A Perception Pipeline for an Autonomous Formula Student Vehicle

Diogo Laranjeiro Amaro Serrão Morgado
Instituto Superior Técnico, Lisboa
diogo.s.morgado@tecnico.ulisboa.pt

***Abstract*—Autonomous vehicles need to accomplish all the tasks a human would while driving a common vehicle, which includes being able to perceive and comprehend the environment around the vehicle, in real-time and using a combination of high-tech distance sensors and cameras. Following the trends of the automotive industry world, Formula Student, one of the Europe's most established educational engineering competition, introduced the Driverless class, which challenges students to build a high-performance autonomous race car. In such competitions, the tracks through which the autonomous car should navigate are unknown and delimited by different colored cones. This thesis presents a perception pipeline for a Formula Student car that exploits the best features of each perception sensor to identify the cones and, using custom Convolutional Neural Networks, classify their color. Furthermore, due to the simplicity of the classifiers used and in order to mitigate possible misclassifications, it was developed a tracking algorithm that chooses the cone's color by taking into account the color of previous observations. An ablation study was conducted on the methods used to identify the cones and, both the classifiers and the tracking module, were evaluated individually. Results showed that the proposed perception pipeline is able to accurately detect and classify the cones in real-time, even when running the pipeline on CPU, something that proved not to be possible using other deep learning object detectors.**

***Index Terms*—Cone detection, sensor fusion, classification, perception, convolutional neural networks, tracking, autonomous driving, Formula Student.**

I. Introduction

The increased safety in addition to an improved comfort promised by autonomous vehicles leads the automotive industry to use a big portion of their investments for the development of autonomous driving. When taking the human out of the equation, the vehicle must be able to respond to unknown situations in order to ensure a reliable system. Therefore, it needs to accomplish all the tasks a human would while driving a common vehicle. This includes perceiving the environment, estimating its position, predicting what other road users will do, planning a trajectory accordingly and giving the correct acceleration and steering input to follow that trajectory. The investments made by the industry led us to the current state-of-the-art in autonomous driving, with several companies already having their autonomous vehicle prototypes being tested on real-life scenarios without any human interference.

A key part when dealing with any kind of autonomous driving problem is being able to perceive and comprehend the environment around the vehicle, in real-time and using a combination of high-tech distance sensors and cameras, combined with state-of-the-art perception algorithms. This is still a very challenging task in autonomous vehicles due to the extremely low acceptable error rate, as it is a crucial task to ensure the safe and reliable operation of the vehicle by being the source of information used by the path-planning and decision-making algorithms that dictate what the vehicle should do or where it should go.

A crucial aspect for an autonomous vehicle to reach its full autonomous capabilities is the ability to operate the vehicle close to its limits of handling. When it comes to new technologies in the automotive industry, racing has often played a key role in fuelling innovation and pushing cars to the limit of what is possible. Autonomous racing has proven to be an essential platform to develop, test and validate new technologies under challenging conditions. It provides a unique opportunity to test autonomous driving software such as redundant perception pipelines, failure detection algorithms and control in challenging conditions. Competitions, by themselves, allow researchers and developers to test the applicability of different solutions and their robustness. Racing competitions, in particular, present additional challenges related to computational speeds, power consumption, and sensing.

A. Formula Student Competition

Formula Student[1] is the Europe's most established educational automotive engineering competition that challenges university students from worldwide top universities to design, manufacture, build and test electric and combustion race cars following a strict set of rules [12] that prioritize safety. These students are then challenged to race their cars and compete against other teams in international competitions. Backed by the automotive industry and by high-profile engineers, Formula Student Germany (FSG), one of the most reputable competition organizers, introduced, in a constant effort to follow the trends of the automotive industry world, a new competition class, the Formula Student Driverless (FSD) class. This is a new competition class that challenges students to build an high-performance autonomous race car. In this class, the autonomous student-developed prototypes must be able to compete in different events fully autonomously without any human interaction and without prior knowledge of the track's layout.

In the FSD class, the track boundaries are delimited by blue cones on the left and yellow cones on the right, small orange cones delimit stopping zones and big orange cones mark timekeeping zones. The shape, dimensions and color pattern of these cones is regulated by the FSG rules [12]. Since the layout of the track is unknown, in order to safely navigate through the unknown environment the perception pipeline implemented in the prototype must be able to identify

[1]https://www.formulastudent.de. Accessible on July 30, 2021.

the cones' position and color using the data retrieved from the available perception sensors. Having detected the cones that delimit the track, it is then necessary to compute a valid path between the track boundaries, something that is done by the path planning pipeline. Finally, in order to navigate through the track using the computed path one needs to determine a speed target and a steering input, which is performed by the control pipeline.

B. Motivation

Formula Student Técnico[2] (FST) is the team that represents Portugal and Instituto Superior Técnico (IST) in international Formula Student competitions. FST has been developing prototypes for this competition since 2001, with ten prototypes developed so far and proven results in the most prestigious competitions of Europe. In 2019 the team, with more than 60 members from different engineering courses, decided to embrace a new challenge: empower the previous electric prototype (FST09e) with autonomous driving capabilities and compete in the FSD class in the summer of 2020. Initially composed of only eight students, the autonomous systems team successfully developed, in less than one year, a complete autonomous pipeline [13], comprising perception, estimation and control pipelines.

Unfortunately, due to the COVID-19 pandemic, the formula student competitions that FST was going to compete in, were canceled. Because of this, an opportunity for extensively testing the developed algorithms emerged. As this was the team's first year to develop an autonomous car, it would be expected to encounter problems during this testing phase, which turned out to be the case. Even though the different pipelines were extensively tested in simulation and using synthetic data, one pipeline that was notoriously hard to test on simulation was the perception pipeline. Due to this, several problems related to the perception pipeline and its computational expensiveness arose during the testing phase. In order for the car to be competitive, it needs to be able to navigate through the track at a very fast pace, which means that an accurate, robust and fast perception pipeline that takes advantage of all the available sensors must be developed.

II. Related Work

In this chapter, a literature review is performed on the existing methods used in the perception pipelines implemented by Formula Student Driverless (FSD) teams. Depending on the sensors available, different approaches can be considered: those that solely rely on either cameras (one or more), described in Sec. II-A, or LiDARs, described in Sec. II-B and those that fuse the information of both sensors, described in Sec. II-C.

