r_y and r_x are the rotation angles along the respective axes in radians and w and h are the image width and height respectively.

4.2 Learning object motion

Eq. 1 can propagate frame inconsistency losses to z, R and t at every pixel. However without further regularization, the latter trio remain greatly underdetermined. While continuity of z, R and t is a powerful regularizer, we found that further regularization helps significantly. In particular, we impose constancy of R throughout the image, and allow t to deviate from a constant value only at pixels that are designated as "possibly mobile". This mask can be obtained from a pretrained segmentation model. Unlike in prior work [7], instance segmentation and tracking are not required, as we need a single "possibly mobile" mask. In fact, we show that a union of bounding boxes is sufficient (see Fig. 2). In addition, an $L1$ smoothing operator is applied on t.

4.3 Occlusion-aware consistency

When the camera moves relatively to a scene, and / or objects move, points in the scene that were visible in one frame may become occluded in another, and vice versa. In pixels that correspond to these points, cross-frame consistency cannot be enforced by a loss. Given a depth map and a motion field in one frame, one could actually detect where occlusion is about to occur, and exclude the occluded areas from the consistency loss.

While detecting occluded pixels is doable, it requires some sort of reasoning about the connectivity of the surface represented by the depth map, and z-buffering. Keeping the mechanism differentiable and efficient, to be suitable for a training loop, may pose a challenge.

We therefore take a different approach, as illustrated in Fig. 3. For each pixel (i, j) in the source frame, the predicted depth z_{ij} and the camera intrinsic matrix are used to obtain the respective point in space, (x_{ij}, y_{ij}, z_{ij}). The point is moved in space according to the predicted motion field. In particular, the depth changes to z'. The new spatial location is reprojected back onto the camera frame, and falls at some generally-different location (i', j') on the target frame. i' and j' are generally non-integer. Therefore obtaining the depth on the target frame at (i', j'), $z^t_{i',j'}$, requires interpolation.

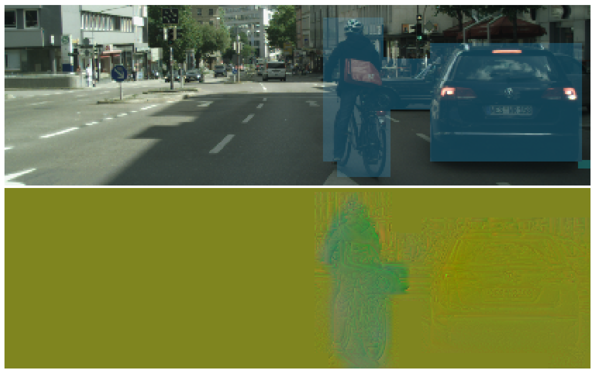
Figure 2: Use of a "possibly mobile" mask to regularize the translation field. An object detection network identifies all instances of objects that are capable of motion, such as pedestrians, cyclists and cars. The union of the bounding boxes comprises the "possibly mobile" mask, within which the translation field is allowed to vary. The top picture, from Cityscapes, illustrates the mask, and the bottom one is the translation field predicted by the network (x, y, z coded as RGB). The **golden background** corresponds to motion in the negative z direction, as the entire scene is moving towards the camera. The **greenish silhouette** is the cyclist moving slightly to the left and slightly towards the camera. Note that the network carves the silhouette out of rough mask.

Occlusions happen at (i', j') where z' becomes multivalued. At such points, color and depth consistency should be applied only to the visible branch of z', that is, the branch where z' is smaller. If the source and target frames are nearly consistent, the visible branch will be close to target depth at (i', j'), $z^t_{i',j'}$. The way we propose to detect that is to include in the losses only points (i', j') where $z'_{i',j'} \le z^t_{i',j'}$. In other words, only if a transformed pixel on the source frame lands in front of the depth map in the target frame, do we include that pixel in the loss. This scheme is not symmetrical with respect to interchanging the source and target frames, which is why we always apply it in a symmetrized way: We transform the source onto the tagret, calculate the losses, and then switch the roles of source and target. Fig. 3 illustrates the method. This way of applying losses can be invoked for many types of loss, and we shall refer to it in this paper as "occlusion-aware" loss or penalty.

