TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Mustafa Furkan Kopar 194236IASM

Camera-Based Vehicle Location Detection

Master's thesis

Supervisor: Uljana Reinsalu
PhD
Jürgen Soom
MSc
Mairo Leier
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mustafa Furkan Kopar 194236IASM

Kaamerapõhine sõiduki asukoha tuvastamine

Magistritöö

Juhendaja: Uljana Reinsalu
PhD
Jürgen Soom
MSc
Mairo Leier
PhD

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mustafa Furkan Kopar

02.08.2021

Acknowledgement

I would like to state my appreciation to everyone who has been supporting me during the development of this thesis, especially my supervisors, family, and friends. The work presented here could not come to fruition without all these people.

I would like to thank my supervisors who have been addressing all the necessary questions so that the thesis can improve as much as possible. Their feedback has been extremely helpful.

I wish to thank my mother, my father, and my sister for always believing in me and supporting me whenever I needed motivation. Their faith helped me wanting to contribute to the study more and more.

I wish to thank my dearest friends who always try their best to come to my aid in times of need. Especially, Ibrahim Koc, Gulriz Yaman, and Berke Nohut deserves all of my appreciation for their valuable opinions and contributions in this process.

Lastly, all of my love and thanks go to my beautiful niece Defne. Her smile has always been my brightest light even in the darkest days.

Abstract

Detecting the precise locations of objects from the image data collected by a monocular camera continues to be a challenge getting more attractive day after day. Although the task is less effortful with the usage of dual cameras, the advantages a monocular camera brings to the concept are quite appealing. One of the main reasons this issue is particularly serious nowadays is the necessity of this technology by self-driving vehicles that are soon going to become an indispensable part of people's lives. Autonomous transportation is surely a leading motivation for advancing image processing applications. On the other hand, focusing only on one possible use case to develop a coordinate estimation unit with a monocular camera can be degrading for other industries in need of advanced image processing features. As the monocular front cameras implemented in vehicles stand parallel to the ground, the algorithms they use for modifying the visual data certainly differ from the ones installed with a pitch angle to survey a specific location.

An obvious case where an inclined monocular camera with the ability to map the object coordinates is the surveillance camera system. Particularly, this project aims to address the demand for such an application in roll-on/roll-off ships. The vessels, like ferries, that are carrying vehicles as cargo need to make sure that these vehicles are placed in their assigned locations correctly. Therefore, the vessels need to properly locate the vehicles at any time so that the cargo can easily be tracked down during loading, transportation, and unloading. Camera-Based Vehicle Location Detection is developed to cover this need. The system helps create a projection of the vessel it is implemented on, by monitoring the cargo and providing data regarding their positions. The parking deck of the vessel is recorded with the usage of cameras whose coordinates on the vessel are fixed. Using the camera feed and the distance estimation algorithm, the system deduces the exact coordinates of each cargo on the vessel.

There are two main challenges concerning the development of the project. First, vehicle detection from the real-time camera feed should be carried out to obtain the areas of interest in the point cloud. Then, the spatial locations of the detected objects are deduced

with respect to the location of the observing camera. The collected data is processed to deduct the distance of every detected vehicle in the vessel with respect to the camera. These deductions can later be utilized to create a mapping of the vessel where the location of each detected vehicle is indicated.

The thesis shows the existing related work regarding vehicle detection and distance estimation as well as the motivation behind the project development while also touching on the method descriptions and test results of some selected algorithms.

This thesis is written in English and is 45 pages long, including 6 chapters, 16 figures, and 3 tables.

List of abbreviations and terms

AGV	Automated Guided Vehicle
API	Application Programming Interface
BA	Bundle Adjustment
BiFPN	Bi-Directional Feature Pyramid Network
CNN	Convolutional Neural Network
EP	Error Percentage
FN	Frame Number
FP	False Positive
FPN	Feature Pyramid Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
ID	Identifier
LiDAR	Light Detection and Ranging
mm	millimeter
MP	Megapixel
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
NMS	Non-Maximum Suppression
PANet	Path Aggregation Network
QR	Quick Response
RADAR	Radio Detection and Ranging
R-CNN	Region-Based Convolutional Neural Network
RFID	Radio Frequency Identification
RMSE	Root Mean Squared Error
Ro-Ro	Roll-on / Roll-off
RoI	Region of Interest
RTLS	Real-Time Location System
SAM	Self-Attention Module

SAT	Self-Adversarial Training
SfM	Structure-from-Motion
SONAR	Sound Navigation and Ranging
SPP	Spatial Pyramid Pooling
SSD	Single Shot MultiBox Detector
XML	Extensible Markup Language
YOLO	You Only Look Once

Table of contents

List of figures

List of tables

1 Introduction

Advancements in image processing have brought about an increase in the usage of visual data for various computation tasks. Applications like object detection, image classification, pattern recognition, segmentation, tracking, and more have been addressing different needs in various fields including but not limited to medical practices, machine vision, smart vehicles, and surveillance. Even though some of the common computer vision implementations are used in distinct disciplines, many of the state-of-the-art object detection and distance estimation algorithms have been developed to accommodate the demands of autonomous vehicles as this is a significant area of current interest. On the other hand, considering the fact that this technology focuses on visual data that is parallel to the ground, other domains of development like surveillance requiring the manipulation of data from images with some pitch angle usually cannot utilize the advancements of the visual data processing for the self-driving vehicles without further modifications. Hence, building dedicated algorithms fulfilling the need of processing angled images correctly calls for additional study to alter the algorithms for driverless transportation.

Having said that the surveillance systems are generally in need of more advanced computer vision applications for certain cases due to their pitch angle, this thesis work in fact stems from a special use case of such systems, namely the roll-on/roll-off (Ro-Ro) ships. Cargo ships that carry vehicles have long been in use for transporting their loads across a body of water. These vessels, also referred to as the Ro-Ro ships, are equipped to transport numerous types of cargo including private cars, buses, vans, semi-trailers, project cargo, and passengers [1]. To be able to properly load and unload their cargo, these ships should have the ability to monitor the positions of the vehicles on the vessel at all times. When each vehicle has been assigned to a certain location on the vessel, loading and unloading the cargo can be carried out in as little disarray as possible and more rapidly. However, the ships require a system to make sure that the vehicles are positioned at the right spot during the trip. Camera-Based Vehicle Location Detection

provides a mapping of the vessel indicating the real-time position of each vehicle on the ship so that each of them can be tracked down accurately.

The system is mainly a real-time embedded platform that takes the video feed as input which is coming from the cameras implemented on the parking deck and produces a mapping of the vessel as output. The location detection algorithm along with the distance estimation determines the exact position of the object with respect to the camera. Using the obtained position and the coordinates of the camera, the system can deduct the coordinates of the vehicle from the corresponding image. These coordinates can then be utilized to come up with a complete mapping of the vessel where the position of each vehicle is shown in real-time.

The work presented in this thesis focuses on the detection of the vehicles from a camera system with a pitch angle and then calculating the distances of the detected vehicles from the observing system. First, some of the most commonly used object detection methods are analyzed so as to come up with a satisfactory vehicle detection implementation in terms of accuracy, speed, ease of use, and real-time compatibility. Afterward, various distance estimation techniques introduced in the literature are studied to obtain accurate distance results from monocular camera images. The most trusted conventional methods rely on the utilization of an additional sensor to calculate the distance between an object and the system despite its various disadvantages. To propose a more compact and still reliable solution, this thesis aims to estimate the distances only from the camera image and additional parameters that can easily be procured or calculated. The essential criteria for suitable performance are precision, low calculation complexity to fulfill the embedded resource requirements, and ease of parameter tuning at setup. The outcomes of these techniques are verified with the usage of the data from a radar sensor.

The remainder of the paper is organized as follows. Section 2 presents the existing state-of-the-art related to the Ro-Ro ship organization, vehicle detection, and distance estimation using image processing. Section 3 provides the methods taken into account during the algorithm development for the project while Section 4 introduces the implementation procedures of these methods. Section 5 demonstrates the results of the implementations addressed in Section 4 discussing their advantages and disadvantages. Finally, Section 6 concludes the paper with a summary.

2 Literature Review

The existing methods and technologies that are of help for developing the Camera-Based Vehicle Location Detection are studied in three main divisions: cargo organization in Ro-Ro ships, object detection, and distance estimation. The research on effective freight placement in Ro-Ro ships is conducted so as to describe the need for this work more clearly. To obtain a camera-based solution, the project integrates two topics in image processing so that it can generate the cargo coordinates, that is, vehicle detection and distance estimation while coordinate evaluation is also carried out during distance estimation.

2.1 Cargo Organization in Ro-Ro Ships

The proper organization and handling of the cargo in the Ro-Ro ships have been being studied for some time. The new applications in various tracking technologies enabled plenty of research to be conducted to optimize the cargo tracking techniques. Applying Radio Frequency Identification (RFID) technology to create a Real-Time Location System (RTLS) [2] was proposed to enhance the port operation system performance. This technology focuses on large cargo ships whose containers need to be tracked down in real-time. Although it addresses a similar need in transportation, this method is not intended for vehicle tracking in smaller Ro-Ro ships such as ferries. Furthermore, RFID was intended to be utilized along with GPS to obtain the real-time position of the loaded goods as GPS coordinates [3]. Yet, this study also lacks to consider some particular types of loads like wheeled cargo. Another technology suggests using Automated Guided Vehicles (AGV) for managing the material flow while AGV placement should also be controlled by the RFID transponders on the floor [4]. Like the previous ones, the feasibility of this approach is low for handling wheeled cargo on ferries. Other technologies such as QR code, barcode, or magnetic ID card reading [5] were proposed for addressing vehicle tracking in Ro-Ro ships. This thesis proposes a more instantaneously accurate solution to the tracing of the cargo vehicles in a Ro-Ro ship during their loading, transportation, and unloading.

2.2 Object Detection

The number of distinct and improved vehicle detection techniques has been rapidly increasing as more and more object detection algorithms are studied. The vehicle detection in the Camera-Based Vehicle Location Detection requires to be carried out with a real-time fast algorithm trained by the most compact dataset possible so that the performance of the detection can be optimized. This is expected to be achieved by analyzing the state-of-the-art vehicle detection algorithms to choose the most suitable one and revising their neural network architectures to check if further optimizations are possible. The object detection algorithms that can also be implemented for detecting vehicles are usually studied in two categories [6].

2.2.1 One-Stage Detector

Single-stage detectors behave as simple regression models that try to learn the probability score and the coordinates of the bounding box of the object from the image [7]. You-Only-Look-Once (YOLO) scheme is one of the most well-known implementations of this segmentation technique. YOLOv3 is widely used for practical purposes and further studied for implementations. Including dilated convolution and self-attention module (SAM) to YOLOv3 yielded efficient and accurate SA-YOLOv3 [6]. Another improvement to YOLOv3 making it faster and more accurate is released as YOLOv4 in April 2020, utilizing the CSPDarknet53 backbone rather than the Darknet53 network of YOLOv3 [8]. An utterly different release in PyTorch implementation called YOLOv5 is also made available the same year being even faster and more accurate than EfficientDet [9].

EfficientDet is also a fast, small, and accurate one-stage detector family proposing a weighted bi-directional feature pyramid network (BiFPN) and a compound scaling method [10]. Yet another common detection method is Single Shot MultiBox Detector (SSD) with a straightforward approach as the proposal generation and subsequent feature resampling stages are removed from the computation [11].

2.2.2 Two-Stage Detector

Two-staged detectors define a region of interest from the input image before object segmentation, making this a two-stage process. They are usually slower than one-stage detectors yet reaching better accuracy rates [7]. The majority of these detectors benefit

from specific artificial neural networks for image processing called Convolutional Neural Networks (CNN). These models execute convolution operations in some of their layers where a feature map describing the input properties is generated [14]. Therefore, the neurons in the convolutional layers are able to provide their findings of the part of the image they apply the convolution to. The distribution of work results in the creation of smaller imagery outputs which are to be inputted to the following layer.

One of the most studied two-stage object detector models is the Region-based Convolutional Neural Network (R-CNN) and its derivatives. R-CNN generates a set of candidate detections constructed by the region proposals regarding the whereabouts of the object, which is fed to the convolutional neural network for the segmentation [12]. This approach is further improved by the Fast R-CNN which produces a feature map speeding up the segmentation process [13]. Later, Faster R-CNN is introduced which adds a region proposal network before the Fast R-CNN to generate object proposals with objectness scores [14]. Faster R-CNN is expanded by Mask R-CNN which introduces a parallel unit to predict segmentation masks on every Region of Interest (RoI), keeping the classification and bounding box regression section [15]. Other two-stage detectors to be considered for the Camera-based Vehicle Location Detection project are the SINet, introducing context-aware region of interest pooling and multi-branch decision network techniques for fast object detection with a large scale [16], and SqueezeNet, a very compact object detection model for better on-chip implementation results in regards to the model size [17].

This thesis work analyses the state-of-the-art detector models addressing the needs of Camera-Based Vehicle Location Detection. The results of these analyses can provide information about the model to be used so that the most suitable vehicle detection scheme can be implemented in the design.

2.3 Distance Estimation

Evaluating the correct distance of the objects with respect to the observing camera has been a widely studied challenge especially due to the rapidly growing autonomous vehicle technology. Calculating the real distances of the objects from certain points of view is a subject also studied by 3D reconstruction techniques such as Structure-from-Motion (SfM). SfM utilizes a set of images of a certain environment to come up with a 3D model

while applying steps like feature extraction, matching, geometric verification, two-view reconstruction, image registration, triangulation, outlier filtering, and bundle adjustment (BA) [18]. The resulting 3D model also reveals the object distance to the implementation points although the construction of the model requires marker objects to be present in the environment making the performance of this method bound to the layout preparation.

One approach utilizes the triangulation method and the binocular camera model for calculating the target distance [19], while another one introduces sparse 2D laser range data to estimate the monocular depth of the object [20]. A different technique for distance estimation is to use the disparity of the target object detected by the stereo camera system [21]. Machine learning methods such as regression modeling are also inspected to be used in distance estimation where a predictive model for calculating the target distance is created [22]. Some well-known distance estimation algorithms like pose from orthography and triangle similarity were also analyzed to compare their performances which resulted in favor of the triangular similarity method [23].

Camera-based Vehicle Location Detection aims to perform a distance estimation that is very fast, lightweight, and accurate. To satisfy these needs, the existing models are inspected, and necessary implementations are carried out so that the platform can correctly estimate the object distance with respect to the monocular camera while considering the coordinates of the object for mapping.

3 Methods

The implementation requirement of Camera-Based Vehicle Location Detection essentially consists of developing two types of algorithms sequentially. First, some methods for vehicle detection are performed for comparison. After this step is done, the images with the determined bounding boxes of the detected objects are inputted to various distance estimation algorithms to assess the outcomes. These two successive branches are described in detail under the corresponding headings.

3.1 Object Detection Methods

This thesis work mainly investigates some of the most commonly known single-stage detector algorithms in detail and compares their results on a set of surveillance images. The detectors implemented for the work are MobileNet SSD, YOLO v3, and YOLO v4. The main reasons why these are selected are their availability, ease of use, speed, and various size options. Since these approaches are utilized extensively, they offer numerous sources to be employed providing rather short development time. Moreover, as the idea behind their algorithm is not extremely complex and the models are easily available for use, configuring these methods for special needs is quite straightforward.

Conventionally, one-stage detectors operate at a considerably higher speed compared to two-staged ones since they do not waste operation time to define a region of interest. The presence of an additional step made the two-stage algorithms got discarded from the scope of this thesis, thus only a selection of single-stage detectors is implemented. Furthermore, every YOLO detector comes with differently-sized packages, namely tiny and full models, making them a desired choice for various applications in terms of size restrictions.

3.1.1 MobileNet SSD

The Single Shot MultiBox Detector is a single-stage object detection framework based on deep networks which, unlike the two-stage frameworks, operate without pixel resampling and bounding box hypothesizing. The two-stage detection systems rely on

these two steps and applying a high-quality classifier. Omitting the aforementioned phases results in a considerable improvement in the object detection speed. Moreover, SSD reaches large accuracy rates as a consequence of the use of a small convolutional filter for object class and offset prediction, particular filters for distinct aspect ratio detections, and the application of the said filters to several feature maps from the later stages of a network so that the object detection can be carried out at multiple scales [11].

The main advantage SSD brings is the computational speed in object detection. However, even with the improvements leading to high accuracy rates, SSD is not as accurate as the existing single-shot detection models. The first base network SSD is introduced with was VGG16 having six feature maps with specific dimensions for the back-end network. The outputs of the network model are later fed to a non-maximum suppression (NMS) method where the detection with the highest confidence is selected as the final output. Despite its ability of good feature extraction, the network architecture is in fact rather sizable to be convenient for real-time systems [24]. Figure 1 presents a basic illustration of VGG16-SSD architecture starting from the input image until the NMS to create the output with the detections having the highest confidence scores.

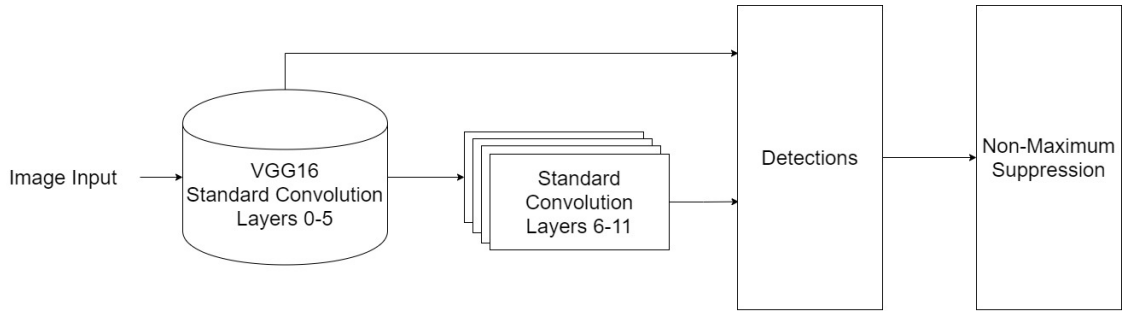


Figure 1. The network architecture of an SSD framework with VGG16.

In order to compensate for the large size and complexity that VGG16-SSD has, Google implemented the SSD framework replacing the VGG16 network with the MobileNet backbone model. Even though this network model results in a certain amount of decrease in the object detection accuracy, it is rather compact and therefore a friendly detector for real-time embedded systems. Figure 2 describes the main steps in a MobileNet-SSD architecture where the input image is fed to the backbone network and the convolution layer outputs produce the detection unit input. A non-maximum suppression is again used to eliminate the detections with low confidence. Hence, the front-end structure remains the same while the MobileNet network as the backbone leads to better real-time performance.

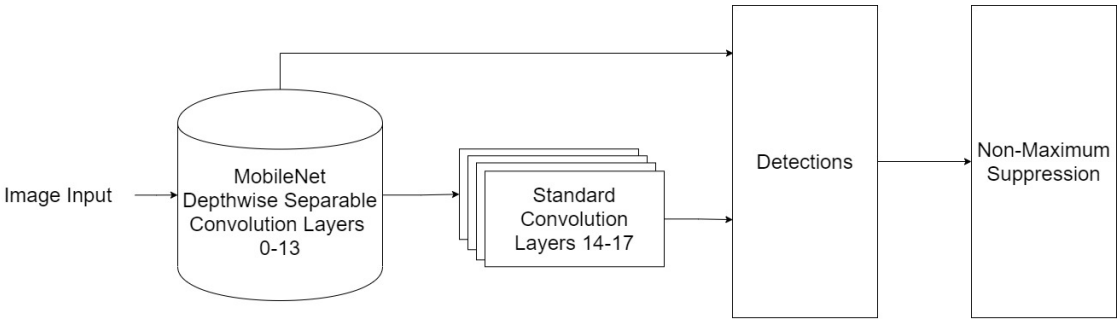


Figure 2. The network architecture of an SSD framework with MobileNet.

Further improvements on MobileNet-SSD were made with MobileNet-SSDv2 to increase the speed and the accuracy rate along with the required memory size. This version introduces a Feature Pyramid Network (FPN) before the detections unit to enhance the back-end detection network execution [24].

