CHALMERS
UNIVERSITY OF TECHNOLOGY

3-D object tracking through the use of a single camera and the motion of a driverless car

Master's thesis in Complex Adaptive Systems

Andreas Ovnell

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

3-D object tracking through the use of a single camera and the motion of a driverless car

ANDREAS OVNELL

Department of Mechanics and Maritime Sciences
Division of vehicle engineering and autonomous systems
Chalmers University of Technology
Gothenburg, Sweden 2021

3-D object tracking through the use of a single camera and the motion of a driverless car
ANDREAS OVNELL

Supervisor: Ola Benderius, Department of mechanics and maritime sciences
Examiner: Ola Benderius, Department of mechanics and maritime sciences

Technical report no 2021:85
Department of Mechanics and Maritime Sciences
Division of vehicle engineering and autonomous systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Digitaltryck
Gothenburg, Sweden 2021

3-D object tracking through the use of a single camera and the motion of a driverless car
ANDREAS OVNELL
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

There has been a very large increase in interest and development of partially or fully driverless cars in recent years. For these driverless cars to function, they need to be able to navigate to their destination while avoiding nearby objects. This can be done using *simultaneous localisation and mapping* (SLAM). SLAM is the task of simultaneously creating a map of the surrounding objects while keeping track of the car's position within this map. This thesis will look into the feasibility of using a single camera attached on a driverless car to perform SLAM on cones detected by the real-time object detection system *You only look once* (YOLO).

Three different methods were tested. All of these require a calibrated camera that is capable of determining horizontal and vertical angles from the pixel positions. The first 'triangulation' method uses that the distance travelled and rotation between two frames is known. The second 'plane projection' method is an optimisation problem which consists of finding the variables which result in lowest error, and through this determine the cone distances and car speed. The map of the surrounding cones is moved according to the estimated velocity and rotation of the car such that the car is always placed at the origin, allowing for use of multiple detections to improve accuracy. The third 'distance from cone height' method works by using the size of the cone detections in order to determine the approximate distance of each cone, use this to determine the approximate angle of the camera and then use the median angle to make the final distance estimates.

The triangulation method was shown to be completely unsuitable for mono-camera use. The plane projection method was shown to be unreliable, likely due to a relatively small number of visible cones and a too large noise amplitude of detections from YOLO. The distance from cone height method was shown to be the best out of the tested methods, as it was simple, fast and quite reliable. However, this method still had an error approximately 1.4 times larger than what is advertised by commercial stereo camera systems.

Keywords: SLAM, driverless, autonomous, depth vision, 3D, mono camera

Acknowledgements

I would like to thank Ola Benderius for his guidance, help and patience while I was working on this thesis. Another thanks to my family for allowing me to bounce off ideas during the early stages of the project.

Andreas Ovnell, Gothenburg, Dec 2021

Contents

List of Figures

1

Introduction

Autonomous vehicles is a field which has become of great interest in recent years. This is especially the case for driverless cars. Many dream of a near future where we can simply give our destination and let our car drive us wherever we want to go, whether that be work, friends, family, or anywhere else. This has the ability to free up a considerable amount of time which could be used for a wide variety of things such as reading news, work, do makeup, watch videos, or chat with family and friends. However, this growth of interest is also tied with a growth of development within the automotive and tech industries. Many large companies within these industries have started developing their own autonomous vehicles, especially cars, within the last 20 years. Some of these autonomous projects have matured to such a degree that they are likely going to be available to consumers within the near future. This means that there is currently a great interest in researching things related to autonomous vehicles.

For driverless cars to mature enough to be sold to consumers, there are at least two main problems that need to be solved. The first problem is how to get the car to its desired destination. This involves using GPS and maps in order to calculate a route to the destination, but it also includes things such as making the car capable of staying in the correct lane, managing intersections, or simply be able to stay on the road under difficult circumstances such as unmarked/dirt roads or snow. The second problem is how to do this safely without damaging the car, anything or anyone in its surroundings. This is because the car will never be alone on the street. There will be animals, pedestrians, cars and vast amount of other objects on the road, many of which might not follow traffic rules. As a crash could result in serious monetary loss or, worse, loss of life, the driverless car therefore needs to be extremely good at preventing accidents. It is therefore of very great importance for driverless cars to be able to detect and track the relative movement of surrounding objects. As previously mentioned, this is partially the case for safety reasons, because the car should try to avoid collisions with pedestrians and other cars. But it is also done in order to detect the outlines of the road and help the car navigate in its desired direction. To do this, objects of importance need to be detected, and then tracked, as the car is moving. To implement and test a new kind of tracking system is the goal of this project.

1.1 Aim

The aim of this project is to design and implement a new system that can use a single camera to track the positions, in 3 dimensions, of certain detected objects, for use in the Chalmers Formula Student Driverless car. This car is designed to follow a track which is outlined by cones of different colors. The old system simply used the screen coordinates to track cones, which had limitations in how accurately the car could detect the direction of the road. The old system was also lacking in its ability to remember and track previously detected cones. Therefore, there was a need for a more advanced system that could solve these problems, and that is the aim of this project.

This new system is built upon previous projects, and will make use of the outputs from these previous projects which detect cones through the use of *You only look once* (YOLO). These previous projects send information such as cone position and cone size, which will be used as inputs for this system.

The system starts by removing lens effects caused by the camera lens. Afterwards, a connection or equation needs to be found between the pixel locations and the real angles of the cones, relative to the direction of the car. The angles will then be used by the system to estimate the real positions of the cones through three different approaches. Finally, the estimate will be improved over time through the use of multiple estimates.

The completed system will be used in order to answer the following research questions:

- How reliably can depth perception be generated from a mono-camera system under motion, in the given race-car setting?
- How does the system compare to a commercially available stereo-camera system?

1.2 Limitations

The system will be limited to tracking stationary objects placed on a quite flat road. The objects will most likely only be cones (but other stationary objects should also be possible). The system will also be limited to use on a single known camera, but the system should be adaptable so that it will be possible to switch the camera with as little work as possible. The system will also use a simple vehicle model that is unique for the car, which includes things such as e.g. the distance between the wheel axles and the positioning of the camera on the car, in order to predict the movement of cones relative to the car. Additionally, the tracking of the cones will only continue until they leave the visible area, at which point they will be discarded.

2
Background

Before continuing, it's of importance to look at recent research in order to get an idea of the current state of the field.

