Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Expanding immersion in virtual reality

Bachelor Thesis

2024
Tomáš Búcsi

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Expanding immersion in virtual reality

Bachelor Thesis

Study Programme:	Applied Informatics
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	Mgr. Iveta Bečková

Bratislava, 2024
Tomáš Búcsi

50591009

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Tomáš Búcsi
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Expanding immersion in virtual reality
Rozšírenie imerzie vo virtuálnej realite

Anotácia: Virtuálna realita (VR) ako moderná počítačová technológia poskytuje svojim používateľom množstvo užitočných výhod v rôznych oblastiach života. Aby bola VR ešte efektívnejšia, musí byť pokročilá a ponúkať vysokú úroveň pohltenia, aby používatelia takmer neboli schopní rozlišovať medzi fyzickou a virtuálnou realitou.

Cieľ: 1. Na základe dostupnej literatúry preskúmajte princípy pohlcujúcej VR.
2. Pomocou simulovaného humanoidného robota NICO, importovaného do virtuálneho prostredia Unity, manipulujte s niekoľkými vybranými vlastnosťami robota v rôznych modalitách, aby sa simulovaný NICO viac podobal na svoj fyzický náprotivok.
3. Pripravte scenár, ktorý sa bude dať použiť pri testovaní participantov, ktorí budú ponorení do VR s NICOm, a zhodnoťte jeho slabé stránky.

Literatúra: Bowman, D. A. and McMahan, R. P. (2007). Virtual reality: how much immersion is enough? Computer, 40(7):36–43.
Tham, J., Duin, A. H., Gee, L., Ernst, N., Abdelqader, B., and McGrath, M. (2018). Understanding virtual reality: Presence, embodiment, and professional practice. IEEE Transactions on Professional Communication, 61(2):178–195.
Wilkinson, M., Brantley, S., and Feng, J. (2021). A mini review of presence and immersion in virtual reality. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, pp. 1099–1103.

Vedúci:	prof. Ing. Igor Farkaš, Dr.
Konzultant:	Mgr. Iveta Bečková
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. RNDr. Tatiana Jajcayová, PhD.
Dátum zadania:	19.10.2023

Dátum schválenia: 01.11.2023

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

....
študent

....
vedúci práce

50591009

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Tomáš Búcsi
Study programme:	Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Expanding immersion in virtual reality

Annotation: Virtual reality (VR), as a modern computer-based technology, provides a number of useful benefits to its users in various areas of life. To be even more effective, VR needs to become advanced, offering a high-level of immersion such that the users would almost not be able to differentiate between the physical reality and the VR.

Aim: 1. Based on available literature, explore the principles of immersive VR.
2. Using the simulated humanoid robot NICO, imported in Unity virtual environment, manipulate a few selected features in various modalities making the simulated NICO more similar to its physical counterpart.
3. Prepare a script that will be for participants who would be immersed in VR with NICO, and assess its limitations.

Literature: Bowman, D. A. and McMahan, R. P. (2007). Virtual reality: how much immersion is enough? Computer, 40(7):36–43.
Tham, J., Duin, A. H., Gee, L., Ernst, N., Abdelqader, B., and McGrath, M. (2018). Understanding virtual reality: Presence, embodiment, and professional practice. IEEE Transactions on Professional Communication, 61(2):178–195.
Wilkinson, M., Brantley, S., and Feng, J. (2021). A mini review of presence and immersion in virtual reality. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, pp. 1099–1103.

Supervisor:	prof. Ing. Igor Farkaš, Dr.
Consultant:	Mgr. Iveta Bečková
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	19.10.2023
Approved:	01.11.2023

doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.... Student

.... Supervisor

Acknowledgments: Many thanks to my supervisor, Prof. Ing. Igor Farkaš, Dr., who has guided me and helped me complete this thesis through countless hours. I would also like to thank my consultant, Mgr. Iveta Bečková, whose consultations were not only helpful but also insightful. Without their help, this thesis could not have been written.

Abstrakt

V tejto práci sme sa ponorili do fenoménu imerzie a študovali rôzne spôsoby, ako sa dá zlepšiť používateľská skúsenosť. Na dosiahnutie tohto cieľa sme vylepšili existujúci projekt v prostredí Unity, ktorý obsahuje polohumanoidného robota NICO. Do tohto projektu sme pridali funkcie, ktorých cieľom je zvýšiť pocit imerzie vo virtuálnej realite. Tieto vylepšenia zahŕňali tréning neurónových sietí prostredníctvom posilňovacieho učenia na ovládanie pohybov robota NICO, zlepšenie jeho vizuálnej reprezentácie cez modelovania jednotlivých časti robota, ako aj pridanie pohybov a zvukových effektov. Nakoniec sme vytvorili skript na zefektívnenie používania konkrétnych vylepšení, aby bolo možné testovať, ktoré vylepšenia najefektívnejšie zvýšia pocit imerzie jednotlivca.

Keywords: virtuálna realita, imerzia, NICO, učenie posilňovaním, Unity

Abstract

We delved into the phenomenon of immersion and studied various ways to enhance the user experience. To achieve this goal, we enhanced an existing Unity project, which contains the semi-humanoid robot NICO by adding functionalities with the goal of increasing the sense of immersion in virtual reality. These enhancements included training neural networks through reinforcement learning to control NICO's movements, improving its visual representation as well as added additional movement and sound effects. Lastly, we created a script to streamline the use of specific improvements, to allow the testing of which improvements most effectively increase the individual's immersion.

Keywords: virtual reality, immersion, NICO, reinforcement learning, Unity

Contents

List of Figures

List of Abbreviations

VR - Virtual Reality
FoV - Field of View
HRTF - Head-Related Transfer Function
RL - Reinforcement Learning
ML-Agents Toolkit - Machine Learning Agents Toolkit
PPO - Proximal Policy Optimization
SAC - Soft Actor Critic

Chapter 1

Introduction

Virtual reality (VR) has long existed as an ambitious concept confined to the realms of science fiction literature, evoking imaginations of a distant future. However, in recent years, this concept has been transformed into an available technology which starts to blur the line between reality and computer generated content.

Its wide availability is thanks to the introduction of inexpensive consumer-grade reality enhancing headsets, making it possible for a larger consumer-base to have access to this technology. This in turn made it into an attractive platform for developers to create content, such as games, for these devices.

Even though the biggest focus of VR is in the entertainment and the video game industry, the use of VR has been integrated into other sectors of life. These industries include medicine, research, education and manufacturing. One way companies use VR, is that they use them as a training tool.

In businesses, where training is necessary, but may come with risks of financial loss or accidents, VR is an alternative, which is a cost-effective and safe way to educate new talents. It also engages personnel more than a traditional training program due to the gamification of the whole process. Addition of game-like elements to non-game environments such as game-mechanics, aesthetics and story, enhances productivity and raises interest in the presented topic (Cavusoglu et al., 2019).

VR also holds potential use in the realm of healthcare. It could be used for patients battling with fears of spiders, heights and public speaking along with other phobias. Additionally, VR could aid recovering patients in the rehabilitation process (Halbig et al., 2022).

Since VR is supposed to imitate the real world, it is important to recreate reality as faithfully as possible. To accomplish this, one must fool the users senses while they interact with the virtual world. Most systems try to achieve such feat with head mounted goggles, headphones and a pair of controllers. Though these may replicate a part of our world, creating a near perfect doppelgänger will require more sophisticated equipment aimed at every human sense.

For this reason, it is important to study what improvements are more impactful at creating a genuine representation of the world. These improvements are aimed at either refining the quality of the senses involved in modern VR experiences or incorporating additional other senses, thereby increasing the quantity of sensory information experienced.

Chapter 2

Background

To be able to fully study immersion and the improvements that enhance it, we first need to understand what immersion is and which elements are responsible for increasing the sense of immersion. Therefore, in this chapter, we will first be exploring immersion and all of its forms, after which we will start researching the software and algorithms that can be used to enhance immersion.

2.1 The Concept of Immersion

For VR to be useful in the future it needs to become advanced enough for humans to not be able to differentiate between traditional reality and the man-made world VR is trying to present. To achieve such a feat, it is essential for us to comprehend the nature of what we seek to replicate within the virtual setting. Common physical reality refers to the tangible interactions confined within the spatial, temporal, and material constraints of the physical world. On the other hand, virtual reality involves the creation of artificial simulations, typically recreating real-life environments, to enhance the perception of an imagined reality or situation (Tham et al., 2018). To blur the barrier of reality we need to increase the users presence or immersion, ideally both. While both these expressions have been somewhat convoluted and at times, even used interchangeably, they are both distinct terms (Wilkinson et al., 2021).

2.1.1 Presence versus Immersion

As mentioned, these terms are often used synonymously even though in the context of VR they hold distinct meanings. Immersion was defined by Kim et al. (2017) as follows:

> “Immersion, leveraging the user’s five senses, offers a remarkable ability to transport individuals into a virtual realm where they can perceive their

> surroundings, interact with others, and engage in activities as if they were genuine experiences. It encompasses the depth of engagement a user feels, evoking a range of emotions within the virtual space and manifesting them in relation to the simulated environment. In contrast, the concept of presence revolves around the phenomenon where users genuinely believe and feel as though they are physically present in the computer-generated world presented by the display technology. Consequently, immersion serves as the term employed to describe the technology that enables the emergence of presence, providing users with an immersive and captivating virtual experience."