A. Camera Only Approaches

These methods are highly dependent on the number and type of cameras used (monocular *vs.* stereo), as depth estimation on stereo cameras can be easily obtained. Regardless of the type of camera used, these implementations start by detecting the cones and estimating their correspondent color, either by using deep learning and state-of-the-art object detectors or using more classical computer vision techniques. Regarding the deep learning techniques, some teams [14], [19] chose to use well-known object detectors such as YOLOv2 [23], YOLOv3 [24] and their correspondent lightweight versions, YOLOv2-tiny [22] and YOLOv3-tiny [1], respectively, while others [28] opted to use custom object detectors, based on the YOLOv3 architecture, to detect the cones, and a custom 7-layer CNN to estimate their correspondent color. By training these real-time and powerful object detectors to detect cones from four different classes (blue, yellow, orange and big orange), the teams are able to accurately get bounding boxes around the detected cones, along with the confidence scores for each detection.

In order to estimate the 3D position of the detected cones, there is a consensus that, for monocular setups, using a keypoint regression alongside the Perspective-n-Point (PnP) [11] algorithm is the best approach. The known cone's shape and size are exploited to perform a keypoint regression and find specific feature points, on each of the detected bounding boxes, that match their 3D correspondences, whose locations can be measured from a reference frame. A classical computer vision approach [19] was initially explored in order to extract these keypoints. First, the RGB bounding box of the detected cones is converted to the LAB color space and the "b" channel, which represents the color axis from blue to yellow, is extracted. Then, an Otsu threshold [21] is used to obtain a binary image where, performing contour fitting, three vertices from the top region of the cone can be identified. These three vertices are then extended, using the known dimensions of the cone, to obtain further four more keypoints. This approach has however shown not to be robust in edge cases, which led the teams to resort to deep learning methods. Here, custom CNNs [9], [28] with a ResNet-based architecture [15] were developed to detect "corner-like" features on the input image – the bounding boxes of the previously detected cones. The detected keypoints are used as 2D points on the image to make correspondences with the respective points on the 3D model of a cone. Using the correspondences and the camera intrinsics, PnP is used to estimate the 3D position of every detected cone. As for stereo camera setups, the 3D position of the detected cones can be estimated using stereo matching algorithms that compute the points' depth by getting the disparity map between the two images.

Regarding the more classical computer vision techniques, two methods were tested by Zeilinger *et al.* [31] in order to detect the cones using a stereo camera. The first one performs a block-matching algorithm [3] on a depth image, which allows to extract and remove 3D planes. Since the track floor is geometrically known, it can be pre-segmented, making cone proposals appear as isolated objects in the image. These proposals are then further evaluated using classical image processing steps on the estimated location in the image plane, similar to the approach taken in [30]. The second method is able to detect cones in both images separately by using an algorithm that computes the disparity, *i.e.* , the distance

[2]https://fstlisboa.com. Accessible on July 30, 2021.

between two corresponding points in the left and right images of the stereo camera, the z-depth, and, consequently, the 3D position of the cone. In both methods, the color of the cones is heuristically estimated by evaluating the RGB pixel values of the cone's centroid, which should be within a white or a black stripe, and moving downwards until another color (blue, yellow, or orange) is found.

B. LiDAR Only Approaches

These implementations [2], [4], [14] have in common their two first LiDAR processing steps. Firstly, the ground plane is removed using an adaptive ground removal algorithm [16] that adapts to changes in the inclination of the ground using a regression-based approach. This algorithm divides the LiDAR's FoV into angular segments and it splits each segment into radial bins from which a line is then regressed using the lowermost points of all the bins in a segment. Finally, all the points that are within a threshold distance to this line are classified as ground points and are removed. The second LiDAR processing step consists of clustering the point cloud. Here, the Euclidean Clustering method is used to group the point cloud into clusters by performing clustering extraction in a Euclidean sense, *i.e.* , by computing the Euclidean distance between each of the 3D points of the point cloud and, using heuristic methods, evaluate whether that point corresponds to the cluster or not. Having the cones clustered out, Agarwal *et al.* [2] and Gosala *et al.* [14] apply some heuristic-based filters to estimate the likelihood of the clusters being cones, before going for the cones' color estimation step. First, the dimensions of a bounding box created around the cluster and the elements of the covariance matrix are used to filter candidates. Then, using the dimensions of the cones and the vertical and horizontal angular resolutions of the LiDAR, a rule-based filter is used to check whether the number of points in that cluster is in accordance with the expected number of points in a cone at that distance. Only the clusters that successfully go through these filters are considered to be cones and are forwarded to the color estimation step.

To estimate the color of the validated cones, all the teams chose to use different deep learning methods. These teams exploit the fact that the cones used in Formula Student can be distinguishable not only by their color pattern but also by their intensity pattern. Yellow cones have a yellow-black-yellow pattern whereas blue cones have a blue-white-blue pattern, which results in differing LiDAR intensity patterns. Gosala *et al.* [14] exploits the differing LiDAR intensity patterns of the cones to develop a custom CNN whose input image corresponds to an image created by mapping the 3D bounding boxes of the validated clusters and whose pixel values store the intensities of points in the point cloud. Similarly, Agarwal *et al.* [2] use binary classification techniques by developing a shallow 1D CNN that exploits the mean intensity and number of points per ring in the validated clusters. Following a different route, and inspired by the different colors of cones that distinguish the left and right boundary of the track, Chen *et al.* [4] chose to use a custom CNN to distinguish the geometric distribution of cones and, consequently, find the color of the validated clusters.

C. LiDAR and Camera Approaches

These implementations [13], [29] start by identifying the cones using methods similar to those described in Sec. II-B, *i.e.* , methods that only rely on the LiDAR. First, the raw point cloud is filtered using either a box region to remove points that are outside it or a pass-through filter that performs a simple filtering along any specified axis, removing points that lay outside the defined range for each axis. Then, the ground points are removed from the point cloud using Random Sample Consensus (RANSAC) [8] with the assumption that the ground plane is flat. Finally, Euclidean clustering is used to cluster the point cloud into groups and to compute their correspondent centroids, which are then projected onto the image plane.