3.1.2 YOLO v3

YOLO is one of the most renowned detectors in usage on account of its speed and accuracy. The system is fast and simple as it regards object detection as a single regression problem rather than repurposing classifiers as the two-stage detectors do. Furthermore, contrary to the region proposal techniques like sliding window, YOLO takes the entire image into account while predicting objects, eliminating the background errors [25]. Even though this approach might harm the accuracy, the results generally exceed expectations. Succeeding the first version, YOLOv2 and YOLO9000 aim to improve the accuracy while remaining fast. Therefore, approaches such as batch normalization, k-means clustering, and high-resolution classifier along with a custom network called Darknet-19 result in a better, faster, and stronger detector [26].

YOLOv3 proposes some advancements in detecting small objects by introducing a superior bounding box prediction with logistic regression. Feature extraction is performed with another backbone network with an increased number of convolution layers called Darknet-53. Having 53 convolutional layers instead of the old 19-layered network, the network architecture of YOLOv3 has become more powerful than YOLOv2 [27].

YOLOv3-tiny is yet another variant of YOLOv3 where the detector can also address the object detection needs of technologies that the speed and the size of the framework become of utmost importance. By decreasing the depth of the network, YOLOv3-tiny sacrifices the accuracy of the detections while making the overall system approximately 442% faster than the former variants of YOLO [28]. The reduction of the convolution

layers also allows this variant to have a smaller size making it an appealing implementation for real-time embedded systems that are aiming for fast and small algorithms.

3.1.3 YOLO v4

A state-of-the-art improvement to the YOLO detector was made with the release of YOLOv4. As the backbone of the YOLOv4 detector, the CSPDarknet53 neural network is utilized while Spatial Pyramid Pooling (SPP) is added over it to enlarge the receptive field. FPN in YOLOv3 is replaced with Path Aggregation Network (PANet) serving as the neck of the architecture. Furthermore, a new data augmentation technique Mosaic was introduced along with Self-Adversarial Training (SAT). All of these improvements lead to an increase of 10% in precision and 12% in speed compared to YOLOv3 [8].

A lightweight variant of YOLOv4 is introduced as YOLOv4-tiny where the CSPDarknet53 backbone network is replaced with CSPDarknet53-tiny. Unlike YOLOv4, this detector still utilizes FPN while SPP and PANet are removed in order to ensure rapid detection so that the model can be a more appropriate choice for real-time mobile and embedded systems [29]. Just like with the YOLOv3 and YOLOv3-tiny, this small-sized fast alternative obtains its qualities by relinquishing accuracy. Yet, the advantages of the tiny variant can outweigh the drawbacks when used in a real-time embedded system.

3.2 Distance Estimation Methods

The approaches presented in this thesis for estimating the object distances all make use of a camera implemented to a location with a pitch angle. The real-time feed from the camera is first to be put into a certain object detection algorithm for detecting the bounding box coordinates of the objects to be investigated. The object detector used for testing the distance estimation methods is YOLOv4. The reason why this method is chosen is to be able to detect as many objects as possible in a frame. Although the real-time needs of the Camera-Based Vehicle Location Detection project suggest a faster and smaller detector such as YOLOv4-tiny, the distance estimation testing requirements mostly aim for the ability to detect more objects for having as many results as possible to be compared. Furthermore, the reasoning for selecting YOLO v4 for the distance estimation experiment is also clarified in Section 5.1 with the object detection

experimental results as YOLO v4 proved to have the capability of making more and accurate detections.

The project intends to use only a camera for estimating the distances of the objects. Other methods for distance estimation such as RADAR, LiDAR, and SONAR sensors are in fact more accurate than using vision-based estimation. However, while these systems can be attractive to large systems, a small embedded system usually requires a simpler solution. The aforementioned techniques demand an extra sensory device to be implemented to the design, and this additional hardware definitely increases the complexity of a compact embedded system. Moreover, the added equipment for distance estimation increases the cost of the project making a small system to become unnecessarily expensive. As a reasonable development price is another requirement of the Camera-Based Vehicle Location Detection system, not including extra hardware and utilizing the already existent visual resource is considered as a goal of this thesis.

This work mainly investigates three techniques to be utilized in vision-based distance estimation. The first one makes use of the intrinsic camera parameters, while the second approach that is examined carries out a geometrical analysis of the environment to come up with the location of the objects in the frame. Lastly, a machine learning model has been trained on a sample implementation to predict the object coordinates on the images.

3.2.1 Using Camera Parameters

The basic functionality of a camera is to capture the light rays emitted from the objects via its lens and collect these light rays reflected from the lens on an image plane. The working principle of a camera lens is described in Figure 3.

The lens mainly generates a virtual representation of what has been captured from the object plane. The distance between the center of the camera lens and the image plane (film) is defined as the focal length in photography. Focal length can in fact be described as a representation of the object distance from the camera lens. The object plane represents the vertical plane in the real world where the object is in, whereas the image plane is the virtual depiction of the object plane produced by the camera lens.

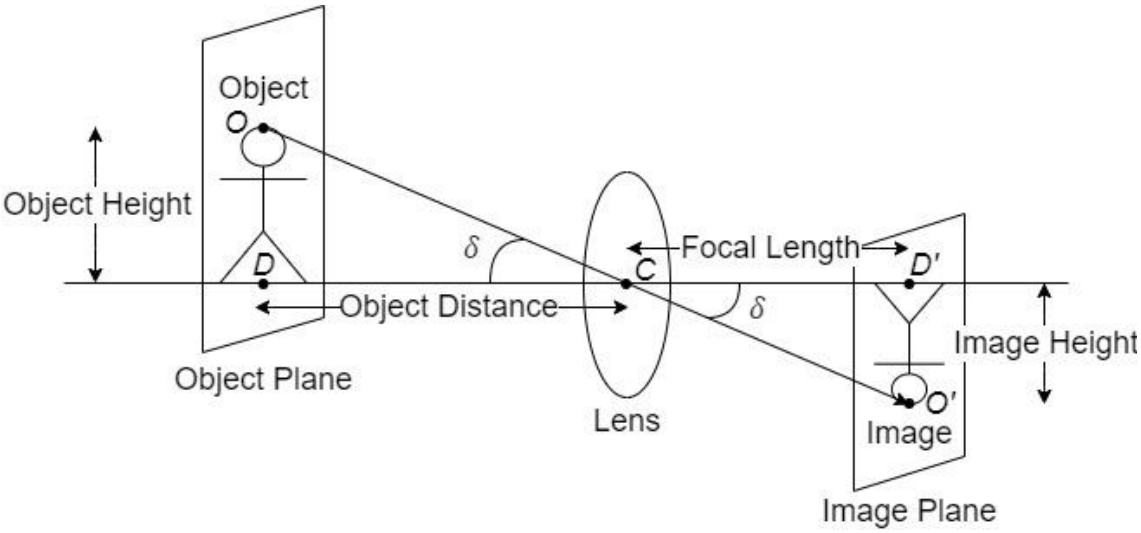


Figure 3. The basic working principle of a camera lens.

With the provided analysis, it can be concluded that the *ODC* triangle formed in the real space and the *O'D'C* triangle generated in the image space are similar. This triangle similarity can be utilized in estimating the object distance in the real space from the camera. Therefore, the proportional relationship between the real distance of the object and the distance of the image is as follows [30]:

$$D = \frac{f \times O}{I} \tag{1}$$

In Equation (1), D is the distance of the object to the camera lens in real space, f is the focal length of the lens, O is the height of the object in real space while I represents the height of the object on the image plane. The unit of all of the variables in Equation (1) is in millimeters. However, after the object detection is performed with a detector algorithm, the output result is the bounding box coordinates giving the object height on the image plane in pixels. Hence, a way to convert these pixel values to millimeters to obtain the value of I is required. To be able to carry out this conversion, the size of the image sensor is needed. This size is different in both horizontal and vertical axes meaning that two new equations are obtained to estimate the object distance:

$$D = \frac{O_x \times f \times P_x}{x \times S_x} \tag{2}$$

$$D = \frac{O_y \times f \times P_y}{y \times S_y} \tag{3}$$

O_x in Equation (2) is the dimension of the image in the horizontal plane, and S_x is the sensor height. Both values are measured in millimeters. Moreover, P_x is the horizontal sensor size while x represents the image dimension in the horizontal plane where these

two values are in pixels. Similarly, O_y in Equation (3) depicts the dimension of the image in the vertical plane in millimeters as y represents the same aspect in pixels. Furthermore, S_y is the sensor width measured in millimeters, and P_y demonstrates the vertical sensor size in pixels.

Even though Equation (2) and Equation (3) are able to give a general idea of how the distance of an object can be predicted, this distance is in fact valid provided that the object and the image planes are parallel as illustrated in Figure 3 [30]. However, the environment where the cameras are implemented for the Camera-Based Vehicle Location Detection system requires these devices to have some pitch angles larger than zero meaning that the object and the image planes are not parallel anymore. A straightforward approach to address the pitch angle problem is basically to divide the distance found in Equation (2) or Equation (3) by the cosine of the pitch angle of the camera resulting in the following relationship [31]:

$$D_O = \frac{D}{\cos\gamma} \tag{4}$$

Equation (4) describes the distance of the object standing with a pitch angle with respect to the image plane denoted by D_O. D is the parallel distance calculated by Equation (2) or Equation (3) while γ is the pitch angle of the camera. The relationship between these variables is also illustrated in Figure 4. The distance between the object in the real space and the camera is now indicated as the oblique distance while the parallel distance denotes how far the camera is away from a virtual representation of the object where the image plane and the virtual object are parallel. In other words, parallel distance is the vertical projection of the oblique distance onto the ground.

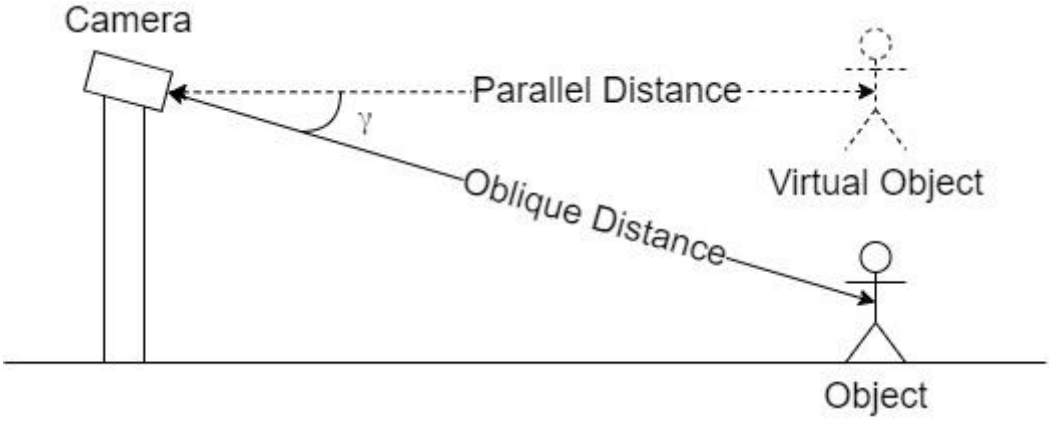


Figure 4. The relationship of the parallel and oblique distances with an inclined camera system.

The approach described in this section is able to give an intuition about the distance of an object to the camera. However, it should be noted that the algorithm depends highly on the intrinsic camera parameters such as the sensor size and focal length. The study proposing this method [30] was conducted with high-quality cameras which makes it possible for the cameras used in this project to be not as practical as their more advanced alternatives. Additionally, the outcomes of the study showed that constant errors might be present requiring calibration of the cameras while errors related to the equation parameters can also be the cause of faulty estimations. A further issue with this approach is that it only focuses on the depth estimation rather than the location of the object meaning that the deviations can occur when the object is not located near the center of the image plane. Therefore, the method should be improved taking other dimensions into account to be able to represent the real world as discussed in the following heading.

3.2.2 Using Geometrical Approach

A technique that takes the object coordinates into consideration suggests that the image location in the real space should be utilized to come up with a geometrical representation of the distance vector. According to this methodology, the pixel grid of an image and the real coordinates of the object with respect to the camera have a trigonometric relationship [32]. To identify this relationship, it is assumed that the optical center of the camera is the origin of the camera-based coordinate system as shown in Figure 5. The axis represented as *x* is parallel to the ground, and its values increase as the points move to the right side of the image plane as depicted in Figure 5 by an arrow. Similarly, the *y*-axis is also parallel to the ground but indicating the depth of the object like the parallel distance in Figure 4. It should be noted that the values in this axis can never be negative as the points with negative *y* values would indicate the objects behind the camera. Lastly, the *z*-axis is used to indicate the height of the camera from the ground. Since the camera is taken as the origin of the system having the coordinates $(x_{camera}, y_{camera}, z_{camera}) = (0, 0, 0)$, the objects on the ground should have the *z* value equal to the height of the camera.

It can be concluded that when the values x_o, y_o, and z_o in Figure 5 remarking the object coordinates are known, the object distance to the camera becomes the Euclidean distance between the origin and the coordinate of the object which can be represented as:

$$D = |(x_o, y_o, z_o), (0, 0, 0)| = \sqrt{{x_o}^2 + {y_o}^2 + {z_o}^2} \quad (5)$$

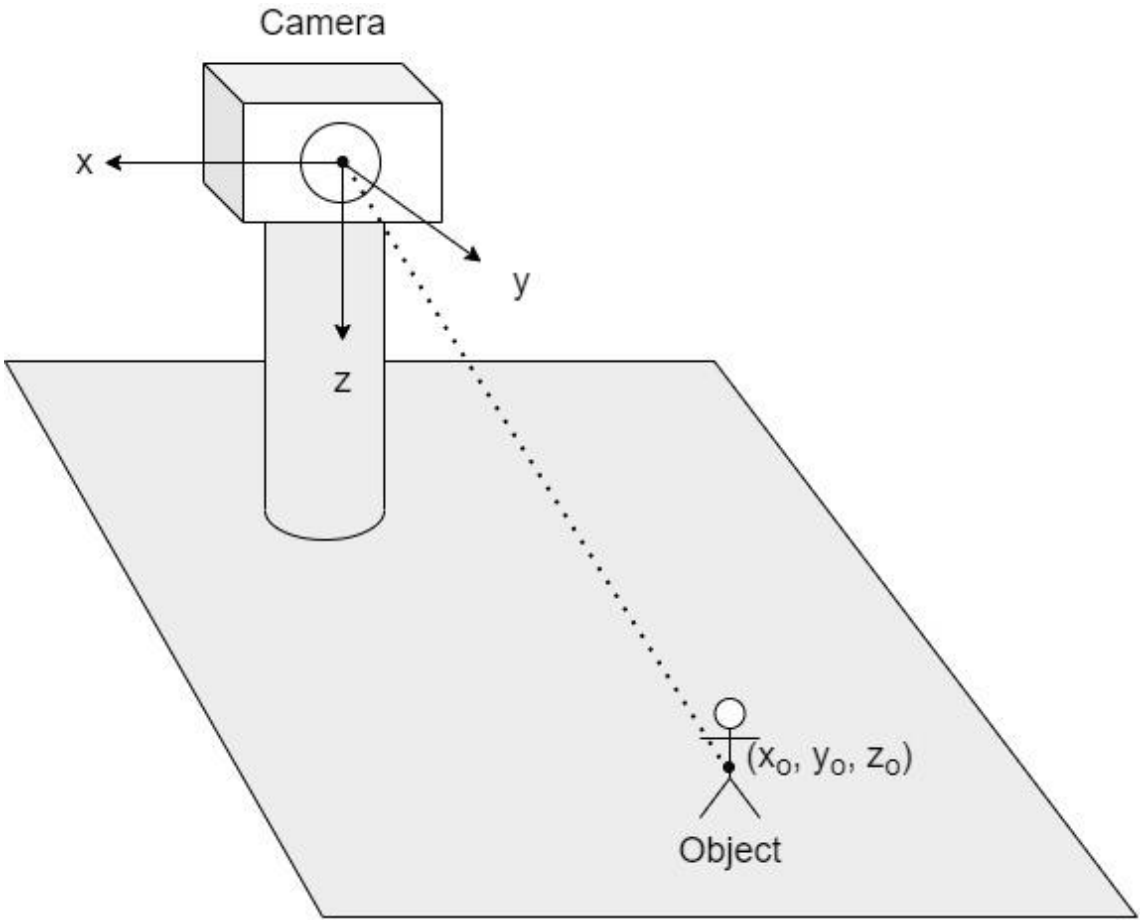


Figure 5. Camera-based coordinate system.

As Equation (5) defines the distance between two points, both the object and the camera are treated as point objects. With this assumption, the coordinates of the object (x_o, y_o, z_o) in fact stand for the middle point of the rectangular bounding box enclosing the object. Since the z_o value in Equation (5) is known as the camera height provided that the object is on the ground, this value is easy to measure during system installation and can be inputted into the algorithm. However, the x_o and y_o values are different for every point in the image which makes it necessary to devise a generic geometrical description for the image.

The values on the *x* and *y*-axes are investigated utilizing the different views of the system area. Figure 6 illustrates the top view of the environment used to generate an equation for x_o while Figure 7 depicts the side view of the domain in order to formulate the y_o value.

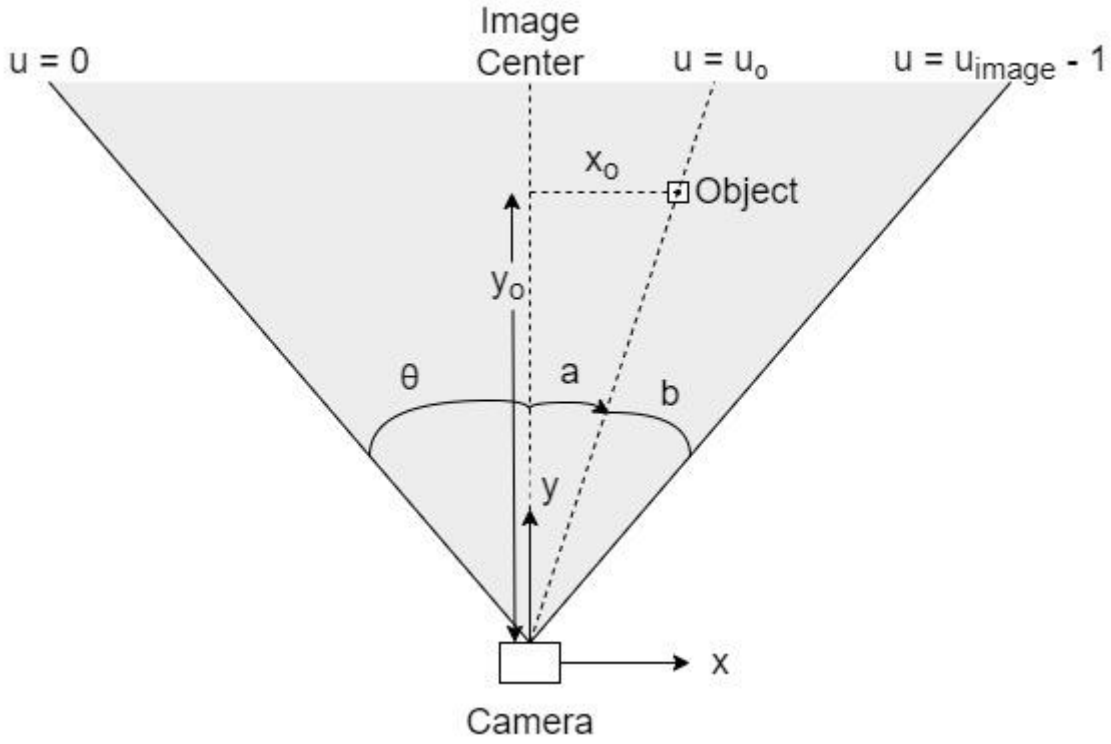


Figure 6. Camera's field of view from the top.

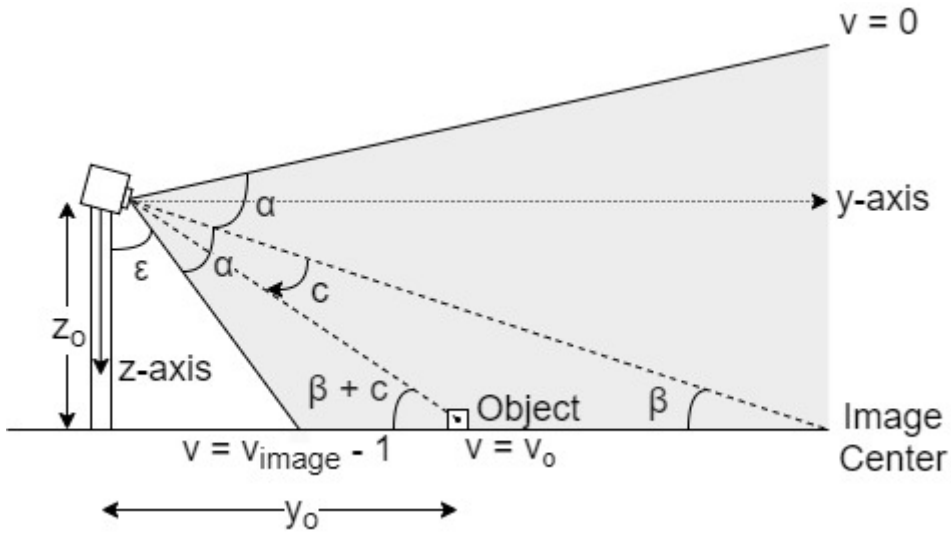


Figure 7. Camera's field of view from the side.

