Designing an Immersive Virtual Reality Avatar Animation Software

Ramit Bag

University College London, Department of Computer Science

Research Supervisor: Prof Anthony Steed

Funded by the Laidlaw Undergraduate Research and Leadership Programme

Acknowledgements

This project could not have been completed without the ongoing support of my supervisor Professor Anthony Steed, as well as Dr Sebastian Friston, Dr Klara Brandstaetter for their invaluable support, and the generous funding from Lord Laidlaw and the Laidlaw Foundation.

Contents

Abstract

Currently, there exists no open-source VR avatar animation software for the whole body. Most social VR systems that exist use avatars that stop at the waist level such as Meta's horizon world [1]. Immersive systems such as the Quest utilise three point tracking (head, and both hands) to generate the upper body of the avatar using inverse kinematics (IK) to calculate the body position of the avatar. But what about the lower body? How can we simulate the positions and locomotion of the legs with just these 3 points?

Figure 1: Meta's Horizon world [1]

Some VR systems such as the HTC vibe utilise "vibe trackers" which allow additional trackers to be placed on the legs. This has been used in VR Chat to generate kinematic models of avatars that follow deterministic points for the leg. But currently majority of VR users don't poses such hardware and only use the controllers and HMD.

Final IK [2] (commercial asset) is used to generate foot animations, but is rather crude as it only deals with kinematic motion. Our first goal is to recreate the Final IK software that can generate lower body motion which appears realistic. In order to improve the algorithm, we must use machine learning and combine it with IK. This will produce agents that are able to behave and traverse the world realistically. However, the IK must be first developed for the leg before combining it with Motion Capture data.

Introduction

For the purpose of implementing this project, we will be using Unity [3], an open source, cross platform game engine. Unity's humanoid model for avatars consists of a 15-bone skeleton structure which mimics a real human skeleton (shown in figure 2)

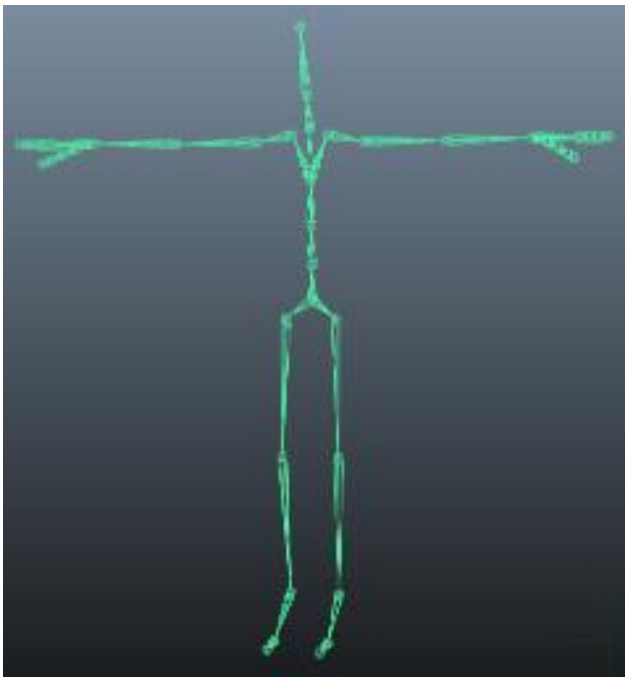

Figure 2: Biped Skeleton, positioned in T-pose [4]

Figure 3: Skeletal hierarchy of an avatar

As we can see, the hips are the root of the skeleton and the other skeleton parts are all arranged in hierarchy of distance from the root. Since the only 3 deterministic points are the head, left and right hands, we must calculate the position and rotation of the other skeletal parts in order to animate the avatar in real time.

Without using any external software/tools we can animate the avatar in unity already. This works by tracking the position and rotation of the HMD (Head mounted display) and if the position is moved in a certain direction, a prefabricated animation will play. For example. If we move the HMD forward (relative the direction the user faces), the “Walk forward” animation will play, and if the HMD is moved backwards, the “Walk backwards” animation will play. This animation system exists in the “animator” tab in unity, shown in the figure below:

