# Continuous-Time Human Motion Field from Event Cameras

Supplementary Material

## 8. Implementation Details

### 8.1. Training Details

**Multi-state Training.** We deploy a multi-stage training strategy, as briefly described in Sec. 4.5. First, we train the Event Human Motion Predictor (E-HMP) described in Sec. 4.2 for 10 epochs. During this training, we use a dummy translation prediction network that directly regresses the poses as in [74], which is then later removed. In parallel, we train the Glboal Motion Predictor (GMP) described in Sec. 4.3 for 5 epochs. At this point, the GMP network is slightly overfitted to the training data. We then freeze the GMP and merge with the E-HMP to help convergence, and then unfreeze it after 1 epoch and decrease the learning rate to 1e-5 to jointly fine-tune the two networks. Empirically, we find that this iterative training approach helps convergence for both the GMP and the E-HMP. For each component of the multi-stage training, we use a start learning rate of 1e-3 and then decay it to 1e-4 after one epoch to help stabilize training. We use batch size of 16 for all networks.

**Hyperparameters.** In all experiments, we set $\lambda_{ori} = 10$, $\lambda_t = 10$, $\lambda_{3D} = 20$, $\lambda_{2D} = 20$, $\lambda_{flow} = 0.1$ and $\lambda_c = 0.1$. The starting learning rate is 1e-3 for all networks trained from scratch and decayed by 10 after the first epoch. We use a learning rate of 1e-5 for fine-tuning the GMP network. We use Adam as optimizer for all experiments.

**Global Motion Field versus Global Motion Predictor** Here, we disambiguate between the two types of "global" motions mentioned in the main paper. The global motion field, which is decoded from the latent code $z_g$, represents the relative rotation of the root joint. These rotations are important to convert joint positions into the root-adjusted global pose, which are needed for the Global Motion Predictor. On the other hand, the Global Motion Predictor (GMP), predicts the root velocity based on the input joint positions, joint velocities, joint orientations, and rotational velocities. The GMP does not use $z_g$ directly. Instead, it uses pelvis-centered local poses (decoded from $z_l$) rotated by each root orientation (decoded from $z_g$) to compute the translational velocity estimate.

## 9. Architecture

### 9.1. Event Human Motion Predictor (E-HMP)

First, we use a pre-trained ResNet50 image encoder pre-trained on ImageNet to compute features from the event volumes. The initial convolutional layers with three input channels are replaced with ones having eight input channels. The output features at different steps are processed with a Gated Recurrent Unit network that recurrently updates a hidden state. In the last step $t_T$, we use a linear layer to project the hidden state to the a vector of size $1024+256$, which is then divided into the global code $z_g$ and $z_l$. The GRU has one layer with $2048$ hidden size with no dropout and with one direction.

**Multi-layer Preceptron Motion Decoder** Our continuous decoding rely on an MLP that takes timestamps as input and outputs the pose parameters. We used an MLP with 11 linear layers with ReLU nonlinearities and skip connections. The input timestamps are mapped to positional encoding and concatenated with the latent codes, as input to the MLP.

### 9.2. Global Motion Predictor (GMP)

Following [2], we use Skeleton Convolutions to aggregate features on the graph defined by the SMPL skeleton. The local poses parameters pass through 3 layers of convolution layers, followed by four more 1D convolution layers and average pooling layers to obtain the final translational velocities. Each pose predicts on velocity vector. No temporal blending is performed at this step.

## 10. Additional Human Mesh Prediction Results

In Fig. 10, we show the results of human predictions at five different frame rates. Due to the continuous nature of our Event Human Motion Predictor (E-HMP), the poses can be queried at any timestamp during the event duration. We only show 120 FPS here because it is difficult to visualize a higher frame rate in a 2D image. Animated results can be found in the supplementary video. Each row corresponds to a 1 second sequence of human motion. We overlaid the predictions at different timestamps onto the same image. It can be observed that our network allows 120 FPS decoding in parallel, which produces smooth and continuous-time movements. We note that the continuous human motion field is predicted at once from the events, and sampling poses at different timestamps requires only inputting arbitrary timestamps to the lightweight MLP.

