Bachelor thesis

Visual Detection of Traffic Cones for Autonomous Student Formula

Roman Šíp

Vedoucí: Ing. Jan Čech, Ph.D.

Fakulta Elektrotechnická

Katedra kybernetiky

May 20, 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Šíp Roman**	Personal ID number:	**492273**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Visual Detection of Traffic Cones for Autonomous Student Formula

Bachelor’s thesis title in Czech:

Vizuální detekce dopravních kuželů pro autonomní formuli

Guidelines:

Traffic cones are used to delineate race tracks in competition of the autonomous student formula. The race vehicle is equipped with cameras and a lidar besides other sensors.
1. Select a suitable image-based object detector.
2. Train the detector for traffic cones of given classes (yellow, blue, and orange cones).
3. Adjust the detector to be sensitive enough for distant cones, which appear small in the image.
4. Evaluate the detector accuracy and computational time on real data captured by the vehicle camera.
5. Calibrate the camera with respect to the vehicle coordinate system, so the image detections are mapped to the vehicle surroundings.

Bibliography / sources:

[1] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You only look once: Unified, real-time object detection. In CVPR, 2016.
[2] J. Redmon, A. Farhadi. Yolov3: An incremental improvement. In arXiv:1804.02767, 2018.
[3] A. Dhall. Real-time 3D Pose Estimation with a Monocular Camera Using Deep Learning and Object Priors. Technical Report. ETH Zurich, 2018.
[4] N. Gosala et al. Redundant Perception and State Estimation for Reliable Autonomous Racing. In arXiv:1809.10099v1, 2018.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Čech, Ph.D.   Visual Recognition Group FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **27.01.2022**   Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jan Čech, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt   Student’s signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date................................. ....

Signature

Acknowledgment

I would like to thank my thesis supervisor Ing. Jan Čech, Ph.D for his guidance and expertise. For the entire year, giving me weekly consultations, providing me with direction and giving advice. It was an invaluable experience and I have learned a lot from him.

I also want to thank the entire eForce team for its existence and for giving me the opportunity to participate in developing the autonomous system for the car. I would like to especially thank the members of eForce Driverless, namely: Daniel Štorc, Ondřej Šereda, Michal Horáček, Ondřej Kuban, Tomáš Roun, Hynek Zamazal, Marek Szeles, Marek Boháč, Matěj Zorek and many more.

Lastly, I would like to give special thanks to my mom for financially supporting me in my studies and also for everything else.

Annotation

This bachelor thesis presents a camera based vision system for an autonomous formula participating in the Formula Student Driverless competition. In the competition, the car autonomously drives several laps on a racing track marked by traffic cones. The objective of the visual perception system is to obtain 3D positions of traffic cones, which are visible in the camera image, creating a local map of the scene. The local map of the scene is subsequently used for planning the car's trajectory by its autonomous system. The presented vision system uses a neural network based object detection model YOLOv3 to detect traffic cones in an RGB image, achieving 85.3 mAP at 259 FPS. The traffic cone detections as bounding boxes are converted to the cone base centers, which are subsequently projected by a homography into the car's coordinate system. The proposed perception system achieves localization error of 0.247 meters for traffic cones up at 10 meter distance from the car. The thesis also proposes a method for automatically estimating the homography mapping between the image and the ground plane based on the current scene. This is done by robustly finding the correspondences between image and lidar–based detections of traffic cones.

Key-words: Object detection, YOLO, RANSAC, Camera calibration, Autonomous driving, Formula Student Driverless

Anotace

Tato bakalářská práce prezentuje kamerový systém pro autonomní formuli, která se účastní univerzitní soutěže Formula Student Driverless. V této soutěži, plně autonomně řízené formule jezdí po závodním okruhu vyznačeném pomocí dopravních kuželů. Cílém kamerového systému formule je získat 3D pozice dopravních kuželů viditelných na obrazu z kamery a tím vytvořit lokální mapu oblasti před formulí. Získaná mapa okolí formule je následně použita k plánování trajektorie jízdy autonomním systémem formule. Představený kamerový systém využívá algoritmus pro detekci objektů YOLOv3, který je založen konvolučních neuronových sítích, na detekci dopravních kuželů v RGB obrazu. Detektor kuželů dosahuje 85.3 mAP pří 259 snímcích za sekundu. Detekce kuželů v obraze jsou následně projektovány do souřadného systému formule pomocí předem spočítaného projektivního mapování mezi souřadným systém obrazu a souřadným systémem formule. Představený systém dosahuje průměrné lokalizační chyby 0.247 metrů pro dopravní kužely vzdálené až 10 metrů. Tata práce dále představuje algoritmus pro automatickou kalibraci kamery, pomocí estimace projektivního mapování mezi obrazem z kamery a rovinou tratě. Toto je dosaženo pomocí robustního hledání korespondujících detekcí dopravních kuželů v obrazu a ve scéně.

Keywords: Detekce objektů, YOLO, RANSAC, Kalibrace kamery, Autonomní řízení, Formula Student Driverless

0 Contents

0 List of Figures

0 List of Tables

1 Introduction

This bachelor thesis presents a camera based vision system of an autonomous formula of the team FEE eForce Driverless, participating in the Formula Student Driverless competition. Formula Student Driverless is a world-wide university competition, in which each year teams compete in building an autonomous formula.

This thesis aims to provide a perception system for the purpose of autonomously driving through a race track marked by traffic cones. The task of the vision system is to compute the 3D positions of the cones in real time using a single RGB camera. The positions of the cones seen through the camera are then used by the car's autonomous system to drive.

The thesis has the following structure, in the rest of the section 1, we give context for the competition disciplines the perception system is designed to operate in. We also provide brief overview of the car's autonomous system that the perception pipeline presented in this thesis is built to work within. In section 2 we outline the theoretical background and present the design of each part of the perception system.

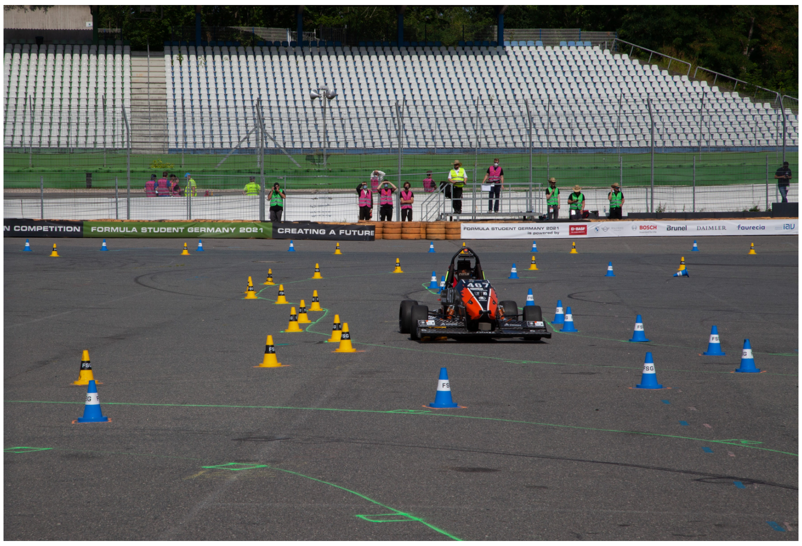


Figure 1: The eForce Driverless formula during the trackdrive discipline at FSD Germany 2021.

1.1 Formula Student Driverless

In the Formula Student Driverless competitions, there are static and dynamic disciplines. The static disciplines are focused on rating the car's design, the documentation of each of the car's systems, and the management of the team. In dynamic disciplines, the physical and autonomous capabilities of the car are tested. In all dynamic disciplines, the car is placed on a track, marked by four types of traffic cones, as depicted in Figure 2. There are four different dynamic disciplines; there is Acceleration, Skid-pad, Trackdrive and Autocross.

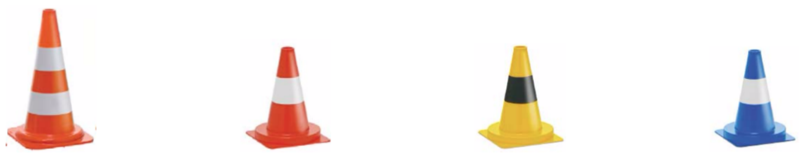


Figure 2: The traffic cone variants used to delineate the track in dynamic disciplines. The big orange with two stripes is used to make the start and end of the race track. Blue and yellow cones are used to mark the left and right sides of the road, respectively.

In the acceleration dynamic event, depicted in Figure 3, the car has to drive through a straight track as fast as possible. The acceleration discipline is designed to test the car's speed capabilities as well as it's ability to steer in a stable way to keep a straight direction.

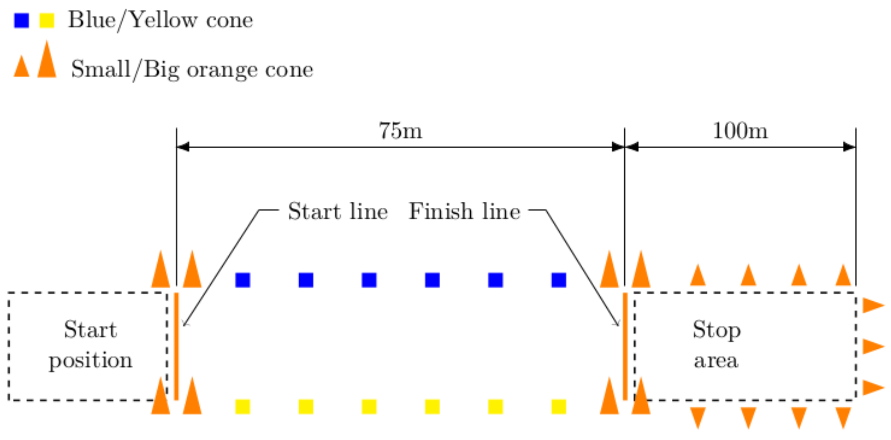


Figure 3: Layout of the acceleration dynamic discipline.

In the skid-pad dynamic event, the car has to drive in an eight-shaped circuit, as depicted in Figure 4. The car first drives two laps on the right side of the track and then transitions and drives two laps on the left side after which it exits the track. The skidpad discipline tests the ability of the autonomous system to correctly track the stage the car is in and execute the following transitions.

The trackdrive and autocross dynamic disciplines are similar to each other. They both consist of driving in a closed–loop unknown track of length up to 1 km. The difference being that in the trackdrive discipline, the car has to drive only a single lap, while in the autocross the car drives 10 laps. By measuring only a single lap time in the trackdrive, the focus is put on the car's ability to drive through a track it has never seen before, while in the autocross discipline the focus is on the car's ability to adapt to the track and optimize its driving driving throughout the 10 lap drive. In Figure 1 you can see FEE eForce Driverless formula driving through a trackdrive track.

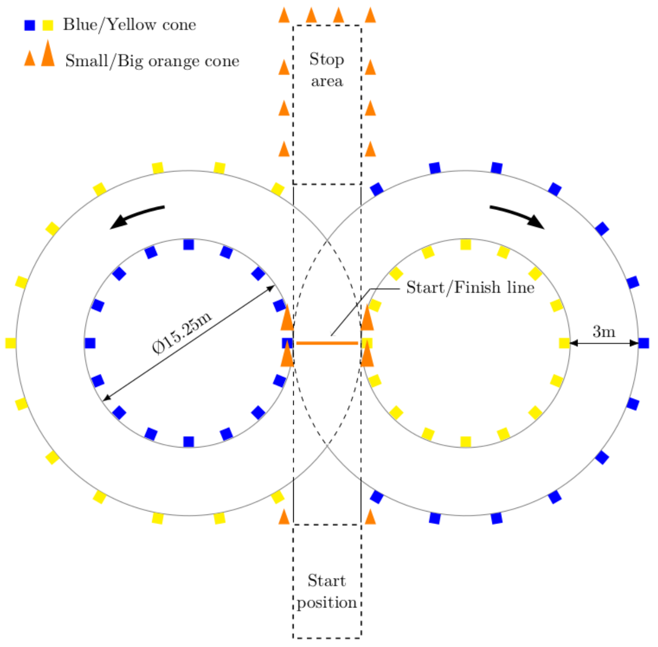


Figure 4: Layout of the skid-pad dynamic discipline.

