ΤΜΗΜΑ ΕΠΙΚΟΙΝΩΝΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΜΕΣΩΝ
ΣΧΟΛΗ ΚΟΙΝΩΝΙΚΩΝ ΚΑΙ ΑΝΘΡΩΠΙΣΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
«Ολόσωμη παρακολούθηση και ενσωμάτωση σε VR»

του/της
Νικόλαου Μαρμαρά

Επιβλέπων: Δρ. Πρωτοψάλτης Αντώνιος

Μέλη της Τριμελούς: 1) Δρ. Κωνσταντίνος Καρπούζης

2) Δρ. Νικόλαος Πλόσκας

Μάρτιος, 2024

Contents

Chapter 1 Introduction 7
Chapter 2 Full-Body tracking 9
2.1 Outside-In Tracking 9
2.2 Inside-Out Tracking 9
2.3 Self-Tracking Technology 11
2.3.1 Lighthouse Tracking System 12
2.3.2 VICON 13
2.3.3 Microsoft Kinect 14
2.3.4 Sony PlayStation Eye Toy 15
Haritora X 16
2.3.5 Tundra Trackers 16
2.3.6 SlimeVR 17
2.3.7 Sony Mocopi 18
2.3.8 Vive Trackers 1.0, 2.0 and 3.0 19
2.3.9 Vive Ultimate Trackers 20
2.4 Comparative Discussion 21
Chapter 3 Embodiment in Full body Avatars 23
3.1 Embodiment and animation of Humanoid Avatars 25
3.2 Inverse Kinematics (IK) 25
3.3 IK calibration using VR editor 27
3.4 Heuristic IK Solvers 27
3.4.1 FABRIK 28
3.4.2 CCD 29
3.5 IK for Unity's Animation Rigging Package 30
3.5.1 Constraint for Hands and Feet 32
3.6 Notable IK solvers 34
3.6.1 SAFullBodyIK 34
3.6.2 Final IK 34
3.6.3 Mecanim IK 36
3.6.4 Cinema IK 37
3.6.5 Easy IK 38
3.6.6 Fast IK 39
3.7 Comparative Discussion 40
Chapter 4 JARVRIKS (Just Another Robust VR IK Solution) IK 42
4.1 The JarvriksFullBodyIK class 44
4.1.1 JarvriksFullBodyIK and its associated classes 45
4.1.2 Saving and Loading functionality 47
4.1.3 Calibration functionality 50
4.2 JarvriksFullBodyIKJob: C# Graphic Job 50
4.2.1 IK Solve Analysis 53
4.2.2 JarvriksFullBodyIKJob methods 55
4.3 Editors for IK customization 55
4.3.1 IK editor: The Class JarvriksFullBodyIKEditor 57
4.3.2 VR IK Editor: The class JarvriksVREditor 58
Chapter 5 Use Case Validation 59
5.1 Use Case 1: SuperHot 59
5.1.1 Scene Creation 60
5.1.2 Weapons and interaction 60
5.1.3 Enemies Implementation 61

5.1.4 Nav Agent 61
5.1.5 Nav Mesh Surface 62
5.1.6 Convert Skinned Mesh Renderer to Mesh Renderer 62
5.1.7 SuperHot script 63
5.1.8 Time Manager 63
5.1.9 Remarks 63
5.2 Use Case 2: Beat Saber 63
5.2.1 Scene and effects 63
5.2.2 Lightsaber 64
5.2.3 3D object modification within Unity 64
Create BeatSaber logic 66
BlockData: This class is used to store data for each block that appears in the game. 66
5.3 Use Case 3: Soccer game 66
5.3.1 Scene creation 67
5.3.2 Implementation 67
5.3.3 Remarks 68
5.4 Menu UIs 68
5.4.1 Basics of Interaction 68
5.4.2 Interaction Process 69
5.4.3 Script MenuSpawn 69
5.4.4 Script SceneManager 69
5.4.5 ScreenFader script 69
5.4.6 Evaluation of Full Body Tracking 70
5.4.7 Measuring Tracking Accuracy 71
5.4.8 Tracking Accuracy Results 72
Chapter 6 Conclusion 76
6.1 Future Work 77

Acknowledgements

Words cannot express my gratitude to my professor Antonis Protopsaltis for his invaluable patience and feedback. I would be remiss in not mentioning my family, especially my parents, and sister. Their belief in me has kept my spirits and motivation high during this process.

I am also grateful to my friends and coworkers, especially my office mates, for their editing help, late-night feedback sessions, and moral support. Their advice to clear my mind and focus on the end goal proven to be life saving for this project.

"Not every hero wears a mask. Some heroes save the day in the simplest of ways. By just being there for us, or letting us know we are believed in"

Abstract

This thesis presents an extensive examination of motion tracking technologies, focusing on the comparative analysis of outside-in and inside-out tracking methodologies, pivotal in advancing virtual reality (VR) applications. By delving into the intricacies of inverse kinematics (IK), the study identifies the limitations and strengths inherent to existing IK solutions within Unity, such as Mecanim, CinemaIK, and others, and proposes the JARVRIKS Full Body IK system as a novel, integrated solution designed to address the identified gaps.

The JARVRIKS Full Body IK system, developed for Unity, leverages advanced IK algorithms and motion capture data to facilitate the creation of lifelike human avatars. Through meticulous engineering, it introduces a series of improvements over traditional IK systems, including enhanced accuracy, flexibility, and user-centric calibration processes. Its innovative architecture is detailed, demonstrating its applicability in VR environments and its compatibility with various motion tracking technologies.

To validate the effectiveness and versatility of the JARVRIKS Full Body IK system, this thesis documents its implementation across three distinct VR game case studies: Superhot, Beat Saber, and a soccer simulation focused on lower body movement. These applications serve as practical demonstrations of the system's capability to provide accurate, responsive, and immersive user experiences. Through quantitative and qualitative analysis, the research evaluates user interaction, movement fidelity, and the overall impact of the JARVRIKS Full Body IK system on gameplay immersion and realism.

Conclusively, this thesis contributes to the field of VR development by offering a comprehensive IK solution that bridges the gap between complex motion tracking technology and accessible, immersive VR content creation. The JARVRIKS Full Body IK system not only enhances the development workflow for VR game designers but also elevates the user experience in VR applications, paving the way for future innovations in virtual embodiment and interactive design.

This abstract encapsulates the thesis's scope, methodologies, key findings, and contributions to the field of VR and game development. It succinctly conveys the research's purpose, the development and application of the JARVRIKS Full Body IK system, and its potential impact on the industry.

Abstract in Greek

Αυτή η διπλωματική περιέχει μια εκτενή εξέταση των τεχνολογιών παρακολούθησης κινήσεων, επικεντρώνοντας στη συγκριτική ανάλυση των μεθοδολογιών παρακολούθησης εξωτερικής προς εσωτερική και εσωτερικής προς εξωτερική, καθοριστικές για την προώθηση των εφαρμογών εικονικής πραγματικότητας (VR). Με την εμβάθυνση στις περιπλοκότητες της αντίστροφης κινηματικής (IK), η μελέτη εντοπίζει τους περιορισμούς και τα πλεονεκτήματα που είναι εγγενή στις υπάρχουσες λύσεις IK μέσα στο Unity, όπως το Mecanim, το CinemaIK και άλλα, και προτείνει το σύστημα JARVRIKS Full Body IK ως μια νέα, ενσωματωμένη λύση σχεδιασμένη για να αντιμετωπίσει τα εντοπισμένα κενά.

Το σύστημα JARVRIKS Full Body IK, αναπτυγμένο για το Unity, εκμεταλλεύεται προηγμένους αλγορίθμους IK και δεδομένα παρακολούθησης κίνησης σε πραγματικό χρόνο για να διευκολύνει τη δημιουργία ρεαλιστικών ανθρώπινων αβατάρ. Μέσω μεθοδικής μηχανικής, εισάγει μια σειρά βελτιώσεων σε σχέση με τα παραδοσιακά συστήματα IK, συμπεριλαμβανομένης της βελτιωμένης ακρίβειας, ευελιξίας και διαδικασιών βαθμονόμησης κεντρικές προς τον χρήστη. Η αρχιτεκτονική του περιγράφεται λεπτομερώς, αποδεικνύοντας την εφαρμοσιμότητά του σε περιβάλλοντα VR και τη συμβατότητά του με διάφορες τεχνολογίες παρακολούθησης κίνησης.

Για την επικύρωση της αποτελεσματικότητας και της ευελιξίας του συστήματος JARVRIKS Full Body IK, η διατριβή καταγράφει την εφαρμογή του σε τρεις διακριτές περιπτώσεις μελέτης παιχνιδιών VR: Superhot, Beat Saber και μια προσομοίωση ποδοσφαίρου εστιασμένη στην κίνηση του κάτω μέρους του σώματος. Αυτές οι εφαρμογές λειτουργούν ως πρακτικές αποδείξεις της ικανότητας του συστήματος να παρέχει ακριβείς, ανταποκρινόμενες και εμβυθιστικές εμπειρίες χρηστών. Μέσω ποσοτικής και ποιοτικής ανάλυσης, η έρευνα αξιολογεί την αλληλεπίδραση του χρήστη, την ακρίβεια κίνησης και τη συνολική επίδραση του συστήματος JARVRIKS Full Body IK στην εμβύθιση και τον ρεαλισμό του παιχνιδιού.

Συμπερασματικά, η διπλωματική συνεισφέρει στον τομέα της ανάπτυξης VR προσφέροντας μια ολοκληρωμένη λύση IK που γεφυρώνει το χάσμα μεταξύ της περίπλοκης τεχνολογίας παρακολούθησης κίνησης και της προσιτής, εμβυθιστικής δημιουργίας περιεχομένου VR. Το σύστημα JARVRIKS Full Body IK όχι μόνο βελτιώνει τη ροή εργασίας για τους σχεδιαστές παιχνιδιών VR αλλά επίσης ενισχύει την εμπειρία του χρήστη σε εφαρμογές VR, ανοίγοντας τον δρόμο για μελλοντικές καινοτομίες στην εικονική ενσάρκωση και την αλληλεπίδραση.

Chapter 1 Introduction

In the evolving landscape of virtual reality (VR) and augmented reality (AR), full-body tracking stands as an important technology enabling immersive and interactive experiences that closely mimic real-world interactions. At the heart of this technology lies the concept of embodiment—the psychological sensation or perception of a user being inside, and having control over, a body within a virtual environment. This not only enhances the sense of presence within a virtual space but also significantly impacts the user's interaction with the virtual world.

Full-body tracking technology is a cornerstone in bridging the gap between the physical and virtual worlds. It enables the translation of a user's movements into the digital realm, allowing for a synchronized and interactive experience within virtual environments. This technology not only captures the positional data of the user's limbs and body but also interprets these movements to animate a virtual avatar in real-time. The accuracy and responsiveness of full-body tracking are crucial for maintaining immersion, as any delay or inconsistency can disrupt the user's sense of presence within the virtual environment.

Embodiment takes the concept of presence a step further by integrating the user's cognitive and emotional responses to being represented in a virtual form. It is not merely about the visual replication of physical actions in a virtual space but encompasses the user's perception of the avatar as an extension of their own body. This phenomenon is central to achieving a fully immersive VR experience, as it directly affects how users interact with the virtual environment and how they interpret sensory feedback from it. A strong sense of embodiment can enhance the realism of the VR experience, leading to more meaningful and engaging interactions within the virtual world.

However, achieving a high level of embodiment is fraught with challenges. One of the main hurdles is the technological limitation in accurately and consistently tracking complex human movements. The human body can perform a wide range of motions, and capturing these nuances requires sophisticated tracking systems that can process vast amounts of data in real-time. Furthermore, the subjective nature of embodiment means that what works well for one user may not be effective for another, adding another layer of complexity to designing universally effective full-body tracking systems. This variability makes it challenging to design a one-size-fits-all solution that consistently delivers a high level of embodiment across all users. Instead, systems must be adaptable and customizable to accommodate the diverse needs and preferences of users to ensure an immersive and comfortable virtual reality experience for everyone.

Moreover, the psychological aspects of embodiment raise important considerations. The uncanny valley [1]—a term used to describe the discomfort felt when a human-like figure looks or moves almost, but not exactly, like a real human—can significantly impact the user's sense of embodiment. Achieving a balance where the avatar is responsive and lifelike without being unsettling is crucial for maintaining immersion and comfort in VR.

These challenges underscore the importance of our research in exploring new methodologies and technologies for full-body tracking and embodiment. By addressing the limitations of current systems, we aim to push the boundaries of what is possible in VR and AR, creating more immersive, intuitive, and interactive virtual experiences that can be applied across various domains.

Historically, full-body tracking technologies have been categorized into two main approaches: outside-in and inside-out tracking. Outside-in tracking, often reliant on external hardware to monitor and interpret user movements, has been instrumental in early advancements in VR and AR. Inside-out tracking, on the other hand, utilizes sensors and cameras attached directly to the device or the user's body, offering a more versatile and unencumbered experience. Despite their contributions to the field, both approaches come with inherent challenges. Issues such as latency, occlusion, limited range of motion capture, and the requirement

for an unobstructed environment have often impeded the seamless integration of the user's physical actions with their virtual counterparts.

These challenges highlight the necessity for innovative solutions that can provide accurate, efficient, and intuitive full-body tracking. Such solutions are crucial not only for enhancing user experience but also for expanding the applicability of VR and AR technologies across various sectors including gaming, healthcare, and professional training. This underlines the significance of our research's, focusing on a novel inverse kinematics (IK) solution designed to address these prevalent issues by improving tracking accuracy and user embodiment in virtual environments.

The forthcoming chapters of this thesis delve deeper into the realm of full-body tracking and embodiment, beginning with an extensive review of existing technologies in Chapter 2. This includes a detailed exploration of both outside-in and inside-out tracking systems, setting the groundwork for understanding the complexities and limitations of current methodologies. Chapter 3 shifts focus towards the embodiment within avatars, elucidating how a user's perception of virtual embodiment can significantly affect their VR experience. In Chapter 4, we introduce "JARVRIKS," our innovative IK solution aimed at overcoming the limitations of existing tracking technologies, thereby enhancing user embodiment and interaction within VR environments. Chapter 5 aims to validate our novel IK approach developing three use case games, facilitating upper and lower body tracking. These games, inspired by popular titles like Superhot and Beat Saber, alongside a soccer game for lower body tracking, serve as practical use cases to demonstrate the capabilities and potential of our solution. Finally, Chapter 6 culminates with the conclusions drawn from our research, summarizing the key findings, implications for the field of VR and AR, and suggesting avenues for future work.

By addressing the known challenges within full-body tracking and embodiment, this thesis contributes to the advancement of VR and AR technologies, paving the way for more immersive, intuitive, and interactive virtual experiences.

Chapter 2 Full-Body tracking

Full Body Tracking refers to the ability to detect and visualize all the user's body movements within a virtual world. Using a range of sensors and cameras, the system can transfer the user's physical movements to the virtual character, offering a more realistic and immersive experience.

Full body tracking and embodiment in virtual reality (VR) is a technology that captures and translates the user's physical movements and visualizes them in an avatar within the virtual environment. As such the virtual avatar animates with the same movements as the real user. The user usually wears sensors or trackers on different parts of his/her body, such as your hands, feet, and head. As the user moves around, these sensors send motion data to a computer or VR system, which then accurately replicates those movements in the game or simulation the user is experiencing. This technology allows for a much more immersive and interactive VR experience, as it feels like the user is present in the virtual world, able to interact with any entity using the entire body. There are two main approaches in full body tracking: Outside-in and Inside-out tracking.

2.1 Outside-In Tracking

Outside-In Tracking involves placing external sensors or cameras around the play area that detect and track the movement of the VR headset and any other tracked devices, like controllers or specialized equipment for body tracking. These external devices continuously monitor the position and orientation of the VR headset and controllers from the outside, hence the term "Outside-In." This method can provide accurate tracking by having a fixed reference point outside the user's body. However, it requires a dedicated setup with sensors placed in specific locations around the room, which can limit the play area and increase the setup time and complexity.

For example, the Lighthouse tracking system used by HTC VIVE PC VR headsets is cited as a classic example of outside-in tracking. While this system does rely on external devices placed in the player's immediate area, those devices don't collect any information themselves. Recent studies reveal that incorporating full-body avatars can significantly enhance the sense of embodiment and immersion, indicating a trend towards the use of marker-based systems for multiplayer modes due to their robustness in tracking multiple users in real-time [2]. Additionally, innovative approaches aim to simplify full-body tracking by integrating upper-body VR tracking systems with a single external webcam, offering a more accessible and streamlined setup process [3].

2.2 Inside-Out Tracking

Inside-Out Tracking, on the other hand, reverses this approach. The sensors or cameras are located on the VR headset itself, and they use the headset's position relative to the environment to determine the user's movement. This method often involves the headset scanning the environment and using fixed points or features in the room to track movement without the need for external

sensors. Inside-Out tracking systems are more portable and easier to set up since they don't rely on external hardware placed around the play area. However, their accuracy and responsiveness can vary based on the environment's complexity and the technology used.

A novel inside-out magnetic tracking system optimized for VR/AR applications demonstrates this approach's potential. It can achieve high-definition 6-DoF tracking with minimal jitter and latency, making it suitable for personal area tracking. This system is compact and portable. [4]. Moreover, an evaluation [5] of the Oculus Rift S tracking system in room-scale virtual reality environments provides insight into the capabilities and limitations of inside-out tracking technologies, which indicates that while the Oculus Rift S achieves an average translation error of about 1.83 cm and an average rotation error of about 0.77°, it still falls short of the accuracy attainable with motion capture systems, highlighting the trade-offs involved in inside-out tracking systems [5]. See Table 1: The table contains the differences between the Inside-Out and Outside-In Tracking.

	Inside-Out	Outside-In
Setup Complexity	Simpler setup, relying on sensors within the VR HMD. Reducing the need for external hardware.	Requires more complex setup with external sensors or cameras placed around the play area.
Mobility and Portability	Portable and easier to set up in different environments since it doesn't rely on external sensors.	Less portable due to the need for external sensors or cameras and their specific setup requirements.
Tracking Range	Limited by the range of sensors within the headset, which may result in less precise tracking in larger play areas.	Potentially offers larger tracking volumes, depending on the number and placement of external sensors.
Tracking Precision and Accuracy	Possible limitations in tracking accuracy, particularly when the headset moves out of the sensors' field of view or in environments with poor lighting conditions.	Generally, it offers higher precision and accuracy as external sensors can track the headset and controllers more precisely.
Occlusion Handling	Susceptible to occlusion when objects or body parts obstruct the sensors on the headset.	Can provide better occlusion handling since external sensors have a broader field of view and can detect movements even when parts of the body are obstructed.
Hardware Requirements	Requires a VR headset equipped with built-in sensors for tracking.	Requires external sensors or cameras, which may involve additional costs and setup.
Cost	Cost-effective since it doesn't require additional external sensors or cameras.	More expensive due to the need for external hardware.
Latency	Can have lower latency as tracking data is processed internally within the HMD.	Latency may be slightly higher due to the need to transmit tracking data from external sensors to the VR system.
Multi-User Support	May be limited in multi-user scenarios, especially in larger spaces where multiple headsets might interfere with each other.	May offer better support for multi-user experiences, especially in environments with multiple external sensors.

Table 1: The table contains the differences between the Inside-Out and Outside-In Tracking

2.3 Self-Tracking Technology

Self-tracking technology [6] represents an advancement in virtual reality (VR) and motion capture systems, addressing many of the limitations associated with traditional tracking methods. It integrates Computer Vision (CV) and Inertial Measurement Unit (IMU) technologies to offer a comprehensive solution for precise, real-time motion tracking without the need for external base stations.

IMU technology is instrumental in the enhancement of tracking and motion analysis solutions that gauge positional and orientational changes of objects and limbs absent of any external point of reference. Utilizing data on acceleration and rotational velocity, IMUs are at the heart of these systems, contributing to an object or individual's path computation [7]. Noteworthy research [8] has unveiled the development of a cost-efficient IMU-based framework capable of simultaneously providing real-time 3D visualization of movements and orientation tracking across different body parts, all while surmounting communication barriers by using a uniquely developed 2.4 GHz protocol. This study points to the far-reaching possibilities that IMU technology holds for broadening the scope of precise and accessible motion tracking across diverse applications such as virtual reality, athletic sciences, and therapeutic interventions.

While IMUs offer significant benefits, they are not free from constraints. Issues such as error accumulation resulting in drift are notable, prompting the need for periodic recalibrations to assure enduring accuracy [9]. In side-by-side evaluations with high-accuracy solutions like the VICON system, IMUs have been shown to exhibit some level of positional estimation error, signaling a need for further refinement. Additionally, despite their adaptability and ease of operation in various settings, IMUs alone cannot supply absolute locational data without additional spatial positioning aids [10].

The fusion of IMUs with other systems, including monocular cameras or sophisticated algorithms for noise mitigation, presents a promising avenue toward resolving these limitations [11] [12]. This collaborative approach has the potential to fortify the resilience and precision of IMU-based motion tracking against environmental variances and under diverse usage cases. With their ample application possibilities, IMUs stand out as a pivotal factor in driving progress within the domain of motion tracking, hinting at their inevitable proliferation across multiple fields.

At the core of self-tracking technology is Computer Vision, which leverages two wide-angle cameras integrated into the tracking device to visually sense the environment, like how VR headsets operate. This visual data, when combined with IMU's relative positioning information, enables the system to accurately determine the tracker's location and movements within a space. AI algorithms play a crucial role in processing this data, calculating the tracker's absolute position in real-time with high precision.

One of the most significant benefits of self-tracking technology is its portability and lightweight design, eliminating the need for cumbersome setup processes associated with base stations. This not only makes the system more user-friendly but also greatly enhances its mobility, allowing for a more flexible and accessible tracking experience. The high precision of this technology ensures that every movement is captured with great accuracy, enhancing the realism and

immersion of VR experiences.

Despite its many advantages, self-tracking technology does face some challenges. One of the main issues is its lower occlusion resistance, meaning that the tracker's performance might degrade when its line of sight with the environment is obstructed. Additionally, the system requires sufficient lighting to function optimally, as the cameras rely on visual data to track movements. The HTC VIVE Ultimate Tracker is a prime example of self-tracking technology in action. Launched in 2024, it showcases the potential for motion tracking. With the ability to support up to five trackers simultaneously and requiring only a small, USB-sized wireless receiver for connection, it offers a seamless and efficient solution for capturing full-body movements in VR, making the technology both intuitive and reliable for users.

2.3.1 Lighthouse Tracking System

The evolution of monitoring technologies in motion capture has been marked by a transition from high-cost, outside-in specialized systems, like VICON, OptiTrack, and VisualEyez, to more accessible, inside-out off-the-shelf VR solutions, such as the HTC Vive. This shift reflects significant technological advancements and a growing need for versatile, cost-effective motion capture options across various fields, including robotics and VR. The Vive system, with its innovative lighthouse tracking mechanism and the subsequent development of improved open-source algorithms, exemplifies this evolution. It addresses the limitations of earlier systems by offering a more affordable, adaptable solution without sacrificing accuracy or precision. This evolution not only democratizes access to high-quality motion capture technology but also fosters innovation in areas previously constrained by cost and complexity.

The HTC Vive's Lighthouse system represents an advancement in motion capture technology for virtual reality (VR) applications, employing a sophisticated approach to tracking. Utilizing synchronized light sweeps emitted by base stations, this system, through trackers equipped with photodiodes, calculates the user's position by measuring the timing of these light pulses. This method, known as Angle-of-Arrival (AoA), is enhanced by an IMU to maintain smooth trajectory movements, offering a blend of efficiency and cost-effectiveness for motion capture.

While the original algorithms of the Vive system were engineered to prioritize smooth tracking, they could compromise precision in more dynamic scenarios. However, the evolution of tracking algorithms and calibration processes has significantly amplified its application potential, especially in the realm of robotics. For instance, studies like [32] delve into the system's spatial tracking performance, suggesting standards for velocity limits to ensure reliable tracking [13]. Another research, [33], evaluates the Lighthouse system's accuracy and repeatability for motion tracking, demonstrating its capability for millimeter-level accuracy and sub-degree precision in position and orientation measurements, crucial for tasks requiring high precision such as robotic applications [14].

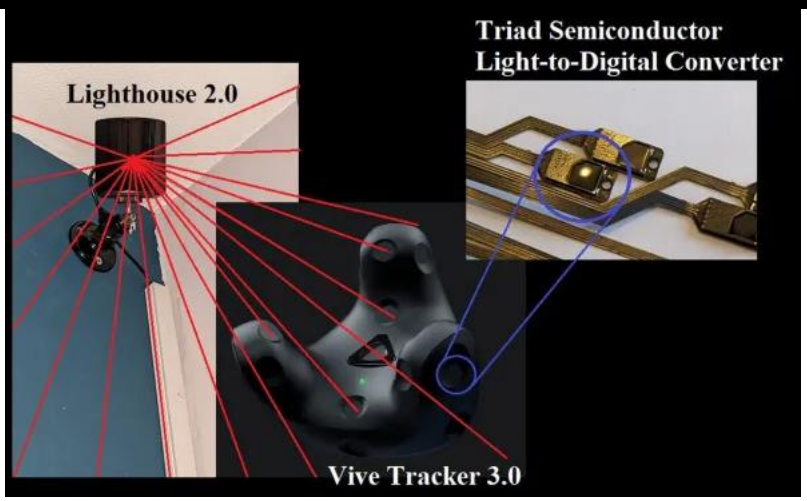

Figure 1: The laser will encode the base station ID as well as the current laser sweep angle. There is no longer a 'timing pulse', the base station does the timing internally and sends the angle directly to the Vive on the laser. [15]

2.3.2 VICON

The VICON system [16] is recognized as a gold standard for outside-in full body tracking due to its accuracy in motion capture. It's frequently used in comparative studies to evaluate other motion tracking technologies. The VICON motion capture system is a sophisticated tool used in various fields, including biomechanics, sports science, and entertainment, to capture and analyze detailed human movements. It operates on an optical tracking mechanism where multiple cameras are strategically placed around the area of interest to track reflective markers attached to the subject. This system provides high-precision, three-dimensional movement data by triangulating the positions of these markers.

While the VICON motion capture system is renowned for its accuracy in tracking human motion, it is confronted with certain constraints. One of the main challenges is its requirement for significant space, which can be an obstacle when deploying the system in confined spaces like manufacturing cells [17]. Cost considerations are another barrier, as the high-priced nature of the VICON system limits its accessibility for cost-conscious projects or for those seeking to validate more economical motion capture solutions like IMUs [18].

Comparative inaccuracies have also been observed when VICON is assessed against other motion tracking systems, particularly in capturing fine motions of smaller joints or in different capturing environments [19] [20]. Covering expansive spaces comprehensively is another area where VICON falls short due to the extensive number of sensitive cameras needed and their limited depth of field, which creates practicality issues [21]. For ergonomic assessments, the system's precision in measuring axial trunk rotation has revealed notable inaccuracies, potentially leading to an underestimation of associated health risks. Inconsistencies emerge as well when examining activities such as the forward leap in children, with VICON displaying variable results depending on the body segment and type of movement being captured [22].

In summary, the VICON motion capture system, despite its stature as a benchmark in the field, comes with a set of limitations ranging from financial, spatial, and technical factors that can affect its application in various contexts.

Figure 2: Vicon has a tradition in the area, as it has been involved in the industry from the beginning. From the first blockbuster movie to use motion capture in 1995 to the creation of the first out of the box head mounted facial capture system [23]

2.3.3 Microsoft Kinect

Microsoft's Kinect [24], released for Xbox 360 and later Xbox One, was a revolutionary device that allowed games and apps to be controlled through body movements. The Microsoft Kinect sensor represents a leap in motion capture technology, combining a depth sensor, a color camera, and a four-microphone array to offer full-body 3D motion capture, facial recognition, and voice recognition capabilities. Its operation hinges on an IR projector and camera duo that utilizes structured light principles to create a depth map by projecting and analyzing a pattern of IR dots, a method developed in collaboration with PrimeSense.