Though the increase of immersion is desirable, miss-handling or incorrect implementation may lead to unfortunate side effects such as nausea, headaches and vertigo, also called VR sickness, directly caused by the discrepancy between visual information and the vestibular information received from the VR simulating movement (Tanaka and Takagi, 2004). As we now have an understanding of what immersion is, the next section will contain how it can be improved.

2.2 Improving Immersion

In a case study participants reported that a VR environment which provided them a rich sensory stimulation, such as visual, auditory and tactile feedback increased their sense of presence (Tham et al., 2018). Their findings also suggest that higher VR fidelity contributes to a greater sense of embodiment among participants. In other words, when the visual and auditory cues in the virtual environment are of higher quality and realism, individuals tend to feel more fully present and engaged within that virtual world. This enhanced fidelity strengthens the connection between the user's physical self and their virtual representation, resulting in a more immersive and convincing VR experience. To them, reality in the context of VR is defined by how effectively the VR hardware convinces their senses in any given scenario (Tham et al., 2018).

As immersion is inherently tied to technology that directly enhances presence, as a result we can enhance immersion by improving the realism of the VR environment through advancements in graphics and sound quality. However, immersion can also be further enhanced by augmenting the quantity and quality of sensory feedback that users receive during their VR experience. By providing users with more comprehensive and realistic sensory inputs, such as tactile feedback, haptic sensations, and spatial audio cues, we can deepen the level of immersion (Hecht et al., 2008).

Most modern commercially available VR systems come with a Head-Mounted Display (HDM), controllers, some variation of tracking sensors and they also may provide integrated or separate audio systems. This means that most devices mainly focus on the visual and audio aspects of VR, neglecting other senses such as touch. That's why we will first focus on audio and visual immersion.

2.3 Visual Immersion

One of the most sought after aspects of VR is its ability to be able to see, what was once on a screen, before us, in 3D, therefore visual immersion plays a critical role in VR systems. There are multiple aspects with which we can improve visual immersion, since we need to trick ourselves to believe we really are a part of the computer generated world. The most important aspects are: realistic rendering, improved graphics and an appropriate field of view (FoV) (Bowman and McMahan, 2007).

2.3.1 Rendering

To ensure a realistic depiction of the environment, it is important that the user continually sees their environment without interruption. For a seamless experience a high enough frame rate is needed to ensure adequate immersion. For this to be viable, a rendering engine with sufficient power is essential. This ensures that the image changes swiftly in response to the users head movement, maintaining the illusion of real-time interaction (Regan and Pose, 1994).

VR systems may use stereoscopic rendering, to accurately simulate how the human eyes see. The simple reason behind this being, that by stereoscopically rendering each eye, we receive separately rendered images, thus creating the illusion of depth and 3D effect (Forlim et al., 2019).

2.3.2 Graphics

In contrast to older graphics cards that render 3D images using polygonal structures with shading, modern graphics cards like the NVIDIA RTX line, employ advanced techniques to simulate the behavior of light. These cards trace the path that light would naturally take from the human eye through the virtual environment. This capability enables them to generate realistic shadows and refractions, enhancing the visual fidelity of the rendered scenes (Wilkinson et al., 2021).

Another way to increase immersion through graphics is to ensure the user has a smooth experience by optimizing graphics and effectively use hardware resources. These strategies include optimizing the frame rate and resolution to fit the hardware and ensure a smooth experience. Another optimization includes implementing a level of information where objects further away are rendered with less detail, than closer objects, lowers resource use and rendering demands. Additional optimization techniques include using efficient shaders, and removing objects that are not seen by the user, which reduces the overall complexity of the surroundings without impacting the experience of the user (Tytarenko, 2023).

2.3.3 Field of View

While FoV might not seem like an important aspect of visual immersion, its incorrect implementation may lead to the users experiencing VR sickness. This of course leads to lessened immersion and an overall worse experience with VR. One of the ways found to mitigate VR sickness without influencing the persons experiences inside of VR is dynamically changing the FoV of the user. When the user is in a stationary position the users have regular, unrestricted FoV. The change occurs when the users are in motion because their FoV contracts drastically. This blocks their peripheral visual motion preventing a disconnect between what they see, and what they are doing (Teixeira and Palmisano, 2021).

2.4 Auditory Immersion

Sound is paramount in crafting an environment that exudes vibrancy and authenticity, ultimately contributing to a heightened sense of realism and immersion. There are multiple factors that contribute to the creation of an immersive environment such as spatial audio, binaural rendering and realistic sound effects.

2.4.1 Spatial Audio and Binaural Rendering

As the name suggests, spatial audio gives the user the remarkable sensation that sound emanates from distinct and specific locations within the 3D space, which helps create a dynamic and captivating environment that envelops the end user in a truly immersive experience (Schissler et al., 2016).

An important part of spatial audio is the shaping of the head-related transfer functions (HRTF), which describes how a listener and its geometry affects the sound that he is exposed to. It can be described as a filter which maps incoming sounds to the head of the user, and then delivers the necessary sounds to the left or right ear (Schissler et al., 2016). The audio also includes cues for the human ears to experience certain sounds from different directions, distance, which is created through binaural rendering (Zaunschirm et al., 2020).

2.4.2 Realistic Sound

The utilization of 3D sound technology, based on HRTF, faces a limitation in accurately reproducing realistic sound. That is due to its inability to reflect the physical impact on object materials and the surrounding environment within the virtual space. This limitation can be overcome through sound rendering. This is a technique which involves simulating the physical attributes of sound that occur between the sound source and the listener within the virtual space. Sound rendering consists of sound synthesis, sound propagation and sound generation (Hong et al., 2017):

- Sound synthesis is the generation of the sound when an event occurs for instance through user interaction.
- Sound propagation is the act of processing the characteristics of the environment into the sound. These may include absorption, reflection or transmission among others.
- Sound generation then takes all the characteristics processed by the sound propagation step and incorporating them into the original sound.

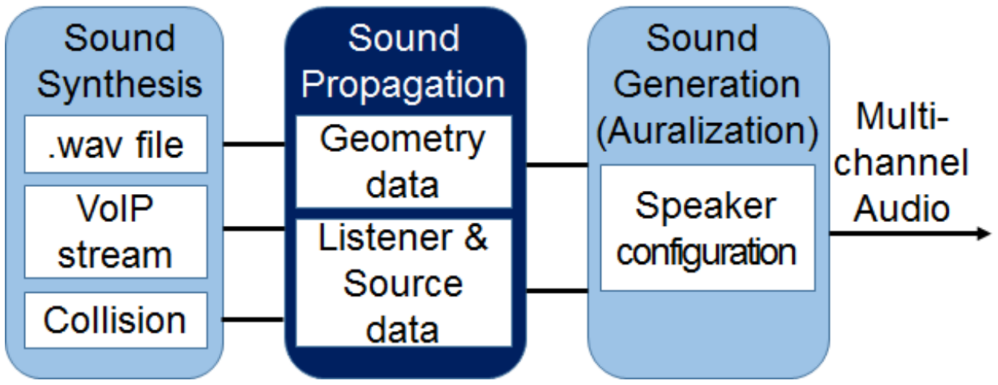


Figure 2.1: The pipeline used for rendering realistic sound (Hong et al., 2017).

2.5 Haptic Feedback

While visual and auditory immersion undeniably form significant pillars of the VR experience, achieving true immersion necessitates the development of precise and lifelike methods for interacting with objects within the computer-generated environment (Kim et al., 2017).

Haptic feedback can be categorized into passive haptic feedback and active haptic feedback. Passive haptic feedback is generally a physical object that is simultaneously generated in the VR, which simulates the physical properties of the object, such as it's weight, texture and shape (Viciana-Abad et al., 2010). Active haptic feedback on the other hand, is haptic feedback that can be actively programmed and felt through a haptic display such as vibrations and heat (Nilsson et al., 2018).

The majority, if not all, of commercially available VR systems currently offer limited or no haptic feedback. This limitation arises from the inherent challenge of accommodating the diverse physical attributes of individual users while maintaining a balance between wearability, lightweight design, and the inclusion of adequate hardware to deliver a convincingly realistic sense of touch (Perret and Vander Poorten, 2018).

2.6 Walking in Virtual Reality

Among the routine tasks we engage in on a daily basis, walking stands out as one of the most formidable challenges to authentically simulate in virtual reality, aiming to capture the essence of natural movement with a genuine sense of reality.

The primary hurdle in achieving a natural walking experience lies in the challenge of crafting expansive and immersive virtual worlds within the confines of our homes. The task at hand involves ingeniously navigating the limitations of physical space while delivering captivating environments that invite exploration and evoke a genuine sense of presence (Nilsson et al., 2018). Another concern may be, how to accurately simulate all the outside stimuli our bodies receive with each step, such as sight, sound and tactile feedback (Nilsson et al., 2018).

Virtual travel can be classified based on various factors, providing us with different categories to explore. Firstly, we can differentiate between mobile and stationary travel techniques, depending on whether the user physically moves or remains in a fixed position throughout the experience. In the realm of mundane travel techniques, users are bound by the limitations of physics, biology, and current technological constraints, while in the realm of magic travel techniques, such as using portals, teleporting, these constraints are transcended. Additionally, we can discern between vehicular and body-centric travel techniques. Body-centric travel techniques simulate movement by incorporating human motions such as walking or running, whereas vehicular techniques allow users to control a simulated vehicle without the need for physical movement (Nilsson et al., 2018).