1.2 Autonomous System

The autonomous system of the car is responsible for everything from observing the environment using sensors, planning a trajectory through the track from the observations and sending the right commands to the car's actuators to correctly follow a trajectory. In Figure 6 you can see the design of our car's autonomous system. All the nodes can be divided into several categories, the Perception node's objective is to build a local map of the scene in front of the car, while nodes such as Reactive-path planning, SLAM and Optimal path-planning use that information to plan the actions the car should take and finally the Motion control node's task is to make the car follow the planned path by sending commands to the car's actuators.

1.2.1 Sensors

For visual sensors, we use a single RGB camera, specifically the ZED1 model. The camera mounted on the main hoop of the car, as can be seen in **??**, provides us with an RGB image of the scene in front of the car, which is then used to detect and localize the traffic cones visible in the image, by the perception system, which is the subject of this thesis. The car also uses a LIDAR, specifically the OUSTER1

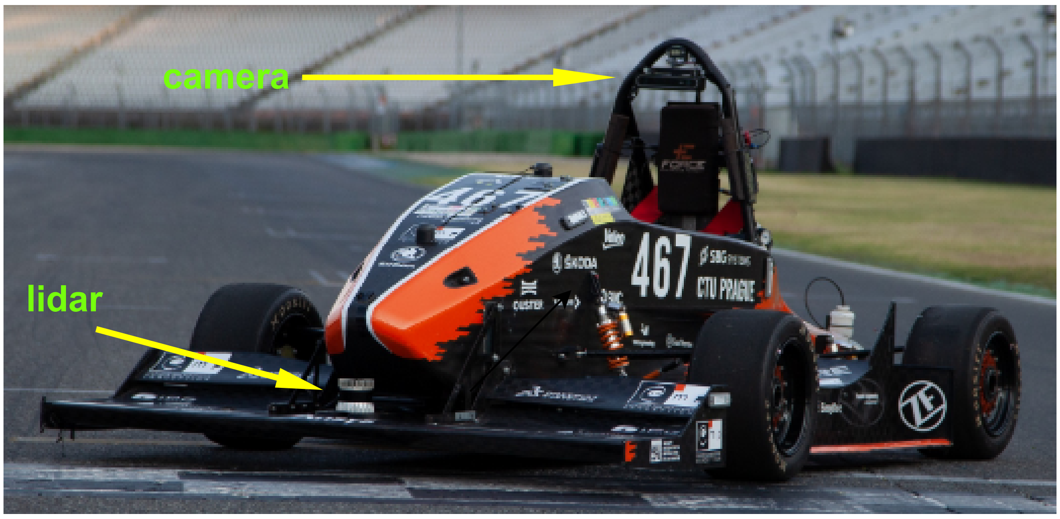


Figure 5: Photo of the FEE eForce Driverless autonomous formula. The lidar is positioned on the front wing and the camera is mounted on the main hoop of the car.

LIDAR, as an alternative vision sensor, although so far it is being used only as a secondary option, mainly for camera calibration, due to the difficulty of discerning the colors of objects, that is needed to differentiate between different types of traffic cones.

The car also contains an INS, or inertial navigation system, providing the Autonomous system with the car's position in the world coordinates, as well as with the car's current orientation and speed. The car's position and orientation are then used in the SLAM node for mapping and localization. The CAN system of the car also provides the autonomous system with readings from several internal car sensors, such as the wheel speed.

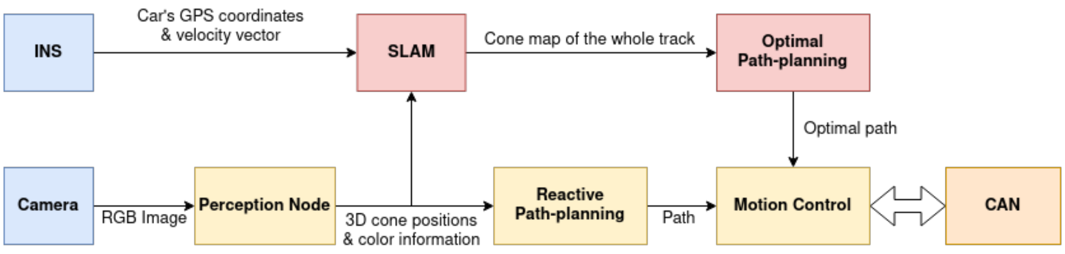


Figure 6: Diagram of the individual components making up the autonomous system of our driverless formula.

1.2.2 Autonomous driving

Even though for each dynamic disciple a specific driving strategy is employed, our autonomous system works in two different modes of driving: the reactive and optimal driving mode. The reactive driving mode means the car is driving through a track it has never seen before. It drives based only on the current picture of the scene in front of the car provided by the RGB camera. A more conservative path and speed profile are computed by the Reactive Path-planning node, which is subsequently passed to the Motion control node that sets the torque and steering angle of the wheels for the car to drive along the planned trajectory. The reactive driving mode is used for the Accelaration, Autocross and the first lap of the Trackdrive dynamic disciplines.

The optimal driving mode requires a map of the whole track to be known, therefore it is only used for the autocross dynamic discipline, since the other disciplines consist of driving through a track only a single time. During the autocross discipline, while the car is driving reactively, the SLAM node of the Autonomous system is using the bird-eye view frames from the perception node combined with the car's world position and velocity vector provided by the INS to build a map of the track. When a map of the whole track is built, the car switches to the optimal driving mode. The map of the track is passed to the optimal path planning [1] node that computes an optimal path and speed profile for the whole track based on the car's physical capabilities.

2 Related Work

2.1 Object detection

The task of object detection consists of detecting objects in an image by predicting bounding boxes around them and correctly determing the class of each detected object. It is a common and popular problem, that has been studied in the fields of computer vision and artificial intelligence for several decades.

Most notable historical approaches, before the rise of neural networks in popularity, include the Viola-Jones [2] and HOG object detectors [3]. The Viola-Jones detector would perform facial detection by splitting the task into two steps. Sliding window would be moved across the image to check for all possible face locations, on which subsequently an AdaBoost face recognition model, trained on predefined features would be used to determine whether the current window contained an image of a face. The HOG(histogram of oriented gradients) detector, worked by dividing the image into a grid of cells. From the pixels within each cell, directional gradients would be computed, conceptually representing edges and regional color transitions of objects. This regional map of gradients, called HOG feature descriptor, would then be used to train a support vector machine [4] classifier, which would then determine if an object is present in a given region of an image.

In 2012, new state-of-the-art result has been achieved in image classification by convolutional neural network(CNN) [5] model AlexNet [6]. Demonstrating the CNNs ability of automatically learning features from large sets of images. Aswell as starting a new trend of research in the fields of computer vision and artificial intelligence focused on neural networks. In 2013, the R-CNN (Regions with CNNs) [7] object detection model set the new state-of-the-art result by achieving mean average precision (mAP) of 58.5% on the VOC-2007 dataset [8] compared to the previous state-of-the-art of 33.7%. The R-CNN detector, split the task into several steps, first using a selective search algorithm, it would generate 2000 proposed regions in the image, subsequently for each region, a 4096 dimensional feature vector was computed, using a pretrained CNN image classification model stripped of its last layer. Subsequently an SVM [4] classifier was trained to detect presence of an object within the proposed region and to correctly predict object's class. Due to a large number of candidate regions to classify per image, each test image took around a minute of computation time, making the model not viable for real time detection. In the following years, there have been several models improving on the R-CNN, notably the SPPNet [9], Fast R-CNN [10] and Faster R-CNN(2015) [11], achieving new state-of-the-art results in terms of both mAP and detection speed, Faster R-CNN achieving 73.20% mAP on the VOC-2007 [8] dataset with detection speed of 6 images per second. Due to splitting the object detection task into two stages, stage of region proposing and stage of region classification, these detectors are commonly referred to as "two stage object detectors" [12].

In 2016, object detection model YOLO(You Only Look Once) [13] was introduced, achieving significantly better detection speeds, 45 frames per second (FPS) for the large version of the model and 155 FPS for the smaller version, while achieving competitive mAP with the state-of-the-art detectors at the time. The YOLO approach, is to frame the object detection task as a single regression problem, on which a convolutional neural network is trained end-to-end, receiving an image as input and outputting positions of bounding boxes and their classes as output. Following the original YOLO paper, there have been several publications improving the original YOLO algorithm, namely YOLO9000 [14] and YOLOv3 [15], aswell as proposing other models, also using a single CNN end-to-end approach to object detection, such as SSD (Single Shot Multi-box Detector) [16], Retina-net [17] and SqueezeDet [18], matching and surpassing the two-stage detectors both in mAP and in detection speed. These models are commonly referred to as "single stage object detectors" [12].

2.2 YOLO

As mentioned in subsection 2.1, YOLO is a CNN based object detection architecture known for achieving a low detection time, while being competitive in terms of accuracy with other state-of-the-art object detection models. Since the first YOLO paper, the original authors published two more papers, YOLO9000 [14] and YOLOv3 [15], introducing several innovations to the model with each version, each time imporoving the models performance. In this section, we first give a brief description of how the original YOLO model was designed and subsequently highlight the key innovations introduced in the second and the third versions of YOLO.

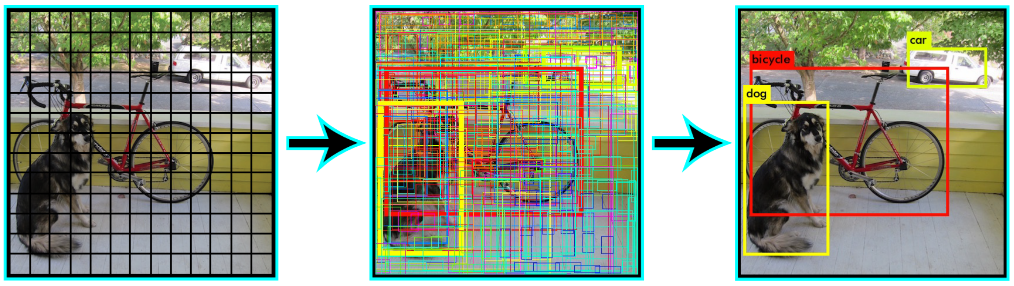


Figure 7: Depiction of how the YOLO neural network splits the image into an $S \times S$ grid of cells, generates predictions for each one and finally filters out detections with low confidence and reduces overlapping detections using non-maximum supression.

The YOLO object detection process can be divided into three separate steps. First, the image is rescaled to a pre-defined, typically square, resolution, eg. 448x448. Secondly, the rescaled image is passed through a convolutional neural network returning the predictions encoded as a tensor of shape $S \times S \times B \times (5 + C)$. Meaning the YOLO network divides the image into a even $S \times S$ grid of cells. For each cell it predicts B bounding boxes, where each bounding box prediction contains probability that the cell contains an object p, center coordinates (x, y), width and height (w, h) and C class probabilities for the object inside the bounding box. Lastly, all bounding box predictions with confidence below a predefined confidence threshold are filtered out and overlapping bounding box predictions of the same class are reduced using the non-maximum supression method to avoid detecting the same object multiple times. The process of splitting the image into a grid, generating detections and lastly selecting the best predictions is depicted on Figure 7.

The YOLO network is trained using a multipart cost function, combining three different losses: object loss, box loss and classification loss. During training, for each ground truth bounding box a cell containing the bounding box center is selected as “responsible” for its detection. From the B bounding boxes, predicted by the selected cell, the bounding box with the highest IOU(intersection over union) with the ground truth box is selected. The object loss incentivices the network to predict $p = 1$ for selected bounding boxes and $p = 0$ for the other bounding boxes. The box and classification losses, representing the predicted bounding boxes and the predicted class probabilities are computed only for the selected boxes.

The YOLO convolutional neural network can be divided into two parts, the backbone and the head. The backbone part, comes first in the network, fulfilling the task of feature extraction from the image. After the backbone comes the head, turning the extracted features into the detection tensor. The backbone part of the network is typically pretrained on an image classification dataset such as ImageNet [19], so that when the network is trained on the task of object detection, it already has a preexisting knowledge about extracting object features from images.

