CHALMERS
UNIVERSITY OF TECHNOLOGY

Real time object localization based on computer vision

Cone detection for perception module of a racing car for Formula student driverless

Maksims Svecovs
Felix Hörnschemeyer

Department of Mechanics and Maritime Since
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Real time object localization based on computer vision

Cone detection for perception module of a racing car for Formula student driverless

Maksims Svecovs, Complex Adaptive Systems (MPCAS)
Felix Hörnschemeyer, Embedded Electronic System Design (MPEES)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Since
Chalmers University of Technology
Gothenburg, Sweden 2020

Real time object localization based on computervision
Cone detection for perception module of a racing car for Formula student driverless
Maksims Svecovs
Felix Hörnschemeyer

Examiner: Ola Benderius

Master's Thesis 2020:53
Department of Mechanics and Maritime Since
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover illustration: Chalmers University of Technology

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

Using a Formula student car to race autonomously and find the right path has been around since the introduction of this class in the Formula student competition in 2017. This project work attempts this problem using a pure computer vision approach by utilizing a stereo vision camera only and running a convolutional neural network to localize the object in a field of view. To make sure the approach works at a stable high frequency, the detection of the cone-objects which are marking the path is run on a GPU. As the output of the system, the two dimensional coordinates in a local frame are calculated from each detection made in each frame, using merged approaches of *triangulation* and *distance by size* in the frame. The result is a solution with a sufficiently precise detection up to a distance around 12m, running at a frequency of 25Hz with more than 12 detected objects per frame, the system proved to be well enough designed to provide data for path planning module.

Keywords: Convolutional neural network, GPU, TinyYOLOv3, Formula student, autonomous driving, object detection.

Acknowledgement

We would like to thank everyone that was involved in the process of developing, testing and verifying our project. We want to thank our examiner Ola Benderius and the whole team of the Revere lab at Chalmers University of Technology for their help and guidance throughout our project and thesis.

Maksims Svecovs and Felix Hörnschemeyer, Gothenburg, July 2020

Contents

List of Figures

List of Tables

1
Introduction

While computer vision as a field of engineering has been around since the late 60s, it has mainlz involved deterministic algorithms that revolved around feature detection. In case of lane detection, since lane markings are a line feature, traditional methods such as edge detection, template matching, colour transformations, and other techniques were used.
These methods had problems with generalization due to variations of lane marking appearances, road lane types and weather conditions. To address this, the data-driven methods based on machine learning started to arise. Data-driven methods allows for variety in both lane-markings and conditions. The variety is achieved by a careful construction of the dataset, that is, the more dispersed the data is, the more variable the algorithm can become. During the last years, deep learning algorithms advanced to become the most popular ones because of the appearance of affordable GPUs, which are required to make algorithms training viable time-wise [2][3][4][5].

1.1 Formula student

Formula student is a student engineering and design competition event that was growing from its start in 1998 in the UK, with a *combustion* class in which teams could compete against each other, over the additional class of *electrical* vehicles from 2010 in Germany until the introduction of the *driverless* vehicle class in 2017 (also first in Germany, with others following). The main goal of this event is to design and build a race car according to specified rules (formulas). An exclusion from that is given in the *driverless* class. Here, a car built in the previous years can be taken to be transformed into an autonomous, driverless race car. The reason for taking a car from a previous year is, to give the teams more time and possibilities while developing an autonomous system for the car, since the hardware is already existing. Since a lot of teams advanced from that in the last three years of the class' existence, the driverless class will be included into the two other classes (*combustion* and *electrical*). From the season of 2021, at least the dynamic *acceleration* event has to be completed without a driver.

	CV & EV	DV
Static Events:		
Business Plan Presentation	75 points	75 points
Cost and Manufacturing	100 points	100 points
Engineering Design	150 points	300 points
Dynamic Events:		
Skid Pad	75 points	75 points
Acceleration	75 points	75 points
Autocross	100 points	100 points
Endurance	325 points	-
Efficiency	100 points	75 points
Trackdrive	-	200 points
Overall	1000 points	1000 points

Figure 1.1: Point distribution in Formula student Germany, FSEast and FSCzech.

Figure 1.1 shows the point distribution for the three different classes during a 2019 Formula student competition. As it can be seen, a competition is divided into three static events and five dynamic events. The point distribution for the DV class however shows a clear division of nearly 50% between the two different classes of events. Therefore, it is as important that the team can defend its design choices in the *engineering design event* and justify the costs of the car, as take part in the dynamic events and achieve good results. To be authorized to drive in dynamic events, the car needs to pass a safety inspection, the so called *scruteenering*. During these inspections, the car is checked to be rules compliant and safe.

1.2 Aim

The aim of this project was to design and implement a computer vision algorithm for object detection for a specific scenario of Formula student driverless (FSD). Since the lanes in the competition are formed by cones, the traditional lane-detection approaches are not applicable here, since they rely on the distinct lane-shaped markings. Therefore, it was crucial to design an algorithm that focuses on particular specifics of the application such as high-speed driving and cone-based lane markings. Moreover, the weak point of traditional approaches, that is weather and lightning conditions, persists. Because of that, the aim was to use a deep learning approach.

Therefore, the following points were needed to be fulfilled by the system:

- Firstly, it needs to implement camera calibrations and pre-processing procedures for object-detection algorithms to be accurate and effective.
- Secondly, it should aim to design the algorithm that focuses on specifics of the application such as high-speed driving and cone-based lane markings. It should

also consider available hardware resources such as particular high-frequency camera and on-board GPU. The designed algorithm should aim to be robust to weather and light conditions changing.

- Thirdly, the designed algorithm needs to be implemented to be run as a microservice within the existing architecture with a support of parallel computations. For robust aimpoint calculation based on the algorithm output the aim was to achieve the processing speed of 50 frames per second with minimal requirement of 20 frames per second.

As an additional goal, the additional LiDAR system should be removed by utilizing a purly camera based approach.

1.3 Project description

This section gives an overview of the sensors and modules used as well as the dataflow and communication between these modules. As the sensor input to the system, a ZED stereo-vision camera with an adjustable frame rate and resolution is used. The API provided by the manufacturer is used to get the frame with depth information for each pixel. This frame is then processed by the CNN, which classifies and localizes all the cones in the frame and outputs a set of bounding boxes with confidence level for each detection. Finally, the so called *birdview* module processes the data further into two-dimensional coordinates which are sent out to the path calculation module. Since the task of processing 50 frames per second is computationally demanding, a GPU is utilized as the computing unit. An overview of the system can be found in Figure 1.2.

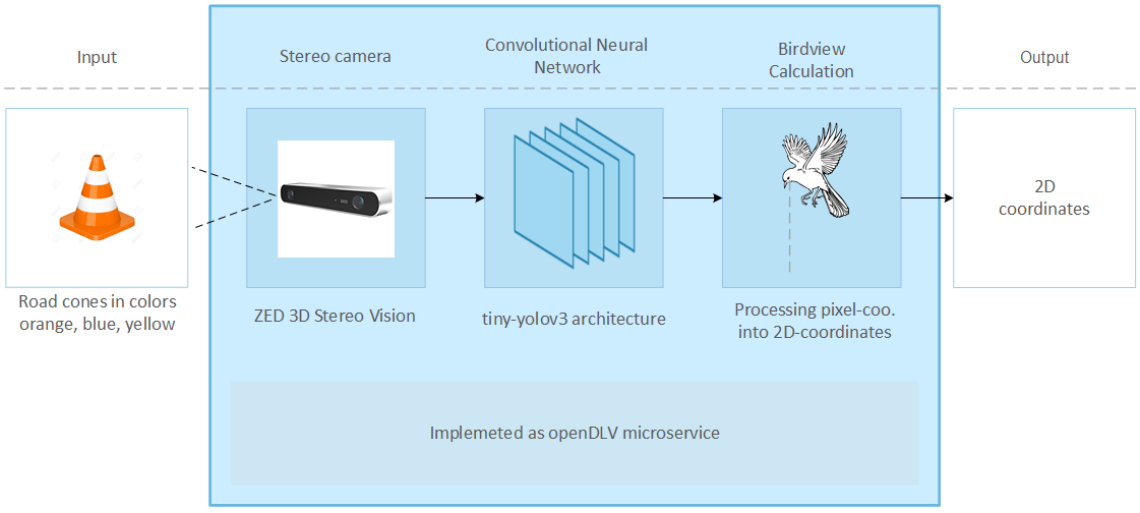


Figure 1.2: System overview with all modules.

1.3.1 Deep neural network

The neural network architecture that was aimed to be used in this project is a variation around convolutional neural networks (CNNs). This class of deep neural networks is suited well for image processing, since it aims to investigate regions of the images (by applying so called filters) in opposed to every pixel investigation. The filters are adjusted during the trainings, which reduces the pre-processing required for the data. Advantage of using the filters is the reduced number of parameters that are required to train, improving the performance as a result. Focusing on the performance and independence of prior knowledge makes CNNs the logical choice for image and video recognition. However, a regular CNN performance is usually not good enough for live video recognition. To address this problem various attempts to improve the architecture were made in forms of Fast-CNN, R-CNN, Fast-R-CNN, and YOLO.

Because CNNs fall in realm of supervised machine learning algorithms, they require an annotated dataset. That is, a collection of video frames and corresponding hand-made annotations, which ought to be used as a ground truth during the learning process. Annotated data collection is one of the biggest challenges for machine learning, since it is a very labour-intensive process. To address this problem, it was decided to utilize a transfer learning by training the network on a public available dataset that would potentially have features similar to the cones [6]. The weights of the network will then be initialized to the trained values in order to converge faster on a smaller cone-dataset.

1.3.2 GPU implementation

By nature, deep learning algorithms require simple operations performed over the matrix of data. This means that bandwidth-optimized computations are superior to latency-optimized computations. Moreover, the simple operation will be repeatedly performed on the entire matrix, which means a great potential for parallel computations. Both advantages are achievable by using GPUs which are optimized exactly for bandwidth and floating-point matrix multiplication.

