Integrating Live Skeleton data into a VR Environment

Tom Hoxey & Dr Ian Stephenson
NCCA, Bournemouth University, UK

Abstract

The aim of this project is to be able to visualse live skeleton tracking data in a virtual analogue of a real world environment, to be viewed in VR. Using a single RGBD camera motion tracking method is a cost effective way to get real time 3D skeleton tracking data. Not only this but people being tracked don't need any special markers. This makes it much more practical for use in a non studio or lab environment. However the skeleton it provides is not as accurate as a traditional multiple camera system. With a single fixed view point the body can easily occlude itself, for example by standing side on to the camera. Secondly without marked tracking points there can be inconsistencies with where the joints are identified, leading to inconsistent body proportions. In this paper we outline a method for improving the quality of motion capture data in real time, providing an off the shelf framework for importing the data into a virtual scene. Our method uses a two stage approach to smooth smaller inconsistencies and try to estimate the position of improperly proportioned or occluded joints.

Keywords: Motion Tracking, Motion Capture, Skeleton Tracking, Kinect, Real Time, Virtual Reality, Telepresence

1 Introduction

This research forms a part of a larger project to create a virtual double of the high power laser lab at the University of Southampton. Due the the nature of the research conducted in the lab every person inside the lab increases the likelihood of the data being contaminated in some way. As such the data collected in the lab is pushed in real time to a central server that can be accessed via a virtual double of the lab. The aim is for the virtual double of the lab to be an equivalent or even better experience than being in the actual lab itself.

To fully recreate the lab a visualisation of the people in the real lab is needed. To do this some sort of skeleton tracking system is required. A full motion tracking system is not at all feasible for the lab. They are expensive to purchase, especially considering the cameras needed, the computer hardware to run it and the cost of technical staff needed to set it up. Not only this but often these require special markers being worn by the people being tracked. As such the Xbox Kinect is far more suited to the job, it is (at the time of writing) an order of magnitude cheaper than a multiple camera alternative, and people are not required to wear any special tracking markers. As well as this it is far more accessible than any other RGBD cameras on the market, it broke the world record for fastest selling consumer electronics device at its first release.[Zha12]

Our method focuses on smoothing the actual skeleton data itself and not the captured image. It should be noted that a computer vision approach to increase the accuracy of the skeleton acquisition is also an important part of the process, however our method is designed to be after this stage has occurred. Smoothing the skeleton is a lesser researched area but we believe it still has value in both real time and offline applications. In real time working with the skeleton is a much smaller data set than the whole captured image, as such you can perform more complex operations with a smaller overall cost. This is especially worthwhile on applications, such as ours, that transfer the data across the network. Even in offline applications unlike ours there may be cases in which after the data has been captured there is often a cleanup stage. If part of this cleanup can be automated this has potential to increase artist productivity.[1]

Before we get into detail regarding the method it is worth talking over the basic structure of our implementation as it will provide necessary context for the rest of the paper. As illustrated in figure 1 the data is captured in an application on the device with the Kinect. Next we send that data as ASCII to a central server, in our implementation it is an MQTT broker running on a Raspberry Pi. Finally a Unity application pulls the capture data from the server and performs our correction method on it and then renders the skeleton.

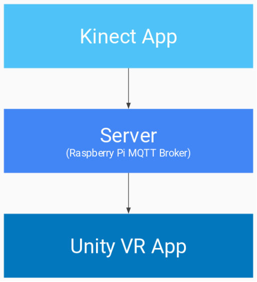

Figure 1: *A diagram to show how the data flows through our implementation*

2 Existing Methods

The problems that our method aims to overcome are very specific and as such there is very little research that covers our exact problem statement. Other optical tracking methods that estimate skeletal parameters such as [KOF05] often use some sort of body tracking markers. To keep our solution usable in a non studio environment markers prove to be far too impractical.

Using inverse kinematics in motion capture is a standard technique for cleaning motion capture data, it can help to stop joints penetrating the other geometry in the scene. [MM05] This is certainly a worthwhile technique to implement in many motion capture cases. However due to the nature of our implementation it is not necessary to implement in our case. Since the data we are capturing is being played in a virtual reconstruction of the environment in which it was recorded it would be impossible for an actor to penetrate the in scene geometry, since they would have to penetrate the real life object. There may be penetrations due to tracking errors, however this is unlikely due to the nature of the skeleton acquisition in the Kinect application. A limb on some sort of hard surface is the best case for the tracking as it has an unambiguous background.