2.2.1 YOLOv2

In the first version of YOLO, height and width of each bounding box is predicted directly, leaving the problem of learning object aspect ratios for the network to learn. In YOLOv2 [14], the concept of anchor boxes was introduced, instead of having the network predict the bounding box size directly, it predicts an offset to a predefined width to height ratio, called an anchor box. Anchor boxes provide the network with prior information about the width to height ratio of objects it is detecting. For example, pedestrians have a width to height ratio of around $1 : 3$, while cars have a width to height ratio of about $3 : 1$. Giving the network this information before training, insentivises it to detect humans with the first and cars with the second bounding box, assuming $B = 2$. The anchor box values are typically computed by clustering width to height ratios of all objects in the training dataset using k-means algorithm.

2.2.2 YOLOv3

In the first and second versions of YOLO, the features extracted by the backbone were through several convolutional layers transformed to form an $S \times S \times B \times (5 + C)$ prediction tensor. In YOLOv3, a method of predicting multiple predictions at multiple different scales, inspired by feature pyramid networks [20], was introduced. Three prediction tensors of three grid sizes would be predicted, doubling the grid size with each prediction tensor. The network would first predict a $S \times S \times B_1 \times (5+C)$, then a $(2 \cdot S) \times (2 \cdot S) \times B_2 \times (5 + C)$ and lastly a $(4 \cdot S) \times (4 \cdot S) \times B_3 \times (5 + C)$ prediction tensor. The three prediction tensors would be merged together into a single tensor of shape $N \times N \times B \times (5+C)$. Adding multi-scale detections increased YOLO’s detection accuracy, especially at detecting smaller objects [15].

2.3 RANSAC

RANSAC or Random Sample Consensus [21], is a general and robust algorithm for fitting mathematical models on data containing outliers. RANSAC works by iteratively selecting random subsets of points of minimal size to determine the model, using the drawn subsets. Subsequently, it computes the model's error for each data point. Data points with errors less than a predefined threshold are considered inliers and points with errors above the threshold are considered outliers. If the number of inliers is high enough, the algorithm recomputes the model on all inliers and terminates, otherwise another random subset is selected and the algorithm continues. In Algorithm 1, overview of the basic RANSAC algorithm is given.

Algorithm 1 RANSAC basic algorithm outline

```
Parameters:
select_size - number of points selected for computing model (typically the minimum number of points needed for model estimation)
max_iters - maximum number of iterations
threshold - error threshold for data points to be considered inliers
min_inlier_ratio - ratio of inlier to dataset size required to terminate the model
  iter = 1
  while iter <= max_iters do
    1. RANDOM SELECT
      Randomly select set of select_size points to be used for fitting a model.
    2. FITTING A MODEL
      Fit a model on select_size selected points.
    3. COMPUTE ERROR
      Compute error for each data point, for the computed model and select the
      points with error less than threshold as a set of inliers.
    4. TERMINATION CONDITION
      If the fraction of the number of inliers over the total number of points in the
      dataset exceeds min_inlier_ratio, fit new model on the set of inliers and
      terminate, otherwise continue back to step 1.
  end while
```

In this work, in subsection 3.5, we present an algorithm that uses RANSAC to automatically find correspondences between two sets of points, in our case detections of traffic cones in image and world coordinates.

3 Method

3.1 Overview

We divide the problem of computing 3D positions of traffic cones from an RGB image in real time into three separate steps: image cone detection, cone center estimation and cone localization. There is another separate step of camera calibration, that is not active during the real-time deployment of the perception pipeline. The perception system architecture is depicted in Figure 8.

In the cone detector part, after an RGB image is received from the camera, the traffic cones visible in the image are detected using the YOLOv3 [15] object detection model trained for traffic cone detection. We propose a scaled-down version of the YOLOv3-tiny neural network architecture designed specifically for the task of real-time traffic cone detection for an autonomous formula. We also propose a method of splitting the image into several crops, in order to maximize the detection rate of more distant cones and minimize the computation time.

In the cone center estimation part, we propose a method for estimating the image point corresponding to the center of the traffic cone's base from its bounding box detection. The estimation of the cone base center is realized using a polynomial fitted on a dataset of bounding box and cone base center image point pairs. We describe the process of creating the dataset by projecting the cone base center point from the cone into the image coordinate system, using a homography computed between a plane defined by handlabeled keypoints in the image and the cone coordinate system defined using 3D model of a traffic cone.

Subsequently, the cone base center image points are projected into the world coordinates using a separetely computed homography matrix, creating a local map of the scene.

Lastly, we propose a method of automatically calibrating the camera by computing a homography matrix between image and world coordinate system. Given a set of cone detections in an image and a set of cone detections in a point cloud, the proposed method finds correct correspondences between the sets of detections using a modified version of the RANSAC[21] algorithm.

We describe each part of the detection system in the following sections, namely cone detector in subsection 3.2, cone center estimator in subsection 3.3, cone localizer in subsection 3.4 and method of automatic camera calibration in subsection 3.5.

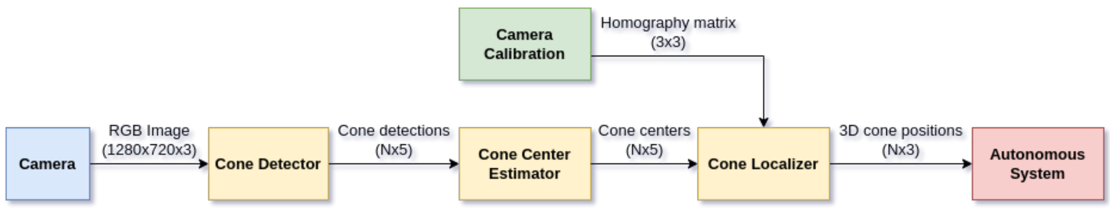


Figure 8: Diagram denoting the individual components of the perception system.

3.2 Image Cone Detection

In this section, we present the design of the image cone detector part of the pipeline. The process of detecting cones in an image is depicted in Figure 9. The image is first split into several crops, which are subsequently batched together and processsed with a YOLOv3 model trained for detecting traffic cones. The bounding box detections in each image crop are then converted into original image coordinates and merged together.

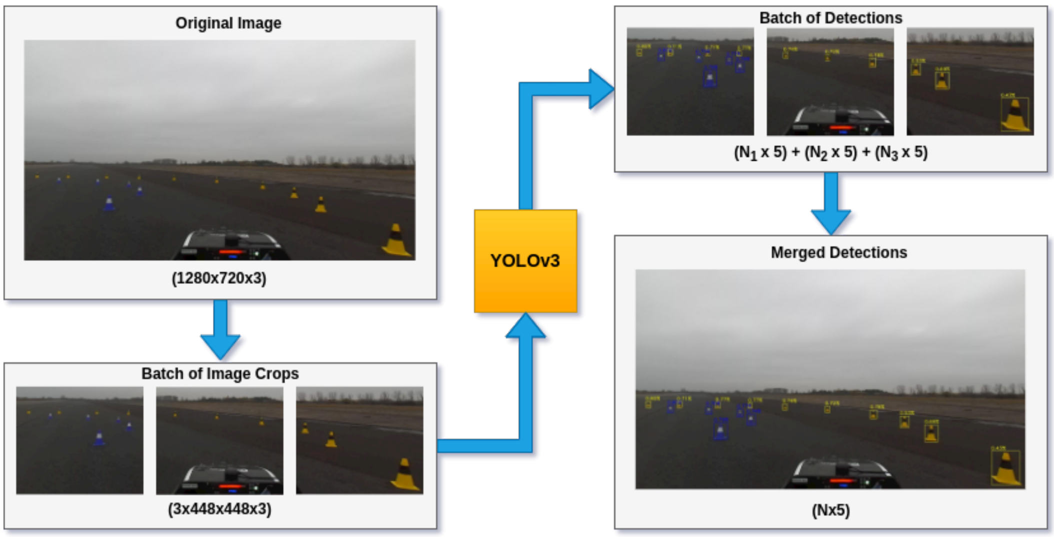


Figure 9: The steps of the process of detecting cones in an image. The image is first split into several sub–crops, on each sub–crop cones are detected using YOLOv3 detection model. Lastly, the detections are merged back into the original image.

3.2.1 YOLOv3 for Detection of Traffic Cones

As discussed in subsection 2.2, YOLOv3 is an object detection algorithm known for exceeding other state–of–the–art models in detection speed, while keeping up in the detection accuracy metrics. Detection speed in autonomous racing is comparable to the reaction time of the human driver in classical racing, making it crucial for the perception system to focus on minimizing its computation time, in order for the autonomous system to be able to control the car at high speeds. For these reasons, we chose YOLOv3 as the object detection algorithm to use for the detection of traffic cones delineating the track.

To train YOLO models for traffic cone detection, we use the FSOCO(Formula Student Objects in Context) dataset [22]. It is a dataset of tens of thousands of hand–labeled images containing traffic cones, that has been crowd–sourced by many Formula Student Driverless teams. After manually discarding the images containing inaccurate labels or traffic cones incompatible with the competition rules, the final dataset we use for training contains 14877 images. The class distribution of blue, yellow, orange and big orange traffic cones is show in Table 1.

id	Class name	Occurences
0	blue cone	11382
1	yellow cone	10579
2	orange cone	2172
3	big orange cone	1949

Table 1: Distribution of cone classes in the FSOCO dataset.

Due to the small number of classes and small complexity of the objects, we expect the task of detecting four types of colored traffic cones to be simpler compared to the detection tasks of benchmark datasets such as COCO [23] or PASCAL–VOC [8], containg many complex classes with high intra–class variability such as a dog and a person. Therefore, a model with fewer parameters than state–of–the–art object detectors should be sufficient for our task of detecting traffic cones. For these reasons, we trained two YOLOv3 based models. First, we trained the YOLOv3–tiny, a smaller and faster version of the original YOLOv3. Secondly, we designed and trained an even smaller model, removing several layers from the YOLOv3–tiny neural network architecture and reducing number of filters in all the convolutional layers, we propose YOLOv3–cones. Comparison of these models, in terms of number of parameters, neural network layers and size of the output tensor can be seen in Table 2 and a full listing of all layers in both models is shown in Table 3. We train each of the two models, on three different resolutions, specifically 224x224, 448x448 and 640x640, giving us 6 models with different detection capabilities in terms of detection speed and detection quality. We can use each of the models in the perception system depending on the requirements by the rest of the autonomous system.

Model	Layers	Parameters	GFLOPs	Output tensor
YOLOv3	333	61539889	154.9	$3087 \times (5+4)$
YOLOv3–tiny	59	8676806	12.9	$2940 \times (5+4)$
YOLOv3–cones	54	2367066	4.4	$1568 \times (5+4)$

Table 2: Comparison of YOLOv3 model architectures in terms of the number of layers, parameters, computational requirements of a single forward pass and the dimensions of its output tensor.

No.	**Layer type**	**Filters**	**Kernel/Stride**
-	input img.	-	-
1	conv.	16	3x3/1
2	maxpool	-	2x2/2
3	conv	32	3x3/1
4	maxpool	-	2x2/2
5	conv	64	3x3/1
6	maxpool	-	2x2/2
7	conv	128	3x3/1
8	maxpool	-	2x2/2
9	conv	256	3x3/1
10	maxpool	-	2x2/2
11	conv	512	3x3/1
12	maxpool	-	2x2/2
13	conv	1024	3x3/1
14	conv	256	1x1/1
15	conv	512	3x3/1
16	conv	256	1x1/1
17	YOLO	-	-
18	route	-	-
19	conv	128	1x1/1
20	upsample	-	-
21	route	-	-
22	conv	256	3x3/1
23	conv	256	1x1/1
24	YOLO	-	-

(a) YOLOv3–tiny

No.	**Layer type**	**Filters**	**Kernel/Stride**
-	input img.	-	-
1	conv	8	3x3/1
2	maxpool	-	2x2/2
3	conv	16	3x3/1
4	maxpool	-	2x2/2
5	conv	32	3x3/1
6	maxpool	-	2x2/2
7	conv	64	3x3/1
8	maxpool	-	2x2/2
9	conv	128	3x3/1
10	maxpool	-	2x2/2
11	conv	256	3x3/1
12	maxpool	-	2x2/2
13	conv	512	3x3/1
14	conv	256	1x1/1
15	conv	256	3x3/1
16	conv	256	1x1/1
17	YOLO	-	-

(b) YOLOv3–cones

Table 3: Full listing of layers for YOLOv3–tiny and YOLOv3–cones convolutional neural network architectures. Each convolutional layers is also followed up by a ReLU and BatchNorm layer.