Although simulating natural walking in VR poses challenges, there exists a multitude of solutions that can be classified into three distinct walking techniques.

2.6.1 Repositioning Systems

Repositioning systems are responsible for nullifying the movement done by the user, thus the user still moves virtually without having moved in reality. These systems can be classified into two distinct categorize active and passive repositioning systems (Nilsson et al., 2018).

Active repositioning systems are complex mechanisms that counteract any movement done by the users. Such systems are for instance linear treadmills. Though these offer the user to move forward turning and walking in any other direction must be done in a indirect matter. One way to mitigate this problem is to use omnidirectional treadmills which allow the user to walk in any desired direction. A limitation of these devices is that sudden movements, like side stepping, may cause the user to lose balance (Nilsson et al., 2018).

Passive repositioning systems are on the other hand mechanically simpler and often use friction-free floors, which prevent the steps of the user to take effect and move them. These systems are available commercially to the public, for instance KatVR's Kat Walk is one of such devices (Nilsson et al., 2018).

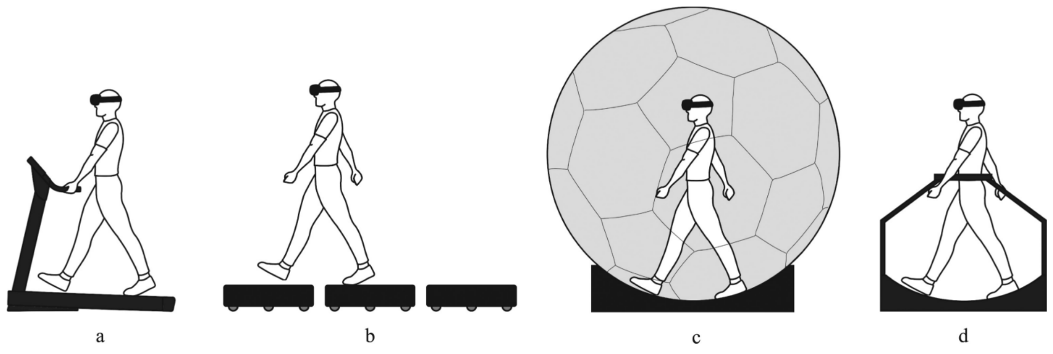


Figure 2.2: Examples of repositioning systems used for walking in VR: a) a traditional treadmill, b) motorized tiles, c) a enlarged hamster ball, d) friction less platform (Nilsson et al., 2018)

2.6.2 Proxy Gestures

Proxy gestures are actions which try to mimic the act of walking without actually moving the user from the spot they are in. These gestures can be categorized based on which part of the body the gesture is made with. Based on this we categorize them into upper and lower body gestures (Nilsson et al., 2018).

Due to the fact that we are trying to simulate walking, proxy gestures more often rely on lower body movement. The most common gesture is the walking-in-place gesture, which is comparable to marching in place. These gesture are easier and less expensive to track through motion tracking, or a physical device. They also provide feedback comparable to the act of walking, thus increasing the level of presence and helping users with spatial orientation (Nilsson et al., 2018).

Upper body gestures may seem to be less desirable to use, but these movements do not seem to have any impact on decreasing the users presence. Though they may hinder the interaction of the user with with their environment while walking. But, for some applications on systems that only track the head and arm movement they may seem as a more likely alternative (Nilsson et al., 2018).

2.6.3 Redirected Walking

This walking technique does include the user physically walking, while the environment around them warps in a way to direct them, without the user noticing. This manipulation may be done through altering the path of the user through objects or altering the users virtual point of view. This may include changes of the curvature, rotation and movement gain of the virtual camera. This is done, so that the user is under the illusion of traveling a large distance in a limited space. For this technique to be effective these manipulation need to be small enough for the user not to notice, since misuse may result in the disruption of the users experience, and may feel unnatural (Nilsson et al., 2018).

Even though most of the manipulation has been achieved through visual manipulation, redirection may be achieved through audio, or other senses, when the user has lowered visual stimuli (Nilsson et al., 2018).

2.7 Relevant Software

2.7.1 Unity

Unity is a real-time development platform, which can be used to create 2D or 3D projects, ranging from simple games to immersive VR experiences. Its popularity is thanks to its widespread availability on all major platforms, including Linux, Apple and Windows. Renowned for its ease of use, flexibility and its strong feature set, Unity caters not only to beginner, but also to experienced developers creating high-performance products (Hussain et al., 2020).

Unity, equipped with its rendering and physics engine and a graphical user interface, the Unity Editor, allows for rapid prototyping in development. Within each Unity project, various components, termed Assets, form the foundation. The main building block is an Asset called a Scene, which is the primary element of the project. It contains the hierarchy of GameObjects that constitute the scene's contents. All projects come with preexisting objects, such as Cameras, Meshes and others, but additional Gameobjects and behaviour can be easily integrated through scripts written in C# (Juliani et al., 2018).

2.7.2 Blender

Blender is a versatile and accessible 3D creation software, freely available and open-source. It provides support the whole creation process, tasks such as modeling modeling, rigging, animation, simulation, rendering, composition and motion tracking. One of its major features is its flexibility, allowing any user to use its API to create custom tools and functionalities for their specific need. These customized additions are often added to the software at a later release. Blender is also cross-platform compatible, making sure users on all major platforms may use its features (Blender-Foundation, nd).

2.7.3 Audacity

Audacity is a free, user-friendly, and open-source software designed for recording, editing, and modifying audio. Its open-source nature it allows user to study and modify its source code as well as install plugins created by other users making it a versatile and flexible software catered to the users individual needs. Audacity supports a wide range of audio formats, such as VAW, AIFF, MP2 and MP3. It is compatible with all major platforms: Windows, macOS, and Linux, providing accessibility for a wide range of users across varying operating systems (Washnik et al., 2023).

2.8 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning, where a task is gradually learned through a process of trial and error, while also trying to uncover such a sequence of actions, which yield the highest amount of cumulative reward. Unlike supervised learning, which relies on external labeled data, and unsupervised learning, focused on uncovering latent data structures, RL places its agent within an unknown environment, permitting it to explore various action choices autonomously (Sutton and Barto, 2018).

The agent initially perceives the state of his surroundings and can subsequently perform a number of actions which change it. This iterative process continues until the agents successfully preforms its given task, or reaches a state, where no additional actions can be taken. Throughout this process, each action is given a reward value, which is derived from the newly created state. It informs the agent, how impactful the given action was, to the completion of his end goal. This reward is calculated by the reward function, who through these calculations defines what the goal is for the agent, and also decides what states and or actions are objectively good or bad. Its objectiveness is protected, because the agent has no way of influencing or altering the outcome of the function (Sutton and Barto, 2018).

While the reward function shows the immediate benefit of an action the value function determines the positives of a decision in the long run. It promotes choosing an action with lower immediate rewards if it promises greater profitability in the future, over an action with higher immediate rewards but lower long-term yield. While the reward function lays the groundwork, the value function has a higher priority in the decision making, as it prioritizes maximizing overall value rather than immediate rewards (Sutton and Barto, 2018).

Subsequent actions are always chosen based on the policy the agent has. This function links actions to the possible result states that the given action will lead to. The policy chooses the next best action either through deterministic or stochastic means, always trying to find the sequence of actions that promises the highest cumulative reward over time. Basically, it defines how an agent behaves in any situation. A lot of algorithms keep refining their policy during the learning process. This practice is predominantly used to encourage exploration. That is because the agent is incentivised to favor routes with the highest established rewards amongst the ones he knows, which may block him from discovering superior alternatives. That is why most algorithms must always incorporate some trade-off between using what the agent knows, and options not yet explored. This prevents the agent from becoming trapped in a local maxima (Sutton and Barto, 2018).

2.9 Machine Learning Agents Toolkit

The Unity Machine Learning Agents Toolkit (Ml-Agents Toolkit) stands as an open-source project that enables developers and researchers to construct learning environments tailored for training intelligent agents. It also acts as a central platform where new algorithms and other advances in the AI field can be evaluated and made available for the wider audience of developers. These advancements can be used for research purposes or to be integrated into new game titles, providing developers with access to the latest innovations to elevate their projects (Juliani et al., 2018).

The Ml-Agents Toolkit is made up of four primary high level components, the Learning Environment, the Python Low-Level API, the External Communicator, and the Python Trainers.

The Learning Environment describes the Unity scene as well as all the objects inside the scene. This is where the agent observes its surroundings, performs actions, and is taught through reward signals. It's crucial to design the learning environment in a manner that is beneficial to learning a given task. If a task is very limited a more general learning environment may be enough. But for more complex tasks a more customized environment is needed. For the creation of custom environments, the Toolkit offers a specialized ML-Agents Unity SDK capable of converting any Unity scene into a learning environment (Juliani et al., 2018).

The Python Low-Level API serves as a Python interface, providing low-level functionality for modifying and interacting with the Learning Environment. Unlike the environment itself, this interface operates external, meaning it can communicate with the environment only through the External Communicator. Its main role is to communicate and manage the Academy class, although it can also be used to control Unity as a simulation engine for custom algorithms (Juliani et al., 2018).

