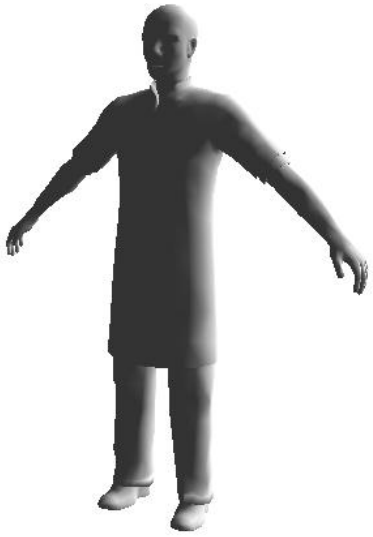
Universiteit Utrecht

EDC = EUROPEAN DESIGN CENTRE

Global inverse kinematics solver for linked mechanisms under joint limits and contacts

The BIRTH game



Master thesis Mathematical Science
30 august 2012
Student: M.E. Bartelink, BSc
Studentnumber: 3032043

Supervisor EDC:
Gino van den Bergen

Supervisors Utrecht University:
Arjan Egges
Roberto Fernández

.

Abstract

To control the motion of a virtual character with 6-DOF controllers inverse kinematics techniques have to be used. One of these inverse kinematics techniques is the Jacobian method. This report gives a new quaternion based Jacobian method to solve the inverse kinematics problem for the real time motion control of a virtual character. This includes the calculation of the Jacobian matrix which represents the change in end effector position and orientation given the change in configuration. Here, the configuration is given by the log values of the quaternions which represent the relative orientation between the bones. The method uses the quaternion properties to constrain the rotational freedom of the joints with only three joint limits. Finally, the report proposes some methods to give a response when the virtual character collides with a virtual object.

Acknowledgement

As much as I enjoyed writing this thesis, I could not have done it alone. First of all I would like to thank The European Design Centre for their hospitality and their involvement in all stages of this project. I would especially like to thank Gino van den Bergen for sharing his expertise and for helping me to get a visual insight in the problem. Also, I would like to thank Steijn Kistemaker, who helped me understand the C++ code in the BIRTH game.

From Utrecht University I would like to thank Arjan Egges. He helped me get on track with his knowledge on games and computer animations. During our regular meetings we could share our thoughts which kept me motivated. Also Roberto Fernández deserves my gratitude for serving as second reader.

Of course I would like to thank my family and friends for their love, support and encouragement. Last but not least, I would like to thank my fellow students Frédérique and Adriaan for their suggestions, cooperation and encouragement.

Contents

1 Introduction

In the last decades many new devices have arisen to control virtual characters in games. Nowadays there are next to the keyboard, mice, joystick and game pads, more advanced devices like the touchscreen, motion controllers, motion sensors and other special purpose devices like steering wheels and laser guns. A new device for the control of a virtual character also needs a new system to manipulate a virtual character by this device.

The BIRTH game is a serious game aimed at skill training for obstetricians. In the BIRTH game a virtual obstetrician is controlled with a set of Razer Hydra motion controllers. The obstetrician has to position his arms and fingers when he is performing (breech) deliveries. The Razer Hydra controllers which are hold in the hands of the game player give a position and orientation which can be used to control the hands of the virtual obstetrician.

The virtual obstetrician is implemented as a skeleton which is given by a hierarchy of bones whose relative pose is defined by a dual quaternion. A dual quaternion is a way to describe the relative position and the relative rotation of the joints. Knowing the position of the end effectors (the hands) we need to position the intermediate bones. This problem can be solved with the help of inverse kinematics techniques. We create a motion for the arm with the successive postures found as solutions of the inverse kinematics method.

For the Birth game the inverse kinematics solutions for the configuration of the arm of the virtual obstetrician should satisfy several aspects:

- Naturalness: the obtained solutions should represent a human motion as well as possible, such that the interface between the game player and system appears natural,
- Stability: the solutions should be stable to create a fluent motion, i.e. the postures of the arms should not flip up and down from one solution to the other,
- Efficiency: many solutions have to be calculations at a high speed to be able to create a real time motion,
- Joint constraints: the orientations of the joints in the solution should satisfy the rotational limits of those joints,
- Contact constraints: the obtained posture should not collide with virtual objects in the game environment or with another body part of the virtual obstetrician.

Many of those aspects influence each other. The use of an efficient method may lead to a less natural motion, than when we allow more computational time. The contact constraints decrease the interface between the game player and the virtual obstetrician. The additional constraints to the joints will require a higher computational time, but benefit the naturalness of the postures.
It is not easy and there does not yet exist a convenient way to create an inverse kinematics method which satisfies all the above mentioned aspects.

The Jacobian methods have a very high computational speed and iteratively solve from one configuration of the chain of joints and bones to the other by bringing small changes to the configuration

which creates a stable motion. A Jacobian method calculates a new solution with the help of a Jacobian matrix which represents the change in end effector position and orientation with respect to the change in configuration. The configuration of a chain depends on the orientations of the joints.

The Jacobian methods found in the literature all have some disadvantages:

- They use Euler angles as parameters for the orientation, while Euler angles are interdependent and suffer from a problem called Gimbal lock (see section 2.6),
- They only solve for the special case where a joint is restricted to the rotation around a single axis,
- They only solve for the end effector position, but not for the end effector orientation.

Therefore, we give a new Jacobian method which can be used for general chains, solve for the end effector position and orientation, allows the rotation of the joint in any direction and use independent parameters for the configuration which are obtained from the quaternions which represent the orientations of the joints. The three parameters which determine a quaternion can be obtained from its exponential map representation.

The joint constraints can easily be added to this Jacobian method. More difficult is it to add contact constraints to the method. Three Jacobian based methods to deal with contact constraints are proposed. The methods use a Jacobian matrix for the contacts which are the points of collision on the virtual character to make the contacts collision free again. The proposed methods are compared on naturalness, stability and on whether they succeed to make contacts free from the virtual objects.

The proposed methods are for general skeleton structures, but we illustrate and make them applicable within the BIRTH game. It is shown how the Razer Hydra controllers are used to control the arms of the virtual obstetrician and how we can in addition create an inverse kinematics based mechanism to let the virtual obstetrician grip an object.

2 Background material and Related work

This research was carried out in cooperation with the European Design Centre. The assignment required the improvement of the inverse kinematics method used within the BIRTH game which is part of the BIRTH project. This chapter starts with a background on the BIRTH project and the BIRTH game. It gives a background on inverse kinematics methods, quaternions and collision methods. On the end of this chapter the objectives of this research are given together with the conclusion on the background.

2.1 European Design Centre

The European Design Centre (EDC) is a creative research lab operating at the intersection of ICT, creativity and design. It was founded in 1988 by the Design Academy Eindhoven, the Ministry of Economical Affairs and the Ministry of Education and Culture. The center has a competitive staff of more than twenty people and it has two offices in the Netherlands, namely in 's-Hertogenbosch and Eindhoven, and also one office in Brussels in Belgium. EDC began as a design research and promotion centre, but in the last two decades it evolved into a commercially focused engine for generating new businesses.
The centre leads research initiatives, develops profitable projects and creates spin-off companies to specialized commercial activities, for example in the field of medical simulation and serious gaming. Aside from that EDC is an initializer of and participator in a large number of professional networks in Europe with research interests in the field of design management, gaming, international innovation and construction management. To do this EDC connects creative industries, research facilities, the academic community, entrepreneurs and start-ups right across Europe. To manage those research projects an intimate understanding of user needs, technology and economics is needed. Also, it requires a mastery of languages and communication skills to bridge the gap between disciplines present. EDC works with public sector funding bodies to stimulate investment and develop new areas of research and business.
In the last two decades EDC led over 40 projects with a combined value of more than 65 million in collaboration with over 80 partners in 29 European countries.

2.2 Project BIRTH

The BIRTH project was created after the Peristat investigation[1]. This investigation was carried out by order of the European Commission and showed that the number of deaths during childbirth in the Netherlands is one of the highest within Europe. In the Netherlands ten out of a thousand babies die around the time of birth. Research in Great Britain has shown that over 75 percent of babies with a poor start result from non-optimal care. The objective of the BIRTH project is to decrease the number of medical mistakes made during childbirth and to increase the patient's safety. The cause of medical mistakes is often found in poor communication and insufficient cooperation within the nursing team. This can be improved by providing training to teams in simulated emergency situations.
The BIRTH project consists of the development of a serious computer game in which common and rare practical situations during deliveries are simulated and practiced. With this game, gynaecologists, midwives and nurses will be trained to accurately execute sequential activities, to correctly administer medication, to appropriately use the medical instruments and to communicate optimal within the nursing team. The scenarios of the BIRTH game are recognizable practical situations and are in accordance with the protocols for the Dutch hospitals. Midwife nurses in particular are faced with complicated procedures and medical emergencies; some of these procedures and emergencies can be quite rare.

[1] http://www.europeristat.com (Accessed: 24 august 2012)

2.3 Game description

The BIRTH game is a 3D animation game. The central part of the game takes place in a delivery room where the present professionals, the mother and baby play a central role.

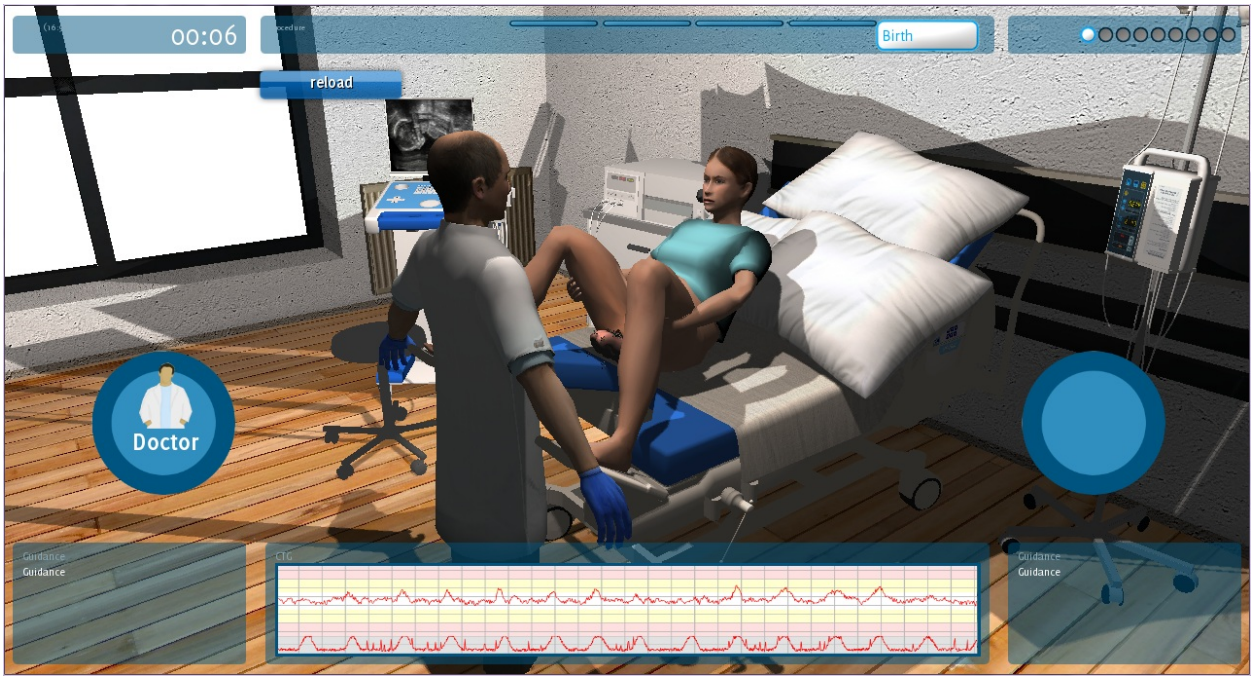


Figure 1: The environment of the BIRTH game

The scenarios of the game are simulations of the delivery of a child where the main focus lays on breech births. The player is present in the delivery room as a virtual doctor. The scenarios have an increasing difficulty level with which the game player can develop himself from a nurse that takes only informative consults, to an experienced gynecologist which can take responsibility over multiple delivery rooms. The player has to make the correct decisions during the delivery and has to perform the correct spatial actions. This is done with a set of Razer Hydra controllers. With these controllers the player can physically give motion to the arms and hands of the virtual doctor in order to practice four obstetrician maneuvers used at breech deliveries. We explain this in more detail in the section 2.4. Furthermore, he can use the controllers to select an option from the two option dials. Here the left dial consists of subjects and locations of actions and the right dial gives depending on the left dial a number of functions. By this the player can take an action like picking a tool, watching an echo or deciding to go to the operation room for a Caesarean section. In the screen we can see a CTG scan which shows the heartbeat of the fetus and the contractions of the mother, which helps the player to make the right decisions. A narrator gives support to the player by giving instructions and feedback. Also, the narrator simulates the role of the other professionals in the medical team to guarantee a team spirit.

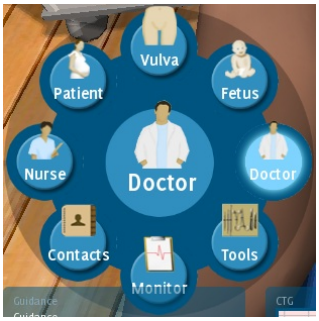


Figure 2: Dial

2.4 The controllers

The game is played with a set of Razer Hydra motion controllers, see figure 3a. This set consists of two controllers and a basis station. The Razer Hydra controllers can be compared with similar controllers for the wii or playstation.

(a) Razer Hydra Motion Controllers (b) Controllers (c) Basisstation

Figure 3: Controllers[1]

The two controllers are held in the hand of the game player as we can see in figure 3b. Each controller has two triggers and a mini-joystick with five buttons around it. The basis station can detect the position and orientation from the controllers till a distance of 1.80 meter. It detects those positions and orientations at a high speed and with a high accuracy. The additional buttons, triggers and joysticks on the Razer Hydra controllers ensure that games can be controlled without the use of another device like a mouse or keyboard. Those properties make the Razer Hydra controllers very useful for the real time motion control of virtual characters.

In the BIRTH game the Razer Hydra controllers are used to control the arms of the virtual doctor. At this moment the game players point of view is from the eyes of the virtual doctor, see figure 4.

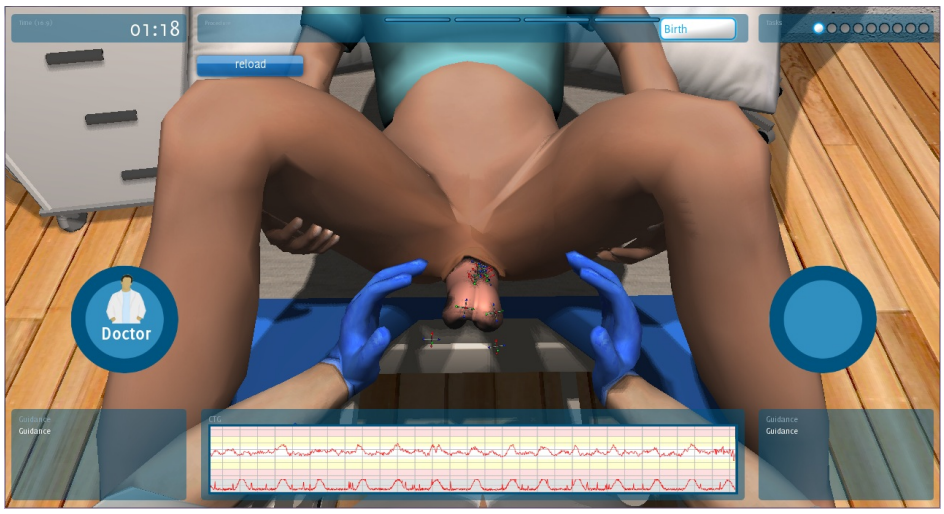


Figure 4: The arms of the virtual doctor

The position and orientation input from the controllers, which corresponds to the position and orientation of the wrists of the game player, are mapped to the position and orientation of the wrists of the virtual doctor. When the game player now gives motion to the controllers, by moving his hands with the controllers in it through the space or by rotating his wrists, we can visualize the same motion in the screen for the hands of the virtual doctor. Aside from visualizing the hands at the right position in the screen, we also need to visualize the arms of the virtual doctor in the screen. The arms can have multiple postures through the freedom in orientation of the joints in the arm. Therefore, we define the posture of the arms is determined by the configuration of the joints. The controllers do not give any information for the configuration of the joints in the arm. All we know is that one side of the arm should be connected to the neck of the virtual doctor and the other side of the arm should be connected to the wrist of the virtual doctor. Since there is no unique way to choose for a posture of the arm of the virtual doctor, which satisfies this connectivity, an inverse kinematics method will determine the posture for the arm that will be visualized. The next section explains the concept of inverse kinematics.

Aside from giving motion to the arms of the virtual doctor, the game player can let the virtual doctor grip an object by pulling the lower trigger of the Razer Hydra controller when the virtual hand is near the object. The grip motion can be visualized with an animation or again by the use of an inverse kinematics method.

Also the other parts in the BIRTH game can be controlled with the Razer Hydra controller. There is no need of a mouse or keyboard to play the BIRTH game. The game player can use the joysticks to go through the left and right option dials and use one of the buttons to select an option. Also, the camera position and orientation to watch the delivery room from another point of view can be changed.

2.5 Kinematics

Kinematics describes the motion of a body; therefore it is also called the geometry of motion. It studies the trajectory of points, lines and other geometric objects, as well as their velocity and acceleration. Kinematics is used in different fields. Some examples are: computer animation to give a real time representation to the motion of a multilink object, robotics to control the motion of a robot or machine, astrophysics to describe the motion of astrological objects like the moon, stars, planets and asteroids and biomechanics to study the structure and function of the biological systems from a human, animal, plant, organ or cell.

In this report we will restrict ourselves to the kinematics for a virtual character. The motion of a chain of bones and joints in a coordinate system will be studied. We assume that the beginning of the chain is fixed in this coordinate system and that the position and orientation of the end of the chain, which we will call the end effector, is variable in this coordinate system. We study the influence between the joint configurations and the end effector position and orientation.

Forward kinematics
We speak of forward kinematics when we want to know the end effector position and orientation given the configuration of the joints. This means that knowing the orientations of the joints we want to determine the position and orientation of the end effector. Solving forward kinematics is straightforward, there is always a unique solution.

Inverse kinematics
We speak of inverse kinematics when we want to know the configuration of all the joints given that we know the end effector position.

In the case of *redundancy*, there are multiple or even infinitely many configurations possible which will come to the same end effector position and orientation. As a consequence solving a system of equations to get a unique solution for the joint configurations is not possible. In this case the inverse kinematics problem is *under-constrained*.

It might also happen that the inverse kinematics problem is *over-constrained*. In this case no possible configuration of joints gives the required end effector position and orientation. The end effector is unreachable. A configuration of the chain which solves as close as possible to the end effector position and orientation is still required.

Simplifications and naturalness
When we apply inverse kinematics to a chain of bones and joints that represents a human limb, then we want the inverse kinematics method to compute solutions which results in a natural motion for the chain. A natural motion of the chain is considered to be a realistic motion of the corresponding human limb. This means that our inverse kinematics method should only give solutions resulting in postures a real human can make, and disregard awkward moves. One solution to avoid awkward postures is by constraining the rotational freedom of the joints. Aside from this the inverse kinematic method should give stable solutions in order to be natural, meaning that the successive configurations should result in a fluent motion of the chain. This means that two successive configurations should lay close to each other such that the chain can not flip from one

posture to another. The use of an inverse kinematics method which iteratively solves from one solution to another by bringing small changes to the joint configuration benefits a stable motion.
We also need to make some simplifications at the cost of the naturalness of the inverse kinematics solution such that the problem does not get to hard to solve. We have already simplified the model of the human limb by only considering a skeleton structure with straight bones and joints with rotational freedom. In reality there is also some translational freedom between the bones. Of course we still put skin on this skeleton structure, but the real motion of the flesh of the limb is ignored. Furthermore, it is assumed that the rotational freedom of the different joints are independent of each other, while for a real limb there is a dependency between the joints caused by the muscles.

