UP-Drive
Automated Urban Parking and Driving

Research and Innovation Action - Horizon 2020
Grant Agreement Number 688652
Start date of the project: 01.01.2016, Duration: 48 months

Deliverable D 4.3

Initial version of higher-level perception functions

Type: R

Status: Revision 1.0

Lead contractor for this deliverable: UTC
Due date of deliverable: 30.06.2017
Coordinator: VW

Project co-funded by the European Commission within HORIZON 2020
Dissemination Level: PU

PU	Public
PP	Restricted to other programme participants (including the Commission Services)
SEAM	Restricted to partners of the SEAM Cluster (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Executive Summary

The deliverable provides an initial design and implementation of the higher level perception functions referring to road surface and obstacle perception, parking spot detection, road users classification, tracking and signaling perception. The higher level perception functions take as input low-level data coming from all possible sensors mounted on the vehicle (360-degree and 4-layer lidars, cameras, etc.). Some high-level perception functions are designed to use data coming only from a single category of sensors (e.g. either lidars or cameras). An initial approach of enhanced high level perception functions based on the supersensor (spatio-temporal and appearance based low level data representation - STAR) is also presented.

Contributing Authors/Partners

	Company/Organisation	Name
Document Manager	UTC	Sergiu Nedevschi
Partner 1	UTC	Florin Oniga
Partner 1	UTC	Ion Giosan
Partner 1	UTC	Robert Varga
Partner 1	UTC	Arthur Costea
Partner 1	UTC	Mircea Paul Muresan
Partner 1	UTC	Andra Petrovai
Partner 1	UTC	Flaviu Vancea
Partner 2	CVUT	Martin Matoušek
Partner 2	CVUT	Radim Šára
Partner 3	VW	Jens Spehr
Partner 3	VW	Markus Koechy
Partner 3	VW	Umair Rasheed

Contents

1 Introduction

This deliverable provides a report on the initial design and implementation of the higher-level perception functions. The main goal is to build high-level environment perception functions that uses the low-level data coming from all available sensors.

In order to make the perception system robust, there should exist functions that use only data coming from a specific type of sensors (e.g. either cameras or lidars) such that the environment may be perceived even in the case when some sensors cannot provide reliable data (e.g. night, adverse weather conditions, failures). In the case when data is available from all sensors, the spatio-temporal and appearance based low level representation (STAR) is built. An initial design and implementation of high-level perception functions based on STAR representation is proposed.

Several modules are identified and an initial design and implementation is proposed for each of them:

- Road surface detection and 3D object segmentation based on 3D voxels using only the lidar point cloud data
- Parking spot detection from lidar point cloud data
- Semantic segmentation of images and instance based object segmentation from area-view image data
- Signaling perception of road users from image data
- Road users perception by taking into account the problem of occlusions and associating semantic labels to the 3D objects
- Road users perception based on STAR representation
- Road users tracking and object position correction

2 Point cloud based perception module

2.1 Ground and object segmentation from point cloud data

The proposed approach for ground and obstacle segmentation consists of two main steps. The first step is the detection of the road surface, which is done initially on each individual 360-degree lidar data, and then the results are fused in order to get a global road surface. The second step is the detection of 3D obstacles by using a proposed voxel representation that allows the fusion of all the lidar outputs. Given the five 360-degree lidars (and the additional data from 4-layer lidars), the lidar measurements will be first fused into the same representation (the voxel space), and then the 3D obstacles will be detected.

2.1.1 Road surface detection

The final result of the road detection will be a model free road surface, represented as elevation values in a rectangular grid that covers a region of interest around the ego vehicle.

The approach for road surface detection starts by processing each individual lidar data in the cylindrical layer / azimuth representation. This representation is built straight forward from the lidar output (Fig. 1), showing the panoramic structure of the 3D data organized by layers and azimuth angles, each location containing the 3D features of the lidar measurement.

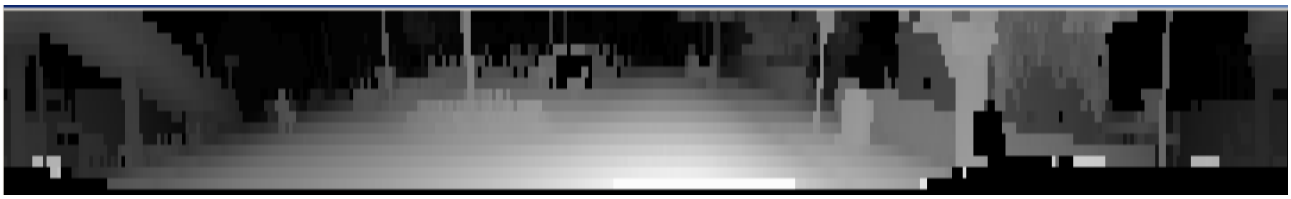

Figure 1: Cylindrical / panoramic representation of the front left lidar output. The complementary logarithm value of the depth is shown in this image, for better visualization

A preliminary classification of the panoramic representation, into road / obstacle (Fig. 2), is performed based on the range difference to an ideal flat road model, for each 360-degree lidar. This classification has the goal of providing a rough separation in order to reduce the number of road outliers for the next processing step.

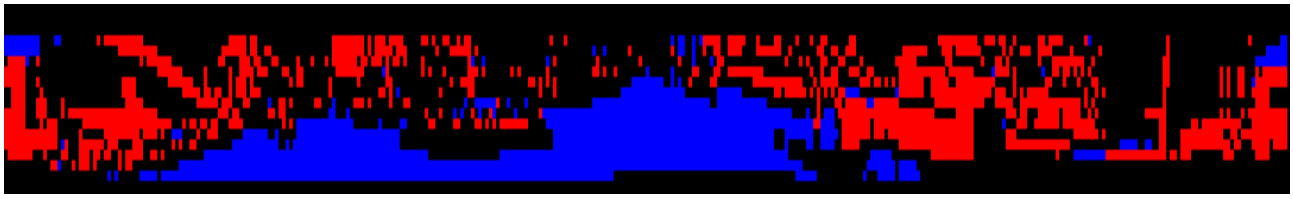

Figure 2: Preliminary classification results (road / obstacles) on the cylindrical / panoramic representation of the front left lidar output

Next, on each lidar data, the predominant planar road patch is detected from the road classified area, using a random sample consensus approach (similar to the connected components RANSAC presented in the literature, but with a fast implementation due to the grid-like representation of the panoramic space). The random sample having the largest number of inliers, which are in the same patch in the panoramic grid, is selected. The road patch is

the set of connected inliers of the best sample. This patch is transformed into a Cartesian representation using Bresenham interpolation.

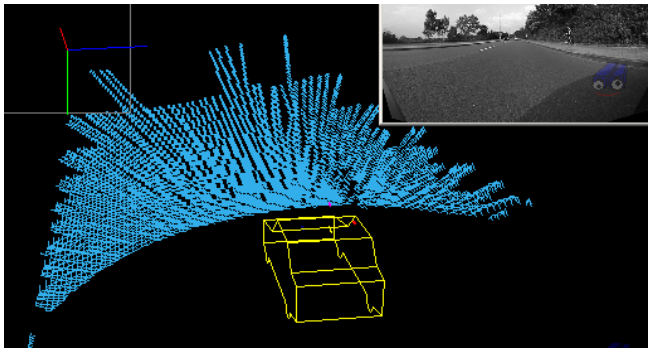

Figure 3: The dominant road patch in a Cartesian representation

Finally, the five Cartesian road patches (Fig. 4) from each lidar are fused in the same grid (Cartesian, covering a rectangular area of 60x15 meters around the ego vehicle, Fig. 5). When dealing with multiple road elevation (form more than one lidar) in a grid location, the highest estimation is stored. This is to minimize the probability that other road measurements at the same location might be classified later as obstacles (in the voxel space).

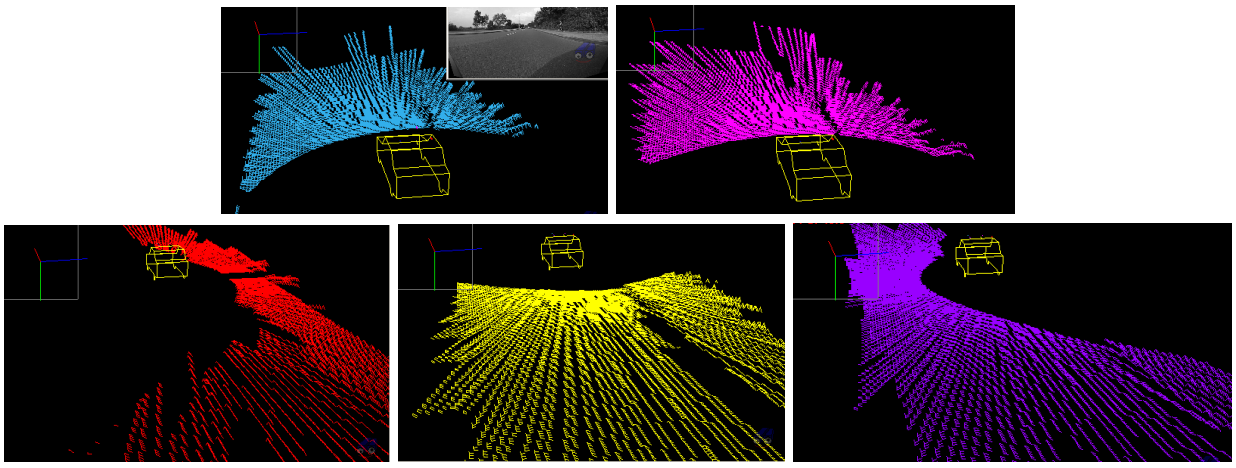

Figure 4: The dominant road patches for each of the lidars: on the top row the front-left and the front-right 360-degree lidar, and on the bottom, the rear-right, rear-center, and rear-left 360-degree lidars

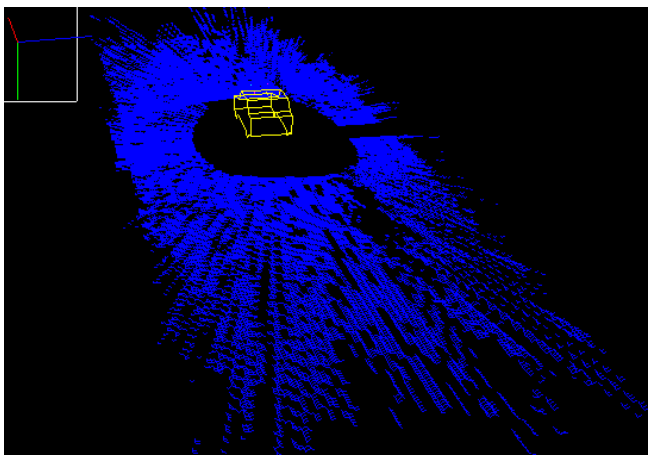

Figure 5: The result of fusing in the same grid the five road patches detected previously

The road surface obtained so far has valid road elevation values only in some of the locations. In order to use the road surface for obstacle / road separation, the road elevation value should be available for every 3D (grid) location. Thus, an estimate of the road elevation in each grid location is computed as an interpolation based on the nearest neighbors that contain road elevation (Fig. 6).

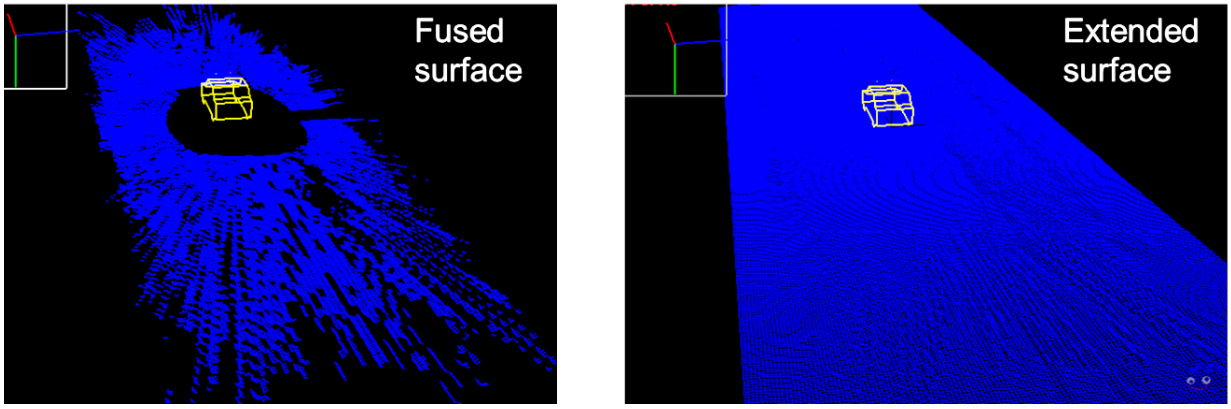

Figure 6: The fused surface, more or less dense, is transformed into a fully dense surface (extended), by propagating the nearest elevations to empty cells

The dense road surface provides the road elevation for every scene location covered by the grid. However, there are large areas where no initial road measurement was available, so when using the nearest available road elevation, the local model is a flat road model. In order to avoid classifying as obstacles certain road areas that are not flat, but are in the blind spot of the lidars, an uncertainty elevation value can be computed for each grid location that is too far away from the real road measurements. For each such location, a conservative road elevation is estimated based on the nearest elevation value, by assuming a maximum local pitch of the road (light blue areas in Fig. 7).

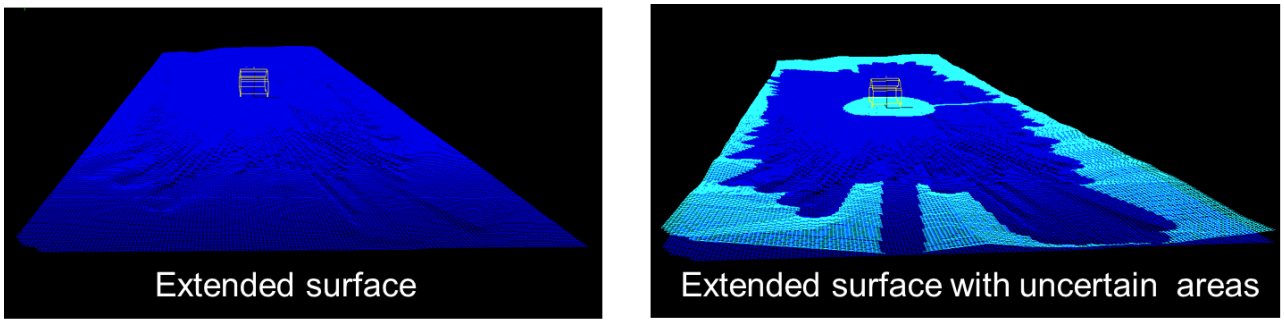

Figure 7: A conservative road elevation value is computed (light blue areas on the right image) for each cell that is far enough from the neighbors that initially provided the road elevation of the cell. This conservative road value will be used for obstacle / road separation in those locations

2.1.2 Object segmentation based on 3D voxels

The lidar measurements will be first fused into the same representation (the voxel space). The size of the voxel space is similar to the road surface grid on the top view, 60m x 15m (longitudinal x lateral directions), and spans 4 meters on the elevation (2 meters up/down relative to the road level = 0 elevation).

