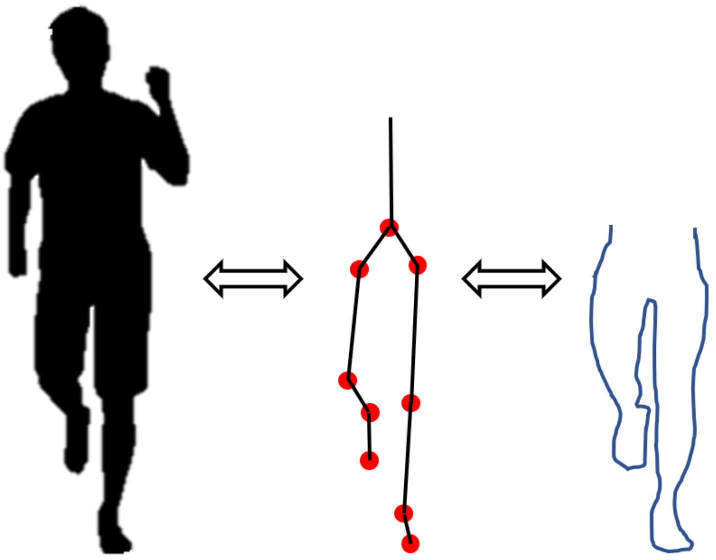
BACHELOR THESIS - BIOMEDICAL TECHNOLOGY

Quality of posture of a humanoid character in a reactive stepping animation obtained using inverse kinematics.

Benthe E.M. Gijzel



Faculty of Engineering Technologies, Biomechanical Engineering

Document number:
BE-943

Date:
July 2023

UNIVERSITY OF TWENTE.

Acknowledgements

I am extremely grateful to Aurora, Edwin and Rai for the amount of support, motivation and feedback they have provided me throughout the project. I had the pleasure of working under the wonderful guidance of Aurora, whose positive attitude and kindness kept me in good spirit and gave me the space and ability to put health before work when needed.

Many thanks to Lotte Hagedoorn for giving up her valuable time to help me record the reactive stepping motions at the Balance Lab in RadboudUMC Nijmegen and showing me around the lab.

Special thanks to my family and friends for always believing in me and providing me with the confidence to keep working.

Furthermore, I would like to thank Sietse Hoekstra at 8D games for telling me about their work and providing me with their 3D character model.

Lastly, I want to mention the HEROES team for their hard work to improve post-stroke healthcare.

Abstract

ENGLISH:

Introduction - For stroke patients, poor balance is one of the most common medical complications. The HEROES project aims to prevent falls and related injuries in patients by designing an exergame to train their reactive responses. The quality of animations of virtual characters play a key role in game experience. Natural human motions can be obtained using motion capture, but this requires expensive equipment and can become time-consuming if multiple takes are required per movement. Inverse kinematics can provide a faster way of making humanoid animations by finding the bone angles of the knee or elbow when you set the hand or foot to a target position. However, not all inverse kinematics methods result in natural body posture. To help animators decide between motion capture and inverse kinematics, the quality of posture during a reactive stepping animation obtained through inverse kinematics will be studied.

Research question - What is the effect of using inverse kinematics for animating reactive stepping motions on the quality of a character's posture?

Method - Reactive stepping motions in 5 different directions were recorded with a single subject through motion capture: forwards, diagonally forwards, sideways, diagonally backwards and backwards. The recorded motions were retargeted to a character. The reactive stepping motions were then replicated using inverse kinematics in Blender, an open-source 3D design software. The posture obtained through inverse kinematics was compared to that obtained through motion capture. Differences between both methods were defined by root-mean-square errors (RMSE) and correlation coefficients.

Results and discussion - The largest differences in posture of the leg were found in the x-direction for all stepping movements, with a mean RMSE between 1.60 and 2.26 cm and mean correlation coefficients between 0.68 and 0.96. Furthermore, the spine showed the largest differences in the upper-spine, neck and head joints in y-direction for the backwards, diagonally backwards and/or sideways movements, with a mean RMSE ranging between 1.29 and 2.92 cm and a mean correlation coefficient of 0.35 to 0.81. However, the differences between the characters in the spine, neck and legs were found small enough to partly disappear in the mesh of the character when the animation is viewed from a distance, as it will be when used in the exergame.

NEDERLANDS:

Introductie - Evenwichtsproblemen zijn een van de meest voorkomende complicaties na een beroerte. Het HEROES project probeert de kans op een val te verminderen door het evenwicht te trainen met hehulp van een exergame. De kwaliteit van de animaties is erg belangrijk voor de ontwikkeling van de exergame. *Motion capture* is een manier om natuurlijke bewegingen *real-time* vast te leggen en deze kan worden omgezet tot een animatie, maar dit process kan tijdrovend zijn en vereist speciale apparatuur. *Inverse kinematica* is een manier om sneller animaties te maken door de houding van een lichaamsdeel wiskundig te bepalen aan de hand van de positie van de hand of voet. Het nadeel van inverse kinematica is dat het niet altijd natuurlijke houdingen oplevert. Deze studie probeert inzicht te werven in de kwaliteit van de houding van het virtuele karakter tijdens de animatie van een reactive stap gemaakt met behulp van inverse kinematica.

Onderzoeksvraag - Wat is het effect van inverse kinematica op de houding van een karakter indien gebruikt voor het animeren van reactieve stappen.

Methode - Tijdens een opnamesessie is de beweging van een reactive stap in vijf verschillende richtingen vastgelegd met behulp van *motion capture*: voorwaarts, schuin voorwaarts, zijwaarts, schuin achterwaarts en achterwaarts. Deze bewegingingen zijn omgezet tot verschillende animaties. Deze animaties zijn vervolgens nagebouwd met toepassing van inverse kinematica in Blender, een 3D ontwerp software. De houdingen tijdens de animaties verkregen na de *motion capture* en na toepassen van de inverse kinematica zijn met elkaar vergeleken. Verschillen werden gedefinieerd met behulp van het kwadratisch gemiddelde (RMSE) en de correlatiecoefficient.

Resultaten en discussie - De grootste verschillen in houding van het been zijn gevonden langs de x-as voor alle stappende bewegingen, met een gemiddelde RMSE tussen de 1.5 and 2.5 cm en een gemiddelde correlatiecoefficient tussend de 0.679 and 0.956. Daarnaast zijn grote verschillen gevonden in het bovenste gedeelte van de ruggengraat en het hoofd langs de y-as tijdens de achterwaarts, schuin achterwaarts en zijwaarts stappende bewegingen, met een gemiddelde RMSE tussen de 0.849 and 2.2919 cm en een gemiddelde correlatiecoefficient van 0.348 tot 0.991. Dit verschil zal minder op te merken zijn als het karakter's *mesh* is toegepast en de animatie van een afstand wordt bekeken, zoals bij de exergame gebeurt.

Contents

1 Introduction

As the population is aging, the number of patients suffering from cardiovascular diseases is rising. With the recent COVID-19 pandemic increasing the chance of strokes in patients with cardiovascular diseases, a large group of patients is currently seeking post-stroke healthcare [1]. An increase in stroke incidence of 70% was found in the period from 1990 to 2019 [2]. Gait and balance disorders are very common in patients suffering from strokes [3],[4]. Moreover, a stroke patient shows a 2-4 times higher incidence of fractures in case of a fall compared to a patient without a history of strokes [5]. Hence, restoring balance plays an important role in post-stroke care. Balance rehabilitation focuses on training a patient's reactive stepping. Reactive stepping is a stepping motion made to counter a temporary imbalance to prevent a fall. These personalized rehabilitation programs are costly, lengthy and often located inconveniently to patients. This can decrease the motivation of patients to start or continue with their rehabilitation program. Exercise intensity has been linked to an increase in patient recovery [6]. Moreover, lack of physiotherapy can lead to permanent disabilities. Motivation for and enjoyment of these exercises are therefore of key importance.

The HEROES project is designing a video game to provide patients with a home-based rehabilitation program, so that they can remain motivated while training their reactive stepping. The concept of the game is as follows: it displays a character on a raft making reactive responses to stay in balance [7]. The player follows the stepping motions to train their reactive stepping and their movement is tracked to allow visual feedback. The game therefore consists of three main components: a character playing the stepping animations on screen, motion tracking and the real-time feedback. A previous study has been performed to explore different types of (optical) motion trackers and their advantages and disadvantages [8]. This study will focus on the movement in the character animation shown on screen.

Animating a character is based on one of two methods: forward kinematics (FK) and inverse kinematics (IK). A character has a skeleton, or armature, with multiple bone chains (legs, arms, spine e.g.) consisting of bones that can rotate. In FK, all angles between two bones in a chain are adjusted individually to change the character's posture. With IK, only the position of the end-effector (such as a hand or foot) is changed manually, forcing the intermediate bones to change angles accordingly. There are often multiple combinations of joint angles possible to get to the same end-effector position due to the many degrees of freedom, as shown in Figure 1. Some of these combinations give unnatural posture. IK methods take this into account by using rotational constraints. However, the mathematical equations used in IK are often not solvable, because there are no or too many solutions. Moreover, research shows that in using it for character animations, there is always a trade-off between computational costs and quality of posture [9],[10].

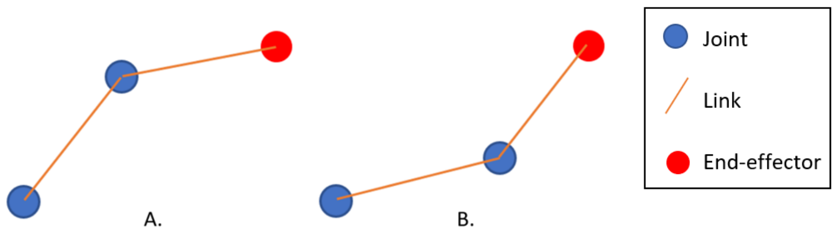


Figure 1: Example of two different arm configurations that result in the same end-effector position.

When there are many bones present in the character, manually rotating each joint becomes tedious. These animations can be created using motion capture (MOCAP). During MOCAP, the movement of a subject is recorded in real-time. Although many human motions have already been recorded and are freely available on platforms such as Mixamo[1], specific movements require separate recording. High quality motion capture takes time, requires expensive equipment and does not allow for much flexibility. To design characters therefore, using IK methods would be preferable. However, for the rehabilitation game it is important that the character moves smoothly and copies natural body movement closely, which is easier to achieve using the MOCAP method.

To let the game developers make an informed decision on which method to use, the goal is to ***study the quality of posture in a reactive stepping animation made using inverse kinematics***. For this, the IK-based stepping animations will be compared to the same stepping animations created using motion capture. To make a valid comparison, there is one important criterion for the IK animation: the joints in the start and end of the bone chains must make the exact same movements as those in the MOCAP animation.

First, a general introduction to character design is given. Next, an overview of different inverse kinematics methods was created to get an insight in how they perform in an animating environment. Different methods of recording and processing MOCAP data and different types of animation software were explored in order to set up a well-designed experiment (see chapter 2).

The methods used in this study are divided into three parts and can be found in chapter 3. The first section describes how the MOCAP recording and data processing was performed and how the MOCAP animations were obtained. In the second section, the making of the IK animations in Blender is presented. The last section introduces the quantitative analysis, where the location of each joint in the MOCAP and IK armatures is obtained and a difference in joint position between that of the MOCAP and IK armature is calculated. The root mean square error ($RMSE$) and correlation coefficient (CC) is then calculated to define how well posture of both characters match throughout the animations. Furthermore, the 'area under the curve' of each bone chain is calculated to find the 'worst-case scenario's'. The results of this quantitative analysis can be found in chapter 4). The findings from the results are presented and discussed in chapter 5. A list of all citations can be found under References. The appendices are given at the end of this document and show further details about different parts of the methods used in this research.

