution of 1920x1208.

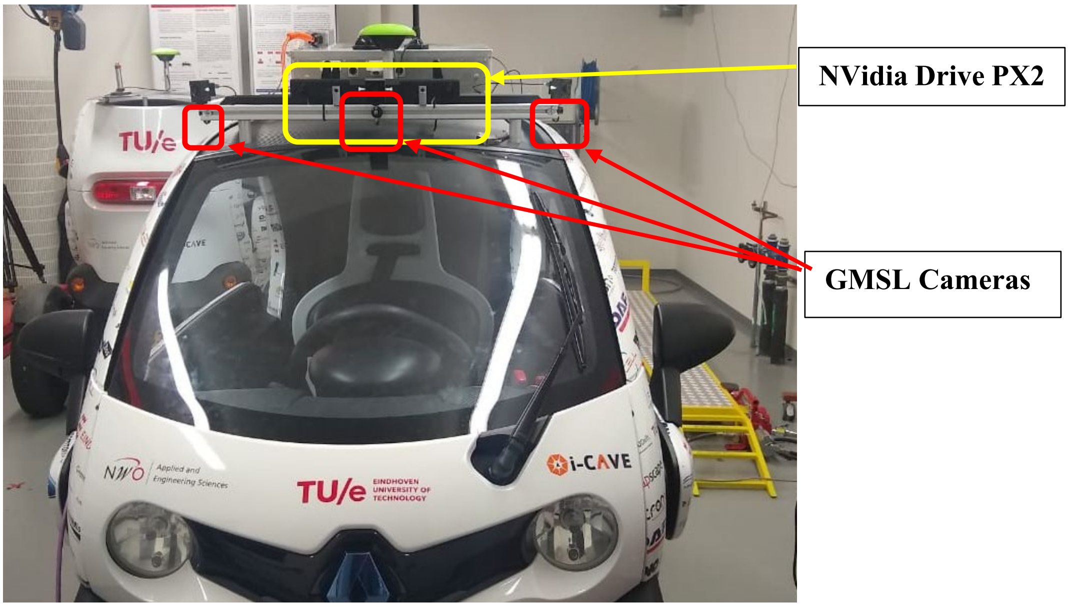


Figure 44: Ego-vehicle fitted with GSML cameras and connected to the DRIVE PX 2

The Drive PX 2 is placed on the roof of the ego-vehicle. It is powered with a 12V DC power supply. It receives input from each camera and communicates with the Vehicle control PC via CAN. The Drive PX 2 consists of two independent SoCs, namely Tegra A and B. The SVS functionality is deployed on the Tegra A for testing purposes. For visual verification of the system by the operator, a HDMI display is fixed (at the back side of vehicle front seat) inside the vehicle.

7.2 Unit Testing

To validate the information computed by each module as per requirements, the output information from each module is rendered on the display. Unit Testing was performed using the Carla simulator as well as the Drive PX 2. The following table highlights the unit-testing results of each module.

Table 2: Unit Testing Observations

Module	Sub-module (level-1)	Sub-module (level-2)	Validation of outputs through Rendering	Performance Observations	Remarks
CIA	NA	NA	Input frames from each camera are captured in the required dimensions and processed successfully	Distortion observed near the image corners while testing with GMSL Cameras	Image Rectification is necessary when using a lens. Due to technical issues, image rectification was kept out of scope.
Object Perception	Object Detector	NA	Objects belonging to all desired classes are detected and encapsulated in a bounding box.	Many false detections observed in heavy rainy weather conditions during simulator test and indoor environments during vehicle tests.	The DriveNet Detector is trained with data collected in sunny weather conditions with cameras placed horizontally.
	Vehicle Tracker	NA	Every detected vehicle object belonging to each camera is marked with a unique instance ID.	Tracked box size and is noisy for objects beyond 30m. False tracks initialized when stationary. The Vehicle tracks are stable for objects up to 40m.	Tracker Parameters need to be fine-tuned for different driving conditions.
	Tracked object Manager		Non-Tracked vehicle objects are killed after not being tracked for 3-consecutive frames.	No memory leaks observed.	
		Vehicle Position Estimator	X,Y Positions of Tracked Vehicle objects are computed in vehicle coordinates using Kalman Filter.	Minimum distance of objects are limited to 7m. As objects move further away from the ego vehicle, the error related to longitudinal distance increases.	Detected Object ground contact points are out of camera FOV. Position estimation accuracy is dependent on the accuracy of camera extrinsic and the complete fitting of bounding on the detected object. Inaccurate fitting at large distances leads to significant errors.
		Other objects Position Estimator	X,Y Positions of Bicycles and Pedestrians are computed in vehicle coordinates	False reading for Objects under 7m.	For objects under 7m, detected object ground contact points are out of camera FOV.
		Target Classifier	All vehicles are classified when a new track is instantiated.	Classification is inaccurate.	Inadequate variety in training data
		Target Pose Estimator	Target Vehicle Pose Estimated.	Pose Estimation evaluated considering the target vehicle as a Van. Pose estimation for multiple targets in	We assume the surfaces of the vehicle planar and use a 3d box mesh to extract the 3d points. Single target setup

				image not implemented.	
Lane Perception Module	Lane Detection	NA	Lanes detected are rendered correctly along with lane types and position types. Lanes detected on Front and Rear cameras.	NA	NA
	Lane Polynomial Fitting	NA	Tested with parallel straight lanes on the left and right of the ego-vehicle.	Able to estimate the offset distance by $\pm 0.1m$ accuracy.	NA
Free space Perception Module	NA	NA	Free space detected on all cameras.	World points computed with 10cm accuracy up to 30m	
CAN Communication			Tracked Vehicles with Track ID, Position and Target status are communicated. Lane & free space information is provided.		

7.2.1. Unit Testing Rendering Results:

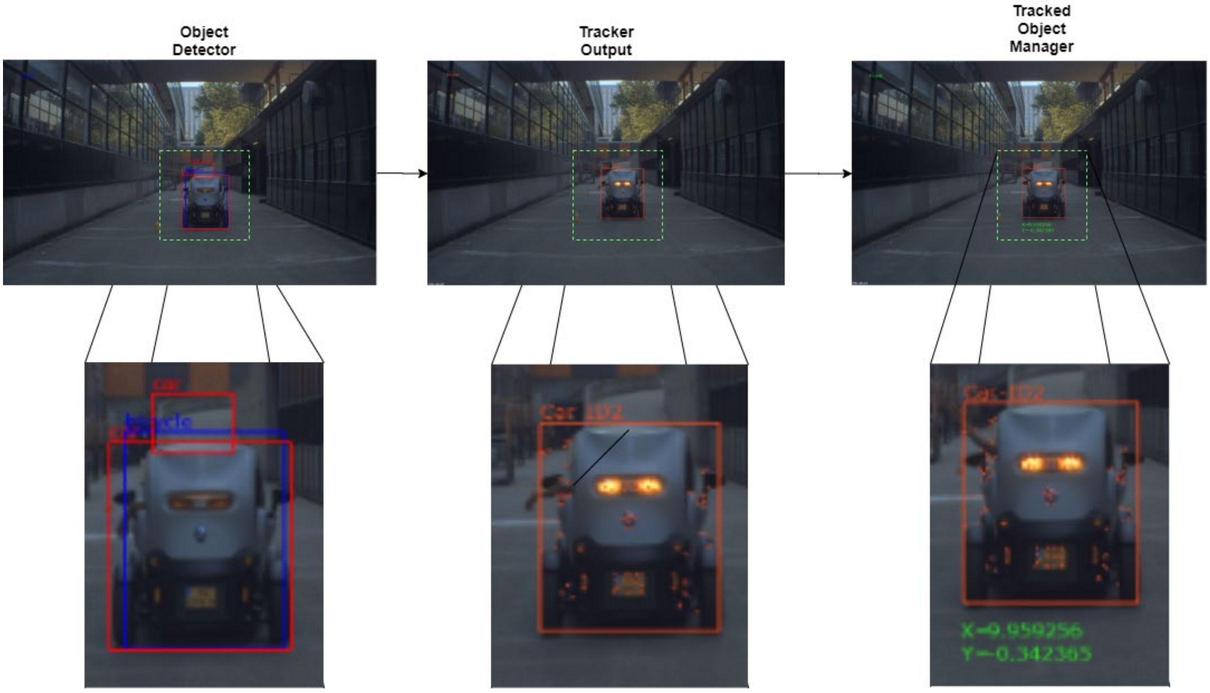


Figure 45: Unit Testing of Object Perception Module

Figure 45 describes the outputs of the object detector, tracker and position Estimator. The object detector accurately detects the preceding vehicle as a car and fits a bounding box accurately over the detection. The detector misclassifies the same vehicle as a bicycle, and there are also false car detections. The vehicle object tracker filters false car detection and provides a stable track with a unique track ID. The stable tracked object x and y position are estimated with respect to the ego-vehicle as per the requirements.