Being one of the leading GPU manufacturers, Nvidia provides a great toolkit for enabling parallel computations on their cards. The CUDA toolkit makes it easy to control memory transfer between GPU memory and RAM and enables parallel computations by giving access to dedicated threads and cores inside the graphical card. Most of the machine learning platforms like Tensorflow and Pytorch are supporting the CUDA toolkit and utilize the GPU capabilities in built-in functionality. Having a Nvidia video card, the aim was to utilize these capabilities and build a network based on the existing library like Tensorflow or Pytorch.

1.4 Limitations

The project is mainly limited by two factors: available hardware and data. First, the existing camera has specifications that must be worked with. Also, the installed GPU is a consumer-level GPU with limited compute-capabilities in terms of deep learning applications. Ideally, a dedicated set-up to train the network could be used and the resulting trained weights would be transferred to the target set-up for inference afterwards.
Then, most machine learning algorithms have a demand for huge data sets to be trained successfully. Deep learning algorithms are no exception, in fact, they are highly dependent on the amount of data. The task of detecting cones acquire specific datasets. Datasets are shared by some teams, but the specific way of every annotation strategy makes them unusable to a known extent. Since there is no dataset available it was inevitable to do that manually, in a very time-consuming process.

1.5 Research questions

The following research questions are investigated in this thesis:

- Is it possible to build a camera-only reliable perception system for the limited scenario of Formula student driverless? Assuming the reliable perception system is always able to detect the road-plane and perform at an acceptable frame rate (20Hz or more).
- What level of hardware is required to implement a working solution? Assuming the working solution is the system reliably performing at 20 Hz.
- How is the solution performing compared to the first year Formula student approach of 2018?
- Is a data-driven approach better than traditional methods for a Student Formula driverless scenario?

2
Background

2.1 Overview

In the autonomous driving industry, the problem of detecting objects is usually solved by sensor fusion between several sensors, including camera, LiDAR and radar [7, 8]. Because autonomous driving is targeting the complex and uncontrolled environment of consumers autonomous driving (amongst others), safety requirements are very strict and play a main role in the development of perception techniques [9]. While attempts in direction of camera-only object detection approaches are made by various companies and sensor manufacturers [10, 11, 12], current camera-only approaches are deemed as futuristic options, because they do not provide necessary safety and precision for complex road environments as for now [9]. However, considering autonomous driving for non-complex environments, camera-only approaches are developed in other fields, for example UAV path corrections [13] and blind persons assistance [14]. Currently, the desire to use a camera-only computer vision approach arises mainly from various restrictions of costs and power consumption. One example for that would be applications running on embedded systems, where power consumption is crucial [15], or systems where installing a LiDAR or radar is not possible due to weight or size restrictions [16].
When it comes to localization problem, there are several sensor technology options as well as algorithm choices. The most interesting topic in connection to this work is vision sensors. Notable technologies in this field are LiDARs, radars and cameras.

2.2 Sensors and instrumentation

This section is about the sensors used in the system and how they were instrumented.

2.2.1 LiDAR

LiDAR (Light detection and ranging) is a laser scanner instrument that emits light impulses and measures time of arrival (TOA) of mirrored wave to determine distance to the range and bearing of an object [17]. The device consists of light source, which emits light beams and receiver, which receives the light reflected from surrounding objects. As discussed in [17], the exact technique of measuring a reflected light varies depending on the manufacturer, resulting in different resolution and precision of the specific sensor, however the basic idea is the same. It usually involves oscillating

mirror for 2-D measurements and rotating mirror for 3-D measurements. The ways to determine range varies and well—some methods use the elapsed time between emission and arrival while others observe the phase shift between the transmitted and received light. A LiDAR gives the benefit of observing wide field-of-view and long ranges while being very precise. One drawback, however, is that it is very expensive compared to other sensors, sensitive to atmospheric conditions, not durable, and currently rather bulky compared to other sensors [17].

2.2.2 Radar

A radar is a sensor like the LiDAR in the sense that both estimate the time of arrival. However, as opposed to light waves used in LiDARs, radars utilize radio waves. Radio waves have much lower frequencies and thus, are less affected by weather conditions and can detect much more distant objects. In addition, radars usually consider the Doppler effect of the echo thus, observing velocity of the moving object without further numerical processing [17]. Radars are also superior to LiDARs in terms of cost, which makes it more practical for consumer applications. The drawback of radars is the resolution; because radio waves are significantly smaller compared to light waves in LiDARs, recognition of an object is currently not possible with radars, and they should be used in conjunction with object recognition methods via other sensors to implement complete a perception solution.

2.2.3 Vision sensors

Originally vision sensors were challenging to use for localization, since vision transforms the 3-D world to plane world. While methods for extracting that information were developed, they were not addressing computational complexity to retrieve that information. However, the recent advances in computer technology sparked new attention to vision sensors for perception purposes. Vision sensors provide plentiful information about immediate surroundings and are used to detect an object, recognise lane marks, traffic lights and traffic signs. One of the most interesting and promising sensors in the field is a stereo camera with two lenses. The depth information could be recovered since the scene is shot from two different perspectives. The regular approach is to use the triangulation method for this calculation, which will be discussed later in this paper. Vision sensors are significantly cheaper than range sensors such as LiDARs and radars and hence are very appealing for usage in consumer applications for localization purposes.

2.3 Autonomous driving in racing

This section should give a short overview about existing autonomous racing classes.

2.3.1 Roborace

Connecting the area of autonomous driving to the topic of making a Formula student race car drive autonomous, it is worth to look in what is happening aside the main intention of making a streetcar go without a driver. Professional companies have come up with driverless competitions in the field of racing. One of the most known classes is the competition *Roborace*, that evolved as a side class of the well-known *Formula E* competition. [18]. In this class, it is the goal to drive on the same circuits the Formula E cars drive, with a specific for autonomous driving developed race car. The race cars itself are equipped with several sensors [19]. The first version of the car from 2017 makes use of different sensors like LiDAR, radar, cameras and GPS, merged to calculate the path, car position and further things to enable the autonomous driving. As the computing united it uses a Nvidia Drive PX 2 computer. It's computing power of 8 teraflops guarantee enough resources for making those calculations in the live system.

2.3.2 Formula student

To be able to make a direct comparison, it is further worth looking in what other teams in the Formula student competition could achieve until now and what technology they used. More precisely, it is a good idea to look at the approaches the most successful teams of the last 2018-year competition developed. Last year's winner of the competition in Germany was the team of ETH Zürich. The approach they utilized for their technical system for environment perception was based on sensor fusion of a LiDAR, three cameras (mono and stereo) and an IMU and GNSS module. In addition to that, the car had a subsystem that fuses motion detection from different sensors. Localization and mapping were utilizing a SLAM algorithm. The path planning algorithm the team was using takes the outputs of these systems to control the car. Furthermore, the team also developed a framework which can make improvements on the lap time by complimenting the sensed environment with additional data on every lap.
Especially for this thesis, it is sufficient to take a deeper look only into their perception system. For failure protection reasons, the team developed a system that merged results from LiDAR and vision together while also providing functionality on its own in case of the shutdown of one of the systems. To detect cones, the LiDAR based system makes use of a CNN which uses the intensity pattern for each cone to determine which colour the detected cone has. In order to detect the cone itself, the LiDAR (mounted on the front wing, thus at the height of the cones) uses a ground removal algorithm as well as some noise reduction algorithms to get return points for each cone.
The camera sensing makes use of a neural network in addition with a key point regression and pose estimation. The neural network was based on the YOLOv2 architecture and returned a bounding box for each detected cone in an image plus the confidence for each. Afterwards, the key-point regression adds key points (top and bottom of cone e.g.) to each cone and by measuring the distance between those key points on the cone the distance can be estimated. To locate their position in real world, making a 3D to 2D conversation is done by using triangulation [20].

Another successful team of the 2018 competition was Team e-gnition from Hamburg. The sensor system they were using to percept the cones was similar to the AMZ approach. It also was a sensor fusion between LiDAR and camera.

3
Theory and implementation

This chapter gives an overview of the theory behind the modules that were developed.

3.1 Camera frame capturing

The input sensor to the system is the ZED stereo vision camera by Stereolabs [21]. The camera stores the distance information for each pixel through triangulation and by making use of its two lenses [22].
In autonomous driving, monocular, RGB-D or stereo cameras are used for most cases. Since the goal was to reduce the number of sensors on the car, it was decided to go with a stereo camera. This type of camera can provide a depth for each pixel, which is needed to calculate the distance of a detected cone from the car. Table 3.1 contains the specifications of three cameras that were considered for this thesis project.

	Stereovision ZED [21]	Multisense S7 [23]	E-con Sys. Tara [24]
Max. Resolution	4416x1242	2048x1088	1504x480
Max. Frame Rate	120Hz	30Hz	60Hz
Rate @ Resolution	1280x720 @ 60Hz	2048x1088 @ 30Hz	1504x480 @ 60Hz
Depth Range	0.5 to 20m	0.4m to 1.5m	0.5 to 3m
Field of View	90° H, 60° V, 110° D	80° H, 45° V	60° H

Table 3.1: Specifications for different cameras of choice

By looking at the specs for each camera, it gets clear that the *Stereovision 3D camera ZED* has the best depth characteristics. The necessity of looking further ahead was a crucial point in the system, since the aim was to drive the car at speeds of higher than $15m/s$. Further on, the ZED camera was already present in the set-up of the last year's team, which favoured its reuse. Therefore, it was decided to keep the existing hardware. The downsides of having great specifications is the computational power that is needed to process the large number of high-resolution images provided by the camera. Therefore, it was necessary to step down with some of the specifications. A resolution of *1280*x*720* was chosen, since that ensured to feed the CNN with the accuracy that was needed to make predictions on cones. Further, it was not necessary to run the camera at 120Hz but at 60Hz, since the output generated was aimed to have a frequency between 30Hz and 60Hz.

3.1.1 Camera API

Stereolabs provides an SDK for the camera, which let us control settings like resolution and frame rate. Furthermore, it calculates the depth of each pixel by comparing and processing two simultaneously recorded frames (left and right). This information is important for the post-processing of the data. In case a cone gets detected in a specific pixel area, the depth of the location can be returned by the API to figure out how far the cone is away from the car. Given that, plus that the software is officially provided by the manufacturer of the camera made it an obvious choice to include that into the system.