This setup enables the sensor to triangulate each dot's position in 3D space, allowing for accurate depth perception. Calibration issues, potentially caused by environmental factors like heat or vibration, are mitigated through a recalibration technique using a special card. The heart of Kinect's innovation lies in its skeletal tracking system, designed to work seamlessly for every individual without the need for user calibration. By employing a per-pixel, body-part recognition strategy using a deep randomized decision forest classifier trained on an extensive array of synthetic human images [24], Kinect achieves real-time tracking of body joints. This process is highly efficient, running at up to 200 frames per second on the Xbox 360's GPU, ensuring broad applicability across different body sizes and shapes, even in varied household environments. This combination of advanced sensing, machine learning, and image processing technologies enables Kinect to provide a highly interactive and immersive user experience.

The Microsoft Kinect system has faced various challenges highlighted in academic research. These include issues like unrealistic representations and limited customizability [25], which affect its utility, especially in VR educational settings. Concerns over its financial feasibility, alongside physical and psychological discomfort, including simulator sickness, have been noted, impacting its role in VR labs [25]. Moreover, when Kinect is used in commercial games not designed for

rehabilitation, it emphasizes the need for more specialized tools to aid physical rehabilitation at home, aiming to cut healthcare costs [26]. Additionally, Kinect's reliance on 2D cameras for human motion recognition struggles against variations in light intensity and texture, reducing its effectiveness [27]. Inefficiency, resource wastage, and subjective measurement outcomes, particularly when juxtaposed with manual methods like the goniometer, mark significant drawbacks [27]. High production costs and the requirement for controlled clinical settings further limit its application [28]. In tasks involving 3D interaction, Kinect's difficulties in accurately recognizing foot movements bring up questions regarding its efficacy compared to other 3D vision sensors for computer vision tasks [28]. Despite these challenges, Kinect's structured light system, designed for simplistic real-time pose tracking, faces criticism for its sensitivity to lighting and material properties, along with relatively low accuracy [29] [30]

2.3.4 Sony PlayStation Eye Toy

The Sony PlayStation EyeToy serves as a landmark device in the realm of interactive gaming, merging physical movement with virtual gameplay. As evidenced by its incorporation in various rehabilitation settings, its capacity for engaging users in therapeutic tasks underscores its intuitive nature [31]. Although specific advantages and limitations are not explicitly delineated in the literature, the EyeToy's contribution to the video game and virtual reality-based motor rehabilitation systems has gathered positive initial feedback within physical therapy communities, particularly among patients with spinal cord injuries (SCI), traumatic brain injuries (TBI), and amputations.

When it comes to usability, the EyeToy has been pinpointed as more user-friendly than alternatives like the Wii-mote, despite the occasional challenge some may face with its menu navigation [32]. Its role in clinical rehabilitation settings is underlined by its ease of use, which is a significant factor for widespread adoption.

With respect to postural balance, the EyeToy, used alongside the AntiGrav™ game, has shown potential in enhancing balance control via lateral head, body, and arm movements. This points to the device's capability in driving improvements in specific balance metrics [33]. However, it’s important to note that while beneficial outcomes have been observed, there is a chance of experiencing mild simulator sickness, which seems to subside as users adapt through repeated play.

Expanding its therapeutic impact, the EyeToy has also been adapted for virtual reality therapy for children with hemiplegic cerebral palsy, successfully encouraging targeted neuromotor movements [34]. This showcases the EyeToy's versatility and hints at its future acceptance in home-based therapy regimens, potentially boosting upper extremity function.

In summary, the Sony PlayStation EyeToy exemplifies the convergence of interactivity and therapeutic efficacy, albeit with a minor caveat concerning simulator sickness and navigation issues.

Haritora X

Tsthe HaritoraX Wireless [35] tracking system counterparts by utilizing Inertial Measurement Unit (IMU) technology for full-body tracking in virtual reality (VR). This makes the HaritoraX Wireless system more versatile and portable, as it does not require the setup of external hardware beyond the VR headset and controllers. The wireless nature of HaritoraX with IMU technology offers users the freedom to move without the constraints of cables or the need to stay within the sightline of base stations, catering to a wide range of applications from gaming to professional VR simulations where ease of movement and setup flexibility are paramount. They are also compatible with SteamVR tracking.

This IMU-based tracking in the HaritoraX Wireless system, comes with its own set of challenges and trade-offs. While IMUs provide significant convenience and support for untethered VR experiences, they may also be susceptible to drift over time without external reference points to recalibrate positional data. This means that while users can enjoy more freedom of movement, the system might require periodic recalibration or adjustments to maintain optimal tracking accuracy.

Figure 3: HaritonaX Wireless system weared by the user on the thigh, lower leg and the chest [35]

2.3.5 Tundra Trackers

The Tundra Tracker [36] on the other hand, is also a device developed for enhancing full-body tracking in virtual reality environments, and it supports the lighthouse tracking technology as well. Similar to the HaritoraX, the Tundra Tracker is compatible with SteamVR Tracking, meaning it works with the same base stations as the HTC Vive, Valve Index, and other VR systems utilizing lighthouse technology. This compatibility ensures that Tundra Trackers can accurately track the

user's movements by detecting the IR light sweeps from the lighthouse base stations. The Tundra Tracker distinguishes itself with its compact size and lightweight design, aiming to provide a less obtrusive and more comfortable VR experience. Its high compatibility and support for lighthouse tracking technology make it a versatile choice for users seeking reliable full-body tracking in VR applications ranging from gaming to professional development.

Figure 4: The Tundra Trackers [36]

2.3.6 SlimeVR

The SlimeVR [37] system, an innovative tool designed for full-body tracking in virtual reality (VR). The system is adept at enabling a variety of physical activities without the need for cables or external devices. This feature is particularly useful in martial arts training within VR, where users can spar with virtually simulated opponents, gaining feedback from realistic, motion-Captured interactions [38]. SlimeVR also offers advanced data integration capabilities, merging eye-tracking and motion capture information for in-depth analysis of visuomotor coordination [39].The system is adept at enabling a variety of physical activities without the need for cables or external devices. This feature is particularly useful in martial arts training within VR, where users can spar with virtually simulated opponents, gaining feedback from realistic, motion-Captured interactions [38]. SlimeVR also offers advanced data integration capabilities, merging eye-tracking and motion capture information for in-depth analysis of visuomotor coordination [39].

At its core, SlimeVR utilizes IMUs for orientation and movement tracking, deploying a network of sensors like magnetometers, accelerometers, and gyroscopes. These sensors work in tandem to

record body movements in real-time, accurately translating them into a dynamic 3D model, which is then reflected in the VR environment [40].

The research [30] specifically aimed to capture and analyze two fundamental motion postures: walking and walking up and downstairs, to assess the SlimeVR's accuracy and reliability in real-world applications. By focusing on these common movements, the study provides valuable insights into the effectiveness of SlimeVR in capturing intricate human motions, thus validating its utility in VR and motion analysis domains. This analysis not only underscores SlimeVR's potential as a motion capture tool but also highlights its significance in enhancing VR experiences and applications requiring precise movement tracking. The findings of this research contribute to the broader understanding of motion tracking technologies' capabilities and applications in VR environments, offering a solid foundation for future advancements in the field.

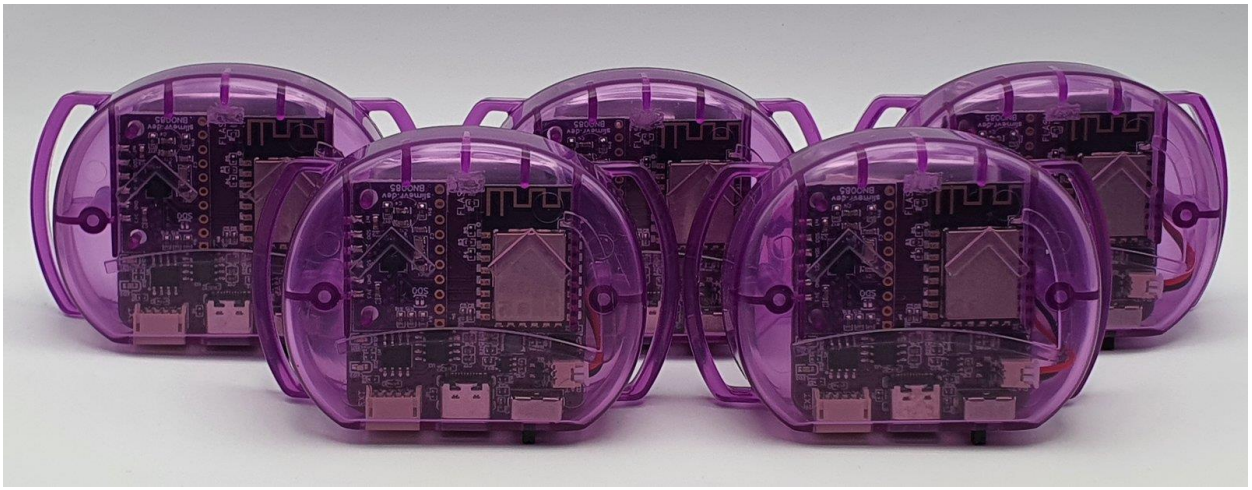

Figure 5: The SlimeVR trackers [37]

2.3.7 Sony Mocopi

Sony's Mobile Motion Capture technology [41] offers a portable solution that transcends traditional studio-bound systems. Unlike conventional motion capture that necessitates a full-body suit with numerous markers and a studio filled with cameras, Sony's approach utilizes just six small, lightweight sensors attached to key points on the body: the head, waist, hands, and legs. This technology allows for the digitization of a person's movements in real-time, in any setting, without the need for specialized attire, making it highly versatile for use indoors, outdoors, or in any location. The core of this technology lies in its use of accelerometers and gyro sensors to capture the acceleration and angular velocity of a person's movements.

These sensors, combined with advanced AI algorithms, enable the system to accurately estimate the position and posture of every joint in the body, including intermediate joints like elbows and knees where sensors are not directly attached. Sony's unique AI model is pre-trained on a vast array of human movements, allowing for precise full-body estimation and the correction of potential position errors that typically arise from integral calculations based solely on acceleration data.

The process involves two primary steps: a) the estimation of reference joint positions, where the system digitizes human movement, by integrating acceleration and gyro sensor data, using integral calculation to

determine three-dimensional positions; Sony's AI model then estimates these joint positions, effectively correcting any accumulated position errors, b) the estimation of intermediate joint positions, due to the complex structure of the human body and the high degree of freedom at the joints, simple geometric calculations are insufficient for accurately determining the positions and postures of intermediate joints; Sony's AI model comes into play again, naturally interpolating joint positions based on training from diverse human movements, ensuring a comprehensive estimation of joint positions across the entire body.

Figure 6: Sony mocopi capture trackers [41]

2.3.8 Vive Trackers 1.0, 2.0 and 3.0

The evolution of Vive trackers from versions 1.0 to 3.0 has marked significant advancements in virtual reality tracking technology. While specific academic papers detailing each version's nuances were not identified, insights into the latest iteration, the HTC VIVE Tracker 3.0, underscore the technological strides made in capturing human movements for full-body visualization in VR environments. The VIVE Tracker 3.0 is celebrated for its enhanced measurement accuracy, especially when utilized alongside recommended base station configurations and during activities characterized by lower pedaling frequencies. This tracker exhibits a commendable balance of precision in comparison to established systems like the Vicon, highlighting its utility in creating immersive and accurate VR experiences [42].

This technology ensures the accurate capture and nuanced interpretation of complex human motions, making it applicable across a spectrum of uses from ergonomic evaluations to the creation of immersive

gaming experiences. A particular study leveraging these trackers, in conjunction with an inverse kinematic model, offers detailed insights into joint angles, demonstrating their effectiveness in conducting ergonomic risk assessments [43]. This highlights the trackers' ability to produce precise and responsive VR interactions, essential in both entertainment and therapeutic settings. Additionally, their application in a comparative study that investigates the influence of virtual self-representation on spatial judgments further underscores their role in enhancing virtual interaction fidelity, thereby marking a significant contribution to the evolution of motion capture technology and its expansive utility [44].

Figure 7: Vive Trackers 1.0, 2.0, 3.0

2.3.9 Vive Ultimate Trackers

Advancing Towards a New Generation of Tracking Solutions, the Vive Ultimate Trackers [45] are an autonomous tracking devices, compact and easy to carry that eliminates the need for external base stations. The system works with obstructed views and requires a well-lit environment to function effectively. To overcome the drawbacks associated with prior technologies, a new approach known as autonomous tracking has been developed. This technique relies predominantly on Computer Vision, with an IMU serving as a supplementary component.

This device gathers relative positioning data via IMU technology while also capturing spatial information through two wide-angle cameras integrated into the tracker (akin to those found on VR headsets). It then employs artificial intelligence to accurately determine the tracker's exact location in real-time. A notable feature is the capability to simultaneously accommodate up to five VIVE Ultimate Trackers, all connecting through a wireless receiver as compact as a standard USB dongle. By sidestepping the typical inaccuracies associated with IMUs and eliminating the need for setting up base stations, this self-tracking device offers users a more straightforward and precise way to achieve VR full-body tracking, thereby enhancing the intuitiveness and reliability of virtual reality experiences.

Figure 8: HTC Vive Ultimate Trackers(called Self Tracking Trackers on Development) [45]

2.4 Comparative Discussion

A detailed analysis of VICON application and accuracy was performed by comparing the VICON system with the ViMove wireless motion sensor system [25], measuring lumbar region inclination motion in the sagittal and coronal planes. They found a clinically acceptable level of agreement between the two systems, indicating the VICON system's reliability for clinical and research applications. This study emphasized the VICON system's accuracy in capturing complex body movements, a critical factor in clinical diagnoses and treatment efficacy evaluations.

Further, [46] evaluated the Kinect sensor against the VICON system for gait kinematics analysis. Through a Cartesian calibration procedure, they aimed to ensure that kinematics data were written in the same reference frame, and joint centers were consistently defined for both systems. Their findings validated the Kinect sensor for gait kinematics analysis, underscoring the VICON system's role as a gold standard in motion capture technology and its versatility in validating emerging, less expensive motion capture solutions.

Additionally, the VICON system's integration with various software for different applications, such as autonomous drone flight, further illustrates its adaptability and precision. A study [47] highlighted how the VICON motion capture system was integrated with Paparazzi UAV autopilot software to enable autonomous flight of the AR Drone 2.0, demonstrating the system's utility beyond human motion capture and into robotics and unmanned vehicles.

The comparison between Vive trackers and the Vicon system reveals insightful contrasts and similarities in their application for motion analysis, particularly in virtual reality (VR) environments. The VICON system is renowned for its precision and is often regarded as a gold standard in motion capture technology, whereas Vive trackers offer a more accessible, less costly alternative with substantial accuracy.

[48] highlighted the agreement between Vive and VICON systems in monitoring lumbar postural changes, showing that Vive trackers are accurate within 0.68 ± 0.32 cm translationally and 1.64 ± 0.18° rotationally when compared to the VICON system. This suggests that Vive trackers could

serve as a cost-effective alternative for motion analysis in clinical and possibly other settings, considering their accuracy and affordability. Another work [42] evaluated the HTC VIVE Tracker 3.0 against the VICON system for full-body tracking in VR, showing that spatial differences between Vive and VICON were smallest at the lowest pedaling frequency of 80 rpm, with differences of 10.4 mm ± 4.5 mm at the knee and 11.3 mm ± 5.1 mm at the ankle, using the rectangular arrangement of base stations recommended by the manufacturer. Additionally, a custom tracking method based on RGB-D images [49] demonstrated a mean per joint position error of 11.7 mm compared to a VICON system, validating its accuracy for full-body tracking in VR.

These studies underscore the evolving landscape of motion capture technology, where traditional, high-cost systems like Vicon are being challenged by more accessible solutions like Vive trackers. The Vicon system continues to be revered for its unmatched precision and reliability across diverse applications, from clinical research to animation and game development. Conversely, Vive trackers, with their ease of setup and lower cost, present a viable option for VR applications requiring good enough accuracy and flexibility, including VR gaming, rehabilitation, and sports training.

While the Vicon system's precision is critical for applications demanding the highest levels of accuracy, the Vive trackers offer a practical solution for developers and researchers seeking to incorporate motion capture into their projects without the significant investment required by traditional systems. The choice between these systems ultimately depends on the specific requirements of the project, including the needed accuracy level, budget constraints, and application domain.

This following Table 2: All tracker technologies analysissummarizes the comparison of the previous analyzed tracker technologies. [6]

Tracker	Tracking Technology	Weight (grams)	Battery Life	Connection Method	Base Station Setup	Maximum Space Range	Maximum Space Range	Supported VR Devices
VIVE Ultimate Tracker	Inside-Out Tracking	94	7	Wireless signal receiver, supports 2.4GHz and 5GHz wireless network connection	No	High	10 meters	VIVE series (Plans to support other standalone and PCVR headsets in the future)
VIVE Tracker 3.0	Lighthouse	75	7.5	Through wireless signal receiver	Yes	High	10 meters	Various
Tundra Tracker	Lighthouse	46-50	7-9	Through wireless signal receiver, can receive multiple trackers depending on the model	Yes	High	10 meters	Various
Sony Mocopi	IMU	8	10	Through Bluetooth connection to smartphone	No	Low	Bluetooth coverage range	Various
SlimeVR	IMU	50	15	Through 2.4 Ghz Wi-Fi connection	No	Medium	Wifi coverage range	Various
HaritoraX Wireless	IMU	17	20	Bluetooth or wireless signal receiver	No	Medium	Bluetooth coverage range	Various

Table 2: All tracker technologies analysis

Chapter 3 Embodiment in Full body Avatars

In the realm of virtual body modelling and simulation, the use of a rigid multibody system is fundamental. This system, consisting of interconnected rigid links, is pivotal for creating lifelike human posture control in computer animations. By modelling these structures, developers can simulate realistic movements, crucial for a variety of usage, such as general use, animated storytelling. The architecture of multibody systems, characterized by hierarchical links and joints, mirrors the complexity and diversity of human motion, laying the groundwork for sophisticated simulation applications.

Building upon these rigid multibody foundations, manipulators like robot arms and animated characters are further refined. The hierarchical nature of these models, where motion in one joint influence the subsequent ones, underpins the realistic animation of humanoid avatars in platforms like Unity. These avatars employ advanced animation techniques such as Motion Matching to achieve a level of realism and naturalness in movement, crucial for applications across entertainment, virtual reality, and educational tools designed for individuals with disabilities. The nuanced study of joint mechanics enhances the fidelity of these avatars, making them indispensable tools in digital human representation.

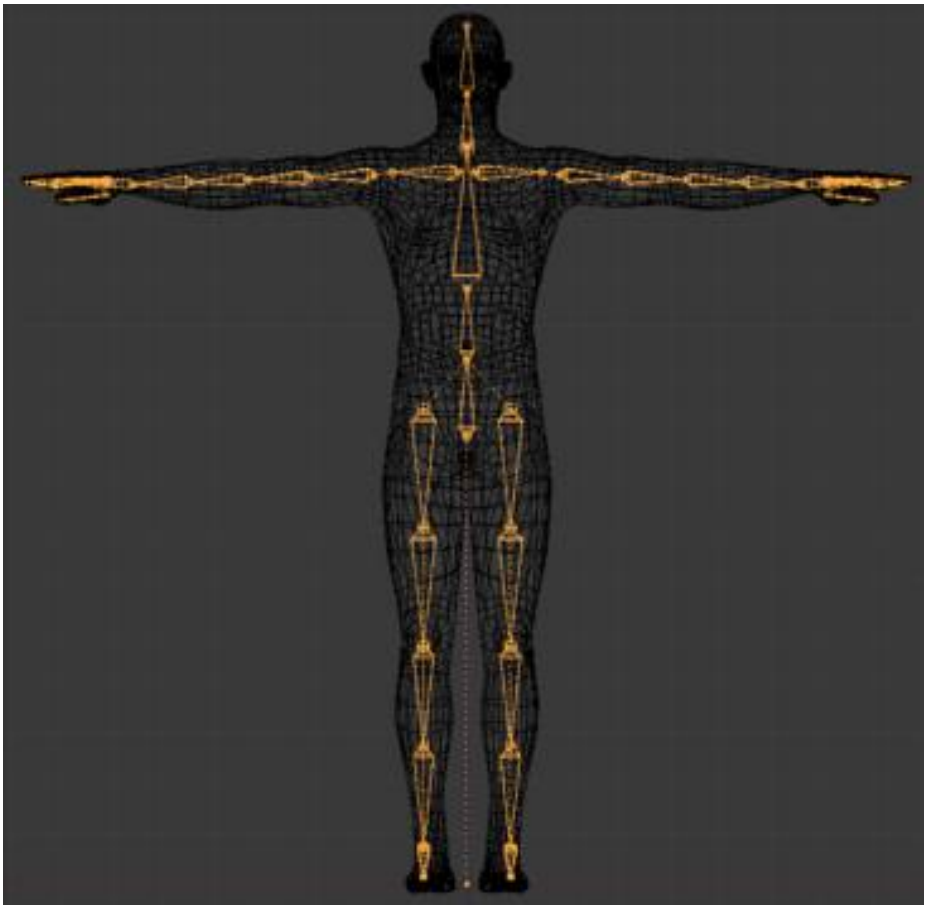

Figure 9: Body with Skeleton armature https://docs.unity3d.com/560/Documentation/Manual/BlenderAndRigify.html

The intricate modelling of joints and body segments introduce the realm of Inverse Kinematics

(IK), a cornerstone in achieving realistic and dynamic posture and movement in virtual characters. IK techniques enable animators and developers to simulate complex motions with high accuracy, applicable across various fields from gaming to ergonomics and rehabilitation. Adjusting the degrees of freedom of the multibody links in articulated figures, allows this computational approach to bridge the gap between desired outcomes and mechanical constraints, enriching the animation and simulation landscape with lifelike virtual beings.

Traditional Forward Kinematics (FK), compute the end position of the animated skeleton bones, of the rigged 3D model, by applying geometric transformations (transform, rotate, scale) on the skeleton bones. Such procedure is not suitable with full-body tracking systems, as the end positions of the bones are known from the tracking devices. If one decides to use FKs in such systems the animator would painstakingly adjust each joint in sequence from the base to the extremity. IKs simplify this process by allowing the end point to be defined first. This shift is particularly crucial for embodying characters with trackers, as it aligns virtual movements with physical motions captured in real-time. In the context of using trackers for motion capture or virtual reality (VR) applications, IK is indispensable. Trackers provide positional data from specific points on a user's body, but translating this data into smooth, naturalistic movement across the entire character requires the sophisticated calculations that IK offers. By focusing on the end goal - such as the position of a hand or foot - IK algorithms work backwards to determine the most efficient and realistic paths through which all connected joints should move. This ensures that virtual characters accurately mirror the complex, fluid motions of their human counterparts, enhancing immersion and interaction in digital environments. Moreover, IK facilitates the animation of characters interacting with objects or environments in a physically plausible manner. Whether it's reaching for an object, walking across uneven terrain, or simulating weight and resistance, IK provides the flexibility and precision necessary to achieve believable interactions. This capability is essential in creating more engaging and interactive VR experiences, where players expect their movements and actions to have a direct and realistic impact on the virtual world.

The calibration of full-body tracking systems within VR environments further amplifies the realism and immersion in digital experiences. Ensuring accurate user-avatar synchronization is critical, particularly in VR applications where engagement and user experience are paramount. Despite the sophistication of IK solutions, the absence of standardized calibration methods poses challenges, necessitating innovative approaches to tailor animations and interactions to individual users. Techniques like Damped Least Squares (DLS) and learning-based models [6] offer pathways to enhanced motion accuracy, though they also highlight the balance between generalizability and personalized calibration in immersive technologies.

Transitioning into the domain of Virtual Reality (VR) editors, these tools represent a paradigm shift in interactive design and content creation. By allowing users to directly manipulate and configure virtual spaces, VR editors unlock new potentials in architectural design, education, and game development. These editors not only democratize content creation by simplifying the development process but also enable novel pedagogical approaches and immersive training experiences. However, the innovation brought forth by VR editors also comes with its set of challenges, including accessibility and resource demands, underscoring the ongoing evolution in how we interact with digital environments.

Each of these segments, from body modelling to VR editing, illustrates the multifaceted approach that is required to accurately embody, simulate user presence and interaction in the virtual world. The

progression from theoretical modelling to practical application in VR underscores the dynamic interplay between technology and creativity, paving the way for future advancements in digital simulation and interaction.

3.1 Embodiment and animation of Humanoid Avatars

Humanoid avatars in Unity are designed to offer realistic and natural animations, leveraging advanced techniques such as Motion Matching [50]. Research in Motion Matching for character animation, Pose Estimation for educational use, and De-anonymization in the Metaverse showcases innovations in VR and avatar technologies, particularly enhancing animations for more realistic avatar movements in virtual settings. Meanwhile, Jun Lee's research on real-time pose estimation caters to creating interactive avatars for metaverse home training, with significant implications for disabled or autistic individuals' education and assistance. This approach enhances versatility and smoothness in character animations, making them particularly suitable for various fields including entertainment, virtual reality, and the metaverse. These avatars are further optimized for interactive applications, enabling control over game avatars through movements using pose estimation technology [51]. This feature is beneficial for educational and assistance programs targeted at disabled or autistic individuals, promoting a more engaging user experience. Additionally, humanoid avatars provide a realistic movement signature that can be utilized in applications like de-anonymization attacks, showcasing their utility in both development and research contexts [52]. The research into de-anonymization attacks within the metaverse explores the security implications of virtual environments. Such studies highlight the dual-use nature of avatars, serving both development and research purposes, by demonstrating their application in areas such as security analysis and user identification

3.2 Inverse Kinematics (IK)

Inverse kinematics (IK) is a computational procedure used to determine the posture of an articulated figure or robot, by calculating the necessary adjustments to each joint's degree of freedom to achieve a specific task, such as reaching a designated position with an end-effector or tracker. This entails solving a system of nonlinear equations that relates the joint parameters to the position and orientation of the end-effector in space.

For a two-link robotic arm as an example, the IK problem can be translated into finding the angles θ_1 and θ_2 that position the end-effector at a target point p = (x, y). This involves solving the equations that relate these angles to the end-effector's position through the link lengths l_1 and l_2 [53].

In IK, the positions of trackers are translated into joint angles or positions using mathematical models like the homogeneous coordinate transformation, which consider inter-joint dependencies and spatial relationships established by methods such as the Denavit-Hartenberg convention. These models allow the calculation of vector equations for the system, based on geometric relationships within the mechanism, accounting for variable length levers and additional constraints imposed by the mechanical structure [53]. Parallel computational algorithms extend these principles to multi-legged systems, leading to more efficient solutions through parallel

processing [54]. Recursive and modular algorithms, as with the application of DeNOC matrices, streamline the process for systems with multiple closed kinematic loops [55]. Closed-form solutions for the Gough-Stewart platform are derived from cartesian dynamic model elements, reducing the computational complexity further [56].

This approach to IK has proven essential not only in computer graphics and robotics but also in ergonomics, rehabilitation medicine, protein structure prediction, and other fields, due to its ability to accurately model complex, articulated structures in digital and physical environments.

Calibration in full-body tracking and embodiment within virtual reality (VR) environments is paramount for achieving a high degree of accuracy and realism in user-avatar synchronization. It ensures that the virtual representations of users reflect their real-world movements accurately, enhancing the immersive experience and the efficacy of VR applications in fields such as rehabilitation, training, and entertainment. Despite the critical role of calibration, many widely recognized inverse kinematics (IK) solutions do not inherently include a calibration approach. This omission often leads to discrepancies between a user's actual physical movements and the avatar's actions in the virtual space, potentially diminishing the user's sense of presence and the overall effectiveness of the VR application.

An exploration into various IK solutions, with a focus on the Damped Least Squares (DLS) method, reveals an effective strategy for lower-body motion accuracy. However, the reliance on DLS does not directly address individual differences in user physiology, which could lead to less personalized avatar movements [57]. While DLS provides a robust mathematical framework for solving IK problems, ensuring smooth and realistic motions, it inherently lacks a mechanism for personal body dimension calibration, which could limit its effectiveness in applications requiring high personalization.

In contrast, a personalized VR motor rehabilitation system incorporates a calibration process tailored to the user's specific body metrics, using HTC Vive trackers. This approach significantly enhances the accuracy of motion tracking, crucial for rehabilitation purposes [58]. However, this method's dependence on manual input for calibration may introduce variability based on the user's ability to accurately replicate the required poses or measurements, potentially affecting the system's overall precision.

