Dynamic 3D Environment Perception Using Monocular Vision and Semantic Segmentation

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

***Abstract*—This paper presents a complete system for traffic environment perception based on a single color image source. The acquired color images are segmented into road and obstacle areas using a U-Net style Convolutional Neural Network (CNN). The segmented image is mapped into a bird's-eye view using the automatically calibrated camera parameters. Obstacle scans are extracted from the bird's-eye view image, highlighting the contact points between the obstacles and the road. The scans are post-processed to increase the connectivity between obstacle parts. The processed scans are used to generate the measurement likelihood values for all observable cells of a dynamic occupancy grid, taking into consideration the expected measurement errors with respect to the distance from the camera. A particle-based occupancy grid is used to track the environment at cell level, and then the occupied cells are grouped into individual objects. The system is able to estimate stable cuboids with measured width, length, orientation and speed for the moving individual objects such as vehicles and pedestrians, and also to identify generic occupied areas for the continuous structures such as fences or barriers.**

Keywords—color monocular vision; semantic segmentation; occupancy grids; measurement model; obstacle detection

I. Introduction

Modern object detection systems used for driving assistance or autonomous driving rely on various perception sensors used in conjunction with each other. Such complex systems may make use of multiple cameras, lidar, radar or combinations of them [1]. The main disadvantage of these systems is the cost factor: accurate and well-performing perception sensors are not cost effective, and they may sometimes be more expensive than the vehicle itself. Another disadvantage is that they require special calibration and must be synchronized with each other. Stereo-vision based systems might be more cost efficient, but they also rely on special calibration that is usually performed in controlled environments.

Monocular vision is a rich source of information about the environment. The images can be used to detect and classify the road features and obstacles alike, the only downside being that there is no direct 3D information about the environment, which lead to less precise measurement and sometimes to false or incomplete detection. The recent emergence of deep learning techniques brings improvements to monocular vision by improving the detection of environment elements, and even by inferring 3D information from single images.

We propose a system that is based on CNNs to detect the object candidates, infers the 3D information by analyzing the bird's eye view image. Due to the usage of different cameras with various intrinsic parameters, we rely on a self-calibration algorithm in order to automatically extract the camera parameters and to construct the inverse perspective mapping image. The detected objects from the segmented image are tracked using an occupancy grid based particle filter, and finally cuboids are extracted from groups of cells from the grid.

II. Related work

Scene understanding plays a key role in developing perception systems for autonomous vehicles or robots. Using convolutional neural networks for the task of semantic segmentation has been widely popular in recent years, especially since the introduction of the "Fully Convolutional Networks" [2]. The input of such a network is the color image, whereas the output represents the segmented image, each channel representing a different object class. In [3] the authors present a CNN architecture called "U-Net" which became very popular, and is also based on an "encoder-decoder" architecture with the same number of layers in the encoding and decoding sub-networks. Semantic segmentation CNNs are trained using pairs of images consisting of the color image (the observed scene) and the labeled image (where each object class has a different color).

CNNs can be trained to estimate depth maps from monocular color images. This can be achieved by training them using stereo-vision based datasets. In [4], the authors present an approach to generate such a depth (disparity) image using a CNN. In [5] the authors train a neural network to produce the left image from the right image, given the pair of left-right images acquired from a stereo-camera rig. They also generate the right image from the left one, and the network is trained using what is called "left-right consistency" loss that validates each generated image. Other approaches try to generate the 3D information of the objects in the scene. In [6], authors train an artificial neural network to predict the 3D bounding box of vehicles from a road scene, given their image space detection rectangle. The same idea is improved in [7], where the authors use a different approach for regressing the local orientation. In [8] the authors propose a different approach using a CNN, based on the idea that reasoning about the obstacle location can be performed in the inverse perspective mapping (IPM) image and not in the perspective image. The authors introduce an orthographic feature transform that can map the features extracted from the color image into the bird's eye view image.

III. Solution Overview

The proposed system tracks the observed scene continuously, based on the following data: a color input image, speed and yaw rate information for the ego vehicle, and the timestamps of the acquired frames.

The acquired image is segmented into road and obstacle areas by means of a CNN. The image regions that are recognized as road areas are used for automatic camera calibration based on road markings, and the obstacle areas are used for generic obstacle recognition. Using the automatically calibrated camera parameters, the segmented image is mapped into a bird's-eye view image. Obstacle delimiters are extracted by analyzing the bird's-eye view image, and based on them a measurement probability model is created, which takes into account the error model of the bird's-eye view based distance measurement. The measurement model is used to update a dynamic occupancy grid based on moving particles, and finally the connected regions from the occupancy grid are grouped into cuboid shape objects.

The whole process is depicted in Fig. 1.