2.1 Autonomous vehicle challenges

The software of modern autonomous vehicles is typically structured as a pipeline of individual software components, each with a different purpose. This pipeline begins with the sensor inputs, which are then linked to the processing of inputs, which in turn is then linked to the motor outputs. This means that any error upstream will propagate downstream, where this error then has the ability to cause many additional errors. It is therefore of great importance to consider the downstream effect of potential errors, and to carefully and thoroughly address these potential errors. However, downstream components should also be capable of handling the errors of upstream components. Interpretability of each software component, as well as inputs and outputs, help greatly in preventing and handling these potential errors [1].

Visual inputs have been shown to be extremely important, especially for vehicle and pedestrian detection, lane detection and drivable surface detection [2].

2.2 Depth perception

Depth perception can be achieved through a few different approaches, with some particularly noteworthy ones being: mono-camera in motion, stereo-camera, Single-image depth, and LIDAR.

2.2.1 Mono- versus stereo-camera

In order to determine the distance of objects from a camera feed, a couple of things are needed. The distance is usually determined through triangulation from two points, either from one mono-camera in two different positions, or from simply using a stereo-camera. It is also necessary to have a known distance, preferably the distance between the two camera positions or between the two cameras, in order to determine the distance of objects. The distance is however not necessary for the camera or object localization itself, but only for the scale of said localization. This means that it is generally comparatively more difficult to find proper scaling for the mono-camera case, even though both cases have been shown to, in most cases, be fully capable of generating accurate scaleless localization. The mono camera does have the benefit of generally being cheaper and requiring less powerful hardware than using a stereo-camera [3].

2.2.1.1 ZED camera

The commercial stereo camera ZED by Stereo Labs is advertised to have an error of 1% at the near range and 9% at the far range, likely under optimal circumstances. However it is mentioned that the accuracy can be affected by outliers on homogeneous and textureless surfaces [4].

2.2.2 Single-image depth

Recent research has shown that it possible to estimate the relative depth of an image. This is done through specifically designed algorithms and a *deep neural network* (DNN). The depth is done as a pixel-wise prediction, meaning it predicts the depth at every image pixel. This approach has been shown to work well for determining the order in which the objects are distanced (or in other words: if a certain object is closer than another object), but rather poor at determining the exact distance of objects. The approach also suffers from being scaleless, and requires something of a metric scale in order to get exact distances [5] [6].

DNNs trained for single-image depth have been shown to be poor at reliably determining the exact depth, in many cases being worse at this than humans. This is likely due to humans being able to understand the approximate size and geometry of objects which are known to them: e.g. roads and walls are flat, as well as cars, humans and floors of buildings having the same approximate sizes. Approaches that can use the approximate geometry of a known environment perform much better [6] [7].

2.2.3 LIDAR

LIDAR is the method of using light, usually in the form of a laser, in order to measure the distance of detected objects. This is done through measuring the time needed for the reflected light to return. This data is then generally processed by a DNN. LIDAR has been shown to, in most cases, be great for vision based detection, and a great tool for measuring the distance to nearby objects [8]. LIDAR does however have a number disadvantages: It has somewhat limited range and resolution, it can often fail on transparent or specular objects, has a comparatively high price, and is still quite lacking in DNN training datasets for environments of non-manmade structures [5]. LIDAR generally also lacks the ability of color vision, and therefore relies on being combined with an RGB-camera in order to get color vision. This can cause problems in combining the two systems. This is likely best solved by feeding the data from both into a DNN [9].

LIDAR has the added benefit of being able to be fused with camera data for training purposes. This is due to the usability of the reliable range measurements from the LIDAR to calibrate the camera systems [10].

2.3 Visual odometry

Visual odometry is the use of, usually multiple, visual inputs in order to find the position of this visual input relative to the positions of visually detected objects, and is usually done with the camera feed from one or several cameras in motion.

Visual odometry generally relies on the camera being in a mostly stationary environment, and unidentified moving objects can have a negative impact on performance [11]. The approach uses the positional change of objects between frames in order to determine the positions of camera, as well as the positions of nearby detected objects. This can done by applying a geometric approach to the surrounding and determining the positions through finding the set of parameters that minimizes the photometric error [12].

Visual odometry can also be achieved by feeding the visual input of successive image pairs through a DNN, an approach which has been shown to be very accurate and robust in certain situations [13]. But the use of DNNs may result in system failure for specific input patterns, and to accurately estimate the failure rate and failure conditions is non-trivial and can be extremely time-consuming [2].

2.3.1 Keypoint sampling

There are several different approaches to visual odometry. One such approach is keypoint sampling. This relies of some form of keypoint detector, and uses these keypoints for localization. This requires a keypoint detector that can reliably and accurately determine the positions of these keypoints, as errors from the keypoint detector could lead to severe localization errors. In addition to this, the use of keypoint sampling can become a problem in situations with none or an insufficient number of useful keypoints. A small number of keypoints can also lead to a decreased positional accuracy [12].

Keypoint sampling can however have great advantages in certain cases, such as when there already exists a detailed map of the surrounding environment. The use of keypoints sampling then allows for matching of these newly detected keypoints to keypoints in the pre-existing map. This leads to a much improved accuracy of self-localization [10].

2.3.2 pixel sampling

A different approach is to use pixel sampling. In this approach, sampling for intensity gradients, edges, and smooth intensity variations is used for localization. How the sampling is done varies, but there are two key parts to point out: The density of sampling and the distribution of said sampling. Dense sampling improves the level of detail of the detected objects, allowing for capturing of complete surface shapes, which can then be given semantic labels. Sparse sampling is only capable of capturing partial scene information, making them generally only useful for localisation [14]. However, a sparse sampling with even sampling distribution has been shown to be superior for camera localization [12]. Dense sampling also has much higher dimensionality, making it computationally costly, as opposed to sparse sampling [14] [15].

3
Method

3.1 Camera position calibrations

In order to get any useful information out of the camera feed, it is necessary to first find equations that can describe the relationship between the screen pixel positions and some form of real positions. This was done by first removing lens distortion and then finding corresponding real angles for each pixel position on the screen. The angles in themselves aren't able to determine the distance, but require some form of real distance to be known, such as the distance between two frames, the distance between two cameras, or the height at which the camera is positioned above the ground.

3.1.1 Remove lens distortions

The system starts by removing the barrel distortion caused by the lens. As the system is going to remove the barrel distortion of points, it is therefore necessary to find equations which can project these points from their incorrect distorted positions to where they would be with no distortion. The points are given as screen coordinates.