The idea behind AvatarPoser is to use a method based on learning to figure out entire body positions from just a few motion cues. It enhances the accuracy of where arm joints are supposed to be by using optimization techniques combined with inverse kinematicst [57]. This innovative approach allows for a broader application scope by reducing the need for extensive motion capture equipment. Nevertheless, the reliance on predictive models may introduce inaccuracies due to model assumptions or limitations in capturing complex or highly individualized movements, thereby affecting the fidelity of the avatar's representation.

Each of these methods highlights the trade-offs between accuracy, personalization, and the practicality of implementation within VR systems. The absence of a standardized calibration process in many IK solutions underscores the need for further research and development in this area, aiming to bridge the gap between generic IK algorithms and the nuanced requirements of personalized full-body tracking in VR environments.

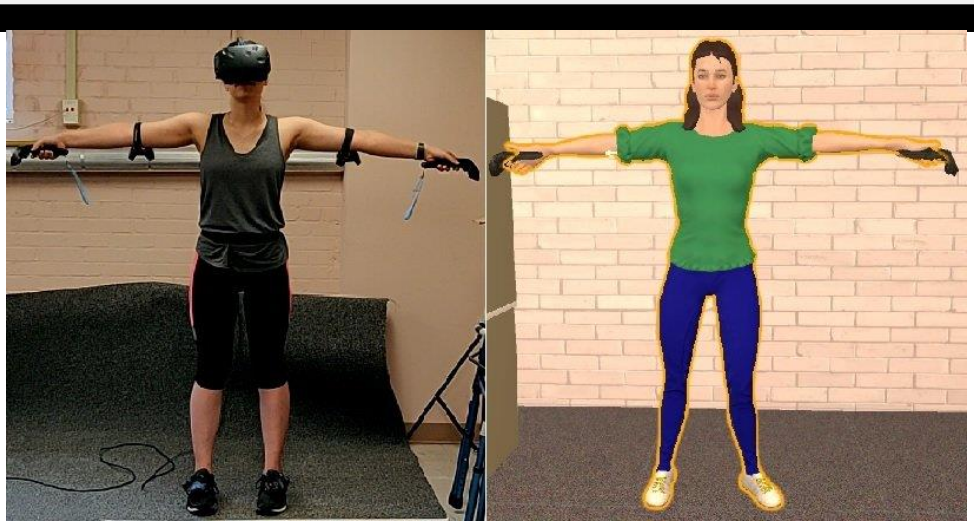

Figure 10: T-pose made by participants for tracker calibration and avatar generation [59]

3.3 IK calibration using VR editor

Virtual Reality (VR) editors represent a transformative technology that allows users to intuitively interact with and calibrate virtual environments, which is particularly beneficial for full-body tracking applications. They enable the creation and manipulation of content in a simulated 3D space, offering an immersive and hands-on experience that traditional interfaces cannot match. For example, a VR editor can improve the calibration process for full-body tracking by providing real-time feedback and adjustments, addressing subjective issues such as personal body dimensions and movement styles [60]. This could enhance the precision of virtual representations and interactions within the environment.

When it comes to calibrating full-body tracking in VR, subjective problems often arise as each user's physicality and perception of movement can differ. VR editors geared towards calibration could provide solutions such as individualized adjustment tools that account for these variances [61]. By integrating advanced calibration methodologies directly within the VR editor, users would be able to fine-tune tracking parameters to match their unique kinematic profiles, leading to more accurate and responsive virtual interactions. This self-calibration could potentially improve the overall user experience by reducing motion sickness and increasing the natural feeling of presence within the VR environment.

VR editors also present the possibility of enhancing the calibration process by using statistical models and machine learning algorithms to predict and adjust for individual user behaviors [62]. Such approaches might include adaptive algorithms that learn from a user's previous interactions to continuously refine the tracking accuracy over time. Furthermore, the integration of multi-sensor data gathering within VR editors could offer comprehensive tracking capabilities, capturing not just the visible body parts but also inferring the positions of occluded limbs [63]. These methods promise more robust performance and adaptive tracking that can respond to dynamic changes in the user's environment and behavior, further reducing the subjective discrepancies in full-body VR tracking.

3.4 Heuristic IK Solvers

In the field of animation and computer graphics, simulating human body movements with high accuracy is a complex challenge. Two heuristic algorithms, FABRIK (Forward And Backward Reaching Inverse

Kinematics) and CCD (Cyclic Coordinate Descent), are particularly notable for their ability to efficiently tackle inverse kinematics problems. These methods provide innovative solutions for determining the positioning of limbs in three-dimensional space, which is crucial for producing realistic and smooth animations. This section will examine the principles and applications of FABRIK and CCD, highlighting their significant contributions to enhancing motion capture and character animation technologies.

3.4.1 FABRIK

The FABRIK (Forward And Backward Reaching Inverse Kinematics) IK solver algorithm [64], , presents a novel heuristic method that eschews traditional angle rotation calculations in favor of a more efficient positional approach. This method relies on iterative forward and backward adjustments of joint positions within a chain, minimizing system error by focusing on finding points along lines between joints. Initiated from the chain's end effector and progressing towards the root (and vice versa), FABRIK calculates the positions of each joint based on their distances to subsequent joints, significantly reducing computational time and complexity.

A key aspect of FABRIK is its ability to determine the reachability of a target by comparing the total length of the joint chain against the distance to the target. If the target is within reach, FABRIK performs two main stages in an iteration: first, adjusting joint positions from the end effector towards the root, and second, from the root back towards the end effector, ensuring the manipulator base remains unchanged if required. This approach allows FABRIK to achieve closer approximations to the target position with each iteration, terminating when the end effector is sufficiently near the target or when a maximum number of iterations is reached without significant improvement.

Traditional Inverse Kinematics (IK) solvers, which are used for calculating the necessary joint parameters that enable an articulated figure or a robot to arrive at a specific target position and orientation, encompass a variety of methods. One widely employed strategy is the inversion of the Jacobi matrix, which is applied to obtain numerical solutions in IK problems, typically within the context of robotic arms [65]. Iterative algorithms are also prevalent, delivering solutions for complex models, such as humanoid arms with multiple degrees of freedom, by refining the approximations until the end-effector's calculated position closely matches the target position [66].

Another method is the normal form approach, which provides benefits by effectively dealing with kinematic singularities encountered in non-redundant robot mechanisms, superior to singularity-robust and null-space based techniques [67]. Analytical methods, including ANFIS, offer exact solutions by utilizing artificial neural networks and fuzzy logic, which attain a balance between computational efficiency and error resilience, particularly for robots like the PUMA 560 [68].

Particle Swarm Optimization (PSO) is a heuristic-based method that exploits swarm intelligence to explore the solution space for IK problems, adjusting parameters such as inertia weights and swarm sizes to ensure convergence to the optimal solution [69] .Furthermore, neuro-fuzzy systems have been tailored to solve inverse kinematics in robot manipulators like the Denso 6-DOF, cementing the union of traditional mechanics with modern computational intelligence [70] .

In comparison, FABRIK (Forward And Backward Reaching Inverse Kinematics) is an alternative

iterative algorithm that solves IK problems by reaching both forward towards the end-effector target and backward towards the base of the kinematic chain. It's known to converge faster and is computationally efficient, making it advantageous for real-time applications like virtual reality and complex manipulations where traditional IK solvers might struggle due to computational load and convergence difficulties [71]. FABRIK-R, an extension of FABRIK, excels in handling highly constrained situations, providing a robust solution to singularities that traditional methods may not effectively address [72].

FABRIK's simplicity and efficiency stem from its direct handling of joints as points in space, avoiding complex mathematical operations often associated with traditional IK solvers.. This makes FABRIK particularly effective for real-time applications, including motion capture and animation of multibody systems with multiple end effectors. By extending the algorithm to accommodate models with various kinematic chains and constraints, FABRIK can solve complex IK scenarios with improved accuracy and reduced computational overhead, offering a versatile and powerful tool for animators and developers in creating realistic and dynamic character movements [71]. The FABRIK solver has even been mentioned in the literature concerning virtual reality applications in mental health, denoting its versatility and potential for wide-ranging VR applications [73]

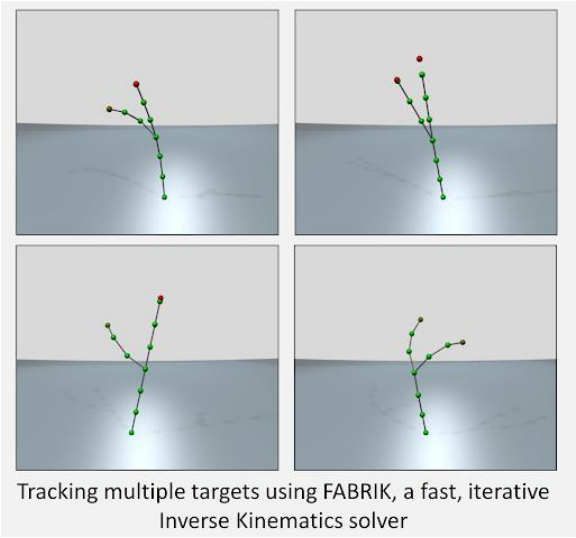

Figure 11: FABRIK: a fast, iterative solver for the inverse kinematics problem [64]

3.4.2 CCD

The Cyclic Coordinate Descent (CCD) algorithm [74] is an iterative method designed for solving inverse kinematics (IK) problems by adjusting the positions and orientations of joints in a chain to move the end effector as close as possible to a target. The process involves sequentially transforming each joint variable to minimize discrepancies in both position and orientation, with the distance between the end effector and the target being recalculated after each iteration to determine if the desired proximity has been achieved. To prevent the algorithm from entering endless loops in pursuit of unattainable targets, a maximum iteration limit is established. This procedure is repeated for each joint in the chain, working backward from the end effector to the root, ensuring all joints are updated in a single iteration.

The algorithm's stability is noted, although it is also mentioned that CCD can produce oscillations and discontinuities due to significant angle rotations. Since CCD operates on a local scale, adjusting one joint at a time, the implementation of global constraints is a challenging task. This limitation can lead to unrealistic movements, particularly in models of highly articulated characters where maintaining plausible motion is critical. Advantages of the CCD algorithm include its simplicity and effectiveness in real-time applications, making it suitable for interactive environments. The CCD IK algorithm is known for its suitability for real-time applications, including virtual reality (VR), due to its speed and efficiency. In particular, the CCD algorithm can be optimized to take advantage of high-speed GPU parallel programming, allowing for significantly faster computations compared to CPU-based processing. This optimization can generate virtual view images up to 10 times quicker than traditional CPU processing, which is highly beneficial for real-time VR applications [75]. Moreover, other implementations of the CCD IK algorithm have been able to achieve real-time performance on hardware like embedded CUDA GPUs and ARM CPUs, reaching speeds up to 46 FPS at VGA image resolution, demonstrating that it is well-suited for interactive VR environments [76].
However, its difficulty with global constraints and the potential for generating non-smooth motions underscore the need for careful application and possibly supplemental solutions or adjustments when using CCD in complex animation or robotics projects.

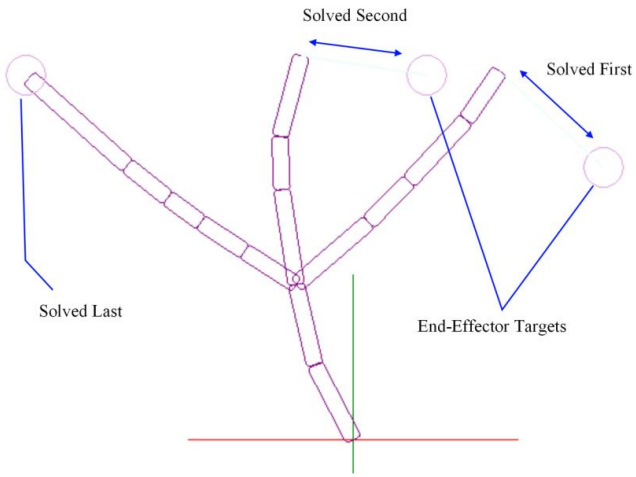

Figure 12: Multiple end-effectors (last end-effector updated using CCD is primary)

3.5 IK for Unity's Animation Rigging Package

Animation Rigging Package [77] offers a tool for creating and editing character movements, particularly useful in VR game development, as it allows movement creation based on inverse kinematics (IK). Unity's Animation Rigging toolkit equips users with a collection of tools that enable the procedural manipulation of character animations directly within Unity. This innovative package empowers users to craft custom rigging setups that can dynamically alter animations post-processing. It's an interactive system that not only provides extensive tutorials to facilitate learning but also allows for further customization and functionality through C# scripting. By enhancing Unity's own animation capabilities, the package serves as a versatile option for creating intricate, responsive, and modifiable animations that can be adjusted in real-

time during gameplay [78].

The implementation of the Animation Rigging package in Unity contains the addition of C# Graphics Jobs starting from Unity version 2018.2 This feature extends the capabilities of animation programming by utilizing the C# Job System, enabling developers to write multithreaded code that significantly improves performance through parallel processing. ,C# Graphics Jobs are designed to run custom scripts at particular moments within the *PlayableGraph*. This feature takes advantage of multicore processors to boost animation quality and performance, ensuring operations remain safe across different threads. The *PlayableGraph* itself is a structure used in Unity to manage and play animations, where these custom jobs can be inserted to optimize how animations are processed and displayed

The application of C# Graphics Jobs [79] spans various domains, from achieving character animations to creating custom animation algorithms and mixers. This flexibility allows for detailed manipulation of character anatomy, such as implementing foot-locking features or dynamically animating character tails without traditional animation clips. Furthermore, it enables the creation of specialized *LookAt* functionalities targeting specific bones. The core of this system is the *AnimationScriptPlayable* and *IAnimationJob* structs, which allow for the manipulation of animation data within the *PlayableGraph*. The animation processing is thoughtfully divided into two passes—*ProcessRootMotion* for root transform motions and *ProcessAnimation* for all other animation aspects—providing a structured approach to animation processing.

The significance of multithreading in this context cannot be overstated. By allowing animation tasks to be parallelized, C# Graphics Jobs alleviate the computational load on the main thread, facilitating smoother and more responsive gameplay. This parallel processing is crucial for the Animation Rigging package, as it enables the handling of complex, real-time animations without compromising game performance. The ability to manually process input streams and control the flow of animation data offers developers the opportunity to optimize their animation systems efficiently, tailoring them to the specific needs of their projects.

Incorporating C# Graphics Jobs into the Animation Rigging package thus represents a significant advancement in Unity's animation capabilities. It not only broadens the scope of what developers can achieve with animations but also enhances the overall performance and responsiveness of animated characters and objects within the game environment. My thesis delves into the technical intricacies of this integration, underscoring its impact on the effectiveness of the Animation Rigging package and its contribution. This detailed analysis sheds light on the practical benefits of multithreading in animation programming and its pivotal role in the evolution of animation systems within Unity.

A critical feature within this package is the Bone Renderer component, which, when activated makes the skeletal structure of the character visible within the editor. This visibility is crucial for direct manipulation and customization of the bones' appearance. The Bone Renderer allows the list of all bones associated with a character or object and adjustment of their shape, color, and size.

To effectively utilize the components provided by the Animation Rigging package in Unity, a structured approach is necessary, starting with the creation of a new rig that supplements the original animation rig, which facilitates the foundation of an additional layer of animation control. The recommendation to instantiate a new Rig GameObject for each new constraint underscores

the flexibility this system offers, allowing animators to selectively override aspects of existing animations without disrupting the original animation framework.

Upon establishing the first Rig, Unity automates the inclusion of a Rig Builder component on the character's root GameObject. This critical component serves as the central hub for managing multiple Rigs, each of which can be individually toggled on or off within the Rig Builder's Rig Layer component. This layering mechanism provides a granular level of control over animation behaviors, enabling precise adjustments and experimentation without permanent alterations to the base animation.

To apply specific constraints, a GameObject is designated as a child under the Rig object, serving as the anchor point for the constraint component. Essentially, you can assign one component to act as a guide or controller for another, enabling targeted and refined animation behaviors. This method ensures animations move smoothly and accurately, according to the setup defined within the rigWhile source objects, such as targets, are positioned, for organizational clarity, within the same GameObject hierarchy, Unity's flexible system allows for alternative configurations to accommodate diverse animation requirements and creative preferences.

3.5.1 Constraint for Hands and Feet

3.5.1.1 *Two Bone IK Constraint*

The Two Bone IK Constraint [80] is particularly useful for animating articulated structures like arms and legs, which typically consist of three main joints: the upper joint (shoulder/hip), middle joint (elbow/knee), and end joint (hand/foot). This constraint applies inverse kinematics to these three joints, enabling the end effector (the hand or foot) to reach a specified target position and orientation. By doing so, it automatically calculates the appropriate angles for the middle and upper joints to achieve this, resulting in naturalistic movement patterns that would be cumbersome to animate manually. The "target" in an IK setup refers to the final position and orientation that the end effector (the terminal part of the chain, such as a hand, foot, or any other point that follows the movement) aims to reach. The IK system adjusts the rotations of all joints in the IK chain to ensure that the end effector matches the target's position and rotation as closely as possible. In essence, the target is the goal for the end effector's movement. The "hint," on the other hand, is an optional aid used by certain IK solvers to guide the bending direction or orientation of the IK chain, primarily to resolve ambiguities in how joints should bend. For example, in a human leg or arm, the hint helps to ensure that the knee or elbow bends in the correct direction. The hint is typically used in chains with three or more joints and serves to improve the naturalness and predictability of the joint orientations throughout the IK calculations.

3.5.1.2 *Multi-Parent Constraint for the Shoulder*

Using the Multi-Parent Constraint [81] on shoulder bones of an articulated structure allows better freedom of movement and a nuanced control of shoulder movements, by blending influences from multiple parent objects, that ultimately provide natural animation. The Multi-Parent Constraint allows an object, in this case, the shoulder joint, to have its transformation (position, rotation, and scale) influenced by more than

one parent object. This means that the shoulder can simultaneously follow the movements and orientations of multiple parts of the body, such as the spine, chest, or even the head, depending on the desired animation effect.

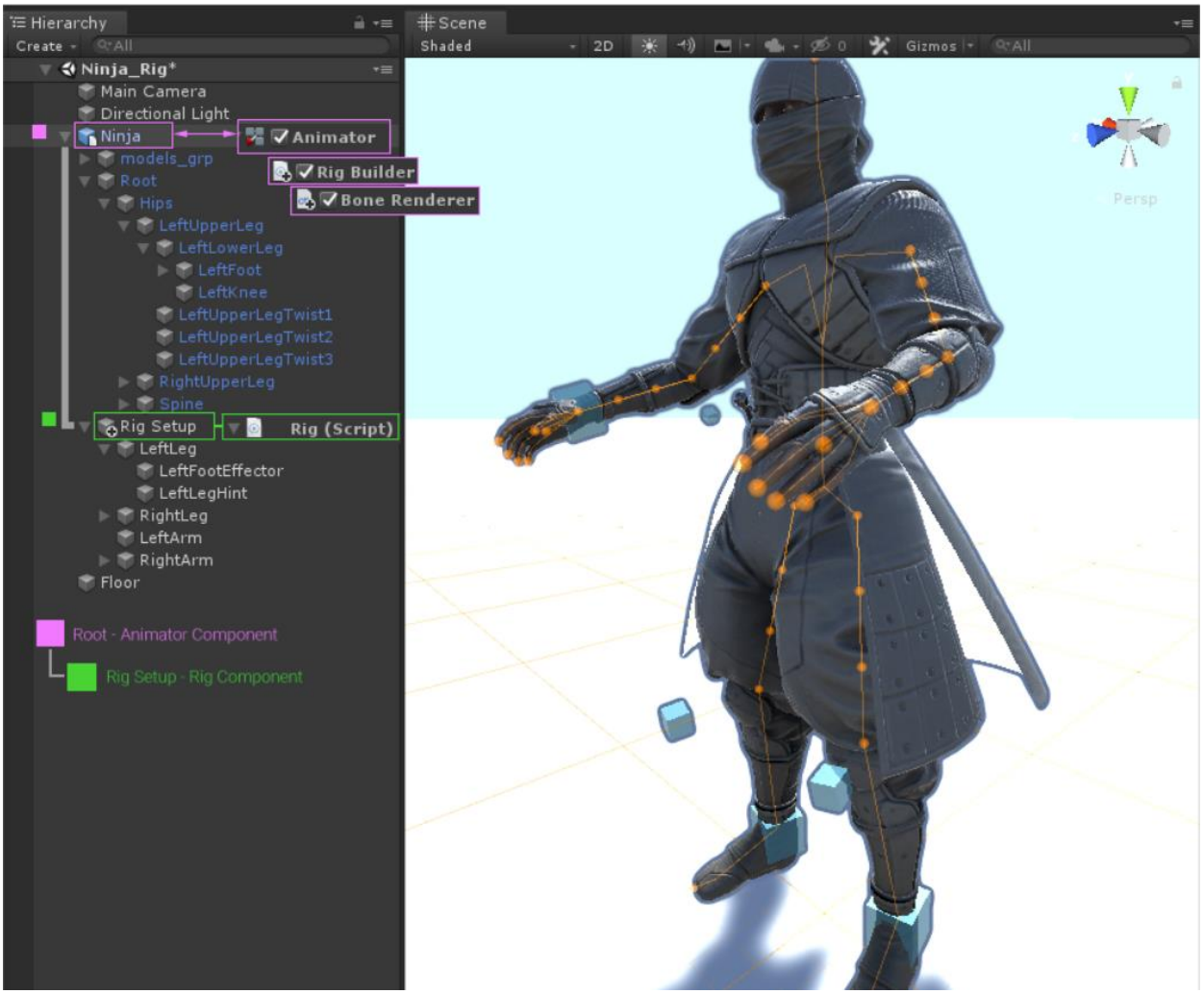

Figure 13: The following illustration represents a schematic overview of the interdepencies between the Animator and the Animation Rig components [77]

The comprehensive study [78] , meticulously outlines the procedural methodology for implementing the package, from basic setup to advanced rigging techniques, such as creating custom rigs that can dynamically adapt to various game scenarios and player interactions.

A significant methodological highlight is the package's reliance on Unity's constraint system, which allows animators and developers to define more complex relationships between different parts of a character's body. This system enables the creation of procedural animations that respond to game physics and user inputs in real-time, presenting a versatile approach to animation rigging beyond the traditional keyframe method.

However, the same studyimplicitly underscores the limitations inherent in the Unity Animation Rigging package. One notable challenge is the requirement for a thorough understanding of Unity's constraint-based rigging system, which can present a steep learning curve for developers and animators new to the platform. Additionally, while the package offers significant flexibility and control over animation dynamics, there is an implicit trade-off in terms of setup and configuration time, particularly for complex characters or animations requiring high levels of precision and customization.

Moreover, the research indicates that although the Animation Rigging package offers robust capabilities, its success largely depends on the user's proficiency in incorporating it into the wider array of Unity tools and functionalities. This encompasses knowledge on optimizing rig setups for better performance, which becomes particularly important in demanding scenarios like VR or mobile gaming, where achieving a smooth frame rate is essential This includes an understanding of how to optimize rigging configurations for performance, especially in resource-intensive applications such as VR or mobile gaming, where maintaining a high frame rate is critical.

3.6 Notable IK solvers

3.6.1 SAFullBodyIK

SAFullBodyIK (Skeleton Animation Full Body Inverse Kinematics) [82] is a complex system designed to provide accurate and realistic full-body tracking and animation in virtual environments, particularly within the Unity game engine for VR applications. It focuses on enhancing VR and animation experiences. It's code defines a comprehensive IK system, structuring the human body into various bone and effector classes, including specific categories for body, head, arms, legs, and fingers. It employs a full-neural approach to blend 2D poses from a webcam with upper-body positions from VR headsets into accurate 3D poses. It's particularly noted for its ease of setup, requiring minimal calibration, and its compatibility with current VR applications that support full-body tracking on SteamVR.

Figure 14: SAFullBodyIK: How to setup full body IK with Unity [82]

3.6.2 Final IK

The Final IK [83] plugin for Unity emerges as a comprehensive solution for implementing inverse kinematics in video game development, with a particular focus on animating biped characters. This plugin is distinguished by its inclusion of FullBodyBiped IK (FBBIK) and LookAt IK components. The FBBIK component is instrumental for animating humanoid models, enabling the association of avatars with detailed skeletal mappings that define the roles of various model parts such as legs, arms, head, and body.

This facilitates a streamlined animation process, significantly reducing the complexity typically associated with rigging and animation in Unity .

An essential aspect of Final IK is its initial setup phase, which can be executed manually or automatically, leveraging auto-detection functions for efficient bone segment matching. This capability simplifies the preparation process, allowing developers to bypass extensive rigging setups. The plugin supports the configuration of kinematic chains—critical for defining the movement and constraints of body parts—with the body acting as the root chain from which limbs extend. This hierarchical structure enables precise control over limb movements and orientations, including the application of bend constraints for joint angles. Final IK's auto-detection function automatically identifies and sets up effectors for various body parts, facilitating the manipulation of end-effectors like hands and feet, as well as mid-body and multi-effectors. The differentiation in effector types underscores the plugin's versatility, offering nuanced control over character postures and movements. For instance, the body effector, as a multi-effector, simplifies body positioning by influencing thigh effectors, enhancing the naturalism of animations.

Target definition is another critical feature, allowing developers to specify the goals effectors should achieve, with adjustable rotation and position weights to modulate effector behavior. This flexibility ensures that multiple effectors can be active concurrently, adapting to the dynamic requirements of game scenes. Despite its comprehensive features, Final IK necessitates careful verification of limb rotations and bend directions to ensure the naturalness of movements. Incorrect configurations may require adjustments, either through minor segment rotations or scripting interventions, highlighting the importance of precise setup for optimal results. Additionally, Final IK offers parameters like pull, reach, push, and spine stiffness to fine-tune the interaction between different body parts, enhancing the realism and responsiveness of character animations. The LookAt IK component further complements these capabilities by directing bone sets, such as the spine and head, towards specific targets, contributing to the immersive quality of character interactions.

The VRIK component from Final IK, is an advanced inverse kinematics solution tailored for animating virtual characters within the Unity engine. It stands out for its specialized focus on bipedal characters, facilitating the animation of human-like models with high precision and minimal setup. VRIK automates the process of matching the character's limbs to targets in the 3D space, allowing for realistic and dynamic movement animations. This component simplifies the complexity typically associated with rigging and animating humanoid avatars by offering automatic detection and alignment of bone segments to predefined targets, such as hands, feet, and look-at points, ensuring that the avatar's movements are both natural and responsive to the environment. VRIK's capability to manage full-body movements and interactions makes it particularly useful for developers aiming to create immersive VR experiences, interactive games, or simulations where character realism and responsiveness are paramount. Its integration within the Final IK package empowers creators with the tools to efficiently animate and control character poses, movements, and interactions, significantly enhancing the animation workflow in Unity projects.

Figure 15: FinalIK by RootMotion [83]

A comprehensive analysis [84] was conducted to evaluate the performance of the VRIK solver from the FinalIK package within a virtual motion capture system, specifically tailored for use in virtual reality applications with Unity. Utilizing a simulated environment based on HTC Vive trackers and IMUSim [84] for tracking, the research aimed to discern the accuracy and limitations of VRIK in replicating full body movements, particularly for walking and dancing animations. It was found that while VRIK is fast and widely used, significant limitations emerged when trackers for certain joints, like the hips in walking animations or elbows in dancing animations, were omitted. The absence of a hip tracker, for instance, led to substantial inaccuracies, revealing a systemic flaw where the hip joint would inaccurately revert to its initial position due to its dependency on the positioning of other body parts. Similarly, the exclusion of elbow trackers in dance animations resulted in a decrease in capture quality, underscoring the necessity of bend goals for precise arm movement replication. These findings underscore the critical need for comprehensive tracker setups to ensure the fidelity of captured motions, highlighting the solver's limitations in handling complex animations without adequate tracking support. This study provides valuable insights into the constraints of using VRIK for full body tracking in VR, emphasizing the importance of proper tracker configuration to achieve accurate and realistic motion capture results.

