Inverse Kinematics for Arm Pose Reconstruction in Virtual Reality

Erik Stolpe

Stockholm University

Bachelor's Thesis in Computer Science
at Stockholm University, Sweden, 2021

Inverse Kinematics for Arm Pose Reconstruction in Virtual Reality

Erik Stolpe

Bachelor's Thesis in Computer Science (15 ECTS credits)
Bachelor's Programme in Computer Science
Stockholm University year 2021
Supervisor at the Department of Mathematics was Woosok Moon
Examiner was Lars Arvestad

Department of Mathematics
Stockholm University
SE-106 91 Stockholm, Sweden

Abstract

Immersion is a key property of virtual reality technology. It has previously been shown that providing the user with a virtual avatar inside VR applications can increase their embodiment and therefore immersion. However, most consumer grade VR only includes tracking hardware for head and hands, making user reconstructions of the upper body non trivial, and accurate full body reconstructions not possible. In this paper we examine how two heuristic inverse kinematics methods, forward and backward reaching inverse kinematics, and cyclic coordinate descent, together with a pole target for the elbow, can be used for arm reconstruction using a VR headset and two VR hand controllers. Both methods are run on an arm model with three joints, and compared with respect to their computational speed and ability to replicate user arm poses. The results show that forward and backward reaching inverse kinematics is the faster method, but that both methods are fast enough for real time applications, and perform very similar in regards to pose recreation when paired with a pole target. Both methods are able to recreate observably accurate arm poses, but the result is dependent on the pole position. Without a pole, most recreations do not match the user. Based on the results, we believe both methods can be used for arm reconstruction, but suggest using the forward and backward reaching inverse kinematics method since it is faster.

Inverterad kinematik för armrekonstruktion i virtuell verklighet

Sammanfattning

Immersion är en nyckelegenskap hos virtual reality-teknologi. Det har tidigare visats att genom att förse användaren med en virtuell avatar i VR-applikationer kan man öka känslan av kroppslig närvaro och därmed immersionen. De flesta VR-konfigurationer av konsumentkvalitet har dock bara spårningshårdvara för huvud och händer, vilket gör rekonstruktioner av användarens överkropp icketrivial, och noggranna helkroppsrekonstruktioner ej möjliga. I det här arbetet undersöker vi hur två heuristiska inverterad kinematikmetoder (*forward and backward reaching inverse kinematics* respektive *cyclic coordinate descent*), tillsammans med en referenspunkt för att styra armbågens position, kan användas för armrekonstruktion med VR-headset och två VR-handkontroller. Båda metoderna körs på en armmodell med tre leder och jämförs med avseende på deras beräkningshastighet och förmåga att replikera användarens armpositioner. Resultaten visar att forward and backward reaching inverse kinematics är den snabbare metoden, men att båda metoderna är tillräckligt snabba för realtidsapplikationer och fungerar mycket lika när det gäller armrekonstruktion med referenspunkt. Båda metoderna lyckas återskapa observerbart noggranna armpositioner, men resultatet är beroende av referenspunktens position. Utan en referenspunkt matchar de flesta rekonstruktioner inte användaren. Baserat på resultaten tror vi att båda metoderna kan användas för armrekonstruktion, men föreslår användning av forward and backward reaching inverse kinematics eftersom den är snabbare.

Acknowledgements

I would like to thank my supervisor Woosok Moon for supporting my chosen project idea, and for advising on how to work with and plan this kind of project. I also wish to thank my family and friends for all support throughout the work.

Contents

1 Introduction

The market for Virtual Reality (VR) has seen a steady increase in size during recent years as headsets get cheaper, more consumer friendly and offer higher performance. Since the industry for VR games and experiences is relatively new, and is not yet a mainstream technology adopted by the computer entertainment audience, there have been few triple-A titles. A majority of popular titles are developed by smaller studios. Accordingly, one could speculate that common standards in VR experiences have yet to be established.

One implementation detail that differs between games or applications is how the user views their own virtual body. A common approach is to make most of the user body invisible to the user, and only show a pair of floating hands which are used to interact with the virtual environment. The floating hands are occasionally accompanied with a floating torso. Physical hand controllers tracked in real time are included in most VR systems. Therefore, the floating hands are easy to implement and can naturally follow the users own hand movements. However, a more comprehensive virtual body is essential to increase user immersion. Due to the lack of other tracked points on the body, recreating and animating naturally moving body parts other than the hands requires sophisticated mathematical methods.

Given the position and rotation of a VR hand controller, and a model of the human arm consisting of limb segments and joints connected to each other. It is possible to calculate the parameters of the elbow and shoulder joints that result in the hand controller position. The mathematical process is called Inverse Kinematics (IK). Elbow and shoulder joint parameters can in turn be used to recreate the arm pose. Therefore, by continuously solving the IK, the arm can be animated in a realistic fashion. In this paper we examine two commonly used IK algorithms and test them individually, and on a VR setup.

1.1 Problem description

Inverse kinematics is a mathematical process used to calculate the parameters needed to place the end of a kinematic chain in a desired position. There exist two distinct groups of methods used to solve IK problems, analytical and iterative. Analytical solutions are closed form and can be significantly faster than iterative methods. However, they are not suitable for computer animation due to complex geometry for long kinematic chains. Iterative methods

start with an initial guess or kinematic chain configuration and calculates an improving approximate solution every iteration until they converge.

We want to investigate how IK methods can be used to enhance the embodiment of a virtual reality user by implementing realistically moving arms. Two commonly used heuristic iterative methods, Forwards and Backwards Reaching Inverse Kinematics (FABRIK) (Aristidou & Lasenby 2011) and Cyclic Coordinate Descent (CCD) (Wang & Chen 1991), are to be implemented in the Unity3D game engine. Both methods are computationally fast making them well suited for real time applications. In addition, some extension to improve the realism of the arm movement should be considered.

A simple kinematic chain representing a human arm should be implemented in Unity3D with the help of in-engine parent child relationships. Both methods need to be able to run in real time on the kinematic chain with the targets being the tracked VR hand controllers. The shoulder positions can be offset from the tracked headset position. Tracking of the VR hardware is made possible by Unity support for VR development.

Furthermore, the performance of the methods should be compared based on specified testing scenarios and the results discussed with relevant background theory. To evaluate performance, the calculation time, number of iterations, and error, need to be considered.

Finally, we aim to suggest the method most suited for VR experiences, based on individual performance, and how well the method followed the VR users arm pose while wearing both headset and hand controllers.

1.2 Research questions

Which method had the highest performance in regard to calculation time and convergence rate? Which method could best replicate the movements of the human arm, and produced least unnatural poses?

1.3 Delimitations

Implementations will be tested on an arm model with three joints. The Unity3D VR application will be a proof of concept, not containing any extra assets apart from the necessary scripts for the methods, and a simple model of the arm.

2 Background

This section of the report covers necessary background required to better understand the results and discussion. Specifically, it contains related work, a brief introduction to kinematics, forward kinematics, and inverse kinematics. As well as the theory behind each implemented method. As a prerequisite, the reader should have an understanding of linear algebra.

2.1 Related work

Efforts of examining ways to increase user immersion in virtual reality with the help of inverse kinematics have been done before. Human Upper-Body Inverse Kinematics for Increased Embodiment in Consumer-Grade Virtual Reality (Parger et al. 2018) investigates the viability of using the limited tracking options provided by most consumer grade VR setups (hand and head tracking) together with their own upper body inverse kinematics solution to increase embodiment in the virtual world. In their user study, the participants preferred their method to floating hands, but also to a motion capture system.

An analysis of the Jacobian inverse IK methods and how they can be used for full body reconstruction in VR was done by Caserman et al. (2019). To make a full body reconstruction they tracked the head, hands, feet, and hip using HTC Vive controllers and trackers. Additionally, a user study was conducted for subjective evaluation. The users found that they felt a high presence in the virtual world when using the recreated body.

In this paper we investigate how two other commonly used IK methods FABRIK and CCD can be used to recreate human arm poses in VR using only tracking data of head and hand positions. In addition to the base methods, a pole target is implemented and tested to see how it can be used for elbow positioning. Furthermore, the methods are compared based on computational performance. Prior performance review of known IK methods has been done by Aristidou & Lasenby (2009).

2.2 Kinematics

Kinematics is the study of the geometry of motion of points, bodies, and systems of points or bodies, without considering the mass and forces applied to those objects (Beggs 1983). In general, a kinematics problem involves

describing the structure of the system of objects, setting initial conditions and constraints, and then solving for specified parameters (position, rotation, etc) on some part(s) of the system.