The voxel representation is built as follows:

- each lidar measurement above the road surface is included (360-degree / 4-layer lidar) in its corresponding voxel (the voxel is considered occupied)
- voxel space is densified based on the cylindrical layer/yaw lidar space. If two lidar measurements are adjacent in the panoramic representation:
  - Neighboring layers and the same azimuth, and the 3D distance between the measurements is below a threshold, then intermediate voxels are marked as occupied
  - The same layer and consecutive azimuth values, and the 3D distance between the measurements is below a threshold, then intermediate voxels are marked as occupied

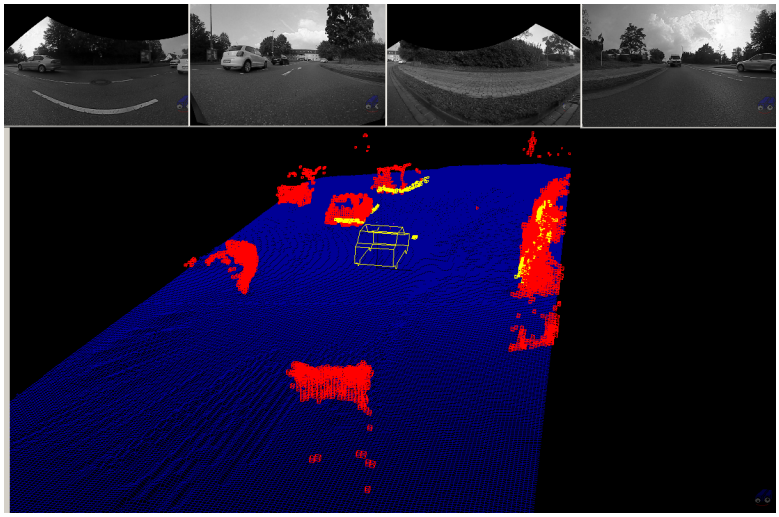

Figure 8: The voxels space (densified) composed of the 360-degree measurements (red voxels) and 4-layer lidars measurements (yellow voxels). The road surface elevation is displayed for each vertical column of the voxel space (the blue mesh)

Due to the densification of the voxel space, most of the lidar measurements, adjacent in the panoramic representation, belonging to the same 3D obstacle are connected (there are several intermediate voxels, even if there are not intermediate measurements, that form a continuous path between them).

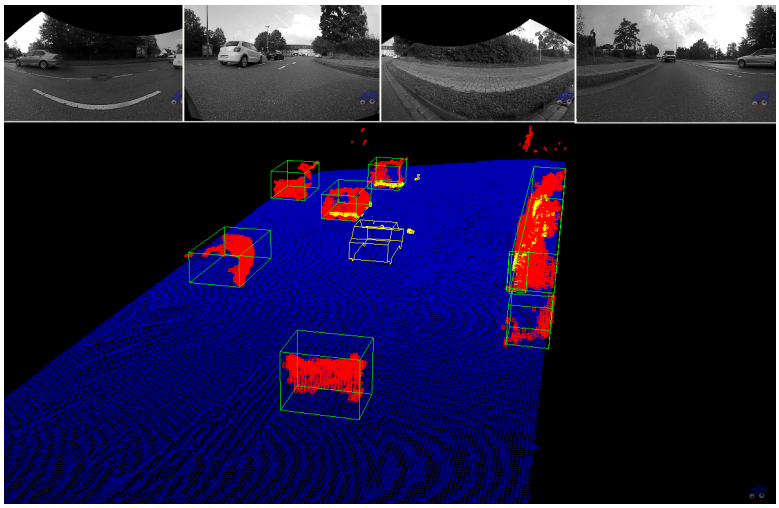

Figure 9: The voxels space and the corresponding obstacles depicted as cuboids around each connected component

3D obstacles are extracted as the connected components of occupied and intermediate voxels. The last step is the clustering of the voxel space into obstacle candidates. This is performed as a breadth-first search to extract the connected 3D blobs. For each obstacle candidate various features are computed, such as size, number of voxels, number of 3D points measured by the lidars, etc. Some of the candidates that do not have these features in an expected range are removed. Finally, in order to reduce fragmentation, cuboids are merged based on criteria as their relative position (from the lidars' point of view), size, and overlap. Results are shown in Fig. 9.

2.2 Parking spot classification

The purpose of this sub-module is to classify parking spot locations as being free (for parking) or occupied (in the physical, not legal sense). It is assumed, that discrete parking spot locations, orientations, and sizes are stored in a map and are available during driving. Prior to the free/occupied classification, the parking spot locations are continuously being pre-sorted based on their distance to ego-vehicle, such that only a reasonable subset of the spots in the vicinity of the vehicle is classified. The classification then uses the 3D point clouds provided by LiDARs. An example of an input data frame is illustrated in Fig. 10.

The proposed approach consists of two steps: 1) continuous aggregation of 3D point data into moving occupancy grid, and 2) query-based classification of parking spots.

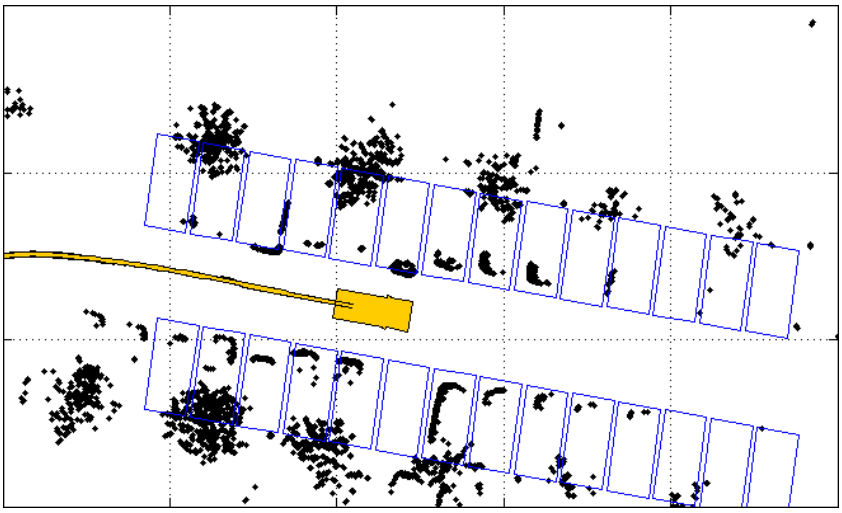

Figure 10: Top-view of a point cloud from a single frame from all LiDARs and several parking spots extracted from a digital map. The ego-vehicle is shown with its passed trajectory.

2.2.1 Occupancy grid aggregation

We use a probabilistic volumetric approach for representing the local map in the vicinity of the ego-vehicle. LiDAR sensor readings are integrated into 3D occupancy grid, which is well established tool for this problem domain [5].

The occupancy grid represents occupancy probability in the vicinity of the ego-vehicle up to 20 m lateral distance in every direction and 3 m of height. The grid is composed from cubic cells of fixed size 0.1 m and its orientation is fixed w. r. t. the world coordinate system. As the vehicle is moving, the grid position in the world coordinate system is kept static as long as the vehicle is located in the central part of the area covered by the grid. At least 5 m around

the vehicle and 15 m in the direction of egomotion should be covered by the map. When this requirement is violated by vehicle movement towards a boundary, the grid position in the world system is updated. This update takes into account the direction of egomotion and the size of the map, such that the new position could be kept fixed as long as possible. The new position is always chosen with a resolution given by the cell size. Since the orientation of the grid is constant, the update does not involve any re-sampling of underlying probability function and possible loss of accuracy.

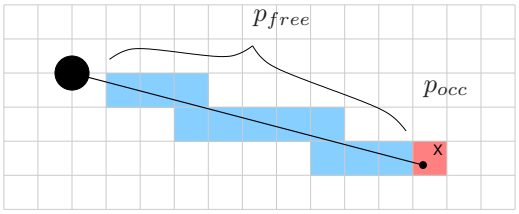

Figure 11: Sensor model for LiDAR-provided 3D points.

The cell values are updated using binary Bayes filter with static state [9]. We use a simple inverse sensor model $p(m_{x,y,z}|X_i)$ for each single point measurement X_i provided by a LiDAR, as illustrated in Fig. 11, in which $m_{x,y,z}$ is a binary random variable representing whether the particular cell (x, y, z) is empty $(m = 0)$ or occupied $(m = 1)$. The cell to which the point belongs to is updated using the probability p_{occ}, the cells along the ray from the sensor to the point are updated using the probability p_{free}. The constants p_{occ} and p_{free} are parameters of the sensor model for a particular sensor, chosen with respect to the cell size, sensor accuracy and sampling density in a typical distance for the parking spot detection application.

We use a popular log-odds representation of probability of a binary variable,

$$L(m_{x,y,z}) = \log(\frac{p(m_{x,y,z})}{p(\neg m_{x,y,z})}) = \log(\frac{p(m_{x,y,z})}{1 - p(m_{x,y,z})}) \,. \tag{1}$$

The occupancy grid represents the evolution of occupancy probability as new measurements are coming. Under the usual simplifying assumption of statistical independence of grid cells and of independence of sensor readings given a scene, the grid is updated cell-by-cell using static state binary Bayes filter as

$$L(m_{x,y,z}|X_1, \ldots, X_i) = L(m_{x,y,z}|X_i) + L(m_{x,y,z}|X_1, \ldots, X_{i-1}) - L(m_{p_{x,y,z}}) \,, \tag{2}$$

where X_i is the new sensor reading (a 3D point), $X_1, \ldots, X_{i-1}$ are sensor datapoints already incorporated into the grid, $L(m_{x,y,z}|X_i)$ is log-odds of the inverse sensor model and $L(m_{p_{x,y,z}})$ is a prior (currently zero, which corresponds to probability of 0.5). At the beginning and during the coordinate system update empty cells are initialised to the zero prior.

An example of accumulation of LiDAR data into the occupancy grid during a drive is shown in Fig. 12.

2.2.2 Query based classification

The occupancy grid is being continuously kept up-to-date by a process described in the previous section. When a parking space is needed, an upper-level decision-making process starts extraction of reasonable set of parking spots from a map. The spots from the current set are then repeatedly queried for classification – a label from a set {*free, occupied, unknown*}, together with its confidence, is being assigned to every parking spot. The classification estimates the probability of every label and the decision is chosen as a maximum.

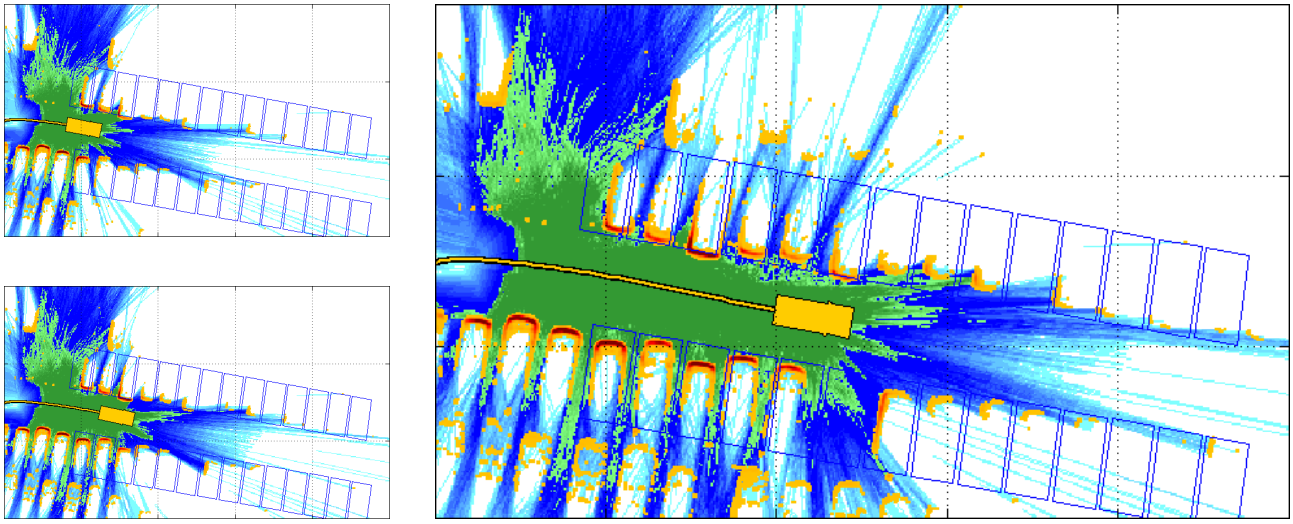

Figure 12: Accumulation of LiDAR data into the occupancy grid during drive; top-view. Cells aggregated across the map height are shown: orange to red intensity shows probability of occupied cells, blue to green intensity shows probability of free cells, white is for unknown state. The ego-vehicle is shown with its passed trajectory, the grid of dotted lines has 10 m pitch. Note that map location update is not demonstrated in this figure.

Since the parking spot can be occupied by any object (or independent objects), not just a car, we do not assume any spatial structure here. The classification is based only on a number of cells that are observed to be occupied (with some probability) and a number of cells that are observed to be free (with some probability). Also the amount of cells that have not yet been observed is taken into account. The classification works in two steps, first, occupancy probability is assigned to every planar (x, y) cell coordinate inside a particular parking spot by aggregating across the z-coordinate. Second, these probabilities are aggregated across the whole parking spot to estimate probability of the *free*, *occupied*, and *unknown* labels.

For every planar cell coordinate inside the parking spot in question, the log-odds are summed-up over the z-coordinate,

$$L^0(m_{x,y}) = \sum_z L(m_{x,y,z})\,. \tag{3}$$

Additionally, another sum in the vertical column is computed only from cells that have positive log-odds (i.e. are likely to be occupied; probability > 0.5).

$$L^+(m_{x,y}) = \sum_{z|L(m_{x,y,z})>0} L(m_{x,y,z})\,. \tag{4}$$

The L^+ is motivated by the requirement, that occupied cells with reasonable probability will veto any number of free cells. The occupancy probabilities p^0, p^+ of a planar cell are then computed from the summed-up log-odds as $p = \frac{e^L}{e^L+1}$. When the probability p^+ is higher than threshold θ_p, it vetoes the p^0, otherwise p^0 is used;

$$p_{x,y} = \begin{cases} p^+ & \text{if } p^+ > \theta_p\,, \\ p^0 & \text{otherwise}\,. \end{cases} \tag{5}$$

In the second step, a weighted sum of free and occupied planar cells is counted across the parking spot, with weights $2p - 1$ for a probably occupied ($p > 0.5$) cell and $1 - 2p$ for a probably free ($p < 0.5$) cell (the weights are normalised to $\langle 0, 1\rangle$),

$$n_O = \sum_{x,y|p_{x,y}>0.5} (2p_{x,y} - 1)\,, \quad n_F = \sum_{x,y|p_{x,y}<0.5} (1 - 2p_{x,y})\,. \tag{6}$$

Then the weighted counts of free and occupied planar cells are compared to a threshold and probabilities of free P_F, occupied P_O, and unknown P_U state of the spot are assigned as

$$P_O = \begin{cases} 1.0 & \text{if } n_O > \theta_{n_O}\,, \\ \frac{n_O}{\theta_{n_O}} & \text{otherwise}\,, \end{cases} \qquad P_F = (1 - P_O)\min(1, \frac{n_F}{a_{n_F} N})\,, \quad P_U = 1 - P_F - P_O\,, \tag{7}$$

where N is total number of planar cells in the spot, θ_{n_O} is a threshold for a number of occupied cells to decide the spot being occupied, and a_{n_F} is a weight representing the area of parking spot that should be visible to reasonably decide that the spot is free.