Efficiency and Real time
An inverse kinematics method should be as simple as possible, so that the problem remains easy to solve. Aside from this, a simple method might benefit the computational time. In some applications a short computational time is very important. For example, this is the case when inverse kinematics is used for the real time motion control of a virtual character. More computational time is allowed for the inverse kinematics methods when used for example in modeling or creating animations.
Note that allowing more computation time to find a solution to the inverse kinematics problem will benefit the precision of the result.

Various inverse kinematic methods
Inverse kinematics is studied intensively during the last few decades. Many methods arose to solve the problem. The most well-known families of iterative methods will be discussed here.
The most popular family of methods is the family of Jacobian methods. The Jacobian methods use a Jacobian matrix to find a linear approximation to the inverse kinematics problem. The Jacobian matrix gives the linearization of the change in end effector position and orientation with respect to the change in joint orientation. The Jacobian methods require the computation or approximation of the inverse of the Jacobian matrix. There are several ways to do this, which led to different variations of the Jacobian method. These are the Inverse Jacobian, the Jacobian Transpose, the Pseudo Inverse Jacobian and the Damped Least Square method. We can find a description of those methods in [Buss, 2009]. Some of these Jacobian methods can have singularity problems. However, the Jacobian methods are considered to give smooth solutions, but at the high cost of the computation of the jacobian matrix.
Another family of inverse kinematic methods is the family of Newton methods. The Newton methods search for the roots (zeros) of a real valued function. For the inverse kinematics problem, this function gives the difference between the current and desired end effector position and orientation and has as variables the parameters of the joint configuration. This way the Newton method will give the joint configuration that minimizes the distance between the current and desired end effector position and orientation. The Newton methods require the computation of first and second order differentials, see [Engell-Norregard and Niebe, 2007]. Therefore the Newton methods are computational more expensive than the Jacobian methods. However, they do not suffer from singularity problems as some of the Jacobian methods do.
There is also a family of heuristic methods that solves the inverse kinematics problem. Those methods find a solution to the inverse kinematics problem by improving the solution locally. There are several heuristic methods. The most famous one is the Cyclic Coordinate Descent method which

improves the solution of the inverse kinematics problem for each joint separately. Other heuristic methods are the triangulation inverse kinematics method and the sequential inverse kinematic method, see [Aristidou and Lasenby, 2009]. The advantage of the heuristic methods is that they are computationally very fast. The disadvantage of the heuristic methods is that the solution can get stuck in a local minimum which can give an unnatural result.
Finally, there is the family of dynamical methods. The dynamical methods satisfy the physical laws and therefore require object descriptions like the center of mass, the total mass and the moments and products of inertia. They define motion by the relation between force, mass and acceleration. This makes the dynamical methods time dependent. They provide a high level of realism to the motion of objects or characters, but the dynamical methods are difficult to implement and are computationally expensive. [Welman, 1993] describes some dynamical methods.

2.6 Quaternions

To solve inverse kinematics we have to work a lot with rotations, especially to represent the orientation of a joint. There are several ways to represent a rotation. The most common representations are the Euler angles, the rotation matrix, the axis/angle and the quaternion. Euler angles represent a rotation by the angles with which we successively rotate around the three basis axes of a coordinate system. Those three rotations are also called "yaw, pitch and roll". The rotation matrix represents a rotation with a 3x3 orthogonal matrix of which the three columns represent the rotation around the basis axes. The axis/angle is a 4-dimensional vector which represents a rotation by a unit vector and an angle with which we rotate around this vector. The unit quaternion represents a rotation around a unit vector with a rotation angle, but is written as a 4-dimensional vector in quaternion space.

Each representation of a rotation has its advantages and disadvantages. We will compare the four different representations of a rotation given above on their properties considering gimbal lock, visualization, complexity, efficiency and interpolation.

Gimbal lock
A gimbal is pivoted support that allows the rotation of an object around a single axis. We can connect two gimbals to be around each other by letting their pivot axes cross each other orthogonal at the centre of the gimbals. A set of three connected gimbals can be used to let the innermost gimbal remain independent of the gimbal that it is connected to.

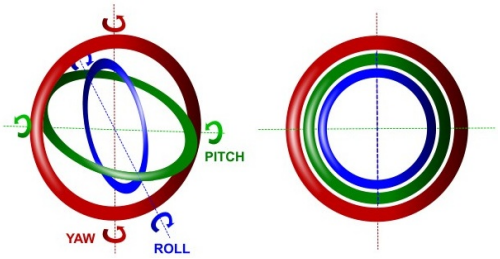


Figure 5: On the left we see three gimbals with their pivot axes denoted by yaw, pitch an roll. On the right we see a gimbal lock.[2]

Gimbal lock is the loss of one degree of freedom in a three dimensional space that occurs when the axes of two of the three gimbals get in a parallel configuration. For a system based on these gimbals, this leads to a loss of one dimension, after which we can only rotate in a 2-dimensional space. See for an illustration of gimbal lock figure 5. Quaternion representations do not suffer from Gimbal lock, since they represent a rotation with a certain angle around one axis. The axis/angle representation can have singularity problems when the angle of rotation equals zero, which means that the rotation vector can be in an arbitrary direction. For the Euler angles there are three successive rotations around three different axes, after which Gimbal lock can occur. Also, the rotation matrix can suffer from Gimbal lock when the matrix is constructed with the use of Euler angles.

Visualization
For a visualization of a rotation there is a necessity for the ability to read out the parameters of our rotational representation and interpret them. For quaternions this is not that difficult. In chapter 4 we show how the rotation vector and angle can be extracted from the quaternion. The interpretation of the axis/angle representation is the easiest, it gives a rotation vector and the angle

of rotation. The interpretation of the Euler angles is less convenient, since the order in which we rotate around the three basis axes is of big importance. A different order in rotation around the basis axes results in a different rotation. There is a dependency between the rotations around the three basis axes. This also means that one rotation can be represented by multiple Euler angles. Also the quaternion has for every rotation two representations. However, this gives less trouble than with the Euler angles, since there exist simple tricks to determine which of the two representations should be used. The rotation matrices give a one-to-one mapping to the orientations.

Complexity and efficiency
For the comparison of the efficiency of the different representations of a rotation, we first look at the number of elements used to store the representations. The Euler angles use three, the axis/angle and the quaternions use four and the rotation matrix uses nine elements. Another aspect for which we can judge the efficiency of the rotational representations is the calculation time to find the representation of two successive rotations. For quaternions and rotation matrices this can be done quite efficient by the multiplication of the quaternions and the multiplication of the rotation matrices respectively. Here we notice that the multiplication of quaternions is faster than the multiplication of rotation matrices. For Euler angles there is no quick way to find the combined rotation, it needs extra work to avoid the Gimbal lock problems mentioned above and we need to be careful with the order of rotation. The axis/angle representation has no method at all to determine a combined rotation without the use of one of the other representations. Finally, we have to take into account the effects of rounding errors that occur after a lot of computations with rotations and errors which arise in numerical integration. This means that the constraints which lay on a representation have to be encountered. The quaternion has one constraint, namely that it has to be unit. This means that as a result of errors it might be necessary to normalize the quaternion. The rotation matrices have six constraints, namely every row has to be unit and each column should be mutually orthogonal. This means that as a result of errors it might be necessary to orthogonalize the rotation matrix. Normalizing a vector is an easy operation, but orthogonizing a matrix is more time consuming. The Euler angles and axis/angle only have constraints on their rotation angles which should be in the range $[-\pi, \pi]$ to maintain their precision.
We can conclude that the quaternions are the most efficient representation for a rotation followed by the rotation matrices. This makes the quaternions the most useful for an animation-engine that requires a high computational speed. However, the Euler angles remain more intuitive than the quaternions or rotation matrices, so they are more useful for the use in an animation tool.

Interpolations
When we are working with rotations it is often necessary to interpolate between two orientations. Here an interpolated orientation is an orientation which lies in the range between two orientations. To find an interpolated orientation with Euler angles is quite difficult. The coordinates of each basis axis have to be interpolated independently. This means that the interdependency between the axes is ignored, which can cause strange interpolations. With quaternions the interpolation between orientations will be a lot easier as we will later see in chapter 4.6. The interpolation of rotation matrices is slightly more difficult and gives less smooth results than the interpolation obtained with quaternions. The interpolation of axis/angle representations can turn out quite well, but can also give strange results.

2.7 Collisions

The basic inverse kinematic methods give motion to a virtual character in an obstacle free space. However, most game environments for virtual characters are not obstacle free. Therefore the inverse kinematic methods have to be extended such that they can compute a collision free motion in an environment with obstacles. To deal with obstacles we first of all need to detect when and where the virtual character collides with an object. This is called collision detection. Once a collision is detected, the next inverse kinematics solution should not move the virtual character further through the obstacle. This is a collision response. Our main focus is not the detection of a collision, but the response to a virtual character once a collision is detected. This is still a very new topic. We will discuss some of the collision response inverse kinematic methods found in the literature.
A first group of collision response methods use the redundancy of the inverse kinematics problem. The methods of [Maciejewski and Klein, 1985], [Fratu *et al.* 2010] and [Ding and Chan, 1996] are based on the use of the null-space operator of the Jacobian matrix which defines the freedom that the configuration has for the same end effector position and orientation. By this we can as secondary goal solve for a configuration which takes the largest distance to the closest objects, but which still satisfies the primary goal of solving for the end effector position and orientation. These are collision avoidance methods, since they do not guarantee that the resulting solution is collision free. There is not always a collision free configuration for a given end effector position and orientation. The collision avoidance methods require the computation of the shortest distance to objects at every iteration of the inverse kinematics method. This increases the computational time. The extension to the Jacobian method is quite efficient.
Another group of collision response methods are path planning methods. Path planning methods are mostly used for reaching or grasping an object in an environment with obstacles. They use the fact that the virtual character wants to grab or reach to an object and plans a path for the end effector towards this object. [Bertram *et al.*, 2006] for example searches for a collision free path with Rapidly-exploring Random Trees. To plan a collision free path it is required to that we know the goal of the virtual character. In general such a goal is unknown. However, path planning algorithms can be useful for example for the creation of an animation. Path planning algorithms are moreover computationally expensive. Aside from having no clear goal this makes the path planning algorithms not useful for example for the real time motion control of a virtual character.
There is also a group of collision response methods which are based on the dynamical inverse kinematic methods. The method of [Choi and Kim, 2000] uses the difference between the velocities of the objects, instead of the distances between objects as the most methods do. We have already seen in the previous section that the dynamical methods are complicated and computational expensive. However, they give the most precise result considering collision avoidance. Another advantage of the fact that the dynamical methods have no problem with moving objects, which for other methods might be an issue.
Finally, there are the methods for collision response which are specialized for the virtual human. For example [Kallmann, 2008] gives an inverse kinematics method for the human arms and legs which take into account self-collision, meaning that the arms and legs do not collide with other body parts of the virtual character. He uses an analytical inverse kinematics formulation based on a swing-and-twist parameterization and the coupling of the three joints which represent the arm or leg. Since the self-collision is a special case of the collision with objects, we can consider the virtual character as an object as well. Therefore this method is only useful when there are no other objects present and self-collision is the only factor to consider.

2.8 Objectives

This research is carried out in assignment of the European Design centre. They gave as task to implement and design an inverse kinematics method to control the arms and fingers of the virtual doctor in the BIRTH game with the Razer Hydra controllers. The reach and configuration of the arms and fingers of the obstetrician are limited by joint limits and collision (contact) constraints. Therefore, the goal is to construct an inverse kinematics method which will be able to find a configuration of the skeleton which satisfies the constraints (joint limits and contacts) and is plausible/natural. A collision detection method is provided and implemented which provides feedback at the first contact between the virtual character and an obstacle.

The BIRTH project is still in development. At the start of this research an inverse kinematics method based on the Cyclic Coordinate Descent method is already implemented in C++. This method uses a skeleton for the virtual doctor which is given by a hierarchy of bones whose relative pose is defined by a dual quaternion. However, this method has some problems that need to be improved:

- The inverse kinematics solver uses a local optimization scheme and can be caught in a "local minimum" which results in awkward poses, where more natural poses are possible.
- The inverse kinematics solver does not take contact constraints into account.
- In cases where the inverse kinematics problem is under-constrained, the solver may pick a pose that looks unnatural.
- In cases where constraints are conflicting, e.g. target object out of reach, the current solving method may result in oscillations.

Another task is to remove the errors in the linked mechanism between the arm and the fingers of the virtual doctor. This mechanism is used in order to grip an object.

2.9 Conclusion

The most problems are caused by the local optimization character of the implemented inverse kinematic method which is based on the Cyclic Coordinate Descent method. A solution which is a local optimum to the inverse kinematic problem can give bad and unnatural results. Aside from that the Cyclic Coordinate Descent method is not the easiest method to extend with a collision response. Therefore, we decided to use another type of method, namely the Jacobian method. The Jacobian methods iteratively solve the inverse kinematics problem by bringing small changes to the configuration which leads to a smooth motion. Another reason to choose for the Jacobian methods is because they are computationally fast enough to use for the real time motion control of virtual characters. Furthermore, we prefer to use the Jacobian methods because they can use the null operator of the Jacobian matrix to find a solution which solves for a secondary goal as well. This might be useful when we want to handle collisions.
It is desired to create a pure dual quaternion based inverse kinematics method, i.e. without the use of conversions to other representations for orientations, which can deal with joint limits and contacts and gives a natural result. The dual quaternions have no problems like Gimbal lock, they are efficient for fast computations, they give smooth results when used for interpolation and they are already implemented in the BIRTH game. A pure dual quaternion based Jacobian method requires computing the Jacobian matrix with respect to the quaternion parameters. In most articles the Jacobian matrix represents the change in end effector position with respect to the change in Euler angles. Instead, we need the change in end effector position and also the change in end effector orientation with respect to the change in quaternion parameters. To define those quaternion parameters and to calculate the Jacobian matrix, the properties of the quaternions and dual quaternions will be explored.
Aside from using a Jacobian matrix for the end effector, we can also define a Jacobian matrix for the contacts obtained from the collision detector.

3 The rigid body structure of the virtual character

In this chapter we describe how a model for a virtual character which can imitate the motions of a human can be created. First we will give a structure to the body of the virtual character. Then we show how this body structure is built out of a skeleton which creates a rigid body. Once we can bring motion to the skeleton structure by adapting the orientations of its joints, it is possible to give the skeleton a human skin.

3.1 The rigid body

A rigid body is a body built out of rigid objects, which are connected to each other by joints. The *joint* is located at the point where two rigid object are connected and is the point where the rigid objects can rotate relative to each other while remaining connected. This means that the joint holds a relative orientation between two rigid objects. A *link* represents a connection to a joint, which is mostly between two joints. This means that every rigid body part is represented by exactly one link. Every link is connected to another link by a joint. To model the motion of a human or animal we use their skeleton to obtain a rigid body structure of links and joints. After giving a skin to this structure of joints and links, the model for our virtual character is complete.

3.2 The hierarchy of the skeleton

The human skeleton in figure 6(a) is used to obtain a rigid body structure for a human, which we can manipulate for rigid body motion. We place the joints at the positions between bones which can rotate relative to each other. We place the links at the position of those bones which can rotate at a joint. Bones that are connected to a joint, but that can not rotate around it, will be disregarded and not be represented by a link. Therefore, there is no link representing the rib bones in the example of figure 6(b). When there are multiple bones in a series which can rotate only very small at their joints, we can represent this series by a single link between the begin and end joint in this series. This means that the original joints and bones between the begin and end joint in this series will be disregarded. We see this in figure 6(b) at the spine of the human skeleton, where we use only a few joints between the vertebrae. When there is more than one bone between the same two joints, then we define only a single link between those two joints. We see this for example at the lower arm.
We have now constructed a simplified skeleton of joints and links. The length of the links can be calculated with the corresponding lengths of the bones of the human skeleton. On this simplified skeleton of the human we can choose a *root* joint. This gives us a tree structure as we can see for example in figure 6(c). A tree structure is a connected graph without cycles. From now on we assume that we work with rigid body structures which have this tree structure. Given a root note and a simplified skeleton we call the corresponding tree structure the skeleton tree.

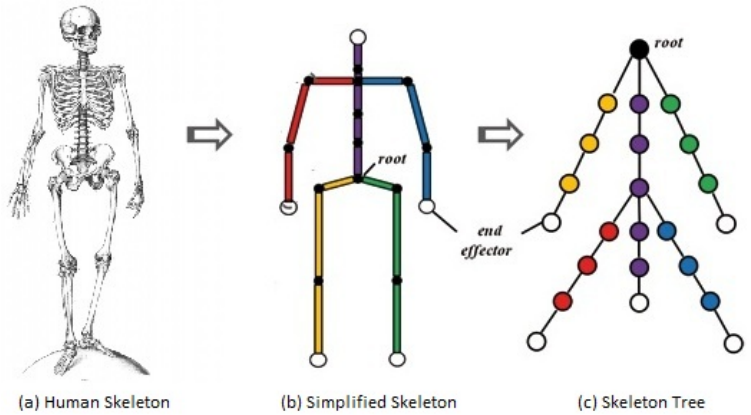


Figure 6: Skeleton structure[3]

On the skeleton tree we define *end effectors*, which are the leaves of the skeleton tree. Here the leaves of the skeleton tree are the endpoints of the links which are connected to only one joint.

On the skeleton tree we define the *parent* of a joint as the joint connected to it by the link which lays on the path towards the root joint. Every joint has a unique parent except the root joint which has no parent. We define the *child* of a joint as the joint connected to it by a link lying on the path towards an end effector. Every joint has at least one child, except the last joint on the path towards the end effector which has no children.

On the skeleton tree we define chains. A *chain* is a list of unique joints which are connected to each other by links which define a path on the skeleton tree. The first joint in the chain is called the *base* joint and the last joint in the chain is called the *end* joint. All joints on the chain have a unique child, except the end joint which has no child and all joints on the chain have a unique parent, except the base joint which has no parent. We can see this in figure 7.

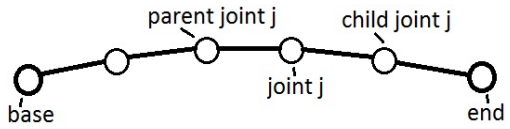


Figure 7: Chain

The chains on a skeleton tree are defined uniquely if no link can be in two different chains, while a joint is allowed to be in multiple chains. The chains cover the skeleton tree completely, if every link of the skeleton tree is in one of the chains, with an exception of the links which connect the end effectors.

The inverse kinematic method described in chapter 5 is a method that attempts to bring the end effector position closer to a given target position by changing the joint orientations in the skeleton tree. In the next section we show how those joint orientations are defined.

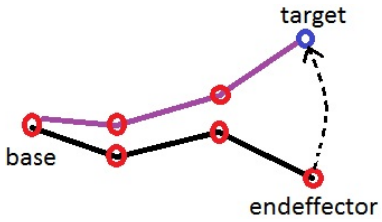


Figure 8: End effector to target

The chains on the skeleton tree have to be defined in such a way that a chain is the part of the skeleton tree which gets influenced by an end effector. Here it is important to choose the chains uniquely and let them cover the skeleton tree completely, such that two different end effectors do not influence the same part of the skeleton tree. The root is preferably chosen at the base of one of the chains. In figure 6 we can see the different chains for the human skeleton which are represented by different colors. Now that we have unique chains on the skeleton tree with each one end effector, we can solve the end effector to the target position with the inverse kinematic method for each chain separately. Here we use a specified order in which we perform inverse kinematics to

the chains. When the base joint of chain A is connected to a non-base joint of chain B , then we perform inverse kinematics to the chain B before performing inverse kinematics to chain A. When two chains have the same base joint, then the order in which we perform inverse kinematics to the chains does not matter.

