# Quo Vadis: Supplementary Material

The supplementary material complements our work with additional information on the bird's-eye-view reconstruction in Appendix A. Furthermore, we provide implementation and training details on different components and networks used in our method in Appendix B. Finally, we present visual examples as visualizations and videos in Appendix C.

## A Information on Bird's-Eye View Reconstruction

The paper presents our approach to constructing a bird's-eye-view (BEV) representation for a static tracking sequence. Here, we extend the explanation by adding a description of moving cameras and how we linearize the homography transformation for farther objects to avoid enormous distances and unrealistic velocities.

### A.1 Linearization of Homography

To get a bird's-eye-view (BEV) representation of the tracking scene, we estimate the homography $H$ between the image and the ground plane. Hence, the homogenous pixel positions transform accordingly to Equation (2) as follows:

$$s \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = H \cdot \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}. \tag{2}$$

This approach assumes that objects move on a perfect plane and object's position in the image is represented as the bottom mid-point of the object's bounding box. Depending on the perspective transformation of the camera, we find that minor changes in pixel value lead to enormous distances in BEV. Given a homography matrix:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}, \tag{3}$$

the BEV coordinate $y$ is computed as:

$$y = \frac{h_{21} \cdot p_x + h_{22} \cdot p_y + h_{23}}{h_{31} \cdot p_x + h_{32} \cdot p_y + h_{33}}. \tag{4}$$

As the denominator in Equation (4) is approaching zero, the $y$-coordinate grows hyperbolically. This behavior is undesired for trajectory prediction because these large jumps in the object's position result in unrealistic velocities for the object. Therefore, we define a threshold for which we linearly extrapolate the transformation such that the transformed distance between two neighboring pixel points is maximal $0.2m$ as shown as a red line in Figure 6b. This formulates the condition as

$$\| \frac{h_{21} \cdot p_x + h_{22} \cdot p_y + h_{23}}{h_{31} \cdot p_x + h_{32} \cdot p_y + h_{33}} - \frac{h_{21} \cdot p_x + h_{22} \cdot (p_y + 1) + h_{23}}{h_{31} \cdot p_x + h_{32} \cdot (p_y + 1) + h_{33}} \| \leq 0.2m \tag{5}$$

We call the $p_y$ value for which the inequality Equation (5) is equal, the linearization threshold $p_y^T$. The pixel point where the denominator of Equation (4) becomes 0 is called the horizon because no point on the plane is projected on a lower point in the image.

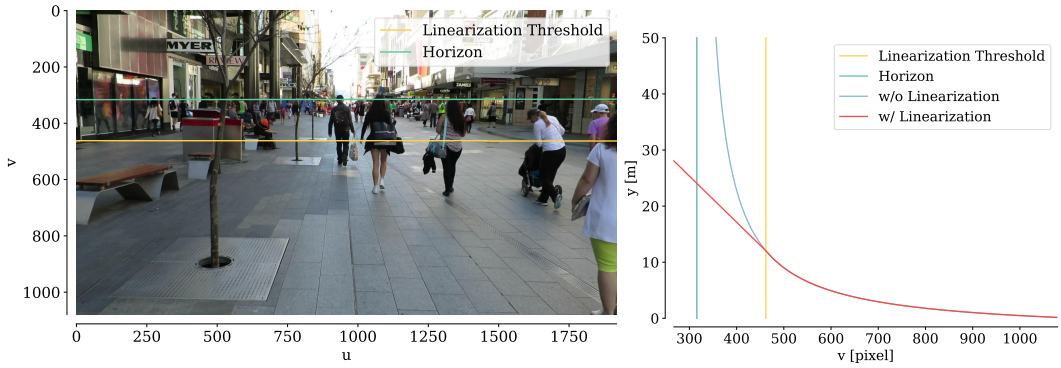
To prevent this hyperbolic growth for image points closer to the horizon, we linerarize Equation (4) around $p_y^T$ and apply the linear transformation for all $p_y \leq p_y^T$ as shown in Figure 6b. Thus, we stabilize the distance between two points to prevent very unrealistic velocities, which would make the transformed values pointless. To transform from pixel space to BEV and back, we also inverse the linearized transformation to get a one-to-one mapping.

## B Implementation Details

In this section, we provide additional information on the implementation of our method and its key components. The source code is available at `https://github.com/dendorferpatrick/QuoVadis`.

### B.1 Synthetic Training Data

For training the trajectory predictor (Appendix B.2) and the depth estimator (Appendix B.3) we use the MOTSynth dataset [19] which provides ground truth 3D positions of objects and image depth information. We use the split suggested by Fabbri *et al.* [19] with 576 sequences in the training set and 192 in the validation set.




(a) Horizon and Linear Threshold in Scene.

(b) Linearization of BEV coordinates.

