Simulating avatar self-embodiment using 3-points of tracking.

MICHELLE FLY, Aalborg University, Denmark

Avatar self-embodiment in Virtual Reality (VR) is simulated using the available tracking data from the VR headset, which most commonly is 3-points of tracking (The Head Mounted Display (HMD) and two controllers). These 3-points of tracking can accurately represent the head and hands of a user, however the remaining body such as torso, hips, legs, etc. needs to be simulated by adding rules for how the avatar should behave. The solution presented in this paper is based on physics and behavior of a real human body. The avatar evaluates is balance and determines whether or not it needs to counteract any imbalance by taking a step by moving the avatar's feet.

CCS Concepts: • **Computing methodologies** → *Procedural animation*; **Virtual reality**.

Additional Key Words and Phrases: VR, Avatar self-embodiment, Balance

1 INTRODUCTION

Virtual Reality (VR) in the form of a Head Mounted Display (HMD) simulates a virtual world that obscures the users real environment, including the users own body. To accommodate for this many VR developers simulate the users body to various degrees, e.g. only the hands using the tracking data from either controller's or hand tracking, or a full avatar using the available tracking data from HMD, controllers and any additional tracking points to approximate how a full avatar would act.

Most VR experiences takes place in a first person perspective, where the user would not be able to view all of their avatar self-embodiment, thus multiplayer experiences are more likely to use some degree of full avatar, e.g. VR Chat, Dead and Buried, Rec Room and Horizon, see figure 1 a, b, c, and d. As seen in figure 1 Rec Room and Horizon simulates the upper body (head, torso, hands and potentially arms), while VR Chat and Dead and Buried simulates a full avatar.

This papers focus will be on simulating an avatar using 3-points of tracking, i.e. HMD and two controllers. The only example of this seen in 1 is Dead and Buried, which uses 3-points of tracking and the Software Development Kit (SDK) Final IK to simulate the avatar self-embodiment [15], while VRChat uses additional tracking points for hips and feet and Inverse Kinematics (IK) [13].

1.1 Developer tools used to create Avatar self-embodiment

Developers wanting to include avatar self-embodiment in their product need to either make their own tool or use an existing SDK.

Root-Motion is a company focused on research and development of real-time character animation for Unity. They have released two SDK's [19];

- *Final IK:* is a tool for IK which supports VR development with up to 6-points of tracking.

(a) VR Chat [14]

(b) Dead and Buried [10]

(c) Rec Room [12]

(d) Horizon [9]

Fig. 1. Example of body simulation in VR

Author's address: Michelle Fly, mfly16@student.aau.dk, Aalborg University, Rendsburggade 14, Aalborg, Denmark, 9000.

- *Puppet Master:* is used for active ragdoll animation, which allows for a 3D model to be affected by physics in the environment. The SDK is not intended for use in VR but have been modified by some developers, e.g. Disassembly VR [11].

Both of these SDK's are intended to be used for animating a virtual model (both bipedal and other creatures). Therefore a developer would still need to include their own logic for how to balance a full avatar. Additionally, Puppet Master is not intended for VR development, so a developer would have to choose between either including physics or developing for VR using Final IK.

Deepmotion has made a physics-based solution using inverse dynamics and natural joint constraints [6], to simulate a avatar self-embodiment experience in VR. Deepmotion's system supports anywhere from one to six points of tracking [5] and allows for physics based interactions, e.g. grabbing another users arm and pulling on it, kicking a virtual ball, etc.

While this tool is intended for VR, and includes physics and logic for how to balance a full avatar, it does not seem that it is available outside of industry use.

In this paper I want to try to combine some of the logic from RootMotion and DeepMotion, i.e. physics is used to determine balance, while a rudimentary IK and joint connections is used to connect and move the body parts of the avatar.

2 IMPLEMENTATION OF AVATAR SELF-EMBODIMENT USING 3-POINTS OF TRACKING

The avatar self-embodiment can be split into two parts, the upper- and lower-body. Upper-body consist of the trunk, arms and head, while the lower-body consist of hips and legs, see figure 2.

