Automatic extrinsic camera parameters calibration using Convolutional Neural Networks

Razvan Itu, Diana Borza, Radu Danescu
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
razvan.itu@cs.utcluj.ro

***Abstract*—Camera calibration is essential for accurate computer vision, and automatic calibration of some extrinsic parameters is needed in case the camera is placed on a mobile platform. The pitch and yaw angles, which are the most likely ones to change as the vehicle moves, can be inferred from the image coordinates of the vanishing point (VP). In this paper we present an artificial neural network approach for detecting the vanishing point position in road traffic scenarios. The network is trained using 2500 images which are first automatically annotated using a classical vanishing point detection algorithm, and then manually validated. The training and test datasets are made publicly available. The trained network was tested on more than 250 images not previously seen by the network, locating the VP accurately in more than 90% of the cases.**

Keywords—vanishing point; convolutional neural networks; calibration; automatic; camera; extrinsic; pitch; yaw

I. INTRODUCTION

Camera calibration is essential for accurate computer vision applications that involve reasoning about the 3D world. Such applications include sensing the environment for mobile robots, driving assistance applications, or surveillance. The most robust techniques for inferring the 3D information from images are based on stereovision, but monocular vision can also be used in limited applications, relying on assumed properties of the environment (such as flat road, known width of obstacles or lanes, etc).

Both stereo and monocular environment perception techniques need accurate camera calibration in order be able to retrieve the 3D information. Camera calibration is usually a static process, involving reference objects, accurately measured, in a controlled environment. However, in the case of vision applications that involve mobility (mobile robots, driving assistance), the parameters that are calibrated can quickly change. This is especially true in the case of consumer cameras, or smartphones, placed in vehicles, which do not benefit from a professional mounting system. The angles of the camera with respect to the reference system of the world (pitch, yaw and roll) can change rapidly as the mobile platform moves in the 3D world, and automatic, real time calibration is needed to cope with this problem.

This Vanishing Point (VP, the point of the perspective image where the parallel lines in the 3D scene intersect) can be used to extract valid information regarding the extrinsic parameters of the camera, such as the pitch and yaw angles. Also, a reliable detection of the vanishing point is important in the case of monocular driving assistance applications, as the vanishing point is also the further point of the road surface, and the intersecting point of lane markings.

In this paper we propose a novel method, based on a convolutional neural network (CNN), to predict the position of the vanishing point. The solution was tested against a classical VP detection solution, using our own annotated dataset which includes 2684 images.

The paper is structured as follows: section 2 presents recent and similar work for detecting vanishing points, section 3 explains the dataset preparation, section 4 presents the neural network and the architecture of the proposed system, and also presents the computation of the pitch and yaw angles from the detected VP, section 5 presents the testing procedure and results and finally, section 6 describes the conclusion and further work.

II. RELATED WORK

Recent work in vanishing point detection is based solely on computer vision algorithms that analyze the texture [1], [2] or geometric properties such as edges and lines in the image [3], [4] or [5]. Usually edge based methods perform well only in scenarios where the road surface is clear and well delimited by painted lane markings or borders. In [3] the authors present a method based on Gabor filters with different orientations in order to estimate the main orientation of edges. A voting scheme is used to update a vanishing point accumulator for multiple vanishing point locations. This approach works well for unstructured roads where road markings might not be present. Texture based methods, such as [1], [2] or [3] generally work well on both structured and non-structured roads. These methods use texture information and have similar steps as the edge based methods: determine the orientation of the textures by applying various filters, determine vanishing point candidates and voters and finally, applying a voting scheme to obtain a vanishing point. In the literature, local voting schemes have been proposed to increase the accuracy of the voting by introducing confidence levels.

978-1-5386-3368-7/17/$31.00 ©2017 IEEE

Road surface segmentation using CNNs has received vast research interest in the past years. Using such approaches, the drivable road surface can be estimated from neural networks [6], [7]. The CNNs are used to perform pixel wise predictions regarding the scene. From a segmented image which includes a drivable road the vanishing point can be found as the top of the road surface. However, this approach is usually computationally expensive and imprecise due to pixel-wise segmentation errors, and assumes that the road is always flat, or that the vanishing point is always a visible road point in the image.