The system illustrated in Figure 6 and Figure 7 represents the top and side views of a tilted camera whose image plane has the width u_{image} and the height v_{image} in pixels. The angle θ in Figure 6 shows half of the horizontal field of view, meaning that the whole visual range in that plane is 2θ. Similarly, α in Figure 7 is the semi-range of the camera in the vertical plane signifying that the vertical field of view is 2α. Furthermore, ε is the blind angle which expresses that the camera fails to see this area as the zone is too close to it. Utilizing these figures, x_o and y_o coordinates of an object are to be found whose mid-

point on the image plane has the coordinates (u_o, v_o). In order to obtain the y_o value, a trigonometric equation should be constructed:

$$y_o = z_o \times cot(\beta + c) \quad (6)$$

The angle β is the tilt angle of the camera measured with respect to the ground plane. Moreover, c demonstrates the angle between the image center line and the object line. Hence, its value depends on the distance of the object on the y-axis. It is worth noting that the angle is positive when the object is under the horizontal image center on the frame, but negative when it is above. Besides, as the c value gets closer to $-\beta$, y_o approaches infinity meaning that Equation (6) cannot produce meaningful results starting from this value. Therefore, the region of interest in this approach is the part of the frame lower than the $z = 0$ plane. When the object point is at the very bottom of the frame, indicating that $v_o = v_{image} - 1$, the angle c becomes equal to α. Additionally, $c = 0$ is reached if v_o is at the central horizontal line with the height $\frac{v_{image} - 1}{2}$. Consequently, c can be represented as:

$$c = \frac{2 \times \alpha \times v_o}{v_{image} - 1} - \alpha \quad (7)$$

Combining Equation (6) and Equation (7), the y-axis distance of the object becomes:

$$y_o = z_o \times \cot\left(\beta + \frac{2 \times \alpha \times v_o}{v_{image} - 1} - \alpha\right) \quad (8)$$

Hence, y_o coordinate of any object under the $z = 0$ plane on the image can be calculated in millimeters provided that the height of the camera z_o (mm), the height of the point on the image v_o (pixel), the image height v_{image} (pixel), the angle denoting the half of the vertical field of view α (degree), and the tilt angle β (degree) is known. Although the study proposing this approach [32] formulates the y-axis distance using the blind angle ε and the tangent function, the equation was altered to address the representation with the tilt angle β as it is easier to measure during the system implementation. Along with y_o, the x-axis distance x_o can also be formulated as a trigonometric equation:

$$x_o = y_o \times \tan a \quad (9)$$

As indicated in Figure 6, a in Equation (9) is the angle between the image center line and the object line similar to the angle c in Figure 7. This angle is also dependent on the location of the object in the frame, specifically its x_o value, and is positive if the object is

on the right side of the image center but negative when it is on the left. Furthermore, the angle at the leftmost side of the image is -θ while the angle at the rightmost part is θ. In addition, the angle at the image center with the pixel $\frac{u_{image}-1}{2}$ becomes *0*. Thereby, a is calculated as:

$$a = \frac{\theta \times (2 \times u_o - u_{image} + 1)}{u_{image} - 1} \tag{10}$$

Inserting the a value found with Equation (10) to Equation (9), the x_o value representation becomes:

$$x_o = y_o \times \tan\left(\frac{\theta \times (2 \times u_o - u_{image} + 1)}{u_{image} - 1}\right) \tag{11}$$

The horizontal placement of the object with respect to the camera, x_o is estimated in millimeters according to Equation (11) utilizing the y_o (mm) value found as a result of Equation (8), the width of the point on the image u_o (pixel), the image width u_{image} (pixel), and one-half of the horizontal field of view θ (degree).

Thus, supposing that the horizontal and vertical field of view, the tilt angle, and the camera resolution is known, the object distance can be calculated from its image coordinates. However, if the angle values are not known, they can be calculated by generating an equation system provided that certain solutions to Equation (8) and Equation (11) are available. Letting that two solutions to the aforementioned equations are (x_1, y_1, z_o) and (x_2, y_2, z_o) where their corresponding image coordinates are (u_1, v_1) and (u_2, v_2) respectively, the solution set for Equation (8) becomes:

$$y_1 = z_o \times \cot\left(\beta + \frac{2 \times \alpha \times v_1}{v_{image} - 1} - \alpha\right) \tag{12}$$

$$y_2 = z_o \times \cot\left(\beta + \frac{2 \times \alpha \times v_2}{v_{image} - 1} - \alpha\right) \tag{13}$$

By taking the z_o values to the left side of the equation as a multiplicative inverse and then applying the inverse cotangent function in Equation (12) and Equation (13), the set can be solved for the angular parts. Moreover, it should be noted that the inverse cotangent of a value, $\cot^{-1}(p)$ is always equal to the inverse tangent of the reciprocal of the same value, $\tan^{-1}(\frac{1}{p})$ which transforms the solutions to:

$$\beta + \frac{2 \times \alpha \times v_1}{v_{image} - 1} - \alpha = \cot^{-1}\frac{y_1}{z_o} = \tan^{-1}\frac{z_o}{y_1} \quad (14)$$

$$\beta + \frac{2 \times \alpha \times v_2}{v_{image} - 1} - \alpha = \cot^{-1}\frac{y_2}{z_o} = \tan^{-1}\frac{z_o}{y_2} \quad (15)$$

Later, Equation (15) is multiplied by *-1* and the derived equation is summed with Equation (14) in order to get:

$$\frac{2 \times \alpha}{v_{image} - 1} \times (v_1 - v_2) = \tan^{-1}\frac{z_o}{y_1} - \tan^{-1}\frac{z_o}{y_2} \quad (16)$$

Rearranging Equation (16) for representing the angle α:

$$\alpha = \frac{v_{image} - 1}{2 \times (v_1 - v_2)} \times \tan^{-1}\left(\frac{z_o \times (y_2 - y_1)}{{z_o}^2 + y_1 \times y_2}\right) \quad (17)$$

To find the angle β, the resulting α description of Equation (17) should be inserted in either Equation (14) or Equation (15). Solving β for equation (14) yields to:

$$\beta = \frac{1}{2(v_1 - v_2)}\left[\left(v_{image} - 2v_2 - 1\right)\tan^{-1}\frac{z_o}{y_1} - \left(v_{image} - 2v_1 - 1\right)\tan^{-1}\frac{z_o}{y_2}\right] \quad (18)$$

The solution (x_1, y_1, z_o) with image coordinates (u_1, v_1) can also be used to formulate θ in a similar manner. The solution for Equation (11) with the mentioned coordinate becomes:

$$x_1 = y_1 \times \tan\left(\frac{\theta \times (2 \times u_1 - u_{image} + 1)}{u_{image} - 1}\right) \quad (19)$$

The angle θ can be solved in Equation (19) as:

$$\theta = \frac{u_{image} - 1}{2 \times u_1 - u_{image} + 1} \times \tan^{-1}\left(\frac{x_1}{y_1}\right) \quad (20)$$

As a result, it can be deduced that given two solution points from the frame, the angular parameters α, β, and θ can be estimated when they cannot be retrieved as a camera parameter or measured. However, the solution set should be selected properly so that accurate results can be obtained. For instance, if the first solution is taken from the center of the frame where $x_1 = 0$, the angle θ will be calculated as *0* as well. Specifically, the experimental results showed that Equation (17), Equation (18), and Equation (20) provide outcomes with smaller errors when the solution set is close to the center where x_1 is not equal to *0*. These experiments with the calculated angles are compared with the values measured and procured from the lens datasheet and presented in Section 5.2.

Calculating the object distances to the camera utilizing the geometrical relationship between the object point and the optical center of the camera proves to be a useful approach as the computational complexity is low and the algorithm is feasible to be computed in real-time. Moreover, as all of the axes in the camera coordinate system is considered during calculation, the methodology can produce reliable results. Although this technique also relies on some camera parameters such as resolution and field of view where the field of view angles can be calculated with a small solution set, its point-specific approach can make it a more preferable solution.

3.2.3 Using Machine Learning

The last approach studied in this thesis work is to use a machine learning algorithm that is trained for the system to be installed in order to detect object distances. When a large and appropriate enough dataset is provided for training, machine learning algorithms with proper models are known to produce estimations that are adjacent to the real results.

Both supervised and unsupervised learning methods have been previously studied to overcome autonomous transportation challenges. While the unsupervised learning algorithms are mainly utilized for depth segmentation [33], [34], distance estimation with numerical proposals primarily demands a supervised learning algorithm to be able to discover the properties of the environment [35], [36]. Similar to the geometrical approach for formulating the x_o and y_o points, the machine learning technique aims to train the system so that it can produce educated estimations on the same coordinates of an object.

The first challenge to be addressed during training is to be able to procure a valid dataset where sufficient information is provided for the system to generate a model expressing the relationship between the object location on the image as bounding box coordinates and its distance to the camera. Most of the existing datasets available for object detection and depth estimation fail to deliver explicit distance annotations that also indicate the types of the objects. Moreover, the studies that focus on object-specific distance annotations benefit from the KITTI [41] and nuScenes [42] datasets which provide annotations regarding only the depth of the object, y_o [35]. While these include comprehensive examples in terms of object types and distances, they are constructed for autonomous transportation studies offering images that are parallel to the ground. Yet, Camera-Based Vehicle Location Detection requires a dataset that consists of inclined images where the camera has a certain pitch angle. Furthermore, the system studied in

this thesis calls for detection of the object coordinates as well to be able to map the object locations meaning that training should be done for both x_o and y_o points rather than only for y_o. Hence, existing datasets cannot address the problem definition of Camera-Based Vehicle Location Detection thoroughly which is why a new dataset generated for a different project at Tallinn University of Technology has been utilized.

The studies on computer vision, whether they focus on distance estimation or not, mostly implement a deep learning model due to its numerous advantages compared to conventional learning models which make deep learning an easily available algorithm. Deep learning is also able to handle a large amount of data for training which is a crucial requirement for image processing applications while more accurate results can be obtained with deep learning [37]. Consequently, due to its ease of access, extensiveness, and efficacy on large datasets, deep learning is a more preferable method of machine learning than the traditional models.

Selecting the proper attributes of the model to fulfill the project-specific needs, which is named hyperparameter optimization, is a major concern in deep learning implementations. This task calls for an efficient algorithm to devise an appropriate model to be utilized in training. Furthermore, training a deep learning model takes more time and requires more advanced hardware with high performance than a conventional approach. Despite these challenges, studies conclude that deep learning can find solutions to complex and non-linear functions in a simplified way [38] while being the most effective, supervised, and stimulating machine learning approach [37].

4 Implementation

Camera-Based Vehicle Location Detection is a project whose end product should run as a real-time embedded system able to process visual data. Hence, the camera input ought to be loaded to a high-performance graphics processing unit (GPU). A suitable alternative to meet the hardware requirements of the project is considered to be the NVIDIA Jetson AGX Xavier Developer Kit with a 13-megapixel (MP) 4-camera board whose image is provided in Figure 8 since it is a rather effective tool for graphical processing and deep learning with low power consumption and ease of use. As this thesis work focuses on comparing the algorithms of the system, hardware discussion is generally omitted from this work.

Figure 8. NVIDIA Jetson AGX Xavier Developer Kit connected to the camera board, retrieved from [43].

The system software is developed in Python programming language with version 3.8. Even though there are other languages that are more embedded-oriented such as C++, Python is chosen as it embodies a massive number of standard and open-source libraries for numerous applications. Considering the fact that this thesis study utilizes various features including object detection, trigonometric calculations, solving trigonometric equation systems, and machine learning, Python is found to be able to provide a more consistent software development process due to its competence to address tasks as complex as end-to-end learning with its selection of libraries. Furthermore, having a simpler syntax allowing more straightforward development has also made Python a more

preferable choice. The link to access the source codes of the project is presented in Appendix 5.

4.1 Object Detection Implementation

Composing the object detection algorithms described in this thesis work presents a generic approach to be employed as the implementation of these detectors requires some specific functions defined in the OpenCV library which is an open-source platform developed for conducting computer vision and image processing operations. In order to get the video from the camera or a file, OpenCV provides the class *VideoCapture* where the video source can be inputted as a parameter to its constructor, either as a file path or as the camera feed. Each frame in the video can be decoded utilizing the *read* function in the class which returns the particular frame. Therefore, the function call is made in a *while* loop which executes as long as the video capturing continues. The pseudocode for decoding frames from a connected camera is presented in Figure 9 where *cv2* denotes the OpenCV library. Furthermore, when only one frame is of interest, OpenCV computes the decoding operation with the *imread* function to be able to read an image where the file path is inputted as a parameter.

```
Initialize the VideoCapture object
While the video feed runs
       Read the subsequent frame in the video feed
       Perform object detection activities on the frame
```

Figure 9. The pseudocode for video decoding with OpenCV.

Even though the algorithmic approach for the object detection on a frame with both MobileNet SSD and YOLO detectors are principally the same, their implementations have been carried out divergently. Moreover, the configurations and models for MobileNet SSD and YOLO are constructed in separate formats. Hence, these techniques are discussed further in detail in different headings.

4.1.1 MobileNet SSD

The implementation of MobileNet SSD covers a list of 20 object classes including but not limited to bicycle, bus, car, person, and motorbike. The network model to be utilized for the detector is loaded in the format of the Caffe framework with the usage of the *cv2.dnn.readNetFromCaffe* function call. The function takes the network configuration file path in *.prototxt* type and the trained weights file path in *.caffemodel* format as

parameters. As a result, the artificial neural network object from the *Net* class is returned. Later, inside the *while* loop where one frame is read, a blob is created from the current frame which is a transformation of the image to a format in which certain shapes on the image are highlighted. With the scaling and resizing parameters, *cv2.dnn.blobFromImage* function allows the creation of such a blob. This blob needs to be inputted to the neural network which is achieved with the *setInput* function from the *Net* class where the blob is a parameter to the function. The output of the network is obtained in a list format via the *forward* function from the same class. After the execution of this function, the information of the detected objects is ready to be analyzed.

Each element in the acquired list of the detection information is examined in a *for* loop to filter out some of the detections with low confidence values that are smaller than a decided threshold of 20% and to visually label the objects on the frame. The left bottom and the right top frame coordinates of the detected objects can also be extracted from the list so that the bounding boxes can be generated on an output frame using the *cv2.rectangle* function. Moreover, the classes indicating the type of the detected objects are also present in the list for each element so that they can be indicated as text on the output image via the function *cv2.putText*. This labeling operation is especially useful for obtaining the performance analysis of the detector as presented in Section 5.1.

4.1.2 YOLO v3 and YOLO v4

The implementation algorithm of the YOLO v3 and YOLO v4 detectors are nearly the same as the MobileNet SSD method since all of these detectors are programmed utilizing the OpenCV framework. However, for the execution of the YOLO techniques, a new Python class with the name *YoloDetector* is generated so that the access of the neural network, the object detection sequence, and filtering as well as labeling the objects can be performed in a more straightforward manner.

The constructor of the mentioned class takes the file paths of the network configuration, trained weights, and the object class dataset to generate the neural network and read the labeled classes. The network configuration file is of *.cfg* type, and the weights are in *.weights* format meaning that the YOLO detector framework Darknet is utilized. The labeling dataset is acquired from the COCO dataset that includes 80 object classes involving person, bicycle, car, motorbike, bus, and truck. To load the network model from a Darknet framework as a *Net* object, OpenCV provides a function named

cv2.dnn.readNetFromDarknet similar to the Caffe framework loading described with the MobileNet SSD. It should also be noted that a more generic function *cv2.dnn.readNet* also exists that can take different framework models as input since the file types are separate. Following this loading operation, the names of the output layers are required to be procured in two steps as the names are needed to input to the detection function later. These steps are initially to get the names of all the layers with the *getLayetNames* function and then to obtain the names of the output layers from the returned list of the preceding function via the *getUnconnectedOutLayers* function both of which are accessed through the generated *Net* object.

The *YoloDetector* class has two more functions named *detect* and *drawAndLabel*. The first one takes the frame on which object detection should be performed as an input parameter and returns four lists that deliver information about the names, confidences, bounding box coordinates, and indexes of the detected objects. The detection sequence is almost equivalent to the detection order of MobileNet SSD; the image blob is built with the *cv2.dnn.blobFromImage* function and its output is given as a parameter to the *setInput* function to start the detection. The function *forward* is also utilized to obtain the results of the detector, but the output layer names from the *getUnconnectedOutLayers* function become a parameter to the function call this time. The confidence threshold is set as 50% with YOLO algorithms while a non-maximum suppression on the frame is necessary as the YOLO model does not apply this operation during detection, which is executed using the *cv2.dnn.NMSBoxes* function whose output is the index list that forms one of the returning lists of *detect* function. Furthermore, labeling the objects on an output frame is accomplished with the identical approach as described with MobileNet SSD implementation inside the *YoloDetector* class function *drawAndLabel*.

The methodology described in this section is valid for both YOLO v3 and YOLO v4 variants along with their tiny adaptations. Although the implementation algorithm is uniform, loaded configuration and weights are unique for each alternative as the neural network models are different.

4.2 Distance Estimation Implementation

The distance estimation methods described in Section 3.2 can be classified into two categories in terms of their variance in implementation. Utilizing camera parameters and

geometrical approach requires similar strategies although their calculation formulas are dissimilar. Moreover, implementing the machine learning approach expects more advanced frameworks and functions to be utilized for model generation and training while some other libraries are needed for generating the proper datasets.

4.2.1 Using Camera Parameters and Geometrical Approach

The requirements for the implementation of the distance estimation methods based on a formula derived either by utilizing the camera parameters or via the geometrical approach are mainly alike. Programming Equation (4) where the parallel distance D is obtained from Equation (3) requires basic arithmetic operations along with a trigonometric *cosine* operation which can be executed via the Python built-in module *math* while the same interpretation is valid for executing Equation (8) and Equation (11) with *tangent* and *cotangent* functions. It is worth mentioning that the trigonometric functions of the *math* library take the angle values in radian meaning that the degree values need to be converting utilizing the *radians* function in the same library. The bounding box coordinates of the detected objects on the frame under investigation can be retrieved from the object detection implementation presented in Section 4.1.2 while the resolution of the mentioned frame is obtained utilizing the *shape* method provided by OpenCV which in fact benefits from the Numpy library.

Generating the strategy for utilizing camera parameters calls for three values to be inputted to the program externally which are the camera focal length, sensor height, and pitch angle. The geometrical approach, on the other hand, necessitates four inputs, namely the camera height and the three angular parameters, α, β, and θ. The remaining attributes can be fetched from the object detection result and the *shape* method for both techniques. Estimating the distances is achieved in a *for* loop where all detections that match the output of the non-maximum suppression are analyzed.

Inside the loop for the calculation with the camera parameters, the bounding box coordinates for the current detection are procured so that the height of the box can be determined. Moreover, this algorithm expects the real height of the object to calculate the distance, which cannot be precisely resolved for each object present. Hence, a mean height of 1535 mm was fixed for each object classified as a car while all the other objects are treated to have a height of 2835 mm which is a mean height of a truck. Thus, it can be concluded that this implementation gives a generic result for cars and trucks which

constitute the major part of the area of interest even though the program is not reliable for other types of objects as well as some cars and trucks whose height deviates from the mean value largely. With all of the values assigned, Equation (3) divided by the *cosine* of the pitch angle is ciphered and labeled as text on the corresponding frame.

The loop iteration for the geometrical approach commences similarly by fetching the bounding box coordinates of the object to be investigated and deciding the height of the box. Furthermore, the mean values of the aforementioned coordinates are calculated as well. Equation (8) has to be computed before since Equation (11) utilizes also the result of the previous estimation. After both x and y coordinates are determined, the Euclidean distance is evaluated by merging the camera height into the measurement. As before, the resulting distances are written above the bounding box drawing on the respective frame.