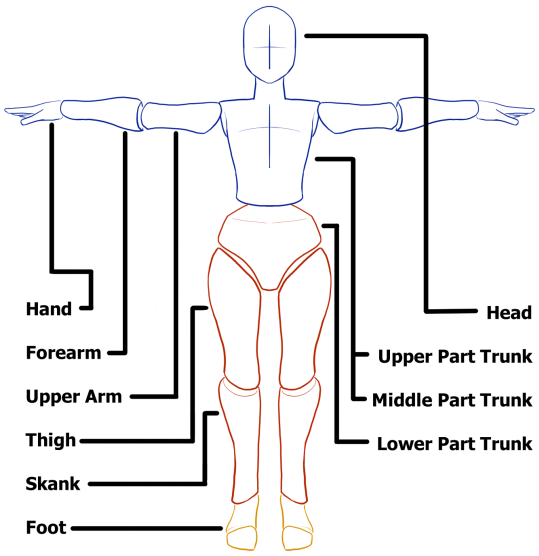


Fig. 2. Representation of an avatar, upper-body is blue, lower-body is red, and the feet are orange

The upper-body follows the tracking data from the HMD and controllers, i.e. the head follows the position and rotation of the HMD and the hands follow the position and rotation of the controllers. The body parts in between these tracking points, e.g. upper arm and forearm is moved via IK [8], and the trunk (upper part trunk and middle part trunk) is linked together via configurable joints.

The lower-body needs to rely on the same tracking data from the HMD and controllers, without being dragging across the floor. The body parts are linked together via configurable joints, i.e. the hip (lower part trunk) is connected to the upper-body and the legs are connected to the hip. The feet are positioned to stand still on the ground.

Each frame the avatar evaluates its balance, based on the current position of Center of Mass (COM), see section 2.1 and the current position of Center of Pressure (COP), see section 2.2. If the avatar is imbalanced it will try to counteract the imbalance by taking a step in order to restore balance, see figure 3. As the avatar will always aim to have one foot on the ground, the avatar is unable to jump or run. Furthermore, due to imbalance being corrected by taking a step and the trunk (both upper, middle and lower) being determined by the position of the HMD, the avatar is also unable to lean or bend forward.

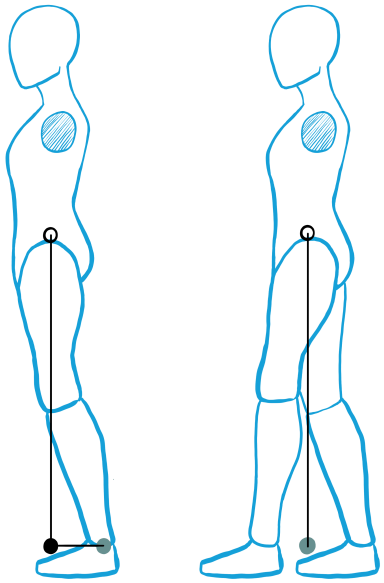


Fig. 3. Left side is imbalanced, right side is balanced by taking a step. COM is an empty black dot, COM on ground is a black dot and COP is a faded blue dot.

2.1 Center of Mass

This section is based on [1] and [2].
Center of Mass (COM) is a position in virtual space found by the mass (m) and position (P) of all body parts of the avatar. It is commonly used in biomechanics when analysing sport events or in studies regarding body posture and balance.

The position changes over time, but the mass remains consistent.

$$P(t) = [x(t), y(t), z(t)]$$
$$COM(t) = \frac{\sum_{i=1}^{N} m_i * P_i(t)}{\sum_{i=1}^{N} m_i}$$

2.1.1 ***Assigning weight to body parts.*** To find COM of the avatar, the body parts need to have a value similar to that of a real humanoid. An average human female weighs 61.9 kg and is 1.735 m tall, and an average male weighs 73 kg and is 1.741 m tall [4]. The percentage values for each body part, see table 1, can be used to calculate an accurate weight for each body part at any weight.

Table 1. Percentage values from the article *Adjustments to zatsiorsky-seluyanov's segment inertia parameters* [4]

Average Male	
Head:	6.94%
Trunk (Torso):	43.46%
Upper Part Trunk:	15.95%
Middle Part Trunk:	16.33%
Lower Part Trunk:	11.17%
Upper Arm:	2.71%
Forearm:	1.62%
Hand:	0.61%
Thigh:	14.16%
Skank (leg):	4.33%
Foot:	1.37%

Average Female	
Head:	6.68%
Trunk (Torso):	45.57%
Upper Part Trunk:	15.54%
Middle Part Trunk:	14.65%
Lower Part Trunk:	12.47%
Upper Arm:	2.55%
Forearm:	1.38%
Hand:	0.56%
Thigh:	14.78%
Skank (leg):	4.81%
Foot:	1.29%

2.2 Center of Pressure

Center of Pressure (COP) refers to the position on the ground where the ground reaction force, act on the body [2], i.e. both/one of the feet when standing upright.