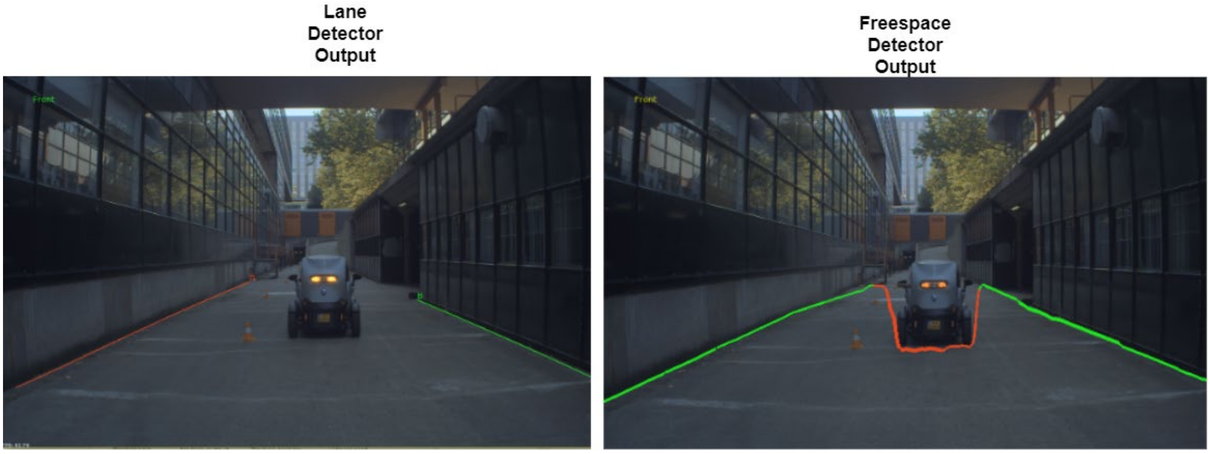


Figure 46: Unit Testing of Lane and Free space module

Figure 46 represents the results of the lane and free space module. The lane module clearly detects the rad boundary to the left and right with different colours. The free space module also clearly identifies the free space in the image by providing free space boundary points as rendered in the image. Free space points near the detected vehicles vicinity are highlighted in red. These points are provided to the CAN interface.
For combined system results check the following video link : https://www.youtube.com/watch?v=4tIrmb65i8A

7.2.2. Position Estimation Static Results

To validate the accuracy of the position estimator, we place a lead vehicle at a distance between 10 to 40m from the ego vehicle and measure the accuracy in steps of 10m, as shown in figure 47.

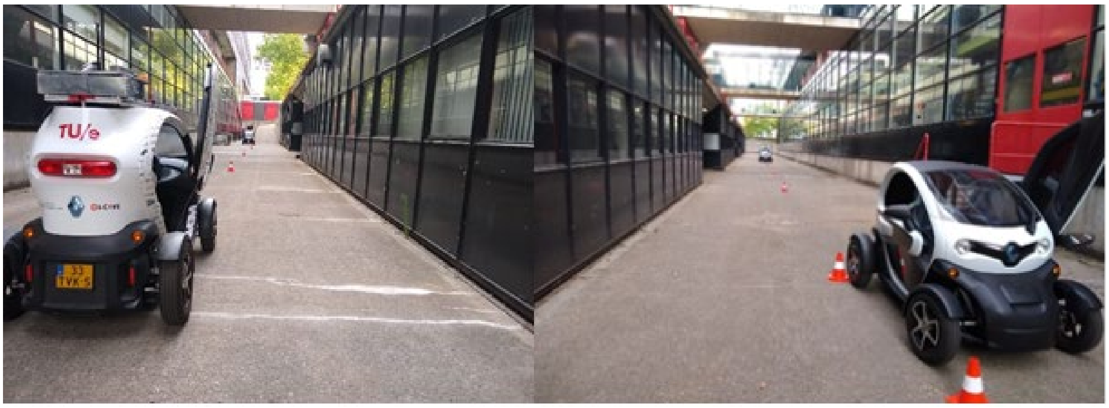


Figure 47: Static Test Setup with cones placed at steps of 10m.

Table 3: Position Estimation static testing results

Intervehicle Distance Ground Truth(m)	Intervehicle Distance Measured (m) Δ	Absoluter Error (Ground Truth - Measured Position) (m) Δ_{error}	Standard Deviation of Measurement Data (m) σ
10	10.05	0.05	0.05
20	20.06	0.06	0.15
30	32	2.0	0.8
40	45.3	5.3	0.8

As per Table 3, it is observed that the absolute error (Δ) depends on the error in the camera pose, the intervehicle distance and the detection accuracy (how well the box fits the detected vehicle). The standard deviation (σ) on the other hand is inversely proportional to the camera resolution.

7.2.3. Position Estimation Dynamic Results

In relation to Position Estimation, for evaluating the performance of Kalman filtering, the following tests are performed by placing cones in steps of 10m distances to mark the ground truth positions.

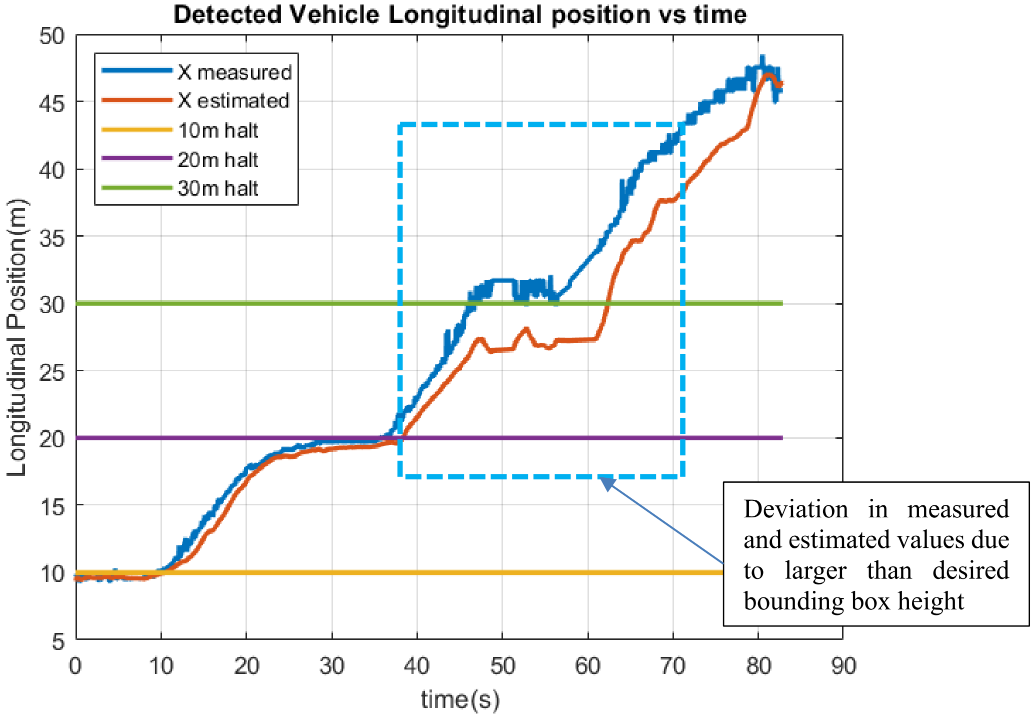
1) Preceding Vehicle moving away from ego-vehicle from 10m to 30m.
2) Preceding Vehicle approaching ego-vehicle from 30m to 10m.

1) Preceding Vehicle moving away from ego-vehicle from 10m to 30m.

In this scenario, the preceding vehicle starts at 10m and departs away from the ego-vehicle, intermittently halting at 20m and 30m. Figure 48 correlates the measured longitudinal position, the estimated longitudinal position by the Kalman filter and the ground truth locations of the cones (10m, 20m, 30m). Between 10m to 20m, it is observed that the error between the measurements and the Kalman filter estimates are small. As the filter accounts for the past historical data, the filter estimates lag the measurement readings.

Between 25m and above, it is observed that the error between the measurements and the Kalman filter estimates increase. This is due to the poor fitting of the detected vehicle's bounding box during this phase(as per visual observation). The height of the tracked bounding box provided by the tracker is larger than desired. As the box height and the measured longitudinal distance are used to obtain the preceding vehicle's height, the calculated value is higher than expected, as shown in figure 49 (between 37-70sec). The larger than expected box-height measurement produces an estimate that is shorter than the actual longitudinal distance.

The accurate fitting of the bounding box with the detected vehicle in the image depends on the parameters of the tracker used for clustering the neighbouring bounding box proposals provided by the object detector. These parameters need to be tuned such that the performance is satisfactory in all environments.



Deviation in measured and estimated values due to larger than desired bounding box height

Figure 48: Longitudinal position results for vehicle departing away from the ego-vehicle

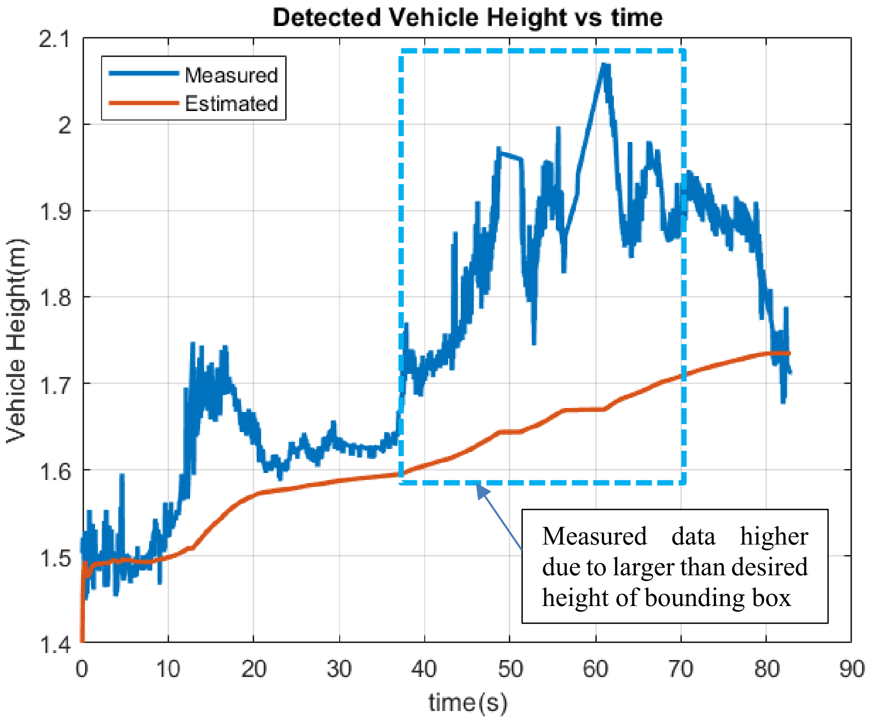


Figure 49: Vehicle Height measurements & estimates of the detected preceding vehicle

2) Vehicle approaching an ego Vehicle 30-10m.