There is promising research into using multiple Kinect cameras to create a more accurate overall picture, such as in [BRS$^+$11], how-

[1] While this research uses the Kinect any equivalent skeleton tracking system could be used instead, the method itself is not reliant on any special features of the product. However the Kinect tracking method is very robust and is recognised to be a solid grounding to build from.[HSXS13] To render the virtual scene we are using the Unity game engine, once again the techniques descried in this paper don't use any specific features of Unity, it was chosen for convenience sake.

ever for every new sensor we add the cost of setting up the system increases. This is in terms of both cost to buy the equipment, and time required to set-up the system. As such we decided to only use a single camera for a more plug and play approach. However if the research were to be further developed this seems to be the most promising first avenue.

3 Our Method

We developed a two stage method for smoothing the tracking data, correction followed by filtering. In the correction stage we change the distance between joints to avoid variation in limb length. We can also perform an estimation of the position of untracked joints using measurements from previous frames. In the filtering stage we apply a Kalman filter to each joint, this helps to reduce smaller jitters in the data as well as interpolating our data across frames so we can have a higher frame rate than the rate of data transmission.

3.1 Developing the network back end

In this subsection I will briefly outline how we set up the network transfer and the rationale behind it, this provides some necessary context to the rest of the paper.

To transfer the data we chose the MQTT network protocol (We use the Mosquitto implementation of MQTT), it is a publish/subscribe based model and as such allows for multiple virtual scenes to be running at once. As well as this there is a lot of existing support that made implementing the network code much simpler.

Firstly we rewrote one of the default Kinect demos to send the position and name of the joint to the server in a custom ASCII format. This is updated at 20Hz as this was found to be a good balance between keeping the data live while leaving a smaller footprint on the network. Secondly we subscribe to the data server on a machine running the Unity application, there is no restriction to what this machine needs to be, it can run on a different platform, a different network or it can be on the same machine as is running the Kinect app.

This is useful as it means that the platform dependent aspect (the Kinect app) is an independent system that once running doesn't need to be touched, it can be left to run in the background forever. While it is always looking for tracking data if there is no one in view it doesn't send anything, as such there is no unnecessary network overhead. Since it is a publish and subscribe network model it is possible to run the Unity application at any time with no extra set up needed. This kind of plug and play system is far more convenient for users, since they don't have to undergo a tedious configuration process every time they launch.

3.2 Average Fit

Algorithm 1 Pseudocode for updating the average distance value

```
procedure UPDATE AVERAGE($newValue$)
                                        ▷ Get New Average
    oldNum = size of $averagesList$ * $currentAverage$
    difference = $newValue$ - $averagesList[0]$
    newAverage = (oldNum - difference) / size of $averagesList$
                                        ▷ Update the List
    Remove $averagesList[0]$
    Append $newValue$ to $averagesList$
```

In this stage we apply a relatively simple correction to tracked points. Over a series of frames an average distance to the parent joint is recorded. At each frame we check if the new position is going to be five percent beyond the current average, if so we set the joint to be at the current average distance away. This helps to correct the error of proportions shifting during runtime.

Our implementation currently stores five averages in a list inside each joint. This is updated every time we get a new tracked point that is within the average bounds. To update, delete the head of the list and insert a new entry at the tail. This is only $O(1)$ complexity and therefore is a cheap operation and using a list is faster than a static memory structure in this case due to constant reassignment. This updating average method is outlined in algorithm 1.

3.3 Untracked Joint Position Estimation

Algorithm 2 Pseudocode for Estimating the position of Untracked Joints

```
procedure IS LIMB($CurrentJoint$)
    if $CurrentJointParent \rightarrow ChildCount > 0$ then
        $CurrentJoint \Rightarrow Limb$
    else
        $CurrentJoint \Rightarrow Fixed$
procedure ESTIMATE POSITION(Skeleton)
    for all $Joints$ in $Skeleton$ do
        if IS LIMB($CurrentJoint$) then
            $CurrentJoint\vec{Direction} \rightarrow Parent\vec{Direction}$
        else
            $CurrentJoint\vec{Direction} \rightarrow Last\vec{Direction}$

        $CurrentJointPosition \Rightarrow AverageDistancealongCurrentJoint\vec{Direction}$
```

If we have no tracked data for the joint we have to try and approximate its position, to do this we use the averages that we have been collecting and the last recorded direction of either the parent or the joint itself, depending on the joint. To do this we have developed a simple algorithm as outlined in algorithm 2. It should be noted that shoulders are an exception to this in the Kinect joint system as they connect directly to the spine.

3.4 Kalman Filter

First outlined in [Kal60] the Kalman filter is a linear filtering method based on predictions formed from previous measurements. Although the Kinect's internal workings are propitiatory it is likely that it employs a Kalman based system to track the objects in 3D such as in [LKK95].

