### Abstract

_In this supplementary document, we first (Sec. 1) provide more details about our method and recording setup. In Sec. 2, we provide implementation details. In Sec. 3, we provide more details about our experimental setup._

### 1. Method

#### 1.1. 3D Scene Reconstruction and RGB image Database

To obtain a representation of the 3D scene, we use a standard commercial solution to obtain a very dense scene point cloud: NavVis M6 [2] mobile capture system. It makes use of 4 LiDAR sensors and 6 RGB cameras. The scene is reconstructed from the LiDAR data and RGB images using a proprietary algorithm. The algorithm, in the process, also provides the extrinsic and intrinsic parameters for all the RGB images. It took six hours to scan all the 8 scenes of the dataset.

Potentially, other SfM or MVS methods using cheaper sensors are also an option - the exact type of the 3D scanning method is unimportant as long as it registers captured RGB images within the scene.

#### 1.2. Camera Localization

To obtain 2D-3D correspondences, for each image in the dataset we produced a rendering of the scene point cloud using known extrinsic and intrinsic camera parameters (Fig. 1). For rendering, we use the surface splatting technique [17] with fixed splat size. Together with the color rendering we produce a point mapping of the image. The resolution of the map is the same as the resolution of the color image. Each pixel of this map contains the index of the point in the pointcloud which is visible in that pixel. As NavVis scanners use fisheye cameras, photos obtained from them are heavily distorted. This affects keypoint detection performance. To alleviate this, we undistort both camera images and point mappings by generating a mapping between our fisheye camera image plane and a fixed perspective camera plane. Undistorted images are run through the SuperPoint [7] keypoint detector. In the end, each keypoint is mapped to the position of the 3D point on the scan according to the generated point mapping.

#### 1.3. IMU and head-mounted camera setup

For the head-mounted camera, we use a GoPro camera. We ensure that it remains tightly attached at the forehead throughout recordings. For IMUs, we use Xsens Awinda [12] system consisting of 17 units attached at the body. IMUs provide the orientation and acceleration in the local coordinate frame of the sensor which is then used to estimate the pose of the person.

#### 1.4. IMU-Camera synchronization

To synchronize IMU and camera data, we ask each subject to clap at the beginning of recordings. We detect
these claps in both modalities to obtain synchronized starting frames. To check for time drift between the two data streams, we perform the following experiment: we made a special 1 hour long recording with a subject clapping every 5 minutes. We synchronize our system with the first clap and measure the difference in time we get for each consecutive clap after that. We noticed a small accumulation of drift (Fig. 2) – around 87 ms per hour. We take this into consideration while performing long recordings.

1.5. Body Part Vertices

We manually define vertices corresponding to the heels and toes of the two feet and also hips (for our modified algorithm in Sec. 1.6) in Blender [5] to enforce contact constraints. These vertices are shown in Fig. 4 and Fig. 5.

1.6. Modified algorithm

To deal with persistent camera localization outliers (outdoor scenes, indoor scenes with repetitive patterns) we implemented a modified version of our algorithm. This algorithm differs from the one described in the main paper in the minimization objective and initialization algorithm. Notation remains the same as in the main paper.

Minimization objective is formulated as follows:

\[ E(\theta_{1:T}, t_{1:T}) = w_i E_{\text{int}} + w_s E_{\text{scene}} + w_{sm} E_{sm} \]  

We optimize with respect to pose \( \theta_{1:T} \) and translation \( t_{1:T} \) parameters, as in the main paper. We will now explain each of the terms in more detail.

Scene Contact Term \( E_{\text{scene}} \): While IMUs can detect foot contacts, our subjects often sit down on flat surfaces in the
scene. To detect sitting motion, we develop a heuristic that involves IMU velocities and pose. First we manually define a vertex \( v \) on the heel of the left foot. To detect frames with hip contacts, we first compute the \( d_{sit} \) - distance between the foot vertex and the root vertex in a particular frame and \( d_{stand} \) - the distance between the foot vertex and the root vertex when the person is in T pose. If \( d_{sit} < 0.66d_{stand} \), we assume that the hips in that frame should be in contact with the scene. See Fig. 3.

Let \( d^2(\theta, t) = ||M(\theta, t)v_n - t||^2 \). If \( d^2(\theta^f_j, t^f_j) < \delta_1d^2(0, t^f_j) \) and \( ||t^f_j - t^l_{j-1}|| < \delta_2 \), we assume that the hips are in contact with the scene at frame \( j \). We set \( \delta_1 = 0.66 \) and \( \delta_2 = 0.001 \).