In this scenario, the preceding vehicle starts at 30m and approaches the ego-vehicle, intermittently halting at 20m and 10m. Figure 50 correlates the measured longitudinal position, the estimated longitudinal position by the Kalman filter and the ground truth locations of the cones (10m, 20m, 30m). The measurement readings are accurate w.r.t to the ground truth, and the filter estimates also do not deviate significantly from the measured data. As discussed in the earlier scenario, the filter estimates lag the measurement reading.

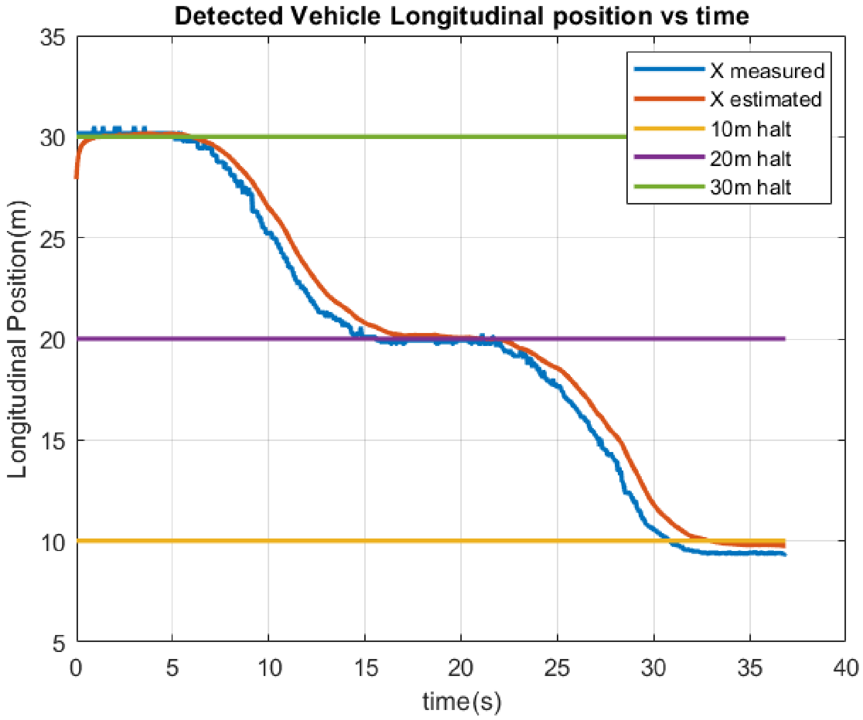


Figure 50: Position estimation results for vehicle approaching the ego vehicle

Figure 51 compares the measured and estimated values for the height of the detected vehicle. When a new tracked object is detected, the initial estimate quickly converges to the measurement values since the Kalman gain is high during the initial readings. The Kalman gain drops with subsequent measures making the filter less sensitive to measurements reading as the track ages, which is desirable.

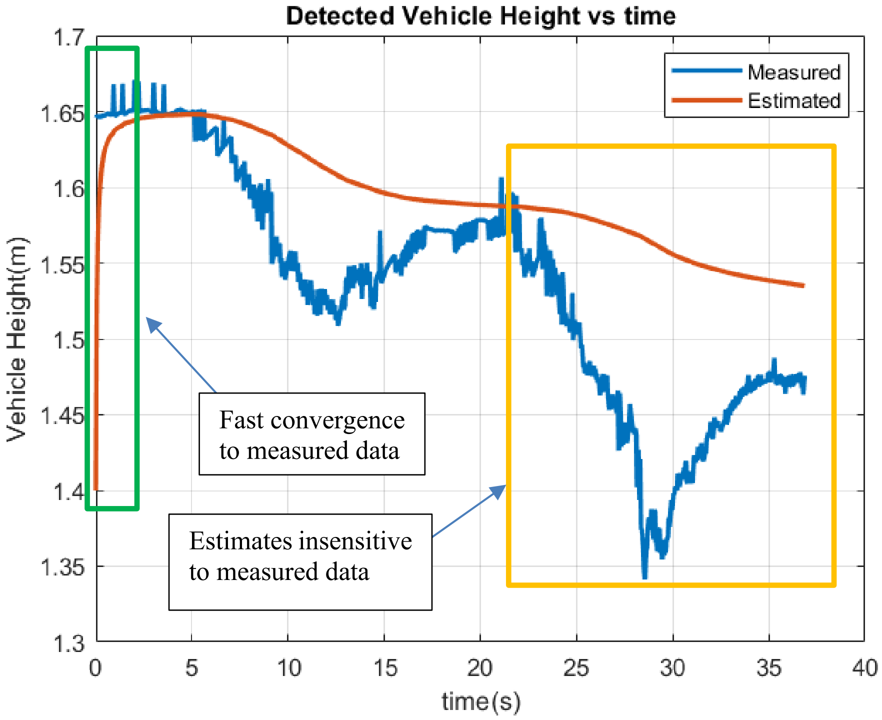


Figure 51: Vehicle Height measurements and estimates of preceding vehicle

Form the above two scenarios it can be observed that the position estimation method implemented is vehicle agnostic and provides the position of detected objects fairly accurately upto 30m. For objects beyond 30m, a conservative estimate is provided (lower measurement than actual). The filter does a good job of removing any disturbance or noise due to detected box size or due to road undulations.

7.2.4. Target Pose (Heading) Estimation Results

To validate the pose estimate more specifically the heading angle, the validation is scoped to testing a single target vehicle detected by the front camera of the ego-vehicle. The testing is performed in accordance with driving scenario 4. Figure 52 describes the test setup where a stationary target is placed in front of the ego vehicle, and the ego-vehicle is driven back and forth at a slow speed (3kmph) on a curved path by locking the steering wheel towards one side. The detailed expected behaviour is described in Appendix A (DS 4).

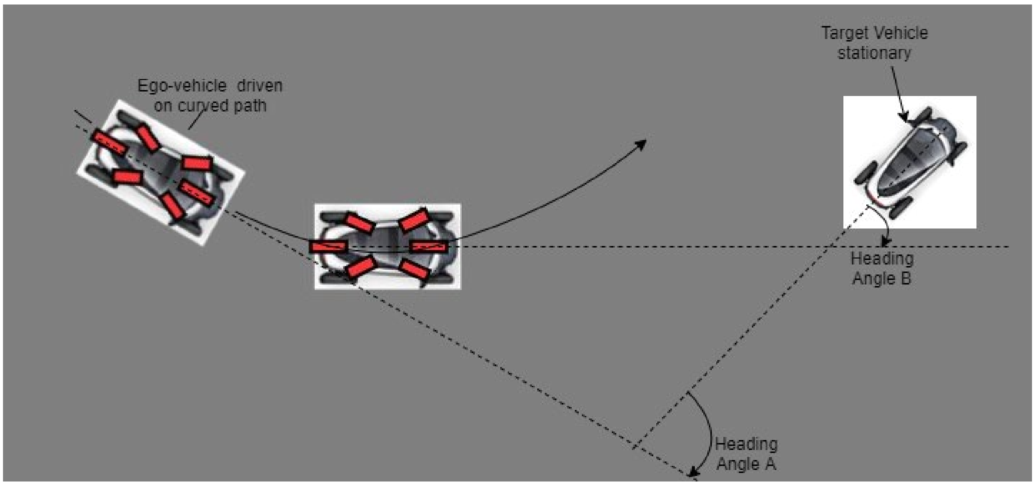


Figure 52: Test Setup for Target Pose Estimation

During execution, we detect features within the bounding box of the target. These features are then matched with the features in the template model. It is observed that a high number of matches (at least more than 30) provides a more accurate pose estimate. Among the matches, inliers are computed using the RANSAC and PnP algorithms to obtain an optimal estimate of the pose. If the interpixel distance between the inliers are greater, the precision of the pose estimate is higher. As the inter-vehicle distance between the target and ego-vehicle increases the inter-pixel distance between the inliers drops, leading to a drop in precision.

For validation, an IMU sensor is placed in the ego-vehicle, which will provide the change in yaw values.
Figure 53 shows the heading angle of the target vehicle computed by the front camera and the relative change in the heading of the ego-vehicle measured by the IMU. Since the target vehicle is stationary, the change in the ego-vehicle's yaw should match the change in the heading angle of the target. A good correlation is observed in the two plots.

