Real-time 3D Traffic Cone Detection for Autonomous Driving

Ankit Dhall, Dengxin Dai and Luc Van Gool

***Abstract*— Considerable progress has been made in semantic scene understanding of road scenes with monocular cameras. It is, however, mainly related to certain classes such as cars and pedestrians. This work investigates traffic cones, an object class crucial for traffic control in the context of autonomous vehicles. 3D object detection using images from a monocular camera is intrinsically an ill-posed problem. In this work, we leverage the unique structure of traffic cones and propose a pipelined approach to the problem. Specifically, we first detect cones in images by a tailored 2D object detector; then, the spatial arrangement of keypoints on a traffic cone are detected by our deep structural regression network, where the fact that the cross-ratio is projection invariant is leveraged for network regularization; finally, the 3D position of cones is recovered by the classical Perspective n-Point algorithm. Extensive experiments show that our approach can accurately detect traffic cones and estimate their position in the 3D world in real time. The proposed method is also deployed on a real-time, critical system. It runs efficiently on the low-power Jetson TX2, providing accurate 3D position estimates, allowing a race-car to map and drive autonomously on an unseen track indicated by traffic cones. With the help of robust and accurate perception, our race-car won both Formula Student Competitions held in Italy and Germany in 2018, cruising at a top-speed of 54 kmph `https://youtu.be/HegmIXASKow?t=11694`. Visualization of the complete pipeline, mapping and navigation can be found on our project page `http://people.ee.ethz.ch/~tracezuerich/TrafficCone/`.**

I. Introduction

Autonomous driving has become one of the most interesting problem to be tackled jointly by the computer vision, robotics and machine learning community. Recent years have witnessed great progress in autonomous driving [33], [19], [15], leading to announcements that fully automated vehicles are just around the corner. Yet, significant technical obstacles such as the necessary robustness against adverse weather/illumination conditions [29], [6], [34], or the capability to cope with temporary, unforeseen situations such as road work and accidents [23] must be overcome before assisted driving can make way for fully automated driving.

Traffic control devices, such as traffic signs, traffic lights and traffic cones, play a crucial role in ensuring safe driving. Recent years have witnessed great progress in detecting traffic signs [28], [22] and traffic lights [16], [8]. Traffic cones, however, have not received due attention. Traffic cones are conical markers and are usually placed on roads or footpaths to safely and temporarily redirect traffic. They are often used to separate or merge lanes during road construction projects or automobile accidents. These situations need to be addressed internally with on-board sensors – even high-definition (HD) maps cannot solve this problem due to the erratic and arbitrary nature of traffic cones.

All authors are with ETH-Zurich `adhall@ethz.ch`, `{dai, vangool}@vision.ee.ethz.ch`

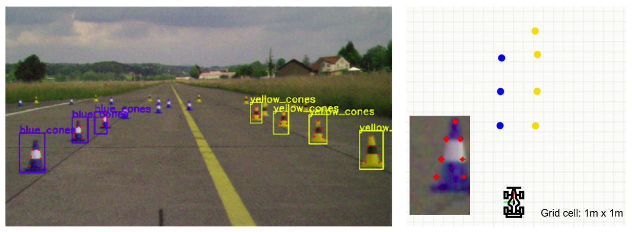

Fig. 1. The pipeline can be subdivided into three parts: (1) object detection, (2) "keypoint regression" and (3) 2D-3D correspondence followed by 3D pose estimation from a monocular image.

One might be tempted to attempt an end-to-end approach to directly map images to the throttle and steering commands [15]. We, however, believe that fusing end-to-end and part-based approaches with interpretable sub-modules is the promising future direction. Therefore, specialized detectors such as ours for traffic cones are still very necessary.

It is interesting to note here that although these traffic cones are static objects themselves, they are frequently replaced and moved around the urban driving scenario. Cars may break down unexpectedly and new constructions zones may pop up more often than anticipated. Although, buildings and landmarks can be mapped with ease and used for localization, one needs to actively detect and estimate the position of these traffic cones for safe, automated driving.

A range based sensor, such as a LiDAR would also be able to output accurate 3D position, but due to the low resolution of LiDAR scans (compared to images) and the low-resistance of LiDAR to ego-vehicle's bumping, detecting small objects and their color and texture becomes quite challenging. To add to that, LiDAR sensors are usually expensive. Recent advance in computer vision show that images from even a monocular camera can be used to not only reveal *what* is in the scene but also *where* it is physically in the 3D world.

In this work, we tackle 3D position estimation and detection of traffic cones from a single image. The task can be broken down into three steps: 2D object detection, landmark keypoints regression, and 2D-to-3D mapping. In particular, cones are detected in images by a tailored 2D object detector; the detected 2D bounding boxes are fed into another neural network to regress seven landmark keypoints on the traffic cones, where the fact that cross-ratio is projection invariant is leveraged for robustness; finally, the 3D position of cones is recovered by classical Perspective n-Point algorithm. In order to train and evaluate our algorithm, a new dataset of traffic cones is introduced. The advantage of using a monocular camera is that a multi-camera setup is is not required, making the system more cost-effective and easy to maintain.