The External Communicator serves as the bridge between the Learning Environment and the Python Low-Level API, making essential communication between these two vital parts possible. It is contained inside of the Learning Environment making it a part of it (Juliani et al., 2018).

The Python Trainers house all the algorithms used for the training of agents. As the name suggest all algorithms are implemented in Python, and this separate package solely communicates with the Low-Level API (Juliani et al., 2018).

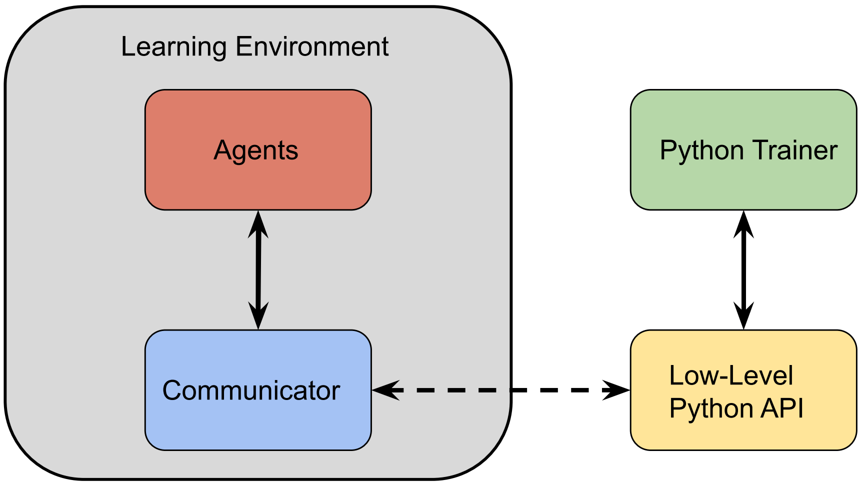


Figure 2.3: Abstract representation of the key components of the ML-Agents Toolkit (Juliani et al., 2018)

2.9.1 ML-Agents SDK

As previously stated, any Unity scene may be converted into a learning environment. For it to happen, we need to add three scripts to an existing project:

- Agent script, a Unity script which defines what GameObject is going to be the agent that is going to be trained (Metz, 2020).
- Behaviour Parameters script, which acts as the control center for the agent, which communicates the observations and the rewards between the Agent and the Training API, and also decides which actions to take (Metz, 2020).
- Decision Requester, which is responsible for the communication between the Agent and the Academy (Metz, 2020).

In an environment created through the ML-Agent SDK, the fundamental components are the Sensors, Agents and the Academy. The Sensors provide observations of the environment to the Agents. These Sensors may relay different kinds of information, based on the type of sensor, for instance they relay rendered images or length vectors. Each Agent is a designated GameObject, with each containing a policy labeled with a *behaviour name*. Multiple Agents may share the same policy with identical *behaviour name* values. All of the Agents will act upon the same policy and all of them will contribute data to the refining of the policy. Any number of policies may be added to a scene, making it easy to create multi-agent environments, where each agent may act upon a distinct policy (Juliani et al., 2018).

The reward function of an agent may be modified dynamically during the simulation at any point by the Unity scripting system. Similarly, the state of the agent, or the entire environment can transition into a state of “done”. This change occurs either when the agents have reached their objective, defined by a call from a Unity script, or when they have taken the predefined maximum number of steps. This state signals the end of an episode of the simulation (Juliani et al., 2018).

The Academy refers to a singleton class that oversees the entire simulation, maintaining control over its progression. It tracks the amount of steps taken within the simulation, and is responsible for the management of the agent. The Academy has the capability to shape the learning environment of the agent by introducing new objects or altering physical properties when the agent reaches a level of proficiency. This is done for the goal of further development of the Agent. Also it can be used to reuse the same environment for both training and testing purposes with minimal modifications (Juliani et al., 2018).

The Agents can undergo training using state-of-the-art algorithms drawn from a diverse array of machine learning methods, including reinforcement learning, imitation learning, neuroevolution and more. However, for the scope of this thesis, we will only be focusing on the reinforcement learning methods. As such we will discuss the two algorithms this toolkit provides, which are Proximal Policy Optimization and Soft Actor Critic (Juliani et al., 2018).

2.9.2 Proximal Policy Optimization

The Proximal Policy Optimization (PPO) is a RL algorithm, which focuses on stabilizing the training of the policy. It achieves this by limiting the magnitude the policy may change at the end of each epoch, preventing excessively large updates that could disrupt the training stability (Simonini, 2022; Schulman et al., 2017).

A more stable training of the policy is preferable, because smaller policy updates tend to converge more reliably towards the optimal solution. Another reason to employ this algorithm is, mitigating the risk of large policy change potentially making the policy ineffective, prolonging training time, or failing to recover altogether (Simonini, 2022; Schulman et al., 2017).

To ensure the policy hasn't undergone excessive changes, we need to calculate the ratio of how much the current policy has changed relative to the alteration made to the former one. This is accomplished by determining the ratio between the current and former policy. We adjust this ratio so that it falls within the range of $[1-\epsilon, 1+\epsilon]$, which prevents excessive alteration of the current policy when compared to the previous one (Simonini, 2022; Schulman et al., 2017).

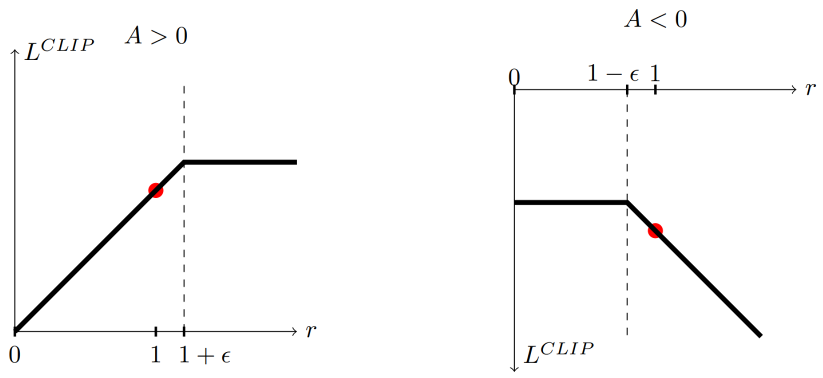


Figure 2.4: Two functions depicting one time step of the function L^{CLIP}, when the ratio function $r = 1$. The left one shows when the advantage is positive, while the right shows when the advantage is negative (Schulman et al., 2017).

The PPO algorithm is carried out through the function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{2.1}$$

where $r_t(\theta)$ is the ratio function:

$$r_\theta = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)} \tag{2.2}$$

In Eq. 2.2, $\pi_\theta(a_t|s_t)$ denotes the probability of taking action a_t in the s_t state. If the value of $r_t(\theta) > 1$, then we know that this action in the current state is more likely to happen in the new policy than the old one. And if $0 < r_t(\theta) < 1$, then it is more likely for this action/state pair to happen in the older policy rather than the new one. This is the easiest way to asses the difference between the newer and the older policy.

The main term in Eq. 2.1 is the minimum of two probability ratios, the clipped and unclipped one. That is how we make sure the policy only changes inside the range of $[1 - \epsilon, 1 + \epsilon]$, and not more drastically (Simonini, 2022; Schulman et al., 2017).

2.9.3 Soft Actor Critic

An Actor Critic is an RL architecture characterized by the fact that it simultaneously trains both a policy (actor) and an estimated value function (critic) at the same time. In this architecture, the actor chooses an action to take based on the policy. This action is then given feedback by the critic, regarding its long-term quality. Based on this evaluation the policy is updated by the actor (Ezzeddine et al., 2023).

One of the advantages of AC algorithms is their ability to leverage the strengths of both policy-based and value-based RL methods. This integration allows the algorithm to converge on an optimal solution more rapidly and learn more efficiently. Additionally, these algorithms have a higher proficiency in navigating complex and high dimensional action spaces, rendering them well-suited for complex input representation (Ezzeddine et al., 2023).

Despite their numerous advantages, these algorithms also present limitations. One such drawback is the need for a substantial amount of samples for the value function to create a more accurate estimate. Another negative is the need to regulate the entropy of the policies; which if not properly managed may lead the model to converge to a deterministic policy, which results in less exploration and potentially yielding sub-optimal result (Ezzeddine et al., 2023).

To resolve such issues a solution was introduced in the form of the Soft Actor Critic (SAC) architecture. SAC not only tries to maximize the expected reward value, but also to maximize the entropy of the action taken. This is to ensure that there always remains a certain level of uncertainty regarding which action will be chosen. This ensures that there remains a balance between exploration and exploitation, making sure the policy doesn't remain sub-optimal (Ezzeddine et al., 2023).

The SAC algorithm is made of three different networks: a state value function, a soft Q-function and a policy function. The estimation provided by the state value and the soft Q-function contributes to the convergence of the algorithm (Ezzeddine et al., 2023).

Chapter 3

Implementation

This thesis is built upon an existing Unity 2.7.1 project[1] containing a simulated model of the semi-humanoid robot NICO in Unity. The robot is constructed using multiple GameObjects. Each distinct GameObject has its own Mesh and ArticulationBody, and adds an additional Degree of Freedom (DoF) to the movement. The Unity project contains a script, that allows an RL agent to be trained to manipulate the right arm of NICO. Additionally, it contains three scenes: two where the RL agent operates with varying DoF, and one where the body parts may be manipulated freely using sliders.