[1]Mixamo is a platform owned by Adobe that offers a library of full-body animations.
Link: https://www.mixamo.com/#/

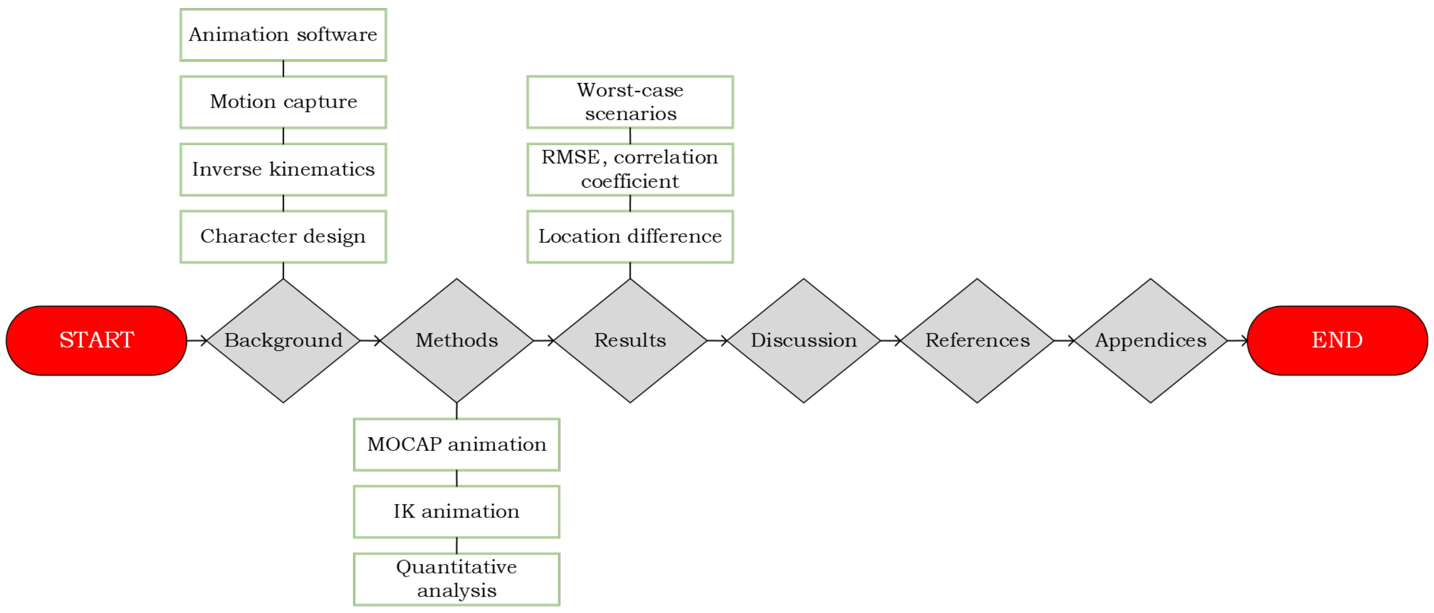


Figure 2: Flowchart showing an overview of the different chapters document.

2 Background

2.1 Character design and animating

A virtual character can be designed using any 3D software, such as Blender, Cinema 4D or Wings3D. A character consists of two main components: the mesh and the armature. A mesh gives shape and texture to the character. The armature allows movement of the character and is required for animating. An armature, or rig, is a simplified skeletal structure consisting of bones intersected by joints and is attached to the mesh through parenting. The hierarchy of the bones play a key role in animating. For example, when you rotate the shoulder joint, the rest of the arm moves with it. When you rotate the wrist, only the hand moves with it, but not the shoulder. The shoulder is therefore considered a *parent* to the elbow and wrist. Any rotation of a *parent* will result in movement of a child. The reverse is also true: for the hand to move, its closest parents needs to move too. This is the basic principle of inverse kinematics. The root bone is defined as the start of the hierarchy and controls the position and rotation (transform) of the entire character.

2.2 Inverse kinematics

Over the years, many inverse kinematics methods have been introduced, each with their own advantages and disadvantages. Following are a few examples:

In robotics, the Jacobian method is often used to determine what forces and torques are acting on a joint when the end-effector moves to a target position with a certain velocity, but it can be used for animating as well. It uses a matrix consisting of partial derivatives, called the Jacobian matrix, to find a linear approximation to the IK problem [10]. Methods include the Jacobian Transposed, the Pseudoinverse method and the Damped Least Squares (DLS) or Selectively Damped Least Squares (SDLS) methods [11],[12],[13],[14]. The Jacobian Transposed method gives fast but poor results, while the DLS methods gives better results but works slower. Disadvantages of these methods are their high computational costs, which make them often unsuitable for real-time application, and occurrence of singularity problems.

Another method used in robotics and animation is the cyclic coordinate descent (CCD) method [15],[16]. This method approaches the IK solution with one degree of freedom (DoF) at a time, which makes the equations more simple and leads to faster computations. A disadvantage of this method is that it results in unnatural poses of joints close to the end-effector, but improvements have been put forward to make the human animations look more natural [17],[18],[19].

More recently, a research group proposed a method that trains the model to learn a set of predetermined poses [20]. "*Given a set of constraints, the system can produce the most likely pose satisfying those constraints, in real-time*". Training session have to be held separately, but once recorded it can be used for multiple game scenarios. A disadvantage is that the poses are limited to those studied by the models during the training session.

A research group working for the university of Cambridge came up with a method that gets around the use of rotational angles [9]. The Forward And Backward Reaching Inverse Kinematics (FABRIK) "*finds each joint position via locating a point on a line*". This method was tested against the CCD and Jacobian methods in multiple scenarios, including the use of multiple end-effectors and in real-time. The results show faster computation, better tracking accuracy and more consistency in natural body posture. It can also be combined with MOCAP, where only the end-effector position needs to be tracked. A disadvantage of this method is that applying constraints to the bones is hard to implement in a code for amateurs and extensions for animation software using this method are expensive.

2.3 Motion capture

There are different methods of recording motions currently in existence. Inertial systems use sensors directly on the body. They measure the relative positions and angles of the sensors, but require additional systems to measure their positions in space. Mechanical systems use mechanical motion capture skeletons to track the actor's joint angles directly. Optical motion capture systems measure the positions in 3D space of specific markers placed on the moving body. Optical systems are generally more accurate and more freely available than the other systems.

The markers used in optical motion capture can give light themselves (active), or reflect incoming light (passive). A minimal of two cameras is required to receive data in 3D. Sometimes during recording a motion, a body part can hide one or more markers from the cameras view. Moreover, if the targets are in close proximity to each other, a camera might not be able to distinguish them. Using multiple cameras can therefore increase measuring accuracy. Additionally, accuracy of the cameras can be negatively influenced by lighting, colors and objects in the background [8]. High-quality MOCAP therefore requires a special environment that minimizes light reflection in the background.
The data obtained during optical motion capture consists of the positions of each individual marker in 3D space. This does not equal the positions of the joints, as the markers are not directly placed on the joint, but on different parts of the body. The data will therefore need to be processed before using it for the character animation. Companies such as Qualisys and Vicon specialize in motion capture and offer software that helps with the data processing, such as gap-filling and skeleton solvers.

Applying the recorded animation to a character is called animation retargeting. The skeleton generated by the MOCAP processing software will almost always be different in shape and composition to the character's armature. During animation retargeting, each bone of the source armature (the skeleton fitted to the MOCAP data) is matched to those of the target armature (the character's armature), to account for the differences in size and orientation of the bones between the two armatures.

2.4 Animation software

Designers often use Blender for character animations, as it is open source and contains many design and animation features. Blender offers two types of IK: the *inverse kinematics* bone constraint and the *spline IK* bone constraint. The Blender manual[2] (Animation & Rigging > Armatures > Posing > Bone Constraints > Introduction) does not explain what method of IK is used in the standard IK, but it implies that some form of Jacobian method is used [21] through the second paragraph. The IK uses two inputs: a target bone and a pole bone. The target bone is used to steer the end-effector to a target location, while the pole bone is used to control the direction in which the bone chain bends. Spline IK is used to bend the bone chain along a predetermined curve and can be used to animate spines and tentacles. It is not known if this method uses a specific IK method either.

[2]Link: https://docs.blender.org/manual/en/latest/animation/armatures/posing/bone_constraints/inverse_kinematics /introduction.html (june 2023)

Unity Real-Time Development Platform, or Unity in short, is a game development platform. It offers different ways of applying inverse kinematics. Their standard method consists of writing basic IK parameters, such as the target orientation, using a *C#*-script applied to the character [22]. Another way is to use the *Animation Rigging* package in Unity to give the character an armature [23]. It applies IK via the *Two Bone IK constraint* and the *Chain IK constraint.* The Two Bone IK constraint allows IK control via a target and a hint object, similar to Blender's IK bone constraint [24]. The Chain IK constraint uses only a target object and is said to implement the FABRIK solver. Lastly, a whole set of assets can be found in the Unity asset store for implementing inverse kinematics. They all have different uses, and some of them require payment.

Animation retargeting can be performed in Blender and in Unity. Unity matches both the target and the source rig to a standard rig and naming system implemented by the software. Blender requires a Rokoko Studio Live extension for animation retargeting, which uses an automatically or manually generated bone list to match the source bones directly to the target bones.

2.5 Related work

Most studies involving IK are in the field of robotics. However, with the rise in game development and animated movies over the last 20 years, the use of IK in animation has been studied increasingly. Some evaluate the IK method based on how well it reaches the target, while others focus more on computational costs or try to obtain more natural movement [9],[16],[19]. Research also differs in the type of body movement targeted. For interacting with the (virtual) environment, hand and arm movement are of key importance. Most of the research therefore choose to simulate arm, hand or finger movement [25],[26],[27],[28]. Combining IK of each part of the body for full-body control has also been proposed [9],[29]. Using IK for lower-body movement, such as walking or running, has been studied less. One study proposes a momentum-based IK method to simulate reactive responses. [30]. However, this method could be too sophisticated for your goal. By researching the quality of posture through reactive stepping animations based on an IK method that is more widely available, we hope to provide enough insight so that a well-informed decision can be made about using IK methods instead of motion capture.

3 Methods

This chapter describes the methods used to obtain the MOCAP and IK animations and perform the quantitative analysis. The chapter is divided into three sections. The first section describes how the MOCAP recording and data processing was performed and how the MOCAP animation was obtained. In the second section, the method of obtaining the IK animation in Blender using bone constraints is explained. The last section introduces the quantitative analysis, in which the location data of each joint in the MOCAP and IK armatures was obtained using a Python script. The location data was then used to find the difference in position of each joint between that of the MOCAP and IK armature. The root mean square error (RMSE) and correlation coefficient (CC) for each joint along each axis were calculated to define how well the two armatures match throughout the animations. Furthermore, the ‘area under the curve’ of each bone chain was calculated to find the ‘worst-case scenarios’ (WCS). The following flowchart shows an overview of the method section.

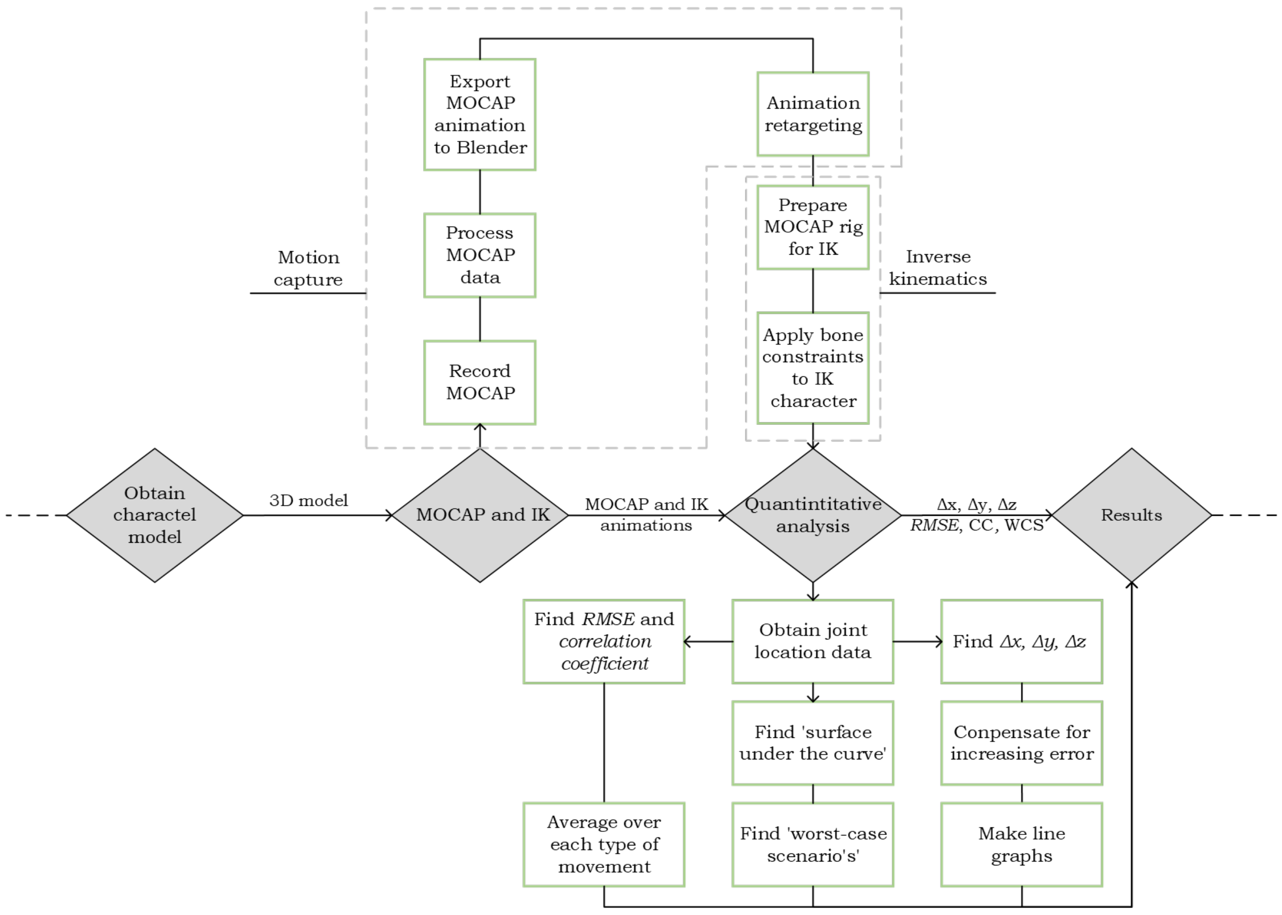

Export MOCAP animation to Blender
Animation retargeting
Process MOCAP data
Prepare MOCAP rig for IK
Motion capture
Inverse kinematics
Record MOCAP
Apply bone constraints to IK character
Obtain charactel model
3D model
MOCAP and IK
MOCAP and IK animations
Quantintitative analysis
Δx, Δy, Δz
RMSE, CC, WCS
Results
Find RMSE and correlation coefficient
Obtain joint location data
Find Δx, Δy, Δz
Find 'surface under the curve'
Conpensate for increasing error
Average over each type of movement
Find 'worst-case scenario's'
Make line graphs

Figure 3: Flowchart showing the different parts that form the methods chapter.

