REAL-TIME 3D HUMAN POSE AND MOTION RECONSTRUCTION FROM MONOCULAR RGB VIDEOS

Anastasios Yiannakidis

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2019

APPROVAL PAGE

Bachelor of Science Thesis

REAL-TIME 3D HUMAN POSE AND MOTION RECONSTRUCTION FROM MONOCULAR RGB VIDEOS

Presented by

Anastasios Yiannakidis

Research Supervisor _____

Yiorgos Chrysanthou

University of Cyprus

June, 2019

ABSTRACT

Real-time 3D pose estimation is of high interest in interactive applications, virtual reality, activity recognition, but most importantly, in the growing gaming industry. In this thesis, we present a method that captures and reconstructs the 3D skeletal pose and motion articulation of multiple characters using a monocular RGB camera. This method deals with the challenging, but useful, task by taking advantage of the recent development in deep learning that allows 2D pose estimation of multiple characters, and the increasing availability of motion capture data. We fit 2D estimated poses, extracted from a single camera via OpenPose, with a 2D multi-view joint projections database that is associated with their 3D motion representations. We then retrieve the 3D body pose of the tracked character, ensuring throughout that the reconstructed movements are natural, satisfy the model constraints, are within a feasible set, and are temporally smooth without jitters. We demonstrate the performance of the method in several examples, including human locomotion, simultaneously capturing of multiple characters, and motion reconstruction from different camera views.

Anastasios Yiannakidis – University of Cyprus, 2019

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to my advisor prof. Yiorgos Chrysanthou for his guidance, for showing me trust and giving me the opportunity to work in this project.

I would like to thank my advisor Andreas Aristidou, for patiently mentoring me throughout my research. His enthusiasm, continuous guidance and support gave me motivation to work, to experiment and the will to learn.

Without the massive help and contribution of my advisors, the publication of the journal and this thesis wouldn't be possible. I am grateful for our collaboration and for welcoming me to the lab and letting me learn from diverse and interesting projects.

I am thankful to RISE for giving me the opportunity to work as an intern last summer. I would like to thank the people in the Graphics Lab for supporting me and helping me overcome numerous obstacles I faced throughout my research.

Last but not least, I would like to thank my family and friends for their encouragement and their spiritual support.

CREDITS

Journals

- **Anastasios Yiannakides**, Andreas Aristidou, Yiorgos Chrysanthou. 2019. Real-time 3D Human Pose and Motion Reconstruction from Monocular RGB Videos. Comput. Animat. Virtual Worlds. 30(3-4):1-10, doi:https://doi.org/10.1002/cav.1887

Presentations

- Real-time 3D Human Pose and Motion Reconstruction from Monocular RGB Videos. Presented at the Department of Computer Science, University of Cyprus, Nicosia, Cyprus, XX May, 2019

- **R**eal-time 3D Human Pose and Motion Reconstruction from Monocular RGB Videos. Will be presented at the 32nd International Conference on Computer Animation and Social Agents (CASA 2019), that will be held on July 1-3, 2019 in Paris, France.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Contents

1.1 Motion Capture Using Monocular Cameras

Motion capture (mocap) is the technological process used for acquiring three-dimensional (3D) position and orientation information of a moving object. Despite recent advances in motion capture technology, that allows high quality acquisition and portrayal of movements, common capture systems are cost-demanding, and require a setup of capturing devices that is not accessible to the general public, especially for home use. Recently, scholars turned their attention in using more affordable technologies, such as single RGB or depth cameras. Capturing and reconstructing the motion of multiple characters using single, monocular cameras has a wide spectrum of applications in surveillance, human-computer interaction, activity recognition, behavioral analysis, and virtual reality. It is also of high interest in the growing gaming industry, where users require cost effective, and easy configurable systems for real-time human-machine interaction. Real-time pose

estimation and 3D motion reconstruction using data only from a single RGB camera is, however, a challenging task. This is because skeletal pose estimation, using such sparse data, is a highly under-constrained problem. A human pose is usually represented and parameterized by a set of joint positions and rotations, whereas its heterogeneous, dynamic, and highly versatile nature, as well as the changes in camera viewpoint, makes motion reconstruction a difficult job.

Most papers in the literature estimate the human pose in 2D for one or multiple characters by localizing joint keypoints in pixel space [1, 2] or by extracting the shape silhouette and then retrieving the closest neighbor from a database [3, 4, 5]. More recently, and with the advent of Deep Learning (DL), the community is moving to learning-based discriminative methods, where the effectiveness of 2D human pose estimation has greatly improved [6, 7, 8]. The 3D skeletal reconstruction, though, is a much harder problem [9, 10]. Even though there are methods that are effective at 3D pose reconstruction, they are usually not real-time implementable, and suffer from depth, and scale ambiguities. In addition, the reconstructed motion is temporally inconsistent and unstable since they treat each frame independently, and do not employ bone lengths constraints. The big challenge in this context is to learn rich features to encode depth, spatial and temporal relation of the body parts so as to ensure smooth motion reconstruction [11, 12]. While some recent methods run at high-frame (e.g., [11, 13]), they are still unsuitable for use in closely interactive characters or crowds. This is because they require a tracked bounding box for each person, and thus, they can only reconstruct the motion of a single person at a time.

The novel contribution of this thesis, in principle, is that it fits 2D deep estimated poses, taken from a single, monocular camera, with the 2D multi-view joint projections of 3D motion data, to retrieve the 3D body pose of the tracked character. Our method takes advantage of the recent advances in deep and convolutional networks that allow 2D pose estimation of multiple characters, and the large (and increasing) availability of motion capture data. More particularly, our method

infers the 3D human poses in real-time using only data from a single video stream. To deal with the limitations of the prior work, such as the bone length constraints violations, the simultaneously capturing of multiple characters, and the temporal consistency of the reconstructed skeletons, we generate a database with numerous 2D projections by rotating, a small angle at a time, the yaw axis of 3D skeletons. Then, we match the input 2D poses (which are extracted from a single video stream using the OpenPose network [8]) with the projections on the database, and retrieve the best 3D skeleton pose that is temporally consistent to the skeleton of the previous frames, producing natural and smooth motion.

Our approach is capable of estimating the posture of multiple characters in real-time, while the reconstructed pose is always within a natural and feasible set. The performance of our method in reconstructing 3D articulated motion is demonstrated in several examples, including video streams taken from different points of view, and using multiple characters in the scene.

1.2 Method Overview

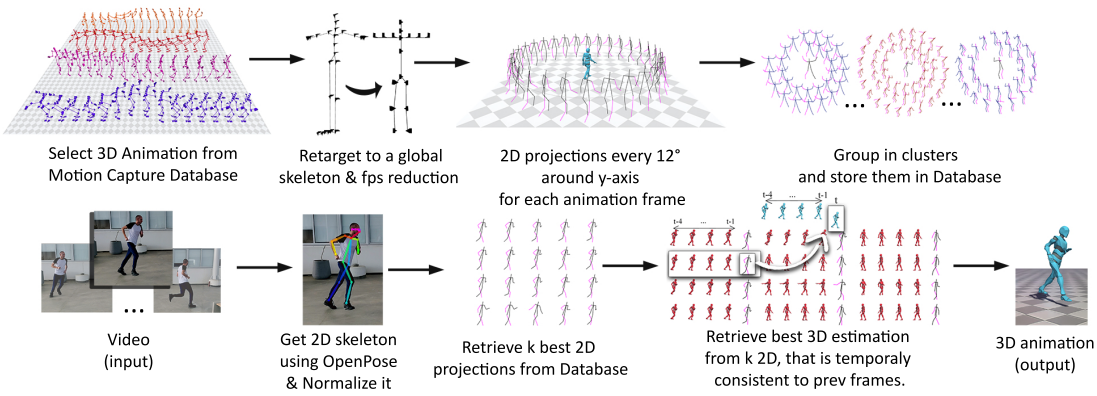

Figure 1: An overview of the proposed method.

Our approach for motion reconstruction can be decomposed into two main parts: (a) the pre-processing step, whereas a 2D pose database is defined by motion capture data projections, and its entries are associated with 3D skeletons, and (b) the run-time step, where 2D joint positions extracted from a video are matched to the 2D pose projections database in order to recover the 3D skeleton and reconstruct the motion.

More specifically, as a first step, we retarget motion data taken from an online motion capture database [14] into a universal skeleton format, and then compute the 2D projections of the 3D skeleton from many different views. The 2D projections are associated with their 3D skeletons and their viewing information, and stored in a database. The 2D projections are thereafter grouped into mutually exclusive clusters, and for each cluster, we allocate a representative pose. Then, in the second stage, for an input frame taken from a video stream, we extract in real-time its 2D pose using the OpenPose network [8]. The 2D pose is then rescaled so as to be consistent with the 2D projections stored in the database. We select the k-best matches from the database to the input 2D pose, and retrieve their 3D skeletons. To achieve smooth reconstruction of the articulated

motion, we select the 3D skeleton from the k-matched projections that is temporally consistent to the previous frame.

Chapter 2

Literature Review

Contents

2.1 Marker-based Systems

There has been a growing demand in recent years for realistic 3D animation in media and entertainment, as well as for research and training purposes. 3D motion acquisition can be roughly categorized as being achieved either using *marker-based*, or *marker-less* motion capture systems.

Marker-based systems are generally producing high-quality, realistic animations and are mainly used by the movies and entertainment industries. They use fiduciary markers which are attached near each joint to identify motion, and they provide real-time acquisition of labeled or algorithmically tracked data to reconstruct the subjects' articulated motion. More specifically, they utilize data captured from special markers (passive and active) to triangulate the 3D position of a subject inferred from a number of high speed cameras[15, 16]. The main strengths of optical systems are their high acquisition accuracy, speed, and high sample rate capture. However, optical

hardware is expensive, intrusive by nature, and lacks portability; calibration is needed for precise application of the markers, while extensive and tedious manual effort is required to recover the misapplied or (self-)occluded markers [17, 18].

More recently, a number of autonomous systems have been developed that use a variety of sensors, such as inertial measurement units (IMUs) [19, 20, 21]. Inertial mocap technology use a number of gyroscopes and accelerometers to measure rotational rates, while these rotations are translated to a skeleton model. Inertial systems do not require external cameras to capture the sensors, and thus, they are portable and functional in outdoor environments. Nevertheless, marker-based systems are costly and thus not suitable for home use, and more particularly, for interactive applications or gaming.

2.2 Marker-less Systems

In general, there is a tendency to reduce the required equipment and the objects attached to the body for motion tracking and reconstruction. One way to deal with sparse data is to model the trajectories of the end effectors, and then apply kinematics models to fill in the gaps [22, 23]. The most popular way, though, is the use of **marker-less** systems (or vision systems), that are becoming more and more popular since they are easy to set-up, have low cost, and are less intrusive, meaning that subjects are not required to wear special equipment for tracking. Usually, the subject's silhouette is captured from a single or multiple angles using a number of vision or RGB-depth cameras [24, 25]. A voxel representation of the body is extracted over time, while animation is achieved by fitting a skeleton into the 3D model, e.g., [26, 27, 28, 29, 30]. These approaches can be broadly classified into discriminative, generative and hybrid approaches. Generative methods reconstruct human pose by fitting a template model to the observed data [31, 32]. Discriminative methods infer mode-to-depth correspondences, cluster pixels to hypothesize body

joint positions, fit a model and then track the skeleton [33, 34]. Hybrid methods use a combination of the aforementioned techniques to achieve higher accuracy [35]. Other works also include methods for motion capturing using an architecture of multiple color-depth sensors to better deal with occlusions or unobserved view angles [36, 37].

Skeletal pose estimation from a single camera is a much harder problem; the problem has been tackled by localizing the joint keypoint positions in pixel space [1, 2], or by extracting the shape silhouette and then retrieving the closest neighbor from a database [3, 4, 5]. For a recent overview of 3D pose estimations, refer to [38].

More recently, deep learning has brought revolutionary advances in computer vision and graphics, including efficient methods for pose estimation. Learning-based discriminative methods, are quite popular for real-time 2D pose estimation of single or multiple characters [6, 7, 39, 8]. More recently, a number of methods tackled a much harder problem, that of 3D skeletal estimation from single color images or videos [9, 10, 13, 40]. A common limitation of the discriminative methods in 3D pose reconstruction, though, is that they typically run off-line, and since they do not take into consideration the temporal consistency of motion (they reconstruct the 3D joint positions on per image), the generated motion is oscillating. Moreover, they are restricted to the viewpoints learned from the training data [41], while they suffer from depth, and scale ambiguities. The problem of different camera-views can be dealt by training the network to predict the same pose using images from multiple views [42, 43]. Another major limitation is that the reconstructed 3D pose is not assigned on a character model (3D joint positions are estimated independently), to enforce kinematic constraints, resulting in temporal bone length violations. The methods in [11] (V-Nect), and [12] animate a modelled character, work in real-time, and produce smooth animations in terms of their temporal coherence. However, since they require each character to be tracked by a

bounding box, they only reconstruct single-person skeletons at a time, making them unsuitable for closely interacting characters.

Our method overcomes most of the prior work limitations, including the 3D capturing of multiple characters, the skeletal model constraints, and the production of smooth animation for the articulated character.

Chapter 3

Data Acquisition

Contents

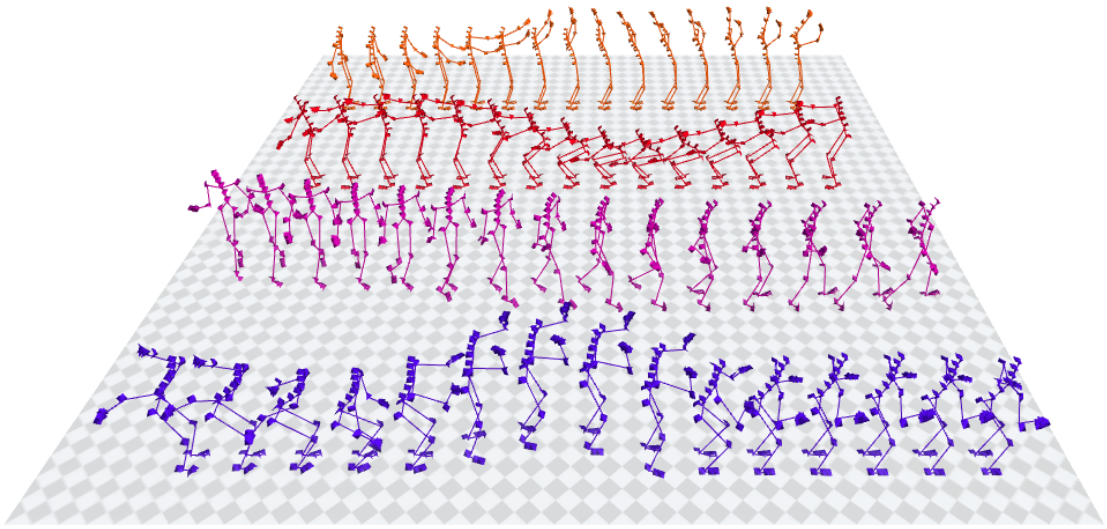

Figure 2: A number of skeleton sequences used in our dataset $\mathfrak{D}$, taken from the CMU motion capture library.