3.3 Joint orientation

The configuration of the simplified skeleton is determined by the lengths of the links and the orientations of the joints. The length of a link is a constant, given by the distance between the two joints it is connected to. The orientation of a link is not constant and can be rotated. This freedom in the orientations of the joints leads to different postures of the simplified skeleton.

The orientation of a joint is determined by the relative orientation of the links which are connected to it. We give every joint a local coordinate system in which we represent the link towards the parent joint by a translation vector in this local coordinate system. Here the length of the translation vector equals the length of the link and the direction of the translation vector gives the orientation of the link. Then the orientation of a joint is determined by the relative orientation of the joints local coordinate system with respect to its parent joints local coordinate system. In the chapter 4 we explain how we can use quaternions to represent these relative orientations. In section 4.5 we give a full detail description on how we can initialize a skeleton with dual quaternions for the relative orientations and translations.

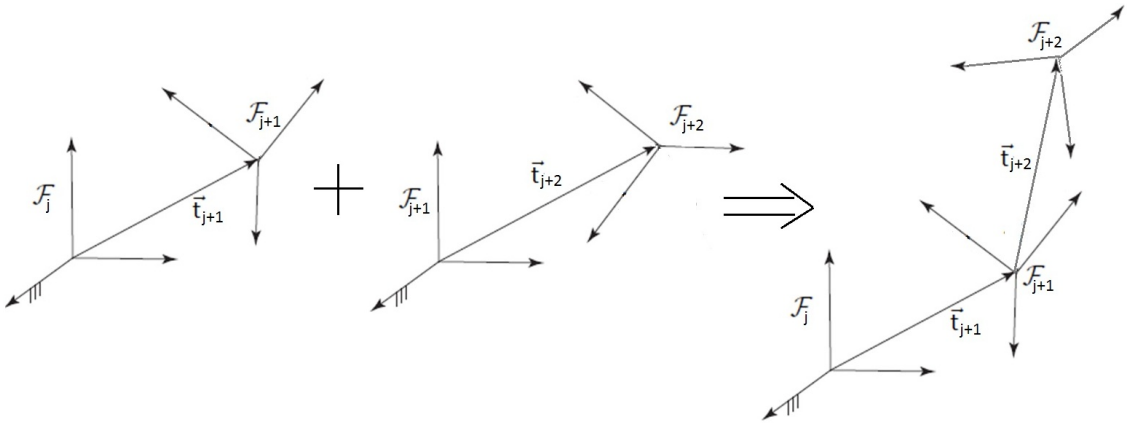


Figure 9: Chain of two joints with translation and orientation

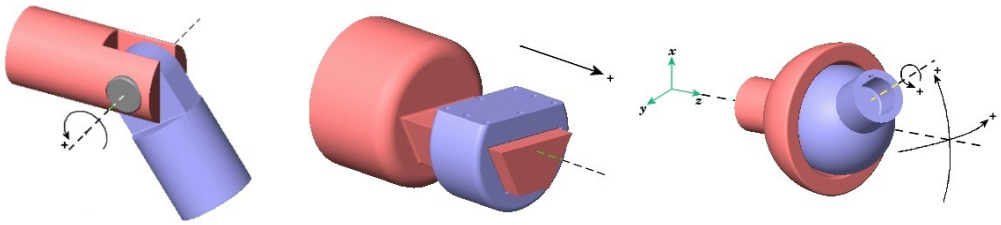
Given is a chain of joints $1, 2, 3, \ldots, k$. Let $\mathcal{F}_j$ be the local coordinate system of joint $j \in 1, \ldots, k$. The orientation of joint $j+1$ is determined by the orientation of its own coordinate system $\mathcal{F}_j$ with respect to its parents coordinate system $\mathcal{F}_j$. We represent the link from joint $j+1$ to joint j by translation vector $\vec{t}_{j+1}$. We can see the representation of this joint and link on the left of figure 9. In the same way we represent joint $j+2$ by the relative orientation between its own coordinate system $\mathcal{F}_{j+2}$ and its parent coordinate system $\mathcal{F}_{j+1}$ and we represent the link connecting it to joint $j+1$ by translation vector $\vec{t}_{j+2}$. When we connect link $\vec{t}_{j+1}$ to $\vec{t}_{j+2}$ by placing coordinate systems $\mathcal{F}_{j+1}$ on each other, as we can see on the right of figure 9, then we see how the relative orientation between $\mathcal{F}_{j+1}$ and $\mathcal{F}_j$ represent the relative orientation between the links represented by t_{j+2} and t_j. When we rotate the orientation of $\mathcal{F}_{j+1}$ with respect to $\mathcal{F}_j$, while keeping the translations in the local

coordinate systems constant, then we see that this rotation influences the configuration of the chain.

The root joint holds a translation with respect to the world coordinate system and an orientation with respect to the worlds coordinate system. The world coordinate system is the coordinate system where the human skeleton is standing in. Mostly we place the world coordinate system at the position of the root joint, since this position is considered to remain constant. When we have relative orientations and translations of a joint with respect to its parent's joint, we can also compute the relative orientation and translation of a joint with respect to the world coordinate system. We do this by taking a chain from the root joint to a given joint and transform the orientation and translation to world coordinates at every joint in this chain, starting at the root joint and ending at this given joint. We note that the translation of a joint in world coordinates does not remain constant when we rotate the relative orientation between a joint and its parents joint.

3.4 Degree of Freedom

The degree of freedom, in short DOF, is the number of independent parameters which determine a configuration. The configuration of a joint can have rotational and translational freedom. When we rotate and translate in a 3-dimensional space, then we see the degrees of freedom as the number of axes around which we can rotate plus the number of axes along which we can translate. This means that a 1-DOF rotational joint, which is also called a revolute or hinge joint, can only rotate around a single axis. A 1-DOF translational joint, which is also called a prismatic joint or slider joint, can only translate along one axis. Similarly, there exists a 2-DOF rotational joint, also called a revolute joint, which can rotate around 2 different axes. A 2-DOF translational joint, which is also called a planar joint, can translate along 2 different axes. A 2-DOF rotational and translational joint, also called a cylindrical joint, which can rotate around a single axis and translate along a single axis. Also, we can have a 3-DOF rotational joint, which we call the ball-and-socket joint, which can rotate around all three axes. We can have a 3-DOF translational joint which can translate along all 3 axes. Furthermore, we can have a 6-DOF joint which can rotate around all 3 axes and translate along all 3 axes.



(a) hinge joint (b) slider joint (c) ball-and-socket joint

Figure 10: Joint types[4]

The rigid body as we described it has only rotational freedom at the joints, this means that we can only have 1-DOF, 2-DOF or 3-DOF rotational joints. We simplify this by specifying only 3-DOF ball-and-socket joints and 1-DOF hinge joints. This means that we consider the 2-DOF joints as a special case of the ball-and-socket joint. The joints in the rigid body which have only rotational freedom have orientations relative to their parent joint. When we consider the joints as orientations relative to the world coordinate system, then these are 6-DOF joints in world space.

The term degrees of freedom can also be used for the whole rigid body and then gives the number of parameters needed for the complete configuration of the rigid body. In this case the degree of freedom of the rigid body is the sum of the degrees of freedom of all the joints in the rigid body.

3.5 Joint constraints

In the previous section we saw how we constrain the rotational and translational freedom of a joint by specifying its degree of freedom. For the rigid body we decided to differentiate only between hinge joints and ball-and-socket joints. The hinge joint is therefore constrained by having only one axis around which it can rotate. This constraint to the hinge joint is not enough, we still need to constrain the angle around which we can rotate around this axis. This in order to create natural configurations for the hinge joints. For example, a human hinge joint can never rotate more than 360 degrees around its hinge axis, the rotation angle is therefore limited to a smaller range. The ball-and-socket joint has rotational freedom over all three basis axes in $\mathbb{R}^3$, but a ball-and-socket joint can not rotate over the complete sphere. Therefore the ball-and-socket joint needs to be constrained as well. In section 4.5 we show how to constrain the hinge and ball-and-socket joint with their rotation angles. When we constrain the joints, we ignore the fact that in nature the rotational limits of a joint depend on the configuration of the other joints in the chain.

4 Quaternions and Dual Quaternions

We will use a quaternion to represent an orientation and we will use a dual quaternion to represent a translation and a rotation. Therefore, this chapter starts with the defenitions and properties of quaternions and dual quaternions. Furthermore, it will be explained how we can perform rotations and translations with dual quaterions. Also, the dual quaternions can be used to represent the skeleton structure of a virtual character. The rotational parts of those dual quaternions represent the orientations of the joints which determine the configuration of the skeleton. To determine the three parameters on which the orientation of a joint depends, the exponential map representation of a quaternion is given. We use this to define the derivative of a quaternion and dual quaternion with respect to the log value of a quaternion. Furthermore, contact limits will be defined to constrain the rotational freedom of the joints. Finally, the interpolation methods for quaternions and dual quaternions will be discussed.

4.1 Definitions and properties of the (dual) quaternion

Before we give the definition of a quaternion we will first give the definition of a complex number. The quaternions are an extension to the complex numbers.

Definition 4.1.1. (Complex number) *A complex number z is given by*

$$z = a + b\mathbf{i}$$

with $a, b \in \mathbb{R}$ and $\mathbf{i}$ the imaginary unit with $\mathbf{i}^2 = -1$. We call a the real part and b the complex part of z.

Complex numbers have the basis elements $\mathbf{1} = (1, 0)$ and $\mathbf{i} = (0, 1)$. For quaternions this is extended to the basis elements $\mathbf{1} = (1, 0, 0, 0)$, $\mathbf{i} = (0, 1, 0, 0)$, $\mathbf{j} = (0, 0, 1, 0)$ and $\mathbf{k} = (0, 0, 0, 1)$. With these basis elements the quaternion is defined as follows.

Definition 4.1.2. (Quaternion) *A quaternion q is represented by*

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

with $w, x, y, z \in \mathbb{R}$ and basis elements $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$ for which $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$ and $\mathbf{ijk} = -1$.

In the literature quaternions are represented in many different ways. Notation 4.1.3 gives the representations of a quaternion which are used in this paper. We will use different notations in order to keep expressions with quaternions as short as possible.

Notation 4.1.3. *The following representations of a quaternion q are equivalent*
(a) $q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$
(b) $q = (w, x, y, z)$
(c) $q = (r, \vec{v})$ *with scalar part $r = w$ and vector part $\vec{v} = (x, y, z)$.*

The quaternion with zero scalar part is used a lot when using quaternions for rotations. Therefore this quaternion gets its own notation, see notation 4.1.4.

Notation 4.1.4. *For the quaternion with zero scalar part and vector $\vec{u} = (u_1, u_2, u_3)$ use $\vec{\mathbf{u}} = (0, \vec{u}) = 0 + u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}$.*

Now that we know the definition of a quaternion we can define the collection of all quaternions.

Definition 4.1.5. (Set of quaternions) *The set of quaternions H is given by*

$$H = \{(w, x, y, z) | w, x, y, z \in \mathbb{R}\}$$

In order to do computations with quaternions we need to define the addition and multiplication of quaternions.

Definition 4.1.6. *Given $(r_1, \vec{v}_1), (r_2, \vec{v}_2) \in H$, addition and multiplication are defined by*
(a) $(r_1, \vec{v}_1) + (r_2, \vec{v}_2) = (r_1 + r_2, \vec{v}_1 + \vec{v}_2)$
(b) $(r_1, \vec{v}_1)(r_2, \vec{v}_2) = (r_1 r_2 - \vec{v}_1 \cdot \vec{v}_2, r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$
Here we use "$\times$" for the cross product and "$\cdot$" for the dot product of two vectors.

Here multiplication from the left is not equal to multiplication from the right, which means that the multiplication of a quaternion is not commutative.

Property 4.1.7. *The multiplication of quaternions is not commutative, i.e. given $q_1, q_2 \in H$ in general $q_1 q_2 \neq q_2 q_1$.*

Proof. Since we have the following: $\mathbf{ij} = -\mathbf{ji} = 1$, $\mathbf{jk} = -\mathbf{kj} = 1$ and $\mathbf{ki} = -\mathbf{ik} = 1$, a quaternion multiplication can not be commutative. See [Dam *et al.* 1998, 10]. □

As the following proposition states, the order in which we multiply does not matter since the quaternion multiplication is associative and distributive.

Proposition 4.1. *Given $q_0, q_1, q_2 \in H$, the following holds true*
(a) $q_0(q_1 q_2) = (q_0 q_1)q_2$ *(associativity),*
(b) $q_0(q_1 + q_2) = q_0 q_1 + q_0 q_2$ *and* $(q_1 + q_2)q_0 = q_1 q_0 + q_2 q_0$ *(distributivity)*

Proof. See [Dam *et al.* 1998, 10]. □

Aside from the addition and multiplication properties, the quaternion has a norm, conjugate and inverse.

Definition 4.1.8. *We have the following definitions for $q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \in H$.*
(a) *The conjugate is defined by $q^* = w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k}$.*
(b) *The norm is defined by $||q|| = \sqrt{w^2 + x^2 + y^2 + z^2} = \sqrt{qq^*}$.*
(c) *The (multiplicative) inverse is defined by $q^{-1} = \frac{q^*}{||q||^2}$ for $q \neq 0$, i.e. $qq^{-1} = q^{-1}q = 1$.*

Now that we have defined the norm of a quaternion we can give the definition of a unit quaternion as given in definition 4.1.9. The unit quaternion is the type of quaternion that we will use for rotation.

Definition 4.1.9. *Let $q \in H$, then q is unit if $||q|| = 1$.*

We have given the definition of the set of quaternions. In the same way we can give the definition of the set of unit quaternions. The set of unit quaternions is a subset of the set of quaternions.

Definition 4.1.10. (Set of unit quaternions) *The set of unit quaternions H_0 is defined by*

$$H_0 = \{q \in H \,|\, ||q|| = 1\}$$

The multiple of two unit quaternions is a unit quaternion.

Property 4.1.11. *Given $q_0, q_1 \in H_0$, we have $q_0 q_1 \in H_0$.*

Proof. See [Dam *et al.* 1998, 14]. □

Computation of the inverse of a unit quaternion is faster than the computation of the inverse of a general quaternion.

Property 4.1.12. *For all $q \in H_0$ the inverse is given by $q^{-1} = q^*$.*

Proof. See [Dam *et al.* 1998, 14]. □

We now know the definition and properties of a quaternion and a unit quaternion. Before we will give the definition and properties of a dual quaternion we will first give the definition and properties of a dual number. Dual numbers are an extension to the real numbers.

Definition 4.1.13. (Dual number) *A dual number $\hat{z}$ is given by*

$$\hat{z} = a + b\epsilon$$

with $a, b \in \mathbb{R}$ and ϵ the dual element for which $\epsilon^2 = 0$.

The definition of the set of dual numbers is given as follows.

Definition 4.1.14. (Set of dual numbers) *The set of dual numbers $\mathbb{D}$ is given by*

$$\mathbb{D} = \{a + \epsilon b | a, b \in \mathbb{R}\}$$

The rules for addition and multiplication of dual numbers are given in the following definition.

Definition 4.1.15. *The addition and multiplication of $a_1 + \epsilon b_1, a_2 + \epsilon b_2 \in \mathbb{D}$ is defined by*
(a) $(a_1 + \epsilon b_1) + (a_2 + \epsilon b_2) = (a_1 + a_2) + \epsilon(b_1 + b_2)$
(b) $(a_1 + \epsilon b_1)(a_2 + \epsilon b_2) = (a_1 a_2) + \epsilon(a_1 b_2 + b_1 a_2)$

The square root of a dual number is given in the following property.

Property 4.1.16. *The square root of $a + b\epsilon \in \mathbb{D}$ with $a > 0$ is given by*

$$\sqrt{a + b\epsilon} = \sqrt{a} + \epsilon \frac{b}{2\sqrt{a}}$$

Proof. See [Kavan *et al.* 2008, 16]. □

One of the most useful properties of a dual number is given in the next property.

Property 4.1.17. *Given $a + \epsilon b \in \mathbb{D}$ and a function of dual argument f, we have the following equality*

$$f(a + \epsilon b) = f(a) + \epsilon b f'(a)$$

Proof. Follows from the Taylor expansion of f. See [Kavan *et al.* 2008, 16]. □

Now that we have defined a dual number as in definition 4.1.13, we can come to the definition of a dual quaternion.

Definition 4.1.18. (Dual Quaternion) *A dual quaternion $\hat{q}$ is represented by*

$$\hat{q} = q_0 + \epsilon q_\epsilon$$

with $q_0, q_\epsilon \in H$ and ϵ the dual element.

For the dual quaternions we can use different notations.

Notation 4.1.19. *The following representations of a dual quaternion $\hat{q}$ are equivalent.*
(a) $\hat{q} = q_0 + \epsilon q_\epsilon$,
(b) $\hat{q} = (q_0, q_\epsilon)$,
(c) $\hat{q} = w_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k} + w_2\epsilon + x_2\mathbf{i}\epsilon + y_2\mathbf{j}\epsilon + z_2\mathbf{k}\epsilon$ *with quaternions* $q_0 = w_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}$ *and* $q_\epsilon = w_2 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}$,
(d) $\hat{q} = \hat{w} + \hat{x}\mathbf{i} + \hat{y}\mathbf{j} + \hat{z}\mathbf{k}$ *with dual numbers* $\hat{w} = w_1 + w_2\epsilon$, $\hat{x} = x_1 + x_2\epsilon$, $\hat{y} = y_1 + y_2\epsilon$ *and* $\hat{z} = z_1 + z_2\epsilon$.

We can extend the set of quaternions to a set of dual quaternions.

Definition 4.1.20. (Set of dual quaternions) *The set of dual quaternions $H^{\mathbb{D}}$ is given by*

$$H^{\mathbb{D}} = \{\hat{q} = q_0 + \epsilon q_\epsilon | q_0, q_\epsilon \in H\}$$

In definition 4.1.21 the rules for addition and multiplication of dual quaternions are given.

Definition 4.1.21. *Given $q_0^A + \epsilon q_\epsilon^A, q_0^B + \epsilon q_\epsilon^B \in H^{\mathbb{D}}$, addition and multiplication are defined by*
(a) $(q_0^A + \epsilon q_\epsilon^A) + (q_0^B + \epsilon q_\epsilon^B) = (q_0^A + q_0^B) + \epsilon(q_\epsilon^A + q_\epsilon^B)$,
(b) $(q_0^A + \epsilon q_\epsilon^A)(q_0^B + \epsilon q_\epsilon^B) = q_0^A q_0^B + \epsilon(q_0^A q_\epsilon^B + q_\epsilon^A q_0^B)$

A dual quaternion has a norm, inverse and two different conjugates. The dual quaternion has a quaternion conjugate and a complex conjugate. We always have to specify which of the two conjugates is used.

Definition 4.1.22. *We have the following definitions for $\hat{q} = q_0 + \epsilon q_\epsilon \in H^{\mathbb{D}}$*
(a) *The quaternion conjugate is defined by* $\hat{q}^* = q_0^* + \epsilon q_\epsilon^*$,
(b) *The complex conjugate is defined by* $\bar{\hat{q}} = q_0 - \epsilon q_\epsilon$,
(c) *The norm is defined by* $||\hat{q}|| = \sqrt{\hat{q}\hat{q}^*} = \sqrt{\hat{q}^*\hat{q}}$,
(d) *The inverse is defined by* $\hat{q}^{-1} = \frac{\hat{q}^*}{||\hat{q}||^2}$ *for* $\hat{q} \neq 0$.