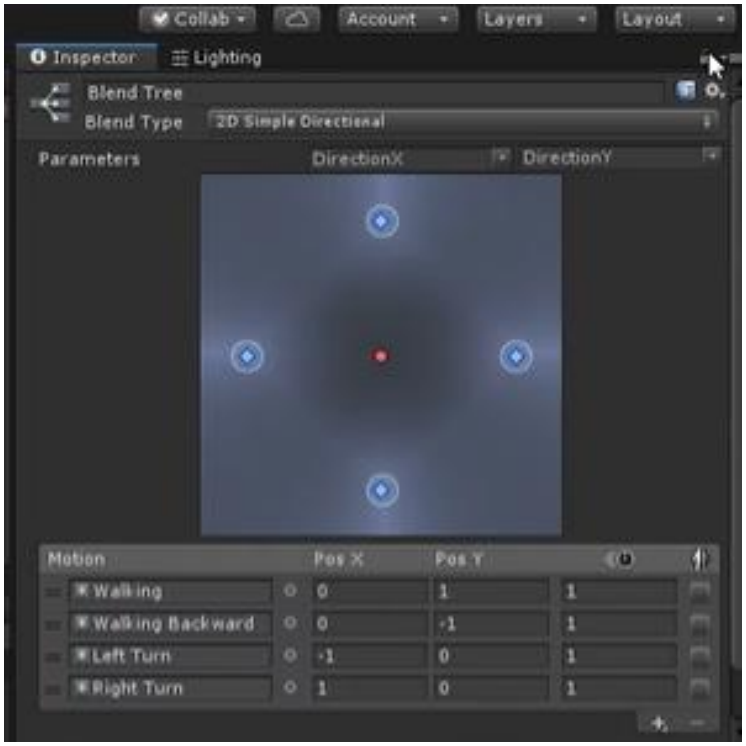

Figure 4: Blend tree for dictating animation

In this blend tree, 4 animations have been configured (forward, backward, left and right) and the animation will play based on the change in the x and y position relative to the previous position in real time. However, the animation produced is vastly inaccurate to the real user's motion. The problems with this method are:

1. Doesn't take into account the hands' locomotion. Only a generic arm swing is played when moving.
2. If the user's head position is still, but they move their arms, the animation will not represent this at all.
3. Doesn't take into account velocity. The same animation will play whether the user walks slowly or runs
4. If the user crouches/ jumps, no animation plays (as the x and y of the HMD remain the same).
5. The avatar doesn't animate any simple tasks such as pointing, shaking their hands, kicking, waving etc.

Therefore this 1-point tracking method is unreliable as a way to animate the avatar close to how a real human move. Therefore a 3 point tracking algorithm must be used to accurately display the user's hand movements. The current available software that is used frequently in both industry and by developers is Final IK [2]. Using this paid software, the position and rotation of each joint in the avatar skeleton can be calculated using an algorithm known as Inverse Kinematics.

Inverse Kinematics

Before explaining Inverse kinematics, we must understand forward kinematics. Forward kinematics is used to calculate the Cartesian position and rotation of the end effector given the input joint angles and length and start point. Let's look at the example diagram below:

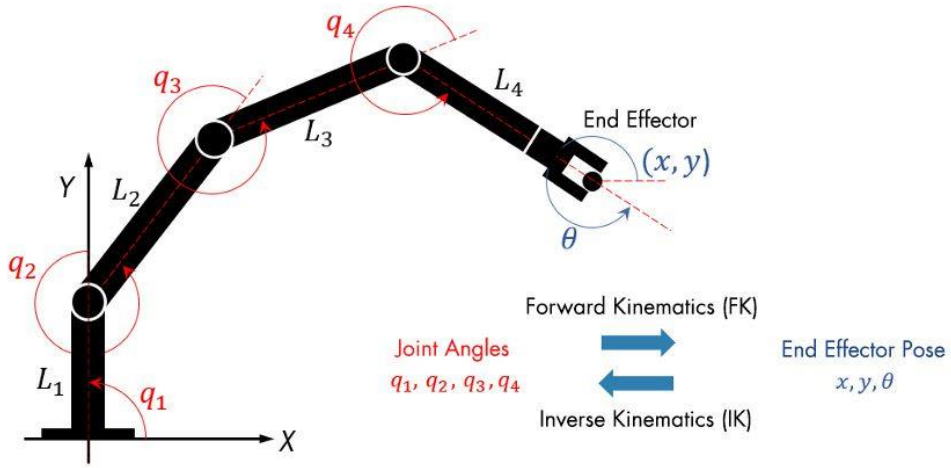

Figure 5: Forward Kinematic chain diagram [6]

If take the lengths (L1, L2, L3, L4 in the diagram) and the angles (q1, q2, q3, q4) we can calculate the end effector using trigonometry.