For this study, 8D Games B.V. kindly allowed the use of the character designed for the HEROES exergame in Blender, see figure 4. Here, the mesh is shown in grey and the armature in black. The pelvis-bone is marked as the root-bone. The axis in the Blender file were defined as follows: the positive y-axis points backwards, the positive z-axis points upwards and the positive x-axis points towards the right as seen from the front.

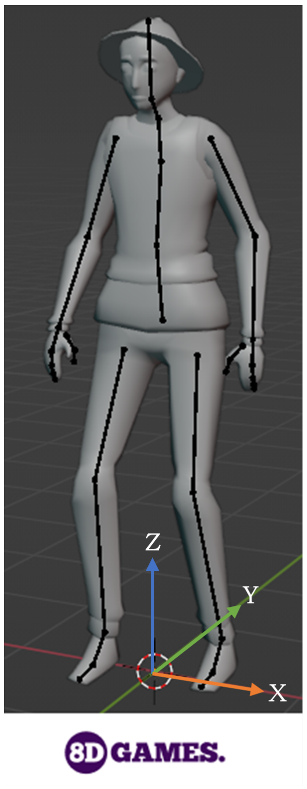


Figure 4: The HEROES of the sea adventurer's character presented in Blender as shared by 8D Games B.V.. The character's mesh is shown in grey, while the black lines represent the character's armature.

3.1 Animation: MOCAP

The process of making the MOCAP-based animation can be separated into 3 main steps: First, the MOCAP data was recorded. Secondly, the collected data was processed so that it can be used to build the character animation. Lastly, the animation was retargeted to the character. The reactive stepping motions were recorded using optical motion capture. The data was processed using QTM software by Qualisys. The animation retargeting was performed in Blender, as it was found better equipped for retargeting the armatures obtained by the Qualisys software and allowed more control over linking individual bones.

3.1.1 Recording motion capture

Experimental setup
The motion capture took place in the balance lab of the RadboudUMC Nijmegen, which is fully equipped for recording reactive stepping motions. The room contained a balance platform on ground level and a Vicon camera system (Vicon Motion Systems, United Kingdom). Eight Vicon Vero cameras were placed around the platform on the walls and ceilings, facing the middle of the platform marked by a white mat (see figure 5). The Vicon Vero cameras send infrared light and record the reflection from the markers. One Vicon Vue camera was also placed on the wall, facing the white screen diagonally from the left, to record the room setup.

To record data from the cameras, Vicon Nexus software (version 2.10.03) was installed on the computer connected to the cameras. Platform Motion software was used to instruct the platform. Calibration was performed before recording to account for background reflections and to define the workspace. The calibration also determined the coordinate system as follows: the y-axis faces away from the white screen, the z-axis upwards and the x-axis towards the left (as seen from figure 5). The origin is defined at the middle-back side of the platform at platform level.

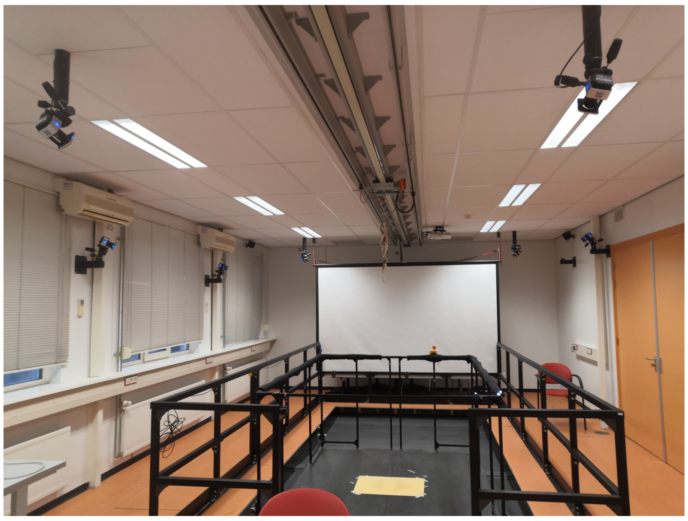


Figure 5: A photo of taken of the motion capture setup used during the recording session. Shown are some of the Vicon Vero cameras, the white screen facing the balance platform and the white mat representing the middle of the platform.

Passive markers were placed on the body according to the Qualisys Animation Marker Set, see figure 6. For naming and further details of the marker placement, see Appendix I. A combination of non-reflective one-sided and double-sided tapes was used to secure the markers on the subject's body. Three reference markers were placed on a corner of the balance platform.

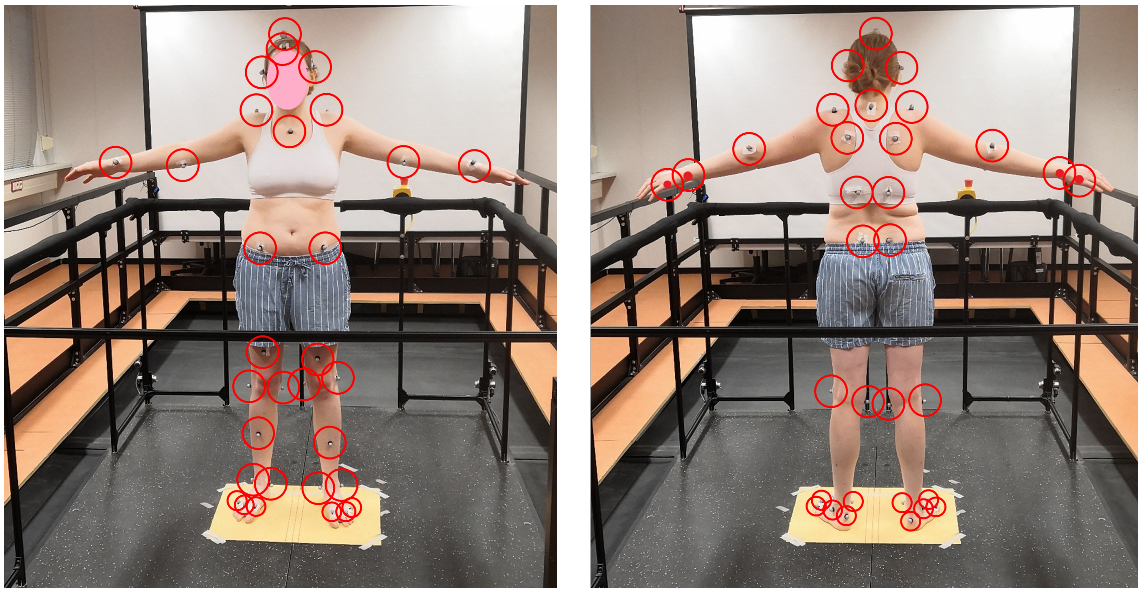


Figure 6: Photo of the marker placement on the subject used during the MOCAP session, as instructed by the QTM animation marker set (see Appendix I). Left: front view of the subject. Right: back-view of the subject

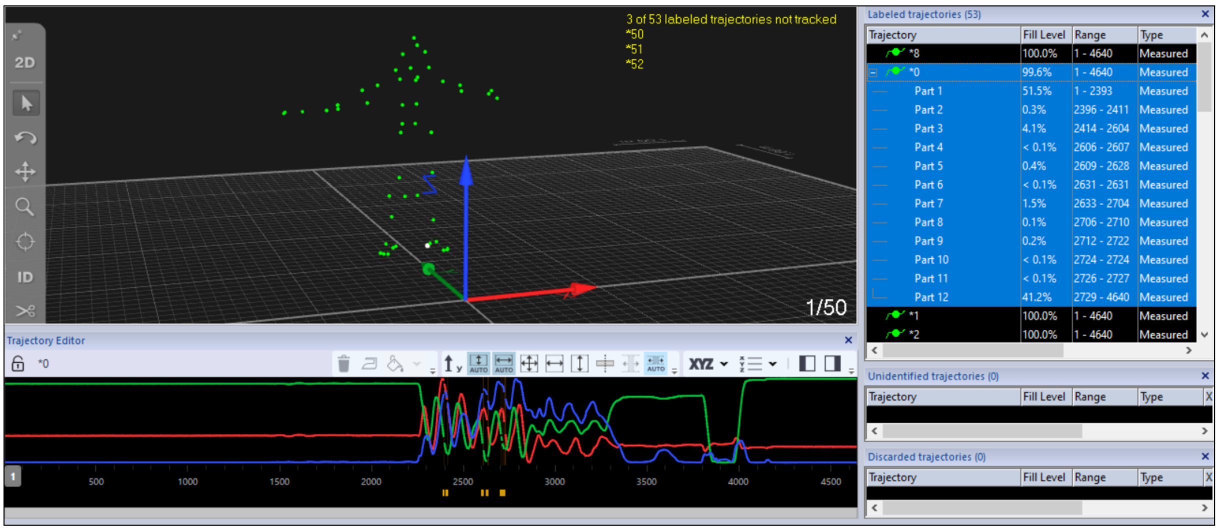
Recording the movements

The Vicon Nexus software is not compatible with the Qualisys animation set. Therefore, the markers could not be labeled beforehand. According to the Qualisys manual and tutorials [31], an AIM-model is required for automatic labelling of the markers and skeleton calibration in the QTM software later on. Therefore, an AIM-trial was recorded first: the subject started in a T-pose position and then performed head, arm and leg rotations and lunges. Next, the movement trials were recorded. During each recording, the subject started in a T-pose before moving to a rest pose (A-pose). This allows for recalibration of the skeleton later on. Next, the platform was instructed to move, forcing the subject to make a stepping motion in the opposite direction to regain balance. Afterwards, the subject moved back in position. The subject made stepping movements in five different directions: forwards, diagonally forwards, sideways, diagonally backwards and backwards. Each of these five movements was recorded three times to ensure that usable data was obtained. During all movement trials, the subject started at the center of the platform (white mat) and was facing the screen, or negative y-direction. Reactive stepping was always performed by the right foot. All trials were recorded at $100fps$.

3.1.2 Processing the MOCAP data

For processing the MOCAP data, Qualisys Track Manager (QTM) software was used. The software is specialized for animating purposes and includes a skeleton solver [31]. The skeleton solver is trained to fit a skeleton to the complete set of measured markers that can be used for animation retargeting. The QTM software also offers an AIM-model feature for automatic identification of the markers. The AIM-model is trained to recognize the animation marker positions on a subject using the aforementioned AIM-trial. To fill the gaps in the positional data as a result of hidden markers, Qualisys offers multiple gap-filling methods for different gap sizes and types. All aforementioned methods are explained in detail in the QTM manual [31].

The recorded trials were each exported as *.c3d* files and imported into QTM. Upon opening a *.c3d* file, a 3D view and a list of trajectories appeared, see figure 7. A trajectory is the measured path of a single marker. Each trajectory is split into parts that contain the measured data and are separated by gaps if any are present.



Labeled trajectories (53)

Trajectory	Fill Level	Range	Type
*8	100.0%	1 - 4640	Measured
*0	99.6%	1 - 4640	Measured
Part 1	51.5%	1 - 2393	Measured
Part 2	0.3%	2396 - 2411	Measured
Part 3	4.1%	2414 - 2604	Measured
Part 4	< 0.1%	2606 - 2607	Measured
Part 5	0.4%	2609 - 2628	Measured
Part 6	< 0.1%	2631 - 2631	Measured
Part 7	1.5%	2633 - 2704	Measured
Part 8	0.1%	2706 - 2710	Measured
Part 9	0.2%	2712 - 2722	Measured
Part 10	< 0.1%	2724 - 2724	Measured
Part 11	< 0.1%	2726 - 2727	Measured
Part 12	41.2%	2729 - 4640	Measured
*1	100.0%	1 - 4640	Measured
*2	100.0%	1 - 4640	Measured

Unidentified trajectories (0)

Trajectory	Fill Level	Range	Type	X

Discarded trajectories (0)

Trajectory	Fill Level	Range	Type	X

Figure 7: The standard overlay of the QTM software, containing a 3D view of the markers, a trajectory list and trajectory editor, as obtained when opening the *Aim03.c3d* file.