The first step of our method, in a preprocessing time, is to define a pose database that will be used to retrieve the 3D pose estimates. The database contains normalized 3D skeletons from motion data sequences, associated with their 2D projections from different views. In order to allow fast skeleton indexing and retrieval, the 2D projections are grouped into mutually exclusive clusters and each cluster is associated with its representative 2D projection.

3.1 Motion Capture Data

We used a small dataset $\mathfrak{D}$ of mocap data, taken from the Carnegie Mellon University (CMU) mocap library [14] (see Figure 2). CMU mocap library hosts a variety of mocap data (about 2,500 files), in the biovision hierarchy format (.bvh), in a consistent skeleton hierarchy as defined in Figure 3. The dataset that we used consists of animation that is highly related to our input.

In our experiments, we only used a small dataset of different actions, in total 5 minutes of motion ($72,000$ frames). Considering that our input consists of ordinary human motions such as

walking, running, jumping, squats and hand movements like arbitrary rotations and boxing, we selected animation frames mainly from 86 and some frames from 91, 111 and 127 CMU BVH directories. To avoid repetitiveness of the same motion, We manually selected (using Autodesk MotionBuilder) animation sequences, so we can achieve variation of motion in just 5 minutes of motion.

3.2 Biovision Hierarchy Format

A BVH file consists of two parts. The first part is the **header section** which defines the skeleton hierarchy. The definition is recursive, as each segment (joint) of the hierarchy defines its children and contains data relevant only to itself. Each segment contains the translational offset it has from its parent, and some channels that define the segments format. A channel can be either a rotational offset or a translational offset from the segment's parent. Usually, Root segment has 6 channels (Xpos, Ypos, Zpos, Zrot, Xrot, Yrot) and the other segments have 3 channels (Zrot, Xrot, Yrot). The second part is the **motion section** which indicates the number of frames, the sampling rate of the data and the actual motion data. The motion data is described by frames, and each frame is defined in a single line of numbers appeared in the order of the channel specifications as defined in the skeleton hierarchy.

3.3 Skeleton Retargeting & Fps Reduction

The human poses in the CMU motion capture data are represented by $n = 31$ joint positions and rotations (some joints overlap, see figure 4 for skeleton definition). In our work we use 2D poses that are estimated from an input monocular video using OpenPose [8] (see Section 4.1 for more details), that are represented by $m = 14$ joint locations as seen in the figure 11. Thus, in order to have a uniform and comparable skeleton, we retarget the CMU bvh data to a 3D skeleton

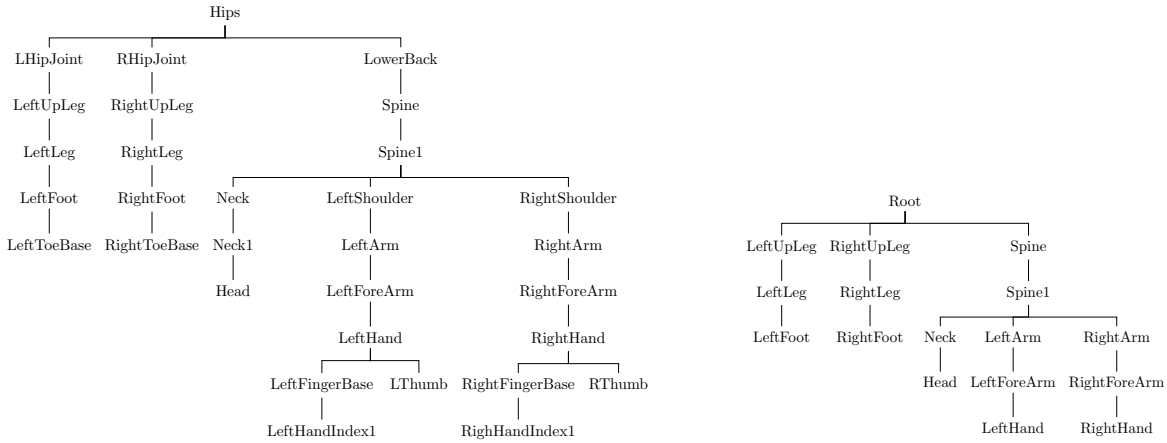

Figure 3: CMU and our BVH Skeleton Hierarchy.

that its projection in T-pose[1] matches the 2D skeleton (in T-pose) returned by OpenPose (see figure 5 for skeletons comparison).

The CMU motion capture data is originally sampled at 120 frames per second (fps), but since human motion is locally linear, and in order to reduce the computational cost, we resample it to 24 fps (using Autodesk Motion Builder) without much loss of the temporal information (see [44]).

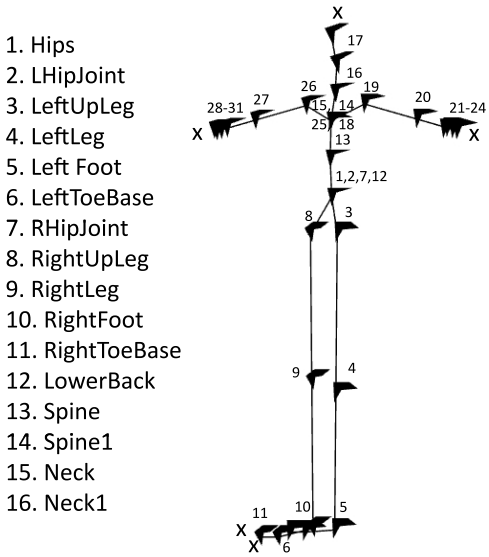

Figure 4: CMU BVH Skeleton illustration

[1]T-Pose is the default state of a 3D model described by stretched arms and legs so they form the letter T.

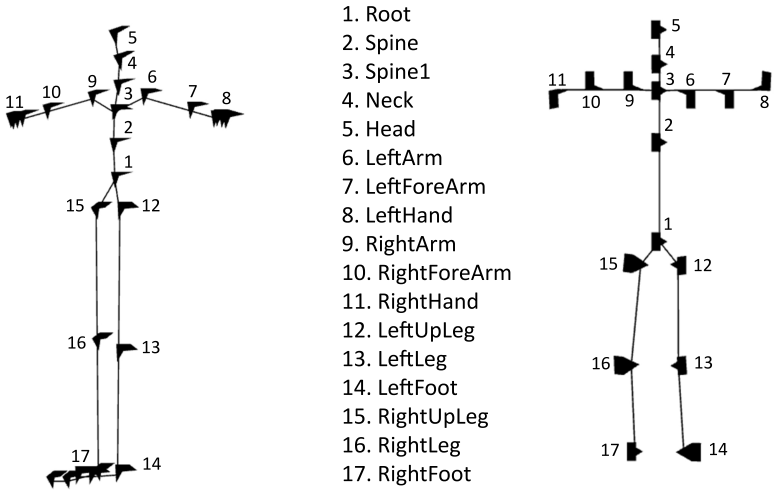

Figure 5: Retarget CMU skeleton to a simplified skeleton that matches the 2D skeleton returned by OpenPose.

3.4 Normalized 2D Projections

After the skeleton retargeting and fps reduction, we extract 2D normalized projections from each animation frame. The projections have the new skeleton (as defined in figure 5) but with the joints 1,2,4 being removed. We remove these joints because they do not give additional information about the posture, however they are necessary for the definition of the bvh skeleton hierarchy. As a result, the skeleton now matches exactly the skeleton of OpenPose skeleton (see figure 11 for OpenPose skeleton). The frame processing of each .bvh file, is implemented in C# in *Unity 3D*, using *BvhImporterExporter* from *Winterdust*, a library that allows us to import .bvh files in *Unity 3D* and get data about the joints in each frame. Thus, for each frame we get the 3D positions P of the joints and we reposition P so each joint is relative to the *Root*[2] and *Root* joint is positioned at $< 0, 0, 0 >$. In addition we scale P by a $ScalingFactor$ which is defined and discussed in Section 3.5.

$$P_{Root} = (P_{LeftUpLeg} + P_{RightUpLeg})/2;$$

[2]Root is the middle point of LeftUpLeg and RightUpLeg (see figure 11)

$$P_i = P_i - P_{Root};$$

$$P_i = P_i * ScalingFactor_i;$$

Then we create 2D Projections from different angles, by rotating the set P by an angle a around y-axis each time:

$$P_i = Quaternion.Euler(0, a, 0) * P_i;$$

Z-value is omitted (or equals to zero) when referring to the set *P'* as a 2D projection; however we keep the 3rd dimension (z) as it is useful to create (later on) articulated motion using the set *P'* as a set of 3D positions (see Appendix A).

Skeleton retargeting and scaling are crucial since we will compare the input to our database pixelwise the joint locations of the 2D skeletons joint-by-joint.

Projection Angle

In order to make our method invariant to the camera view, and robust against fine-grained pose variations, for each frame we compute the 2D projections of the 3D skeleton, rotated by an angle θ at a time ($\theta = 12^o$) on the yaw axis, resulting in total to 30 projections per frame. The main idea is to assign for each 3D pose multiple 2D projection representation from many different camera viewpoints, and is illustrated in Figure 6. Each 2D projection is associated with its initial 3D skeleton (pointed to its frame on the bvh animation), by indexing the corresponding frame on the animation, and its rotation angle. Note that, in this work, we place the camera at 1.5 meters above earth plane (common height for video recording), and similarly, we project the 3D skeletons assuming that the camera is at the same height.

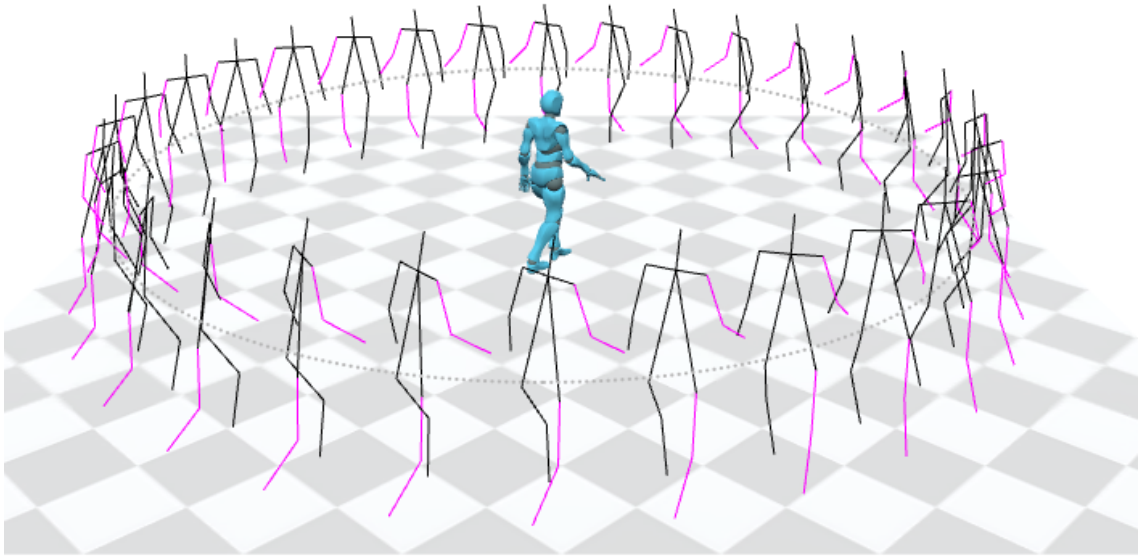

Figure 6: Multi-view skeleton projection. For each frame, we rotate the 3D skeleton by an angle θ at a time on the yaw axis, and create 2D projections.

3.5 Scaling 2D Figures

The same scaling method is used in normalizing both the 2D figures of the Database and the 2D figures of the input. We have examined two methods of calculating the scaling factor of a 2D figure: *Scale by bones length* and *Scale by bounding box*. The first method, scales a 2D projected figure to a global skeleton, using bones length; a bone is defined as the edge between two joints. We concluded that this method does not always guarantee correct scaling because bone length can be altered simultaneously when 2D projected. More specifically, this method may fail when the bone lengths in the torso area are altered. The second method is simpler as it scales all 2D projected figures so their bounding box remains constant; meaning that all figures will have the same height. We use this method as bones length alteration does not affect the reliability of the results.

3.5.1 Scaling using Bones Length

We are going to take a glimpse into this method because despite its disadvantage described in the introduction, it works in most cases. Particularly in this method, the scaling factor is calculated by the ratio of a bone of the figure and the corresponding bone of the global skeleton (eq. 1).

$$ScalingFactor(Bone_i) = \frac{Length(GlobalSkeleton, Bone_i)}{Length(Figure, Bone_i)} \tag{1}$$

In the equation we use the bone that its length has the least alteration when it has been projected. To determine which bone is the best to use, we test all bones that belongs to the torso area (see Algorithm 1). We omit the limbs because their length change more often when they are 2D projected. For each bone, we create a new figure that is scaled using the scaling factor from eq. 1. Finally, we select the scaling factor that scales a figure so the torso bone lengths have the minimum difference from the corresponding bones of the global skeleton.

