F Additional Experiments

F.1 Naive Combination: DROID + ZoeDepth

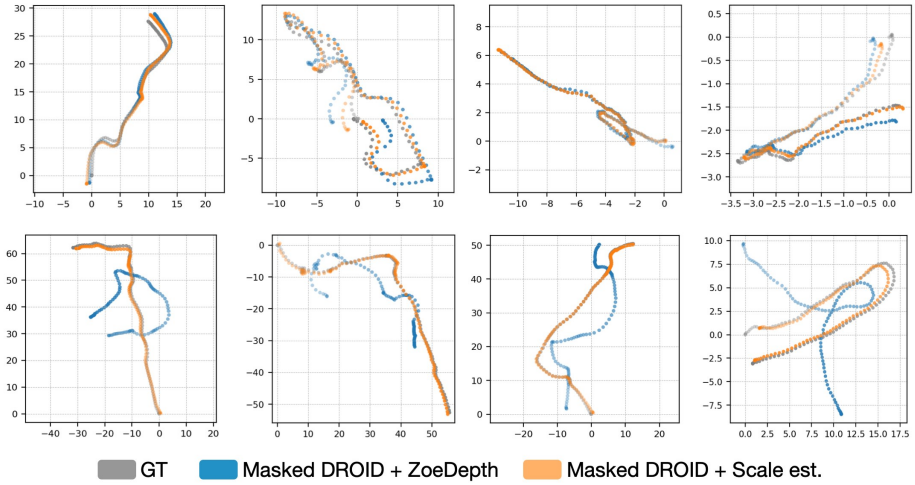

Fig. 7: Camera trajectory. Top row: examples where the naive combination achieves comparable results. Bottom row: naive combination leads to large error.

Our method TRAM derives camera motion scale from the background, by using a robust optimization procedure to align SLAM depth with predicted metric depth (Sec 3.3). As an alternative, we could give metric depth prediction to SLAM along with the input images as pseudo RGB-D inputs. RGB-D SLAM will then return trajectory in metric scale. As indicated in the main text, this naive approach leads to an average ATE-S of $3.09m$, while our method has an average ATE-S of $0.66m$. We visualize this difference in Figure 7. As shown, DROID diverges in roughly half of the sequences due to noisy or spurious depth predictions. Metric depth prediction cannot be treated as RGB-D inputs for a SLAM system.

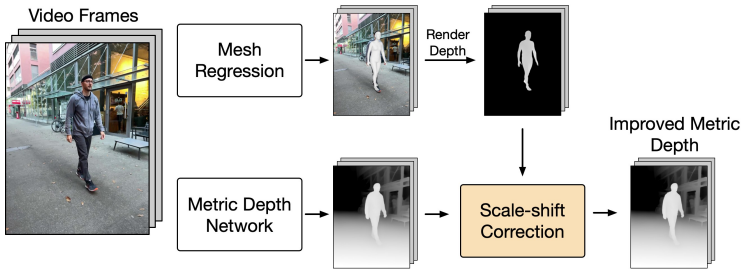

Fig. 8: Improving depth prediction. We render depth from SMPL reconstruction, and use this depth map to correct any shift or scale biases in the metric depth prediction. We scale and shift the depth prediction so that its human region would align with the rendered SMPL depth.

	EMDB 2 (ATE-S)			
Scale Estimation using	Short(5)	Medium(10)	Long(10)	Average
ZoeDepth	0.48	0.62	**0.78**	**0.66**
ZoeDepth + shift correction	0.37	**0.36**	1.41	0.78
ZoeDepth + scale-shift correction	**0.35**	0.37	1.43	0.79

Table 6: Camera scale estimation. Using SMPL depth rendering to correct scale and shift in depth prediction produces mixed results.

F.2 Using SMPL Depth to Improve ZoeDepth

We have shown that ZoeDepth prediction is not always accurate. Particularly, there could be shift and scale biases. In such cases, the metric depth prediction can be regarded as affine-invariant depth prediction. If there are objects of known depth in the image, we can use them to correct the shift and scale. Can we use human mesh reconstruction to help correct the biases? Specifically, could we estimate shift and scale variables s and t to correct depth prediction $\hat{D} = s*D+t$?

Figure 8 illustrates this approach. We solve for the scale and shift correction by aligning the human region in the depth prediction to the rendered depth from SMPL reconstruction, through energy minimization similar to the robust optimization in Sec 3.3 of the main text. We report the quantitative results in Table 6. We observe mixed results: it improves scale estimation in some sequences but decreases accuracy in others. Specifically, we observe that it decreases ATE-S (better) by 20% in 10/25 sequences but increases ATE-S (worse) by 20% in 5/25 sequences. The average ATE-S is slightly worse, because worse cases happen to be long sequences, so a small error in scale estimation could lead to a much higher translation error.

The effectiveness of this approach is also influenced by the accuracy of the mesh reconstruction. If the predicted human shape is more accurate, it will be more effective. Inaccurate shape prediction (e.g., the predicted human being is taller than the ground truth) will produce inaccurate depth rendering.

