1. Obtaining a human model for OpenSim

The first step in the pipeline is to obtain a human model for the OpenSim software. These models are available on the official OpenSim website, each designed with different characteristics. Some focus on the upper limbs, while others represent the entire body.

For our simulation-driven platform, we used the upper extremity model by Lee et al. [1], which specifically focuses on the upper limbs, including the hands. This model can be downloaded from this link. If you use it, please ensure you credit the authors and cite their work appropriately.

2. Obtaining motion files to animate the human model

The second step in the process is to animate the human model with motion data. To achieve this, we first need to obtain motion files. One approach is to use human pose estimation methods applied to video data. These methods generate as output a time series of 3D positions for key human body landmarks. In the case of the hand, these landmarks correspond to finger joints.

The `examples/interhand/raw_txt` folder contains several output samples generated using MediaPipe, Google's human pose estimation method, applied to the InterHand dataset. These files follow a 2D structure, where one dimension represents time and the other corresponds to different human body landmarks.

The first line of the text file contains column names, where the first column represents the timestep, and the remaining columns correspond to landmarks on the human body. For the next steps in the pipeline, it is crucial to know the precise locations of these landmarks on the body.

3. Preparing the motion files to be used in OpenSim

To animate the human model using motion files extracted from MediaPipe, inverse kinematics must be applied to them. The inverse kinematics (IK) module calculates a sequence of joint angles for the human model. These angles are determined such that, when applied to the model, they produce a motion that best matches the movement described by the input motion data. Essentially, IK ensures that the model's simulated motion accurately follows the tracked motion data from sources like MediaPipe.

The Inverse Kinematics (IK) module in OpenSim requires motion files to be in a specific format: the TRC (Track Row Column) format. This format ensures compatibility with OpenSim's processing pipeline. For detailed information on the structure and requirements of TRC files, refer to OpenSim's official documentation.

The `convert_to_trc.py` script converts MediaPipe-generated output (e.g., files in the `raw_txt` folder) into TRC files that are compatible with OpenSim. During the conversion, the script also renames the column headers for better clarity. The new column names are:

- **Thumb:** `"THUMB_TIP", "THUMB_IP", "THUMB_MCP", "THUMB_CMC"`

- **Index Finger:** `"INDEX_TIP", "INDEX_DIP", "INDEX_PIP", "INDEX_MCP"`
- **Middle Finger:** `"MIDDLE_TIP", "MIDDLE_DIP", "MIDDLE_PIP", "MIDDLE_MCP"`
- **Ring Finger:** `"RING_TIP", "RING_DIP", "RING_PIP", "RING_MCP"`
- **Pinky Finger:** `"PINKY_TIP", "PINKY_DIP", "PINKY_PIP", "PINKY_MCP"`
- **Wrist:** `"WRIST"`

Note that the conversion process also converts 3D positions from millimeters (as provided by MediaPipe) to meters for compatibility with OpenSim. This behavior is controlled by the variable `DATA_IN_MM`. When set to `True`, the script assumes that the input values are in millimeters and automatically converts them to meters. The TRC files are saved in the `raw_trc` folder after conversion.

4. **Adding markers to the OpenSim model**

Now, open the human model in OpenSim and add markers corresponding to the human body landmarks for which we have 3D position data (obtained from running pose estimation with MediaPipe). When creating these markers in OpenSim, it is crucial to use the same names as those used in the TRC files. More information on how to add and edit markers is available here. After adding the markers to the model, you can save the model to be utilized later.

The model located in the `models/modified` folder contains all the markers for the finger joints. Additionally, in this model, we have removed the muscles to simplify the subsequent inverse kinematics (IK) process.

5. **Adapting the TRC files for the model with added markers**

The final step before applying inverse kinematics is to modify the TRC files located in the `raw_trc` folder. This modification ensures the scale of the markers matches the model, and adapts the translation and orientation of the markers using the Kabsch algorithm. For more detailed information on this transformation, we recommend referring to our publication. Here's an excerpt from the publication:

The coordinate system of the extracted 3D hand poses does not match that of the virtual environment. More importantly, it may be arbitrary and vary between videos or even between time steps within a video. This arbitrariness arises due to the challenges of estimating 3D information from monocular videos. To address this, we apply translation and rotation transformations to the data before using it in the inverse kinematics module. Without these transformations, the inverse kinematics module would generate virtual motions that are significantly different from the real-world motions captured in the video.