Algorithm 1 Calculate Scaling Factor using best bone

Require: B, List of bones in Torso area
GS, Global Skeleton
F, Figure to be scaled

1: **procedure** GETBESTSCALINGFACTOR(F, GS)
2: **for each** $b \in \mathcal{B}$ **do**
3: $error = 0$
4: $sf \leftarrow ScalingFactor(b)$
5: $SF \leftarrow Scale(F, sf)$
6: **for each** $b \in \mathcal{B}$ **do**
7: $error = error+ \mid Length(SF, b) - Length(GS, b) \mid$
8: **end for**
9: $\Delta \leftarrow \langle error, sf \rangle$
10: **end for**
11: **return** $minError(\Delta)$
12: **end procedure**

To evaluate Algorithm 1 we compare the results of the algorithm to already known scaling factors of various figures. Particularly, we scaled a figure F by x units, that its 3D skeleton S

is known. Then, we compare the scaling factor that GETBESTSCALINGFACTOR(F,S) returns, to the x value. We run this test for $2,000$ various figures and we conclude that the bone defined by the Spine and Root joint (the point between RightUpLeg and LeftUpLeg, see figure 11) is always selected as the best bone to use in the calculation of the scaling factor, with the numerical difference from x being 0.00876 on average. Despite the fact that this method does not guarantee correct scaling due to the possibility of torso length alternation, it is effective in any other case.

3.5.2 Scaling using Bounding Box

This method is simpler; it scales the 2D pose projections so their bounding box remains constant over time. It ensures that the height, which is defined as the distance of the lowest point L and the highest point H of the figure, is the same for all 2D pose projections. More specifically, the scaling factor is the ratio of the height to a constant number X; meaning that all 2D pose projections will have height equal to X.

3.6 Data Clustering

To allow fast skeleton indexing and retrieval, we position the 2D pose projections into d-dimensional space using Multi-Dimensional Scaling (MDS) [45] and we apply the k-means algorithm to group the data into clusters. All steps are implemented in Matlab R2018b, using $mdscale()$ and $kmeans()$ built-in functions.

3.6.1 Dissimilarity Matrix and Multi-dimensional Scaling

First step of the clustering procedure, is to create a dissimilarity matrix that represents the correlations of our data (2D pose projections) and apply Multi-Dimensional Scaling (MDS) [45] for dimensionality reduction. We tried different dimensionalities, $\mathbb{R}^d$, and the quality of embedding seems to be equally well for $2 \leq d \leq 5$. The distance metric used to compute the distance between the 2D skeletal projections to create the dissimilarity matrix, is defined as:

$$dist_{ij} = \sum_{k=1}^{m} d\left(\mathbf{p}_k, \mathbf{q}_k\right), \tag{2}$$

where m is the number of joints, and $\mathbf{p}_k$, $\mathbf{q}_k \in \mathbb{R}^2$ are the joint positions of the two skeletons i and j. The term $d\left(\mathbf{p}_k, \mathbf{q}_k\right)$ represents the Euclidean norm between the two joint positions.

Since for each frame we extract 30 projections (rotated by an angle $\theta = 12^o$ on the yaw-axis), our 2D projection database $\mathfrak{D}$ consists in total with $216,000$ pose projections. Considering the computational complexity of MDS ($O(N3)$), in order to reduce the processing time and data usage, we randomly group our 2D projections set into smaller groups and apply the clustering procedure to each group separately. Thus, we grouped the $216,000$ projections into 10 groups.

3.6.2 K-Means

2D projection representations in $\mathbb{R}^d$ are then grouped into mutually exclusive clusters. We use the k-means clustering algorithm (Lloyd's algorithm) to create clusters that are separated by similar characteristics (see figure 8 for clustering visualization). For each cluster, we also define a representative pose that is the one closest to the centroid of the cluster. Figure 7 portrays the 2D poses of five selected clusters and their representative skeletons. As can be seen in Figure 7, similar 2D projections corresponds to many different 3D skeletons.

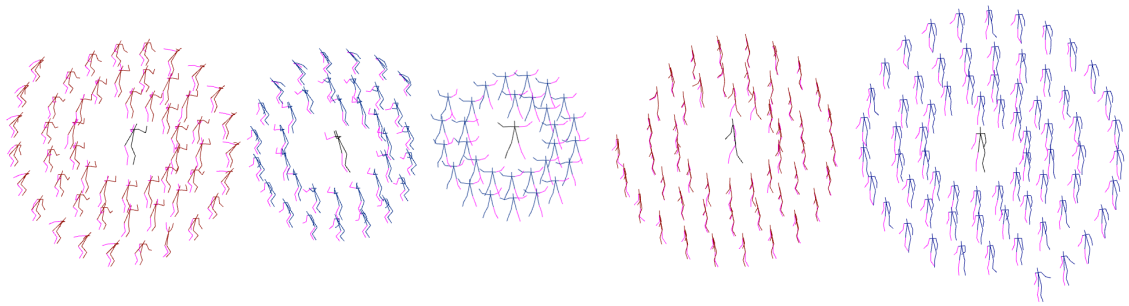

Figure 7: Five selected clusters with their representatives. The projected poses with the smallest distance to the centroid are placed nearest to the center, and as we move further from the center, the poses are less correlated to the centroid.

Selecting the right number of clusters for organizing $\mathfrak{D}$ is very important since it will allow fast indexing, and searching of poses, but it will also guarantee the quality of motion retrieval. To find the ideal number of clusters, we increased the number until the mean distance between the projections of each cluster and its centroid, do not change more than 1% (see Table 1 and Figure 9).

We need to acknowledge that the k-means algorithm does not always converge to the optimal solution, because the algorithm depends on the initial cluster centroid positions. As a result, some clusters may contain projections that are irrelevant to the representative. Let's take a deeper look into the algorithm and explain how to minimize this error. k-means as implemented in Matlab

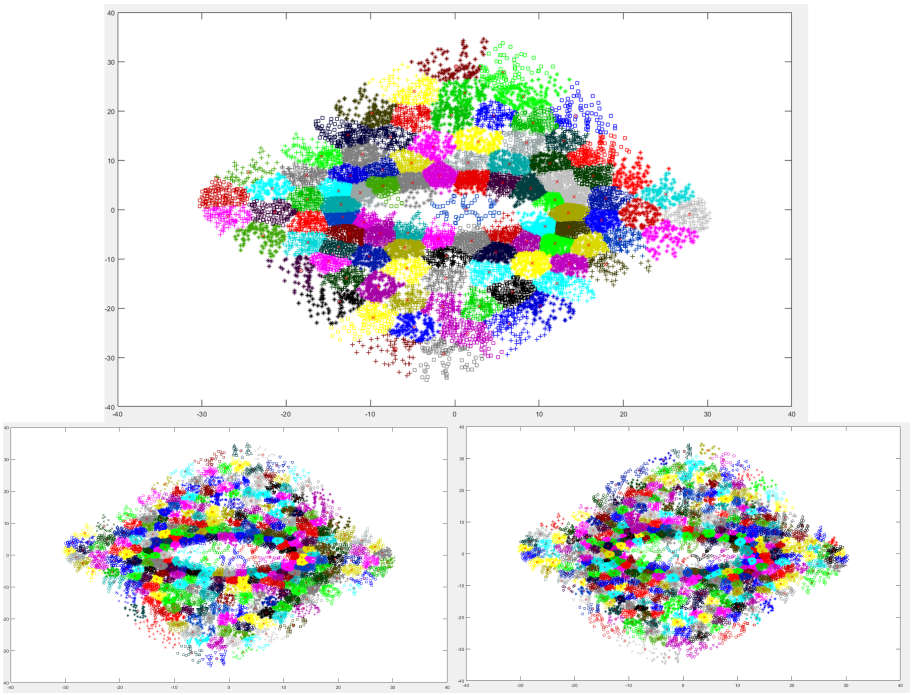

Figure 8: Visualization of Clusters: 100, 300, 400 clusters for the same sample of data. *Colors only demarcate a cluster from its neighbours*. The x-mark in each cluster represents the centroid of the cluster.

R2018b, uses a two-phase iterative algorithm to minimize the sum of point-to-centroid distances. The first phase is described as *batch updates*, where all points are assigned to their nearest cluster followed by recalculation of cluster centroids; this phase by its very nature does not guarantee to a local minimum solution. During the second phase, described as *online updates*, points are individually reassigned reducing the sum of point-to-centroid distances, and then the centroids are recomputed. Using the second phase, the algorithm converges to a local minimum, but undoubtedly there is no guarantee that this is the global minimum. In conclusion, in Matlab R2018b, in order to get satisfactory results we need to enable the *online updates* phase, set the *Replicates* option to repeat the clustering r times and get the solution with the local minimum. Executing the clustering multiple times is not time consuming if we execute them in parallel, by enabling the *Use Parallel* option which activate as many workers as we need, considering the available virtual cores of the machine.

Group id #	Number of Clusters	Mean distance from Centroid
1	490	0.6859
2	470	0.7226
3	440	0.6499
4	480	0.6574
5	440	0.6443
6	360	0.5372
7	400	0.5527
8	400	0.5897
9	500	0.7135
10	410	0.6089

Table 1: Indicates the number of clusters for each group of projections, when the mean distance between the projections of each cluster and its centroid do not change more than 1% while increasing the number of clusters.

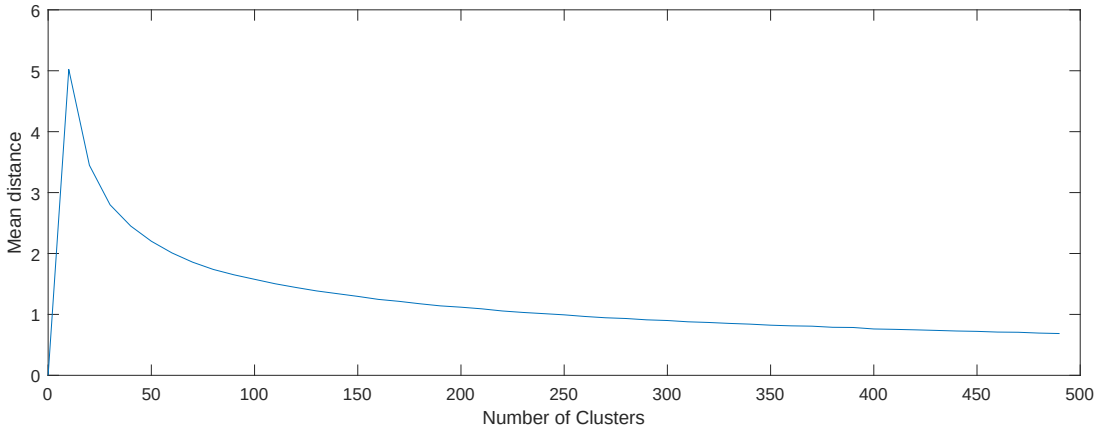

Figure 9: The rate of change of the mean distance between representative and projections is decreasing while the number of clusters is increasing.

Chapter 4

Motion Reconstruction

Contents

4.1 Input Processing

4.1.1 OpenPose

For an input video stream, we use OpenPose to estimate, in real-time, the 2D pose of the characters. OpenPose is a bottom-up approach that uses Part Affinity Fields, a set of 2D vector fields that encode the location and orientation of limbs over the image domain, offering robustness to early commitment. It offers state-of-the-art accuracy, it returns the 2D key points of the joints for multiple articulated characters at the same time, but most importantly, it decouples runtime complexity from the number of people in the image, achieving low computational cost.

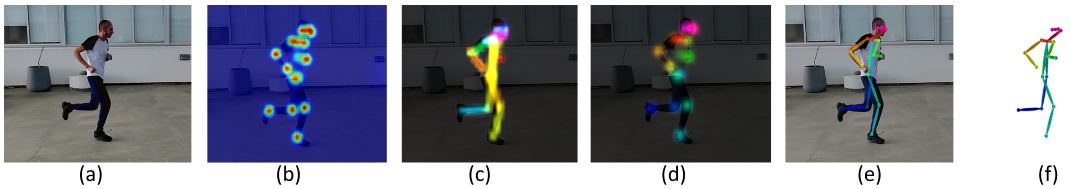

Figure 10: OpenPose pipeline: (a) Input image in the CNN which predicts the (b) confidence maps for body part detection and (c) PAFs for (d) body parts association. (e,f) The final result is the assembly of the body parts into a full pose.

In this project, we assume that the character's skeleton is fully visible at the camera, meaning that all 14 joint 2D positions are known. However, in a video sequence some joints may be unidentified due to occlusions, body parts may be out of frame or not stand out from the background. Future work will see the introduction of a pose recovery method (see section 6.2).

4.1.2 Scaling the Input

Since the 2D joint locations are inferred from a video, the size of the estimated pose is depended on the size of the tracked character (e.g., adult or child), and its depth (e.g., how far or

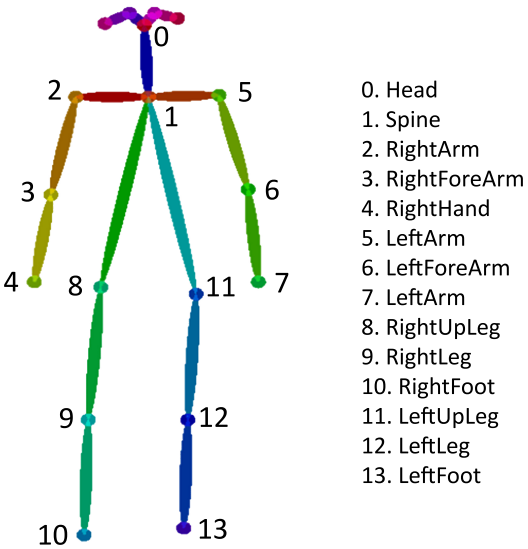

Figure 11: OpenPose skeleton example.

near is to the camera). To deal with this size ambiguity, and be consistent with the 2D projections stored in the database, we introduce a scaling step. For each OpenPose 2D pose, we export its bounding box and rescale it so as the height of the bounding box remains constant over time (same scaling method used in 2D projections, in section 3.4). Figure 12 shows a sequence of 2D poses of a walking character moving closer to the camera (the size of its pose increases over time), as inferred by OpenPose, and its rescaled pose that its size remains constant over time.

