Supplementary Material for
Propagate And Calibrate: Real-time Passive Non-line-of-sight Tracking

1. NLOS-Track Dataset

1.1. Real-shot Data

1.1.1 Video collection and position calibration

To obtain paired wall-shooting videos and ground truth trajectories, we record the relay wall with one camera and stick another camera on the ceiling to have a corresponding top-viewed video at the same time. Subjects walk around the hidden room with Aruco code on head so that we can track him with the top-viewed video. To improve the accuracy and robustness of tracking, we put four Aruco codes on the four corners of a hard and flat board. We use the Aruco API provided in OpenCV [43] to obtain the 3D pose of Aruco codes with respect to ceiling camera. In each frame of the top-viewed video, only when all four codes are detected, we take the average of four codes' coordinates as the character's position coordinate. Then translation and rotation are applied to transform the character's coordinate from the camera system to the world system.

1.1.2 Stream alignment and cropping

We use two pulse signals that only last for a short moment to mark the beginning and the ending time point of a single clip. Two signals are designed to be visible by exposing them to both two cameras. Since the time intervals of the two signals in the two videos are equal, we are allowed to manually align the two videos at a frame-level precision by aligning the beginning time point and the ending time point in the timeline.

1.1.3 Data cleaning

Although we use four codes to improve the tracking robustness, there are still some point-sequences that we fail to track due to jitters and blurs of pictures. Assume that the missing points in one sequence are $\{\vec{p}_i\}$, we use a linear interpolation to complete them:

$$\begin{aligned}\vec{p}_i &\approx \frac{N-i}{N}\vec{p}_0 + \frac{i}{N}\vec{p}_N \\ &= \vec{p}_0 + \frac{i}{N}\left(\vec{p}_N - \vec{p}_0\right), \quad i = 1, 2, ..., N-1,\end{aligned} \tag{8}$$

where i is the index of the missing point. $\vec{p}_0$ denotes the recorded point previous to the missing sequence while $\vec{p}_N$ denotes the subsequent recorded point. To perform this interpolation, we make a reasonable assumption that there are no sharp turns or changes in speed within N frames. In addition, we set a threshold of $N \leq 10$ when conducting the interpolation, which reinforces this assumption so that we won't introduce unbearable errors during data cleaning.

After completing missing points and excluding clips that can not be completed, we crop the remaining paired videos and ground truth trajectories into many 250-frame clips. The videos are save as `.npy` files and trajectories are saved as `.mat` files, along with corresponding room size.

1.2. Random Trajectory Generation

To simulate the realistic situation of people walking in the room, we use a heuristic algorithm for generating near-real continuous trajectories frame by frame. Given the room size, we first select a random position $\vec{p}_0 = (p_0^x, p_0^y)$[1] within the room as the initial position and a random vector $\vec{v}_0 = (v_0^x, v_0^y)$ as the initial walking direction. Considering the effect of inertia and momentum in real world, in each subsequent frame we make a slight and random change $\Delta\vec{v}_t$ to the current walking direction as that of the next step:

$$\vec{v}_{t+1} = \vec{v}_t + \tau \cdot \Delta\vec{v}_t \ , \ t = 0, 1, ..., \tag{9}$$

where τ is an adjustable parameter called *turning rate*. By setting τ properly, one can control the curvature degree when generating trajectories. Specifically, we set the turning rate τ to 0.15 in our dataset because this value guarantees a reasonable continuity and rotation magnitude of generated trajectories. In each frame, the character takes a step forward in the new direction $\vec{v}_{t+1}$ and a continuous trajectory can be generated:

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t \ , \ t = 0, 1, ... \tag{10}$$

Note that the distance (in meters) of each frame step is not fixed, but is randomly sampled with a uniform distribution $U(0.03, 0.04)$ for the sake of being realistic.

[1] Here we use p to denote position instead of x, which is distinguished from x used to denote direction.

In addition, we ensured that the characters did not cast any direct shadow on the wall that would be visible to the naked eye while generating the trajectory.

1.3. Data Format in Dataset

We store all video clips in `.npy` format after cropping them to squares and resizing them to a dimension of $T \times C \times H \times W$, where $C = 3$ is the number of channels and $H = W = 128$ is the spatial dimension. $T = 250$ in real-shot data while $T = 320$ in synthetic data.