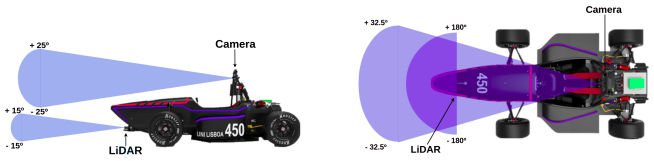
Tian *et al.* , in [29], create a Region of Interest (ROI) around the projected centroids and, from these ROI boxes, gradient features are extracted using Histogram of Oriented Gradients (HOG) [7]. Furthermore, using the extracted HOG features, a Support Vector Machine (SVM) [6] classifier is trained to distinguish cones from other clustered objects. The RGB color space of the ROI patches is then converted to the HSV color space and the color of the cones is estimated by extracting the main color present on the ROI.

In [13], the FST Lisboa team proposes using YOLOv3-tiny [1], a lightweight version of the 2D real-time object detector YOLOv3, to classify and identify the location of the cones on the camera image. Then, for each projected centroid, it is evaluated if it falls inside any of the bounding boxes that surround the detected objects. Those who do, receive the color associated with the class of the correspondent bounding box. Although this approach combines the best features of each sensor by using the position estimated by the LiDAR and the color estimated using the camera, it is a very calibration dependent approach, in the sense that a single projected point is used to evaluate if it matches any bounding box and, since the further the detected cones are, the smaller the bounding boxes will be, this matching task becomes more demanding and less forgiving for further away cones. Moreover, this approach involves using the full image to detect the cones, something that is very computationally expensive, even running, on GPU, a lightweight version of a real-time object detector like YOLOv3-tiny.

The perception pipeline proposed in this work fits inside this last approach category, in the sense that it fuses the information between the two sensors to identify the cones and classify their color. However, the proposed perception pipeline differs by not using heavy object detectors to detect the different colored cones in the full camera image. Instead, the cones are detected using the LiDAR information and are then projected onto the camera image plane, where 2D bounding boxes are created around the cones. These 2D bounding boxes are cropped from the original image and, using custom lightweight CNNs, the resultant image patches are classified, something that is significantly less computationally expensive.

III. Proposed Pipeline

This section describes the vehicle setup, the proposed perception pipeline and how it helps solve the problems identified



(a) Side view. (b) Top View.

Fig. 1: Position and Field of View (FoV) of LiDAR and Camera.

while using the perception pipeline previously implemented by the FST team. The vehicle setup is described in Sec. III-A. In Sec. III-B it is described the process of identifying the cones that delimit the Formula Student tracks using only the LiDAR sensor. Then, in Sec. III-C, it is detailed how the sensors can be fused to help the cone's color classification task described in Sec. III-D. Finally, in Sec. III-E, it is described the implementation of a tracking algorithm that allows to mitigate misclassifications by taking into account previous observations.

A. Vehicle Setup

The vehicle chosen to be transformed to compete in the Formula Student Driverless (FSD) class was the last prototype developed by the Formula Student Técnico (FST) team, the FST09e. This was the ninth prototype developed by the team and it is currently the most successful and reliable prototype developed and built by the team. In order to meet the set of challenges imposed by the FSD competition, the prototype was equipped with a set of sensors that replaces the driver's ability to perceive the environment. The first sensor added to the car was a Velodyne VLP-16 LiDAR sensor, which allows to accurately get the distance to obstacles that are within a 100-meter range and inside a 360 degrees horizontal Field of View (FoV). The LiDAR was positioned on top of the front wing of the car as it corresponds to the position that maximizes the number of point returns per cone, allowing to detect further away cones. The second perception sensor added to the prototype was a Lucid Triton TRI032S camera, responsible for collecting rich and colorful information of the environment in the form of an RGB image, complementing the sparse information, in the form of a point cloud, retrieved by the LiDAR. This camera offers a useful 65 degrees of FoV with a 2048×1536 resolution and a frame rate of 30 FPS. The camera is positioned on the main hoop of the car, close to where the eyes of the pilot would be, as this positioning allows to reduce the occlusion among cones and to still be able to perceive cones that are placed one behind the other (in line of sight). The positioning of these sensors as well as a representation of their FoV are represented in Fig. 1.

B. Cones Detection

This section describes the process of identifying and generating cone proposals, and their corresponding position, using only the point cloud generated by the LiDAR. The generated cone proposals will later be used in the classification step described in Sec. III-D, where it will be assessed which proposals actually represent cones and with what color. These cone proposals are generated through a multi-step point cloud processing that includes a pass-through filter to trim the LiDAR Field of View (FoV), a ground removal algorithm to remove points that belong to the ground, a clustering algorithm to extract and group the point cloud into clusters, a cone reconstruction algorithm to retrieve points incorrectly removed during the ground removal process and a final cone validation step to remove any remaining outliers.

1) Pass-through Filter: As described in Sec. III-A, the LiDAR used in the FST autonomous prototype has a range of around 100 meters and a horizontal FoV of 360 degrees with a 30-degree vertical FoV angle. Due to specifications and positioning of the LiDAR, most of the points of the generated point cloud lay outside the Region Of Interest (ROI). Therefore, in order to remove these points from the point cloud and avoid using them on future point cloud processing steps, which would increase their total execution time, a pass-through filter, from the Point Cloud Library (PCL) [26], is applied to the raw point cloud generated by the LiDAR. This pass-through filter performs a simple filtering along any specified axis, removing points that lay outside the defined range for each axis. Thus, applying this pass-through filter on all three axes allows not only to remove all the points that are behind the vehicle, but also points that are much higher than the height of a cone used in Formula Student competitions and points that are much farther, widthwise, than the width of a Formula Student track.

2) Ground Removal: Although most of the points that lay outside the ROI were removed in the previous LiDAR processing step, there are still several points in the ROI that were generated by the reflection of the LiDAR laser beams on the ground. Since these points can be considered noisy data, as they mostly belong to the ground plane and do not represent cones, the same principle of removing these points to avoid using them in LiDAR processing steps is followed. To accomplish this, an implementation of the iterative Random Sample Consensus (RANSAC) [11] method, from the sample consensus module of the PCL library, was used. This implementation allows to choose a mathematical model to be used as the mathematical model whose parameters will be estimated by the RANSAC method. RANSAC is a resampling technique that generates candidate solutions by using the minimum number of observations (data points) required to estimate the underlying model parameters. In this case, the points to be removed belong mostly to the ground, which in the Formula Student context can be approximated to a plane. Thus, the iterative RANSAC method was implemented using the PCL sample consensus plane model, which defines a plane model for 3D plane segmentation. This allows identifying the ground plane and removing it from the point cloud.

