Supplementary Material

A Data Recording Platform

Below (Fig. 6), we schematically illustrate the data recording platform used for recording our dataset. It features a 32-beam Velodyne LiDAR mounted on top of the robot, a tilting LiDAR in the front, a Stereo RGB camera facing forward, and an IMU unit. The total height of the robot is approx. 1.80 m, offering camera viewpoints similar to those of a pedestrian. A photograph of the platform is redacted to reduce the risk of submission anonymity violation.

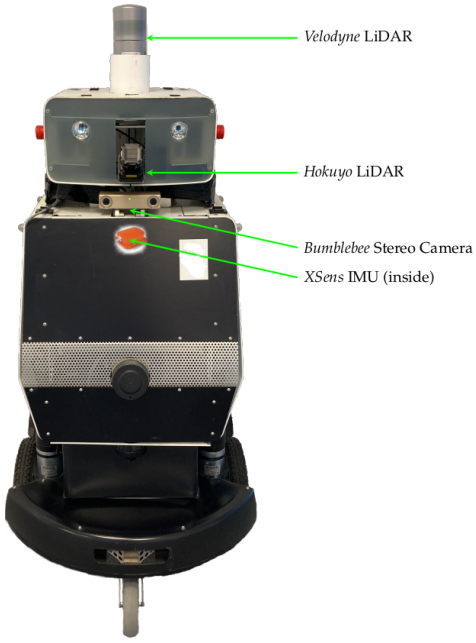

Figure 6: Our robotic data recording platform is equipped with LiDAR, RGB vision, and an IMU unit.

B Trajectory Projection

In the following, we illustrate the precise scheme of pixel-annotations based on a list of poses. These poses may be the robot ego-poses or the observed tracklets of other traffic participants. Fig. 7 illustrates the geometric construction of the tracklet annotations from a list of poses. Adjacent poses (p_i, p_{i+1}) are connected and form a 3D surface.

We illustrate a rendered ego-trajectory surface in Fig. 8. The trajectories of observed traffic participants are rendered using the same method. The object base width (i.e. footprint) changes depending on the type of traffic participant.

C Obstacles

We found that the stixel-based approach described in Sec. 3.1 leads to some false-negative Obstacle annotations. We, therefore, additionally leverage the bounding boxes obtained from our object tracker and mark all pixels within detected bounding boxes as obstacles.

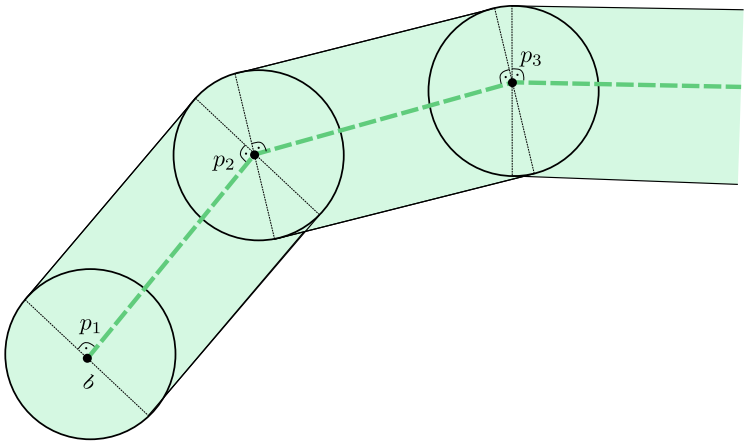

Figure 7: Geometric construction of the tracklet annotations from a list of poses. The scalar b denotes the object base width.

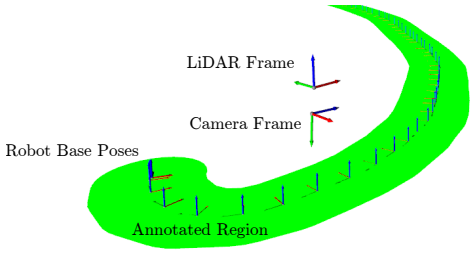

Figure 8: Visualization of the one specific LiDAR coordinate frame and camera coordinate frame, and multiple object base poses translated to the ground plane. We visualize the robot trajectory as a green surface.