Extensive experiments show that traffic cones can be detected accurately in single images by our method. For instance, the 3D cones position estimates deviate by only 0.5m at 10m and 1m at 16m distances when compared with the ground-truth. We further validate the performance of our method by deploying it on a critical, real-time system in the form of a life-sized autonomous race-car. The car can drive at a top-speed of 54 kmph on a track flanked by traffic cones.

The main contribution of this paper are threefold: 1) a new dataset for traffic cone detection to facilitate research in this direction; 2) a novel method for real-time 3D traffic cone detection in single images; and 3) a system showing that the accuracy of our cone detection is enough for navigating an autonomous car at a top-speed of 54 kmph. The video showing our vehicle navigating through a track flanked by traffic cones can be found at `https://youtu.be/HegmIXASKow?t=11694`.

II. Related Work

Fast object detection. One of the first successful fast object detector is Viola-Jones' face detector [35], which employs weak learners to accurately detect faces using Haar-based features. The next class of well-known object detectors use deep learning in the form of convolutional neural networks (CNNs). The string of R-CNN [10], [26], [9] schemes use CNN-based features for region proposal classification. YOLO [24] cleverly formulates object detection as a regression task, leading to very efficient detection systems. The idea was extended to 3D object detection as well [17]. While great progress has been made, the accuracy on small object such as traffic cones at larger distance requires further investigation.

Traffic device detection. A great progress has been made for detecting traffic sign [28], [22] and traffic light [16], [8] in the past years. To aid in the efforts for bench-marking, a 100,000 annotated image dataset for traffic signs has been released [37]. Li et al. [20] propose a generative adversarial network (GAN) to improve tiny object detection, such as distant traffic signs. Lee et al. [18] explore the idea of detecting traffic signs and output a finer mask instead of a coarse rectangle in the form of a bounding box. The work briefly discusses triangulation of points using the extracted object boundary across 2 frames, but only in simulation. Our work focus on traffic cone detection and 3D position estimation from single images.

Keypoint estimation. One of the main contributions of this work is to be able to accurately estimate traffic cone's pose using just a single frame. A priori information about the 3D geometry is used to regress highly specific feature points called *keypoints*. Previously, pose estimation and keypoints have together appeared in other works. [30], [12]. Glasner et al. [12] estimate pose for images of cars using an ensemble of voting SVMs. Tulsiani et al. use features and convolutional neural networks to predict view-points of different objects [32]. Their work tries to capture the interplay between viewpoints of objects and keypoints for specific objects. PoseCNN directly outputs the 6 degrees-of-freedom pose using deep learning [36]. Gkioxari et al., [11] use a k-part deformable parts model and presents a unified approach for detection and keypoint extraction on people. Our method leverages the unique structure of traffic cones, more specifically the projective invariant property of cross-ratio, for robust keypoint estimation.

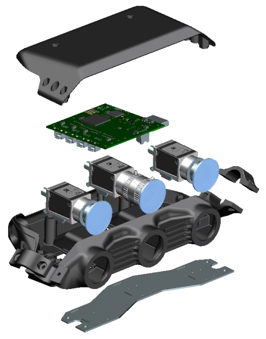
Fig. 2. The left and right cameras (on the extremes) in the housing act in a stereo configuration; the center camera is a stand-alone monocular camera and uses the pipeline elaborated in this work.

III. Monocular Camera Perception Pipeline

A. Sensor Setup and Computation Platform

It is extremely favourable to detect objects while simultaneously estimating their location in 3D space.

We use 2 megapixel cameras (CMOS sensor-based) with a global shutter to avoid image distortion and artifacts due to fast motion. Figure 2 shows our camera setup. The center camera, which is the monocular camera described in this work, has a 12mm lens to allow long range perception. The left and right cameras use short focal length lenses of 5.5mm and act as stereo pair to triangulate nearby cones. The cameras are enclosed in a customized 3D printed, waterproof shell with polarized filters in front of the lenses. The cameras transmit data over Ethernet through a shared switch, allowing for a neat camera housing. The cameras are screwed to the metallic plate at the bottom and are in direct contact to keep them at operating temperatures. Raw camera data is directly transmitted to the Jetson TX2 which acts as a slave to the master PIP-39 (with Intel i7) on-board "gotthard driverless". The pipeline is light enough to run completely on a low-powered mobile computing, the Jetson TX2, at the rate of 10Hz.

B. Pipeline Overview

Although pose estimation from a single image is an ill-posed problem, but with a priori structural information of the object-of-interest, it is tractable. With the availability of tremendous amounts of data and powerful hardware (GPUs) deep learning has proven to be good at tasks that would be difficult to solve with classical hand-crafted approaches.