In our method we use a Kalman filter on three dimensional coordinates over time.This has two advantages to smoothing at the image processing stage. Firstly it reduces the amount of data that is required to be sent over the network, to stream 1080p RGBD capture data across the network requires a much higher rate of transfer than the just the points (24kb/s). Especially since the points values can remain absolute whereas to send the video would most likely require some sort of compression, losing accuracy. Secondly the Kalman filtered points give smoother animation arcs than the raw data, which is much more visually appealing. While it may appear that this is less accurate I would add that if the tracked data was provably absolutely accurate then this would be the case, but since we know the data is flawed this trade-off seems worth it.

4 Results and Discussion

4.1 Critical Analysis

Overall the implementation of our method shows that it makes a measured improvement when compared to the raw input data. While most frames only benefit from the smoother motion due to the second filtering, on a per frame basis there can be much more important improvements made by the first stage.

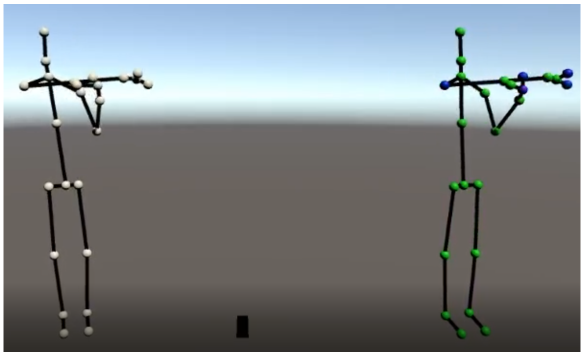

Figure 2: *Tracking data of a Shoulder Posterior Stretch (Appendix A). Showing how average fit is used to constrain proportions (Left Skeleton is raw data, Right is the corrected skeleton, joints in blue have been Average Fitted)*

The average fit on tracked joints is the most subtle of the correction methods that we have implemented, however it helps to constrain the skeletons proportions. This will be especially useful if this method was to be applied to control a full 3D mesh. Figure 2 shows how when performing an extreme case such as a stretch the average fit ensures our body stays in proportion. The right shoulder has been pulled towards the body compared to the raw skeleton where the arm has appeared to increase in length.

While it does not make as dramatic a difference as the other stages it is very cost effective considering the data is collected for the estimation stage in any case. As such I think it is a worthwhile addition considering the tracked skeleton often has mistakes in where is positions the joints at the capture stage.

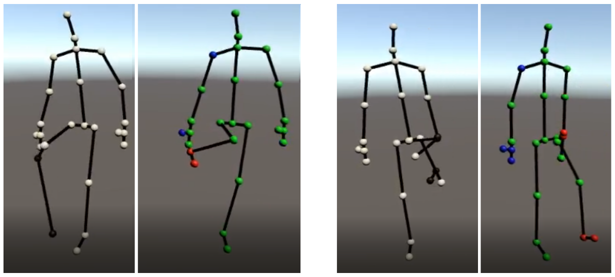

Figure 3: *Images showing capture data of a person grabbing performing a quadricep stretch (See appendix B). (In each pair: Left Skeleton is raw data white joints are tracked, black are untracked, Right is the corrected skeleton, joints in blue have been Average Fitted and red have been estimated)*

The estimation stage provides the most dramatic changes from the raw capture data. As seen in the left pair of images in figure 3 the estimated data manages to correctly determine the location of the foot. Even in the left pair of images where the estimation cannot determine the foots location it puts the skeleton back into a rest position, this is far more preferable to the bunched up joints shown in the raw data.

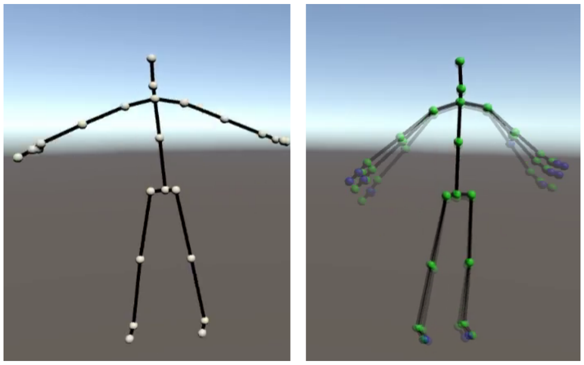

Figure 4: *3 frames of overlapped footage, showing that the smoothed implementation updates 3 times while the raw data only updates once*

The second filter stage is illustrated in figure 4. As this image clearly demonstrates our filtered method is updated 3 times more often than the raw tracking data. This provides a far superior quality of playback than the raw data alone. The filtered skeleton updates at sixty frames per second compared to only twenty for the raw data. [2]