3.1.2 Camera calibration

In order to get the data like focal length or the principal point offset for the calibration matrix K (3.3), the manufacturer provides a calibration tool [25]. In addition to that, each camera comes with its own, camera specific calibration file. Stereolabs calibrates every camera individually after manufacturing. By that, it is ensured, that every camera is calibrated in the best possible way by using its calibration file. By experimenting with the calibration tool and comparing those results to the factory calibration, it was evident that the factory calibration is good enough and does not change much from transportation or the environment.

3.2 Object detection

After capturing the frame, the task of detecting a cone position within the frame is performed by a neural network. Since the scope of the project is a fast-moving environment of the Formula student competition, inference time should be as small as possible. Networks that perform on frequencies greater than 30 FPS are referred to as real-time object detection networks and are deemed suitable for live systems. Current state-of-the-art detectors are divided into three categories: classical object detection, two-stage detectors and one-stage detector. Classical detectors, such as R-FCN, use the sliding-window technique. The essence is to apply a fully convolutional network to the selected region of the image, which is called window. This window is then moved (hence sliding window) and the process is repeated. Two-stage detectors, such as R-CNN, have a two-step detection procedure. First stage is to generate a set of candidate proposals, which are then classified by a second stage to respective foreground or background classes. As opposed to two-stage detection models, one-stage detectors are not generating any candidate proposals, but rather running the classification on the full image simultaneously with class probability calculations. Examples of such detectors are OverFeat, SSD, and YOLO. Such detectors are tuned for speed and usually have a major trade-off of speed vs accuracy. Since the speed is crucial for the project usage, while accuracy is not critical, one-stage detectors are the right fit for the task of detecting cones in real time.
For this project it was decided to focus on a one-stage detector called *you only look once* (YOLO). The core idea of YOLO is that the network is trained on a full image and running a bounding box prediction plus class probability calculations in a

single go. As opposed to sliding-window or region-proposal techniques, YOLO sees the entire image during test and training. Because of this, the network has a larger context and has a better grasp of contextual information.

The first YOLO version (YOLOv1) consisted of 24 convolutional layers followed by two fully connected layers. The task of the convolutional layers is to extract features of an object while the fully connected layers are responsible for output probabilities and coordinate prediction. The network was designed for and pre-trained on a 1000-class ImageNet. YOLO has a unique loss function that only penalizes classification error if the object is present in the respective grid cell. Moreover, it is only penalizing bounding box coordinate error if that predictor has the highest *intersection over union* of any predictor in that grid cell. In YOLOv1, each grid cell can only predict two boxes and have one class, therefore it limits greatly the number of nearby objects the algorithm can predict. Essentially it struggles to predict groups of small objects, and if several objects are only covered by a few grids, the network would not have enough class predictions and bounding box positions to cover them all. Another problem is that the loss function is penalizing errors for small bonding boxes for the same amount as for the large ones. Small error in a big box is relatively insignificant, while small error in a small box imposes much greater effect on IOU.

The second version of YOLO (YOLO9000) is aiming to address the problems of the first one. It focuses on improving localization and recall, while maintaining classification accuracy. Usually such improvements are achieved by making a network deeper, however, since all YOLO networks are designed for the live system application and this would result in greatly increased inference time, the creators of the network did not utilize this approach. Instead, improvements of layers were made, such as adding batch normalization to improve regularization and convergence speed, using anchor boxes in convolutional layers to improve recall and potential improvement space. A vast improvement was made in training for detection and localization to address issues of generalization and different input sizes.

The most recent version of the network is YOLOv3. While YOLOv2 was one of the fastest and most accurate networks at the time, it was still performing poorly with small clusters of objects because of major down sampling throughout the network (32 times from the original image size). This was partially addressed by introducing identity mapping-concatenation of feature maps from the previous layer to capture low level features. To address these and other problems of previous version, YOLOv3 implements several state-of-the art mechanisms that are now a staple in most modern algorithms for object detection. Such techniques include up sampling, residual blocks, and skip-connections. As seen in architecture representation on Figure 3.1, YOLOv3 is performing a detection on 3 different scales, down sampling original dimensions by 32 and then up sampling to 16 and 8 respectively, which allows to detect objects of various sizes. The features extracted before each detection are fused in convolutional layers following respective detection blocks. The shape of the detection kernel is $1 \times 1 \times (B \times (5 + C))$, where B is number of bounding boxes each individual cell of feature map can predict, and C is the number of classes, represen-

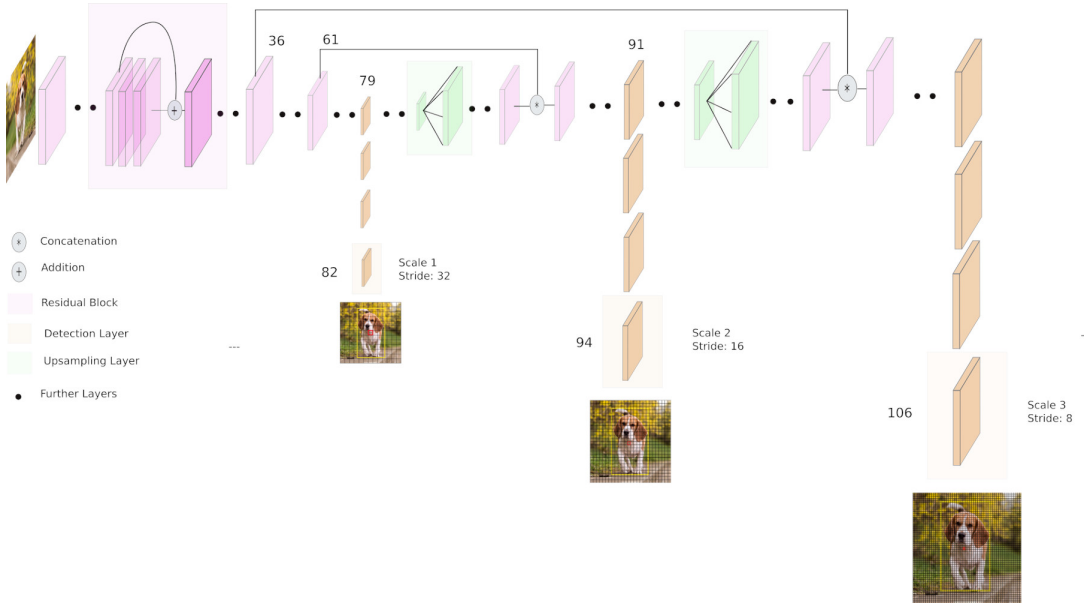


Figure 3.1: YOLOv3 network architecture.

tation of the kernel can be seen in Figure 3.2. In first versions of the network the loss function was constructed in a way that would penalize errors for small bounding boxes equally to the large ones, which resulted in network overfitting for small bounding boxes. To address this problem, loss function was modified in YOLOv3. Namely, squared errors in a loss function were replaced with cross-entropy error terms.

All the improvements in latest YOLO version came with a cost of inference speed. To make a detection in three different scales authors had to make network 106 layers deep. This decreased inference speed 1.5 times. However, with calculation power improvements and cost reduction of GPU this reduction is acceptable in many cases.

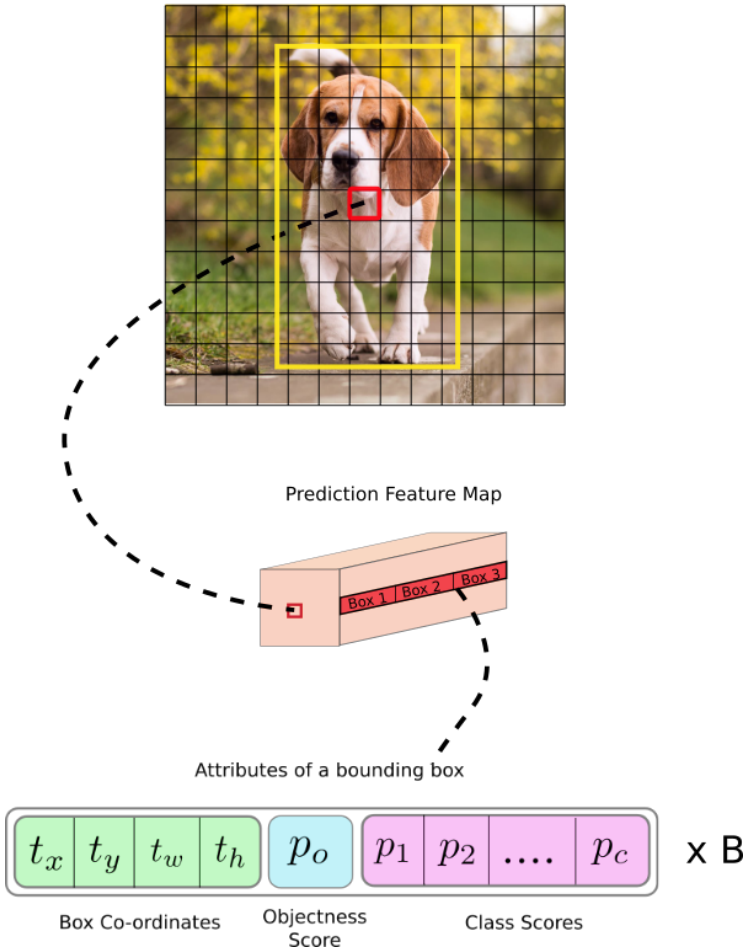


Figure 3.2: Image grid. The red grid is responsible for detecting a dog.

3.3 Network

This section discusses preparation and usage of the data, tuning, modification, and training approaches for the selected YOLO version.