Preparing the AIM-model

First, the markers in the AIM-trial needed to be labeled correctly. For this purpose, all trajectories labelled as numbers by the Vicon software were unidentified and the old trajectory names deleted. Next, 51 new trajectories were added to the trajectory list and renamed to the 48 markers from the QTM animation marker set and the three reference markers of the balance platform. The unidentified trajectories were individually inspected and labeled using the new trajectory list. Markers that switched name within a trajectory were separately identified. When all trajectories were correctly identified, a $B\_$ prefix was added to the 48 animation markers to tell the software those belong to the same subject.

Next, any gaps in the data resulting from hidden markers were removed using the gap-filling methods offered by QTM. All small gaps of ≤ 100 frames were filled with the polynomial type that uses interpolation. When the gaps were separated by one frame, as in figure 8, or were present at the beginning or end of the trajectory, the gaps remained unfilled, since interpolation is then impossible. These gaps were filled instead using the linear type, followed by the kinematic type, which fills the gaps based on the movement of the skeleton segment. The current frame number was set such that all trajectories were tracked and the subject was in a T-pose position and the skeleton was calibrated before performing the kinematic gap-filling.

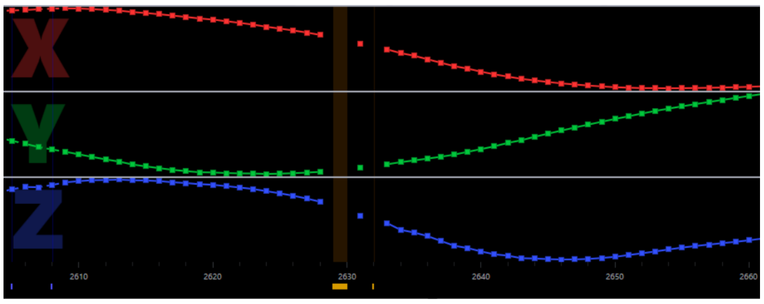


Figure 8: A graph showing the x-, y- and z-coordinates of the *LHeelBack* trajectory plotted against the number of frames. This is an example of a gap that remains unfilled after polynomial gap-filling is applied.

The file is now ready for generating the AIM-model. For this, the 48 trajectories were selected and a new AIM-model was generated from the selection. For further details, see Appendix II.

Processing the movement trials

The data from the movement trials was processed in a similar way to that of the AIM-trial, but instead of manually labelling the markers and creating the marker connections, the AIM-model was applied to the data so that it labels automatically. Before filling the gaps, each trajectory was inspected and adjusted if needed to ensure correct marker identification. To finish up, the skeleton solver was reprocessed to ensure the skeleton was fitted to the completely gap-filled data. The resulting skeleton is shown in figure 9. While processing the data, it was found that one of three recorded diagonally-backwards movement trials was accidentally overwritten.

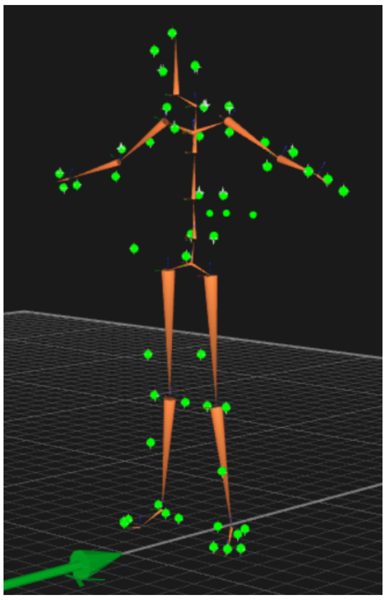


Figure 9: An image of the skeleton (in orange) created by the QTM skeleton solver software to fit the marker data (the green dots).

3.1.3 Animation retargeting

For animation retargeting, it is easier if the bone hierarchy of the source armature matches that of the target rig. When comparing the character's armature to that generated by the QTM software, it was found that the rig's do not match around the shoulder and neck area, so either the source or the target armature needed to be adjusted to match the other. The easiest option was to build a new armature in Blender. For this, the original armature was replaced by a basic human meta-rig offered by Blender's *rigify* add-on. The two breast bones and the 5th spine bone of the new armature were removed before editing the bones to match the character's mesh. The character's rest-pose was set to a T-pose. Figure 10 shows the old and new armatures.

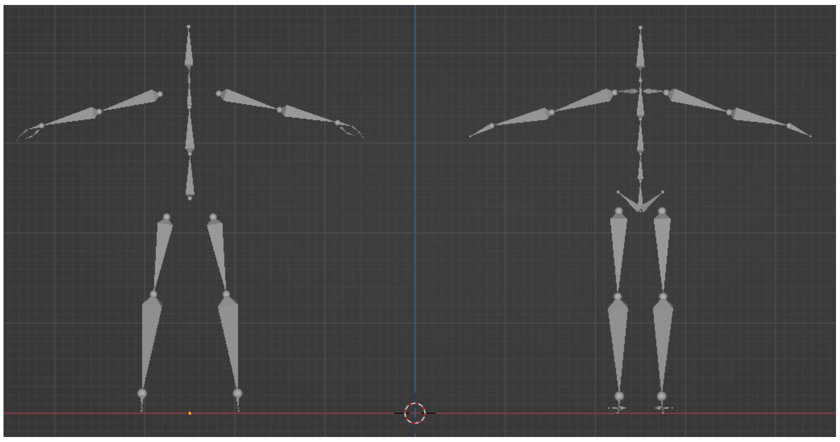


Figure 10: Capture of the character's armature in Blender in rest-pose before (left) and after (right) adjustments were made in preparation of animation retargeting.

Animation retargeting was performed in Blender (version 3.3.0), as it allowed direct control over the bone matching. To prepare the MOCAP files in QTM for export, the timeline control was adjusted to limit it to the reactive stepping motion only. The data files of each of the movement trials were then exported as *.fbx* file in binary format, including only the skeleton and time code.

The *.fbx* files containing the MOCAP data were all separately imported into Blender using automatic bone orientation, force connect children and Z-forward/Y-up orientation. The resulting source armature was then scaled and re-positioned so that the head (start joint) of the root bone (pelvis bone) of both armatures matched. Each animation was then retargeted using the Rokoko Studio Live extension (version 1.4.0). For this, auto-scaling was deselected and the target armature put into rest-pose (T-pose).

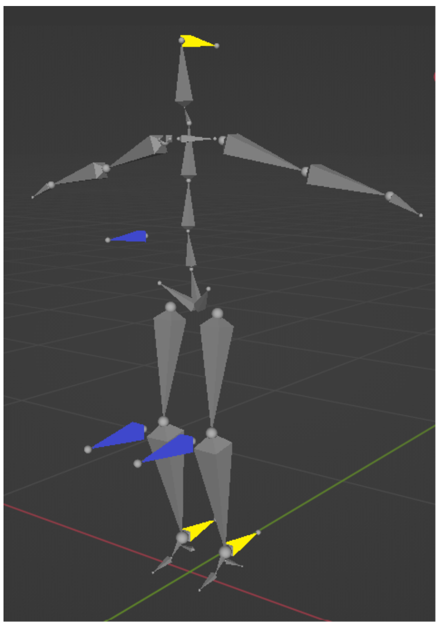
3.2 Animation: Inverse Kinematics

The IK-based animations were made in Blender, as it represents a wider community of animators and is more compatible with the obtained character. The file in which animation retargeting was performed (see chapter 3.1.3) were used as the base file and a second character was created through duplication to represent the IK character. The animating process can be divided into three steps: First, both armatures were adjusted in preparation of the IK. Second, a set of constraints was used on the IK character to meet the IK criterion (matching the locations of the start and end of each bone chain). Finally, Blender's standard IK was applied to all the chains through the IK constraint.

3.2.1 Preparing the rig for IK

For IK to be used in Blender, a separate target and pole bone needed to be added to each bone chain. The character consists of 5 chains: each of the arms, the spine and the left and right legs. However, for the analysis of reactive stepping only the (stepping) leg and spine bone chains are important. Target bones were therefore extracted from the shin bones in y-direction and from the head bone in y-direction.

Pole bones were extracted from the thigh bones (at the knee) and 2nd spine bone (around the navel), in y-direction and pointing away from the bending direction of the chain. They were moved away from the armature in y-direction to allow room for bending. See figure 11 for the resulting armature.



🟦 Pole bone

🟨 Target bone

Figure 11: Capture of the character's rig in Blender after it was adjusted in preparation of applying IK. The IK target bones are shown in yellow, the IK pole bones in blue.

3.2.2 Applying the bone constraints

As the target and pole bones (non-deform bones) are all parented to parts of the MOCAP rig, they move with the armature during the animations. To make the IK-based animation comparable to the MOCAP-based animation, the position and rotation data (transform) of these bones must be copied to the same bones in the IK-character throughout the animation. For this, the *copy transform* bone constraint was used (see Appendix III for more information).

The *copy transform* bone constraint was applied to the pelvis bones, thigh bones and all non-deform bones in the IK character (see figure 12) to ensure that the joint at the start and end of each bone chain are in the exact same position as in the MOCAP character. The complementary bones in the MOCAP character were set as the target and *local with parent* space definitions were used.

The *inverse kinematics* bone constraint was used to apply IK to each bone chain as follows:

- The IK bone constraint was added to the shin bones and the head bone.
- The target and pole inputs were set to the correct target and pole bones.
- The pole angle was set to $-90°$ for the legs, and to $90°$ for the spine.
- The number of iterations was set at 500 as by default.
- The chain length of the legs were set to 2, and that of the spine to 6.
- Use tail was selected, stretch was deselected and influence was set to 1 for all chains as by default.

All non-deform bones were de-parented from the rest of the rig to make the IK constraint work properly. Note that the foot bones are not influenced by the inverse kinematics of the leg.

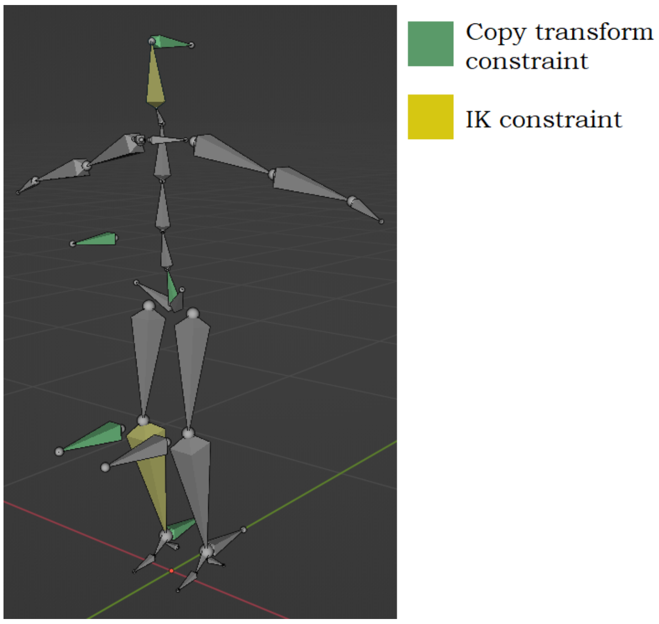


Figure 12: Capture of the character's rig in Blender showing the bones to which the bone constraints were applied: the *inverse kinematics* bone constraint is colored yellow and the *copy tranform* bone constraint is colored green.

3.3 Quantitative analysis

With all MOCAP-based and IK-based animations ready, a quantitative analysis could be performed. The analysis was limited to the right leg (stepping leg) and spine bone chains and focused only on the reactive stepping movement. First, the location data was obtained from Blender using a Python script in Blender's text editor. The location data was then analyzed further in Microsoft VS code using Python. The parameters used in the quantitative analysis are:

- $RMSE$ and CC: the root mean square error and the correlation coefficient
- Δx_r, Δy_r, Δz_r: the difference in x-, y- and z-positions of each joint between the IK and MOCAP character's armatures, compensating for the increasing error carried through the bone chain.
- $\Delta S_{xz,leg}$, $\Delta S_{xz,spine}$, $\Delta S_{yz,leg}$, $\Delta S_{yz,spine}$: the area under each bone chain w.r.t the global axis in the xz- and yz-plane.

The maximum values of ΔS were used to find the animation and keyframe numbers of the worst-case-scenario's (see also figure 3). All parameters were obtained using a Python script.

3.3.1 Finding the 3D position data of the joints

For the analysis, all bones in the arms and left leg as well as the two pelvis and heel bones were removed from each armature in a new file, so that only the bones that play significant part in the reactive stepping remain. The characters were then re-scaled equally on all axes until it measures 1.69m in the z-direction, to match the length of the test subject. Lastly, the character's mesh was hidden from the Viewport, so only the armature is visible.