For the quaternions we defined the unit quaternions. Also for the dual quaternions we have a unit dual quaternion.

Definition 4.1.23. *A unit dual quaternion $\hat{q}$ is a dual quaternion with $||\hat{q}|| = 1$.*

We define the set of unit dual quaternion which is a subset of the set of dual quaternions.

Definition 4.1.24. (Set of unit dual quaternions) *The set of unit dual quaternions $H_0^{\mathbb{D}}$ is given by*

$$H_0^{\mathbb{D}} = \left\{\hat{q} \in H^{\mathbb{D}} | \, ||\hat{q}|| = 1\right\}$$

To know when a dual quaternion is unit we have the following correspondence.

Theorem 4.1.25. *We have $q_0 + \epsilon q_\epsilon \in H_0^{\mathbb{D}}$ if and only if $q_0 \in H_0$ and $q_0 q_\epsilon^* + q_\epsilon q_0^* = 0$.*

Proof. $||\hat{q}|| = \sqrt{\hat{q}\hat{q}^*}$ and $\hat{q}\hat{q}^* = (q_0, q_\epsilon)(q_0^*, q_\epsilon^*) = (q_0 q_0^*, q_0 q_\epsilon^* + q_\epsilon q_0^*) = (1, 0)$.
See [Kavan *et al.* 2008, 17]. □

The product of two unit dual quaternions is again unit.

Property 4.1.26. *Given $\hat{q}_0, \hat{q}_1 \in H_0^{\mathbb{D}}$, we have $\hat{q}_0\hat{q}_1 \in H_0^{\mathbb{D}}$.*

Proof. See [Kavan *et al.* 2008, 17]. □

At last we have the following property for the product of two dual quaternion conjugates.

Property 4.1.27. *Given* $\hat{q}_1, \hat{q}_2 \in H^{\mathbb{D}}$*, we have* $(\hat{q}_1\hat{q}_2)^* = \hat{q}_2^*\hat{q}_1^*$.

Proof. See [Kavan *et al.* 2008, 17]. □

We now have the definitions and properties of quaternions and dual quaternions. In the next section we will show how the quaternions are used for rotation and the dual quaternions for screw motion.

4.2 Rotation and translation with (dual) quaternions

One of the properties of dual quaternions is that they can be used to represent rotations and translations. In this section we will show how this can be done. We will start by showing how a quaternion can be used to represent rotation and end by showing how we can add translation to it by creating a dual quaternion.

4.2.1 A quaternion for rotation

A unit quaternion describes a rotation by a unit vector $\vec{u} \in \mathbb{R}^3$ and angle $\theta \in [-\pi, \pi)$. Here we rotate around $\vec{u}$ with angle θ by the right hand rule as shown in figure 11. Here we rotate anticlockwise if we view from the tip of the thumb/vector. We can also use the cross product to determine the direction of rotation. In this case a rotation from vector $\vec{a}$ to vector $\vec{b}$ by angle θ will be around $\vec{u} = \vec{a} \times \vec{b}$. Here we can note that $\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$.
Now that we have defined a rotation by a unit vector and a rotation angle we can define its representation by a unit quaternion.

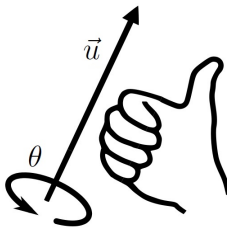


Figure 11: [5]

Definition 4.2.1. *$q \in H_0$ represents a rotation around a unit vector $\vec{u}$ with angle $\theta \in [-\pi, \pi)$ by*

$$q = \left(\cos\frac{\theta}{2}, \vec{\mathbf{u}}\sin\frac{\theta}{2}\right)$$

It is important that $\vec{u}$ is a unit vector in this definition, since this implies that q is also unit. We use the following notation for the set of unit vectors.

Notation 4.2.2. *The set of unit vectors in $\mathbb{R}^3$ is represented by $\mathbb{R}^3_u = \left\{\vec{u} \in \mathbb{R}^3 \,|\, ||\vec{u}|| = 1\right\}$.*

Having a unit quaternion q for rotation we can always find a corresponding rotation angle and unit vector as the following property states.

Property 4.2.3. *Given $q = (r, \vec{v}) \in H_0$, there exist $\theta \in [-\pi, \pi)$ and $\vec{u} \in \mathbb{R}^3_u$ such that $q = \left(\cos\frac{\theta}{2}, \vec{\mathbf{u}}\sin\frac{\theta}{2}\right)$.*

Proof. See [Dam *et al.* 1998, 14]. □

Now that we know that there exists a rotation angle and unit vector for every unit quaternion we can compute them.

Property 4.2.4. *Given $q = (r, \vec{v}) \in H_0$, the rotation angle $\theta \in [-\pi, \pi)$ and rotation vector $\vec{u} \in \mathbb{R}_u$ are given by*

$$\theta = 2\arctan 2(||\vec{v}||, r)$$
$$\vec{u} = \frac{\vec{v}}{||\vec{v}||}$$

Proof. By property 4.2.3 we have that $q = \left(\cos\frac{\theta}{2}, \vec{\mathbf{u}}\sin\frac{\theta}{2}\right)$ exists. Here $\vec{v} = \vec{u}\sin(\frac{\theta}{2})$ implies that $||\vec{v}|| = \sin(\frac{\theta}{2})$ since we require $\vec{u}$ to be unit. Hence $\vec{u} = \frac{\vec{v}}{||\vec{v}||}$ and θ follows from

$$\frac{\theta}{2} = \arctan 2\left(\sin(\frac{\theta}{2}), \cos(\frac{\theta}{2})\right) = \arctan 2\left(||\vec{v}||, r\right) \in \left[\frac{-\pi}{2}, \frac{\pi}{2}\right)$$

Note that the *arctan2* function is different from the *arctan* function.[2] □

Note that we do not use $\theta = 2\arccos(r)$ since this implies that θ can only lie in the interval $[0, \pi)$, while the possible rotation angles lie in the larger interval $[-\pi, \pi)$. However,there are cases in which we use the arccosine to compute a rotation angle. For example when we want to find the smallest positive angle between two vectors.

Property 4.2.5. *Given $u_1, u_2 \in \mathbb{R}^3_u$, the shortest angle between u_1 and u_2 is given by* $\arccos(u_1 \cdot u_2)$.

Proof. $\vec{u}_1 \cdot \vec{u}_2 = ||\vec{u}_1||\,||\vec{u}_2|| \cos(\theta)$ with θ the shortest angle between u_1 and u_2. □

We now know the correspondence between a rotation quaternion and the rotation vector and angle. The next property will explain how a unit quaternion is used in order to perform the rotation of a vector.

Property 4.2.6. *Given $q = \left(\cos(\frac{\theta}{2}), \vec{\mathbf{u}}\sin(\frac{\theta}{2})\right) \in H_0$, $p_1 \in \mathbb{R}^3$ and $\vec{\mathbf{p}}_1 = (0, \vec{p}_1) \in H$, then $\vec{p}_2 \in \mathbb{R}^3$ is vector $\vec{p}_1$ rotated around $\vec{v}$ by angle θ and can be found by*

$$(0, \vec{p}_2) = \vec{\mathbf{p}}_2 = q\vec{\mathbf{p}}_1 q^{-1}$$

Proof. See [Dam *et al.* 1998, 19]. □

Remember that the inverse of a unit quaternion q equals its complex conjugate, $q^{-1} = q^*$.

Finding the rotation quaternion that equals a succession of rotations is not that hard since we only need to multiply the successive rotation quaternions in reversed order.

Property 4.2.7. *Given $q_1, q_2 \in H_0$, the rotation by q_1 followed by rotation by q_2 is equal to rotation by $q = q_2 q_1$.*

Proof. See [Dam *et al.* 1998, 21]. □

Every rotation has two unit quaternions that can represent it. This can be seen in the following property.

Property 4.2.8. *Let $q \in H_0$, then q and $-q$ represent the same rotation.*

Proof. This follows since $q * \vec{\mathbf{p}} * q^{-1} = (-q) * \vec{\mathbf{p}} * (-q)^{-1}$ for a unit quaternion q. □

This means that rotating around unit vector $\vec{u}_1$ by angle $\theta_1 \in [-\pi, \pi)$ is equivalent to rotating around $u_2 = -\vec{u}$ by angle $\theta_2 = -\theta_1 \in [-\pi, \pi)$. However, the result of rotations q and $-q$ are the same, there are cases in which it is important to take the right representation. For example to find the smallest angle between two orientations. An orientation can be represented by a rotation and therefore be represented by a unit quaternion.

Property 4.2.9. *Given $q_1, q_2 \in H_0$, the rotation angle between q_1 and q_2 is smaller than the rotation angle between $-q_1$ and q_2 if $q_1 \cdot q_2 > 0$.*

Proof. The rotation angle between q_1 and q_2 is given by $2\arccos(p_1 \cdot p_2)$, since $q_1 \cdot q_2$ is the scalar part of $q_1 q_2^*$. $\arccos(q_1 \cdot q_2) < \arccos(q_1 \cdot q_2)$ if $q_1 \cdot q_2 > 0$, meaning that the rotation angle between q_1 and q_2 is smaller than the rotation angle between $-q_1$ and q_2 if $q_1 \cdot q_2 > 0$. □

This property is important when we do spherical interpolations as described in section 4.6.

[2]See for the definition of arctan2 *http://en.wikipedia.org/wiki/Atan2* (Accessed 24 august 2012)

4.2.2 A dual quaternion for rotation and translation

A dual quaternion is used to represent a rotation combined with a translation. This can be obtained in two different ways. We can either first rotate and then translate or first translate and then rotate. This is given in definition 4.2.10.

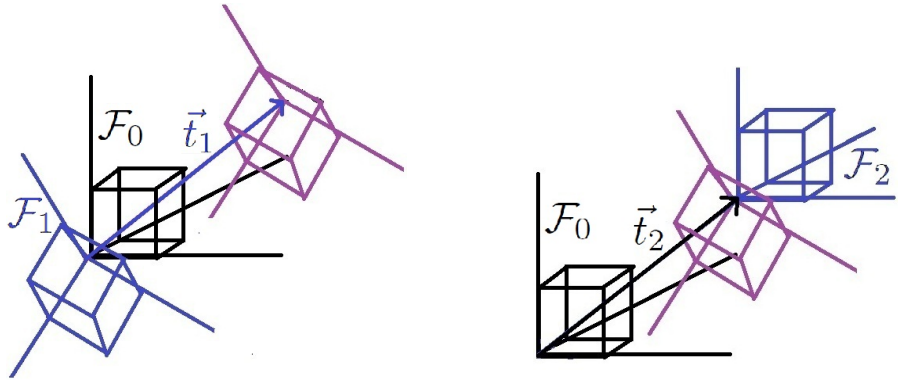
Definition 4.2.10. *Let $q_0 \in H_0$, $\hat{q} \in H_0^{\mathbb{D}}$ and $\vec{t}_1, \vec{t}_2 \in \mathbb{R}^3$. Let $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2$ be coordinate systems, where $\mathcal{F}_1$ is $\mathcal{F}_0$ rotated by q_0 and $\mathcal{F}_2$ is $\mathcal{F}_0$ translated by $\vec{t}_2$.*
(a) *The rotation q_0 in $\mathcal{F}_0$ followed by the translation $\vec{t}_1$ in $\mathcal{F}_1$ is represented by*

$$\hat{q} = q_0 + \epsilon \frac{1}{2} q_0 \vec{\mathbf{t}}_1$$

(b) *The translation $\vec{t}_2$ in $\mathcal{F}_0$ followed by the rotation q_0 in $\mathcal{F}_2$ is represented by*

$$\hat{q} = q_0 + \epsilon \frac{1}{2} \vec{\mathbf{t}}_2 q_0$$

In figure 12 we can see the visual interpretation of this definition. It shows the difference between first rotating and then translating and first translating and then rotating. It is important to notice that the rotation quaternions in both cases are the same though they are performed in different coordinate systems. The translation vectors in the two cases are different due to the fact that the translations are performed in different coordinate systems.



(a) Rotation followed by translation (b) Translation followed by rotation

Figure 12

In our rigid body we had a chain of joints which have translations and rotations with respect to their parent joints. We can now understand the choice to first translate and then rotate as in definition 4.2.10(b). The reason lays in the independency of the translation vector t_1 with respect to the rotation quaternion q_0. This enables us to let the translation vectors be fixed, which we want since the bone lengths are fixed as well. Hereby we can manipulate the rotation quaternions, meaning that we manipulate the orientations of the joints, without influencing the translations. Notice that in definition 4.2.10(a) the translation vector does depend on the rotation quaternion q_0 since the coordinate system that we translate in $\mathcal{F}_1$ depends on q_0.

For rotations we used unit quaternions, in the same way we use unit dual quaternions for rotations

combined with translations. The dual quaternion for rotation and translation is unit, because its rotation part is unit as the next property states.

Property 4.2.11. *Given $q_0 \in H_0$ and $\vec{t} \in \mathbb{R}^3$, we have $\hat{q} = q_0 + \frac{1}{2}\epsilon\vec{\mathbf{t}}q_0 \in H_0^{\mathbb{D}}$.*

Proof. $q_0(\frac{1}{2}\vec{\mathbf{t}}q_0)^* + (\frac{1}{2}\vec{\mathbf{t}}q_0)q_0^* = \frac{1}{2}(\vec{\mathbf{t}}q_0q_0^*)^* + \frac{1}{2}(\vec{\mathbf{t}}q_0q_0^*) = \frac{1}{2}\vec{\mathbf{t}}^* + \frac{1}{2}\vec{\mathbf{t}} = 0$. By theorem 3.1.18 this implies that $||\hat{q}|| = 1$ □

The following property shows how to rotate and translate a vector with a dual quaternion.

Property 4.2.12. *Let $\vec{v}_1, \vec{v}_2 \in \mathbb{R}^3$, $\hat{q} \in H_0^{\mathbb{D}}$ and $\hat{v}_1 = 1 + \epsilon\vec{\mathbf{v}}_1 \in H^{\mathbb{D}}$. Then $\vec{v}_2$ which is $\vec{v}_1$ rotated and translated by $\hat{q}$ can be obtained by*

$$1 + \epsilon\vec{\mathbf{v}}_2 = \hat{q}\hat{v}_1\bar{\hat{q}}^*$$

Proof. See (Kavan 2008, pag 17). □

Finding the rotation and translation dual quaternion that is equivalent to a succession of rotations and translations is not that hard since we only need to multiply the successive rotation and translation dual quaternions in reversed order.

Property 4.2.13. *Given $\hat{q}_1, \hat{q}_2 \in H_0^{\mathbb{D}}$, the rotating and translating by $\hat{q}_2$ followed by the rotation and translation by $\hat{q}_1$ equals rotation and translation by $\hat{q} = \hat{q}_1\hat{q}_2$.*

Proof.

$$\hat{q}_1\left(\hat{q}_2(1 + \frac{1}{2}\vec{\mathbf{v}})\bar{\hat{q}}_2^*\right)\bar{\hat{q}}_1^* = (\hat{q}_1\hat{q}_2)(1 + \frac{1}{2}\vec{\mathbf{v}})(\bar{\hat{q}}_2^*\bar{\hat{q}}_1^*) = (\hat{q}_1\hat{q}_2)(1 + \frac{1}{2}\vec{\mathbf{v}})(\overline{\hat{q}_1\hat{q}_2})^*$$

□

When we have a dual quaternion for rotation and translation we can extract the rotation quaternion and translation vector from it.

Property 4.2.14. *Given $\hat{q} = q_0 + \epsilon q_\epsilon \in H_0^{\mathbb{D}}$.*
(a) *$\hat{q}$ represents rotation by q_0 followed by translation by $\vec{\mathbf{t}} = 2q_0^{-1}q_\epsilon$.*
(b) *$\hat{q}$ represents translation by $\vec{\mathbf{t}} = 2q_\epsilon q_0^{-1}$ followed by by rotation q_0.*

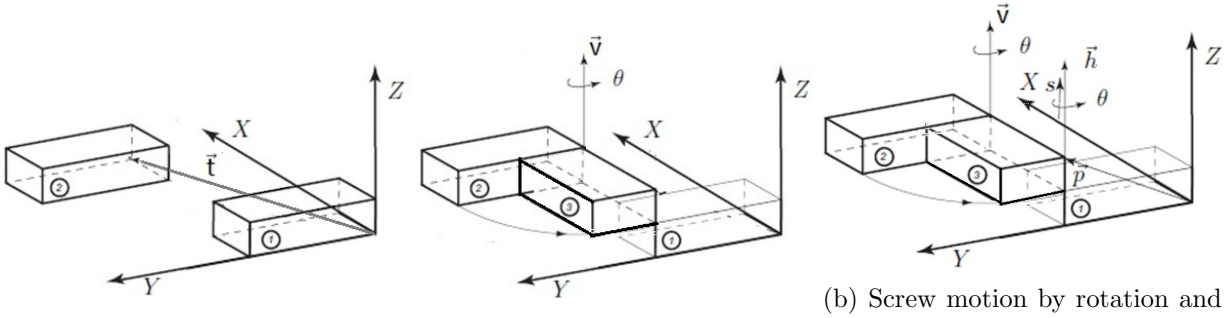
Proof. (a) $\hat{q} = q_0 + \frac{1}{2}\epsilon q_0\vec{\mathbf{t}}$. Hence $q_\epsilon = \frac{1}{2}q_0\vec{\mathbf{t}}$ gives $\vec{\mathbf{t}} = 2q_0^{-1}q_\epsilon$.
(b) $\hat{q} = q_0 + \frac{1}{2}\epsilon\vec{\mathbf{t}}q_0$. Hence $q_\epsilon = \frac{1}{2}\vec{\mathbf{t}}q_0$ gives $\vec{\mathbf{t}} = 2q_\epsilon q_0^{-1}$. □

We have now shown all important properties when we want to rotate and translate with a dual quaternion.

4.2.3 A dual quaternion for screw motion

In the previous section we saw how we could make a screw motion by successive rotation and translation. Instead we could also make one direct screw motion by defining a screw axis around which we rotate and translate. The direct screw uncouples the rotational and translational part of a screw. The next example illustrates the difference between a screw made by succesive rotation and translation and a direct screw.

Example
In figure 13a we see how we can make a screw motion by first translation followed by rotation as we described in the previous section. We first translate the brick by $\vec{t}$ from position 1 to position 2 and then rotate the brick around vector $\vec{v}$ with angle θ from position 2 to position 3. Instead we could have done a direct screw motion as in figure 13b. Here we screw the brick from position 1 to position 3 along line h, where the screw angle equals θ and the translation factor is s.



(a) Screw motion by first translation and then rotation

(b) Screw motion by rotation and translation along one single axis

Figure 13: Two different ways a dual quaternion represents a screw motion.[6]

The direct screw motion, where we rotated and translated along one line can be described by a dual quaternion as in the following definition.

Definition 4.2.15. *Let* $\hat{q} = \cos(\frac{\hat{\theta}}{2}) + \hat{h}\sin(\frac{\hat{\theta}}{2}) \in H_0^{\mathbb{D}}$*, then* $\hat{q}$ *represents a screw motion with dual screw angle* $\hat{\theta} = \theta + \epsilon s$*, with rotation angle* θ *and translation scalar* s *and screw axis* $\hat{h} = \vec{\mathbf{v}} + \epsilon\vec{\mathbf{d}} \in H^{\mathbb{D}}$ *given by Plücker coordinate* $(\vec{v}; \vec{d})$*.*

This means in general that we get a screw motion as shown in figure 14.