Inverse Kinematics works in the opposite way. The base and end effector's cartesian position and rotation are given, and we must calculate all the remaining cartesian position and rotation for the joints in between. There are multiple methods of applying inverse kinematics, but all of them rely on an iterative optimisation technique that produces an approximate solution. This is because inverting the forward kinematic equation is difficult given the large number of possible solutions in an empty solution space. Techniques exists such as the "Jacobean inverse [7], CCD (cyclic coordinate descent) [8], and FABRIK (Forward-and-Backward Reaching Inverse Kinematics, and the main algorithm driving the Final Ik software is FABRIK [5]

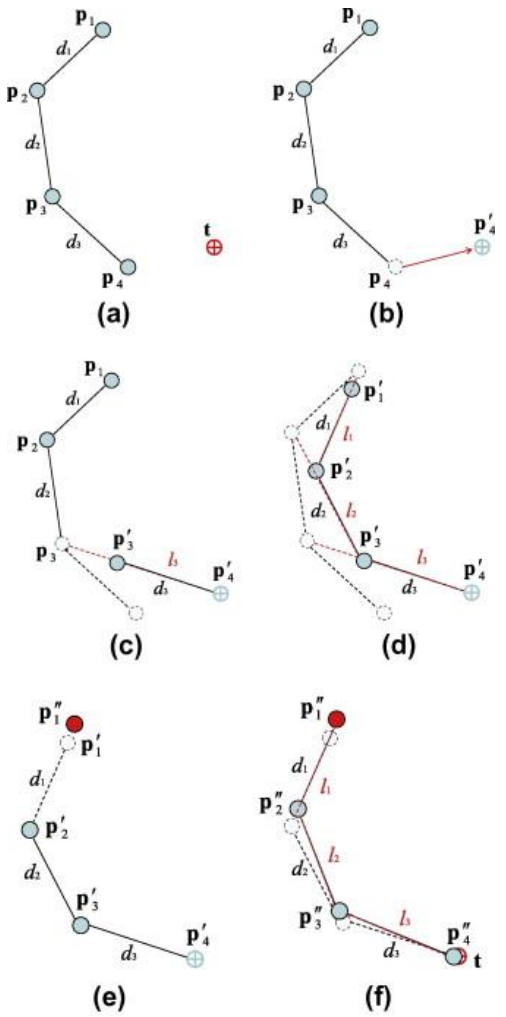

Figure 6: A full iteration of FABRIK [5]

Fabrik is a great algorithm as it has a low computational cost and we can add constraints to the rotations. This is important because the human skeleton is constrained in the movements it can produce e.g. elbows can't swing 360 degrees in all directions. Thus, by having the ability to fix these constraints, we can produce a more realistic animation.

Procedural Animation

Since we understand the theory behind calculating joint position and rotation given the end effector, we can now establish the 3 deterministic points (controllers and HMD) as the end effectors. Unlike the Blend Tree method mentioned in the introduction, this method tackles all 5 issues mentioned.

The avatar animation can be broken down into 2 sub problems: upper body and lower body animation. This is because the strategy behind generating these animations are vastly different. Whilst we can simply use a version of Inverse kinematics for the upper body, the lower body animation must be extrapolated from the upper body.

Upper Body Animation

In order to animate the upper body, we can set up the head, left hand, and right hand as a Rig and then use Unity's "Animation Rigging" package to add the IK Script in unity known as "Two Bone IK Constraint" for the hands. Using this script, we can constraint the tip as the wrist, and then the root as the upper arm. For the head, we use the "Multi Parent Constraint" script since we don't just want the head to move but also the rest of the body (Including the hands when rotating the head). All we have to do now is stream the position and rotation of the 3 points and then fix the respective points to the avatar. In this case we have used the "Oculus quest" as our physical device, and we will be using the prebuilt package "XR Rig" to stream the data from the physical devices onto Unity.

Figure 7: IK Script for hands

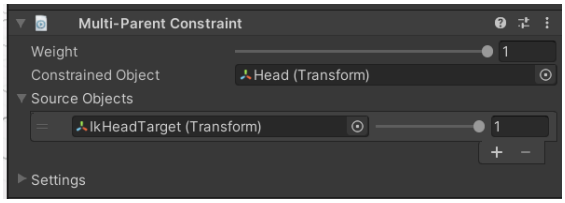

Figure 8: IK Script for head

In order to use this data and manipulate the 3 end effectors on the avatar, I have written a C# script. In summary, it takes the input of the VR target (eg left hand controller), the avatar target (eg left wrist) and sets the position in real time. I have chosen to add an offset variables for the rotation and position in case the user finds that the orientation of the avatar is slightly off or the position (different user may grip the controllers differently). I have also added a turn smoothness variable, which slows down the rotation animation of the body relative to the head, so when the user turns around, the avatar's body doesn't turn sharply, instead gradually. This further mimic the way humans move when turning in real life