When the IMUs detect a foot contact or our heuristic detects a hip contact, we force that body part to be in contact with the ground by using an energy term consisting of two subterms \( E_{scene} = w_cE_{contact} + w_sE_{slide} \). Mathematically the two subterms are almost the same as eq. 5 and eq. 6 in the paper:

\[
E_{contact} = \frac{1}{5T} \sum_{j=1}^{T} \sum_{k=1}^{5} \sum_{n \in \mathcal{B}_k} \frac{1}{|\mathcal{B}_k|} ||M_n(\theta_j, t_j) - v(n)||^2 ,
\]

Similarly to our main algorithm, to prevent the feet and hips from sliding when in contact with the scene, we constrain the distance between foot or hip parts in contact with the scene in two successive frames to be zero

\[
E_{slide} = \frac{1}{5(T-1)} \sum_{j=1}^{T-1} \sum_{k=1}^{5} \sum_{n \in \mathcal{B}_k} \frac{1}{|\mathcal{B}_k|} c^j_k c^j_{k+1} ||M_n(\theta_j, t_j) - M_n(\theta_{j+1}, t_{j+1})||^2 .
\]

**Smoothness Term** \( E_{sm} \): We impose smoothness constraints on global translation, and global orientation, but not the head orientation:

\[
E_{sm} = w_T E_T + w_G E_G,
\]

where the translation term equals:

\[
E_T = \frac{1}{T-1} \sum_{j=1}^{T-1} \max(||(t_j - t_{j+1})||^2 - ||(t^f_{j} - t^l_{j+1})||^2, 0) .
\]

\[
E_G = \frac{1}{T-1} \sum_{j=1}^{T-1} ||(\log((R^G(\theta_j))^T R^G(\theta_{j+1})))^\top||_2
\]

The function \( R^G \) is the same as described in the main paper.

**Interpenetration Term** \( E_{int} \): We also add a robust interpenetration term. We force points inside SMPL to be in the space unoccupied by the scene. We first define an occupancy field of the scene by voxelizing the scene point-cloud. By trilinearly sampling the occupancy field, we define a function \( S : \mathbb{R}^3 \rightarrow [0, 1] \) that maps every point in space to an occupancy value. To obtain points inside the mesh, we project face centroids inside the mesh using the face normals. We define:

\[
N(f, \theta, t) = \frac{(M(\theta, t)_{f_1} - M(\theta, t)_{f_2}) \times (M(\theta, t)_{f_3} - M(\theta, t)_{f_2})}{||M(\theta, t)_{f_1} - M(\theta, t)_{f_2}|| \times (M(\theta, t)_{f_3} - M(\theta, t)_{f_2})||^2}
\]

\[
P(f, \theta, t) = \frac{\sum_{k=1}^{3} M(\theta, t)_{f_k}}{3} - \pi N(f, \theta, t)
\]

where \( f \) is a face in SMPL defined by three vertex indices \( f_1, f_2, f_3 \) and \( \pi \) is a hyperparameter that controls the depth of projection (selected randomly from 0 to 0.1 for each face). The interpenetration loss is defined as

\[
E_{int} = \frac{1}{T|F|} \sum_{f \in F} \sum_{j=1}^{T} (S(P(f, \theta_j, t_j)))
\]

where \( F \) is the set of SMPL model faces.

**Initialization:** To initialize translation parameters \( t_j \) we use filtered camera localization estimates \( t^f_j \) obtained from the filtering algorithm Alg. 1. This algorithm is specifically designed to deal with the frequent camera localization outliers: we use IMU location estimates to identify erroneous camera localizations and using correct camera localizations, we orient IMU trajectories to match the person’s overall trajectory and interpolate between the filtered frames using these corrected IMU trajectories.

For pose initialization we use the same method as in our main paper (see eq. 11).

### 2. Implementation

#### 2.1. Joint Optimization

In each iteration of our optimization algorithm, we optimize for 100 frames in a single batch. To ensure a smooth transition between results from adjacent batches, we force the translation and global orientation of the first frame in a batch to be the same as the corresponding parameters of the last frame in the previous batch. We implement our method in Pytorch [11]. Specifically we use the Adam optimizer [9] and optimize each batch of frames with 100 to 2000 gradient steps. Using an Nvidia Volta V100 GPU, one gradient step takes about a quarter of a second.
Algorithm 1: Filtering Algorithm

Input: \( \eta, \epsilon \), IMU positions: \( t_{t_1:N}^{C} \), Camera localizations: \( t_{C}^{C} \), a function that returns an array of values - (start, end) of contiguous blocks in an array of indices.