Using annotated testing data, we learnt the parameters as $\theta_p = 0.55$, $\theta_{n_O} = 10$, $a_{n_F} = 0.4$.

The label with the maximum probability is then assigned to the parking spot, and the probability is used as a confidence of this decision.

Each particular parking spot is classified independently, using current state of the occupancy grid. This means, that the label and the confidence of a particular spot are evolving in time, as new measurements become available. Usually the confidence of spot assigned as $free$ or $occupied$ increases as more data become available, but also a change from $free$ to $occupied$ label is possible when a smaller object is located inside a parking spot and it becomes visible only when the ego-vehicle approaches the spot.

The evolution of labels and confidences of parking spots is illustrated in Fig. 13.

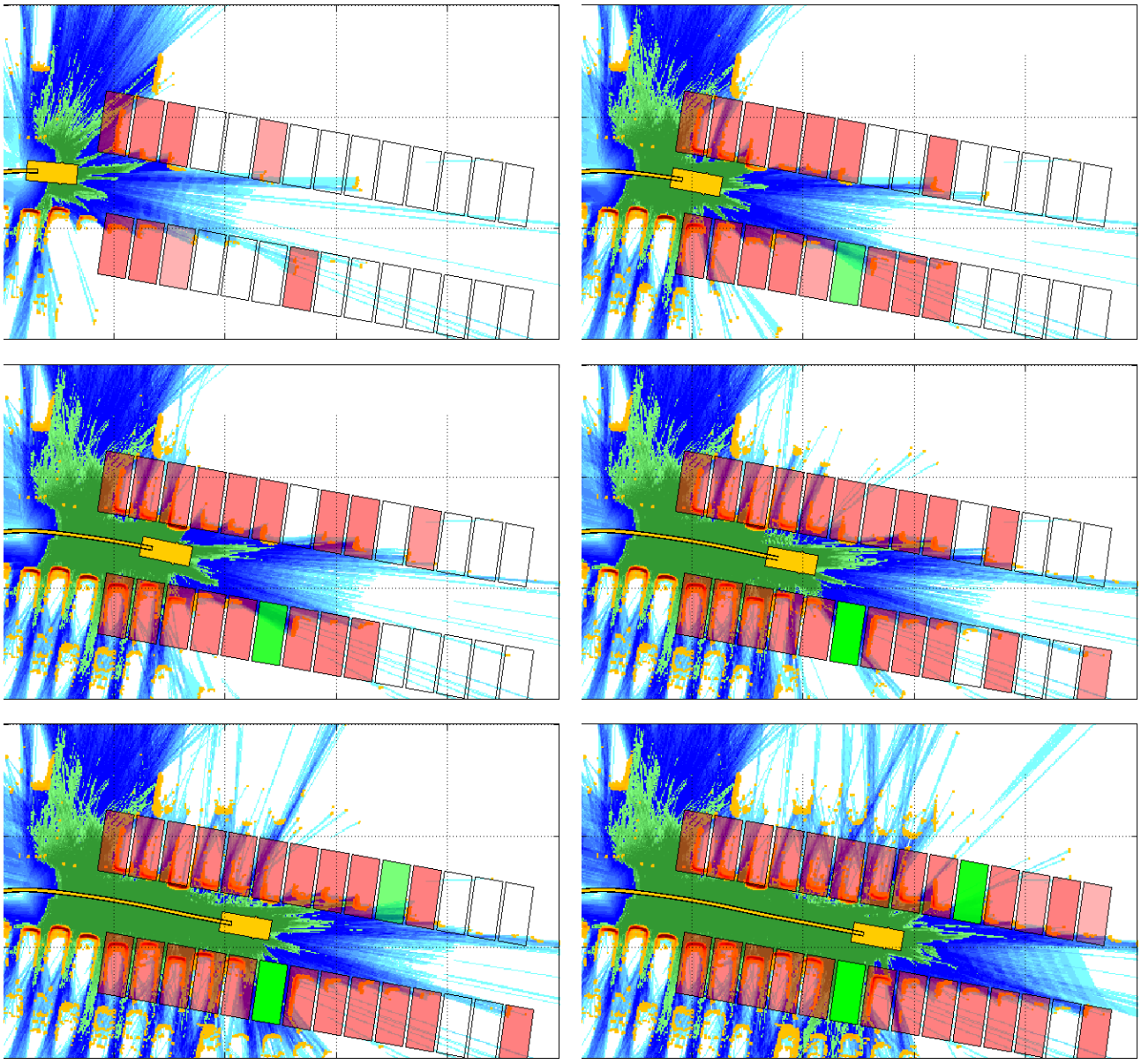

Figure 13: Evolution of classification of parking spots extracted from a map during a drive. The occupied spots are marked in red, the free spots are marked in green, the intensity of the colour encodes the confidence. Spots labelled as unknown are shown in white.

3 Image based perception module

3.1 Object segmentation and classification from image data

3.1.1 Semantic segmentation of images

Semantic segmentation represents the task of assigning a semantic class to each pixel in the image. It allows a detailed understanding of the scene by delimiting background from foreground objects. Nowadays, many state-of-the-art approaches are using deep convolutional neural networks. Two major factors have contributed to the increase in popularity of deep learning methods: larger datasets and more powerful hardware. Fully Convolutional Neural Network [6] have been extensively used for this task and have shown superior results to algorithms based on hand-crafted features. In the following part, we will describe the semantic classes, the dataset used for training, the architecture of the convolutional neural network and the training procedure.

A. Semantic classes

In this work, we have chosen the most relevant 23 visual classes for the traffic scene. Classes were selected based on their frequency and compatibility with other public datasets.

Table 1: Category and classes used for semantic segmentation

Category	Class
Flat	Road Pedestrian walking area Ground Parking Markings Curb
Sky	Sky
Nature	Nature object Nature surface
Traffic object	Traffic sign Traffic light
Construction	Building Fence Pole
Person	Pedestrian Rider
Two wheeled	Bicycle Motorcycle
Vehicle	Train Bus Truck Car
Other objects	Other objects

B. Image dataset

Deep learning algorithms need large dataset of annotated ground truth images for training. Medium-sized datasets that focus on the complex traffic scene are publicly available, from which the Cityscapes [2] dataset is the most complex, containing 5000 pixelwise annotated images and another 20000 images with coarse annotations for 19 classes. The dataset assures environment variability by recording inner-city scenes in 50 different cities from Germany.

Cityscapes images (Fig. 14) are significantly different from the UP-Drive area view images due to the nature of the employed lenses. However, the dataset captures complex traffic scenes, so we decided to use them for training our neural network along with our UP-Drive dataset. Our dataset (Fig. 15) contains 950 undistorted area view annotated images for 23 classes, from which 19 are the same as in the Cityscapes dataset.

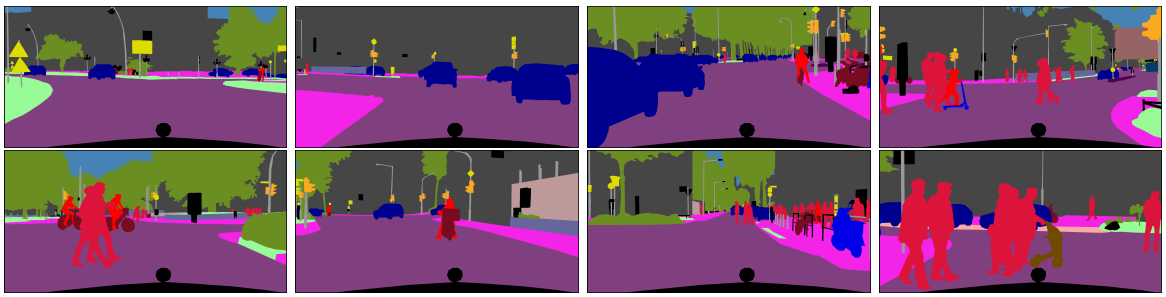

Figure 14: Cityscapes ground truth images with 20 classes

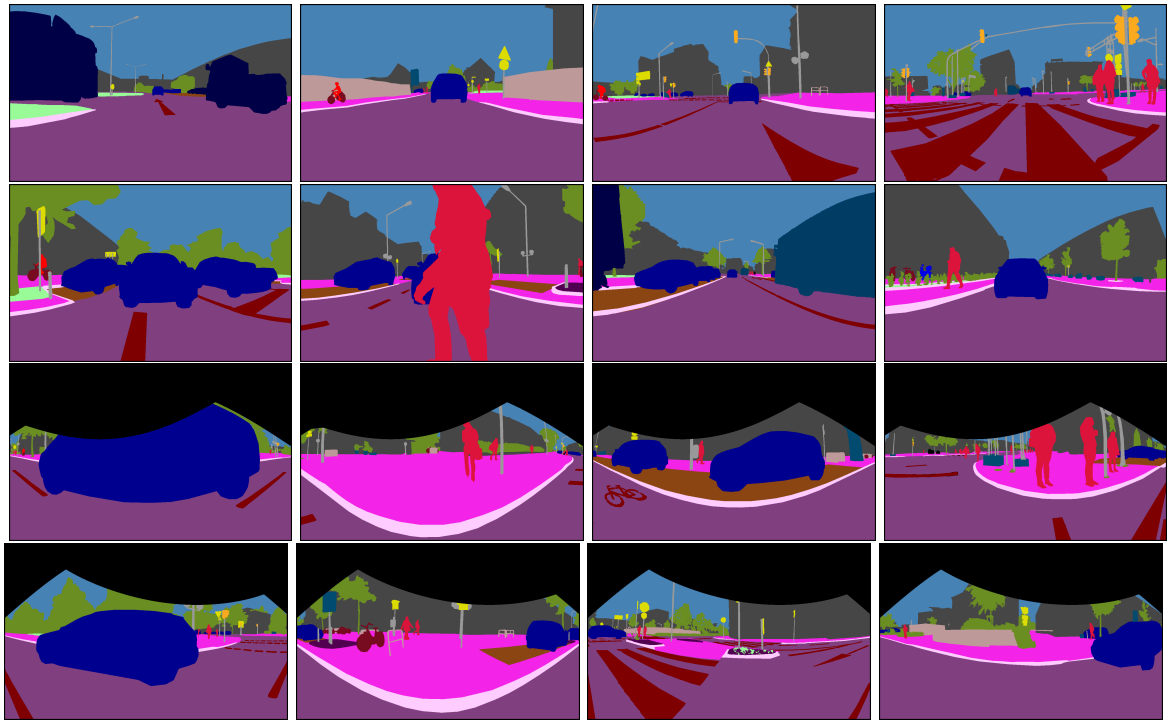

Figure 15: UP-Drive ground truth images with 23 classes: 1st row front view, 2nd row back view, 3rd row right view, 4th row left view

C. Convolutional Neural Network architecture

For the task of semantic segmentation we propose a Fully Convolutional Neural Network with an encoder-decoder architecture as seen in Figure 16. The network contains an encoder module which computes features at different scales and a decoder module which combines the features to obtain a higher resolution representation.

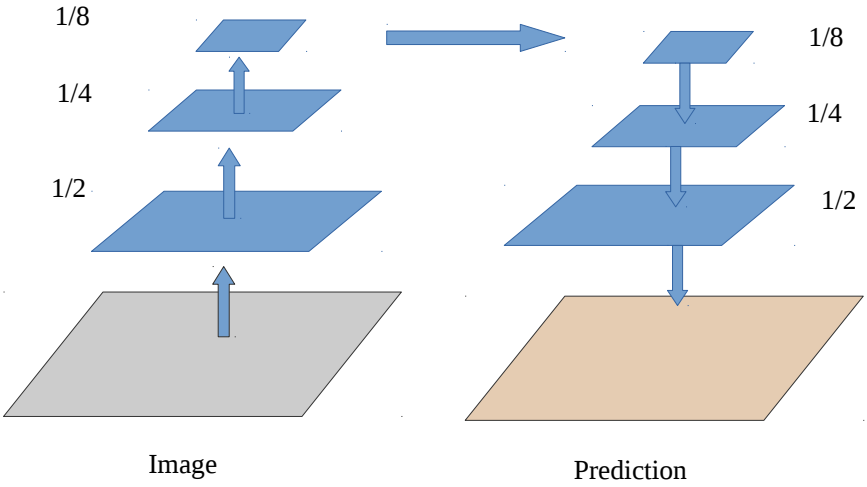

Figure 16: Encoder-decoder architecture

The network is entirely made of residual blocks [4] (Fig. 17) which alleviate the problem of vanishing gradient in training deep convolutional neural networks.

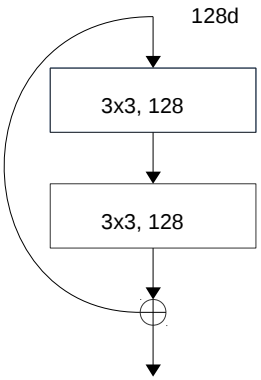

Figure 17: Residual block with two 3x3 convolutions and an identity connection

The network is made of 42 layers as seen in Fig. 16. First we perform resolution reduction by applying 3x3 strided convolutions. Next, a sequence of residual blocks generate features at 1/4 resolution. After a downsample operation, residual blocks with dilations are used for capturing context at different scales. We chose to design a lighter decoder with only 11 layers that recovers spatial and semantic information from the last layer of the encoder.

Table 2: Architecture for semantic segmentation of 1024x512 images

Encoder	layer name	output size	42-layer
		1024x512	
	conv1	512x256	Conv(3x3x64, stride 2)
	conv2	256x128	Conv(3x3x64, stride 2)
	conv3	256x128	ResBlock(3x3x64, 3x3x64) x 6
	conv4	128x64	Conv(3x3x128, stride 2)
	conv5	128x64	ResBlock(3x3x128, 3x3x128, dilated) x 8
Decoder	conv6	256x128	ConvTransposed(3x3x64, stride 2)
	conv7	256x128	ResBlock(3x3x64, 3x3x64) x 2
	conv8	512x256	ConvTransposed(3x3x16, stride 2)
	conv9	512x256	ResBlock(3x3x16, 3x3x16) x 2
	conv10	1024x512	ConvTransposed(3x3x23, stride 2)

C. Experimental results

The proposed architecture is trained and evaluated on the dataset comprised of Cityscapes and UP-Drive images having 1024x512 resolution. The entire dataset is split in 3815 images used for training and 605 for validation. We perform augmentation for UP-Drive images at training by using random flip and shift. We train several configurations, in which we use 3 resolutions: 1024x512, 1024x256, 512x256. Table 3 shows the evaluation results on the validation set and the metric used is Intersection over Union.