3.1 Motivation and Objectives

As we previously mentioned in chapter 2.2, to improve immersion we have to increase the fidelity of available sensory feedback, or we increase the amount of senses that are getting feedback. We chose to build upon existing simulation of the robot NICO, aiming to enhance and incorporate features to it that are present on the physical version of the same robot. That is because we can not only test different versions of the simulated variants, but we have a physical form of the robot to compare it to.

That is why, the main aim of this thesis was to create a script, that will help in future studies of immersion in VR. The main focus was to distinguish which changes have a higher impact on enhancing an individual's immersive experience and which alterations may have the opposite effect and diminish it. If this is studied further, we may be able to grant insight into what makes a digital world immersive and engaging.

To develop a script where immersion may be studied we have to enhance the model of NICO. For these enhancements to be noticeable, we have to make NICO perform an activity where we may detect these additional features. The activity to be performed by NICO was chosen to be it pointing at a specific object that appears on a table situated before him. The enhancements themselves are categorized based on which senses they

[1]This project can be found at: https://github.com/iveta331/NICO

try to enhance. In this thesis, we have implemented improvements targeting visual and auditory senses of the user.

3.2 Visual Improvement

Since visual immersion plays an important role in the overall VR experience, our initial focus was set on elevating the visual aspects of the existing project. The original project did include a basic model of the robot NICO as well as the ability to control certain parts of the body. Specifically, the head, neck and right arm could be manually manipulated in the scene without any RL agents. In the scenes where RL agents were implemented, only the right arm was moved through the agent.

We decided to first enhance the model of NICO, and after that to implement the code, which made its movements seem more real.

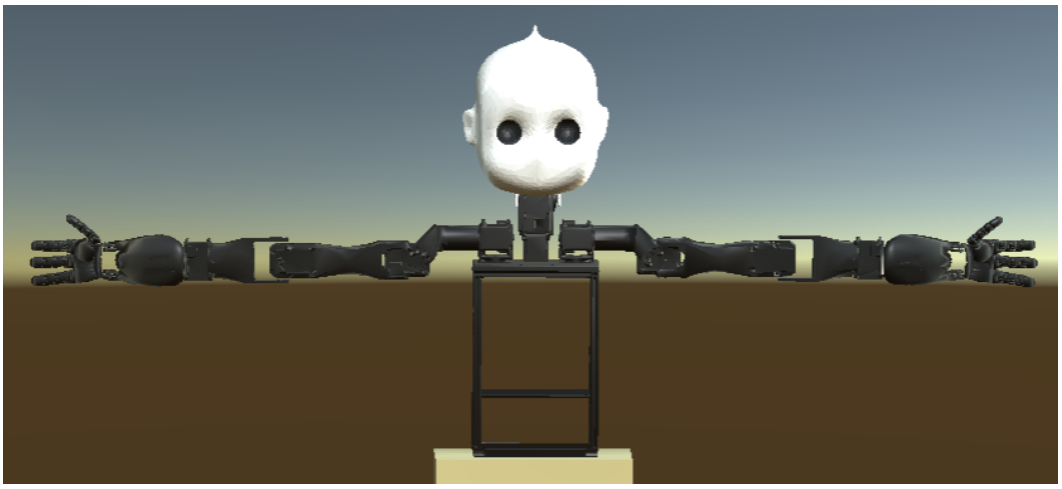
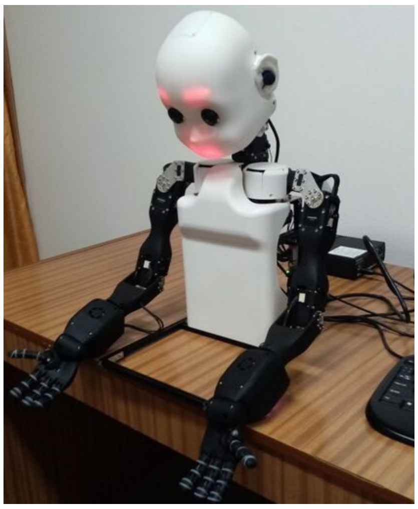


Figure 3.1: The model of robot NICO without any visual improvements inside of the Unity editor.

3.2.1 Improvement of the Robotic Model

As mentioned, the project came with a basic model of NICO, as seen in fig. 3.1, consisting primarily of parts that are either needed for the robot's structural integrity or those which may be manipulated with. However, the real NICO, as seen in fig. 3.2a, features additional aesthetic components, such as covers, which improve its appearance rather than functionality. To increase immersion, we decided to include at least the most prominent of such components. This cover masks the robot's empty torso, since it covers it from the front, and is the most noticeable visual difference between the real and virtual NICO.


(a)

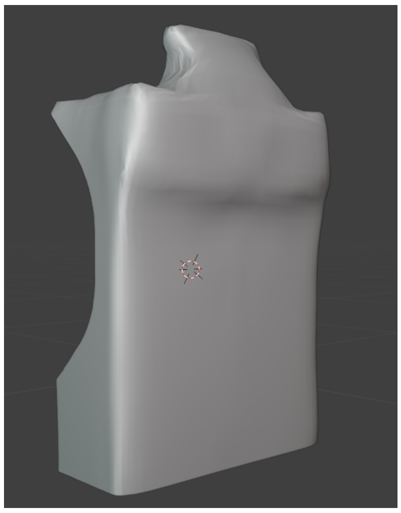

(b)

Figure 3.2: a) the physical NICO, which served as a guide for the modeling. b) the front cover created for the enhancement of the visual immersion made in Blender.

We created the front cover using the 3D modeling software Blender (see in sec. 2.7.2). To guide the modeling process, we began by capturing pictures of the robot's front and side profiles. The initial modeling was based on the front profile, followed by modifying and refining the model using the side profile. During this process, we ensured that the mesh was made up of only triangles and quadrilateral, to comply with the best practices and making sure no polygons where present in the mesh, as these are considered bad practice in 3D modeling. After the modeling was finished it was exported as an .fbx file format and added to the Unity editor, where it was positioned on the preexisting model as seen in fig. 3.3.

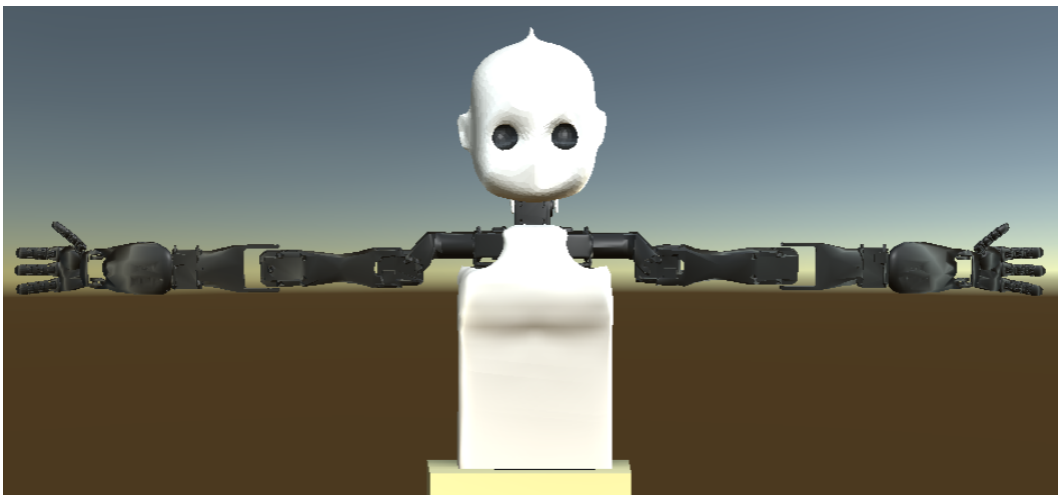


Figure 3.3: The model of robot NICO with the added front cover inside the Unity editor.

3.2.2 Improvement of Movement

While the outer look of the objects in VR play a crucial role in creating an immersive environment, their unnatural behaviour and movement may be able to break the illusion and negate the aesthetic of the whole experience. For this reason we decided to improve the movement of the robot NICO.

All the improvements to the movement were made to ensure that the action of pointing appears more natural and believable. The first change made, was the positioning of the left arm and and the fingers of the right arm, which doesn't change through the duration of the performed action.

The left arm has been positioned to be parallel instead of perpendicular to the body. The fingers on the right arm have been arranged as to make a pointing gesture, where the index finger extended and the other fingers curled inwards.

There was an issue we encountered was with the fingers closing. The speed at which the motion happened caused the fingers to started moving in a matter resembling uncontrollable flailing, which did not subside until we restarted the program. Each finger is made up of three joints, while the thumb consists of four, and the act of closing the fingers may have caused two joints to overlap. This overlap should not be happening, due to the collision boxes of the joints. The flailing may be the result of the unsuccessful attempt at correcting this.

After some experimentation with the digits, we discovered that the problem didn't occur within a threshold. This solution was favourable for the thumb, but for the other digits another solution had to be found, as it caused the hand to stop resembling the intended gesture. Therefore, we removed the collision boxes of the middle joint of the two remaining fingers. This adjustment allowed for them to fully close without the unintended flailing, making the intended gesture.

The next improvement we made, was making the head of NICO move to appear as if it is looking at the object it is pointing to. The movements of the head are controlled by two joints: the "neck" joint controlling the heads horizontal movement, and the "head" joint, which manages the heads vertical movement.