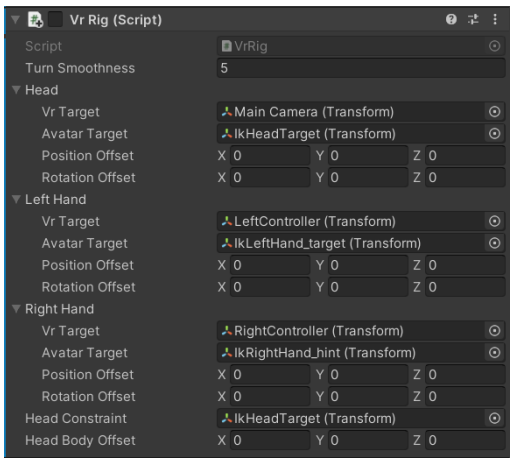

Figure 9: Rig Script for the upper body

As a result, we have animation that plays for the upper body of the avatar. The following problems have already been resolved from the previous method:

- Doesn’t take into account the hands’ locomotion. Only a generic arm swing is played when moving.
- If the user’s head position is still, but they move their arms, the animation will not represent this at all.
- The avatar doesn’t animate any simple tasks such as pointing, shaking their hands, kicking, waving etc.

(To see the animation being playing, check out the project on GitHub)

Lower Body animation

This is the most challenging part of the solution. Unlike the upper body, where the end effectors were deterministic with the head and controllers, the lower body has no trackers. The problem could be solved if we introduced trackers in VR, however popular devices such as the Oculus quest or HTC Vive don’t come with feet trackers.

Method 1

The first strategy for procedural animation is using the Inverted pendulum method [9] shown below:

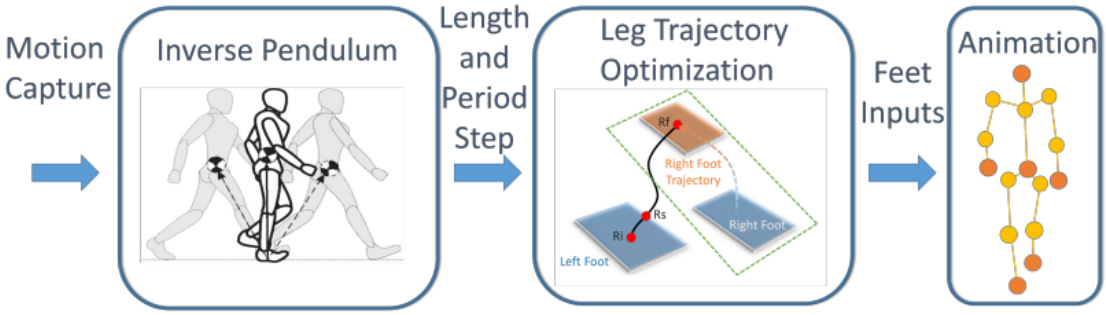

Figure 10: Inverse Pendulum model for generating animations [9]

If we look top down of the avatar (bird eye view), we see the following:

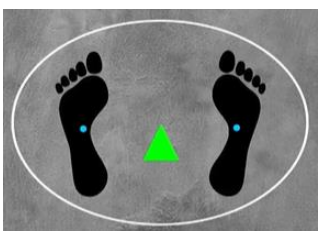

Figure 11: Stable avatar [10]

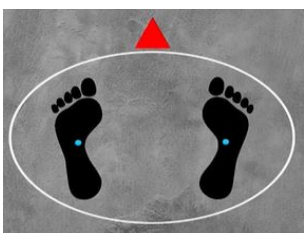

Figure 12: Unstable avatar [10]

The triangle in figure 11 represents the centre of mass of the avatar and the two feet correlates to the foot position of the avatar. As the centre of mass moves forwards (the user

leans forward), it will eventually go beyond the ellipse shown in figure 12. When this occurs, the avatar must step forward to keep balance, or fall over.