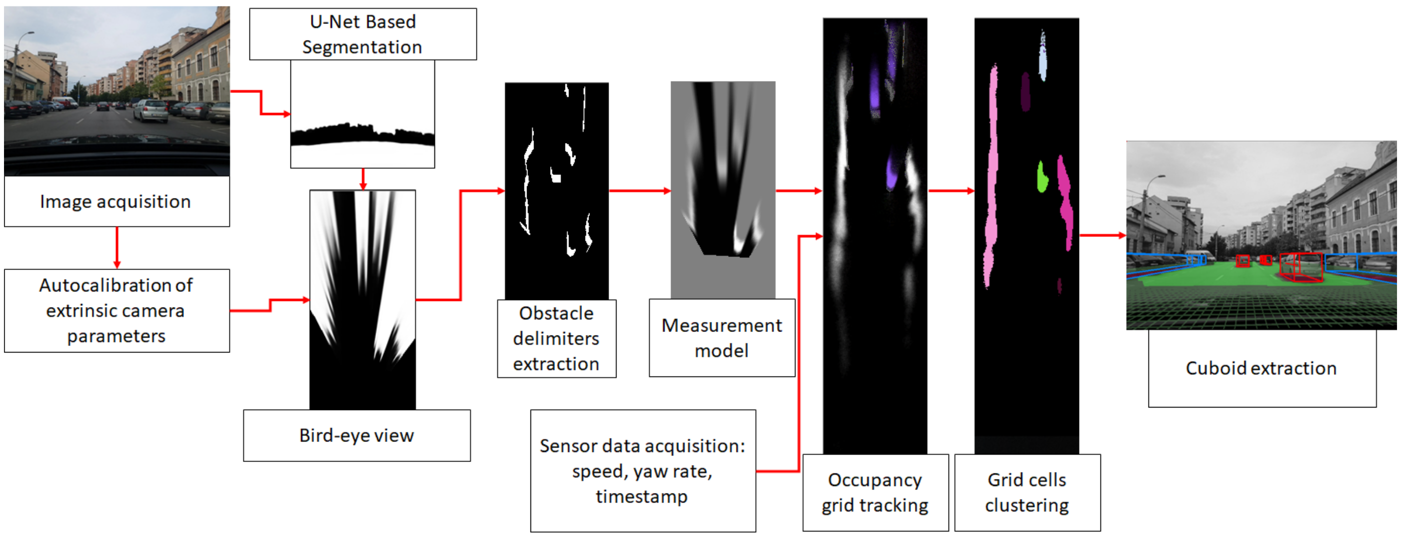

Fig. 1. The obstacle detection and tracking solution.

IV. Semantic Segmentation

The network used in our solution is based on the U-Net architecture, having 5 layers for encoding the data and 5 layers for decoding. The network also features a central layer between the encoding and decoding layers. A typical encoder layer consists of these operations: convolution using a 3x3 kernel, batch normalization followed by ReLU activation. These three operations are applied again, and then they are followed by a max pooling (with 2x2 stride). The middle layer has the same operations, minus the max pooling, whereas a typical decoder layer features: upsampling (deconvolution) with a 2x2 kernel, and concatenation with the homologous layer from the encoder. The decoder layer then features another deconvolution layer (with 3x3 kernel), batch normalization and ReLU activation, each applied three times. The final output of the network is given by the sigmoid activation function applied to a 1x1 convolution in the last layer. Both the input and the output images have the same size, 256x256 pixels.

A visual representation of the CNN is illustrated in figure 2.

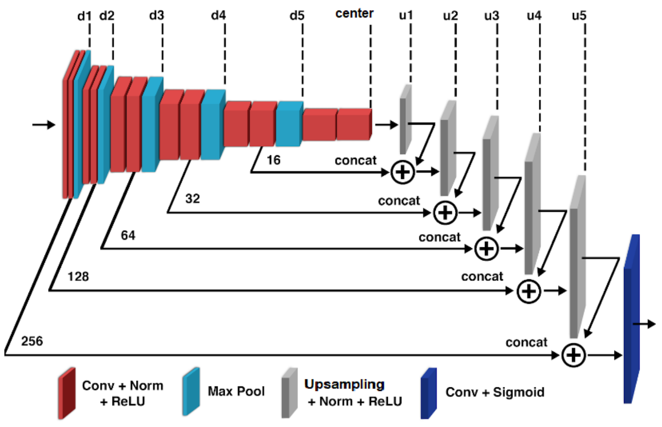

Fig. 2. The U-Net based architecture used for semantic segmentation of the scene.

The network was trained for a total of 50 epochs using the binary cross entropy loss function. We have also monitored the intersect over union (IoU) during the training phase and the results are illustrated in figure 3. Training is automatically stopped if there is no improvement (in this case it was stopped after 34 epochs).

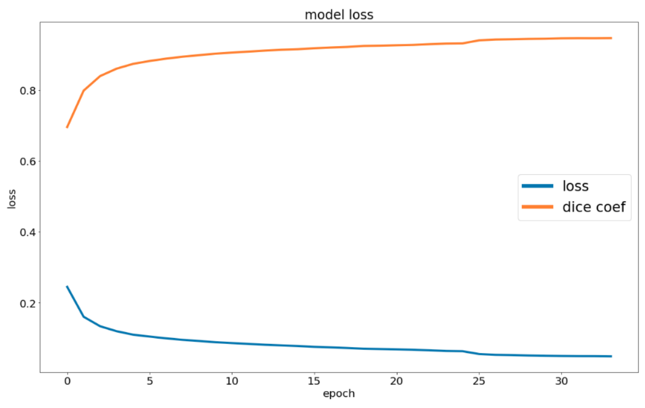

Fig. 3. Training the CNN using cross entropy ("loss"). The IoU score is also presented and it converges towards 1.