4.2 Retrieve Nearest 2D Poses

Having the rescaled 2D pose from OpenPose, we then need to search and retrieve the nearest pose to the entry from the projections database, and associate it with its corresponding 3D skeleton. However, one of the main challenges to deal with is that multiple 3D poses may correspond to the same 2D pose after projection. This introduces severe ambiguities in 3D pose estimation. Moreover, applying per-frame pose estimation on a motion sequence does not ensure temporal

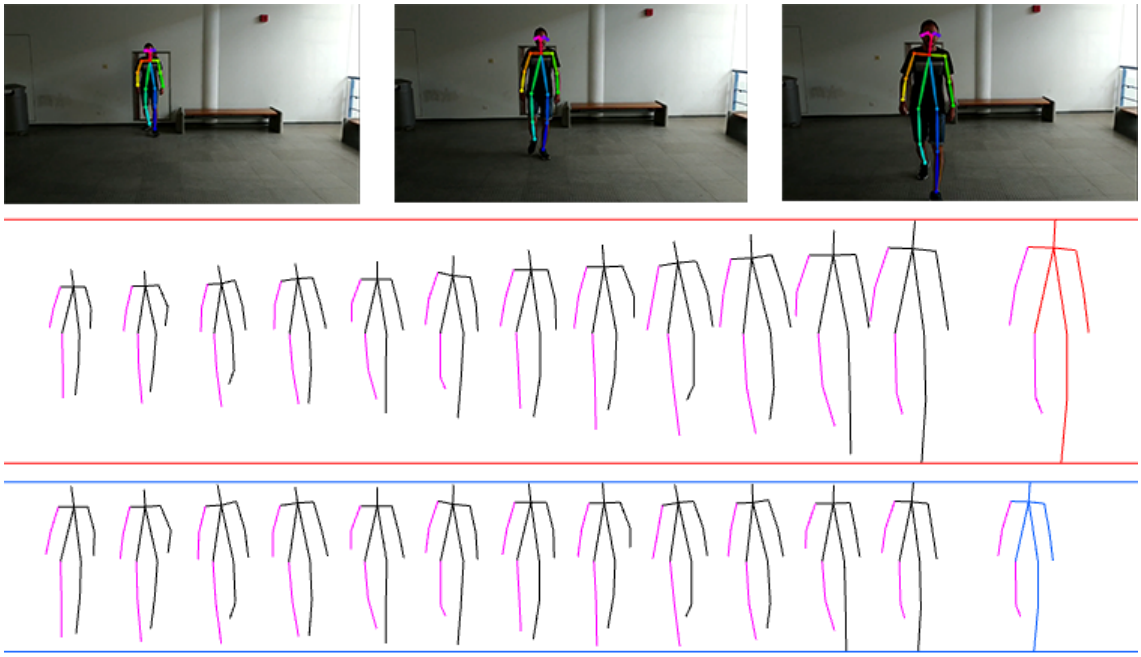

Figure 12: Scaling the OpenPose 2D pose estimation to ensure comparison consistency.

consistency of motion, and small pose inaccuracies lead to temporal jitter and motion oscillations. We dealt with these challenges in two steps.

First, we compare the entry pose with the cluster representations, and then select the i closest clusters with the smallest Euclidean distance (see equation 2) between the input pose and their representative. Note that, searching on a single cluster is not a good idea since there is no guarantee that the closest representative contains the closest projection for our entry. The number of clusters to select, should be at least the number of groups; a group is defined in section 3.6.2. The more clusters we search, the bigger the possibility is to find the best 2D match (at the expense of execution time). For each of those i clusters, we retrieve the j nearest poses projections using the same distance metric (equation 2). Figure 13 portrays j selected clusters (spiral shapes) and i nearest poses projections (blue figures) of an entry pose.

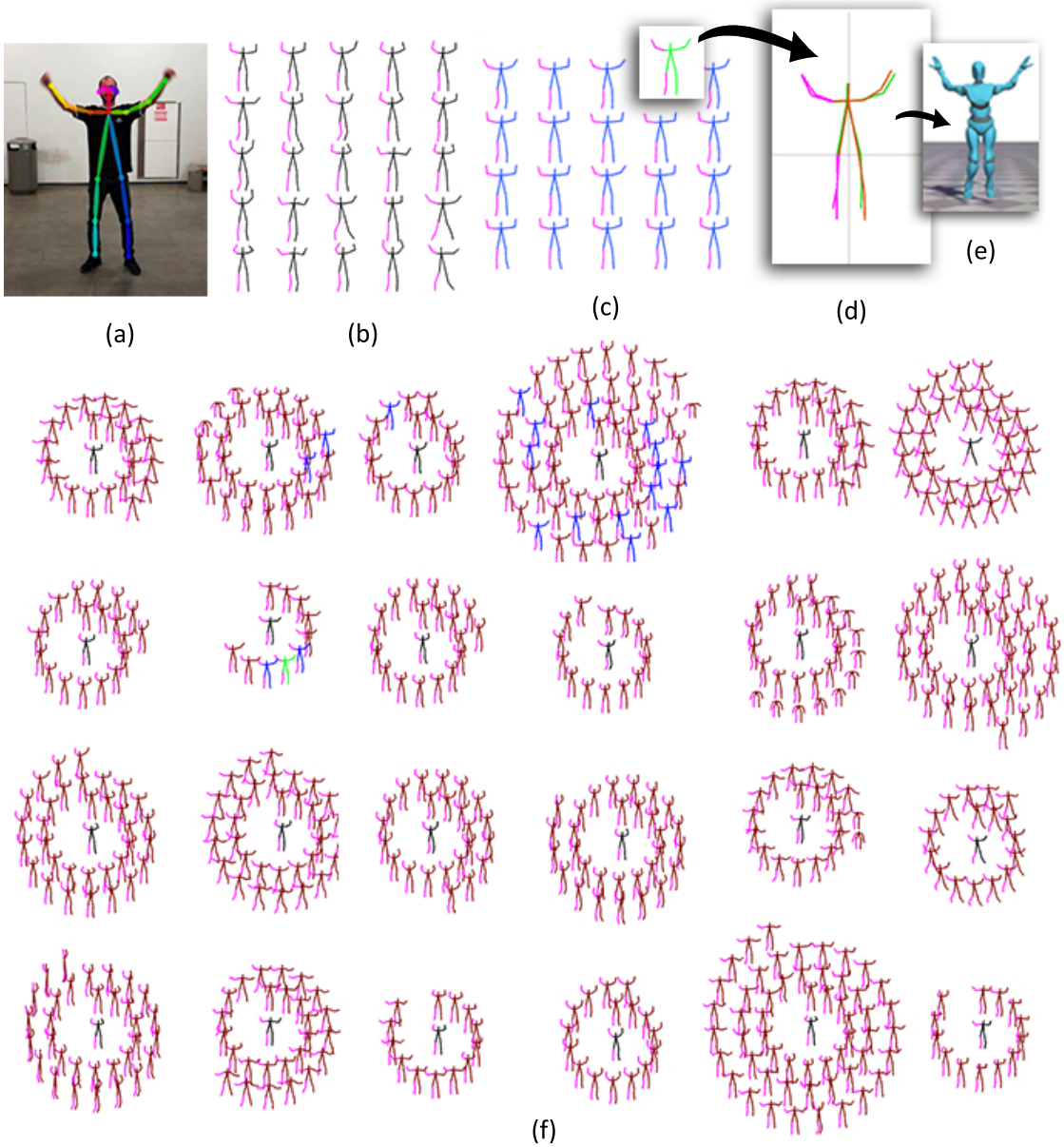

Figure 13: Illustration of nearest cluster representatives (b) and nearest projections (c) of input (a). The nearest clusters are illustrated in (f), where blue figures denote that they belong in the group of i nearest projections. The green figure visualizes the projection of the input's 3D estimation (e). Sub-figure (d) shows the green figure overlaid by the input.

4.3 Retrieve Temporally Consistent 3D Estimation

Then, by taking advantage of the property that motion is locally linear, we retrieve the 3D skeleton that is temporally more consistent to the reconstructed poses of the previous frames. More specifically, for each of the $l = i \times j$ selected nearest pose projections, we get their location to the bvh animation; recall that a projection is extracted from a 3D skeleton that comes from a specific frame in a bvh animation. Thereafter, we compare the 3D skeletons of their w previous frames, in the bvh animation, with the w previously reconstructed skeletons (each of the w skeletons is weighted differently), and select the one that its w previous frames have the smallest average distance (see Figure 14 for a visual explanation). The window length w is determined by the fps of the bvh files that we used in our database. We empirically conclude that $w = 4$ covers enough previous motion.

To compute the distances between the 3D skeletons, we use the Lee *et al.* [46] distance metric, that is the sum of the difference in rotation between joints. The distance between two skeletons is defined as:

$$dist_{ij}^2 = \sum_{k=1}^{m} \| \log \left(q_{j,k}^{-1} q_{i,k} \right) \|^2, \tag{3}$$

where m is the number of joints, and $q_{i,k}$, $q_{j,k} \in \mathbb{S}^3$ are the complex forms of the quaternion for the k-th joint of the two skeletons i and j, respectively. The log-norm term $\| \log \left(q_{j,k}^{-1} q_{i,k} \right) \|^2$ represents the geodesic norm in quaternion space, which yields the distance from $q_{i,k}$ to $q_{j,k}$ on $\mathbb{S}^3$.

The selection of the 3D estimation is strongly affected by the estimation of the previous frames. Thus, it is very possible that successive frames will have the same estimation, resulting in a frozen motion for an amount of frames as Figure 15 illustrates. To avoid that, we can put a *restriction* on the 3D estimation: reject the previous frame 3D estimation. This restriction assures that two

successive frames do not have the same 3D estimation, however it will cause more jitter that is more visible when the human figure (input) stands still or moves slowly.

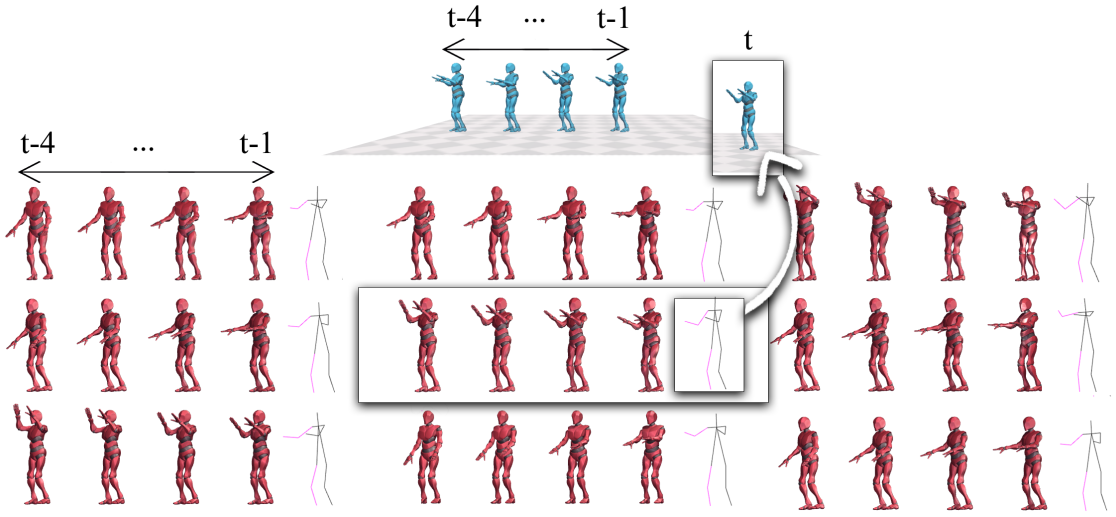

Figure 14: To ensure temporal consistency in motion, for each candidate skeleton, we compare its w previous frames in the original animation (shown in red) with the w previously reconstructed skeletons (shown in blue), and then select the candidate skeleton that its previous frames return the smallest average distance (highlighted box).

4.4 Save the 3D Estimation

At this point, we need to recall that 3D human motion is described by rotations of joints (and the translation of the root). Lets take a look into two approaches of saving a 3D motion sequence into files that can be used by 3D computer graphics software toolsets like Blender or MotionBuilder.

The **first** approach is to use the corresponding BVH rotations and adjust only the root rotation (rotate by θ deg around yaw-axis) to match the projection angle. Having the correct oriented BVH rotations for each frame, we can export the 3D motion sequence as a BVH file. In order to do that, we use the same skeleton hierarchy that we used for our database and we append the motion part which is the translation of the root and the joint rotations (refer to Section 3.2 for the structure of

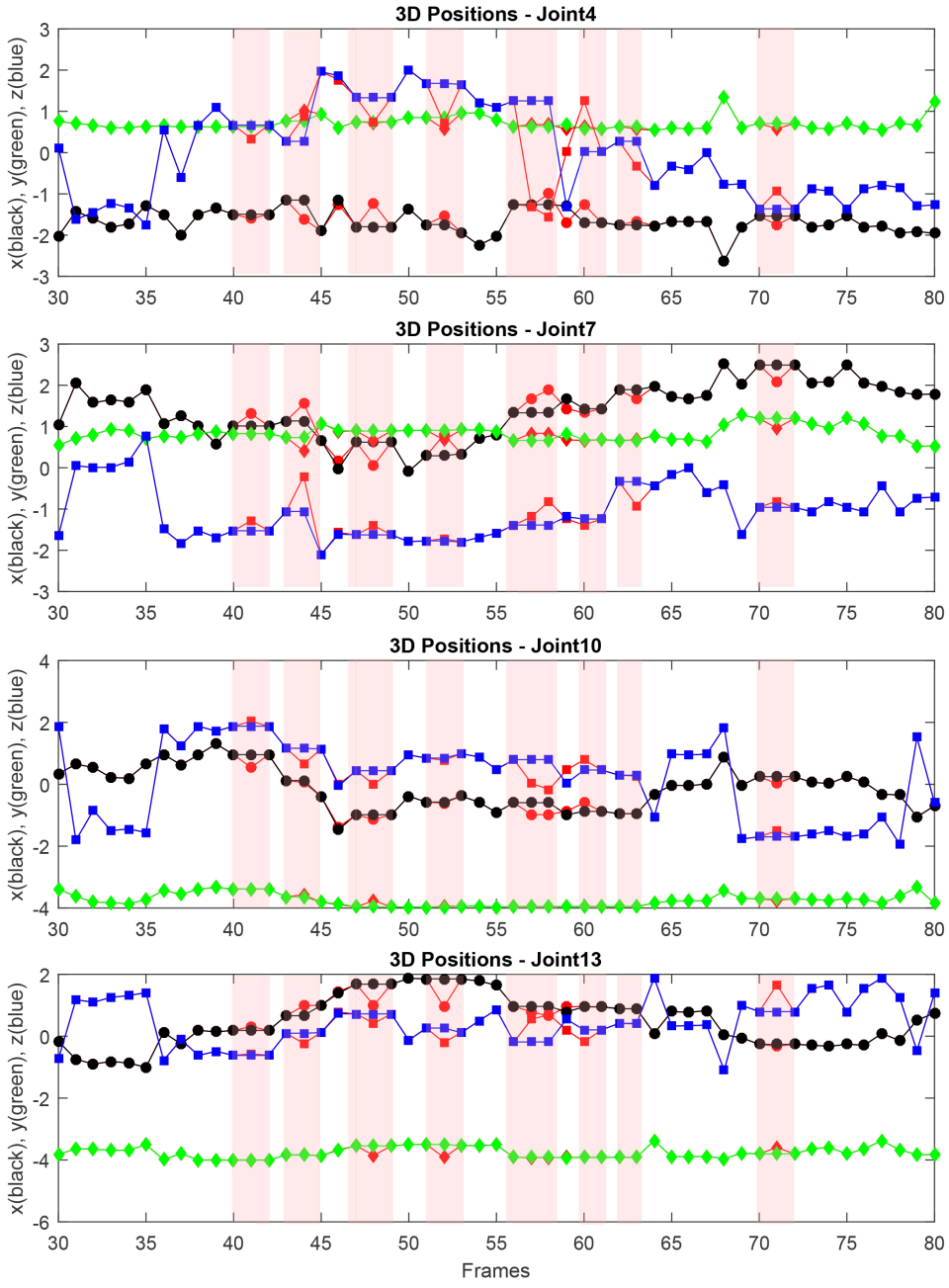

Figure 15: Comparison of raw-unfiltered 3D joint positions (RightHand, LeftHand, RightFoot, LeftFoot) with and without applying the *restriction*: "Rejecting the previous frame 3D Estimation".Blue-squares, green-diamonds and black-circles illustrate the x,y,z components respectively without applying the *restriction* and red points denote the difference in the estimation when the *restriction* is applied. Red areas mark the frames where motion is frozen when *restriction* is not applied.

a BVH file). This approach assure us that each animation frame has a realistic pose, as the joint rotational translation comes directly from recorded motion (BVH file).