The first step to this was to get an object with clear checkerboard pattern on it. The checkerboard pattern forms vertical and horizontal lines which are parallel and have identical distances between each other. This can be used in order to observe the lens curvature at different points of the screen. However, the main reason for using this is that the pattern of parallel lines with identical distances between each other must be achieved for the correction of the distorted image. If this can be achieved with high precision for all screen coordinates, the found equation can be considered as acceptable. However, this approach can, perhaps, lead to an incorrect scaling factor on the screen coordinate distance from the centre of the lens. Because of to the way the screen coordinate to angle relation will be found in the next step, Section 3.1.2, this was not considered a problem, due to it having no effect on the end results. A visualisation of how a distorted checkerboard pattern can look is shown in Figure 3.1. A visualisation of how the correction of a distorted checkerboard pattern can look is shown in Figure 3.2.

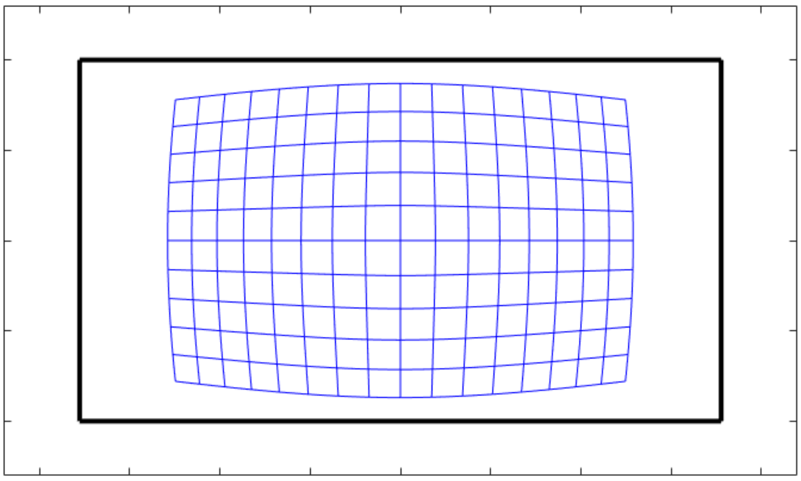

Figure 3.1: *A visualisation of a grid with an added barrel distortion. The grid is shown in blue. The thick black line represents the edge of the field of view of the camera.*

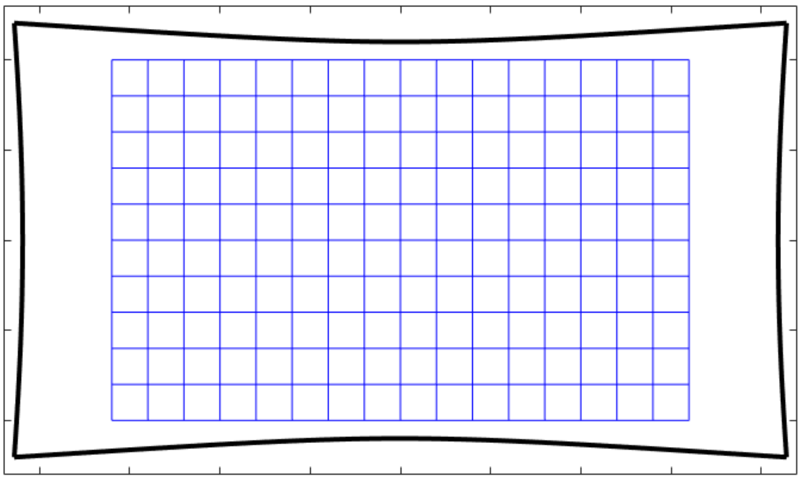

Figure 3.2: *A visualisation of a grid with a correction for an added barrel distortion. The grid is shown in blue. The thick black line represents the edge of the field of view of the camera. Note that the field of view has been stretched and no longer has a rectangular shape.*

The next step was to make these lines on the checkerboard pattern into screen coordinate points. This was done by placing said checkerboard pattern in front of the camera, and taking pictures of the video feed. A MATLAB script was implemented which could be used to create groupings of points, and then saved these into one file for each image. Each grouping of points corresponded to a sufficiently large number of points placed on each line of the checkerboard pattern. The points were manually placed on a highly zoomed-in image by clicking with consistent intervals along the line. MATLAB registered and saved each click.

For the next step, the files created in the previous step was loaded into another MATLAB script. This script was implemented to try different lens correction equations and the error for each equation. In order to do this, the script used each grouping of points, which, as previously mentioned, belong to a single line of the checkerboard pattern. Each grouping was first projected with a lens correction equation. Following this, a line would be fitted with the polyfit command in MATLAB, and the mean square error was calculated between the line fitting and the given points. This entire process was repeated for a large amount of parameters through doing a parameter sweep. The script would output the best parameter values for the given equation, as well as the errors for these parameter values. The best equation and parameter values were obtained by looking for the lowest error.

3.1.2 Finding pixel to angle relationship

The next step for the system was to find an equation which could accurately describe the relationship between the pixel positions and real angles for each detected object. The pixel position was the output resulting from Section 3.1.1. The angles of interest are the horizontal (sideways) and vertical (up/down) angles. These are the physical angles between the direction at the centre of the camera lens and the angles of each detected object.

In order to find the physical angles, the camera was placed facing directly at a physical plane. Five points were placed on top of the video feed, at known pixel positions. These five points were placed in a plus-sign shape, which was used to make sure the video and physical coordinate systems aligned. The physical plane was then marked in such a way that all markings perfectly aligned with the directions on the video feed. The distance between the camera and the origin of the coordinate system on the physical plane was measured. Following this, the plane was marked with a large amount of points, each with its physical coordinates written down next to its marked position. The exact positions for all the physical points were known, and this made it possible to easily find the physical angle for each point. After a sufficiently large amount of points were placed, a screenshot was taken.

Finally, an equation was needed to describe the relationship between the pixel positions and real angles for each detected object. The pixel positions were obtained simply by taking the pixel positions found on the screenshot. These pixel positions could be matched with the real angles for each of these points. This data was then input into MATLAB, where an equation could be found.

3.2 Simulation

In order to ease with implementation as well as to have the ability to easily verify that every part was working as intended, a vehicle simulation was implemented in MATLAB. This simulation was made to work in a manner which replicated the behaviors of the camera on the real car.