2.2.1 Kinematic chain

Kinematics are applied to many different structures of bodies. This report focus on kinematic chains in a three dimensional space since that is what is used to model the arms in the application.

A kinematic chain consists of a finite number of joints and links connected in a hierarchical structure using parent-child relationships. Joints and links are rigid bodies, which means all points in the body keeps the same distance and angle in relation to each other at all times, i.e. the body is solid and can not deform (West 2015). Joints are constrained by the parent position and rotation such that if the parent joint translates or rotates, all child joints move accordingly.

A kinematic chain with $n \in \mathbb{N}$ joints can be represented with a set of three dimensional vectors $P = \{\vec{p_1}, \vec{p_2}, \ldots, \vec{p_n}\} \subset \mathbb{R}^3$, where $\vec{p_1}$ is called the root joint and $\vec{p_n}$ is called the end effector. The rest being called intermediate joints. A joint $\vec{p_i}$ is the child of $\vec{p_j}$ if $i = j + 1$ and a grandchild if $i > j + 1$, conversely $\vec{p_i}$ is the parent of $\vec{p_j}$ if $i = j - 1$ and a grandparent if $i < j - 1$, with i and j such that $p_i, p_j \in P$. The root joint is therefore a parent or grandparent of all other joints, and the end effector is a child or grandchild of all other joints. The length of any link between two joints in P is determined by distance between the joints at the links endpoints $||\vec{p_{i+1}} - \vec{p_i}||$, with $0 < i < n$. The links inward and outward joints are $\vec{p_i}$ and $\vec{p_{i+1}}$ respectively. The set of joint positions can be used to determine the pose of the kinematic chain, which is normally given as the set of joint rotations.

Figure 1 shows a representation of a simple two dimensional kinematic chain with four joints and its pose.

2.2.2 Degree of freedom

A Degree of Freedom (DoF) within kinematics refers to the ability to translate or rotate about an axis. In a three dimensional space, a joint can translate along three axis, and rotate about three axis, it therefore has six DoF. The number of DoF is dependent on how many dimensions the joint is in, and any constraints applied on the joint.

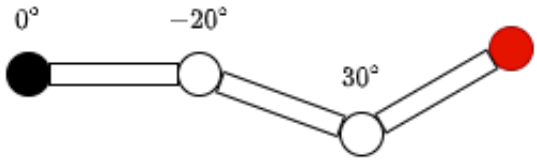


Figure 1. Example kinematic chain with four joints. The black joint represents the root and the red the end effector. The counterclockwise rotation from a horizontal axis is marked on each joint, making up the pose of the kinematic chain in this example.

2.2.3 Constraints

When modeling a real world scenario using kinematic chains, putting constraints on certain joints often becomes a necessity. A human arm for instance, is very limited in terms of how many degrees its individual joints (shoulder, elbow and wrist) can rotate about certain axis. The elbow only has two rotational DoFs. With rotational constraints on the relevant joints, this can be modeled more accurately.

2.2.4 Forward kinematics

A kinematic chain can be positioned using two opposite processes, Forward Kinematics (FK) and Inverse Kinematics (IK). In both processes, the lengths of each link connecting the joints, and the root joints position are given. Additionally, in FK the rotation of each joint is given. Using these known parameters, FK calculates the position of the end effector. Effectively, a FK function takes the root position, link lengths, and pose as input, and calculates the position of the end effector as its output. A commonly used convention for dealing with FK is the Denavit-Hartenberg convention (Kucuk & Bingul 2006).

2.2.5 Inverse kinematics

In many real world scenarios, the pose is unknown. What is really interesting is how we can configure the kinematic chain, such that the end effector gets placed at a specified target position. This is a common problem in robotics, and is also of interest in computer animation (Aristidou et al. 2018). Inverse kinematics solves this problem. An IK function is the inverse of a FK func-

tion, it takes the end effector position as input, and calculates the pose or joint positions as the output using an initial guess or joint configuration. Note that several joint configurations may result in the same end effector position, but the same IK function should always give the same output given a specific end effector position. Figure 2 shows an example of inverse kinematics on a simple chain.

There exist a variety of numerical solutions for IK. They mostly fall in two categories, Jacobian inverse methods and heuristic methods. An extensive review of existing methods was done by Aristidou & Lasenby in 2009. In the two following chapters two commonly used heuristic IK methods are examined, FABRIK and CCD.

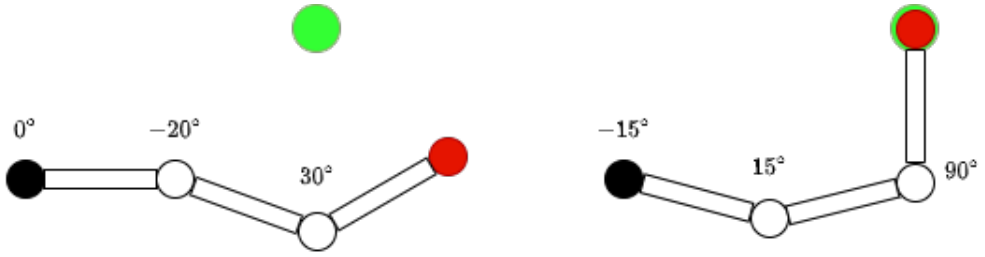


Figure 2. Example of IK. Before IK on the left, and after IK on the right. The black joint represents the root, and the red the end effector. The green disk is the target position.

2.3 Forward and backward reaching inverse kinematics

Forward and backward reaching inverse kinematics is an iterative method used to solve the IK problem (Aristidou & Lasenby 2011). The method uses simple calculations involving points, distances, and lines, to calculate the joint configuration. Computationally expensive operations such as matrix manipulation are not involved, making it fast, and avoiding problems such as matrix singularities. In an iterative fashion, FABRIK uses the joint positions of the previous iteration to adjust the kinematic chain such that the distance from the end effector to the target is reduced, while also keeping the proportions of the chain intact. A brief overview of the ideas and methodology involved in FABRIK is given before moving onto a formal description.

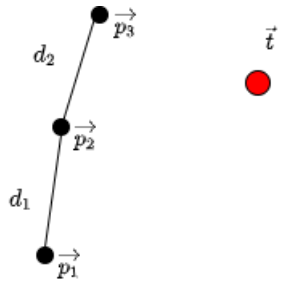
The method is composed of two stages throughout each iteration. A forward, and a backward stage. The forward stage starts at the end effector

and traverses the kinematic chain until it reaches the root joint, adjusting joint positions along the way. The backward stage moves in the opposite direction and is needed to correct unwanted movement of the root. Assuming a kinematic chain and a target position, we must first measure the distance to the target from the root. If the total length of the kinematic chain is shorter than the measured distance, the target is unreachable and there are no solutions. In this case, the kinematic chain is stretched towards the target. If the target is reachable, an iteration can begin, starting with the forward stage. The forward stage starts with updating the end effector position to have the same position as the target. Any given joint between end effector and root is adjusted according to the their childs new position (the previously adjusted joint), their current position, and the link length between them and their child given by the chain. Finally, the root joint is adjusted, and the forward stage ends. The movement of the root joint is undesirable, and is corrected in the backward stage. The backward stage begins with setting the root joint position back to its original position. It then adjusts the following joints in a similar way as the forward stage, until the end effector is reached and adjusted. Note that the forward stage set the end effector position to the same as the target position, but the backward stage then moved it, creating a distance between the target and the end effector (a consequence from correcting the root). The error tolerance for this distance has to be set by the user. The method iterates until the tolerance is reached, or until it reaches a maximum iteration limit set by the user. It can be shown that this distance does indeed decrease with each iteration. One iteration of FABRIK is presented in figure 3.

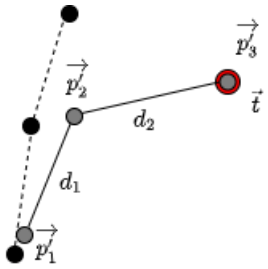
Continuing with the formal description. Assume a kinematic chain (as described in section 2.2.1) with an arbitrary set of joint positions $P = \{\vec{p_1}, \ldots, \vec{p_n}\}$ where $\vec{p_1}$ is the root position and $\vec{p_n}$ is the end effector, and a target position $\vec{t} \in \mathbb{R}^3$. To check if the target is reachable, we calculate the distance $d_i = ||\vec{p_{i+1}} - \vec{p_i}||$ between each pair of joints $\vec{p_i}$ and $\vec{p_{i+1}}$ for $i = 1, \ldots, n-1$. And sums the distances to get the maximum length of the chain,

$$chainlength = \sum_{1}^{n-1} d_i. \tag{1}$$

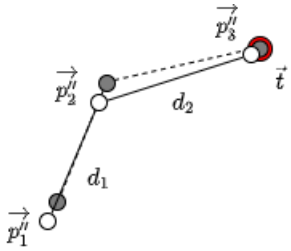
Now, we proceed to check if the maximum length of the chain is shorter than the distance from the root to the target,



(a) Initial positions of kinematic chain (black) and target (red). Links are black line segments.