D Mesh Rendering

To render the semantic mesh generated with our prediction aggregation scheme, we use the py-OpenGL framework. Below, we list an exemplary code snippet for initializing the shaders required to render a mesh into an image and setting parameters for the virtual camera capturing the mesh. We use a flat shader which does not consider any lighting effects, thus leading to monochrome surfaces for each surface types. When training a model on this data, the mesh colors can be mapped to a one-hot class encoding for each pixel.

```
window = glfw.create_window(self.w, self.h, "Projection", None, None)
glfw.make_context_current(window)

VERTEX_SHADER = """
                  #version 330
                  in vec3 position;
                  in vec3 color;
                  out vec3 newColor;

                  uniform mat4 projection;
                  uniform mat4 world_2_cam;

                  void main() {
                    //gl_Position = projection * vec4(position, 1.0f);
                    gl_Position = projection * world_2_cam * vec4(
    position, 1.0f);
                  newColor = color;
                  }
              """

```

```
FRAGMENT_SHADER = """
                #version 330
                 in vec3 newColor;
                 out vec3 outColor;
                void main() {
                   outColor = floor(newColor * 1.99);
                }
            """
shader = OpenGL.GL.shaders.compileProgram(OpenGL.GL.shaders.compileShader(VERTEX_SHADER, GL_VERTEX_SHADER),
                                          OpenGL.GL.shaders.compileShader(FRAGMENT_SHADER, GL_FRAGMENT_SHADER))
VBO = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, VBO)
glBufferData(GL_ARRAY_BUFFER, vertices.itemsize * len(vertices), vertices, GL_STATIC_DRAW)

# Create EBO
EBO = glGenBuffers(1)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.itemsize * len(indices), indices, GL_STATIC_DRAW)

# get the position from  shader
position = glGetAttribLocation(shader, 'position')
glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, vertices.itemsize * 6, ctypes.c_void_p(0))
glEnableVertexAttribArray(position)

# get the color from shader
color = 1
glBindAttribLocation(shader, color, 'color')
glVertexAttribPointer(color, 3, GL_FLOAT, GL_FALSE, vertices.itemsize * 6, ctypes.c_void_p(12))
glEnableVertexAttribArray(color)

glUseProgram(shader)
glClearColor(0.0, 0.0, 0.0, 1.0)
glEnable(GL_DEPTH_TEST)  # avoids rendering triangles behind other triangle

# specify virtual camera intrinsic parameters "camera_intrinsics"
proj_loc = glGetUniformLocation(shader, "projection")
glUniformMatrix4fv(proj_loc, 1, GL_FALSE, camera_intrinsics)

# set virtual camera pose according to actual robot pose "world_2_cam"
world_2_cam_loc = glGetUniformLocation(self.shader, "world_2_cam")
glUniformMatrix4fv(world_2_cam_loc, 1, GL_FALSE, world_2_cam.T)
```

Listing 1: OpenGL Mesh Shader initialization and setting of virtual camera parameters

E Dataset Details

In Tab. 3, we list the duration of each of the data collection runs in our *Freiburg Pedestrian Scenes* dataset.

E.1 Ground Truth BEV Semantic map

As part of our *Freiburg Pedestrian Scenes* dataset, we also annotated major sections of traversed regions from a BEV perspective. The rendered annotations are shown in Fig. 9. This map can serve as a reference to aggregated maps for qualitative and quantitative evaluations.

Table 3: *Freiburg Pedestrian Scenes* dataset collection runs

Collection Run Name	Duration [min]
Run01	36.3
Run02	27.1
Run03	6.1
Run04	192.8
Run05	70.3
Run06	23.5
Run07	5.3
Run08	3.1
Run09	2.3
Run10	1.4
Run11	6.8
Run12	6.6
Run13	17.5
Run14	57.9
Run15	20.7
Run16	16.5
Run17	114.9
Run18	119.0
Run19	37.1
Run20	14.9
Run21	41.7
Run22	6.6
Run23	17.5