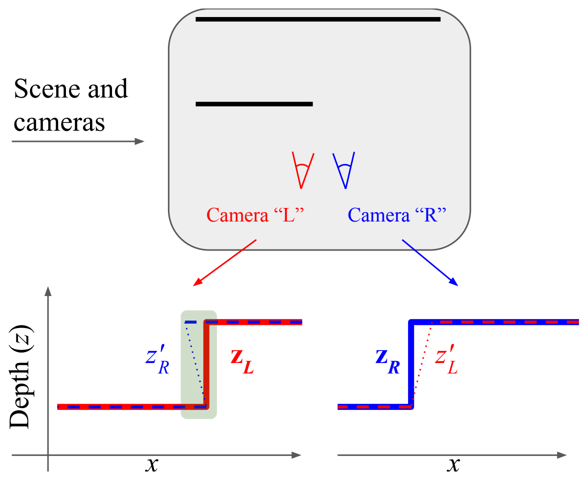

Figure 3: An illustration of our proposed method for handling occlusions. At the top we show a two-dimensional "scene", consisting of two straight surfaces, one partially occluding the other. Two cameras, left ("L") and right ("R"), are observing the scene. Our method is *monocular*, so these represent two locations of the same camera that moved, and "left" and "right" are used for convenience. At the bottom, the depth map observed by each camera is illustrated in as a solid line on the respective side ($\mathbf{z}_L$ and $\mathbf{z}_R$). A dashed line shows the depth map obtained from warping one view onto the other (z'_R and z'_L). The warped depth map can become a multivalued function, which indicates occlusions (see green shaded rectangle). To handle that, we apply photometric and geometric losses only at pixels where $z'_R \leq \mathbf{z}_R$ and $z'_L \leq \mathbf{z}_L$. When the depth maps and motion estimation are correct, the loss in this scheme would indeed evaluate to zero.

4.4 Networks, losses and regularizations

Networks We rely on two convolutional networks, one predicting depth from a single image, and the other predicting egomotion, object motion field relative to the scene, and camera intrinsics from two images. The depth prediction network is a UNet architecture with a ResNet 18 base and a softplus activation ($z = \log(1 + e^{\ell})$) to convert the logits (ℓ) to depth (z).

The motion estimation network is a UNet architecture inspired by FlowNet [9]. A stack of convolutions with stride 2 (the"encoder"), with average pooling in the last one, forms a bottleneck of 1024 channels with a 1x1 spatial resolution. Two 1x1 convolutions with 3 output channels each predict the global rotation angles (r_0) and the global translation vector (t_0). The latter two represent the movement of the entire scene with respect to the camera, due to camera motion. Each of the intrinsic parameters is predicted by a 1x1 convolution stemming from the bottleneck, with softplus activations for the focal lengths and distortions. The next layers progressively refine (that is, increase the spatial resolution of) the translation, from a single vector to a residual translation vector field $\delta t(x, y)$, by a factor of 2 in the height and width dimension each time.

It is here where we utilize the foreground mask $m(x, y)$ described in Sec. 4.2: The translation field is expressed as the sum of the global translation vector plus the masked residual translation:

$$t(x, y) = t_0 + m(x, y)\delta t(x, y). \quad (4)$$

$m(x, y)$ equals one at pixels that could belong to mobile objects and zero otherwise.

Losses Given a pair of frames, we apply an occlusion-aware L1 penalty for each of the RGB color channels and the depths. For the motion fields, we demand cycle consistency: The rotation and translation at pixel (i, j) of the source frame must form the *opposite* transform of the ones at (i', j') of the target frame, and vice versa. Occlusion awareness is invoked here too.