For training the network we have used the current available datasets for semantic segmentation: Cityscapes [9], KITTI [10], Mapillary Vistas [11] and Berkeley Deep Drive [12]. We have used a total of over 31000 images from the 4 datasets combined. The images with less than 2500 annotated road pixels were filtered out, resulting in 28000 images for training and 3500 images used for validation during training. All images and labels were scaled down to 256x256. The main objective is to determine the drivable road area, therefore we have only used the road class from the datasets, meaning that the image pairs used for training will have the following structure: the input image (color image of the road scene) and the label image (with the road annotated as 255, and the background/non-road annotated with 0).

V. Automatic Camera Calibration

The system is designed to work without using complex and elaborate camera calibration methodologies. However, accurate camera parameters are needed in the process of 3D coordinates reconstruction from the segmented image areas, and therefore a robust automatic calibration algorithm had to be designed. The algorithm is based on the road features. The process of lane marking extraction is helped by the fact that the road area is already segmented, and therefore the obstacle features will not affect calibration.

Based on the extracted lane markings, the current lane's width is computed for any possible image line. The results are accumulated in a 2D voting array. After enough frames are processed, a linear pattern is expected to emerge, as seen in figure 4. The intersection of the line with the 0 column indicates the horizon row, the geometric locus of the vanishing point. Subsequent computation of the vanishing point is restricted to this locus plus/minus an error band, and the vanishing point results are used for computing the pitch and yaw angle of the camera. The slope of the line will provide us the camera height, assuming that the lane width is (statistically) known.

More details about the automatic calibration process are provided in [13].

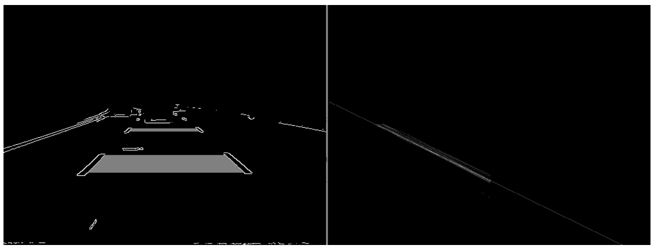

Fig. 4. Detecting candidate lane widths (left) and accumulating them versus the image row coordinate (right). The resulted linear relation is used to compute the camera height and the pitch angle.

VI. Occupancy Grid Based Tracking

A. The world model

The perceived scene is modeled as a 120 x 500 cells occupancy grid, each cell representing a 20 cm x 20 cm area of the road, seen from above. Therefore, the whole tracked scene is 24 x 100 m. The camera is assumed to be positioned at the row coordinate 250, and at column coordinate 60, therefore the camera is in the middle of the grid, facing forward. This means that half of the grid is not observable directly, but only predicted from the observable cells. This extended grid can be useful for dynamic map building, or for sensor fusion, if multiple cameras or other types of sensors can be added to observe the hidden areas.

Each cell in the grid can contain dynamic particles, which are entities having position and speed. The particles will move from one cell to another based on their own speeds, and based on the speed and yaw rate of the ego vehicle, which are used in the prediction phase of the tracking algorithm. The probability of a cell to be occupied is given by the number of particles that belong to that cell. A maximum number of N_C=100 particles per cell is allowed. The particles in a cell are created, multiplied and destroyed based on measurement information. More details about the particle-based occupancy grid can be found in [14].

At the end of each grid-based tracking cycle, the occupied cells (cell that have more than 75 particles) are grouped into individual objects based on neighboring and speed compatibility criteria. A cuboidal model is fitted to these clusters, producing a list of objects having position, length, width, orientation and speed, and a default height of 1.5 m.

B. The measurement model

The CNN based semantic segmentation is able to reliably find the road and the generic obstacle areas in the image space. These results must be mapped in the world's 3D space, so that they become useful as sensorial data for environment perception and tracking.

Using the camera parameters obtained from automatic calibration, we can generate a bird eye's view of the segmented image. Such an image is useful for assigning a longitudinal coordinate (Z) and a lateral coordinate (X) to any point in the image that is assumed to be on the road surface, therefore having its height coordinate (Y) equal to zero. As seen from figure 5(a), this mapping is only partially useful for retrieving the 3D information about the segmented obstacle areas, due to the fact that the obstacle points are mostly not on the road, and therefore only the points of contact between road and obstacle surfaces are to be taken into consideration. Therefore, the first step in processing the obstacle map is to generate polar scans and identify the obstacle-to-road contact areas.

```
Algorithm 1 – Polar distances computation
Input: the bird eye view segmented image $B$
Output: polar distances $d(a)$, for each $a = 0…180$
For each $a = 0…180$, set $d(a)$ to $\infty$
For each row $r$ and column $c$ of $B$
    If $B(r,c)>T_B$
        $a(r, c) = \text{atan2}(r, c) \cdot 180/\pi$        (1)
        $a_i = \lfloor a(r,c) + 0.5 \rfloor$        (2)
        $d_i = \sqrt{r^2 + (c - w_B/2)^2}$        (3)
        $d(a_i) = \min(d(a_i), d_i)$        (4)
    End If
End For
Return $d$
```

In algorithm 1 we assume that the row coordinate is proportional to the forward distance from the camera, and the column coordinate is proportional to the lateral distance, and the camera is at coordinates ($r = 0$, $c = w_B/2$), w_B being the width of the bird's-eye image B. The threshold T_B is used to classify between obstacle and road areas in the continuous image B.