In order to run this simulation, it required to be given a map of the cones and a pre-determined camera path. For each iteration, it would start by moving the car along the pre-determined path. The distance moved was decided by the speed of the car as well as the frequency of the camera.

The next step was to calculate the positions of the cones relative to the position and direction of the camera. This was done in order to ease with the next step—calculating the camera pixel positions for each cone. Each pixel position was given as the horizontal and vertical angles, rather than integers. This was done in order to match the angles described in Section 3.1.2. However, before calculating the pixel positions, a temporary point was added above each of the cones, indicating where the top of the cone was located. The height of these points were decided by a parameter which set the cone height. When the pixel positions of these temporary points were calculated, they gave the information of the cone sizes for cones at different distances. This information was then used to remove cones that were covered by other cones.

The remaining cones then had noise added to them. This noise was added as a normal distribution rounded to an integer multiplied by the pixel size of the camera, and was done in order to replicate the way YOLO marks the position of a detected cone. A vertical sinusoidal noise was also added in order to replicate the shaking behavior of a real camera.

Finally, cones that were outside the field of view of the camera were removed. This field of view was decided by the parameter for the horizontal angle, a parameter for the height-to-width ratio of the camera, and a parameter for the angle at which the camera was mounted.

3.3 Car test

The tests on the real car were rather limited, and consisted of thoroughly going through recorded videos with the intent of finding and imitating observed behavior, as well as finding suitable values of parameter to use within the simulation.

3.4 Approach one—Triangulation

In order to estimate the distance of each cone, equations needed to be found that could achieve this. One way of doing this was through triangulation. This triangulation was done between each two sequential frames. Therefore, a new estimate was obtained for each new frame. In order to do triangulation, 4 different kinds of data were required. The first of these is the horizontal (sideways) angles of all cones in the first frame. The second is the horizontal angles of all cones in the second frame. The third is the distance the camera travelled between the two frames. The fourth and final is the camera rotation that occurred between two frames.

3.4.1 Cone tracking

In order do this triangulation, it was necessary to have a way of tracking the movement of each cone between two frames. This is because it is necessary to make sure that the same cone is used in each frame, in order for the distance estimate to work. As new cones could appear or old ones disappear, as well as the pixel position movement of cones close to the camera being much greater than the pixel position movement of cones far away from the camera, this meant that it was required to have an adaptive model that could handle these challenges. This was achieved through attempting to predict the positions of all cones in the next frame, by using the fact that the height as well as the approximate angle of the camera were known. This information could be used in order to project the cone positions from the pixel plane to the ground plane, then add the movement and rotation of the car between the two frames to the cone positions, and lastly projecting back from the ground plane to the pixel plane. The motion was simplified as being only in the forward direction. The prediction was therefore obtained through:

$$\alpha_{pred} = arctan(-\frac{y}{x}) \tag{3.1}$$

$$\beta_{pred} = arctan(\frac{-H}{\sqrt{x^2+y^2}}) + \theta \tag{3.2}$$

where x and y are given by:

$$x = \frac{H}{tan(\theta-\beta)\sqrt{1+tan^2(\alpha)}} - v\Delta t \tag{3.3}$$

$$y = \frac{-Htan(\alpha)}{tan(\theta-\beta)\sqrt{1+tan^2(\alpha)}} \tag{3.4}$$

for which H is the height of the camera above the ground, θ is the approximate vertical angle of the camera in the downwards direction, and v is the car velocity. α represents the angle in the rightwards direction, and β the angle in the upwards direction on the screen. These predicted cone positions from the last frame would then be compared to the cone pixel positions of the current frame. If the predicted and current positions were within a certain pixel distance of each other, they were considered to be the same cone, and a distance estimate could be obtained for this cone. The maximum allowed pixel distance between the prediction and current cone positions was decided by a parameter. If there was a prediction that did not match a current cone position, it was assumed that a cone had disappeared, and the prediction was therefore discarded. If there was a current cone position that did not match a predicted cone position, it was assumed that a new cone had appeared.

3.4.2 Triangulation

After it was known which cone positions that belonged together in the two consecutive frames, it was now possible to do the triangulation. This triangulation was done by taking the angles from the two frames and adding the rotation between the frames to the latter frame. By taking the velocity and multiplying it with the time between two frames, it was possible to get an estimate of the distance travelled. By then using the Law of sines, and for each cone inputting the angles and distance travelled, it was possible to calculate the distance between each cone and the camera in the latest frame. A visualisation of the law of sines can be seen in Figure 3.3. This position of the cone was obtained from:

$$x = dcos(\alpha_n) \tag{3.5}$$

$$y = -dsin(\alpha_n) \tag{3.6}$$

where d, the distance of a cone, was obtained through:

$$d = \frac{v\Delta t sin(\alpha_{n-1})}{sin(\alpha_n - \alpha_{n-1} - \omega\Delta t)} \tag{3.7}$$

where ω is the rotational velocity of the car.

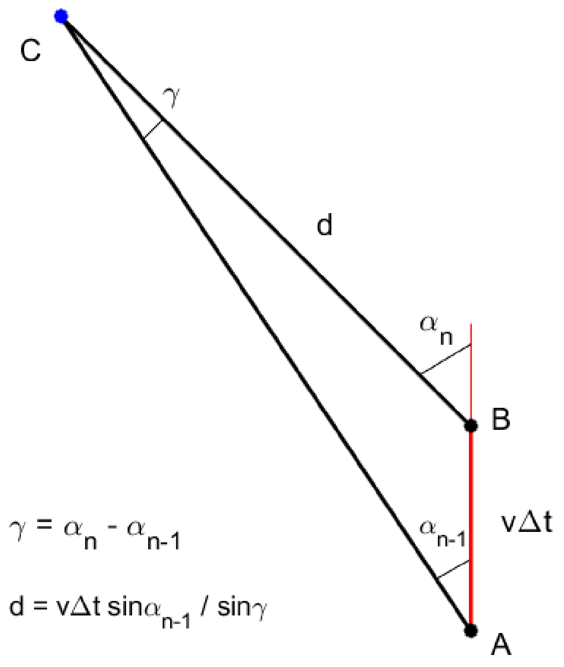
C
γ
d
α_n
B
$v\Delta t$
α_{n-1}
A
$\gamma = \alpha_n - \alpha_{n-1}$
$d = v\Delta t \sin\alpha_{n-1} / \sin\gamma$

Figure 3.3: *A visualisation of a rewritten form of the law of sines. A is the position of the car in the previous frame. B is the position of the car in the current frame. C is the position of a cone. $v\Delta t$ is the distance travelled between the two car positions. d is the distance between the car and the cone at the time of the latest frame. α_n and α_{n-1} represent the angles of the cone in the current and previous frames respectively. The rotation between two frames is not shown.*

In order to prevent noise from occasionally resulting in a close to infinitely large cone distance or, in the case of a tiny negative angle, an infinitely large negative distance, a maximum cone distance parameter was used. Any distance below zero or larger than this parameter was set to the value of this parameter. The parameter should be set approximately to that of the maximum cone detection distance.