The global location and rotation data of the joints can not be directly obtained in Blender, as bone movement is always defined with respect to the bone's parent and not to the global world. To go around this problem, empty objects were added into the scene and placed at the position of each joint throughout the animation. Global position data of the joints could then be obtained from the empty objects instead. To place an empty object at the joint, the snapping tool in Blender was used. For more information about how this tool is used, see Appendix IV. As the snapping tool can only snap the empties to the head (start) of each bone, the endpoint of each bone chain would normally be left out of the analysis. The head of the spine target and toe bones were used to obtain these positions. Obtaining the locations of the start and end of each chain allows to check if they are indeed the same in both animations (IK criterion), otherwise the animations can not be compared to each other.

Manually snapping an empty object to each individual joint for all keyframes would be time consuming and tedious. Therefore, a Python (v3.10.8) script was written using Blender's *text editor* to obtain a matrix containing the x,y and z-positions of all of the joints. For more information about this Python code and how to use it, see Appendix IV. The location data was expressed in meters.

3.3.2 Finding Δx, Δy, Δz, the root mean square error and correlation coefficient

Before performing the analysis, each cell in the location data Excel file was multiplied by 100 so that it was expressed in centimeters. The following parameters were found using new Python script (v. Python 3.9.13) in Microsoft VS code. The column order in each sheet was changed to fit the Python file.

Δx, Δy, Δz (cm) were found by subtracting each data point in the MOCAP data from the corresponding data point in the IK data. These errors were then used to find the root mean square error (RMSE) for each animation, each joint and each axis. Additionally, a correlation coefficient (CC) was calculated. As there are multiple animations of each of the five reactive stepping motions (backwards, diag. backwards, diag. forwards, forwards and sideways), a mean RMSE and CC was calculated for each of those five stepping motions, for each joint and axis separately.

$$RMSE_{j,x} = \sqrt{\frac{\sum \Delta x_j{}^2}{n}}$$

$$CC_{j,x} = \frac{n \sum x_m x_{ik} - \sum x_m \sum x_{ik}}{\sqrt{(n \sum x_m{}^2 - (\sum x_m)^2)(n \sum x_{ik}{}^2 - (\sum x_{ik})^2)}}$$

With j the joint, x the axis and n the number of data points, or keyframes, x_m the x location of the joint in the MOCAP armature (cm) and x_{ik} the x location of the joint in the IK armature (cm).

As the error carries through to the bones further on in the chain, extra calculations were required to find the real Δx, Δy and Δz of each individual joint. These can show behaviour of each joint more clearly. The following formula was applied:

$$\Delta x_{i,r} = \Delta x_i - \Delta x_{i-1,r}$$

Where i represents a joint in the character's armature following the bone hierarchy in each bone chain (see Appendix V for an overview of the joint names and hierarchy). Note that this formula is only applied to the intermediate joints.

3.3.3 Finding the worst-case-scenarios

To visualize how Blender's inverse kinematics method behaves at it's worst, the animation and keyframe number associated with the worst-case-scenarios needed to be found first. The worst-case-scenario was defined as the point in the animation where the difference in summed area (2D) under a bone chain (w.r.t the global z-axis) between that of the MOCAP and IK armature is the largest. This was seperated into four situations, as there are two bone chains (leg & spine) and two planes (xz & yz). The surface S (cm^2) created by each individual intermediate bone in a bone chain was calculated as follows:

For $x_i > x_{i+1}$:

$$S_{x,i} = (0.5 \cdot dz \cdot dx) + (x_{i+1} \cdot dz)$$

For $x_i < x_{i+1}$:

$$S_{x,i} = (0.5 \cdot dz \cdot dx) + (x_i \cdot dz)$$

Where i represents a joint in the character's armature following the bone hierarchy in each bone chain, and dx the difference between x_i and x_{i+1} (see figure 13).

The individual surfaces were summed to find the total surface under each bone chain. Next, ΔS was calculated by subtracting the S found in the MOCAP character from that found in the IK character. Finally, the maximum ΔS for the right leg chain and the spine chain in xz- and yz-plane and the corresponding frame number and name of the animation were obtained through the Python code.

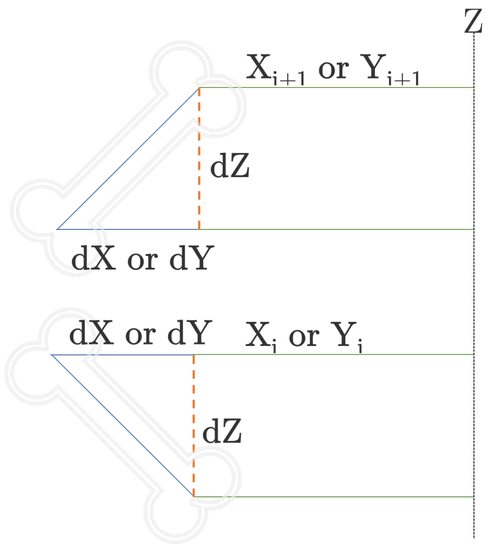


Figure 13: Simplified schematic of the surface created by each bone with respect to the z-axis. Shown are the parameters x, y, dx, dy and dz. Here, i represents the joint number in the character's armature following the bone hierarchy of each bone chain (see Appendix V).

4 Results

4.1 General observations

After closer inspection, the snapping method used by the python script in Blender did not snap the empty objects center to the exact center of each joint (see figure 14). Instead, it has a slight offset that is not equal in direction or magnitude over each joint. It also appears to differ in magnitude and direction throughout an animation. The offset magnitude is estimated to be maximally 2 mm in total length (hypotenuse). However, the offset occurring when the object is snapped to a joint in the MOCAP armature is the same as when it is snapped to the same joint in the IK armature, so Δx, Δy and Δz remain the same.

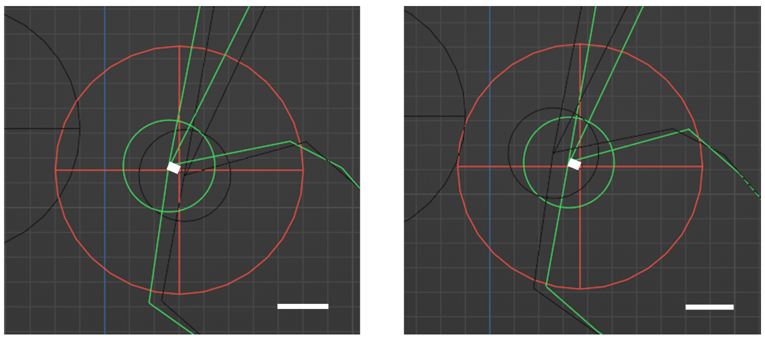


Figure 14: Images of the knee joint (green lines) and the *knee* empty object (red circle) in Blender in the yz-plane. Left: MOCAP armature. Right: IK armature. The white stripe represents the offset between the center of the joint and the center of the empty object. The white bar in the bottom right corner indicates 1 cm.

Before observing the results of the quantitative analysis, it was checked if the locations of the start and end of each bone chain was indeed equal for both characters throughout all animations. The Δx, Δy and Δz for the hip, spine and head tip all neared 0 throughout the animations. However, the ankle joint showed generally larger Δx, Δy and Δz than the other joints. It's Δx, Δy and Δz remained under 0.003 cm throughout all animations.

4.2 Δx_r, Δy_r and Δz_r

The Δx_r, Δy_r and Δz_r of each joint were plotted against the progress of the animations (in keyframes). As each data point of the MOCAP dataset was subtracted from the corresponding data point of the IK dataset, the x-axis serves as a baseline representing the MOCAP data with each of the curves showing how the IK dataset deviates from the MOCAP dataset. A negative value for Δx, Δy and Δz means that the joint in the IK armature is located more in negative x, y or z-direction with respect to the same joint in the MOCAP armature. See Appendix VI for the definitions of the axis in Blender.

The $|\Delta z_r|$ always remained under 0.8 cm for all of the joints and was noticeably smaller than $|\Delta y_r|$ and $|\Delta z_r|$ under all circumstances (see figure 23). The $|\Delta x_r|$ of the *knee* joint was larger then any of the spine joints throughout all animations, during peak movement (where the foot makes an arch). Furthermore, $\Delta x_{r,knee}$ remained negative throughout all animations, while $\Delta y_{r,knee}$ stayed positive. The other joints did not show the same behaviour. The *neck* joint showed the largest $|\Delta y_r|$ of all spine joints throughout all animations, followed by the *upper-spine* joint. An example line plot of Δx_r, Δy_r and Δz_r showing these behaviours can be found in Appendix VII.

4.3 RMSE and correlation coefficient

The joint at the start and end of each bone chain showed an RMSE of near 0 and correlation coefficients of near 1 for all axis. The bar plots in figures 15 and 16 present the mean RMSE and correlation coefficients over each stepping movement for each intermediate joint in x- and y-direction. The mean RMSE and correlation coefficients of all joints in the z-direction were found as 0-0.5 cm and close to 1.000 respectively.

The largest mean RMSE values and smallest correlation coefficients were found for the *knee* joint in the *x-axis* for all stepping movements (mean RMSE: 1.60-2.26 cm, mean CC: 0.68-0.96) and for the *upper spine*, *neck* and *head* joints in the y-axis for the backwards, diagonally backwards and/or sideways stepping movements (mean RMSE: 1.29-2.92 cm, mean CC: 0.35-0.81).

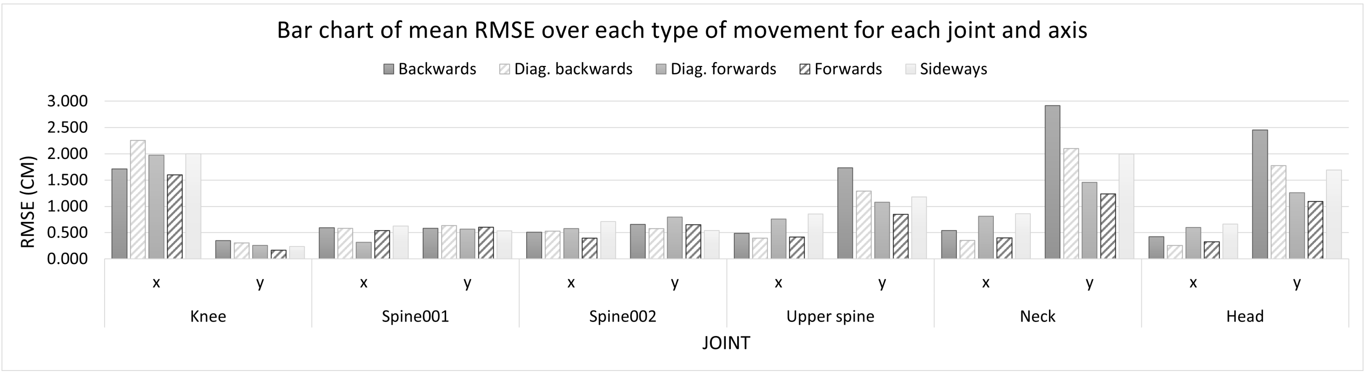


Figure 15: Bar plot showing the mean RMSE over each stepping movement for each joint and axis. See for calculations chapter 3.3.2.

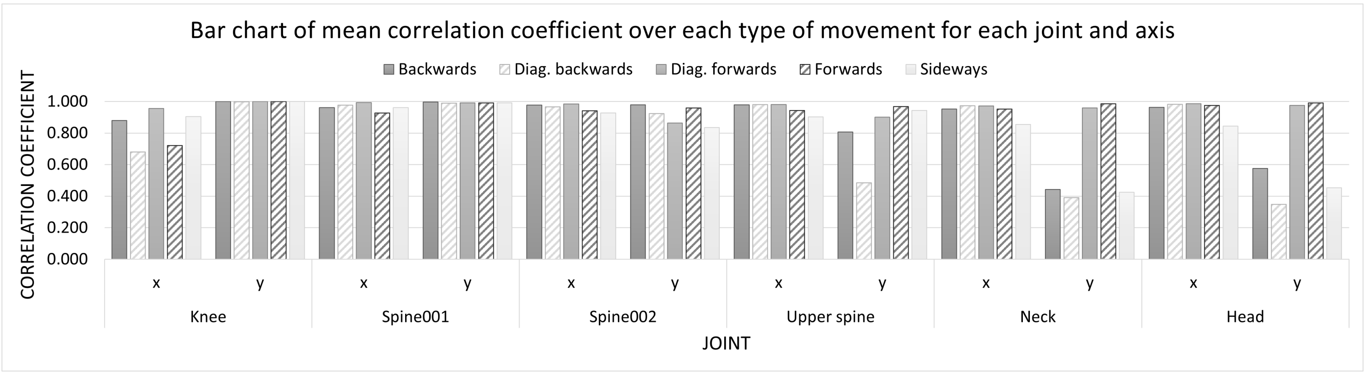


Figure 16: Bar plot showing the mean correlation coefficient over each stepping movement for each joint and axis. See for calculations chapter 3.3.2.

4.4 Visualizing the worst-case-scenarios

The maximum values of $\Delta S_{xz,leg}$, $\Delta S_{yz,leg}$, $\Delta S_{xz,spine}$ and $\Delta S_{yz,spine}$ were found within the diagonally backwards, backwards and sideways reactive stepping animations, with values 132.14 cm^2, 34.65 cm^2, 133.09 cm^2 and 119.53 cm^2 respectively. See for the axis definitions Appendix VI. Figure 17 show the armatures of the IK and MOCAP character in Blender during these worst-case scenario's.