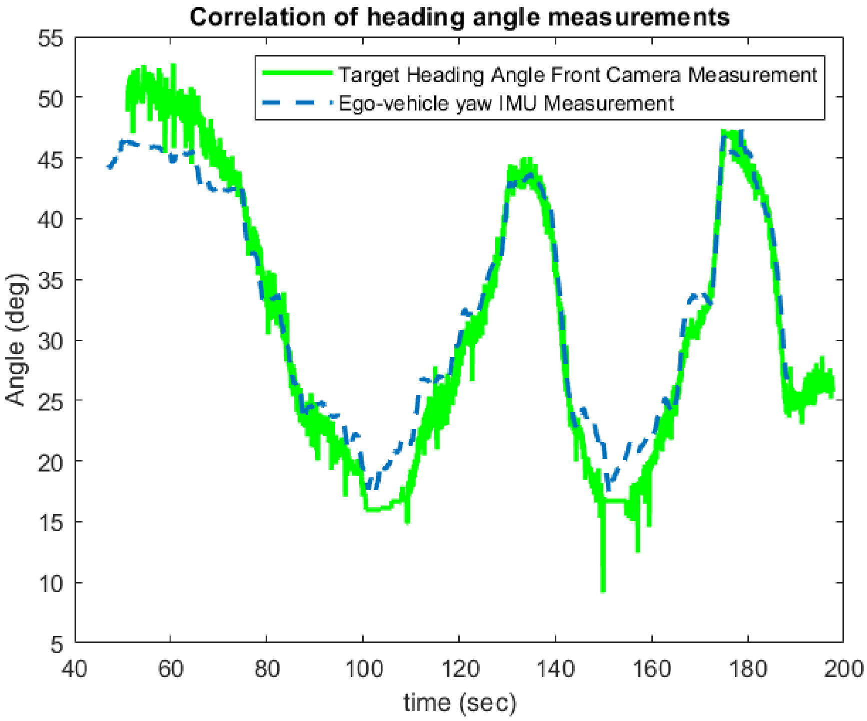


Figure 53: Correlation of heading angle measurements by camera with IMU measurements

For the online testing results click on https://www.youtube.com/watch?v=sDuzfNlXUEA

From the above results, it can be concluded that the pose estimation works well for intervehicle distances up to 30m and for small heading angles (0-30deg). If the target vehicle is wide (e.g. Truck) the interpixel distances of the features detected will be wider than the existing setup, leading to better precision and greater intervehicle distances. Thus, this method is suitable for highway platooning driving scenarios where the relative heading angles between vehicles are lower.

7.2.5. Lane Polynomial Fitting Results

The plot in figure 54 indicates the road boundary points detected by the Lane Detector on the left and right side of the ego vehicle. In the image, lanes that are nearer to the ego-vehicle are represented by a greater number of pixels. When these pixels are mapped to world points, a higher resolution and precision is obtained for points nearer to the vehicle.

It should be noted that the position of these 2D points is reasonably accurate compared to the ground truth. However, the accuracy is dependent on the camera extrinsic parameters which are affected by the vehicle motions (pitch, bounce).

Third-order polynomials are fit through these 2D road points. Thus, the lane coefficients provided by the system are accurate and are useful for path planning purposes.

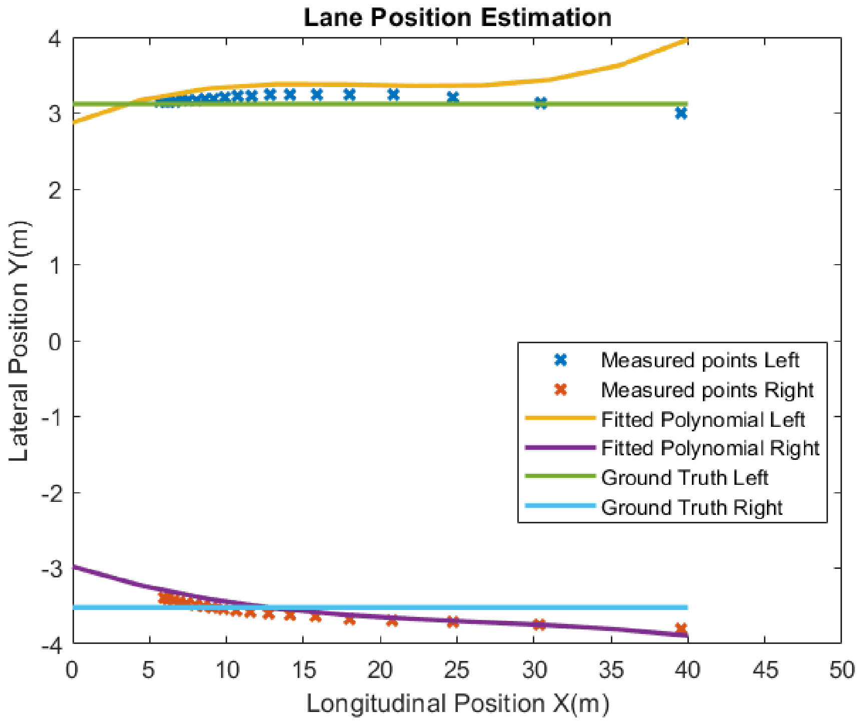


Figure 54: Lane Polynomial Fitted with measured data

7.3 System-Level Testing

In system-level testing, we evaluate the response of the SVS in different driving scenarios by driving the ego-vehicle in the TU/e campus. In the video described in the following link, we drive in an urban environment (TU/e campus) following a target vehicle in front of the ego-vehicle.

Video-link: https://www.youtube.com/watch?v=G0XBVVIsJOU

Since the test is conducted in an urban scenario, many detections were observed for objects beyond 100m which were irrelevant. Since each object is processed sequentially, it leads to longer execution time. We thus tuned our parameters to filter out objects which are smaller than a threshold box size limiting the detection range to approximately 35m.

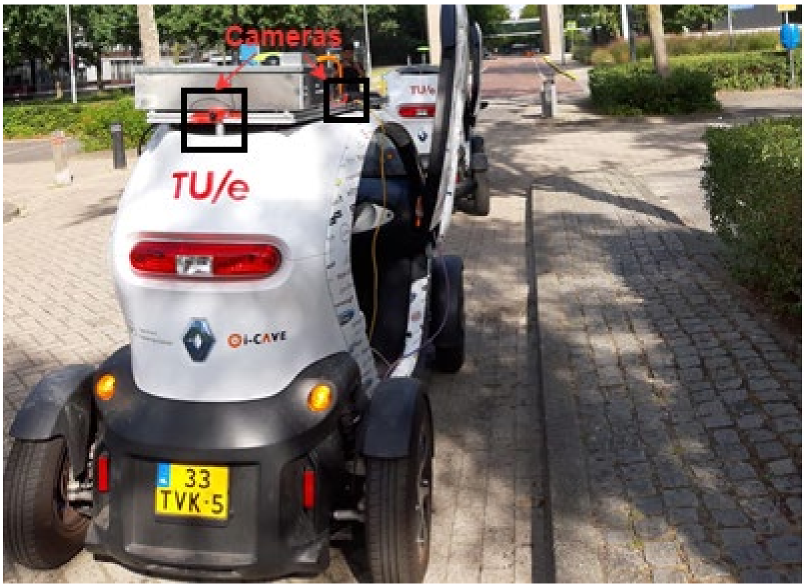


Figure 55: On-Road testing with target and ego vehicle

7.3.1. System Profiling:

To analyze the overall execution time taken per input frame by the system, we breakdown and measure the time taken by each module. As per the architecture since object, free space and lane detections are independent processes, it is desired to execute these processes on separate threads. However, due to limited computational resources resulting in internal switching between processes, we observed that there were no significant improvements in running these processes in parallel compared to sequentially [19]. Additional research is necessary to investigate this problem.

The timeline shown in Figure 56, describes the execution time for the front camera when a new vehicle track is created due to a new vehicle detection or due to a track change.
Figure 57 describes the execution time when existing tracked vehicles are detected, and no new tracks created. The total processing time is given by

$$\tau_{sys} = m\{\tau_{CIA} + \tau_{OD} + \tau_{FSD} + \tau_{LD} + \tau_{TR} + n\tau_{Cl} + l\,\tau_{p} + \tau_{pose}\}$$

Where

$\tau_{CIA}: Image\ Acquistion\ \&\ Processing$ $\tau_{OD}: Object\ Detection$
$\tau_{FSD}: Freespace\ Detection$ $\tau_{LD}: Lane\ Detection$
$\tau_{TR}: Vehilce\ Tracking$ $\tau_{Cl}: Vehilce\ Classification$
$\tau_{p}: Vehicle\ position\ estimation$ $\tau_{pose}: Target\ pose\ estimation$
$m: Number\ of\ Cameras$
$n: Number\ of\ new\ vehicle\ object\ tracks$
$l: Number\ of\ live\ vehicle\ object\ tracks$

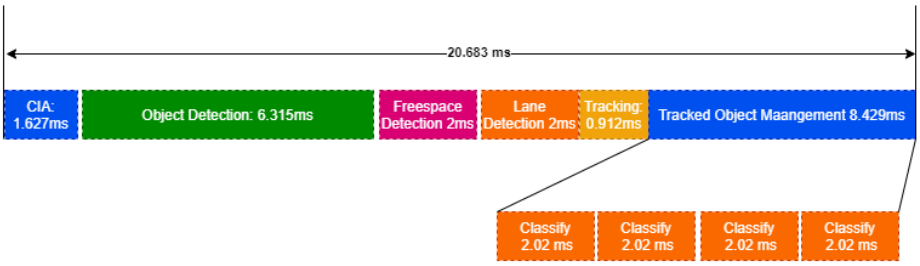


Figure 56: Execution time for Front Camera when four new tracks are created

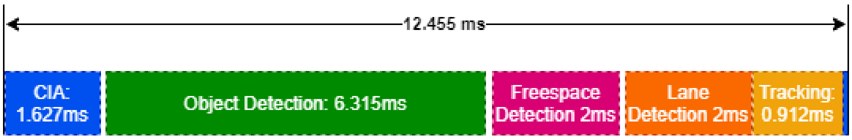


Figure 57: Execution time for Front Camera with existing tracks.

From the above two figures, it can be concluded that by performing classification only once, a clear improvement in execution time is observed. It is also observed that the execution time proportionally increases with the number of cameras.