(b) The forward stage. The end effector is placed at the same position as the target and the rest of the joints are positioned on lines passing through their old position and their children's new positions. New positions are gray. Old links are dashed lines.



(c) The backward stage. The root is repositioned at its original position and the rest of the joints are positioned as in (b) but in the opposite direction. New (final) positions are white. Old links are dashed lines.

Figure 3. Display of one iteration of forward and backwards stages of FABRIK on a 2D kinematic chain with 3 joints. The final configuration is shown in (c).

$$\sum_{1}^{n-1} d_i < ||\vec{t} - \vec{p_1}||. \tag{2}$$

If the inequality (2) is true, the forward stage begins. If it is false, the target is unreachable, and the chain is stretched towards the target. In any case, let $P' = \{\vec{p'_1}, \ldots, \vec{p'_n}\}$ be the set of new joint positions. In the case where (2) is *false*, these are; $\vec{p'_1} = \vec{p_1}$, and

$$\vec{p'_i} = \vec{p'_{i-1}} + d_{i-1}\frac{\vec{t} - \vec{p_1}}{||\vec{t} - \vec{p_1}||}, \tag{3}$$

for $i = 2, \ldots, n$. Note that $\vec{p'_i}$ is calculated using $\vec{p'_{i-1}}$, hence (3) has to be calculated recursively.

We continue with the case where the target is reachable. Then the forward stage begins. The forward stage starts at the end effector $\vec{p_n}$ and works through the joints of the kinematic chain until it reaches $\vec{p_1}$. It starts by letting the new end effector position be the same as the target position, $\vec{p'_n} = \vec{t}$. The next joint position to be set is the end effectors parent position $\vec{p'_{n-1}}$. Which is positioned d_{n-1} units from $\vec{p'_n}$ along the line passing through $\vec{p'_n}$ and $\vec{p_{n-1}}$. That is,

$$\vec{p'_{n-1}} = \vec{p'_n} + d_{n-1}\frac{\vec{p_{n-1}} - \vec{p'_n}}{||\vec{p_{n-1}} - \vec{p'_n}||}. \tag{4}$$

Figure 4 contains a visual representation of this step.

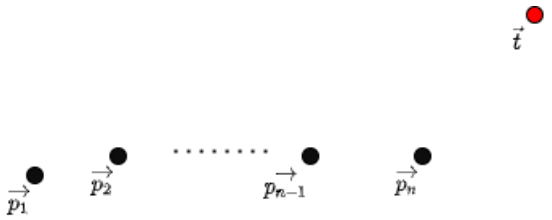
Naturally, the next position to be set is $\vec{p'_{n-2}}$, and the process continues until $\vec{p'_1}$ is reached. Consequently, in the forward stage, any joint position $\vec{p'_i}$ with $i = 1, \ldots, n-1$ is placed according to the (backwards) recurrence

$$\vec{p'_i} = \vec{p'_{i+1}} + d_i\frac{\vec{p_i} - \vec{p'_{i+1}}}{||\vec{p_i} - \vec{p'_{i+1}}||}, \tag{5}$$

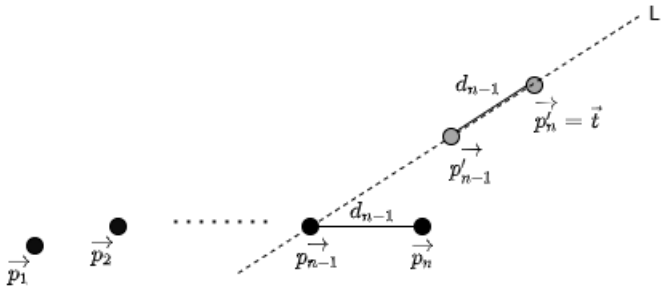
starting with $\vec{p'_i} = \vec{t}$ for $i = n$.

With $i = 1$ in (5) the position of the root joint is changed. The backwards stage is needed to reposition the root joint and move the rest of the joints accordingly.

The backward stage uses the same positioning strategy as the forward stage, but moves in the opposite direction, starting at the root joint and



(a) Initial positions of target $\vec{t}$ (red), and elements in P (black).



(b) End effector $\vec{p'_n}$ (gray) set to target position. End effector parent $\vec{p'_{n-1}}$ (gray) positioned along line L according to expression (4).

Figure 4. Visual representation of a typical situation in the forward stage.

finishing at the end effector. Let $P'' = \{\vec{p''_1}, \ldots, \vec{p''_n}\}$ be the final set of joint positions. First, the root is relocated to its starting position, $\vec{p''_1} = \vec{p_1}$. The next joint position, $\vec{p''_2}$, is then positioned d_1 units from $\vec{p''_1}$ along the line passing through $\vec{p''_1}$ and $\vec{p'_2}$. The rest of the joints are positioned in the same way. In a comparable way to the forward stage, we place the final joint positions according to the following recursion,

$$\vec{p''_i} = \vec{p''_{i-1}} + d_{i-1}\frac{\vec{p'_i} - \vec{p''_{i-1}}}{||\vec{p'_i} - \vec{p''_{i-1}}||}, \tag{6}$$

for $i = 2, \ldots, n$, and $\vec{p''_i} = \vec{p_i}$ when $i = 1$. When the backward stage reaches the end effector, a full iteration is completed, and the configuration of the kinematic chain is contained in P''.

With $i = n$ in (6), the final end effector position is changed, and is not the same as the target. The error tolerance has to be specified by the user. If the error $||\vec{p''_n} - \vec{t}||$ is larger than the tolerance, another iteration begins with initial positions set to those in P''.

2.3.1 Convergence

A formal proof of convergence is given in the follow up paper to FABRIK by Aristidou et al. In this section we investigate which cases cause the algorithm to not converge, and examine the given proof.

There exist three cases in which FABRIK, without any modifications, is not able to converge. In the first case the target is unreachable in regard to the maximum length of the chain, see equation (2). Then the chain is extended towards the target as in equation (3). The second case is similar, the target is reachable in terms of distance, but the lengths of the links are constraining movement such that it is ultimately unreachable. For instance, a kinematic chain with three joints where one link is long and the other one is short, in such a way that the target cannot be reached. Seen in figure 5. Formally, given a kinematic chain containing a link d_{max} longer than all other links combined $d_{max} > \sum_{i=1}^{n-1} d_i - d_{max}$ and a target located in a distance less than $2d_{max} - \sum_{i=1}^{n-1} d_i$ from the root, the method will not converge. In an implementation, this can be handled by putting a maximum number of iterations, or by comparing the position of the end effector between iterations and stopping if the distance is not large enough according to some set tolerance.

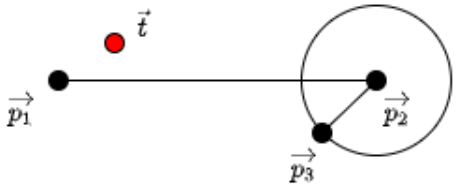


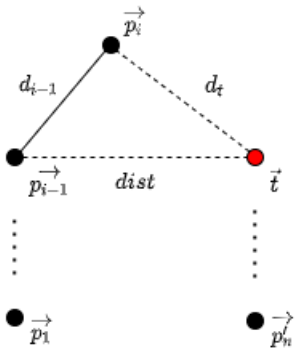
Figure 5. No solution exists as long as $\vec{t}$ is outside of the circle with origin $\vec{p_2}$ and radius $||\vec{p_2} - \vec{p_3}||$.

In the third case, the kinematic chain is straight, and the target is located on any of the line segments (links) connecting each joint position. In this situation, performing the forward and backward stages of the algorithm always results in a straight chain and the end effector is unable to reach the target. Aristidou et al. suggests allowing a small degree of sideways bending in the backward stage of the first iteration to account for this.

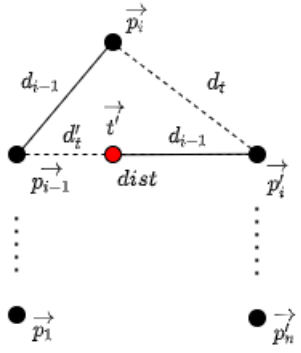
Assuming none of the three cases above are current, it has been shown that the solution converges. Let P be the set of n initial joint positions,

and let P' be the set of new joint positions produced by the forward stage. An iteration consists of two identical, albeit reverse, processes. The forward stage has a target F_t (the root), and the backward stage has a target B_t which coincides with the IK target. Forward stages bring us closer to F_t while backward stages bring us closer to B_t. We wish to show that after some number of iterations, the end effector reaches the target position B_t. We do this by examining what happens in each of the steps in one of the stages.