Table 3: Results

Resolution	23 classes	Forward time
1024x512	64.46 IoU	30 ms
1024x256	60.96 IoU	17 ms
512x256	59.35 IoU	9 ms

Our implementation is built on a custom C++/ CUDA framework based on Nvidia CuDNN. As far as execution time is concerned, the network runs in real-time as can be seen in Table 3. In Fig. 18 the segmentation results using 23 classes are presented for the area-view cameras.

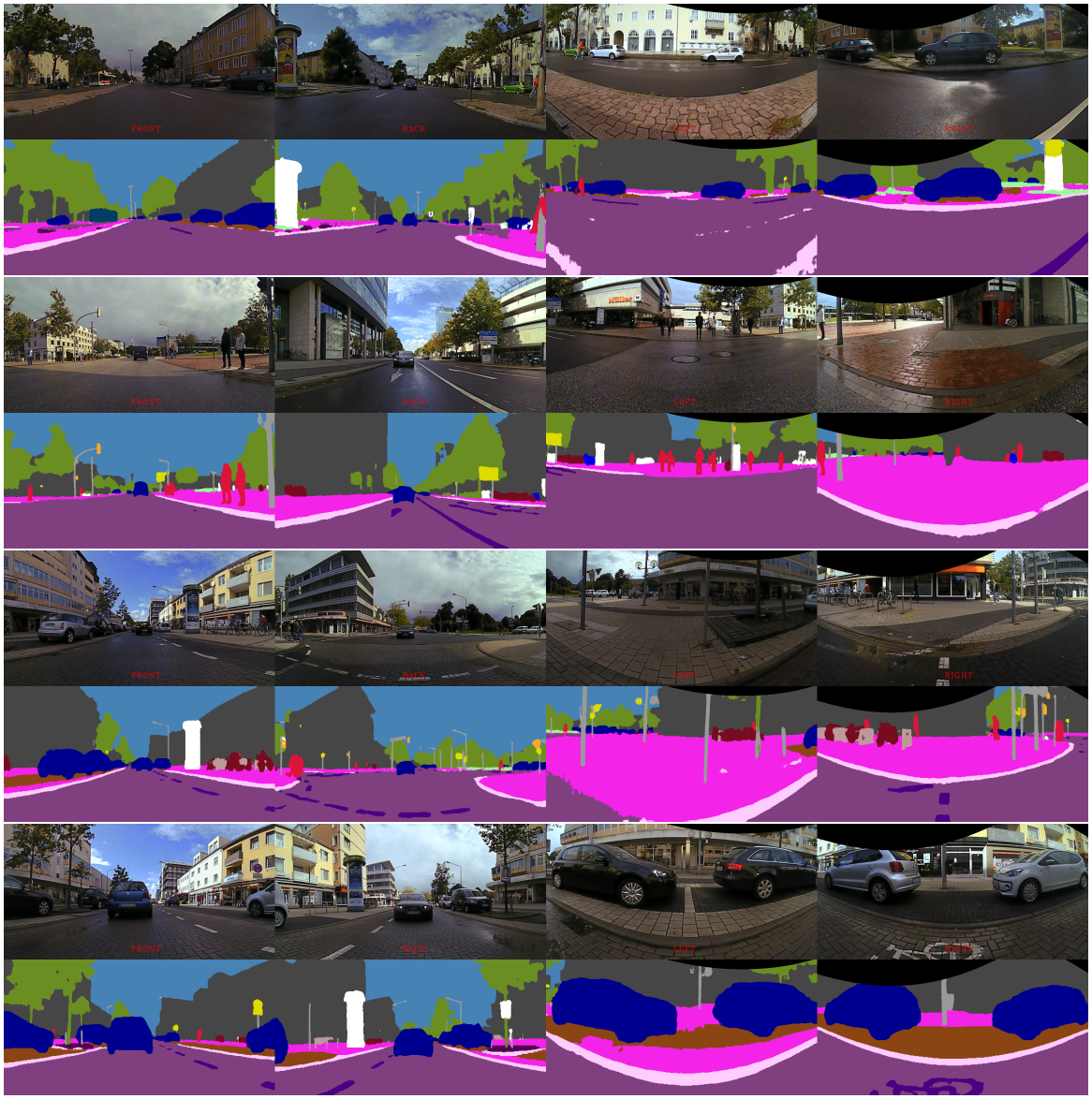

Figure 18: Segmentation results using 23 classes: 1st column front view, 2nd back view, 3rd left view, 4th right view

3.1.2 Instance based object segmentation from images

Semantic segmentation provides a pixel level semantic representation of an image, which relies on pixel level predictions. The receptive field of the prediction of a single pixel can be very large due to the complex multi-layer architecture of the employed deep convolutional neural networks. Semantic segmentation performs particularly well for background or stuff semantic classes that are represented by different textures. In the case of thing or object classes, semantic segmentation can struggle because prediction is achieved at pixel level (even with large receptive field) instead of being achieved at instance level. Object detection approaches are ideal because objects are recognized as a whole and each object is detected individually. Instance segmentation is a combination between semantic segmentation and object detection. Objects are detected individually and the objects are recognized at pixel level, i.e. a pixel mask is provided additionally to the object bounding box.

Multiple approach types have been considered for instance segmentation in the literature. Currently, the best results in literature are achieved by the Mask-RCNN framework [3] based

solutions. Objects are recognized using a deep CNN based region proposal network for object detection, classification and bounding box refinement. Each resulting bounding box is aligned with a 2D grid and binary mask is predicted for each grid point using also a fully convolutional neural architecture. The main advantage of this solution type is that it recognizes object instances using a powerful object detector, then it applies a pixel level fully convolutional newtork for finding fine object instance boundaries.

To achieve both semantic segmentation and instance segmentation over area view images for image based scene perception in the context of the UP-Drive project, we propose a common deep CNN based architecture. In order maximize the robustness of object detection we employ a feature pyramid network (FPN) to generate features for objects at different scales. The fusion of the pyramid layers is an efficient alternative for semantic segmentation compared to using dilated convolutions, resulting in similar recognition performances at an approximately three-fold reduction of memory requirements. The execution time of the fused solution is dominated by the feed-forward process of the base CNN body that generates the multiscale feature pool for both tasks. Typical architecture choices for best results consist of ResNet of ResNext architecture using 101 or 152 layers, however these architectures provide also high computational costs and struggle to perform at over 4 FPS on high-end GPUs. The deep CNN described in the previous section for real-time semantic segmentation represents an efficient tradeoff between recognition performance and computational cost.

The robustness of both tasks is also improved due to the joint end-to-end learning with the same common backend architecture. In oder to further improve the robustness of the two outputs we apply a simple fusion scheme. The semantic segmentation is used to divide the image into semantic categories, resulting in fine and precise boundaries between foreground and background classes. The semantic subclasses for background are determined from the semantic segmentation, and for instance categories from object detection and instance segmentation. This way an object is recognized always as a whole, and the object boundaries are determined from foreground/background separation at pixel level. Note that the standard object instance segmentation masks are based on the binary classification of 28 x 28 grids aligned with the object bounding boxes and struggle to find fine boundaries for large scale objects due to the very low resolution of the alignment grid.

In Fig. 19 we illustrate instance segmentation results for area-view images.

Figure 19: Instance segmentation results for area-view images

3.2 Road users signaling perception

For the purpose of signaling perception vehicle taillight pairs have been extracted from detected vehicles. Two methods have been explored to extract candidate taillight regions. Extracted candidate regions are paired by comparing their sizes, positions and colors. Each taillight of a detected pair is then tracked using Kalman filtering in order to deal with false negatives.

A. Candidate taillight extraction

1. Red region extraction using thresholds

The first method explored for candidate taillight extraction is proposed by Almagambetov et al. [1] and uses explicit thresholds for on images represented in L*a*b* color space in order to identify red pixels. Colors in the L*a*b* space are defined in the following way: The L* channel represents the lightness with L*=0 being the darkest regions and L*=100 being the brightest regions, the a* channel represents green at negative and red at positive values and the b* channel represents blue at negative and yellow at positive values. The thresholds used were determined empirically and are show in Table 4. After candidate region extraction morphological opening followed by morphological closing is done to eliminate noisy pixels and to merge regions that are close together. Convex hulls are then extracted tor each region. The main problems with this approach are that it does not work for red vehicles and that it is sensitive to illumination changes. The result of this method can be viewed in Figure 22.

*Table 4: L*a*b* thresholds*

Channel	Minimum	Maximum
L*	0	255
a*	140	255
b*	129	255

2. Taillight extraction using deep learning

The second method explored uses deep learning to segment taillights and was initially proposed in [11]. A fully convolutional network (FCN) [7] based on the VGG16 [8] architecture was fine-tuned for the purpose of taillight segmentation. VGG16 is composed of 16 weight layers out of which 13 are convolutional layers and 3 are fully-connected layers. VGG16 also has 5 pooling layers, each downsampling its input by a factor of two.

A FCN is obtained by replacing all fully-connected layers from a network with convolutional layers. A fully-connected layer can be transformed into a convolutional layer by setting the size of the filter to the size of the layer input and to set the number of filters equal to the number of neurons in the fully-connected layer. For example, in VGG16, transforming the first fully-connected layer with 4096 neurons, which receives its input of the shape 512x7x7 from the fifth pooling layer, gives a convolutional layer with 4096 filters of size 7x7. Unlike fully-connected layers, convolutional layers can have an input of any size therefore a FCN can also have an input of any size.

Due to the presence of the 5 pooling layers in the VGG16 architecture, the output of the network is 32 times smaller then the input. Transposed convolutional layers are used to upsample the output back to the original size. Figure 20 depicts the transposed convolution operation. Upsampling the network output 32 times gives very coarse results. In order to obtain finer results, the output of the third pooling layer is fused with the output of the fourth polling layer upscaled 2 times and the output of the last convolutional layer upscaled 4 times.

The fusion result is then upscaled 8 times to obtain the image of the original size. The fusion operation is an elementwise addition.

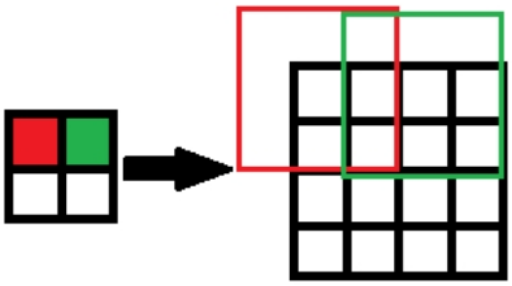

Figure 20: 3x3 transposed convolution with stride 2 and padding 1. The filter weights are copied over the output image and then they are multiplied with the input pixel value. The red pixel is multiplied with the weights situated inside the red window and the green pixel is multiplied with the weights in the green window. The weights where the windows intersect are added together.

The taillights from 677 images of the CompCars [10] dataset have been manually labeled and used to fine-tune a FCN. Figure 21 illustrates taillights segmented by FCN. The main advantage of the deep learning approach is that it can extract taillights from red vehicles.

Figure 21: FCN segmentation results

B. Taillight pair identification

As in [1], extracted regions are grouped into pairs if they pass two tests. The first test is a distance test where one candidate region, called "master", is compared with all the other regions. If the distance along their Y-axis between the "master" and another region is less then the height of the master and the area of the other region is $\pm 25\%$ of the height of the "master" region, then the first test is passed and the two regions are considered a pair. The second test is a color test where a 3D histogram is computed for each region that has passed the first test. To compute the 3D histogram, each color channel is binned into 8 bins obtaining a 8x8x8=512 bin histogram for each region. The histograms of each pair regions are compared using the Bhattacharyya coefficient and if the coefficient is smaller then 0.7

then the pair is removed. The Bhattacharyya coefficient measures the overlap between two statistical samples and is given in (8) where *n* is the number of bins in the histogram, *p* is the normalized histogram of the first taillight and *q* is the normalized histogram of the second taillight. The result of taillight pair identification can be viewed in the bottom-right image of Figure 22.

$$BC(p,q) = \sum_{x=0}^{n-1} \sqrt{p(x)q(x)} \tag{8}$$

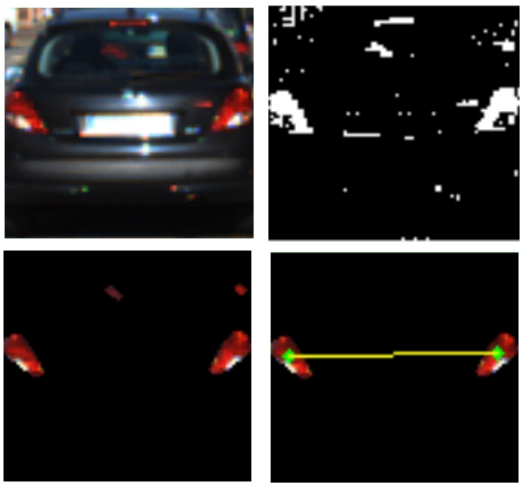

Figure 22: Vehicle detection result. Top-right: Thresholding result. Bottom-left: Result after morphological operations and convex hull extraction. Bottom-right: Detected pair after distance and color test

C. Taillight tracking

Taillight pairs may not always pass the distance and color test due to occlusions or changes in illumination. To deal with this, Kalman filters are used to track the centroid of each detected taillight over time. A Kalman filter can predict the next position of the centroid that it is tracking based on the position in the current frame.

4 Enhanced perception module

4.1 Road users perception by associating semantic labels to 3D objects

4.1.1 Depth map generation and occlusion handling

In this section we discuss an important practical problem which arises during sensor fusion and address it with two different approaches.

During the information fusion between the LIDAR point cloud and the segmentation results, semantic labels can be transferred to 3D points erroneously. This happens when the LIDAR sensors observes an object that is occluded in the camera view. The LIDAR points that originate from the occluded object are projected onto the occluding object in the camera view. Any semantic label transferred to the occluded object in this way is a mistake.

We presume that objects in the scene are static or, equivalently, their position has been corrected. The presented approaches work only in this context. For objects such as moving cars, additional information is needed to correctly estimate their position and consequently to handle occlusions.

We propose two main approaches: one based on only the 3D point cloud and one which makes use of the available semantic segmentation of the images.

Occlusion handling in the 3D domain

The first step is to project the 3D point cloud onto the image. This results in sparse measurement points with distances associated to them. We call this representation the raw or sparse **depth map** and denote it by $D(i, j)$, where we either have the real distance in position (i, j) or we signal that no measurement is available.

To solve the sparsity issue we consider cells of fixed size s (e.g. 10x10 pixels) and construct a lower resolution depth map by taking the minimum distance in each cell.

$$D_{lr}(x, y) = min_{i,j}\{D(i, j)|x = [i/s], y = [j/s]\} \tag{9}$$

Other more sophisticated interpolation methods can be applied if higher accuracy is required such as fitting a plane onto the existing measurements in each cell, although this is correct only if the measurements originate from the same object.