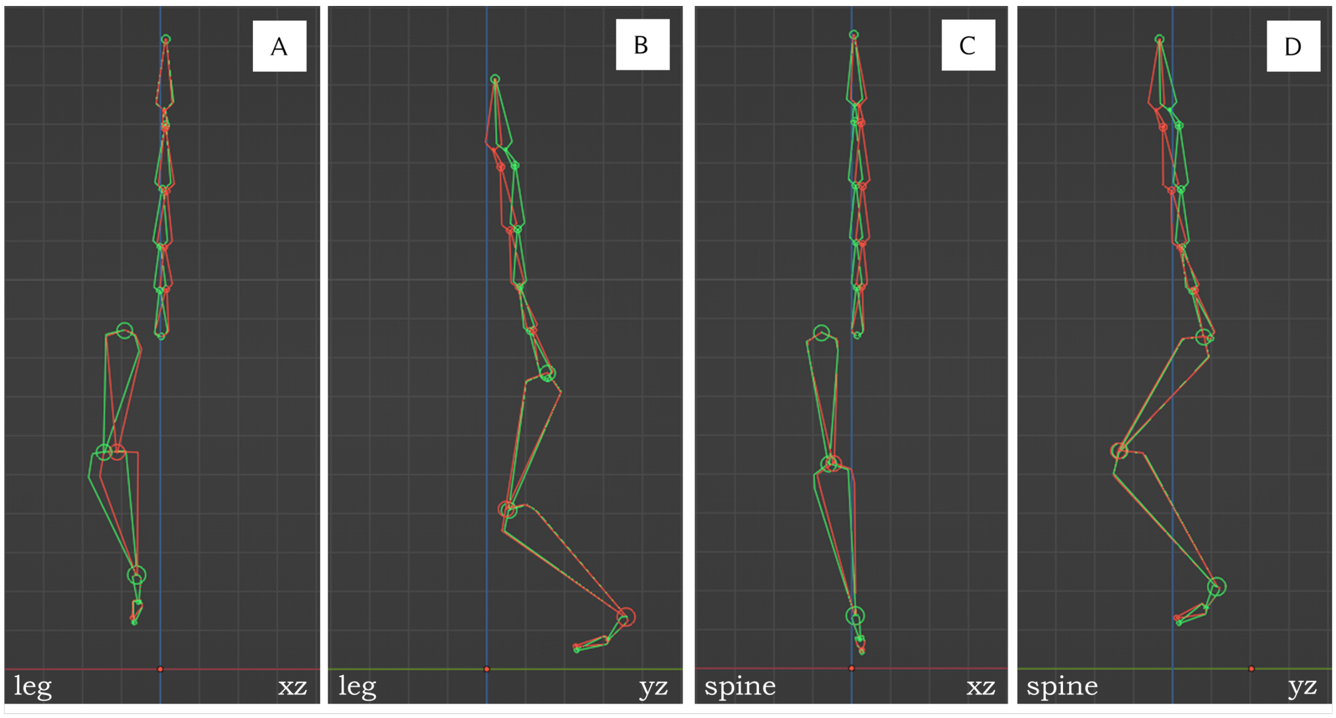


Figure 17: Images taken from the MOCAP (red) and IK (green) armatures in Blender (in wire mode) to show the worst-case-scenarios. A) Shows the worst-case scenario for the leg bone chain in the xz-plane, as found in a diagonally backwards stepping animation. B) Shows the worst-case scenario for the leg bone chain in the yz-plane, as found in a backwards stepping animation. C) Shows the worst-case scenario for the spine bone chain in the xz-plane, as found in a sideways stepping animation. D) Shows the worst-case scenario for the spine bone chain in the yz-plane, as found in a sideways stepping animation.

5 Discussion

The outcomes of this research have provided insight in the quality of the inverse kinematics methods used by the Blender animation software when applied to animating reactive stepping motions. Overall, the IK performed well in the leg chain, with a mean RMSE of the *knee* joint between 1.60 cm and 2.26 cm and a mean correlation coefficient between 0.68 and 0.96 for the *x-axis*, and a low mean RMSE and high correlation for both the y- and z-axis. The images taken of the MOCAP and IK armatures at the moment of worst-case scenario's show a small difference for the knee joint of the IK armature when compared to that of the MOCAP armature in the yz-plane and a small difference in the xz-plane.

As the error carries through the chain towards the end-effector, the errors are generally the largest in the joints closer to the target joint. It is therefore not surprising that the IK performed less well in the (upper part of the) spine bone chain, as it also contained more bones. Where the mean RMSE is generally low (< 1 cm) and mean correlation coefficient is high (> 0.8) for the spine joints in the x- and z-direction, the joints in the upper part of the spine (*upper-spine*, *neck* and *head* joints) show bad correlation between the MOCAP and IK armatures in the y-direction for the backwards, diagonally backwards and sideways stepping movements (mean RMSE: 1.29-2.92 cm, mean CC: 0.35-0.81). This corresponds to the line graphs of Δy_r. The difference in posture of the spine between the IK and MOCAP armature in the worst-case scenario's is noticeable, but not very large, in both xz- and yz-plane.

When both armatures are reapplied to the character, the differences are less noticeable as the error is 'hidden' partly in the mesh enveloping the armatures. On its own, the IK character's posture looks natural throughout the reactive stepping animations. When put into practice, the character animation is seen by the player from far away, as they stand in front of the screen. Depending on the size of the screen, the differences may become negligible from that distance. The inverse kinematics method is therefore expected to provide good enough posture to use in the exergame environment.

As the specific IK method used by Blender is unknown, it is hard to compare the results to relatable research. Given the comparison between MOCAP and the FABRIK, CCD, Jacobian transposed and DLS methods for human poses[3], the Blender inverse kinematics method seems to show slightly worse posture quality than the FABRIK and CCD methods and better than the Jacobian transposed and DLS methods [9].

As mentioned under 'general observations' in Results (see chapter 4.1), the snapping method shows a small offset between the empty object's location and the joint's true location. As the offset is always the same over the IK and MOCAP armatures for the same joint, it has no effect on Δx, Δy, Δz, RMSE or CC. Yet, the offset is different in magnitude and direction throughout the animation, between the different animations and between the joints, which does influence Δx_r, Δy_r, Δz_r and ΔS. Furthermore, it makes comparisons between the the joints less accurate. However, as the offsets is estimated as < 2 mm in length (hypotenuse), its influence on these parameters is limited. No corrections were therefore applied to these calculations. One way to work around this problem is to perform coordinate transformations instead of using the snapping method. By using coordinate transformations, you can find a joint's location expressed in the global coordinate system by using the joint's location expressed in its local coordinate system and the definitions of that coordinate system.

[3]FABRIK: A fast, iterative solver for the Inverse Kinematics problem, chapter 5.2.

Another observation is that the Δx, Δy, Δz of the *ankle* are larger then those of the *hip, spine* and *head tip*. Although it is small enough (< 0.003 cm) to fulfil the IK criteria, it is worth mentioning. Perhaps the multiple bone constraints around the joint (on the shin, IK target and foot bones) and/or the unusual bone hierarchy could have interfered with each other.

As mentioned in the Method chapter (3.2.1), the placement of the pole bone can influence the way the bone chain bends. If it is placed too close to the armature, it presents in weird spine or leg posture or it makes the chains bend the other way, as there is no 'room' for proper bending. The pole bones where therefore placed well outside of that zone. The pole bones also control the general direction in which the bone chain bends, so this may feel like it is 'cheating the system'. However, the pole bone acts similar to rotation limits on joints used by other IK methods: it does not force the knee to point exactly to its location, but more to its general direction. The existence of a Δx, Δy, Δz in the knee joint proofs this too.

Noticeably, the *knee* joint and the joints in the upper part of the spine showed lower correlation and higher $RMSE$ between the IK and MOCAP armatures for the backwards, diagonally backwards and sideways stepping movements compared to the forwards and diagonally forwards stepping movements. However, as the mean RMSE and correlation coefficients were taken over a maximum of 3 datasets per stepping direction this could be down to chance. Further investigation is required to see if the same effect can by found for larger datasets.

Furthermore, the knee joint showed higher RMSE and lower correlation between the MOCAP and IK armatures in the x-coordinates compared to the y-coordinates. This is expected for an IK chain with only two bones. Given that the x- and y-coordinates of the joint at the start and end of the chain are restricted and the bone lengths can not change, the y-coordinate of the knee joint is restricted too. However, the knee can be freely moved along the x-axis without causing bone deformation or changing the position of the start- and end of the bone chain. For the joints in the upper part of the spine chain, it showed the opposite behaviour. As there are more bones in the spine IK chain, movement along the y-axis of the intermediate joints in the IK is less restricted than in the knee joint, while rotation of the bones around the z-axis is more restricted in the spine bone chain then in the leg bone chain. However, as the data is only recorded in duple or triplo, it can not be concluded that the correlation between the IK and MOCAP armatures in the x-coordinates for the upper spine joints is always lower than that in the y-direction. Further research can study if this behaviour can be found for larger datasets as well.

This research focused on reactive stepping movements only. Results can differ when the motion is changed, for example, to running or jumping. It only represents the workings of the IK methods used by Blender, not those of other animation software. It would be interesting to follow up on this research and see how the results would compare to IK methods used by other animation software, such as the *final IK* extension in Unity. It is known that for animating humanoid characters using IK methods, there is always a trade-off between quality of posture and high computation. Another follow-up research could therefore study the difference in computation time of Blender's IK method compared to other IK methods, which is especially important for games using real-time input.

It was found that the upper part of the spine bone chain performed the worst. One way of improving the spine posture could be to separate the head from the spine IK to shorten the length of the IK chain. The rotation between the head and neck could then be limited using rotational constraints. Perhaps using spline IK instead of standard IK could give even better posture.

References

1. Qureshi AI, Baskett WI, Huang W, Shyu D, Myers D, Raju M, Lobanova I, Suri MFK, Naqvi SH, French BR, et al. Acute ischemic stroke and COVID-19: an analysis of 27 676 patients. Stroke 2021; 52:905–12
2. Feigin VL, Brainin M, Norrving B, Martins S, Sacco RL, Hacke W, Fisher M, Pandian J, and Lindsay P. World Stroke Organization (WSO): global stroke fact sheet 2022. International Journal of Stroke 2022; 17:18–29
3. Li N, Li J, Gao T, Wang D, Du Y, and Zhao X. Gait and balance disorder in patients with transient ischemic attack or minor stroke. Neuropsychiatric disease and treatment 2021; 17:305
4. Inness EL, Mansfield A, Lakhani B, Bayley M, and McIlroy WE. Impaired reactive stepping among patients ready for discharge from inpatient stroke rehabilitation. Physical therapy 2014; 94:1755–64
5. Ramnemark A, Nyberg L, Borssén B, Olsson T, and Gustafson Y. Fractures after stroke. Osteoporosis International 1998; 8:92–5
6. Langhorne P, Wagenaar R, and Partridge C. Physiotherapy after stroke: more is better? Physiotherapy Research International 1996; 1:75–88
7. HEROES of the sea: home-based exergame for enhancing resistance to falls after stroke. Available from: `https://www.4tu.nl/health/poster-gallery/posters/03/` [Accessed on: 2023 Jun 1]
8. Nieuwenhuys WW. The comparison of the accuracy of camera-based skeleton trackers during the stepping response for recovering balance in a home based environment. MA thesis. University of Twente, 2022
9. Aristidou A and Lasenby J. FABRIK: A fast, iterative solver for the Inverse Kinematics problem. Graphical Models 2011; 73:243–60
10. Buss SR. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. IEEE Journal of Robotics and Automation 2004; 17:16
11. Wolovich WA and Elliott H. A computational technique for inverse kinematics. *The 23rd IEEE Conference on Decision and Control.* IEEE. 1984 :1359–63
12. Whitney DE. Resolved motion rate control of manipulators and human prostheses. IEEE Transactions on man-machine systems 1969; 10:47–53
13. Phuoc LM, Martinet P, Lee S, and Kim H. Damped least square based genetic algorithm with Gaussian distribution of damping factor for singularity-robust inverse kinematics. Journal of Mechanical Science and Technology 2008; 22:1330–8
14. Buss SR and Kim JS. Selectively damped least squares for inverse kinematics. Journal of Graphics tools 2005; 10:37–49
15. Wang LC and Chen CC. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. IEEE Transactions on Robotics and Automation 1991; 7:489–99
16. Song W and Hu G. A fast inverse kinematics algorithm for joint animation. Procedia Engineering 2011; 24:350–4
17. Shin HJ, Lee J, Shin SY, and Gleicher M. Computer puppetry: An importance-based approach. ACM Transactions on Graphics (TOG) 2001; 20:67–94
18. Kulpa R and Multon F. Fast inverse kinematics and kinetics solver for human-like figures. *5th IEEE-RAS International Conference on Humanoid Robots, 2005.* IEEE. 2005 :38–43