Another full body IK (OcAcO) system developed [85] for VR applications, which calculates upper body movements based on head and hand positions. Although this system demonstrates innovative ways to determine neck forward direction and waist positioning depending on the user's stance, it encounters challenges with accurate neck or shoulder placement. This difficulty suggests a potential limitation in the approach itself or its implementation.

3.6.3 Mecanim IK

Unity's Mecanim animation system [86] stands as a cornerstone for animators and developers seeking a streamlined workflow for character animation within the Unity engine. It employs a sophisticated Animator interface, which orchestrates animations through a state machine, facilitating intuitive control over animation sequences with transitions and event triggers. At its core, Mecanim provides an inverse kinematics (IK) solution specifically tailored for humanoid characters, necessitating a correctly configured avatar to function optimally. This IK functionality operates by dynamically adjusting the positions and

rotations of character joints in real-time, ensuring that end effectors, such as hands and feet, reach target positions and orientations specified by the animator. This allows for more natural and contextually appropriate character movements, such as adjusting a character's foot to rest properly on uneven terrain or reaching for objects in a realistic manner. However, despite its utility, Mecanim IK's effectiveness is bounded by its design constraints, primarily its exclusive compatibility with humanoid characters. This specialization inherently limits its application to avatars conforming to a human-like structure, specifically those with no more than four limbs and limbs comprising up to three joints each [87]. Such a limitation significantly narrows the scope of Mecanim's IK utility, precluding its use for animating characters with complex or unconventional anatomies, like creatures with multiple arms or segmented bodies. This delineation underscores the necessity within the Unity ecosystem for more adaptable IK solutions capable of accommodating a broader spectrum of character forms, to ensure animators and developers can achieve precise and lifelike animations across a diverse range of virtual entities. The current limitations of Mecanim IK, while providing a robust solution for humanoid avatars, spotlight the need for innovation and expansion in Unity's IK capabilities to fully harness the creative potential of 3D animation in gaming and interactive media.

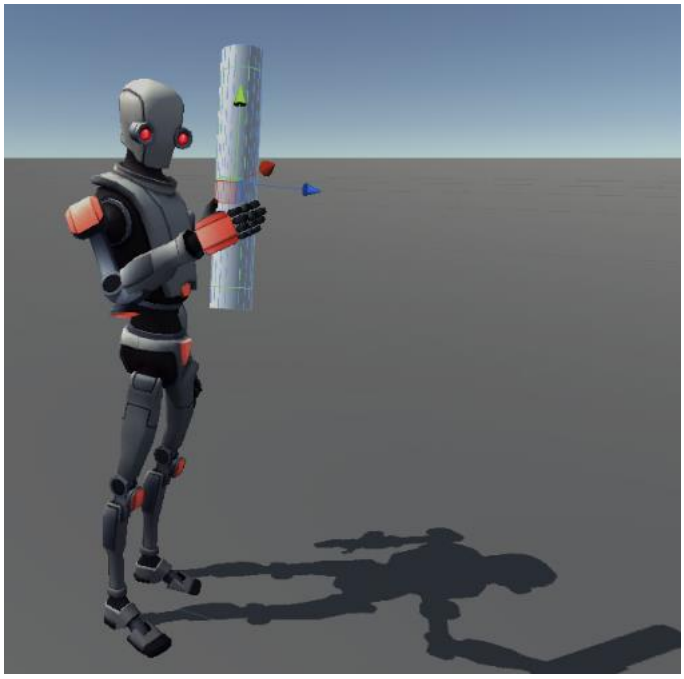

Figure 16: Mecanim IK by Unity [86]

3.6.4 Cinema IK

CinemaIK [88], available on the Unity Asset Store, enriches Unity's ecosystem by facilitating the animation of humanoid characters with an intuitive and accessible interface. This asset integrates closely with Unity's Mecanim system, leveraging its inverse kinematics (IK) capabilities tailored for humanoid avatars. Despite the simplicity and user-friendliness offered by CinemaIK's panel, which allows for precise control over characters' gazes and limbs through independent objects, it inherits the fundamental limitation of Mecanim's IK solutions: the restriction to humanoid characters that do not exceed four limbs with up to three joints each. Recognizing the constraints posed by this limitation, the development of an asset inspired by CinemaIK's streamlined interface was undertaken, aiming to transcend the skeletal restrictions by implementing custom solvers that accommodate a wider variety of character anatomies. This initiative reflects a deliberate effort to extend the flexibility and applicability of IK solutions within Unity, making it possible to animate characters beyond the traditional humanoid framework. The CinemaIK asset itself

comprises several key components: the main CinemaIK component, the IKData class for storing inverse kinematic solutions such as poses and weights, and the CinemaIKAnchor, which serves as an animator component anchor to relay IK information from the main component to the IKData script. Integration with Unity's Mecanim requires enabling the IK Pass on the Animator and adding the CinemaIK component to a new empty GameObject for optimal setup. The animator of the target character and the limbs' targets are configured within the CinemaIK interface, complete with sliders to adjust the influence on the kinematic chain.

To capture movements, the target GameObject is added to Unity's Timeline, creating a new Animation Track. Existing animations can be modified or new ones crafted within this framework, utilizing the CinemaIK component added to the Timeline to record character movements with keyframes. This workflow not only underscores CinemaIK's synergy with Mecanim and Unity's Timeline for creating complex animations and cinematic content but also highlights the broader ambition to overcome the inherent skeletal limitations through innovation and the development of new IK solutions.

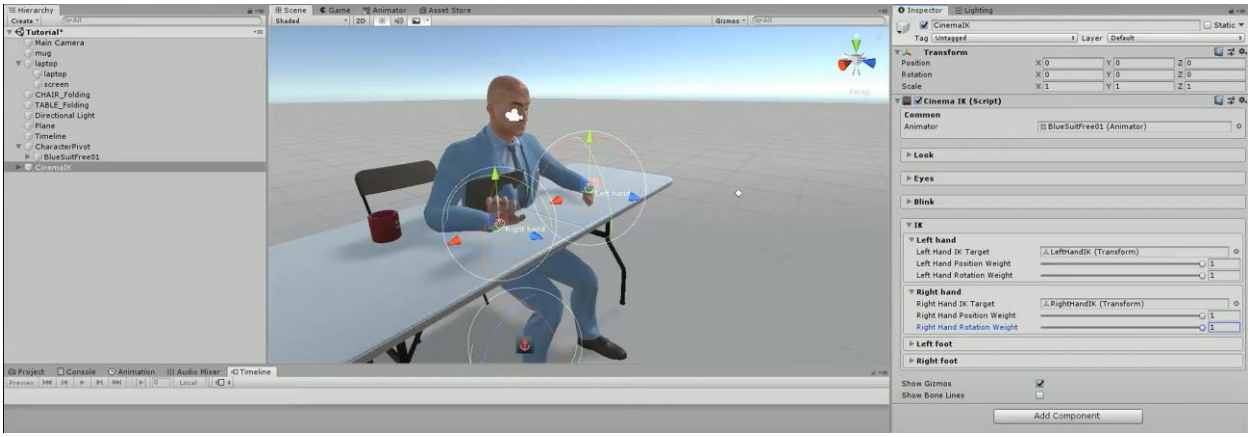

Figure 17: Cinema IK [88]

3.6.5 Easy IK

The EasyIK package [89] for Unity is an implementation of the FABRIK inverse kinematics solver tailored for developers working on VR and full body animation projects. It'script emphasizes a procedural approach to IK, allowing for dynamic adjustment of character limb positions towards a specified target, making it a potentially effective tool for real-time applications such as VR. The script is configured to handle a predefined number of joints and performs iterative calculations to align these joints with an IK target, utilizing a specified number of iterations and a tolerance threshold to determine the precision of the alignment.

Key features include the ability to define a pole target for controlling the orientation of intermediate joints, like elbows or knees, within a three-joint chain, which is crucial for achieving natural limb positioning. Additionally, it offers debugging options to visualize joint positions, rotations, and the pole target alignment through Unity's Gizmos, enhancing the development and troubleshooting process.
However, the script's requirement for manual setup on each joint and the effector script attachment on each bone might introduce complexity, making it less less straightforward for programmers unfamiliar with IK setups This could potentially limit its programmer-friendliness, particularly for complex rigs or those new to IK concepts. Despite this, the script's performance in VR environments, where real-time computation and responsiveness are critical, could be seen as a strong point due to FABRIK's efficiency in solving IK problems with lower computational overhead compared to more traditional angle-based solvers.

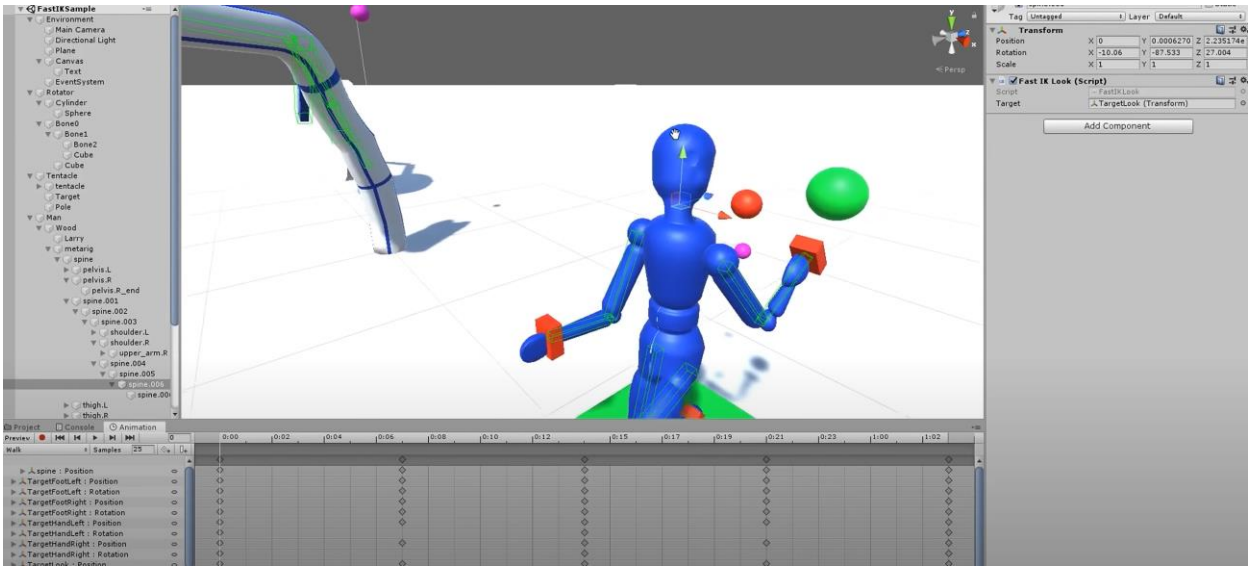

Figure 18: Easy IK [89]

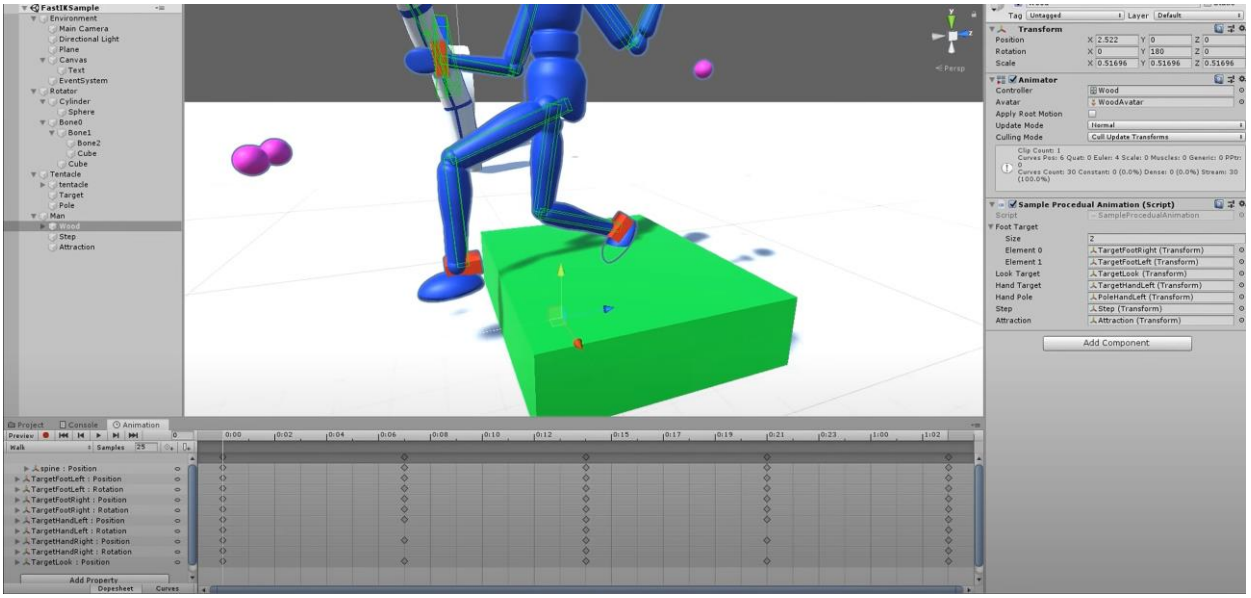

Figure 19: Fast IK Root for the IK seems complicated [89]

3.6.6 Fast IK

The provided package [90] represents an implementation of the FABRIK algorithm within a Unity environment, environment, bearing functional similarities with EasyIK package. It operates by dynamically adjusting a chain of joints to meet a specified target position, allowing for intuitive setup and control of IK systems, particularly in scenarios requiring precise manipulation of character limbs.

Analyzing the script reveals its structured approach to solving IK problems. It initializes by calculating bone lengths and setting up arrays to hold joint positions and rotations, thus preparing the system for the iterative solving process. One notable strength of the scriptit uses, is its adaptability to chains of varying lengths, as it accommodates any chain length specified by the user. This flexibility makes it applicable to

a wide range of use cases, from simple arm setups to more complex structures like tails or tentacles. The incorporation of a pole target provides additional control over joint orientations, helping to prevent unnatural bending and twisting that can occur in IK systems.

However, the script's effectiveness is contingent upon the manual setup of its parameters, including the chain length and target positions, which could pose challenges for users unfamiliar with IK concepts or those working with highly complex character rigs. Additionally, while the script includes a mechanism to handle scenarios where the target is unreachable by stretching the chain towards the target, this might result in unrealistic movements, highlighting a potential limitation in handling edge cases.

Another point of interest is the commented-out section related to a different solver, suggesting that the script was intended to support multiple IK solving techniques, though these capabilities are not fully implemented or integrated into the main functionality. This hints at potential extensibility but also indicates a current focus on FABRIK without leveraging the benefits other solvers might offer.

Figure 20: Fast IK joints and movement [90]

3.7 Comparative Discussion

The comparative analysis of various Inverse Kinematics (IK) packages presented here is grounded in assessments drawn from a collection of paper references, alongside a thorough examination of each package's source code, except the commercial ones. This analysis considers several critical metrics to gauge the suitability and functionality of each IK solution for virtual reality applications. A pivotal factor in this assessment isprogrammer-friendliness, defined here as the capacity to control the IK system and its corresponding effectors through a singular script interface, which streamlines the integration and manipulation process for developers. Another key aspect evaluated is whether these packages offer a dedicated VR IK editor, which can greatly facilitate the fine-tuning of avatars and objects within a VR setting. Additionally, the availability of the package at no cost is a vital mjaetric, significantly impacting accessibility for a wider range of users, from indie creators to educational purposes.
While some packages, like SAFullBodyIK, are lauded for their solver's VR compatibility and cost-free access, they may lack in areas such as ease of use and the availability of a VR editor, which can affect the overall user experience.

Conversely, no solutions incorporate these user-friendly features and tools, with a comprehensive VR editor and calibration tools, thta could offer a more intuitive and accessible approach for developers, albeit potentially at a cost.

In contrast, both Final IK and SAFullBodyIK exhibit limitations in providing an IK solver for the chain from head to shoulder, necessitating external inputs for ground truth shoulder locations to reduce complexity. These solvers employ parametric models to deduce elbow positioning from the hand position relative to the shoulder, yet neither method accounts for hand orientation, which could prevent unnatural wrist bends. The comparison of these IK solvers, using publicly available motion capture data covering a range of motions typical in VR games, reveals that specialized solvers can achieve lower errors in joint positioning than FABRIK, particularly for the shoulder and elbow joints. This suggests that while FABRIK's avoidance of joint limits contributes to its efficiency, incorporating detailed knowledge of typical joint movements could enhance accuracy. The analysis further indicates that methods relying on ground truth for shoulder positioning tend to perform better in datasets with diverse motions, highlighting the importance of comprehensive solutions that consider the entire kinematic chain for achieving high fidelity in character animation. The following Table 3: IK Solutions analysis on their Solver, Usage for VR, User Freindly, VR Editor, Calibration and if they are free.analyzes each IK solution discussed in the previous sections based on their technologies.

IK Package	Solver	Use for VR	User Friendly	VR editor	Calibration	Free
Animation Rigging Package	FABRIK	Yes	No	No	No	Yes
Final IK	FABRIK	Yes	Yes	No	No	No
SAFullBodyIK	FABRIK	Yes	No	No	No	Yes
Cinema IK	FABRIK	Yes	Yes	No	No	No
FastIK	FABRIK	Yes	No	No	No	Yes
FreeIK	FABRIK	Yes	No	No	No	Yes
Mecanim IK	FABRIK	Yes	No	No	No	Yes

Table 3: IK Solutions analysis on their Solver, Usage for VR, User Freindly, VR Editor, Calibration and if they are free.

Chapter 4 JARVRIKS (Just Another Robust VR IK Solution) IK

Maintaining immersion in VR simulations is paramount, necessitating swift and optimized computations to ensure seamless experiences. This imperative extends to full-body tracking and embodiment, which often entail complex inverse kinematics (IK) computations. To preserve immersion, these heavy-load IK calculations must be executed with minimal latency, ensuring that users' movements are accurately reflected in real-time within the virtual environment. Achieving this requires a delicate balance between computational efficiency and precision, where the speed of calculations is optimized without compromising the fidelity of tracking. By prioritizing fast and efficient IK solvers, VR systems can uphold immersion while empowering users with responsive and lifelike interactions in their virtual surroundings.

JARVRIKS (Just Another Robust VR IK Solution), aims to advance the state of the art with an Inverse Kinematics (IK) multi-threading solution for Unity VR simulations. For users to fully experience the advanced capabilities of full-body tracking within VR mode, it is imperative to utilize trackers compatible with SteamVR-OpenXR. This requirement stems from JARVRIKS' dynamic search for connected trackers, ensuring that every movement is accurately captured and rendered within the virtual environment. The use of C# Graphics Jobs multi-threading will ensure optimal performance in VR environments. JARVRIKS is designed to significantly enhance user accessibility and interaction with IK configurations, introducing a comprehensive yet intuitive panel in VR that allows for advanced customization, including features such as pull weight for each IK target, providing an extra level of detail for the strength of attracting an avatar's body part to its target position, like a magnet. A higher pull weight means the body part will closely follow the target, while a lower weight allows more freedom and natural movement deviation from the target.

Additionally, JARVRIKS incorporates a calibration method that utilizes a uniform scale adjustment of the avatar ensuring an accurate representation of the user's physical dimensions in the virtual space. To address common issues with avatar limb distortions, particularly at the wrists, JARVRIKS will integrate a 'wrist twister' mechanism, preventing unnatural wrist breaks and enhancing the realism of character movements. Furthermore, JARVRIKS will feature an in-VR IK editor, allowing users to seamlessly configure and calibrate their avatars directly within the VR environment. This editor will facilitate automatic target attachment to controllers or trackers for full-body tracking, streamlining the setup process for a wide range of VR applications. In that respect, the users will be able to customize their avatar in VR, through a custom-implemented user-friendly component editor. By combining these features, JARVRIKS aims to set a new extension for IK systems in VR, offering an unprecedented level of control to developers, and ease of use and enhanced presence and immersion for virtual user experiences.

As we dive into the implementation of JARVRIKS, three key developments are set to elevate the immersive experience: the *FullBodyIK* implementation for the avatar movement, the Animation Job integration describing the performance of the IK through mathematical computations, and the VR IK Editor for customization of the avatar directly within the VR environment.

The `*JarvriksFullBodyIK*` class is a sophisticated framework designed for creating immersive virtual reality (VR) experiences by providing realistic full-body tracking and inverse kinematics (IK) solutions. This system is constructed using a modular approach, incorporating various classes that interact to simulate accurate and responsive movements of a virtual avatar corresponding to the user's physical actions.

At the heart of this system is the `*JarvriksFullBodyIKJob*` class, which is responsible for the heavy lifting of the IK computations. It holds references to `*GeneralEffectorHandle*` and `*LimbJoints*` instances, representing different parts of the avatar's body, such as hands, feet, and their respective limb segments. This class processes animation data and updates avatar poses on a frame-by-frame basis, ensuring that the virtual representation mimics the user's movements with minimal latency. The computation of movements is further refined through parameters like stiffness and pull iterations, offering fine control over the rigidity and fluidity of motions.

`*EffectorData*` and `*AnimationInput*` classes serve as data structures that store detailed configurations for each effector and the mappings between user inputs and animation triggers, respectively. These configurations determine how different body parts should move and react to external inputs, making the system adaptable to various gameplay mechanics or interactive scenarios.

The `*JarvriksFullBodyIK*` class acts as the central hub, linking the IK logic with Unity's animation system. It interfaces with Unity's `Animator` and `*PlayableGraph*` to integrate the IK computations seamlessly into the game's existing animation pipeline. Through this class, developers can adjust IK settings, manage effector targets, and apply uniform calibration techniques to align the avatar's proportions with those of the user, enhancing the believability of the virtual embodiment.

To facilitate ease of use and customization, the system includes `JarvriksVREditor` and `*JarvriksFullBodyIKEditor*` classes. These editor extensions provide graphical interfaces for configuring IK settings directly within Unity's Editor, offering immediate visual feedback and simplifying the process of fine-tuning the avatar's movements. Additionally, the `*JarvriksIKSettingsManager*` class provides functionality for persisting and loading IK configurations, ensuring consistency across sessions or projects.

Supporting the interaction between these components are utility classes like `*DummyEffector*`, which acts as placeholders for effector targets in the VR environment, allowing for dynamic repositioning and adjustment of effectors based on real-time user movements.

In summary, the `JarvriksFullBodyIK` system exemplifies a well-architected approach to solving complex IK challenges in VR. *Through* a combination of modular design, extensible interfaces, and tight integration with Unity's animation system, it provides a robust foundation for developing VR experiences that require precise and naturalistic avatar movements. This framework not only enhances the immersive quality of VR applications but also offers developers the flexibility to tailor the system to meet specific interaction requirements or narrative goals.

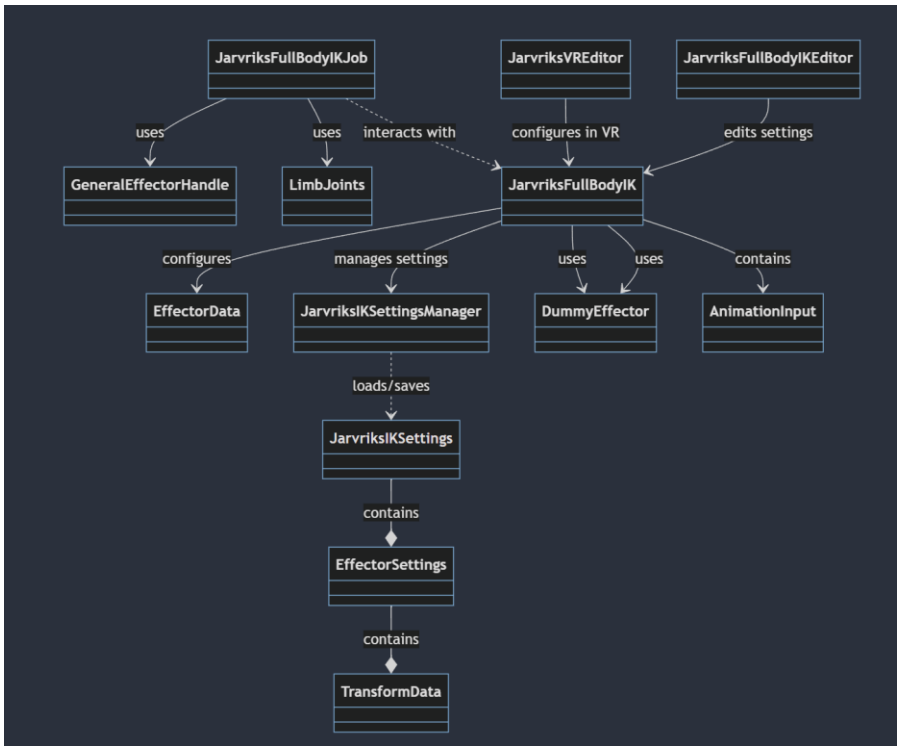

Figure 21: The Jarvriks System Diagram

4.1 The JarvriksFullBodyIK class

The *JarvriksFullBodyIK* class is a comprehensive solution crafted for the Unity engine, aimed at enhancing virtual reality experiences through detailed full-body tracking and inverse kinematics (IK) application. At its core, the script focuses on creating a bridge between the physical movements of users and their digital avatars, enabling a synchronization that is both fluid and precise. It utilizes a variety of effector data structures—specifically designed for different body parts like hands, feet, head, and the body—to fine-tune and control the avatar's movements in response to the user's actions.

Designed with flexibility in mind, this script allows for the adjustment of several parameters, including stiffness, pull iteration limits, and default weights for position, rotation, and pull. These settings are critical for tailoring the IK behavior to suit various animation requirements and achieving the desired levels of responsiveness and realism in avatar movements. The script is equipped with mechanisms to handle animations input, enabling it to process and respond to user interactions dynamically.

A notable feature of the *JarvriksFullBodyIK* script is its capability for uniform avatar calibration. This process adjusts the avatar's scale based on the user's physical dimensions, such as arm span and height, ensuring the virtual representation closely matches the user's real-world proportions. Such calibration is essential for immersive VR experiences, as it enhances the believability of the avatar's interactions within the virtual environment.

The script operates within Unity's Playable API framework, allowing for seamless integration with the engine's animation systems. It establishes a playable graph that manages the flow of animations and IK solutions, ensuring that movements are smoothly blended and accurately represented. Through careful manipulation of effector targets and weights, the script translates the intricate details of human motion into the virtual realm, providing a foundation for sophisticated interaction models in VR applications.

To facilitate the practical application and fine-tuning of IK settings, the script supports the instantiation of dummy objects as placeholders for real-world tracking devices or controllers. This feature is particularly useful in VR setups, where physical space and user movements must be accurately mirrored by the avatar. Additionally, it incorporates a system for loading and saving configuration settings, which aids in maintaining consistency across different sessions or projects.

The *JarvriksFullBodyIK* script embodies a multifaceted approach to bridging the gap between virtual and physical realms. By leveraging Unity's animation and XR frameworks, alongside a detailed IK system, it sets the stage for creating VR experiences that are not only engaging but also deeply connected to the natural movements and intentions of the user. Through this script, developers are equipped with the tools necessary to push the boundaries of VR immersion, offering users a seamless extension of their physical selves into digital landscapes.