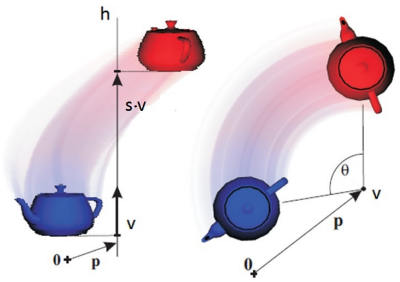


Figure 14: Screw motion[7]

The dual quaternion for a direct screw as in definition 4.2.15 equals the dual quaternion for a screw with translation and rotation as in definition 4.2.10. Property 4.2.16 shows how these dual quaternions correspond to each other.

Property 4.2.16. *Let* $\hat{q} = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}})(\cos(\frac{1}{2}\theta), \vec{v}\sin(\frac{1}{2}\theta))$*, then* $\hat{q} = \cos(\hat{\theta}) + \hat{h}\sin(\hat{\theta})$ *with*
(a) $\hat{\theta} = \theta + \epsilon||\vec{t}||$ *and* $\hat{h} = \frac{\vec{\mathbf{t}}}{||\vec{t}||} + \epsilon\vec{0}$ *if* $\theta = 2\pi k$ *for some* $k \in \mathbb{Z}$*,*
(b) $\hat{\theta} = \theta + \epsilon(\vec{t} \cdot \vec{v})$ *and* $\hat{h} = \vec{v} + \epsilon\frac{1}{2}((\vec{v} \times \vec{t})\cot(\frac{\theta}{2}) + \vec{t}) \times \vec{v}$ *if* $\theta \neq 2\pi k$ *for all* $k \in \mathbb{Z}$*.*

Proof. See (Kavan 2008, pg.18). □

The line along we screw in theorem 4.2.15 is given by Plücker coordinates. Therefore we briefly show what a Plücker coordinate is.

Plücker coordinates
A Plücker coordinate is given by a couple $(\vec{d}; \vec{v})$ with $\vec{d}, \vec{v} \in \mathbb{R}^3$. The Plücker coordinate represents a line h which is defined by all points $\vec{p} \in \mathbb{R}^3$ for which $\vec{d} = \vec{p} \times \vec{v}$. We can see this in figure 15. Line h lays always parallel to vector $\vec{v}$.

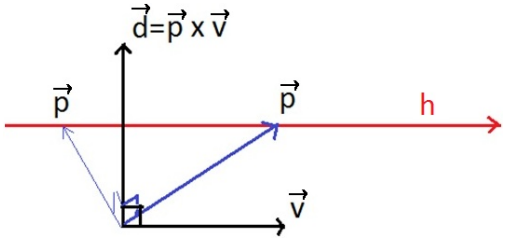


Figure 15: Plücker coordinates

Now the dual quaternion $\hat{h} = \vec{\mathbf{v}} + \epsilon\vec{\mathbf{d}}$ corresponds to the Plücker coordinate $(\vec{d}; \vec{v})$.

4.3 The exponential map and the log of a quaternion

In this section we will give an explanation of the exponential map and the log of a quaternion. We will later see how this can be used to get the three variables that determine the value of a quaternion. We will first go back to some basic concepts and end with the expression for the quaternion.

We now give Euler's formula which is the well known exponential map of a complex number.

Theorem 4.3.1. (Euler's formula) *Given $\theta \in \mathbb{R}$,*

$$e^{\mathbf{i}\theta} = \cos(\theta) + \mathbf{i}\sin(\theta)$$

Proof. Follows from the infinite sum representation of the exponential function. □

This formula can be generalized by an extension to the complex numbers; aside for the basic vector $\mathbf{i}$ we include the basic vectors $\mathbf{j}$ and $\mathbf{k}$. This brings us to the next theorem.

Theorem 4.3.2. *Given unit vector $\vec{u} = (u_1, u_2, u_3) \in \mathbb{R}^3$ and angle $\theta \in \mathbb{R}$,*

$$e^{(u_1\mathbf{i}+u_2\mathbf{j}+u_3\mathbf{k})\theta} = \cos(\theta) + (u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k})\sin(\theta)$$
$$e^{\vec{\mathbf{u}}\theta} = \cos(\theta) + \vec{\mathbf{u}}\sin(\theta)$$

Proof. Write this out as an infinite sum also. □

Aside from the exponential map of $\vec{\mathbf{u}}\theta$, we can take the exponential map of an arbitrary vector $\vec{w}$.

Theorem 4.3.3. *Given vector $\vec{w} \in \mathbb{R}^3$,*

$$e^{\vec{\mathbf{0}}} = (1, \vec{0})$$
$$e^{\vec{\mathbf{w}}} = \cos(||\vec{w}||) + \frac{\vec{\mathbf{w}}}{||\vec{w}||}\sin(||\vec{w}||) \quad \textit{if} \quad \vec{\mathbf{w}} \neq \vec{\mathbf{0}}$$

Proof. Follows directly from theorem 4.3.2 by taking $\vec{\mathbf{u}} = \frac{\vec{\mathbf{w}}}{||\vec{w}||}$ and $\theta = ||\vec{w}||$. □

We can note that the exponential map of a quaternion with zero scalar part results in a unit quaternion. The other way around is also true, every unit quaternion has an exponential map representation.

Theorem 4.3.4. *The exponential map representation of $q = \cos(\frac{1}{2}\theta) + \vec{\mathbf{u}}\sin(\frac{1}{2}\theta) \in H_0$ is given by*

$$q = e^{\frac{1}{2}\vec{\mathbf{u}}\theta}$$

Proof. Follows directly from theorem 4.3.2 with $\theta = \frac{1}{2}\theta$. □

The exponential map of a quaternion $\vec{\mathbf{w}}$ has singularities when $||\vec{w}|| = 2n\pi$ for $n \in \mathbb{N}\setminus\{0\}$. A rotation of angle $2n\pi$ around an arbitrary axis results in no rotation at all and is therefore equivalent to a rotation of angle 0. To avoid these singularities we restrict $||\vec{w}||$ to the range $[0, \pi]$. We do this by replacing $\vec{w}$ by $(1 - \frac{2\pi}{||\vec{w}||})\vec{w}$ when $||\vec{w}||$ is larger than π. This way we restrict the rotation angle to be in the range $[-\pi, \pi)$ with which we can still get all the possible rotations.

Since the unit quaternion has an exponential map representation, we can take the logarithm of this unit quaternion.

Theorem 4.3.5. *The logarithm of* $q = \cos(\frac{1}{2}\theta) + \vec{\mathbf{u}}\sin(\frac{1}{2}\theta) \in H_0$ *is given by*

$$\log q = \frac{1}{2}\vec{\mathbf{u}}\theta$$

Proof. Follows from theorem 4.3.4 and the fact that the logarithmic function is the inverse function of the exponential function. □

From the exponential map representation of a unit quaternion we know that the unit quaternion depends only on three variables, namely the vector part of its logarithm.

We use the following convention for the power of a unit quaternion.

Definition 4.3.6. *Let* $q \in H_0$ *and* $t \in (0, 1)$*, then* q *to the power* t *is given by*

$$q^t = e^{t\log(q)}$$

The next property shows that the t-th power of a rotation quaternion represents the fraction of a rotation, where t is the fraction of the rotation angle of the quaternion.

Property 4.3.7. *Let* $q \in H_0$ *and* $t \in (0, 1)$*, then* $q^{1-t}q^t = q^t q^{1-t} = q$*.*

Proof. See [Dam *et al.* 1998, 16] □

We use this property for the spherical linear interpolation which we will describe in section 4.6.

The dual quaternion also has an exponential map representation as the following definition gives.

Definition 4.3.8. *The exponential map representation of* $\hat{q} = \cos(\frac{\hat{\theta}}{2}) + \hat{\mathbf{h}}\sin(\frac{\hat{\theta}}{2})$ *is given by*

$$\hat{q} = e^{\hat{\mathbf{h}}\frac{\hat{\theta}}{2}}$$

The logarithm of a dual quaternion is given by the following definition.

Definition 4.3.9. *The logarithm of a dual quaternion* $\hat{q} = \cos(\frac{\hat{\theta}}{2}) + \hat{\mathbf{h}}\sin(\frac{\hat{\theta}}{2})$ *is given by*

$$\log(\hat{q}) = \hat{\mathbf{h}}\frac{\hat{\theta}}{2}$$

This means that the unit dual quaternion depends on six variables, three for its rotation part and three for its translation part.

Property 4.3.10. *Let* $\hat{q} \in H_0^{\mathbb{D}}$ *and* $t \in (0, 1)$*, then* $\hat{q}$ *to the power* t *is defined by*

$$\hat{q}^t = e^{t\log(\hat{q})}$$

4.4 Derivative of a (dual) quaternion

In the previous section we saw that a quaternion equals the exponential map of its logarithm. The logarithm of a unit quaternion depends on three independent variables. This means that the unit quaternion depends on only three variables. When we want to study the change of a quaternion by its variables, then we have to study the derivative of a quaternion with respect to its log values.

We start with giving the definition of a quaternion with respect to a real variable.

Definition 4.4.1. (derivative of a quaternion) *Given $t \in \mathbb{R}$, the derivative of $q(t) = w_0(t) + w_1(t)\mathbf{i} + w_2(t)\mathbf{j} + w_3(t)\mathbf{k} \in H_0$ is given by*

$$\frac{\partial q(t)}{\partial t} = \frac{\partial w_0(t)}{\partial t} + \frac{\partial w_1(t)}{\partial t}\mathbf{i} + \frac{\partial w_2(t)}{\partial t}\mathbf{j} + \frac{\partial w_3(t)}{\partial t}\mathbf{k}$$

We note that the derivative of a unit quaternion is a quaternion, but in general not a unit quaternion. The following theorem shows that the product rule as we know for real functions is still applicable for quaternions.

Theorem 4.4.2. (product rule for quaternions) *Given $t \in \mathbb{R}$ and $q_1(t), q_2(t) \in H_0$, we have the following equality*

$$\frac{\partial \left(q_1(t)q_2(t)\right)}{\partial t} = \left(\frac{\partial q_1(t)}{\partial t}\right) q_2(t) + q_1(t) \left(\frac{\partial q_2(t)}{\partial t}\right)$$

Proof. See article (Dam *et al.*,pg 24). □

Also the chain rule as we know for real functions is still applicable for quaternions.

Theorem 4.4.3. (chain rule for quaternions) *Given $t \in \mathbb{R}$, $q_1 : \mathbb{R} \to H_0$ and $q_2 : H_0 \to H_0$, the following equality holds*

$$\frac{\partial}{\partial t} q_2(q_1(t)) = q_2'(q_1(t)) \frac{\partial}{\partial t} q_1(t)$$

Proof. See article (Dam *et al.*,pg 24) □

In order to show that we can take the derivative of a dual quaternion we first need to define the derivative of a dual number, as the following definition gives.

Definition 4.4.4. (Derivative of a dual number) *Given $t \in \mathbb{R}$, the derivative of a dual number $z(t) = a(t) + b(t)\epsilon$ is given by*

$$\frac{\partial z(t)}{\partial t} = \frac{\partial a(t)}{\partial t} + \frac{\partial b(t)}{\partial t}\epsilon$$

This leads us to the definition of the derivative of a dual quaternion.

Definition 4.4.5. (Derivative of a dual quaternion) *Given $t \in \mathbb{R}$ the derivative of dual quaternion $\hat{q}(t) = q_0(t) + q_\epsilon(t)\epsilon$ is given by*

$$\frac{\partial \hat{q}(t)}{\partial t} = \frac{\partial q_0(t)}{\partial t} + \frac{\partial q_\epsilon(t)}{\partial t}\epsilon$$

The product rule is applicable to dual quaternions as the following theorem states.

Theorem 4.4.6. (Product rule for dual quaternions) *Given $t \in \mathbb{R}$ and dual quaternions $\hat{q}_1(t)$ and $\hat{q}_2(t)$, the following equation holds*

$$\frac{\partial \hat{q}_1(t)\hat{q}_2(t)}{\partial t} = \frac{\partial \hat{q}_1(t)}{\partial t}\hat{q}_2(t) + \hat{q}_1(t)\frac{\partial \hat{q}_2(t)}{\partial t}$$

Proof. Write it out with $q_1(t) = q_0^I + q_\epsilon^I \epsilon$ and $q_2(t) = q_0^{II} + q_\epsilon^{II}\epsilon$, use the chain rule for quaternions and keep in mind that $\epsilon^2 = 0$. □

Now that we have seen that we can take the derivative of a quaternion with respect to a real variable, we can also define the derivative of a quaternion with respect to its log. This is possible since the quaternion depends on its log values by its exponential map representation and the log of a quaternion depends on three real variables. With this knowledge the following theorem is developed to derive the partial derivatives of a quaternion with respect to its log values.

Theorem 4.4.7. *Let $\vec{w} = (w_1, w_2, w_3) \in \mathbb{R}^3$ and let $q = q(\vec{w}) = e^{\vec{\mathbf{w}}} \in H_0$, then the partial derivatives of q to $\vec{w}$ for $||\vec{w}|| \neq 0$ are given by*

$$\frac{\partial q(\vec{w})}{\partial w_k} = (a_k - w_k)\frac{\sin(||\vec{w}||)}{||\vec{w}||} + w_k \vec{\mathbf{w}} \left(\frac{\cos(||\vec{w}||)}{||\vec{w}||^2} - \frac{\sin(||\vec{w}||)}{||\vec{w}||^3} \right)$$

and the partial derivatives of q to $\vec{w}$ for $||\vec{w}|| = 0$ are given by

$$\frac{\partial q(\vec{w})}{\partial w_k} = a_k$$

for all $k \in \{1, 2, 3\}$, where $a_k = \mathbb{1}_{(k=1)}\mathbf{i} + \mathbb{1}_{(k=2)}\mathbf{j} + \mathbb{1}_{(k=3)}\mathbf{k}$.

Proof. For the first case with $||\vec{w}|| \neq 0$ we write q as in theorem 4.3.3 and derive the partial derivatives of it as in theorem 4.4.1.
For the case where $||\vec{w}|| = 0$ we notice that $\lim_{||\vec{w}|| \to 0} e^{\vec{\mathbf{w}}} = 1 = e^{\vec{\mathbf{0}}}$. Therefore, we take as its derivative $\lim_{||\vec{w}|| \to 0} \left((a_k - w_k)\frac{\sin(||\vec{w}||)}{||\vec{w}||} + w_k \vec{\mathbf{w}} \left(\frac{\cos(||\vec{w}||)}{||\vec{w}||^2} - \frac{\sin(||\vec{w}||)}{||\vec{w}||^3} \right) \right) = a_k$. □

When we have a unit dual quaternion and we wish to take its derivative to the log values of its rotation part, then we can use the product rule as in theorem 4.4.6 to otain the following equality.

$$\frac{\partial}{\partial \vec{w}} \left((1 + \frac{1}{2}\epsilon\vec{\mathbf{t}})e^{\vec{\mathbf{w}}} \right) = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}})\frac{\partial}{\partial \vec{w}} \left(e^{\vec{\mathbf{w}}} \right)$$

When we take the product of two unit dual quaternions $\hat{q}_1$ and $\hat{q}_2$ of which only $\hat{q}_2$ has a dependency on the log of its rotation quaternion $\vec{w}$, then we obtain the following equality.

$$\frac{\partial}{\partial \vec{w}} \left(\hat{q}_1 \hat{q}_2(\vec{w}) \right) = \hat{q}_1 \frac{\partial}{\partial \vec{w}} \left(\hat{q}_2(\vec{w}) \right)$$

We will use the derivative of a quaternion to its log value and those last two properties when we derive the jacobian matrix in chapter 5.

4.5 Joint Constraints

In this section we explain how we initialize our skeleton with local coordinate systems at each joint. Those local coordinate systems will be used to describe the relative orientation of the joints. We show how to do this in a proper way such that we can constrain the freedom of the joints to at most three variables, called the joint limits. The decomposition of a quaternion can then be used to see if we are satisfying the given constraints of our joint limits. We end by giving the algorithm for clamping our rotation within the joint limits.

4.5.1 Initializing the skeleton

In chapter 2 we explained how the orientation of a joint is given as a relative orientation between its own coordinate system and its parent joints coordinate system. In order to do this we first need to define the local coordinate systems of the joints.
We let the z-axis be the axis laying in the positive direction of the child joint. For a joint without a child, we consider the end effector which is connected to it as the child joint. We let the y-axis be the axis laying perpendicular to the z-axis and perpendicular to the axis laying in the positive direction of the parent joint. In order to get a coordinate system we define the x-axis such that it is perpendicular to the y- and z-axis. Therefore, we define the x-axis by the cross product of the z- and y-axis, $\vec{x} = \vec{z} \times \vec{y}$. Figure 16 shows this local coordinate system of a joint.

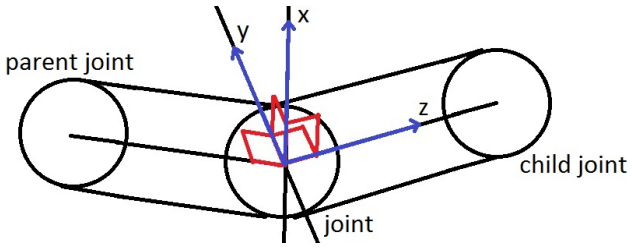


Figure 16: The local coordinate system of a joint

The orientation of the local coordinate system with respect to the parent joints coordinate system depends on the configuration of joint, but the local coordinate system itself does not depend on the configuration of the joint.
Once all joints have a local coordinate system, we can determine the dual quaternions which describe the relative orientation and translation between the joint and his parent joint.
We end by defining the initial configuration of the joints, which is determined by the initial quaternions which describes the relative orientation between the joints local coordinate system and its parents local coordinate system. The choice of the initial configuration of the joints depends on the joint limits which we will define next.

4.5.2 Joint limits

As we said in chapter 2 we consider two different joint types, the hinge joint and the ball-and-socket joint. We will constrain each joint type in its own way.

The hinge joint is a joint with one degree of freedom, meaning that it can rotate around a single axis, the hinge axis. As we can see in figure 16 the hinge axis is the y-axis of the local coordinate

system of the joint. To constrain the hinge joint we need to give limits to the rotation angle around this hinge axis. This means that given an initial configuration of the joint we determine a minimum rotation angle α_{hinge} and a maximum rotation angle β_{hinge} of which we allow rotation around the y-axis. Here we take as initial configuration of the joint a relative orientation which is in the range of natural orientations for this joint. We want the range of any rotation angle to lay within $[-\pi, \pi)$. Therefore we define α_{hinge} and β_{hinge} such that any rotation of the initial relative orientation of the joint around the hinge axis with rotation angle θ_{hinge} is such that $-\pi < \alpha_{\text{hinge}} \leq \theta_{\text{hinge}} \leq \beta_{\text{hinge}} < \pi$.

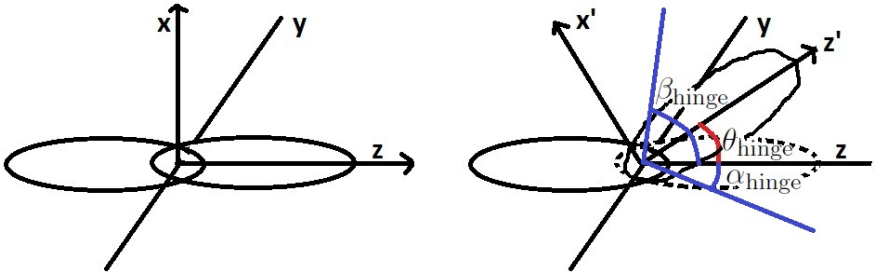


Figure 17: hinge limits

In figure 17 we see how the initial orientation of the local coordinate system xyz with respect to its parents local coordinate system gets rotated by angle $\theta_{\text{hinge}} \in [\alpha_{\text{hinge}}, \beta_{\text{hinge}}]$ around the y-axis. This gives the new orientation represented by the local coordinate system denoted by the x'-, y- and z'-axis with respect to the parents local coordinate system.