3.3.1 Data annotation

Since object detection is a supervised learning model, data had to be gathered and annotated to be able to train the network. In a first approach this was done by accessing the data from CFSD2018, but since they were using different object detection approach, the annotations were not applicable. To annotate the data, the open-source annotation tool Yolo-mark[26] was used for relatively fast and convenient data annotation. The result of single image annotation is a label-file that represents ground truth for one cone detection per line: class ID and bounding box coordinates relative to image size. An example of a label-file for image shown in Figure 3.3 is presented in Table 3.2. The data set consisted of images in different weather and light conditions. Trying to cover as big variety as possible, snow, rain, good sunlight and evening conditions were included. The annotations were made

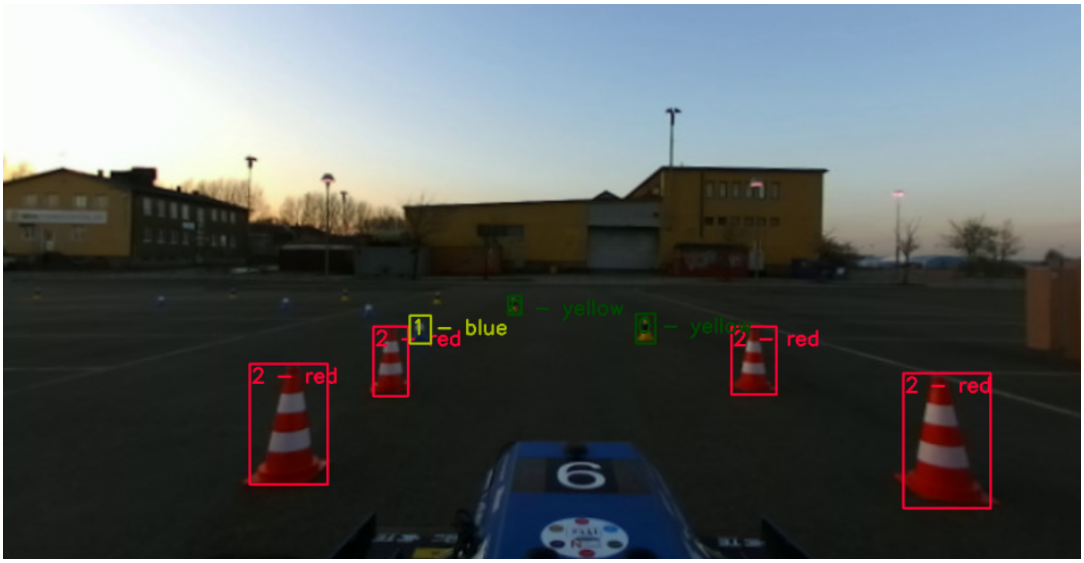


Figure 3.3: Image annotated by labels from Table 3.2.

with the assumption that the smallest detectable cone is no smaller than 10×10 pixels on the original image. The resulting data set consisted of around 800 image and label-file pairs.

class ID	x_{center}	y_{center}	height	width
2	0.387500	0.645139	0.032812	0.093056
1	0.402344	0.602778	0.017188	0.041667
2	0.333203	0.705556	0.053906	0.144444
2	0.666797	0.638194	0.035156	0.098611
2	0.779297	0.718056	0.058594	0.155556
0	0.587891	0.603472	0.017969	0.045833
0	0.477344	0.572222	0.014062	0.033333

Table 3.2: Labels for the image at Figure 3.3

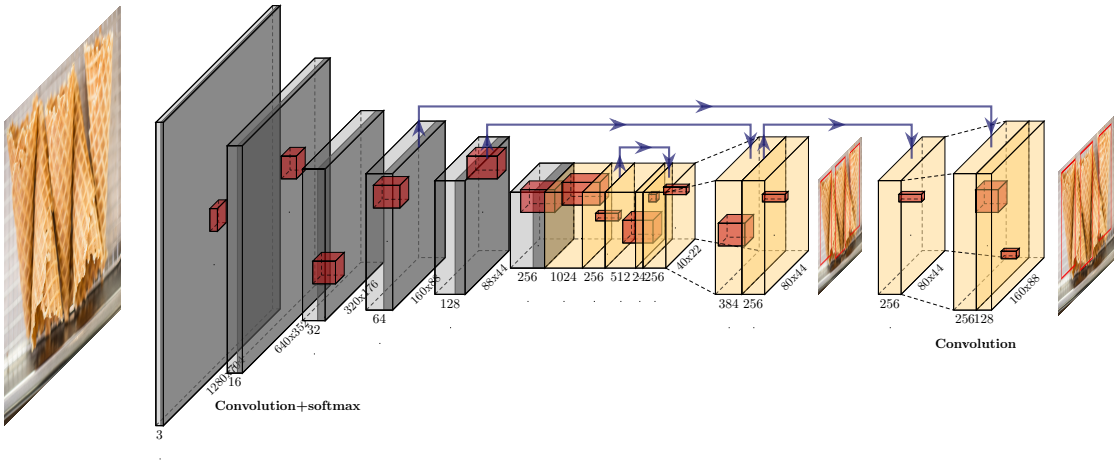
3.3.2 Network architecture and hyperparameters

The data of interest from the image is the position of the cones. The task of this module is to perform object detection relative to the camera position and pass this information to post-processing. The detection is performed by neural network based on YOLOv3 architecture discussed in theoretical section. The full YOLOv3 network is designed for a huge number of various objects, which have different shapes, sizes and colours. Essentially it is designed to learn big feature variety. As discussed in theoretical section, to have a better detection of various object sizes, the architecture implements detection on 3 separate scales.

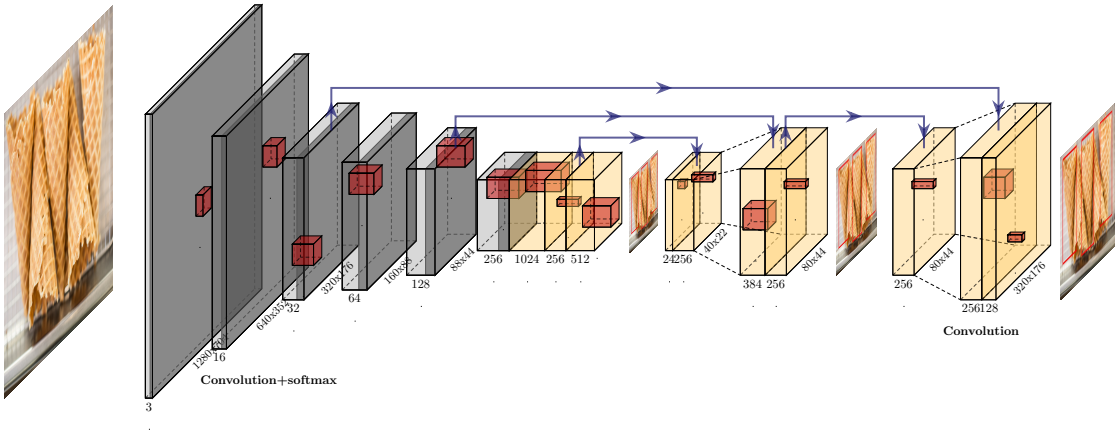
Since our goal is to detect cones of three colours and two possible sizes, there is not much feature variety or size difference for the objects. Therefore, a complicated architecture was not needed, which sacrifices inference speed to accuracy. However,

there is a minimal version of YOLOv3 called TinyYOLOv3, which is designed for systems with low computational power. It still retains detection on three different scales, but has decreased number of convolutional layer for feature detection. It was decided to base the architecture on this minimal version.
Firstly it was decided to remove the YOLO detection layer, responsible for detecting larger objects. Network architecture for this change is represented in Figure 3.4a. The network consists of 27 layers in opposed to 106 layers for YOLOv3. The reasoning behind these changes to the original architecture is discussed below.
As discussed in theoretical section, the first detection layer is responsible for detecting the large objects, the second one is detecting medium objects, and the last one detecting the small objects. Since it is not expected to have cones that, relatively to picture frame, can be considered big, it was decided to remove the first detection layer from the architecture. Together with the first detection an attempt to do something about respective convolutional layers was made. Removing down sampling at layer ten and up sampling at layer 17 resulted in a much slower inference time, while detection precision remained the same. Removing all convolutional layers responsible for first detection resulted in a great precision lost, since too much feature information was lost. In the end one convolutional layer was left, followed by a softmax layer and five fully convolutional layers. This composition gave same precision on our data as the model with the detections, while lowering inference time (three fully convolutional layers were removed and one YOLO detection layer).
Since the only objects of interest are the ones that are roughly having the same aspect-ratio, a number of anchor boxes was reduced from nine (default) down to five and aspect ratios of these anchors were adjusted to be more representative of cone sizes. Individual cone could cover as much as approximately 1/20 of the picture, and the smallest cone is about 10×10 pixels big (assumption taken during the annotation). With that in mind, the anchor box sizes were set according to Table 3.3 to cover variety of cone sizes. The number of anchor boxes was reduced from nine down to five, with boxes IV, V for the last detection layer and I, II, III for the first detection layer.

After experimenting with two detection layers-based network it was realized that the network is not detecting the smallest objects far away and it was decided to get back the additional detection layer and back to the original nine anchor boxes set-up. However, instead of using the default anchor boxes values as before, it was decided to calculate the anchor values specific for given data. K-means with nine clusters were used to get a nine anchor box values out of available training and test data. Values of the new anchor boxes are presented in Table 3.4. Detection layers were responsible for sets of three anchors each, going from bigger anchors to smalls ones with each up sampling respectively.
To address the problem of detecting small cones, along with anchor box improvement, it was decided to increase upsampling value for the last detection from two to four. That also meant that the skip connection had to be moved to the convolutional layer of respective size. The final network is presented in Figure 3.4b.



(a) First version.



(b) Second version.

Figure 3.4: TinyYOLOv3 architecture adaptation.