Predicting the angular parameters from the provided solution set can be performed utilizing the Python library Sympy. Equation (8) and Equation (11) can be generated with the help of this library so that the solution set can be inputted to obtain the intended parameters. The function *symbols* is utilized to create symbol instances for each attribute of the aforementioned equations. Later, these symbols are employed to generate the equations where the trigonometric functions *tan* and *cot* are retrieved from the Sympy library. To substitute the variables with their known values in the solution set, the function *subs* is called iteratively where the first argument is the symbol to be replaced with its value for the corresponding solution and the second parameter is the actual value of the variable in the current solution. The left-hand side and the right-hand side of the equation are defined with the class constructor *Eq* which stands for Equality class where the first parameter is the created equation and the latter one is the resulting distance. Lastly, to solve the equation system, the function *solve* is used with the attributes equation and the symbol to be solved, which returns either a dictionary or a list containing the solution. It must be mentioned that the solution set for estimation of the angular values should be selected carefully in order to obtain results that are parallel to the real values. The experimental outcomes have shown that the x-axis value is not important for α and β meaning that it is sufficient to obtain the y-axis measurement which should be chosen from the points that are relatively in the middle of the image plane. Following a similar logic, the coordinate that is of significance for θ is represented by the x-axis, which implies that one horizontal solution is enough for calculating this angle provided that this solution is also rather central on the image plane except for 0.

4.2.2 Using Machine Learning

Since the Camera-Based Vehicle Location Detection project focuses on not only the distance of a vehicle but rather its coordinates to create an accurate map of the environment, the machine learning model constructed for this thesis work does not aim to estimate the overall distances, instead to predict the *x* and *y* coordinates separately. To address this target, two different learning models have been generated that are relatively small and plain as shown in Figure 10. Another approach fulfilling the objective could be to build one model with the capability to forecast the values in both axes to output one set of predictions covering all needs. However, this model practically needs to be more complex than the two models anticipating the coordinates independently as it would require more layers and neural connections to devise acceptable estimations for both of the expected outputs. Therefore, as a substitute for this large and compound design, each axis value has been predicted by compact and unconnected networks. Both of the models are inputted by the same image dataset which contains certain information related to the detected objects which are utilized by the neural network for training and generating prediction outputs for any upcoming object information.

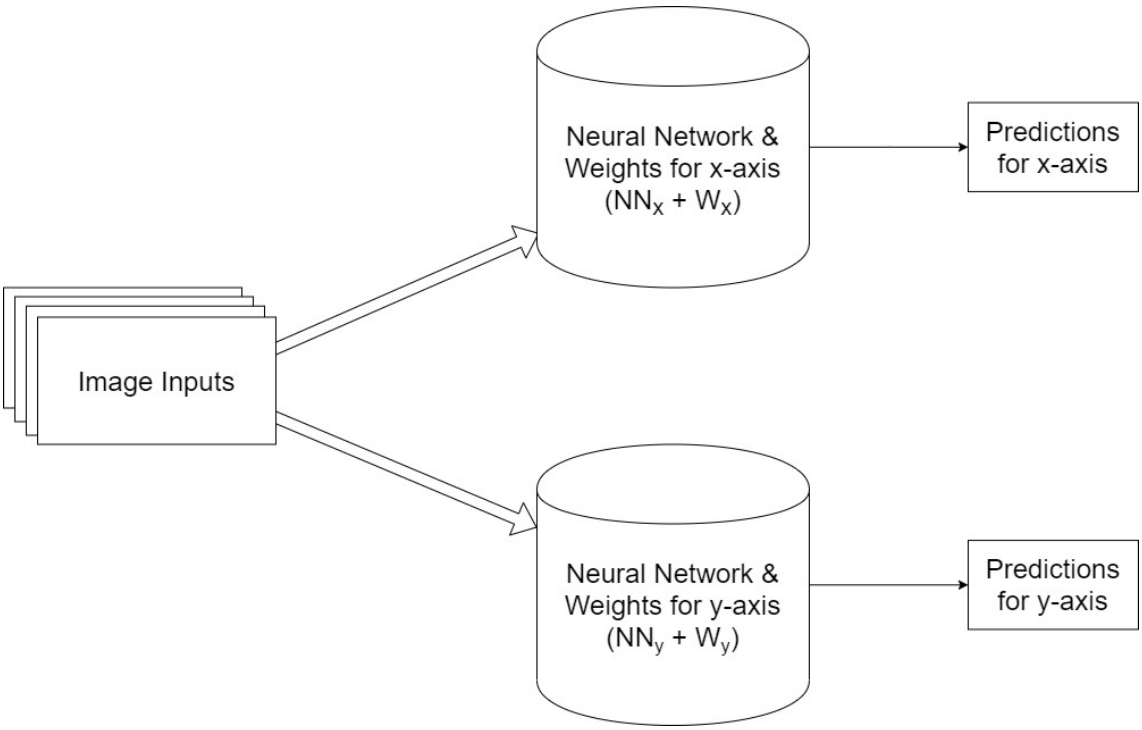


Figure 10. The generic model of the machine learning implementation.

Applying a deep learning algorithm to estimate the object distances requires some primary stages to be implemented which can be classified as follows [39]:

- Generating annotations and dividing the dataset to train and test clusters
- Creating the learning models to be utilized and deciding the hyper-parameters
- Training the models with the training dataset
- Testing the algorithm by producing predictions out of the test annotations

The initial phase for the intended method is to build a proper dataset to train and test the models to be crafted. A different study being conducted at the Tallinn University of Technology regarding embedded image processing has procured a video recording of a road with an integrated camera and a radar system so that the object distances can also be known, which is utilized for testing the machine learning methodology of this thesis work as well. The data was divided into frames where the object type, detection confidence, bounding box coordinates, and whether the box is truncated or not are specified in *.xml* files for each frame. Furthermore, the radar data indicating the distances of the objects in *x* and *y*-axes are presented as *.json* files for every frame. The names of these files are first read into two lists inside a *for* loop scanning every file in the directory according to their filetypes where one list holds the file name of each *.json* file and the other having the names of the *.xml* files with the corresponding index of each element being its frame number. This operation benefits from the Python modules *os* and *re* as *os* allows communication with the operating system for reading file names in the directory, and *re* enables handling regular expressions for configuring the file names to get their frame numbers which serve as list indexes. Later, each file in the aforementioned lists is opened frame by frame in another *for* loop where the information is fetched to a Pandas data frame utilizing the *json* module and the XML document object model API *xml.dom* as well as the *statistics* module. The data frame includes the frame number, the type of the detected object, the detection confidence, bounding box coordinates, the object distance in *x* and *y*-axes, and whether the bounding box is truncated or not. The generated annotations are then divided into two datasets for training and testing where 90% is randomly selected and spared for training with the use of the *rand* function in the *numpy.random* module and the rest is left for testing. The reason for the separation of 90% by 10% between training and testing is to be able to exploit the dataset as much as possible to prepare the models for creating sufficiently accurate results since the annotations used can be considered a rather small dataset with approximately 1000 object information stored.

Constructing deep learning models with the proper attributes asks for hyperparameter optimization which can be achieved with the Python library *hyperopt*. The library embodies a *Trials* class to store the necessary information regarding the hyperparameters. A Python wrapper called *hyperas* enables a convenient optimization performance through the function *optim.minimize* which takes a model creation function, a train and test data generation function, a *hyperopt* algorithm, maximum optimization runs, the *Trials* object, and some other evaluation parameters as inputs. In the model creating function *CreateModel*, a sequential model instance is generated through the Tensorflow framework and Keras API. The neural network is assembled by either three or four layers, which is to be decided by the hyperparameter optimization, each of which is depicted by the *Dense* function where the neuron number of the second layer is also to be determined as a hyperparameter. Moreover, the optimizer in the model configuration function *compile* is also programmed as a hyperparameter as well as the batch size attribute of the training function *fit*. The last operation of the model creation is to evaluate the loss and accuracy of the model to be reported for the optimization. On the other hand, the data generation method *Data* fetches the attribute under investigation from the train and test datasets, standardizes, and transforms these annotation values. Since the predictions are desired for both *x* and *y*-axes, the hyperparameter optimization is performed twice; one for the *x* coordinates, and the other for *y*. The results of the optimization program indicating the best options for both of these models are presented in Figure 11. In the end, the parameters to be chosen by this program are:

- The number of layers in the model
- The number of neurons in the second layer
- The optimizer of the model
- Model batch size

The evaluation results of the best model for *x* and the best model for *y* can be observed in order to construct the optimized network structure and train the models with their corresponding annotations. Similar to hyperparameter optimization, training the model for the dataset is conducted separately for both axes. The aforementioned *CreateModel* method is mainly repeated with the determined hyperparameters for developing the training program. First, the train and test data are fetched and classified as inputs and outputs where inputs are the information that the system learns from and the outputs are the coordinates to be predicted. The training inputs and outputs are then standardized and

transformed before building the models. The sequential models with three and four layers are constructed and configured with the appropriate number of units and the suitable optimizers decided by the previous phase before training the system with the certified batch sizes. Afterward, the trained models are saved at the project directory as *.json* files as well as the weights as *.h5* files. The summary of each model outputted by the function *summary* is provided in Figure 12.

```
Hyperopti_X;

Test accuracy:
0.13825437426567078
100%|██████████| 3/3 [06:05<00:00, 121.85s/trial, best loss: 0.1277075558900833]
5/5 [==============================] - 0s 500us/step - loss: 0.1277 - mae: 0.2272
Evaluation of best performing model for x-coordinates: [0.1277075558900833, 0.2271677553653717]
Best performing model chosen hyper-parameters for x-coordinates:
{'Dense': 4, 'Dense_1': 'three', 'batch_size': 2048, 'optimizer': 'sgd'}


Hyperopti_Y;

Test accuracy:
0.10068554431200027
100%|██████████| 3/3 [06:06<00:00, 122.11s/trial, best loss: 0.05031649023294449]
5/5 [==============================] - 0s 750us/step - loss: 0.0503 - mae: 0.1741
Evaluation of best performing model for y-coordinates: [0.05031649023294449, 0.1740970015525818]
Best performing model chosen hyper-parameters for y-coordinates:
{'Dense': 4, 'Dense_1': 'four', 'batch_size': 1024, 'optimizer': 'rmsprop'}
```

Figure 11. The results of the hyperparameter optimization program evaluated for both models.

Resembling the preceding stage, producing the predictions for the test dataset is conducted independently for both axes under investigation. The process begins with retrieving the test annotations and dividing the input and output attributes followed by standardization and transformation of the data. The model is then loaded and attained from the *.json* file through the Tensorflow method *model_from_json*, and the weights through the *load_weights* function of the *Model* class. Since this model is not compiled yet, the compilation is done with the estimated optimizer. Subsequently, the function call *predict* from the *Model* class with the input annotations performs predictions where the results need to be inversely transformed. The list obtained after all of these operations are the prediction annotations that can be analyzed for accuracy. Utilizing the *pandas* and *numpy* libraries, the product annotations are organized such that the true coordinates and distances are represented along with their corresponding predictions.

```
Model Summary for X Axis:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 66
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 28
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 5
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________

Model Summary for Y Axis:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 28
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 3
=================================================================
Total params: 107
Trainable params: 107
Non-trainable params: 0
_________________________________________________________________
```

Figure 12. The summary of each neural network outputted by the function *summary*.

5 Results

All of the implemented methodologies to be used both for object detection and distance estimation during the development of the Camera-Based Vehicle Location Detection project have been tested with certain samples so that the performance of each approach can be compared with others statistically. Therefore, MobileNet SSD and the YOLO v3 and YOLO v4 families have been compared with the same samples for the object detection efficiency while camera parameter utilization, geometrical approach, and machine learning methods have been juxtaposed to see their distance estimation accuracy scores.

5.1 Object Detection Results

The experiment for comparing the performance results of different object detection methodologies requires these detectors to be tested out on the same frames. Even though the detectors under investigation have more performance criteria for the Camera-Based Vehicle Location Detection project, this study mainly compares them in terms of accuracy since this parameter also affects the distance estimation experiment crucially.

To construct the experiment, five different sample images from a recording of the traffic have been selected and the algorithms MobileNet SSD, YOLO v3, YOLO v3-tiny, YOLO v4, and YOLO v4-tiny have all been inspected to see which algorithms detect the most vehicles with high confidences. The frames taken from flowing traffic allow the procurement of various types of vehicles while generating a challenging environment for the detectors due to the lighting and weather conditions. Even though the images from the decks of the vessels do not pose as much concern in terms of environmental distresses, testing the algorithms in a harsher system can uncover various issues of the models to be realized.

Although the implementation of these detectors is able to notice the objects that are not necessarily vehicles such as traffic lights, these detections do not contribute to the accuracy evaluation of the tests in this study as these objects are out of scope for the

project. Some images with the detection results of each model are provided in Appendix 2, and a sample frame with the outputs of the YOLO v4 detector is presented in Figure 13. Furthermore, the object detection results from all of the models whose implementations are studied in this work are compared in Table 1.

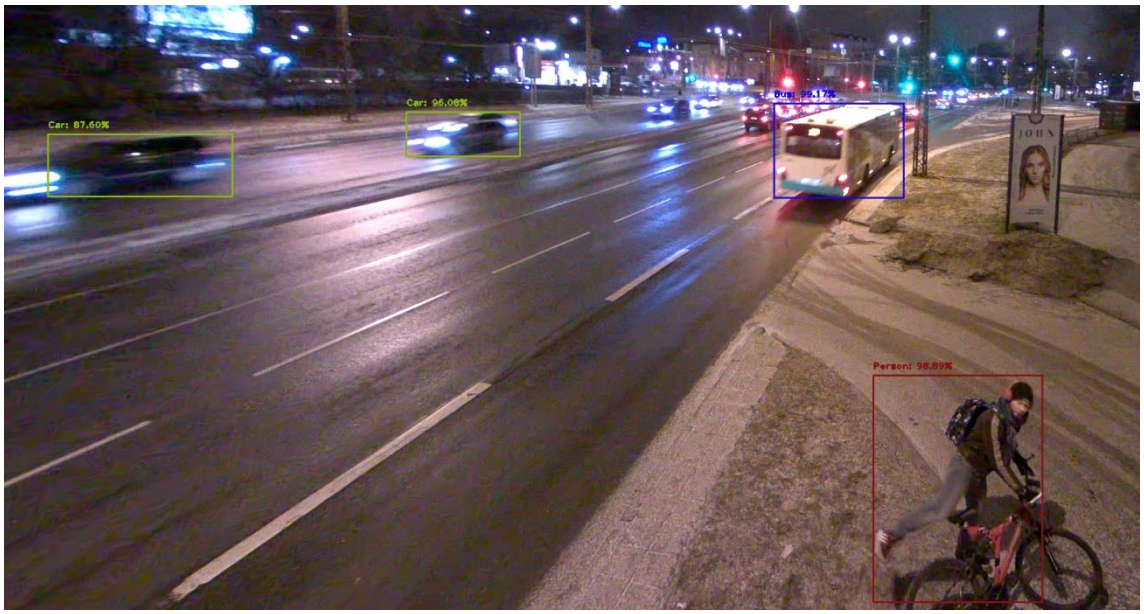


Figure 13. A sample frame with YOLO v4 model used for object detection (Frame #3 in Table 1).

As observed from Figure 13, the test frames in fact contain numerous vehicles if the ones that are quite distant from the camera are also counted. However, the experiment expects the detections of only the objects that can fairly be identified which creates an interest radius of approximately 30 meters. As the distance information from the radar sensor embedded into the test setup suggests, objects that are farther away than this value tend to become indistinguishable by the detectors. This borderline has been considered while determining the number of vehicles on the frames provided in Table 1.

Table 1. The experimental results of the object detection methods.

Frame Number	**Number of Vehicles**	**MobileNet SSD**	**YOLO v3-tiny**	**YOLO v3**	**YOLO v4-tiny**	**YOLO v4**
Frame #1	3 cars	1 car 1 bus (FP)	No objects detected	2 cars	1 car	2 cars
Frame #2	5 cars	1 car 1 bus (FP)	1 car	5 cars	1 car	4 cars
Frame #3	2 cars 1 bus 1 bicycle	1 bicycle 1 bus (FP)	No objects detected	2 cars 1 bus 1 bicycle	1 car 1 bus 1 bicycle	2 cars 1 bus

Frame Number	Number of Vehicles	MobileNet SSD	YOLO v3-tiny	YOLO v3	YOLO v4-tiny	YOLO v4
Frame #4	3 cars	1 car 1 bus (FP)	1 car	1 car	2 cars	2 cars
Frame #5	3 cars 1 bus	1 bus 1 bus (FP)	No objects detected	2 cars 1 bus	1 car	2 cars 1 bus

The frames in the table are indexed from 1 to 5 where each detected object on the frames is indicated for all of the detectors to be tested. Both the figures in Appendix 2 and the results shown in Table 1 suggest that YOLO v3 and YOLO v4 are the best choices addressing the need of being able to detect as many vehicles as possible on the frame while the other algorithms have either failed to detect any objects on some of the frames or detected some false positive (FP) objects. The lighting coming from an advertisement on the top left of the images has always been incorrectly deduced as a bus by MobileNet SSD while the number of vehicle detections on the frames has not reached a decent score. The panel has never been mistaken for a bus on the detectors of the YOLO family and the detection numbers appear to be acceptable. YOLO v3 and YOLO v4 models performed the top-notch results in this elementary experiment while their tiny variants did not function as successfully which is a predictable outcome due to their sacrifice in accuracy while growing faster and more compact. Moreover, the tests on the frames from a ferry, taken specifically for the Camera-Based Vehicle Location Detection project that cannot be shown in this study due to confidentiality reasons also concluded that YOLO v4 performs better than its v3 variant on the complex cases where vehicles are positioned close to one another.

5.2 Distance Estimation Results

Resembling the object detection experiment, the distance estimation techniques have been analyzed for their accuracy in a similar test setup where the estimation with camera parameters, geometrical approach, and machine learning are assessed with various sample frames. The true distance and the findings of each methodology are compared to obtain a percentage of error for every estimation benefiting the relation between the results and the error:

$$Error\ Percentage\ (EP) = \frac{Experimental\ Result - Accepted\ Result}{Accepted\ Result} \times 100 \qquad (21)$$

The reason that the numerator in Equation (21) is not used as the absolute value of the subtraction is to be able to determine whether the estimation is smaller or larger than the correct value. When the experimental result outputted from a distance estimation algorithm is smaller than the accepted result delivered by the radar system, the error percentage is denoted as a negative value while it becomes positive provided that the predictions are larger than the radar measurements. Later, a mean error for each of these methods is calculated in order to devise a reasonable judgment regarding the precision of the alternatives. Some of the images with the estimated distance values for the camera parameters and the geometrical approach are presented in Appendix 3 although the machine learning estimations are not adjoined since those predictions are obtained as a data frame and not integrated with their corresponding images. Moreover, an example frame having the distance estimation results conducted with the geometrical approach is presented in Figure 14. The outcomes of all of the methods in meters along with their calculated error percentages for each frame are demonstrated in Table 2 as well as the measured distance with radar sensor in meters. The frame number (FN) indicates an index for the test samples for proper tracking.

Table 2. The experimental results of the distance estimation methods.