Now we have the distance from the camera to the nearest obstacle structure, for every angle from 0 to 180 degrees. Only a subset of these angles will have a valid distance. For the angles that are outside of the camera's field of view, and for the angles that cast rays that do not meet obstacles in the *XZ* range defined by the bird's-eye view transformation, the assigned distance will remain the invalid infinity.

Another problem with the bird's-eye view is that most obstacles do not touch the road surface completely. For example, cars touch the surface with their wheels, but the road is visible between them. This gap gets amplified in the bird's-eye view image, leading to a false perception of the obstacle's distance, as seen in figure 5(b).

In order to obtain a more relevant detection of the obstacle distances, we will fill the gaps caused by imperfect contact with the road by generating convex hulls of the radial distances. First, we need to group the rays into clusters, based on their distance difference. The clusters will represent continuous structures such as individual objects, fences, etc. The polar clustering is achieved by Algorithm 2, and is based on the difference in distance between consecutive rays, which must be lower than a threshold T_K. The output of the algorithm is a vector K, which, for each ray angle a, will contain the cluster label. Each valid ray (with a distance that is not infinite) will have a non-zero label assigned to it.

The threshold T_K controls the clustering process, and is set by trial and error to the grid cell count equivalent to 3 meters. To increase the quality of the clustering result, we have employed a hysteresis-like re-clustering step. Small clusters, containing less than 10 rays, can be joined together if neighboring rays have a distance less than 3 times the initial threshold T_K. This will improve the estimation of the smaller objects, while preventing large structures, such as side fences, to be joined with vehicles and pedestrians.

Now, based on the clusters, the polar distances are adjusted to create convex objects.

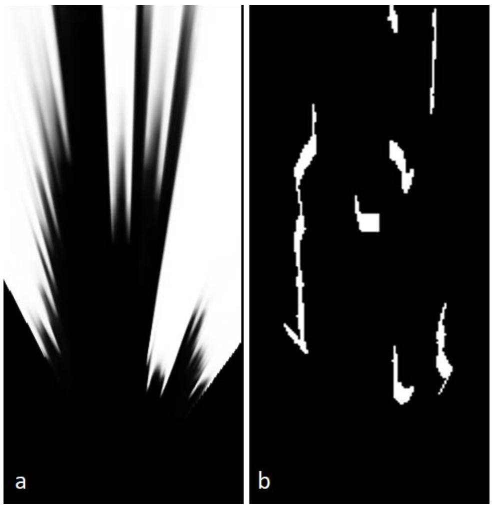

Fig. 5. From bird's-eye view of the segmented image to obstacle delimiter scans.

```
Algorithm 2 – Polar clustering
Input: polar distances $d(a)$, for each $a = 0…180$
Output: cluster labels $K(a)$, for each $a = 0…180$
For each $a = 0…180$, set $K(a)$ to 0
Set initial cluster number $N = 0$
For $a = 1…180$
    If $d(a)<D\_inf$
        If $|d(a)-d(a-1)|<T_K$ and $K(a-1)>0$
            $K(a) = K(a-1)$
        else
            $N = N+1$
            $K(a) = N$
        End if
    End if
End for
Return $K$
```

```
Algorithm 3 – Create convex objects
Input: polar distances $d(a)$, and clusters $K(a)$, $a = 0…180$
Output: refined polar distances $d(a)$
$Changed$ = true
While $Changed$
    $Changed$ = false
    For $a = 1…179$
    If $d(a)<d(a-1)$ and $d(a)<d(a+1)$ and $K(a)=K(a-1)=K(a+1)$
        $d(a) = (d(a-1) + d(a+1))/2$
        $Changed = true$
    End if
    End for
End while
Return $d$
```

After this step, the rays can be used to create a “scan image”, as seen in figure 5(b).

The measurement data will be further processed to generate the measurement model, a map that will express the conditional probability of a cell to be occupied, given the known measurement data. This model must incorporate the measurement errors, uncertainties and limitations. As the measurement is based on detecting the contact point between the obstacle and the road, transposed in the bird’s-eye view space, the following errors and limitations are taken into consideration:

a. The longitudinal errors along the observation rays. These errors are caused by the limits of the Inverse Perspective Mapping. We use the equation from [15] to compute this error:

$$\sigma_z = h\left(1 + \left(\frac{z}{h}\right)^2\right)\sigma_\alpha + \sigma_0 \qquad (5)$$