To calculate the necessary movements of the two joints for NICO's head to "look" at the target, we have to first designate what the target is, and what we can designate as the head. By defining both the target and head, we can extract their X, Y and Z coordinates. Using these lets us calculate, through the simple trigonometric function of hyperbolic tangent, what the required angles are for the "neck" and "head" joints.

After accurately calculating the required angle, we still must determine where the target is relative to the head both horizontally and vertically. This step is crucial because, for instance, if we calculate that the head needs to move 30 degrees horizontally, we still need to ascertain whether to move it 30 degrees left or right.

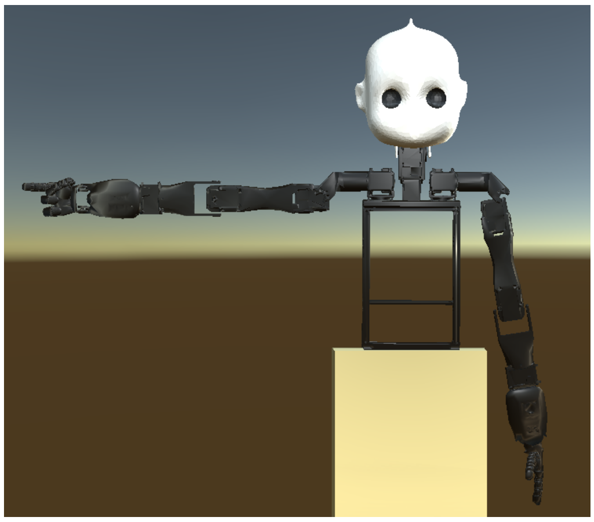


Figure 3.4: The model of robot NICO with its left arm and fingers on the right arm adjusted.

The final improvement done to the movement of the robot NICO involved training an agent, through the ML-Agents Toolkit (see sec. 2.9), to control the right arm to point at the target. We opted to train an agent for this task because, creating a script for the arms movement, though possible, would prove to be a challenge, that may result in the final action being choppy and unrefined. This could be attributed to the arms complicated composition, being made up of six joint, if you exclude the joints that make up the fingers.

As discussed in 2.9.1, any existing Unity project may be turned into one with a teachable agent or agents, by adding three scripts. The existing project did contain all three necessary scripts, but for the applications that we needed, some required to be modified or rewritten from scratch.

The Decision Requester script comes in-built into the ML-Agent Toolkit. We only had to decide the frequency of communication between the agent and the Academy, which is a singleton class (see sec. 2.9.1), which we set to four.

The Behaviour Parameters is another script that is prepared for use in any project. Here we modified the agent control center with the *behaviour name* NICO. Since we wanted our agent to only move six joints that make up NICO's right arm, we changed the size of the Vector Action array to match the amount of joints, so for each joint an action is chosen. We also changed the Vector Observation, to match the amount of observations made by the agent. These include the current position of all joints moved by the agent, and the distance of the end-effector from the target. The last changes made were only after we had trained the agent, and that was, we set the model of the agent to the trained neural network and changed the Behaviour Type to Interference, which meant that all decisions were made by the neural network we trained, and chose to use.

The agent script had to be rewritten from scratch to fit our specific requirements of our project. The existing project operated on the principle, that every part of NICO, which had the physics component ArticulationBody, which was responsible for making the component movable, was automatically included in the agent's control and moved according to its neural network. One exception to this, was the possibility to toggle the exclusion of the fingers from the agent's control. But even with this exception, this script was not usable because of the previous improvements we made. We needed certain joints of the robot to have an ArticulationBody component while also being controlled by our code. This led to a clash between our code and the agent's attempt to try move the same joints.

To address this problem, we decided to create a function called GetParts, which separates the components under the agent's control from the ones we don't want the agent to influence. For this we utilized Unity's Tag system, tagging all the parts we already use in other functions, while leaving all the joints under the agent's control untagged. We then created a list of all the tags corresponding to the joints we didn't want the agent to control and based on this filtered out all of them in the function. This solution is practical because it allows for easy adjustments: if we wish to add another tag, we simply add it to the existing list, and if we wish to grant the agent control over additional body parts, we simply have to remove the corresponding tag from the list.

After constraining the agent's control over chosen body-parts, we had to rework other functions vital for the agent's correct operation. These functions had the issue of either changing all the joints with the ArticulationBody component, or contained extensive code for the purpose of controlling the finger joints, both of which were incompatible with the functionality we required. Thus we changed that functions such as CollectObservation, which collects observations from the environment for the agent, and OnActionReceived, which when receives actions from the agent changes the positions of the joints and calculates the reward for the given action. These changes ensured that only joints under the agents control would be affected by this script and there be no clashes with other written instructions.

After setting up the Unity project with all the necessary scripts, we were not sure which of the two RL algorithms to use, either PPO or SAC, so we decided to train two separate neural networks. To train a neural network with the ML-Agents Toolkit, we needed to first create a conda environment, where we could run the command:

```
mlagents-learn <trainer-config-file> --env=<env_name> --run-id=<run-identifier>
```

`--run-id=<run-identifier>` parameter is a unique identifier which differentiates between different training runs.

`--env=<env_name>` is an optional parameter, which contains the path to the Unity project. If not inputted the training will start only after pressing the play button in the Unity editor.

`<trainer-config-file>` is the path to a .yaml path which contains all the hyper parameters needed for the training. If left blank a basic configuration file is used. Since we wanted to train based on two different algorithms, we created two different configuration files. Each file was created based upon the recommendation of the ML-Agents Training Configuration File documentation norms.

```
behaviors:
  NICO:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size: 50000
      learning_rate: 0.00001
      beta: 0.001
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 10
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 2000000
    time_horizon: 1000
    summary_freq: 10000
```

Figure 3.5: PPO configuration file used for training an agent for 2 000 000 steps.

The training consisted of NICO trying to point at a red cube object, which changed position after every five-thousand steps made. We employed a reward function present in the original project which comprised of three elements:

- For every movement made by a joint NICO was penalized to discourage any additional movement to make the movement more smooth and fluid.
- If NICO got closer to the target compared to its previous position a reward is given, if it got further away, it was penalized.
- NICO was penalized for the distance from his target, to incetivize getting closer to the target.

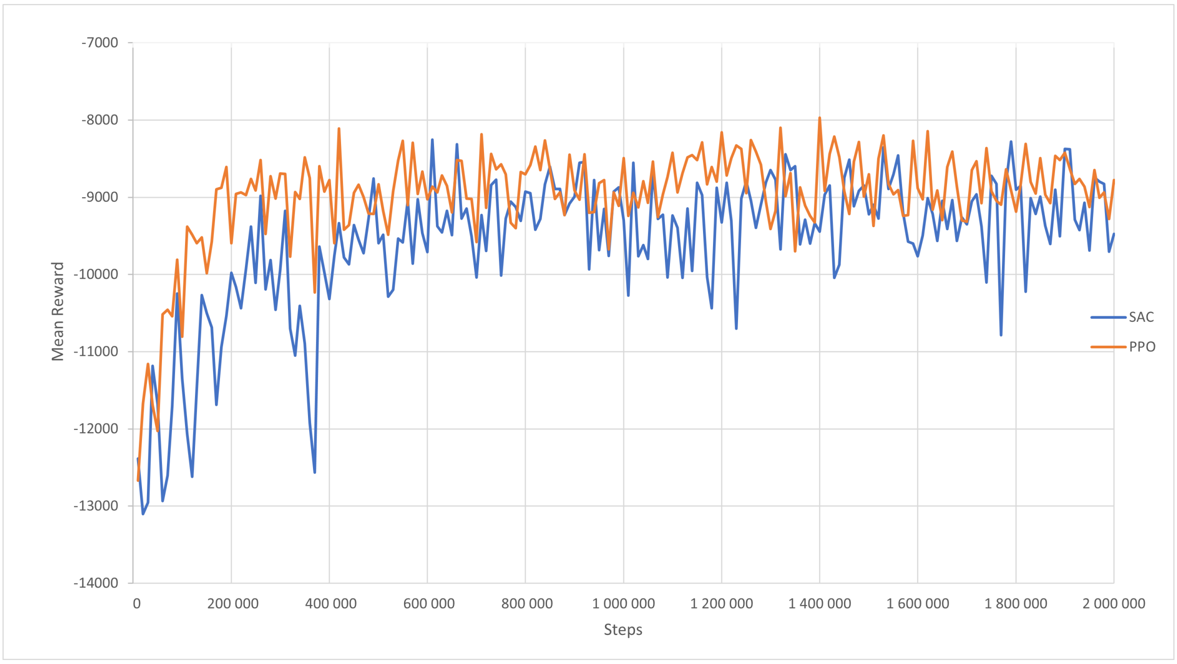


Figure 3.6: Cumulative Reward throughout the training of both PPO and SAC algorithms.

We finished training after two-million steps, due to the reward increasing only marginally close to hitting a plateau. The results showed that the neural network taught with the PPO algorithm was more successful than the one trained with the SAC algorithm.

After we reviewed the finished neural network NICO was successful at pointing an object farther away, but for object near it, the result was that NICO approached the red cube from the side and tried to force its finger inside the box instead of pointing at it.