In the translation step, the 3D coordinates in the motion data were moved so that the point of interest located on the wrist coincided with the one on the hand model. The final step —rotation— made use of the Kabsch algorithm [3] to find an optimal rotation matrix which, when applied to the motion data, aligned the hand that was made using the motion data with the hand model. The Kabsch algorithm expects two paired sets

of points as input. We utilized the points of interest in the motion data and their corresponding counterparts in the hand model as the required input. For the pre-processing, both strategies for hand motion data capture were used to provide coordinates for the inverse kinematics. That is, the output of the preprocessing step comprised refined motion files that contained 3D marker positions.

To apply the scaling, translation, and rotation transformations, the script `adapt_trc.py` is used. It is important to specify the folders where the raw TRC files and the model are located. Additionally, you must provide the save folder where the processed files will be stored. By default, the save folder is named `processed_trc`.

6. **Running the inverse kinematics**

To run inverse kinematics, use the `run_ik.py` script. In the script, you will need to specify the folder containing the processed TRC files (obtained from the previous step), as well as the location of the human model. The script will then generate MOT files, which contain a time series of joint angles for the OpenSim model. These MOT files will later be used to animate the model while collecting IMU data.

To visualize the MOT files, open the human model in OpenSim and click on Load Motion. This will load the corresponding motion into the model, which can then be replayed. The model should display the motion as it was seen in the original video used for MediaPipe pose estimation.

7. **Evaluating the result of the inverse kinematics**

The `evaluate_ik.py` script is designed to evaluate the results of the inverse kinematics process. As with the other scripts, ensure that you specify the correct file paths for all required inputs. Specifically, you will need to provide the paths to the generated MOT files, TRC files, and the human model. The output generated by this script will be in a format similar to the one shown below:

```
Avg. marker error [m]: 0.0056
Std. marker error [m]: 0.0041
Number of constraint violations: 19323
Constraint violation frequency: 0.1513
Avg. angle of constraint violation: 12.89°
Std. angle of constraint violation: 11.50°
```

You can use the results to filter out certain MOT files that result in large errors or a high number of constraint violation.

8. **Extracting acceleration, angular speeds, and orientations**

After generating the MOT files and filtering out those with significant errors, the next step is to extract accelerations, angular velocities, and orientations of the markers. This is achieved

by running the script `simulate_from_mot.py`. The script will play the MOT files provided as input with the human model in OpenSim (also provided as input) and capture the position, orientation (in terms of quaternion), and velocity data from the markers. It's important to note that the markers now function as IMU sensors, meaning you are not restricted to using the same markers as before—you can define an entirely different set. However, for simplicity, we choose to use the same markers as in the previous steps.

Note that the scale of the hand can also be modified since the MOT files contain joint angles rather than 3D positions (as is the case with TRC files). As described in our paper, we applied hand scaling procedures specifically to the fingers. You can adjust this scaling using the `scale_multiplier` variable in the script.

The output consists of `.CSV` files containing raw acceleration, angular velocity, and orientation data. In the next step, this raw data will be further processed to generate more realistic IMU simulations.

9. Simulating IMU sensors utilizing MATLAB

The CSV files generated in the previous steps serve as input for the `matlab_imu.py` script. This script utilizes the MATLAB library for Python to simulate sensor readings, including accelerometer, gyroscope, and magnetometer data. To install the MATLAB library for Python, access this link**.**

The output of the `matlab_imu.py` script is stored in a folder named `processed_simulations`. You can modify the save location by changing the `OUTPUT_FOLDER` variable in the script.

Additionally, the script extracts the filenames of the InterHand files to determine the activity being performed. Each activity is assigned a unique numerical ID, which is then used when saving the final simulated IMU files in the `processed_simulations` folder. This ID ensures proper classification and retrieval of activity-specific IMU data.

References

[1] Lee, J. H., Asakawa, D. S., Dennerlein, J. T. & Jindrich, D. L. Finger muscle attachments for an OpenSim upper-extremity model. PLOS ONE 10(4), 0121712 (2015).

[2] Moon, G., Yu, S.-I., Wen, H., Shiratori, T., Lee, K.M. Interhand2.6m: A dataset and baseline for 3d interacting hand pose estimation from a single rgb image. In: ECCV (2020).

[3] Kabsch, W. A solution for the best rotation to relate two sets of vectors. Acta Crystallogr. Sect. A 32(5), 922–923 (1976).