Most vanishing point detection methods based on traditional computer vision algorithms rely on pixel information and voting, meaning that such systems are generally computationally expensive.

III. Dataset preparation

We have used traditional computer vision algorithms for computing the ground truth vanishing points, the method is based on [5]. We process images taken from OpenStreetCam [8] and create a dataset annotated with the following data: image, text file and image vote map. The text file contains the image name, the vanishing point x and y coordinates obtained by the algorithm, the tracked vanishing point x and y coordinates obtained by a tracking algorithm. The tracking algorithm is based on a ring buffer where the most recent 6 frames are stored and the median value is selected. This introduces a delay in the vanishing point values, but overall produces better results and filters out cases when the classical algorithm might fail. The images are of a 4:3 scale and size of 640 x 480 pixels. We chose to crop the top part of the image in order to avoid unnecessary processing of irrelevant data (such as sky and trees) and also to improve the overall performance, both the training time and the prediction time. Then the images are resized and the final size of the input is actually 160 x 48 pixels. The initial dataset consists of 2828 total images and labels, of which 2233 represent highway scenarios and 595 for city. The dataset was adjusted so that negative y coordinates of the vanishing point are removed after the image resizing and cropping, resulting in a dataset containing 2090 highway images and labels and 594 images and labels for city, resulting a total size of 2684. The initial dataset is further split into training (90%) and test (10%).

As the automatic detection of the vanishing point is prone to errors, we have also analyzed the processed sequences manually, and corrected the wrong results.

One of the properties of CNNs is that they work better when the training process uses a lot of data. One of the easiest ways of generating more training data is to augment the existing annotated data. This can be done in multiple ways, such as: random image flips, translations, applying rotations or even by adjusting the brightness and contrast of the input data. In our case, we have applied a horizontal flip on the entire dataset and we have adjusted the vanishing points accordingly. The resulted final training dataset includes 4830 images, with their annotated vanishing points and the test dataset consists of 269 images and annotations. During training phase of the neural network, the training set is further split in two: training 90% and validation 10%. The distribution of all the vanishing points in the image space is illustrated in figure 1.

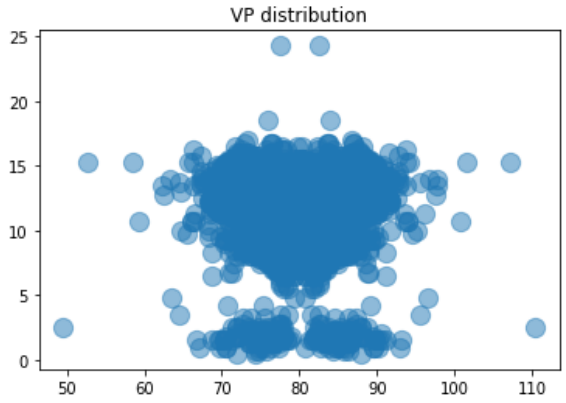

Fig. 1. Vanishing point distribution for the training dataset.

IV. The proposed solution

This section presents the proposed solution for vanishing point estimation, based on a convolutional neural network.

A. Solution overview

We present an original vanishing point detection framework based on convolutional neural networks. The framework is inspired from [9], where Nvidia used a neural network to predict the steering angle of a vehicle directly from a single image. A system overview of our framework is presented in figure 2.

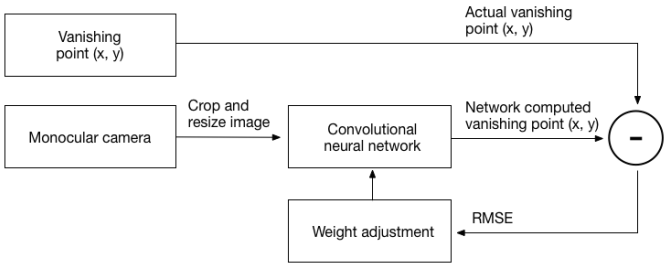

Fig. 2. System overview

We deploy a convolutional neural network (CNN) in order to process cropped and resized input images that were acquired using monocular camera systems. We then predict the vanishing point position and compare it using the ground truth vanishing point.