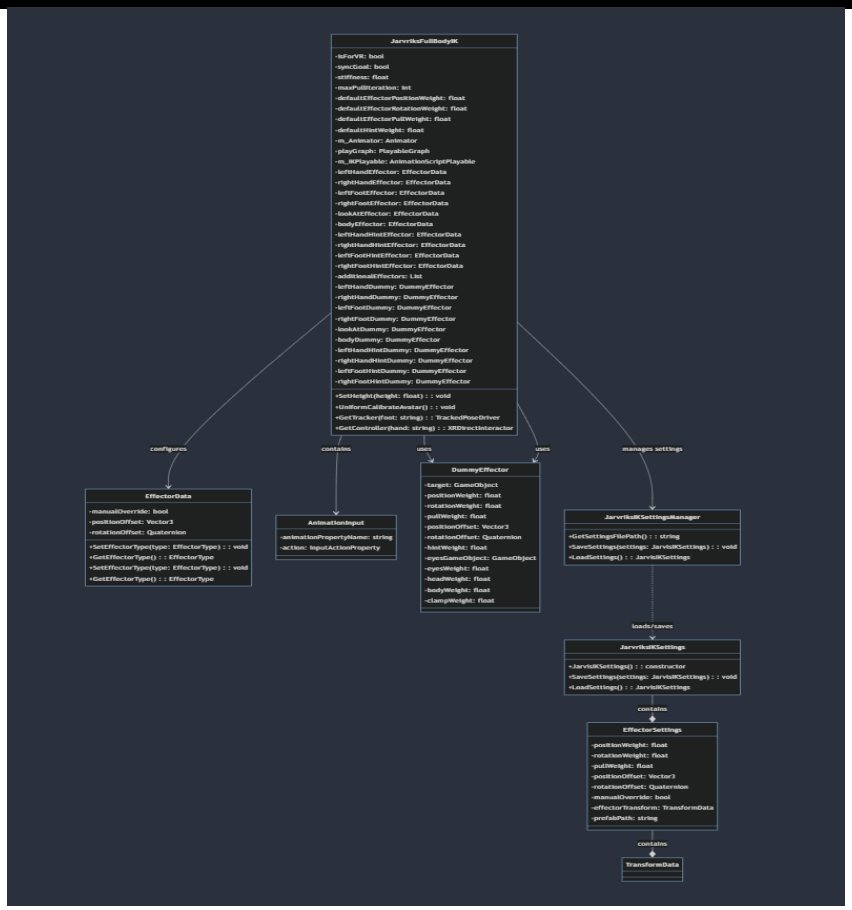

Figure 22: The JarvriksFullbodyIK class diagram containing the relations between the EffectorData, AnimationInput, DummyEffector, JarvriksIKSettingsManager, JarvriksIKSettings and EffectorSettings class

4.1.1 JarvriksFullBodyIK and its associated classes

The *JarvriksFullBodyIK class* and its associated classes play a crucial role in facilitating advanced Inverse Kinematics (IK) functionalities in Unity, allowing for dynamic and realistic character animations. Let's analyze the implementation of this class in detail.

The *JarvriksFullBodyIK* class is the core component orchestrating the IK system, linking Unity's animation system with the custom IK functionalities.

- *isForVR, syncGoal*: Flags to configure the system for VR use and whether to synchronize IK goals with the current animation pose.
- *Stiffness, maxPullIteration*: Parameters controlling the rigidity of the IK solution and the iterations for solving pull-based IK, affecting the smoothness and responsiveness of movements.
- *Default weights*: Default influence weights for newly added effectors, providing a baseline for their impact on the IK solution.
- *Animator, PlayableGraph, IKPlayable*: Unity components and structures for integrating the IK system with the Animation system, enabling the application of IK adjustments within the animation playback loop.
- *EffectorData* Instances: Specific instances of *EffectorData* for various body parts, enabling detailed control over their IK behaviors.
- Animation Inputs: List of *AnimationInput* instances for dynamic animation control based on player

inputs or game events.
- VR Components: References to VR components like controllers and trackers for integrating IK with VR inputs.
- Initialization and Update Methods: Functions for initializing effectors, updating their states based on the animation and inputs, and applying the IK solution each frame.

The *EffectorData c*lass stores the configuration and state of each effector within the IK system. It's designed to accommodate various effector types, with properties tailored to their specific roles.
- *effectorType*: Specifies the type of effector (Effector, Hint, LookAt, Body).
- *Target*: The GameObject which the effector manipulates or targets.
- *positionWeight*, *rotationWeight*, *pullWeight*: Influence weights for position, rotation, and pull adjustments, respectively. They determine the effector's impact on the IK solution.
- *hintWeight*: For Hint effectors, defines the influence of the hint in guiding the IK solution.
- *eyesGameObject*, *eyesWeight*, *headWeight*, *odyweight*, *clampWeight*: For LookAt effectors, these properties control the look-at behavior and its influence on different body parts.
- *manualOverride*, *positionOffset*, *rotationOffset*: Allow for manual adjustments to the effector's position and rotation, enabling fine-tuning of the IK solution or dynamic changes during runtime.

The *EffectorType* Enum defines the types of effectors used in the IK system:
- ***Effector***: Represents a standard IK effector for manipulating limbs (e.g., hands, feet).
- ***Hint***: Used for additional guidance in the IK solution, such as elbow or knee directions.
- ***LookAt***: Controls the direction in which the character's head or eyes should be looking.
- ***Body***: Pertains to the character's central body movements and positioning.

The *AnimationInput c*lass holds data for animating effectors through Unity's Input System, linking animation properties with input actions for dynamic control.
- animationPropertyName: the name of the animation property to be controlled.
- *InputActionProperty:* drives the animation property's value, allowing for real-time animation adjustments based on player inputs or game events.

The *InitializeGeneralEffector* function plays an important role in setting up the inverse kinematics (IK) system for a character in a VR environment. It dynamically assigns effectors to parts of the character's body (like hands and feet) and configures their behavior based on whether the system is operating in a VR context and whether specific VR controllers or trackers are connected.

The function distinguishes between VR and non-VR contexts, defaulting to placeholder targets (e.g., a hint prefab) when not in VR. This flexibility allows the IK system to function in a variety of scenarios, including development and testing environments where VR hardware might not be available. When in VR, the function searches for connected controllers and trackers using the *GetController* and *GetTracker* methods.

The *GetTracker f*unction searches and returns the VR tracker (e.g., Vive Tracker) associated with a specific foot, identified as either "Left" or "Right". It uses *UnityEngine.InputSystem.XR.TrackedPoseDriver* to interface with VR hardware, allowing for tracking of position and orientation. *FindObjectsOfType* is called to gather an array of all *TrackedPoseDriver* instances in the scene. The true parameter indicates that even inactive objects should be considered, ensuring a comprehensive search. The function iterates through each *TrackedPoseDriver* instance, checking the name for a match with the specified foot parameter ("Left" or "Right"). This approach relies on naming conventions to identify the correct tracker. If a match is found, the corresponding *TrackedPoseDriver* instance is returned, enabling its positional and rotational data to be used for the specified foot's IK target. This method also heavily depends on naming conventions ("Left", "Right") to identify the intended tracker. While simple, it might require consistent naming standards across different VR setups.

The *GetController method* finds and returns the XR controller (e.g., Oculus Touch or Vive Controller) corresponding to a specified hand, either "Left" or "Right". Utilizes *XRDirectInteractor*, part of Unity's XR Interaction Toolkit, which is designed for direct interaction in VR, such as grabbing or using objects.

Like *GetTracker*, it gathers all instances of *XRDirectInteractor* present in the scene, considering inactive objects as well. The function then iterates through the collected array, comparing the name property of each controller against the specified hand parameter. The match is based on exact names ("Left Controller" and "Right Controller"). Upon finding the matching controller, its instance is returned, allowing its tracking data to be utilized for hand IK targets. Like *GetTracker*, this method's effectiveness depends on consistent naming conventions. Inconsistent naming or custom controller names might require adjustments to the search logic. By using *XRDirectInteractor*, this function aligns with Unity's broader ecosystem for VR development, facilitating integration with other XR Toolkit features like interaction and physics.

This dynamic detection ensures that the IK system can adapt to the specific hardware setup of the user, accommodating a range of VR devices. For each effector (like hands and feet), the function checks if a target GameObject already exists. If not, it creates a new target based on the detected VR controllers or trackers, positioning it relative to the controller's or tracker's location and orientation. This approach ensures that the IK targets accurately represent the physical location and orientation of the user's limbs in the virtual environment.

The function adds a *DummyEffector* component to each target GameObject. This component stores IK-related properties like position and rotation weights, allowing for fine-grained control over how each body part follows its IK target. It uses the *Animator.BindSceneTransform* and *Animator.BindSceneProperty* methods to bind each effector's properties to the animation system. This binding enables the IK system to manipulate the character's animations based on the effector's configuration, ensuring that movements are smooth and responsive to the user's actions in VR.

4.1.2 Saving and Loading functionality

Figure 23: The Loading/Saving functionality into class diagram containing the relations between the JarvriksSettingsManager, JarvriksIKSettings and EffectorSettings classes.

The static utility class *JarvriksIKSettingsManager* provides mechanisms for persisting and retrieving the IK configuration (JarvriksIKSettings) to and from a file. This class works in tandem with the *JarvriksIKSettings* script to ensure that the user's IK settings are maintained across VR sessions, contributing significantly to a seamless user experience. Here's a detailed analysis of how these two scripts interplay and the functionality provided by *JarvriksIKSettingsManager*:

The *GetSettingsFilePath method* constructs and returns the file path where the IK settings should be saved or loaded from. It uses Unity's *Application.persistentDataPath* to ensure that the settings are stored in a directory that persists across VR sessions and is accessible without needing special permissions. The method combines the persistent data path with the filename "JarvriksIKSettings.json" to form a complete file path. This standardized naming and location make it straightforward to locate the settings file for both saving and loading operations.

The *SaveSettings method* serializes the *JarvriksIKSettings* object into JSON format and writes it to a file, effectively saving the current IK settings. It utilizes *JsonUtility.ToJson* to serialize the settings object, including formatting for readability (true parameter). The resulting JSON string is then written to the file at the path determined by *GetSettingsFilePath*. This approach ensures that any modifications to the IK settings are persisted beyond the current VR session, allowing them to be reloaded in future sessions.

The *LoadSettings method* loads the IK settings from a file, if it exists, deserializing the JSON content back into a *JarvriksIKSettings* object. It checks if the settings file exists at the path provided by *GetSettingsFilePath*. If the file is found, it reads the JSON content and uses J*sonUtility.FromJson* to deserialize it into a *JarvriksIKSettings* object. If the file does not exist, it returns a new instance of *JarvriksIKSettings* with default values. This mechanism provides resilience by ensuring that the application can handle situations where the settings file might be missing or deleted.

The *JarvriksIKSettings* class is a comprehensive structure designed to serialize and store the settings related to the Inverse Kinematics (IK) configuration of an avatar within a virtual reality (VR) or animation context. This script facilitates the saving and loading of IK settings, ensuring a persistent user experience across VR sessions by retaining the customized positions, rotations, weights, and other properties of each effector involved in the IK system. Following we delve into the components and functionalities of this script:

The *TransformData* class holds the positional, rotational, and scaling information of an object in 3D space. This class is essential for capturing the state of an effector's target transform, enabling its restoration upon loading settings.

The fields of *JarvriksIKSettings c*lass are global settings affecting the overall behavior of the IK solver as well as individual settings for each effector, such as stiffness, *maxPullIteration*, *defaultEffectorPositionWeight*, *defaultEffectorRotationWeight*, *defaultEffectorPullWeight*, *defaultHintWeight.* These fields represent global settings that influence the IK solving process, such as the rigidity of the effectors' movements, the number of iterations for pull adjustments, and default weightings for effector positions, rotations, and pulls.

The other effectors such as *leftHandEffector*, *rightHandEffector*, *leftFootEffector*, *rightFootEffector*, *lookAtEffector*, *bodyEffector* are fields as an instance of the *EffectorSettings* class, tailored to store settings specific to an individual effector. This approach allows for granular control over each part of the IK system. The constructor initializes each effector setting with default values, ensuring a consistent starting point for customization.

The *EffectorSettings* class encapsulates the settings for an individual effector, including its weighting in the IK calculations and any manual overrides or transform adjustments made by the user. Its fields are *positionWeight*, *rotationWeight*, *pullWeight* and they control the influence of the effector's position, rotation, and pull (how strongly it affects the avatar's pose) within the IK calculations.

The *positionOffset*, *rotationOffset* fields allow manual adjustments to the effector's target position and rotation, useful for fine-tuning the avatar's pose. Also the *manualOverride*, a boolean indicating whether the user has manually overridden the effector's automatic positioning, ensures that custom adjustments are respected. The *effectorTransform* captures the current transform state of the effector's target.
The *prefabPath* optionally stores the path to a prefab for the effector's target, facilitating dynamic instantiation or referencing within the scene.

These functions are directly called from JarvriksFullBodyIKScript to store all informations about every effector, On the OnApplicationQuit function is ensured that IK settings are saved when the application is closing. It creates a new instance of JarvriksIKSettings and calls SaveSettings to capture the current IK configuration. Utilizes JarvriksIKSettingsManager.SaveSettings to persist the captured settings, ensuring they are available for loading in the next session.

By marking the JarvriksIKSettings and its nested classes as [Serializable], Unity can easily serialize the entire structure, allowing it to be saved to and loaded from persistent storage (e.g., a file or PlayerPrefs). This feature is crucial for maintaining user-customized settings across VR sessions, enhancing the user experience by remembering their preferences and adjustments made within the VR environment or

animation setup.

4.1.3 Calibration functionality

The *UniformCalibrateAvatar* method within the JarvriksFullBodyIK script is a key feature for ensuring a realistic and immersive VR experience by aligning the virtual avatar's proportions with those of the player. This calibration process is particularly important in VR environments, where a mismatch between the player's physical and the avatar's virtual movements can break immersion or cause discomfort.

Its functionality begins by measuring two critical dimensions – the user's arm span and height. The arm span is determined by calculating the distance between the positions of the left and right VR controllers, which the user holds in each hand. Similarly, the user's height is assessed based on the vertical position of the VR headset, which tracks the user's head position. With these measurements, the script calculates scaling factors to adjust the avatar's size. The scaling factor for the arm span is the ratio of the user's arm span to the avatar's default arm span (*defaultAvatarArmSpan*). The scaling factor for height follows a similar principle, comparing the user's height to the avatar's default height (*defaultHeight*). The script then determines a uniform scaling factor by averaging or combining the scaling factors for arm span and height.

This unified approach ensures that the avatar's proportions are adjusted uniformly, maintaining the avatar's anatomical correctness while matching the user's physical dimensions. Finally, the uniform scaling factor is applied to the avatar's root transform (*avatarRoot*), effectively resizing the avatar in all dimensions. This scaling ensures that the avatar's movements, when controlled by the user through the VR controllers and headset, closely mirror the user's real-world actions, enhancing the sense of presence within the VR environment.

Proper calibration ensures that the avatar's movements in the virtual environment correspond precisely to the user's real-world movements, deepening the sense of immersion. Misalignments between a player's physical actions and the avatar's responses can lead to discomfort or VR sickness. Calibration minimizes these discrepancies, making the VR experience more comfortable and enjoyable over extended periods. This calibration function allows the VR system to adapt to users of various sizes and builds, making the VR application more accessible and inclusive.

4.2 JarvriksFullBodyIKJob: C# Graphic Job

The Unity Graphics Job System is a powerful tool within Unity aimed at improving the performance of games and applications. By utilizing multi-threading, specifically through the Unity Graphics Job System, developers can significantly boost optimization within Unity. This system enables graphics-related tasks to run in parallel with the main game loop, effectively utilizing CPU cores to their fullest potential. This results in smoother frame rates and more complex graphics operations without slowing down the game. This system is particularly useful for projects that require detailed and high-quality animations, such as those involving full-body inverse kinematics (IK) solutions.

The *JarvriksFullBodyIKJob* struct is a sophisticated representation of an animation job in Unity that utilizes inverse kinematics (IK) for animating characters, specifically focusing on full-body IK solutions. At its core, this job struct involves managing and manipulating multiple *GeneralEffectorHandle* instances, each corresponding to different parts of the avatar's body such as hands, feet, and the torso, alongside specialized effectors for hints and gaze control. These handles are important for defining how each body part should move and rotate according to the animation's requirements, with detailed control over aspects

like weight and offset for position and rotation, thereby allowing for nuanced animation behaviors.

Each *GeneralEffectorHandle* contains scene handles for the effector and its properties, enabling real-time adjustment of the character's pose based on the animation data stream. The job struct also defines *IKLimbHandle* for each limb, specifying the limb's structure from top (proximal) to end (distal) joints and including a maximum extension property that dictates how far the limb can stretch and this helps for creating realistic and physically plausible movements within the character's constraints.

The animation job functions by first establishing the configuration of each effector handle and limb handle, setting up the initial conditions for the animation. Through the *ProcessAnimation* method, it then dynamically updates these configurations based on the current state of the animation stream. This includes applying transformations to the avatar's joints and limbs to achieve desired poses and movements, informed by the specified effector handles and limb handles. The process involves calculating the necessary adjustments to each joint's position and rotation to align with the animation's target states, employing techniques such as pull solving to simulate the effects of physical forces on the avatar's body.

The core of the IK animation job's importance lies in its 'solve' function, which mathematically determines the positions and orientations of each joint to achieve a desired end effector location. This calculation involves considering the constraints and degrees of freedom of each joint, ensuring that the final posture is physically plausible. The solve function iteratively adjusts the avatar's joints, starting from the end effectors and working back toward the root of the skeleton. This backtracking ensures that movements are goal-oriented, directly linking the character's interactive behavior with environmental factors and user inputs.

The JarvriksFullBodyIKJob represents a use case of Unity's animation system, specifically utilizing the IAnimationJob interface for implementing custom Inverse Kinematics (IK) in animations. This job handles various IK setups, including single effectors (like hands or feet), hints (used to guide IK solving, like elbow or knee directions), and look-at targets. The job is designed to manipulate an animated character's pose based on dynamic IK targets, blending them seamlessly with the existing animation data. Let's break down the key components and the underlying mathematics.

The core of the IK solving process involves adjusting the character's pose based on target positions and rotations for each effector, while respecting constraints like limb lengths and joint limits.

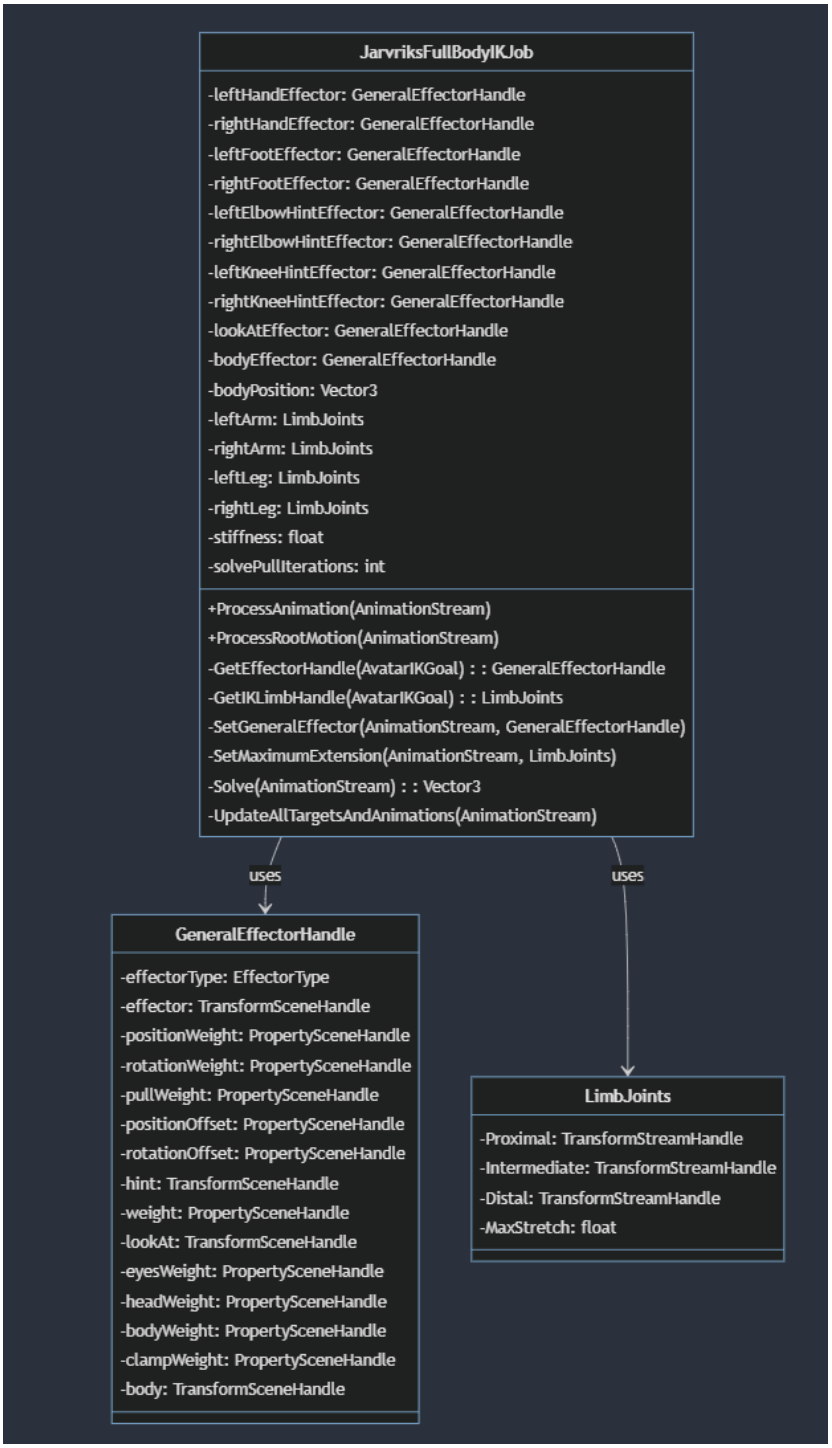

Figure 24: The JarvriksFullBodyIKJob class diagram which uses both GeneralEffectorHandle class and the LimbJoints class with all fields, methods used.

4.2.1 IK Solve Analysis

The Solve method first initializes the preparation for the IK solving process by calculating limb positions relative to the body's position and setting up goal positions and weights for each limb. For each limb, the function calculates its current position relative to the body's position. This is done using vector subtraction:

$$RelativePos = jointTopPos - updatedBodyPos.$$

AnimationHumanStream provides access to the humanoid animation data in a format that can be efficiently processed. When Solver calls methods like *humanStream.GetGoalPosition((AvatarIKGoal)goalIter)*, it fetches the current target positions for the IK goals (like hands or feet) as determined by the animation system. This information is crucial for setting up the initial conditions for the IK solution.

The function *GetGoalPosition* also establishes a conceptual "goal" or target for each limb, including its desired position and influence weights (position, rotation, and pull). These are foundational for the IK solving process as they dictate the direction and magnitude of adjustments needed for each limb to align with the animation targets.

The functions *GetEffectorHandle* and *GetIKLimbHandle* serve as accessors to fetch the appropriate effector or limb handles based on the IK goal (e.g., left hand, right foot). This design encapsulates the logic for mapping high-level IK goals to the specific data structures that hold their current state, simplifying the main IK solving logic. The other functions *GetGoalWeightPosition* and *GetFloat* are for accessing specific properties of the effectors, such as their influence weights or custom properties like pull weight. *GetFloat* is particularly generic, enabling the retrieval of any float property associated with an effector, enhancing the flexibility and reusability of the IK system.

The chosen data structure for the IK goals is native array primarily for performance reasons in the context of Unity's job system and Burst compiler. *NativeArray* offers a structure that is optimized for high-performance parallel jobs, allowing efficient memory layout and access patterns that are conducive to SIMD (Single Instruction, Multiple Data) operations and multithreading, crucial for real-time animation tasks. When solving IK in a real-time application like a game, performance is paramount. *NativeArray* minimizes garbage collection pressure and provides a low-level, array-like data structure that the Burst compiler can further optimize, leading to faster execution times for the IK calculations, based on Unity manual.

The Solve function is the core IK solving function iteratively adjusts the character's body and limbs to approach the target positions. Through a loop (controlled by *solvePullIterations* which is a value from 5 to 50), the function gradually moves the limbs towards their targets. Each iteration refines the body's position based on the limbs' pulls towards their targets. Conceptually, forces are applied to the limbs to move them towards their goals. This is calculated as a directional vector from the limb's current position towards the target, adjusted by the limb's stiffness and pull weight.

The force applied to move a limb towards its target can be modeled as:

$$appliedForce = Mathf.Max((currStretch - stretchRest) * limbData[goalIndex].Rigidity, 0.0f)$$

Where *currStretch* is the current distance from the limb to the target. *stretchRest* is the natural length of the limb (maximum extension without stretching).

The change in position (*updatedBodyPos*) for the body in each iteration is influenced by the forces from all limbs. Here, *appliedForce* is normalized to ensure its directionally correct but maintains a consistent magnitude.

- Let Δ_p be the cumulative position change vector.
- Let d_i be the normalized direction vector from limb i to its goal.
- Let F_i be the applied force magnitude for limb i.
- Let w_i be the target weight for limb i.
- Let s_i be the pull strength for limb i.

The equation for the position change, due to the influence of all limbs aiming towards their respective goals is given by:

$$\Delta p = \sum_{i=1}^{N} d_i \cdot F_i \cdot w_i \cdot s_i$$

$\sum_{i=1}^{N}$, denoting the summation over all N limbs or effectors involved in the IK solving process, is applied the calculated magnitude of change along the direction vector. This equation succinctly captures the essence of calculating the overall position change based on the contributions from each limb involved in the IK system. Each limb's contribution is determined by its direction towards the goal, the force applied to move it closer to the target, and the respective weights that modulate the influence of the goal position and pull strength.

```
private Vector3 Solve(AnimationStream stream)
{
    AnimationHumanStream animHumanStream = stream.AsHuman();

    Vector3 initialBodyPos = animHumanStream.bodyPosition;
    Vector3 updatedBodyPos = initialBodyPos;

    NativeArray<LimbIKData> limbData = new NativeArray<LimbIKData>(4, Allocator.Temp);

    for (int iteration = 0; iteration < solvePullIterations; iteration++)
    {
        if (iteration < limbData.Length)
        {
            AvatarIKGoal ikTarget = (AvatarIKGoal)iteration;
            GeneralEffectorHandle effectorData = GetEffectorHandle(ikTarget);
            LimbJoints limbDataHandle = GetIKLimbHandle(ikTarget);
            Vector3 jointTopPos = limbDataHandle.Proximal.GetPosition(stream);

            limbData[iteration] = new LimbIKData
            {
                Rigidity = this.stiffness,
                PullStrength = effectorData.pullWeight.GetFloat(stream),
                TargetPos = animHumanStream.GetGoalPosition(ikTarget),
                TargetWeight = animHumanStream.GetGoalWeightPosition(ikTarget),
                RelativePos = jointTopPos - updatedBodyPos,
                MaxStretch = limbDataHandle.MaxStretch,
            };
        }
        Vector3 positionChange = Vector3.zero;
        for (int goalIndex = 0; goalIndex < limbData.Length; goalIndex++)
        {
            Vector3 limbTop = updatedBodyPos + limbData[goalIndex].RelativePos;
            Vector3 directionToGoal = limbData[goalIndex].TargetPos - limbTop;
            float restLength = limbData[goalIndex].MaxStretch;
            float currentLength = directionToGoal.magnitude;

            directionToGoal.Normalize();

            var appliedForce = Mathf.Max((currentLength - restLength) * limbData[goalIndex].Rigidity, 0.0f);

            positionChange = positionChange+ directionToGoal * appliedForce * limbData[goalIndex].TargetWeight * limbData[goalIndex].PullStrength;
        }

        positionChange = positionChange /(solvePullIterations - iteration);
        updatedBodyPos = updatedBodyPos + positionChange;
    }

    limbData.Dispose();

    return updatedBodyPos - initialBodyPos;
}
```

Figure 25: The Solve function used to solve the IK

4.2.2 JarvriksFullBodyIKJob methods

This UpdateAllTargetsAndAnimations function updates the effector targets based on current animation states or external inputs and prepares the IK system for solving. In the begginig, it calculates and sets the maximum extension (length) of each limb (arms and legs). This length is crucial for determining the reach of each limb during the IK solving process, ensuring movements stay within realistic human anatomical limits. After that, the SetGeneralEffector method updates the properties of each IK goal (hands, feet) and hint (elbows, knees) based on their current states. This includes their positions, rotations, and weights, which dictate how strongly an IK goal should try to reach its target position or orientation. It ensures that the animation system's current pose is considered in the IK calculations, allowing for dynamic adjustments during gameplay or animation playback. Finally, Effector types (Effector, Hint, LookAt, Body) differentiate how different parts of the IK system are treated. For instance, effectors directly influence limb positions, hints provide auxiliary positioning hints (e.g., for elbows and knees to avoid unnatural bending), and LookAt controls the direction the character's head and eyes should face.

While the *ProcessRootMotion* method is empty, in more complex IK systems, it could be used to handle adjustments to the character's root position based on the IK calculations. Root motion is often used to create more realistic movements by allowing the animation to drive the character's movement and position in the game world. In our case, it wasn't necessary to manipulate this function.