Output: Filtered positions: \( t_{F}^{C} \)

Init: \( vel_{\text{max}} = \infty, t_{F}^{C} = t_{C}^{C} \), invalids = []

while \( vel_{\text{max}} > \epsilon \) do
  for \( j \) in \( 1, \ldots, N \) do
    \[ v_{j}^{F} = \| t_{j+1}^{F} - t_{j}^{F} \|_2, v_{j} = \| t_{j+1}^{C} - t_{j}^{C} \|_2 \]
    if \( v_{j}^{F} > \eta v_{j}^{C} \) then
      invalids = invalids + \([j]\)
  for \( (\text{start}, \text{end}) \) in contiguous(invalids) do
    \[ v_{ta} = \frac{v_{\text{end}+1}^{F} - v_{\text{start}-1}^{F}}{\| v_{\text{end}+1}^{F} - v_{\text{start}-1}^{F} \|_2}, \]
    \[ v_{in} = \frac{v_{\text{end}+1}^{C} - v_{\text{start}-1}^{C}}{\| v_{\text{end}+1}^{C} - v_{\text{start}-1}^{C} \|_2} \]
    for \( k \) in \( \text{start} \ldots \text{end} \) do
      \[ t_{k}^{F} = t_{k}^{C} + \exp(v_{ta} \times v_{in}) (t_{k}^{F} - t_{k-1}^{F}) \]
    \[ v_{F}^{C}_{t_{1:N} - 1} = \| t_{F}^{C}_{t_{1:N} - 1} - t_{F}^{F}_{t_{1:N} - 1} \|_2 \]
    \[ vel_{\text{max}} = \max(v_{\text{in}}) \]
  end
end

2.2. Camera self-localization pipeline

For feature extraction and matching, we use SuperPoint [7] and SuperGlue [13] PyTorch implementation pretrained on MegaDepth [10] and ScanNet [6] datasets. We detect 4096 keypoints at max for each image and perform 40 sinkhorn algorithm iterations at matching stage. For database prefiltering we use NetVLAD PyTorch implementation pretrained on Pittsburgh 250k dataset [16]. We select 40 most similar database images based on the cosine distance between query and database NetVLAD descriptor. We use COLMAP [14, 15] software to minimize the reprojection error of the matched keypoints. Overall, the pipeline takes around 3s to localize a frame at 1920 × 1080 resolution using Nvidia Q8000 GPU.

2.3. Camera calibration

To retrieve intrinsic parameters of the head-mounted camera, we take several photos of a checkerboard pattern and use OpenCV [4] camera calibration tools to get the parameters. In our camera self-localization pipeline, we use an OpenCV camera model with 2 radial and 2 tangential distortion coefficients.

3. Experiments

To obtain the ground truth data for our comparisons we use a system of 3 Azure Kinect RGB-D sensors [1] in a time-synchronized mode. The system is set up in a way that it covers all regions of the body and provides a 360° view of the subject every 1/30 of a second (Fig. 6).

To obtain the RGB-D videos from Kinects we use a custom recorder written in C++ running on a separate computer for each Kinect. To ensure time synchronization, Kinects are connected sequentially through the special time synchronization input and recorders are controlled over the wireless network. The video is recorded at 30 FPS with a color resolution of 1920 × 1080 pixels and a depth resolution of 640 × 576 pixels.

**Ground-truth Point clouds:** To register the ground truth point cloud to the static 3D prescanned scene, we use the following 3-stage method:

- Visual localization: We obtain an approximate camera position for each depth-camera using the same visual localization method described in the paper.
- Depthmap-to-scene ICP: We align the partial point cloud of each depth camera with the scene using ICP [3]. ICP is initialized using the camera parameters from the previous stage.
- Manual correction: Some results can still be erroneous due to depthmap artifacts or due to incorrect ICP initialization. These results are corrected manually. As a final step, we run ICP again.

**Video recording preparation.** Prior to the actual ground-truth video recording, we record a short video of the scene from all kinects with no one present in the field of view. Frames are averaged and average depth is used in 2 ways: in depthmap-to-scene ICP alignment and for the background subtraction.

**Background subtraction.** To compute our metrics using obtained point clouds we need to make sure that we get
only points corresponding to a subject. To achieve this, we first subtract the background from depth videos by ignoring all pixels with depth more or equal to the prerecorded empty scene depthmap. After that, we unproject the points from all 3 Kinects to the 3D space and run DBSCAN \cite{8} clustering with the maximum distance between clusters of 0.12 meters. Finally, we remove all points that are not in the biggest cluster. Example of the result is shown in Fig. 6.

References


\cite{8} Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise.


\cite{12} Monique Paulich, Martin Schepers, Nina Rudigkeit, and G. Belluscì. Xsens MTw Awinda: Miniature Wireless Inertial-