7.3.2. Comparison with Stereo Vision System.

The following table provides a comparison with of the execution time taking by the proposed SVS system and the earlier stereo system on a desktop with NVidia 1060 GPU and an Intel Core i7-4790K CPU.

Table 4: Comparison of the execution time of SVS and Stereo Vision systems on Host PC

Functions	**SVS System per camera**	**Stereo Vision system**
Image Acquisition and Processing	1.627ms	1.627ms
Object Detection & Tracking	7.2 ms	7.2 ms
Lane Detection	2 ms	2 ms
Free space Detection	2 ms	2 ms
Position Estimation/Depth estimation	0.1ms (considering 100 vehicle object tracks)	94 ms
Target vehicle Classification	2ms	---
Target vehicle Pose Estimation	164 ms	---

It can be concluded that the estimation time for position estimation is 940 times faster than the earlier stereo vision system. The high execution time of 164ms for Target vehicle Pose estimation is due to the OpenCV CPU implementation. OpenCV GPU implementation is recommended.

■■■

8.Project Management

This chapter gives an overview of the time and resource management for the project. First, a description of the project plan is provided, which is used in the second section to make a risk assessment for the most critical risks for completion of the project, according to the goals, and on time. After this, an assessment is made of how well the plan was followed.

8.1 Project Planning

At the start of the project, a work breakdown was constructed to identify the sub-tasks required for successful implementation of the concept on the prototype vehicle. To achieve the main goal, the following sub-tasks were identified:

1) Identify and construct the functional requirements
2) Setup the necessary software packages required for training Deep Neural Networks and DriveWorks
3) Evaluate different neural network architectures and compatibility with NVidia DriveWorks.
4) Visualize the system interconnections (architecture, relation to existing systems)
5) Deploy and test existing networks to identify initial errors in the architecture.
6) Build a vehicle Test setup to generate data for training custom detector/classifier.
7) Label and train DNN.
8) Implement system functions on the host machine (personal PC) using C++ and DriveWorks Libraries.
9) Deploy system on Drive PX 2.
10) Perform on Unit and on-Road Testing.

Based on this work-breakdown, an initial plan was created, as shown in figure 58. To control the process, progress meeting with the stakeholders and supervisors were defined bi-weekly with the university supervisor as well as the company supervisor. To be able to compensate for delays in the project, it was planned to finish at the beginning of August, leaving the remainder time for other delayed activities.

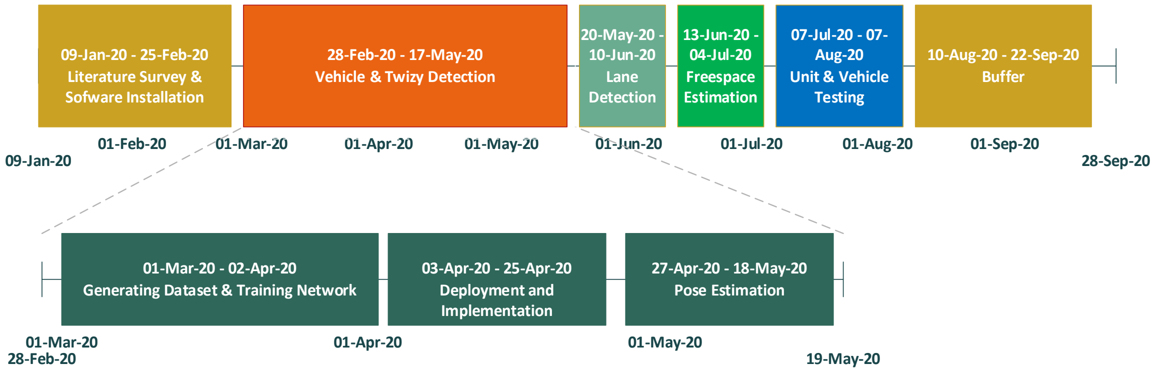


Figure 58: Initial Project plan constructed at the beginning of the project.

8.2 Risk management

With the project plan from the previous section, the main risks for the completion of the project goals according to the requirements within the given time were identified. This activity was repeated in the beginning of every month based on the knowledge gained during the literature study and implementation trials. Exploring and experimentation with the provided code samples of DriveWorks provided an insight on what solutions are optimal, compatible and feasible within the given time-frame.

The main risks, along with their likelihood and impact, are listed in Table 5. Possible mitigation strategies are also listed in this table. Risks R1, R2 & R7 actually happened, resulting in the decision to go ahead with CARLA simulator testing and training for a different type of target vehicle available in the simulator. Due to unavailability of access to the labs, there was a lot of uncertainty related to the implementation on the Drive PX2. However, when access was granted, fortunately due to the unit testing carried out on the simulator, no serious software bugs were observed.

Table 5: Risk Management Table

ID	Description	Likelihood	Impact	Time Horizon	Mitigation Strategy
R1	Unable to generate enough training data for the Object Detector/Classifier using the vehicle mounted cameras due to COVID19 restrictions	5	4	Medium	Use a vehicle simulator to generate training data for classifier. Use Nvidia's trained object detector.
R2	Drive PX 2 unavailable for testing due to corona restriction	5	5	Long	Preform all unit and system test on a simulator to validate all functions.
R3	Insufficient time and manpower for labelling training data	4	3	Long	Accept the risk and adjust the scope of the project to only building the software pipeline for the neural network. Inform all stakeholders about the problem and arrive at a mutual agreement.
R4	Testing availability limited due to COVID19 university lab restrictions	3	3	Long	Plan tests precisely in advance; prepare necessary experiments off-line to minimize required time with the vehicle.
R5	DriveWorks not compatible with certain network architectures	3	3	Short	Study and analyse network architectures of existing state of the art networks and identify reasons for incompatibility at an early stage.
R6	Limited time for implementation of pose estimation for targets	4	5	Short	Assign higher priority to methods that have available code which can be easily modified and integrated in less time.
R7	Unable to validate the system for highway driving conditions due to COVID19 restrictions	5	3	Long	Accept the risk.

8.3 *Project task execution*

The project plan present in Section 8.1, as is usually the case could not be executed fully according to plan. Figure 59 shows how the planned tasks were actually executed.

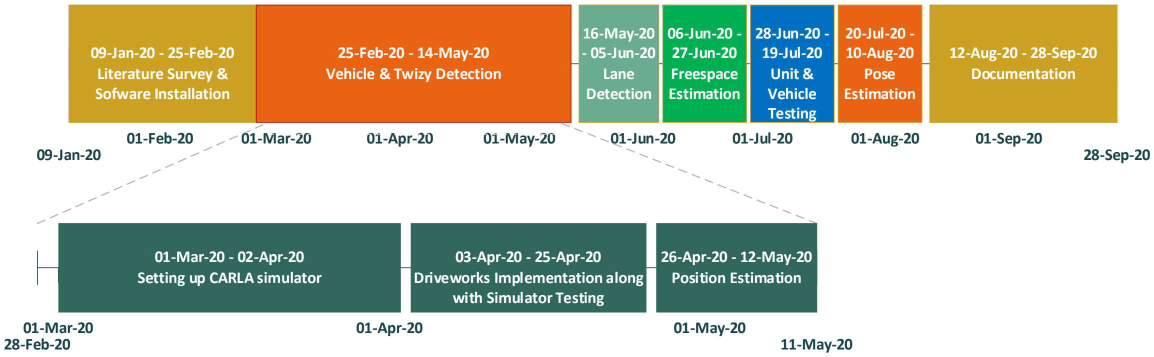


Figure 59: Actual Executed Plan

The main delay was caused due to the unavailability of access to the vehicle and the Drive PX 2 till the month of July resulting in the scheduling of all vehicle testing activities from mid-July. Furthermore, the Drive PX 2 was expected to be flashed with the latest version of DriveWorks 1.2 and ready, which was not the cause. Additional two weeks were lost in understanding the processes related to flashing the Drive PX 2, which were not accounted for earlier. Thankfully, due to the rigorous unit testing performed earlier on the simulator, few runtime errors were observed. The final phase of the project between July to mid-September was critical where it was needed to push harder to integrate the Pose Estimation module designed in OpenCV with DriveWorks. It was possible to integrate and test for only a few controlled cases; however, on-road testing for different use-cases seemed infeasible due to COVID-19 restrictions.

■■■

9. Conclusion and Recommendations

This report started with a motivation for self-driving cars, the benefits they offer, and the challenges present to reach level 5 self-driving. The need for a surround vision system was described to meet the goal for cooperative and autonomous driving. The system was then designed using a system engineering approach which involved using the CAFCR methodology for identifying the stakeholder's concerns and key drivers. The SEMAS methodology was used to describe driving scenarios, derive requirements and propose the architecture of the system by describing the desired behaviour at different levels of abstraction. Functional modules, which were derived from the architecture, were implemented considering different design choices available in the literature and the constraints. Different Deep Neural Network architectures were evaluated with respect to implementation effort, time and compatibility. Conventional computer vision techniques were used to estimate the position of objects and pose for target vehicles. The designs were implemented using NVidia DriveWorks and OpenCV libraries. The designed software was deployed on the Drive PX 2 embedded system. Each module was validated with respect to the driving scenarios. Finally, this chapter closes the report first by providing an account of the achievements made in this assignment. Furthermore, some recommendations and final thoughts on future work are presented at the end of the chapter.

9.1 Conclusions

The main technical goal of this assignment was to design and deploy a prototype of a real-time surround-vision system in the demonstrator vehicle Renault Twizy. The main focus of this project was to investigate the possibility of using monocular cameras to detect and estimate the position of surrounding objects, identify a potential target among the detected vehicles for platooning and estimate their pose.
The following are the key conclusions observed while carrying out this project:

1) **System Architecture:** The Architecture of the system is designed using object-oriented principles, each function is decoupled, and its interfaces are clearly identified. The architecture is abstract and independent of the implementation. This allows different possible implementations in the future.