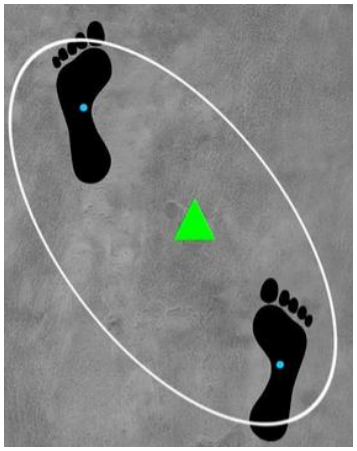

Figure 13: Left foot step [10]

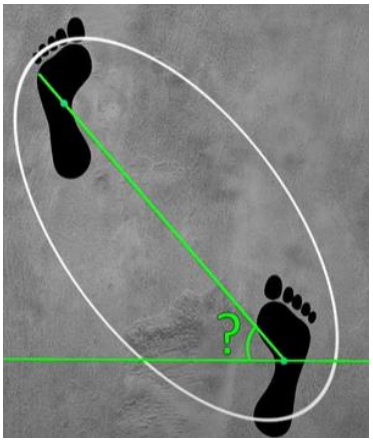

Figure 14: Calculating the angle [10]

Whilst it doesn't matter whether left or right foot steps forward first, for the purpose of this algorithm, the left foot is the primary leg. By calculating the angle between the Centre of mass and the right foot, we can extrapolate the position of the left foot to rebalance the avatar.

Once we have calculated where the foot should be placed in 2d, we need to ensure the foot sticks to the ground in 3d. This is because the terrain may not be flat, and thus it is vital that the angle the feet points is the normal to the ground. Unity has a feature called "Ray Cast" which can be used to find the normal to the plane, thus parallel to the plane the avatar steps on. This also can be used to generate foot animation for jumping/ crouching since when crouching, the foot position can be set to have a minimum value of the terrain it is currently standing on (Using Ray Cast). This is vital to prevent the avatar from falling below the terrain.

This algorithm can be run for the other foot too, and by repeating this we can generate an animation cycle, which causes the avatar to continuously calculate the new foot position if the centre of mass moves beyond the ellipse. The next step is to ensure the foot animation is smooth, and by using Unity's "Coroutines" we can spread the foot step animation across many frames. This time delay allows a smooth foot placement that almost mimics a human.

Evaluation:

Using this method, we have solved one more issue:

- If the user crouches/ jumps, no animation plays (as the x and y of the HMD remain the same).

However, multiple issues remain with only using this method. Unlike the animator method, we have now introduced a new problem (number 1)

1. The foot slides across the floor instead of arching above the ground
2. Doesn't take into account velocity. The same animation will play whether the user walks slowly or runs

Thus, we must take a different approach to tackle the following issues.

Method 2

Here is a simple walk cycle a typical human performs:

Figure 15: Human walk cycle [11]

As we can see the feet height follows an arch above the ground when it is lifted (instead of sliding across the ground). This arc of a whole walk cycle can be abstracted into a double loop:

Figure 16: A single loop shown for 1 step

In figure 16, half a step cycle is shown (right foot step). The foot position follows a semi ellipse pattern, where the height always remains above the ground. Thus, if we use a semi ellipse, we can model the foot placement in real time.

One additional benefit to using this method is that the shape of the semi ellipse can be adjusted to fit various stride lengths. This solves the final remaining issue of the avatar's animation not taking into account the velocity. For example, when a human walks slowly the stride length is smaller than the stride length of a human running. In order to accommodate this, we must keep track of the avatar's velocity (distance moved per second) and scale the size of the ellipse accordingly.

To implement this in unity we create a “Step scaler” game object. When the centre of mass (approximated by the HMD position) is moved, the velocity is calculated and adjusts the step scaler’s value. The avatar’s feet are mapped to the step scaler accordingly (based on the left/ right feet) and the walk cycle plays. If the user moves faster, the step length increases to match a faster walk/ run motion. By repeating this process for both legs, we can essentially implement the inverse pendulum model.

Evaluation:

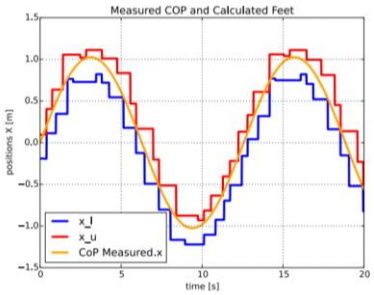

Figure 17: Position of the Contact Area enclosed by the feet during Ideal Double loop walking [9]

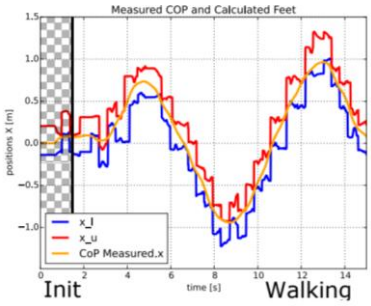

Figure 18: Position of the Contact Area enclosed by the feet during Motion Capture [9]

Figure 17 and 18 show the comparison between the double loop prediction (left) and the real walking motion of a human (on the right). According to this method [9], the double loop method maintains an 80% contact stability over time [9].