In order to keep balance COP must align with COM on the ground, e.g. taking a step in the direction of imbalance in order to restore balance. COP is technically the point where the feet touches the ground, but for simplicity when evaluating the avatars balance COP can be determined to be the midpoint between the left and right foot.

2.3 Balance

In order to maintain balance the avatar must adjust its lower-body so that COP align with COM projected on the ground. For this an angle between the imbalance direction and a forward facing direction from HMD is used to determine how the feet should move to adjust COP to follow COM projected on the ground.

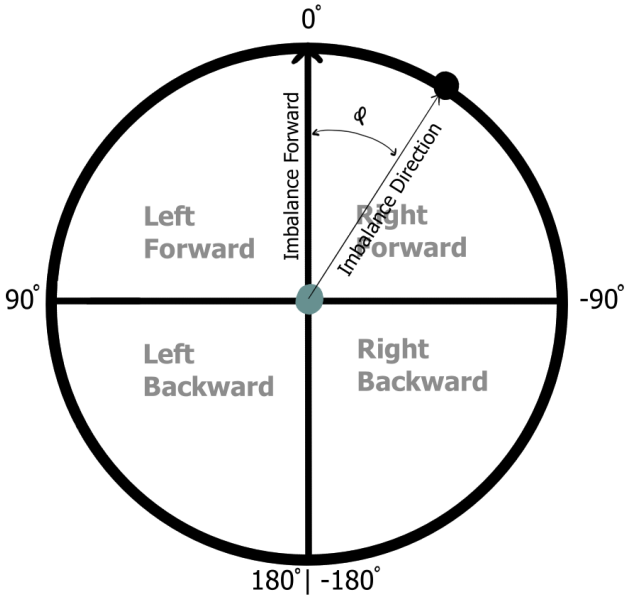


Fig. 4. The angle φ between $\overrightarrow{ImF}$ and $\overrightarrow{ImD}$ is between 180° and -180° The faded blue dot is COP and the black dot is Q

Hereafter referred to as;

(1) Q: COM projected on the ground.
(2) $\overrightarrow{ImD}$ (Imbalance Direction): Vector from COP to Q.
(3) $\overrightarrow{ImF}$ (Imbalance Forward): Vector facing forward in the direction of the HMD

This enables the avatar to calculate the angle φ between -180° and 180°, which is used to determine which foot to move, i.e. left/right, and which direction it should move, i.e. forward/backward, see figure 4.
Since the angle φ could determine either the left/right foot should move, I name the foot that should move Dynamic Foot (F_D) and the foot that should remain still Static Foot (F_S). For example in figure 4, F_D would be the right foot, and F_S would be the left foot, see Algorithm 1.

2.3.1 ***Adjust Balance.*** However, before moving F_D the system needs to know where F_S is in relation to Q and COP. The distance between COP and F_S determines whether F_S is getting too far away from the avatar and would need to be move instead of F_D, see Algorithm 2.

- $|\overrightarrow{F_SCOP}|$ <= Threshold; Move F_D
- $|\overrightarrow{F_SCOP}|$ > Threshold; Move F_S

The foots target position is found via the direction and magnitude for the foot that isn't moving to Q, see Algorithm 3, as Q is the desired position of COP. For now let's assume that F_D is the moving foot.

$$TargetPosition = Q + (-\overrightarrow{P_{F_S}Q})$$

A new target position is calculated any time a foot needs to move due to imbalance.

2.3.2 ***Pseudo code.*** Algorithm 1 is favored towards checking if the user is moving forward with either the left or right foot, and will afterwards check if the user is moving backward with.