2) **Object & Target Vehicle Detection and Tracking:** Deep Neural Networks are used for robust detection of objects. Different network architectures were evaluated considering the compatibility with DriveWorks and Drive PX 2. After careful consideration of the available resources (Time for creation and labelling of the dataset, human resources) it was decided to use an existing trained object detector for detecting vehicles and train a classifier for classifying vehicle detections as target vehicles (Twizys). The bounding boxes are tracked to mitigate false detections. The proposed design efficiently detects and tracks vehicles and other objects and provides stable detections to the vehicle controller through the CAN interface. Thus, with all the stable & robust information of objects from the surroundings, the vehicle controller can take safe decisions.

3) **Target Classifier Training:** A training methodology for data collection, labelling of the dataset for classification and deployment in the system is described. However, in this project, it was not possible to train the classifier robustly due to insufficient training data. For future work, additional data can be generated using the existing camera setup by driving in all desired scenarios using the proposed data-generation and training methodology.

4) **Position estimation of Objects:** Position of objects are estimated by solely relying on monocular vision. It is assumed that the ego-vehicle and the detected object lie on the same ground plane and the height of the camera from the ground plane is known. The assumption is valid for highway driving conditions and identifies the position of objects accurately up to 30m. The algorithm takes negligible memory and is 1000x faster than stereo disparity computations, thus suitable for real-time applications.
For vehicle objects, it is assumed that the height of the vehicle is constant and additionally, the height of the bounding box is used to provide a more robust estimate which is independent of the ground plane assumption. Thus, by combining these two measurements, the system can work in all possible scenarios, and only depend on accurate detection (location and fitting of the bounding box) of the objects. The position and class of all detected objects are available on the CAN interface which can be used for fusion with other sensors.

5) **Target Vehicle Pose Estimator:** The pose of a target is computed given the prior information of the shape of the target. In this project, the target is a Renault Twizy. ORB Features and their corresponding 3D locations are extracted and used as a template to match the features of the same object in the scene. Based on the matches, the pose of the vehicle is computed. Since ORB features are used, the algorithm is robust to illumination. The heading of the target is validated using an IMU sensor. The precision of the pose/heading drops as the intervehicle distance increases. In this project, the testing was scoped to only extracting features points from the rear & left faces of the model. However, further testing is required to identify how the algorithm performs in different scenarios.

6) **Lane and Free space Detection:** Existing neural networks provided by DriveWorks are implemented to detect lanes and free space. The coefficient of the lanes computed are validated during testing. These coefficients are provided to the CAN interface. However, further testing is required to validate the Lane Polynomial Coefficients, Object Free Space Points on CAN data.

9.2 Future Work and Recommendations

The section outlines the work required to be done for the completion of the project along with the possible future extensions.

- As mentioned in the above section, the architecture of the system was designed with a purpose of building a prototype in mind. For further improvement, it is necessary to do a functional safety analysis of each functional component. Using the existing functional modules, high levels modules that can fuse the information from both modules can be designed. For e.g.
    - Computing the lane on which a detected vehicle is running on. Higher priority can be assigned to those vehicles which are running on adjacent lanes.
    - Using the free space and the bounding information to determine nearest objects. This will aid in dynamic path planning.
- It is suggested to use the Robot Operating System (ROS) framework, where the functional modules are mapped to ROS nodes. This will improve the flexibility of the system and make it easy to build and add more functionality without disturbing the existing code.
- The Drive PX 2 consists of two identical and independent Parker System on Chips (SoC)GPUs, called Tegra A and B respectively. At present, all neural networks are deployed only on the Tegra A SoC. Even though the detection modules are run on independent CPU threads and CUDA streams, the modules operate sequentially due to limited resources. It is recommended to exploit the full capabilities of the Drive PX 2 by deploying a few of the networks on the Tegra B SoC and use TCP/IP to communicate the results between both SoCs.
- In relation to pose estimator, the rear shape of the Twizy was assumed as a 3d Box, and features points were extracted from only the left and rear side of the target. It is recommended to fit a 3d mesh of a Twizy to extract all the points on the external surface of the vehicle. This will lead to better precision and accuracy for targets beyond 30m.
- Object Detectors like Yolo V4 and Yolo V5 which are recently proposed in 2020 provide a higher accuracy and detection speed. It is recommended to evaluate how much improvement in execution time, accuracy and memory can be achieved by deploying these networks on the Drive PX 2. Specific attention needs to paid for unsupported layers of DriveWorks TensorRT.
- The change in camera extrinsic positions due to the compliance of the ego vehicle's suspension can be corrected by online camera calibration.
- The current software implementation is scoped to estimating the pose of a single target vehicle present in the scene. It is recommended to improve the software to implement multiple dynamic targets. Furthermore, the implementation of pose estimation can be moved to the GPU for faster than present CPU execution.
- Presently fusion of measurements from different sensors occurs at a late stage, e.g. Processing information from each sensor like radar, camera independently and then fusing the position measurements. Also, only limited information can be transmitted by each sensor due to the limited bandwidth of the CAN network. If the raw point cloud data of the radar was clustered using the camera bounding box detections at an early stage, this would lead in the improvement of the detection accuracy.

Glossary

CIA	Camera Image Acquisition
CPU	Central Processing Unit
CUDA	Compute unified Device Architecture
DeepIm	Deep Iterative Matching for 6D Pose Estimation
DNN	Deep Neural Network
D&C	Dynamics and Control Group
FOV	Field of View
GMSL	Gigabit Multimedia Serial Link
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
i-CAVE	Integrated Cooperative Autonomous Vehicles
IO	Input/output
MPS	Mobile Perceptions Group
OS	Operating System
ORB	Oriented Fast and rotated Brief
PnP	Perspective-n-Point
PVNet	Pixel-wise Voting Network for 6DoF Pose Estimation
RANSAC	Random sample consensus
SoC	System on chip
SVS	Surround Vision System
TU/e	Eindhoven University of Technology
V2V	Vehicle-to-Vehicle
Yolo	You only look once

Bibliography

References

[1] "NHTSA: Automated Vehicles for Safety," [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety. [Accessed 02 September 2020].

[2] "I-Cave -- Integrated cooperative automated vehicles," [Online]. Available: https://i-cave.nl/.

[3] Z. Luo, "LiDAR based perception system: Pioneer technology for safety driving. PhD Diss," 2017.

[4] G. Muller, CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity, 2004.

[5] M. M. H. a. A. K. Hedderich, "SEMAS-System Engineering Methodology for Automated Systems. The world described in layers.," in *22nd Workshop-Methods and Description Languages for Modelling and Verification of Circuits and Systems*, 2019.

[6] W. J. Z. Y. Feng Liu, "A Multi-object Tracking Method Based on Bounding Box and Features," in *Advances in Computer Science for Engineering and Education II*, 2006.

[7] G. K. a. J. Cho, "Vision-based vehicle detection and inter-vehicle distance estimation," *International Conference on Control, Automation and Systems, JeJu Island, 2012,* pp. 625-629, 2012.

[8] O. M. A. S. G.P. Stein, "Vision-based ACC with a single camera: Bounds on range and range rate accuracy," in *Intelligent Vehicles Symposium*, 2003.

[9] J. Lee, "Intervehicle distance estimation through camera images," *Journal of Electronic Imaging,* vol. 27, no. 6, 2018.

[10] "NVIDIA DriveWorks," [Online]. Available: https://developer.nvidia.com/drive/driveworks. [Accessed 25 08 2020].

[11] D. Simon, Optimal State Estimation: Kalman H-infinity and Non-linear Approaches, John Wiley & Sons, 2006.

[12] "Sekonic GMSL Camera Datasheet.," [Online]. Available: https://www.leopardimaging.com/uploads/LI-AR0231-GMSL_datasheet_Update.pdf.

[13] "Möller–Trumbore intersection algorithm - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm. [Accessed 10 08 2020].

[14] "Wikipedia: Random Sample Consensus," [Online]. Available: https://en.wikipedia.org/wiki/Random_sample_consensus#:~:text=Random%20sample%20consensus%20(RANSAC)%20is,as%20an%20outlier%20detection%20method.. [Accessed 25 August 2020].

[15] "Real Time pose estimation of a textured object," [Online]. Available: https://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html.

[16] "statsdirect.com Polynomial Regrerssion," [Online]. Available: https://www.statsdirect.com/help/regression_and_correlation/polynomial.htm.