3) Euclidean Clustering: After the previous two LiDAR processing steps, one gets an unorganized point cloud where all of the 3D points are contained in the ROI, the majority of them belonging to cones. However, despite knowing that most of these 3D points belong to cones, one does not know to which cone each of these points belongs, *i.e.* , at this

point, there can be either as many cones as the number of 3D points in the point cloud or a single cone to which every 3D point belongs. To address this problem, the Euclidean Distance Clustering method [27], is used, which performs cluster extraction in a Euclidean sense, *i.e.* , it computes the Euclidean distance between each one of the 3D points of the point cloud and it groups them into clusters based on predefined parameters such as the minimum and the maximum number of points that a cluster can contain in order to be considered valid, the maximum distance between two points so that they can be considered part of the same cluster, and the spatial tolerance for new cluster candidates. These parameters can be tuned taking into consideration the dimensions of a cone used in Formula Student competitions and the expected distance between cones.

4) Cone Reconstruction: During the ground removal process, it was noticed that the bottom-most layer of points hitting a cone was often being removed, as it was being associated with the ground plane. Given the small number of LiDAR layers hitting cones resulting from the not so fine LiDAR vertical resolution, which is an hardware drawback, these layers can act as invaluable information, especially in the cone classification step presented in Sec. III-D1. Thus, an extra step in the LiDAR processing is done in order to retrieve these so valuable layers of 3D points. To retrieve these points, a cylindrical area, with similar dimensions to those of a Formula Student cone, is created around each cluster using its corresponding centroid. The cylindrical areas, which are an approximation to the geometrical shape of a cone, are then used to assess if any of the 3D points from the removed ground point cloud is contained inside it. Those that are contained inside the created cylindrical area are then added back to the filtered point cloud and associated with its corresponding cluster.

5) Cone Validation: Even though in Formula Student competitions the tracks are in a substantially controlled environment when compared to urban traffic, there can still be obstacles or objects nearby that are not cones and therefore not part of the track. Despite not being part of the track, these obstacles can be present in the LiDAR FoV and in the desired ROI, which may cause them to be considered as clusters while not being cones. To avoid using these outlier clusters in the classification task, one last LiDAR processing step is done. In this step, and similarly to what was done in Sec. III-B4, a cylindrical area with similar dimensions to those of a Formula Student cone, is fitted around each of the cone proposal's centroid and it is assessed if all the points of the proposal's cluster are contained inside the respective cylinder. At this point, only the clusters whose all points fall inside the correspondent cylindrical area are considered and used to generate cone proposals to later be classified and assigned a color.

C. Sensor Fusion

The clusters validated in the last step of the LiDAR, which may not all be cones, are considered to be cone proposals. These cone proposals need to be classified in order to validate if they are actually cones and with which color. Since these cone proposals may be outside the Field of View (FoV) of the camera, which commonly happens in tight corners or when the cone proposals are very close to the car, two different Convolutional Neural Networks (CNNs) are used to classify the proposals: one that resorts to the camera image to classify the image patches of the proposals and one that resorts to the point cloud intensity to generate grayscale images of the cone proposals. The choice of the CNN to be used in the classification task is based on whether the camera is able to see the cone proposal or not. To determine this, both sensors are fused by fitting a 3D bounding box, with similar dimensions to those of a cone used in Formula Student competitions, around each 3D cone proposal's centroid and then projecting it onto the image plane. Considering that on an image it is possible to define a bounding box using only its top-left and bottom-right points, only the two correspondent 3D points need to be projected onto the camera's image plane. To compute these two points for each 3D centroid $\mathbf{P} = (X, Y, Z) \in \mathbb{R}^3$, two new points, $\mathbf{P}_1$ and $\mathbf{P}_2$, are created. These points are created by keeping the centroid's X coordinate, which represents the depth of the cone proposal's centroid, and by taking into account the cone's dimensions, as follows:

$$\mathbf{P}_1 = (X, Y + w/2, Z + h/2), \tag{1}$$

$$\mathbf{P}_2 = (X, Y - w/2, Z - h/2), \tag{2}$$

where w and h represent the width and height of a cone, respectively. Here, the first point, $\mathbf{P}_1$, represents the top-left point of the bounding box while the second point, $\mathbf{P}_2$, represents the bottom-right point of the bounding box.

These two points can be mapped to points in pixel coordinates on the image plane using the 3×4 perspective projection matrix P, as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim K \begin{bmatrix} R|T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{3}$$

with $P = K \begin{bmatrix} R|T \end{bmatrix}$, where K represents the camera's intrinsic matrix and $\begin{bmatrix} R|T \end{bmatrix}$ represents the extrinsic matrix. The camera's intrinsic matrix K transforms 3D camera coordinates into 2D homogeneous image coordinates using a perspective projection modeled by the ideal pinhole camera model and is parameterized as

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4}$$

where f_x and f_y represent the camera's focal length, *i.e.* , the distance between the camera's pinhole and the image plane, and c_x and c_y represent the principal point offset, which is the location of the principal point relative to the image plane origin. The extrinsic matrix $\begin{bmatrix} R|T \end{bmatrix}$, given by

$$\begin{bmatrix} R|T \end{bmatrix} = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{array}\right], \tag{5}$$

is a 3×4 matrix that takes the form of a rigid transformation by concatenating a 3×3 rotation matrix R, whose columns

represent the directions of the LiDAR axes in camera coordinates, and the 3×1 translation column-vector T, which can be interpreted as the position of the LiDAR origin in camera coordinates. The extrinsic matrix describes how the LiDAR is transformed relative to the camera, *i.e.* , it allows to map points from the 3D LiDAR coordinate system to the 3D camera coordinate system.