B. Convolutional neural network architecture

Convolutional neural networks have been widely used in the past years, especially since computational power has become more affordable. Notable changes occurred in 2012 when [11] used a CNN for the ImageNet classification challenge and won by a large margin. The classical steps of image based classification, such as: feature pre-processing, extraction and classification are replaced by neural networks that automatically learn the relevant features.

Our chosen CNN architecture is illustrated in figure 3.

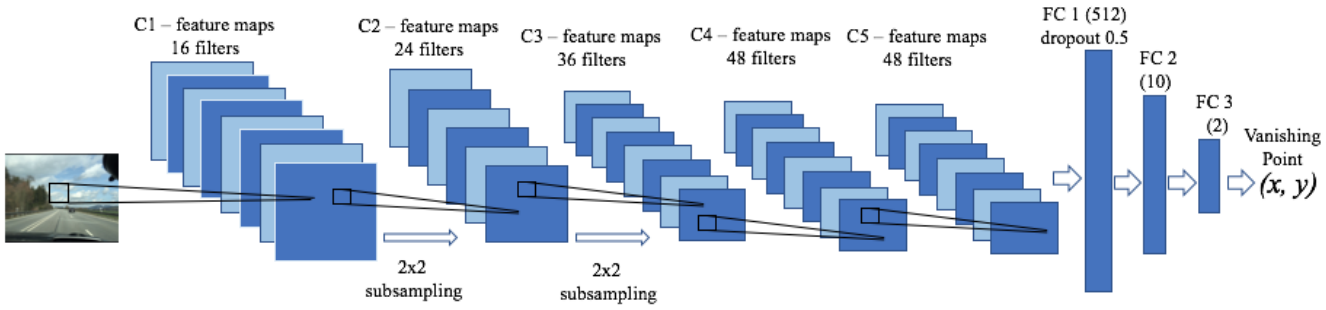

Fig. 3. Proposed convolutional neural network architecture.

For our network, the images are cropped and then scaled down to 160 x 48 resolution and fed directly as input to the neural network. The layers are organized as follows:

- First layer represents batch normalization (B)
- The second layer is actually the first convolution layer (C1), with 16 filters using a 3x3 kernel
- The third layer is also a convolution layer (C2), with 24 filters using a 3x3 kernel
- The fourth layer represents another convolution (C3), with 36 filters using a 3x3 kernel
- The fifth layer is also a CONV (C4) with 48 filters using a 2x2 kernel
- The sixth layer is the last CONV (C5) featuring 48 filters with a 2x2 kernel
- The next layer is a flatten operation that flattens the input
- The 7th layer represents the first fully-connected layer (FC1) that contains 512 neurons, followed by a 0.5 dropout with ReLU as activation function
- The 8th layer is another FC layer (FC2) with 10 neurons with ReLU as activation function
- The final layer represents a fully connected layer (FC3) with 2 neurons

Batch normalization is used in order to achieve faster learning and increase the overall accuracy of the network. The convolutional layers will feature filters with weights adjusted so that the output images will work similar to line detectors.

The 2 neurons from the final layer actually represent the values for the predicted vanishing point: the *x* and *y* coordinates.

The response of the first convolutional layer (with its 16 3x3 filters) for a sample input image is shown in figure 4.