Deep learning does well to learn sophisticated representations while established results from classical computer vision and mathematics provide robust and reliable estimates. In our work we strive to combine the best of both in an efficient way holding both performance and interpretability in high regard with a pipelined approach.

The sub-modules in the pipeline enable it to detect objects of interest and accurately estimate their 3D position by making use of a single image. The 3 sub-modules are (1) object detection, (2) keypoint regression and (3) pose estimation by 2D-3D correspondence. The pipeline's sub-modules are run as nodes using Robot Operating System or ROS [4] as the framework that eases handling of communication and data messages across multiple systems as well as different nodes. The approach will be detailed in Section IV.

IV. Approach

A. Object Detection

To estimate 3D position of multiple object instances from a single image, it is necessary to be able to detect them. For the task of object detection, we employ an off-the-shelf object detector in our pipeline in the form of YOLOv2 [25]. YOLOv2 is trained for the purpose of detecting different colored cones which are the principal landmarks demarcating the track. Thresholds and parameters are chosen such that false positives and misclassification are avoided. For this particular use-case YOLOv2 is customized by reducing the number of classes that it detects, as it only needs to distinguish and detect 'yellow', 'blue' and 'orange' cones each with a particular semantic meaning.

Since the bounding boxes for cones have a height to width ratio of greater than one, such prior information is exploited by re-calculating the *anchor boxes* used by YOLOv2 (see [25] for details).

Weights trained for the ImageNet [7] challenge are used for initialization. We follow a similar training scheme as in the original work [25]. The detector is fine tuned as more data was acquired and labeled. Refer to Section V-A.1 for more details of data collection and annotation.

B. Keypoint Regression

This section discusses how object detections in a single 2D image can be used to estimate their 3D positions. Doing this from a single view of the scene in itself is challenging because of ambiguities due to scale. However, with prior information about the 3D shape, size and geometry of the cone, one has can recover the 3D pose of detected objects from a single image. One would be able to estimate an object's 3D pose, if there is a 2D-3D correspondence between the 3D object and the 2D image, along with camera's parameters.

To this end, we introduce a feature extraction scheme that is inspired by classical computer vision but has a flavor of learning from data via machine learning.

1) Keypoint Representation: The bounding boxes from the object detector do not directly correspond to a cone. To tackle this, landmark features need to be extracted. In the context of classical computer vision, there are mainly three kinds of features. "Edges" are more interesting than "flat" regions as they have a gradient in one particular direction but suffer from the aperture problem [31]. By far, the most interesting features are the "corners" which have are most distinctive.

Previous feature extraction work includes renowned Harris corners [13], robust SIFT [21] and SURF [5]. A desirable property that many of these possess is invariance to transformations such as scale, rotation and illumination. Most of these work well as general feature detectors and can be used in many applications.

The issue with using pre-existing feature extraction techniques is that they are generic and detect any kind of features that fall within their criteria. For instance, a Harris corner does not distinguish whether the feature point lies on a cone or on a crevasse on the road. This makes it difficult to draw the relevant correspondences and match them correctly to their 3D counterparts. Another issue is when a patch has a low resolution, it may detect only a couple of features which will not provide enough information to estimate the 3D pose.

2) Keypoint Regressor: With the issues in mind, we design a convolutional neural network (CNN) that regresses "corner" like features given an image patch. The primary advantage over and commonly used feature extraction technique is that here data can be leveraged to robustly detect extremely specific feature points. Although, in this work we focus on cones, the "keypoint regression" scheme can be easily extended to different types of objects. The 3D locations corresponding to these specific feature points (as shown in Figure 4) can be measured in 3D from an arbitrary world frame, $\mathcal{F}_w$.

There are two primary reasons to have these keypoints at those places. First, the keypoint regressor locates position of 7 very specific features that are visually distinct and can be considered as "corners". Second and more importantly, these 7 points are relatively easy to measure in 3D from a fixed world frame $\mathcal{F}_w$. For convenience $\mathcal{F}_w$ is chosen to be the base of the 3D cone, enabling measurement of 3D position of these 7 points in this world frame, $\mathcal{F}_w$. The 7 keypoints are the apex of the cone, two points (one on either side) at the base of the cone, 4 points where the center stripe, background and upper or lower stripes meet.

The customized CNN, made to detect specific "corner" features, takes as input a $80 \times 80 \times 3$ sub-image patch which contains a cone, as detected by the object detector, and maps it to $\mathbb{R}^{14}$. The spatial dimensions are chosen as 80×80, the average size of detected bounding boxes. The output vector of $\mathbb{R}^{14}$ are the (x, y) coordinates of the keypoints.

The architecture of the convolutional neural network consists of basic residual blocks inspired from ResNet [14] in PyTorch [3].