4.2 Future Work

While our system provides a measured improvement over the raw tracking data, it does have limitations. This is especially notable regarding the estimation of joint position. The estimations that it provides mostly return the joint to a rest position as seen in the far right image of figure 3. A continuing goal of the system is to improve the rate at which we can correctly estimate the location of the joint. However since the Kinect does not provide rotation data for the joints it would most likely require the implementation of a full forward kinematics system.This did not seem necessary for the first prototype of the project as we were aiming for verisimilitude as opposed to accuracy. Meaning that for a telepresence application we want the data to be plausible and appropriate as opposed to entirely accurate.

In future if we wanted to increase the accuracy of our system using multiple Kinects in conjunction with each other would be the most effective way to do this. A software approach may be possible, however the development cost of this system would far outweigh the cost of a second camera. The second camera would remove many of the possible self occlusion cases as well as providing a fuller view of the room.

[2] Twenty was decided due to network limitations, the Kinect is capable of capturing up to thirty frames per second, but the required transfer rate was shown to put too much stress on the network we tested on.

References

Kai Berger, Kai Ruhl, Yannic Schroeder, Christian Bruemmer, Alexander Scholz, and Marcus A Magnor. Markerless motion capture using multiple color-depth sensors. In *VMV*, pages 317–324, 2011.

Nathan Cowley. Photography of woman in pink tank top stretching arm, oct 2017.

J. Han, L. Shao, D. Xu, and J. Shotton. Enhanced computer vision with microsoft kinect sensor: A review. *IEEE Transactions on Cybernetics*, 43(5):1318–1334, Oct 2013.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

Adam G. Kirk, James F. O'Brien, and David A. Forsyth. Skeletal parameter estimation from optical motion capture data. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) 2005*, pages 782–788, June 2005.

Joon Woong Lee, Mun Sang Kim, and In So Kweon. A kalman filter based visual tracking algorithm for an object moving in 3d. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 342–347 vol.1, Aug 1995.

Amanda Mills. female, quadriceps, stretching, position, 2015.

Michael Meredith and Steve Maddock. Adapting motion capture data using weighted real-time inverse kinematics. *Comput. Entertain.*, 3(1):5–5, January 2005.

Zhengyou Zhang. Microsoft kinect sensor and its effect. Technical report, April 2012.

A Shoulder Posterior Stretch

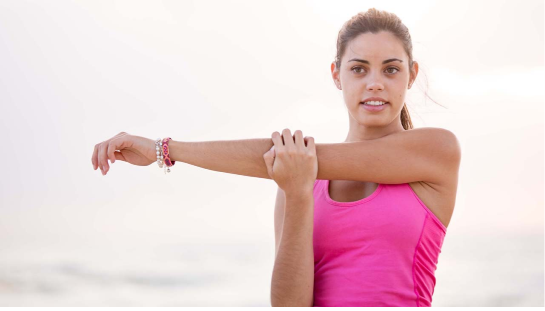

Image from: [Cow17]

B Quadricep Stretch

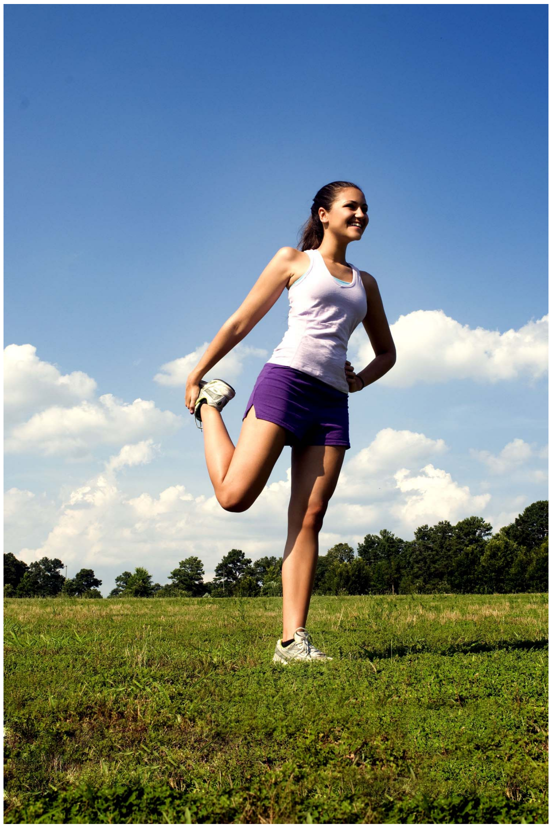

Image from: [Mil15]