The **second** approach is independent of the actual BVH rotations; all the joint rotations are recalculated for an articulated character, using the 3D joint positions of the 3D estimation (see figure 16). We can export the 3D motion sequence as an FBX file, recording the articulated motion each frame using the class *GameObjectRecorder* that belongs to *UnityEditor.Animations* namespace. Refer to Appendix A for further information on how to create articulated motion from a set of 3D positions. This approach is more generic as we can animate any skeleton (rigged model) we want, however it is exposed to animation artifacts such as unrealistic joint rotations and ambiguities in roll rotations of limbs and neck.

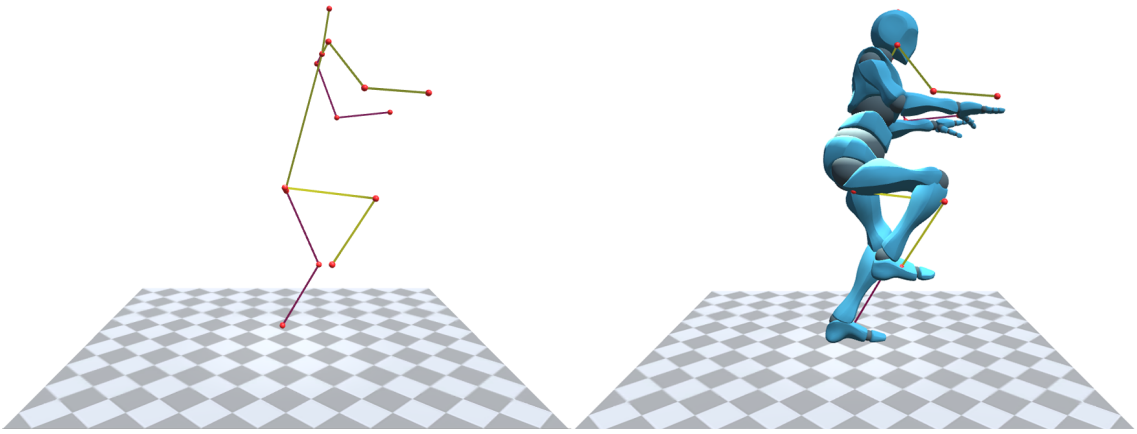

Figure 16: The 3D joint rotations are calculated using the available 3D joint positions.

4.5 Motion smoothing

Retrieving the pose that is more related to the previous frames will still cause some unwanted oscillation. Hence, we need to additionally smooth the reconstructed 3D motion using filtering methods, e.g., the SavitzkyGolay [47] or the 1 Euro filter [48]. The filtering can be either applied to the 3D joint positions or to the 3D joint rotations (in quaternions) of the estimation sequence.

Motion data can be further edited, again in real-time, using FBBIK [49] to avoid common synthesis artifacts, such as foot sliding and floor penetration.

4.5.1 1 € Filter

1 € Filter ("One Euro filter") is an algorithm that improves noisy signal quality. It uses a first order low-pass filter with an adjustable jitter and lag: when the speed is low, a low cutoff stabilizes the signal (reduces jitter) at the expense of latency, but at high speeds the cutoff is increased to reduce lag rather than jitter. This adaptive cutoff frequency suits to the nature of 3D animation: in low movement speed the jitter is more visible to the eye and we care less about the lag, but as the speed increases we become more sensitive to the lag and we mind less about the jitter.

In the first order low-pass filter that is being used, every time the frequency doubles, the signal amplitude is reduced by half. The equation 4 describes a first low-pass filter where X_i is the new raw data and $\hat{X}_i$ the filtered data at time i, while α being the smoothing factor $\in [0, 1]$. As α decreases, the jitter decreases but lag increases, as the first term of the equation denotes the contribution of new input data value and the second term denotes the previous values. Comparing to the standard n moving average filter based algorithms, the low-pass filter has less lag since the contribution of older values exponentially decreases.

$$\hat{X}_i = \alpha X_i + (1 - \alpha)\hat{X}_{i-1} \tag{4}$$

$$\alpha = \frac{1}{1 + f\frac{1}{2\pi f_c}} \tag{5}$$

$$f_c = f_{c_{min}} + \beta|\dot{\hat{X}}_i| \tag{6}$$

The algorithm adjusts the equation 4 so it considers the time intervals between samples. Parameter α can be now computed as a function of the sampling rate f and the cutoff frequency f_c (see eq. 5). The main challenge is to find the ideal f_c value (defined in eq. 6) which configures the trade off between lag and jitter. So, we need to increase the cutoff slope β if we want to minimize lag (in high speeds), and decrease the minimum cutoff frequency $f_{c_{min}}$ in order to minimize jitter(in low speeds). Proper initial values for these parameters are $\beta = 0$ and $f_{c_{min}} = 1$; note that even a very small increment in β value (for example by 0.007 may show some lag reduction).

The 1 € Filter is easy to understand and implement, and it is suitable for real time filtering as it uses only previous samples. However, the main downside is the constant need of adjusting the parameters β and $f_{c_{min}}$ in order to achieve the ideal trade-off between jitter and lag. We concluded that these values: $< \beta = 0.4,\ f_{cmin} = 1 >$, balance the trade-off, resulting in an acceptable real-time filtering (see 17 for results). The filter can be applied on the 3D positions or on the quaternions that describe the rotational movement applied on an articulated body.

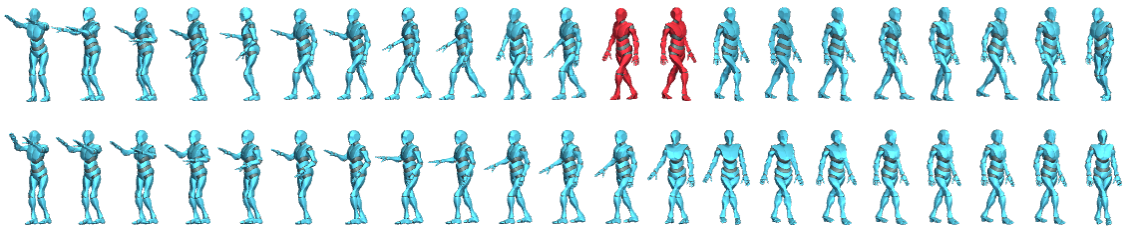

Figure 17: In Raw 3D estimation (first line), there is some visible discontinuation in movement especially in hands and the at the exact time when the root orientation is changing (red figures). In Filtered results using 1 € Filter (second line) the movement of the figure is much smoother.

4.5.2 Savitzky Golay Filter

Savitzky and Golay method was published in 1964, and their paper [50] was described as one of the best papers ever published in the *Analytical Chemistry* journal. This method is used

to smooth out a noisy signal while maintaining the tendency of the signal. It uses a convolution process that fits consecutive subsets of $2m + 1$ digital data points by a polynomial of degree p ($p \leq 2m$) based on linear-least squares (m is the half width of the approximation interval). Ronald W. Schafer in his article [51] gives a smooth and detailed explanation on the algorithm.

The main idea of the algorithm is to calculate the coefficients of a polynomial that minimize the mean-square approximation error for a set of digital data points ($2m + 1$: also referred as frame length) centered at k. The output value for the kth digital point is equal to the kth polynomial coefficient.

Savitzky Golay Filter might not be as simple as 1 € Filter, however its results are fairly reliable. We used the Matlab's built-in $sgolayfilt()$ function using as input the 3D joints positions, and we concluded that the filter works fine with polynomial order being the $3rd$ and the frame length being 11.

4.5.3 FBBIK

Our system lacks of obtaining the space translation, meaning that the output has a fixed root translation preventing the skeleton to move around in space. However, root translation feature is included in our future work, thus an IK solver is essential to be used in order to resolve any animation artifacts such as foot sliding, floor penetration and joint rotation abnormalities.

FBBIK (Full Body Biped Inverse Kinematics) is a system implemented for Unity Game Engine, designed specifically for bipeds that applies IK to an animated articulated body and it belongs to a general collection of inverse kinematics solutions named Final IK. The 3D model displayed in figure 16 is based on an articulated body model which is a tree set of rigid links connected to joints which can be rotated or translated. The leaves of that tree are called end effectors, which are basically the ending position of the most outboard links. In *Forward Kinematics*, the position

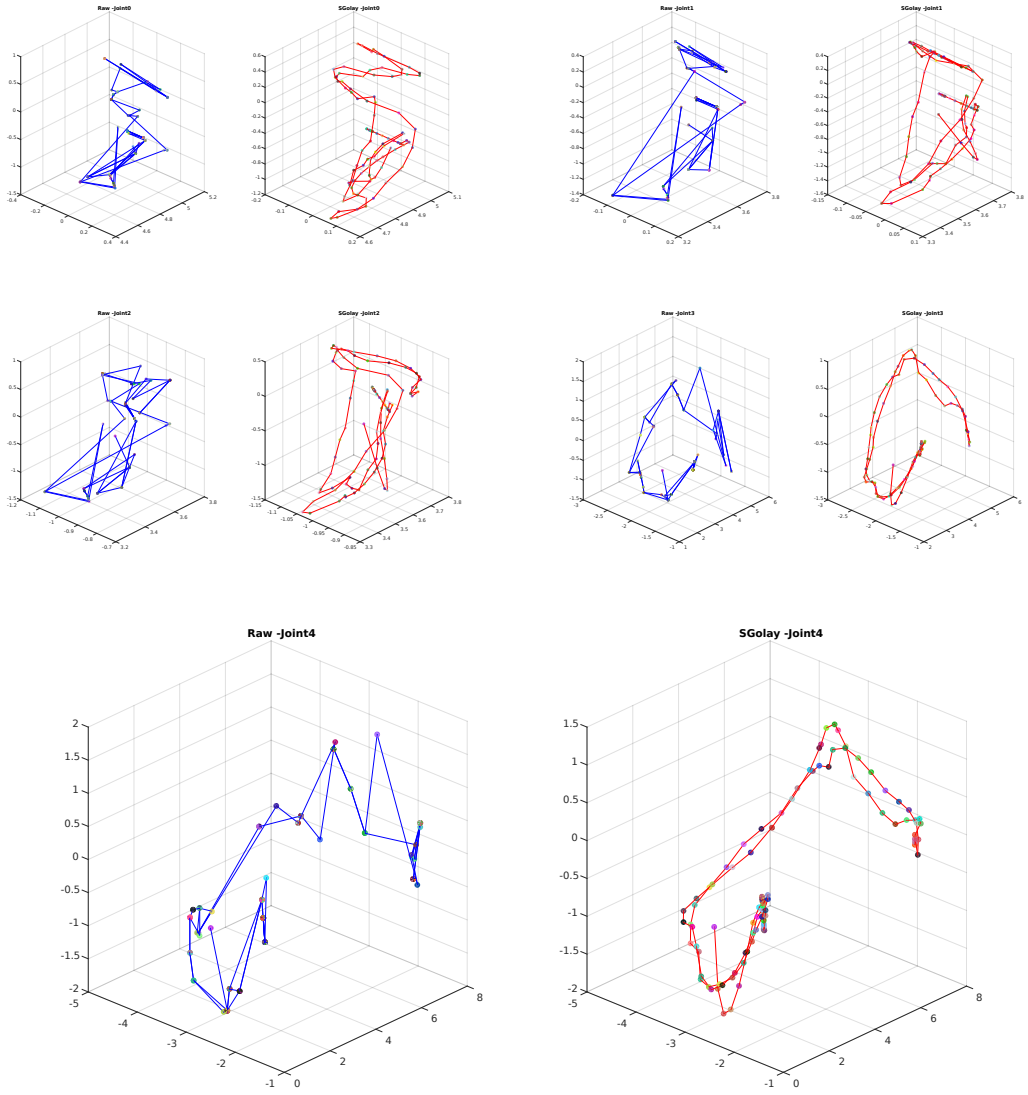

Figure 18: Illustration example of 3D positions for 100 frames, of joints 0 to 4 (Head, Neck, RightArm, RightForeArm, RightHand). The blue (left) graph illustrates the raw 3D estimation, and the red (right) graph illustrates the 3D estimation after applying S-Golay filter with 3rd order polynomial and frame length 11. Successively frames in the raw data may maintain the same (or similar) 3D position, but S-Golay filter will interpolate those points. Note that the filtered output maintains the number of points (frames) of the raw data.

of the end effectors are calculated when applying the definition of the whole pose (rotation and translation of all joints). However, in *Inverse Kinematics*, defining only the end effectors position results in the definition of the whole body pose.