As analyzed in [27], with more layers, the tensor volume has more channels and smaller spatial dimensions, implying the tensors contain more global and high-level information than specific, local information. We eventually care about location of keypoints which are extremely specific and local. Using such an architecture prevents loss of spatial infor-

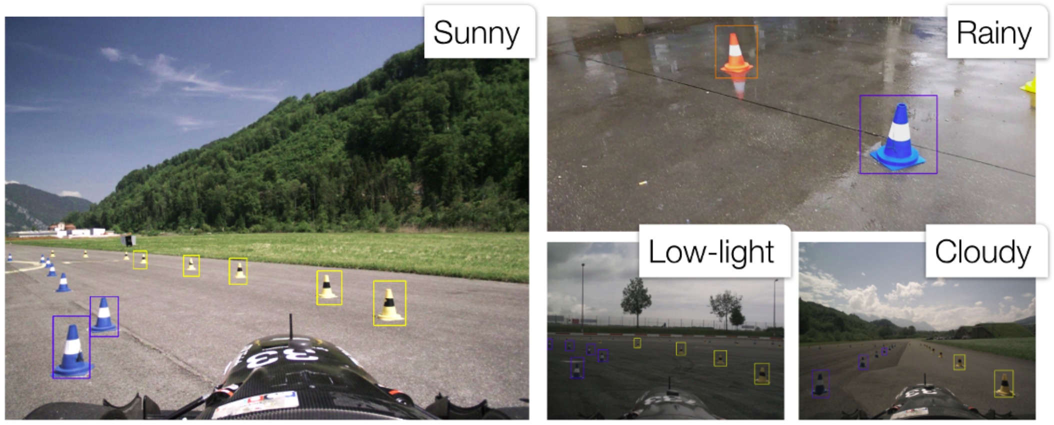

Fig. 3. Detection under varying lighting and weather conditions for ‘yellow’, ‘blue’ and ‘orange’ cones.

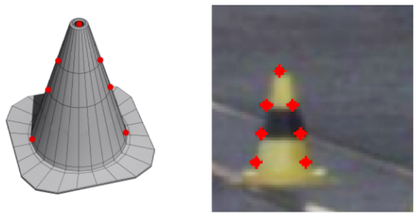

Fig. 4. 3D model of the cone and a representative sub-image patch with the image of the cone. The red markers correspond to the 7 specific keypoints the “keypoint network” regresses to given a cone patch.

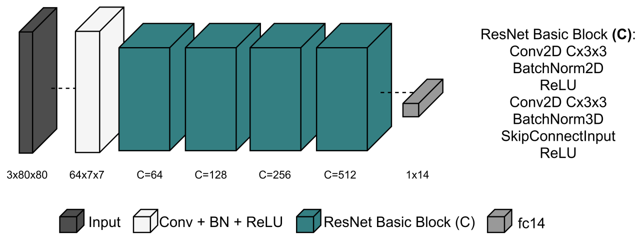

Fig. 5. Architecture of the “keypoint network”. It takes a sub-image patch of $80 \times 80 \times 3$ as input and maps it to $\mathbb{R}^{14}$, the (x, y) coordinates for the 7 keypoints. It can process 45-50 cone patches per second on a low-powered Jetson TX2.

mation as it is crucial to predict the position of keypoints accurately.

The first block in the network is a convolution layer with a batch norm (BN) followed by rectified linear units (ReLU) as the non-linear activation. The next 4 blocks are basic residual blocks with increasing channels $C \in \{64, 128, 256, 512\}$ as depicted in Figure 5. Finally, there is a fully-connected layer that regresses the location of the keypoints.

3) Loss Function: As aforementioned, we use object-specific prior information to match 2D-3D correspondences, from the keypoints to a physical cone. In addition, the “keypoint network” also exploits a priori information about the object’s 3D geometry and appearance through the loss function via the concept of the *cross-ratio*. As known, under projective transform, neither distances between points nor the ratio of these distance is preserved. However, a more complicated entity known as the *cross-ratio*, which is the ratio of ratio of distances, is invariant and remains constant under a projection. While widely used in classical computer vision approaches, the cross ratio has seldom been used in recent works, especially the ones using deep neural networks.

The cross-ratio (Cr) is a scalar quantity and can be calculated using 4 collinear points or 5 or more non-collinear points [1]. Since it is invariant under a projection and taking pictures by a camera essentially is a projective transform, this implies that the cross-ratio is preserved, both on the image plane and in 3D space.