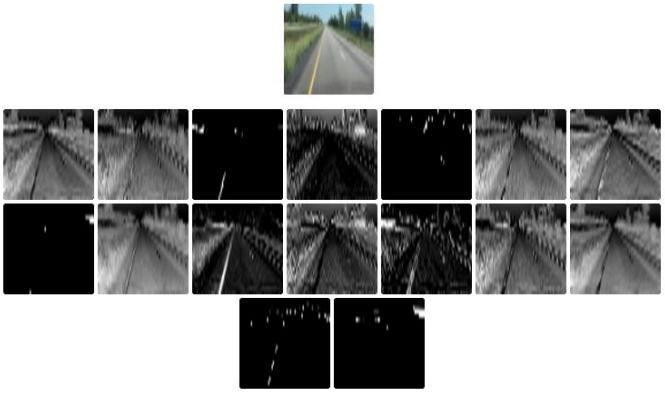

Fig. 4. Output of the first convolutional (C1) layer for a sample input image.

C. CNN training

We have trained the network in 20 epochs, using a batch size of 256 images. In figure 5 we plot the loss function evolution on training and test datasets. As seen in the figure, after 5 epochs the loss function stops converging, therefore training can be stopped earlier.

The loss function of the network is defined as the root mean squared error (RMSE) between the ground truth and the predicted position of the vanishing point. The network's weights are adjusted by using back-propagation by gradient descent and using an Adam optimizer [10] with a learning rate of 0.001.

As described in section 3, we have split the dataset into training and test (90% - 10%). From the remaining training set we have further obtained the validation set (90% training - 10% validation) that is used during the training phase.

Training the network on a GPU powered machine usually takes around 2 seconds per epoch, whereas on a CPU based machine it takes around 20 seconds per epoch. Therefore, the use of a GPU can bring significant speedup, but the system is capable of running in real time (for the prediction) also on a decent CPU.

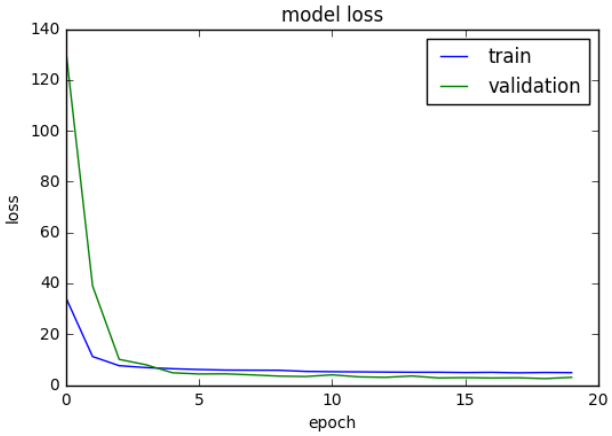

Fig. 5. Loss function result evolution while training, computed using training and validation sets.

D. VP prediction, pitch and yaw computation

The network is trained by using images with annotated vanishing points. After training, the network can predict the image coordinates of the VP on new, unseen images. An example of VP prediction, compared with ground truth, is presented in figure 6.

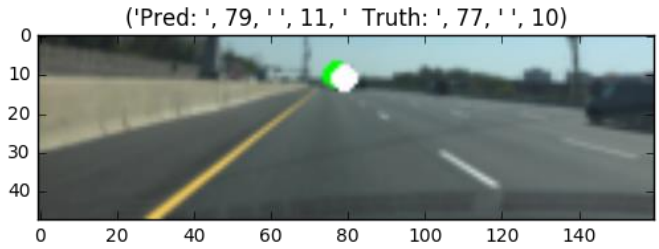

Fig. 6. Example of a predicted vanishing point. The green circle represents the ground truth and the white circle is the prediction.

From the detected VP coordinates, we can extract the pitch and yaw angles of the camera with respect to the road, using the following equations:

$$pitch = original\ pitch + \Delta pitch \tag{1}$$

where Δ pitch is obtained as follows:

$$\Delta pitch = \frac{\Delta y}{focal\ length} \tag{2}$$

$$\Delta y = original\ vp.y - vp.y \tag{3}$$

The equations for computing the yaw angle are similar to the ones for the pitch, with the only difference that we use the x vanishing point coordinate coordinate. Pitch and yaw angles provide relevant information about the camera pose in the 3D world scene.

The original pitch and yaw angles are the initial known angles of the camera, which can be calibrated in static conditions when the camera is set up on the mobile platform. If such calibration cannot be performed, these angles can be assumed to be zero, and the pitch and yaw differences computed from VP differences become the actual angle values.