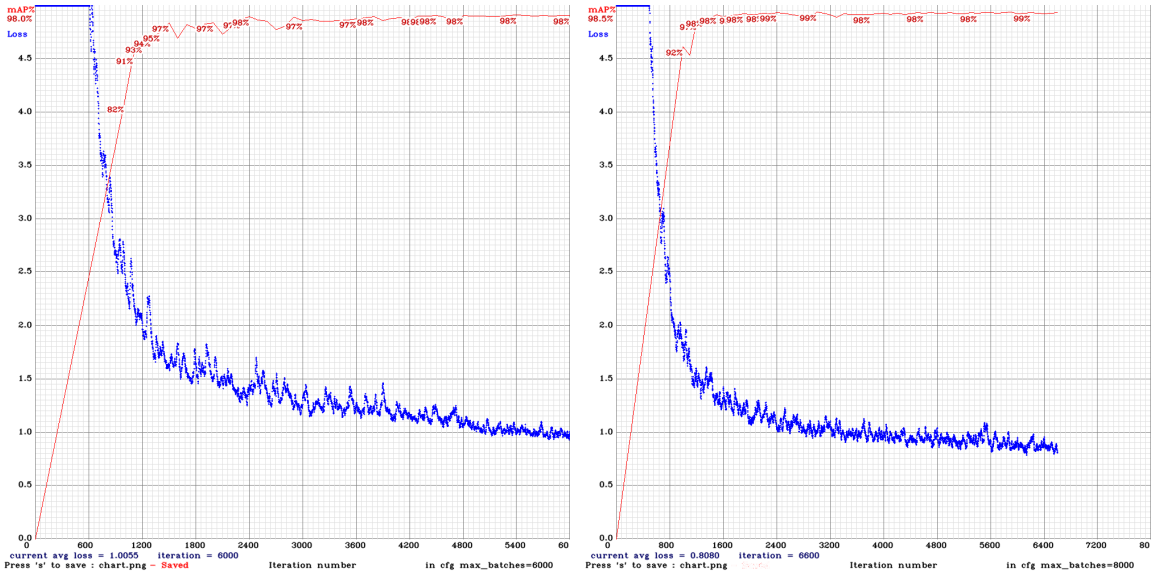
To train the network the data was split into training and testing datasets with proportions of 80% and 20%. The training was done with a batch-size of 64 and a learning rate of 0.001. It was recommended by the YOLO creators to train the network for 2000 steps for each class, so the training cycle was limited to 10000 steps, using learning rate decay of 0.0005 after 8000 steps. With the training taking 8000 steps the learning rate was adjusted after 7100 steps and 7500 steps.
For the better accuracy a training resolution was set to 608×608, which was later changed to 1280×704 for the inference.

anchor	I	II	III	IV	V
width	4	7	13	25	41
height	7	15	25	42	67

Table 3.3: Anchor box sizes for the first version of the network

anchor	I	II	III	IV	V	VI	VII	IIX	IX
width	5	6	8	9	12	15	18	24	33
height	13	16	19	23	28	34	42	54	75

Table 3.4: Anchor box sizes for the second version of the network



(a) First version **(b)** Second version

Figure 3.5: Different network version training results.

3.4 GPU

Since the approach discussed in this work involves a deep neural network that has to make object detection in a real time, it was crucial to utilize a GPU power for computing. The darknet library is providing a CUDA implementation of all layers. Using GPU greatly improved speed of the application, making it possible to train the 604×604 network in 3 hours on the NVIDIA RTX 2080.

3.5 Post processing

After detecting a cone from a frame, pixel coordinates are used to calculate 3D coordinates from it. In addition to the captured pixel coordinates, some additional, camera specific factors had to be included in the calculation to get the 3D coordinates.

3.5.1 Cone distance

Figure 3.6 illustrates the mathematical connection between the two points—p on the image plane in a 2D coordinate system and P, the real world 3D coordinate of point p. The centers of both coordinate systems are in C, were the camera is located. Point O is a principle point, which is the projection of the center of the camera in the image. The focal length f is the distance between points C and O [1].

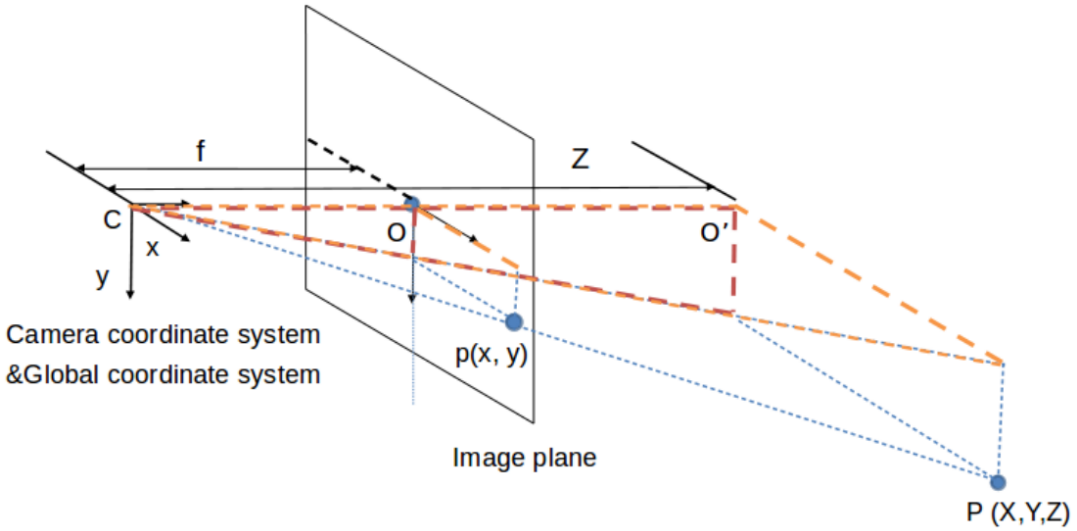


Figure 3.6: [1] 3D-point P and its projection p on the image plane. Their mathematically link can be described by the calibration matrix K.

From the two triangles *C-O-p* and *C-O'-P* the following mathematical link can be derived: f to Z (Z is a perpendicular distance of P from camera) is equal to x to X (Figure 3.1). The same is valid for the connection between y and Y figure 3.6).

$$\frac{x}{X} = \frac{f_x}{Z} \tag{3.1}$$

$$\frac{y}{Y} = \frac{f_y}{Z} \tag{3.2}$$

In addition to that, the principal point offset has to be included in the calculation. This offset is the distance between the origin of the film (or frame plane) and the principal point. As shown by Figure 3.7 [27], the principal point is located where the line from the *pinhole model* camera and the frame plane intersect.

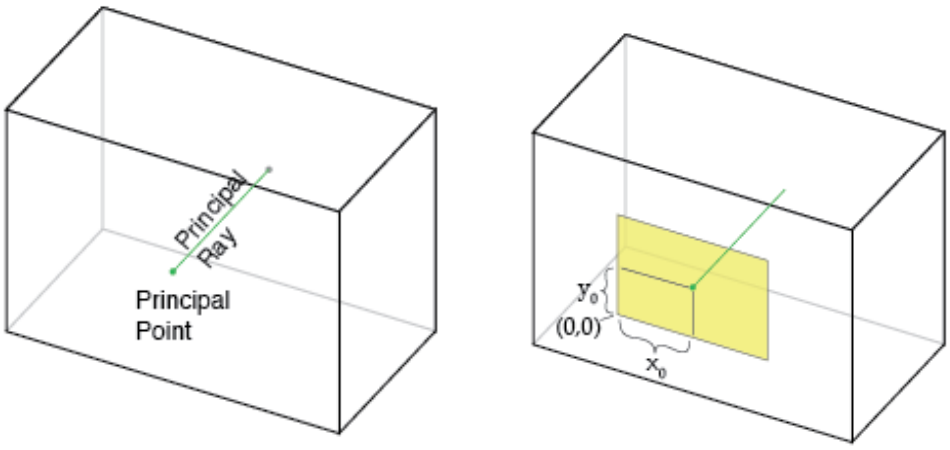


Figure 3.7: The principal point offset.

Connecting all the equations and their relations together leads to equation (3.4) and to the intrinsic, or calibration matrix K (3.3) as parameterized by Hartley and Zisserman [28].

$$K = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$Z \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{3.4}$$

For the 3D points X and Y from equation (3.4), (3.1) and (3.2) can be concluded:

$$X = \frac{Z(x - x_0)}{f_x} \tag{3.5}$$

$$Y = \frac{Z(y - y_0)}{f_y} \tag{3.6}$$

3.5.2 Cone distance by size

Another way to calculate the distance to a cone is by the size of the respective bounding box. The distance to an object **d** in m can be calculated with a knowledge of the real object height $\mathbf{h}_o$ in m multiplied by the focal length **f** in mm, divided by the object height in the frame $\mathbf{h}_{fmm}$ in mm:

$$d = \frac{h_o * f}{h_{fmm}} \tag{3.7}$$

To get the object height on the sensor (in the frame) $\mathbf{h}_{fmm}$ in mm, the sensor height $\mathbf{h}_{smm}$ in mm is multiplied by the object height $\mathbf{h}_{op}$ in *pixel* and divided by the sensor height $\mathbf{h}_{sp}$ in *pixel*:

$$h_{fmm} = \frac{h_{smm} * h_{op}}{h_{sp}} \tag{3.8}$$

To get the coordinates for a two-dimensional view after calculating the perpendicular distance to a cone, the 3D X coordinate can be calculated according to equation 3.5.

3.5.3 Implementation

To process the data in order to get the 2D coordinates X and Y for each cone according to 3.5 and 3.6 a software module is created that processes the data. For that, the calibration matrix 3.3 filled with the data from the calibration file mentioned before is used. The camera is calculating the pixel from the left camera lens, therefore, the calibration data for the left camera lens is taken to process the data.

In the first approach, the coordinates are taken from the camera API, which automatically calculates and outputs them for the requested pixels where a cone was detected. Because the camera can guarantee depth sensing only with a mistake of about 20% in accuracy for objects that are around twenty meters away, it was decided to implement a second approach the cone distance is calculated from its pixel size in the frame it was detected. Having the height of a cone in pixels allows to calculate the cone distance given the actual height of a cone. The cone size is known from the competition rules.
To get a better evaluation of the cone distance X, both approaches were merged and the perpendicular distance is taken from the method that gives a higher confidence for the given cone.
By using the merged approach results improved. However, in a range to the camera of around four meters the depth estimation was way more precise than the *distance by cone size* estimation. To improve the post processing, a threshold of four meters was implemented which made the system more accurate.

3.6 Tools

This section gives an overview of the main tools that were used in the perception software module.