This procedural animation method has tackled all the 5 issues mentioned in the introduction. However, some issues prevent it from achieving natural motion:

1. The user is only allowed to walk/ run forwards and backwards. If the user moves directly sideways, the legs collide with each other. This is because the Centre of mass moves directly over one feet, causing the other feet to pivot abnormally. For example, if the user moves directly to the left, the right foot collides and goes through the left foot in the animation. A human in real life would move their right foot close to their left foot before moving their left foot again.

2. When crouching, the movement is highly unrealistic since humans tend to balance themselves on their toes instead of the whole feet.

3. The movement is robotic. Whilst it is able to balance the avatar correctly, it lacks variation which is represented when humans walk/ run. From the 1st person perspective, the animation will play moderately well, but when a 3rd person views this, it is comparable to a video game avatar, not a realistic human.

Motion Matching Method

There is a limit to how well procedural animation can accurately animate avatars that mimic human behaviours. Thus, a machine learning approach should be used, which predicts the orientation of the avatar using a database of MoCap data collected from humans. By focusing on creating a high quality MoCap database that represents VR users (high frequency of looking-around gestures) [12] and featuring a variety of poses, the pose and orientation generated would be more accurate than the procedural animation.

The main issue with procedural animation is miscalculating the orientation of the torso given the HMD. In this approach, we use a neural network to predict the correct body orientation given the HMD and hand controllers. Then using a Mo-cap database tailored to VR users, the lower body of the avatar are queried instead of a walking algorithm (e.g. Inverse Pendulum). Finally, an IK solver is used to generate the upper body motion and a foot lock is placed to lock the feet to the floor/ terrain.

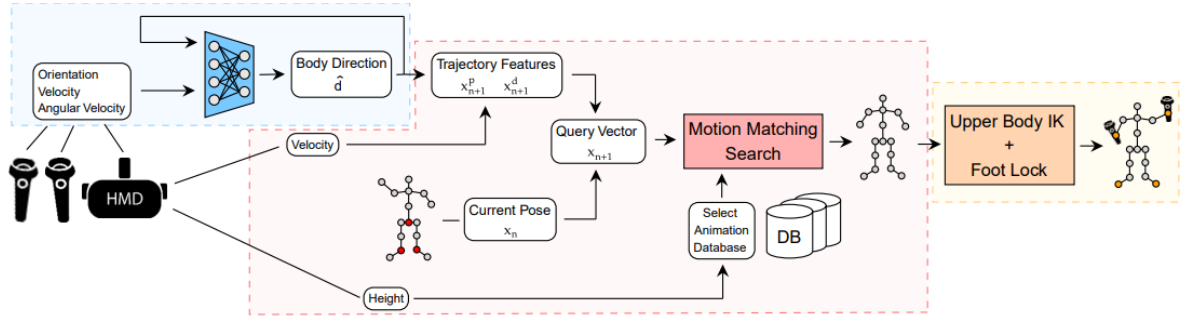

Figure 19: Overall strategy of Motion Matching Algorithm [12].

1. Neural Network

The input vectors are the velocity and rotations of the 3 trackers. A feedforward neural network is featured with 2 hidden layers (32 units each) using ReLU activation function. Volunteers were asked to play video games on the Oculus, and extreme motions were added in order to have a breadth of poses. This resulted in a total of 500k poses and 2.4 hours of motion capture. During the training processes, the MSE loss is compared between the predicted avatar orientation and the ground truth.[12]

2. Motion Matching

The predicted trajectory and pose features extracted from the current pose are used to create a query vector. Every 10 frames a Motion Matching search is performed using the query as input to find the motion that best matches the current pose and trajectory.
It is vital we have a Mocap data for crouching/ bending. When crouching/ tip-toeing, we ideally want the feet to angle itself so that the avatar stands on its toes instead of feet. But this should be only done below/ above a height, as small – medium knee bends don't result in tip toeing.[12]

3. Upper body Ik + Foot Lock

The upper body is implemented using IK. Thus, we can use the same strategy from the procedural animation for this part. To prevent the foot sliding issue faced in procedural animation, a foot lock is implemented. The feet will not move until the Motion Matching changes the pose/ a maximum distance between the feet are reached.[12]