Algorithm 1 Evaluate Balance

1: θ = angle between $\overrightarrow{COMCOP}$ and $\overrightarrow{InverseGravity}$
2: φ = angle between $\overrightarrow{ImD}$ and $\overrightarrow{ImF}$
3: **if** $\theta > 10°$ **then**
4: **if** φ is between 0° to 90° **then**
5: Assign left foot as F_D, and right foot as F_S
6: Run Function **Check Distance**
7: **else if** φ is between -90° to 0° **then**
8: Assign right foot as F_D, and left foot as F_S
9: Run Function **Check Distance**
10: **else if** φ is between 90° to 180° **then**
11: Assign left foot as F_D, and right foot as F_S
12: Run Function **Check Distance**
13: **else if** φ is between -180° to -90° **then**
14: Assign right foot as F_D, and left foot as F_S
15: Run Function **Check Distance**
16: **else**
17: Adjust left AND right foot's y position to be at ground height
18: **end if**
19: **end if**

Algorithm 2 is favored towards moving the F_D.

Algorithm 2 Check Distance

Require: F_D AND F_S
1: **if** $|\overrightarrow{P_{F_S}COP}|$ OR $|\overrightarrow{P_{F_D}COP}|$ < 0.5 **then**
2: **if** $|\overrightarrow{P_{F_S}Q}|$ <= 0.3 AND F_S is not moving **then**
3: Assign Direction = $\overrightarrow{P_{F_S}Q}$
4: Run Algorithm **Adjust Balance** for F_D
5: Adjust F_D AND F_S y position to be at ground height
6: **else if** F_D is not moving **then**
7: Assign Direction = $\overrightarrow{P_{F_S}Q}$
8: Run Algorithm **Adjust Balance** for F_S
9: Adjust F_D AND F_S position to be at ground height
10: **end if**
11: **else**
12: **if** $|\overrightarrow{P_{F_S}COP}|$ AND $|\overrightarrow{P_{F_D}COP}|$ >= 0.5 **then**
13: Reset F_D AND F_S by running Algorithm **Reset Foot**
14: **else if** $|\overrightarrow{P_{F_D}COP}|$ >= 0.5 **then**
15: Reset F_D by running Algorithm **Reset Foot**
16: **else if** $|\overrightarrow{P_{F_S}COP}|$ >= 0.5 **then**
17: Reset F_S by running Algorithm **Reset Foot**
18: **end if**
19: **end if**

Algorithm 3 assumes that a F_D and F_S has been assigned.

Algorithm 3 Adjust Balance

Require: Direction AND Q
1: Assign Target Position = Q + -Direction
2: **if** Foot is not in motion **then**
3: Run Algorithm **Follow Foot Path**
4: **end if**

2.4 Step logic

In order to perform a step, i.e. gradually moving the F_D from it's start position to the target position, the foot needs a path to follow and a duration for how long it should take.
The average time for one step, also know as a gait cycle, ranges from 0.98 to 1.07 seconds for men [7]. While this source only references men, it seems that women has a similar time of approximately 1 second pr. step (from looking at videos of people, both men and women, walking I could count 1 second between starting and completing a step).

The step logic follows the following rules;

(1) Always have one point of support (i.e. one foot should always be in contact with the ground)
(2) Complete the step withing 1 second.

In order to always have one point of support the avatar cannot initiate a new step while either of the feet are in motion.

Using the target position and the current position of the F_D (before beginning a step), the F_D can be set to complete a path within a time frame of 1 second. The position of the foot in motion is set to update at each frame using the following equation to move on the path at time t, see algorithm 4.

$$position[t] = (1-t)^2 * P0 + 2(1-t) * t * P1 + t^2 * P2$$

- P0 → Start position/Current position
- P1 → Midpoint between P0 and P2, at the height of the magnitude of a step
- P2 → End position/Target position
- t → Time between frames * speed modifier

2.4.1 ***Speed modifier.*** The speed at which the foot moves needs to be at least within one second. The foot's position moves on the path at time t, which can be slowed down or sped up by multiplying the time between frames with a speed modifier. The aim is to increase the speed at which the foot moves, and the speed modifier value therefore needs to have a value above one.

Furthermore, the speed modifier should behave according to both the magnitude of a step and the speed of the user's movement.

$$SpeedModifier = magnitude + HMD_{Speed}$$

The magnitude of a step is generally lower than one. To accommodate for this the magnitude is multiplied by 10, as to increase the value.