In our case, we use 4 collinear points p_1, p_2, p_3, p_4 to calculate the cross-ratio as defined in Equation 1. Depending on whether the value is calculated for 3D points $(D = 3)$ or their projected 2D counterparts $(D = 2)$, the distance Δ_{ij}, between two points p_i and p_j is defined.

$$\begin{aligned} Cr(p_1, p_2, p_3, p_4) &= (\Delta_{13}/\Delta_{14})/(\Delta_{23}/\Delta_{24}) \in \mathbb{R} \\ \Delta_{ij} &= \sqrt{\Sigma_{n=1}^{D}(x_i^{(n)} - x_j^{(n)})^2}, D \in \{2, 3\} \end{aligned} \tag{1}$$

In addition to the cross-ratio to act as a regularizer, the loss has a squared error term. This forces the output to be as close as possible to the ground-truth annotation of the keypoints. The effect of the cross-ratio is controlled by the factor γ and is set to a value of 0.0001.

$$\begin{aligned} \Sigma_{i=1}^{7}(p_i^{(x)} - p_{i\_groundtruth}^{(x)})^2 + (p_i^{(y)} - p_{i\_groundtruth}^{(y)})^2 \\ +\gamma(Cr(p_1, p_2, p_3, p_4) - Cr_{3D})^2 \\ +\gamma(Cr(p_1, p_5, p_6, p_7) - Cr_{3D})^2 \end{aligned} \tag{2}$$

The second and third term minimize the error between the cross-ratio measured in 3D (Cr_{3D}) and the cross-ratio calculated in 2D based on the “keypoint regressor’s” output, indirectly having an influence on the locations output by the CNN. The second term in Equation 2 represents the left arm of the cone while the third term is for the right

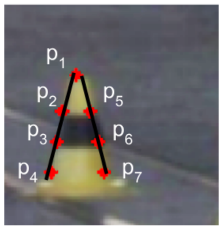

Fig. 6. An exemplary 80×80 cone patch with regressed "keypoints" overlaid in red. Depiction of the left (p_1, p_2, p_3, p_4) and right arm (p_1, p_5, p_6, p_7) of the cone. Both of which are used to calculate the cross-ratio terms and minimize the error between themselves and the cross-ratio on the 3D object (Cr_{3D}).

arm, as illustrated in Figure 6. For the cross-ratio, we choose to minimize the squared error term between the already known 3D estimate ($Cr_{3D} = 1.39408$ from a real cone) and its 2D counterpart. Equation 2 represents the loss function minimized while training the "keypoint regressor". The training scheme is explained in the following section.

4) Training: During the training, Stochastic Gradient Descent (SGD) was used for optimization, with $learning\_rate = 0.0001$ and $momentum = 0.9$ and a batch size of 128. The learning rate is scaled by 0.1 after 75 and 100 epochs. The network is trained for 250 epochs. The keypoint regressor is implemented in PyTorch and used via ROS on gotthard driverless. Refer to Section V-A.2 for more information about the dataset.

C. 2D to 3D Correspondences

The "keypoint network" provides with accurate locations of the specific features, the keypoints. Since, there is a priori information available about the shape, size, appearance and 3D geometry of the object, the cone in this case, 2D-3D correspondences can be matched. With access to a calibrated camera and 2D-3D correspondences, it is possible to estimate the pose of the object in question from a single image.

We define the camera frame as $\mathcal{F}_c$ and the world frame as $\mathcal{F}_w$. $\mathcal{F}_w$ can be chosen arbitrarily, as long. In this case, we choose the world frame, $\mathcal{F}_w$ to be at the base of every detected cone, for easy measurement of the 3D location of the keypoints (with respect to $\mathcal{F}_w$) and convenience of calculating the transform and eventually the cone position.

We use Perspective n-Point (PnP) to estimate the pose of every detected cone. This works by estimating the transform ${}^{c}\mathcal{T}_w$ between the camera coordinate system and the world coordinate system. Since, the world coordinate system is located at the base of the cone, lying at an arbitrary location (that we want to estimate) in $\mathbb{R}^3$ this transform is exactly the pose we are looking for. As we are concerned only with the translation between $\mathcal{F}_c$ and $\mathcal{F}_w$, which is exactly the position that we care to estimate, we can discard the orientation due to the cone's symmetric geometry.

To estimate the position of the cone accurately, we use the non-linear version of the PnP implemented in OpenCV [2] that uses Levenberg-Marquardt to obtain the transformation. In addition, RANSAC PnP is used instead of vanilla PnP, tackling noisy correspondences. RANSAC PnP can be done for each detected cone, i.e. extract the keypoints by passing the patch through the "keypoint regressor" and use the pre-computed 3D correspondences to estimate their 3D position. One can obtain the position of each cone in the car's frame by a transformation between the camera and the car.

TABLE I

Summary of datasets collected and manually annotated to train and evaluate different sub-modules of the pipeline. Acronyms: (1) OtfA?: On-the-fly augmentation?, (2) NA: Not applicable.