### 10.1. Smoothness

Temporal Smoothness Evaluation We provide additional visualization of improved temporal smoothness than per-frame prediction in Fig. 9. By decoding the entire sequence at once, our model leverages temporal smoothness to produce smooth and coherent predictions, particularly for mo-

Figure 8. **Left**: Accurate tracking under smooth motion with dense event observations. **Middle**: Sudden acceleration causes the model to lose track of the head. **Right**: Failure in regions with sparse events, such as the static legs. Faces masked for anonymity.

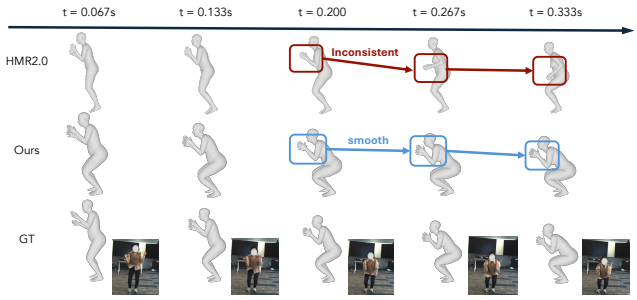



Figure 9. **Temporal smoothness.** Comparison between our model and HMR2.0 on the *Starjump* sequence. HMR2.0 predicts temporally inconsistent meshes due to single frame prediction, while our model maintains smooth motion.

tions such as arm folding. In contrast, HMR 2.0 tends to rely on frame-wise predictions, making it more prone to local errors and leading to reduced temporal consistency across the sequence.

### 10.2. Failure Cases

In Fig. 8, we illustrate several common challenges in event-based motion estimation arising from sensitivity to contrast thresholds. When the contrast between the foreground and background is low, the resulting events contain insufficient information for the network to reliably track the human subject. Conversely, increasing the sensitivity leads to the generation of extraneous noisy events, which can degrade performance. We leave the resolution of this trade-off to future work.

## 11. Beam-splitter Event Agile Human Motion Dataset (BEAHM)

### 11.1. Collection setup

BEAHM uses four high-speed Flir RGB cameras (1920 $\times$ 1080) to collect images for annotations, will be referred to as GT cameras. And a beam splitter with a Flir RGB camera (2448 $\times$ 2048) and a Prophesee Event camera (640 $\times$ 480) to capture paired event data and images. The beam splitter and trigger box design are shown in Fig. 13. The setup for data collection is shown in Fig. 14.

Before data collection, all cameras, including the event camera, are calibrated using Kalibr [12]. The calibration procedure for the event camera follows the methodology in [43]. The re-projection error for each camera is sub-pixel. The calibration result, raw data, ground-truth labels and data collection software will be publicly released as part of BEAHM.

During data collection, ground truth cameras ran at 125 Hz with an exposure time set to 5 ms to minimize motion blur. The RGB camera within the beam splitter is triggered at 15.625 Hz (division of 8) with an exposure time of 50 ms to produce intentionally blurry frames for comparison. All RGB cameras are set to autogain. The event camera receives the same trigger rate as the ground-truth cameras. We evaluated the trigger drift of our 125 Hz synchronization signal. As shown in Fig. 15, the trigger interval has a minimal drift of on average 0.01938 ‰.

For visualization, we plot the speed distribution of non-static joints in BEAHM and MMHPSD dataset. As shown in Fig. 12, BEAHM has a larger proportion of high-speed joints ($>$ 3 m/s), featuring fast and diverse motions for training and evaluation.

Using four views from different directions, data in BEAHM is annotated with Easymocap [1], which employs 2D joint detection and triangulation to determine 3D joint locations. Examples of the annotation results are shown in Fig. 11. In the figure, the annotated SMPL model is overlaid on the original images to visually assess annotation quality.

In our collections 9 out of 160 sequences are discarded due to inaccurate annotations, primarily caused by occlusions of certain parts of the human body.