A ball-and-socket joint has three degrees of freedom. We saw that the hinge joint, which has only one degree of freedom, needs two limits to constrain. You might think that this means that we need six limits to constrain the ball-and-socket joint, but only three are needed.
The unconstrained ball-and-socket joint can rotate over the complete sphere. To constrain the ball-and-socket joint we allow only rotation in the cone of angle γ. This cone is the range in which the z-axis of the joints local coordinate system can rotate relative to its parents local coordinate system. We need to orientate this cone with respect to the parents local coordinate system. We do this by placing the z-axis through the centre of the cone. This makes the relative orientation of the cone to be the same as the relative orientation of the z-axis. This way we have found the initial orientation of the local coordinate system of the ball-and-socket joint. We can see this in figure 18.

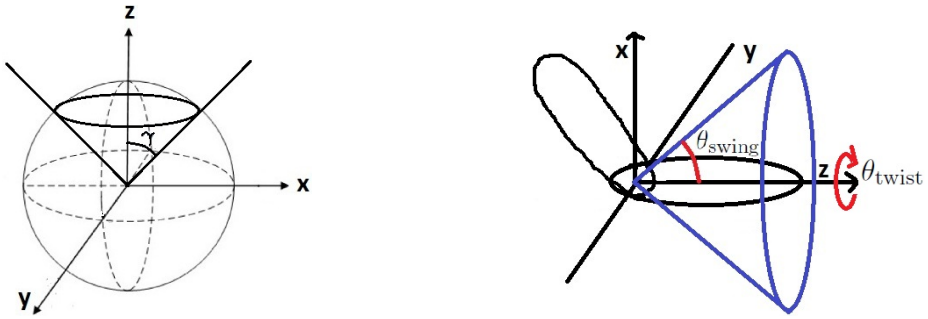


Figure 18: Constraint of the ball-and-socket joint

When we rotate the initial orientation of the local coordinate system of a joint, then we can divide

this rotation by a swing and twist component. Here the twist rotation is the rotation of the orientation of the local coordinate system around the z-axis. The swing rotation is the rotation of the orientation of the local coordinate system which rotates the z-axis in a direction in the x, y-plane. We constrain the swing rotation angle θ_{swing} by a maximum swing angle γ_{swing}. The twist rotation angle θ_{twist} gets constrained in an equivalent way as we constrained the hinge joint by giving it a minimum twist angle α_{twist} and maximum rotation angle β_{twist}. We note that there is no dependency between angle with which we can swing and the angle with which we can twist. We now have a way to constrain the ball-and-socket joint, by constraining its swing and twist angle such that $0 \leq \theta_{\text{swing}} \leq \gamma_{\text{swing}} < \pi$ and $-\pi < \alpha_{\text{twist}} \leq \theta_{\text{twist}} \leq \beta_{\text{twist}} < \pi$.

When we perform a rotation to the initial relative orientation of a joint, then this rotation has to satisfy the joints rotational constraints. To verify whether the rotation satisfies its joint limits we need to derive the hinge or swing and twist angle of this rotation and verify whether those angles lay within the given range. In the next section we give a definition of the decomposition of a quaternion and show how we use it to derive the hinge, swing and twist angle.

4.5.3 Decomposition of a quaternion

A quaternion describes a rotation by a rotation vector $\vec{u}$ and a rotation angle θ as we saw in section 3.3. We decompose this rotation in two successive rotations which together form the original rotation. Let the first rotation part be around one specified basis axis which we call the main axis. Let the second rotation part be a rotation around a vector laying in the plane spanned by the remaining two basis axes. The resulting quaternions of this decomposition are given in the following theorem for each choice of main axis.

Theorem 4.5.1. *Let $w_0 \neq 0$. The decompositions of $q = (w_0, w_1, w_2, w_3) \in H_0$ are given by*

(a) $q_x = \left(\frac{w_0}{\sqrt{w_0^2+w_1^2}}, \frac{w_1}{\sqrt{w_0^2+w_1^2}}, 0, 0\right)$ *and* $q_{yz} = \left(\sqrt{w_0^2+w_1^2}, 0, \frac{w_0w_2-w_1w_3}{\sqrt{w_0^2+w_1^2}}, \frac{w_0w_3+w_1w_2}{\sqrt{w_0^2+w_1^2}}\right)$ *with* $q_{yz}q_x = q$,

(b) $q_y = \left(\frac{w_0}{\sqrt{w_0^2+w_2^2}}, 0, \frac{w_2}{\sqrt{w_0^2+w_2^2}}, 0\right)$ *and* $q_{xz} = \left(\sqrt{w_0^2+w_2^2}, \frac{w_0w_1+w_2w_3}{\sqrt{w_0^2+w_3^2}}, 0, \frac{w_0w_3-w_1w_2}{\sqrt{w_0^2+w_2^2}}\right)$ *with* $q_{xz}q_y = q$

(c) $q_z = \left(\frac{w_0}{\sqrt{w_0^2+w_3^2}}, 0, 0, \frac{w_3}{\sqrt{w_0^2+w_3^2}}\right)$ *and* $q_{xy} = \left(\sqrt{w_0^2+w_3^2}, \frac{w_0w_1+w_2w_3}{\sqrt{w_0^2+w_3^2}}, \frac{w_1w_3-w_0w_2}{\sqrt{w_0^2+w_3^2}}, 0\right)$ *with* $q_{xy}q_z = q$.

Here q_x, q_y and q_z denote the rotations around the x-, y- and z-axis respectively and q_{xy},q_{xz} and q_{yz} denote the rotations around a vector in the xy-, xz- and yz-plane respectively.

Proof. Simply calculate $q_{yz}q_x$, $q_{xz}q_y$ and $q_{xy}q_z$ and show that those expressions equal q. □

Since we always rotate with an angle between $-\pi$ and π, the assumption of $w_0 \neq 0$ in theorem 4.5.1 is always satisfied.

The hinge joint has only rotation freedom around the hinge axis, which is the y-axis. We constrained the rotation angle θ_{hinge} around the hinge axis by $-\pi < \alpha_{\text{hinge}} \leq \theta_{\text{hinge}} \leq \beta_{\text{hinge}} < \pi$ for given minimum and maximum rotation angles α_{hinge} and β_{hinge}. When we rotate the hinge joint with a quaternion q, then we can find its decomposition in rotations q_y and q_{xz} as in theorem 4.5.1b. Here q_y is the rotation part around the y-axis. This means that θ_{hinge} is the rotation angle of q_y. For a hinge joint there should not be rotation around the x- nor z-axis, meaning that q_{xz} should be one.

The ball-and-socket joint has rotation freedom around all axes. We decomposed the rotation of the initial relative orientation of the joint by a swing and twist component. We constrained the swing rotation angle θ_{swing} by $0 \leq \theta_{\text{swing}} \leq \gamma_{\text{swing}}$ and we constrained the twist rotation angle θ_{twist} by $-\pi < \alpha_{\text{twist}} \leq \theta_{\text{twist}} \leq \beta_{\text{twist}} < \pi$. The twist component of the rotation was the rotation around the z-axis. We can decompose the rotation quaternion q by q_z and q_{xy} as in theorem 4.5.1c. Here q_z gives the rotation part around the z-axis, meaning that θ_{twist} is the rotation angle of q_z. We remain with showing that θ_{swing} is the rotation angle of q_{xz}.
Rotation quaternion q_{xz} represents the rotation around a vector $(x_1, y_1, 0)$ in the (x, y)-plane with angle θ_{xy}. Let $(0, 0, z_1)$ be the vector representing the z-axis. The rotation of $(0, 0, z_1)$ by $q_{x,y}$ results in the vector $(y_1 z_1 \sin(\theta_{xy}), -x_1 z_1 \sin(\theta_{xy}), z_1 \cos(\theta_{xy}))$. We note that $\sqrt{(y_1 z_1 \sin(\theta_{xy}))^2 + (-x_1 z_1 \sin(\theta_{xy}))^2} = z_1 \sin(\theta_{xy})$, since $x_1^2 + y_1^2 = 1$ by the unity of a rotation vector. This means that $(y_1 z_1 \sin(\theta_{xy}), -x_1 z_1 \sin(\theta_{xy}), z_1 \cos(\theta_{xy}))$ lays on the circle in the $(x, y, z_1 \cos(\theta_{xy}))$-plane of radius $z_1 \sin(\theta_{xy})$. When we swing $(0, 0, z_1)$ with angle θ_{swing} in an arbitrary direction, then the resulting vector lays on the circle in the $(x, y, z_1 \cos(\theta_{\text{swing}}))$-plane of radius $z_1 \sin(\theta_{\text{swing}})$. Hence, θ_{swing} equals θ_{xy} which is the rotation angle of q_{xy}.

4.5.4 Clamping

In the previous section we saw how we can verify whether the rotation of the initial relative orientation of a joint satisfies its joint constraints. Once we know whether a rotation satisfies the constraints or not, we need to define what we will do with the rotation. We say that we clamp a rotation when we push the rotation within its allowed range. This means that instead of making the full rotation we stop rotating once we hit one of the joint limits. We let CLAMPED(q) be the functions which given a rotation returns a clamped rotation. When the rotation satisfies the joint constraints, then the function CLAMPED(q) gives the original rotation q back, but when the rotation does not satisfy the joint constraints then it gives the clamped rotation back.

The following algorithm gives the derivation of CLAMPED(q) for a hinge joint given a unit quaternion q and joint limits $-\pi < \alpha_{\text{hinge}} \leq \beta_{\text{hinge}} < \pi$.

Algorithm 1 Clamping of a hinge joint

Require: $q = (w_0, w_1, w_2, w_3) \in H_0$ and $-\pi < \alpha_{\text{hinge}} \leq \beta_{\text{hinge}} < \pi$
Ensure: $q_{\text{clamped}} = \text{CLAMPED}(q)$
  $q_y := (y_0, 0, y_2, 0) = (\frac{w_0}{\sqrt{w_0^2+w_2^2}}, 0, \frac{w_2}{\sqrt{w_0^2+w_2^2}}, 0)$
  $\theta_{\text{hinge}} = 2 \arctan 2(y_0, y_2)$
  if $\theta_{\text{hinge}} < \alpha_{\text{hinge}}$ **then**
    $\theta_{\text{hinge}} = \alpha_{\text{hinge}}$
  end if
  if $\theta_{\text{hinge}} > \beta_{\text{hinge}}$ **then**
    $\theta_{\text{hinge}} = \beta_{\text{hinge}}$
  end if

  return $q_{\text{clamped}} = (\cos(\frac{\theta_{\text{hinge}}}{2}), 0, \sin(\frac{\theta_{\text{hinge}}}{2}), 0)$

The following algorithm gives the derivation of CLAMPED(q) for a ball-and-socket joint given a quaternion q and joint limits $-\pi < \alpha_{\text{twist}} \leq \beta_{\text{twist}} < \pi$ and $0 < \gamma_{\text{swing}} < \pi$.

Algorithm 2 Clamping of a ball-and-socket joint

Require: $q = (w_0, w_1, w_2, w_3) \in H_0$ and $-\pi < \alpha_{\text{twist}} \leq \beta_{\text{twist}} < \pi$ and $0 < \gamma_{\text{swing}} < \pi$
Ensure: $q_{\text{clamped}} = \text{CLAMPED}(q)$

$q_z := (z_0, z_1, z_2, z_3) = \left(\frac{w_0}{\sqrt{w_0^2+w_3^2}}, 0, 0, \frac{w_3}{\sqrt{w_0^2+w_3^2}} \right)$
$q_{xy} := (s_0, s_1, s_2, s_3) = \left(\sqrt{w_0^2 + w_3^2}, \frac{w_0 w_1 + w_2 w_3}{\sqrt{w_0^2+w_3^2}}, \frac{w_1 w_3 - w_0 w_2}{\sqrt{w_0^2+w_3^2}}, 0 \right)$
$\theta_{\text{twist}} = 2 \arctan 2(z_0, z_3)$
$\theta_{\text{swing}} = 2 \arccos(s_0)$
$n := (n_1, n_2, n_3) = (s_1, s_2, s_3)$
$n = \frac{n}{||n||}$
if $\theta_{\text{twist}} < \alpha_{\text{twist}}$ **then**
  $\theta_{\text{twist}} = \alpha_{\text{twist}}$
end if
if $\theta_{\text{twist}} > \beta_{\text{twist}}$ **then**
  $\theta_{\text{twist}} = \beta_{\text{twist}}$
end if
if $\theta_{\text{swing}} > \gamma_{\text{swing}}$ **then**
  $\theta_{\text{swing}} = \gamma_{\text{swing}}$
end if
$q_{\text{twist}} = (\cos(\frac{\theta_{\text{twist}}}{2}), 0, 0, \sin(\frac{\theta_{\text{twist}}}{2}))$
$q_{\text{swing}} = e^{\frac{1}{2}\mathbf{n}\theta_{\text{swing}}}$

return $q_{\text{clamped}} = q_{\text{swing}} q_{\text{twist}}$

4.6 Interpolations

When working with inverse kinematics and computer animations it is often necessary to interpolate between rigid body transformations. Interpolation is a method to construct new data points within the range of a discrete set of known data points. Before we show how to interpolate between quaternions and dual quaternions we first show how we geometrically interpolate, meaning that we interpolate between vectors in $\mathbb{R}^3$. We give a comparison between linear interpolation, normalized linear interpolation and spherical interpolation. We end with the algorithms to find the quaternion which creates an interpolated point.

4.6.1 Geometric interpolation

We speak of a geometric interpolation when we interpolate between vectors in a d-dimensional space. Here we interpolate between a begin and end vector with fraction t, meaning that the interpolated vector lays with a factor t on the range between the begin and end vector. The following definition gives the formal definition of an interpolated vector.

Definition 4.6.1. *Given begin vector $p_0 \in \mathbb{R}^d$, end vector $p_1 \in \mathbb{R}^d$ and fraction $t \in [0,1]$, $T(p_0, p_1; t) \in \mathbb{R}^d$ is an interpolated vector if $T(p_0, p_1; 0) = p_0$, $T(p_0, p_1; 1) = p_1$ and $T(p_0, p_1, t) = T(p_1, p_0, 1-t)$.*

This definition states that the interpolated vector for a fraction $t = 0$ equals the begin vector and the interpolated vector for a fraction $t = 1$ equals the end vector. Also, this definition states that an interpolated vector remains the same when we exchange the order of begin and end vector.
Since we work in 3-dimensional space, we will for simplicity assume that we interpolate between vectors in $\mathbb{R}^3$, but in fact we can interpolate between vectors of any dimension.

We speak of linear interpolation when the interpolated vector lays on the line segment between begin and end vector where the interpolation fraction is the fraction of this line segment where the interpolated vector lays. The shorthand for linear interpolation is LERP. The following definition gives the formula for the linear interpolated vector.

Definition 4.6.2. (LERP) *Given $p_1, p_2 \in \mathbb{R}^3$ and $t \in [0,1]$,*

$$LERP(p_1, p_2; t) = (1-t)p_1 + tp_2 \in \mathbb{R}^3$$

The interpretation of LERP can be seen in figure 19, where p is the linearly interpolated vector between begin vector p_1 and end vector p_2 with fraction t. Often we interpolate between unit vectors and wish to find an interpolated vector which is a unit vector again. When we linearly interpolate between two unit vectors, then there is no guarantee that the interpolated vector is again a unit vector.
We speak of normalized linear interpolation, when the interpolated vector is the normalized version of LERP. The shorthand for normalized linear interpolation is NLERP. The following definition gives the formula for the normalized linear interpolated vector between two unit vectors.

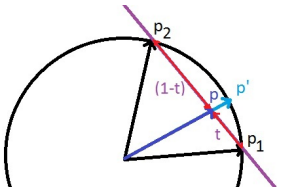


Figure 19

Definition 4.6.3. (NLERP) *Given $p_1, p_2 \in \mathbb{R}^3$, $||p_1||, ||p_2|| = 1$ and $t \in [0,1]$*

$$NLERP(p_1, p_2; t) = \frac{(1-t)p_1 + tp_2}{||(1-t)p_1 + tp_2||} \in \mathbb{R}^3_u$$

With NLERP we make sure that the interpolated vector between two unit vectors is again unit. This means that the begin, end and interpolated vector lay all on the same sphere of radius one. The interpretation of NLERP can be seen in figure 19 where p' is the normalized linear interpolated vector between vector p_1 and p_2 with fraction t. We can see that vector p' is the normalized version of vector p which was the linear interpolated vector between p_1 and p_2.

The drawback of NLERP is that t is not the fraction how far we are on the sphere of radius one between begin and end point. There is a method called spherical interpolation which does this correctly.
We speak of spherical linear interpolation when the interpolated vector of two unit vectors lays on the sphere of radius one and the interpolation fraction is the fraction that we are on this sphere. The shorthand for spherical linear interpolation is SLERP. The following definition gives the formula for an spherical linear interpolated vector.

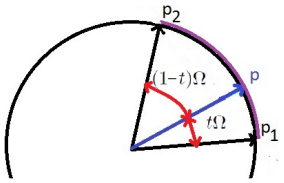


Figure 20

Definition 4.6.4. (SLERP) *Given* $p_0, p_1 \in \mathbb{R}^3_u$, $t \in [0,1]$ *and* $\Omega = \arccos(p_0 \cdot p_1) \in [0, \pi]$,

$$SLERP(p_0, p_1; t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} p_0 + \frac{\sin(t\Omega)}{\sin(\Omega)} p_1 \in \mathbb{R}^3_u$$

In figure 20 we can see the interpretation of a spherical interpolated vector. Here p is the spherical linear interpolated vector between begin vector p_0 and end vector p_1 with fraction t.

4.6.2 Quaternion interpolation

Now that we know how we can geometrically interpolate between unit vectors in $\mathbb{R}^3$, we can give the definition of the interpolation between unit quaternions. Since a unit quaternion resamples an orientation we have that the quaternion interpolation between begin unit quaternion q_1 and end unit quaternion q_2 gives an interpolated quaternion in the range between those two orientations with a fraction $t \in [0,1]$. The following definition gives the formal definition of the interpolation of two unit quaternions.

Definition 4.6.5. *Given begin quaternion* q_0*, end quaternion* q_1 *and fraction* $t \in [0,1]$*,* $T(q_0, q_1; t)$ *is an interpolated quaternion if* $T(q_0, q_1; 0) = q_0$*,* $T(q_0, q_1; 1) = q_1$ *and* $T(q_0, q_1, t) = T(q_1, q_0, 1-t)$*.*

We speak of quaternion linear interpolation when the interpolated quaternion lays on the straight line between begin and end unit quaternion in quaternion space where the interpolation fraction is the fraction on this line segment. The following definition gives the formula for the linear interpolated quaternion.

Definition 4.6.6. (LERP) *Given* $q_1, q_2 \in H_0$ *and* $t \in [0,1]$*,*

$$LERP(q_1, q_2; t) = (1-t)q_1 + tq_2 \in H$$

In general the quaternion linear interpolation of two unit quaternion does not lead to a unit quaternion.

We speak of quaternion normalized linear interpolation when we take as interpolated quaternion the normalized quaternion linear interpolated quaternion. The quaternion obtained by the NLERP of two unit quaternions is a unit quaternion. The following definition gives the formula for the normalized linear interpolated quaternion.

