Designing an Immersive VR Avatar Animation Software

Ramit Bag, Supervised by Prof Anthony Steed, Department of Computer Science

UCL

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 . 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?

Final IK (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.

Meta's Horizon world

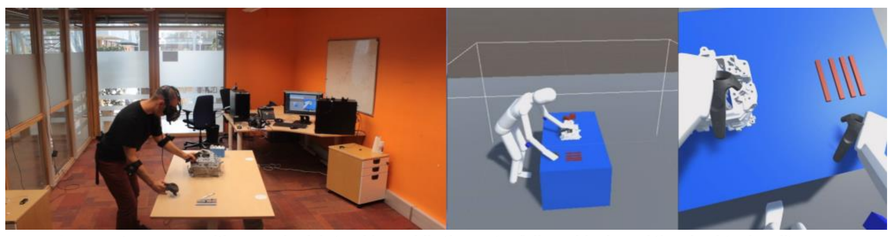
Source: https://www.cnbc.com/2019/09/25/facebook-introduces-a-virtual-world-called-horizon-for-oculus-users.html

Procedural Animation

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 (FABRIK), the lower body animation must be extrapolated from the upper body.

Upper body

Oculus provides 3 deterministic points (2 controllers and HMD) which we can use as the end effectors. In Inverse kinematics, 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. A C# script can be used to map the oculus points onto the avatar in real time. Unity's "Animation Rigging" package allows us to use FABRIK to move the head, body and the entire arm given those 3 points.

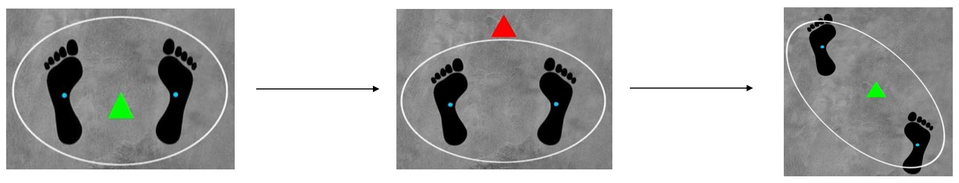


Source: https://dl.acm.org/doi/10.1145/3359996.3364240

An operator moves, with upper-body's tracking, while the avatar follows him in real time in virtual reality

Lower Body

We can use the Inverse Pendulum model to extrapolate and generate the leg animations.

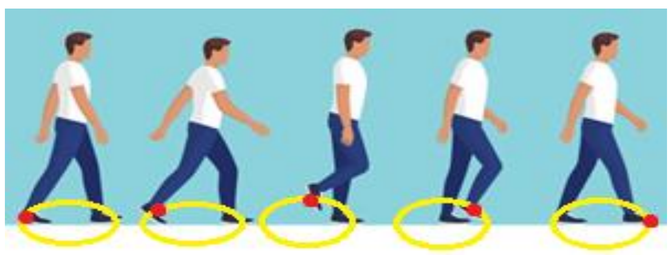


Source: https://www.youtube.com/watch?v=VMRpgIAaw6k&ab_channel=L%C3%A9oChaumartin)

Birds eye view of the avatar. The triangle represents the centre of mass of the avatar and the two feet correlates to the foot position of the avatar.

The ellipse is the region of area the COM (centre of mass) of the avatar can move without loosing balance. As the COM moves forwards (the user leans forward), it will eventually go beyond the ellipse. When this occurs, the avatar must step forward to keep balance, or fall over. This algorithm can be run for the other foot too, and by repeating this we can generate an animation cycle.

The issue with this method is the foot slides across the floor instead of arching above the ground, and the same animation will play whether the user walks slowly or runs. A human walk cycle (both left and right feet) can be abstracted into a double loop:

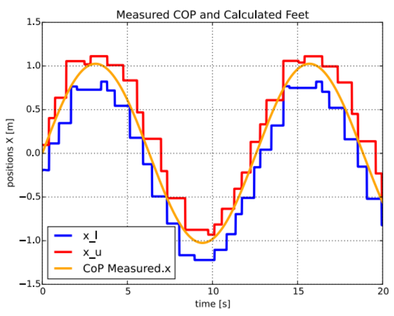


A single loop shown for 1 step (Right foot)

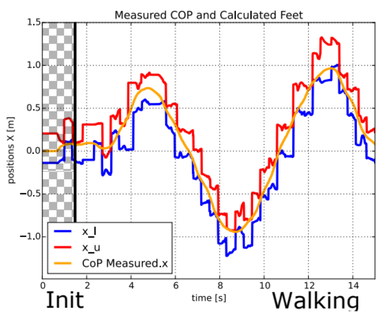
Source: https://stock.adobe.com/bg/search/images?k=animation+cycle.

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. The 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.

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 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.



Position of the Contact Area enclosed by the feet during Ideal Double loop walking

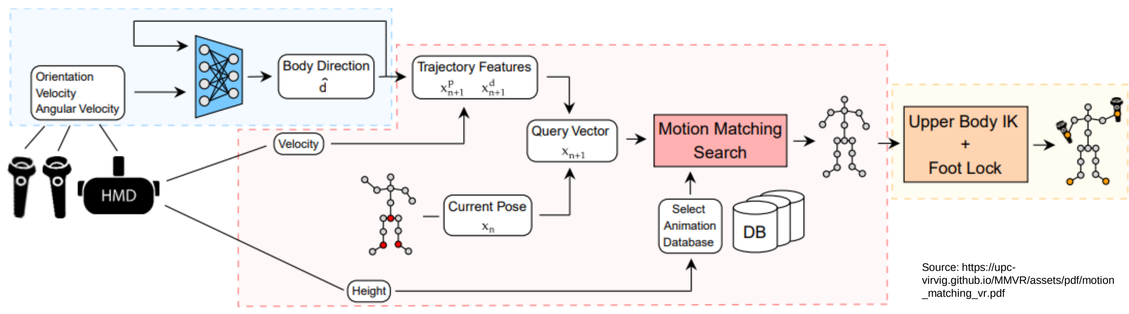


Position of the Contact Area enclosed by the feet during Motion Capture Walking walking

As we can see, the Motion Capture data used to validate the double loop method produces very similar feet trajectory to each other.

Combining Motion Matching and Orientation Prediction

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.



Source: https://upc-virvig.github.io/MMVR/assets/pdf/motion_matching_vr.pdf

Overall Strategy of Motion Matching Algorithm

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.

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.

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.

Motion Matching algorithm run time animation.

Comparison and Conclusion

The Motion Matching algorithm has numerous advantages over the procedural animation:

+ More natural walking animation and smoother transitions between the HMD orientations
+ Head and torso don't always face the same direction, so torso orientation is more accurate
+ When crouching/ tip-toeing, the feet angles itself so that the avatar stands on its toes instead of feet

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.

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.

Email: ramit.bag.21@ucl.ac.uk

LAIDLAW SCHOLARS

VECG Virtual Environments and Computer Graphics

UCL Computer Science