In the above equation, h is the camera height above the road plane, z is the forward distance, and σ_α is an angular expected error, which can be caused by either the limited resolution of the image, which limits the accuracy of the angle measurement of the obstacle contact point with the road surface from the image data, or by the pitching motions of the ego vehicle. An experimental value of 0.1 degrees was chosen. By σ_0 we denote an error that accounts for the non-modeled sources, and this value is set to 0.1 meters.

b. The limitations of observation. As the measurement is expressed by distances along viewing rays, we can only know that before the ray hits the obstacle, it passes through free space, and that we have an obstacle structure at the distance indicated by the ray. We don’t know what is beyond the obstacle. Assuming that along a ray cast at angle a (from 0 to 180 degrees) we have the obstacle at distance d, and that an obstacle has a minimum depth w, we can define the occupancy probability along a ray for a given distance d as:

$$p_{ideal}(a, z, d) = \begin{cases} 0, if\ z < d \\ 1, if\ z \geq d\ and\ z < d + w \\ 0.5, if\ z \geq d + w \end{cases} \qquad (6)$$

The above equation states that we are certain that the cells are free before we hit the obstacle, we are certain that they are occupied for at least a small depth w, and after that the probability of being free or occupied are equal, 0.5.

The probabilities expressed by (6) are ideal, as they do not take into consideration the sources of errors. In order to account for possible segmentation errors, the values 0 and 1 are replaced by p_0 and 1-p_0, respectively, where p_0 is a small value, experimentally set to 0.05. In order to account for the distance errors, we define a Gaussian convolution kernel G(d), based on the standard deviation computed from equation (5). This kernel is used to convolve the p_{ideal} values:

$$p_{real}(a, z, d) = p_{ideal}(a, z, d) * G(d) \qquad (7)$$

Now we have for each angle a and each possible distance z the occupancy probability value $p_{\text{real}}(a,z)$. We will omit the d parameter, as it is no longer useful for us.

The next step is to map these polar coordinate probabilities into the Cartesian grid space. Each row of the grid will get an assigned probability, using the following algorithm:

Algorithm 4 – Create measurement probability grid

```
Input: polar probabilities $p_{\text{real}}(a,z)$
Output: grid probabilities $p_{\text{measured}}(r,c)$
For each grid row $r$
    For each grid column $c$
        If $(r,c)$ is in camera's field of view
            $a_{\text{f}} = \text{atan2}(r - r_{\text{cam}}, c - c_{\text{cam}})$    (8)
            $z_{\text{f}} = \sqrt{(r - r_{cam})^2 + (c - c_{cam})^2}$    (9)
            $p_{\text{measured}}(r,c) = \text{LinearInterpolation}(p_{\text{real}}, a_{\text{f}}, z_{\text{f}})$
        Else
            $p_{\text{measured}}(r,c) = 0.5$
        End if
    End for
End for
Return $p_{\text{measured}}$
```

In Algorithm 4, the grid is assumed to be larger than the observable scene, and the position of the camera, which is the point from where the rays are cast, is located in the grid at coordinates (r_{cam}, c_{cam}). In our implementation, the camera is located in the middle of the tracked grid. The row and column coordinates r and c are used to compute a distance z_{f} and an angle a_{f}, as floating point values. As the polar probability matrix p_{real} is computed for integer values, the function LinearInterpolation is used to compute a weighted mean of the neighboring values of p_{real}, based on their proximity of their integer coordinates to z_{f} and a_{f}. This way, we’ll ensure a complete and smooth coverage of the grid cells with measurement probability values.

The probability computation process is depicted in figure 6. Two rays are shown as example: one will meet an obstacle scan far away from the camera, and one will meet it at a closer distance. The Gaussian kernels used for the two rays are adjusted accordingly, the one for the far away obstacle being much more spread out, depicting the uncertainty of the measurement growing with the distance. The final measurement probability grid depicted in the right side of the image is the result of linear interpolation of the ray probability values for each grid column and row. The grid cells that are not inside the camera’s field of view get the default 0.5 probability, and the same probability is assigned to the grid cells behind the camera, which are not shown in the figure.

C. Updating the occupancy grid

Before measurement, the probability of a grid cell at coordinates (r, c) to be occupied is given by the number of particles in that cell, particles that have been moved using the motion equations of the ego vehicle (the forward movement expressed by the speed, and the angular movement expressed by the yaw rate), and the motion equations of the particles themselves (uniform motion based on constant speed, combined with random alterations of the position and speed), as described in [14]. We’ll denote this predicted probability as $p_{\text{pred}}(r,c)$, the ratio between the number of predicted particles in the cell and the maximum capacity of the cell.