Definition 4.6.7. (NLERP) *Given* $q_1, q_2 \in H_0$ *and* $t \in [0,1]$,

$$NLERP(q_1, q_2; t) = \frac{(1-t)q_1 + tq_2}{||(1-t)q_1 + tq_2||} \in H_0$$

We speak of quaternion spherical linear interpolation when the interpolated quaternion lays on the arc of the sphere of radius one between begin and end unit quaternion where the interpolation fraction is the fraction of this arc. When we define the quaternion which rotates the begin quaternion to the end quaternion with rotation vector $\vec{v}$ and rotation angle Ω, then the interpolated quaternion is the quaternion which rotates the begin quaternion around rotation vector $\vec{v}$ with rotation angle $t\Omega$. The following definition gives the formula for the spherical linear interpolated quaternion.

Definition 4.6.8. (SLERP) *Given* $q_1, q_2 \in H_0$ *and* $t \in [0,1]$,

$$SLERP(q_1, q_2; t) = q_1(q_1^* q_2)^t \in H_0$$

Here, $(q_1^* q_2)$ is the quaternion which rotates q_1 to q_2 and t is the fraction of the rotation angle of $(q_1^* q_2)$, meaning that $(q_1^* q_2)^t$ gives a rotation by a fraction t of the rotation angle. Hence, $q_1(q_1^* q_2)^t$ is q_1 rotated by $(q_1^* q_2)^t$.
We speak of log quaternion linear interpolation when we linearly interpolate in the log-space of the quaternions. The shorthand for log quaternion linear interpolation is LogLERP. The formula of a log quaternion linear interpolated quaternion is given in the next definition.

Definition 4.6.9. (LogLERP) *Given* $q_1, q_2 \in H_0$, $t \in [0,1]$, $\vec{\mathbf{w}}_1 = \log(q_1)$ *and* $\vec{\mathbf{w}}_2 = \log(q_2)$,

$$LogLERP(q_1, q_2; t) = e^{Lerp(\vec{\mathbf{w}}_1, \vec{\mathbf{w}}_2; t)}$$

The idea behind this method is that a rotation quaternion is approximately linear in a small neighborhood of the identity quaternion ($q = 1$).

4.6.3 Dual quaternion interpolation

We have now seen how we can interpolate quaternions linearly, nomalized linearly and spherical linearly. We can also define those forms of interpolations for the dual quaternion.

We speak of dual quaternion linear interpolation when the interpolated dual quaternion lays on the line segment in dual quaternion space between begin and end dual quaternion and the interpolation fraction is the fraction of this line segment. The formula for a linear interpolated dual quaternion is given in the next definition.

Definition 4.6.10. (LERP) *Given* $\hat{q}_1, \hat{q}_2 \in H_0^{\mathbb{D}}$ *and* $t \in [0,1]$,

$$LERP(\hat{q}_1, \hat{q}_2; t) = (1-t)\hat{q}_1 + t\hat{q}_2 \in H^{\mathbb{D}}$$

We speak of dual quaternion normalized linear interpolation when we take as interpolated dual quaternion the normalized dual quaternion linear interpolated dual quaternion. The next definition gives the formula for a dual quaternion normalized interpolated dual quaternion.

Definition 4.6.11. (NLERP) *Given* $\hat{q}_1, \hat{q}_2 \in H_0^{\mathbb{D}}$ *and* $t \in [0,1]$,

$$NLERP(\hat{q}_1, \hat{q}_2; t) = \frac{(1-t)\hat{q_1} + t\hat{q_2}}{||(1-t)\hat{q_1} + t\hat{q_2}||} \in H_0^{\mathbb{D}}$$

Screw linear interpolation for dual quaternion is a generalization of SLERP for quaternions. The shorthand for screw linear interpolation is ScLERP and the formula to obtain an screw linear interpolated dual quaternion is given in the next definition.

Definition 4.6.12. (ScLERP) *Given* $\hat{q}_1, \hat{q}_2 \in H_0^{\mathbb{D}}$ *and* $t \in [0,1]$,

$$ScLERP(\hat{q}_1, \hat{q}_2; t) = \hat{q_1}(\hat{q_1}^* \hat{q_2})^t \in H_0^{\mathbb{D}}$$

4.6.4 Blending

Aside from interpolation between two (dual) quaternions we can also take a convex combination of multiple (dual) quaternions. We call this blending.

Quaternion linear blending is the generalization of the normalized linear interpolation of quaternions. The shorthand for quaternion linear blending is QLB and the formula to obtain a quaternion linear blended quaternion is given in the following definition.

Definition 4.6.13. (QLB) *Given* $q_1, \ldots, q_k \in H_0$ *and* $w_1, \ldots, w_k \in [0,1]$ *with* $\sum_{i=1}^{k} w_i = 1$,

$$QLB(q_1, \ldots, q_k; w_1, \ldots, w_k) = \frac{w_1 q_1 + \ldots + w_k q_k}{||w_1 q_1 + \ldots + w_k q_k||} \in H_0$$

Dual quaternion linear blending is the generalization of the normalized linear interpolation of dual quaternions. The shorthand for dual quaternion linear blending is DLB and the formula to obtain a quaternion linear blended dual quaternion is given in the following definition.

Definition 4.6.14. (DLB) *Given* $\hat{q}_1, \ldots, \hat{q}_k \in H_0^{\mathbb{D}}$ *and* $w_1, \ldots, w_k \in [0,1]$ *with* $\sum_{i=1}^{k} w_i = 1$,

$$DLB(\hat{q}_1, \ldots, \hat{q}_k; w_1, \ldots, w_k) = \frac{w_1 \hat{q}_1 + \ldots + w_k \hat{q}_k}{||w_1 \hat{q}_1 + \ldots + w_k \hat{q}_k||} \in H_0^{\mathbb{D}}$$

Log quaternion linear blending is the generalization of the loq quaternion linear interpolation. The shorthand for log quaternion linear blending is LogLB. The formula to obtain a log quaternion blended quaternion is given in the following defenition.

Definition 4.6.15. (LogLB) *Given* $q_1, \ldots, q_k \in H_0$ *and* $w_1, \ldots, w_k \in [0,1]$ *with* $\sum_{i=1}^{k} w_i = 1$,

$$LogLB(q_1, \ldots, q_k; w_1, \ldots, w_k) = e^{QLB(q_1, \ldots, q_k; w_1, \ldots, w_k)} \in H_0$$

4.6.5 Comparison

When we compare interpolation methods for quaternions there are three properties which a good interpolation should satisfy. We will give them here.

- *constant speed*: the angle of interpolated rotation is linear with respect to parameter t,
- *shortest path*: the rotation from one orientation to another is around one axis with smallest angle,
- *coordinate invariance*: converting to another coordinate system before or after interpolation is the same.

Additional, it might be usefull when there exists a generalization of the quaternion interpolation to the blending of multiple quaternions. The next table gives the properties of the normalized linear interpolation, the spherical linear interpolation and the log linear interpolation of quaternions.

	constant speed	shortest path	coordinate invariant	generalizable to blending
quaternion NLERP	no	yes	yes	yes
quaternion SLERP	yes	yes	yes	no
quaternion LogLERP	yes	no	yes	yes

Table 1: Properties of the linear interpolations

None of the three interpolations satisfies all the properties that we would like an interpolation to satisfy. It depends on the case which properties for interpolation are more important.
Next to this table, we can notice that the difference between NLERP and SLERP is very small for rotation angles $\ll \pi$ which makes NLERP almost constant speed. Since NLERP is faster and easier to compute than SLERP, it might be better to use NLERP in case of small rotation angles. Also the LogLerp method is computationally faster than the SLERP method. Even though the LogLerp method is not shortest path, which makes it an obscure interpolation method, in many cases where the angle between interpolated quaternions is small its result comes close to the result of SLERP.

4.6.6 Final algorithm for interpolation of quaternions

To compute the quaternion which rotates a begin vector to its spherical linear interpolated vector we use the following algorithm.

Algorithm 3 Compute q_{rel} which rotates vector v_1 to vector $\text{SLERP}(v_1, v_2; t)$

Require: $\vec{v}_1, \vec{v}_2 \in \mathbb{R}^3$, $t \in (0, 1)$
Ensure: $q\vec{v}_1 q^* = \text{SLERP}(\vec{v}_1, \vec{v}_2; t)$
$\vec{v}_1 = \frac{\vec{v}_1}{||\vec{v}_1||}$, $\vec{v}_2 = \frac{\vec{v}_2}{||\vec{v}_2||}$
$\theta = \arccos(v_1 \cdot v_2)$, $\vec{v} = \vec{v}_1 \times \vec{v}_2$
$\text{Log}q_{\text{rel}} = \frac{1}{2} t\theta\vec{v}$
return $q_{\text{rel}} = e^{\text{Log}q_{\text{rel}}}$

To compute the quaternion which rotates a begin orientation to its spherical linear interpolated orientation we use the following algorithm.

Algorithm 4 Compute q_{rel} which rotates orientation q_1 to orientation $\text{SLERP}(q_1, q_2; t)$

Require: $q_1, q_2 \in H_0$ and $t \in (0, 1)$
Ensure: $q_1 q_{\text{rel}} = \text{SLERP}(q_1, q_2; t)$
$q_3 := (r, \vec{w}) = q_1^* q_2$
$\theta = \arccos(r)$ and $\vec{v} = \frac{\vec{w}}{||\vec{w}||}$
$\text{Logq}_3 = \frac{1}{2}\theta\vec{v}$
return $q_{\text{rel}} = e^{t\text{Logq}_3}$

A dual quaternion represents a translation and rotation. This means that we can extract a translation vector and a rotation quaternion from it. When we want to spherical linear interpolate the translation vectors of two dual quaternion and at the same time want to spherical linear interpolate the rotation quaternions of these dual quaternions, then we need to find a way to combine these two interpolations. We remark that this is not the same as a spherical linear interpolation of two dual quaternions. We achieve this combined interpolation by performing a log quaternion linear interpolation to the two quaternions which we used to obtain the spherical linear interpolated vector and quaternion. Here we choose a log quaternion linear interpolation since the methods for spherical interpolation as we described in algorithm 3 and 4 already give us the logarithms of the quaternions which we wish to interpolate.

Algorithm 5 Compute q_{rel}=LogLERP$(q_{rel}^A, q_{rel}^B, \alpha)$, where q_{rel}^A is the rotation from q_1 to $\text{SLERP}(q_1, q_2; t)$ and q_{rel}^B is the rotation from v_1 to $\text{SLERP}(v_1, v_2; t)$, where $\hat{q}_1 = (1 + \frac{\epsilon}{2}\vec{v}_1)q_1$ and $\hat{q}_2 = (1 + \frac{\epsilon}{2}\vec{v}_2)q_2$

Require: $\hat{q}_1, \hat{q}_2 \in H_0^{\mathbb{D}}$ and $t \in (0, 1)$
$q_1^{\mathbb{R}} = \frac{\hat{q}_1 + \bar{\hat{q}}_1}{2}$ and $q_1^{\mathbb{D}} = \frac{\hat{q}_1 - \bar{\hat{q}}_1}{2\epsilon}$
$q_2^{\mathbb{R}} = \frac{\hat{q}_2 + \bar{\hat{q}}_2}{2}$ and $q_2^{\mathbb{D}} = \frac{\hat{q}_2 - \bar{\hat{q}}_2}{2\epsilon}$
$q_3 := (r, \vec{w}) = (q_1^{\mathbb{R}})^* q_2^{\mathbb{R}}$
$\theta_3 = \arccos(r)$ and $\vec{v}_3 = \frac{\vec{w}}{||\vec{w}||}$
$Logq_3 = \frac{1}{2}\theta_3\vec{v}_3$
$\vec{v}_1 = 2q_1^{\mathbb{D}}(q_1^{\mathbb{R}})^*$ and $\vec{v}_2 = 2q_2^{\mathbb{D}}(q_2^{\mathbb{R}})^*$
$\vec{v}_1 = \frac{\vec{v}_1}{||\vec{v}_1||}$ and $\vec{v}_2 = \frac{\vec{v}_2}{||\vec{v}_2||}$
$\theta_4 = \arccos(v_1 \cdot v_2)$ and $\vec{v}_4 = \vec{v}_1 \times \vec{v}_2$
$Logq_4 = \frac{1}{2}\theta_4\vec{v}_4$
return $q_{rel} = e^{t(0, \alpha Logq_3 + (1-\alpha)Logq_4)}$

We use this algorithm in the next chapter.

5 Kinematics

This chapter gives the definition of the forward and inverse kinematics problem. It starts by showing that the inverse kinematic problem is hard to solve analytically. Furthermore, it gives the explanation and algorithm for the Cyclic Coordinate Descent method using dual quaternions. Also, it gives explanations and algorithms for the different Jacobian methods including the derivation of the end effector Jacobian matrix with respect to the log values of the relative quaternions of the joints.

5.1 Forward kinematics

With forward kinematics we determine the end effector position and orientation given a configuration of the chain. We first give some notations for the structure of the chain which we will use throughout this chapter.

Notation 5.1.1. *Given an ordered chain of m joints $\{1, 2, \ldots, m\}$ and joint 0 to which joint 1 is linked.*

- $\hat{q}_i^W = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_i^W)q_i^W \in H_0^{\mathbb{D}}$ *gives the position* $\vec{t}_i^W \in \mathbb{R}^3$ *and orientation* $q_i^W \in H_0$ *of the coordinate system of joint i with respect to the world coordinate system,*
- *for* $i \neq 0$, $\hat{q}_i^R = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_i^R)q_i^R \in H_0^{\mathbb{D}}$ *gives the position* $\vec{t}_i^R \in \mathbb{R}^3$ *and orientation* $q_i^R \in H_0$ *of the local coordinate system of joint i with respect to the local coordinate system of joint $i-1$,*
- $\hat{q}_{end}^W = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_{end}^W)q_{end}^W \in H_0^{\mathbb{D}}$ *gives the position* $\vec{t}_{end}^W$ *and orientation* q_{end}^W *of the end effector with respect to the world coordinate system,*
- $\hat{q}_{end}^R = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_{end}^R)q_{end}^R \in H_0^{\mathbb{D}}$ *gives the position* $\vec{t}_{end}^R$ *and orientation* q_{end}^R *of the end effector with respect to the coordinate system of joint m.*

For the motion of a chain we assume that the distances between the joints are fixed and the relative orientations of the joints are variable. The distance and relative orientation from the end effector with respect to the end joint are fixed also. Therefore we say that the end effector position and orientation in world coordinates depends on the relative orientations of the joints. The relative orientation of joint i is given by q_i^R. In section 4.3 we showed that a unit quaternion depends on three parameters, which are its log values. This means that the relative orientation of q_i^R depends on $\log(q_i^R)$ which we call the configuration of joint i. The configuration of the chain is given by the set of configurations of its joints. We use the following notation for the configuration of a chain.

Definition 5.1.2. *The configuration of a chain with m joints $\{1, 2, \ldots, m\}$ is given by*

$$\Phi = (\vec{\Phi}_1, \vec{\Phi}_2, \ldots, \vec{\Phi}_m)$$

where $\vec{\Phi}_i$ is the configuration of joint i given by $\log(q_i^R)$.

The end effector holds a position and orientation with respect to the end joint or with respect to the world coordinate system. We define $\vec{e} = (\vec{t}_{\text{end}}^W, \vec{o}_{\text{end}}^W)$, where $\vec{t}_{\text{end}}$ is the end effector position in world coordinates and $\vec{o}_{\text{end}}^W$ is the logarithm of the end effector orientation with respect to the world coordinate system, $\vec{o}_{\text{end}}^W = \log(q_{\text{end}}^W)$.

This leads us to the formalization of the forward kinematics problem of determining the end effector position and orientation $\vec{e}$ given a configuration of the chain Φ.

Definition 5.1.3. *The forward kinematics problem is described by*

$$\vec{e} = f(\Phi)$$

where f is the function which determines the end effector position and orientation $\vec{e}$ given configuration Φ.

f is a nonlinear function which has a unique solution to the forward kinematics problem.

Since we defined the parameter $\vec{\Phi}_i$ as the log of the rotation quaternion q_i^R, we have $q_i^R(\vec{\Phi}_i) = e^{\vec{\Phi}_i}$. Together with the fixed translation vector $\vec{t}_i^R$, this gives $\hat{q}_i^R(\vec{\Phi}_i) = (1 + \frac{\epsilon}{2}\vec{\mathbf{t}}_i^R)q_i^R(\vec{\Phi}_i)$.

The unit dual quaternion representing the orientation and translation of joint i relative to the world coordinate system can be found by

$$\hat{q}_i^W(\vec{\Phi}_1, \vec{\Phi}_2, \ldots, \vec{\Phi}_{i-1}, \vec{\Phi}_i) = \hat{q}_0^W \hat{q}_1^R(\vec{\Phi}_1)\hat{q}_2^R(\vec{\Phi}_2)\cdots\hat{q}_{i-1}^R(\vec{\Phi}_{i-1})\hat{q}_i^R(\vec{\Phi}_i)$$

This means that we can compute the dual quaternion $\hat{q}_{\text{end}}^W$ of the end effector position and orientation relative to the world coordinate system by

$$\hat{q}_{\text{end}}^W(\Phi) = \hat{q}_m^W(\Phi)\hat{q}_{\text{end}}^R = \hat{q}_0^W \hat{q}_1^R(\vec{\Phi}_1)\hat{q}_2^R(\vec{\Phi}_2)\cdots\hat{q}_{m-1}^R(\vec{\Phi}_{m-1})\hat{q}_m^R(\vec{\Phi}_m)\hat{q}_{\text{end}}^R$$

From $\hat{q}_{\text{end}}^W = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_{\text{end}}^W)q_{\text{end}}^W$ we can extract the end effector position $\vec{t}_{\text{end}}^W$ and orientation q_{end}^W with respect to the world coordinate system as in proposition 4.2.14(b). This gives us our desired solution $\vec{e} = (\vec{t}_{\text{end}}^W, \log(q_{\text{end}}^W))$ of the forward kinematics problem. The following algorithm gives the computation of the end effector position and orientation $\vec{e}$ in world coordinates given the configuration Φ of a chain.

Algorithm 6 Forward kinematics, compute $\vec{e} = f(\Phi)$

Require: configuration $\Phi = (\vec{\Phi}_1, \vec{\Phi}_2, \ldots, \vec{\Phi}_m)$
Ensure: $\vec{e} = f(\Phi)$
  for $k = 1, k = m$ **do**
    $\hat{q}_k^R = (1 + \frac{1}{2}\epsilon\vec{t}_k^R)e^{\vec{\Phi}_k}$
    $\hat{q}_k^W = \hat{q}_{k-1}^W\hat{q}_k^R$
  end for
  $\hat{q}_{\text{end}}^W = \hat{q}_k^W\hat{q}_{\text{end}}^R$
  $q_{\text{end}}^{\mathbb{R}} := (r, \vec{w}) = \frac{\hat{q}_{\text{end}} + \bar{\hat{q}}_{\text{end}}}{2}$, $q_{\text{end}}^{\mathbb{D}} = \frac{\hat{q}_{\text{end}} - \bar{\hat{q}}_{\text{end}}}{2\epsilon}$
  $\vec{o}_{\text{end}} = \arccos(r)\frac{\vec{w}}{||\vec{w}||}$
  $\vec{t}_{\text{end}} = 2q_{\text{end}}^{\mathbb{D}}(q_{\text{end}}^{\mathbb{R}})^*$

  return $\vec{e} = (\vec{t}_{\text{end}}, \vec{o}_{\text{end}})$

5.2 Inverse Kinematics

With inverse kinematics we wish to find the configuration Φ of a chain given the end effector position and orientation $\vec{e}$, as formalized in the following definition.