Task	Training	Testing	OtfA?
Cone Detection	2700	300	Yes
"Keypoint regression"	16,000	2,000	Yes
3D LiDAR position	NA	104	NA

V. Data Collection and Experiments

A. Dataset Collection and Annotation

To train and evaluate the proposed pipeline data for object detection and "keypoint regression" is collected and hand-labeled. To analyze the accuracy of the position estimates from a single image, ground-truth is collected using a LiDAR sensor.

1) Traffic cone detection: The object detector is trained on 90% of the acquired dataset, about 2700 manually-annotated images with multiple instances of cones and performance is evaluated on 10% of the data (about 300 unseen images).

2) "Keypoints" on cone patches: 900 cone patches were extracted from full images and manually hand-labeled using a self-developed annotation tool.

Data Augmentation. The dataset was further augmented by transforming the image with 20 random transforms. These were a composition of random rotations between $[-15°, +15°]$, scaling from 0.8x to 1.5x and translation of up to 50% of edge length. During the training procedure, the data is further augmented on the fly in the form of contrast, saturation and brightness. The annotated dataset is partitioned to have 16,000 cone patches for training and the remaining 2,000 cone patches for testing.

3) 3D ground-truth from LiDAR: In order to test the accuracy of the 3D position estimates, corresponding object positions measured from a LiDAR are treated as ground-truth. This is done for 104 physical cones at varying distances from 4 meters up to 18 meters. The estimates are compared in Figure 9 and are summarized in Section V-D.

This section analyses and discusses results of the monocular perception pipeline, paying special attention to the robustness of the "keypoint network" and the accuracy of 3D position estimates using the proposed scheme from a single image. The "keypoint network" can process multiple cone patches in a single frame in 0.06s, running at 15-16FPS on the Jetson TX2 while running other sub-modules of the pipeline and handling 1 Gb/s of raw image data.

B. Cone Detection

Table II summarizes the performance evaluation of the cone detection sub-module. The system has a high recall

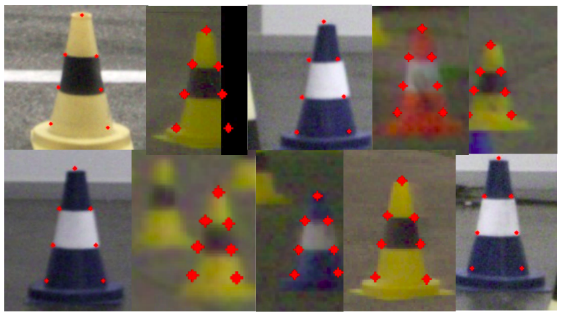

Fig. 7. Robust performance of “keypoint regression” across scenarios. Refer to Section V-C for analysis.

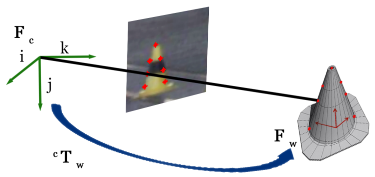

Fig. 8. Schematic illustrating matching of 2D-3D correspondence and estimation of transformation between the camera frame and the world frame.

TABLE II
Performance of YOLO for colored cone object detection.

	Precision	Recall	mAP
Training	0.85	0.72	0.78
Testing	0.84	0.71	0.76

and is able to retrieve most of the expected bounding boxes. With a high precision it is averse to false detections which is necessary to keep the car on track. Figure 3 illustrates the robustness of the cone detection pipeline in different weather and lighting conditions. The detections are shown by colored bounding boxes. The key to driving an autonomous vehicle successfully is to design a system that has no false positives. Incorrect detections can lead to cone (obstacle) hallucination forcing the car off-course.

C. Keypoint Regression

Figure 7 illustrates a montage of 10 sample patches regressed for keypoints after being detected by YOLOv2. There are a couple of interesting cases here. The second cone (from the left) in the top row is detected on the right edge of the image and is only partially visible on the image and has almost a third of it truncated (padded with black pixels). Even with missing pixels and no information about a part of the cone, the “keypoint regressor” regresses correctly. It learns the geometry and relative location of one keypoint with respect to another. Even by just partially observing a cone, it approximates where the keypoint *would have been* in case of a complete image. Its ability to understand spatial arrangement of the keypoints and their geometry through data and by minimizing the loss is quite interesting. Another interesting observation is the second cone (from the left) in the bottom row. Here, there is a cone in the back-ground but the keypoint network is able to regress to the most prominent cone. The error (equation 2) for regressing the keypoints on train and test data is summarized in Table III.

TABLE III
Performance of “keypoint regressor” on training and testing datasets.