3.4.3 Improving range estimate

Due to the possibility of noise leading to potentially large range estimation errors, it is of importance to use several consecutive range estimations, for each cone, in order to improve the distance estimations. The way this was done was through the use of a modified Kalman-filter. This filter used, for each new frame, the new cone distance estimations and combined these with the predictions calculated from previous estimations. This lead to an improved range estimation accuracy for every consecutive frame that a cone has been detected. Some alteration had to be done in order to account for the fact that the noise level very quickly decreased as a cone moved closer to the camera.

3.5 Approach two—Plane projection

The method of determining the distance was simply to do a projection from the screen plane onto the ground plane. In order to do this, two pieces of information are required: the height of the camera, which was known, and the downwards angle of the camera, which had to be calculated. This approach is much better at preserving the shape of the track compared to an approach which does an individual distance estimation of each cone. This is demonstrated in Figure 3.4.

3.5.1 Cone tracking

The cone tracking was done using the same procedure described in Section 3.4.1. First doing a prediction of the pixel movement from the positions in the previous frame. The tracking is then done by pairing the most suitable predictions with the pixel positions of the latest frame. The suitability was done through, for each prediction, finding the closest point and then verifying that the prediction is also the closest prediction of that point. If the distance between the two was above a set threshold, the pair was discarded.

3.5.2 Determining velocity and vertical camera angle

As previously mentioned, the vertical camera angle is used in order to determine the distances of all cones. It is therefore of great importance to get an accurate estimate of the vertical camera angle. Estimating the velocity of the camera is also useful for tracking the cones. In addition to these two variables, a third variable must be introduced. This third variable, ϕ, is the difference in vertical angle between two frames, as the shaking of the camera means the camera angle can not be assumed to be fixed.

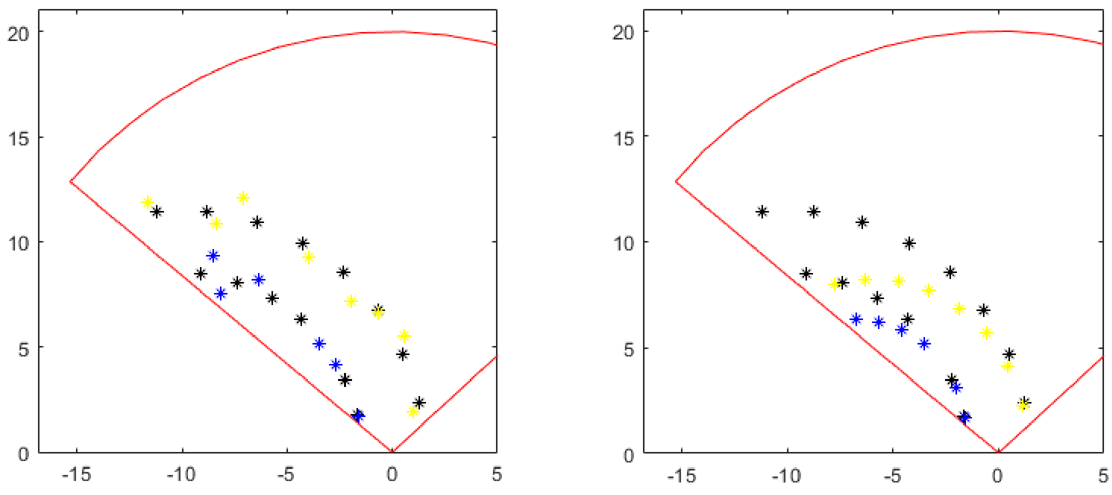

Figure 3.4: *Two figures which demonstrate the differences between a point-based distance approach (left) versus the plane projection approach (right). The black points represent the real cone positions. The blue and yellow points represent the cone position estimates. The outlined red area represents the area where the camera is capable of detecting cones. The image on the right has intentionally been given an error approximately twice as large as the image on the left, but still does a better job at preserving the shape of the track.*

The velocity, vertical camera angle and difference in vertical angle were found through finding the values that resulted in the smallest total error between prediction and real movement of all cone positions. The prediction of one cone was given by:

$$\beta_{pred} = arctan(\frac{-H}{\sqrt{x^2 + y^2}}) + \theta_n + \phi \tag{3.8}$$

where x and y are given by:

$$x = \frac{H}{tan(\theta_n - \beta)\sqrt{1 + tan^2(\alpha)}} - v\Delta t \tag{3.9}$$

$$y = \frac{-Htan(\alpha)}{tan(\theta_n - \beta)\sqrt{1 + tan^2(\alpha)}} \tag{3.10}$$

This optimal values were found through doing a parameter sweep with short step length. The vertical angle was restricted to an upper limit such that the furthest point would not exceed a maximum distance, set by a parameter.

3.5.3 Calculating distance

Finally, the estimated vertical camera angle was used to project the points onto the ground plane and determine their distances. This was done in the following way:

$$x = \frac{H}{tan(\theta - \beta)\sqrt{1 + tan^2(\alpha)}} \tag{3.11}$$

$$y = \frac{-Htan(\alpha)}{tan(\theta - \beta)\sqrt{1 + tan^2(\alpha)}} \tag{3.12}$$

3.6 Approach three—Distance from cone height

The idea of this approach is to use the fact that the height at which the camera is placed, H, as well as the height of the cones, h, are both known. This can be combined with the fact that the angle size, β, of the cones directly results from the pixel heights of the cones, in order to determine the distance, d, of each cone individually. This was done through finding the distance which minimized the error between the right and left sides of the following equation:

$$\beta = arctan(\frac{h - H}{d}) - arctan(\frac{-H}{d}) \tag{3.13}$$

The approximate angle of the cone, α, was then found through:

$$\alpha = arctan(\frac{-H}{d}) \tag{3.14}$$

This was used to calculate an estimate of the angle of the camera, for each individual cone. The final estimate was obtained by taking the median of these previous estimates. The median was chosen over the average in order to reduce the effect of outliers. This final estimate was then used to project the screen positions onto the ground plane, as described in Section 3.5.3.