The applications of IK increase the more creative you get. Generally, IK can be used in configuring the pose when an end effector interacts with a target, for example when arms are reaching, following, or grabbing an object. It can also be used to correctly place feet on a surface avoiding the artifacts of floating feet, floor penetrations and feet sliding. By just defining the ground, the joints can be configured in resulting in a convincing stepping cycle.

What FBBIK actually does, is mapping the biped character for each frame in LateUpdate[1] to a low resolution IK rig with effectors, then configures the final pose and applies the result to the character. This setup uses three types of effectors. The end-effectors (hands and feet) that can be rotated changing the direction of the limb, affecting the mid-body effectors (shoulders and thighs). In addition, multi-effectors are attached to torso area and can be dragged along with the other effectors defining the final pose. In addition, each limb and body are defined as chains that have the ”pull”, ”reach” and ”push” property. ”Reach” property adjusts the update rate of the IK solver and the other two define how easily we can pull or push the character using the effector. These property values are weights between the chains, for example if right foot ”pull” weight equals to 1 and other chains equals to 0, we can pull the character from right foot without being restricted by the other body parts.

Final IK collection among other IK solutions offers the Grounder system. This system is using FBBIK solver, to place the feet on the ground minimizing any artifacts due to the feet and ground interaction. Grounder uses raycasting to find the ground height from each foot and the character’s root’s y-position, and then pulls the feet to the ground if needed and optionally bends the spine. Minimizing feet jitter along with Grounder system will result in a smoothing stepping cycle.

[1]LateUpdate is a function in Unity declared in MonoBehaviour class, that is called every frame after all Update functions have been called and is completely independent from the animating system. Other Update functions are Update() and FixedUpdate().

Chapter 5

Results and Evaluation

Contents

5.1 Motion Examples Results

We conducted several experiments to evaluate the performance of our method. Figure 19 visualizes the 3D pose reconstruction on several motion examples which are included in our Database, such as walking, boxing, jumping, and running. Unlike VNect [11], our method is able to 3D estimate multiple, closely interactive characters, as Figure 19 illustrates. Figure 20 shows the effectiveness of our method in tracking and capturing motion that is temporally coherent. It can be observed that the 3D skeletons of the articulated motion are smoothly reconstructed over time.

5.2 Evaluation using Virtual Video

To quantitatively evaluate the performance of our method, we animated a .bvh file using a virtual character (see Figure 21) using Autodesk MotionBuilder. We then video capture the animated character, and input the video in our system to get the 3D estimation. The virtual character used in the video should be human-like, in order to have satisfactory results from OpenPose; using the robot character we observed weak results. We computed the difference between the poses of the original and reconstructed characters; the average Euclidean distance between their joints is only 10 cm (for a character with height 175 cm).

5.3 Evaluate Input in Different Angles

We further evaluated the effectiveness of our method in pose reconstruction by recording the same action from two different viewing angles, and then reconstructing their 3D motion. We computed the average Euclidean distance between the two 3D skeletons per joint (as Figure 23 illustrates) for recordings with the angle view difference being approximately $135\,\mathrm{deg}$ and $75\,\mathrm{deg}$. The average distance without applying any filtering to our results is 12.64 cm for $135\,\mathrm{deg}$ and 9.3 cm for $75\,\mathrm{deg}$. With filtering, the average distance is 9.4 cm and 6.8 cm respectively. We compared our results to VNect [11] and the corresponding distance for the filtered VNect reconstruction is 16.4 cm and 15.2 cm, respectively. Note that, for the comparison between the two poses, we discard the translation and rotation of the root joint so as to fairly compare the two 3D poses. Figure 22 illustrates the two 3D poses from different angles.

5.4 Implementation Tools and Resources

We have implemented our main system and GUI in the Unity game engine (Unity 2018.3.9f1) using $C\#$. We have used Matlab R2018b to cluster 2D projections and to use SavitzkyGolay [47] Filter built-in function. All experiments were run on a six-core PC with Intel i7-6850K at 3.6GHz, 32GB RAM, and nVIDIA Titan XP GPU.

5.5 Live Input Implementation

The system has been extended to allow live 3D motion reconstruction using a webcam. The 2D poses (skeletons) are estimated live using the OpenPose Unity Plugin and each incoming frame is processed, displaying the results in real-time as Figure 24 illustrates (using the method *From 3D positions to Articulated Motion* described in Appendix A). The bottleneck of the method (using the database as described in section 3) is the OpenPose Unity Plugin that process frames in 15fps on average. However, in every *Unity Update* that is over 60fps, the 3D motion reconstruction frames are interpolated using the 1 € Filter making the 3D animation play smoothly in higher fps.

Figure 19: 3D skeleton reconstruction of various different frames for (a) single character, and (b) multiple interactive characters in the scene. The RGB video frames are shown on the left side of the figure, the selected 2D projections in the middle, and the 3D skeleton reconstruction on the right.

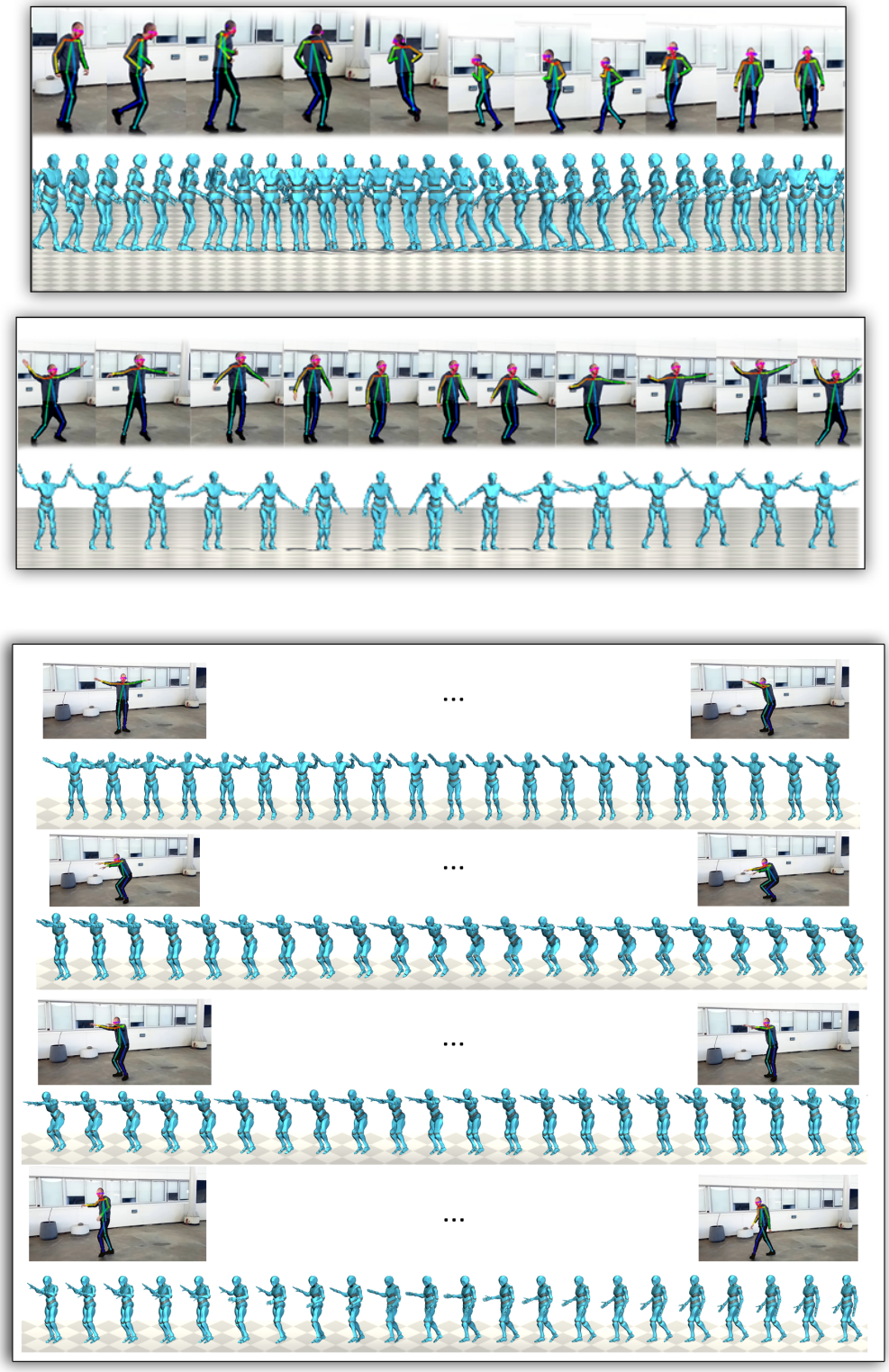

Figure 20: Smooth motion reconstruction in a sequence of skeletons.

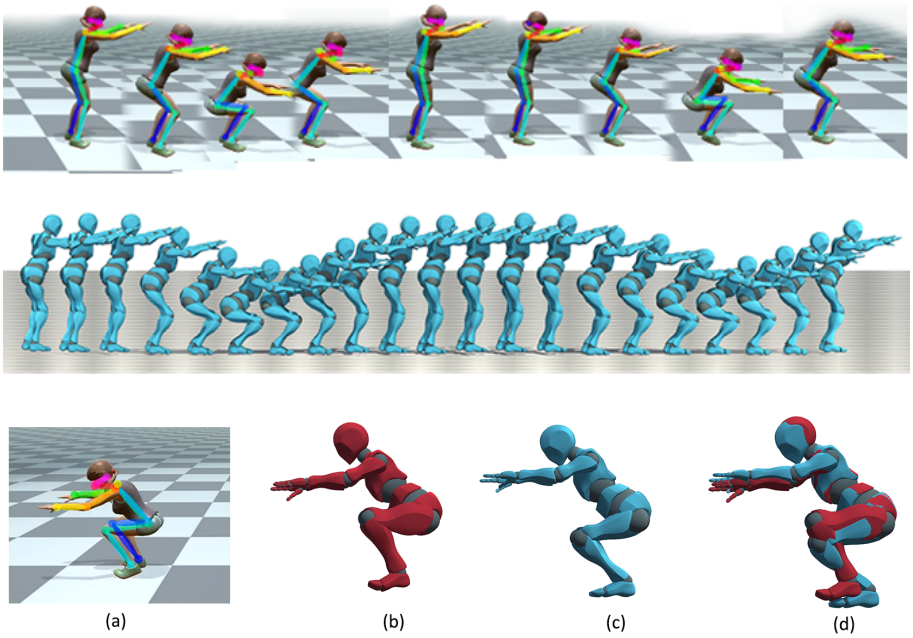

Figure 21: 3D motion reconstruction on virtually animated motion capture data. The output of our method (c) is being compared to the motion capture data (b) used in the video.

Figure 22: 3D pose reconstruction of the same action recorded from different angles. The arrow shows the same pose, but rotated, to be aligned and visually comparable with the pose from the other view.

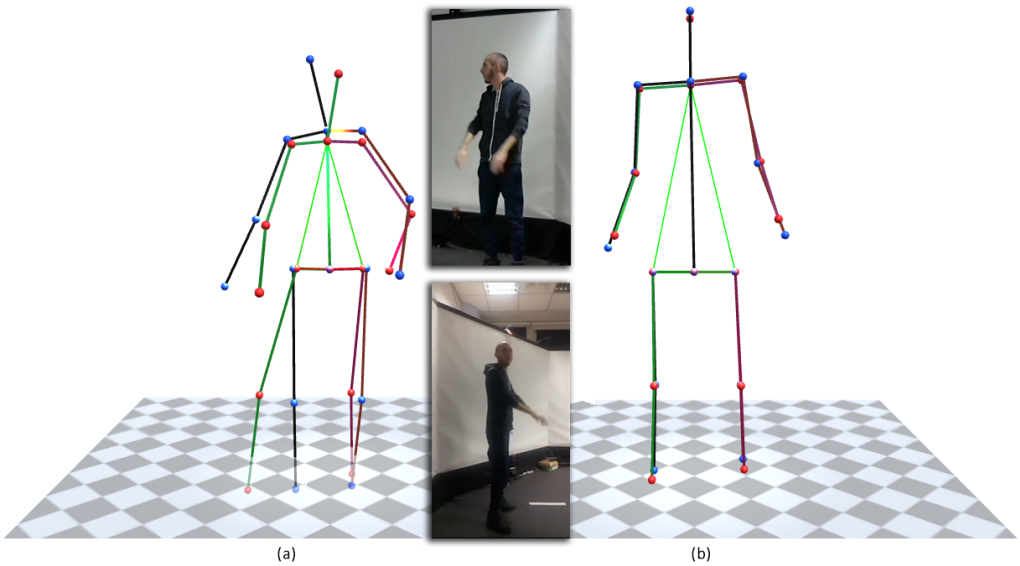

Figure 23: Illustration of comparing 3D pose estimation from two different angles (approx. $135\,\mathrm{deg}$ difference) using VNect (a) and using our method (b). The root translation and rotation of the skeleton of each angle *(skeleton1: red spheres - green lines, skeleton2: blue spheres - black lines, and the reddish lines denote the skeleton's left side)* has been discarded, so the skeletons are aligned are ready to be compared.

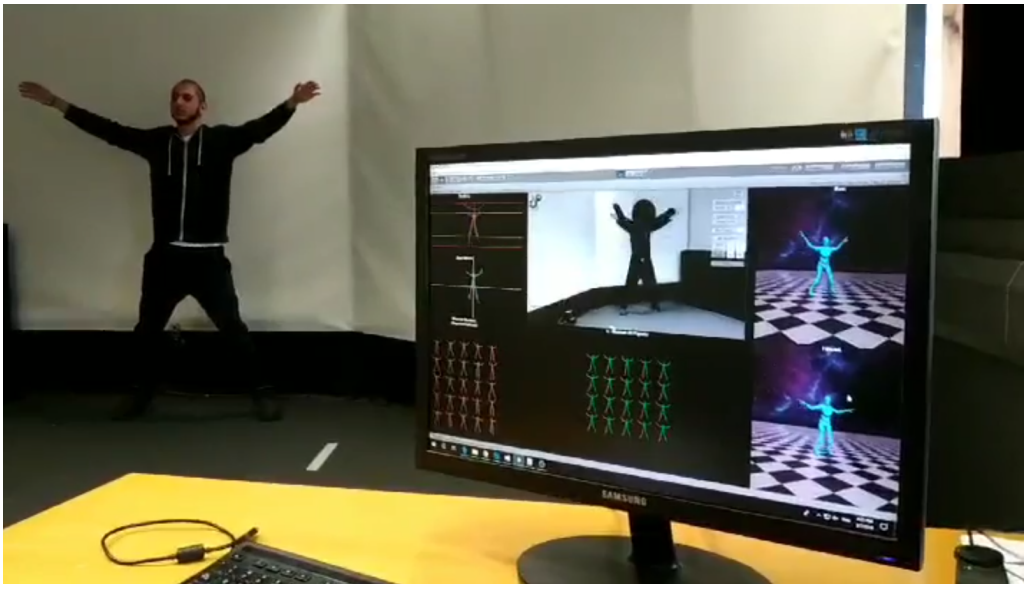

Figure 24: Real Time 3D motion reconstruction.