Besides the sparsity, another common problem is the lack of measurements. The main causes for this are non-reflective surfaces of the objects from the scene. To address this issue we propose a correction scheme.

For each position (x, y) from the lower resolution depth map, find the first neighbor upwards and downwards which have a smaller distance measurement: $D_{lr}(x, y_u) < D_{lr}(x, y) - t_{dist}$ and $D_{lr}(x, y_d) < D_{lr}(x, y) - t_{dist}$. Search is limited to a fixed number of neighbors and stops at the first one which satisfies the condition. If either neighbor is not found, the current value is directly stored in the corrected low-resolution depth map: $D_c(x, y) = D_{lr}(x, y)$. Otherwise, we set the distance to the minimum of these: $D_c(x, y) = min(D_{lr}(x, y_u), D_{lr}(x, y_d))$. The aim of this scheme is to fill in gaps in the measurement data by searching vertically for closer points. This is clearly incorrect for objects that do not have a fronto-parallel profile, but most objects of interest meet this criterion.

Occlusions are detected using the corrected low-resolution dense depth map. Simply mark projected points which have a larger distance than the corresponding cell from the depth map as occluded. To such occluded points no semantic class is associated. Figure 23

illustrates the resulting low resolution depth map.

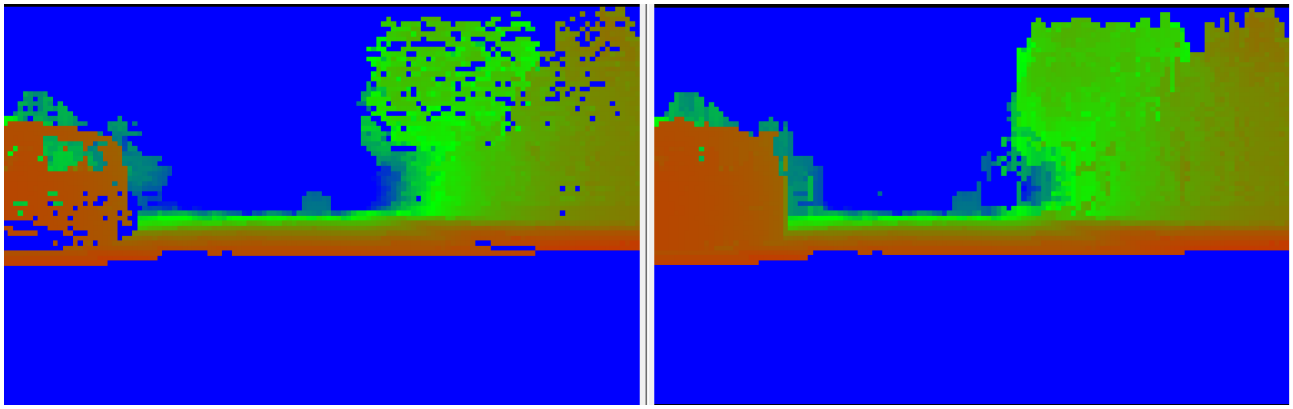

Figure 23: Low resolution depth map and its corrected version - scene depicts a car close by on the left, a car in the middle farther away and foliage on the right. Color encodes the distance from red (close) to blue (far), up to 20 meters.

Occlusion handling using semantic segmentation

We propose the use of semantic segmentation for inferring missing measurement data. If a vehicle is measured partially by laser scanners, the semantic segmentation can help to extend these measurements.

We consider each individual pixel column in the image and partition it based on the semantic segmentation, resulting in a semantic stixel-like representation with stixels having a width of 1 pixel. For each stixel we retain the closest 3D measurement (from camera point of view). We extend the 3D measurement for the whole stixel and generate the depth map by projecting these extended measurements into the image plane as seen in Fig. 24. In order to check if a 3D measurement is occluded the 3D distance has to be compared with the corresponding value in the occlusion mask (depth map). If the distance is larger, it means that the 3D point is occluded and does not receive a semantic class from the image. We extend the 3D measurements only for semantic classes that can represent vertical structures and avoid sky or horizontal background classes such as road, lanemarking, sidewalk or ground.

If each pixel is associated to a unique stixel then we can obtain the depth map in the following way:

$$D_{stixel}(x,y) = min_{y'}\{D(x,y')|y' \in Stixel(x,y)\} \tag{10}$$

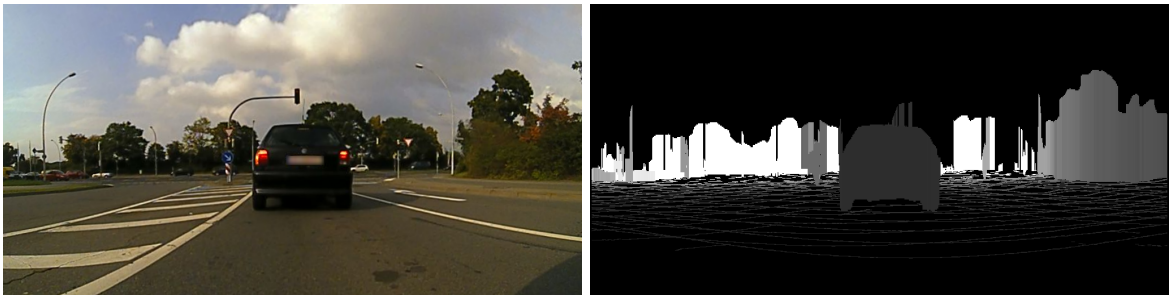

Figure 24: Original image and depth map generated using semantic stixel partitioning. Gray level encodes the distance from dark (close) to light (far)

4.1.2 Semantic label association to 3D objects

After the cloud of lidar measurements is enhanced with semantic information, the semantic class for each 3D obstacle is established. This is done statistically, in order to compensate for various errors in the semantic class of the lidar points (on the borders of dynamic obstacles, or due to the different view point of the cameras versus lidars). The following steps are performed for computing the semantic class for each obstacle:

- For each lidar measurement that has semantic class, the class of the corresponding voxel is updated. If more lidar measurements from the same voxel have different semantic classes then the final voxel class is considered unknown.
- For each 3D obstacle the frequency of each class is computed, for the contained voxels. This is done via a histogram. The reason for this statistical approach is that the borders of moving or thin obstacles (poles) can be contaminated with semantic classes from the background.
  - The class of each 3D obstacle is selected as the most frequent class
  - For further improvements, the most likely k (2, 3?) classes can be provided for establishing the object class with subsequent tracking.

Results for obstacle segmentation are presented in Fig. 25.

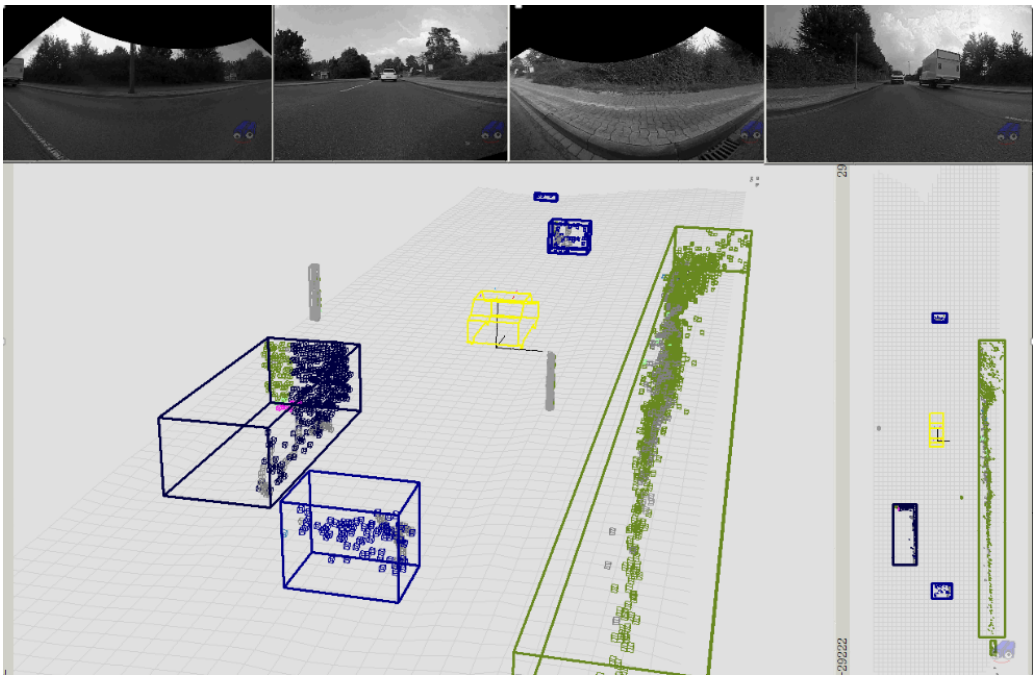

Figure 25: The voxel space enhanced with semantic segmentation (each voxel is labelled accordingly). Classes of vegetation, pole, car, truck are shown with different colors. The cuboids are drawn accordingly with the most frequent class present in the obstacle

4.2 Road users perception based on supersensor low-level data representation (STAR)

Road users can be detected starting from the low-level STAR representation. In this representation each 3D point has associated to it color information based on the position of its projection onto each camera image. If semantic segmentation is available the point also has

access to this data. Also, a grouping of entities based on the image view and the semantic segmentation is available in the form of an object id. This representation has the advantage over the raw 3D points that additional information is available before grouping of the points is performed.

Based on this representation road users can be detected by grouping 3D points from the birds eye view together. The first stage entails grouping together points that have the same semantic class and same object id. The proximity of the points is also a criterion. We provide cuboids which have their sides aligned with the car coordinate system. The limits of the cuboid are determined based on the points inside. The lower height limit is fixed to coincide with the ground plane.

The second stage involves validation and correction of the objects. Several constraints are applied to cuboids and the 3D points belonging to them such as: minimum number of 3D points; dimensions in all three directions must be within a given interval; the semantic classes of the 3D points must be sufficiently uniform. Specific limits are established for vehicles (classes: bus, car, truck, train) and vulnerable road users (classes: pedestrian and rider).

Figure 26 shows an example detection result using the presented approach.

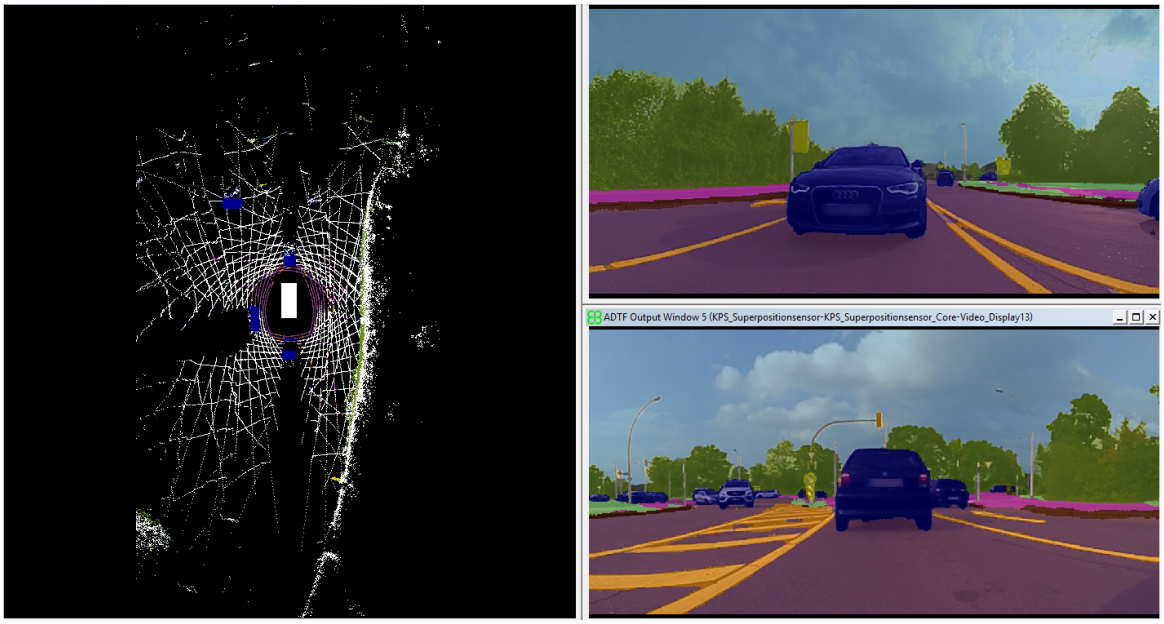

Figure 26: left: Bird's eye view with detected vehicles (blue) and ego-vehicle (white); right top: segmentation of rear view; right bottom: segmentation of frontal view

4.3 Road users tracking and object position correction

4.3.1 Tracking

Object tracking represents a process in which sensor measurements are used in order to determine the location, path and characteristics of certain objects in the scene [1]. Object tracking usually seeks to encapsulate multiple unique traits of the tracked object such as object identities, velocities, positions, orientations and in the context of autonomous driving the class of objects. For the task of vehicle perception, the task of object tracking is essential, as the environmental measurement is useful only if it is filtered (not noisy) and identifiable even in occluded situations such that the vehicle is capable of making use of the measurement

and transform it into an actionable information. The implemented general flow of the tracking process presented in the context of the Bayesian filtering framework is depicted in Fig. 27.

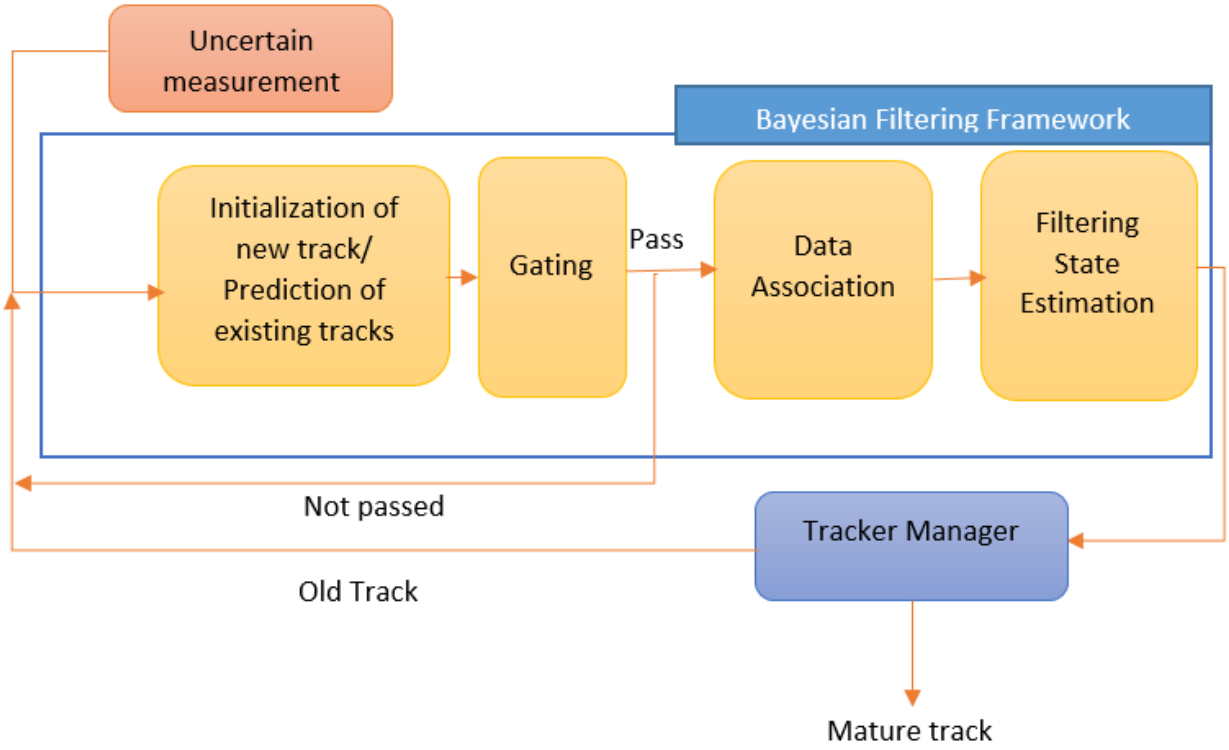

Figure 27: General processing pipeline of the Bayesian filter framework

The objects measured from the 360-degree lidars are provided in the form of cuboids. For the purpose of our problem we will assume that the objects are moving in the x and y directions. There are multiple motion models for describing the vehicle movement. The tracking dynamic model can be described mathematically by a discrete time, stochastic space model in the form of:

$$X_{k+1} = f(X_k, u_k) + w_k \tag{11}$$

$$Z_k = H(X_k, u_k) + v_k \tag{12}$$

The motion model assumed is the CTRV motion model, which has a state vector described by the expression bellow. This model incorporates the vehicle position on x and y, the velocity, heading and heading angle.

$$X_k = \begin{bmatrix} x \\ y \\ v \\ \Psi \\ \dot{\Psi} \end{bmatrix}$$

The evolution of the vehicle motion from time t to time t+T is depicted in Fig. 28.