The speed of the HMD is found by tracking the previous and current position of the HMD each frame and dividing it by the time in seconds between frames. Similarly this means that the HMD speed value is generally lower or close to a value of 1 and is therefore also multiplied by 10.

$$HMD_{Speed} = \frac{(CurrentPosition - PreviusPosition)}{TimeBetweenFrames}$$

The final Speed Modifier;

$$SpeedModifier = (magnitude * 10) + (HMD\_Speed * 10)$$

It is worth noting that multiplying by 10 is not an optimal solution, but this will be discussed further in section 3.

2.4.2 ***Reset Foot***. To account for any loss of tracking, e.g. the controllers disappearing from view or any other mistakes during run time, the legs and feet can be reset for half a second in order to readjust, see algorithm 5. In order to reset all the logic controlling the feet whether stationary or dynamic is disabled. This will allow the foot to fall back to a neutral position before reassigning the rotation and target position of the foot.

2.4.3 ***Pseudo code***. Algorithm 4 is reliant on the equation in section 2.4, and must have an assigned foot as found by algorithm 2, either F_D or F_S, with a target position.

Algorithm 4 Follow Foot Path

Require: An assigned foot AND target position
1: Assign Magnitude as $|\overrightarrow{P_{Current}P_{Target}}|$
2: Assign P0 as current foot position
3: Assign P2 as target position
4: Assign P1 as midpoint between P0 and P1 at the height of the Magnitude
5: **while** t < 1 **do**
6: Increase t by seconds between frames * (Magnitude * 10 + HMD_{Speed} * 10)
7: As t increases from 0 to 1 the foot's path moves from P0 towards P1, then bends to arrive at P2.
8: **end while**
9: Assign t = 0

Algorithm 5 does not require knowledge of F_D or F_S, since it is activated directly from the left/right foot.

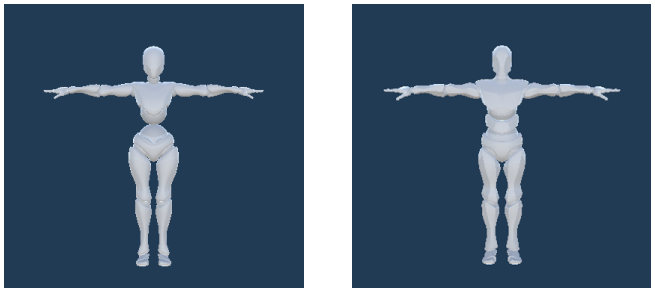
Algorithm 5 Reset Foot

1: Disable control over foot
2: Wait for 0.5 seconds
3: Enable control over foot
4: Set rotation to (0,0,0)
5: Assign target position to foot position

3 EVALUATION

3.1 A short Walk through of the system

When starting up the system the user is presented with a menu where they can select their gender and weight. This will influence which model is used (female variant or male variant, see figure 5) and how the COM is calculated by changing the mass of the rigid bodies of the model. Additionally, though invisible to the user, the user's height is measured via the HMD according to its surroundings, which is used to manipulate the height of the ground.



(a) Female model variant (b) Male model variant

Fig. 5. Models used for the avatar self-embodiment was found at Mixamo [3]

After this the user enters an environment with their avatar self-embodiment. The environment is a simple plane with two mirrors so the user can see themself, see figure 6. In the environment the user can move around as long as they have the space for it in their real environment, and can see how the virtual body behaves in the mirror or by looking down.

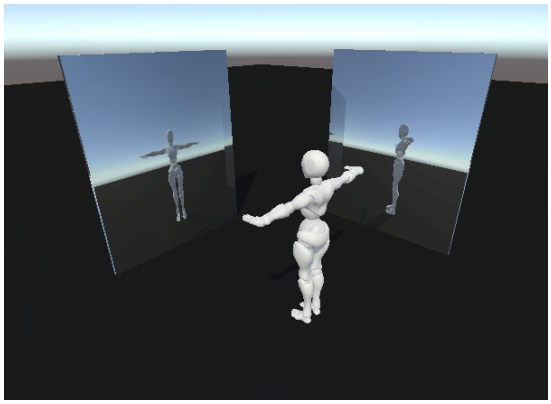


Fig. 6. Virtual Environment

3.2 Alternative solutions to issues regarding model and user size

The models used where found via Mixamo [3] and tended to simultaneously be too large and too small at the same time. Either the legs

are longer than the user and the arms fit, or the arms are to short while the height fits. To solve this, the avatar has a fixed size and the ground is instead moved to fit the height of the avatar according to the eye position of the HMD, i.e. the height of the user, see figure 7.