The *ProcessAnimation* function orchestrates the IK solving process within the animation loop, ensuring the character's pose updates based on the IK calculations. The previous function analyzed, *UpdateAllTargetsAndAnimations*() is called first to ensure all IK targets and hints are up-to-date with the latest animation state or external inputs. In this function the Solver function is called and performs the core of the IK solving process. It iteratively adjusts the character's body position to reduce the distance between current and target positions of each effector, based on the calculated forces (described in previous explanations).After solving, the adjusted body position (*bodyPosition* += *bodyPositionDelta*) is applied back to the animation human stream, and *humanStream.SolveIK*() is called to apply the IK adjustments to the character. This step is crucial for integrating the IK calculations with Unity's animation system, ensuring the character's pose visually reflects the adjustments made by the IK solver.

4.3 Editors for IK customization

The *JarvriksFullBodyIKEditor* script is a Unity editor extension that simplifies the process of configuring full-body inverse kinematics (IK) for humanoid avatars. This tool is important for developers aiming to create character movements within Unity, catering to both traditional gaming and immersive VR experiences. By providing a graphical interface for adjusting IK parameters such as weights and offsets for various effectors (like hands, feet, and gaze direction), it helps with the development workflow, enabling control over character animations. Additionally, its visualization features offer real-time feedback on the skeletal structure and IK adjustments, enhancing in setting up IK systems.

On the other hand, the *JarvriksVREditor* script focuses on optimizing the VR experience by facilitating in-game, real-time adjustments of IK settings. It brings the power of IK configuration directly into the VR environment, allowing players and developers to fine-tune avatar movements on the fly. This direct interaction within VR not only ensures that avatars mirror the player's movements but also allows for immediate feedback and iterative adjustments to improve immersion and performance. The inclusion of a guided calibration process further exemplifies the script's role in personalizing the VR experience, making it adaptable to different player physiques and preferences.

Together, the *JarvriksFullBodyIKEditor* and *JarvriksVREditor* embody the JARVRIKS system's commitment to providing comprehensive and user-friendly solutions for IK configuration. They democratize access to animation techniques, enabling creators of all skill levels to implement character interactions without delving into the underlying complexity of inverse kinematics.

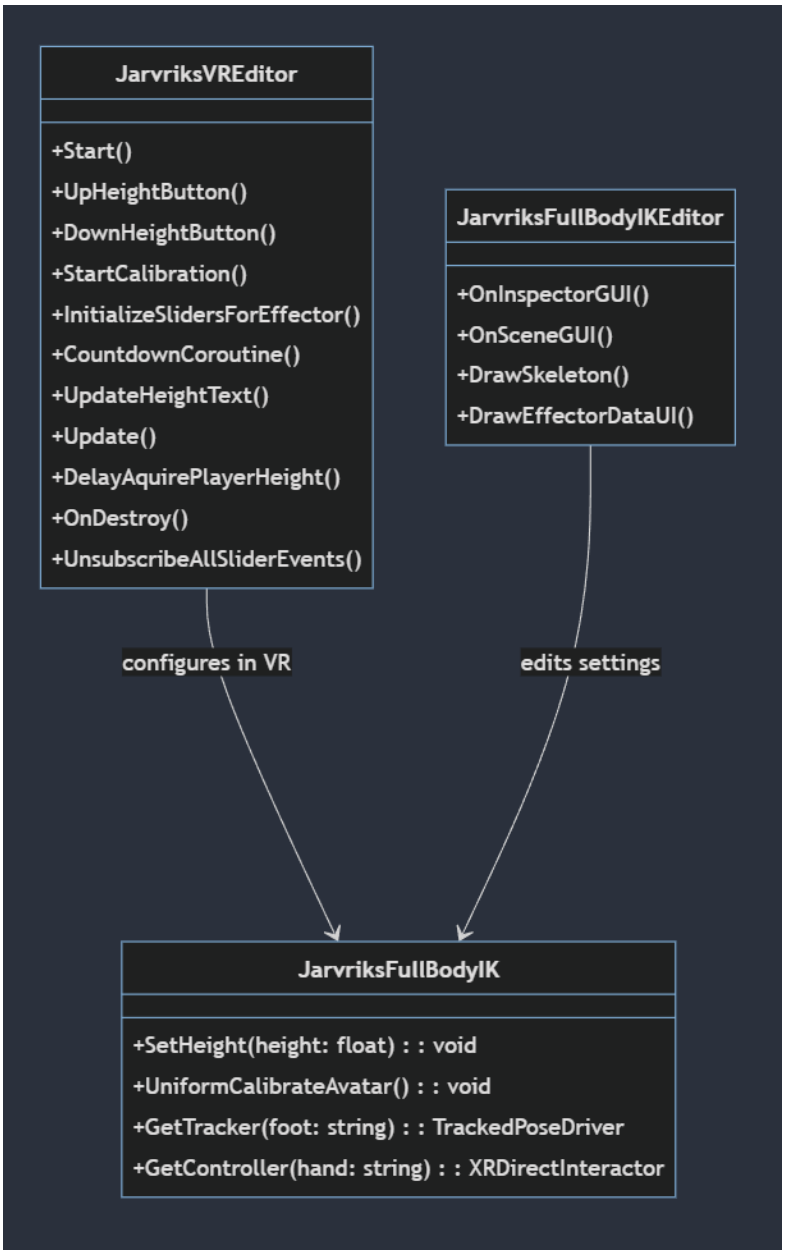

Figure 26: The JarvriksVREditor and JarvriksFullBodyIKEditor class diagram with their relation to the JarvriksFullBodyIK class

4.3.1 IK editor: The Class JarvriksFullBodyIKEditor

The *JarvriksFullBodyIKEditor class* first method is the *OnInspectorGUI* method, in which the script facilitates real-time updates to the IK configuration, reflecting changes immediately within the editor. This method also enables toggling specific settings for VR applications, demonstrating the script's adaptability to various project needs. The addition of drawing custom UI elements for each effector consolidates all necessary controls in one place, streamlining the IK setup process.

OnSceneGUI plays a pivotal role in visualizing the avatar's skeletal structure, employing the Unity Handles class to draw lines between joints. This visual aid is invaluable for developers to understand the spatial relationships within the IK setup, ensuring that each effector is correctly positioned and oriented. The method meticulously iterates through the humanoid bones, excluding the root to focus on the skeletal structure that participates in IK processes. Special consideration is given to the head bone, ensuring it's correctly linked to the neck, further aiding in the setup's accuracy.

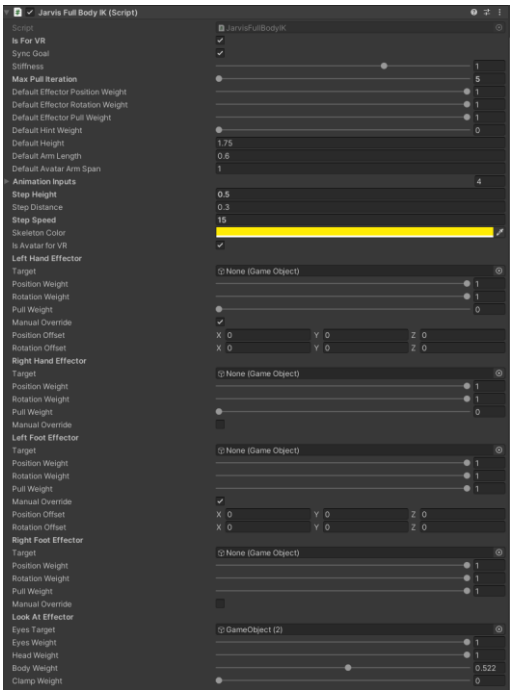

Figure 27: The JarvriksFullBodyIK custom editor

The *DrawSkeleton* function is ingeniously designed to visualize the skeletal structure of the character directly within the Unity scene view. By iterating through the *HumanBodyBones* enumeration, it identifies each bone's position and its parental connection, drawing lines between them to represent the skeletal links visually. This visualization allows for an immediate understanding of the skeletal structure's layout, facilitating more precise placement and adjustment of IK effectors. Special attention is given to ensuring the head bone, a critical part of the character's expressiveness, is accurately represented in relation to the neck, enhancing the realism of the avatar's movements.

The *DrawEffectorDataUI* function exemplifies the script's commitment to offering an accessible and detailed interface for configuring each effector's parameters. Through a series of UI elements, developers can assign targets to effectors, adjust their influence on the character's posture through weight sliders (position, rotation, and pull), and specify offsets for fine-tuning. This level of control is pivotal for achieving natural and responsive character animations, allowing for nuanced adjustments that reflect the complexities of real-world movements. The inclusion of manual override options further empowers developers, offering the flexibility to apply specific positional and rotational adjustments to effectors, catering to the unique requirements of each project.

4.3.2 VR IK Editor: The class JarvriksVREditor

The *JarvriksVREditor* script contains *ikSystem*, a reference to the *JarvriksFullBodyIK* component, which controls the IK operations for the VR character. This connection allows *JarvriksVREditor* to directly manipulate IK settings and effectors. The other reference is the origin, an instance of *XROrigin*, which is crucial for determining the player's position and orientation within the VR space, aiding in accurate calibration and alignment of the VR avatar to the player's physical movements.

For the UI elements and Interactivity, it contains Sliders Dictionaries, separate dictionaries for left hand, right hand, left foot, right foot, look-at, and general sliders. These dictionaries map string identifiers to Slider components, enabling dynamic adjustment of IK parameters such as weight and offset for each effector. The buttons *upButton*, *downButton*, and *calibrateButton*, facilitate user interaction for height adjustment and calibration initiation within the VR environment. It also has a Text component displaying the current height of the player, providing visual feedback for height adjustments. Finally, the countdown Mechanism comprises a GameObject for the countdown UI and a TMP_Text for displaying countdown numbers. This mechanism guides users through a timed calibration process.

Upon startup, the script initializes slider values based on current IK settings and subscribes to their value-changed events for real-time IK adjustment. It also prepares the countdown UI and height adjustment interface. The *InitializeSlidersForEffector* function dynamically binds sliders to specific IK effector parameters, allowing for granular control over the IK system's behavior. This setup is essential for tailoring the VR character's movements to the player's preferences or physical characteristics. Through *UpHeightButton* and *DownHeightButton* functions, users can fine-tune their avatar's height, ensuring a better match with their real-world stature. This adjustment is crucial for immersive and accurate VR experiences.

The Update method monitors for the start of the calibration countdown, launching a coroutine to manage the countdown and subsequent calibration. Meanwhile, *DelayAquirePlayerHeight* attempts to acquire an accurate player height after a brief delay post-startup, ensuring that initial settings are as accurate as possible.

Upon destruction of the *JarvriksVREditor* object, *OnDestroy* ensures all event listeners are properly unsubscribed from sliders to prevent memory leaks or unintended behavior.

For the calibration process the calibration button when it is triggered the *StartCalibration* function is called, and a countdown begins, culminating in the application of new calibration settings to the IK system.

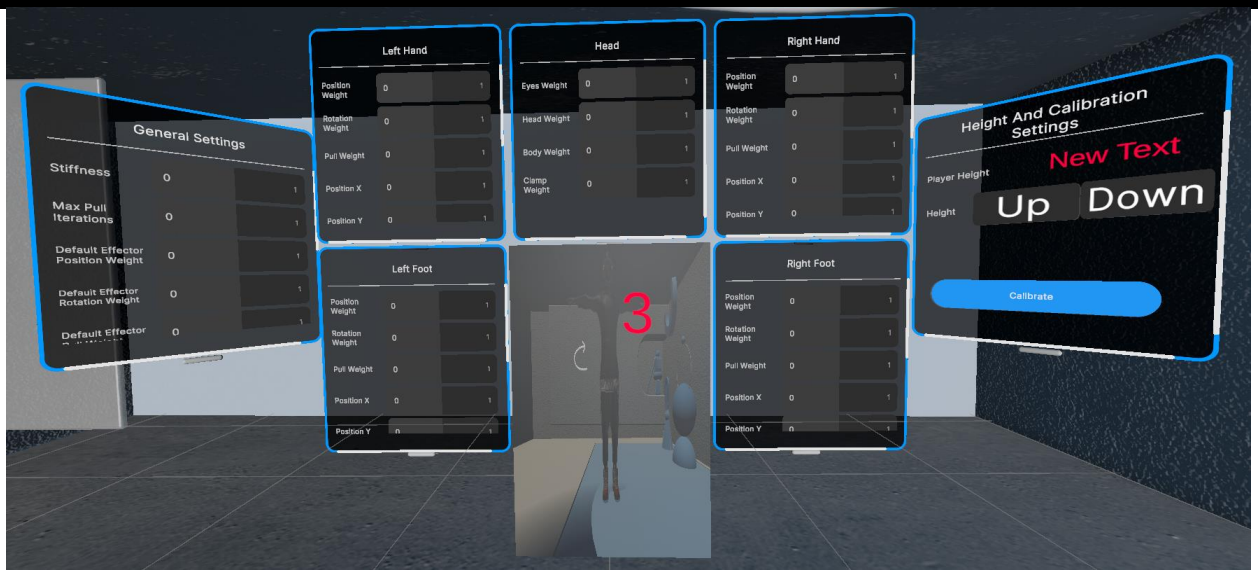

Figure 28: The VR IK Editor for modifying the effector properties on runtime and perform calibration method.

Chapter 5 Use Case Validation

In this chapter we aim to validate the proposed IK approach by developing and testing three use case game applications. Two games aim to analyze hand movements and their interaction with the virtual environment, and the third is for leg movements.

5.1 Use Case 1: SuperHot

SuperHot is an innovative first-person shooter game where time only moves when the player moves. This unique mechanics creates a more strategic and tactical approach to traditional single-shooter games. In SuperHot-style gameplay, enemies and bullets move only when the player moves. This enables the player to plan their moves and actions in a more strategic way. The player needs quick reactions and planning, as each move must be targeted and the strategy must be carefully planned. The game was originally created as a prototype in a game jam in 2013 and later developed into a full-game game. The game takes advantage of Unity Engine technology to visualize graphics and physics.

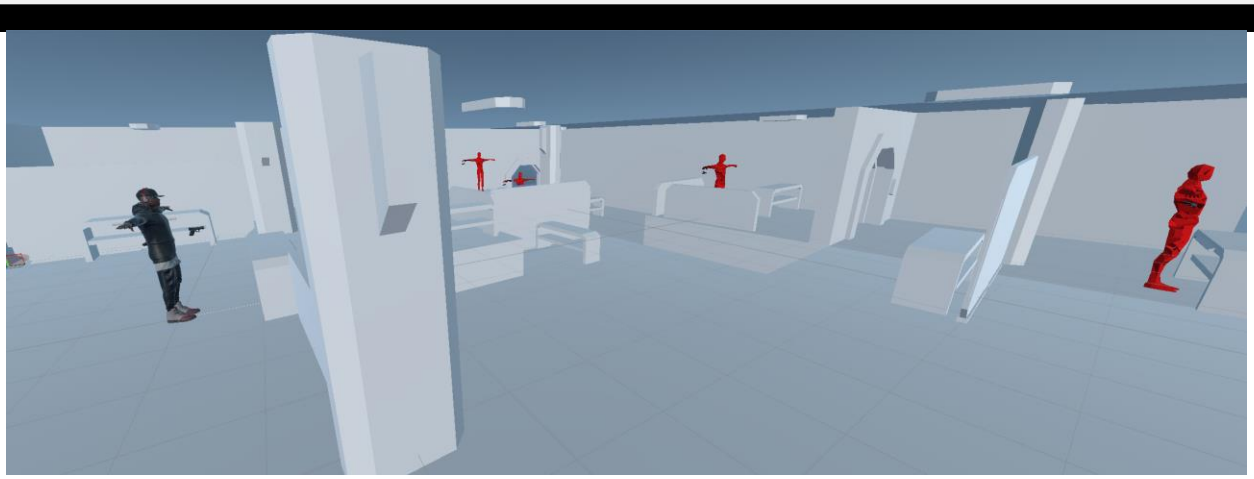

Figure 29: Super Hot Room containing our player, the enemies and the scene created

5.1.1 Scene Creation

The first step was to create the scene. Separate environment with fog effects to make it difficult for the user to deliberately detect enemies later. For this purpose, a script that creates fog is executed when the game starts.At the beginning of the game's mode (in the Start method), the script initializes the fog settings: a) Sets fog to be active (*RenderSettings.fog = true*) and specifies the start and end of fog (*fogStartDistance*, *fogEndDistance*) in the game environment. In that respect, fogging begins at 10 units from the camera and ends at 30 units. *SetFogRoutine* is a coroutine that manages the progressive change of fog and its disabling, by gradually increasing the *fogEndDistance* value until it reaches 45 units, creating a feeling of reduced fog.At the end of the game, the start and end values of fog are set to 0 and the fog is disabled (*RenderSettings.fog = false*), thus creating the impression that fog has completely disappeared from the world.

5.1.2 Weapons and interaction

The next step was to create the weapon and the right interaction. It is quite a complex system, because weapons can be possessed by both the user and the enemies.
Initially, an FBX weapon was imported from a free package on the Unity Asset Store, making it interactive through the XR Grabbable component. This enables the user to grasp and manipulate the object either directly or by using raycasting. To ensure the weapon is held in a realistic manner, the script was enhanced with methods and empty objects serving as offsets, guiding the user on the correct position and orientation for gripping. Furthermore, enhancements were made to facilitate remote interaction, allowing the weapon to be maneuvered from afar. Remote interaction was also extended to allow the user to "dive" to it from a distance.

To enhance gameplay clarity, tags were implemented to distinguish between the player's weapons and enemies. This was a crucial development step, enabling the user to fire bullets accurately in the desired direction with the trigger button, which leaves a red mark upon impact. By assigning distinct tags, confusion between the player's weapons and enemies was effectively eliminated, ensuring smoother interaction and gameplay. . All weapons use the same script, but their functions are separated. When

weapons are wielded by enemies, three extra modes have been created. a) the automatic shot towards the user's head when the enemy has reached a certain distance from our user;this is achieved through coroutines by calling the fire bullet method to the user every 2 seconds, only as long as the weapon is the enemy's, otherwise we turn it into the user's weapon. b) when the user shoots the enemy correctly the weapon leaves the enemy's hands and is thrown towards the user; this is done through the script managed by velocity, calculating the distance between the user and the enemy, the weapon flies through physics and velocity, so that the user can then catch it directly when the enemy is terminated.

The *FireBullet* script determines the target's identity, whether it's a player or an enemy, using the *SetIsEnemy* method. Upon activation (triggered by the *grab.activated* event), it generates a bullet, propelling it with specific direction and velocity. When the bullet from a player hits an enemy, the *SliceDeathSuperHot* script invokes the Death method from the SuperHot class to execute the logic for the enemy's demise. Additionally, the *XRGrabInteractableToAttach* script adjusts the interaction based on whether the player is using their right or left hand, setting the attachment point (*attachTransform*) accordingly.

5.1.3 Enemies Implementation

Enemies play a crucial role in the game. Initially, their models were downloaded in the .fbx [91] format and then imported into the Unity project. These models were converted to humanoid forms to accommodate two key animations: running and targeting. An animator was set up to manage these animations simultaneously, with a single parameter controlling the transition between them. To animate the enemies and enable them to move around the game environment, various tools available within Unity were utilized.

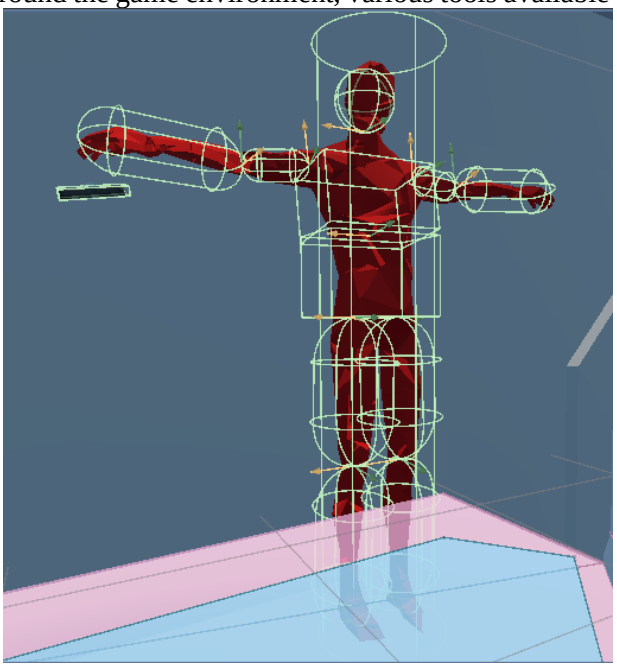

Figure 30: SuperHot Enemy with his colliders, holding a weapon

5.1.4 Nav Agent

Nav Agent is a Unity component used to automatically navigate characters (NPCs - non-Player Characters)

within the game. It allows NPCs to move autonomously through the game world, avoiding obstacles and following specific paths. It determines which areas are accessible to NPCs and which are not, allowing them to avoid obstacles and find routes in the virtual world.

5.1.5 Nav Mesh Surface

The Nav Mesh Surface is a surface on which Nav Agents can navigate. In essence, it is a two-dimensional representation of the game environment, showing where NPCs can move. It controls the movement of NPCs within the game, allowing them to move autonomously from one point to another, follow the player, or move purposefully through the environment.
The Nav Mesh Surface shapes the areas that NPCs can move around, defining a "virtual map" with the accessible and unreachable areas. This helps in the correct and logical movement of NPCs in the environment. By using the Nav Mesh Surface, NPCs can interact more naturally with the environment and various elements of the game, such as doors, stairs, and other obstacles.

We select the environment, adjust the height and area occupied by the nav mesh agent, and then bake the environment. In the selected environment, NPC's have 2 floors to navigate so the appropriate settings were chosen. To allow enemies to behave appropriately towards the user, through code, methods were developed to follow the user based on his distance, when to shoot, when to die, when to fire the weapon. To achieve the death of enemies, physics methods called ragdolls were used.

The Ragdoll is a technique in video games used to give character models a more realistic and natural response to physical forces, such as gravity or collisions. It is often used to depict a character's movement when he "dies" or becomes unconscious in the game. Ragdoll models react to physics forces in a more realistic way. For example, when an enemy is shot, his body will move and fall in a natural way, according to the direction and force of the shot. The ragdoll system allows models to respond dynamically to various situations in the environment, such as the impact of explosions or interaction with other objects. The use of a ragdoll helps increase the immersiveness of the game, as it offers more convincing reactions of the characters to the various events of the game.

In our case, the ragdoll system contributes to all the above plus an extra function when the enemy receives a bullet at a certain point. A tool called Ezy-Slice was used to chop it. Ezy-Slice is a free tool for Unity that provides slicing functions for objects in 3D environments. Ezy-Slice allows developers to slice objects dynamically, that is, cut or divide objects into smaller pieces. This is especially useful in games where there are battle or disaster scenes. The Ezy-Slice can be used to slice enemies at the point where they are attacked. For example, if an enemy is shot, the point of contact of the bullet with the enemy's body can create a visually convincing dissection.

5.1.6 Convert Skinned Mesh Renderer to Mesh Renderer

Typically, Ezy-Slice does not directly support Skinned Mesh Renderers, which are often used for animated character models with moving parts (such as NPCs or enemies). The solution applied, converting the Skinned Mesh Renderer to the Mesh Renderer, allows compatibility with the Ezy-Slice. This conversion usually requires copying the mesh to a new object that uses a Mesh Renderer. After conversion, NPCs or enemies can be realistically shredded when attacked, enhancing immersiveness and in-game interaction. Thus, when a bullet is fired into one of the enemy's skinned mesh renderers, the skinned mesh renderer is automatically converted into a mesh renderer, and the Ezy-Slice creates two upper hall and lower hall pieces to dynamically separate them, and the enemy enters dead mode by entering ragdoll inactive mode. This was achieved through a script attached to each collider of ragdoll. As soon as the bullet tag is detected,

the rest of the functions are activated.

5.1.7 SuperHot script

The *SuperHot* script manages the enemy, using a coroutine (*FireAtIntervalsCoroutine*) to shoot aimed at the player at specific intervals. *NavMeshAgent* is also used to move towards the target (player) and the Animator to visualize movements. It also adjusts the aim of the weapon towards the player. It includes methods for activating and deactivating ragdoll mode, i.e. the realistic performance of body movement during impact or destruction. The Death method manages the various functions that must occur when the enemy is killed, such as applying the impact force, activating the ragdoll, and producing the pieces from the enemy's body. Additionally, it uses the Ezy-Slice library to break the enemy into pieces at the time of death. It includes the ability to add visual effects upon enemy death, such as creating objects after breaking.

5.1.8 Time Manager

Another important element of the game is the regulation of game time and this is handled by the TimeManager script. When the player moves, enemies also move correspondingly fast. This is because the player must make a strategy before moving to avoid or act quickly in order not to leave enemies, named "Dynamic Time Flow Strategy". For this reason, a script with parameters the two controllers and the head, fluctuates the time of the game, based on their velocity. The *TimeManager* script uses a dynamic method to adjust the flow of time in the game, based on the player's movement. This can add an interesting dynamic to the gameplay, as players have control over the speed of the game through their movements. It's a technique that intensifies immersion and adds an extra dimension to the game's strategy and interactions.

5.1.9 Remarks

This game was instrumental in demonstrating how full-body tracking can elevate action-based gameplay. The ability of players to physically manipulate their bodies to avoid obstacles introduced a new level of engagement. Analysis of the game revealed improved player reflexes and a deeper sense of immersion. Challenges encountered such as ensuring fast-moving follow-up response, providing valuable insights to optimize performance for action gaming.

5.2 Use Case 2: Beat Saber

Beat Saber is a VR rhythmic game where players use lightsabers to cut musical notes that match the rhythm of the song. The game combines music, movement and rhythm in a fun and energetic experience. Players stand on a virtual treadmill and use VR controllers as lightsabers to cut approaching notes from various directions. Each note requires to be cut in a specific way, and success depends on the accuracy and pace of the player. It was created and released in 2018, quickly gaining a lot of popularity in the VR community. It uses the Unity Engine to create a smooth and interactive VR experience, with an emphasis on precision and responsiveness of movements.

5.2.1 Scene and effects

Initially, to ensure players experience a sense of isolation immersed in darkness, walls with outward-facing normals were constructed, visible only from the interior. The utilized effects, sourced as free assets from the Unity Asset Store, underwent modifications to suit the project's requirements. Moreover, a structure comprising numerous cubes was employed, alongside a script that gradually rotates each object around its center. To create the fog effects, a shader was created, dealing with the effects computations. Also very important is the trace left by each lightsaber while the user shakes them. This is a script method where the vertices and triangles of the lightsaber are stored, while the script updates the position of the mesh vertices within *LateUpdate*(). This is done based on the position of _tip and _base objects, simulating the top and base of the weapon, respectively. The positions of the vertices are refreshed to form the trace, creating a dynamic graphic effect that follows the movement of the weapon.

5.2.2 Lightsaber

For the lightsaber character we used a free 3D object, downloaded from the internet [92]. The upper part is an emissive color cube which looks like neon light. Then, the corresponding colliders were added to allow the user to interact and cut the corresponding triangles that will come on him. When an object comes into contact with the sword, the script checks to see if the object is of the correct color (red or yellow, respectively) and then calls the *PerformSlice* method to perform the slicing process. It uses the Ezy-Slice library to slice objects at the point of contact with the sword. This involves creating two new items from the original, corresponding to the shredded parts. Produces cutting effects when shredding, to enhance the visual representation of the action. It evokes a feeling of haptic feedback when the two swords collide, offering a sense of plausibility to the interaction. In *Update*, it calculates the current movement of the sword and refreshes the cutting direction depending on the movement. After shredding, new items receive a *Rigidbody* and move based on the force of the blow.

5.2.3 3D object modification within Unity

The pro-builder is a Unity tool that allows the user to modify the edges, faces, or normals of a 3D object within Unity. Internally the tool contains a shader responsible for displaying an emitive color. In our case we used it to directly manipulate and simplify the triangles geometry, optimizing the 3D object to a simpler form, maintaining the emissive color feature internal shader. This aspect is invaluable in creating more dynamic and visually compelling game objects that can react to the game's lighting environment or stand out on their own, adding depth to the game's visual presentation. Such visual cues are essential in rhythm-based games, where visual feedback and synchronization with music are key to the gameplay experience.