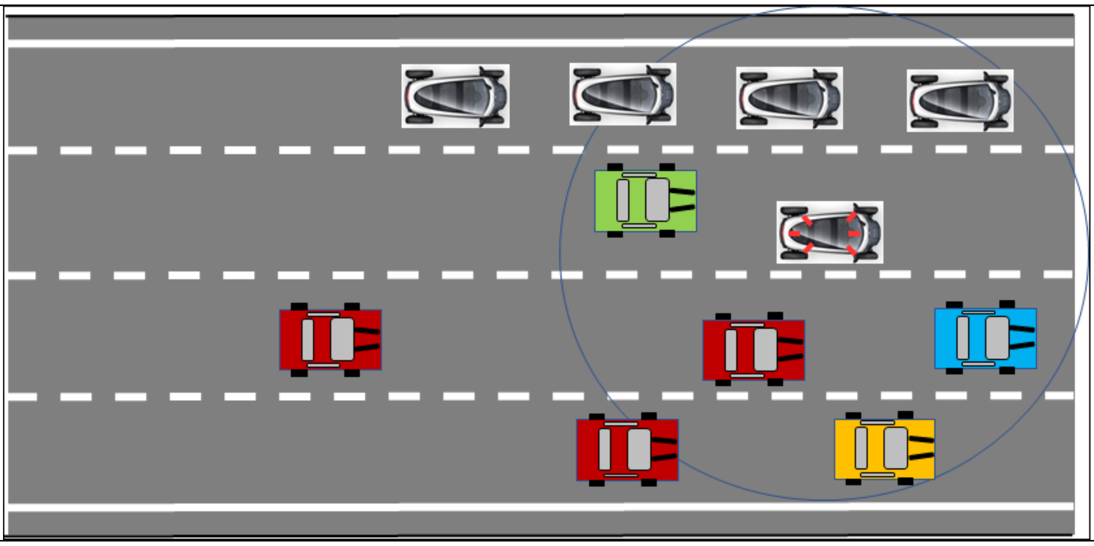
[17] "Eigen Polynomial Module," [Online]. Available: https://eigen.tuxfamily.org/dox/unsupported/group__Polynomials__Module.html.

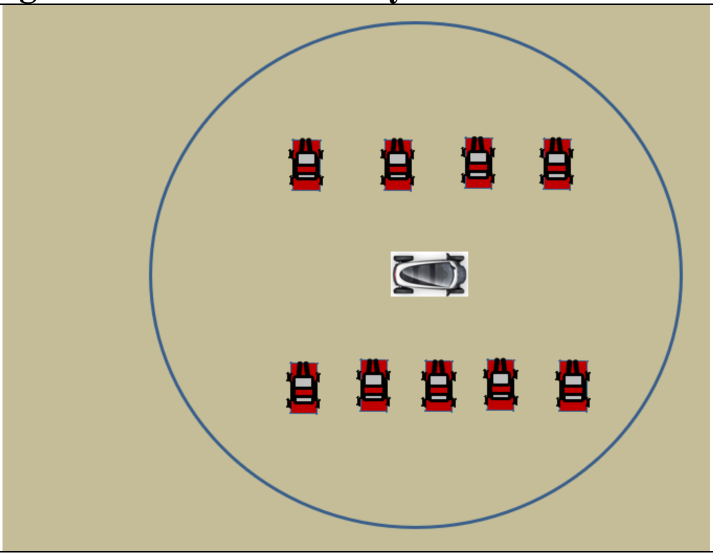
[18] "CARLA Vehicle Simulator," [Online]. Available: https://carla.org/.

[19] "NVidia Forum," [Online]. Available: https://forums.developer.nvidia.com/t/low-execution-time-when-running-4-cameras-in-parallel-drivenet-n-cameras-sample/145708/7. [Accessed 27 August 2020].

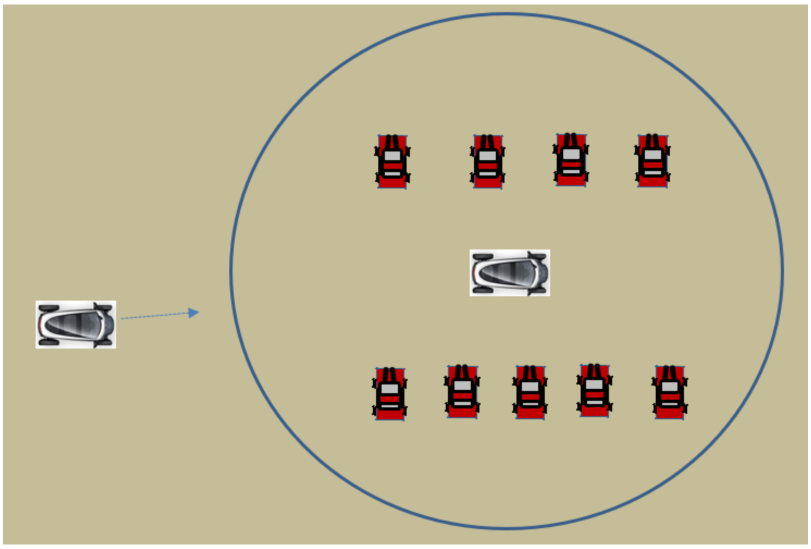
Appendix A.

Driving Scenarios:

DS 1 - CACC

Context
Ego-vehicle driving beside a platoon of target vehicles on a highway.
Actors
Target vehicles (Twizy’s), other vehicles, Lane markings, Lighting conditions, Road conditions
Preconditions
1) The target vehicles are driving in a platoon 2) There are other vehicles around the ego-vehicle 3) The visibility is clear, and the weather is sunny 4) The SVS system is inactive
Trigger
1) SVS system is activated.
Expected ego-vehicle behaviour
1) The ego-vehicle detects and classifies the vehicles which are clearly or partially visible to the cameras as a "Target"(Twizy) or "other vehicle"(cars) around its surrounding
2) The ego-vehicle calculates the pose of the detected target vehicle (Twizy) with respect to itself.
3) The ego-vehicle calculates the position of other detected vehicles.
4) The ego-vehicle computes the curvature and position of the left and right lane markings belonging to the ego vehicle if present
5) The ego-vehicle detects the free space around the vehicle
6) The ego-vehicle displays all perceived information to the operator
Postconditions
1) Detections are provided until the system is deactivated
Key Performance Indicators (KPIs)
1) The number of false negatives detected. (objects incorrectly classified have a higher risk)
2) The detection range of the system. (How far can an object be accurately detected)
3) The accuracy at which the position of objects are estimated
4) The number of target vehicles the system can handle (detect as well as provide pose).
5) The time required to detect and estimate the position of vehicles/objects
6) The accuracy of the vehicle position estimate with respect to the ground truth
7) The execution time required to compute the pose of the target vehicle when detected
8) The accuracy of the computed pose of the target vehicle when detected
9) The rate at which the system updates the position of the targets and other vehicles on the CAN bus

DS 2: Parking Lot Vehicle Stationary

Context
1) Other vehicles parked around the ego Vehicle
2) The visibility is clear.
Actors
Other vehicles
Preconditions
1) There are vehicles parked around the ego vehicle (5-20m radial distance from ego-vehicle).
2) The system is inactive.
Trigger
1) The system is activated
Expected system behaviour
1) The system detects and classifies the vehicles as a target (Twizy) or other vehicle(cars) around its surrounding.
2) The system calculates the position of all vehicles.
3) The system estimates the pose for all target vehicles.
4) The system detects the free space around the ego-vehicle
5) The system publishes all the computed information on the CAN-bus
Postcondition
1) Keeps providing detections till the system is deactivated.
KPIs
1) The number of true detections perceived by the system. Precision, recall, F1score.
2) The max number of detections the detector can handle.
3) The max position of all vehicle from the ego vehicle for detecting the target.(max size of the object in image to be classified suitable for detections)
4) The time required to estimate a detection when the detector is activated. (as a factor of number of detections)
5) The accuracy of the estimated vehicle positions with respect to the ground truth value.

DS 3: Parking lot Vehicle Approaching



Context

1) The visibility is clear.

2) The ego vehicle is in a parking lot with more than one vehicle around it

Actors

Target vehicles(Twizy), other vehicles

Preconditions

1) The system is acitve.

2) The target vehicle is initially out of range and driving towards the ego vehicle

Trigger

1) The target vehicle enters in the range of the ego vehicle's cameras.

Expected vehicle behaviour

1) The system detects and classifies the target vehicles which are clearly or partially visible to the cameras as a “Target”(Twizy) or “other vehicle”(cars) around its surrounding.

2) The system calculates the pose of the detected target vehicle with respect to ego vehicle

3) The system calculates the position of all vehicles.

4) The system publishes all the computed information on the CAN-bus

Postcondition

1) Keeps providing detections till the detector is deactivated.

KPIs

1) The number of true detections perceived by the system. Precision, recall, F1score.

2) The max position of all vehicle from the ego vehicle for detecting the target.(max size of the object in image to be classified suitable for detections)

3) The minimum pixel area of a vehicle required to be worthy for detection

4) The number of target vehicles the detector can handle (detect as well as provide pose) .

5) The time required to estimate a detection when the detector is activated. (as a factor of number of detections)

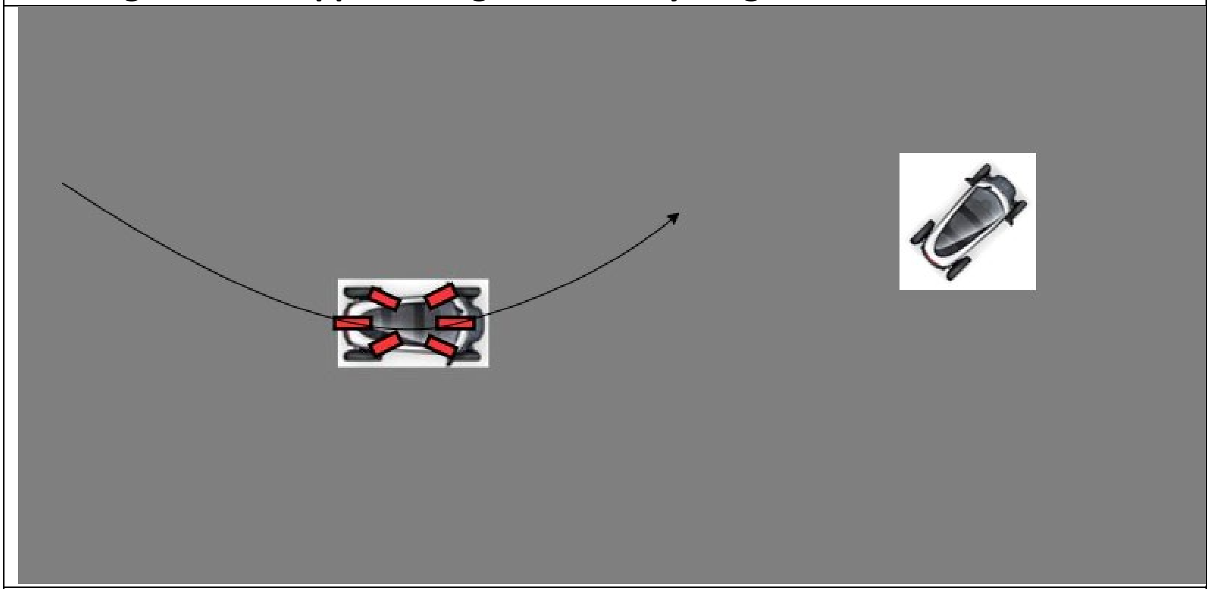
6) The accuracy of the estimated vehicle positions with respect to the ground truth value.