Each stage consist of $n-1$ steps, at each step there are three positions involved. The acting target $\vec{t}(=\vec{p_i'})$, the acting end effector $\vec{p_i}$, and the parent of the acting end effector $\vec{p_{i-1}}$, with $i = 2, \ldots, n$. The acting target is the position we wish to move $\vec{p_i}$ to. We call the distance between the acting end effector and target the residual distance d_t. An overview of the situation is given in figure 6a.



(a) Beginning of a step. Including acting target $\vec{t}$, acting end effector $\vec{p_i}$, parent of acting end effector $\vec{p_{i-1}}$, and residual distance d_t.



(b) End of a step. Including the new acting target $\vec{t'}$, new acting end effector $\vec{p_{i-1}}$, and new residual distance d_t'.

Figure 6. Shows the configurations of joint positions and distances within a step in one of the stages.

Note that the acting target and acting end effector coincides with the real target (B_t) and real end effector ($\vec{p_n}$) in the first step of the forward stage.

In each step, the acting end effector $\vec{p_i}$ moves to the acting target position $\vec{p_i'}$, and the new target position is denoted $\vec{t'}(=\vec{p'_{i-1}})$. The new residual distance d_t' is the distance between the new acting target $\vec{t'}$ and the new acting end effector $\vec{p_{i-1}}$. Resulting situation is shown in figure 6b. Showing

that the residual distance decreases in each step, implies that the distance between end effector and target is decreasing as well. Thus, by generalizing for each iteration and showing that

$$d'_t < d_t, \tag{7}$$

in each step, we can prove that the solution is converging.

Given an arbitrary step, a triangle is formed by the three involved points $\vec{t}$, $\vec{p_i}$, and $\vec{p_{i-1}}$, as shown in figure 6a. Because of triangle inequality we have,

$$d_{i-1} < d_t + dist, \tag{8}$$

and

$$dist < d_t + d_{i-1}. \tag{9}$$

Where d_{i-1} is the length of the link between $\vec{p_i}$ and $\vec{p_{i-1}}$ (as before), and $dist$ is the distance between $\vec{p_{i-1}}$ and $\vec{t}$.

At the end of the given step, $\vec{p_{i-1}}$, p'_i, and $\vec{t'}$ lie on the line defined by $\vec{p'_i}$ and $\vec{p_{i-1}}$. Giving us the following equalities,

$$d'_t = d_{i-1} - dist \text{ if } d_{i-1} > dist, \tag{10}$$

and

$$d'_t = dist - d_{i-1} \text{ if } dist > d_{i-1}. \tag{11}$$

Substituting (8) in (10) and (9) in (11) gives,

$$d'_t < d_t + dist - dist = d_t \text{ if } d_{i-1} > dist,$$

and

$$d'_t < d_t + d_{i-1} - d_{i-1} = d_t \text{ if } dist > d_{i-1}.$$

Hence, proving that $d'_t < d_t$ (Aristidou et al. 2016).

Since the steps within both stages perform the same process only in different directions, this shows that both stages converge toward their respective targets F_t and B_t. With B_t being the target of the IK.

2.4 Cyclic coordinate descent

The Cyclic Coordinate Descent (CCD) method is a commonly used iterative IK solver first introduced by Wang & Chen in 1991. It is computationally fast, and straightforward to implement, making it well suited for real time

applications. The general idea of CCD is to rotate the joint links in the kinematic chain one at a time such that the distance between the end effector and the target always decreases. An outline of the algorithm follows: The algorithm starts at the parent of the end effector, rotating the link between the end effector and its parent such that the distance to the target is minimized. Now moving onto the next joint, we still wish the rotation of this next link to bring the end effector as close to the target as possible. This is done by calculating the angle between the vector from the current joint to the end effector, and the vector from the current joint to the target. Then, rotation of the current link by the calculated angle is done about the axis perpendicular to those two vectors. The same is done at each joint until the root is reached. If the distance between the end effector and the target is less than a set distance, the algorithm stops, otherwise it moves onto the next iteration. One full iteration of CCD on a kinematic chain with three joints can be observed in figure 7. The procedure is described formally below.

Let P be a kinematic chain (section 2.2.1) with n joints, $\vec{t}$ be the target position, and $\vec{p_i}$ be any joint except the end effector ($i = 1, \ldots, n-1$). Assuming all previous links have been properly rotated according to equations (12) and (13). We wish to calculate the smallest angle θ between vectors $\vec{p_n} - \vec{p_i}$ and $\vec{t} - \vec{p_i}$, as well as the rotational axis $\vec{r}$. We use the dot product to calculate the angle, and the cross product to determine the rotational axis. We get,

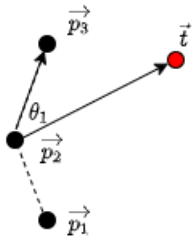
$$\cos\theta = \frac{(\vec{p_n} - \vec{p_i}) \cdot (\vec{t} - \vec{p_i})}{||(\vec{p_n} - \vec{p_i})||\ ||(\vec{t} - \vec{p_i})||}, \tag{12}$$

and,

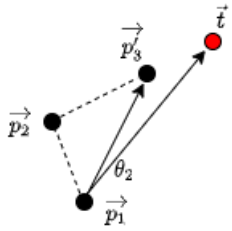
$$\vec{r} = \frac{(\vec{p_n} - \vec{p_i}) \times (\vec{t} - \vec{p_i})}{||(\vec{p_n} - \vec{p_i}) \times (\vec{t} - \vec{p_i})||}. \tag{13}$$

Notice that if we use right hand rotation, the direction of the rotation will always be correct as a result of the properties of the cross product. See figure 8 for clarification.

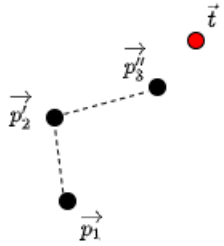
After acquiring the angle axis rotation R at the current joint, we need to rotate the child and grandchild joints positions to update the kinematic chain, that is all joints $\vec{p_j}$ with $i < j \leq n$. This has to be done if the implemented kinematic chain structure is not a rigid armature, if it is, it is enough to rotate only the current joint link (as the rest of the armature will



(a) Initial positions of kinematic chain and target. The angle to rotate is θ_1, the angle between vectors from current joint to end effector and target, current joint being $\vec{p_2}$.



(b) The most outward link has been rotated such that the end effector is as close to the target as possible. The current joint is now $\vec{p_1}$ and the angle to rotate is θ_2.



(c) The final configuration. Finally, the link from the root was rotated by θ_2, moving both $\vec{p_2}$ and $\vec{p_3}$.

Figure 7. Displays one iteration of CCD on a 2D kinematic chain with 3 joints. Two rotations are performed, one per link, and the final configuration is shown in (c). Rotations are clockwise in this example.

rotate with it). Letting the new joint positions be $\vec{p_j'}$, we relocate the joints by,

$$\vec{p_j'} = \vec{p_i} + R(\vec{p_j} - \vec{p_i}), \tag{14}$$

for $j = i + 1, \ldots, n$.

Naturally, this positioning update has to be performed at every joint except the end effector (since we start at the parent of the end effector), resulting in $n - 1$ rotations each iteration of the algorithm, one for each link. When the root joint is reached, we check if the distance between the end effector and target is within the predetermined tolerance. If it is, we stop. Otherwise, a new iteration begins with the final joint positions from the

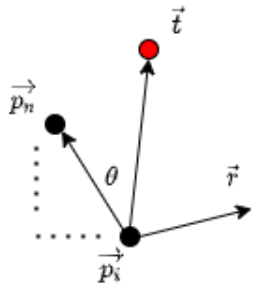


Figure 8. Right hand rotation around $\vec{r}$ will always rotate in the wanted direction. As seen from the opposite direction of $\vec{r}$, the situation in the figure would result in clockwise rotation. Should we swap the positions of $\vec{p_n}$ and $\vec{t}$, $\vec{r}$ would be pointing in the opposite direction, resulting this time in counterclockwise rotation. Both being correct for their respective situation. Dotted lines are joints between current joint and end effector. Vector $\vec{r}$ is perpendicular to both $\vec{p_n} - \vec{p_i}$ and $\vec{t} - \vec{p_i}$.

previous iteration as the new initial set of joint positions. Iterations continue until the error is within the tolerance.