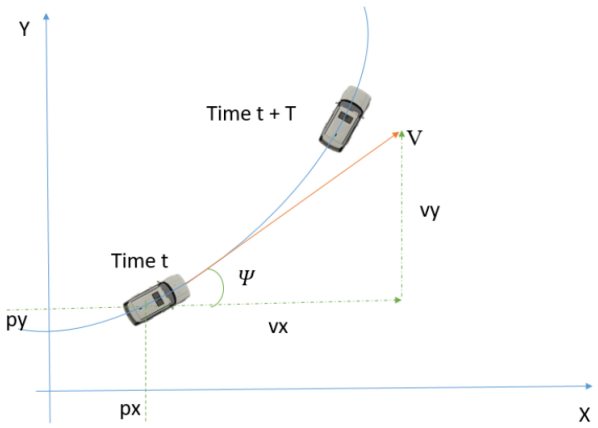

Figure 28: Vehicle transition from time t to time t+T

The transition function between states is:

$$X_{k+1} = X_k + \begin{bmatrix} \frac{v}{\dot{\Psi}}(-\sin(\Psi) + sin(\Delta t\dot{\Psi} + \Psi))) \\ \frac{v}{\dot{\Psi}}(\cos(\Psi) - \cos(\Delta t\dot{\Psi} + \Psi))) \\ 0 \\ \Delta t\dot{\Psi} \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2\cos(\Psi)\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2\sin(\Psi)\nu_{a,k} \\ \Delta t\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2\nu_{\ddot{\Psi},k} \\ \Delta t^2\nu_{\ddot{\Psi},k} \end{bmatrix} \tag{13}$$

In case the yaw angle is 0 the process model becomes:

$$X_{k+1} = X_k + \begin{bmatrix} v_k\cos(\Psi)\Delta t \\ v_k\sin(\Psi)\Delta t \\ 0 \\ \Delta t\dot{\Psi} \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2\cos(\Psi)\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2\sin(\Psi)\nu_{a,k} \\ \Delta t\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2\nu_{\ddot{\Psi},k} \\ \Delta t^2\nu_{\ddot{\Psi},k} \end{bmatrix} \tag{14}$$

In the object tracking problem, sensor measurements and the states of the tracked objects are modeled as random variables. Due to the fact that such variables can take multiple values, the evolution of their values is computed using probabilistic inference. The Bayes filtering is the main theory used in solving and modeling the object tracking problem. In the context of object tracking Bayesian theorem can applied using two main functions:

- State Prediction: the current state of the tracked object is adapted to the most recent measurement time of the most recent sensor observations. The evolution of the system is described by a process function. In the case of autonomous driving such process function is based on a motion model.
- State update: the state prediction results in a probabilistic representation of the system state at the most recent time stamp in the measurement domain. If the sensor observation sample is available for comparison, the likelihood of the sensor model is evaluated at the position of the observed measurement, taking into account the state hypothesis (prediction). Using the likelihood, the state is adapted to the most recent available knowledge. The final result is obtained by the probabilistic combination of the previously accumulated state using the prediction and the latest sensor measurement.

The Kalman Filter is based on the Bayesian method and it computes recursively the optimal parameter estimates from its posterior density. The Kalman Filter assumes the object

dynamic function and posterior density are Gaussian distributions and the process and measurement functions are linear. As we know in the real world the object dynamics cannot be captured by linear motion model so the Kalman Filter had to be adapted to encapsulate a non-linear motion and measurement model. Two approaches were introduced, the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). The EKF uses a first order Taylor series expansion approximation to linearize the process function. However, linearization using Jacobians is computationally expensive so for this reason in this project we have used the UKF, which makes an approximation based on the so called sigma point sampling. The UKF generates a set of sigma points and then propagates them through the non-linear process function. The Gaussian can then be recovered from the newly transformed points. The resulting probability density function is an approximation of the Gaussian distribution, so the UKF is not an optimal algorithm. Even so the UKF is used widely in LIDAR MOT as presented in [2,3,4] due to its low computational complexity comparable to the KF.

4.3.2 The Unscented Kalman filter steps

Generating a set of sigma points: The first sigma point is the mean.

$$X^0_{K|K} = X^*_{K|K} \tag{15}$$

The rest of the points are generated around the mean with a spreading factor of λ

$$X^i_{K|K} = X^*_{K|K} + \sqrt{(\lambda + n_x)P_{K|K}} \tag{16}$$

$$X^i_{K|K} = X^*_{K|K} - \sqrt{(\lambda + n_x)P_{K|K}} \tag{17}$$

Prediction The sigma points are fed through the process function. The weights are calculated as described below. As it can be seen the weights depend on the spreading parameter lambda (Lambda was used before to set how far we want to set the sigma points). Before we had a covariance matrix and generated sigma points, and here we are doing the inverse step, we have predicted sigma points and we want to recover the covariance matrix. So we also want to invert the spreading of the sigma points. This task is performed by using the weights:

$$w_i = \frac{\lambda}{\lambda + n_a}, i = 0 \tag{18}$$

$$w_i = \frac{1}{2(\lambda + n_a)}, i = 2, ..., n_a \tag{19}$$

The mean and covariance are generated using the formulas below.

$$X_{K+1|K} = \sum_{i=1}^{n_\sigma} w_i X_{K+1|K,i} \tag{20}$$

$$P_{K+1|K} = \sum_{i=1}^{n_\sigma} w_i (X_{K+1|K,i} - X_{K+1|K})(X_{K+1|K,i} - X_{K+1|K})^T \tag{21}$$

Update In the update step, since the measurement model is linear (we are receiving x, y coordinates similar to the ones in our state model) we will not have to perform any linearization procedure. We are computing the Kalman gain based on the equation below:

$$K = P_k H^T (H P_k H^T + R)^{-1} \tag{22}$$

$$X_k = X_k + K(z_k - H X_k) \tag{23}$$

In the equation bellow we are updating the state and covariance based on the measurement readings:

$$P_k = (I - KH) P_k \tag{24}$$

$$X_k = X_k + K(z_k - H X_k) \tag{25}$$

The results of the tracking process are depicted in Fig. 29.

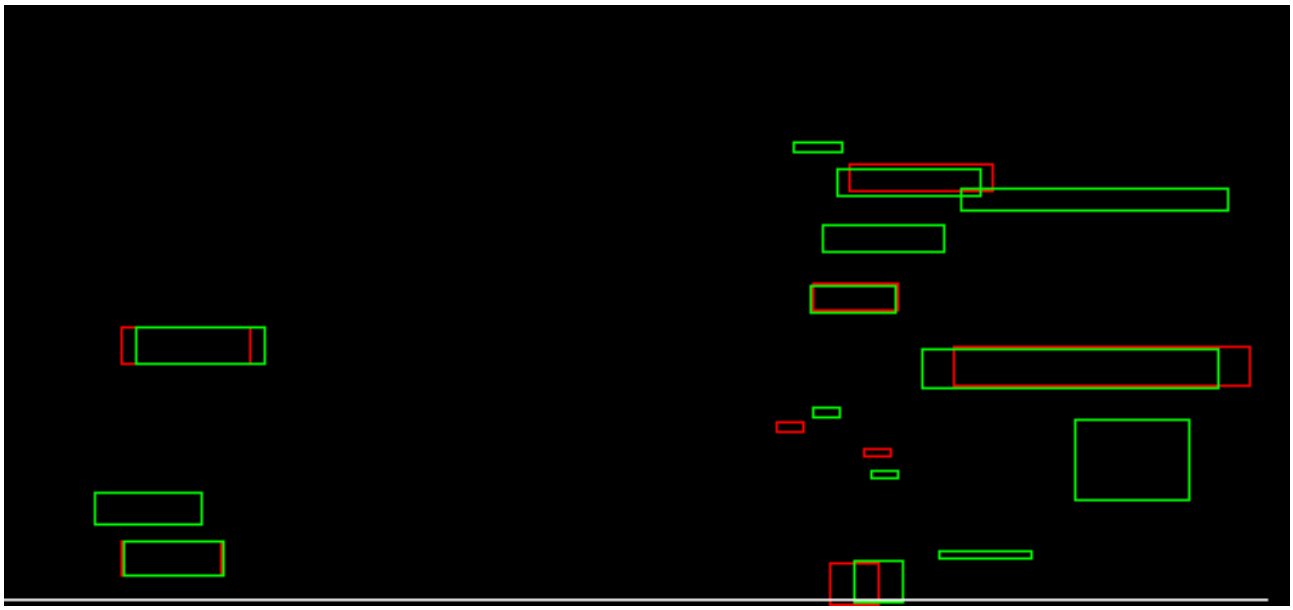

Figure 29: Multi-object tracking example

The object predictions are depicted with green rectangles, the object measurements are illustrated with red rectangles.

4.3.3 Data association

The data association process seeks to group the object detection result into a tracking filter. There are two classes of data association filter: the deterministic data association and the probabilistic filter. One deterministic data association filter is the nearest neighbor filter, which updates each track with its closes measurement. Some metrics for computing the nearest neighbor are the Euclidean distance or the Mahalanobis distance. The probabilistic data association filter performs a weighted update of object state using all association hypothesis, to avoid single hypothesis association errors. The association implemented in the current framework consists of two steps. In the first step we project all object measurements onto a 3D grid. We then project the tracks onto the grid and compute the maximum overlapping between each track and measurement. We perform a gating process such that only measurements that are situated in a validation ellipse around the track are considered. This is performed such that we limit the search space. Each track contains a list containing

the maximum overlapping with each object measurement. We search in this list, and return the object having the maximum overlapping, in case the maximum overlapping percentage is higher than 5%. We then update the track with the found measurement. In Fig. 30 with red we depict the object measurements, with white empty rectangles we illustrate the tracks, the yellow rectangle regions represent the intersection with the measurements and with a thin blue line we have shown the final association between a track and measurement. As we observe there can be intersections with multiple object, however we are taking just the measurement having a maximum intersection. There can also be tracks propagated from previous frames that have not been associated, we maintain a history of 10 frames for each track.

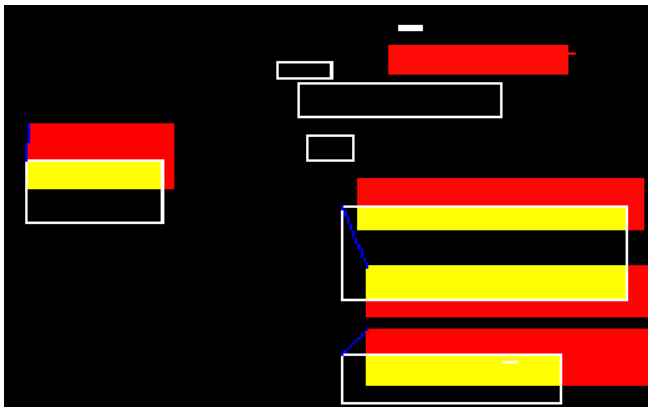

Figure 30: Grid-based multi-object data association example

Since the synchronization between the velodyne sensor and the reference time stamp is not perfect, and also in the point cloud motion compensation task, which is performed before the object detection stage, we are considering a transformation for each point in the original cloud using only the ego motion, and not the speed of each individual point (in case of moving objects), certain position errors are accumulated. The effect of these errors can be that the state predictions may not overlap over the measurements at all. For this reason in the second step of the association we are considering a modified nearest neighbor approach. We have marked all associated tracks and measurements used so far so that we will not use them again for associations in the current frame. For the remaining tracks we try to find the best measurement that minimizes the following function:

$$f(x) = \begin{cases} |px| < 0.0001 \vee |py| < 0.0001, INFINITY \\ euclidean < 11 \vee sizeDiff < 2, INFINITY \\ \frac{0.7Euclidean + 0.3sizeDiff}{sizeDiff + Euclidean} \end{cases} \quad (26)$$

where *abs* refers to the absolute value, *px* and *py* are the *x* and *y* positions of the object, Euclidean represents the Euclidean distance in meters between the position of the track and the position of the measurement, *sizeDiff* represents the size difference with respect to the width of the object and *INFINITY* represents a very large value.

5 Sensor fusion module

The previous sections give an overview of the target high-level perception system as specified and designed at the beginning of the project. However, during the project we found that additional modules are helpful and needed. In order to test the whole processing chain in an early development stage, from sensors via perception and prediction to path planning, and in order to decouple the development of the path planner from those of the previous modules we set up a classical object and grid fusion (see Sec.5.1 and 5.2). Furthermore, inaccuracies of the localization avoid reproducible testing and parameterization of the processing chain. For that, we add an additional localization step which checks the consistency of the map and refines the localization by using curb information (see Sec.5.2.2). Finally, we found crossing intersections as a particular challenge for automated driving. Therefore, we add information about the right of way and conflict area to the road graph representation (see 5.3).

5.1 Object Fusion

To be able to drive autonomously in a dense urban scenario, the ego vehicle should be able to perceive the static and dynamic obstacles and to distinguish them not only from each other but also from the background road surface.