The original vanishing point *x* and *y* coordinates can be computed by projecting a 3D point situated far away on the Z-axis (in a left hand rule coordinate system, where the Z-axis points in front of the ego-vehicle) into the 2D image space. Projection of the theoretical vanishing point is done using a projection matrix where the pitch, yaw and roll angles can be either known from a static calibration process, or assumed to be equal to 0. The focal length is expressed in pixels, and is adjusted for the resized input image.

The roll angle (lateral inclination of the platform) does not affect the vanishing point, and therefore cannot be computed from the VP coordinates. Therefore, this angle is assumed to be zero, if no other means of computing it can be used.

The pitch and yaw angles are then used for computing the extrinsic camera rotation matrix, which is further used to generate the projection matrix that related the 3D world to the image plane. The projection matrix can be used to infer 3D information from the image features (under several assumptions, if monocular vision is used), or can be used for removing the perspective effect, and to generate a bird eye's view image, like in [12].

V. Tests and Evaluation

The evaluation dataset includes 59 images of city traffic and 209 images of highway driving. The mean error values obtained after testing the prediction is presented in table 1.

To evaluate our results, we have used two metrics: the root mean squared error (RMSE) between the predicted and actual (ground truth) vanishing point and a similar metric named: *NormDist* that divides the RMSE by the image diagonal (so that the metric is image-independent). The *NormDist* is actually the distance between the vanishing points divided by the image diagonal, as expressed in equation 4: where *VPP* represents the predicted vanishing point, *VPG* is the ground truth vanishing point and *diag* represents the image diagonal expressed in pixels.

$$NormDist = \frac{||VPP - VPG||}{diag} \tag{4}$$

The *NormDist* distance metric is generally used for computing the VP error in [4] and also in [5]. We ran prediction tests on random batch of images from both datasets and also on the entire city and highway dataset.

TABLE I. Error computation on dataset

Error metric	Random data	City	Highway
NormDist	0.04458	0.05072	0.02612
RMSE	5.19591	8.47350	4.36344

We have also computed the *NormDist* error on each image in the city and highway datasets and the results are displayed in figure 7.

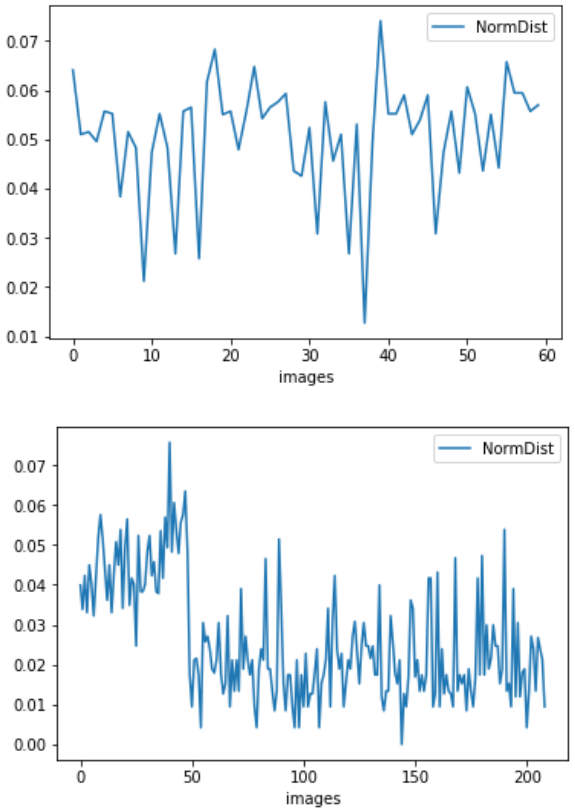

Fig. 7. NormDist error computed on individual images in both city (top) and highway (bottom) test datasets.

Table 2 represents an analysis on the RMSE metric that was computed on the entire vanishing point dataset. We have computed how many predictions have a RMSE value lower than 2, between 2 and 3, between 3 and 5, between 5 and 10 and over 10. The table presents the percentage from the total number of images.