3.6.1 OpenDLV

OpenDLV is *"A modern microservice-based software ecosystem for self-driving vehicles"* [29]. Using this ecosystem allowed all the software modules to be integrated into one microservice with a defined input and an output.
Looking at the system overview 1.2, the input to the microservice is an environment captured by the camera. Inside the microservice (the blue box) Zed camera API, the neural network, and the Birdview processing module resides. As an output for the microservice, the 2D coordinates are sent out to be taken as an input by a path planning microservice which is not in context of this thesis. OpenDLV provides the architecture to set up a microservice with the modules and the communication between other microservices. A common communication protocol is used across the entire OpenDLV system. In case of microservice implemented in this work, a start and stop message is sent out for every frame that was captured and processed. In-between those messages, all the calculated 2D coordinates are sent out together with the cone IDs and their confidence for every cone individually.

3.6.2 Docker

Because every system has different set-up, different libraries, tools, and versions, it is problematic to run a library and tools-dependant software on a different systems. To solve this problem, the system makes use of Docker. The microservice and all necessary libraries, drivers, and calibration files are packed into a Docker image and can be run inside a Docker container on any platform.

4
Results

This chapter unifies the results obtained during this work.

4.1 Bounding box detection and classification

To measure precision of the network, the test data set was built and annotated, consisting of 100 images with approximately 25% representation for each of the four classes. *Intersection over union* (IoU) was then used to measure the localization of the cones for each given detection.

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

The results for all of the four classes are presented in Table 4.1.

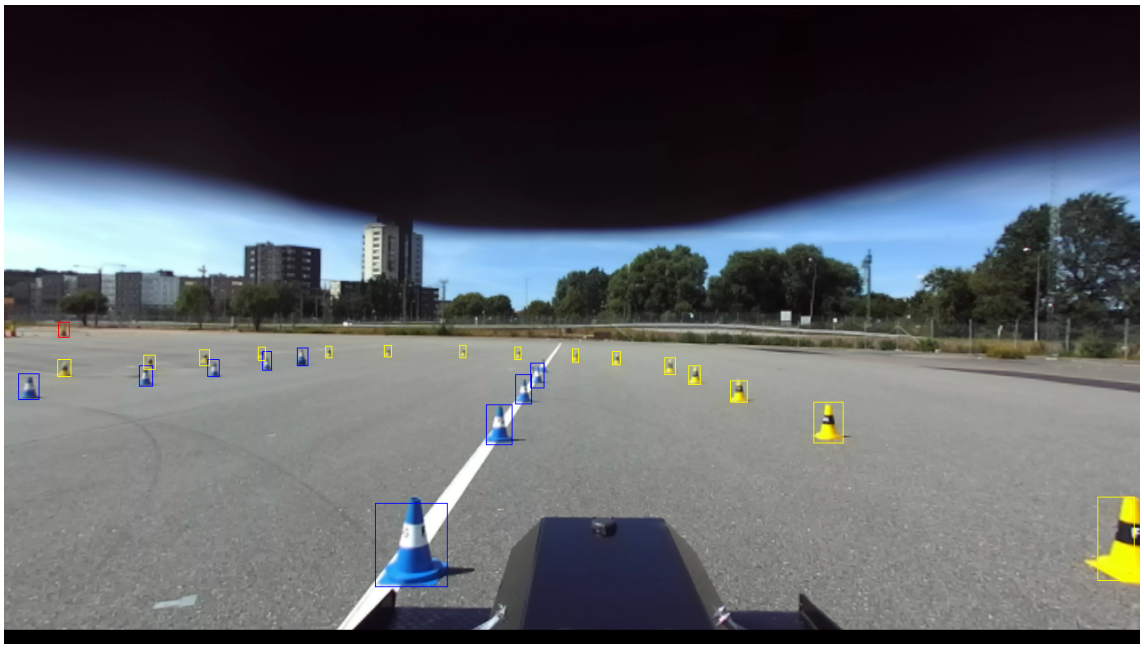


Figure 4.1: Detection result running real-time on a car.

	yellow	red	blue	big red
Avg	95%	96%	87%	98%

Table 4.1: Average intersection over union for the best network on 100 image test set

With respective IoU values mAP (mean Average Precision) for these classes was 56.4, which is much better than mAP for YOLOv3 on the benchmark COCO data set. The reason behind such an improvement over benchmark is having only four similar classes with similar features.

4.2 Accuracy

To measure the accuracy of the system, a test field was set up. The cones were set-up at certain distances and these distances were compared to the output of the system. For this first test, the distance was only calculated by the API and the system was not making use of the distance calculation by the cone size in the frame. The test was done as follows: all three cone colours, orange, blue, and yellow were placed in two, four and eight meters in perpendicular distance, with an offset of one meter to the left and one meter to the right relatively to the camera.

2m	2d coordinates[m]					
	x			y		
	l	m	r	l	m	r
red	-1,02	-	1,07	2,15	-	2,65
blue	-1	-	0,86	2,07	-	2,15
yellow	-1,01	-	0,84	2,09	-	2,12

4m	x			y		
	l	m	r	l	m	r
red	-0,9	-0,04	0,89	3,76	3,93	4,06
blue	-0,9	-0,02	0,85	3,79	3,75	3,92
yellow	-0,89	-0,03	0,88	3,77	3,81	4,03

8m	x			y		
	l	m	r	l	m	r
red	-0,92	-	0,8	7,27	-	7,4
blue	-0,92	-0,07	0,81	7,31	7,6	7,37
yellow	-0,93	-0,05	0,8	7,27	7,53	7,25

Table 4.2: Accuracy test results by distance and position left(l), middle(m), right(r)

Table 4.2 shows the results of this test. The numbers show the measured positions from the optical system. A visualization of this can be found in Figure 4.2 on the right. The figure shows the bird perspective 2D view for the measured numbers.

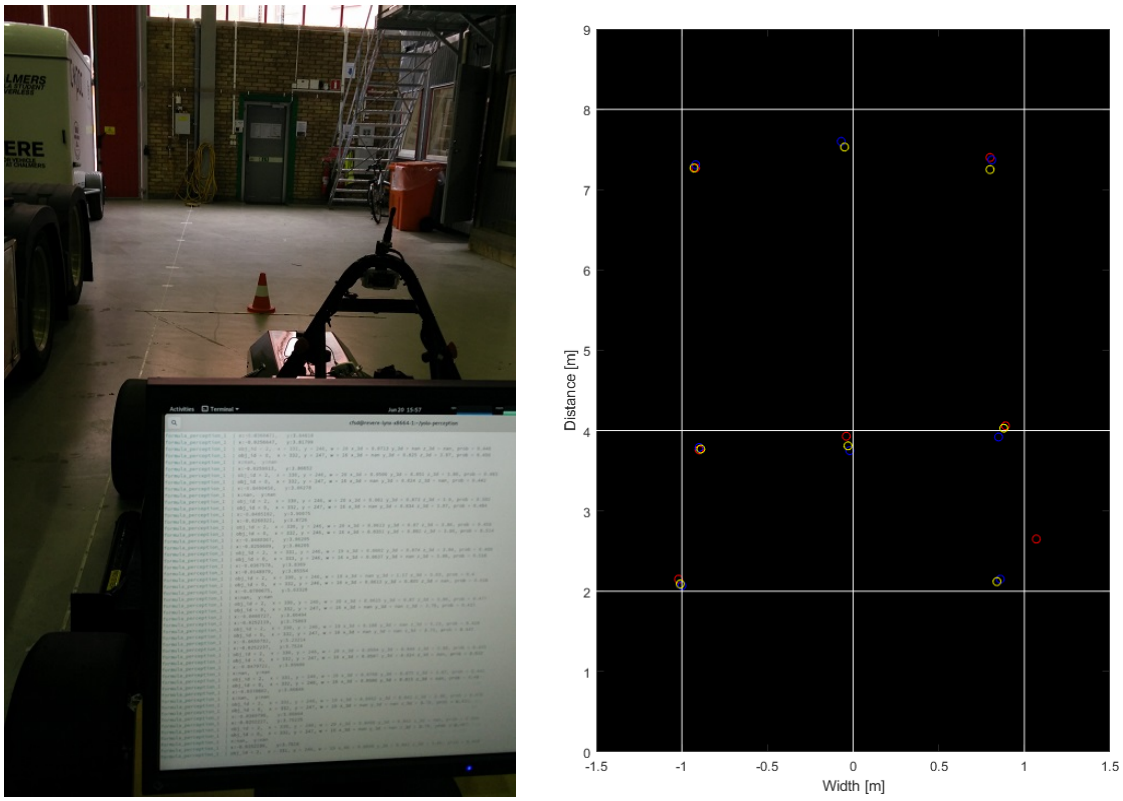


Figure 4.2: Accuracy test organization (left) and result (right).

The camera centre is in position *0,0* of the coordinate system. The white auxiliary lines indicate the positions where the cones where placed in the test field (at their crossings). The different coloured circles indicate the measured positions for each cone colour by the system and therefore the deviation between the real and measured positions.

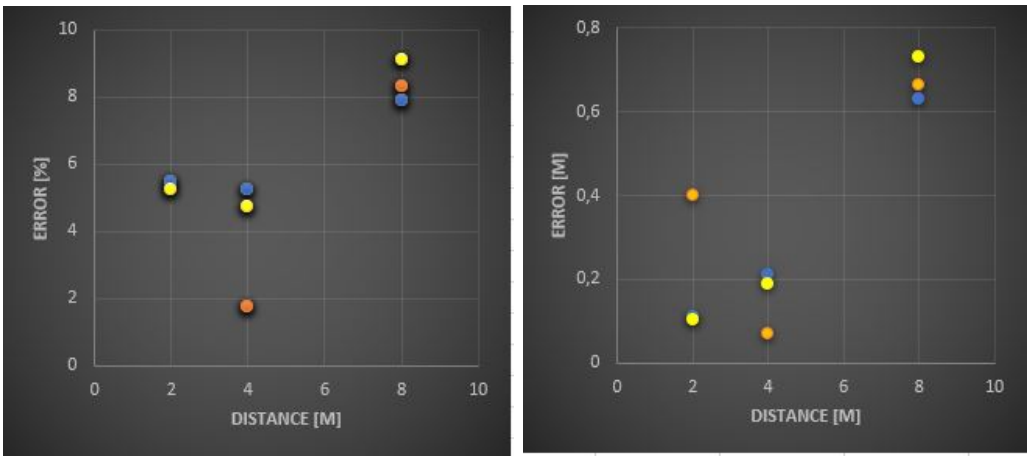


Figure 4.3: Percentage (left) and distance (right) error by color of cone.