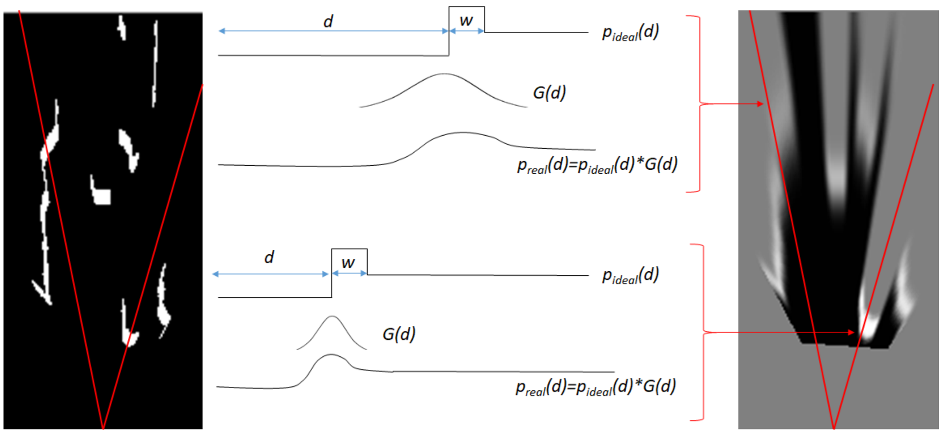
d
w
$p_{ideal}(d)$
G(d)
$p_{real}(d)=p_{ideal}(d)*G(d)$
d
w
$p_{ideal}(d)$
G(d)
$p_{real}(d)=p_{ideal}(d)*G(d)$

Fig. 6. The generation of the measurement model from the scans.

Knowing the predicted probability (the prior probability), and the measurement probability $p_{\mathrm{measured}}(r,c)$, the updated occupancy probability is given by:

$$p(r,c) = \frac{p_{pred}(r,c)p_{measured}(r,c)}{p_{pred}(r,c)p_{measured}(r,c)+(1-p_{pred}(r,c))(1-p_{measured}(r,c))} \qquad (10)$$

As the true state of the grid is given by its component particles, the probability computed by (10) is used to adjust the particle population for each cell. The target number of particles is $p(r,c)N_C$, where N_C is the maximum capacity of a cell (100 in our implementation). If the current number of particles in the cell is lower, they are randomly multiplied, otherwise they are randomly eliminated [14].

For a smoother estimation, we have used an additional step before starting the particle multiplication/deletion process, consisting of a Gaussian smoothing of the $p(r,c)$ array.

D. *Identifying individual objects*

The updated occupancy grid is segmented into individual objects by proximity based labeling. Clusters of occupied neighboring cells are extracted, and cuboids are fitted to them. The speeds of each cell, resulted from the speeds of individual particles within the cell, are used to estimate the speed and orientation of the resulted cuboid. If the speed of the cuboid is too low, or the standard deviation computed from the individual cell speeds is too high, the object is reported as static and no orientation is computed for it.

E. *Handling measurement errors*

Most measurement errors come from fast pitching of the ego vehicle, and occasional false segmentation results. These errors may create false occupied cells in the measurement grid, or lead to a false perception of the occupied cells' distance. We can see the effect this has on the tracking process in figure 7

Some pitching errors are handled by using real time correction of the pitch angle, based on computation of the vanishing point. However, sometimes the vanishing point is incorrectly computed due to lack of reliable road markings in the scene. In these cases, the simplest solution that we found is to ignore the measurement altogether, and rely on the prediction instead. A fitness score is computed between the predicted occupancy grid and the measurement grid, and if this score falls below a threshold, the measurement grid is discarded. The effects of this approach can be seen in figure 8.

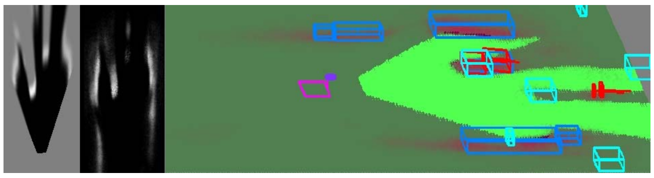

Fig. 7. Effect of uncorrected sudden pitching: false grid cells are created (near vehicle), or the position is wrong (far vehicle). Left: measurement probability, middle: tracked occupancy probability, right: objects reconstructed from the grid. The light blue obstacles are Ground Truth obstacles.

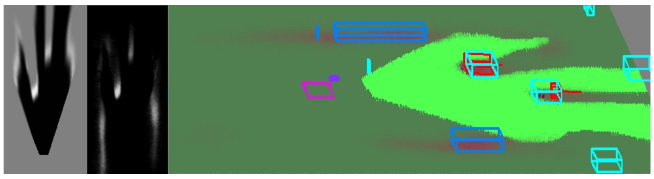

Fig. 8. Effects of ignoring a false measurement, and using prediction instead: the tracked obstacles stay in place. The light blue obstacles are Ground Truth obstacles.

VII. TESTS AND RESULTS

A. *Testing the segmentation performance*

The network was evaluated on the CityScapes validation set and the IoU score is 0.91, whereas the state of the art DeepLab [16] achieves 0.98 (Table 1).

TABLE I. COMPARISON WITH SIMILAR METHODS.

CNN Model	**IoU score for road class**
DeepLab [16]	0.986
E-Net [17]	0.974
Proposed CNN, multi-class	0.922
Proposed CNN	0.911

The difference in score is also given by the size of the images used for training and prediction, as the best results from the state of the art use images at least twice as large as in our proposed model. Also, these networks were trained to predict multiple classes (usually 19 classes that are present in the Cityscapes set), whereas our model was trained only on 1 class (the road area). We also trained our model to predict 4 classes: road, sidewalk, vehicles and sky and the results are improved (0.92 IoU score vs 0.91 on the single class). Also, our model does not use any post-processing or refinement of the predicted image, we simply threshold the image using a fixed value.