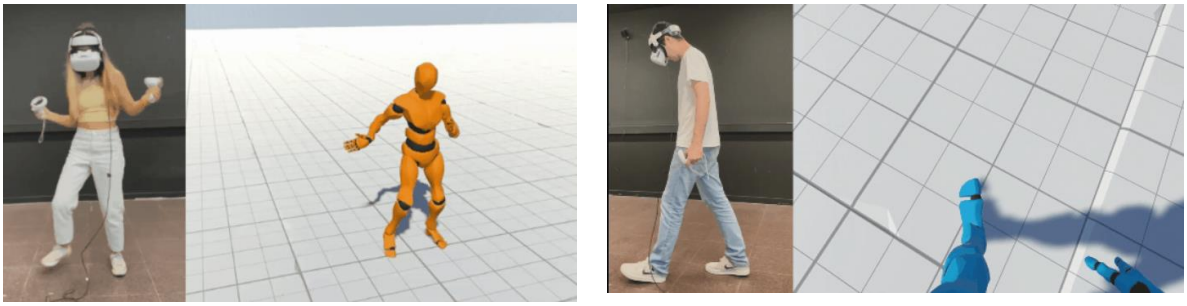

Figure 21: Motion Matching algorithm run time animation [12]

Motion Matching still uses IK for the upper body, and this is because the hand controllers result in too much variability in the upper body positions. It is impractical to store the various movements possible with alternating speeds with various user styles. In order to improve this algorithm in the future, a deep learning method can be used to generate the upper body motion. Since the motion matching database has a limited number of discrete poses, it can only be used to predict movements that are pre-recorded.

In terms of collecting this data, at the UCL VECG lab, I had access to Phase space (a motion capture system) to collect the data of the position of the body in real time.

Figure 22: A human rigged with the PhaseSpace LED lights (Upper body only)

Figure 23: Me (Author) rigged with the PhaseSpace LED lights (Whole body)

The cameras shown in figure 22 and 23 detect the LED lights and capture the position of each of them relative to the room (shown in figure 24). This is captured at a rate of 960 frames per second in HDR at a subpixel resolution of 130 megapixels with less than 3 milliseconds of latency [13].

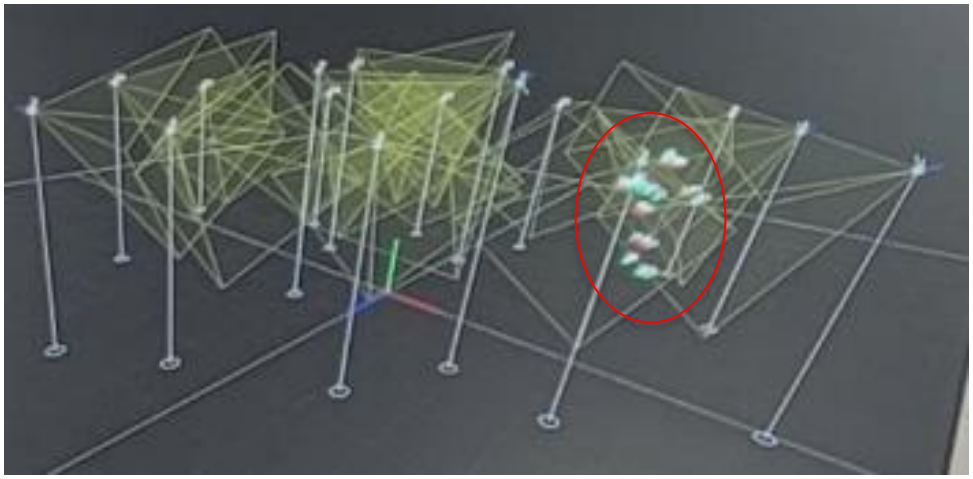

Figure 24: Zoomed version to see the avatar outline and cameras on the computer screen

The positioning of the LED is intentional: they must track the joints (vertices), and anything in between acts as the edges. Since we have a limited number of LEDs, they are placed in key areas such as extremities (arms, legs and head) and main body.

I started by making a list of actions the users should perform to have a complete database:

1. Stay in the idle T-Pose for 10 seconds
2. Stay in the natural idle position for 10 seconds
3. Turn Clockwise 5 times
4. Turn Anti- clockwise 5 times
5. Squat 5 times
6. Jump 5 times
7. Walk to the corner of the room and back to the centre
8. Run to the corner of the room and back to the centre
9. Walk backwards to the corner of the room and to the centre
10. Run backwards to the corner of the room and to the centre
11. Point at the cameras in the room, positioned around the ceiling of the room
12. Point at objects on the floor positioned around the room
13. Walk in a circle 5 times clockwise
14. Walk in a circle 5 times anti-clockwise
15. Run in a circle 5 times clockwise
16. Run in a circle 5 times anti-clockwise

By selecting 5 volunteers (including myself) for this, each person captured 5 minutes of data performing the following set of actions.