FN	**Object Type**	**Object Distance**	**Camera Parameters | Error Percent**		**Geometrical Calculation | Error Percent**		**Machine Learning | Error Percent**	
1	Person	6.91	12.99	87.97%	8.30	20.14%	7.45	7.82%
2	Person	6.92	14.43	108.64%	8.53	23.39%	6.98	0.95%
3	Car	10.78	11.64	7.97%	12.63	17.21%	10.69	-0.82%
4	Car	14.09	10.52	-25.38%	12.78	-9.35%	12.57	-10.80%
5	Car	22.06	19.71	-10.68%	16.95	-23.17%	22.42	1.64%
6	Car	11.05	6.14	-44.38%	10.05	-9.04%	10.14	-8.22%
7	Car	14.09	11.16	-20.80%	12.96	-8.04%	12.75	-9.53%
8	Bicycle	6.23	22.57	262.28%	6.58	5.63%	6.15	-1.33%
9	Car	23.21	21.66	-6.70%	21.31	-8.19%	23.25	0.15%
10	Car	27.57	47.55	72.49%	28.84	4.62%	25.68	-6.85%
11	Car	21.31	24.86	16.65%	19.96	-6.33%	21.77	2.17%
12	Car	15.78	14.88	-5.73%	14.97	-5.15%	13.97	-11.46%
13	Car	17.67	13.34	-24.52%	15.87	-10.20%	16.69	-5.57%

FN	Object Type	Object Distance	Camera Parameters | Error Percent		Geometrical Calculation | Error Percent		Machine Learning | Error Percent	
14	Car	34.03	27.34	-19.66%	20.12	-40.88%	31.01	-8.86%
15	Car	22.64	35.28	55.82%	23.43	3.50%	22.42	-0.96%
16	Car	12.42	8.82	-28.99%	11.68	-6.00%	11.52	-7.29%
17	Car	24.66	29.96	21.48%	21.59	-12.48%	24.34	-1.30%
18	Car	11.13	6.06	-45.59%	9.64	-13.43%	9.83	-11.71%
19	Bus	18.12	8.60	-52.58%	16.16	-10.83%	16.39	-9.57%
20	Car	21.50	29.56	37.48%	20.80	-3.23%	22.63	5.25%
21	Car	23.51	24.58	4.52%	21.89	-6.89%	24.01	2.12%
Mean Error Percentage				*18.59%*		*-4.70%*		*-3.53%*

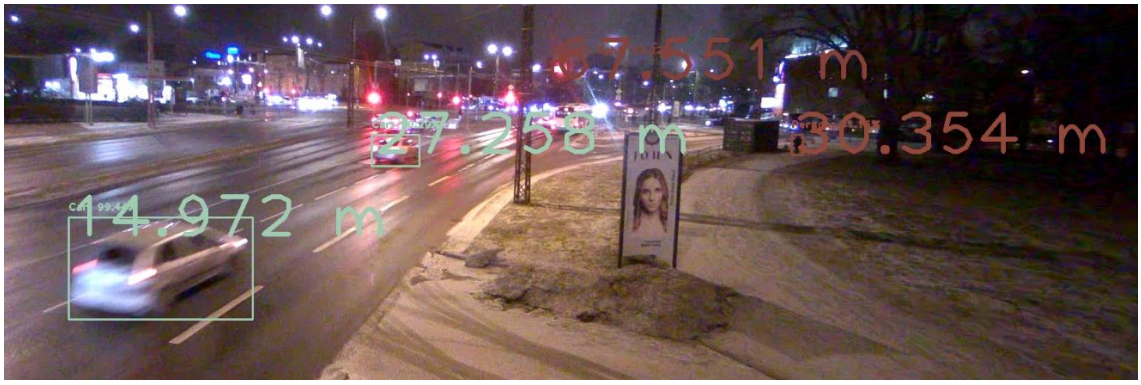


Figure 14. An example frame with the geometrical approach used for distance estimation (Frame #12 in Table 2).

As deduced from Table 2, calculation with camera parameters may lead to large errors due to various reasons. Firstly, this method is applied on an embedded camera although it produces better estimations on advanced cameras. Moreover, the average height assumption made during the implementation of this approach can yield erroneous results when the vehicle whose distance is to be calculated largely deviates from this mean height. Another aspect to be taken under consideration is that the center of the image plane is estimated better with this methodology while the rest of the frame has more errors due to angular differences not considered with this technique. Utilization of the camera parameters has produced its largest errors on the estimation of person and bicycle classes in Table 2 both of which are the only classes with the true distance smaller than 10 meters while the rest of the errors of this technique appear to have occurred evenly on the distance intervals of 5 meters. However, Table 2 results suggest that the geometrical approach and

machine learning produce fairly well approximations between 10 and 15 meters as well as 20 and 25 meters. Although this small experiment set may not be enough to make deductions regarding the operation range of these algorithms, further experiments might be able to generalize these observations with sizable datasets.

Unlike the first practice, the geometrical approach and machine learning result in significantly more precise deductions making them more superior alternatives. Geometrical approximation to model the environment can consider the camera pitch angle and object orientation more fittingly than the formula derived from the camera parameters while machine learning takes time to investigate the environment in detail to devise findings that are nearly exact. Yet, the model construction and training processes applied at the machine learning method make it computationally expensive while the geometrical approach formula can be friendlier to be implemented on an embedded system. Hence, even though its error percentage is mildly higher than the machine learning technique, utilizing a formula derived from geometrical modeling seems to be a more suitable expedient to the Camera-Based Vehicle Location Detection project.

The angle calculation in the geometrical approach provided with Equation (17), Equation (18), and Equation (20) is also experimented to compare the outcomes of the equations with the measured tilt angle and the fields of view obtained from the datasheet of the camera lens. The camera utilized for this test is e-CAM130A_CUXVR whose lens has a horizontal field of view with 87.26° and a vertical field of view with 64.96° [40]. As the angle α is equal to the half of the vertical field of view whereas θ represents one half of the horizontal field of view, the true values for these angles can be deduced as:

$$\alpha = \frac{FOV_V}{2} = \frac{64.96^\circ}{2} = 32.48^\circ \tag{22}$$

$$\theta = \frac{FOV_H}{2} = \frac{87.26^\circ}{2} = 43.63^\circ \tag{23}$$

In Equation (22), FOV_V signifies the vertical field of view while FOV_H in Equation (23) indicates the horizontal field of view. Moreover, the tilt angle β is measured as 9°. To understand the effect of selecting proper points for angle calculation through the equations, three solution sets have been considered. Three frames where the coordinates of the detected objects are in extreme points have been put to the experiment as the first solution set. The second and third solution sets consist of two frames each as two objects create the smallest possible sets. The object coordinates of these points are provided as:

$$Set\ \#1\ \begin{cases}(x_{1,1}, y_{1,1}, z_{1,1}) = (-0.77, 4.03, 4.1), (u_{1,1}, v_{1,1}) = (1297.0, 1412.0)\\ (x_{1,2}, y_{1,2}, z_{1,2}) = (0.0, 34.88, 4.1), (u_{1,2}, v_{1,2}) = (995.5, 785.5)\\ (x_{1,3}, y_{1,3}, z_{1,3}) = (-10.97, 11.99, 4.1), (u_{1,3}, v_{1,3}) = (730.5, 819.5)\end{cases} \quad (24)$$

$$Set\ \#2\ \begin{cases}(x_{2,1}, y_{2,1}, z_{2,1}) = (2.86, 6.75, 4.1), (u_{2,1}, v_{2,1}) = (1945.0, 1251.5)\\ (x_{2,2}, y_{2,2}, z_{2,2}) = (-6.05, 11.39, 4.1), (u_{2,2}, v_{2,2}) = (812.5, 991.5)\end{cases} \quad (25)$$

$$Set\ \#3\ \begin{cases}(x_{3,1}, y_{3,1}, z_{3,1}) = (3.06, 32.48, 4.1), (u_{3,1}, v_{3,1}) = (1439.5, 737.5)\\ (x_{3,2}, y_{3,2}, z_{3,2}) = (-5.96, 6.86, 4.1), (u_{3,2}, v_{3,2}) = (338.5, 1291.5)\end{cases} \quad (26)$$

The angle calculation results of the solution sets provided in Equation (24), Equation (25), and Equation (26) are shown in Table 3. In the first set, the points $(x_{1,1}, y_{1,1}, z_{1,1})$ and $(x_{1,2}, y_{1,2}, z_{1,2})$ along with their representative coordinates on the image plane $(u_{1,1}, v_{1,1})$ and $(u_{1,2}, v_{1,2})$ respectively, are utilized to calculate the angles α and β while the last point $(x_{1,3}, y_{1,3}, z_{1,3})$ and its placement on the image plane $(u_{1,3}, v_{1,3})$ are exploited to obtain the θ angle. However, the first points at *Set #2* and *Set #3* are applied for finding θ as well since the information of one point is sufficient to perform the calculation for it.

Table 3. The results of the angle calculations for different solution sets.

Angle	**Read from the Datasheet / Measured**	**Set #1**	**Set #2**	**Set #3**
α	32.48°	47.02°	33.58°	32.47°
β	9.0°	5.09°	9.54°	8.14°
θ	43.63°	99.23°	43.89°	42.29°

It can be concluded from the results of Table 2 that not all solution sets obtained from the system can be applicable to deduce acceptable estimations for angular variables. In fact, designating a set of points that can induce values with minor errors appears to be a challenging problem. Nonetheless, the experiments indicate that the points near the center of the frame are typically less prone to erroneous estimations. As a matter of fact, having deductions that are close to the actual angles, the values outputted from *Set #2* are utilized to calculate the distances of the points denoted in Table 2.

Certain assessments have been conducted while implementing the machine learning model as well so that the efficiency of the models can be evaluated. As the regression models to address the aim of this project should produce decimal numbers, the performance metrics applied cannot be “accuracy” because of the fact that accuracy checks if the estimated value is exactly the same as the true value. Hence, other criteria

to evaluate the performance are employed instead so that the estimated *x* and *y*-axes coordinates can be analyzed in terms of their proximity to the real values. All of the metrics investigated during this work are related to the errors of the train and test data of the models except for the cosine similarity analysis. Cosine similarity represents the angular difference between the two results. As the values become similar, the angle between their corresponding vectors gets closer to 0 making them alike. The similarity test results are shown for the *x* and *y*-axis models in Figure 15. Furthermore, the losses from the mean squared error of both of the models are provided in Figure 16. The outcomes of the remaining investigations regarding other types of errors are provided for both models in Appendix 4.

The formula for calculating the cosine similarity of two lists, denoted as *A* and *B*, is presented in Equation (27) where the sets become perfectly opposite as the equation outputs -1 and perfectly similar as the product calculates to 1. The angle θ in Equation 24 symbolizes the angle between the two collections to be compared, and the equation divides the dot product of the lists to the multiplication of their magnitudes. A_i indicates an arbitrary element belonging to list *A* of *n* elements whereas B_i is utilized for an arbitrary element of list *B* of the same number of elements as *A*.

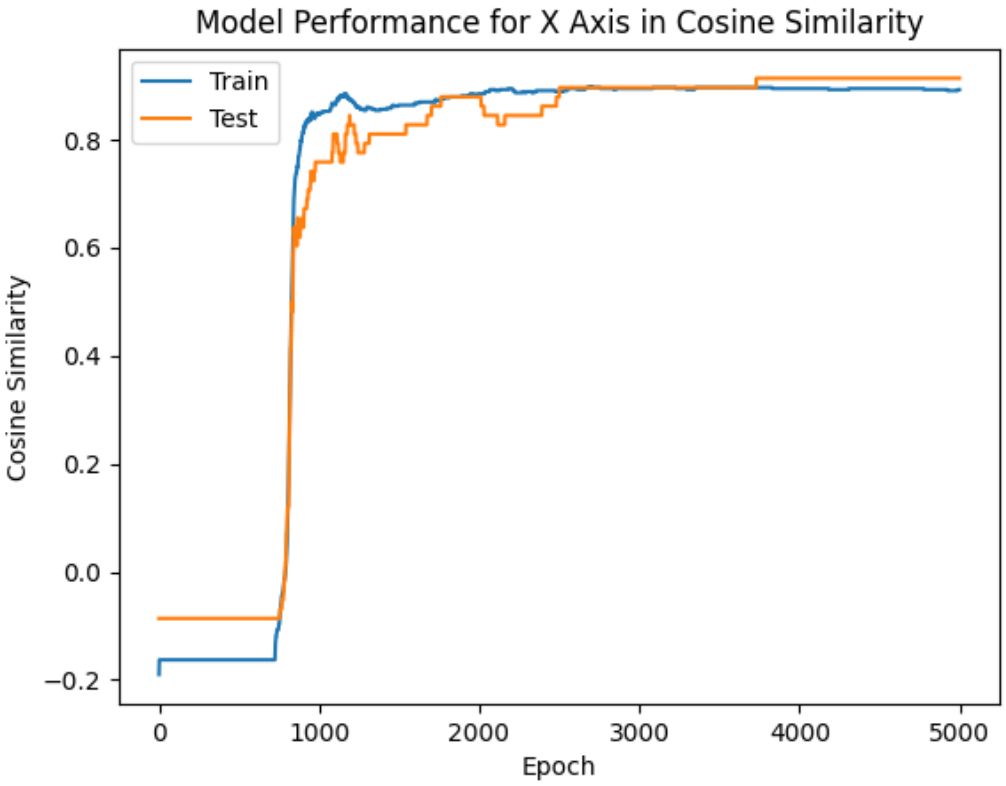
$$Similarity(A,B) = \cos\theta = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} \times \sqrt{\sum_{i=1}^{n} {B_i}^2}} \quad (27)$$

Mean squared error is another valuable statistical analysis as the sizable deviations from the real value add more weight to the analysis while the principle always assures a positive output. As the name suggests, this method takes the average of the sum of the squared errors. Equation (28) describes the calculation of this technique. Similar to Equation (24), *n* denotes the number of elements on which the error is estimated while y_i is an arbitrary true value, and $\tilde{y}_i$ is the estimated representation of this value.

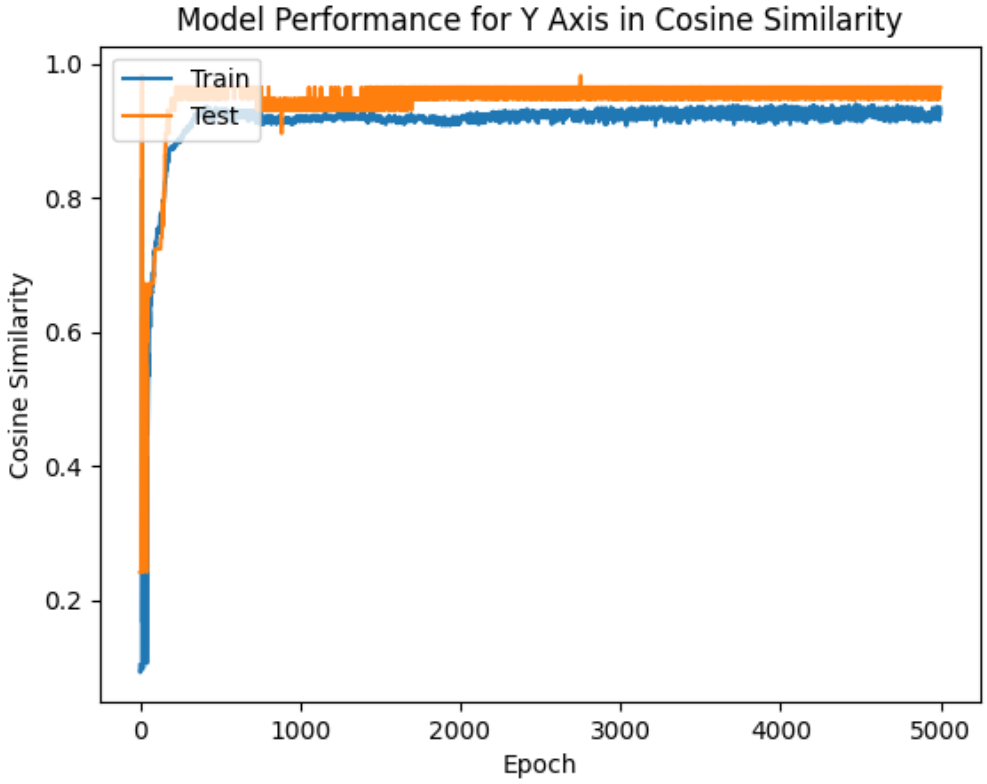
$$Mean\ Squared\ Error\ (MSE) = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 \quad (28)$$

Figure 15 depicts that the cosine similarity of both of the models approaches to 1 as the number of epochs of models increases although the convergence to 1 reaches a steady-state for both train and test data after some epochs. The similarity of the *x*-axis can be said to have reached its maximum near 4000 epochs for the test and 2500 for the train while the maximum reach occurred near 2000 epochs for the *y*-axis. Furthermore, Figure

16 describes the mean squared error with respect to the increasing epoch numbers which indicates that the losses decrease as epochs increase in the train and test data of both of the models as expected. However, especially with the *x*-axis model, the test data can be seen to descend to lower error values than the training set which might indicate that the test samples generalized better as the size of the test samples is rather small. Generating novel train and test sets with different separation percentages can simply illuminate the reason for this phenomenon. Yet, the decrease of the error observed in all datasets of both models to diminutive values after 3000 epochs in *x*-axis and 2000 epochs in *y*-axis, in fact, proves that the networks operate successfully.

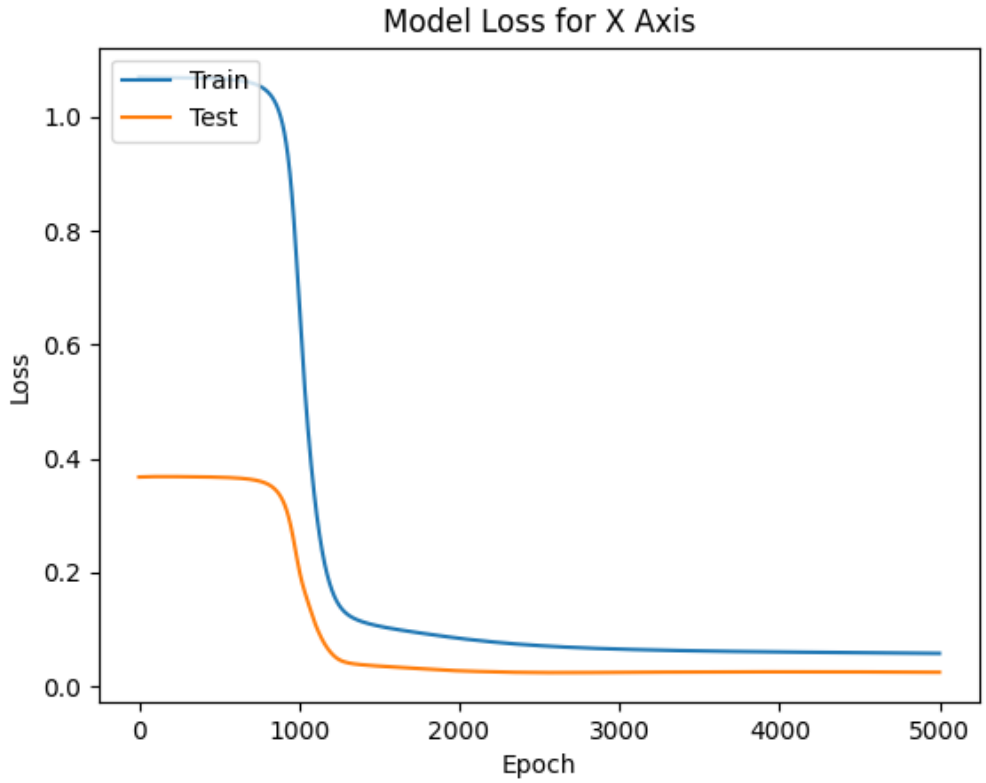


(a)

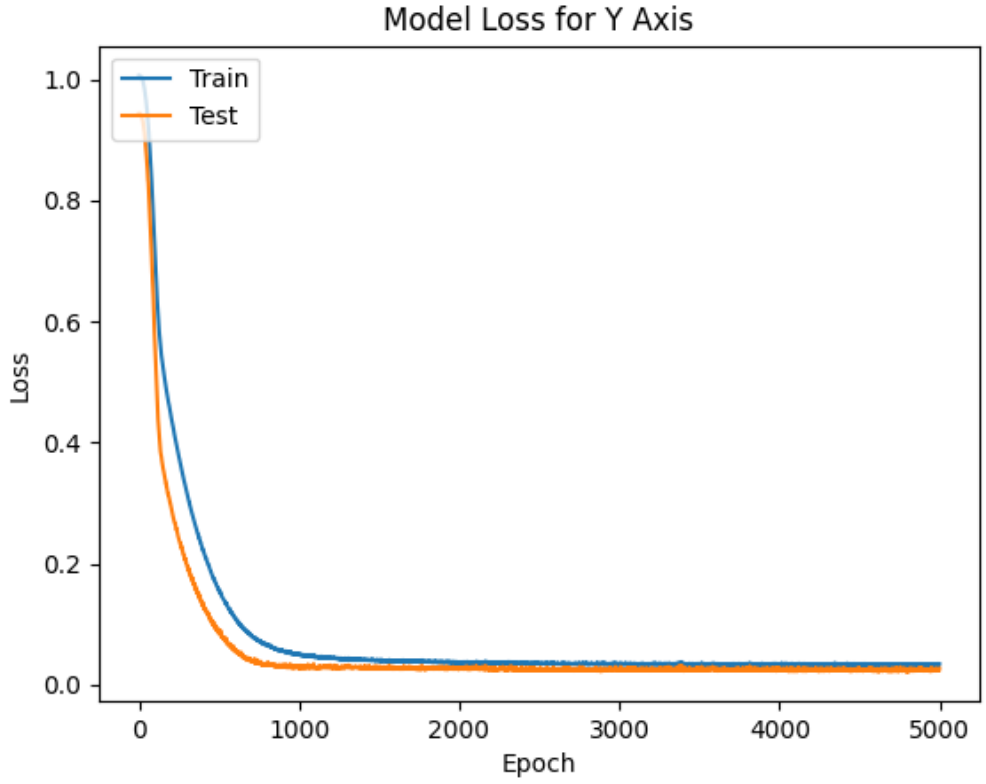


(b)

Figure 15. Model performance analysis via cosine similarity for (a) the x-axis estimations and (b) the y-axis estimations.