Chapter 6

Conclusions, Limitations and Future Work

Contents

6.1 Conclusions

We have presented a method that estimates, in real-time, the 3D pose of a human character using a single, monocular camera, and reconstructs its articulated motion. Our method is capable of tracking and reconstructing multiple 3D human postures at the same time, ensuring that the estimated 3D poses satisfy the character's bone constraints, and are always within a natural and feasible set. Common prior work limitations were efficiently dealt, such as the temporal consistency of poses, and the production of smooth and linear motion. We have evaluated the performance of our method in several examples, including a large variety of locomotion with single

and multiple characters in the scene, on data taken from different points of view, artificially generated data, etc.; our results which have been compared with a state-of-the-art method (VNect), demonstrate the efficiency of the proposed approach.

6.2 Limitations and future work

Our method has some limitations. First, the accuracy of the 2D joint estimation (using OpenPose) can greatly affect our method's 3D estimation performance. There are two ways to overcome this limitation: (a) one way is to use a weighted distance metric and enforce less influence (see eq. 3) on faulty joint positions, or (b) to use an architecture of two or more cameras for a better 2D pose estimation. Moreover, in our work we assume that the characters skeleton is fully visible at the camera; however, this is not always possible due to occlusions from other characters or objects in the scene. Future work will see the introduction of a pose recovery method that selects the most appropriate skeleton, from the database, and matches with the sparse data.

A second limitation lies on the camera's projection angle, which must match the view angle of the camera. A possible way to make our system invariant to different camera viewpoints is to further extend the database with additional projections on different axis, different height (view), or camera parameters. Learning the camera viewpoint to improve 3D pose estimation is currently an active and challenging topic in computer vision [52].

Another limitation of our method is that the results are highly related to the training data. A larger variety of movements will increase the variety of reconstructed actions, but at the same time, it will increase the computational cost. Finally, since there is no calibration of the scene, our method is not trained to account for global translation and rotation. In future work, scene calibration and root translations will be added.

6.3 Publications

During this project, we have achieved the following publication, and it will be presented at the 32nd International Conference on Computer Animation and Social Agents (CASA 2019), that will be held on July 1-3, 2019 in Paris, France.

- **Anastasios Yiannakides**, Andreas Aristidou, Yiorgos Chrysanthou. 2019. Real-time 3D Human Pose and Motion Reconstruction from Monocular RGB Videos. Comput. Animat. Virtual Worlds. 30(3-4):1-10, doi:https://doi.org/10.1002/cav.1887

Appendix A

From 3D Joint Positions to Articulated Motion

Contents

A.1 Introduction

In this guide, we are going through a process that reconstructs articulated motion from a set of 3D joint positions. A common straightforward definition of a 3D pose is a set of 3D world-positions that represent the joints, which are connected by rigid links. Calculating the rotations using oversimplified motion information is a challenging task due to possible ambiguities and non-realistic joint rotations (Figure 27). Ambiguities can arise due to the limited number of known 3D joint positions, most commonly on the roll rotation of the limbs and head; in this project we also face severe ambiguities in hands and feet because we discarded the information of fingers and toes

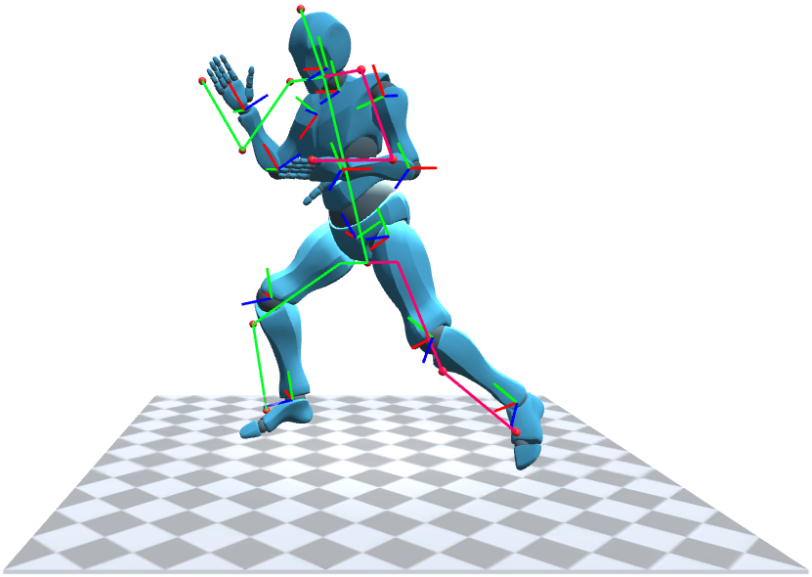

Figure 25: From 3D Positions to Articulated Motion.

in order to simplify the skeleton. Another challenge is to avoid any abnormal rotations that can cause penetration of body parts or over-rotation of joints. This guide promises to be as generic as possible in terms of mapping articulated motion to any rigged 3D model. Any code shown is in $C\#$ implemented for $Unity2018.3.9f1$.

A.2 Different Skeleton Configurations

Rigged humanoid 3D models may differ on their skeleton definition: joint hierarchy and initial rotations of joints may be different. Some skeletons may have a more complicated joint hierarchy than others; for example a complete set of joints for each finger and toe. However, a proper defined humanoid skeleton should at least consist of the basic joints as described in the hierarchy (right) in Figure 3 and illustrated in figure 25. We assume that we have set of 3D positions P for each animation frame, for an immutable set of joints. Using Breadth-first search we can find the joints that correspond to the set of known 3D positions P; we save these joints in an indexed list J, so we can apply any transformations on them directly.

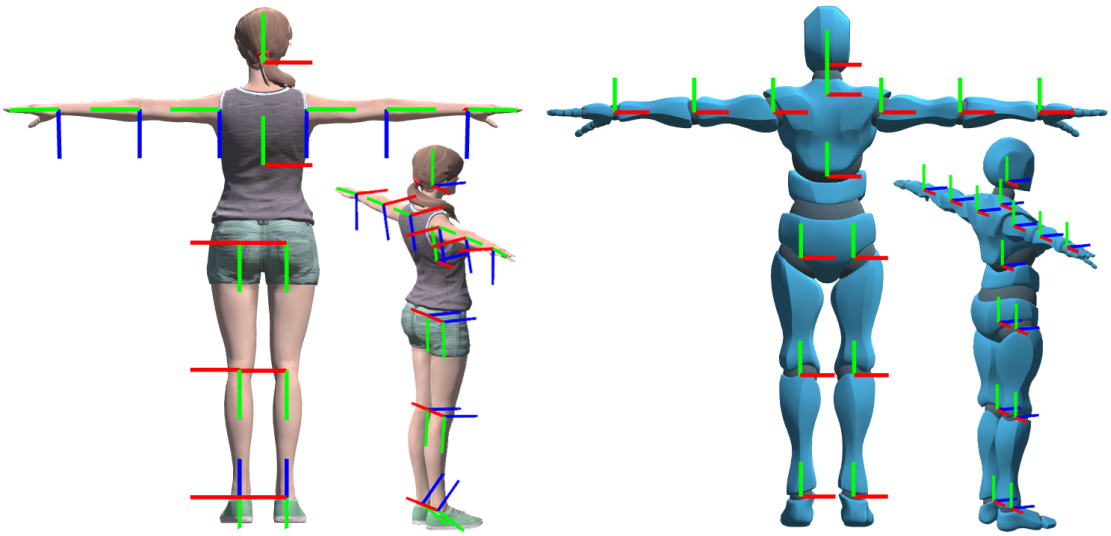

Figure 26: Two examples of rigged 3D models (in T-Pose) that show variation in local axes orientation. The Female model (left) has its local axes defined in a more complicated way than the Robot model (right) which all joints happens to be the oriented the same (with zero local rotation). The figure illustrates the positive x,y,z axes in red, green and blue respectively. Axes x,y,z are also called Right, Up and Forward respectively.

Our goal is to calculate the rotations of the joints J for a set of 3D positions P that describes an animation frame. At first it is important to define the initial local axes orientation of each joint by the T-Pose of the skeleton. $Unity$ uses a left-handed coordinate system which means that *right* is the positive direction of x-axis, *up* is the positive direction of y-axis and *forward* is the positive direction of z-axis; this coordinate system defines the global axes (world space axes) which happen to match the local axes orientation of the Robot's joints, as seen in Figure 26. Thus, we first are going to configure the rotations of the Robot model, and then extend the method to a more generic approach.

A.3 Calculating Rotational Transformations

The green stick figure in Figure 25 is composed by the set P connected by rigid links. Figure 28 illustrates an example of $RightForeArm$ link which is defined by 3D positions a and b.

This link is actually a vector $\vec{V}$ that is equal to $(a-b)$: a vector that starts from point b and ends on point a. Now, we need to calculate the rotation of that joint so its orientation matches the direction of $\vec{V}$. Observing its local axes (Figure 26) we decide that we should align the positive x-axis (red) with the $\vec{V}$, as Figure 28 shows. In the same way we determine for each joint an axis $\vec{A}$ to be aligned with the corresponding vector $\vec{V}$.

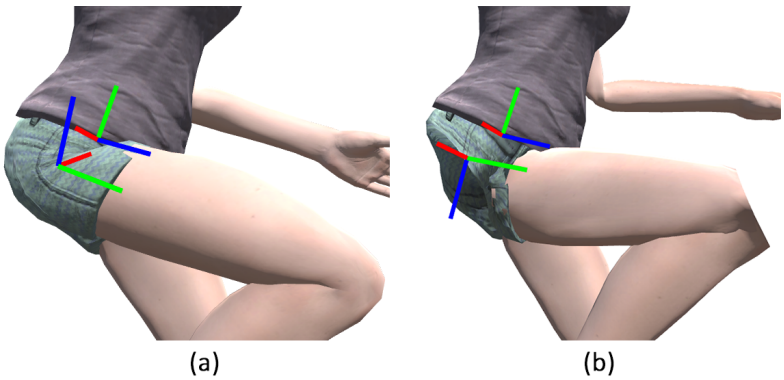

Figure 27: Unrealistic joint rotation artifact (b).

Rotating just $\vec{A}$ to the direction of $\vec{V}$ is not enough because the direction of the other two axes is still undefined; which can cause unrealistic rotations (see Figure 27). Thus, we need to define the direction of a second axis that will look towards the direction of $\vec{U}$; the direction of $\vec{U}$ will define the roll rotation of the joint. In order to be consistent to our formula, we first calculate the rotation $R1$ that points the axis $\vec{A}$ to the global *Forward*-axis; so $\vec{A}$ becomes the z-axis.

$$R1 = Quaternion.FromToRotation(\vec{A}, Vector3.forward);$$

Then we should point $\vec{A}$ to the direction of $\vec{V}$ with y-axis pointing towards $\vec{U}$. Hence, we calculate $R2$ whose z-axis is aligned with $\vec{V}$, x-axis is aligned with cross product between $\vec{V}$ and $\vec{U}$, and y-axis is aligned with cross product between z and x.

$$R2 = Quaternion.LookRotation(\vec{V}, \vec{U});$$

Finally, the product $R = R2 * R1$ is the rotation that should be applied on the joint. Note that the rotations should be applied using a pre-order traversal visit in the joint hierarchy, meaning that the parent should be transformed before transforming its child.

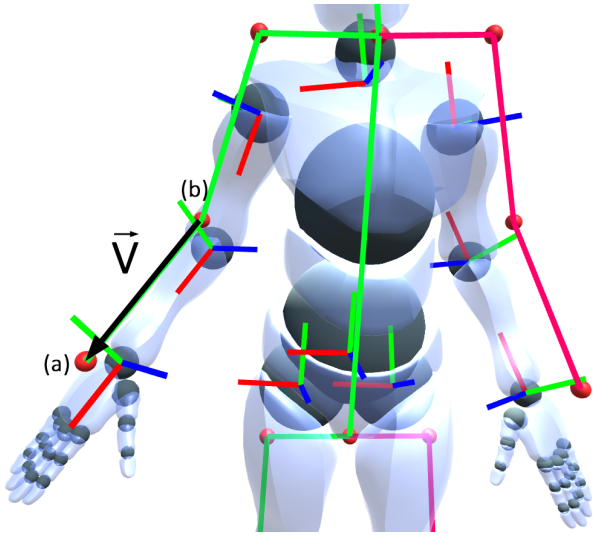

Figure 28: $RightForeArm$'s local axes are rotated so the positive x-axis (red) is aligned with a vector $\vec{V}$ defined by two 3D Positions a and b. Red spheres represent a set of 3D positions connected with links.

A.4 A Generic Approach

As we already pointed out, the joints of a rigged model in T-Pose might have already some initial local rotation, that changes the orientation of the axes, just like in Female model in Figure 25); making the selection of the axis $\vec{A}$ and defining $\vec{U}$ a confusing task. Thus, we need to map the global axes to the local axes, so when we request the *forward*-axis, we get the actual axis that looks at global *forward*. Using the same example (Figure 26), in *RightUpLeg* or in *LeftUpLeg*, the global *up*-axis is actually equal to the local *down*-axis (negatve y-axis). We can automate the axes matches by calculating the smallest angle distance between a local axis and a global axis; this process is done only one time, using the T-Pose.