4 Results

4.1 Camera calibrations

The camera calibrations worked well, showing only minor errors close to the four corners of the screen. Removing the camera distortions does however somewhat distort the bounding box placed by the YOLO-microservice, as these bounding boxes were placed on the distorted image. An example of a bounding box placed on the distorted image can be seen in Figure 4.1

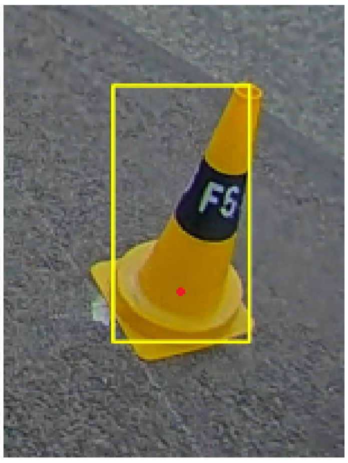

Figure 4.1: *A photo showing the bounding box placed by the YOLO-microservice on a distorted image. Removing the distortion of the image distorts the bounding box. The red dot marks the approximate centre of the bottom of the cone, which is the point of interest.*

4.2 Car tests

The recorded videos showed a cone noise with an amplitude of approximately 2 pixels.

The camera shaking consists of small random camera shakes with an amplitude of up to approximately 0.5 degrees, as well as a larger sinusoidal shaking with an amplitude of up to approximately 1 degrees and a periodic time of approximately 0.5 seconds.

4.3 Approach one—Triangulation

The results of varying the rotational noise at different camera angles can be seen in Figures 4.2 and 4.3. The triangulation approach had the benefit of requiring very little computational power.

The approach of using multiple estimates to improve the position estimate over time was shown to work well, but only in circumstances where the velocity of the car was somewhat well known. The computational power required for this was very low.

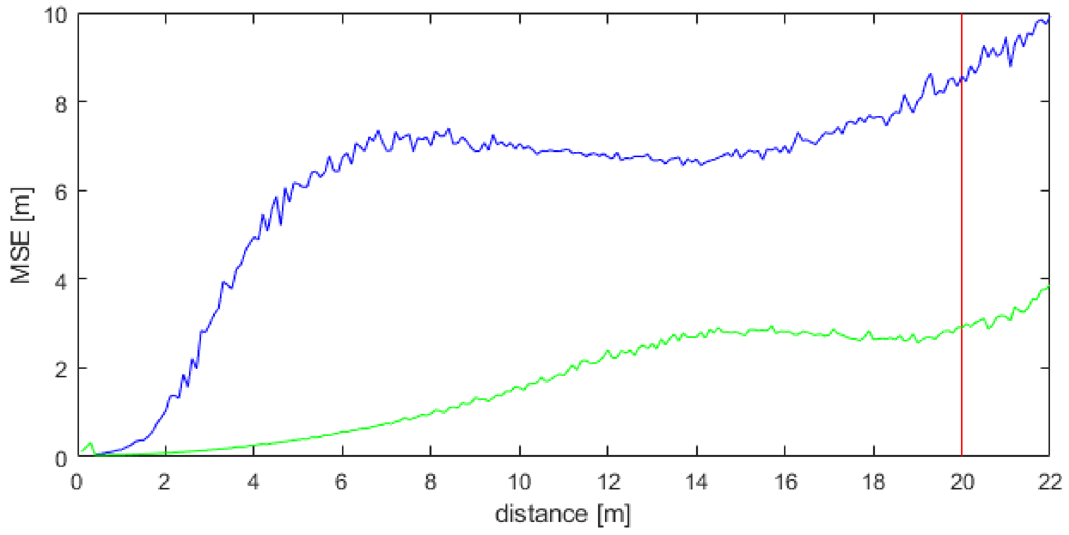

Figure 4.2: *The mean square error (MSE) calculated using different variables. All cones were placed at a* 10 *degree angle from the forward direction. Both green and blue have a speed noise amplitude of* 10% *and a pixel noise of a normal distribution with a standard deviation of* 2 *pixels. Green has no rotational noise. Blue has a rotational noise of a normal distribution with a standard deviation of* 0.2 *degrees. The red line is the upper limit of the cone detection range. Each data point is obtained by taking the average of* 1000 *cones.*

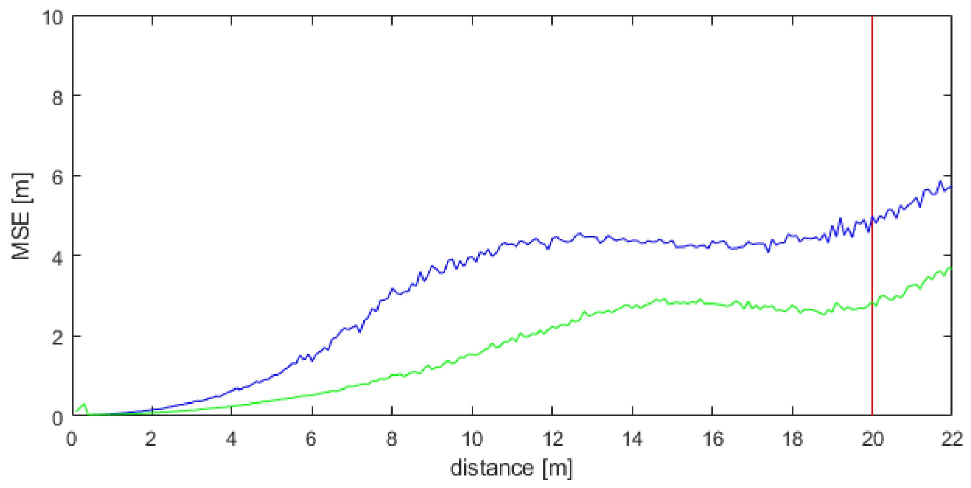

Figure 4.3: *The mean square error (MSE) calculated using different variables. All cones were placed at a* 40 *degree angle from the forward direction. Both green and blue have a speed noise amplitude of* 10% *and a pixel noise of a normal distribution with a standard deviation of* 2 *pixels. Green has no rotational noise. Blue has a rotational noise of a normal distribution with a standard deviation of* 0.2 *degrees. The red line is the upper limit of the cone detection range. Each data point is obtained by taking the average of* 1000 *cones.*

4.4 Approach two—Plane projection

The cone tracking worked close to flawlessly for all cones in all tested circumstances when a camera frequency of 60 Hz was used. The computational power required for tracking was very low.