### 11.2. Sequences

In total, BEAHM consists of 160 sequences with 40 different motions and 200 thousand frames of SMPL annotations. The motions are categorized by prediction difficulty into 3 increasing levels, varying from basic motion such as arm abduction to extreme sports such as Taekwondo, Volleyball and Tennis. The details of captured motions are described in Tab. 4. We present 5 example sequences: Left arm wave, Bicep curl, Left lunge, Lean left, Volleyball of our BEAHM dataset in Fig. 11.

## 12. Baseline Methods

### 12.1. DHP19

Since the annotation method differs between BEAHM and DHP19 data, we re-trained the DHP19 network on our dataset using 24 joints derived from the annotated SMPL model. To generate ground truth for the DHP19 network, SMPL annotations are converted and projected into 24 2D joint positions, which are then rearranged into a heatmap

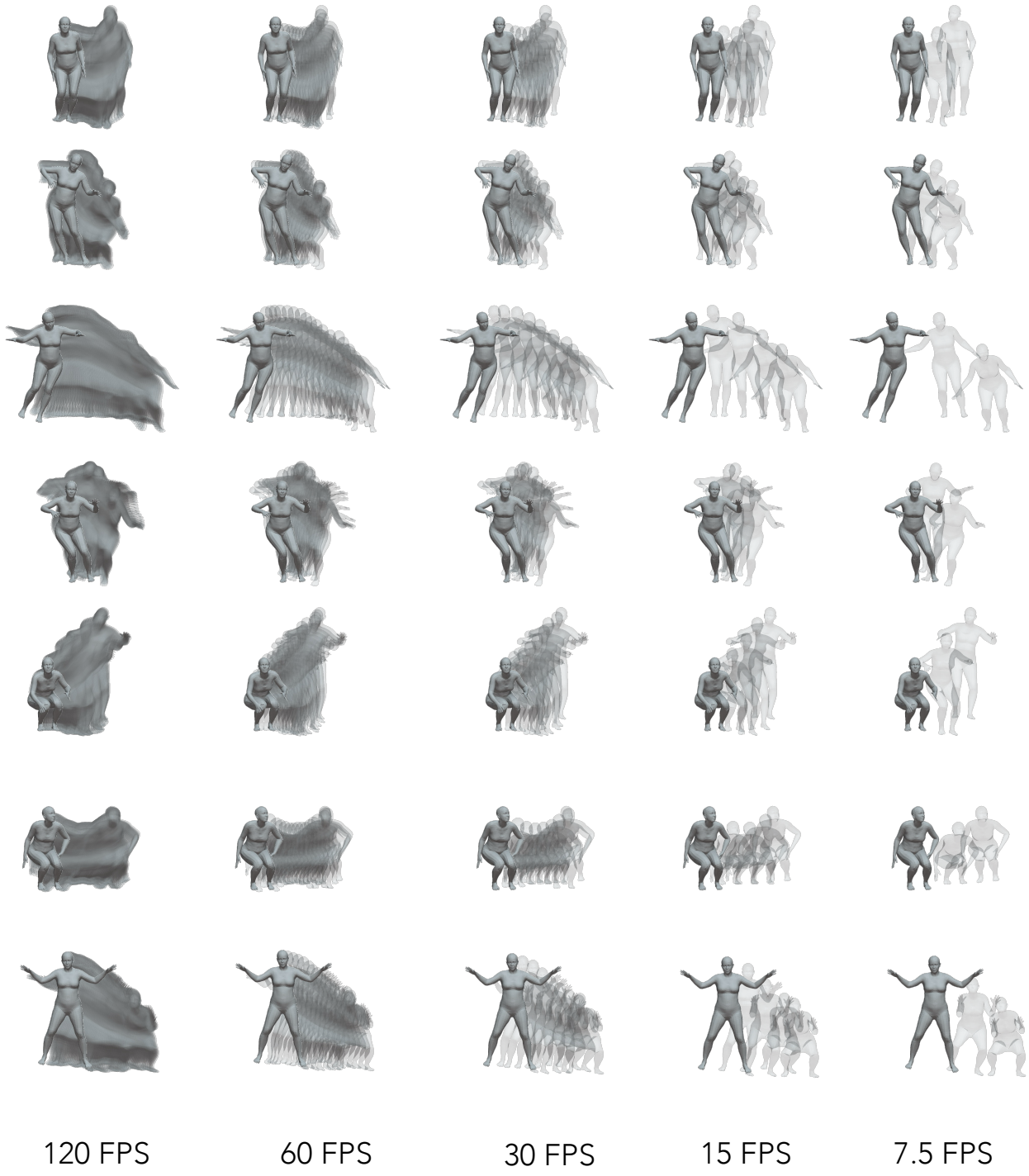



Figure 10. Predicted Human Motion Sequence. We show the sequences of human motion in each image by overlaying predictions at different timestamps. Past predictions are rendered with high transparency.