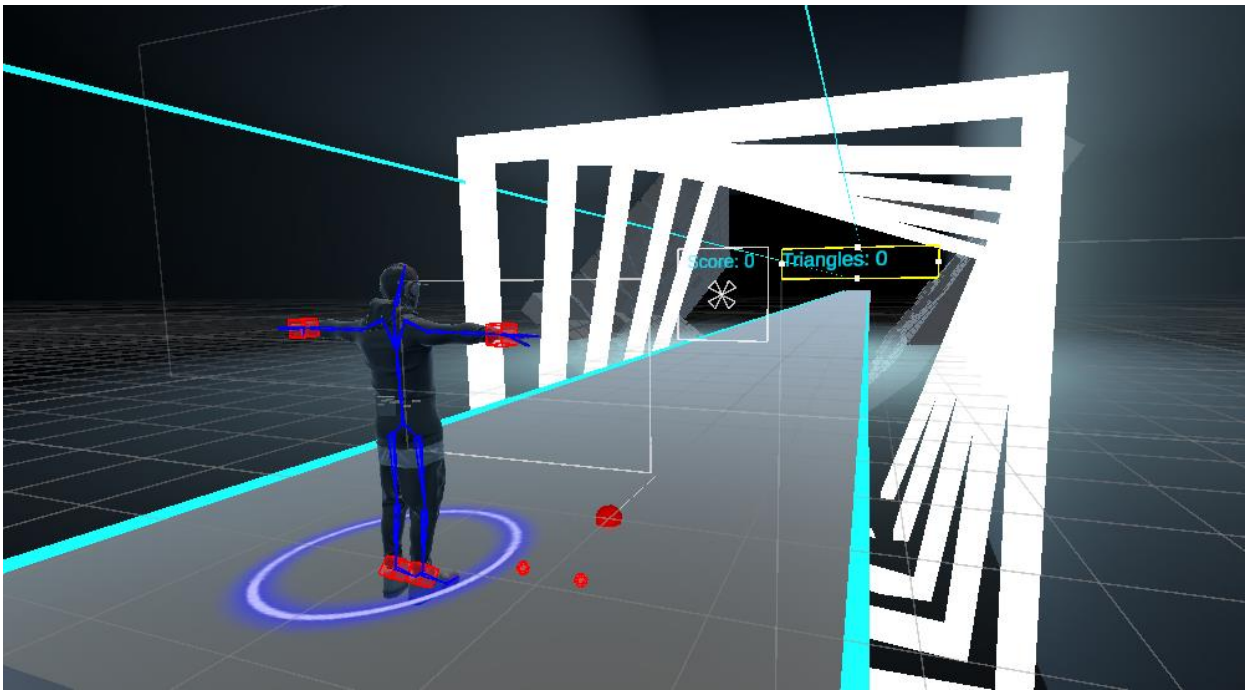

Figure 31: The Beat Saber game with it's visual effects

Create BeatSaber logic

The *BladeSaberManager* script implements functions similar to those of the Beat Saber game in Unity. During initialization, the script loads the prefabs for the green and yellow blocks as well as for the bombs from the application's resources. The *PlayMusic* method starts playing the music with a delay as defined by the *musicDelay* variable. A coroutine is used to load the Beat Saber map data from a file. The data is loaded by using *UnityWebRequest*. The map data is analyzed and converted by JSON into Unity objects. Based on the map data, the script determines when and where blocks will be created in the game. A timer is used to display the blocks synchronized with the music. Blocks appear in the game at set locations and times, based on their location on the map and the rhythm of the song.

In more detail, the game reads an EasyStandard file .dat, essential for generating a dynamic and engaging gameplay environment that aligns with the music. This file is utilized to dictate the characteristics of each block that appears in the game, such as its color (be it green or yellow) and whether it's a bomb. Additionally, it includes details regarding the display of each block in relation to the music's rhythm. Consequently, this facilitates the design of an interactive and rhythmic gaming experience where players engage with blocks in sync with the musical beats.

The script uses *UnityWebRequest* to load the data from a json file and then processes it to create the corresponding blocks in the game. Based on the information in the file, the script dynamically develops the content of the game, creating an environment that changes and adapts according to the pace and difficulty set by the file. Thus, the entire operation of the game is achieved, in what order each triangle will appear, whether they will be together, at what distance either horizontally or vertically. The following classes explain the entire implementation.

BlockData: This class is used to store data for each block that appears in the game.
Time: The time at which the block should appear in the game.
LineIndex: The position of the block on the horizontal axis.
LineLayer: The position of the block on the vertical axis.
Type: The type of block (red, blue, bomb).
CutDirection: The direction in which the block should be cut.
BlockType: A simple enum that defines the different types of blocks that the game can have.
MapData: This class represents the data on a game map.
Notes: A table of type BlockData, containing the data for all blocks that should appear on the map.
BeatSaberFile: This class is used to store data loaded from a Beat Saber file.
_version: The version of the data file.
_notes: A list containing the Block Data for the game map.

5.3 Use Case 3: Soccer game

For interaction with the lower body, and specifically with the legs, a soccer game was created. Essentially, the user has the soccer ball in front of him on a field and can simultaneously with his feet and with the movement to score a goal in front of him.

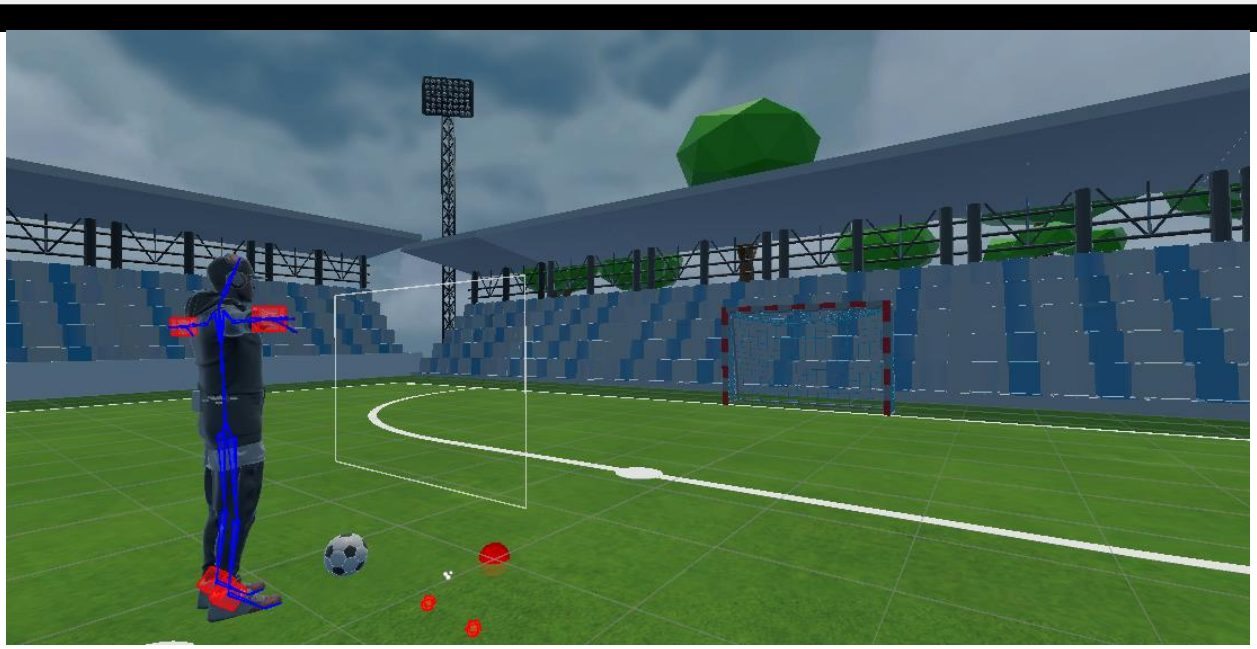

Figure 32: The soccer game scene with it's visual effects

5.3.1 Scene creation

The selection of the suitable stadium model was determined by the number of triangles it comprised and its complexity. Typically, upon investigation, it was found that most models possess a high triangle count. Since the user does not interact with the stadium, the game does not need to use a high level of detail stadium, as it would complicate their integration into VR environments and would increase the rendering load without providing an actual benefit on the application. For these reasons, an analytical, almost low poly 3D model of a stadium was imported [93] where it was modified in the isolated objects it contains, as well as in the colors of each object. One effect added is the cloth component in the goal net. The Cloth Component incorporates fabric physics into the game world. When applied to a soccer goal, it can produce the following effects: The goal net will respond realistically to movement, such as waves from the wind or the effect of the ball when it falls into the net. The Cloth Component allows the net to interact with other objects, such as the ball or players, in a way that mimics physics. To achieve the desired look and feel, several parameters are set:

Elasticity: Determines how elastic the fabric of the net will be.
Wind Resistance: Adjusts how the net will respond to the influence of the wind.
Gravity: Controls the effect of gravity on the net.
Conflict: Regulates how the net will react to collisions with other objects.
Similarly, colliders have been added to prevent the ball from leaving the goal.

5.3.2 Implementation

Obviously, the important element is to create the right ball so that it rolls and interacts with the colliders of the feet. The appropriate adjustments were made to the rigidbody and collider of the ball. Additionally, a script that creates a new ball as soon as it detects the ball entering the goal post, was also created. This may happen as many times as the user wishes.

5.3.3 Remarks

The development of a football simulation game presented whole-body monitoring in a sports context. The game's success in simulating realistic ball physics and player interaction was remarkable. However, it also brought to light challenges such as the need for more physical play space and the difficulty of translating some more subtle foot movements into play.

5.4 Menu UIs

To switch between games, UIs were used and created which have direct interaction with the player's raycasts. More specifically, the interaction of UI buttons with raycast through the XR Interaction Toolkit in Unity is a process that allows the user to interact with UI elements in a virtual environment (VR). The menuwas created in the starting room, where the user is initially located, so the user may select, through hand raycasts the desired game to play.

5.4.1 Basics of Interaction

- Raycast Interactor: This component is part of the XR Interaction Toolkit and acts as a kind of pointer, as mentioned before. It is used to transmit a ray from the user (usually from the VR controller) to the UI.
- UI Buttons: Buttons in the UI must be set to respond to raycast rays. This allows the buttons to work when the raycast "taps" or selects a button.
- Event System: In Unity, the Event System manages input events, such as touches or raycasts. To interact with the UI, the Event System must be configured to recognize actions by the Raycast Interactor.

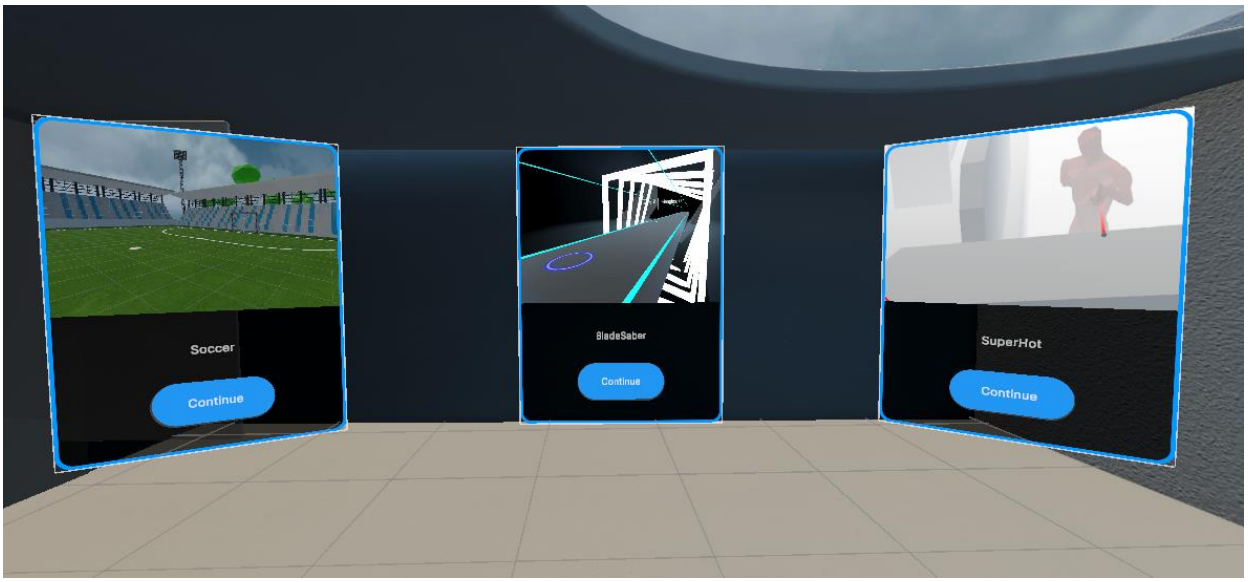

Figure 33: The Menu UI implemented to start each game with any order.

5.4.2 Interaction Process

Radius: When the user points a UI button at the controller, the Raycast Interactor emits a beam connected to the Event system. If the beam is relieved by a UI button, Unity's Event System detects the contact. When the Raycast Interactor "hits" the button, an event such as the *onClick* event of the trigger button is triggered. The button then performs its predefined function, such as changing an item or enabling a feature in the game.

The *MenuSpawn* and *SceneManager* scripts manage the appearance and functionality of menus (Main Menu and Pause Menu) in Unity for a VR game.

5.4.3 Script MenuSpawn

- Menu Display: Detects when the user presses a button (via *button.action*) and activates or deactivates the menu accordingly.
- Reboot to Main Menu: Includes the *BackToRoom* method, which restores the game back to the Main Menu.
- Menu Position Control: Adjusts the menu position in relation to the user's position and orientation (head tracking). It freezes the time when the menu is displayed and restores it when the menu is disabled.

5.4.4 Script SceneManager

- *Scene Management*: Controls switching between various game scenes, such as activating SuperHot, BladeSaber and Soccer game rooms.
- *Transition Effects:* Uses screenFader to create a visual transition effect when changing scenes.
- *Object On/Off Management*: Enables or disables specific objects and scenes depending on the currently active scene.
- *Overall Operation:* The two scripts work together to create a dynamic and interactive menu experience in a VR game. *MenuSpawn* handles the immediacy of appearance, position, and interaction of the menu with the user, while *SceneManager* manages the transition between various scenes and game modes.

5.4.5 ScreenFader script

The *ScreenFader* script is designed to provide a simple fading function to an image in Unity, usually used to create transition effects between different states in the game. In *Start*(), the script ensures that the image (*fadeImage*) is fully transparent at the beginning, setting the alpha channel(s) of the color to 0. The *FadeOutInSequence*() method calls the coroutine *FadeOutAndIn*(), which controls the fade to black process and then back to clear. This is done through a linear interpolation (Lerp) to smoothly transition transparency within a specified time (*fadeDuration*).

5.4.6 Evaluation of Full Body Tracking

To evaluate the fidelity and responsiveness of our full-body tracking system within virtual environments we implemented the *BodyTrackingAnalysis class*. Central to its design is the objective to meticulously compare the movements of an avatar to the user's real-world actions, utilizing physical controllers or trackers as the unequivocal standard for ground truth. This rigorous examination is important for discerning the effectiveness of the tracking system across various applications, embodying a crucial component in the development and refinement of virtual reality technologies.

At the foundation of the class lies its capacity to capture the nuances of human motion, sorting movements into distinct categories—none, small, medium, and large—based on predetermined thresholds. These thresholds are not arbitrarily set; rather, they are carefully calibrated to encapsulate the full spectrum of human movement, from the slightest twitch to expansive gestures. By doing so, the *BodyTrackingAnalysis* class ensures no detail is overlooked, no matter how minute, thus painting a comprehensive picture of the user's interaction within the virtual space.

Operating on a frame-by-frame analysis, the class diligently records the positional differences between the user's physical movements and the avatar's corresponding actions. This continuous logging process, coupled with the categorization of movement sizes, lays the groundwork for an in-depth evaluation of the tracking system's precision. Further enriching this analysis is the calculation of latency for medium and large movements, which sheds light on the system's responsiveness—particularly how swiftly the avatar can mirror significant user movements, a factor critical to user experience.

As the class encompasses a tracking session, it doesn't merely collate the collected data. Instead, it embarks on an organizational phase, where data entries are sorted by tracker type and timestamp, thereby imposing a coherent structure upon the dataset. This meticulous organization facilitates a seamless transition to the analysis phase, where the raw data can be thoroughly examined to unveil trends, pinpoint areas needing improvement, and guide the technological advancements.

The evolution of the *BodyTrackingAnalysis* class has introduced a layer of sophistication to this evaluative process. By generating separate summaries for each controller-tracker, the class offers a bespoke analysis of each tracked limb or object. This granular breakdown is instrumental in identifying specific tracking discrepancies and is complemented by the aggregation of total movements, addressing the potential overestimation of significant movements that a purely frame-by-frame approach might yield. Consequently, the class not only quantifies the frequency of movements but also contextualizes their significance within the broader scope of user activity.

The final form of the class, therefore, is a testament to the intricate balance required to assess the VR tracking system. It harmonizes detailed data collection with strategic summarization, providing both the granular insights necessary for technical scrutiny and the overarching summaries conducive to broader analyses. This dual-faceted output, comprising detailed logs and synthesized reports, ensures that the class caters to diverse analytical needs, from in-depth investigations to executive summaries.

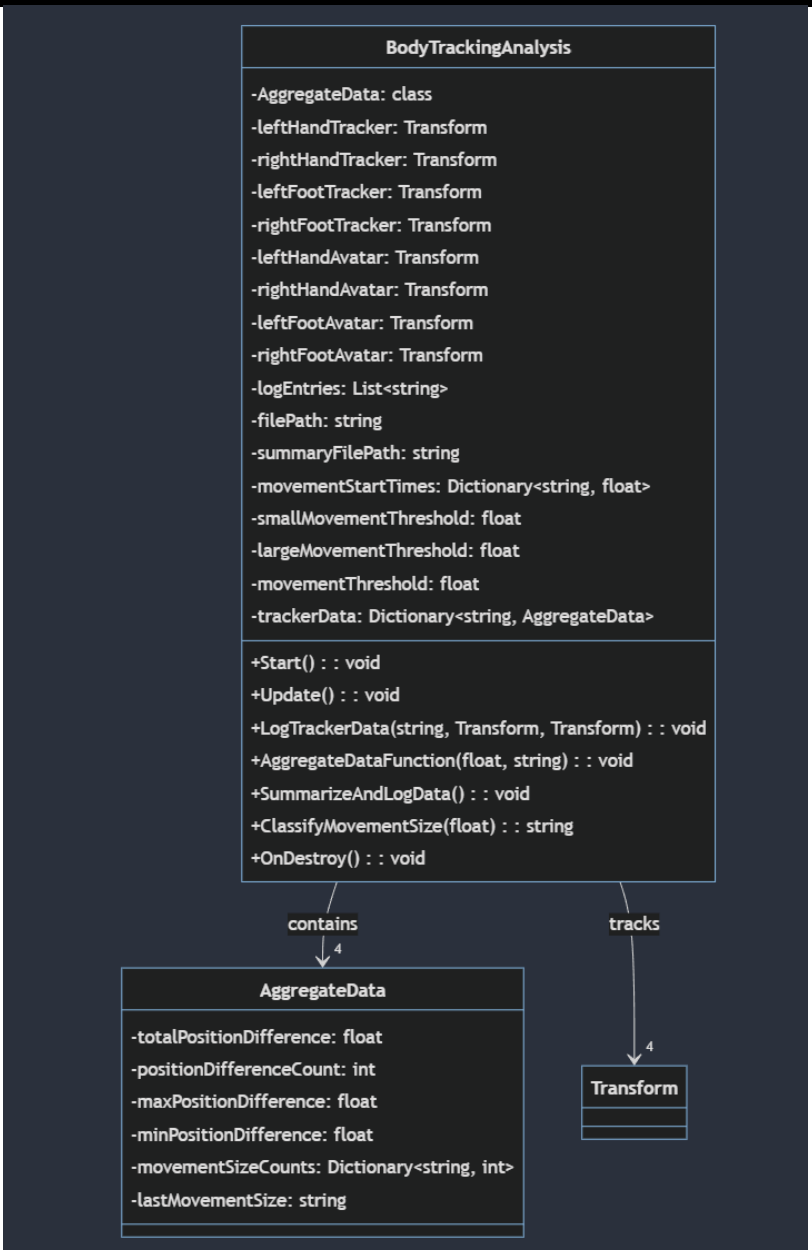

Figure 34: The BodyTrackingAnalysis class diagram which contains the AggregateData class and tracks the Transform class for each effector.

5.4.7 Measuring Tracking Accuracy

This class employs a 30-second gameplay interval for data capture, providing a comprehensive dataset recorded at each frame to derive the specified metrics. The 30-second timeframe was selected to balance the extensive calculations required for in-depth frame analysis with the practical need to adequately test each game's features. This duration aligns with the typical gameplay scenarios implemented in our use cases, ensuring a representative sample for analysis.

For each use case, we provide a table that analyzes the tracking of each part contained in the study. The SuperHot and the BeatSaber games required upper body tracking (hands) while the soccer game required lower body tracking. Each field is described below.

The **Average Position Difference** provides a holistic measure of tracking accuracy over a gameplay session. It is calculated by summing the positional discrepancies between the avatar's limbs and the physical trackers for every frame where a movement is detected, divided by the total number of such frames. This average gives us an overarching view of how closely the system can mimic the user's movements, offering insights into the system's precision. For example, in a fast-paced game like Beat Saber, where players slice through flying objects with virtual sabers, a lower average position difference signifies a high degree of tracking fidelity, ensuring that the avatar's movements mirror the player's actions with minimal deviation.

The **Max Position Difference** underscores the worst-case scenario of tracking deviation within a session. It pinpoints the largest single discrepancy between the user's actual movement and the avatar's representation at any given moment. This metric is particularly revealing in scenarios that demand precise timing and spatial awareness, such as grabbing pistols in mid-air to shoot enemies in Superhot. A high max position difference in such a context could indicate moments where the system failed to accurately capture rapid or complex movements, potentially breaking immersion or leading to gameplay frustrations.

The **Min Position Difference**, on the other end of the spectrum, highlights the best case of tracking accuracy, representing the smallest discrepancy observed. This metric reveals moments where the virtual and physical realms align with near-perfect accuracy. For activities requiring subtle movements, such as aiming in Superhot, a minimal position difference is indicative of the system's ability to faithfully replicate fine motor actions.

The classification of movements, **none, small, medium, and large** within the The *BodyTrackingAnalysis* class determines the type of movement within the classification <**none, small, medium, and large**>**,** based on the position difference between the avatar's hand (or foot) and those of the physical controllers or trackers. This difference is an important metric, serving as the foundational data point from which the class evaluates the accuracy and responsiveness of the virtual reality tracking system. The essence of this measurement lies in comparing the virtual representation (the avatar) to the user's actual physical movements as captured by the controllers-trackers. When a user moves their hand or foot, the controller-tracker records this movement in real space, while the IK system attempts to replicate it in the virtual environment through the avatar's movements. The class calculates the positional difference in each frame, providing a real-time assessment of how closely the avatar's movements mirror those of the user. This difference is then analyzed to classify the movement's type. A "none" movements would indicate no discernible movement or a negligible difference, implying a near-perfect alignment between the avatar's and tracker's positions. "Small" movements suggest minor discrepancies, which could be crucial for applications requiring fine motor control. "Medium" and "Large" movements highlight more significant deviations, which might be acceptable in some contexts but could also indicate potential areas for improvement in tracking accuracy.

Each use case has different limits based on each occasion and they are fully analyzed.

5.4.8 Tracking Accuracy Results

Our methodology's precision was assessed through an evaluation conducted by a user of 1.81 meters in height, who had completed a calibration process. This process involved fine-tuning their controllers and trackers to ensure a seamless match between the positions of the VR controllers and the avatar's limbs, essential for fostering a sense of embodiment within the virtual environment. The used VR headset was a HTC Vive XR Elite with standard VR hand controllers and 2 HTC Vive Ultimate Trackers for the feet.

The proficiency of users with VR technology significantly impacts the data observed during gameplay. More experienced users, accustomed to the nuances of VR interaction, may demonstrate quicker, more precise movements, potentially resulting in lower Min Position Differences in fast-paced games like Beat Saber. Their familiarity could also contribute to more strategic and controlled movements in Superhot, minimizing the Max Position Difference by executing precise actions with fewer exaggerated movements. Conversely, newcomers to VR might exhibit larger discrepancies in their movement tracking as they adjust to the immersive environment, affecting the overall metrics.

The physical attributes of users, including height, reach, and mobility, play a crucial role in the performance metrics of full-body tracking systems. Taller individuals, for example, might exhibit larger Max Position Differences simply due to their larger range of motion, especially in games requiring extensive movements like the soccer simulation. Similarly, the effectiveness of lower body tracking can vary with the user's leg length, impacting the Average Position Difference observed in soccer gameplay. These physical differences necessitate a nuanced approach to analyzing tracking data, considering how individual body mechanics influence the interaction with the virtual environment.

The Beat Saber game offers a rhythm-based VR experience that tests the limits of upper body agility and coordination. In this game, players slice through flying blocks with lightsabers, matching the color and direction indicated on each block, all while keeping in time with the beat of the music. This setup makes Beat Saber an ideal environment for assessing the VR tracking system's ability to handle quick, complex movements. The game challenges the system to track fast, directional slices and rapid changes in body position, providing a comprehensive overview of its tracking speed, accuracy, and the smoothness with which it translates physical movements into in-game actions. Data gathered from Beat Saber gameplay can reveal how effectively the VR system captures the swift, precise motions required to slice through blocks, highlighting its performance in scenarios demanding both speed and precision.

In a 30 second game run, the recorded accuracy results per hand are summarized in this Table 4: Performance analysis of the Jarvriks IK for the Beat Saber game:

Controllers-Trackers	**Average Position Difference (cm)**	**Max Position Difference (cm)**	**Min Position Difference (cm)**	**None Movements**	**Small Movements**	**Medium Movement**	**Large Movements**
Left Hand	1.20	4.11	0.07	5%	10%	30%	55%
Right Hand	1.23	3.54	0.04	3%	12%	35%	50%

Table 4: Performance analysis of the Jarvriks IK for the Beat Saber games showing the position differences and the percentage of total movements performed.

The critical metric of interest is the Min Position Difference, which measures the tracking system's ability to capture the most subtle and precise movements essential for slicing through blocks accurately. In this gameplay analysis, Beat Saber scored an average number on Min Position Difference. This

suggests that while the VR tracking system performs well in recognizing fine movements, there's room for improvement to reach the pinnacle of precision required for mastering the game. The performance indicates a satisfactory level of tracking that allows for an engaging gameplay experience, but further optimizations could enhance the system's sensitivity to subtle wrist and arm movements, elevating player immersion and accuracy in hitting the beats.

In the Superhot game, players find themselves navigating a matrix-like environment where time moves only when they do as analyzed before and the players must dodge bullets, grab weapons, and counter enemies in slow motion. This makes the game an ideal use case for evaluating the upper body tracking capabilities of our IK solution, particularly in terms of acquiring pistols with accuracy and reacting to dynamically changing environments. By analyzing tracking data from Superhot sessions, we can assess how well the system captures nuanced movements with the speed parameter. The ability of the tracking system to replicate these actions accurately and responsively in the avatar provides critical insights into its efficacy under high-stakes, precision-dependent scenarios.

In a 30 second game run, the recorded accuracy results per hand are summarized in this Table 5: Performance analysis of the Jarvriks IK for the Super Hot game:

Controllers-Trackers	**Average Position Difference (cm)**	**Max Position Difference (cm)**	**Min Position Difference (cm)**	**None Movements**	**Small Movements**	**Medium Movement**	**Large Movements**
Left Hand	1.5	6.0	0.3	4%	34%	42%	20%
Right Hand	1.4	5.7	0.25	6%	33%	40%	21%

Table 5: Performance analysis of the Jarvriks IK for the Super Hot game games showing the position differences and the percentage of total movements performed.