Let the types of global axes be *T* = *Up, Right, Fwd, Down, Left, Back*

$$localAxis_{WorldSpace} = transform.TransformDirection(globalAxis_i);$$

$$Distance = Mathf.Abs(Vector3.Angle(localAxis_{WorldSpace}, globalAxis_j));$$

We save the direction of each axis type in an array D, which can be indexed by T. Let a function $getActualAxis(t)$ to return the actual axis that looks at $t \in T$. At this point, we can calculate $R = R2 * R1$ with $R1$ defined as:

$$\vec{A} = D[getActualAxis(t)];$$

$$R1 = Quaternion.FromToRotation(\vec{A}, D[Fwd]);$$

Table 2 shows the direction of $\vec{V}$ and the selected axis $\vec{A}$ for each joint of the list J defined as seen in Figure 29. It is important to point out that we don't really have enough information (considering the limited set of 3D points P) to define $\vec{U}$ which decides the roll rotation. So, for a joint j we can approximate this rotation by using current hips' $TransformDirection(D[getActualAxis(Fwd)])$ only if the *Up*-axes of hips and j match, else $TransformDirection(D[getActualAxis(Back)])$. For example, in Female model in Figure 26, $\vec{U}_{RightLeg}$ equals to $hips.transform.TransformDirection(< 0, 0, -1 >)$ as the *Up*-axes of hips and *RightLeg* are looking at the opposite direction. Now, for the *Hips* joint, there *is* enough information to calculate its $\vec{U}$ direction; two more positions $p[1] \in P$ and $p[14] \in P$ will determine $\vec{U} = p[1] - p[14]$ (see Figure 29 for set P).

A.5 Post Production and Conclusions

In this appendix we have discussed how to create articulated motion using a limited set of 3D positions P. This limitation as we already discussed, causes ambiguities especially in roll rotations of limbs and neck. A future work can be the application of a more sophisticated way of estimating the roll rotation. However, we still can fix any abnormal rotations, by applying angle limitations on the joints. Note that our result from *Unity* can be saved as an *AnimationClip* using a *GameObjectRecorder* instance and then export the animated 3D model as an FBX animation file

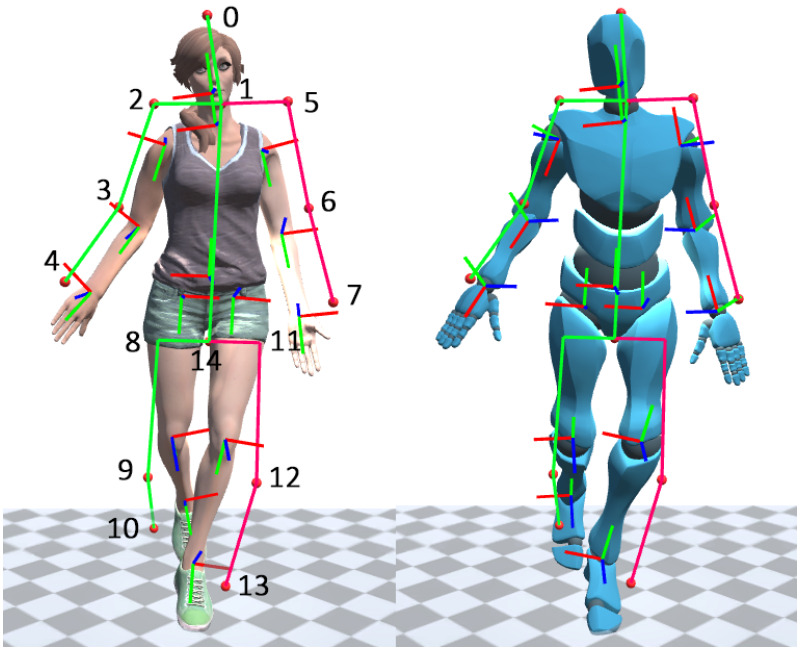

Figure 29: The set of 3D positions P (0...14) are used to calculate the rotations of joints J. $\vec{A}$ axis that is aligned with $\vec{V}$ ($p_i - p_j, p_{i,j} \in P$) may vary between different rigged models. Positive local x,y,z axes are illustrated in red, green and blue respectively.

Joint	$\vec{A_{global}}$	$\vec{V}$
0-Head	-	-
1-Neck	Up	$0-1$
2-RightArm	Right	$3-2$
3-RightForeArm	Right	$4-3$
4-RightHand	-	-
5-LeftArm	Right	$5-6$
6-LeftForeArm	Right	$6-7$
7-LeftHand	-	-
8-RightUpLeg	Up	$8-9$
9-RightLeg	Up	$9-10$
10-RightFoot	-	-
11-LeftUpLeg	Up	$11-12$
12-RightLeg	Up	$12-13$
13-RightFoot	-	-
14-Hips	Right	$8-11$

Table 2: For each joint, we define the axis to be aligned with $\vec{V}$ ($p_i - p_j, p_{i,j} \in P$). $\vec{A_{global}}$ can be mapped to the local $\vec{A}$ which is used in the calculations of the rotations.

using the *FBX Exporter* plug-in. Thus, we can further edit the FBX in a 3D character animation software like *MotionBuilder*, in order to achieve the results we want.

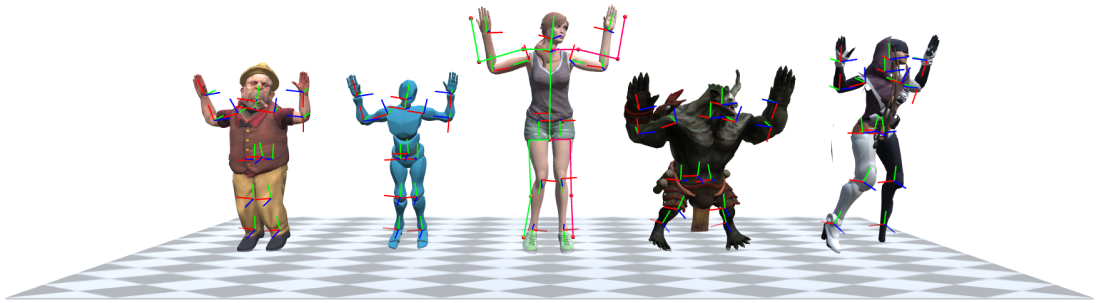

Figure 30: From the same set P we create articulated motion to different rigged 3d models showing respect to their unique skeleton definition.

Bibliography

[1] L. Bourdev, M. Müller, G. Bharaj, H.-P. Seidel, and C. Theobalt. Poselets: Body part detectors trained using 3d human pose annotations. In *Proc. of ICCV'09*, pages 1365–1372. IEEE CS, 2009.

[2] P. F. Felzenszwalb, . B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[3] C. Wang, Y. Wang, Z. Lin, A. L. Yuille, and W. Gao. Robust estimation of 3d human poses from a single image. In *Proc. of CVPR'14*, pages 2369–2376, DC, USA, 2014. IEEE CS.

[4] I. Akhter and M. J. Black. Pose-conditioned joint angle limits for 3d human pose reconstruction. In *Proc. of CVPR '16*, DC, USA, 2016. IEEE CS.

[5] F. D. Atrevi, D. Vivet, F. Duculty, and B. Emile. 3d human poses estimation from a single 2d silhouette. In *Proc. of VISAPP'16 - Volume 4*, pages 361–369, 2016.

[6] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *Proc. of ECCV '16*, 2016.

[7] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *Proc. of CVPR'16*, pages 4724–4732, DC, USA, 2016. IEEE CS.

[8] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. In *Proc. of CVPR '18*, DC, USA, 2018. IEEE CS.

[9] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black. Keep it SMPL: Automatic estimation of 3d human pose and shape from a single image. In *Proc. of ECCV '17*, 2017.

[10] G. Pavlakos, L. Zhu, X. Zhou, and K. Daniilidis. Learning to estimate 3d human pose and shape from a single color image. In *Proc. of CVPR'18*, DC, USA, 2018. IEEE CS.

[11] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt. Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Trans. Graph.*, 36(4):44:1–44:14, 2017.

[12] J. Martinez, R. Hossain, J. Romero, and J. J. Little. A simple yet effective baseline for 3d human pose estimation. In *Proc. of ICCV'17*, pages 2659–2668, 2017.

[13] K. Wang, L. Lin, C. Jiang, C. Qian, and Pengxu W. 3d human pose machines with self-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[14] Carnegie Mellon University. MoCap Library: `http://mocap.cs.cmu.edu/`, 2019.

[15] PhaseSpace Inc. Optical MoCap Systems: `http://www.phasespace.com`, 2019.

[16] Vicon. Vicon Motion Capture Systems: `http://www.vicon.com`, 2019.

[17] A. Aristidou and J. Lasenby. Real-time marker prediction and CoR estimation in optical motion capture. *Vis. Comput.*, 29(1):7–26, 2013.

[18] A. Aristidou, D. Cohen-Or, J. K. Hodgins, and A. Shamir. Self-similarity analysis for motion capture cleaning. *Comput. Graph. Forum*, 37(2):297–309, 2018.

[19] Xsens Technologies B.V. Motion Capture Systems: `http://www.xsens.com`, 2019.

[20] J. Tautges, A. Zinke, B. Krüger, J. Baumann, A. Weber, T. Helten, M. Müller, H.-P. Seidel, and B. Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Trans. Graph.*, 30(3):18:1–18:12, 2011.

[21] R. Slyper and J. K. Hodgins. Action capture with accelerometers. In *Proc. of SCA'08*, pages 193–199, Aire-la-Ville, Switzerland, 2008. EG Association.

[22] A. Aristidou, Y. Chrysanthou, and J. Lasenby. Extending FABRIK with model constraints. *Comput. Animat. Virtual Worlds*, 27(1):35–57, 2016.

[23] P. Carreno-Medrano, S. Gibet, and P.-F. Marteau. From expressive end-effector trajectories to expressive bodily motions. In *Proc. of CASA '16*, pages 157–163, NY, USA, 2016. ACM.

[24] C. Zimmermann, T. Welschehold, C. Dornhege, W. Burgard, and T. Brox. 3d human pose estimation in RGBD images for robotic task learning. In *Proc. of ICRA '18*, 2018.

[25] A. Biswas, H. Admoni, and A. Steinfeld. Fast on-board 3d torso pose recovery and forecasting. In *Proc. of ICRA'19*, 2019.

[26] G. K. M. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *Proc. of CVPR'03*, pages 77–84, DC, USA, 2003. IEEE CS.

[27] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Trans. Graph.*, 27(3):98:1–98:10, 2008.

[28] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph.*, 27(3):97:1–97:9, 2008.

[29] J. Gall, A. Yao, and L. Van Gool. 2d action recognition serves 3d human pose estimation. In *Proc. of ECCV'10: Part III*, pages 425–438, Berlin, Heidelberg, 2010. Springer-Verlag.

[30] Y. Liu, J. Gall, C. Stoll, Q. Dai, H.-P. Seidel, and C. Theobalt. Markerless motion capture of multiple characters using multiview image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2720–2735, 2013.

[31] X. Wei, P. Zhang, and J. Chai. Accurate realtime full-body motion capture using a single depth camera. *ACM Trans. Graph.*, 31(6):188:1–188:12, 2012.

[32] M. Ye and R. Yang. Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera. In *Proc. of CVPR'14*, pages 2353–2360, DC, USA, 2014. IEEE CS.

[33] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Commun. ACM*, 56(1):116–124, 2013.

[34] T. Sharp. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Proc. of CVPR'12*, pages 103–110, DC, USA, 2012. IEEE CS.

[35] A. Baak, M. Müller, G. Bharaj, H.-P. Seidel, and C. Theobalt. A data-driven approach for real-time full body pose reconstruction from a depth camera. In *Proc. of ICCV'11*, pages 1092–1099. IEEE CS, 2011.

[36] D. Vlasic, P. Peers, I. Baran, P. Debevec, J. Popović, S. Rusinkiewicz, and W. Matusik. Dynamic shape capture using multi-view photometric stereo. *ACM Trans. Graph.*, 28(5):174:1–174:11, 2009.

[37] K. Berger, . Ruhl, Y. Schroeder, C. Bruemmer, A. Scholz, and M. Magnor. Markerless motion capture using multiple color-depth sensors. In P. Eisert, J. Hornegger, and K. Polthier, editors, *Proc. of Vision, Modeling, and Visualization*. EG Association, 2011.

[38] N. Sarafianos, B. Boteanu, B. Ionescu, and I. A. Kakadiaris. 3d human pose estimation: A review of the literature and analysis of covariates. *Comput. Vis. Image Underst.*, 152(C):1–20, 2016.

[39] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy. Towards accurate multi-person pose estimation in the wild. In *Proc. of CVPR'18*, DC, USA, 2017. IEEE CS.

[40] X. Zhou, M. Zhu, G. Pavlakos, S. Leonardos, K. G. Derpanis, and K. Daniilidis. MonoCap: Monocular human motion capture using a CNN coupled with a geometric prior. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018.

[41] E. Simo-Serra, A. Quattoni, C. Torras, and F. Moreno-Noguer. A joint model for 2d and 3d pose estimation from a single image. In *Proc. of CVPR'13*, pages 3634–3641, DC, USA, 2013. IEEE CS.

[42] H. Rhodin, J. Spörri, I. Katircioglu, V. Constantin, F. Meyer, E. Müller, M. Salzmann, and P. Fua. Learning monocular 3d human pose estimation from multi-view images. In *Proc. of CVPR'18*, pages 8437–8446, DC, USA, 2018. IEEE CS.

[43] H. Rhodin, M. Salzmann, and P. Fua. Unsupervised geometry-aware representation for 3d human pose estimation. In *Proc. of ECCV'18*, 2018.

[44] K. Forbes and E. Fiume. An efficient search algorithm for motion data using weighted PCA. In *Proc. of SCA '05*, pages 67–76, 2005.

[45] G. A. F. Seber. *Multivariate Observations*. John Wiley & Sons, Inc., NJ, USA, 1984.

[46] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[47] S. J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, Inc., NJ, USA, 1995.

[48] G. Casiez, N. Roussel, and D. Vogel. 1 € filter: A simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of CHI '12*, pages 2527–2530, NY, USA, 2012. ACM.

[49] Root-Motion. FINAL-IK: `http://root-motion.com/`, 2019.

[50] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[51] Ronald W. Schafer. [lecture notes] what is a savitzky-golay filter?, 2011.

[52] M. F. Ghezelghieh, R. Kasturi, and S. Sarkar. Learning camera viewpoint using CNN to improve 3d body pose estimation. In *Proc. of 3DV '16*, 2016.