Using Eq. (3) to project the two points from each of the 3D boundig boxes, it is possible to create 2D bounding boxes around the cone proposals in the image plane. By simply checking if the entire 2D bounding boxes fit inside the camera image, it can be concluded which cone proposals fall inside the camera's FoV and which don't. For those who do, the corresponding bounding box is cropped from the original camera image, resulting in an image patch with the cone proposal. This image patch corresponds to the input image of the CNN further described in Sec. III-D2. On the other hand, the cone proposals that are outside the camera's FoV go through a preparation process that converts their 3D point cloud clusters into grayscale images so that these cone proposals can be classified using only information given by the LiDAR, as further described in Sec. III-D1.

D. Cones Classification

As previously mentioned, two CNNs are used to classify the color of the identified cone proposals. In Sec. III-D2 it will be described the CNN that resorts to the camera image to classify the image patches of the proposals that fall inside the camera's FoV. Then, in Sec. III-D1 it will be described the CNN that resorts to the point cloud intensity to generate grayscale images of the cone proposals that are outside the camera's FoV.

1) Cone's Color Classification with LiDAR: In tight corners it is common for the camera not to be able to see the cones placed on the outer side of the turn, due to the camera being rigidly mounted to the car and pointing forward. This means that the cones placed on the inner side of the curve will not be able to be classified using the camera image. In such situations, and due to the nature of the path planner algorithm developed by the FST Lisboa team, the resulting trajectory may not be representative of a good trajectory for the car to follow, as only one side of track is visible.

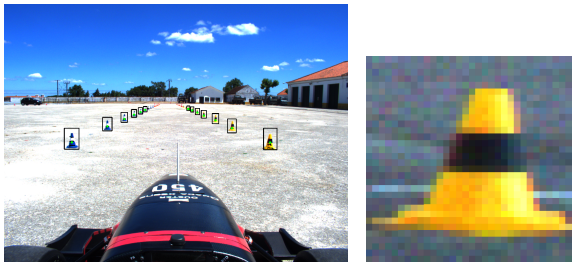
According to the rules imposed by the Formula Student competitions [12], the different cones used in Formula Student Driverless must have always the same dimensions, shape and color pattern. This means that a yellow cone will always be distinguishable by its yellow-black-yellow pattern and a blue cone by its blue-white-blue pattern. Since the intensity of the LiDAR points correspond to the returned strength of the laser pulse that generated the point, which is directly influenced by the object's reflectivity, each cone can also be distinguished in the point cloud according to its intensity pattern. Therefore, since black reflects less than yellow and white reflects more than blue, a yellow cone can be described, from top to bottom and in terms of intensity, as having a high-low-high pattern, while a blue cone can be described as having a low-high-low pattern. The orange cones are not considered here due to their intensity pattern being similar to the intensity pattern of the blue cones. The orange cones can be distinguished by their orange-white-orange pattern which, in the point cloud, is also translated to a low-high-low intensity pattern. Furthermore, these cones are traditionally placed on straights and therefore can be seen by the camera.

(a) Blue cone. (b) Yellow cone. (c) Unknown.

Fig. 2: Representation of the input image of the LiDAR CNN for each class.

This intensity gradient pattern is then exploited to estimate the cone proposals' color using a CNN with a similar architecture to the one proposed by Kabzan *et al.* [19]. The input of this CNN corresponds to a 32×32 grayscale image with the intensity points of the cone proposal and the output corresponds to the probability of the proposal being blue, yellow and unknown. The backbone of the CNN, which performs the feature extraction, consists of four convolutional layers with a max-pooling layer between them, which not only reduces the number of parameters in the model by downsampling the feature map but it also makes feature detection more robust to object orientation and scale changes. In order to introduce non-linearity into the model, the Rectified Linear Unit (ReLU) activation function is used after each convolutional layer and after each of the five fully connected layers that compose the classifier. The output of the network is normalized to a probability distribution over the three classes using Softmax. Furthermore, the aforementioned CNN uses a cross-entropy loss function that increases as the predicted probability diverges from the actual label, penalizing and adjusting the model weights based on how far the predicted probability is from the actual expected value. It is also less computationally expensive when computing the network's gradients, which, consequently, converges faster to the optimal value. Moreover, while training the model, Dropout [17] - a regularization technique that randomly "deactivates" neurons in the neural network - and batch-normalization [18] - a technique that normalizes the distributions of the hidden layer's inputs - are used to prevent complex co-adaptations between neurons, control overfitting and improve the generalization of the network.

The input images, similar to those represented in Fig. 2 are created by mapping every point of the cone proposal's cluster to a 32×32 image whose pixel values store the intensity of the LiDAR beam in that point. These points are mapped into a 32×32 matrix, using a linear interpolation, which leaves a boundary of 3 points around the cone. In order to enhance the intensity pattern of the cones, the pixel values are scaled using a linear scale factor, benefiting high-intensity values and penalizing low ones. This network has been trained



(a) (b)

Fig. 3: 2D bounding boxes created using the 3D-2D projection, represented in (a), and an example of an image patch with the cone proposal, represented in (b).

with a custom dataset of over 19 000 images with the mapped intensity values of blue and yellow driverless Formula Student cones and noisy identified clusters to represent the unknown class.

2) Cone's Color Classification with Camera: The classification of the remaining cone proposals, whose projected 2D bounding boxes fit inside the camera image, is performed using a CNN with an architecture similar to the one described in Sec. III-D1. However, this CNN takes as input a 32×32 RGB image that corresponds to the image patch of the projected bounding box that surrounds the cone proposal, as shown in Fig. 3. Furthermore, the architecture of the network described in this section has one less convolutional layer in the feature extraction stage, a change that allowed to achieve better computational times while maintaining an acceptable accuracy and reliability, as demonstrated in Sec. IV.

This CNN consists of three convolutional layers that perform feature extraction and, similarly to the CNN described in Sec. III-D1, after each convolutional layer a max-pooling layer is used to downsample the feature maps after using Rectified Linear Unit (ReLU) as the activation function to introduce non-linearity to the model. As for the classifier, it consists of five fully connected layers, where to the output of each one is also applied the ReLU activation function. The output of the network is normalized to a probability distribution over the five classes using Softmax. Additionally, while training the model, dropout and batch normalization were also used to randomly "deactivate" some neurons in the neural network and to normalize the distributions of the hidden layer's inputs, preventing complex co-adaptations between neurons, controlling overfitting and improving the generalization of the network. This network has been trained with a custom dataset of over 100 000 images of the four classes of Formula Student Driverless cones and background images to represent the unknown class. The dataset was originally only composed of images from FSOCO [10] - a collaboration between Formula Student teams that aims to accelerate the development of camera-based solutions in the context of Formula Student Driverless - but has since been enlarged with more images gathered from runs on different testing sites, with different weather/light conditions and with different cameras.