(a)



(b)

Figure 16. Model loss analysis via mean squared error for (a) the x-axis and (b) the y-axis estimations.

6 Summary

Camera-Based Vehicle Location Detection is a project to be developed on an embedded system that utilizes image processing features to create a mapping analysis of the system environment. The system is intended to be implemented on the decks of ferries and possibly other types of Ro-Ro ships to be able to monitor the vehicle traffic on the ship in real-time. The camera on the system can produce a live feed of the deck to which certain object detection and distance estimation algorithms are applied. Therefore, the coordinates of the vehicles on the ships can be tracked down in a convenient manner.

This thesis work analyzed certain algorithms for detecting objects and their locations so that the optimized solution methodology could be crafted for the project. For object detection, some of the most common one-stage detectors were implemented and compared for accuracy, namely MobileNet SSD, YOLO v3 along with its alternative YOLO v3-tiny, and YOLO v4 as well as YOLO v4-tiny. Comparing these models showed that YOLO v4 is more successful than the rest in delivering accurate findings with the ability to perform satisfactorily in complex cases. Furthermore, three different distance estimation techniques were put to test in order to inspect their precision. The first formulation using intrinsic camera parameters to represent the object depth functioned more defectively whereas constructing a set of equations to evaluate the distances in different axes for mapping with the help of geometric modeling proved to be a fast, simple, and a rather accurate approach. Lastly, a machine learning model was also developed so as to analyze how much the accuracy improves compared to the geometrical methodology, and the results confirmed that the enhancement on the error percentage was not at a grand scale to make the geometrical representation a poor alternative.

The experiments have mainly demonstrated that the error percentage difference between the geometrical approach and the machine learning models are not grand where the network models performed with an average error percentage of -3.53% from 21 arbitrarily selected frames and the geometrical calculation resulted in -4.70%. As the reasons for errors are relatively larger with camera parameters utilization, the method performed poorly with an average error percentage of 18.59%.

The further tasks to be executed for the Camera-Based Vehicle Location Detection project mainly involves the implementation of the algorithms proposed in this study on the selected hardware platform as well as the detailed performance analysis with the different criterion on the embedded system. Additional detailed and explicit analyses of the angle calculation technique might also uncover more criteria for the proper solution set selection. Finally, although the object coordinates deduction strategies have been implemented, these findings need to be adapted for a generic mapping representation of the vessel which requires some extra work on the outcomes of this thesis work.

References

[1] Puisa, R. (2018). Optimal stowage on Ro-Ro decks for efficiency and safety. *Journal of Marine Engineering & Technology, 20, 1-17*.

[2] Park, D.J., & Choi, Y.B. (2006). Implementation of Ubiquitous Port Operation System Using RTLS. *The Journal of the Korea Contents Association, 6, 128–135*.

[3] Hur, D.C., & Lee, K.Y. (2007). Design and Implementation of Physical Distribution Management System Using RFID and GPS. *Proceedings of the Korean Institute of Information and Commucation Sciences Conference, The Korea Institute of Information and Commucation Engineering*.

[4] Roehrig, C., Heller, A., Heß, D., & Künemund, F. (2014). Global Localization and Position Tracking of Automatic Guided Vehicles using passive RFID Technology.

[5] M'hand, M.A, Boulmakoul, A., Badir, H., & Lbath, A. (2019). A scalable real-time tracking and monitoring architecture for logistics and transport in RoRo terminals. *Procedia Computer Science, 151, 218-225*.

[6] Tian, D., Lin, C., Zhou, J., Duan, X., Cao, Y., Zhao, D., & Cao, D. (2020). SA-YOLOv3: An Efficient and Accurate Object Detector Using Self-Attention Mechanism for Autonomous Driving. *IEEE Transactions on Intelligent Transportation Systems, 1-12*.

[7] Soviany, P., & Ionescu, R. (2018). Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 209-214*.

[8] Bochkovskiy, A., Wang, C.Y., & Liao, H.Y.M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.

[9] Jocher, G., Kwon, Y., guigarfr, Veitch-Michaelis, J., perry0418, Ttayu, Marc, Bianconi, G., Baltacı, F., Suess, D., Chen, T., Yang, P., idow09, WannaSeaU, Xinyu, W., Shead, T.M., Havlik, T., Skalski, P., NirZarrabi, LukeAI, LinCoce, Hu, J., IlyaOvodov, GoogleWiki, Reveriano, F., Falak, & Kendall, D. (2020). ultralytics/yolov3: 43.1mAP@0.5:0.95 on COCO2014. *doi:10.5281/zenodo. 3785397*.

[10] Tan, M., Pang, R., & Le, Q.V. (2020). EfficientDet: Scalable and Efficient Object Detection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10778-10787*.

[11] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., & Fu, C.Y., & Berg, A. (2016). SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science, 21-37*.

[12] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition, 580-587*.

[13] Girshick, R. (2015). Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV), 1440-1448*.

[14] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*.

[15] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask R-CNN.

[16] Hu, X., Xu, X., Xiao, Y., Chen, H., He, S., Qin, J., & Heng, P.A. (2019). SINet: A Scale-Insensitive Convolutional Neural Network for Fast Vehicle Detection. *IEEE Transactions on Intelligent Transportation Systems, 20(3), 1010-1019*.

[17] Forrest, N.I., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.

[18] Schönberger, J., & Frahm, J.M. (2016). Structure-from-Motion Revisited. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4104-4113*.

[19] Zhang, J., Hu, S., & Shi, H. (2018). Deep Learning based Object Distance Measurement Method for Binocular Stereo Vision Blind Area. *International Journal of Advanced Computer Science and Applications, 9*.

[20] Liao, Y., Huang, L., Wang, Y., Kodagoda, S., Yu, Y., & Liu, Y. (2016). Parse Geometry from a Line: Monocular Depth Estimation with Partial Laser Observation.

[21] Mustafah, Y., Noor, R., Hasbi, H., & Azma, A. (2012). Stereo vision images processing for real-time object distance and size measurements. *2012 International Conference on Computer and Communication Engineering (ICCCE), 659-663*.

[22] Akepitaktam, P., & Hnoohom, N. (2019). Object Distance Estimation with Machine Learning Algorithms for Stereo Vision. *2019 14th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP), 1-6*.

[23] Pratama, M., Budi, W., Dimyani, S., Praptijanto, A., Nur, A., & Putrasari, Y. (2019). Performance of Inter-vehicular Distance Estimation: Pose from Orthography and Triangle Similarity. *2019 International Conference on Sustainable Energy Engineering and Application (ICSEEA), 37-41*.

[24] Chiu, Y.C., Tsai, C.Y., Ruan, M.D., Shen, G.Y., & Lee, T.T. (2020). Mobilenet-SSDv2: An Improved Object Detection Model for Embedded Systems. *2020 International Conference on System Science and Engineering (ICSSE), 1-5*.

[25] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788*.

[26] Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger.

[27] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement.

[28] Adarsh, P., Rathi, P., & Kumar, M. (2020). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), 687-694*.

[29] Jiang, Z., Zhao, L., Li, S., & Jia, Y. (2020). Real-time object detection method based on improved YOLOv4-tiny.

[30] Kendal, D. (2007). Measuring distances using digital cameras. *Australian Senior Mathematics Journal, 21*.

[31] Joglekar, A., Joshi, D., Khemani, R., Nair, S., & Sahare, S. (2011). Depth Estimation Using Monocular Camera.

[32] Taylor, T., Geva, S., & Boles, W. (2004). Monocular Vision as a Range Sensor.

[33] Jin, F., Zhao, Y., Wan, C., Yuan, Y., & Wang, S. (2021). Unsupervised Learning of Depth from Monocular Videos Using 3D-2D Corresponding Constraints. *Remote Sensing, 13(9)*.

[34] Xiong, L., Wen, Y., Huang, Y., Zhao, J., & Tian, W. (2020). Joint Unsupervised Learning of Depth, Pose, Ground Normal Vector and Ground Segmentation by a Monocular Camera Sensor. *Sensors, 20, 3737*.

[35] Zhu, J., Fang, Y., Abu-Haimed, H., Lien, K.C., Fu, D., & Gu, J. (2019). Learning Object-specific Distance from a Monocular Image. *2019 IEEE/CVF International Conference on Computer Vision (ICCV), 3838-3847*.

[36] Haseeb, M. (2018). DisNet: A novel method for distance estimation from monocular camera.

[37] Dargan, S., Kumar, M., Ayyagari, M., & Kumar, G. (2019). A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning. *Archives of Computational Methods in Engineering, 1-22*.

[38] Chauhan, N., & Singh, K. (2018). A Review on Conventional Machine Learning vs Deep Learning. *2018 International Conference on Computing, Power and Communication Technologies (GUCON), 347-352*.

[39] Patel, H. (2020). KITTI distance estimation. *Retrieved from* https://github.com/harshilpatel312/KITTI-distance-estimation. (Accessed in 2021).

[40] e-con Systems™ (2020). e-CAM130A_CUXVR Lens Datasheet. *Revision 1.4*.

[41] Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., & Geiger, A. (2017). Sparsity Invariant CNNs. *International Conference on 3D Vision (3DV)*.

[42] Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., & Beijbom, O. (2020). nuScenes: A multimodal dataset for autonomous driving.

[43] e-CAM130A_CUXVR - Multiple Camera Board for NVIDIA® Jetson AGX Xavier™. *Retrieved from* https://www.e-consystems.com/nvidia-cameras/jetson-agx-xavier-cameras/four-synchronized-4k-cameras.asp. (Accessed in 2021).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Mustafa Furkan Kopar

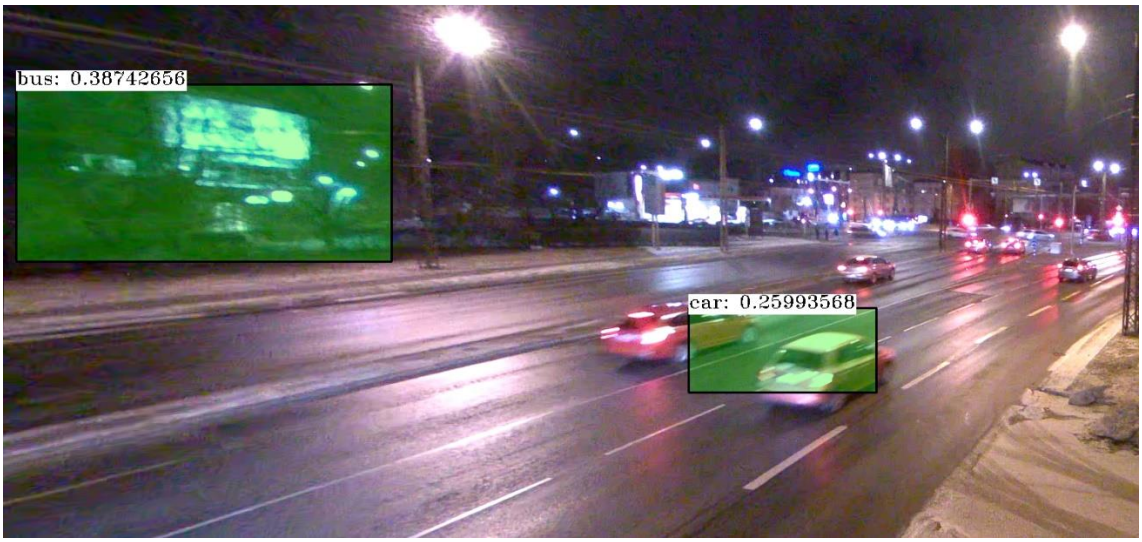
1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Camera-Based Vehicle Location Detection”, supervised by Uljana Reinsalu, Jürgen Soom and Mairo Leier
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

02.08.2021

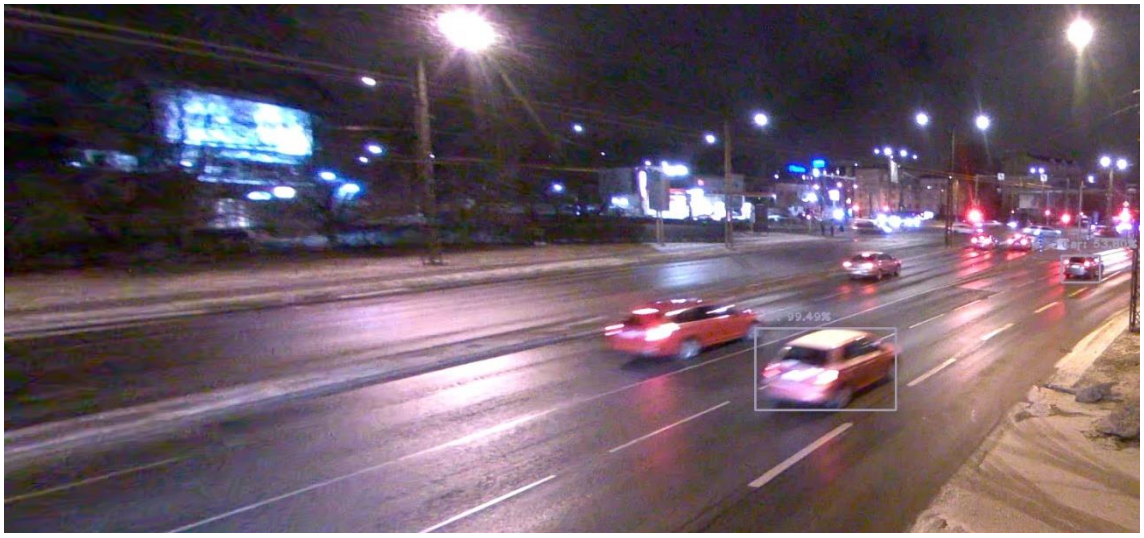
1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Object detection results for selected images

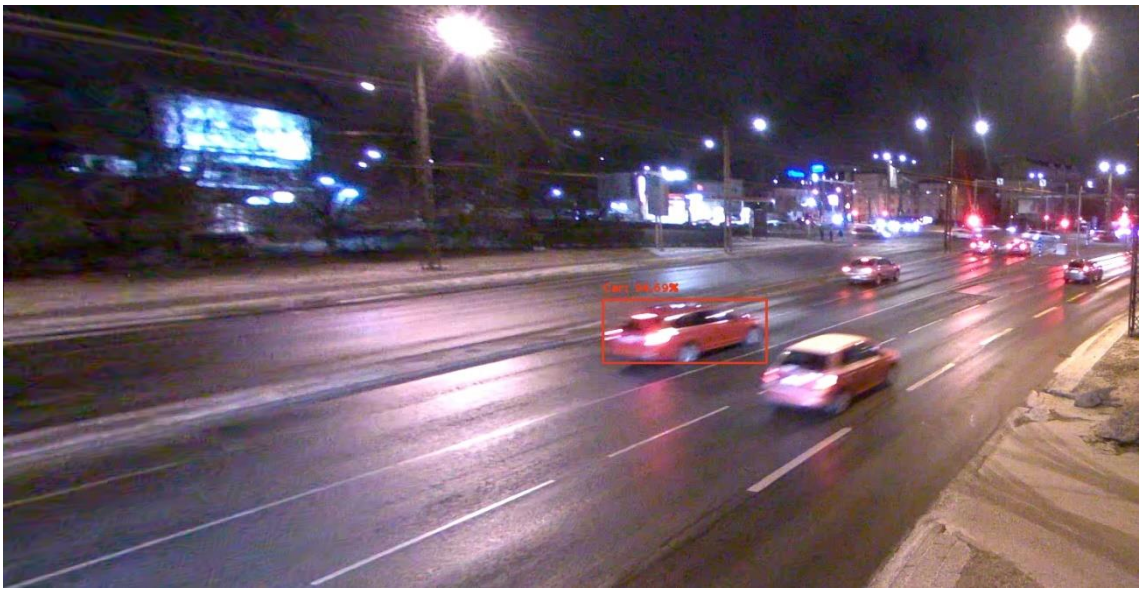
The experimental results for object detection conducted by Mobile SSD, YOLO v3, YOLO v3-tiny, YOLO v4, and YOLO v4-tiny detectors are presented from Figure 17 to Figure 19 provided that objects were able to be detected on the samples by the corresponding detectors. The tiny variant of YOLO v3 was not able to detect any vehicles on the sample image in Figure 17 while MobileNet SSD has a false positive detection of a bus in each figure. The deduction of YOLO v3 and v4 being superior to the tiny variants in terms of the number of detections can also be derived from these images.



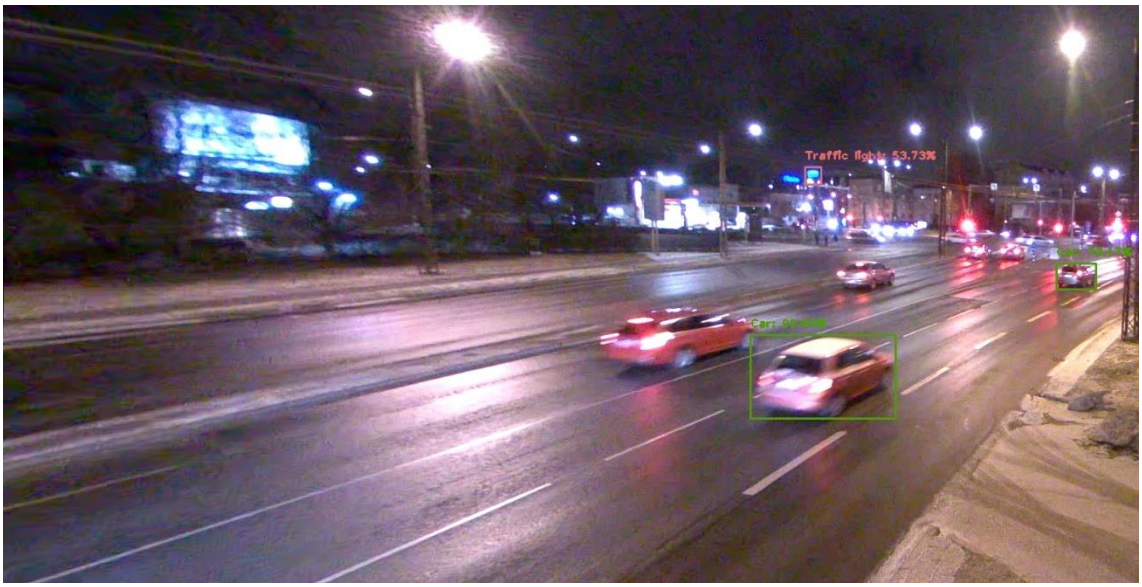
(a)



(b)