From table 2 we can clearly see that the city dataset has a larger uncertainty, whereas the highway predictions are more accurate (over 76% of the predictions have a RMSE smaller than 3 and over 93% have a RMSE smaller than 5). The higher uncertainty from the city dataset is caused by the presence of a higher number of vehicles and other obstacles in front of the ego vehicle, which make the road surface and the painted road markings less visible.

Sometimes the predicted vanishing point is better than the one obtained from the classic algorithms, as can be seen in figure 8. This is usually caused by additional obstacles in front of the ego vehicle which can influence the results of traditional algorithms.

Fig. 8. Example of a predicted vanishing point that is better than the vanishing point computed by traditional algorithms. The green circle represents the ground truth and the white circle is the prediction.

Figure 9 illustrates more prediction examples on random images from the datasets. As opposed to the previous example, the predicted value is not always accurate (for example, the middle image in the last row of the 2nd column).

TABLE II. RMSE ANALYSIS ON CITY AND HIGHWAY DATASETS

City (594 total images)				**Highway (2090 total images)**			
interval	**count**	**interval**	**percent**	**interval**	**count**	**interval**	**percent**
< 2	68	< 2	11.4 %	< 2	1122	< 2	53.6 %
[2 - 3)	65	< 3	22.3 %	[2 - 3)	487	< 3	76.9 %
[3 - 5)	215	< 5	58.5 %	[3 - 5)	340	< 5	93.2 %
[5 - 10)	238	**< 10**	**98.6 %**	[5 - 10)	139	**< 10**	**99.9 %**
> 10	8	> 10	1.34 %	> 10	2	> 10	0.09 %

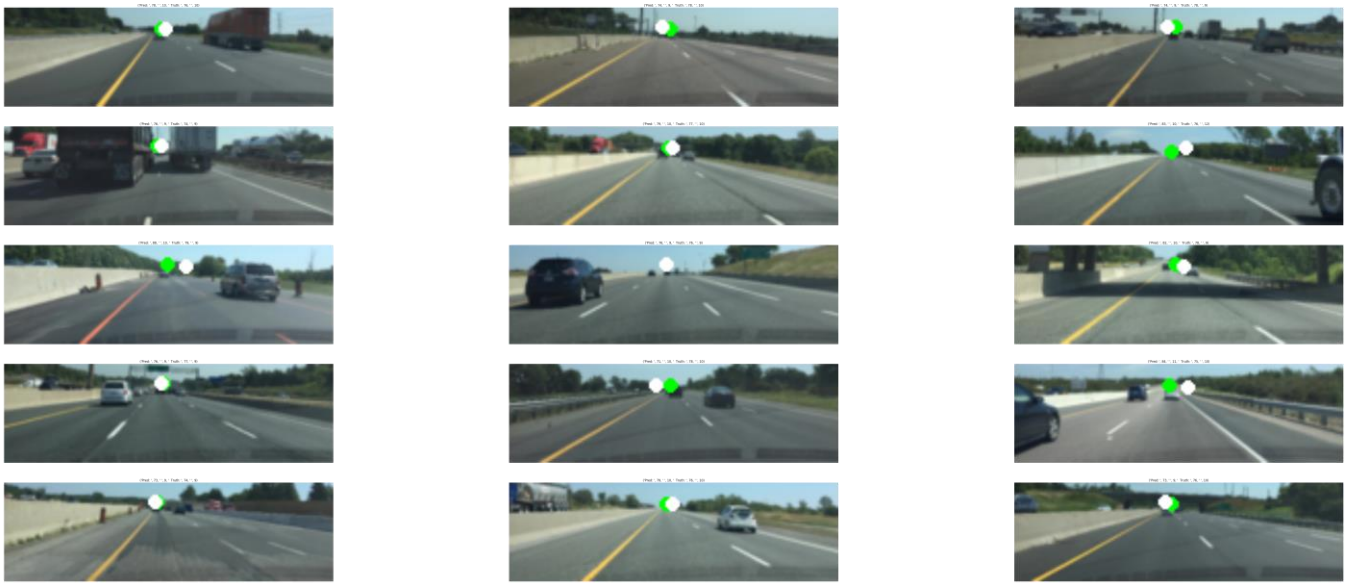

Fig. 9. Comparison between network predicted vanishing point and ground truth vanishing point. The green circle represents the ground truth and the white circle is the prediction.