E. Tracking

Given the simplicity of the CNNs described in Sec. III-D1 and in Sec. III-D2, which allow to classify and associate a color to the cone proposals generated through the LiDAR processing described in Sec. III-B, a Nearest Neighbor [5] based tracking algorithm was developed in order to mitigate possible misclassifications returned by either CNNs. By taking into account previous observations, it is possible to keep track of the colors associated with each cone proposal and thus decide the cone's color based on the number of times each color was associated with the proposal.

This tracking algorithm starts by fusing the information returned by both neural networks, the cone detections with an associated position and color. Once all the cone detections are fused, a different ID is assigned to each of them and, simultaneously, the cone's color is registered. Having assigned the first IDs, the Nearest Neighbor based tracking algorithm is used to associate the next set of cone detections to the already existing and tracked cone detections. So, for each previously tracked cone detection, it is searched for the closest cone detection, within the new set of cone detections, and the tracked cone's position is updated and the color is registered, both using the information of the closest cone detection. It should be noticed that, to avoid miss associating new cone detections to a tracked cone that, although is the closest one, it is too far away, a searching radius is used taking into account the minimum distance between cones defined by the Formula Student competitions rules. The cone detections that were not associated with any previously tracked cone are considered to be new cones and an ID is assigned to them, allowing them to be used as tracked cones in the next iteration. On the other hand, the previously tracked cones that were not associated with any new cone detection are registered as being missed. By keeping note of the number of times a tracked cone was missed, it is possible to remove not only outliers that stopped being identified but also cones that have already left the LiDAR FoV, *e.g.* , cones that have already been passed by the car and are now behind it.

Furthermore, the cone's color is decided by searching for the color that was most associated with that cone ID. For example, a cone tracked with the ID 56 that was classified eight times as being blue, if the last color classification happens to classify the cone as a yellow cone, the cone will still go through as being a blue cone, because it has already been previously classified as being blue many more times. The implemented tracking algorithm has proven to be a powerful filter, as it helps to smooth out cone color estimation and make the pipeline more robust to misclassifications, something that significantly helps the path planning algorithms in a previously unknown track.

IV. Experimental Results

In this chapter, the overall results of the proposed perception pipeline will be evaluated in terms of performance and computation cost. First, the overall results of the Convolutional Neural Network (CNN) that solely relies on the information from LiDAR are presented in Sec. IV-A. Similarly, in Sec. IV-B,

it is presented the overall results of the CNN that relies on the created image patches with the cone proposals. Then, in Sec. IV-C, an ablation study on the LiDAR processing methods is conducted in order to justify the design choices made on the LiDAR pipeline. Finally, in Sec. IV-D, it is performed a comparison of the overall perception pipeline results with and without the tracking module.

A. LiDAR Classification

The model of the LiDAR CNN was trained on an NVIDIA GeForce GTX 1070 for 100 epochs with a batch size of 128 and using the Adam optimizer [20] with a learning rate starting on 0.0001. Resorting to a learning rate scheduler, the learning rate is dynamically reduced by a factor of 0.1 whenever the validation loss stops decreasing for more than 10 epochs, which indicates that the learning has stagnated.

First, to validate the importance of the using the LiDAR CNN, a study was conducted on the number of cones that it helps classifying. For this, the full perception pipeline was run over raw data gathered from a lap performed around a Formula Student Driverless track with and without using the LiDAR CNN. The results of this study showed that using the LiDAR CNN allows identifying 16 more cones in a track with a total of 93 cones, which represents an increase of 17.2% in the number of identified cones. From Fig. 4 it is possible to conclude that the LiDAR CNN is essential to identify the cones placed in the inner side of the curves, where mostly cones from the opposite class can be seen by the camera.

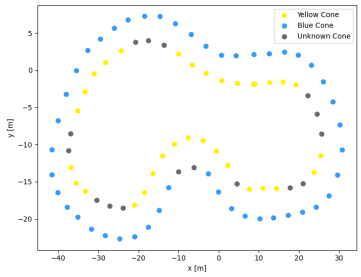


Fig. 4: Number of cones classified without using the LiDAR CNN.

Then, in order to validate the importance of the cone reconstruction step detailed in Sec. III-B4, which retrieves cone points that were incorrectly removed in the ground removal step, a comparison between the LiDAR CNN results with and without performing this reconstruction step was conducted. To perform this comparison, a new test dataset was created with the same cone images used in the splitted test dataset but without performing cone reconstruction. The results obtained from the trained model on both test datasets are detailed in Table I.

Observing Table I, one can conclude that the overall results obtained when cone reconstruction is not performed is around 10% worse across all performance metrics. This can be explained by the missing bottom-most layer of points in the input images of the LiDAR CNN when no reconstruction is performed. This is especially noticeable for further away cones which, due to the not so fine vertical resolution of the LiDAR, lack on the number of beams that hit them. For these cones, the missing layer of points breaks the intensity pattern used as the base for this CNN, turning the intensity pattern of the cones into low-high and high-low intensity patterns, respectively for a blue cone and a yellow cone. These intensity patterns are much less distinguishable, which justifies the increase of false positives for the unknown class, evidenced by its 15% lower precision.

TABLE I: Results of the trained LiDAR CNN model on the test dataset.

Method	Class	Precision (%) ↑	Recall (%) ↑	F1 (%) ↑	Accuracy (%) ↑
With Cone Reconstruction	Blue Cone	93.34	85.98	89.51	87.77
	Yellow Cone	94.38	84.02	88.90	
	Unknown	76.51	94.93	84.73	
Without Cone Reconstruction	Blue Cone	90.08	73.03	80.66	77.17
	Yellow Cone	89.14	69.01	77.79	
	Unknown	60.87	92.99	73.58	

B. Camera Classification

The camera CNN model was trained on an NVIDIA GeForce GTX 1070 for 250 epochs with a batch size of 64 and using the Stochastic Gradient Descent (SGD) optimizer [25] with a learning rate starting on 0.00001. Similarly to the train performed on the LiDAR CNN model, a learning rate scheduler was used to dynamically reduce the learning rate by a factor of 0.1 whenever the validation loss stops decreasing for more than 10 epochs.