3.2.2 Image splitting

As mentioned before, instead of feeding the whole image into the YOLOv3 detection model, we take crops of several pre–defined areas in the image and feed them to the detection model instead. Splitting the image this way has several benefits. Firstly, due to the geometry of the scene and the camera pose, we know that all cones will appear in the bottom half of the image, below the horizon, therefore by excluding the top half of the image from being processed by the detection model, we avoid using computational resources on processing parts of the image, that contain zero relevant information for our task. Secondly, due to the small size of traffic cones and the geometry of the scene, most of the traffic cones appear visually small in the image. If we were to feed the whole image to neural network, after downscaling the image to input resolution for the YOLO detector, most of the cones would be represented only by a few pixels on the image, making them hard to detect. Comparatively, by generating several sub–crops of the image, the downscaling of the images is avoided, since the crops fit the required resolution by YOLO or even need to be upscaled. This way, the YOLO network receives all the pixel information from the relevant areas of the image, which would otherwise be lost in the downscaling process. As can be seen in Figure 9, the distant cones appear larger on the image crops, making them easier to detect.

We define an *image crop* as a following tuple:

$$(C_x, C_y, W, H) \in [0, 1]^4$$

Where (C_x, C_y) is the center of crop, W is the width and H is the height. All the values are in the relative image coordinates between 0 and 1, where $(0, 0)$ indicates the top left corner and $(1, 1)$ the bottom right corner of the image. The complete configuration of image crops for the image detector, is defined as a list of *image crops*. We call the image crop configuration of the detector a *split profile*. For example, the *split profile* used in Figure 9 would be defined as:

$$[(0.75, 0.167, 0.5, 0.333), (0.75, 0.5, 0.5, 0.333), (0.75, 0.833, 0.5, 0.333)]$$

To maximize the benefits of splitting the image, namely increase in detection accuracy of distant cones, while keeping the computation cost low, we propose two ways of structuring *split profile* configurations. First, is the *bottom N–split*, evenly splitting the part of the image below the horizon into N crops. The split in Figure 9 is a *bottom N–split* for $N = 3$. Secondly, since the cones in the image are unevenly distributed(most cones appear just below the horizon) placing the *image crops* closer could potentially lead to improved detections of distant cones compared to the *bottom N–split* strategy. We also propose a *horizontal N–split* way of generating *split profiles*. The line of horizontal *image crops* is positioned in a predefined vertical interval closer to the horizon. Moreover, there is an extra image crop containing the full area below the horizon to cover for detecting traffic cones positioned closer to the car, that appear in the very bottom of the image.

Sometimes a cone in an image is positioned on the border between two *image crops*, causing the cone to be detected by two imprecise bounding boxes. To prevent this, we introduce horizontal overlap between the *image crops* of 5% of their width.

3.3 Cone Center Estimation

In this section, we present the design of the cone center estimator part of the pipeline. The goal is to estimate the image point corresponding to the center of the base of each traffic cone from its bounding box, as is depicted in Figure 10. By obtaining the cone base centers in the image, we can subsequently project them into the world coordinates and obtain accurate positions of the cones in the local map of the scene, as is discussed in subsection 3.4. We achieve this by first creating a dataset of bounding box and cone base center image point correspondences. We obtain these pairs by computing a planar homography projection between cone model and image coordinate system. Secondly, we fit an N-degree polynomial on the dataset of bounding box, cone base center pairs. The polynomial is then used for estimating the cone base centers at test time.

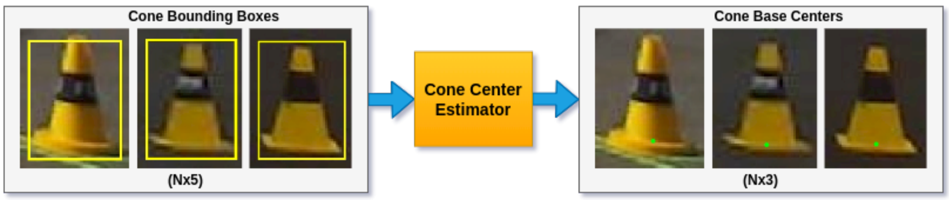


Figure 10: Task of cone base center estimation. From the bounding box information, the goal is to estimate the pixel corresponding to the center of the traffic cone's base.

3.3.1 Cone coordinate system

Since the image point corresponding to the center of the traffic cone base is occluded, it is difficult to label accuratately by a human. To solve this, we hand–label several keypoints on occluding contours of a traffic cone and use them to compute a homography mapping between cone and image coordinates. Using this mapping we subsequently project the cone base center point into the image.

Due to the traffic cone circular shape, we can represent the cone as the cross-section between the traffic cone and a plane perpendicular to the ground, going through the center of the traffic cone, resulting in a shape of isosceles trapezoid, as is visually depicted on the left side of Figure 11. Using our prior knowledge about the traffic cone 3D model, we construct the cone coordinate system. As is depicted on Figure 11, the points on the outer contours on the left and right sides of the cone in the image correspond to the outer left and right lines in the cone coordinate system.

From computer vision, we know that two images of the same planar surface are related by a homography, a projective mapping realized by a 3×3 matrix. Since both the cross-section defining the traffic cone and the image plane are both planar surfaces, a homography mapping between them exists. To compute a homography, at least 4 pairs of corresponding points from both surfaces are required. As depicted in Figure 11, we manually label 8 keypoints in the image, which correspond to the known points in the cone 3D model. This gives us 8 correspondences between the

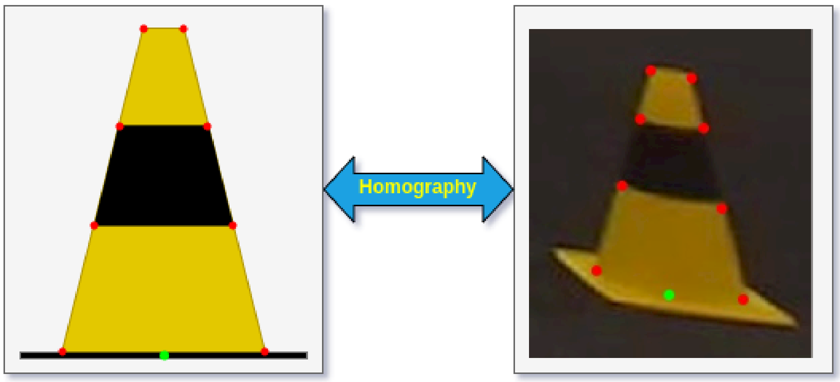


Figure 11: The hand-labeled red points in the image on the right are used to compute a homography mapping between the cone model on the left and the image coordinate system on the right. Subsequently, the green point, representing the traffic cone base center is projected into the image.

cone and image coordinate systems. Using these 8 pairs of corresponding points, we compute a homography between the cone and image coordinate systems, which we subsequently use to project the the traffic cone base center point from the cone to the image coordinate system.

3.3.2 Polynomial regression

For estimating the cone base center from the bounding box, we transform the bounding box into the center of its bottom side, formally:

$$(C_x, C_y, W, H) \rightarrow (C_x + \frac{W}{2}, C_y)$$

As can be seen on Figure 12, assuming the bounding boxes consistently surround the cones tightly, a relationship between the bottom bounding box center and the cone base center image points exists, where the bounding box center is always below the cone base center.

By labeling keypoints and projecting the cone base center from the cone into the image coordinate system for several hundred cone detections, we create a dataset of bottom bounding box center and cone base center image point pairs. Subsequently, we learn the relationship between them by fitting a N-degree polynomial on the task of predicting the correction vector between the bounding box and cone base centers.

Formally, we have two equally sized ordered sets of image points, the bounding box centers and traffic cone base centers:

$$I_{bbox} = \{\mathbf{b}_1, ..., \mathbf{b}_N \mid \mathbf{b}_n \in \mathbb{R}^2\}$$
$$I_{center} = \{\mathbf{c}_1, ..., \mathbf{c}_N \mid \mathbf{c}_n \in \mathbb{R}^2\}$$

We find a polynomial regression model $\hat{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, which minimizes the least squares error of the cone base center predictions, expressed as:

$$\hat{f} = \underset{f}{\operatorname{argmin}} \sum_{n=1}^{N} \|\mathbf{b}_n + f(\mathbf{b}_n) - \mathbf{c}_n\|^2 \tag{1}$$

The process of using the bounding box center point $\mathbf{b}_n$ to predict the correction vector $f(\mathbf{b}_n)$ using the polynomial regression model from Equation 1 is depicted in Figure 12. In the experiments it is shown that the polynomials of second or higher degrees overfit the dataset, while the first degree polynomial or affine function fit the dataset well.

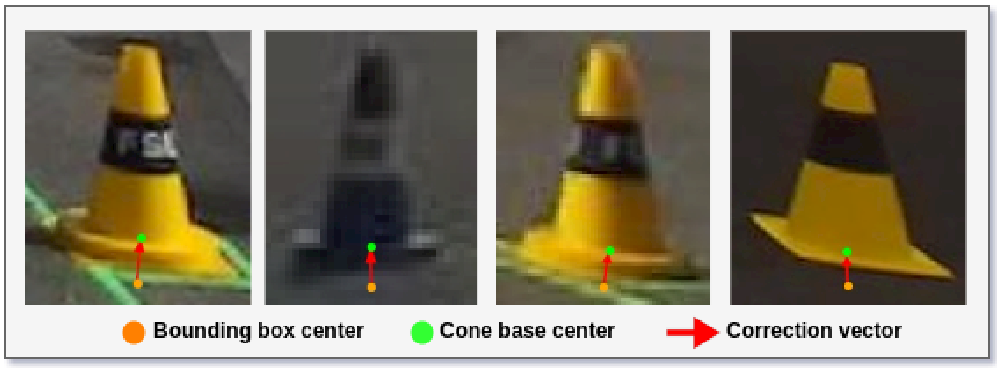


Figure 12: The task of predicting a correction vector for the center of bottom bounding box side to estimate the cone base center image point.

3.4 Cone localization

The final part of the detection pipeline is the Cone Localizer. It receives a set of image points corresponding to the cone base centers, as discussed in subsection 3.3. Since the race track is flat, a homography mapping between the image and the car coordinate system exists. For the sake of this section, we assume the homography matrix has already been computed. The process of computing the homography matrix is described in subsection 3.5. Using the homography matrix, the cone base center image points are projected into the car coordinate system, creating a local map of the scene, as is depicted on Figure 13.

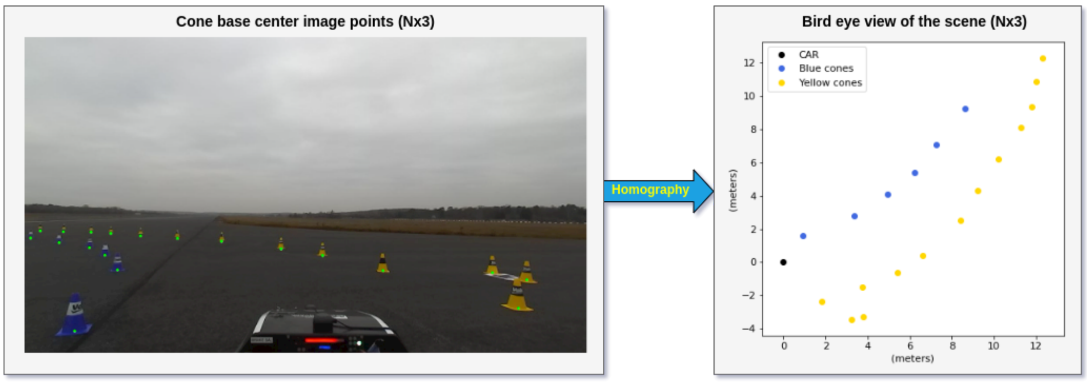

Cone base center image points (Nx3)
Homography
Bird eye view of the scene (Nx3)
CAR
Blue cones
Yellow cones
(meters)
(meters)

Figure 13: The process of projecting cone base center image points into the car coordinates with a homography matrix, creating a local map of the scene.