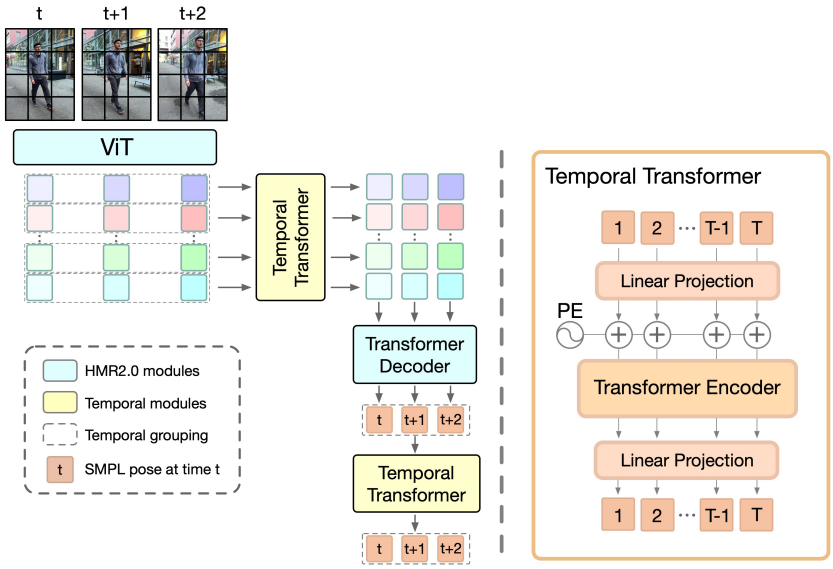

Fig. 9: Architecture of VIMO. Left: the detailed architecture of VIMO, with the yellow blocks denoting the new temporal components. Right: the architecture of the two temporal transformers.

G Implementation Details

G.1 Architecture

We show a more detailed view of the VIMO architecture in Figure 9. VIMO interleaves spatial and temporal modules. Both temporal transformers have 6 layers and 4 multi-head attention. The first temporal transformer (image domain) has an embedding dimension of 512, while the second temporal transformer (motion domain) has an embedding dimension of 384.

G.2 Datasets

We use 3DPW, Human3.6M, and BEDLAM to train our video transformer VIMO. We evaluate on 3DPW and EMDB. **3DPW** is an in-the-wild dataset providing ground truth 3D pose annotations acquired with IMU and videos. 2D and 3D joints are generated from the pose annotation. **Human3.6M** is an indoor multi-view dataset with 2D and 3D joint annotation. Additionally, we use SMPL recovered using MoSH for this dataset. **BEDLAM** is a large synthetic dataset rendered with Unreal Engine 5 and SMPL. Therefore, it has the most accurate SMPL pose and shape. **EMDB** is an in-the-wild dataset with accurate SMPL and trajectory annotations recovered with electromagnetic sensors.

During training, we sample sequences of 16 frames from the three datasets. There are about 1.3k sequences from 3DPW, 19k from Human3.6M, and 305k

from BEDLAM (30fps). So we sample sequences unequally from each dataset to guarantee a good mix of real and synthetic data, with the following ratio: [3DPW: 16.5%, Human3.6M: 16.5%, BEDLAM: 67%].

G.3 ORB-SLAM2

We use the open source ORB-SLAM2 implementation released by the authors in https://github.com/raulmur/ORB_SLAM2. For the masked evaluation, we first process the images in the dataset by setting all pixels within the human masks to a value of 255. We run the entire ORB-SLAM2 pipeline including camera tracking, point reconstruction, and loop closure. We specify our configuration based on the default monocular SLAM parameters for the TUM RGB-D dataset provided in the code and increase the number of features detected at each frame to 4,000. Additionally, because the EMDB videos sometimes demonstrate low contrast with a uniform background, we slightly lowered the minimum fast feature threshold per image patch (more details in the ORB-SLAM2 configuration documentation). Despite these efforts, we still observed tracking failures due to a fast-moving camera as well as textureless background regions. Compared to ORB-SLAM2, DROID performed better in handling these areas because they do not rely on distinctive sparse features; instead, DROID uses optical flow to guide correspondence, which shares the benefits of low-texture texture handling with direct SLAM methods. For loop closure, we use the bag-of-words vocabulary provided in the official repository.

G.4 Training

Acceleration. The training of video models is costly. Because the backbone is often frozen, previous methods pre-compute the features output by the backbone and use them as input to finetune the upper layers (including new components). While this approach reduces forward time, it is impossible to apply data augmentation. To address this issue, we do not pre-compute features but use two other methods: pre-cropping images and half-precision backbone inference. The datasets provide high-resolution images which could take longer to load, crop, and resize. Pre-cropping the images and saving them as crops reduces loading time. Using crops has a different disadvantage: data augmentation such as random rotation and scaling will produce black borders. While it could potentially reduce the effectiveness (it's not clear the extent), it is still better than no augmentation. Secondly, we use half-precision inference for the backbone, which reduces forward time. Since we do not finetune the backbone, using half-precision will not affect the training.

Data Augmentation. We apply standard data augmentation including rotation, scaling, horizontal flipping, color jittering, and occlusion. For video model training, all augmentations except occlusion are applied consistently in the same sequence. For example, the same degree of random rotation should be applied for all 16 frames of a sequence. However, each frame has an independent and equal chance of having occlusion augmentation.