	Training	Testing
MSE	3.535	3.783

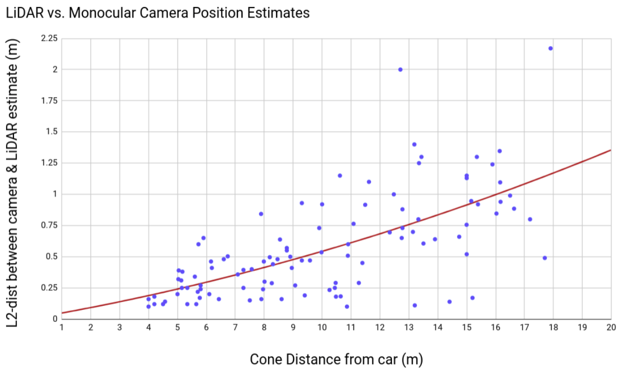

Fig. 9. Euclidean distance between position estimates from LiDAR and monocular camera pipeline for the same physical cone. The x-axis represents the distance of the cone from the car and the y-axis the difference between the LiDAR and camera estimates.

Figure 7 shows the robustness and accuracy of the “keypoint regressor”, but it represents only the internal performance of the “keypoint regression” sub-module. In the following subsections, we analyze how outputs of intermediate sub-modules affect the 3D cone positions. We also show how variability in output of a particular sub-module ripples through the pipeline and influence the final position estimates.

D. 3D position accuracy

As it is deployed on a real-time self-driving car, one of the most crucial aspect is the accuracy of the estimated 3D positions. We compare the accuracy of the pipeline against the LiDAR’s 3D estimates, which is treated as ‘ground-truth’.

Figure 9 is plotted using data from 2 different test tracks. The x-axis represents the depth, in meters, of a physical cone and along the y-axis is the Euclidean distance between the 3D position from the LiDAR estimates and the 3D position from the monocular camera pipeline. The plot consists of 104 data physical cones. Further, a second order curve is fitted to the data, which has mostly linear components. On average, the difference is about 0.5 meters at 10 meters distance away from the car and only about 1 meter at 16 meters. At 5 meters the cone position is off by $\pm 5.00\%$ of its distance, and at 16 meters, it is off by only $\pm 6.25\%$ of its distance. The error is small enough for a self-driving car to drive on a track flanked by cones at speeds higher than 50kmph.

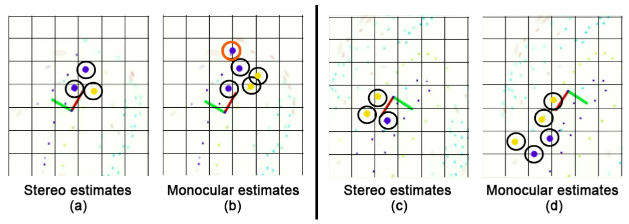

Fig. 10. 3D cones using computer vision are depicted by solid yellow and blue circles highlighted by black circles. In the first panel, (a) & (b), "gotthard driverless", shown as a coordinate system, with red axis pointing forward, approaches a sharp hair-pin turn. The monocular pipeline perceives a blue cone on the other side of the track (marked with an orange circle), allowing SLAM to map distant cones and tricky hair-pins. In the second panel, (c) & (d), the car approaches a long straight. By perceiving longer distances, the car can push harder and accelerate hard. Each grid cell depicted here is $5m \times 5m$.

E. Extended perception range

One of the goals of the method is to have an extended range of perception. In Figure 10 we compare the difference between the ranges of the monocular and the stereo pipeline. We find that our solution based on monocular cameras has larger perception range than the standard triangulation solution based stereo cameras. The larger perception range is mainly due to the fact that monocular cameras have a longer focal lengths than stereo cameras. If the stereo pair also have longer focal length, the field of view reduces which introduces a lot of blind-spots where the cameras cannot see.

F. Effect of 2D bounding boxes on 3D estimates

As mentioned before, we would like to see how sub-modules have an effect on the final 3D position estimates. Here, we take a step back and analyze how variability in output from the object detection sub-module (imprecise bounding boxes) would influence the 3D positions. In this experiment, we randomly perturb the bounding box edges by an amount proportional to the height and width of the bounding box in respective directions. Estimating depth is most challenging using raw data from cameras. Figure 11 shows how for single images, perturbing the boxes by a certain amount ($\pm 1\%$, $\pm 5\%$, $\pm 10\%$ and $\pm 20\%$) influences the variance in depth estimates. As expected, for higher amounts of perturbation, more variance in depth estimates is observed. However, even for a $\pm 20\%$ perturbation, the variance is about 1 meter^2 at 15 meters. Figure 11 shows that even with imprecise and varying bounding boxes, the depth of the cone is consistent and has low variance.

For additional visualizations of the detection, regression, 3D position estimation, mapping and final navigation please visit the project page `http://people.ee.ethz.ch/~tracezuerich/TrafficCone/`

VI. Conclusion

Accurate, real-time 3D pose estimation can be used for several application domains ranging from augmented reality to autonomous driving. We introduce a novel "keypoint regression" scheme to extract specific feature points by leveraging geometry in the form of the cross-ratio loss term. The approach can be extended to different objects to estimate 3D pose from a monocular camera and by exploiting object structural priors. We demonstrate the ability of the network to learn spatial arrangements of keypoints and directly translating to robust regression even in challenging cases. To demonstrate the effectiveness and accuracy, the proposed pipeline is deployed on an autonomous race-car. The proposed network runs in real-time with 3D position deviating by only 1m at a distance of 16m.