Structural similarity (SSIM) is a crucial component of the training loss, and occlusion awareness as defined above is harder to enforce here, because SSIM involves the neighborhood of each pixel. It is possible that $z' \leq z^t$ holds for only part of the pixels in the neighborhood. The solution we found here is to weigh the structural similarity loss by a function that falls off where the depth discrepancy between the frames is large compared to the RMS of depth discrepancy.

Randomized layer normalization Initially our depth prediction network had batch normalization. However we repeatedly observed it leading to anomalous behavior:

- Eval accuracy was consistently better when running inference at the "training mode" of batch normalization. That is, instead of long-term averaged means and variances, the ones obtained from the image itself during inference were used[1], rendering batch normalization more similar to layer normalization [3].

- As we increased the batch size at training, the eval accuracy was consistently worse and worse, no matter how we scaled the learning rate.

These two observations led us to try replacing batch normalization by layer normalization, but the eval quality degraded. Our next postulate was that while batch normalization is actually acting like layer normalization, the other other items in the batch serve as a source of noise on top of that. To test this theory we replaced batch normalization by layer normalization with Gaussian noise applied on the layer means and variances.

Indeed all eval metrics exhibited significant improvements compared to batch normalization. Moreover, now the eval metrics started to slightly improve when the batch size at training increased (accompanied with a linear increase of the learning rate [15]), rather than significantly degrading. The best results were obtained with multiplicative noise. While it has been observed in the past that noise can act as a regularizer, for example with dropout [34] or when applied to the gradients [26], we are not aware of prior work where it is applied with layer normalization.

5 Experiments

In this section, we evaluate our method on depth prediction, odometry estimation, and the recovery of camera intrinsics across a range of diverse datasets, which will be described next.

5.1 Datasets

KITTI The KITTI dataset is collected in urban environments and is the main benchmark for depth and egomotion estimation. It is accompanied with a LIDAR sensor which is used for

Cityscapes The Cityscapes dataset is a more recent urban driving dataset, which we use for both training and evaluation. It is a more challenging dataset with many dynamic scenes. With a few exceptions [30, 7] it has not been used for depth estimation evaluation. We use depth from the disparity data for evaluation on a standard evaluation set of 1250 [30, 7].

EuRoC Micro Aerial Vehicle Dataset The EuRoC Micro Aerial Vehicle (MAV) Dataset [5] is a very challenging dataset collected by an aerial vehicle indoors. While the data contains a comprehensive suite of sensor measurements, including stereo pairs, IMU, accurate Leica laser tracker ground truth, Vicon scene 3d scans, and camera calibration, we only used the monocular videos for training. Since the camera has significant lens distortion, this is an opportunity to test our method for learning lens distortions (see later sections).

YouTube8M videos To demonstrate that depth can be learned on videos in the wild from unknown cameras, we collected videos from the YouTube8M dataset [1]. From the 3079 videos in YouTube8M that have the label "quadcopter", human raters selected videos that contain significant amount of footage from a quadcopter. Naturally, the videos were taken with different unknown cameras, with varying fields of view and varying degrees of lens distortion. The YouTube8M IDs will be made public.

5.2 Depth

KITTI Table 1 summarizes the evaluation results on the KITTI Eigen partition of a model trained on KITTI. The metrics are the ones defined in Zhou et al. [50]. Only the best methods and the first three metrics are displayed in Table 1, the rest are given in the Appendix. As seen in the results, we obtain best state-of-the-art result. More importantly, we observed that learned intrinsics, rather than given ones consistently help performance, as seen too in later experiments.

Cityscapes Table 2 summarizes the evaluation metrics of models trained and tested on Cityscapes. We follow the established protocol by previous work, using the disparity for evaluation [7, 30]. Since this is a very challenging benchmark with many dynamic objects, very few approaches have evaluated on it. As seen in Table 2, our approach outperforms previous ones and benefits from learned intrinsics.

Cityscapes + KITTI Being able to learn depth without

[1]Even at batch size 1, there would still be means and variances over the spatial dimensions.