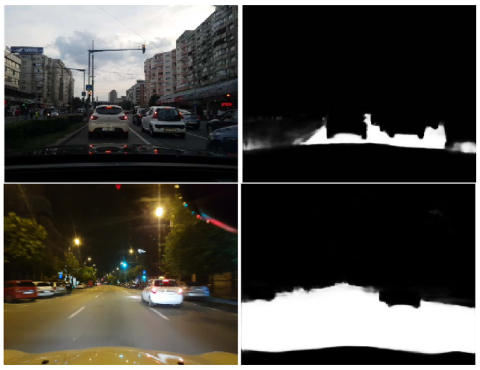
Fig. 9. Example of predictions in complex city scenarios during day time and night time.

The predictions are good enough to be used in a perception system, for the task of tracking the objects in the scene. By identifying the drivable road area, we make the assumption that everything that is not part of the road is an obstacle, and therefore can be tracked. We chose to favor prediction speed rather than state of the art accuracy, therefore our model is capable of predicting the drivable road area in under 15ms on average (on 256x256 input images, total processing time, including grid tracking and object detection is below 100 ms), whereas models trained on 512x512 images usually predict in 33-34ms on average.

Figure 9 shows examples of predictions using the trained CNN with images from our own acquired test dataset (these images have never been seen before by the network during training).

B. *Testing the object detection and measurement performance*

For testing the performance of object detection and the accuracy of 3D measurement, we needed continuous sequences of images (as our method is a tracker, not a single shot detector) annotated with timestamps and vehicle sensor data (speed and yaw rate). Also, the objects in the sequence need to be detected and measured. For these reasons, we have used sequences from a previous project, consisting of binocular image pairs, accurately calibrated, so that we can perform stereovision-based object detection. The objects detected and tracked from the stereovision system are considered as ground truth.

The only downside of the used sequences is that they are monochrome, and the segmentation network was trained for color images. However, we have found that the network still works, even in the absence of color.

Some results can be seen in figures 10 and 11. The red boxes are dynamic objects extracted by the proposed solution, the dark blue objects are static objects, and the light blue objects are the ground truth objects extracted by stereovision. We can see a good match both in the perspective image and in the 3D world. The nature of the objects does not influence the detection: vehicles, scenery and pedestrians are all detected correctly.

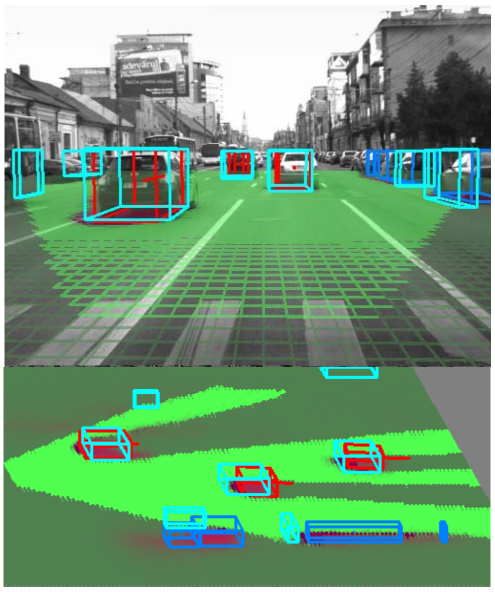
Fig. 10. Monocular versus stereo: detecting moving and parked vehicles.

We have assessed the detection rate performance and the distance measurement performance for multiple intervals, as shown in tables II and III. As expected, the accuracy of the monocular vision system is significantly lower at higher distances, above 30 meters. Also, due to the fact that beyond 40 meters the expected error of the monocular system is very high, the occupancy probability of the grid remained low, and the objects are not always detected, and therefore we can see a significant drop in the detection rate in that interval.

The detection rate is lower than 100% in all intervals, and this is mainly due to the limitations of the grid (the grid only covers a road width of 24 meters), and due to the scanning nature of the algorithm, which only allows one object to be detected along a viewing ray. The stereovision system is able to detect an object even if it is partially obstructed by another, closer one. The detection rate in the distance interval 0-10 m is affected by the ego vehicle's hood obstructing the contact point of the obstacles with the road, even if they are visible enough to be detected by stereovision.

In figure 12 we can see a typical behavior of the monocular system when tracking a moving object. The orange line is the ground truth distance, and the blue line is the distance detected from monocular grid tracking. We can see that the monocular system closely follows the tracked object, but the errors increase with the distance. When the object approaches the 40m mark, it is lost.