3.5 Camera calibration

In this section, we explain our approach for computing homography between the image and the world coordinate system. We propose a method of automatically determining correspondences between two overlapping sets of image and world points using a modified version of the RANSAC algorithm. By world coordinates, we mean the local coordinates with the car's position in $(0, 0)$ point, that change with each frame.

Due to the race trtack ground being flat, computing a homography matrix between the ground plane in the image and the ground plane in the world coordinates is sufficient for calibrating the camera. To compute a homography, at least four correspondences between image and world points are needed. In our case, we generate the image points using the image from the camera and detecting traffic cones in it with the perception pipeline parts up to the cone center estimator. We detect the cones in the world coordinate system from a point cloud provided by the car's LIDAR [24].

Formally, we have two sets, a $N \times 2$ set of image points and $M \times 2$ set of world points. The task is to find the correct mapping between the two sets. We assume, that atleast 4 image points and 4 world points correspond to each other, ie. they are the detections of the same cone in different coordinate systems. Formally, we are looking for a mapping between the set of image points I and a set of world points W:

$$m : I \rightarrow W$$

Where if $m(I_i) = W_j$ means, that the i-th image point corresponds to the j-th world point. If $f(I_i) = \emptyset$, then the i-th image point does not have a corresponding world point.

3.5.1 RANSAC for automatic homography computation

The RANSAC algorithm, as described in subsection 2.3, is a general method for estimating mathematical models on data containing outliers. The general idea in using the RANSAC algorithm for the task of automatically finding correspondences is to be iteratively selecting sets of four possible correspondences, computing a homography for the sampled sets and subsequently verifying the correctness of the homography. In order to use RANSAC for this task, several additional steps and adjustments need to be implemented.

In order to be able to solve this task with RANSAC, we generate a dataset of all possible correspondence between the image and world points. The set of all possible correspondences C is defined as the cartesian product of I and W:

$$C = \{(I_i, W_j) \mid (I_i, W_j) \in (I \times W)\} \qquad (2)$$

In each RANSAC iteration, we first randomly sample four correspondences from the set C. Since it can never occur that two image points are in correspondence with the same world point, we filter out all sampled sets containing the same image or world point twice:

$$\begin{aligned} C_{sampled} = \{(I_{i_1}, W_{j_1}), (I_{i_2}, W_{j_2}), (I_{i_3}, W_{j_3}), (I_{i_4}, W_{j_4})\} \\ \text{where} \quad i_1 \neq i_2 \neq i_3 \neq i_4 \quad \text{and} \quad j_1 \neq j_2 \neq j_3 \neq j_4 \\ (I_{i_1}, W_{j_1}), (I_{i_2}, W_{j_2}), (I_{i_3}, W_{j_3}), (I_{i_4}, W_{j_4}) \in C \end{aligned} \tag{3}$$

Secondly, a homography matrix is computed, projecting between the sampled image points and world points:

$$\mathbf{H} = \mathit{compute\_homography}([I_{i_1}, I_{i_2}, I_{i_3}, I_{i_4}], [W_{j_1}, W_{j_2}, W_{j_3}, W_{j_4}]) \tag{4}$$

When computing the error on the dataset of all possible correspondences C, we must first filter out all correspondences that contain any of the image points or world points from the sample correspondences. The set of remaining correspondences C_{remain} is defined as:

$$C_{remain} = \{(I_i, W_j) \mid I_i \neq I_k \text{ and } W_j \neq W_l \text{ for any } (I_k, W_l) \in C_{sampled})\} \tag{5}$$

We subsequently compute projection error for each pair of image and world points from C_{remain}. The correspondences with error below a predefined threshold are considered inliers, rest of the points are considered outliers.

$$E = \{\|\mathbf{H} I_i - W_j\| \mid (I_i, W_j) \in C_{remain}\} \tag{6}$$

$$C_{inliers} = \{C_i \mid E_i \leq \mathit{threshold};\ C_i \in C_{remain}\} \tag{7}$$

If the number of inliers is higher than the highest number of inliers encountered so far, the counter of maximum inliers encountered is updated and the homography matrix H is saved as H_{best}. Followingly, the termination condition is checked. RANSAC terminates if either the maximum number of iterations has been reached or if the current count of inliers is higher than a predefined threshold of `min_inlier_ratio`.

Pseudo-code description of the RANSAC algorithm for finding correspondences is given at Algorithm 2.

Algorithm 2 RANSAC for automatic correspondence finding algorithm outline

Parameters:

`max_iters` - maximum number of iterations

`threshold` - error threshold for data points to be considered inliers

`min_inlier_ratio` - ratio of inlier to dataset size required to terminate the model

```
iter := 1
max_inlier_count := $min(\mathbf{card}(I),\ \mathbf{card}(W))$
best_inlier_count := 0
while iter <= max_iters do
    1. Random select
        Randomly select 4 entries from the dataset of all possible correspondences as
        is outlined in Equation 3
    2. Fitting a model
        Compute homography matrix on the sampled imaged and world points as
        described in Equation 4
    3. Compute Error
        Select a subset of the dataset of correspondences with no overlap with the
        points used for computing the homography according to Equation 5 and com-
        pute projection error for each correspondence it contains as is shown in Equa-
        tion 6.
    4. Select Inliers
        Consider the points with projection error lower than threshold inliers, as is
        described in Equation 7.
    5. Update
        if best_inlier_count < inlier_count then
            inlier_count := best_inlier_count
            $H_{best} := H$
            $Inliers_{best} := C_{inliers}$
        end if
    6. Termination condition
        if best_inlier_count ≥ min_inlier_ratio · max_inlier_count then
            Compute new $H$ with image and world points from $Inliers_{best}$.
            return $H$
        end if
end while
```

3.5.2 Correspondence dataset optimization

When sampling the $C_{sampled}$ sets from the set of of all possible correspondences C, as denoted in Equation 3, we don't make any assumptions about the points. For example, a correspondence between two points of two traffic cones, which are position on the other side of the scene is just as likely to be sampled as a correspondence between points of traffic cones which are both on the left side of the scene. This leads to the RANSAC algorithm running for many iterations in order to find the correspondence selections which lead to the correct solution. To limit the space of all possible correspondences, we describe a more efficient method for generating $C_{sampled}$, taking prior knowledge about the geometry of the scene into account.

Due to the alignment of the lidar and camera mounted on our car, the X-axis in the image and the Y-axis in the world coordinates both represent the lateral dimension of the scene in front of the car. If we assume the detections are precise enough, so that the ordering between the sets sorted by their lateral dimensions is preserved, then we can cut down the number of possible correspondence samplings $C_{sampled}$. The problem of finding the correct correspondences, can be represent as looking for a matching in a bipartial graph. Then the method of randomly sampling from a Cartesian product of the sets is equivalent to trying out all possible matchings. Comparatively, the method of sampling correspondences, that are consistent with the lateral dimension ordering, as is depicted on Figure 14, is equivalent to considering only the non–crossing matchings.

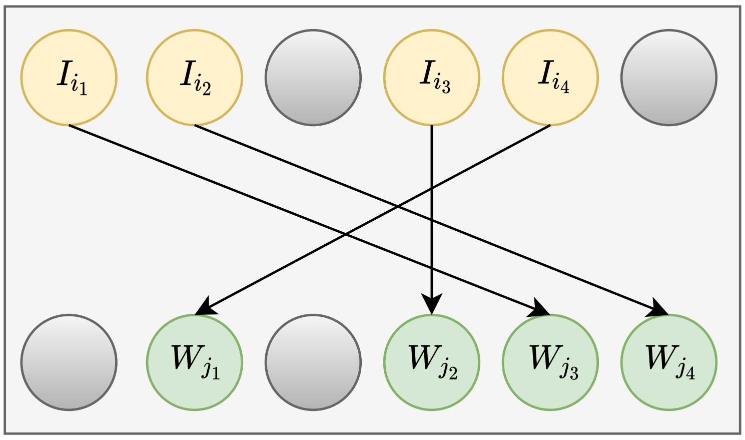
When comparing these two methods of sampling correspondences, we can express the number of possible $C_{sampled}$ sets for the approach of fully random drawing of correspondences as:

$$\mathbf{sample\_count}_{random} = 4!\binom{\mathbf{card}(I)}{4}\binom{\mathbf{card}(W)}{4} \tag{8}$$

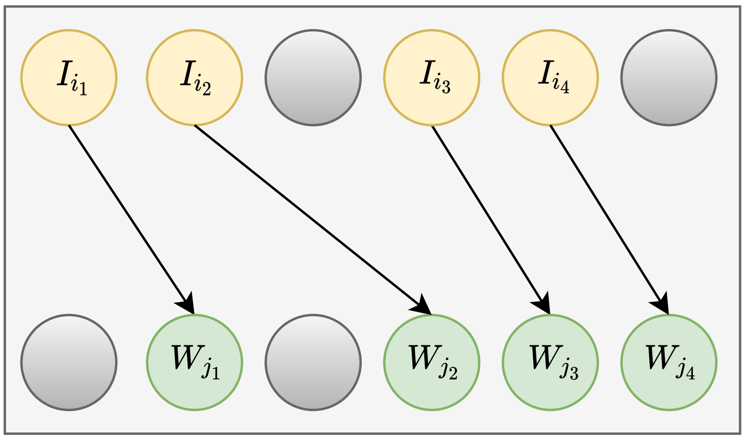
While count of possible $C_{sampled}$ sets, when considering only the non–crossing mappings is only:

$$\mathbf{sample\_count}_{non\_crossing} = \binom{\mathbf{card}(I)}{4}\binom{\mathbf{card}(W)}{4} \tag{9}$$

Meaning that the non-crossing approach eliminates $\frac{23}{24}$ or 95.6% of possible $C_{sampled}$ sets for the RANSAC algorithm to consider. This leads the algorithm both needing less computation time, but also a potentially higher success rate of finding the current mapping, since the eliminated eliminated possibilities could have contained degenerate solutions, leading to a high inlier count, but wrong homography mapping.



(a) example of $C_{sampled}$ with random sampling



(b) example of $C_{sampled}$ sampling when assuming correct lateral order

Figure 14: Comparison of $C_{sampled}$ when sampling the correspondences randomly versus sampling with an assumption of correct lateral ordering, visualized as a bipartite graph. Respecting lateral ordering of the detections leads to a single possible matching due to respecting lateral ordering being equivalent to matching being non–crossing. When not taking the lateral ordering into account, for every two quadruplets of points, there is 12 possible matchings instead of 1. Also the random matching allows for samplings, where points representing detections of cone position on opposite sides of the scene are in correspondence.

4 Evaluation

In this section, we evaluate the individual parts of the perception system. When evaluating each part, we measure their performance in terms of prediction accuracy, reliability and computation time. We compare several configurations of each part in terms of these metrics and discuss their advantages and disadvantages for deployment in the car.

4.1 Image Cone detector

Since the image cone detector is the first step of the perception pipeline, meaning all other parts pipeline rely on its performance. We test several different configurations, in terms of different YOLOv3 models and different *split profiles*. We compute several different metrics relevant to the task of autonomous driving on a custom made test dataset containing images exclusively from our autonomous formula.