The two figures in 4.3 show the error between measured values and real positions. The test showed that in a distance greater than eight meters, the system was not able to detect any cones regardless of the colour. This is due to the fact that the

network was trained with the data from an outside environment. The test was done inside the Revere lab with a different light conditions and a very different background compared to the training data.

4.2.1 Cone distance by size

As described in 3.5.3, a second approach to calculate the distance to a cone was implemented by using the size of the cone in an image frame. The result is displayed in 4.4.

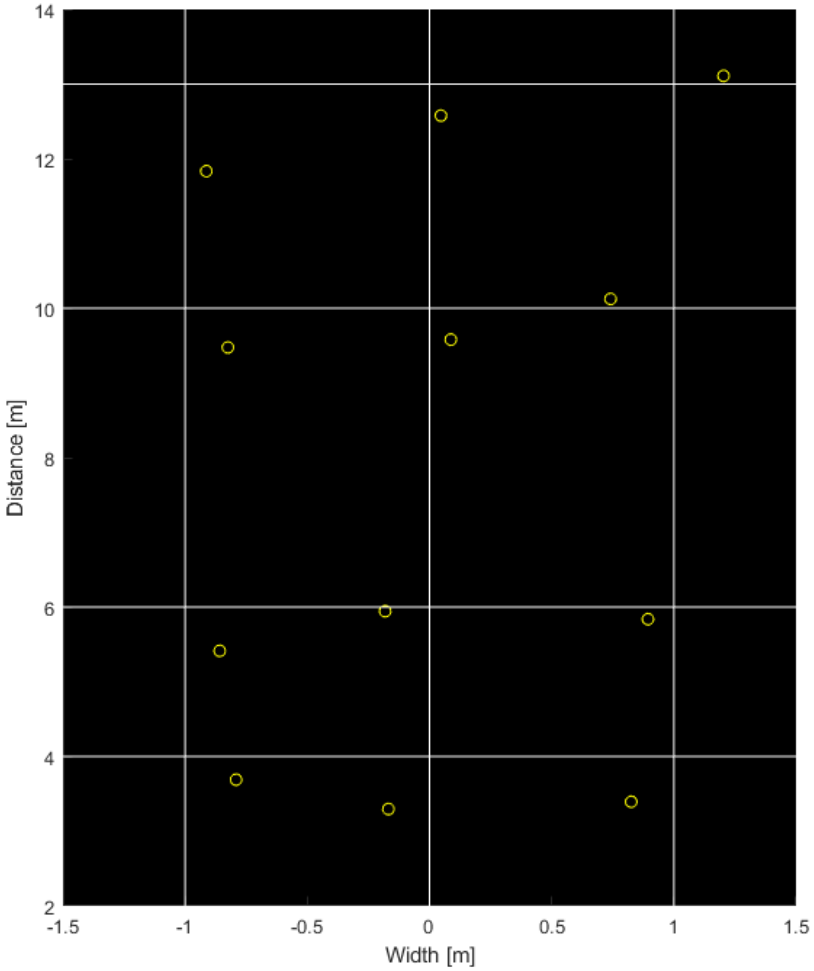


Figure 4.4: Accuracy test result for distance calculation by size.

The test shows similar results in accuracy for both methods.

4.2.2 Merged approach with threshold

Figures 4.5 and 4.6 show the absolute and percentile error in an accuracy for the same test as before. This time the test was done outside to have a more realistic environment in terms of both the light conditions and the intended field of operation.

Both graphs show all the methods evaluated separately, blue for *distance by cone size* and grey for *distance by camera depth*. The red, *merged* solution shows the approach of combining both methods together by using the higher confidence. It is apparent that, using this combined approach, the overall error gets smaller. However, observing the test results and the graph shows, that the approach distance by depth only was more accurate in the closer area to the camera up to four meters. The *merged approach with threshold*, represented by the yellow line, where the threshold of four meters was implemented to take the value of the *depth only approach* reading for every cone under that threshold, if available. Using this method, the overall error was reduced to under *5%* up to a distance of ten meters. Even in a longer distance, the error is never greater than in one of the single-method approaches.
During the tests it was discovered that detecting distance to the cone specifically by the camera triangulation was very problematic in a bright sun. Examination of the confidence map showed that specifically the *blind spot-confidences* of most pixels of a cone were below 50%. To address this problem, it was decided to get the depth value of the pixel with a highest depth confidence across the entire lower half of the cone. This approach greatly improved depth detection, since the module had to content to using cone-by-size calculation more rarely.

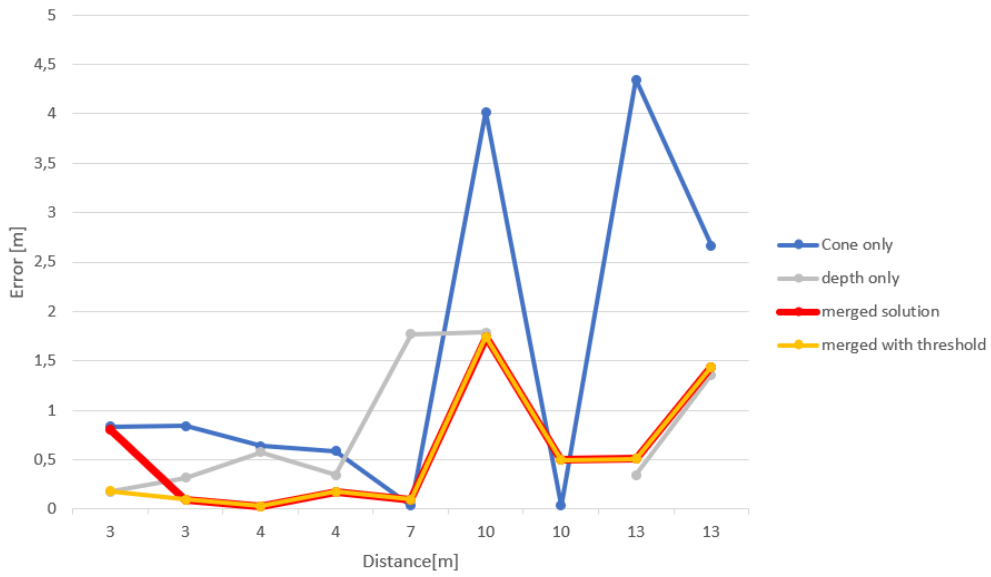


Figure 4.5: Accuracy test result for distance calculation merged approach absolute.

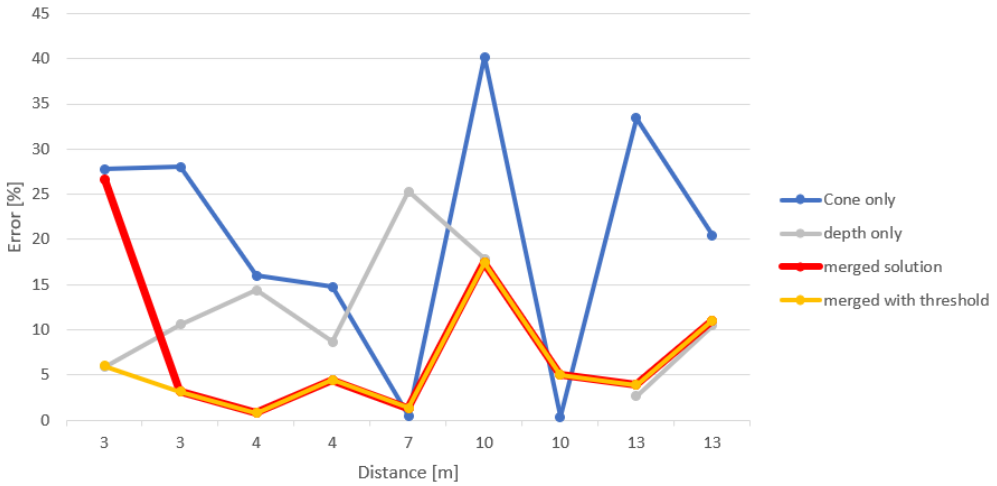


Figure 4.6: Accuracy test result for distance calculation merged approach percentage.

5
Discussion

The following chapter compares the goals that were set up initially and in how far they were fulfilled. Further the evaluation criteria were recalled to make a statement about the success of the project. During the project there were several network design and implementation decisions made, which are worth discussing. By looking into the initial research questions, the presented results can be discussed and compared.

5.1 From a research perspective

The first question stated the word reliable and was asking, if the system was performing in a reliable way. It was also asking, if a system with a camera only approach can work reliable. Comparing this question to the results, the last results of the merged approach show an error of under 5% up to a distance of 10 to 13 meters for the calculated output of the system. Concerning the input of the system, the cones detected in each frame had an accumulated rate of 94% with blue colored cones sticking out negatively, with a rate of 87%. Thinking of the fact, that the system evaluates between 18 and 25 frames every second and the actual speed the car should be moving at, which would not be faster than around 75km/h on a long straight, the system can be called reliable in case of detecting and calculating the cone positions and classes right and sending the data to the next module. This strategy can be supported by the fact that the main approach in Formula student aims to start slowly to be able to perceive the track and make a more precise path planning. When the circuit is perceived, the track gets only updated every round to verify it continuously. Therefore, a higher speed is only applied in a later stage of the process, what means, that the probability of a missed object until that point is going towards zero.

By comparing the result of the project with the perception solution of last year, it is good to discuss two approaches: A pure-vision approach versus a sensor-fusion approach. The scope of this project and the backbone of this year's project was to focus on one sensor, namely, the camera. The supposition was, that with a restricted time-schedule, which is the nature of Student Formula project, it would be beneficial to focus and work towards perfection on one particular sensor, rather than scatter the effort across multiple sensors and perception techniques. Last year's team was using both, LiDAR and camera, doing sensor fusion and detecting cone position only using the LiDAR, while using camera only for classification (color detection).

With such a vast difference in techniques it is hard to compare those numerically. However, while getting similar results for perception overall, the fact that the current solution achieves this with only one sensor and one algorithm is considered to be an overall improvement in the quality-complexity scale.