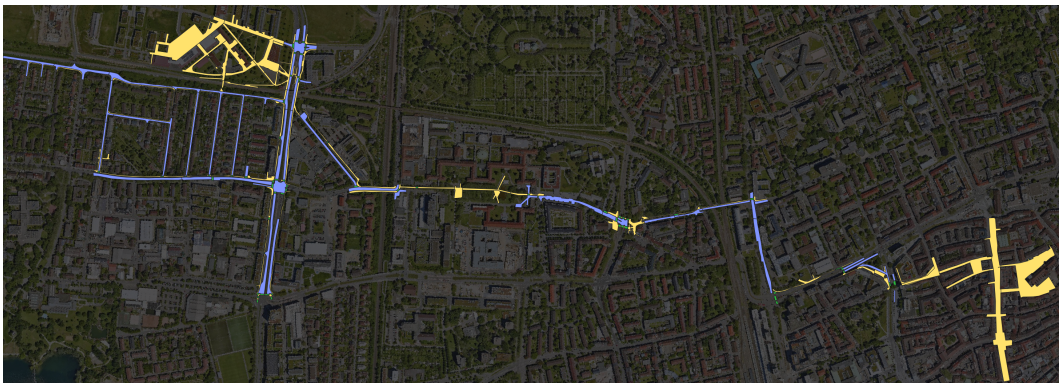

Figure 9: Visualization of our BEV map annotations superimposed on an aligned RGB satellite image layer. Best viewed zoomed in. Color-codes for the semantic classes are: *Road*, *Pedestrian*, *Crossing*.

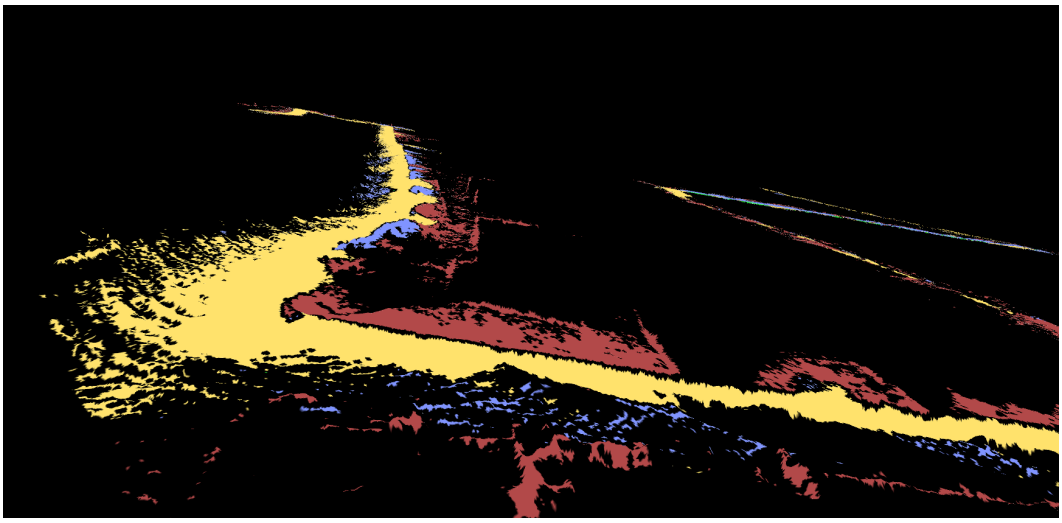

Figure 10: Aggregated 3D ground map, illustrating a non-planar surface structure in the pedestrian area to the left of the image and in the obstacles present on both sides of the pedestrian pathway. Color code: *Road*, *Pedestrian*, *Crossing*, *Obstacle*.

E.2 On the Fraction of Annotated Pixels

Using only the ego-trajectory, we were able to label 53% of all image pixels with the classes *Pedestrian* or *Obstacle*. Using additional tracklets of other traffic participants, we were able to label 70% of all pixels with the classes *Road*, *Crossing*, *Pedestrian*, and *Obstacle*. Finally, using our mesh aggregation scheme we were able to further increase the number of labeled pixels. Concretely, With our aggregated map, we were able to label 87% of all pixels.

E.3 Visualization of 3D surface map

For illustrative purposes, we show an exemplary aggregated map in Fig. 10, including static obstacles close to the ground surface (red color). It features multiple regions of non-planar ground surfaces, showing our ability to model non-flat terrains with our approach.

F Training Details

We train our models using the standard per-pixel weighted cross-entropy loss formulation:

$$\mathcal{L} = -\sum_{k} \alpha_k y \log \hat{y}_k, \tag{4}$$

where α_i denotes the loss weight for class k, $\hat{y}_i$ denotes the model class prediction, and y_i denotes the ground-truth class. For our experiments we select the following loss class weights: $\alpha_{\text{Obstacle}} = 0.2$, $\alpha_{\text{Road}} = 1$, $\alpha_{\text{Pedestrian}} = 1$, $\alpha_{\text{Crossing}} = 5$, and $\alpha_{\text{Unknown}} = 0$.