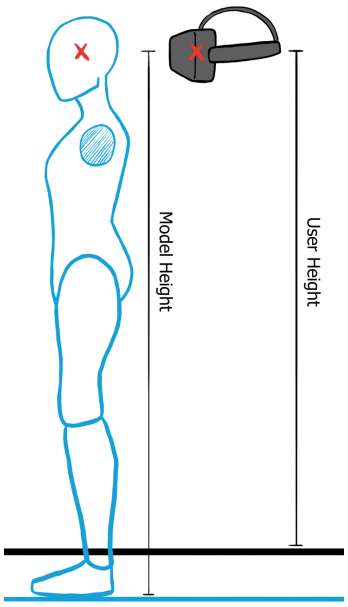


Fig. 7. Height of the model compared to the users height. The red X represent the eye position of the model and HMD

3.3 Speed modifier alterations

Currently the speed modifier used to increase the speed at which the foot is moving is based on the magnitude of a step and the speed of the user multiplied by 10. The value used to multiply the magnitude and HMD_{Speed} was found by trying out different values until I found something that worked, but could likely have been solved by using a low pass filter instead.

Furthermore, the speed modifier would likely be more consistent if it was purely based on either magnitude or HMD_{Speed}. If the HMD_{Speed} is calculated using the following formula. By increasing the amount of frames between current and previus frame, for example 10 frames between the current and the previus frame, HMD_{Speed} would likely have a value that could be used as the speed modifier.

$$HMD_{Speed} = \frac{(CurrentPosition - PreviusPosition)}{TimeBetweenFrames}$$

3.4 User Feedback

The avatar wasn't able to be tested statistically due to the global health restrictions, but was tried by 3 users of both genders with varying body types, who commented on how the avatar self-embodiment felt.

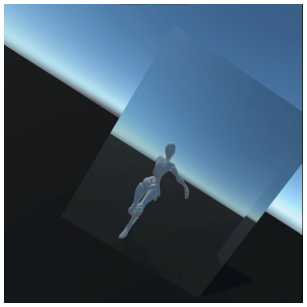
The general observation was that the user's could recognise the avatar in the virtual environment as a representation of their own body. Furthermore, if they tried both model variants, they tended to prefer the model representing their own gender.

All users thought the ground was at an expected height, and did not seem to notice that it was raised or lowered compared to their height. However, they noted that when they looked down at their avatar it felt bigger than they would have expected. One user pointed out that while their first person perspective seemed too big, when they looked in the mirror the avatar seemed to be the correct size.

This might be due to an issue of distance perception accuracy in VR. Many studies have found that the perception of spatial dimensions in VR tend to be skewed, especially in low-fidelity environments that do not represent their current real environment [18] [17], and that the view of an avatar self-embodiment can assist in distance judgement whether it is in first person or third person [16]. Nevertheless, all three sources still conclude that due to the low-fidelity there is still a significant underestimation of distances.

3.4.1 ***Issues with leaning and bending.*** Since the avatar adjust for imbalance via taking a step in the direction of imbalance, the model does not have the ability to lean from side to side while standing in place, or bending forward without bending the legs, see figure 8 and 9. This was pointed out by all the users. However, due to the scope of the project it was never intended for the avatar to be able to lean or bend down.

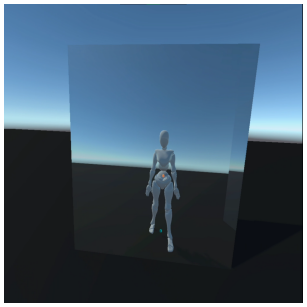
(a) User successfully leaning



(b) Avatar failing to lean

Fig. 8. User and avatar leaning

(a) User successfully stepping



(b) Avatar successfully stepping

Fig. 9. User and avatar stepping

Additionally, most users were not satisfied with the distance the legs could be apart, as the avatar would often end in what was described as "*Doing a split*" where the feet would be in a wide stand.

The feedback from the users lead to a small revision of the system, where the threshold for when to move F_S instead of F_D was lowered as to avoid the wide distance between the feet.