However, I soon came to the conclusion the dataset was not enough compared to existing databases which has more motion capture data than I could produce myself.

The databases were the following:

- Amass: https://amass.is.tue.mpg.de/
- CMU Graphics Lab http://mocap.cs.cmu.edu/

- HDM05 https://resources.mpi-inf.mpg.de/HDM05/

Comparison of the methods:

Having discussed both a procedural animation algorithm and the Motion Matching algorithm for the legs, we must compare the 2 under various categories:

- Walking and running: The Motion Matching algorithm produces natural walking animations and smoother transition between the HMD orientations.

- Head and body relationship: The Motion Matching algorithm is able to separate the head and body orientation. Unlike the procedural animation, where the head and body appear to always face the same direction, using MoCap data, the avatar's orientation is more accurate.

- Crouching: Unlike procedural animation, Motion matching perform a tip toe motion when the HMD is above the default height of the user and crouches with the weight on the toes rather than the whole feet, correctly mimicking human behaviour.

- Static/ idle motion: Unlike procedural animation, if the user turns their head whilst static frequently to look around, the torso won't always rotate to match the head orientation. Similarly, small changes in the HMD position will not result in slow walking animations or feet sliding.

In conclusion, the machine learning method combined with IK to stabilise the end effectors seems to offer the most realistic animation. By combining real human motion and guiding the avatar through IK, we can avoid the robotic animation style of the procedural animation method and also ensure deterministic points such as head and hands are in the correct position.

References

1. Rodriguez, S. (2019). *Facebook will soon let Oculus users build their own avatars so they can play laser tag on the moon*. [online] CNBC. Available at: https://www.cnbc.com/2019/09/25/facebook-introduces-a-virtual-world-called-horizon-for-oculus-users.html.
2. www.root-motion.com. (n.d.). *Final IK - RootMotion*. [online] Available at: http://www.root-motion.com/final-ik.html
3. Unity Technologies. *Unity - Unity*. [online] Unity. Available at: https://unity.com/
4. Technologies, U. (n.d.). *Unity - Manual: Creating models for animation*. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/UsingHumanoidChars.html

5. Aristidou, A. and Lasenby, J. (2011). FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models*, [online] 73(5), pp.243–260. doi:10.1016/j.gmod.2011.05.003.
6. www.mathworks.com. (n.d.). *What Is Inverse Kinematics?* [online] Available at: https://www.mathworks.com/discovery/inverse-kinematics.html
7. nrsyed.com. (2017). *Inverse kinematics using the Jacobian inverse, part 2*. [online] Available at: https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/#:~:text=Therefore%2C%20the%20Jacobian%20inverse%20is
8. sites.google.com. (n.d.). *CCD Algorithm for Solving Inverse Kinematics Problem - sharing_experiences*. [online] Available at: https://sites.google.com/site/auraliusproject/ccd-algorithm
9. Thomasset, V., Caron, S. and Weistroffer, V. (2019). Lower body control of a semi-autonomous avatar in Virtual Reality: Balance and Locomotion of a 3D Bipedal Model. *25th ACM Symposium on Virtual Reality Software and Technology*. doi:10.1145/3359996.3364240.
10. www.youtube.com. (n.d.). *#2 - Animate a character 100% PROCEDURALLY - The first steps - Tutorial Unity3D*. [online] Available at: https://www.youtube.com/watch?v=VMRpglAaw6k&ab_channel=L%C3%A9oChaumartin)
11. Adobe Stock. (n.d.). *Animation Cycle Images – Browse 2,620 Stock Photos, Vectors, and Video*. [online] Available at: https://stock.adobe.com/bg/search/images?k=animation+cycle.
12. Combining Motion Matching and Orientation Prediction to Animate Avatars for Consumer-Grade VR Devices. Availalable at: https://upc-virvig.github.io/MMVR/assets/pdf/motion_matching_vr.pdf
13. Anon, (n.d.). *PhaseSpace Motion Capture – Infinite Possibilities*. [online] Available at: https://www.phasespace.com/

Code

All code mentioned in the report is uploaded to GitHub. The GitHub Link for the repository is: https://github.com/RammBagg/Designing-an-Immersive-Virtual-Reality-Avatar-Animation-Software.git