We use the Adam optimizer with an initial learning rate of $\alpha = 0.001$, and parameters $\beta_0 = 0.9$ and $\beta_1 = 0.999$. The learning rate is adjusted according to an exponential decay with a decay rate of 0.9.

G Evaluation of Aggregated BEV Maps

In order to quantify the validity of the aggregated maps, we evaluate the IoU score, precision, and recall on a per-map basis. Tab. 4 lists the metrics for five regions while Fig. 11 visualizes the aggregated maps and their corresponding aligned ground-truth annotations.

Table 4: BEV map performance evaluation for five maps. We denote all metrics in %.

Map Name	Metric	Road	Pedestrian	Crossing
Map 0	IoU	17.3	22.4	25.4
	Precision	58.0	57.8	34.6
	Recall	18.2	23.7	33.4
Map 1	IoU	0.0	55.0	0.0
	Precision	0.0	57.6	0.0
	Recall	0.0	55.0	0.0
Map 2	IoU	50.2	53.9	2.6
	Precision	57.9	41.3	100.0
	Recall	50.2	54.9	2.63
Map 3	IoU	4.0	62.8	21.2
	Precision	44.0	69.2	70.0
	Recall	40.2	62.9	22.0
Map 4	IoU	24.6	28.2	5.9
	Precision	74.0	64.3	8.4
	Recall	25.1	30.5	9.2

We observe that for most regions, the predicted semantic ground class overlaps with the actual ground class. This hold true even for very complex environments such as the intersection depicted in map 0 and map 3. Furthermore, in most scenarios, the clear border between class *Predestrian* and *Road* is prominent, indicating a clear distinction between these two classes. It is crucially important for an autonomously operating robot to have a robust distinction between sidewalks and roads in order to navigate safely. We also observe misclassifications of surfaces, prominent in map 4. Note that the incompleteness of our aggregated maps stems from the fact that not all ground surfaces visible in the annotated map were visible in the onboard robot camera during the data collection runs.

The quantitative evaluation underlines these findings. Please note that the IoU and recall values are of limited significance for evaluating the aggregated maps due to the incompleteness of these maps. The precision metric, in contrast, is more meaningful in this context. We find that for most maps, decent precision values (values generally above 50 %) are obtained, indicating that when a surface patch is observed in the robot camera, the prediction quality for this patch is high.

H Path Planning Experiments

In addition to the quantitative IoU evaluation of the aggregated BEV maps, we conduct additional path planning experiments. One intended use-case of our map aggregation scheme is the ability for an autonomous robots to perform high-level planning on the aggregated semantic maps. We, therefore, convert the semantic class map into a costmap where each class is associated with a traversability cost. Since our robot is supposed to operate and navigate alongside pedestrians, we associate high cost with the classes *Road* and *Unknown*, while we associate low cost with the classes *Pedestrian* and *Crossing*. Finally, we smooth the produced costmap with a Gaussian filter to encourage the search algorithm to follow pathways that are centered within a given corridor of low-cost traversability such as sidewalks. We subsequently perform an A* search on the costmap to find optimal routes between a start position and a goal position. Fig. 12 illustrates three exemplary planning tasks in complex urban areas.

The results show that it is possible to use the semantic map as a data source for a planning algorithm. The planned path follows legal pathways through complex surroundings such as street crossings and sidewalks. As long as the SLAM solution to a given data collection run is accurate, large-scale maps such as shown in 12, rightmost map, are possible to generate. Fig. 12, leftmost map, shows an interesting failure case where the map does not contain a street crossing that would shorten the overall route length from start position to goal position (indicated with a green circle). In this case, the map contains a longer but also safe route across the street closer to the building where the street surface is correctly classified as a pedestrian area (it turns into a pedestrian area after the crossing).