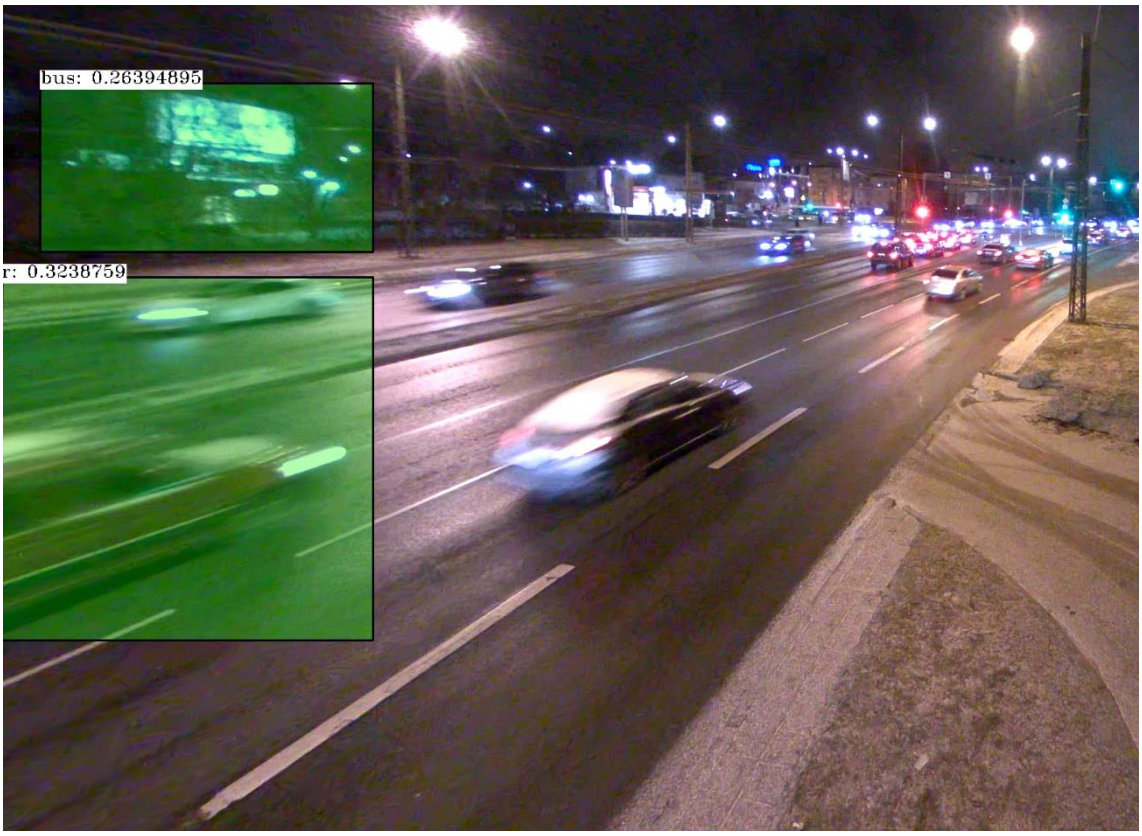


(c)

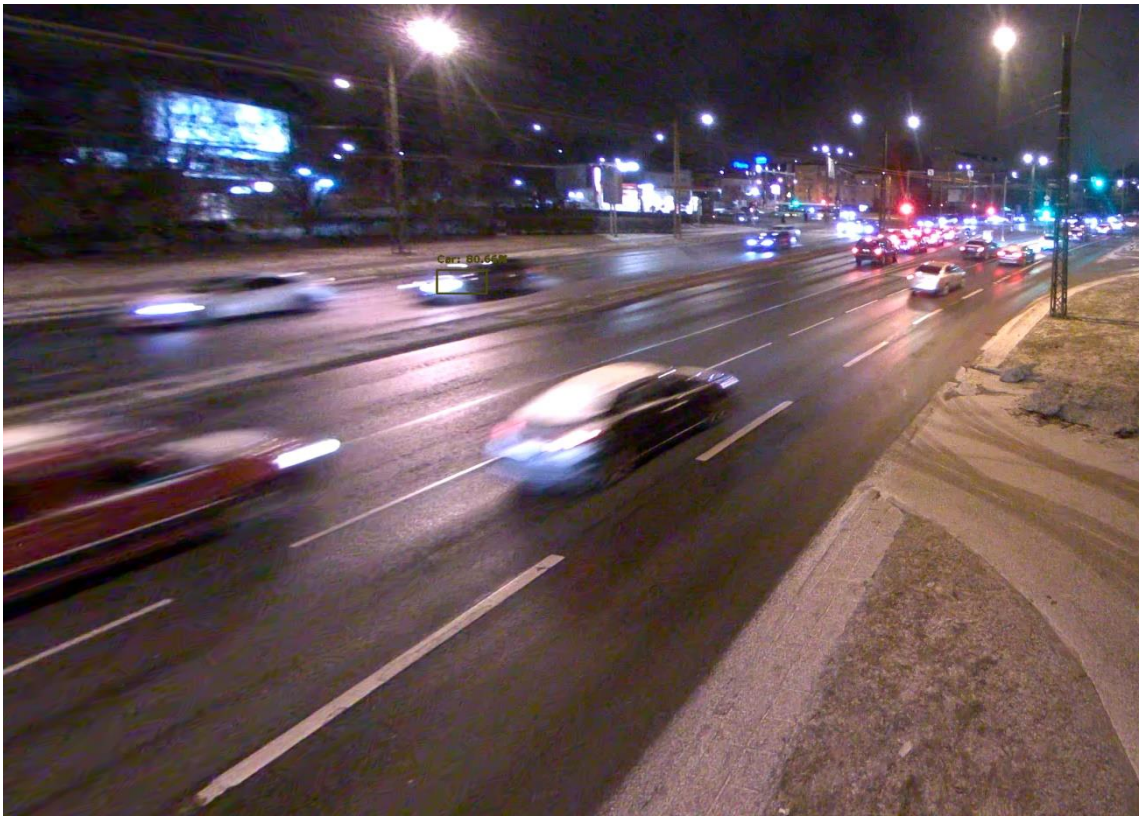


(d)

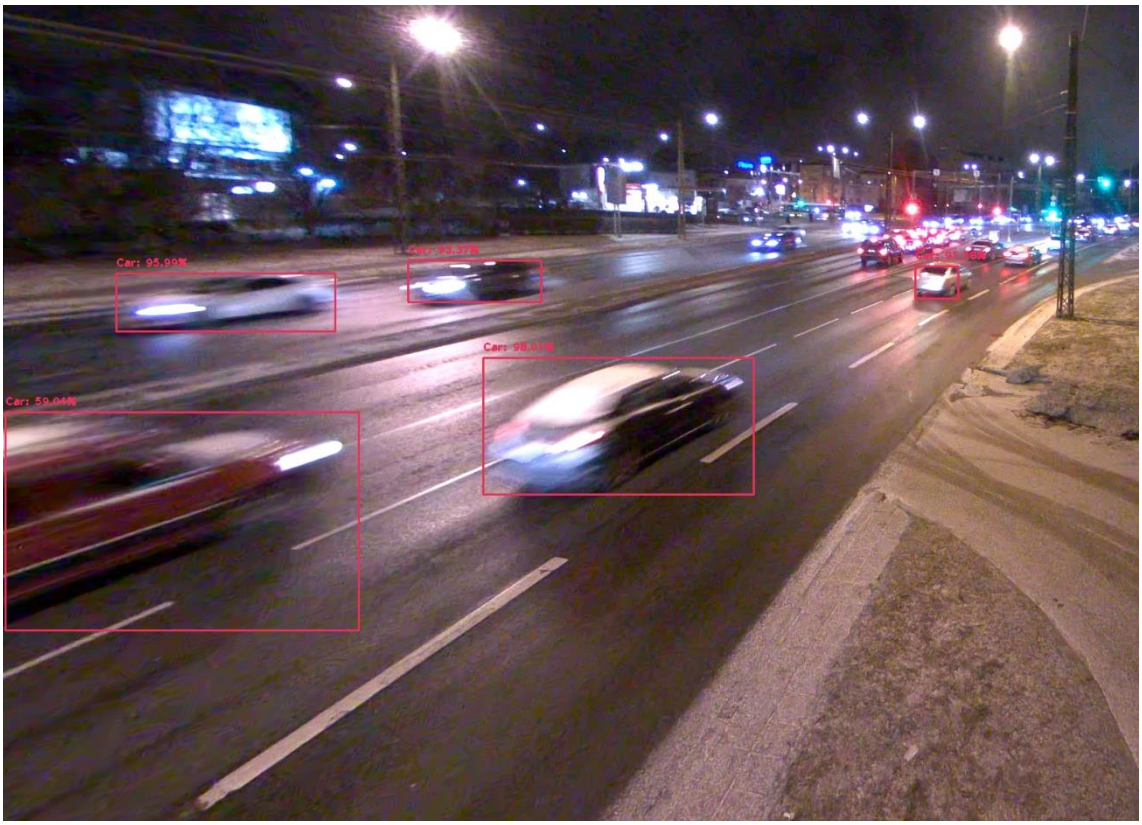
Figure 17. Results of the vehicle detection on a sample image (Frame #1 in Table 1) with (a) MobileNet SSD, (b) YOLO v3, (c) YOLO v4-tiny, and (d) YOLO v4. YOLO v3-tiny failed to detect objects on this sample.



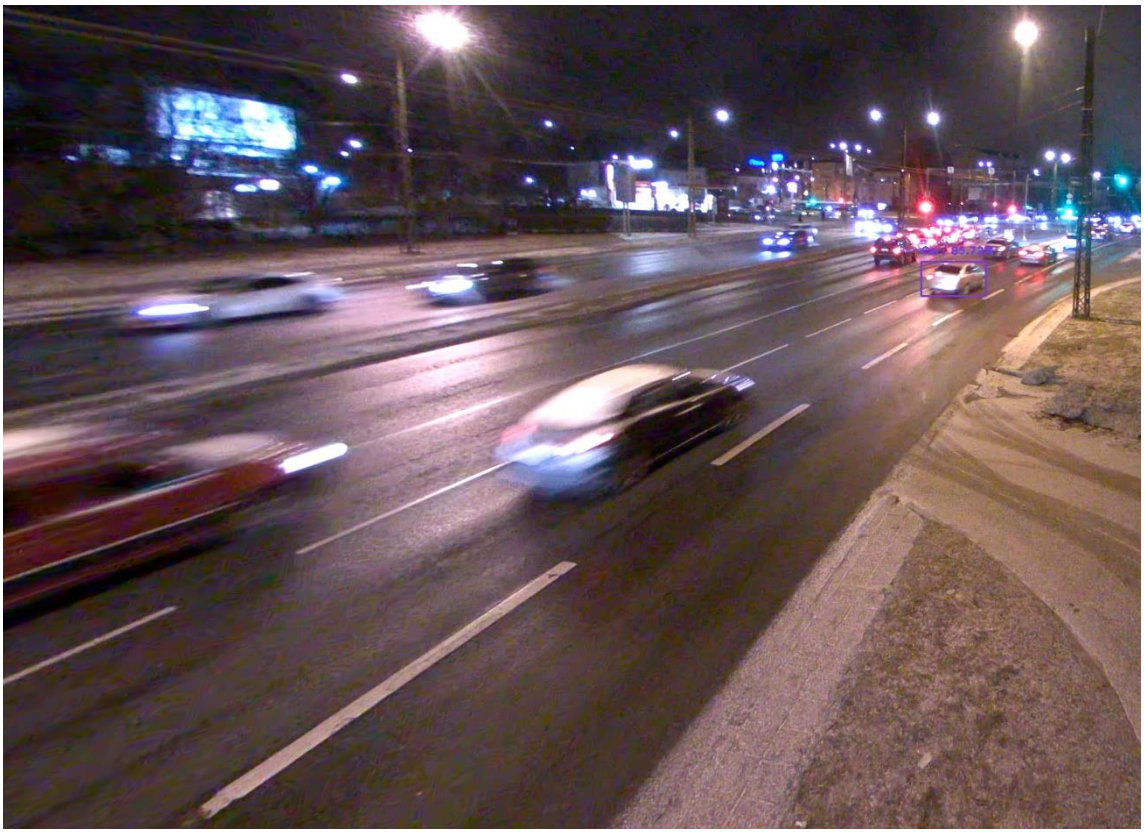
(a)



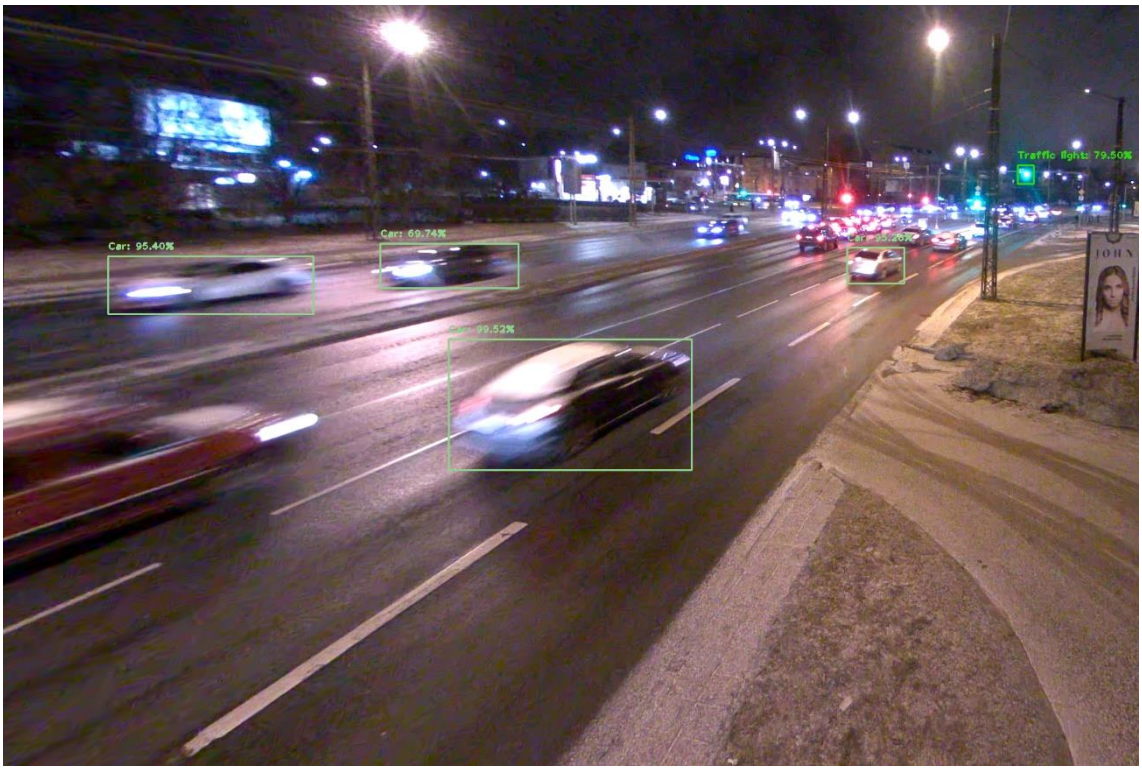
(b)



(c)

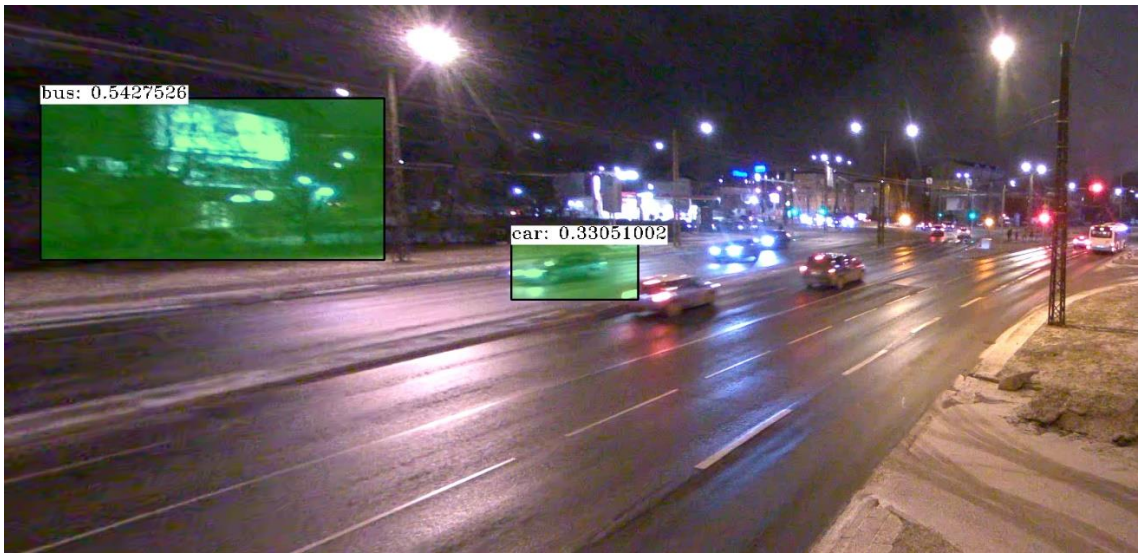


(d)

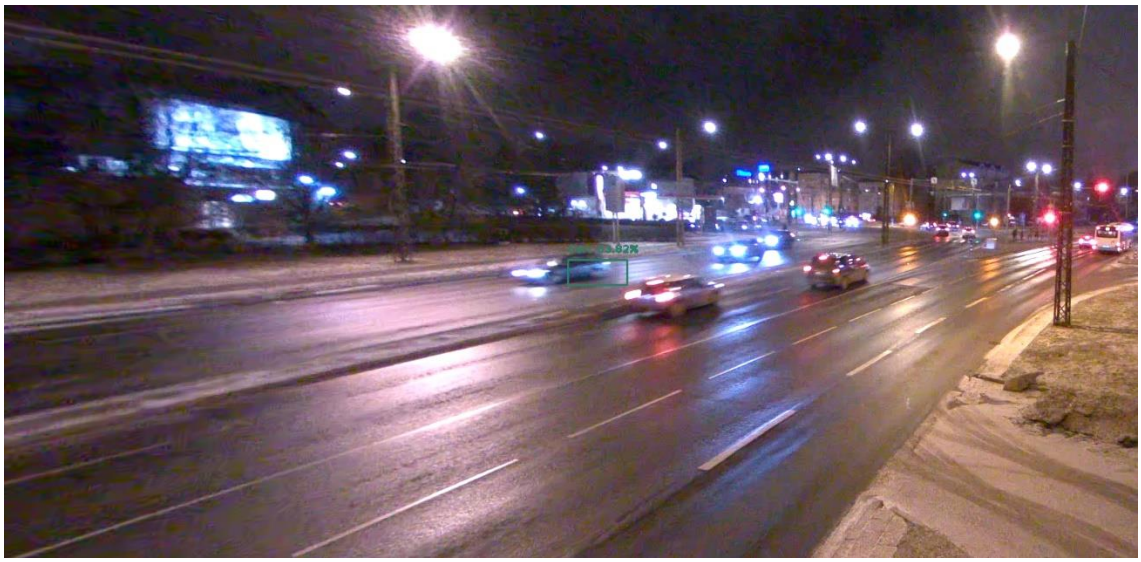


(e)

Figure 18. Results of the vehicle detection on a sample image (Frame #2 in Table 1) with (a) MobileNet SSD, (b) YOLO v3-tiny, (c) YOLO v3, (d) YOLO v4-tiny, and (e) YOLO v4.