7) The time required to compute the pose of the target vehicle when detected.

8) The time required to translate CAN messages and publish on the CAN bus.

9) The rate at which the system updates the position of the targets and other vehicles on the CAN bus

DS 4: Ego Vehicle Approaching a Stationary Target



Context

1) The ego vehicle moves forward towards a Target vehicle (Twizy) parked in front of it in a curved path as shown in above figure.

Actors

Target vehicles (Twizy)

Preconditions

1) The system is active.
2) The visibility is clear.

Trigger

1) The target vehicle starts approaching the target vehicle following a curved path.

Expected system behaviour

1) The system detects and classifies the vehicle as a target vehicle(Twizy).
2) The system calculates the pose of the detected target vehicle with respect to ego vehicle

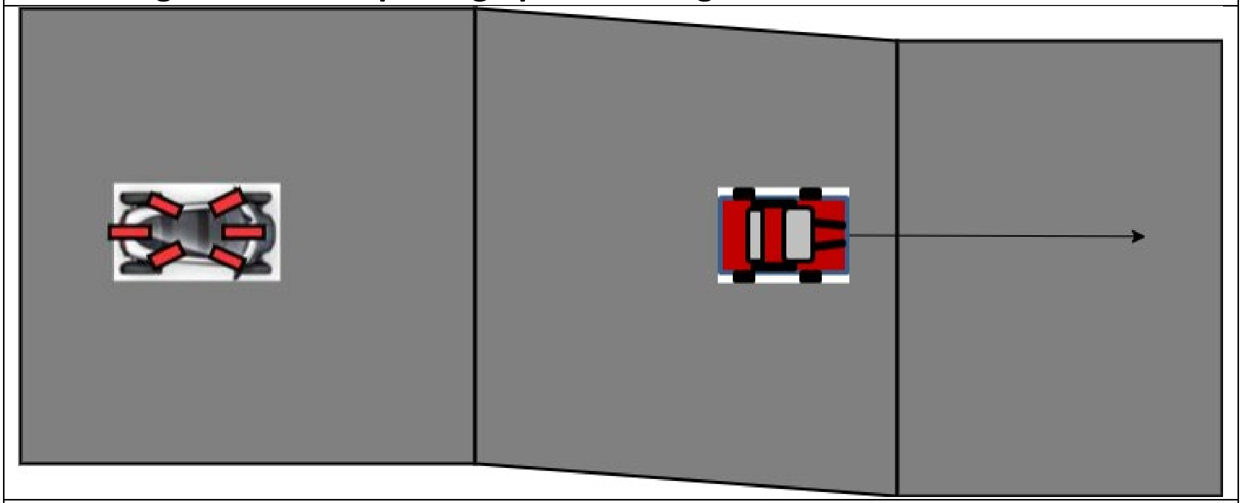
Postcondition

1)Detections are provided until the target gets out of the FOV of the cameras.

KPIs

1) The accuracy of classification of the detected vehicle.
2) The accuracy of the pose estimate of the target with respect to intervehicle distance
3) The variance in the measurement of pose data.
4) The time required to compute the pose of the target vehicle when detected.
5) The rate at which the system updates the position of the targets and other vehicles on the CAN bus

DS 5: Target Vehicle Departing uphill from Ego-Vehicle



Context

1) The ego vehicle is stationary, and a preceding vehicle moves away from the ego vehicle up a ramp.

Actors

Other vehicles (Cars), Road Uphill gradient

Preconditions

1) The system is active.

2) The visibility is clear.

Trigger

1) The preceding vehicle starts from the same plane as that of the ego vehicle and drives up a ramp.

Expected system behaviour

1) The system detects and classifies the vehicle as “other vehicle” (Non-target).

2) The system calculates the position of the detected vehicle with respect to ego vehicle

Postcondition

1)Detections are provided until the target gets out of the FOV of the cameras.

KPIs

1) The accuracy of classification of the detected vehicle.

2) The accuracy of the position estimate of target with respect to intervehicle distance

3) The variance in the measurement of position data.

4) The time required to compute the position of the vehicle when detected.

5) The rate at which the system updates the position of the targets and other vehicles on the CAN bus

Appendix B.

Example description of camera Rig Configuration file for front Camera.

```
    "rig": {
        "sensors": [
            {
                "name": "Front_60FOV",
                "nominalSensor2Rig": {
                    "quaternion": [
                        -0.3536,
                        0.3536,
                        -0.6124,
                        0.6124
                    ],
                    "t": [
                        0.0,
                        0.0,
                        1.42
                    ]
                },
                "parameter": "",
                "properties": {
                    "Model": "pinhole",
                    "cx": "9.674573781498158e+02",
                    "cy": "5.939498223275325e+02",
                    "distortion": "-0.444577318609922 0.224404687754395",
                    "fx": "1.935708341198060e+03",
                    "fy": "1.934157368441432e+03",
                    "height": "1208",
                    "params": "",
                    "width": "1920"
                },
                "protocol": "camera.virtual",
                "sensor2Rig": {
                    "quaternion": [
                        -0.3536,
                        0.3536,
                        -0.6124,
                        0.6124
                    ],
                    "t": [
                        0.0,
                        0.0,
                        1.42
                 ]
                }
            },]
       }
```

Appendix C.

Network Compatibility with DriveWorks:

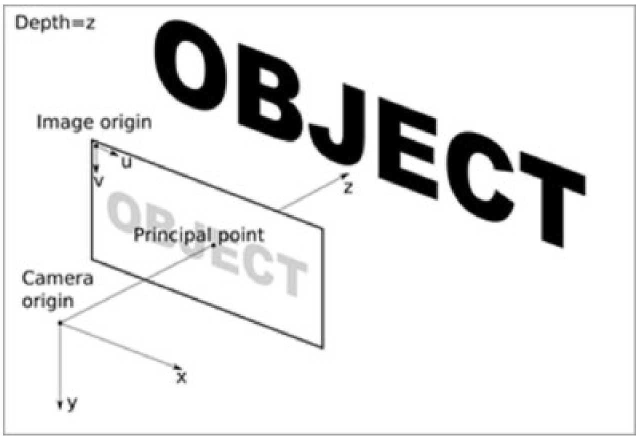
A deep neural network (DNN) is composed of several layers which perform different mathematical operations. Each layer must be compatible with the TensorRT optimization tool provided for DriveWorks to integrate with the DriveWorks APIs. This constraint limits the number of possible network choices.
The following Network types were evaluated to meet the above requirements and constraints.

Network Name	Compatibility Status
MobileNet-SSD	Incompatible due to Permute, Prior-Box Layers
YOLO V3	Incompatible due to Upsample, Flatten, Leaky Relu layers
YOLO V1-Tiny	All Layers compatible
DetectNet	All Layers compatible

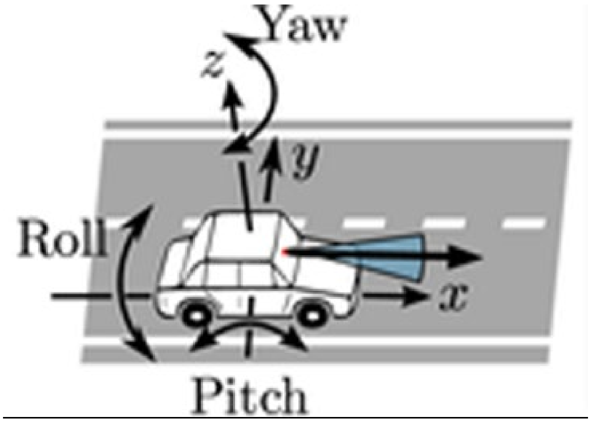
Appendix D.

Coordinate Systems

The camera has a right-handed coordinate system, where the camera origin is at the optical centre of the left camera. The x-axis points to the right of the image plane, the y-axis points to the bottom of the image plane, and the z-axis points forward along the optical axis.



Vehicle Coordinate System: The vehicle uses a right-handed coordinate system, where the vehicle origin is considered to be under the center of the rear axle. The x-axis points forward to the front of the vehicle, the y-axis points to the left of the vehicle, and the z-axis points to the upward of the vehicle.



About the Author

Ashton Menezes is an automotive systems designer with a background in Vehicle Dynamics and Controls. He enjoys working on multi-disciplinary projects related to robotics and software applications.

Since November 2018, he is working on aspects related to automotive systems design where he has been trained on the systems aspects for solving automotive design-related problems. During this training, he assisted various automotive companies in designing next-generation driver assistance systems and defining concepts for automated driving.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng AUTOMOTIVE SYSTEMS DESIGN
Track AUTOMOTIVE SYSTEMS DESIGN

EINDHOVEN
UNIVERSITY OF
TECHNOLOGY