Figure 6: Demonstration of horizon and linearization threshold for sequence image. Linearization of homography transformation is necessary to prevent enormous distances in the transformed coordinates and unrealistic velocities.

### B.2 Trajectory Predictor

For our trajectory model, we use the implementation of MG-GAN [13]. For studying the effect of modeling social interactions on tracking, we implement a social max-pooling module following S-GAN [23].

**Model.** The trajectory prediction model generates a set of $K$ future trajectories $\{\hat{Y}_i^k\}_{k=1,\dots,K}$ with $t \in [t_{obs}+1, t_{pred}]$ given the input trajectory $X_i$ with $t \in [t_1, t_{obs}]$ for each pedestrian $i$. We use $t_{obs} = 8$ observation steps and $t_{pred} = 12$ prediction steps as default for training the model. However, input and output length can vary depending on the observed tracks during inference for the tracking model.

For the multimodal MG-GAN implementation, we use $n_G = 3$ generators from which we sample one prediction from each generator during inference.

**Training.** We construct trajectories of the MOTSynth data with $8$ observation and $12$ prediction steps, each step being $0.4s$. The entire model is trained in a GAN framework using a prediction model and a discriminator network. We train the network on the entire train dataset over $200$ epochs, with a learning rate $\lambda = 10^{-3}$, and using a batch size scheduler [58].

### B.3 Depth Estimator

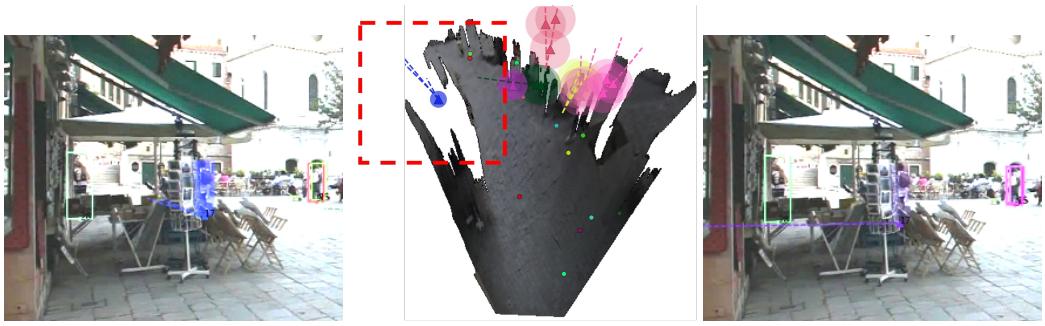
Depth estimation is a crucial part of the BEV estimation in our model. Therefore, we use a vision transformer-based [18] network [6] for monocular dense depth estimation.

**Model.** The transformer-based model regresses the depth prediction as a linear combination of depth range bins of adaptive size. The network encoder-decoder extracts visual features from the image, which are passed to the mVit block. mVit is a lightweight vision transformer based on [18]. The model applies an MLP on top of the mVit's output, predicting the size of the bins for the depth range. The encoder computes the weights of the bins by passing the features through multiple convolutional layers with a final softmax non-linear activation function.

**Training.** The network trains on the synthetic MOTsynth dataset to leverage a large number of tracking scenes of varying perspectives, weather, and light conditions. To better generalize to real-world data, we augment the scenes with ground reflections by mirroring surfaces in the image. This results in better performance, especially for the indoor MOT sequences, with ground reflections. We find the model trained on synthetic data performs well on real data even without fine-tuning.

To increase the default model depth map resolution from $640 \times 480$ to $960 \times 576$ we grow the transformer positional embedding vector size from 500 to 1000.

We trained the model using AdamW optimizer [40] with weight decay $10^{-2}$. Following [57], the maximum learning rate $\lambda_{\text{max}}$ was set to $3.5 \times 10^{-4}$ with linear warm-up from $\frac{1}{4}\lambda_{\text{max}} \rightarrow \lambda_{\text{max}}$ for the first 30% of the iterations followed by cosine annealing to $\frac{3}{4}\lambda_{\text{max}}$. We trained the model for 20 epochs with an image resolution of $960 \times 576$ on a training split of MOTSynth dataset with the batch size of 8 on 4 $RTX8000$ for one week. Then, we trained the model for 30 epochs with an image resolution of $960 \times 576$ on the full MOTSynth dataset with a batch size of 8 on 4 $RTX8000$ for ten days.



(a) Projected MG-GAN prediction (b) MG-GAN predictions in BEV (c) Prediction of Kalman Filter in Pixel Space

Figure 7: Demonstration of prediction for MG-GAN in BEV and Kalman filter in pixel space.