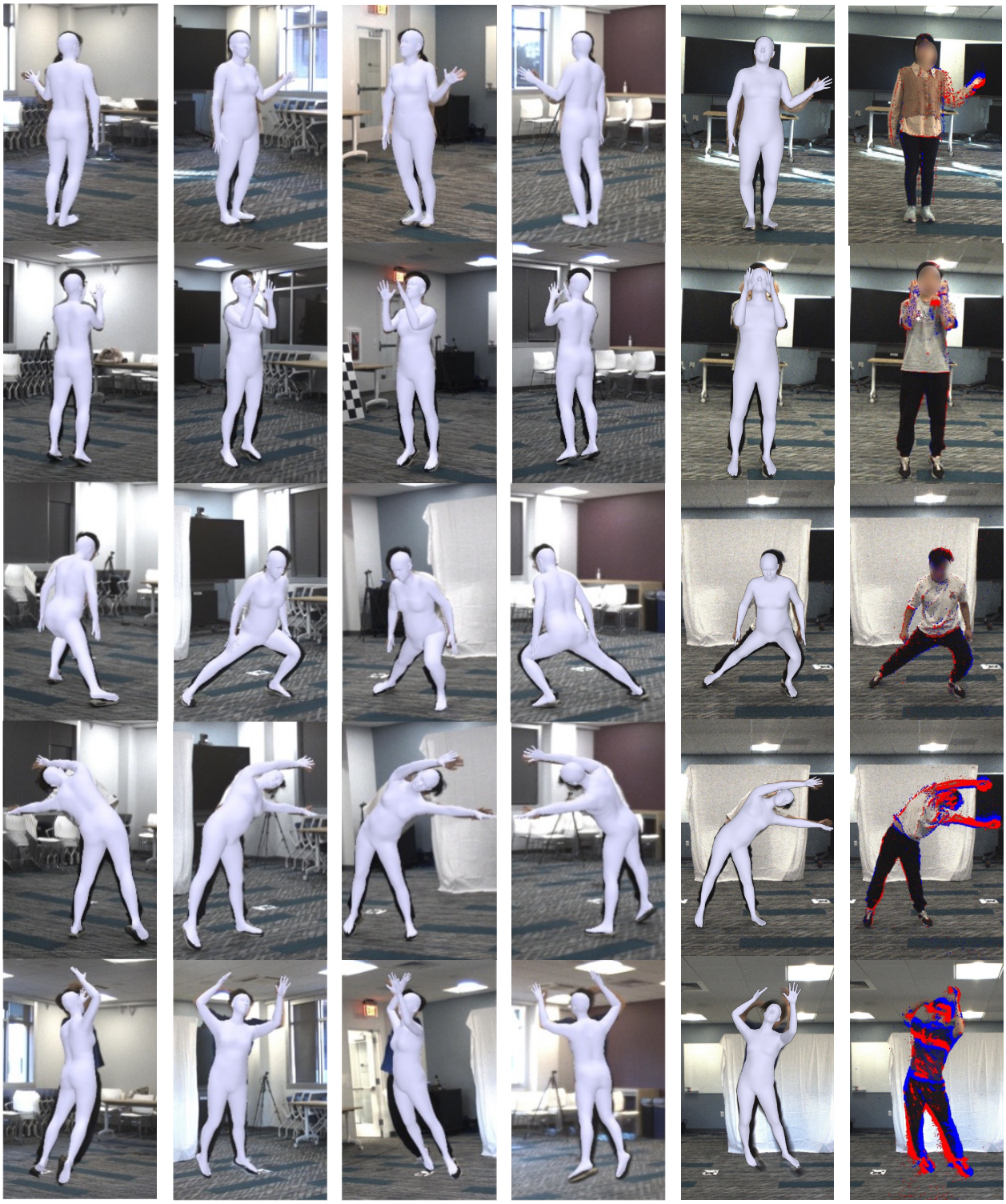



Figure 11. We present six example sequences from BEAHM. Each sequence, from left to right, includes: **(a-d)** Four multi-camera images with SMPL annotation overlaid via EasyMocap [1]. **(e)** The estimated mesh model superimposed on the beam splitter RGB camera. **(f)** Events displayed on the beam splitter RGB camera.

Table 4. BEAHM dataset details. BEAHM consists of 40 different human motions that are categorized into **Basic**, **Medium** and **Extreme** levels. **Basic** activities include motion of arms, legs, main body and head. **Medium** activities are common actions involving more than 2 parts of the human body. **Extreme** activities are competitive sports with fast motions

| **Category** | **Actions** | | | | |
|---|---|---|---|---|---|
| | **Basic** | | | | |