The baseline CCD method favors higher rotation of joint links close to the end effector. This can be regulated by damping factors, which put a restriction on the rotation of certain links. Or by reversing the algorithm to start at the child of the root, which favors higher rotation of links close to the root instead (Kenwright 2012). When modeling human arm motion with CCD, we want to start from the end effector, since the forearm tends to move more than the upper arm when moving the hand.

2.4.1 Convergence

This section discusses the cases where CCD does not converge, as well as give an idea of why the method converges if none of these cases are current.

Three cases exist where the method can not converge, all of them being the same as those in FABRIK. The first one being that the target is out of reach. The second one being the situation depicted in figure 5. In the third case, the kinematic chain is straight, and the target is located on the chain, leading to no rotation or 180 degrees rotation by equation (12).

If the target is out of reach, the chain will straighten towards the target, regardless of the initial joint positions. This happens since the links always rotate such that the end effector gets closer to the target, and eventually it

reaches the closest position possible, which is when the chain is straightened in the direction of the target. See figure 9.

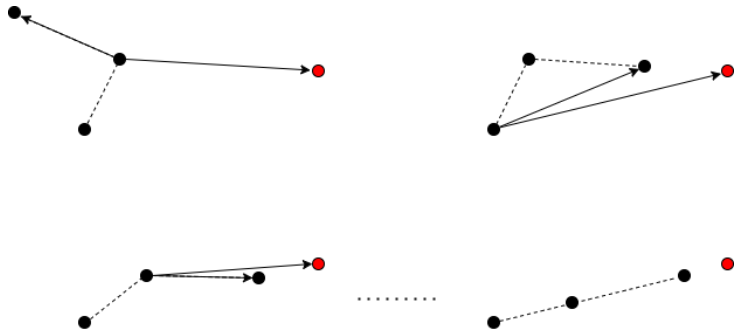


Figure 9. Shows how the CCD algorithm proceeds when the target is unreachable. Eventually, the chain is straightened towards the target. Joints in the chain are black and the target is red.

The second case is handled by putting a restriction on the amount of iterations in the implementation. And the third case can be handled by recognizing the situation and introducing a random rotation of any link in the chain, which would cause the algorithm to proceed normally.

The following gives an idea of why the CCD algorithm converges, it should not be considered a proof. Assume none of the three cases above are current, and let P contain the initial joint positions of the kinematic chain, and $\vec{t}$ be the target position. By the construction of the method, we are looking for the angle θ (see equation (12)) which by rotation around $\vec{r}$ (see equation (13)) brings the end effector as close to the target as possible. Let the distance between $\vec{p_n}$ and $\vec{t}$ before a rotation be called d_t. And let the distance between the end effectors new position $\vec{p'_n}$ and $\vec{t}$ be called d'_t. Assume $d_t \leq d'_t$, then we rotated away from the target, or we did not rotate at all. By the design of the algorithm, we did not rotate away from the target, leaving us with $d_t = d'_t$. In this case, we did not rotate at all, but then θ must be zero. This means the chain is straight and we are either in the first case, where the target is unreachable, or in the third case, where the chain is straight and the target is located on the chain. This contradicts our assumption that none of the three cases were current. We have $d'_t < d_t$ each rotation, and the method converges.

2.5 Pole

In some applications, affecting the way the kinematic chain bends could be an important addition. For instance, the elbow of a human arm tend to bend outward or downward from the body instead of inward when moving the hand. These situations are most often modeled using joint constraints. However, this section discusses a different approach, using the concept of a pole.

A pole is a predetermined position which dictates what direction the kinematic chain should bend towards. Unlike joint constraints, the pole does not affect the chains ability to reach the target. The pole method presented below repositions the intermediary joints such that they end up in the position closest to the pole without constraining the kinematic chain. A 2D simplification of the general idea is shown in figure 10. The pole calculations are performed on all intermediary joints after the final kinematic chain configuration has been obtained by the IK method and can therefore be added to both FABRIK and CCD implementations.

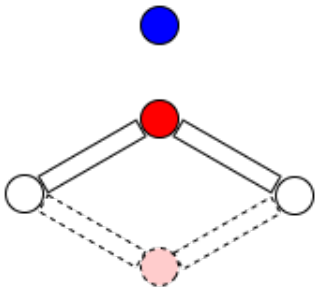


Figure 10. Shows the idea of a pole in two dimensions. There are two valid positions available for the intermediary joint (red), but the position closest to the pole (blue) is chosen. The dashed elements make up the alternative configuration.

In three dimensions, the situation is more complex than what is shown in figure 10. Any intermediary joint is repositioned in the following way. Given a pole $\vec{a}$, and any three joint positions from the final configuration $\vec{u}, \vec{v}, \vec{w} \in P_{final}$ such that $\vec{u}$ is the parent of $\vec{v}$, which in turn is the parent of $\vec{w}$. We wish to move $\vec{v}$ such that the distance to $\vec{a}$ is minimized, without compromising the lengths of the interconnecting links. The solution is based on the idea that $\vec{v}$ can rotate freely around the vector $\vec{n} = \vec{w} - \vec{u}$, as depicted in figure 11. By projecting $\vec{a}$ and $\vec{v}$ onto a plane defined by the normal vector $\vec{n}$

and the position $\vec{u}$, finding the rotation angle θ about $\vec{n}$ which minimizes the distance between the projected points, and translating back into 3D space, we can obtain the updated position of $\vec{v}$.

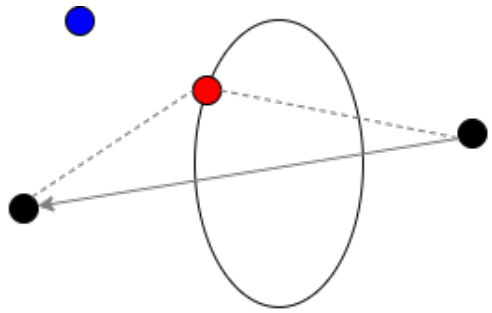


Figure 11. Shows the idea of a pole in three dimensions. The intermediary joint (red) is positioned where the distance to the pole (blue) is the shortest, without compromising the kinematic chain. As long as the intermediary joint resides on the circle (as seen along the vector), the links (dashed lines) can keep their length.

We begin with finding the projections of $\vec{v}$ and $\vec{a}$ on the plane in the direction of the plane normal $\vec{n}$. Let $\vec{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ with $x, y, z \in \mathbb{R}$, then the equation of the plane is,

$$\vec{n} \cdot (\vec{u} - \vec{X}) = 0, \tag{15}$$

where $\cdot$ is the dot product. The projections onto the plane are given by,

$$\vec{v}_{proj} = \vec{v} - \vec{n}t_1,$$
$$\vec{a}_{proj} = \vec{a} - \vec{n}t_2,$$

with $t_1, t_2 \in \mathbb{R}$ such that $\vec{X} = \vec{v}_{proj}$ and $\vec{X} = \vec{a}_{proj}$ solves equation (15). The next step is to find the angle between $\vec{v}_{proj} - \vec{u}$ and $\vec{a}_{proj} - \vec{u}$. And to determine which direction to rotate. By using the definition of the dot product (and the fact that we are in the standard basis) we can find the smallest angle θ between the two vectors,

$$\frac{(\vec{v}_{proj} - \vec{u}) \cdot (\vec{a}_{proj} - \vec{u})}{||\vec{v}_{proj} - \vec{u}||\, ||\vec{a}_{proj} - \vec{u}||} = \cos\theta, \tag{16}$$

using the algebraic definition to calculate the numerator. To determine the direction of rotation we can use the cross product in the following way. Since

the cross product $(\vec{v_{proj}} - \vec{u}) \times (\vec{a_{proj}} - \vec{u})$ is orthogonal to $\vec{v_{proj}} - \vec{u}$ and $\vec{a_{proj}} - \vec{u}$, and points in the direction given by the right hand rule (note that the cross product is anticommutative). There are two cases, (1): it will point in the same direction as the plane normal $\vec{n}$; indicating counterclockwise rotation, and (2): in the opposite direction of the plane normal; indicating clockwise rotation. The full situation is depicted in figure 12. We have,

$$\vec{n} \cdot ((\vec{v_{proj}} - \vec{u}) \times (\vec{a_{proj}} - \vec{u})) \begin{cases} > 0 & \text{in case (1)} \\ < 0 & \text{in case (2).} \end{cases} \tag{17}$$

Finally, the signed angle θ_{sign} is determined by (16) and (17) such that $\theta_{sign} = \theta$ if the expression in (17) returns a positive value, and $\theta_{sign} = -\theta$ otherwise.