### B.4 Image Segmentation

We run a pre-trained Detectron2 [73] segmentation network to get the segmentation masks of the tracking scene images. Explicitly, we use the pre-trained COCO Panoptic Segmentation model with Panoptic FPN [73]. The model outputs semantic labels for 134 COCO classes and panoptic object ids, which are irrelevant to our task.

In our model, we use segmentation labels to mask ground pixels of the scene. Therefore, we combine the following COCO classes to our ground class: `pavement`, `road`, `platform`, `floor`, `floor − wood`, `grass`, `sand`, `dirt`.

#### B.4.1 Optical Flow

We estimate the optical flow using the implementation [12] of an attention-based GMA model[28]. We use the standard MMFlow configuration for the GMA pre-trained model on a mix of the datasets [28, 17, 45, 9, 46, 30]. While the model was pre-trained on images with size (768, 368), we resized the MOTChallenge images to (960, 540) at test time.

## C Visual and Qualitative Results

This section shows a visual example of the difference between a BEV and a 2D image space prediction. Furthermore, we want to point to the additional scene videos also provided in the Supplementary material.

**2D versus 3D.** In Figure 7 we show the trajectory prediction of our MG-GAN projected into the image (Figure 7a), the prediction in BEV (Figure 7b), and trajectory prediction in pixel space using a Kalman Filter (Figure 7c). We find the problem of the model reasoning in pixel space and cannot account for the effect of the camera perspective. As a consequence, this results in unrealistic motion in image space.

In contrast, we see in Figure 7a that our model predicting in BEV understands the spatial structure of the scene and is, therefore, able to predict the correct trajectory for the object and resolves the long-term occlusion.

**Example Videos.** In addition to the written supplementary material, we provide brief video clips of different MOT17 validation and test sequences with ByteTrack and Center Tracks.

In Figure 8, we give a brief description of the format of the provided video sequences. We show our predictor output on the left side, the baseline tracker output in the middle, and the online BEV prediction and reasoning on the right side. For our model, we show the tracker detection and predictions in BEV, including their projection in the image.

## D Information on computation of ID Recall

In the introduction, we present the performance of the baseline trackers compared to our trajectory forecasting model on how well they can re-associate tracks after occlusion from occlusions. We measure the performance as a fraction of ground-truth tracks detected and assigned correctly before and after occlusion.

As the first step, we need to identify the ground-truth occlusion regions for every sequence. We use the visibility scores of objects and threshold those into a binary visibility flag, stating whether an object is visible in a given frame. Then, we apply a minimum rolling window on the visibility flags to get connected components and to smooth the deviations of the visibility flag values. The rolling

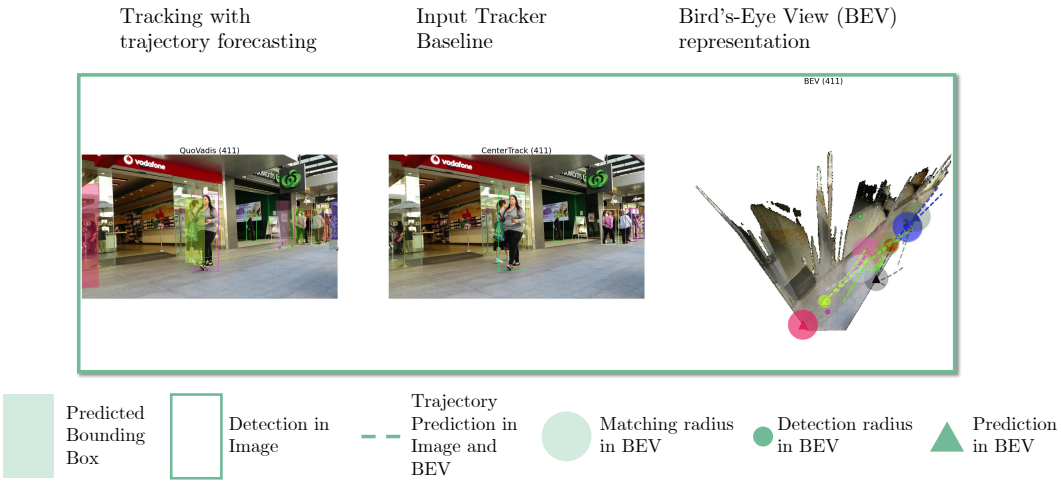



Figure 8: Description of supplementary sequence videos.

window also includes visible frames before and after the occlusion, where the actual IDSW may happen. We compute the frame ids where an occlusion starts and ends by extracting all connected components, with the visibility flag being $0$. We only consider components where the object is visible before and after the occlusion.

Finally, we check for every tracker if the tracker detected an object at the start and the end of the occlusion component and if the track ids between the beginning and start match. We use $\tau_{\text{vis}} = 0.1$ as visibility threshold and a temporal window size $\texttt{ws} = 5$ as hyperparameters.