As mentioned in Sec. III-D2, the architecture of the feature extractor of this CNN was reduced from four convolutional layers to only three. This design choice was based on a study conducted on the camera CNN performance in terms of classification and computation costs. The results obtained from this study are represented in Table II and Table III.

TABLE II: Average inference time of the camera CNN on GPU and CPU for both architectures.

	Average Inference Time (ms) ↓	
Architecture	**GPU**	**CPU**
4 Convolutional Layers	1.9	4.5
3 Convolutional Layers	1.6	2.3

TABLE III: Results of the trained camera CNN model on the test dataset.

Architecture	Class	Precision (%) ↑	Recall (%) ↑	F1 Score (%) ↑	Accuracy (%) ↑
4 Convolutional Layers	Blue Cone	96.32	96.00	96.16	93.58
	Yellow Cone	97.25	94.99	96.11	
	Orange Cone	89.48	90.01	89.74	
	Big Orange Cone	91.00	87.00	88.96	
	Unknown	90.69	95.99	93.26	
3 Convolutional Layers	Blue Cone	94.84	94.00	94.42	90.40
	Yellow Cone	95.37	91.99	93.65	
	Orange Cone	83.23	86.00	84.59	
	Big Orange Cone	86.53	82.00	84.20	
	Unknown	87.26	93.84	90.43	

Regarding the inference times of the CNN, it is possible to conclude that the architecture that resorts only to 3 convolutional layers is faster on both GPU and CPU. Although the decrease in inference time is less noticeable when the model is run on GPU, decreasing only 0.3 ms, on CPU the inference time decreases by almost 50%, from 4.5 ms to 2.3 ms. Furthermore, from Table III, it can be observed that

both models achieve around 90% on almost all performance metrics for almost all classes. It is also possible to observe that the expected loss of performance between the two models is not very accentuated, decreasing only around 3% on all performance metrics, except for the Orange and Big Orange Cone classes, when the model with three convolutional layers is used. The bigger performance decrease in these two classes can be justified by the fact that these two cones are only distinguishable by the extra white stripe on the big orange cones and, since a deeper CNN is able to extract deeper features, the model with four convolutional layers can extract deeper features and thus correctly classify these classes more often. Furthermore, the lower performance on the orange and big orange cones in both models can also be explained by the class distribution of the training dataset, as these cones are around 10% less represented. This small decrease in the classification performance of the camera CNN allied to the possibility of being able to run the CNN on CPU, justifies the choice of decreasing the feature extractor architecture of the camera CNN.

C. LiDAR Processing Ablation Study

To justify the need of all the LiDAR processing steps described in Sec. III-B, an ablation study was conducted on the implemented methods presented in the LiDAR pipeline. The results from this ablation study are depicted in Table IV.

TABLE IV: Number of cone proposals and correspondent distribution per iteration for each CNN after removing certain LiDAR processing steps.

Method	Cones on Track	Number of Cone Proposals	Avg. Number of Cone Proposals per Iteration	
			LiDAR CNN	Camera CNN
Full LiDAR Pipeline	147	168	2.25	5.24
Without Pass-through Filter	147	4 937	44.05	176.19
Without Ground Removal and Cone Reconstruction	147	109	1.46	3.39
Without Cone Validation	147	214	2.86	6.68

Initially, the pass-through filter used to trim the LiDAR FoV was removed, significantly increasing the number of cone proposals identified and, consequently, the number of cone proposals sent to both classification CNNs. Removing the pass-through filter, which, per iteration, only takes an average of 1.65 ms to trim the LiDAR FoV, causes an increase of over 6 ms in the euclidean clustering average execution time. Since the euclidean clustering method is responsible for grouping the point cloud into clusters, the greater is the LiDAR FoV, the greater will be the number of points in the point cloud, which explains this increase in the execution time of the algorithm and in the number of cone proposals identified. Furthermore, these increases in the number of points in the point cloud and in the number of cone proposals identified, cause the subsequent LiDAR processing and classification steps to also significantly increase their execution times per iteration, as there are more information to be processed and more cone proposals to be classified. Consequently, the average execution time per iteration of the full perception pipeline increases by over 2300%, from which can be concluded that the pass-through filter is essential in the proposed LiDAR pipeline.

TABLE V: Analysis of the average execution time per iteration after removing certain LiDAR processing steps.

Method	Average Execution Time per Iteration (ms)							Total Execution Time (ms)	
	Method	Euclidean Clustering	Full LiDAR Pipeline	LiDAR CNN		Camera CNN			
				GPU	CPU	GPU	CPU	GPU	CPU
Full LiDAR Pipeline	4.13	0.33	4.13	3.6	5.18	8.38	12.05	16.11	21.36
Without Pass-through Filter	1.65	6.55	37.23	70.48	101.32	281.9	405.24	389.61	543.79
Without Ground Removal and Cone Reconstruction	2.16	35.71	37.30	2.34	3.36	5.42	7.80	45.06	48.46
Without Cone Validation	0.02	0.33	4.14	4.58	6.58	10.69	15.36	19.41	26.08

The second method removed from the LiDAR processing was the ground removal and, consequently, the cone reconstruction, as it is no longer necessary. Removing the ground removal step from the LiDAR pipeline decreases the number of identified cone proposals and, consequently, the number of classifications performed by both classification CNNs. Although these decreases cause a reduction on the execution time per iteration of the classification task, which is explained by the reduced number of cone proposals to be evaluated, this reduction in the number of cones means that less cones are being identified. Since the ground is not being removed, the euclidean clustering algorithm tends to associate the ground points to cones that are closer to them, which can cause the group of points to exceed the defined maximum number of points that a cone can have and therefore not being considered as clusters. Furthermore, by removing the ground removal step, the number of points in the point cloud significantly increases, even more than when the pass-through filter is removed. As previously explained, this causes a significant increase in the average execution time per iteration of the euclidean clustering algorithm (over 35 ms) and, consequently in total execution time of the full perception pipeline. Therefore, since removing this step decreases the number of cones identified while increasing the total execution time, it is essential to maintain it on the LiDAR processing.