This made us consider updating the reward function. We decided to add another element:

- We penalize NICO for the angle between two vectors. The first one starts from the right palm of NICO and ends at the end of its index finger, which is the end effector. The second one also starts at its right palm, but ends at the target, which is the red cube. This encourages NICO to minimize the angle, thus trying to put all three points on one line, thus making the pointing gesture even if the item is close instead of trying to force its finger into the object.

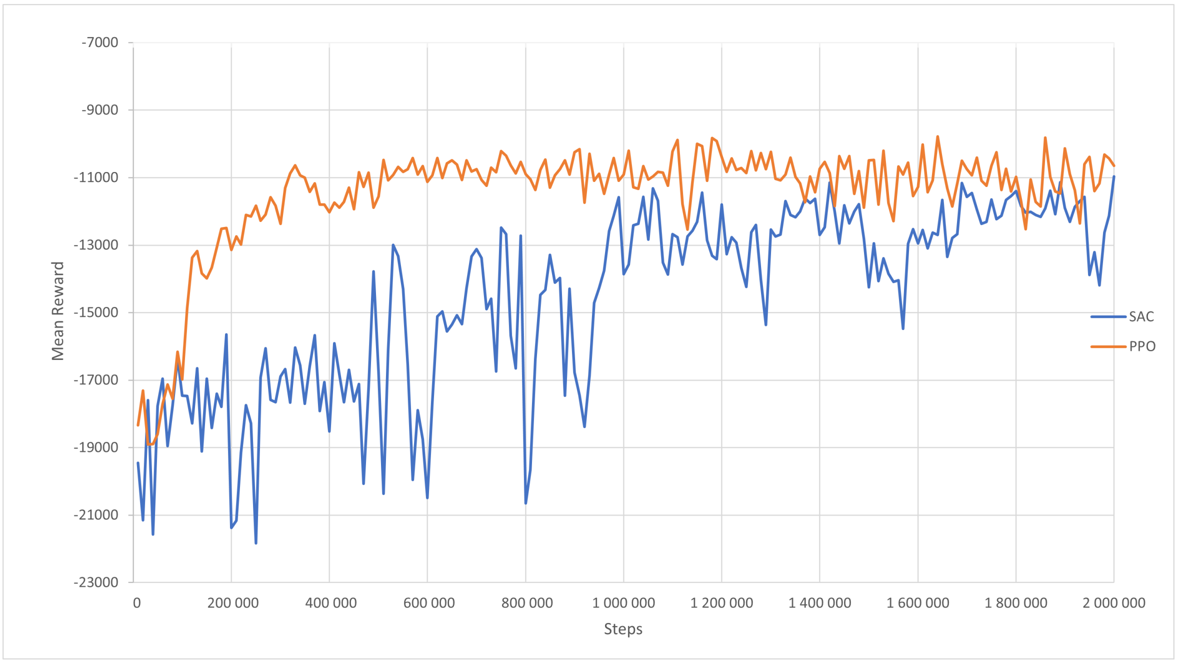


Figure 3.7: Cumulative Reward throughout the training of both PPO and SAC algorithms.

The neural network trained with the PPO algorithm did show improvement compared to the previous neural networks. The one taught with the SAC performed worse with the updated reward function, the arm performed more movement while staying off target more.

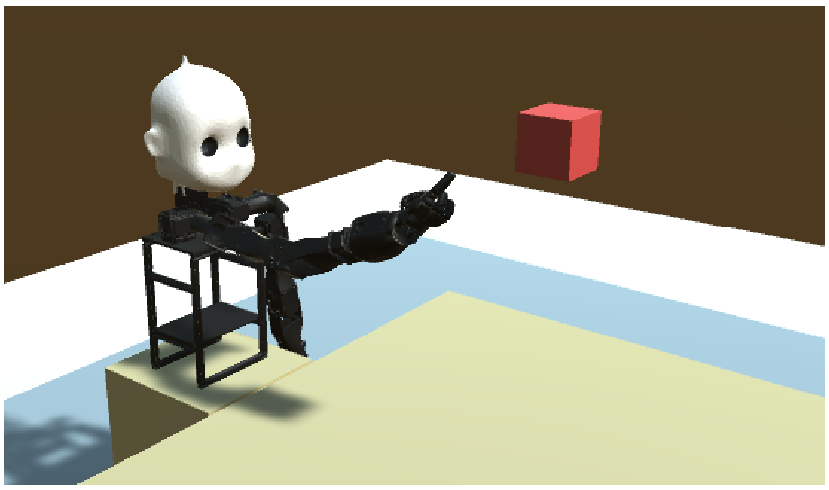


Figure 3.8: NICO pointing at the target red cube using a trained neural network.

Overall the neural networks taught with the PPO algorithm performed better than the ones trained with the SAC algorithm.

3.3 Auditory Improvement

Increasing immersion can be achieved not only by enhancing existing senses that are already implemented, but also with the addition of a new sense. Besides sight, the next most commonly implemented sense into VR projects is audio. In our case this involves including the sounds made by NICO while it is operational.

The first step was to identify the sounds NICO produces. We categorized these sounds into two groups: sounds made while NICO is idle and motionless, and sounds made while NICO is in motion. Passive sounds are produced by the coolers attached to NICO, located in its trunk near the neck. Active sounds are generated by the servos in its joints as they bend and move.

The passive sounds generated by NICO were consistent, meaning only one sound effect was needed. Since the sound made by each joint was indistinguishable from others, one sound effect would be sufficient for the active sound effects as well.

The next step was recording the audio from which we could create the needed sound effects for the project. The recording consisted of first recording while NICO was idle, and then of it moving its arm. While the recording of his passive sounds was straightforward, capturing the sound of its servos during movement proved challenging. The servos were producing sound only while moving rapidly. The first recording only managed to capture these sound in really small bursts which turned out later to be unusable. Only in the next recording session did we capture usable samples. These while better than the first batch of audio files, still didn't contain a sound, which could be used on its own. The problem was that we required a sound which could be used for various lengths of time, since we could not predetermine how long a certain movement would take.

To create the sounds that will be used in the our project, we needed to alter the samples, so they could be played on repeat. This was done so that it didn't matter how long the simulation took, the sound could be played thru-out. For this we used the open-source software Audacity (see sec. 2.7.3).

The passive background sound effect was relatively easily sampled. We took the sampling, cut it, and altered it to create a smaller fragment that had a smooth transition between the end and beginning. This way it could be played for any length of time. The final sound is two seconds long, but before its addition to the project it was listened to on repeat to determine if it really was applicable.

Creating the sound effect for the active movement of NICO posed a bigger challenge.

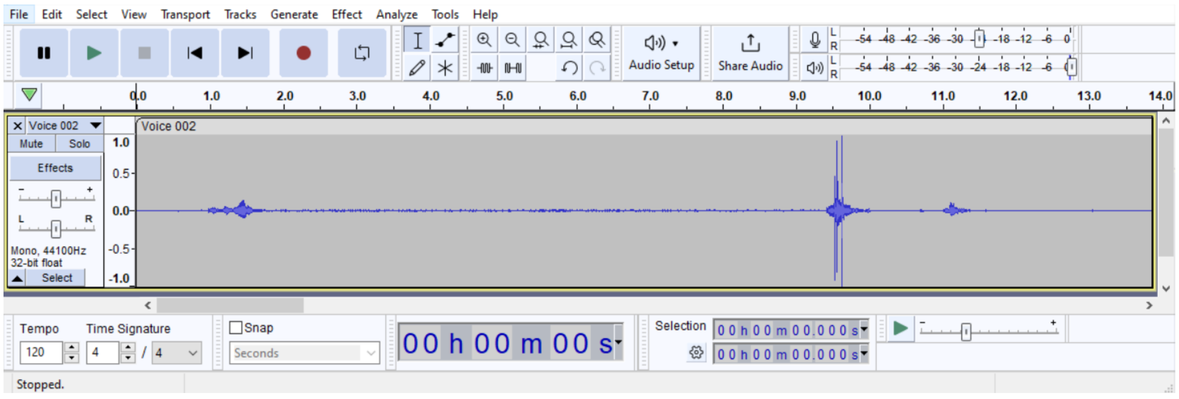


Figure 3.9: Audacity software layout with the final recording, which was used to create all sound effects.

As mentioned earlier, the recording did not contain a sample, which could be made into a looping sound effect. Each movement could be cut into a louder initial sound, a monotonous middle and a progressively quieting sound. Since we needed something that could be used on repeat, the only useful piece of the sound was the middle. These middle parts, while being the best piece, alone they were too short to be played on repeat, since instead of sounding like a motor, they were more similar to a washing machine. The solution we found was to use multiple samples of movement, cut out its more monotonous middle parts, and edit them into one longer sample. With this approach we got a second long sound effect, which could be added to the final project.

Finally we imported the sounds into the project. For the simulation to create sound we had to add an Audio Source component to a part of NICO. The passive sound had its source put into the trunk of NICO because, that is the approximate place where the sound is coming from on the real robot. For all the joint movements we decided it would be better to have a central source, which acts as an output for the whole right arm, instead of having each individual joint outputting sound. This would make sure that the sound is not disruptive and unintelligible. As a result the Audio Source was placed on the right upper arm, which is the approximate center of the whole arm. This means that if any part of the arm moves, the sound effect is played only from the upper arm, no matter which joint is moving.