The processing time for making prediction is close to an average of 1.7 milliseconds per image on an Intel i7 6700K (desktop CPU) and Nvidia GTX 1080 Ti GPU. Tests on the same dataset were performed on a laptop with Intel i7 3615QM (laptop CPU) and the prediction time is close to 100 milliseconds per image. Therefore, the system is capable of computing the vanishing point position in real time on a decent CPU.

VI. Conclusion and Future Work

This paper presents a novel method for obtaining vanishing points using a CNN and a unique dataset that is used to compute the pitch and yaw of a monocular camera system. Even though the metrics and results are good, there are a few improvements that can be done. One major aspect is to compute individually, on each image a proper start index for cropping the top part. Another improvement that can be done is regarding the dataset: we can further increase the amount of data using new trips and using our own acquisition system. Augmentation can be of good help, but new images can be better for training neural networks.

The vanishing point dataset is publicly available at [13] and consists of multiple trips organized into 2 folders: city and highway. Inside of each trip folder there is a text file corresponding to individual images. The text file contains: vanishing point x and y coordinates, the tracked vanishing point x and y coordinates and the web address of the image file.

Automatic self-calibration systems are very useful, but not limited to, the self-driving cars field. By using our method, we can automatically compute most of the extrinsic camera parameters (pitch and yaw angles) of a monocular camera system that is used in driving scenarios.

References

[1] Bui, T.H., Nobuyama, E., Saitoh, T., “A texture-based local soft voting method for vanishing point detection from a single road image”. IEICE Trans. Inf. Syst. 2013, E96-D, 690 - 698.

[2] Bui, T.H.; Saitoh, T., Nobuyama, E., “Road Area Detection Based on Texture Orientations Estimation and Vanishing Point Detection”. In Proceedings of the 2013 SICE Annual Conference (SICE), Nagoya, Japan, 14 - 17 September 2013; pp. 1138 - 1143.

[3] Kong, H., Audibert, J.-Y., Ponce, J., “Vanishing point detection for road detection”. Comput. Vis. Pattern Recognit, 2009, 96–103.

[4] Moghadam, P., Starzyk, J.A., Wijesoma, W.S., “Fast vanishing-point detection in unstructured environments”. IEEE Trans. Image Process. 2012, 21, 425 - 430.

[5] Wu, Z., Fu, W., Xue, R., Wang, W., “A Novel Line Space Voting Method for Vanishing-Point Detection of General Road Images”. Sensors 2016, 16, 948.

[6] Alvarez J.M., Gevers T., LeCun Y., Lopez A.M., “Road Scene Segmentation from a Single Image”, ECCV 2012.

[7] Badrinarayanan V., Kendall A., Cipolla R., “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”, eprint arXiv:1511.0056, 2015, online: https://arxiv.org/abs/1511.00561

[8] OpenStreetCam, online: https://www.openstreetcam.org/

[9] Bojarski M., et. al., “End to End Learning for Self-Driving Cars”, eprint arXiv:1604.07316, 2016, online: https://arxiv.org/abs/1604.07316

[10] Kingma J. Ba, “Adam: A method for stochastic optimization”, ICLR 2015.

[11] Krizhevsky A., Sutskever I., Hinton G.E., “Imagenet classification with deep convolutional neural networks”. Advances in neural information processing systems. 2012, 1097 - 1105.

[12] Danescu R., Itu R., Petrovai A., “Generic Dynamic Environment Perception Using Smart Mobile Devices”. Sensors 2016, 16, 1721.

[13] Vanishing Point public dataset, online: http://users.utcluj.ro/~razvanitu/VPdataset.zip