What remains is to update the original joint position $\vec{v}$ based on θ_{sign}. To do this, we construct the new vector $\vec{v}_{new}$ according to $\vec{v}_{new} = \vec{u} + R(\vec{v} - \vec{u})$. Where R is the angle axis rotation according to θ_{sign} and $\vec{n}$.

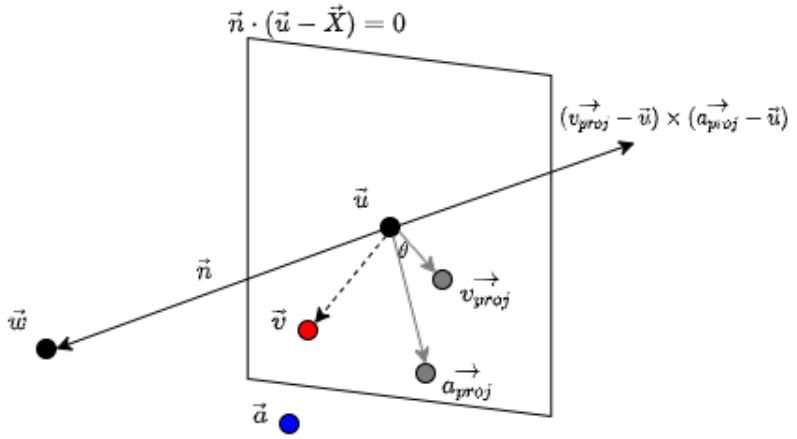


Figure 12. Displays an example situation with the elements included in the pole calculations. The goal is to rotate $\vec{v} - \vec{u}$ (dashed line) about $\vec{n}$ such that $\vec{v}$ (red) is as close to the pole $\vec{a}$ (blue) as possible. The pole and intermediary point are projected (gray) onto the plane defined by $\vec{u}$ and $\vec{n} = \vec{w} - \vec{u}$, and the rotation angle θ is determined by the projected points $\vec{v_{proj}}$ and $\vec{a_{proj}}$. The direction of the rotation is determined by the cross product of the projected points. In this figure the cross product is pointing in the opposite direction of the plane normal $\vec{n}$, which indicates that we should use clockwise rotation.

3 Method

This section lists the hardware used, gives a brief overview of the implementation, and an explanation of the tests conducted on the IK methods.

3.1 Hardware

All tests were run on a Windows 10 (version 19041.572) 64-bit machine with an Intel Core i5-4670 CPU @ 3.4 GHz, 8 GB of RAM, and a GTX 1060 GPU with 6 GB of VRAM. The VR hardware used was a Lenovo Explorer Windows Mixed Reality (WMR) headset, and WMR motion controllers.

3.2 Implementation

The human arm model and IK methods were both implemented using the Unity game engine (version 2019.4.9f1). A Unity plugin called SteamVR (version 2.6.1) was used to handle tracking of the VR hardware.

3.2.1 Human arm model

The arm model was implemented using the base three dimensional cube in Unity, as well as parent child relationships. A total of three cubes were used, representing the shoulder, elbow, and hand. The cubes were configured such that the hand cube was a child of the elbow cube, which in turn was a child of the shoulder cube. Effectively, making it a kinematic chain with three joints, the shoulder being the root, and the hand being the end effector. Each cube in the model had 6 degrees of freedom.

3.2.2 Virtual reality integration

Unity has built in VR support, but for this application, we opted to use the SteamVR plugin maintained by Valve. The plugin handles loading 3D models for VR controllers, input from those controllers, and estimation of what the users hand looks like while using the controllers (Valve 2015). Hand estimation was never used in the application. Headset and controllers were tracked in 6 degrees of freedom.

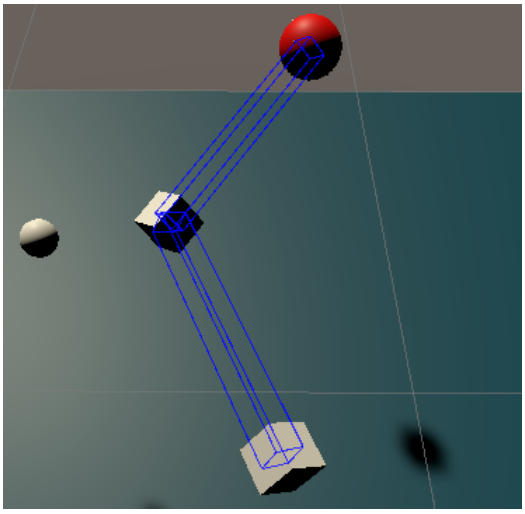


Figure 13. Shows the human arm model with joints as cubes and links as blue outlines. The gray sphere is a pole, and the red sphere is the target. The arms end effector is inside the target.

3.2.3 Inverse kinematics methods

Both methods, as well as the pole, were implemented according to the theory in section two using C# scripts. The scripts were attached to the end effector in the human arm model described in section 3.2.1 and had access to all positional parameters of each joint in the chain. Code for running the IK with a pole were included in both scripts, along with an option to turn it on and off. The error tolerance as well as the maximum number of iterations allowed were configurable. The target of the IK could be any object in Unity with a positional component known as a transform. Therefore, making it easy to use VR controllers as targets. The pole target could be any object with a transform component as well, and was positioned using 3D coordinates. The human arm model, pole, and target, can be observed in figure 13. All implementation code is available on the projects Github page (Stolpe 2020).

3.3 Tests

Two tests were conducted on both methods. The first test examined their ability to converge. It proceeded as follows. The human arm model described above were put in an initial configuration, and an error tolerance

was specified. Then, the selected method was tested on 30 reachable target positions. For each position, the number of iterations and time required to reach the specified error tolerance were measured. Both methods were tested using the same initial configuration of the human arm model, and the same target positions. Seven different values for the error tolerance were tested: 0.1, 0.075, 0.05, 0.025, 0.01, 0.005, and 0.001. Unity units are in meters. For each one, the average number of iterations per position and the average time to converge were computed. Any points on which a method did not manage to converge were not counted towards the averages, but were noted. The pole calculations were excluded during the measurements. Averaging over several target points was done to reduce any unwanted bias towards any of the methods.

The second test examined both methods ability to replicate human arm movement using a pole. The SteamVR CameraRig asset was used to track the two hand controllers and VR headset position in real time. Two human arm models were positioned on opposite sides of the tracked VR headset position, each with a shoulder width offset. The hand controllers were set as targets of the IK scripts, and poles were added to the scripts on both arm models with the intent to improve positioning of the elbow joints. Then, a wearer of the VR hardware performed several poses, and in the meantime both wearer and Unity application were recorded. One recording was done without poles for comparison. The resulting poses of the human arm models were then compared with the real life poses. The same scene view camera angle, and real life poses, were used for both IK methods. The VR setup is shown in figure 14.

All code, Unity scene configuration, and target positions used for testing are available in the projects Github repository (Stolpe 2020).

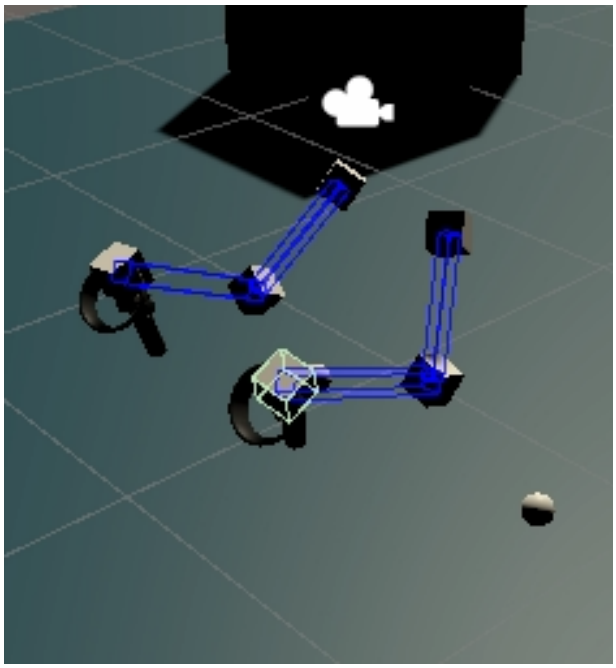


Figure 14. Shows the VR setup used in the Unity application. The white camera is the VR headset position. The sphere is a pole (the other pole is not in line of sight). The cubes are the joints in the human arm model, and the blue outlines are the links.

4 Results

Two tests were conducted on both IK methods, in this section the results are presented. The reader is advised to read section 3.3 were both tests are described in detail before continuing.