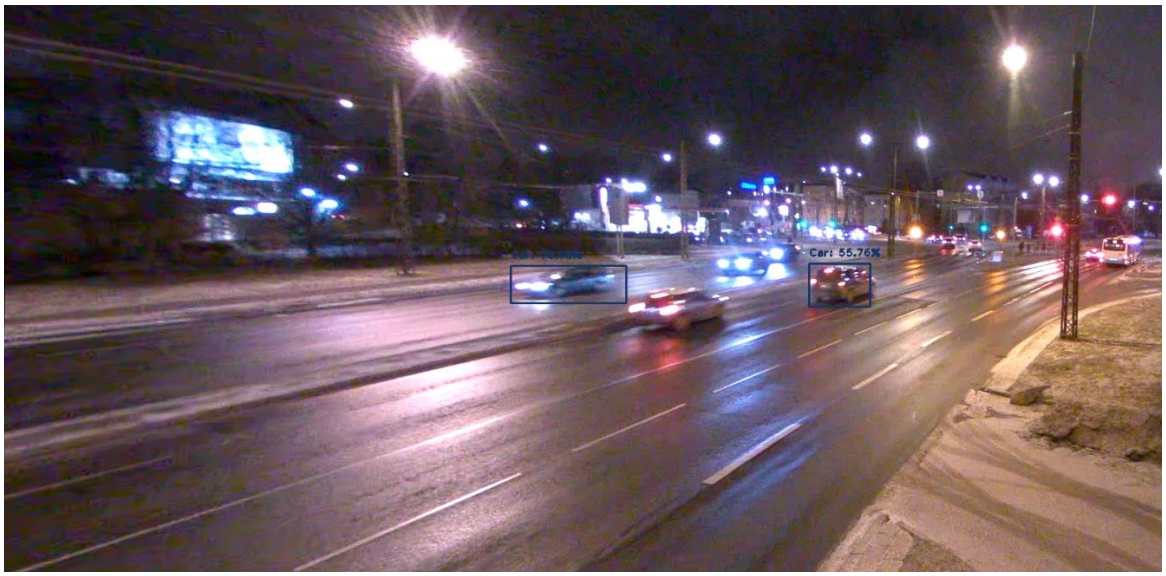


(a)

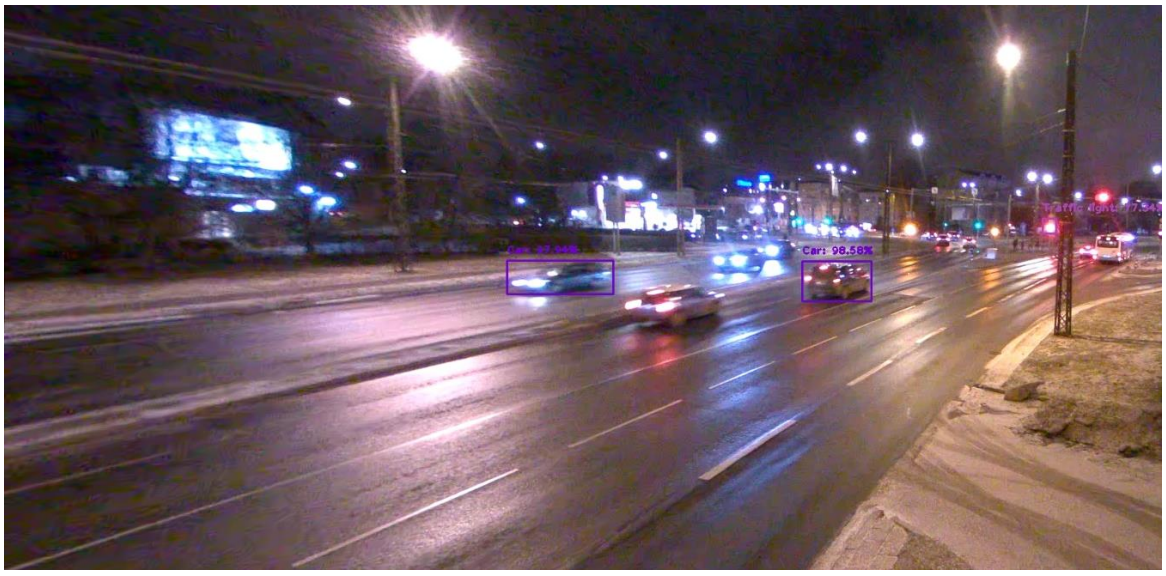


(b)

(c)



(d)



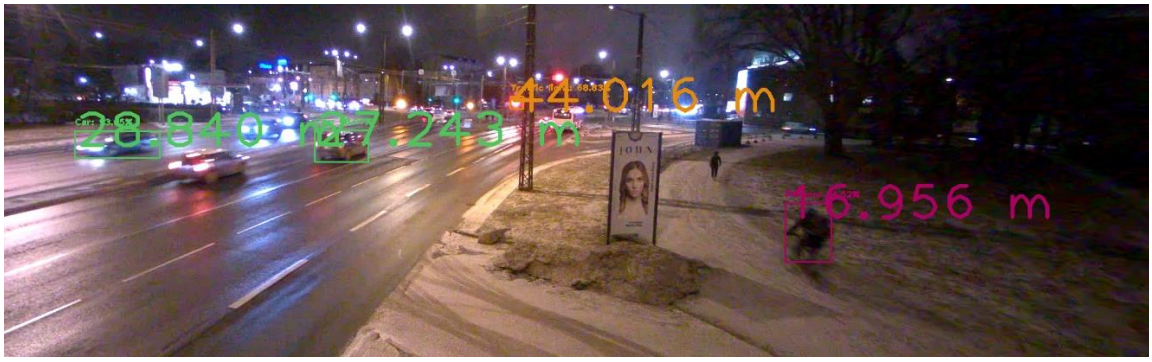
(e)

Figure 19. Results of the vehicle detection on a sample image (Frame #4 in Table 1) with (a) MobileNet SSD, (b) YOLO v3-tiny, (c) YOLO v3, (d) YOLO v4-tiny, and (e) YOLO v4.

Appendix 3 – Distance estimation results for selected images

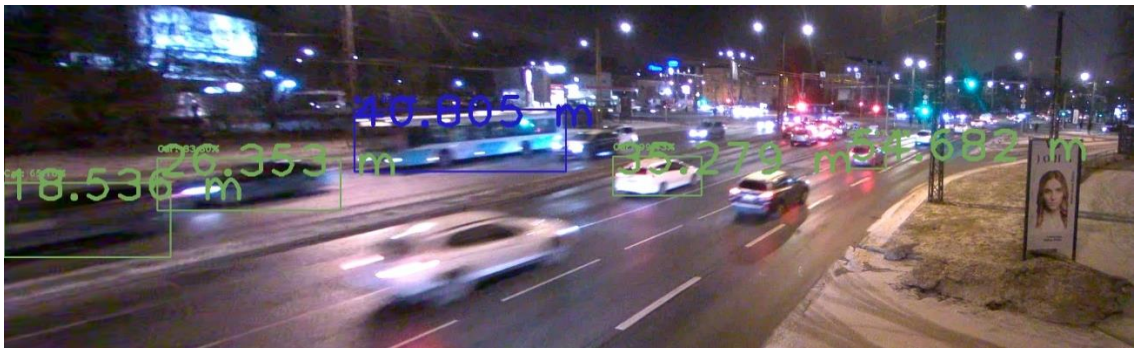
The experimental results for distance estimation conducted by camera parameters utilization and geometrical approach are presented from Figure 20 to Figure 22 where the object detection is performed with YOLO v4 detector. The results from the machine learning methodology were not inspected on the frames but obtained as a resulting dataset. Comparing the results of the two algorithms provided in these frames suggests that their estimations generally differ from each other on a large scale while most of the time, the calculations of the geometrical approach are the ones producing more sensible and accurate results.

(a)

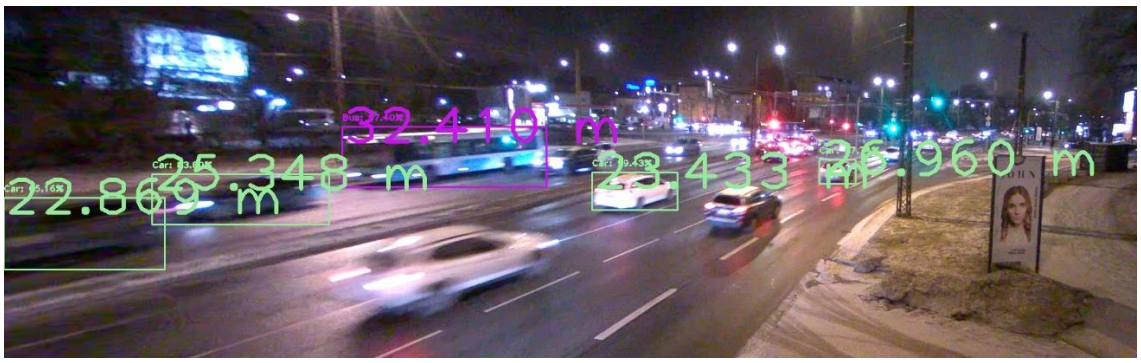


(b)

Figure 20. Results of the distance estimation on a sample image (Frame #10 in Table 2) with (a) camera parameters utilization and (b) geometrical approach.

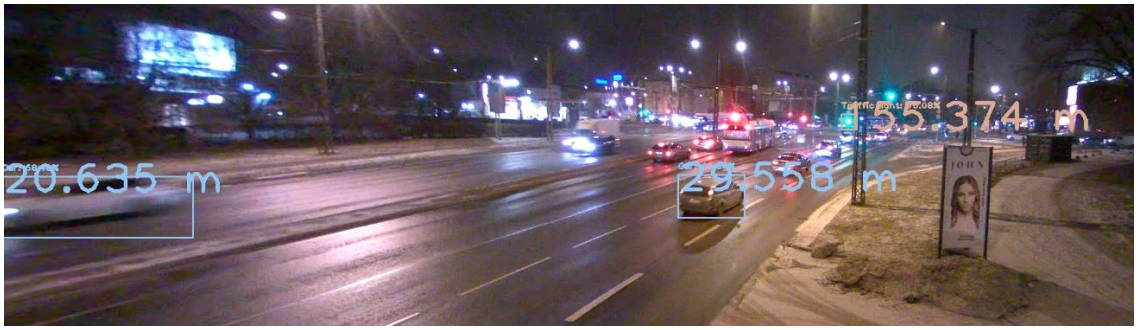


(a)



(b)

Figure 21. Results of the distance estimation on a sample image (Frame #15 in Table 2) with (a) camera parameters utilization and (b) geometrical approach.



(a)

(b)

Figure 22. Results of the distance estimation on a sample image (Frame #20 in Table 2) with (a) camera parameters utilization and (b) geometrical approach.

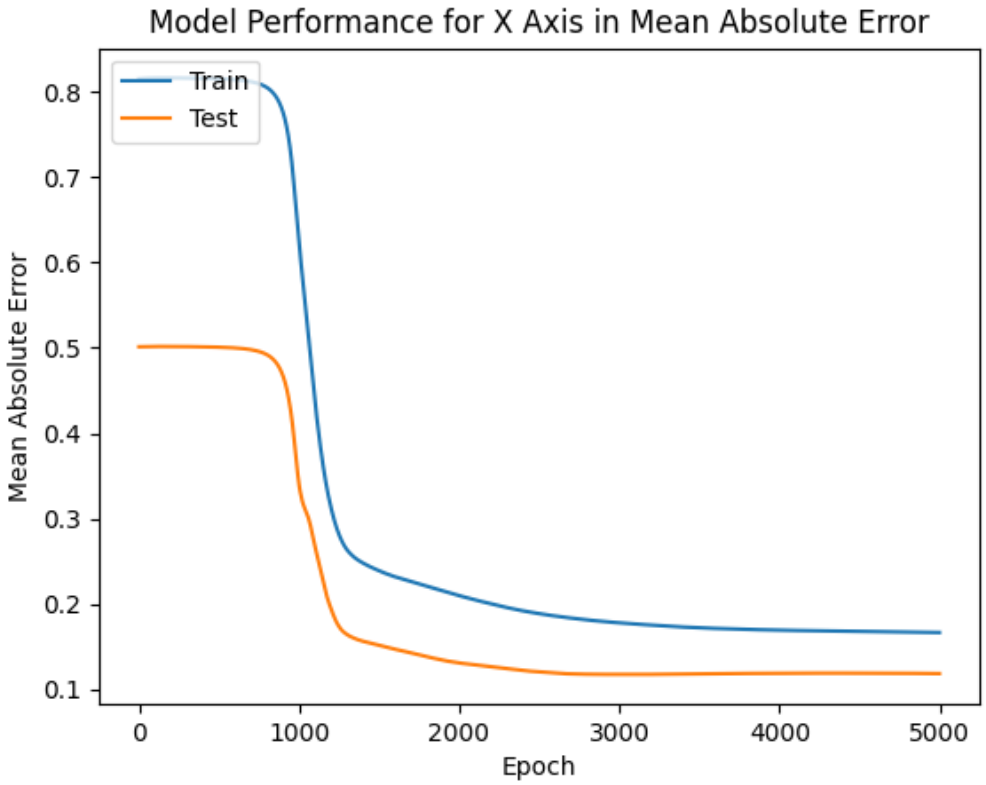
Appendix 4 – Performance analysis for machine learning models

The experimental results for estimating the performance of the machine learning models are presented in Figure 23 and Figure 24. Figure 23 represents the mean absolute error of the models calculated by the formula presented in Equation (29), and Figure 24 shows the root mean squared error of the models according to Equation (30).

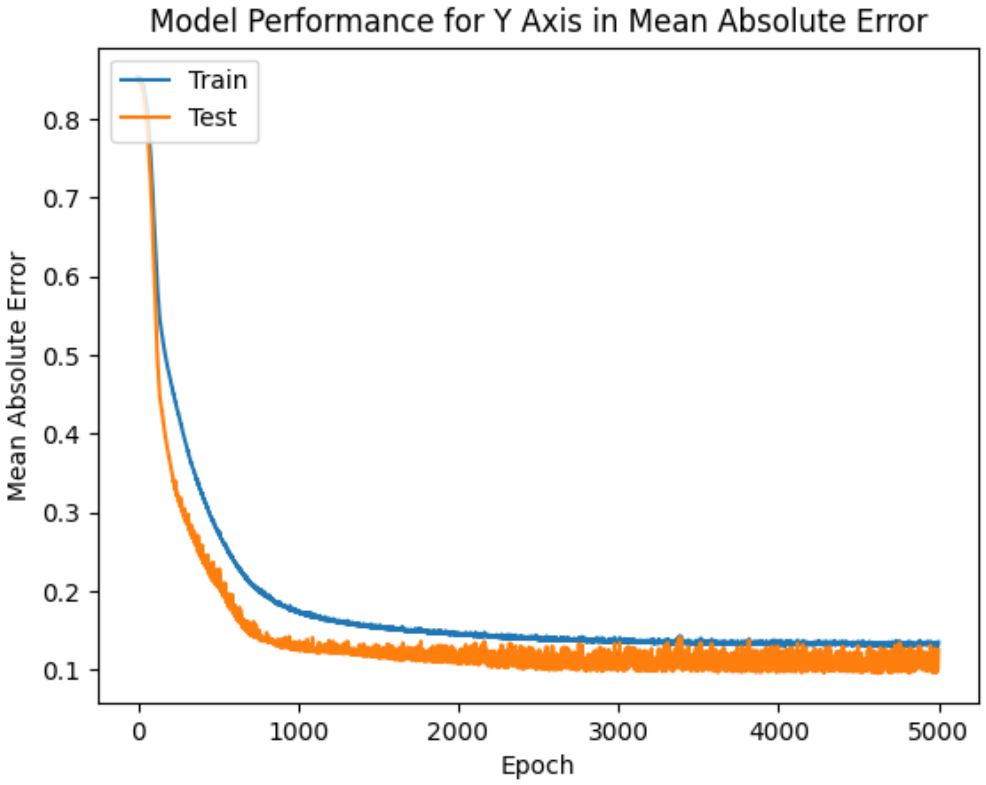
$$Mean\ Absolute\ Error\ (MAE) = \frac{1}{n} \times \sum_{i=1}^{n} |y_i - \tilde{y}_i| \quad (29)$$

$$Root\ Mean\ Squared\ Error\ (RMSE) = \sqrt{MSE} = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2} \quad (30)$$

The results for both MAE and RMSE are indeed rather similar to the results of the loss function MSE provided in Figure 16. The difference between the train and test error performances that is especially more notorious with the *x*-axis model stems from the same analogy described in Section 5.2 where the small test dataset results in a better fit with smaller errors. However, this does not necessarily mean that the models are faulty or performed poorly, new datasets with various sizes can be applied to the model to confirm the reason for this observation. As the error values decrease rather drastically with the increasing epochs, usually around 2000 in the *y*-axis model and near 3000 in the *x*, the networks can be concluded to operate as expected.

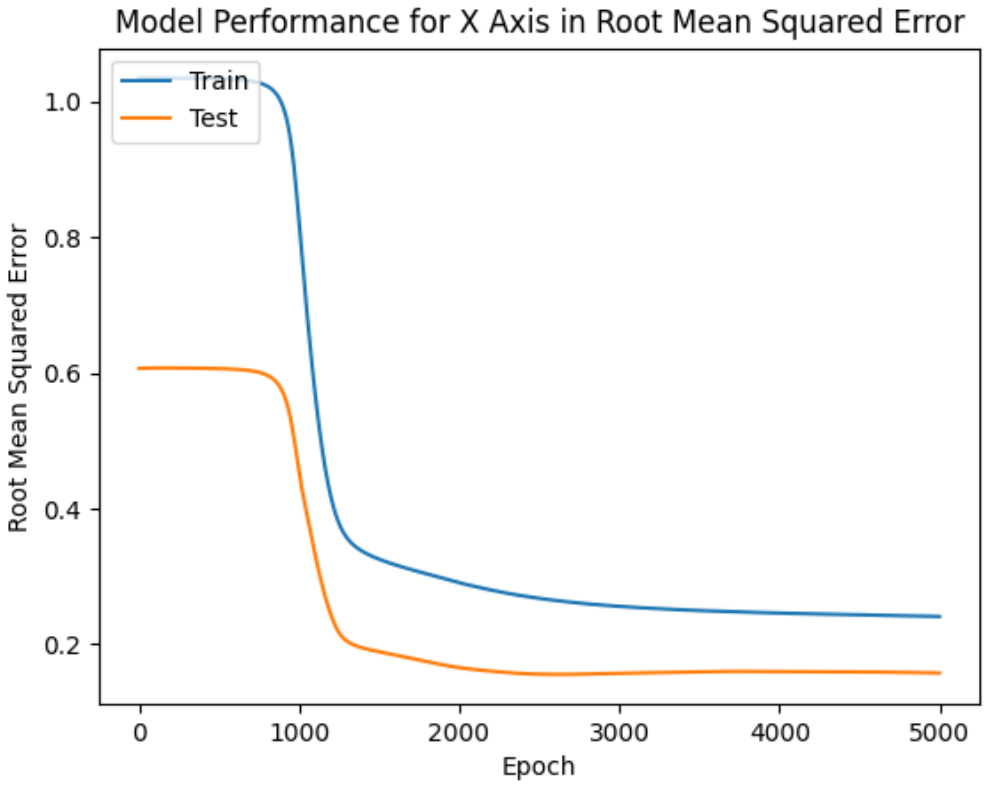


(a)

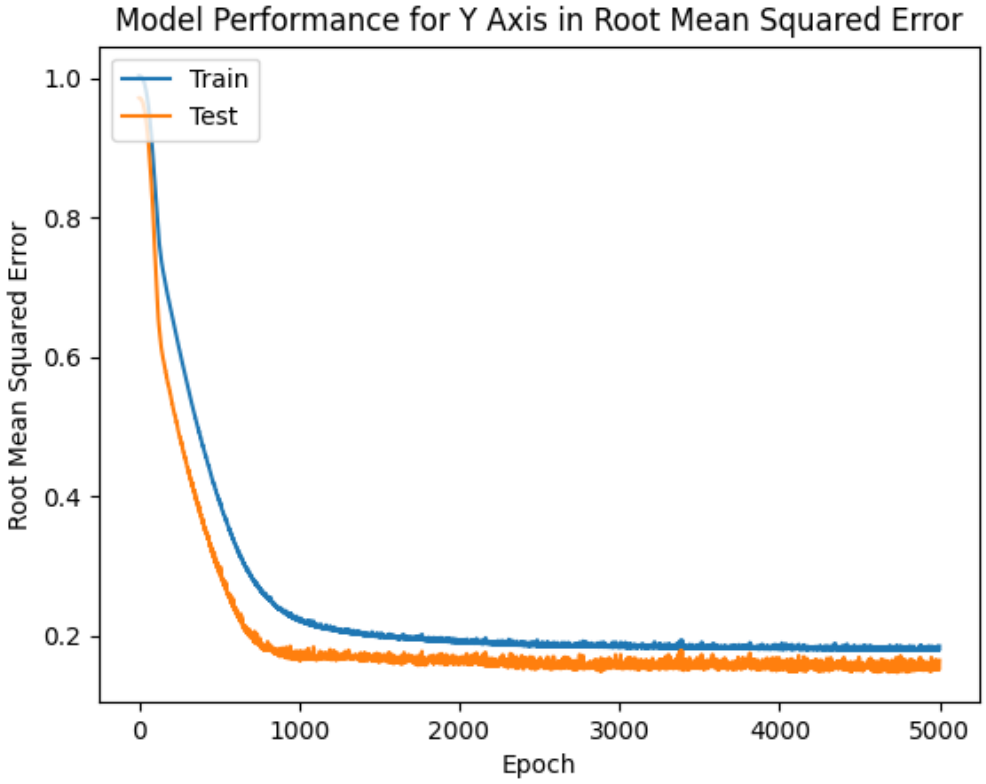


(b)

Figure 23. Model performance analysis via mean absolute error for (a) the x-axis estimations and (b) the y-axis estimations.



(a)



(b)

Figure 24. Model performance analysis via root mean squared error for (a) the x-axis estimations and (b) the y-axis estimations.

Appendix 5 – The repository link of the thesis work

The link to the Tallinn University of Technology GitLab repository which includes the basic data and source codes described in the thesis “Camera-Based Vehicle Location Detection” is as follows:

- https://gitlab.cs.ttu.ee/mukopa/camera-based-vehicle-location-detection