Another question was aiming for the hardware that is required for the solution to run with a good performance. Since the project was building up of the first years approach of the team last year, the existing hardware was considered sufficient to build the new system. The most expensive part computational wise, is the CNN, which is running constantly to detect cones. The use of NVIDIA's *CUDA* architecture to parallelize the process helped to improve the performance on the used GPU. In order to make the system run on a lower hardware setup, taking a simpler architecture to implement the neural network would probably bring the biggest improvement to still be able running the detection on a sufficient frequency. The downside of that would be a drop in the actual detection rate and confidence. The positive effect this would cause is less computational demand what could save energy what is a crucial point in a Formula student car, since the low voltage system gets powered by 24V batteries.

While a decision was made in using the tinyYOLO architecture as a basis for this work, the regular, more complex YOLOv3 architecture was also considered and tested to value the benefits it would bring versus execution time penalties. The full network was trained with the same anchors and same data as the one discussed in this work. While the execution time nearly doubled for the full network, it was apparent that the complete model shares most of the flaws of the tiny model: poor detection of clustered objects and vulnerability to changes in learned object orientation. However, the accuracy of localization was slightly better which would have improved the depth-by-cone-size approach used in this work. With performance still being the main pinnacle of the system, it was decided not to stick with a full network.

To map the way or road in another module, the location of the percepted cones needed to be calculated into the 2d image plain turned around by 90° into a topdown view or *birdview*. The necessary calculations for that were done using *triangulation*. While reviewing the module and its results, it was found that the accuracy was not good enough for the needs of the system. It was therefore decided to implement a second approach for distance calculation, which was running in parallel to the original approach based on triangulation by the camera. The second approach, based on the size of the detected cones in the frame, was therefore an additional method to improve the confidence of the distance for each cone. Hence, the merged approaches gave a better result on the output side, meaning more precise cone coordinates.

5.2 Other findings

Last year's solution for the camera mounting included a shielding for sun, which was supposed to reduce sun interference. However, after multiple experiments it

was apparent that the camera handles the task itself and adopts sensitivity very quickly. It was therefore decided to remove the shielding from the set-up.
Since Formula student is a rather specific scenario with it track limiting cones, finding and creating data to train the network is a big and time consuming part while developing a neural network. Data annotation was done manually with frames taken from the camera installed in the car in order to train the network in a time expensive but best possible way. The good results of the created network justify this work.

5.3 Competitors comparison

The technical systems of the teams from e-gnition Hamburg and in more detail, AMZ from Zürich were introduced in the *Background* chapter. To make a valuable comparison between the approach of AMZ and CFSD19, it makes sense to discuss the results of the perception system made by only the camera on the AMZ side. Their results show an error of less than 10% in a distance up to ten meter, which makes this part of their system not as accurate as the CFSD19 approach with an error of under 5% inside the same distance. However, looking at the combined accuracy of different sensors, their system is more advanced, since the combined output of LiDAR and camera accomplish a more accurate result. Unfortunately, their papers never mention any frequency their perception and algorithms are running at, which makes this important part of the system incomparable.

6
Conclusion

The initial purpose of this thesis was to investigate and develop a perception module for a Formula student race car. The perception module should have perceived the environment around the car and process it. The output of this module should have been constructed in a way that the next-in-chain module could process it and implement a path planning algorithm with it. Looking into the requirements based on the rules made by Formula student, how the event shapes them and the created output from that, a successful outcome could be implemented that suits the demands. As the first module in a chain of different software parts for the autonomous system of the car, an accurate and reliable system could be programmed to achieve a precise perception. Especially the reduction of the sensors by removing the LiDAR from the system, but remaining the accuracy at a speed three to four times higher than the last years implementation, underline this. By that, two expensive parts of the project, the hardware itself, as also the software implementation for the module plus the software implementation for the fusion of both sensors could be saved. The solution can therefore be seen as a simpler solution that achieves a better result and therefore an improvement compared to the earlier solution.

6.1 Future work

While this project is considered to be successful, there are various improvements that could be made in order to achieve better results.
First of all, one might revisit the full YOLO3 architecture in order to achieve better results with cone localization and, potentially, reduce the amount of false negatives. While it was deemed not worth sacrificing the detection frequency for the small detection improvement, there were no architecture adjustments made or hyperparameter fine-tuning done. Perhaps with these adjustments, one might achieve significant improvements in detection accuracy and with better GPU it would become reasonable to use the full model.
Secondly, while this work is purely focused on localization of cones in one picture frame, there is a great potential and field of improvement in building up a system that would have a memory of a few previous frames. The current implementation of YOLO by darknet achieves that by trying to distinguish between several instances of the same class and assign an ID to each one of them.
Thirdly, as suggested by the experience of other CFSD teams that have experience with a camera-only set-up, while the ZED camera is a good "working out of the box" stereo vision camera - it is not the best solution for stereo vision in general. Two

separate cameras mounted so that the base would be long enough would increase both, depth-sensing distance and precision greatly. Moreover, the ZED camera is using rolling-shutter which is inferior to global-shutter cameras in shaky and fast environments such as formula student.

Bibliography

[1] Weiming Li, Yumeng Jiang: *Stereo Visual Odometry using Supervised Detector*. Chalmers University of Technology, Master's thesis in Systems, Control and Mechatronics & Communication Engineering, 2018.

[2] Yan Tian, Judith Gelernter, Xun Wang Weigang Chen Junxiang Gao Yujie Zhang Xiaolan Li: *Lane marking detection via deep convolutional neural network*. Neurocomputing 280 (2018), 2017.

[3] Jianwei Niu, Jie Lu, Mingliang Xu Pei Lv Xiaoke Zhao: *Robust Lane Detection using Two-stage Feature Extraction with Curve Fitting*. Pattern Recognition 59 (2016), 2015.

[4] Sandipann P. Narote, Pradnya N. Bhujbal, Abbhilasha S. Narote Dhiraj M. Dhane: *A review of recent advances in lane detection and departure warning system*. Pattern Recognition 73 (2018), 2016.

[5] Jihun Kim, Jonghong Kim, Gil-Jin Jang Minho Lee: *Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection*. Neural Networks 87 (2017), 2016.

[6] Andreas Geiger (MPI Tübingen), Philip Lenz (KIT), Christoph Stiller (KIT) Raquel Urtasun (University of Toronto): *Road/Lane Detection Evaluation*. http://www.cvlibs.net/datasets/kitti/eval_road.php, 2013.

[7] Kocic, Jelena, Nenad Jovičić Vujo Drndarevic: *Sensors and Sensor Fusion in Autonomous Vehicles*. 11 2018.

[8] Pełka, Michał, Karol Majek, Jakub Ratajczak, Janusz Bedkowski Andrzej Maslowski: *Study of multi-sensor fusion for localization*. 1–2, 09 2019.

[9] Mariani, Riccardo: *An overview of autonomous vehicles safety*. 6A.1–1, 03 2018.

[10] Sun, Dongming, Xiao Huang Kailun Yang: *A Multimodal Vision Sensor for Autonomous Driving*. 08 2019.

[11] Pendleton, Andersen, Du Shen Meghjani Hong Eng Rus Ang Jr: *Perception, Planning, Control, and Coordination for Autonomous Vehicles*. 02 2017.

[12] Feng, Haase-Schutz, Rosenbaum Hertlein Glaser Timm Wiesbeck Dietmayer: *Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges*. 04 2019.

[13] Roggi, Gabriele, Mattia Giurato Marco Lovera: *A computer vision line-tracking algorithm for UAV GNSS-aided guidance*. 09 2019.

[14] Tian, Yingli, Chucai Yi Aries Arditi: *Improving Computer Vision-Based Indoor Wayfinding for Blind Persons with Context Information.* 6180, 255–262, 07 2010.

[15] Chai, Sek: *Mobile Challenges for Embedded Computer Vision*, 219–235. 01 2009.

[16] Alsaedi, Malik, Baraa Albaker, Hussein Dawood, Haider aoun Zeyad tayeh: *Quadcopter Based Object Detection and Localization.* Iraqi Journal for Computers and Informatics (IJCI), 43:5, 06 2017.

[17] Woo, Ami, Baris Fidan, W.W. Melek, Seyed Zekavat R. Buehrer: *Localization for Autonomous Driving.* 1051–1087, 01 2019.

[18] Wollman, Dana: *Formula E is planning the first racing series for driverless cars*, 2015.

[19] O'Kane, Sean: *Roborace details the tech behind its wild autonomous racecar.* 2016.

[20] Juraj Kabzan, Miguel I. Valls, Victor J.F. Reijgwart Hubertus F.C. Hendrikx Claas Ehmke Manish Prajapat Andreas Bühler Nikhil Gosala Mehak Gupta Ramya Sivanesan Ankit Dhall Eugenio Chisari Napat Karnchanachari Sonja Brits Manuel Dangel Inkyu Sa Renaud Dub´e Abel Gawel Mark Pfeiffer Alexander Liniger John Lygeros3 Roland Siegwart: *AMZ Driverless: The Full Autonomous Racing System.* 05 2019.

[21] *3D camera ZED stereovision by Stereolabs.* https://www.stereolabs.com/.

[22] *Getting started with ZED 3D camera by Stereolabs.* https://www.stereolabs.com/docs/getting-started/.

[23] *Multisense S7 stereo vision camera datasheet.* https://static1.squarespace.com/static/503b9985e4b04953d0f47479/t/5c8aba6c7817f7a11e6c785e/1552595571412/MultiSense_Stereo_brochure03-13-2019_c.pdf.

[24] *E-con Systems 3D camera 'Tara'.* https://www.e-consystems.com/3D-USB-stereo-camera.asp.

[25] *ZED calibration tool.* https://support.stereolabs.com/hc/en-us/articles/360011828773-Can-I-recalibrate-my-ZED-camera.

[26] *Annotation tool.* https://github.com/AlexeyAB/Yolo_mark.

[27] Simek, Kyle: *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix.* http://ksimek.github.io/2013/08/13/intrinsic/, 2013.

[28] Hartley, R. I., Zisserman A.: *"Multiple View Geometry in Computer Vision", 2nd edition.* Cambridge University Press, ISBN: 0521540518, 2004.

[29] *OpenDLV.* https://github.com/chalmers-revere/opendlv.