Finally, it was assessed if it was worth performing the cone validation step that allows to remove most of the outliers from the cone proposals identified. Removing this method, slightly increases the number of cone proposals identified and, consequently, the number of classifications performed by the classification CNNs. As this method is performed over the already clustered cone proposals, the average execution time per iteration of the euclidean clustering remains unchanged. This extra step only takes an average of 0.02 ms and it is the only extra delay added to the LiDAR pipeline, *i.e.* , it does not cause any further delay on the LiDAR pipeline. However, with this small delay comes a decrease of over 3 ms on the total execution time of the perception pipeline, as the number of classifications performed by both the LiDAR CNN and the camera CNN are reduced.

D. Tracking

In order to test the influence of the developed tracking algorithm, the proposed perception pipeline was run over raw data from an acceleration track in a controlled environment, where the position and color of the cones is known. The

number of color corrections made by the tracking algorithm is depicted in Table VI, from where it is possible to conclude that 92% of the misclassifications got its color corrected by the tracking algorithm. Most of the remaining misclassifications belong to cases where the cone was associated with the Unknown class several times at the beginning, which can happen for further cones if the camera-LiDAR calibration is not perfect. For these cones, only when the respective color surpasses the unknown classifications, the color is corrected.

TABLE VI: Number of misclassifications per class and number of color corrections.

Class	Number of Misclassifications	Number of Color Corrections
Blue	92	84
Yellow	114	106
Orange	70	65
Big Orange	39	36
Total	**315**	**291**

V. Conclusion

The perception pipeline proposed in this thesis fuses the best features from the two available perception sensors (a camera and a LiDAR) by combining the cone's position estimated by the LiDAR pipeline, which processes the raw LiDAR data to generate cone proposals, with the cone's color likelihood estimated by either the camera CNN, a custom CNN that classifies the image patches of the correspondent generated cone proposals, or the LiDAR CNN, a custom CNN that classifies the color of the cone proposals' cluster resorting to the intensity pattern that distinguishes the different cones. Furthermore, given the simplicity of the custom CNNs used, a Nearest Neighbor based tracking algorithm was developed to correct the estimated cone's color taking into account the estimated color of previous observations.

The results show that the proposed perception pipeline is able to accurately detect and classify the cones even when running the pipeline on CPU, something that can bring benefits on the Efficiency event of the Formula Student Competitions. Furthermore, they also show the importance of the developed LiDAR CNN in order classify the detected cones that the camera is not able to see, which most of times correspond to the cones places in the inner side of the curves. Moreover the developed tracking algorithm has shown to help correcting the color of misclassificated cones, avoiding the propagation of those misclassification to the path planning algorithms that follow. Further work can extend the taken approach to detect and classify other objects in other environments by relaxing the assumptions made for the Formula Student context.

References

[1] P. Adarsh, P. Rathi, and M. Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694, 2020. 2, 3

[2] A. Agarwal, K. Brandes, R. Chang, K. Doherty, M. Kabir, K. Strobel, N. Stathas, C. Trap, A. Wang, and L. Kulik. Robust, High Performance Software Design for the DUT18D Autonomous Racecar, 2018. 3

[3] A. Barjatya. Block matching algorithms for motion estimation. *IEEE Transactions Evolution Computation*, 8:225–239, 01 2004. 2

[4] T. Chen, Z. Li, Y. He, Z. Xu, Z. Yan, and H. Li. From perception to control: an autonomous driving system for a formula student driverless car, 2019. 3

[5] K. Choeychuen, P. Kumhom, and K. Chamnongthai. An Efficient Implementation of the Nearest Neighbor Based Visual Objects Tracking. In *2006 International Symposium on Intelligent Signal Processing and Communications*, pages 574–577, 2006. 7

[6] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20:273–297, 1995. 3

[7] 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, 2005. 3

[8] K. Derpanis. Overview of the RANSAC Algorithm, 2005. 3

[9] A. Dhall. Real-time 3D Pose Estimation with a Monocular Camera Using Deep Learning and Object Priors On an Autonomous Racecar, 2018. 2

[10] D. Dodel, M. Schötz, and N. Vödisch. FSOCO: The Formula Student Objects in Context Dataset, 2020. 7

[11] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981. 2, 4

[12] Formula Student Germany. FS Rules 2020, 2020. 1, 6

[13] Formula Student Técnico Lisboa. Autonomous Design Report - FST10d, 2020. 2, 3

[14] N. Gosala, A. Buhler, M. Prajapat, C. Ehmke, M. Gupta, R. Sivanesan, A. Gawel, M. Pfeiffer, M. Burki, I. Sa, and et al. Redundant Perception and State Estimation for Reliable Autonomous Racing. *2019 International Conference on Robotics and Automation (ICRA)*, May 2019. 2, 3

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition, 2015. 2

[16] M. Himmelsbach, F. Hundelshausen, and H. Wuensche. Fast segmentation of 3D point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565, 2010. 3

[17] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. 6

[18] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015. 6

[19] J. Kabzan, M. de la Iglesia Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart. AMZ Driverless: The Full Autonomous Racing System, 2019. 2, 6

[20] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017. 8

[21] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. 2

[22] J. Pedoeem and R. Huang. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers, 2018. 2

[23] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger, 2016. 2

[24] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018. 2

[25] S. Ruder. An overview of gradient descent optimization algorithms, 2017. 8

[26] B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, 2011. 4

[27] R. Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Künstliche Intelligenz*, 24:345–348, 2010. 5

[28] K. Strobel, S. Zhu, R. Chang, and S. Koppula. Accurate, Low-Latency Visual Perception for Autonomous Racing:Challenges, Mechanisms, and Practical Solutions, 2020. 2

[29] H. Tian, J. Ni, and J. Hu. Autonomous Driving System Design for Formula Student Driverless Racecar. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6, 2018. 3

[30] H. Yong and X. Jianru. Real-time traffic cone detection for autonomous vehicle. In *2015 34th Chinese Control Conference (CCC)*, pages 3718–3722, 2015. 2

[31] M. Zeilinger, R. Hauk, M. Bader, and A. Hofmann. Design of an Autonomous Race Car for the Formula Student Driverless (FSD), 2017. 2