| Arms | Left arm upward | Right arm upward | Bicep curl | Left arm wave | Right arm wave |
| | Left arm circle | Right arm circle | Left punch | Right punch | Punch |
| | Left arm raise | Right arm raise | Left arm outward | Right arm outward | Outward |
| Legs | Left knee lift | Right knee lift | Left hop | Right hop | Left kick |
| | Right kick | Left lunge | Right lunge | | |
| Body | Lean left | Lean right | Rotate torso | Rotate head | Nod |
| | **Medium** | | | | |
| Common Activities | Walk | Jog | Jump up-down | Jump forward-back | Jump sideway |
| | Starjump | Squat | | | |
| | **Extreme** | | | | |
| Sports | Taekwondo | Tennis | Volleyball | Gymnastics | Shot put |

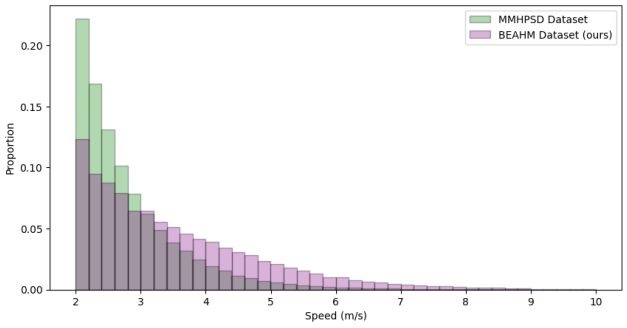
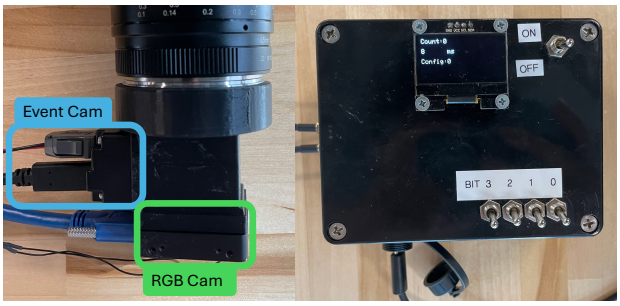



Figure 12. Speed distribution of joints in MMHPSD and BEAHM.