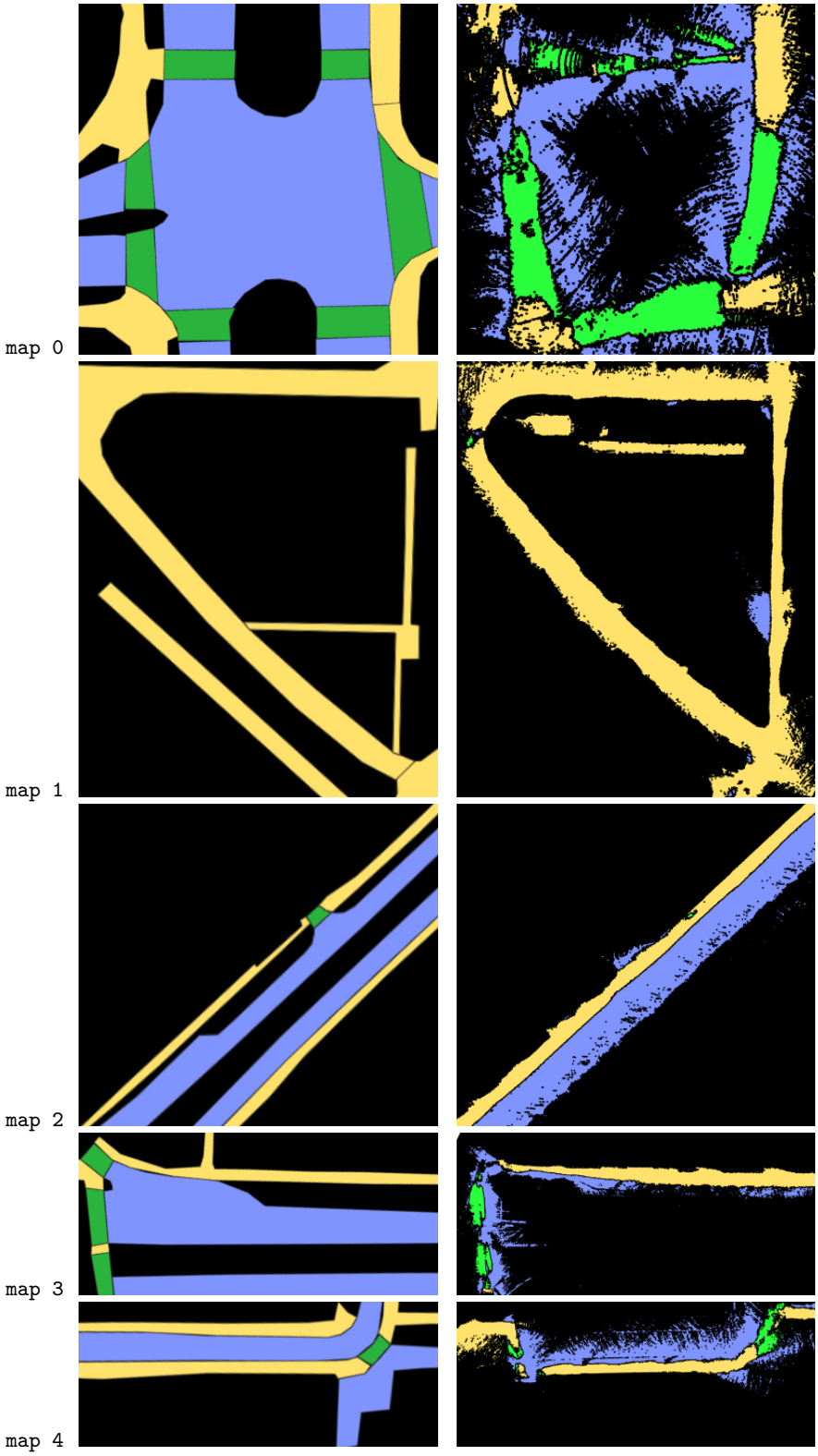

Figure 11: Visualization of ground-truth map annotations obtained from manual labeling efforts (left column) and corresponding crop of the aligned aggregated semantic map obtained with our approach (right column).

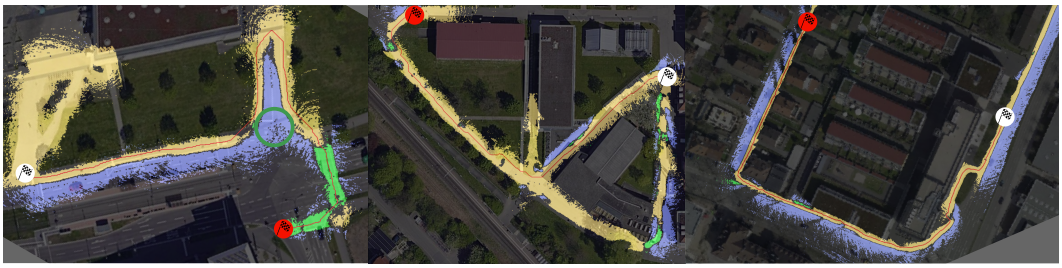

Figure 12: Path planning experiments on three exemplary complex urban areas. We superimpose the color-coded semantic map onto an aligned satellite image. The start and goal positions are indicated with red and white flags, respectively. The planned route according to the semantic map is indicated as a red line. Best viewed zoomed in. Color code: *Road*, *Pedestrian*, *Crossing*.