4.1 Convergence rate towards specific error tolerances

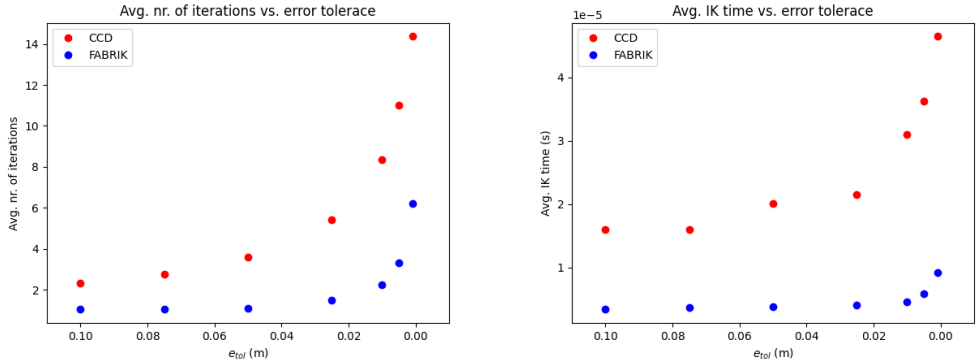
In the first test, both methods were examined based on their ability to converge towards a specified error tolerance e_{tol}. Both the number of iterations and computation times were measured, table 1 shows the averages of both methods over 30 target points.

Table 1. Showing the results of the convergence rate test for FABRIK and CCD. The first column shows all error tolerance values, the second and third column shows the results for FABRIK, and the fourth and fifth column shows the results for CCD. Time is measured in seconds, tolerance in meters. View section 3.3 for details about the test.

	FABRIK		CCD	
e_{tol} (m)	iterations	time (s)	iterations	time (s)
0.1	1.033	$3.393 \cdot 10^{-6}$	2.333	$1.606 \cdot 10^{-5}$
0.075	1.067	$3.640 \cdot 10^{-6}$	2.767	$1.602 \cdot 10^{-5}$
0.05	1.100	$3.799 \cdot 10^{-6}$	3.600	$2.016 \cdot 10^{-5}$
0.025	1.500	$4.029 \cdot 10^{-6}$	5.433	$2.153 \cdot 10^{-5}$
0.01	2.233	$4.649 \cdot 10^{-6}$	8.333	$3.105 \cdot 10^{-5}$
0.005	3.300	$5.905 \cdot 10^{-6}$	11.00	$3.623 \cdot 10^{-5}$
0.001	6.200	$9.274 \cdot 10^{-6}$	14.37	$4.649 \cdot 10^{-5}$

As seen in the table, FABRIK converged in fewer iterations, and in less time, for all error tolerances. Times were in the 10^{-6} second range for FABRIK and in the 10^{-5} range for CCD. Iteration count were at least two times higher for CCD for all tolerances. The times for a single iteration were inconsistent. For example, looking at the FABRIK results, a tolerance value of 0.1 results in a single iteration being approximately $3.3 \cdot 10^{-6}$ seconds, while for a tolerance of 0.001 the same result is approximately $1.5 \cdot 10^{-6}$ seconds.

Graph representations of the data in table 1 can be observed in figure 15. While iterations and IK time seem to increase dramatically with decreasing error tolerance for CCD, FABRIK starts off more linear. This is especially



(a) Shows number of iterations vs. error tolerance (m).

(b) Shows time (s) vs. error tolerance (m).

Figure 15. Graph representations of the data in table 1.

clear for the number of iterations on the first three error tolerance values. For error tolerances 0.01 and below, both time and number of iterations starts increasing rapidly for both methods.

The number of iterations were consistent during several runs of the test. With e_{tol} set to 0.001 CCD failed to converge on three out of thirty target positions, the results for these positions were not counted towards either of the averages. To determine for which error tolerance FABRIK started to fail, it was decreased further than 0.001. Which revealed that FABRIK consistently converged on all target positions down to an error tolerance of 10^{-7} where it failed to converge on one position. Observational differences in the end effectors position were almost unnoticeable when decreasing the tolerance below 0.005.

4.2 Arm pose reconstruction with tracking input

The second test examined how well the methods could replicate movements of the human arm using VR hardware to track hand and head positions, hand positions being set as targets for the IK. Both methods were tested and recorded with, and without, the addition of poles.

Two of the real life poses and resulting arm model poses from the recordings can be observed in figures 16 and 17, poles were used in both. The poses will be referred to as pose 1 and pose 2, respectively. As seen in the figures,

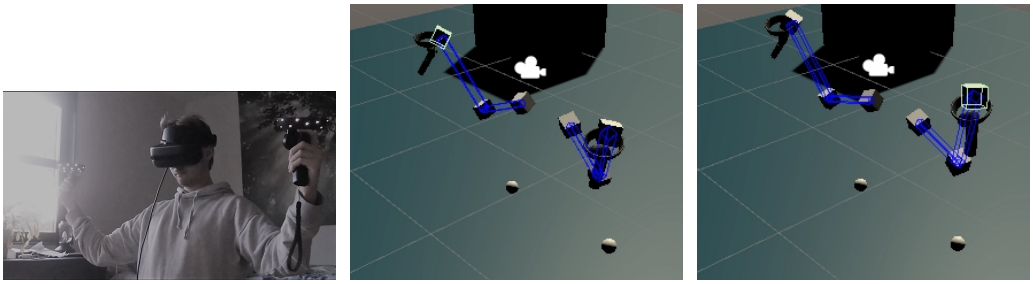
both methods manage to recreate arm poses that are observably similar to that of the VR wearer (note that the wearer is shot from a slightly different angle). Slight differences can be seen in the subfigures comparing CCD and FABRIK, mostly as a result of the wearer not being able to perform the exact same pose twice. Most other resulting poses were also observably similar, excluding those were pole placement did not coincide with the VR wearers elbow position.

Pole positioning was critical for improving the similarity between arm pose recreations and real arm poses, the results of both methods were very similar when the same poles were used. Positioning the poles as in figures 16 and 17 (down and slightly outwards from the shoulders) prevented the arm models elbows from pointing inwards or upwards, and was subjectively considered as the position which resulted in the most natural pose reconstructions. However, those positions also resulted in some reconstructions being inaccurate. For instance, when flaring the elbows outwards in real life, the elbows of the arm models would point downwards instead.

In figure 18 the results of the same poses, performed without poles, can be observed. They are noticeably different from the real life poses, with the elbows flaring inwards in pose 1 and outwards in pose 2. Using the IK without poles often resulted in non accurate positioning of the elbows.

A couple of observations were made by the wearer during the VR tests. Most notably, when using CCD, the end effector position flickered slightly from side to side. The effect was almost unnoticeable, but when focusing on it, it was clearly present. Additionally, when using the methods without a pole, FABRIK resulted in the arm model moving in a more predictable fashion, with the hand position always moving in a line towards the elbow.

A link to a demo recording of FABRIK as well as all additional pictures from the tests are available on the projects Github (Stolpe 2020).

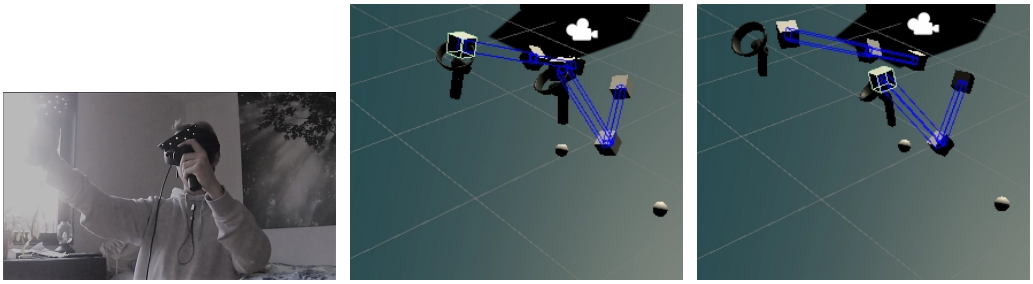


(a) Shows the pose of the VR wearer.

(b) FABRIK result on the human arm models.

(c) CCD result on the human arm models.

Figure 16. Shows the results of the wearer of the VR hardware performing pose 1.