4.1.1 Dataset

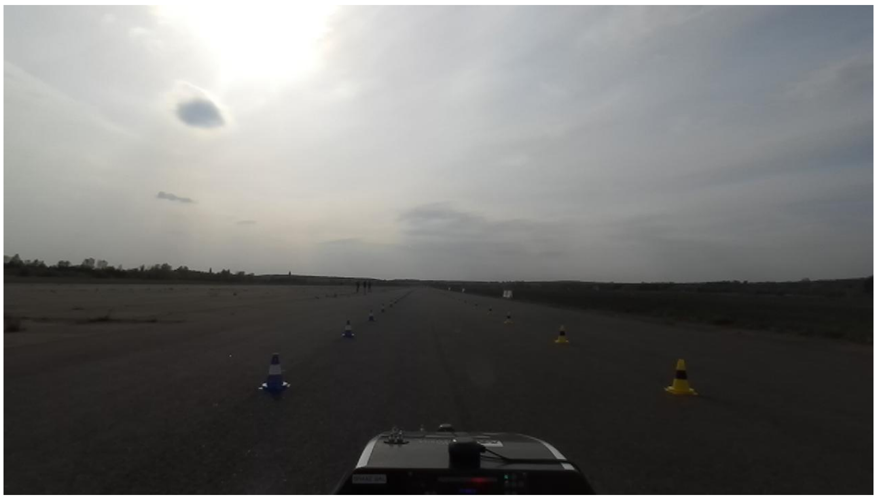
We test the image cone detector configurations on a dataset of 160 images containing 1937 bounding box labels of all four classes of traffic cones. All the images were taken directly from the camera of our autonomous formula either during Formula Student Driverless dynamic events or during test drives. Therefore, all the images are representative of the competition environment the perception system will be deployed in. The images in the dataset contain a wide range of lighting and weather conditions, as is shown on Figure 15.

4.1.2 Metrics

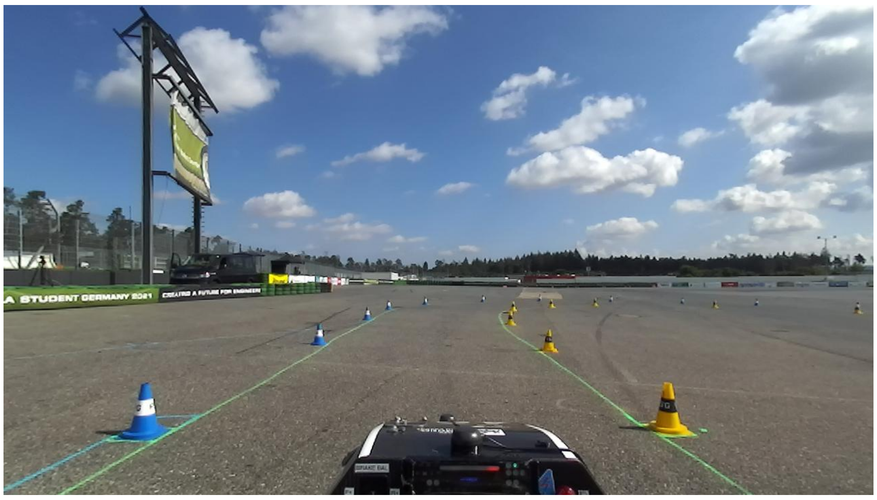
There are several metrics we focus on when testing the cone detector configurations. We test the detectors for detection speed in terms of FPS (frames per second) and accuracy by computing the mAP (mean average precision) at different ranges of bounding box sizes, reflecting the detector's ability of detecting traffic cones at specific distances.

The image cone detector is by far the most computationally intensive part of the perception pipeline due to computing a forward pass of the YOLO object detection model, which contains millions of parameters for each image frame provided by the camera. Computation time of this part determines the detection speed of the whole pipeline. Therefore, prioritizing a configuration with high FPS for deployment is desirable.

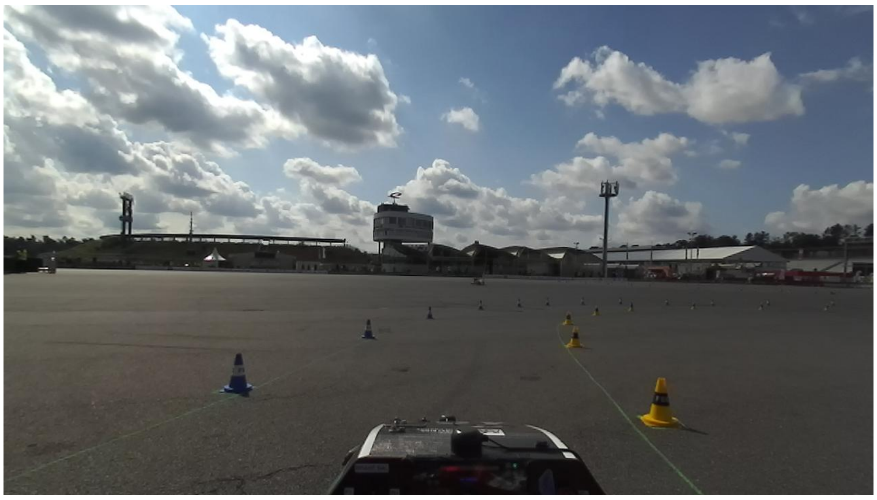
Since the detections from the perception pipeline are used for planning of the path through the track, detecting the traffic cones positioned in closer proximity to the car takes higher priority compared to the cones that are far in the distance. We evaluate the detectors using the mean–average precision (mAP), a widely used



(a) sunny lighting when oriented towards the sun



(b) sunny lighting when not oriented towards the sun



(c) sunny lighting when oriented towards the sun

Figure 15: Camera images from the evaluation dataset with diverse lighting conditions. The third image has been taken only few moments after the second one.

object detection accuracy metric [8]. The mAP metric is computed by taking the area under a precision-recall curve for each class and taking the average across all classes. We consider a detection to be correct, if the IoU (Intersection over union) value of the predicted and the ground truth bounding box is atleast 0.5.

To evaluate the detector's ability to detect cones at different distances, we introduce mAP for minimum bounding box height. Due to the geometry of the scene, the height of the traffic cone's bounding boxes is proportional to their distance from the car. Therefore, by computing mAP only for bounding box labels of a certain minimal height, we are able to evaluate the detector's accuracy at different distances. We use the following notation: $\mathbf{mAP}_{35px}$ stands for mAP for bounding boxes of minimal height of 35 pixels.

4.1.3 Configurations

We evaluate the predictions of several image cone detector configurations. As explained in subsubsection 3.2.1, we trained 2 different YOLO architectures with three different receptive field resolutions, namely, the YOLOv3–tiny and its reduced version YOLOv3–cones. We trained each of the models on three different resolutions: 224×224, 448×448 and 640×640. Moreover, for each of the models, we test its performance with several different *split profile* configurations. Specifically, we evaluate each model, with a *bottom N–split* and *horizontal N–split* split profiles for $N = 1, 2, 3, 4$.

4.1.4 Results

In this subsection, we present the evaluation results of the image cone detector. First, we present the evaluation of all cone detector configurations. Due to the high count of configurations, we will select a few best performing and most notable ones and subject them for further analysis of their performance.

In Table 4, the evaluation of the cone detector configurations is shown. For each cone detector configuration, we compute mean–average precision for the minimum bounding box heights of 0px, 20px and 35px, representing the detection capabilities of traffic cones in all distances. We also measure the detection speed for each configuration.

As can be seen in the evaluation results, for every detector configuration, regardless of resolution and split profile, the YOLOv3–cones architecture achieves up to 80% more frames per second than the YOLOv3–tiny variants do. It can also be observed that the image splitting method leads to an increase in mAP. As expected, a higher mAP increase occurs in the configurations with lower resolution. For most of the configurations, the YOLOv3–tiny performs better at mAP_{35px} compared to YOLOv3–cones, meaning it is better at detecting traffic cones, that are in closer proximity to the car. This can be attributed to the lack of the second detection layer in the YOLOv3–cones architecture, which splits the image into a grid with less cells, focusing more on the detection of objects that appear larger in the image.

Models	**mAP**$_{0px}$	**mAP**$_{20px}$	**mAP**$_{35px}$	**FPS**
(1-bot)-yolov3-cones (224 x 224)	28.3	50.5	75.4	**315.7**
(1-bot)-yolov3-tiny (224 x 224)	28.9	51.8	75.1	263
(2-bot)-yolov3-cones (224 x 224)	49.4	71.3	74.3	230.6
(2-bot)-yolov3-tiny (224 x 224)	53.3	73.9	84	180.3
(2-hor)-yolov3-cones (224 x 224)	50.3	73.4	81.5	181.4
(2-hor)-yolov3-tiny (224 x 224)	53.1	73.1	84.5	139.8
(3-bot)-yolov3-cones (224 x 224)	55.2	70.2	58.5	198.2
(3-bot)-yolov3-tiny (224 x 224)	60.7	76.5	75.3	144.8
(3-hor)-yolov3-cones (224 x 224)	60.8	78.2	83.4	160.3
(3-hor)-yolov3-tiny (224 x 224)	63.4	78.3	83.8	135.5
(4-bot)-yolov3-cones (224 x 224)	55.1	65.9	46.4	175.1
(4-bot)-yolov3-tiny (224 x 224)	63.3	75.7	69.3	120.7
(4-hor)-yolov3-cones (224 x 224)	62.3	79.4	80.3	144.8
(4-hor)-yolov3-tiny (224 x 224)	65.3	79.9	83.2	119.6
(1-bot)-yolov3-cones (448 x 448)	51	76.8	87.8	305.2
(1-bot)-yolov3-tiny (448 x 448)	51.1	76.2	87	217
(2-bot)-yolov3-cones (448 x 448)	66.7	87.1	85.6	205.3
(2-bot)-yolov3-tiny (448 x 448)	69.2	88.2	90.6	150.6
(2-hor)-yolov3-cones (448 x 448)	72.1	87.1	90.1	155.4
(2-hor)-yolov3-tiny (448 x 448)	72	88	89.2	104
(3-bot)-yolov3-cones (448 x 448)	64	80.8	67.2	159.5
(3-bot)-yolov3-tiny (448 x 448)	69.5	87.5	84.3	113.7
(3-hor)-yolov3-cones (448 x 448)	70.2	89.2	87	127.3
(3-hor)-yolov3-tiny (448 x 448)	73.8	88.8	89.9	86.9
(4-bot)-yolov3-cones (448 x 448)	61	74.8	55.9	135.6
(4-bot)-yolov3-tiny (448 x 448)	68.4	85.1	79.3	95.8
(4-hor)-yolov3-cones (448 x 448)	69.8	89.7	87.2	107.2
(4-hor)-yolov3-tiny (448 x 448)	72.7	88.5	90.5	80.1
(1-bot)-yolov3-cones (640 x 640)	61.3	85.3	92.8	259.1
(1-bot)-yolov3-tiny (640 x 640)	59.9	82.9	**93.3**	152.2
(2-bot)-yolov3-cones (640 x 640)	71.4	87	81.9	164
(2-bot)-yolov3-tiny (640 x 640)	64.9	87.7	91	102.3
(2-hor)-yolov3-cones (640 x 640)	**77**	91.1	90.5	119.6
(2-hor)-yolov3-tiny (640 x 640)	65.7	86.2	92.5	74.2
(3-bot)-yolov3-cones (640 x 640)	67.1	82.1	66.4	122.8
(3-bot)-yolov3-tiny (640 x 640)	64.1	89.1	91.2	67.9
(3-hor)-yolov3-cones (640 x 640)	74.5	91.5	91.3	97.4
(3-hor)-yolov3-tiny (640 x 640)	68.5	89.2	91.9	58.1
(4-bot)-yolov3-cones (640 x 640)	62.5	74.3	49.3	104.3
(4-bot)-yolov3-tiny (640 x 640)	65	87.9	89.3	59.4
(4-hor)-yolov3-cones (640 x 640)	74.8	**92.1**	91.4	72.1
(4-hor)-yolov3-tiny (640 x 640)	69.5	90.1	91.9	49.6

Table 4: Evaluation of image cone detector configurations, comparing the detection speed and capabilities of the yolov3-tiny and yolov3-cones architectures. The naming convention is [split profile][YOLO architecture][resolution], where (N-hor) means *horizontal N-split* and (N-bot) for *bottom N-split*. All computations evaluation tests were performed on the car's computer with Nvidia 2070 RTX GPU.