When running the system on a circular path with noiseless cone points and no camera shaking, the cone distance estimates have an average error of 7% for car velocities of less than 1 meters per second and an average error of 0.8% when the car is moving at a velocity of exactly 10 meters per second.

When running the system on a circular path with a pixel noise of a normal distribution with a standard deviation of 2 pixels, the cone distance estimates have an average error of 12% for all tested car velocities. The errors are concentrated mainly in approximately 20% of frames. One such frame is shown in Figure 4.4.

The quality of the results of the plane projection approach was somewhat reliant on the step length used of the parameter sweep. A short step length was required to achieve the results above, resulting in the need for a large amount of computational power needed to run at the desired frequency.

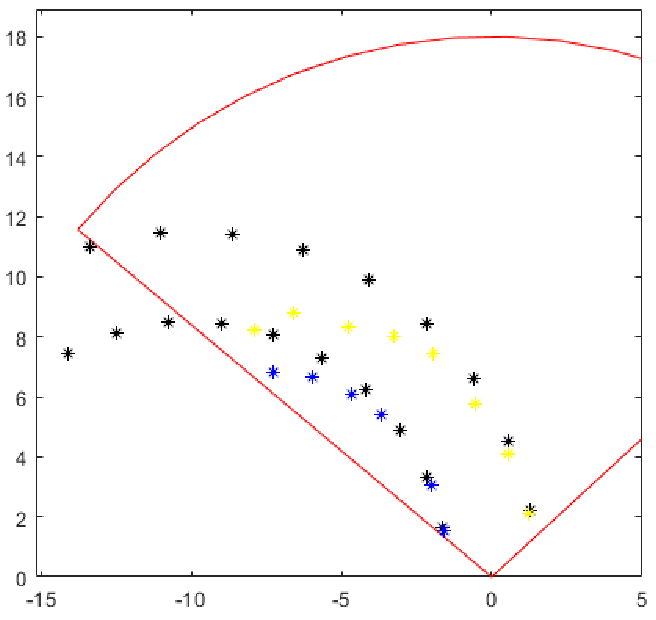

Figure 4.4: *A frame showing an example of a large error that occasionally occurrs when using the plane projection approach. The red area represents the field of view where cone detections are possible. The black points are the real cone positions. The blue and yellow points are the position estimates given by the system.*

4.5 Approach three—Distance from cone height

When running the system on a circular path with a pixel noise of a normal distribution with a standard deviation of 2 pixels, height error multiplier of either 0.95 or 1.05, and a height noise of a normal distribution with a standard deviation of 0.05, the cone distance estimates have an average error of 6%. This approach required very little computational power.

5

Discussion

5.1 Camera calibrations

While the calibrations worked well for points, there were some potential problems spotted in the way the YOLO-microservice places its bounding boxes around detected cones. This was due to these bounding boxes being placed on a distorted image, as seen in Figure 4.1. The point of interest is the point at the centre at the very bottom of the cone, and the YOLO-microservice is not currently capable at finding this exact point. Additionally, the bounding box will no longer retain its shape after removing distortions. This has a negative impact on the information of the bounding box in the undistorted image. Generally speaking, the top of the bounding box is great at finding where the top of the cone is, and the bottom of the bounding box is better at determining the sideways alignment of the cone.

5.2 Approach one—Triangulation

Attempting to triangulate the distances showed quite poor results, as shown in Figures 4.2 and 4.3. When using realistic parameters for pixel noise and rotational noise, the error could in some cases become larger than 100%. The results showed that these errors mainly came from inaccuracies in the estimation of car rotation. This means that triangulation using two consecutive frames of a camera in motion is not feasible when using external sensors to determine the rotation, or through estimating the rotation of the car by using the movement of cone positions (except maybe with a very accurate car velocity estimate).

Additionally, the cone errors were much larger for cones positioned at smaller horizontal angles from the centre of the screen. The reason for this is that cones in this area experienced a much smaller sideways movement as a result of the forward movement of the car. This meant that the angle necessary for triangulation became disappearingly small in comparison to the noise amplitude, resulting in nearly a complete depth blindness at angles of less than 10 degrees left or right from the centre of the screen. This is a problem that is unique to this particular application, as the car movements only allow the camera to move forward or rotate, instead of allowing for a sideways movement which would be required for depth vision. A stereo camera instead has its cameras placed at a known sideways distance away from each other, and would therefore not experience the problem of having a field with depth blindness.

5.3 Approach two—Plane projection

The cone tracking worked extremely well, and this seems to be almost irregardless of the way the position prediction was done. This is due to that the maximum movement a cone between two frames is generally smaller than the distance between the cone and its closest neighbour. This might however no longer be the case if a lower camera frequency is used. However, almost any well designed prediction should work even for frequencies of around 15 Hz.

The plane projection approach worked extremely well for circumstances with no noise. However, it turned out to be rather unstable for realistic noise amplitudes. There seems to be certain combinations of the three variables that results in very low errors. The realistic noise will frequently be large enough that the lowest error will be found on a random erroneous low error combination, often resulting in large errors for all three variables. The likelihood of accurately estimating the variables does however seem to greatly increase if the problem can be reduced to finding the optimal values of only two variables. This could for example be done by installing a velocity sensor in the car.

The causes of the instability is assumed to be because of a combination of a poor spread of data points, too few data points, and the YOLO-microservice giving rather large noise amplitudes. The last two being somewhat similar: the noise either needs to be reduced, or more points are needed so that the noise of these cones are more likely to cancel each other out. The spread of the cones has a somewhat different effect. As mentioned earlier, the cones close to the centre of the camera experience very little movement as a result of the movement of the car. This means that the cones in this area almost only move as a result of the camera rotation. Cones close to the two bottom corners of the screen are the opposite, and the cone movement there is usually mainly from the movement of the car. Having cone positions at the two top corners could greatly improve the accuracy of the velocity and the noisy vertical movement of the camera. It is however not possible to place additional cones at will, and the cones will realistically never be found in the sky. But, if possible, it could be useful to add additional keypoints which can be tracked and used to improve the estimates. Additionally, as the camera is fixed on the car, the camera is unable to do any depth estimates while the car is stationary. This is quite a drawback, as it would otherwise be possible to create well calibrated reference points on which to build the map of surrounding objects.