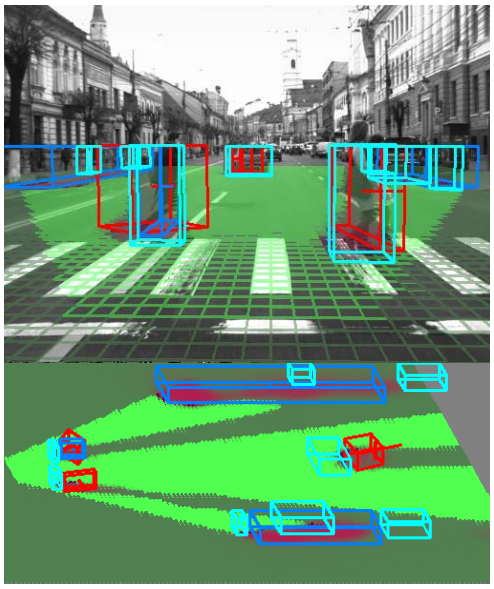

Fig. 11. Monocular versus stereo: detecting pedestrians

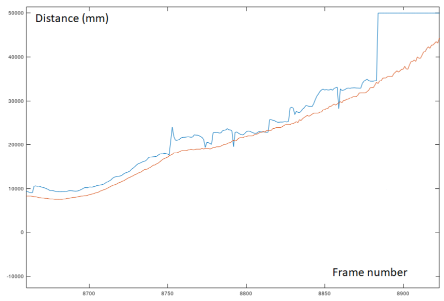

Fig. 12. Tracking a moving object.

TABLE II. DISTANCE ERROR FOR DIFFERENT INTERVALS.

Distance range (m)	**Mean Average Error (m)**
0 – 10	0.78
10 – 20	1.37
20 – 30	2.62
30 – 40	7.21
40 – 50	17.44

TABLE III. DETECTION RATE FOR DIFFERENT INTERVALS.

Distance range (m)	**Detection rate (%)**
0 – 10	88.66
10 – 20	96.03
20 – 30	92.32
30 – 40	80.61
40 – 50	57.43

VIII. CONCLUSION

We have described a traffic scene perception system that relies on the power of the CNNs to achieve robust identification of the obstacle areas in the image space, and on the power of probabilistic modeling and tracking of the scene in the 3D space. By combining these techniques, we have achieved a robust and real time vision system that can be used in diving assistance applications. Due to the probabilistic and 3D nature of the occupancy grid model, the system can be easily extended to use another camera, or another sensor such as a radar or a laser scanner.

ACKNOWLEDGMENT

This work was supported by a grant of Ministry of Research and Innovation, CNCS - UEFISCDI, project number PN-III-P1-1.1-TE-2016-0440, within PNCDI III.

REFERENCES

[1] S. Sivaraman, M. Manubhai Trivedi, ”Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis”, IEEE Trans. Intelligent Transportation Systems vol. 14, no. 4, pp. 1773-1795, 2013.

[2] J. Long, E. Shelhamer, T. Darrell, “Fully convolutional networks for semantic segmentation”, Computer Vision and Pattern Recognition, pp. 3431-3440, 2015.

[3] O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, Vol. 9351, pp. 234-241, 2015.

[4] R. Garg, V. Kumar BG, I. Reid, “Unsupervised CNN for single view depth estimation: Geometry to the rescue”, European Conference on Computer Vision, pp. 740-756, 2016.

[5] C. Godard, O. M. Aodha, G. J. Brostow, “Unsupervised Monocular Depth Estimation with Left-Right Consistency”, Computer Vision and Pattern Recognition, pp. 6602- 6611, 2017.

[6] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, R. Urtasun, “Monocular 3D Object Detection for Autonomous Driving”, Computer Vision and Pattern Recognition, pp. 2147-2156, 2016.

[7] A. Mousavian, D. Anguelov, J. Flynn, J. Kosecka, “3D Bounding Box Estimation Using Deep Learning and Geometry”, Computer Vision and Pattern Recognition, pp. 5632-5640, 2017.

[8] T. Roddick, A. Kendall, R. Cipolla, "Orthographic Feature Transform for Monocular 3D Object Detection", arXiv: 1811.08188, 2018.

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding”, Computer Vision and Pattern Recognition, pp. 3213-3223, 2016.

[10] A. Geiger, P. Lenz, R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”, Computer Vision and Pattern Recognition, pp. 3354-3361, 2012.

[11] G. Neuhold, T. Ollmann, S. R. Bulò, P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes", IEEE International Conference on Computer Vision, pp. 5000-5009, 2017.

[12] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, T. Darrell, “BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling”, arXiv: 1805.04687, 2018.

[13] R. Danescu, R. Itu, ”Camera Calibration for CNN based Generic Obstacle Detection”, 2019 Portuguese Conference on Artificial Intelligence (EPIA), in print 2019.

[14] R. Danescu, F. Oniga, S. Nedevschi, “Modeling and Tracking the Driving Environment with a Particle-Based Occupancy Grid”, IEEE Transactions on Intelligent Transportation Systems, volume 12, no. 4, pp. 1331-1342, 2011.

[15] R. Danescu, R. Itu, A. Petrovai, "Generic Dynamic Environment Perception Using Smart Mobile Devices", Sensors, vol. 16, no. 10, art. no. 1721, 2016.

[16] L.C. Chen Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, “Encoder- Decoder with Atrous Separable Convolution for Semantic Image Segmentation”, arXiv: 1802.02611, 2018.

[17] A. Paszke, A. Chaurasia, S. Kim, E. Culurciello, "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation", arXiv: 1606.02147, 2016.