Models		$\mathbf{mAP}_{0px}$	$\mathbf{mAP}_{20px}$	$\mathbf{mAP}_{35px}$	**FPS**
(1-bot)-yolov3-cones	(224 x 224)	28.3	50.5	75.4	**315.7**
(1-bot)-yolov3-tiny	(224 x 224)	28.9	51.8	75.1	263
(2-bot)-yolov3-cones	(224 x 224)	49.4	71.3	74.3	230.6
(2-bot)-yolov3-tiny	(224 x 224)	53.3	73.9	84	180.3
(1-bot)-yolov3-cones	(448 x 448)	51	76.8	87.8	305.2
(1-bot)-yolov3-tiny	(448 x 448)	51.1	76.2	87	217
(2-bot)-yolov3-cones	(448 x 448)	66.7	87.1	85.6	205.3
(2-bot)-yolov3-tiny	(448 x 448)	69.2	88.2	90.6	150.6
(2-hor)-yolov3-cones	(448 x 448)	72.1	87.1	90.1	155.4
(2-hor)-yolov3-tiny	(448 x 448)	72	88	89.2	104
(1-bot)-yolov3-cones	(640 x 640)	61.3	85.3	92.8	259.1
(1-bot)-yolov3-tiny	(640 x 640)	59.9	82.9	**93.3**	152.2
(2-hor)-yolov3-cones	(640 x 640)	**77**	**91.1**	90.5	119.6
(2-hor)-yolov3-tiny	(640 x 640)	65.7	86.2	92.5	74.2

Table 5: Evaluation of selected image cone detector configurations from Table 4.

We have selected detectors from table Table 4 into Table 5, which achieved the best evaluation results compared to other configurations, for further analysis. In particular, the combination of achieving high mAP in all three pixel intervals and high FPS. We also selected the configurations with only a single image split for comparison with the configurations that utilize multiple splits.

We have selected detectors from Table 4, which achieved the best combination of metrics of all the configurations into table Table 5 for further evaluation. In particular, the combination of achieving both high mAP in all three pixel intervals and high FPS. We also selected the configurations with only a single image split for comparison with the configurations that utilize multiple splits.

In Figure 16, we can see a comparison of the selected cone detector configurations in terms of their mean–average precision based on a minimum bounding box height of detections. We can see that the configurations using the YOLOv3–tiny architecture generally achieve the same results of mAP compared to the YOLOv3–cones detectors. However, when the detection speed is taken into consideration, as is visualized in Figure 17, we see that the YOLOv3–cones configurations achieve in every case higher frames per second with similar mAP scores, outperforming the YOLOv3–tiny configurations.

When choosing the optimal cone detector configuration to use in competition, we are looking for a middle ground between minimizing detection speed and maximizing capability and reliability of detecting traffic cones positioned up to 20 meters away from the car. The **(2–bot)–yolov3–cones** (448 $\times$ 448) and **(1–bot)–yolov3–cones** (640$\times$640) detectors both achieve a frame rate of over 200 fps, while achieving high detection quality for all distances. The first detector being able to detect cones that are further away, while the second detector is more stable at detecting the cones closer to the car. To decide, with which detector the autonomous system performs better, further testing of both detectors in experiments with the whole autonomous system must be conducted to see, which detector yields better results.

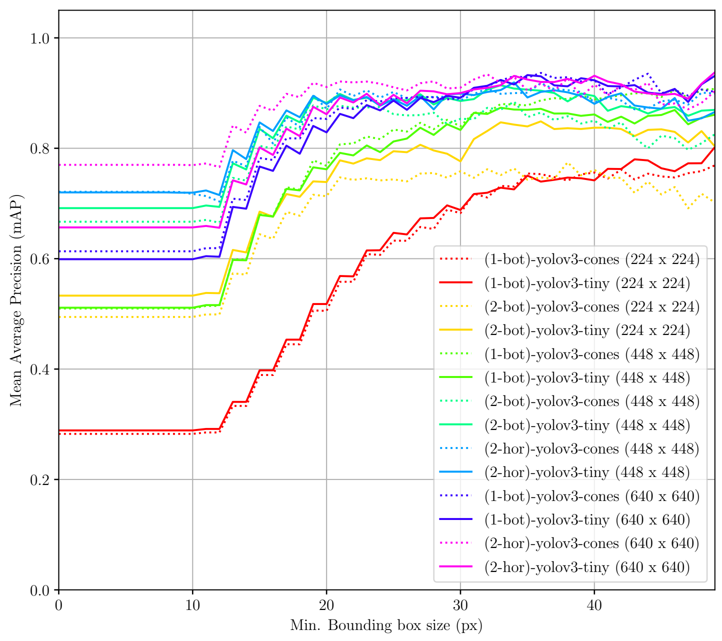


Figure 16: Comparison of selected image cone detector configurations of mean–average precision based on minimal bounding box height. Comparing the ability of the detectors to detect traffic cones as a function of distance from the car.

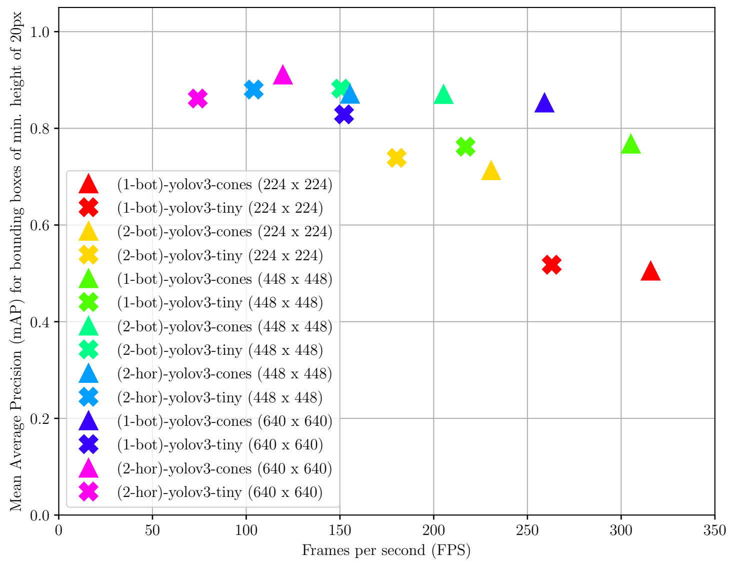


Figure 17: Comparison of selected image cone detector configurations. Comparing the mean–average precision of the detectors, for bounding boxes of minimal height of at least 20 pixels over the detection speed in frames per second.

4.2 Cone center estimator

The cone center estimator, as described in subsection 3.3, estimates the image point corresponding to the traffic cone's base center from its bounding box. The cone base center image points are subsequently projected into the car's coordinate system, obtaining the accurate positions of the traffic cones corresponding to their centers. In this section, we evaluate the accuracy of the polynomial regression model for estimating the cone base center from its bounding box.

4.2.1 Dataset

By the process of labeling traffic cone keypoints, a homography between the cone and the image coordinate system is computed. Subsequently, by projecting the center of the traffic cone's base into the image coordinate system, as is described in subsubsection 3.3.1, we create a dataset of bounding box and traffic cone base center pairs. The dataset consists of 64 images of traffic cones from all positions in the image. For evaluation, we split the dataset into train and test sets, consisting of 48 and 16 pairs, respectively.

4.2.2 Polynomial Regression

We train polynomials of first and second degree on the dataset of bounding box and cone base center pairs for the task of predicting the correction vector for the bounding center. Reliability of the predictions is essential, because a single estimation of the cone's position in the middle of the road, instead of on its edge, can lead to a failure of the car's ability to drive through the track. Therefore, when evaluating the cone base center estimation, we not only compute the mean error, but also focus on the maximal error across the test dataset.

Model	RMSE$_{train}$	RMSE$_{test}$	Max-RSE$_{train}$	Max-RSE$_{test}$
pol-1	1.405	1.592	3.496	4.599
pol-2	1.071	2.192	2.631	8.335

Table 6: Comparison of errors of first and second degree polynomials fitted on the regression task of correcting bounding box center to traffic cone base center, as is visualized on Figure 12. RMSE stands for Root Mean Squared Error and Max-RSE for Max Root Squared Error. All values are in pixels.

Table 6 contains the evaluation of the regression models using first degree and second degree polynomials. We can see that the polynomial of the second degree overfits on the training data, achieving lower train errors and higher test errors compared to the first degree polynomial. The first degree polynomial, achieving a mean test error of 1.6 pixels and maximal test error of 4.6 pixels seems to be fitting the dataset decently well. In Figure 18, you can see the visualization of the first and second degree polynomial estimates on the test data. In Figure 19, cone centers estimation using the first degree polynomial model is depicted.

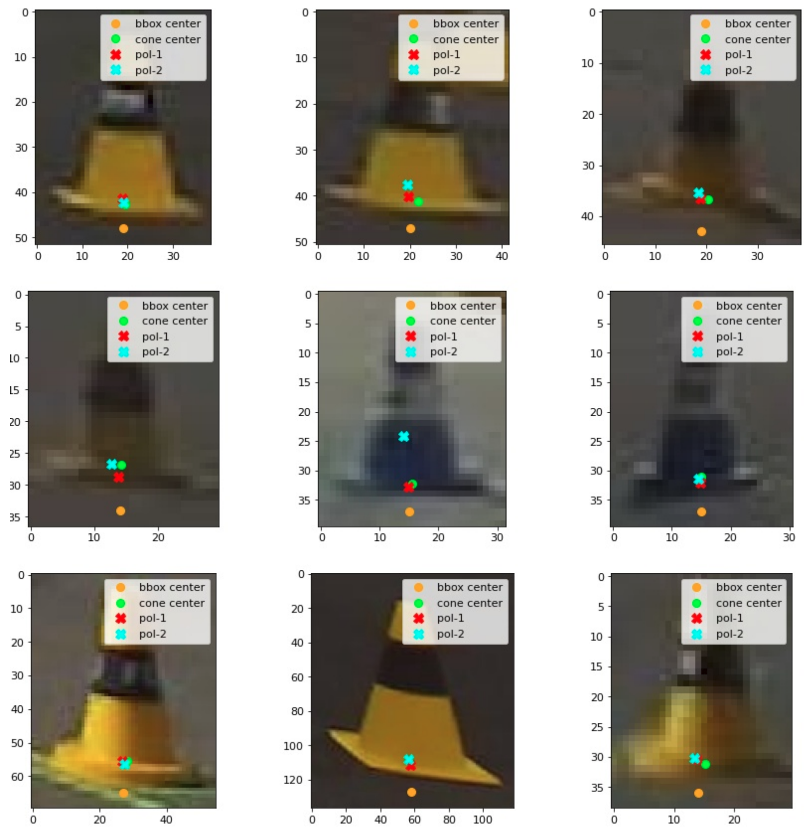


Figure 18: Visual comparison of estimating image point corresponding to the traffic cone base center from it's bounding box using polynomials of first and second degree. The images are cropped around the cones only for visualization purposes, the estimation takes as input the position of the cone in the full image coordinates, taking the geometry of the scene into account.

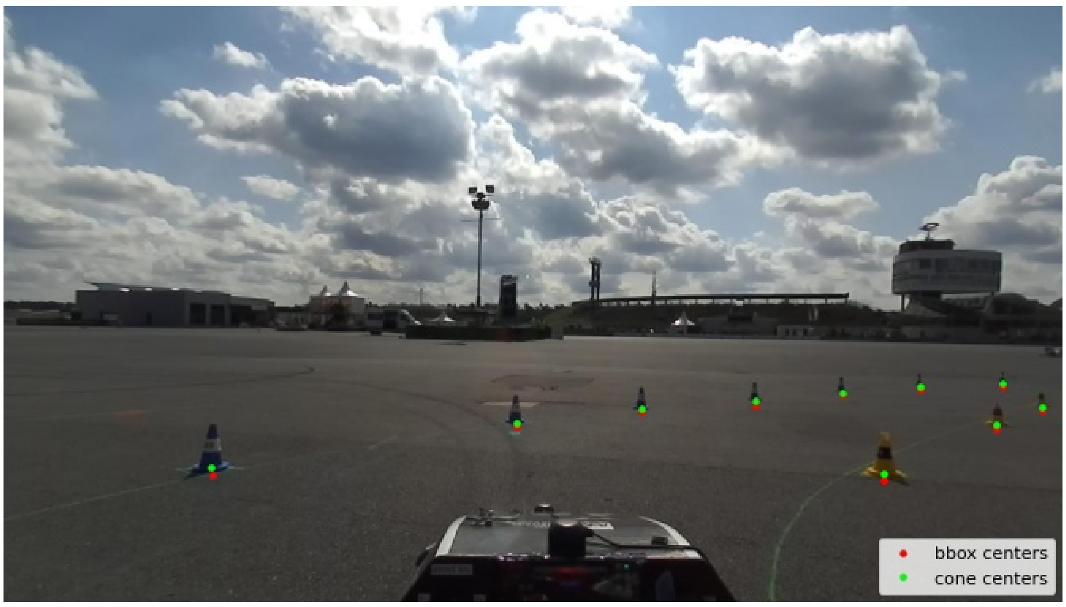


Figure 19: Depiction of cone base center estimation in the entire image.