The approach of finding the vertical angle of the camera and using this to project all cone positions onto the ground plane did seem to give good results. It mostly preserves the shape of the track even in cases with a large vertical angle error. There was usually only a problem of preserving the shape of the track if the cones were estimated to be too far away, but having a set maximum distance prevented this from being a problem.

5.4 Approach three—Distance from cone height

The approach of estimating the distance from the cone height shows some promise inside the simulation, as it gave the best results of all tested approaches. It was the only approach which was able to give cone estimates from a single frame, making it much simpler and, in some ways, more reliable. However, as mentioned in Section 5.1, there are some problems with using the bounding box of the cones. These problems could potentially mean that the used equations are not valid, especially for cones closer to the edges of the camera, as these have greater distortion. Additionally, as shown in Figure 4.1, the size of the bounding box is actually larger than the size would be using only the real height of the cone. These problems could perhaps be remedied by instead relying on real cone pixel height data gathered from placing cones at different distances, combined with an interpolation in between these data points. The pixel height would be compared to this interpolation between data points in order to find the distance of the cone. This approach can handle cones of any known size, which can easily be adjusted by changing a single parameter per cone type/color. It does however have the problem of being unable to handle cones of unknown heights.

6 Conclusion

The cone tracking was found to work close to perfect. This means that it's possible to reliably track each cone individually as they move across the screen. It was also found that having a reliable velocity estimate, such as from a well tuned speedometer, can be used to accurately estimate the rotation and predict the cone movements between frames. All of these things combined can be used in order to use multiple cone distance measurement (one per frame), in order improve the distance estimates of the cones over time. These very accurate cone positions could also be used for detailed mapping when running the same track multiple times. In addition to this, the cone tracking, rotation estimate, and cone movement predictions were all computationally inexpensive. This is great for potential use in a self driving electric car, where the electrical and computational power can be limited.

The triangulation approach was shown to be unsuitable for mono-camera use, due to the field of depth-blindness at the centre of the camera combined with poor depth estimates in general.

The plane projection approach was shown to be unreliable for realistic noise amplitudes, especially at slow speeds. This approach was also very computationally expensive. Being supplied an accurate real-time velocity estimate would greatly reduce the computational power needed while simultaneously improving the results of the distance estimates. The degree of this improvement is however unknown.

The distance from cone height approach was simple and fast. Some of the problems with this approach could possibly be remedied by either better handling of bounding box distortions or using a camera with little to no camera distortion. Fixing these things would however not decrease the distance estimate errors found in the simulation. But, if possible, using estimates from several frames would decrease the distance estimate errors. Being able to determine the cone distances from a single frame made this approach much easier to implement and also meant that it did not have some of the reliability problems of the other approaches.

Overall, the depth perception results were likely not sufficiently accurate and reliable, as all approaches were shown to have some flaw. The distance from cone height approach showed the best results, but still had an error 1.4 times that which can be expected from a commercially available stereo-camera system of good quality. It could therefore be that it is not possible to get an accurate and reliable enough depth vision with the available cone data for the mono-camera approach.

6.1 Further work

The plane projection approach can be improved through supplying it with an accurate real-time velocity estimate. Additionally, there may be other methods more suitable than optimizing through using a parameter sweep. Some form of a DNN may be the most suitable here, due to its ability to learn and adapt. A DNN can be supplied information about the current and previous states, and could possibly use this to filter out the combinations that resulted in large errors. This could result in a more reliable distance estimate. It is therefore of interest to implement and test this kind of approach.

The distance from cone height approach showed some promise, but needs to be looked into further to fully determine how well suited it is. Ideally this would be done with a camera with little to no lens distortion. Cone data needs to be gathered to have a mapping between cone height and cone distance. The finished system then needs to be tested on real data.

Bibliography

[1] R. McAllister, Y. Gal, A. Kendall, M. van der Wilk, A. Shah, R. Cipolla, and A. Weller, “Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning,” *International Joint Conferences on Artificial Intelligence*, 2017.

[2] W. Shi, M. B. Alawieh, X. Li, and H. Yu, “Algorithm and hardware implementation for visual perception system inautonomous vehicle: A survey,” *INTEGRATION the VLSI journal 59*, 2017.

[3] R. Mur-Artal and J. D. Tardós, “Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras,” *IEEE Transactions on Robotics*, 2016.

[4] How does the zed work? [Online]. Available: https://support.stereolabs.com/hc/en-us/articles/206953039-How-does-the-ZED-work-

[5] W. Chen, Z. Fu, D. Yang, and J. Deng, “Single-image depth perception in the wild,” *30th Conference on Neural Information Processing Systems*, 2016.

[6] C. Godard, O. M. Aodha, R. Clark, M. Firman, and G. Brostow, “Digging into self-supervised monocular depth estimation,” *IEEE*, 2019.

[7] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, “Unsupervised learning of depth and ego-motion from video,” *CVPR 2017*, 2017.

[8] B. Li, T. Zhang, and T. Xia. (2016) Vehicle detection from 3d lidar using fully convolutional network. [Online]. Available: https://arxiv.org/abs/1608.07916

[9] A. Asvadi, L. Garrote, C. Premebida, P. Peixoto, and U. J. Nunes, “Multimodal vehicled etection: fusing 3d-lidar and color camera data,” *Pattern Recognition Letters 115*, 2018.

[10] J. V. Brummelen, M. O’Brien, D. Gruyer, and H. Najjaran, “Autonomous vehicle perception: The technology of today and tomorrow,” *Transportation Research Part C: Emerging Technologies*, 2018.

[11] J.-W. Bian, Z. Li, N. Wang, and H. Zhan, “Unsupervised scale-consistent depth and ego-motionlearning from monocular video,” *33rd Conference on Neural Information Processing Systems*, 2019.

[12] J. Engel, V. Koltun, and D. Cremers. (2016) Direct sparse odometry. [Online]. Available: https://arxiv.org/abs/1607.02565

[13] B. Ummenhofer and H. Zhou. (2017) Demon: Depth and motion network for learning monocular stereo. [Online]. Available: https://arxiv.org/abs/1612.02401

[14] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, “Codeslam — learning a compact, optimisable representation for densevisual slam,” *IEEE*, 2018.

[15] Z. Yin and J. Shi, “Geonet: Unsupervised learning of dense depth, optical flow and camera pose,” *IEEE*, 2018.