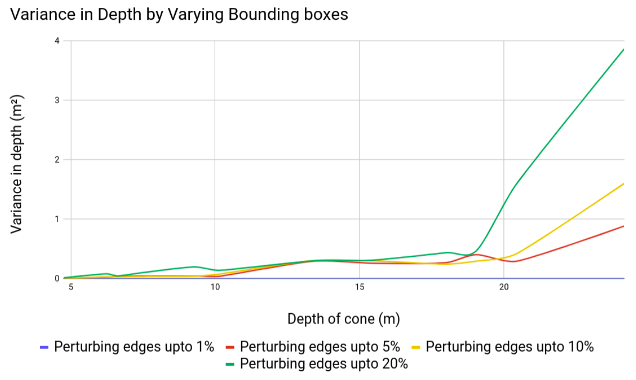

Fig. 11. The variance observed in depth of cones when perturbing the bounding boxes that are input to the "keypoint regressor". On the x-axis is the depth of the cone while the y-axis represents the variance in the cone's depth estimate. Even with imprecise and inaccurate patches, the "keypoint regressor" keeps the depth variance quite low.

Acknowledgement. We would like to thank AMZ Driverless team for their support. The work is also supported by Toyota Motor Europe via the project TRACE-Zurich.

References

[1] Cross ratio. `http://robotics.stanford.edu/~birch/projective/node10.html`.
[2] OpenCV calibration and 3d reconstruction. `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp`.
[3] PyTorch. `https://pytorch.org/`.
[4] ROS: Robot Operating System. `https://ros.org/`.
[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
[6] D. Dai and L. Van Gool. Dark model adaptation: Semantic image segmentation from daytime to nighttime. In *IEEE International Conference on Intelligent Transportation Systems*, 2018.
[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.
[8] A. Fregin, J. Mller, and K. Dietmayer. Three ways of using stereo vision for traffic light recognition. In *IEEE Intelligent Vehicles Symposium (IV)*, 2017.
[9] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
[11] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. Using k-poselets for detecting people and localizing their keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3582–3589, 2014.

[12] D. Glasner, M. Galun, S. Alpert, R. Basri, and G. Shakhnarovich. aware object detection and continuous pose estimation. *Image and Vision Computing*, 30(12):923–933, 2012.
[13] C. Harris and M. Stephens. A combined corner and edge detector. Citeseer, 1988.
[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[15] S. Hecker, D. Dai, and L. Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. In *European Conference on Computer Vision (ECCV)*, 2018.
[16] M. B. Jensen, M. P. Philipsen, A. Mgelmose, T. B. Moeslund, and M. M. Trivedi. Vision for looking at traffic lights: Issues, survey, and perspectives. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):1800–1815, 2016.
[17] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
[18] H. S. Lee and K. Kim. Simultaneous traffic sign detection and boundary estimation using convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 2018.
[19] J. Levinson, J. Askeland, S. Thrun, and et al. Towards fully autonomous driving: Systems and algorithms. In *IEEE Intelligent Vehicles Symposium (IV)*, 2011.
[20] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan. Perceptual generative adversarial networks for small object detection.
[21] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
[22] M. Mathias, R. Timofte, R. Benenson, and L. V. Gool. Traffic sign recognition how far are we from the solution? In *International Joint Conference on Neural Networks (IJCNN)*, 2013.
[23] R. McAllister, Y. Gal, A. Kendall, M. Van Der Wilk, A. Shah, R. Cipolla, and A. Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *International Joint Conference on Artificial Intelligence*, 2017.
[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
[25] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.
[26] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
[27] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.
[28] A. Ruta, F. Porikli, S. Watanabe, and Y. Li. In-vehicle camera traffic sign detection and recognition. *Machine Vision and Applications*, 22(2):359–375, Mar 2011.
[29] C. Sakaridis, D. Dai, and L. Van Gool. Semantic foggy scene understanding with synthetic data. *International Journal of Computer Vision*, 2018.
[30] S. Savarese and L. Fei-Fei. 3d generic object categorization, localization and pose estimation. 2007.
[31] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
[32] S. Tulsiani and J. Malik. Viewpoints and keypoints. *CoRR*, abs/1411.6067, 2014.
[33] C. Urmson, J. Anhalt, H. Bae, J. A. D. Bagnell, and et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466, June 2008.
[34] A. Valada, J. Vertens, A. Dhall, and W. Burgard. Adapnet: Adaptive semantic segmentation in adverse environmental conditions. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4644–4651. IEEE, 2017.
[35] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.
[36] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017.
[37] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.