Superhot places emphasis on the Max Position Difference due to its gameplay mechanics, which involve making large, strategic movements to dodge bullets and engage with enemies. The analysis revealed an average to high number on Max Position Difference for Superhot. This range indicates that the tracking system is capable of handling significant movements to a certain extent, but there may be moments where the accuracy falters, potentially affecting the player's ability to interact precisely with the environment or perform quick, life-saving maneuvers. While the system provides a reliable experience for most gameplay situations, identifying and addressing the causes of higher Max Position Difference could further refine the game's immersive qualities and responsiveness.

The implementation of a classic soccer game within a virtual reality setup, particularly for analyzing lower body tracking, offers a unique and intricate perspective on the capabilities and limitations of VR tracking systems. In this game, players interact with a virtual ball by using their feet, mimicking the actions of kicking, dribbling, and shooting in a real-world soccer game. This setup not only engages users in an immersive experience but also subjects the tracking system to a rigorous test of its ability to accurately capture and replicate complex lower body movements within the virtual environment.

In such a scenario, the precision of lower body tracking is paramount. The game requires the VR system to detect a wide range of movements: from the subtle shifts in stance as the player positions themselves for a kick, to the dynamic and forceful action of shooting the ball towards the goal. Each of these actions provides valuable data for assessing the tracking system. The positional difference between the player's physical foot (as captured by the tracker) and the virtual representation of the foot (as seen in the avatar) becomes a critical measure of tracking accuracy.

By focusing on lower body movements, this soccer game scenario extends the analysis beyond the more

commonly assessed upper body tracking, offering insights into the VR system's comprehensive tracking capabilities. Such an analysis is crucial, as lower body movements tend to be more challenging to capture due to their dynamic nature and the ground-level positioning of the trackers, which may be more susceptible to occlusion and other tracking difficulties.

In a 30 second game run, the recorded accuracy results per foot are summarized in this Table 6: Performance analysis of the Jarvriks IK for the Soccer game

Controllers-Trackers	**Average Position Difference (cm)**	**Max Position Difference (cm)**	**Min Position Difference (cm)**	**None Movements**	**Small Movements**	**Medium Movement**	**Large Movements**
Left Foot	2.5	7.2	0.4	15%	30%	35%	20%
Right Foot	2.8	7.5	0.5	14%	32%	33%	21%

Table 6: Performance analysis of the Jarvriks IK for the Soccer game games showing the position differences and the percentage of total movements performed.

In the case of the soccer game, the Average Position Difference is a key indicator of how well the tracking system captures and replicates a range of foot movements. The gameplay scored a notably poor number on Average Position Difference, reflecting challenges in maintaining consistent accuracy with lower body tracking. This performance level suggests difficulties in accurately capturing the nuances of kicking, dribbling, and general footwork, which are integral to the soccer-playing experience. The notably poor performance points to a need for significant enhancements in lower body tracking technologies, emphasizing the importance of developing more sophisticated solutions to improve realism and player satisfaction in sports simulations.

This thesis advocates for the expansion of whole-body monitoring applications beyond the realm of gaming, venturing into virtual education, rehabilitation, and immersive storytelling. These areas present vast opportunities for engaging users in novel and meaningful ways. Analyzing the Jarvriks IK system across varied VR gaming scenarios—Superhot, Beat Saber, and a soccer simulation—provides insight into its performance, highlighting strengths and pinpointing areas for improvement. The Max Position Difference, particularly noted in Superhot with values up to 4.8 cm, illustrates instances where tracking precision deviates from actual movements during rapid or broad arm actions. This observation underscores the necessity of enhancing the system's accuracy to support a more immersive and responsive gaming experience. Beat Saber, which relies on the precision of slicing movements, achieves commendable results in Min Position Difference. However, the presence of larger Max Position Differences signals a need for increased accuracy in tracking the game's most dynamic actions to ensure consistent tracking performance. The analysis of the soccer game highlights the challenges in lower body tracking, with Average Position Differences indicating a variance in tracking fidelity that could detract from the realism of the gameplay. These findings collectively suggest focal points for future development: reducing Max Position Difference across games to improve action responsiveness and honing Average Position Difference, particularly in lower body tracking for sports simulations, to enhance the overall VR experience.

Chapter 6 Conclusion

This thesis confirms the transformative potential of full-body watching in VR gaming and beyond. While there are challenges to be addressed, developments in this area are quickly overcoming these obstacles, pointing to a bright future for full-body monitoring by creating deep immersive and interactive virtual

The advent of virtual reality (VR) has transformed digital interaction, offering immersive experiences that closely mimic real-world or fantastical environments. Central to this transformation is the sophisticated tracking of user movements and gestures, enabling the seamless translation of physical actions into virtual responses. This thesis delves into the nuanced realms of motion capture technology, focusing on the comparative study of outside-in and inside-out tracking methods—each with unique implications for VR application development.

Exploring tracking solutions reveals a spectrum from highly accurate, studio-grade systems like Vicon, known for their precision in capturing minute movements through a network of cameras and sensors, to consumer-oriented solutions like the Vive Trackers. Vicon systems, emblematic of outside-in tracking, excel in professional environments, offering unparalleled detail but at a significant cost and complexity. Vive Trackers, on the other hand, represent inside-out tracking, granting more accessibility and ease of setup for home users, though they trade off some accuracy and require a clear line of sight to base stations. To deepen the analysis, the exploration of tracking technologies spans from Vicon's studio-grade precision, ideal for professional use with its intricate camera and sensor network, to consumer-friendly Vive Trackers, showcasing the breadth from outside-in to inside-out tracking systems. Other notable mentions include SlimeVR and Tundra Trackers for budget-conscious users, HaritoraX for wireless convenience, and pioneering systems like Microsoft Kinect and PlayStation EyeToy, which democratized motion capture for home entertainment.

For IK solutions within Unity, the landscape is equally diverse. Mecanim provides an out-of-the-box solution for humanoid character animation, allowing for seamless blending of animations but with limited customization for advanced IK manipulations. Final IK and IK Plus extend Unity's capabilities, offering more granular control over limb movements and interactions with the environment, suited for developers needing precise control over character animations. Solutions like CinemaIK and FastIK cater to specific animation needs, while foundational algorithms like FABRIK and CCD solvers provide the mathematical backbone for precise movement simulations. This diverse toolkit underscores Unity's flexibility in addressing various animation and interaction requirements across gaming and VR landscapes. The animation rigging package could not be missed from the equations as it is the Unity standard today IK solution favourable from the developers. These tools cater to a wide range of applications, from game development to VR simulations, each with their pros and cons regarding flexibility, ease of use, and integration into existing workflows.

Inverse Kinematics (IK) solutions and tracking technologies in Unity, while pivotal for VR development, encounter several issues. IK challenges include achieving realistic limb movement without unnatural distortions, managing computational load for real-time applications, and ensuring accurate joint rotation that reflects natural human motion. Similarly, tracking technologies face obstacles like occlusion, where objects block the tracked entity, limiting tracking accuracy. Lighting conditions can also affect sensor-based systems, and setup complexity can deter users. Both IK solutions and tracking technologies strive to balance realism, performance, and user accessibility, continually evolving to address these challenges and enhance VR experiences.

Each IK solution faces several challenges. Beyond technical issues like realistic movement simulation and tracking accuracy, many IK solutions lack user-friendly interfaces and do not include a calibration process for players. This absence complicates adjusting avatars to match individual user dimensions accurately.

Furthermore, none offer a VR IK editor, preventing direct avatar manipulation within VR environments. This limitation significantly increases development time, as adjustments must be made externally in the editor, underscoring the need for more integrated, intuitive tools in VR content creation.

In response to the identified limitations of current IK and tracking solutions, the JARVRIKS Full Body Inverse Kinematics (IK) system has been developed to address these gaps. It offers a comprehensive framework that enhances accuracy and flexibility, crucial for creating lifelike virtual reality experiences. Distinguished by its user-centric calibration process, JARVRIKS enables precise adjustments to avatar dimensions, catering to individual user needs. Its innovative VR IK editor allows for in-environment avatar manipulation, streamlining the development process and significantly reducing time spent in traditional editing workflows. This advanced system marks a significant step forward in IK technology, promising to elevate the realism and interactivity of VR applications. Also, it supports both VR directly effector manipulation saving time directly for VR and saving any changes the user makes without the need of re-changing each effector utilities. Finally, the user can choose if the avatar is especially for VR or for editor game and can manipulate every and any effector from the single user-friendly UI has without manipulating external factors and objects. As a bonus, it supports auto wrist manipulation to avoid breaking avatar wrists.

Collectively, these use cases reveal both the strengths and areas for improvement in current VR tracking technologies. They illustrate the necessity for a balanced approach that combines high accuracy, low latency, and a deep understanding of the nuances of human movement to create truly immersive and interactive VR experiences.

Across these use cases, several lessons emerge. Firstly, the diversity of VR gaming experiences demands equally versatile tracking solutions capable of adapting to a wide range of motion types and intensities. Secondly, the quest for enhanced accuracy and responsiveness in tracking systems is ongoing, necessitating continuous refinement of both hardware components and the algorithms that interpret movement data. Lastly, the user experience benefits from a seamless integration of physical movements with their virtual counterparts, highlighting the importance of intuitive, user-friendly interfaces for calibration and customization. The lessons learned from these analyses advocate for continued technological advancements, emphasizing the importance of comprehensive testing across diverse gaming scenarios to push the boundaries of what's possible in VR interaction. Also, the insights garnered from these analyses contribute to a deeper understanding of the current state of VR tracking technology. They underscore the potential of VR to provide engaging, immersive experiences that closely mirror real-world interactions. However, they also reveal the critical need for ongoing innovation to address the nuances of human movement, ensuring that VR environments not only captivate but also accurately reflect the complexity of our physical actions.

6.1 Future Work

The application of full-body tracking technologies extends far beyond gaming, encompassing fields like virtual education, physical rehabilitation, and immersive storytelling. These sectors offer rich contexts for engaging users in more meaningful, interactive experiences. Future research could focus on customizing full-body monitoring systems for specific educational curriculums, rehabilitation exercises, or narrative experiences, evaluating their impact on learning outcomes, physical recovery, and emotional engagement, respectively.

A significant challenge in advancing full-body monitoring technology lies in merging high-speed data processing with precise motion capture to achieve real-time responsiveness without sacrificing accuracy. Further complicating this endeavor is the integration of sophisticated tracking technologies with artificial intelligence for predictive movements, augmented reality for richer interactive experiences, and cloud

computing for enhanced scalability and accessibility. These integrations face hurdles such as ensuring seamless data interoperability across different platforms and safeguarding user privacy and data security in cloud-based implementations.

Exploring the scalability of full-body tracking systems for large-scale social VR platforms presents a significant area for future research. As virtual spaces become more populated, the challenge of maintaining accurate tracking for hundreds or even thousands of users simultaneously becomes paramount. This research could delve into optimizing network infrastructure and data compression techniques to support large-scale interactions without compromising tracking fidelity or user experience.

Another promising direction involves the convergence of full-body tracking with machine learning algorithms to predict and enhance user movements in real-time. By analyzing vast datasets of user movement, future systems could anticipate user intentions, adjust tracking algorithms dynamically to improve accuracy, and even suggest movement optimizations for tasks within the VR environment. This predictive approach could revolutionize user interaction, making VR experiences more intuitive and satisfying.

Enhancing the accessibility and usability of full-body monitoring systems like Jarvriks IK is paramount for broadening their application beyond gaming to include entertainment, fitness, rehabilitation, and education. Making these systems more accessible involves simplifying the setup process and refining user interfaces to be more intuitive, thereby appealing to a wider audience, including non-gamers and VR novices. Challenges in this domain include designing systems that are straightforward to configure and operate by individuals without technical backgrounds, reducing the intimidation factor associated with high-tech VR systems. Additionally, there is a pressing need to establish universal design standards that accommodate a diverse range of users, including those with disabilities, to truly democratize access to VR technology. Achieving a balance between the complexity inherent in full-body tracking and user-friendly design, without compromising the system's effectiveness, remains a critical hurdle in enhancing both accessibility and usability.

References

[1] R. D. Caballar, What Is the Uncanny Valley? https://spectrum.ieee.org/what-is-the-uncanny-valley, 2024.

[2] C. Polona , G. A. Augusto and G. Stefan , A Survey of Full-Body Motion Reconstruction in Immersive Virtual Reality Applications, IEEE Trans Vis Comput Graph, 2020.

[3] J. Y. Jackie, C. Tuochao , Q. Fang , . S. L. Monica and A. L. James, HybridTrak: Adding Full-Body Tracking to VR Using an Off-the-Shelf Webcam, New Orleans, LA, USA: SIGCHI, 2022.

[4] S. Mohit , A. S. R. and J. Byunghoo , Inside-Out Magnetic Tracking for Virtual/Augmented Reality Applications, IEEE Sensors Journal , 2021.

[5] M. Riccardo and J. Aleotti, Evaluation of the Oculus Rift S tracking system in room scale virtual reality, 2022.

[6] https://blog.vive.com/us/vr-full-body-tracking-guide-pros-and-cons-of-tracker-technologies/, Self Tracking Technology.

[7] A. Filippeschi , N. Schmitz , M. Miezal , G. Bleser, E. Ruffaldi and D. Stricker , Survey of Motion Tracking Methods Based on Inertial Sensors: A Focus on Upper Limb Human Motion, 2017.

[8] S. Tang, W. Yang, . A. Bajenov and Y. Shen, "Inertial-Measurement-Unit (IMU) Based Motion Tracking for Biomorphic Hyper-Redundant Snake Robot., Honolulu, HI, USA: IEEE, 2017.

[9] W. Wei, K. Kurita, J. Kuang and A. Gao, Real-Time Limb Motion Tracking with a Single IMU Sensor for Physical Therapy Exercises, Mexico: IEEE, 2021.

[10] Q. Yuan and I.-M. Chen, Dynamic Behavior Tracking and Localization Method for Inertial Capture System., ScienceDirecter, 2013.

[11] W. Fang, L. Zheng and H. Deng, A motion tracking method by combining the IMU and camera in mobile devices., Nanjing, China: 2016 10th International Conference on Sensing Technology (ICST), 2016.

[12] H. Nyqvist and F. Gustafsson, A high-performance tracking system based on camera and IMU, Istanbul, Turkey: IEEE, 2013.

[13] M. S. Ikbal, V. Ramadoss and M. Zoppi, Dynamic Pose Tracking Performance Evaluation of HTC Vive Virtual Reality System, IEEE, 2020.

[14] S. Sitole, A. K. LaPre and F. C. Sup IV, Application and Evaluation of Lighthouse Technology for Precision Motion Capture, IEEE Sensors Journal , 2020.

[15] Skarredghost, https://skarredghost.com/2022/12/02/vive-trackers-full-body-vrchat-2/.

[16] Vicon Systems, https://www.vicon.com/.

[17] M. Jebeli, A. B. and A. Arshi, A study on validating KinectV2 in comparison of Vicon system as a motion capture system for using in Health Engineering in industry, Nonlinear Engineering, Volume 6, Issue 2, pp.95-99, 2017.

[18] T. U. Siaw, Y. C. Han and K. I. Wong, A Low-Cost Marker-Based Optical Motion Capture System to Validate Inertial Measurement Units, IEEE Sensors Letters, 2023.

[19] N. Rizaldy, F. Ferryanto, A. Sugiharto and A. I. Mahyuddin, Evaluation of action sport camera optical motion capture system for 3D gait analysis, Indonesia: RCMEManuE, 2020.

[20] O. Brunner, A. Mertens, V. Nitsch and C. Brandl, Accuracy of a markerless motion capture system for postural ergonomic risk assessment in occupational practice, International Journal of Occupational Safety and Ergonomics, 2022.

[21] R. J. Aughey, K. Ball, S. J. Robertson, G. M. Duthie, F. R. Serpiello, N. Evans, B. Spencer, S. Ellens, E. Cust, J. Haycraft and J. Billingham , Comparison of a computer vision system against three-dimensional motion capture for tracking football movements in a stadium environment,

Advanced Wellbeing Research Centre, Olympic Legacy Park: ISEA, 2022.

[22] N. Lerma and H. Gulgin, Agreement of a Portable Motion Capture System to Analyze Movement Skills in Children, Taylor & Francis, Inc, 2022.

[23] J. Antunes, https://www.provideocoalition.com/shogun-change-vfx-motion-capture/.

[24] Z. Zhang, Microsoft Kinect Sensor and Its Effect, IEEE MultiMedia, 2012.

[25] M. Zhang, Recent Developments in Game-Based Virtual Reality Educational Laboratories Using the Microsoft Kinect, International Journal of Emerging Technologies in Learning (iJET), 13(01), pp. pp. 138–159. , 2018.

[26] C. M. Tseng, C. L. Lai, D. Erdenetsogt and Y. F. Chen, A Microsoft Kinect Based Virtual Rehabilitation System, Taichung, Taiwan: 2014 International Symposium on Computer, Consumer and Control, 2014.

[27] Z. Gao, Y. Song, Y. Zhou and W. Xiong, Design of joint range of motion measurement based on Kinect, Xi'an, China: 2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC), 2021.

[28] S. Adalberto L. , V. Eduardo , A. Jason and G. Hans , Feet movement in desktop 3D interaction, Minneapolis, MN, USA: IEEE Symposium on 3D User Interfaces, 2014.

[29] G. Fitzpatrick, Kinect in Game-based Physical Rehabilitation.

[30] P. Engström, M. Axelsson and M. Karlsson, Combining structured light and ladar for pose tracking in THz sensor management, Proceedings of the SPIE, Volume 8745, id. 874515 11 pp. (2013), 2013.

[31] S. Flynn, B. Lange and A. Rizzo, Virtual reality rehabilitation - what do users with disabilities want?, International Conference Disability Virtual Reality and Associated Technologies, 2008.

[32] B. Lange, S. Flynn and A. Rizzo, Initial usability assessment of off-the-shelf video game consoles for clinical game-based motor rehabilitation, USA: Institute for Creative Technologies, University of Southern California, 2009.

[33] K. McConville and S. Virk, Evaluation of an electronic video game for improvement of balance. Virtual Reality, 16, 315–323 (2012), 2012.

[34] W. Li, Development and Evaluation of a Virtual Reality Therapy System for Children with Hemiplegic Cerebral Palsy, University of Toronto: Division of Engineering Science, 2007.

[35] HaritonaX Wireless, https://en.shiftall.net/products/haritorax.

[36] Tundra Labs, https://tundra-labs.com/.

[37] SlimeVR, https://docs.slimevr.dev/.

[38] T. M. Takala, Y. Hirao, H. Morikawa and T. Kawai, Martial Arts Training in Virtual Reality with Full-body Tracking and Physically Simulated Opponents, Atlanta, GA, USA: IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2020.

[39] H. . L. Miller, I. R. Zurutuza, N. Fears, S. Polat and R. Nielsen, Post-processing integration and semi-automated analysis of eye-tracking and motion-capture data obtained in immersive virtual reality environments to measure visuomotor integration, ETRA '21 Adjunct: ACM Symposium on Eye Tracking Research and Applications, 2021.

[40] M. I. Majid, A. Gauhar and A. R. Chandio, Three Axis Kinematics Study for Motion Capture Using Augmented Reality, Innovative Computing Review, 2023.

[41] https://www.sony.net/Products/mocopi-dev/en/documents/Home/Aboutmocopi.html, Sony Mocopi.

[42] S. Merker, S. Pastel , D. Bürger, A. Schwadtke and K. Witte, Measurement Accuracy of the HTC VIVE Tracker 3.0 Compared to Vicon System for Generating Valid Positional Feedback in Virtual Reality, Germany: Sports Engineering and Movement Science, Otto-von-Guericke University, 39106 Magdeburg,, 2023.

[43] J. Vox, A. Weber and K. I. Wolf, An Evaluation of Motion Trackers with Virtual Reality Sensor Technology in Comparison to a Marker-Based Motion Capture System Based on Joint Angles for Ergonomic Risk Assessment., Frank Wallhoff's Lab: MDPI, 2021.

[44] A. Bhargava, H. Solini, K. Lucaites, J. W. Bertrand, A. Robb, C. C. Pagano and S. V. Babu, Comparative Evaluation of Viewing and Self-Representation on Passability Affordances to a Realistic Sliding Doorway in Real and Immersive Virtual Environments, Atlanta, GA, USA: 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), 2020.

[45] Vive Ultimate Trackers, https://www.vive.com/us/accessory/vive-ultimate-tracker/.

[46] H. Lamine, S. Bennour, M. Laribi, . L. Romdhane and S. Zaghloul, Evaluation of Calibrated Kinect Gait Kinematics Using a Vicon Motion Capture System, y Informa, 2017.

[47] J. Donald, Indoor Flight of AR Drone 2.0 UAV in Vicon Motion Capture System, SemanticScholar, 2018.

[48] S. M. van der Veen, M. Bordeleau, . P. E. Pidcoe, C. R. France and J. S. Thomas, Agreement Analysis between Vive and Vicon Systems to Monitor Lumbar Postural Changes, MDPI, 2019.

[49] N. Hesse, Concurrent Validity of a Custom Method for Markerless 3D Full-Body Motion Tracking of Children and Young Adults Based on a Single RGB-D Camera, IEEE Transactions on Neural Systems and Rehabilitation Engineering, 2023.

[50] J. L. Ponton, Motion Matching for Character Animation and Virtual Reality Avatars in Unity, Cornell University, 2023.

[51] L. N. Fung and M. Andaya, A Befitting Single-Player Squash Program to Educate and Assist Disabled/Autistic People using Pose Estimation and Unity, USA: California State Polytechnic University, 2023.

[52] Y. Meng, Y. Zhan, J. Li, S. Du, H. Zhu and X. S. Shen, De-anonymization Attacks on Metaverse, New York City, NY, USA: IEEE INFOCOM 2023 - IEEE Conference on Computer Communications, 2023.

[53] W. Wei, Z. Xin, H. Li-li, W. Min and Z. You-bo, Inverse kinematics analysis of 6 – DOF Stewart platform based on homogeneous coordinate transformation Ferroelectrics, 517(1), 9–21, Taylor & Francis, 2017.

[54] C. Gosselin, Parallel Computational Algorithms for the Kinematics and Dynamics of Planar and Spatial Parallel Manipulators. Journal of Mechanical Design, 118(1), 4-9., ASME, 1996.

[55] W. Khan, V. Krovi and S. Saha, Recursive Kinematics and Inverse Dynamics for a Planar 3 R Parallel Manipulator. Journal of Mechanical Design, ASME, 2005.

[56] W. Khalil and S. Guégan, Inverse and direct dynamic modeling of Gough-Stewart robots, IEEE Transactions on Robotics, 20(4), 754–762, 2004.

[57] J. Adolf, J. Dolezal and L. Lhotska, Affordable Personalized, Immersive VR Motor Rehabilitation System with Full Body Tracking, Italy: Proceedings of the 16th International Conference on Wearable Micro and Nano Technologies for Personalized Health, 2019.

[58] J. Jiang, P. Streli, H. Qiu, A. Fender, L. Laich, P. Snape and C. Holz, AvatarPoser: Articulated Full-Body Pose Tracking from Sparse Motion Sensing, ECCV , 2022.

[59] A. Bhargav, Comparative Evaluation of Viewing and Self-Representation on Passability Affordances to a Realistic Sliding Doorway in Real and Immersive Virtual Environments.

[60] Y. Hoshikawa, , Y. Hashimoto,, . A. Moro, K. Terabayashi and . K. Umeda, Tracking of human groups using subtraction stereo, Taylor And Francis , 2011.

[61] ,. H. Yu and G. Liu, Research on calibration of optical see-through head-mounted display, IOP Publishing Ltd, 2019.

[62] S. Tripathi and B. Guenter, A statistical approach to continuous self-calibrating eye gaze tracking for head-mounted virtual reality systems, 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), 2017.

[63] D. Jo, Y. Kim, E. Cho, D. Kim and G. Lee, Tracking and interaction based on hybrid sensing for virtual environments, MDPI, 2013.

[64] A.Aristidou and J. Lasenby, Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver, University of Cambridge: IEEE, 2009.

[65] O. Hock and J. Šedo, Inverse kinematics using transposition method for the robotic arm., ELEKTRO, 2018.

[66] H. V. Nguyen, T. D. Le, D. D. Huynh and P. Nauth, Forward kinematics of a human-arm system and inverse kinematics using vector calculus, 2016.

[67] K. Tchoń and R. Muszynski, A normal form solution to the singular inverse kinematic problem for robotic manipulators: the quadratic case, 1998.

[68] M. Aghajarian and K. Kiani, Inverse Kinematics solution of PUMA 560 robot arm using ANFIS, Incheon, Korea (South): 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2011.

[69] N. Rokbani, R. Kumar, A. Alimi, P. H. Thong, I. Priyadarshini, V. H. Nhu and P. T. T. Ngo, Impacts of heuristic parameters in PSO inverse kinematics solvers, De Gruyter, 2022.

[70] R. P. Rangkuti, Kinematika Balik Menggunakan Neuro-Fuzzy Pada Manipulator Robot Denso, ITS, 2017.

[71] A. Aristidou and J. Lasenby, Fabrik: A fast, iterative solver for the inverse kinematics problem. Graphical Models, ScienceDirect, 2011.

[72] P. C. Santos, M. C. Santos, S. Sivcev, E. Omerdic, G. Dooly and D. Toal, Inverse kinematics of a subsea constrained manipulator based on FABRIK-R, Hampton Roads, VA, USA: OCEANS 2022, Hampton Roads, 2022.

[73] K. Srivastava, R. Das and S. Chaudhury, Virtual reality applications in mental health: Challenges and perspectives Industrial Psychiatry Journal, 23(2), 163–166, Medknow Publications, India: Wolters Kluwer, 2014.

[74] A. Martín, A. Barrientos and J. Del Cerro, The Natural-CCD Algorithm, a Novel Method to Solve the Inverse Kinematics of Hyper-redundant and Soft Robots, LiebertPub, 2018.

[75] P. L. jiao, Z. Y. lian, Q. Z. wen, C. Qin and H. Yue, Real-time virtual view synthesis based on GPU parallel programming, 2015.

[76] B. Ruf, J. Mohrs, M. Weinmann, S. Hinz and J. Beyerer, ReS2tAC—UAV-Borne Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices, 2021.

[77] A. R. Manual, https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.3/manual/index.html.

[78] D. Hunt, O. Dionne and S. B. Zappa, Advanced use cases for animation rigging in unity, Los Angeles: SIIGRAPH, 2019.

[79] Animation C# Jobs , https://blog.unity.com/engine-platform/animation-c-jobs.

[80] https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.1/manual/constraints/TwoBoneIKConstraint.html, Two Bone IK Constraint.

[81] https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.1/manual/constraints/MultiParentConstraint.html, Multi Parent IK Constraint.

[82] https://github.com/Stereoarts/SAFullBodyIK, StereoArts, SAFullBodyIK.

[83] R. FinalIK, http://root-motion.com/, 2017.

[84] A. D. Young, IMUSim: A simulation environment for inertial sensing algorithm design and evaluation, 2011.

[80] F. Herrmann, oCaCo: A Simulator Framework for Motion Capture Comparison.

[86] Unity Mecanim IK, https://docs.unity3d.com/Manual/InverseKinematics.html.

[87] L. C. Soto, Procedural generation of animations, 2021.

[88] CinemaIK., https://assetstore.unity.com/packages/tools/animation/cinemaik-free-demo-186194.
[89] EasyIK, https://github.com/joaen/EasyIK.
[90] FastIK, https://assetstore.unity.com/packages/tools/animation/fast-ik-139972.
[91] https://vrcmods.com/item/4251.
[92] https://sketchfab.com/models/32a5ac91b541438e8f29999e73ac54f1/embed?utm_source=website&utm_campaign=blocked_scripts_error.
[93] https://www.cgtrader.com/free-3d-models/sports/equipment/low-poly-football-stadium.

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Μεταπτυχιακή Διπλωματική Εργασία με τίτλο:

« Ολόσωμη παρακολούθηση και ενσωμάτωση σε VR**»**

καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν,
και η οποία έχει εκπονηθεί στο Πρόγραμμα Μεταπτυχιακών Σπουδών «Ανάπτυξη Ψηφιακών Παιχνιδιών και Πολυμεσικών Εφαρμογών» του Τμήματος Επικοινωνίας & Ψηφιακών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του/της
Αντώνη Πρωτοψάλτη

αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Υπογραφή Φοιτητή/τριας