4.3 Camera calibration

In this section, we evaluate the algorithm for camera calibration by automatically finding correspondences between cone detections in the image and in the world coordinates of scene. We compare the two ways of sampling correspondences in each RANSAC iteration, the random and the non–crossing, as presented in subsubsection 3.5.2. We also compare the performance of the algorithm, when computing the reprojection error in world coordinates (meters) anand in image coordinates (pixels).

When evaluating the RANSAC algorithm variants, we used the same parameter values for each one, the only difference being in the threshold value based on if we are measuring the homography projection error in world or image coordinates. We used the following parameter values:

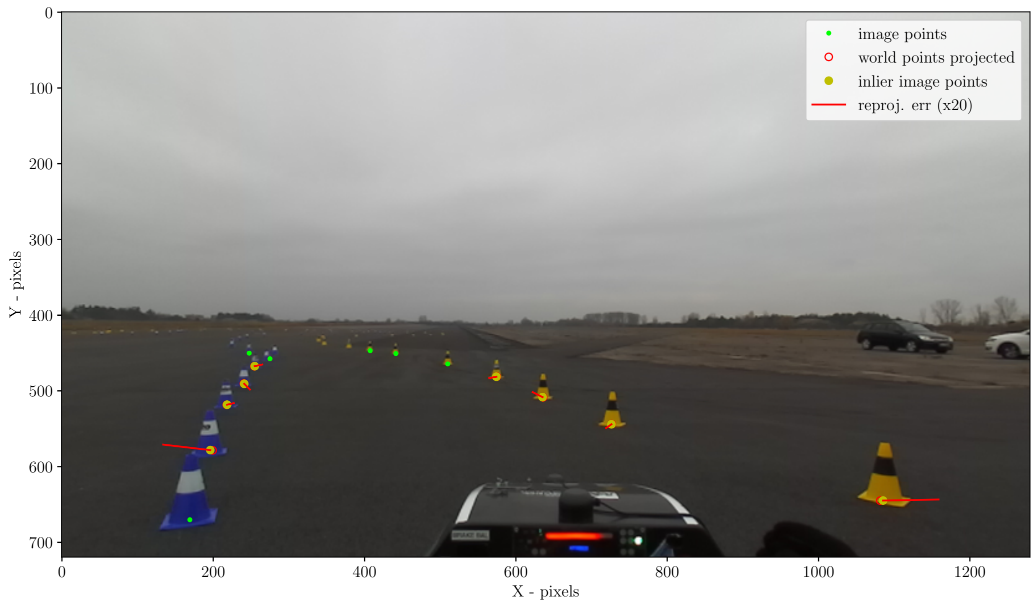
$$\begin{aligned}\texttt{max\_iter} &= 100000\\ \texttt{threshhold(px)} &= 20\\ \texttt{threshhold(m)} &= 0.6\\ \texttt{min\_inlier\_ratio} &= 0.85\end{aligned}$$

For evaluation, we collected 8 different pairs of point clouds and camera images of scenes. We split the data into 5 scenes for the evaluation of the RANSAC algorithm for automatic camera calibration. We have hand–labeled correspondences in the remaining scenes to be used as a test set to verify the correctness of the solutions found by the algorithm. We evaluate the correspondence finding algorithm variants by running them on each of the 5 evaluation scenes and for each computed solution, we evaluate its projection error on the test set scenes. We consider the found solution to be correct, when the mean projection error of the found homography is below 0.5 meters on the test set.

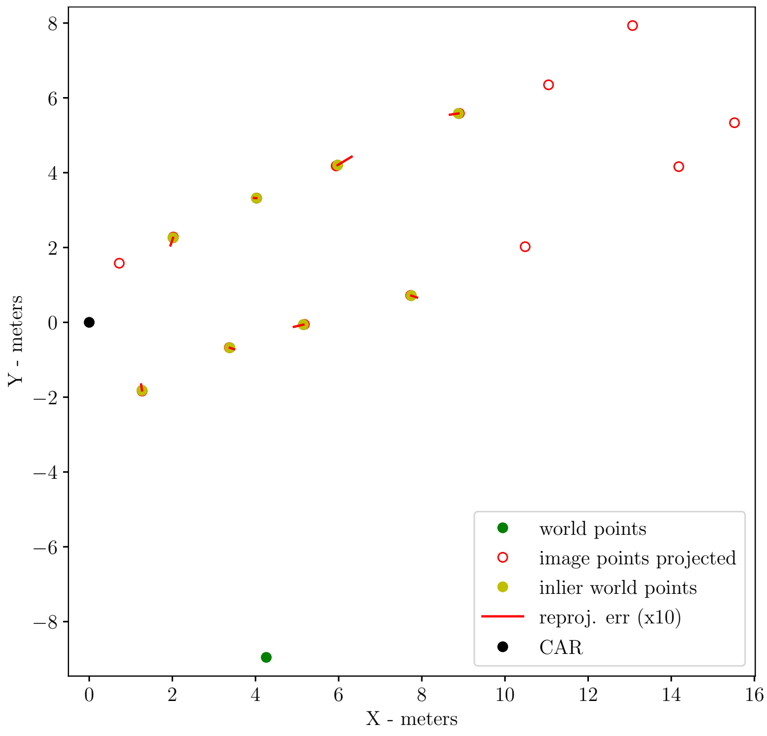
Configuration	Success rate	Avg. iterations	Avg. time
non-cross (m)	100%	13193.2	6.37
non-cross (px)	60%	33086.6	15.07
random (m)	40%	80768.4	136.78
random (px)	60%	84917.2	143.29

Table 7: RANSAC comparison

In Table 7, the evaluation results of each of the algorithm variants are shown. As expected, the non–crossing variants needed fewer iterations and took less computation time to find the same or better solution, compared to the RANSAC variants using random sampling. Since the evaluation set contains only so few examples, it is difficult to make definitive judgements about the reliable accuracy of the algorithm. Figure 20 contains a visualization of a found solution by the **non–cross (m)** RANSAC variant.



(a) automatic correspondence finding results in image coordinates



(b) automatic correspondence finding results in world coordinates

Figure 20: Visualization of homography reprojection in image and world coordinates. The homography was computed using the algorithm for automatic traffic cone detection correspondence finding for camera calibration. The mean reprojection error in this scene is equal to 0.13 meters or 1.24 pixels.

4.3.1 Localization Error

In this subsection, we evaluate the localization error of the traffic cones. To evalute the accuracy of the cone detections projected by the homography into the world, we use the traffic cone detection from lidar point cloud as the ground truth. The lidar detector only detects traffic cones up to around 13 meters away from the car, therefore we only able to evaluate the localization accuracy of traffic cones up to that distance.

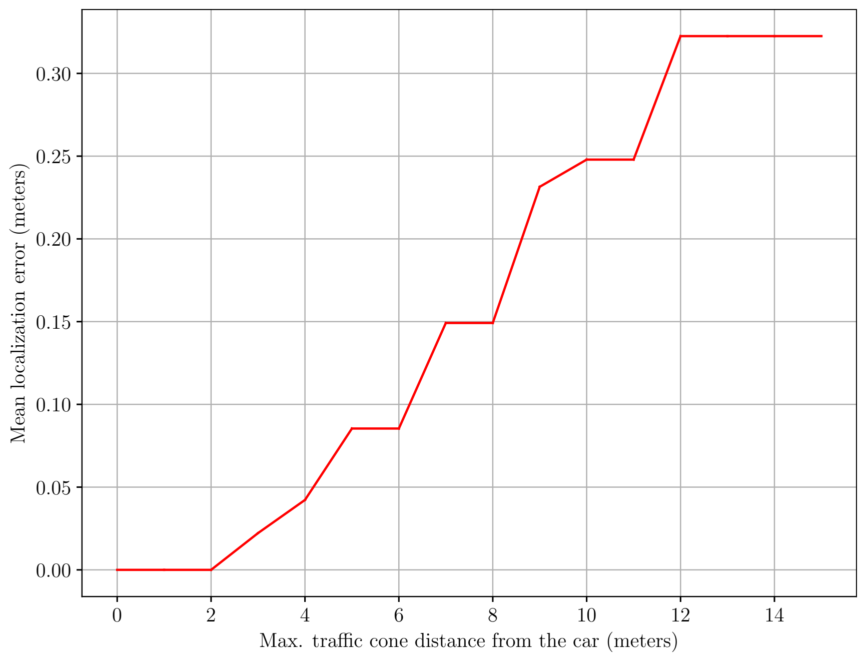


Figure 21: Evaluation of the localization error of the perception pipeline. The plot visualizes the mean localization error of all detections from the test set for a given maximum cone distance from the car. After 12 meters, there no more traffic cone detections in our test set to measure the error, therefore the error is unknown for more distant detections.

In Figure 21 we see, that the localization error of the traffic cones rises with the distance of the traffic cones from the car. Even though the localization error for traffic cone detections, which are more distant than 12 meters is unknown, the detection of cones up to 12 meters covers the most important distance range for accurate detections. The perception pipeline achieves mean localization error of 0.247 meters for traffic cone detections up to 10 meters away from the car and error of 0.323 meters for detections up to 12 meters.

5 Conclusion

In this thesis, we presented a perception pipeline for predicting 3D traffic cone positions using a single RGB camera. The perception pipeline was made up of three parts, the image cone detector, the cone center estimator and the cone localizer.

For the image cone detector, we presented a new YOLOv3 based neural network architecture called YOLOv3–cones, designed specifically for the task of traffic cone detection. We also presented and evaluated a method of splitting the image into several sub-crops to minimize the computational detection cost and maximize the accuracy of detecting distant traffic cones.

For the cone center estimator, we presented a method, which from a traffic cone's bounding box, estimates the image point, which corresponds to the center of the traffic cone's base. The method utilized fitting a polynomial regression model on dataset of bounding box and traffic cone center image point correspondences. The image points corresponding to the traffic cone base centers are subsequently projected by the cone localizer to the car's coordinate system, building a local map of the scene.

Finally, we presented an algorithm for automatically estimating the homography mapping between the image and the ground plane, by finding the correspondences between traffic cone detections in the scene in the car's coordinates and in the image coordinates.

In the future, the perception system needs to be tested in competion–like experiments, working while integrated with the rest of the car's autonomous system. Some future developments of the system would include further optimization of hyperparameters of the YOLOv3 architecture, attempting to achieve higher detection speed and accuracy.

5 References

[1] Michal Horáček. Finding the Fastest Trajectory for Autonomous Student Formula. Bachelor's thesis, Czech Technical University, 2022. To be defended.

[2] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 02 2001.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[4] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152.

[5] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.

[10] Ross Girshick. Fast r-cnn, 2015.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[12] Ch Murthy, Mohammad Farukh Hashmi, Neeraj Bokde, and Zong Woo Geem. Investigations of object detection in images/videos using various deep learning techniques and embedded platforms—a comprehensive review. *Applied Sciences*, 05 2020.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[14] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[18] Bichen Wu, Alvin Wan, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving, 2016.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[20] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[21] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[22] David Dodel, Michael Schötz, and Niclas Vödisch. FSOCO: The formula student objects in context dataset, 2020. arXiv preprint arXiv:2012.07139, Dec 2020.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[24] Daniel Štorc. Detection of Traffic Cones from LiDAR Point Clouds. Bachelor's thesis, Czech Technical University, 2022. To be defended.