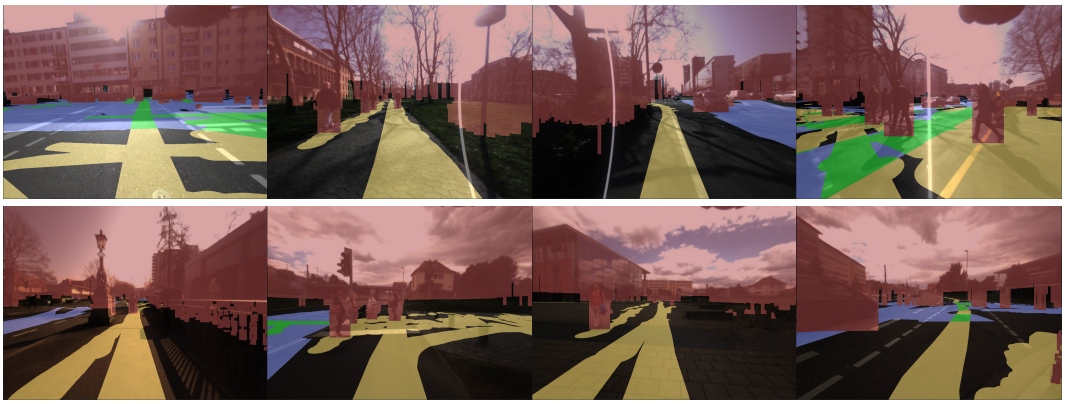

Figure 13: Exemplary visualizations of annotation masks obtained with our tracklet-based annotation scheme. Color code: *Road*, *Pedestrian*, *Crossing*, *Obstacle*.

I Exemplary visualization of Tracklet Annotations

In Fig. 13, we illustrate exemplary semantic annotations obtained from the projected tracklets in each scene (dataset $\mathcal{D}_0$).

J Exemplary visualization of Semantic Map Projections

In Fig. 14, we illustrate exemplary map projections obtained from the aggregated surface maps in each scene (dataset $\mathcal{D}_1$). Note how the number of labeled pixels is increased compared to the annotations in Fig. 13.

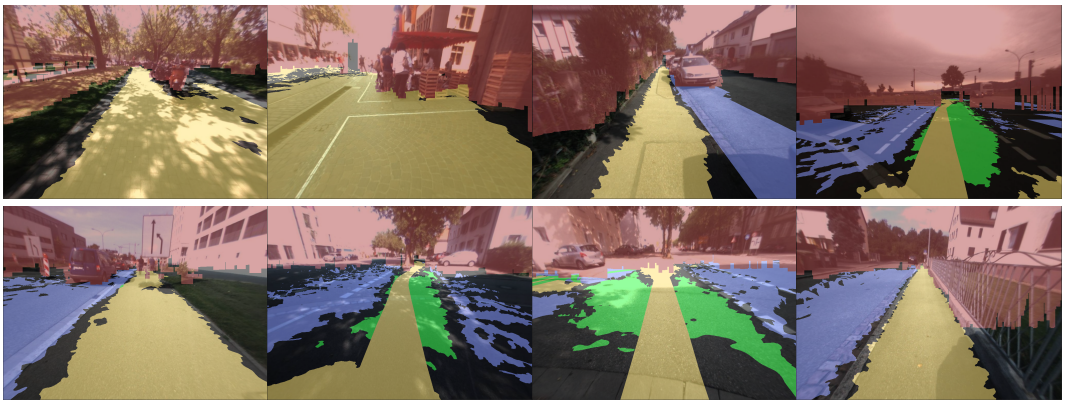

Figure 14: Exemplary visualizations of annotation masks obtained with our map reprojection annotation scheme. Note that the yellow-colored ego-trajectory is superimposed on the projected map for visualization purposes and is not used to provide the annotations for the model. Color code: *Road*, *Pedestrian*, *Crossing*, *Obstacle*.