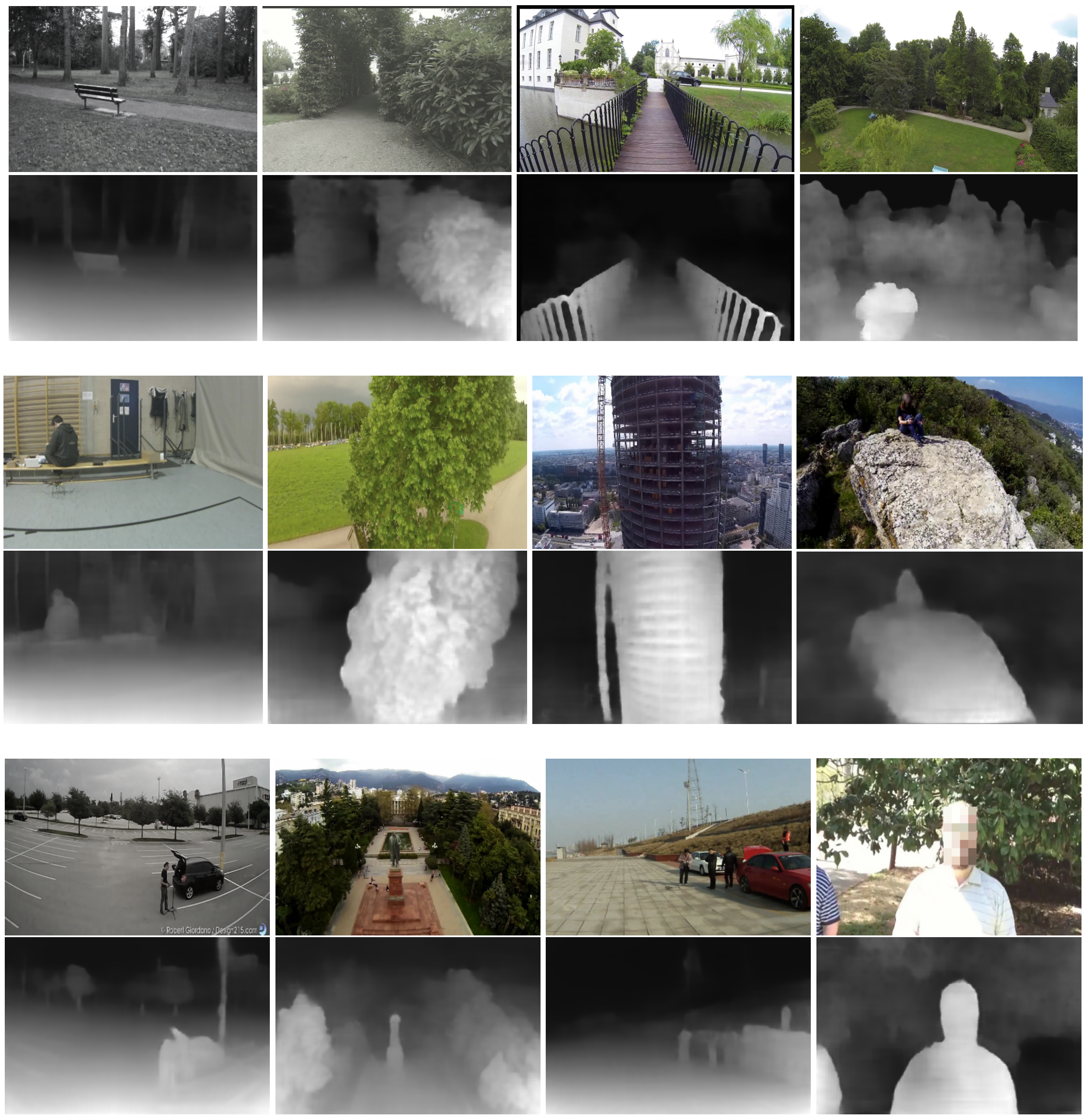

Figure 4: Images and learned disparity maps from the set collected from YouTube8M.