4 CONCLUSION

This paper presents one possible way of simulating an avatar self-embodiment using 3-points of tracking. This is achieved via joint's and simulated physics, which controls the lower-body to adjust for imbalance. The legs are attached to the feet, which follow a path towards their target position. Thus, the feet are not moved via physics, but is rather a rudimentary form of Inverse Kinematics.

4.1 Future Work

There are many things that could be done to improve the avatar. One improvement could be for all of the movement to be controlled by adding force and torque to the body parts instead of the current method.

By adding torque and force to the joints connecting the body parts, the avatar could simulate muscle movement, which would move the body parts according to the current balance evaluation described in 2.3.

A muscle system could be programmed using the configurable joint in Unity and adding a torque and force to rotate and position the body parts correctly. However, it would be a tremendous amount of work to find all the correct values or limits for the body to function as intended. It would likely be preferable to train a machine learning algorithm that could apply the correct values, similar to that of a walking bipedal robot.

ACKNOWLEDGMENTS

Thank you to the users trying and giving feedback on the current avatar self-embodiment and Claus Brøndgaard Madsen for supervision through out the project.

REFERENCES

[1] *Anthropometry*. John Wiley Sons, Ltd, 2009, ch. 4, pp. 82–106.
[2] *Kinetics: Forces and Moments of Force*. John Wiley Sons, Ltd, 2009, ch. 5, pp. 107–138.
[3] Adobe. Mixamo, characters. https://www.mixamo.com/#/?page=1&type=Character, 2021. accessed May 27, 2021.
[4] de Leva, P. Adjustments to zatsiorsky-seluyanov's segment inertia parameters. *Journal of Biomechanics 29*, 9 (1996), 1223–1230.
[5] DeepMotion. Virtual reality tracking. https://www.deepmotion.com/virtual-reality-tracking.
[6] DeepMotion. How to make 3 point tracked full-body avatars in vr. https://blog.deepmotion.com/2018/04/30/how-to-make-3-point-tracked-full-body-avatars-in-vr/, 2018.
[7] Elizabeth, F. Mechanical testing of foot and ankle implants. In *Mechanical Testing of Orthopaedic Implants*. Elsevier, 2017, pp. 1–1.
[8] Erdmann, D. Fast ik. https://assetstore.unity.com/packages/tools/animation/fast-ik-139972, 2019. accessed May 27, 2021.
[9] Facebook. horizon. https://www.oculus.com/facebook-horizon/, 2020. accessed May 27, 2021.
[10] Games, G. Dead and buried. http://gunfiregames.com/dead-and-buried, 2016. accessed May 27, 2021.
[11] Heong, K. C. Disassembly vr. https://store.steampowered.com/app/973700/Disassembly_VR/, 2019. accessed May 27, 2021.
[12] Inc, R. R. Rec room. https://recroom.com/, 2016. accessed May 27, 2021.
[13] Inc, V. Full-body tracking. https://docs.vrchat.com/docs/full-body-tracking. accessed May 27, 2021.
[14] Inc, V. Vrchat on steam. https://store.steampowered.com/app/438100/VRChat/, 2017. accessed May 27, 2021.
[15] Lang, P. Inverse kinematics in dead and buried. http://root-motion.com/2016/06/inverse-kinematics-in-dead-and-buried/, 2016. accessed May 27, 2021.
[16] Mohler, B. J., Creem-Regehr, S. H., Thompson, W. B., and Bülthoff, H. H. The Effect of Viewing a Self-Avatar on Distance Judgments in an HMD-Based Virtual Environment. *Presence: Teleoperators and Virtual Environments 19*, 3 (06 2010), 230–242.
[17] Phillips, L., Ries, B., Interrante, V., Kaeding, M., and Anderson, L. Distance perception in npr immersive virtual environments, revisited. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization* (New York, NY, USA, 2009), APGV '09, Association for Computing Machinery, p. 11–14.
[18] Phillips, L., Ries, B., Kaeding, M., and Interrante, V. Avatar self-embodiment enhances distance perception accuracy in non-photorealistic immersive virtual environments. In *2010 IEEE Virtual Reality Conference (VR)* (2010), pp. 115–1148.
[19] ROOTMOTION. Advanced character animation system for unity. http://root-motion.com/, 2014. accessed May 27, 2021.