19. Muller-Cajar R and Mukundan R. Triangualation-a new algorithm for inverse kinematics. 2007

20. Grochow K, Martin SL, Hertzmann A, and Popović Z. Style-based inverse kinematics. *ACM SIGGRAPH 2004 Papers.* 2004 :522–31

21. Blender 3.3 Manual. Available from: `https://docs.blender.org/manual/en/latest/index.html` [Accessed on: 2022 Oct 5]

22. Unity Manual - inverse kinematics. Available from: `https://docs.unity3d.com/Manual/InverseKinematics.html` [Accessed on: 2022 Nov 28]

23. Working with Animation Rigging. Available from: `https://learn.unity.com/tutorial/working-with-animation-rigging` [Accessed on: 2022 Nov 28]

24. Unity Manual - animation rigging: constraint components. Available from: `https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.1/manual/ConstraintComponents.html` [Accessed on: 2022 Nov 28]

25. Parger M, Mueller JH, Schmalstieg D, and Steinberger M. Human upper-body inverse kinematics for increased embodiment in consumer-grade virtual reality. *Proceedings of the 24th ACM symposium on virtual reality software and technology.* 2018 :1–10

26. Kallmann M. Analytical inverse kinematics with body posture control. Computer animation and virtual worlds 2008; 19:79–91

27. Hoyet L, Ryall K, McDonnell R, and O'Sullivan C. Sleight of hand: perception of finger motion from reduced marker sets. *Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games.* 2012 :79–86

28. Aristidou A and Lasenby J. Motion capture with constrained inverse kinematics for real-time hand tracking. *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP).* IEEE. 2010 :1–5

29. Peinado M, Herbelin B, Wanderley M, Le Callennec B, Boulic B, and Thalmann D. Towards configurable motion capture with prioritized inverse kinematics. Tech. rep. 2004

30. Komura T, Ho ES, and Lau RW. Animating reactive motion using momentum-based inverse kinematics. Computer Animation and Virtual Worlds 2005; 16:213–23

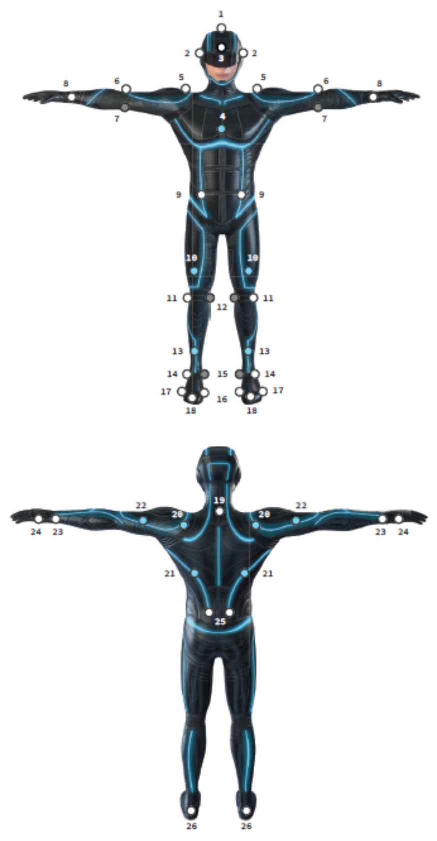
31. Qualisys Track Manager manual. Available from: `https://docs.qualisys.com/getting-started/content/getting_started/introduction.htm` [Accessed on: 2022 Dec 1]

Appendices

Appendix I: Marker placement used during motion capture

The following marker placement was used during the MOCAP session. This marker set is specific for the Qualisys Track Manager software.

Detailed marker description



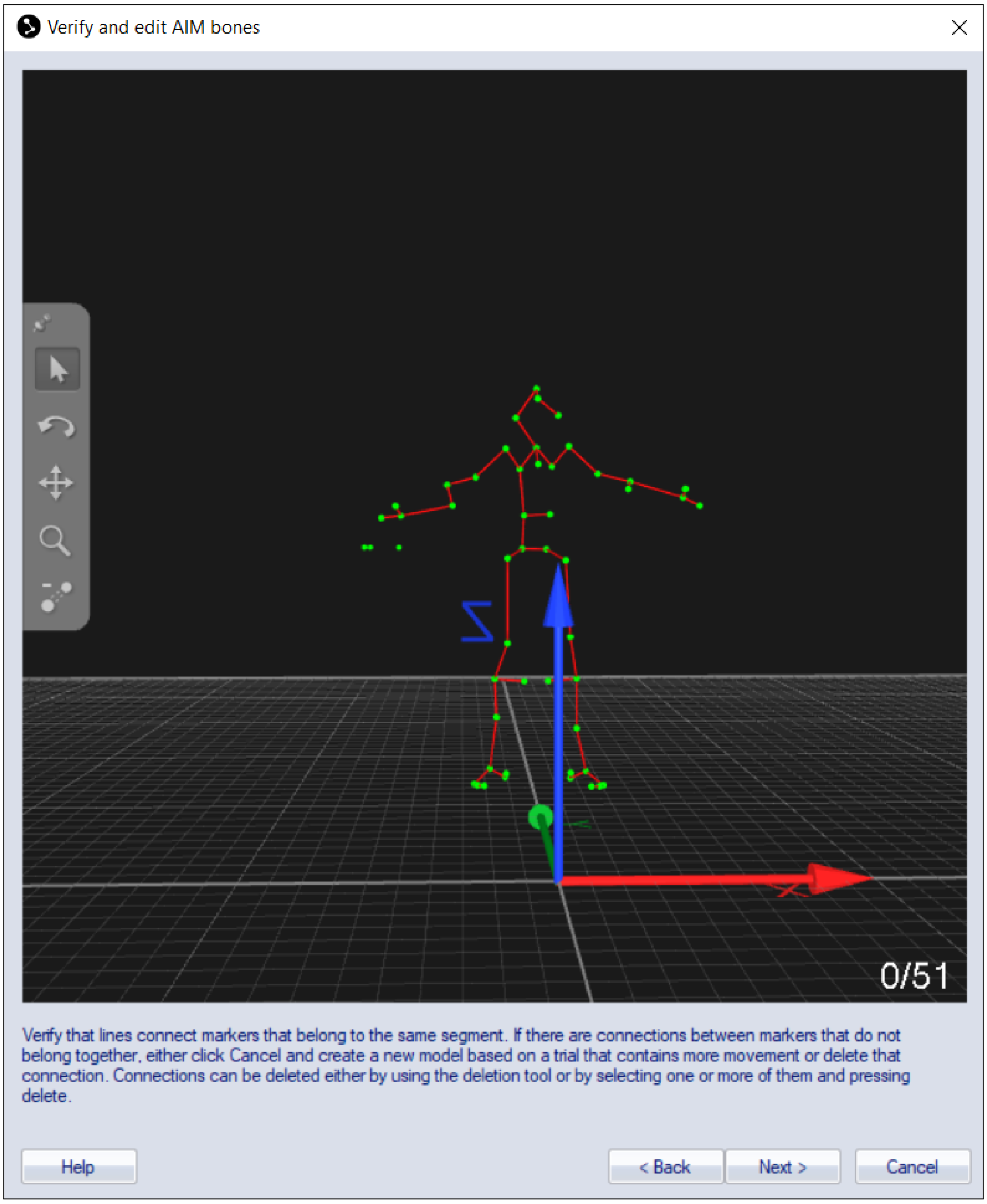
	Display name	Location	Ref[1]	Movable
1	HeadTop	On top of the head vertically above the ears.	-	-
2	HeadL, HeadR	Just above the ear centre.	-	-
3	HeadFront	Forehead, above the nose.	SGL	-
4	Chest	Middle part of the sternum.	-	On the sternum
5	LShoulderTop, RShoulderTop	On top of the shoulder (bony prominence).	SAE	-
6	LElbowOut, RElbowOut	On the outside of the elbow (bony prominence).	HLE	-
7	LElbowIn, RElbowIn OPTIONAL	On the inside of the elbow (bony prominence).	HME	
8	LWristIn, RWristIn	On the inside of the wrist (thumb side, bony prominence).	RSP	-
9	WaistLFront, WaistRFront	On the front of the pelvis (bony prominence).	IAS	-
10	LThigh, RThigh	Above the kneecap.	-	On the thigh
11	LKneeOut, RKneeOut	On the outside of the knee (bony prominence).	FLE	-
12	LKneeIn, RKneeIn OPTIONAL	On the inside of the knee (bony prominence).	FME	-
13	LShin, RShin	Front of the shin.	-	On the shin
14	LAnkleOut, RAnkleOut	On the outside of the ankle.	FAL	-
15	LAnkleIn, RAnkleIn OPTIONAL	On the inside of the ankle.	FAM	
16	LForefootIn, RForefootIn	Base of the toes on the inside.	FM1	-
17	LForefootOut, RForefootOut	Base of the toes on the outside.	FM5	-
18	LToeTip, RToeTip	Top of the foot/shoe tip.	-	-
19	SpineTop	Biggest prominence on the spine.	C7	-
20	LShoulderBack, RShoulderBack	Upper part of the shoulder blade.	SAA	On the shoulder blade
21	BackL, BackR	Below the shoulder blade.	SIA	Below the shoulder blade
22	LArm, RArm	Back of the arm.	-	On the arm
23	LWristOut, RWristOut	On the outside of the wrist (pinky side, bony prominence).	USP	-
24	LHandOut, RHandOut	Base of the fingers on the outside.	HM5	-
25	WaistLBack, WaistRBack	On the back of the pelvis (bony prominence).	IPS	-
26	LHeelBack, RHeelBack	Back of the heel.	FCC	-

FRONT
BACK

Figure 18: Schematic representation and description of the marker placement used during the MOCAP session, as provided by the Qualisys Track Manager software in the installment.

Appendix II: AIM-model generation details

After generating a new AIM-model from the AIM-trial, a window pops up allowing verification of the bones in the AIM-model, see figure 19. There are no connections going across skeleton segments, so no 'bones' (red lines) needed to be deleted.



Verify and edit AIM bones

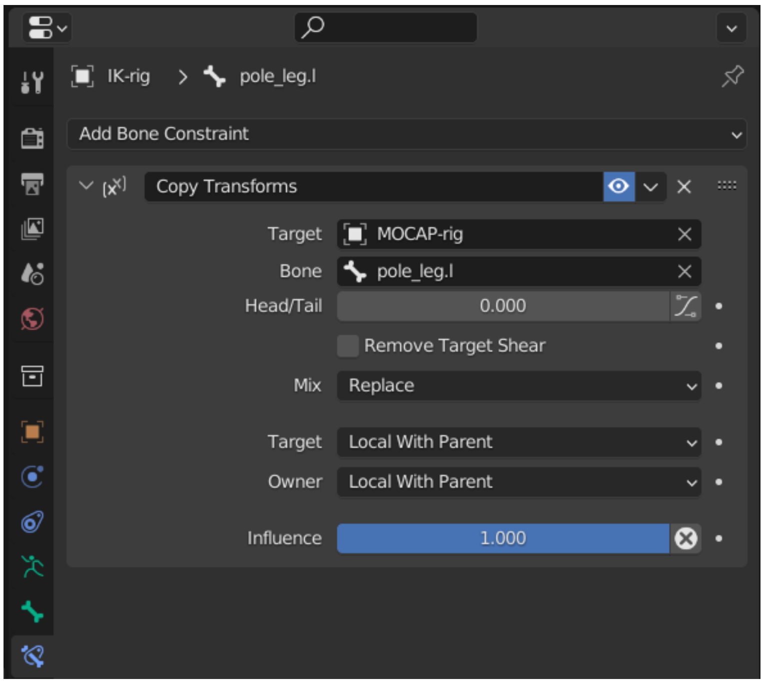
0/51

Verify that lines connect markers that belong to the same segment. If there are connections between markers that do not belong together, either click Cancel and create a new model based on a trial that contains more movement or delete that connection. Connections can be deleted either by using the deletion tool or by selecting one or more of them and pressing delete.

Figure 19: Verification of the AIM-model's bones upon generating an AIM model from 48 animation markers of the processed *Aim03.c3d* file.

Appendix III: Settings used in the copy transform bone constraint

The *copy transform* bone constraint in Blender can be used to copy an object's transform (position and rotation) to another object. In this project it is used to obtain the IK-based animation. The constraint can be found in the *bone constraint properties* menu. It takes multiple inputs. The *target* and *bone* input requires the rig and bone whose transform needs to be copied. With the *head/tail* input you can set the copied movement of the bone to the head or tail of the owner. The *target* and *owner* inputs determine how the owner will move with respect to the target. The input settings used in this project are shown in figure 20.



IK-rig > pole_leg.l

Add Bone Constraint

Copy Transforms

Target	MOCAP-rig
Bone	pole_leg.l
Head/Tail	0.000
	Remove Target Shear
Mix	Replace
Target	Local With Parent
Owner	Local With Parent
Influence	1.000

Figure 20: An overview of the inputs of the copy transform bone constraint in Blender

Appendix IV: A summary of the snapping method in Blender and how it was used to obtain the joint data.