(a) (b)

Figure 13. **Beam splitter and trigger box design (a)** Beam splitter with an event camera and a paired RGB camera. **(b)** Trigger box design with adjustable time interval.

with 24 channels, each representing the probability of one joint. The output dimension of the DHP19 network is adjusted to 24 accordingly. For smoothing the heatmaps, we use a 15-pixel Gaussian blur kernel from Torchvision with sigma of 2.6 pixels for computational efficiency. The whole implementation is built using PyTorch and additional details such as data format, optimizer, and learning rate decay strategy follow the setup of the original paper.

### 12.2. EventCap

EventCap [66] is a pioneering approach for capturing high-speed human motions using a single event camera, achieving human pose estimation at 1000 fps. We followed the steps outlined in the original EventCap [66] paper with modifications to adapt the algorithm to both the MMHPSD dataset and our BEAHM dataset. We adapted the EventCap baseline to use SMPL models instead of the proprietary scanned textured human models in the original implementation. We initialized SMPL parameters using HMR 2.0 outputs through slerp interpolation.

The entire EventCap [66] pipeline is divided into two main components: batch optimization and per-frame event refinement. In the batch optimization stage, we included four loss terms: correspondence loss, 2D loss, 3D loss, and temporal loss. To obtain feature tracks to compute the correspondence loss, we experimented with both event feature tracking [15] and image-based tracking [38]. We found that image feature tracking provided more stable performance. To give EventCap a fair shot, we reported the performance of the better model using the image-based features. In the temporal loss term, we only included joints in the energy term if they were close to events, thereby introducing a temporal stabilization constraint for non-moving body parts. We empirically chose the weight of each loss term for both the MMHPSD dataset and our BEAHM dataset. The chosen weights are $\lambda_{3D} = 10$, $\lambda_{2D} = 10$, $\lambda_{temp} = 0.01$,

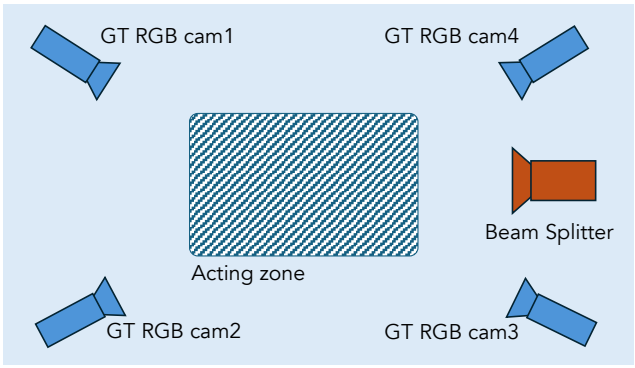



Figure 14. **BEAHM collection setup.** 4 high speed cameras for gt are placed at 4 corners and a beam splitter is placed at one side to capture human in the middle of the room. The space of acting zone is 1.5m $\times$ 2m.

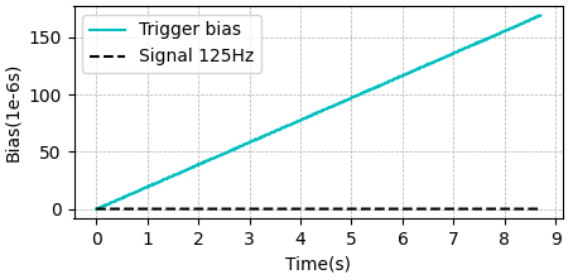



Figure 15. **Hardware synchronization signal evaluation** We evaluated the bias drift of triggers by recording the timestamp of the trigger subtracted by a standard 125 Hz signal, the total drift over 8.6 s is 0.169 ms.

$\lambda_{cor} = 2.5$. The event refinement implementation adhered largely to the original paper[66]. $\lambda_{dist}$ is used to determine the relative scale between spatial distance and temporal distance when finding the closest event to a human contour pixel. We chose $\lambda_{dist} = 0.5$ for both datasets. The entire EventCap pipeline was implemented in PyTorch with SGD as the optimizer. We ran 10,000 iterations for batch optimization and 2,000 iterations for event refinement.