2. PAC-Net

2.1. Model Details

We use PyTorch [48] to build and train our neural networks. Besides ResNets [45], there are only two other basic components in our model, which are GRU cells [44] and a self-designed MLP decoder. The vector dimension of extracted features and hidden state $\mathbf{h}$ are both 128. Before loading the pre-trained weights of ResNets, we substitute the last linear layer in ResNets with another one with an output dimension of 128. The two-layer MLP decoder takes in the hidden state $\mathbf{h}$ as input. The 64-dimensional intermediate vector is activated by a ReLU, followed by a final linear layer that outputs the 2-dimensional coordinate.

2.2. Model Training

During offline training, we use an equivalent implementation of networks to improve the training efficiency. Instead of extracting feature frame-by-frame, we use two ResNets in PAC-Cell to extract features from raw frames and difference frames all at once after loading a video clip. Then static features (from raw frames) and dynamic features (from difference frames) are fed to P-GRU and C-GRU alternatively.

For all networks, we train models for 70 epochs using AdamW [47] optimizer with a weight decay of 2e-3. The learning rate is set to 3e-4 and follows the cosine annealing schedule [46]. The batch size is set to 32.

During training, we load a random T'-frame clip from the whole video clip of T frames, where $T' \leq T$. This practice further enhances the diversity of datasets.

3. More Analysis

3.1. Light sources

We set all three light sources on the ceiling (please refer to Fig.1 in the main paper) so the useful information on the wall mainly comes from the diffuse reflection rather than obvious shadows (please refer to Sec.3 in the main paper). Since the tracked person is always moving, the position of lights relative to the person is not fixed.

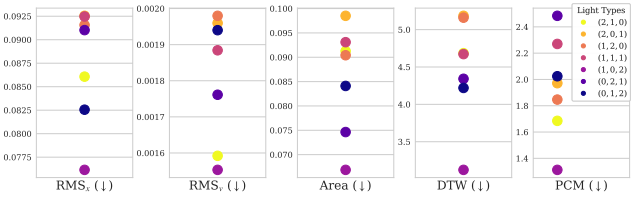

Figure 8. **How light sources affect metrics.** Light type of $(1, 0, 2)$ means 1 point light, 0 spot light and 2 area lights are in the room.

In the synthetic dataset we vary the *type(point, spot, and area), position, rotation*, and *power* of three light sources. With so many factors influencing the light condition, it is a ponderous task to conduct a fair and comprehensive investigation. In Fig. 8, we preliminarily show how metrics vary with different light type combinations. We can see that with 2 area lights, we have relatively good tracking results on average because more area light makes the room brighter, thus can provide sufficient diffuse signal cast on the relay wall for NLOS tracking. We found there is a coarsely positive correlation between light source area and performance.

3.2. Warm-up

When facing a variety of room settings in synthetic data, the Warm-up stage plays an important role in "adapting" to the current room (please refer to the grey dashed lines in Fig.4 in the main paper). In contrast, due to the relatively limited real-shot scenes, the potential of the Warm-up stage cannot be fully demonstrated. We conduct an extra comparison test on a small synthetic dataset with only two room sizes. And we observe only minor differences between w/ and w/o Warm-up (Tab. 3). Thus we conclude that Warm-up requires a diverse dataset to work.

Model	RMS$_x$	RMS$_v$ ($\times 10^{-3}$)	Area	DTW	PCM
w/o Warm-up	**0.1103**	2.50	0.0776	1.698	**2.613**
w/ Warm-up	0.1105	**2.03**	**0.0715**	**1.589**	3.429

Table 3. **Comparison test on warm-up.**

4. More Visualisations

To demonstrate the test results in real scenes in an intuitive way, we attach two demo videos in supplementary materials[2]. In each video, we present the equivalent real-time reconstructed trajectory, along with the corresponding raw stream and difference stream. We use red squares to highlight when and where the faint variation in the relay wall can be detected by naked eyes with difference frames.

[2]Demos can also be found on the project website.

References

[43] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000. 1

[44] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Empirical Methods in Natural Language Processing*, 2014. 2

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 2

[46] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 2

[47] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 2

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 2