The locations of the joints in the armature can not be obtained directly from Blender. Instead, by placing an empty object at the exact same place as the joint, the location of the joint can be found from the location of that empty. A method called *snapping* was used to place an empty object at a bone's tail. The following list shows a summary of the snapping method:

1. An empty (sphere) object must be added into the scene first.
2. The interaction mode must be set to *pose-mode* with the armature of choice selected to be able to select an individual bone. With the bone selected, the cursor is snapped to the bone's **head** using: *pose > snap > cursor to selected.*
3. The interaction mode must be set back to *object-mode* to select the empty object. The object is then snapped to the cursor using: *object > snap > selected to cursor.*
4. The empty object is now placed at the exact same position as the bone's head and it's global transform is shown in the transform menu.

A Python code was written, that loops through each bone in the armature and snaps an empty object (sharing the name of the bone) to a bone's head and places its global position data in a location matrix. The code then repeats these steps for each joint and at each keyframe in the animation. Finally, it prints the results to the console window in a way it can be easily copied to an Excel file.

Before using the *snapping* code, a new collection was created in the ViewLayer called *Joint_objects*. A separate script called *make_empties* was used to create the set of empty spheres with a radius of $0.05m$ and rename them to each of the bones in the armature.

To obtain the location data, a MOCAP-based animation was linked to the MOCAP character's armature and the keyframes of the start (foot-off) and end (foot-on) of the reactive step were entered at the top of the *snapping.py* code and the active rig was set to MOCAP-rig by commenting out the *rig = 'IK-rig'* line. The code was executed and the data was copied from the system console to an Excel file. Next, the active rig was set to IK-rig by commenting out, *rig = 'MOCAP-rig'* instead. The script was executed again and the results were copied to a new worksheet in the Excel file. All [,] and $'$ symbols that were present in the console window were removed from the data in the Excel sheet.

The result of the Python code is an Excel file containing 2×14 sheets: four movements were recorded in triplo and one in duplo (see chapter 3.1.1 and 3.1.2), and for each recorded movement the location data of the joints in both IK and MOCAP character were obtained. Each sheet consisted of a header in row 1 and the location data was spread over a total of 44 columns: 4 columns per bone (keyframe and x-, y- and z-positions) for 11 bones in total (6 spine bones, 3 leg bones and the extra head and foot bone). The number of rows depended on the amount of keyframes defining the reactive stepping movement in the animation. The location data was expressed in meters.

Appendix V: The joint naming system and hierarchy used by Blender and an overview of the bone lengths.

The bone hierarchy in Blender is determined by the parent-daughter connections. It usually follows the chain in head-to-tail order. The hierarchy in the leg and spine bone chains is therefore as follows: *Hip > Knee > Ankle > Foot > Toe; Spine > Spine001 > Spine002 > Upper spine > Neck > Head > Head tip.* Figure 21 shows the position of each of these joints in the character's armature.

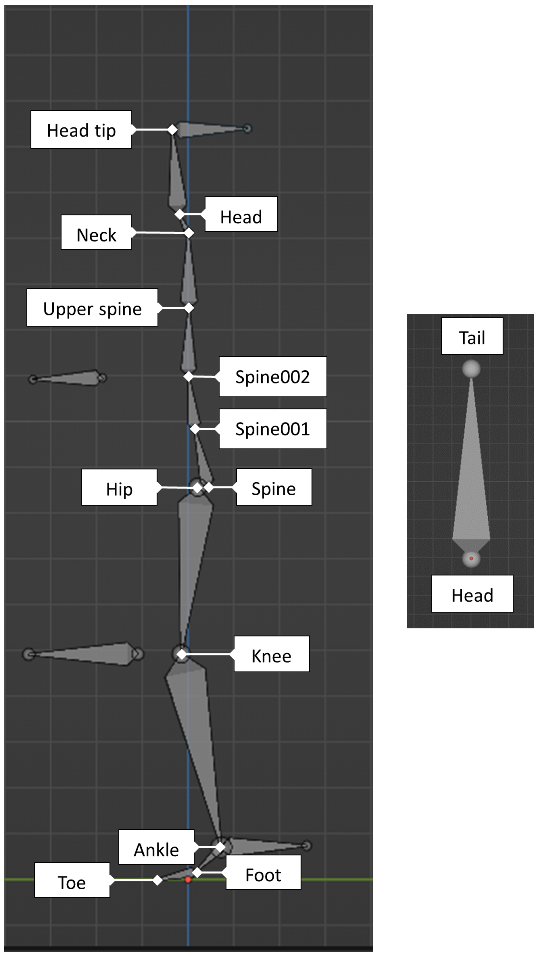


Figure 21: Image of the character's armature in Blender, showing the names of each joint.

The following table presents an overview of the length of each (deform) bone in the character's armature and the joint they control.

Bone List		
Bone name	*Associated joint*	*Bone length (cm)*
Foot	Foot	7.64
Shin	Ankle	43.16
Thigh	Knee	36.58
Root/1st spine	Spine001	13.00
2nd spine	Spine002	11.67
3rd spine	Upper spine	14.76
4th spine	Neck	16.45
Neck	Head	4.75
Head	Head tip	18.31

Appendix VI: An overview of the axis definitions in Blender.

The axis were defined in Blender as follows:

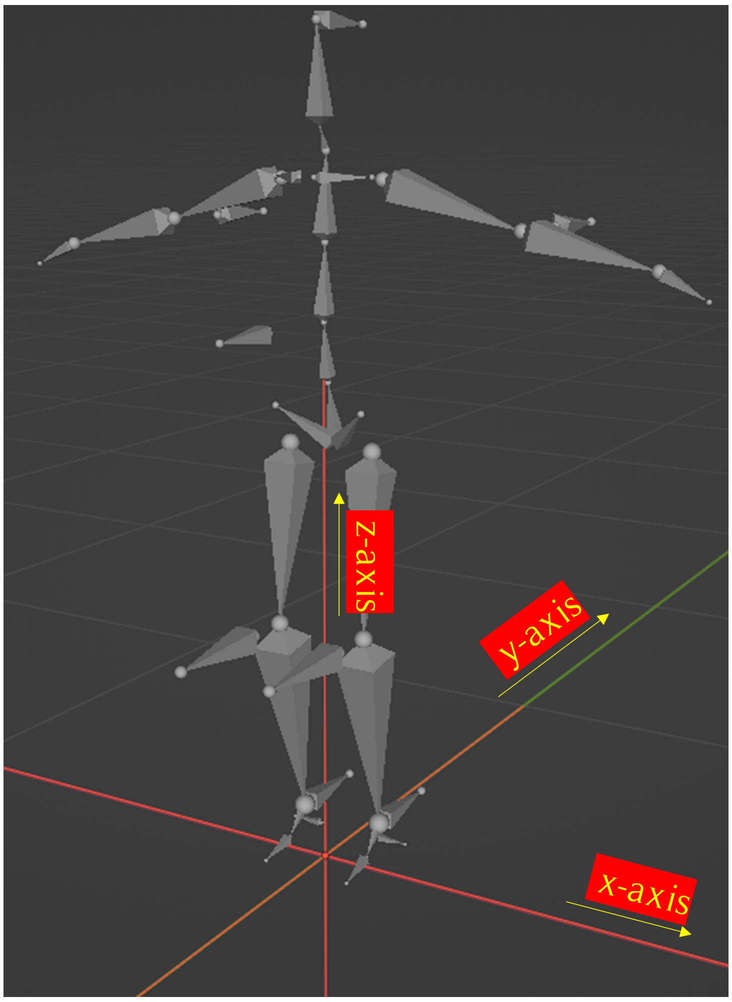
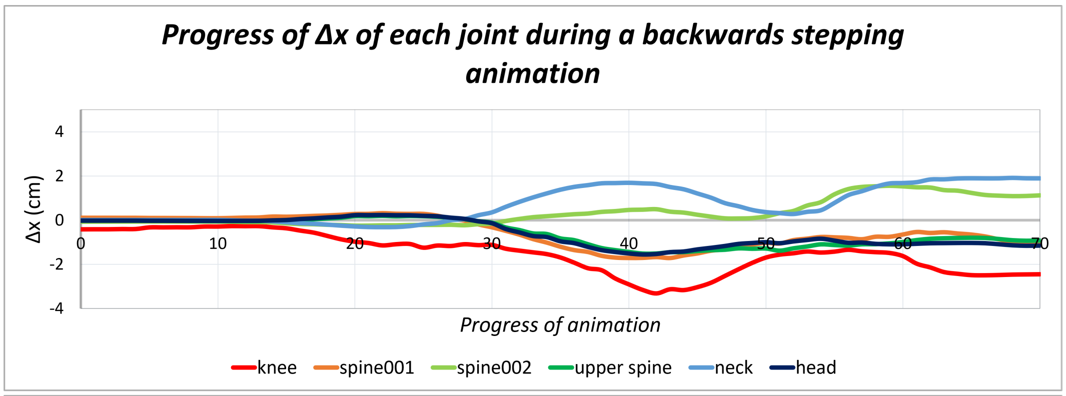
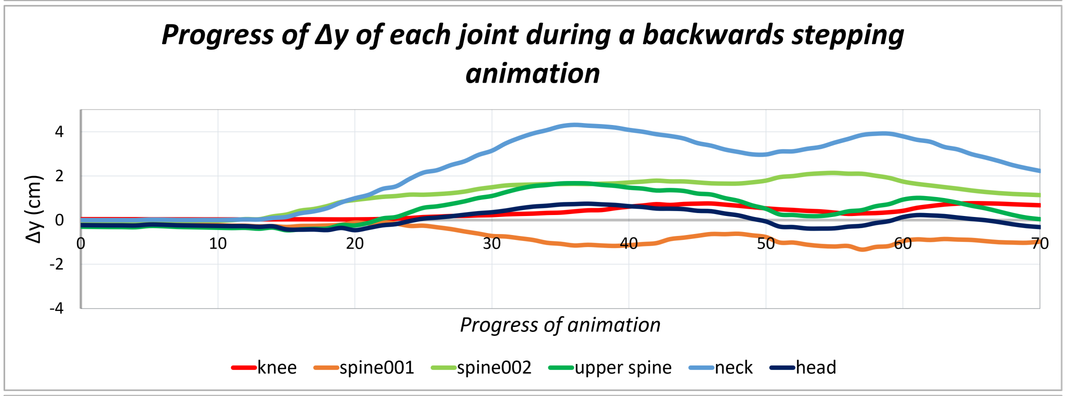


Figure 22: An image showing the axis definitions as defined in the Blender files used during this project.

Appendix VI: Lineplot of Δx_r, Δy_r and Δz_r for a backwards stepping movement.

The following line graphs show the progress of Δx_r, Δy_r and Δz_r throughout the backwards stepping movement. It shows that Δz_r is smaller than Δx_r and Δy_r for all joints. The $|\Delta x_r|$ of the *knee* joint is larger then any of the spine joints, during peak movement. Furthermore, $\Delta x_{r,knee}$ stays negative, while $\Delta y_{r,knee}$ stays positive. This is representative for all stepping movements. The other joints did not show a similar behaviour for all movement. The *neck* joint showed the largest $|\Delta y_r|$ of all spine joints throughout all animations, followed by the *upper-spine* joint.





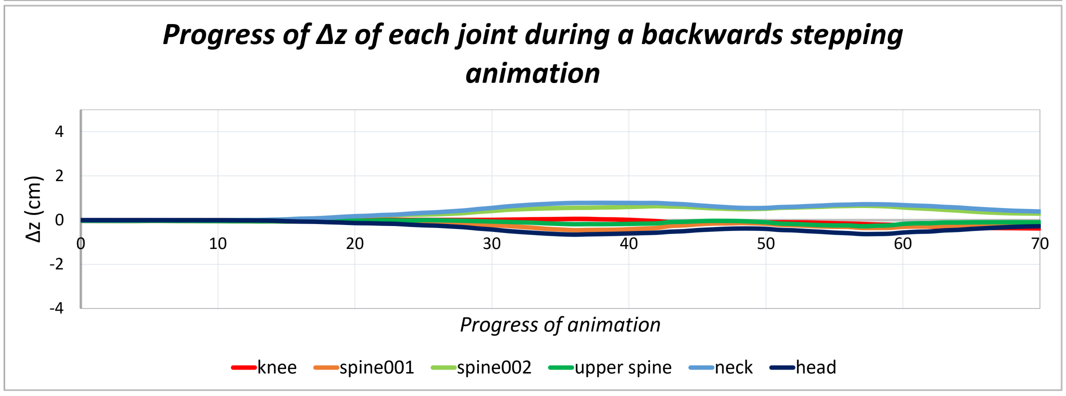


Figure 23: Three graphs showing the progress of Δx (above), Δy (middle) and Δz (below) for each joint during the backwards stepping animation. A negative value means the joint in the IK armature is positioned towards the negative axis with respect to the same joint in the MOCAP character. *These graphs show the data of only one of the backwards stepping animations (backwards.04)*