3.4 Scenario Setup

To facilitate the use of the improvements made in this project for future studies of immersion in VR, we developed a C# script that centralized all the features and streamlines the process of creating different setups of the robot NICO.

The Unity project operates on the basis, that NICO starts performing the action of pointing for a number of steps defined by the agent script, which is refereed to as an episode. At the end of the episode the agent scripts function OnEpisodeBegin is called, which resets NICO and starts the next episode.

Our script utilizes this episodic nature of the project and makes it easier to have different versions of the robot NICO in succession without the need to interrupt the simulation and redefine the parameters.

This script organizes the improvements as distinct functions. Each function is paired with a corresponding boolean, which is initialized at the beginning of every episode. Each frame, through the Update function, depending on the booleans, the respective function is called or bypassed. This structure allows for a highly customizable approach to the needs of the immersive VR study.

To achieve this level of customization we decided to use a list structure, where the user inputs either 0 or a 1. This list is used at the beginning of each episode, when the agent script calls the function SetEnvironment. This function, depending on how many improvements can be toggled on or off, takes the next n values from the list, and based on this sets the corresponding boolean to true or false. If we have reached the end of the list, the next episode will have the same values as the current one.

In our implementation, this entails that the function consistently processes the next three values based on the integer value of "command_size". These customization options include: the front cover of the model, the movement of the head, and the sound effects.

```
SetEnvironment(){
  for each enviromental variable{
    if enviromental variable for this environment is set to 0{
      set enviromental variable to false;
    }
    else{
       set enviromental variable to true;
    }
  }
  reset model to default settings;
  move to next environment;
}

OnEpisodeBegin(){
    SetEnvironment()
    move target to a random location in the designated area;
    reset the parts the agent controls to default position;
}
```

This method is designed with future development in mind, making sure that adding new functionalities is straightforward. To integrate a new functionality, we first need to implement the new improvement along with a corresponding boolean value, increase the "command_size" value, and incorporate the call of the new functionality into the Update function. This kind of approach not only streamline the development process but also strengthens the modularity and scalability of the system, making the continues improvement and expansion of NICO's capabilities possible.

Chapter 4

Conclusion

The objective of this thesis was to study the concept of immersion, implement features which would raise the immersion of an existing Unity project involving the semi-humanoid robot NICO, and finally to develop a script which makes it possible to further study the effects of these improvements and their effect on the project's immersive experience.

We began by researched the concept of immersion, exploring methods to improve immersion inside a VR simulation, and examining the different types of immersion based on the senses they aim to engage. Based on this we decided what senses we wanted to focus on and started to study different software, which we would use in the development, such as the Unity editor for the VR simulation, Blender and Audacity for the creation of different improvements. Finally we chose concrete libraries and algorithms to use for the betterment of an already existing project upon which we built.

After we finished researching the theoretical part of this thesis we moved unto implementing functionalities, which would increase the simulations immersion. First we focused on improving the project from a visual standpoint with the creation of additional assets that where present on the real world counterpart, but not on the model inside the simulation. We then moved unto improving the movement of NICO while performing the action of pointing at a target by including other body parts into the movement and creating a neural networks which controlled its arm during the action of pointing. The last improvement was the inclusion another sense, sound into the project, by sampling the real counterpart and editing the sample to fit our needs.

Finally we developed a script which would ease the use of our functionalities in future studies of immersion, by making it easier to create different version of the robot in quick succession without the need to stop the experiment and changing parameters.

Even though we did successfully enhanced a project with new features and functionalities for future studies, due to technical and time constraints some aspects of it may be enhanced and improved upon in the future.

These include experimenting with the hyper-parameters of the training configuration file and letting the training run for a larger amount of steps to better the final network. Another aspect to improve upon is using better sound recording equipment and the editing the final sample, making sure that the sound effects do not sound periodic in the final product. Finally to model and include all the parts of the physical NICO, that are not yet included in our project, such as the shoulder covers.

Even though the project has some minor shortcomings, we believe we created an enhanced version of the NICO project in Unity which could be used to study what improvements have the biggest impact on the users immersive experience as to know which areas should be expanded upon in future VR applications.

Bibliography

Blender-Foundation (n.d.). Blender foundation - blender.org. `https://www.blender.org`. Accessed: 6.5.2024.

Bowman, D. A. and McMahan, R. P. (2007). Virtual reality: how much immersion is enough? *Computer*, 40(7):36–43.

Cavusoglu, H., Dennis, A. R., and Parsons, J. (2019). Immersive systems. *Journal of Management Information Systems*, 36(3):680–682.

Ezzeddine, F., Ayoub, O., Andreoletti, D., and Giordano, S. (2023). SAC-FACT: Soft actor-critic reinforcement learning for counterfactual explanations. In *World Conference on Explainable Artificial Intelligence*, pages 195–216. Springer.

Forlim, C. G., Bittner, L., Mostajeran, F., Steinicke, F., Gallinat, J., and Kühn, S. (2019). Stereoscopic rendering via goggles elicits higher functional connectivity during virtual reality gaming. *Frontiers in Human Neuroscience*, 13:365.

Halbig, A., Babu, S. K., Gatter, S., Latoschik, M. E., Brukamp, K., and von Mammen, S. (2022). Opportunities and challenges of virtual reality in healthcare–a domain experts inquiry. *Frontiers in Virtual Reality*, 3:14.

Hecht, D., Reiner, M., and Karni, A. (2008). Enhancement of response times to bi-and tri-modal sensory stimuli during active movements. *Experimental Brain Research*, 185:655–665.

Hong, D., Lee, T.-H., Joo, Y., and Park, W.-C. (2017). Real-time sound propagation hardware accelerator for immersive virtual reality 3D audio. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–2.

Hussain, A., Shakeel, H., Hussain, F., Uddin, N., and Ghouri, T. L. (2020). Unity game development engine: A technical survey. *University Sindh Journal Information and Communication Technology*, 4(2):73–81.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kim, M., Jeon, C., and Kim, J. (2017). A study on immersion and presence of a portable hand haptic system for immersive virtual reality. *Sensors*, 17(5):1141.

Metz, L. A. E. P. (2020). *An evaluation of unity ML-Agents toolkit for learning boss strategies*. PhD thesis, Rekjavík University.

Nilsson, N. C., Serafin, S., Steinicke, F., and Nordahl, R. (2018). Natural walking in virtual reality: A review. *Computers in Entertainment (CIE)*, 16(2):1–22.

Perret, J. and Vander Poorten, E. (2018). Touching virtual reality: a review of haptic gloves. In *ACTUATOR 2018; 16th International Conference on New Actuators*, pages 1–5. VDE.

Regan, M. and Pose, R. (1994). Priority rendering with a virtual reality address recalculation pipeline. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 155–162.

Schissler, C., Nicholls, A., and Mehra, R. (2016). Efficient HRTF-based spatial audio for area and volumetric sources. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1356–1366.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Simonini, T. (2022). Hugging face. `https://huggingface.co/blog/deep-rl-ppo`. Accessed: 2.5.2024.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Tanaka, N. and Takagi, H. (2004). Virtual reality environment design of managing both presence and virtual reality sickness. *Journal of Physiological Anthropology and Applied Human Science*, 23(6):313–317.

Teixeira, J. and Palmisano, S. (2021). Effects of dynamic field-of-view restriction on cybersickness and presence in HMD-based virtual reality. *Virtual Reality*, 25(2):433–445.

Tham, J., Duin, A. H., Gee, L., Ernst, N., Abdelqader, B., and McGrath, M. (2018). Understanding virtual reality: Presence, embodiment, and professional practice. *IEEE Transactions on Professional Communication*, 61(2):178–195.

Tytarenko, M. (2023). Optimizing immersion: Analyzing graphics and performance considerations in unity3d vr development. *Asian Journal of Research in Computer Science*, 16(4):104–114.

Viciana-Abad, R., Lecuona, A. R., and Poyade, M. (2010). The influence of passive haptic feedback and difference interaction metaphors on presence and task performance. *Presence*, 19(3):197–212.

Washnik, N., Suresh, C., and Lee, C.-Y. (2023). Using audacity software to enhance teaching and learning of hearing science course: A tutorial. *Teaching and Learning in Communication Sciences & Disorders*, 7(3):4.

Wilkinson, M., Brantley, S., and Feng, J. (2021). A mini review of presence and immersion in virtual reality. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 65, pages 1099–1103. SAGE Publications Sage CA: Los Angeles, CA.

Zaunschirm, M., Frank, M., and Zotter, F. (2020). Binaural rendering with measured room responses: First-order ambisonic microphone vs. dummy head. *Applied Sciences*, 10(5):1631.

Appendix A

User manual

To create episodes with different settings, the user needs to write 0/1 into the “environments” list, making sure that they are written in groups of three for each improvement, to make up one episode.

To change the neural network used in Unity, take these steps:

1. Open the Unity project
2. Double-click on the nico GameObject (in the left panel)
3. In the inspector tab open the Behavioural Parameters script
4. Change the Model parameter

The required version for Unity is 2022.3.4f1, other versions may not be compatible with this project.

Apendix B

Electronic Attachment

The electronic attachment contains the folder NICO_articulation_body, which is a Unity project including all the necessary assets, scenes, scripts and neural network models. These can be found in the Assets folder, where scripts and scenes have their own folders, and neural networks are the .onnx files.