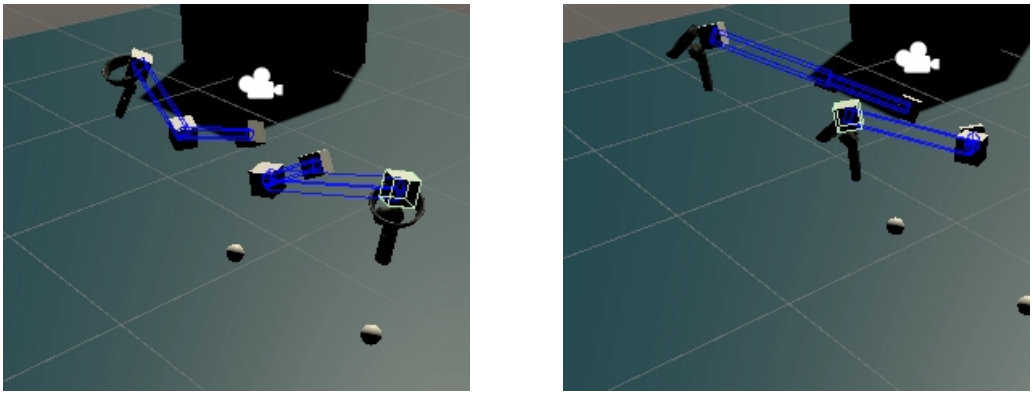


(a) Shows pose of the VR wearer.

(b) FABRIK result on the human arm models.

(c) CCD result on the human arm models.

Figure 17. Shows the results of the wearer of the VR hardware performing pose 2.



(a) Pose 1 performed without poles.

(b) Pose 2 performed without poles.

Figure 18. Shows the results of the wearer of the VR hardware performing pose 1 and 2 without poles attached to the IK. The spheres, which are poles, can be seen in the figures, but they were not enabled.

5 Discussion

Inverse kinematics is a feasible solution to animating user avatars in virtual reality with limited tracking data. Increasing user immersion is a key feature of VR hardware, and accompanying software should strive to enhance this property. Particularly, the users embodiment in the virtual world is an important factor in increasing immersion. In a lot of VR experiences, embodiment is found lacking. The user is often presented with a pair of floating hands, dislocated from their body, as their means to interact with the virtual environment. In most mainstream consumer VR hardware, the only tracked points are the head and hand positions. Thus, making a full body recreation with moving arms and legs matching the wearers is hard. While additional tracking sensors is needed for recreating accurately moving legs, the tracking of head and hands makes it possible to animate the arms of a virtual user avatar using inverse kinematics. Several methods for solving IK exist, and the question becomes what method to use. A novel method for upper body IK was presented by Parger et al. (2018), and Jacobian inverse IK methods were examined by Caserman et al. (2019), both containing user studies which show an increase in immersion when using IK for the users virtual avatar. The purpose of this study was to compare two commonly used heuristic IK methods, FABRIK and CCD, by testing them in regard to computational performance, and the ability to recreate arm poses using tracking of head and hand positions. In addition to the IK methods, a pole was implemented to help with the positioning of the elbow.

We found that both methods' computational performance was sufficient for real time environments, with convergence times less than one tenth of a millisecond for all tested error tolerances. This came as no surprise since both methods are computationally simple, and the human arm model is only three joints. However, FABRIK was consistently faster than CCD by approximately one order of magnitude. This is a result of CCD requiring a significantly larger number of iterations before converging. We can only speculate about why CCD requires more iterations given the provided background. A reasonable speculation would be that since FABRIK always starts with the end effector placed at the target position, and then corrects it slightly based on the movement of the root, the distance to the target is often low after a single iteration. On the other hand, CCD starts with the end effector in its initial position and has to rotate each link to bring it closer, which means the distance it can cover at each step is limited by the rotation of the other links

in the chain. Additional theory would be required to make this clear. Similar time differences between FABRIK and CCD were measured by Aristidou & Lasenby (2009). Another important observation was that CCD failed to converge on some target positions when decreasing the error tolerance below a certain value. This could be because of numerical instabilities in the CCD method stemming from the use of rotations. There is also the possibility that the method was incorrectly implemented. If not, this result would clearly tip the scales in favor of FABRIK from a computational performance standpoint.

In regard to pose recreation, we found that the choice of method did not have any noticeable impact when poles were used. Both methods managed to create observably similar poses to that of the wearer of the VR hardware as long as the pole position was somewhat correspondent of the wearers elbow position. To improve the elbow positioning further, rotational constraints would have to be implemented. Based on the background describing how the pole functions, we see that as long as the root and end effector positions of the human arm model are the same, the elbow (intermediate joint) will end up in the same place, regardless of what the elbow position was in the final joint configuration from the IK method. The different methods would produce slightly different end effector positions, but it would not be enough for a noticeable difference in the elbow position produced by the pole. Consistently recreating arm poses without using poles did not succeed since there would be no correction of the elbow. The preferred method of the wearer was FABRIK since CCD produced a slight side to side flickering of the end effector, probably because of the method being based on rotation.

None of the cases were the methods cannot converge were of any problem during the VR pose testing. With link lengths in the arm model being adjusted for the wearers arm lengths, such situations would not occur since the wearers real arm would limit the movement of the target hand controller. However, note that the arm model used was slightly shorter than the wearers arms to allow for the chain to stretch to its full length when the wearer extended his arms (intentionally allowing for one of the unreachable cases).

Unfortunately, the VR pose testing falls short of doing any proper measurements of the wearer's real arm pose, and only relies on visual observations of the recordings for each method. Any value based comparison of the methods' resulting poses was not done. This is an obvious limitation of the study, but we still think the results show that any of the two methods combined with the usage of poles is a viable option for arm pose recreation.

5.1 Conclusions

To conclude, we try to answer the research questions stated in the introduction. The first question was "Which method had the highest performance in regard to calculation time and convergence rate?" The answer to this question is simply the FABRIK method, as it had both lower IK computation times, and lower iteration count, for all tested error tolerances.

The second question was "Which method could best replicate the movements of the human arm, and produced least unnatural poses?" This question is harder to answer since no proper value based measurements were done to compare the methods. But from an observational standpoint, both methods produced a lot of unnatural poses when used without poles. And with poles, both methods produced sufficient arm poses for a majority of the tested real life poses.

Future directions would include trying the methods on longer kinematic chains with more than one target, perhaps to model the neck and spine. Additionally, to improve elbow positioning further, rotational constraints could be added. And to assess if the arm models increase user embodiment in VR, a user study could be conducted.

Finally, while both methods (with poles added) are viable options to use for arm pose recreation for VR avatars, we suggest using FABRIK based on the results of the convergence test.

References

Aristidou, A., Chrysanthou, Y. & Lasenby, J. (2016), 'Extending fabrik with model constraints', *Computer Animation and Virtual Worlds* **27**(1), 35–57.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1630*

Aristidou, A. & Lasenby, J. (2009), 'Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver'.

Aristidou, A. & Lasenby, J. (2011), 'FABRIK: A fast, iterative solver for the inverse kinematics problem', *Graph. Models* **73**(5), 243–260.
URL: *http://dx.doi.org/10.1016/j.gmod.2011.05.003*

Aristidou, A., Lasenby, J., Chrysanthou, Y. & Shamir, A. (2018), 'Inverse kinematics techniques in computer graphics: A survey', *Computer Graphics Forum* **37**(6), 35–58.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13310*

Beggs, J. S. (1983), *Kinematics*, Hemisphere Publishing Corporation, United States.

Caserman, P., Achenbach, P. & Göbel, S. (2019), Analysis of inverse kinematics solutions for full-body reconstruction in virtual reality, *in* '2019 IEEE 7th International Conference on Serious Games and Applications for Health (SeGAH)', pp. 1–8.

Kenwright, B. (2012), 'Inverse kinematics - cyclic coordinate descent (ccd)', *J. Graph. Tools* **16**, 177–217.

Kucuk, S. & Bingul, Z. (2006), *Industrial Robotics: Theory, Modelling and Control*, InTech, London.

Parger, M., Mueller, J. H., Schmalstieg, D. & Steinberger, M. (2018), Human upper-body inverse kinematics for increased embodiment in consumer-grade virtual reality, *in* 'Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology', VRST '18, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3281505.3281529*

Stolpe, E. (2020), 'Inversekinematicsvr', https://github.com/tayloh/InverseKinematicsVR. Contact the author on erst2297@student.su.se for access.

Valve (2015), 'Steamvr', https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647. Accessed: 2020-11-06. Latest release: 2020-08-11.

Wang, L. . T. & Chen, C. C. (1991), 'A combined optimization method for solving the inverse kinematics problems of mechanical manipulators', *IEEE Transactions on Robotics and Automation* **7**(4), 489–499.

West, M. (2015), 'Rigid bodies', https://dynref.engr.illinois.edu/rkg.html. Accessed: 2020-09-23.