Object fusion is the module in the perception layer that works on the problem of how to combine data from multiple and diverse sensors to make robust and general inferences about the surrounding 360 degrees environment. The idea is to combine data from multiple sensors to perform inferences about the dynamic object states (position, velocities etc.), their identities (pedestrian, motorbike, car etc.) and other detection characteristics (existence probability, false alarm rate, tracked statistics etc.) that are not possible from a single sensor alone. Other advantages include increasing the spatial coverage by individual sensors and increasing redundancy in case of a breakdown and adverse weather conditions.

The core of the object fusion module is based on an extended Kalman filter. An autonomous and high level processing flow architecture approach is used for data fusion. This means that each sensor is allowed to do a maximum amount of processing to generate the state vectors and the identity information. The data alignment, association and classification etc. are then performed on the state vectors rather than on the raw data. This is also known as track to track fusion. Even though these are well known techniques, the overall performance of the module depends on better association methods and on non Kalman state details like the identity information (classification), the existence probability (false positive-negative), perceived intention of the dynamic objects (brake light, blinking light for a turn) etc.

The identification process seeks to combine the identity data measured either by the sensors directly (e.g. in case of a camera) or by using other parametric data that can be indirectly related to the identity of the object (velocity measurements from a Radar sensor or dimension measurements from a Lidar). There are two types of classification approaches, soft decision or a hard decision. Soft decision approach communicates partial evidence in the form of probabilities or associated confidence for a track. A hard decision approach on the other hand uses a threshold value to give binary information regarding the classification. A soft decision approach is adopted in the classification module for object fusion. Several different algorithms were tried to achieve the optimal results for the identity fusion. This includes e.g. (a) priority based approach where sensors are prioritized in the case of conflicts (camera classification is prioritized compared to a Lidar classification), (b) the highest classification

value wins (one Lidar sensor is certain about the classification compared to the other uncertain Lidar sensor), (c) a Hidden Markov Model (HMM) chain is implemented in the case where none of the perception sensor provides the identity information. The model takes then into account the dimensions and the velocity information etc. to guess about the classification. For example, a pedestrian cannot be moving with a velocity higher than 10 m/s or a pedestrian cannot be bigger than a few meters in dimensions. This method however is still prone to errors in the case where the resolution of the sensors is not good enough e.g. it is difficult to distinguish between a pedestrian group and a truck moving slowly because both the dimensions and the velocity profile are the same.

Ancillary functions in a system e.g. database management, evaluation, human machine interaction etc. comprise of a major function of the evaluation function. The interfacing between the sensor layer, the fusion layer and the driving function layer e.g. forms a major portion of these ancillary functions. The conversion from the proprietary sensor interfaces from different vendors to a common standard communication interface is especially important because this if not handled properly, it might lead to the loss and corruption of intended information. The interface should be generic and flexible to not only handle information from diverse sensor types from different senders but also able to adapt to the quickly changing requirements. A new interface (namely KPS-interface) was designed and implemented taking into account the lessons learnt during the span of several years in several other projects. This interface is not used both at the input and the output side of the sensor fusion module.

Long term interpretation and intention prediction (I&P) is an important part of the drivers function. In order to have a well-established I&P module, the intention indicators from different road users need to be perceived and passed on to the I&P module. This e.g. includes the hand signs by the pedestrians and cyclists etc. As a first step to this, the object fusion module implemented the identification of the car brake light indicators and the turning maneuver indicator lights, fusing the given information from different sensors and then passing it to the other modules.

False positives and false negatives is another critical problem that needs to be handled by the object fusion module. A false negative such as missed obstacle, an imperfect reading of the road lane, an incorrect speed estimation of other vehicles, etc. could significantly affect navigational safety (majority of the autonomous car accidents reported today are due to false negatives) and thus methodology to detect and minimize them should be implemented in the object fusion module. However for a comfort driving function, false positives also are unwelcomed as it forces the driving function to break unnecessarily. Therefore a parameter for indicating the confidence of each track in the object fusion module is implemented in the form of a probability. The value of the parameter increases or decreases based on how many number and how many different types of sensor measurements are associated with the track. This also takes into account the field of view of the sensors. The value of the existence parameter decreases e.g. if the sensor is not delivering the measurement for the said track even when it is in the field of view of the said sensor. The positive and negative update rate of each sensor is based on heuristics and is parameterizable.

5.2 Grid Fusion

As described in deliverable 4.1 Section 9.1 we use occupancy grid map as a well-established representation of the static environment. Here, we planed to distinguished between a far range grid for well-structured highway and rural scenarios, a near range grid for urban scenarios including parking areas and a traversability grid for estimation of road side infras-

tructure. However, we found that especially for urban environments the curb geometry and location is also an essential information. Therefore, we also calculate a curb grid in parallel to the other grids.

It is worth noting that for Object and Grid Fusion we employ the Late Fusion method rather than the planned Super-sensor. The spatio-temporal and appearance based representation (STAR) that represents the basis of building a virtual super-sensor is described in more detail in D4.2. Since the extensive development and proper implementation of the super-sensor requires significant amount of time and resources, the project partners took the decision to run both approaches in parallel. It was imperative to provide the necessary data to the other WPs especially WP6 “Scene understanding“ and WP7 “Decision making and navigation“. By employing the Late Fusion method we were able to provide the datasets required for WP6-7 development right after the first vehicle was fully operational in M6. Super-sensor module will be described in D4.6 “Final version of higher-level perception functions“ due in M42.

5.2.1 Curb Detection

We investigated all sensor modalities for their capability of detection curbs. The main challenge for a good detection performance is the wide range of appearance and the small height gap which defines the actual lateral position of the curb. We found that the velodyne sensors are best suited for curb detection in our case. There are two approaches to extract the curb position from the distance measurements of the 360° Lidar sensor.

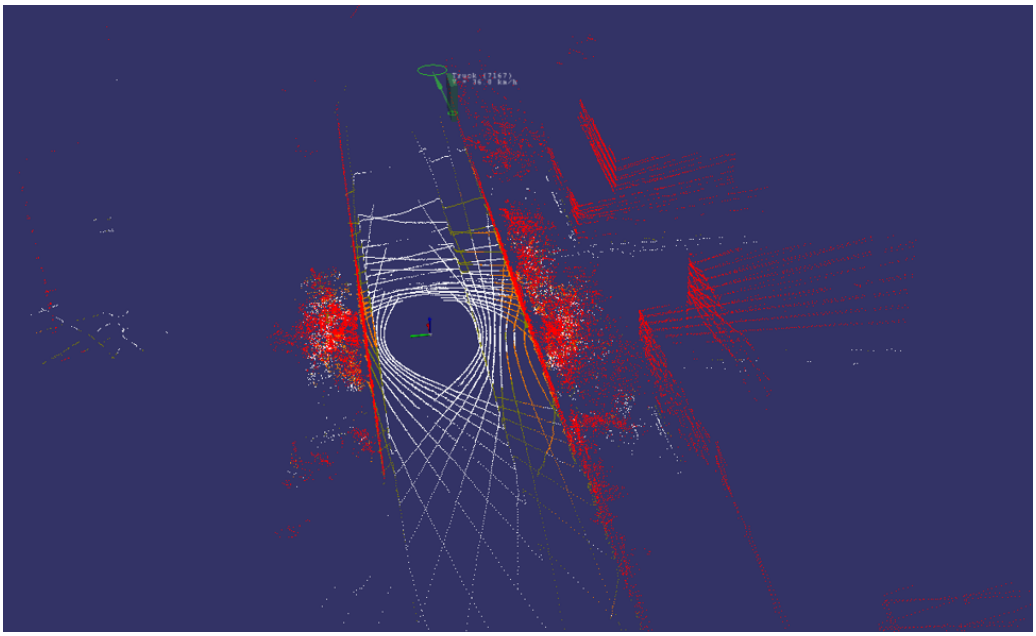

Figure 31: Classified curb points using approach 1 (see text)

(1) This approach tries first of all to approximate the ground surface around the ego vehicle by means of a surface model (Figure 31, white points). Curbs are then detected using a height threshold. So every point higher than the threshold above the ground surface is classified as an curb point (brown points). Points classified as obstacles are shown in red.

(2) The other solution is to extract curb points directly from the scan lines of the 360° Lidar sensor (Figure 32). So instead of searching for height gaps, we try to find gaps in the

scan lines (blue and red points). For that we iterate through each scan line and extract gaps by means of an moving average approach. The main advantage is that even small curb gaps can be detected.

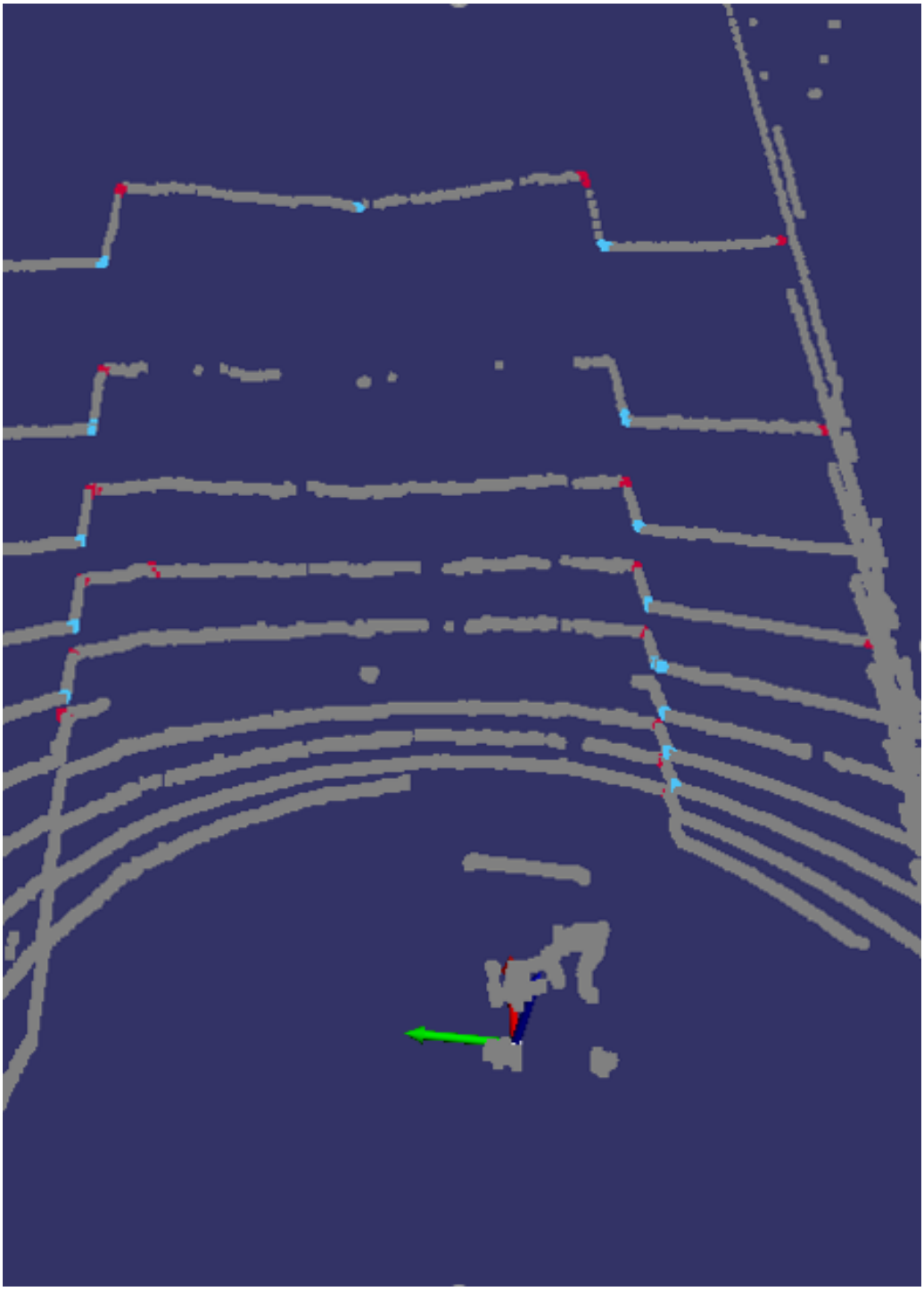

Figure 32: Classified curb points using approach 2 (see text)

Figure 33 shows the final result of the curb grid. As can be seen the curb is clearly detected (red). Only the small detection range of about 15-20m is an open issue. Unfortunately, the accuracy and sensitivity of the 360° Lidar was not good enough to increase this distance.

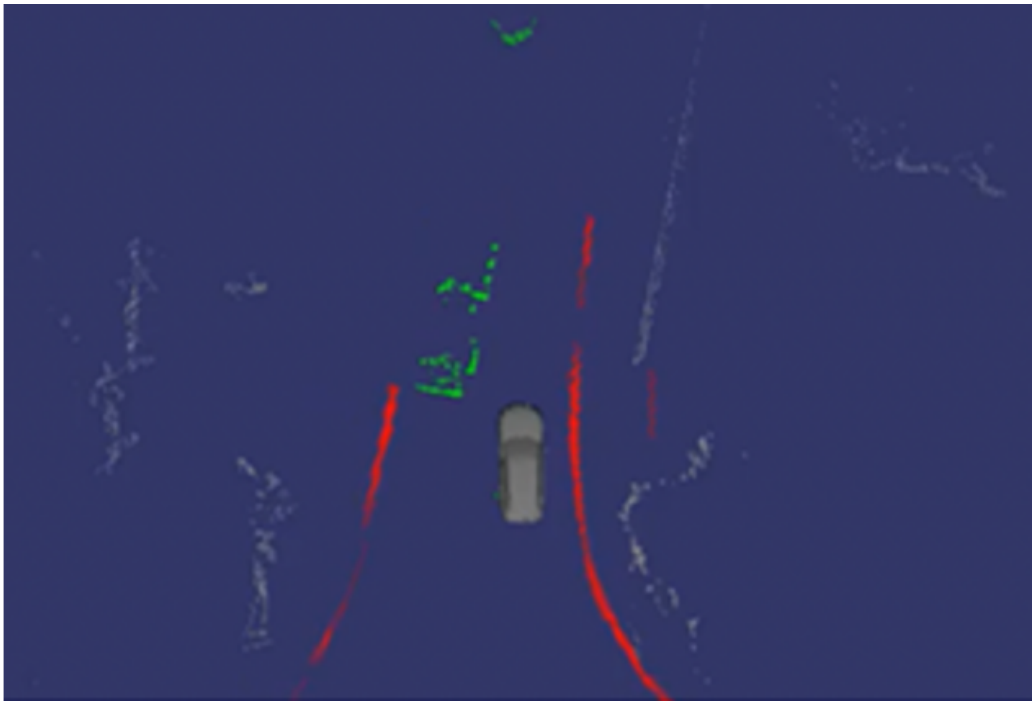

Figure 33: Result of the curb detection (shown in red)

5.2.2 Map Checker

Autonomous driving in urban environments requires an accurate positioning within the lane. Additionally, we need a module which is supervising the consistency of the map including the localization. For this two tasks we set up a module "map checker".

The map checker uses the curb grid as its main sensory feature. The main idea is to compare the curbs detected by the sensors with those expected based on map data. Usually, if the map is updated/accurate and the localization is precise, one would expect perfectly aligned sensor and map curbs. This alignment can be estimated by means of a distance function which in the simplest case is just the sum of squared distances of all grid cells. High distance values indicate that sensor and map curbs are not aligned or dissimilar to each other. Low distance values indicate higher similarities.

The map checker can directly use this distance function to estimate the consistency of the map including the localization. Furthermore, we can use the distance function to improve the localization in a certain range. For that we just have to generate local samples and evaluate each sample using the distance function. The sample with the lowest distance can be taken as the winner and can be used as the new localization result. Figure 34 shows an example. The detected sensor curbs are shown in red, the expected curbs from the map are shown in blue. As can be seen, there is a large lateral offset between the sensor and map curbs (in this case about 3.0m). After applying the map checker the map curbs (green) are aligned to the sensor curbs.

In summary, Map Checker described above could be considered as an additional supplementary tool for Localization, that we use in the context of WP4 for high level validation of the map with the sensors data available. For more details regarding Lifelong Localization and Mapping please refer to D5.2.

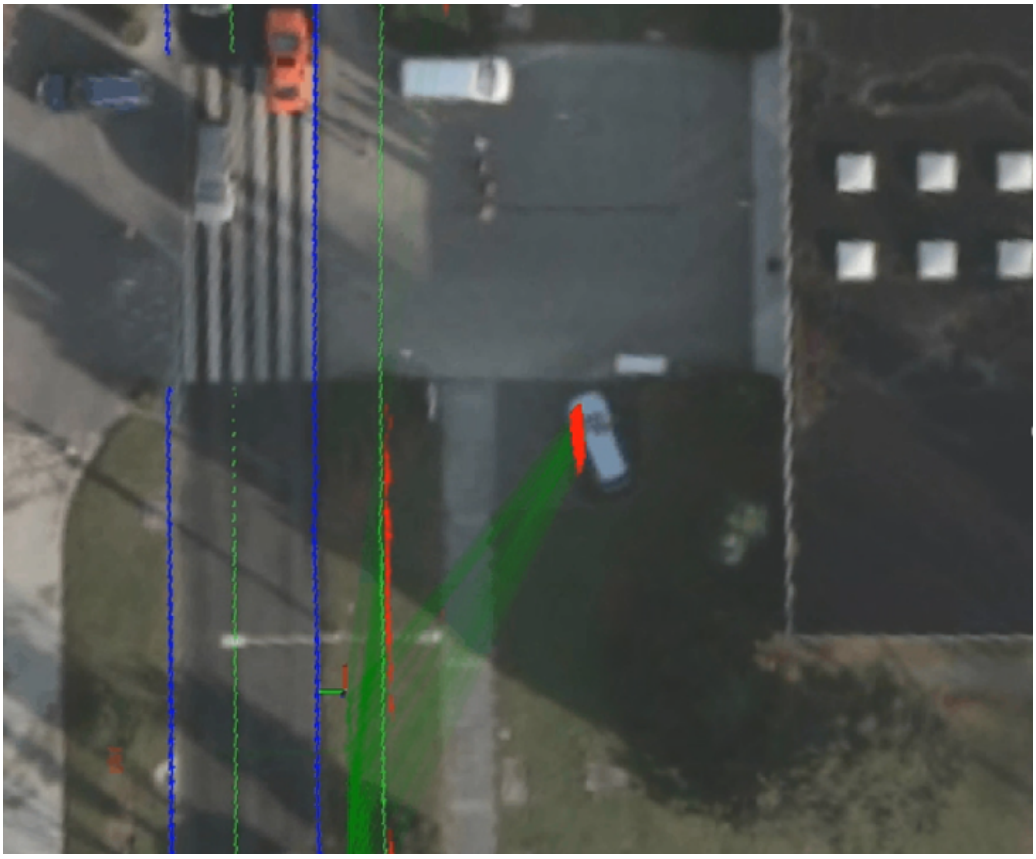

Figure 34: Result of the localization refinement

5.3 Road Graph

Crossing intersections is a particular challenge for automated driving. The self-driving vehicle must know the right-of-way rules that apply to the crossing to be able to follow the rules and to predict the behavior of other road users.

This section explains how the Road Graph derives this information from the digital map.

5.3.1 Right of way

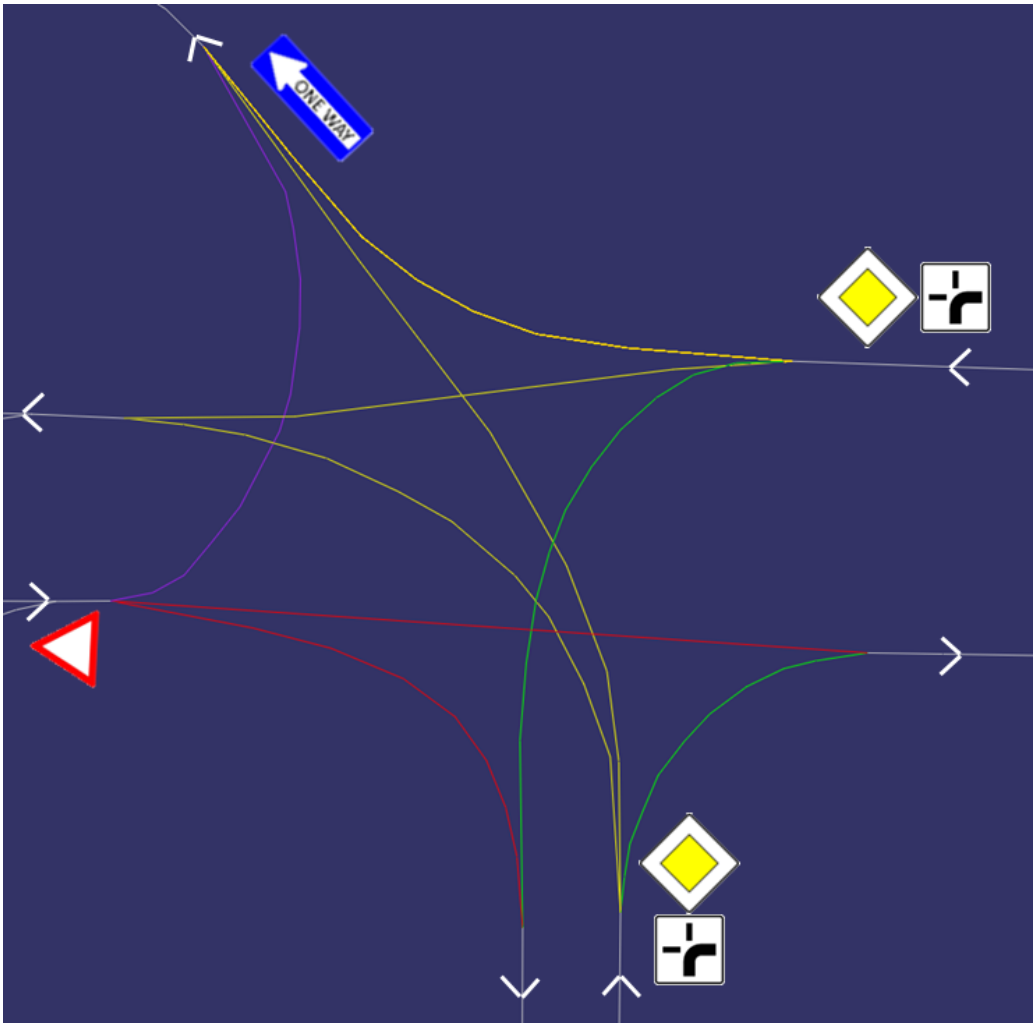

Figure 35: Classified lanes (center lines) of an intersection

The main step is to divide the lanes into five categories or priorities:

(P0) The lane represents a pedestrian crossing (zebra crossing).

(P1) The lane remains on the priority road.

(P2) The lane leaves the priority road

(P3) The lane enters the priority road.

(P4) The lane remains on the side street.

This information can either be stored directly in the digital map or can be obtained using the stored traffic signs. Even in a crossing with equal roads the classification can be made based on the right-before-left rule.

Figure 35 illustrates the classification for an intersection with inflected right of way. The lanes following the priority road (P1) are shown in green. The lanes leaving the priority road (P2) are drawn in yellow. The lanes entering the priority road (P3) are red while the lane which comes from the road with the yield sign and leads into the one-way road are colored purple (P4)

With this classification it can now be decided for each pair of arbitrary lanes which has the right of way:

- Lane A has right of way over lane B if the priority of lane A is higher than the priority of lane B.
- If the two lanes A and B have the same priority class than the right-before-left rule applies. A has priority over B if A is to the right of B or comes from the right of B.

5.3.2 Conflict areas

Conflict areas are the shared spaces of an intersection where the road users have to follow the right of way rules in order to avoid collisions or blockages.

The self-driving vehicle must know where exactly the conflict areas of an intersection are located to be able to stop at a suitable position if it has to give way or to know where it should not stop, so that other road users are not hindered or blocked.

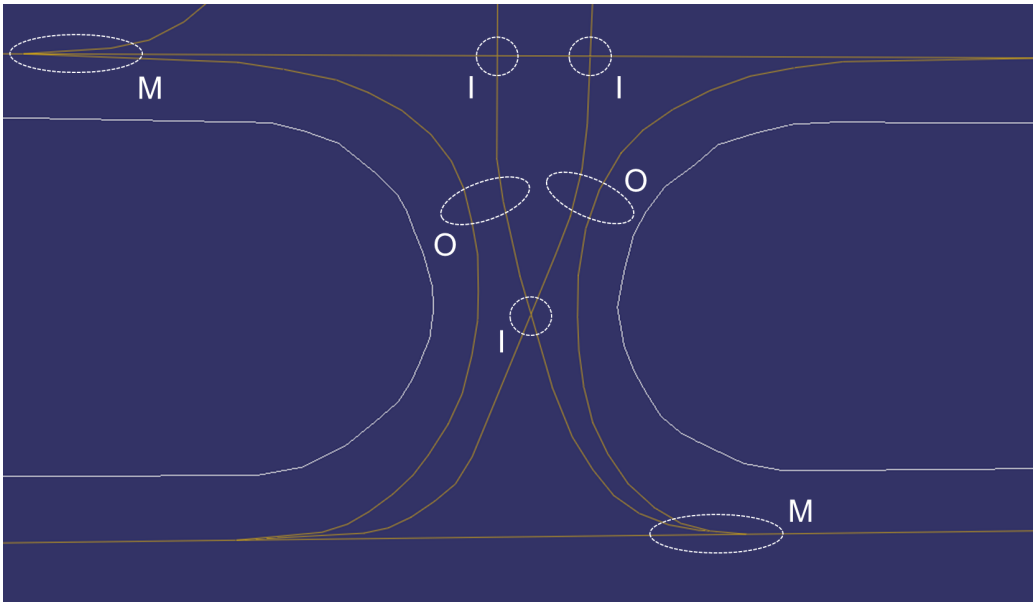

Figure 36: The different types of conflict areas: (I)ntersect, (M)erge, (O)verlap

The Road Graph provides the following information about conflict areas:

1. The type of conflict areas (see also figure 36):

   (I) Intersecting lanes including pedestrian crosswalks

   (M) Merging lanes

   (O) Overlapping lanes

2. The geometric shape of conflict areas: First and foremost, this calculation relies on lane boundaries like curbstones, road markings or traffic islands. If there natural boundaries are not available the algorithm will use the width of the lane or a width profile that is either stored in the map or estimated.

3. Entry points and exit points: The conflict areas are orthogonally projected onto the corresponding lanes. This gives the entry point and the exit point for the conflict on the lane.

Results of this calculation are shown in figure 37:

- The outlines of conflict areas between intersecting lanes are drawn in orange, conflict areas of merging lanes are blue and areas between overlapping lanes are drawn in purple.
- The projected conflict areas are visualized by the entry points (green squares) and exit points (red squares) on the lanes in driving direction.

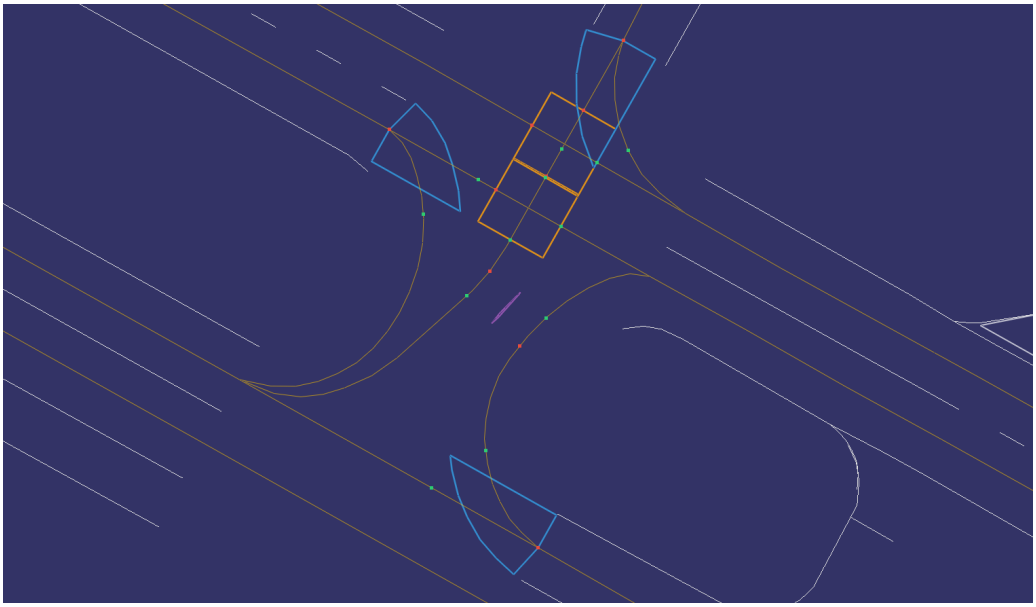

Figure 37: Calculated conflict areas with entry points and exit points

6 Conclusions

Deliverable 4.3 described the initial version of higher-level perception functions used for 360-degree environment perception.

The first design and implementation of environment perception modules were described: point cloud based perception module including ground and 3D object segmentation and parking spot detection; image based perception module including object semantic segmentation and signaling perception; enhanced perception module including road users perception by associating semantic labels to 3D objects, road users perception by using the super-sensor low-level data representation (STAR) and road users tracking and correction of their position.

As the project progresses, all these modules will be refined in order to increase the performance of the actual implementation. The Deliverable D4.6 in M42 will reflect and summarize the progress.

References

[1] A. Almagambetov, S. Velipasalar, and M. Casares. Robust and computationally lightweight autonomous tracking of vehicle taillights and signal detection by embedded smart cameras. *IEEE Transactions on Industrial Electronics*, 62(6):3732–3741, June 2015.

[2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 06 2016.

[5] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[7] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[9] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[10] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. *CoRR*, abs/1506.08959, 2015.

[11] Guangyu Zhong, Yi-Hsuan Tsai, Yi-Ting Chen, Xue Mei, D. Prokhorov, M. James, and Ming-Hsuan Yang. Learning to tell brake lights with convolutional features. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1558–1563, Nov 2016.