Definition 5.2.1. *Given the function f which determines the end effector position and orientation $\vec{e}$ given the configuration Φ, the inverse kinematics problem is described by*

$$\Phi \in f^{-1}(\vec{e})$$

where f^{-1} is the inverse image of $\vec{e}$ under f.

This problem is a lot harder. There are multiple solutions Φ possible and we want one that gives us the most natural pose. Aside from that we want the solution Φ to be stable, meaning that when we slightly change the end effector position and orientation, the configuration of the chain Φ also changes only slight.

When the chain consists of a small number of joints, we can compute the solution Φ analytically, by trying out all possible ways the links can be placed in order to end on the end effector position. We describe this in the next section and will see that the analytical inverse kinematics method is not an easy method to use for a chain with a larger number of joints. This is because there are multiple solutions possible and it is hard to find a convenient way to program a computer to choose solutions in a stable way. Therefore we will have to use an inverse kinematics method which stepwise comes to a solution. We will briefly discuss the Cyclic Coordinate Descent Method in section 5.4 and the Jacobian methods in section 5.5.

5.3 Analytical Inverse Kinematics

In this section we will show how we can analytically find a solution for our inverse kinematics problem for a chain with two joints in a 2-dimensional space, where we only wish to solve for the end effector position. Let the two links of the chain have lengths l_1 and l_2. Let the desired end effector position be the coordinate (x, y). For simplicity we assume that this position is reachable by our chain. We can now make a picture to find the possible configurations of the chain with end effector on position (x, y) as we see in figure 21.

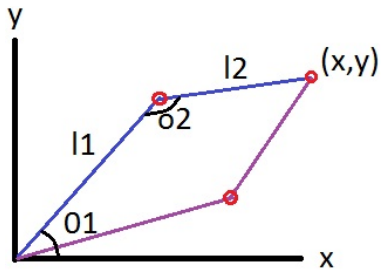


Figure 21: Analytical solution

We can see that there are two possible configurations for our chain with end effector on the position (x, y). The two different solutions for our chain are denoted by the blue and purple line segments. We will calculate the relative rotation angles for the blue chain. The relative rotation angles are denoted by θ_1 and θ_2 in figure 21. Since we know the lengths of the two links l_1 and l_2 and we know the desired end effector position, we can calculate the relative angles.

$$\theta_1 = \cos^{-1}\left(\frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}}\right) \quad \text{and} \quad \theta_2 = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1 l_2}\right)$$

For a chain with two links in a 2-dimensional space this analytical calculation was not that hard. When we extend the chain with an additional joint, the number of possible solutions already increases as we can see in figure 22.

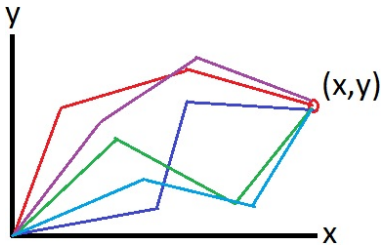


Figure 22: Multiple solutions to the inverse kinematics problem

Aside from that the calculation of the relative joint angles for one of these possible solutions becomes harder when the chain becomes larger, it can also be seen that it is hard to find a convenient way to define which of the solutions we will choose. After all we want our configuration Φ to be stable. Clearly when we go from 2-dimensional space to 3-dimensional space, the situation will be worse, calculations harder and the number of solutions higher.

5.4 Cyclic Coordinate Descent Method

The Cyclic Coordinate Descent method, in short CCD method, is a method which iteratively solves the inverse kinematic problem. Here one iteration consists of bringing the current end effector position and orientation closer to a desired end effector position and orientation, the target, by bringing a small change to the configuration of one of the joints in the chain. We repeatedly perform this iteration for the complete chain, where we start at the end joint and end with the base joint.

The target is given by $\hat{q}^W_{\text{target}} = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}^W_{\text{target}})q^W_{\text{target}} \in H_0^{\mathbb{D}}$ where $\vec{t}^W_{\text{target}}$ is the position and q^W_{target} is the orientation of the target with respect to the world coordinate system.

When we make a small change to the configuration of joint i, then this means that we rotate the orientation of joint i slightly. To get the right rotation of joint i we need to compute the rotation quaternion which rotates the end effector position and orientation relative to the coordinate system of joint i to the target position and orientation relative to the coordinate system of joint i. Figure 23 gives an illustration of this.

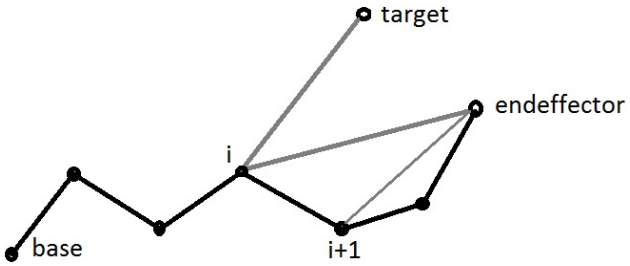


Figure 23: CCD target and end effector rel to local coordinate system

To compute the dual quaternions which represent the translation and orientation of the end effector and target with respect to the coordinate system of joint i we use the following addition notation.

Notation 5.4.1. *Given $j \leq i$, $\hat{q}^j_i$ denotes the position and orientation of the local coordinate system of joint i with respect to the local coordinate system of joint j, where $\hat{q}^i_i = 1$.*

It is not hard to see that $\hat{q}^j_i = \hat{q}^R_{j+1} \dots \hat{q}^R_{i+1}\hat{q}^R_i$ for $j < i$. This gives us the equalities $\hat{q}^W_i = \hat{q}^W_j \hat{q}^j_i$ and $\hat{q}^k_i = q^k_j q^j_i$ for $k < j < i$.

The dual quaternion for the target position and orientation with respect to the coordinate system of joint i can be obtained by $\hat{q}^i_{\text{target}} = (\hat{q}^W_i)^{-1}\hat{q}^W_{\text{target}}$, since $\hat{q}^W_{\text{target}} = \hat{q}^W_i \hat{q}^i_{\text{target}}$. We use the fact that the target position and orientation with respect to the world coordinate system are known. To obtain the end effector in the coordinate system of joint i we use the recursive relation $\hat{q}^i_{\text{end}} = \hat{q}^R_{i+1}\hat{q}^{i+1}_{\text{end}}$ if $i < m$ and $\hat{q}^i_{\text{end}} = \hat{q}^R_{\text{end}}$ if $i = m$. Note that we do not use $\hat{q}^i_{\text{end}} = (\hat{q}^W_i)^{-1}\hat{q}^W_{\text{end}}$ since we do not know the end effector position and orientation with respect to the world coordinate system at every iteration. Updating the end effector dual quaternion with respect to the world coordinate system at every iteration would be computationally inefficient and unneccessary.

Now that we have the dual quaternions $\hat{q}^i_{target}$ and $\hat{q}^i_{end}$ representing the position and orientation of the target and end effector position, we can use algorithm 5 to derive the quaternion which

by rotating the orientation of joint i brings the end effector position and orientation closer to the target position and orientation in the coordinate system of joint i. Here we can choose the fraction with which we bring the end effector closer to the target per iteration. We use algorithm 3 or algorithm 4 when we ignore solving for the orientation or position respectively.

We iteratively go through the chain, solving the end effector closer to the target by rotating a joint, starting at the end joint and ending at the base joint. When we repeatedly iterate through the chain in this way, then the end effector will eventually converge to the closest reachable position and orientation with respect to the target. When we in advance know that the target is reachable by the end effector of our chain, we can stop iterating once the distance between the end effector position and orientation is smaller than a small threshold value ϵ, if $|\vec{g} - \vec{e}| < \epsilon$. In case we do not know whether our target is reachable by the end effector position of the chain, this will not work. The distance between target and end effector might never reach the threshold value, which will lead to infinity iteration, we never stop. In this case it might be better to set a number N for the times that we iterate through our chain.

We should not forget that the rotational freedom of the joints can be constrained as we described in section 4.5. Therefore whenever the quaternions for the relative orientations of the joints are updated we have to clamp these solutions as in algorithm 1 and 2.

Algorithm 7 gives us the final algorithm for the Cyclic Coordinate Descent method. The illustration of this method is shown in figure 24.

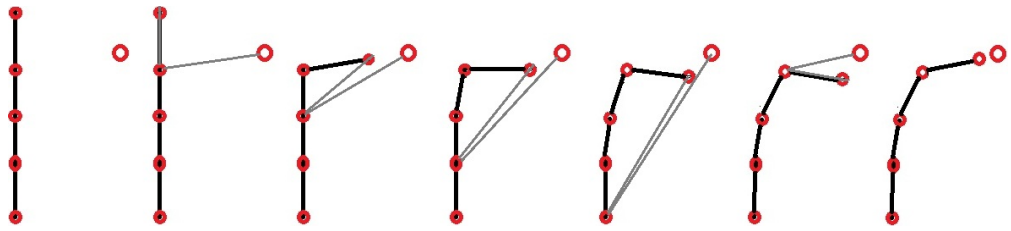


Figure 24: The iterations of the CCD algorithm

On the left we see the initial configuration of the chain and the target. We start with rotating the end joint as close as possible to the target, followed by the other joints of the chain. Once we finished rotating the base joint we start over again with rotating end effector. We see that the end effector converges to the target at every iteration.

Algorithm 7 The Cyclic Coordinate Descent method

Require: begin configuration Φ^0, target $\vec{g} = (\vec{t}^W_{\text{target}}, \vec{o}^W_{\text{target}})$, step factor α and thresholds ϵ and N
(Assume that fixed variables $\vec{t}^R_i \ \forall i$ and $\hat{q}^R_{\text{end}}$ and $\hat{q}^W_0$ are known)

Ensure: $\Phi \approx f^{-1}(\vec{g})$

$\hat{q}^W_{\text{target}} = (1 + \frac{1}{2}\epsilon\vec{t}^W_{\text{target}})e^{\vec{o}^W_{\text{target}}}$
for $i = 1, i = m$ **do**
$\hat{q}^R_i = (1 + \frac{1}{2}\epsilon\vec{t}_i)e^{\vec{\Phi}^0_i}$
$\hat{q}^W_i = \hat{q}^W_{i-1}\hat{q}^R_i$
end for
$\hat{q}^W_{\text{end}} := (q^{\mathbb{R}}_{\text{end}}, q^{\mathbb{D}}_{\text{end}}) = \hat{q}^W_m\hat{q}^R_{\text{end}}$
$q^{\mathbb{R}}_{\text{end}} := (r, \vec{w})$, $\vec{o}^W_{\text{end}} = \arccos(r)\frac{\vec{w}}{||\vec{w}||}$
$\vec{t}^W_{\text{end}} = 2q^{\mathbb{D}}_{\text{end}}(q^{\mathbb{R}}_{\text{end}})^*$
$\vec{e}_0 = (\vec{t}^W_{\text{end}}, \vec{o}^W_{\text{end}})$, $k = 0$
while $(|\vec{g} - \vec{e}_k| > \epsilon$ **or** $k < N)$ **do**
for $i = m, i = 1$ **do**
$\hat{q}^i_{\text{target}} = (\hat{q}^W_i)^{-1}\hat{q}^W_{\text{target}}$
if i=m **then**
$\hat{q}^i_{\text{end}} = \hat{q}^R_{\text{end}}$
else
$\hat{q}^i_{\text{end}} = \hat{q}^R_{i+1}\hat{q}^{i+1}_{\text{end}}$
end if
$q_{\text{rel}} = \text{Algorithm5}(\hat{q}^i_{\text{end}}, \hat{q}^i_{\text{target}}, \alpha)$
$q^R_i = q^R_i q_{\text{rel}}$, $q^R_i = \text{CLAMPED}(q^R_i)$ (see algorithm 1 and 2)
$\vec{\Phi}^{k+1}_i = \log(q^R_i)$, $\hat{q}^R_i = (1 + \frac{1}{2}\epsilon\vec{t}_i)q^R_i$
end for
for $i = 1, i = m$ **do**
$\hat{q}^W_i = \hat{q}^W_{i-1}\hat{q}^R_i$
end for
$\hat{q}^W_{\text{end}} := (q^{\mathbb{R}}_{\text{end}}, q^{\mathbb{D}}_{\text{end}}) = \hat{q}^W_m\hat{q}^R_{\text{end}}$
$q^{\mathbb{R}}_{\text{end}} := (r, \vec{w})$, $\vec{o}^W_{\text{end}} = \arccos(r)\frac{\vec{w}}{||\vec{w}||}$
$\vec{t}^W_{\text{end}} = 2q^{\mathbb{D}}_{\text{end}}(q^{\mathbb{R}}_{\text{end}})^*$
$\vec{e}_{k+1} = (\vec{t}^W_{\text{end}}, \vec{o}^W_{\text{end}})$
$k = k + 1$
end while

return $\Phi^k = (\vec{\Phi}^k_1, \vec{\Phi}^k_2, \ldots, \vec{\Phi}^m_i)$

5.5 Jacobian Methods

The Jacobian method is a method which iteratively solves the inverse kinematics problem. The CCD method solves iteratively by repeatedly changing every joint configuration of the chain such that it brings the end effector position and orientation closer to a target position and orientation. The Jacobian method iteratively solves by repeatedly changing the configuration of the complete chain such that it brings the end effector position and orientation closer to a target position and orientation.

The Jacobian method does this by determining how a change in configuration changes the end effector position and orientation. This correspondence is given by the Jacobian matrix.

The Jacobian matrix of $\vec{e}(\Phi) : \mathbb{R}^{3m} \to \mathbb{R}^6$ is given by:

$$J(\vec{e}, \Phi) = \begin{pmatrix} \frac{\partial e_1}{\partial \vec{\Phi}_1} & \cdots & \frac{\partial e_1}{\partial \vec{\Phi}_m} \\ \vdots & & \vdots \\ \frac{\partial e_6}{\partial \vec{\Phi}_1} & \cdots & \frac{\partial e_6}{\partial \vec{\Phi}_m} \end{pmatrix}$$

Here we have $\vec{e} = (e_1, e_2, \ldots, e_6)$ and $\frac{\partial e_k}{\partial \vec{\Phi}_i} = (\frac{\partial e_k}{\partial \vec{\Phi}_{i,1}}, \frac{\partial e_k}{\partial \vec{\Phi}_{i,2}}, \frac{\partial e_k}{\partial \vec{\Phi}_{i,3}})$ with $\vec{\Phi}_i = (\vec{\Phi}_{i,1}, \vec{\Phi}_{i,2}, \vec{\Phi}_{i,3})$.

The computation of the Jacobian matrix is given in section 5.5.5.

In order to know how we should change the configuration of a chain to change the end effector position and orientation in a desired way, we use the following linear approximation.

$$\Delta\vec{e} \approx \frac{\partial \vec{e}}{\partial \Phi}\Delta\Phi = J(\vec{e}, \Phi)\Delta\Phi = J\Delta\Phi$$

This means that a small change of $\Delta\Phi$ to the current configuration Φ will bring a change of $\Delta\vec{e}$ to the current end effector position $\vec{e}$. However, what we need is the other way around. We need to know how we should bring a small change $\Delta\Phi$ to the current configuration Φ in order to change the current end effector position and orientation $\vec{e}$ by an desired amount $\Delta\vec{e}$. When the Jacobian matrix has an inverse, this can be solved by the following equation.

$$\Delta\Phi \approx J^{-1}\Delta\vec{e}$$

This means that a small change of $\Delta\vec{e}$ to the current end effector position $\vec{e}$ will bring a change of $\Delta\Phi$ to the current configuration Φ. We call the Jacobian method which uses the inverse of the Jacobian the Inverse Jacobian method. We give this method in section 5.5.1.

In many cases the Jacobian matrix is not invertible. Therefore there are other methods which use the Jacobian matrix to determine the change $\Delta\Phi$ in current configuration Φ given a change $\Delta\vec{e}$ in the current end effector position and orientation $\vec{e}$. These methods are the Jacobian method which we give in section 5.5.2 and the Pseudo Jacobian method which we give in section 5.5.3.

5.5.1 Inverse Jacobian Method

The Inverse Jacobian method is a method which only works if the Jacobian matrix of the end effector position and orientation with respect to the configuration of the chain has an inverse. Therefore we assume in this section that the Jacobian matrix has an inverse.

In the previous section we saw that for an invertible Jacobian matrix J, the relation in change $\Delta\Phi$ of the current configuration Φ with respect to the change $\Delta\vec{e}$ in current end effector position and orientation $\vec{e}$ is given by the following equation.

$$\Delta\Phi \approx J^{-1}\Delta\vec{e}$$

We know that the current end effector position $\vec{e}$ depends on the current configuration of the chain Φ and is given by $f(\Phi)$. When $\vec{g}$ is the target position and orientation, then this means that we wish to change our current end effector position $\vec{e}$ by $\Delta\vec{e} = \vec{g} - \vec{e} = \vec{g} - f(\Phi)$. The approximation we use, $\Delta\Phi \approx J^{-1}\Delta\vec{e}$, is only true for small changes $\Delta\vec{e}$. Therefore it is better to only bring the end effector $\vec{e}$ closer to the target g with a fraction $\alpha \in [0, 1]$. This means that we should change the configuration of the chain Φ by $\Delta\Phi = \alpha J^{-1}(\vec{g} - \vec{e})$ such that the current end effector position and orientation $\vec{e}$ changes by $\Delta\vec{e} = \alpha(\vec{g} - \vec{e})$.

We end up with a new configuration $\Phi^{\text{new}} = \Phi + \Delta\Phi$ which leads to a new end effector position and orientation $\vec{e}^{\text{new}} = f(\Phi^{\text{new}})$.

We repeat updating the configuration of the chain in this way till the distance between the end effector and the target is smaller than a given threshold ϵ, or after a fixed number of updates N.

This leads us to the following algorithm for the Inverse Jacobian method.

Algorithm 8 The Inverse Jacobian method

Require: start configuration Φ^0, target $\vec{g}$, optimization factor α and threshold ϵ.
  $\vec{e}_0 = f(\Phi_0)$. Let $k = 0$.
  while $(|\vec{g} - \vec{e}_k| > \epsilon$ **or** $k < N)$ **do**
    $J = J(\vec{e}_k, \Phi^k)$
    $\Delta\vec{e}_k = \alpha(\vec{g} - \vec{e}_k)$, $\Delta\Phi^k = J^{-1}\Delta\vec{e}_k$
    $\Phi^{k+1} = \Phi^k + \Delta\Phi^k$
    for i=1,i=m **do**
      $\vec{\Phi}_i^{k+1} = \log(\text{CLAMPED}(e^{\vec{\Phi}_i^{k+1}}))$
    end for
    $\vec{e}_{k+1} = f(\Phi^{k+1})$
    $k = k + 1$
  end while

  return Φ^k.

Note that we added the clamping of the rotation freedom of the joints. The way we wrote the algorithm is only for the readability, merging the clamping inside the forwards kinematics function f is more efficient.

5.5.2 Jacobian Transpose method

The Jacobian Transpose method is a Jacobian method intended for the case where the Jacobian matrix has no inverse. The Jacobian Transpose method is similar to the Inverse Jacobian method, aside from that it uses the transpose of the Jacobian matrix instead of the inverse of the Jacobian matrix to calculate the change in configuration $\Delta\Phi$ given a change in end effector position and orientation $\Delta\vec{e}$. This means that we use the following expression to calculate $\Delta\Phi$.

$$\Delta\Phi = \alpha J^T \Delta\vec{e}$$

The use of the Jacobian transpose in this formula can be justified in terms of virtual forces. It can also be proved that the new distance between end effector and target $||J\Delta\Phi|| = ||\alpha J J^T \Delta\vec{e}||$ is smaller than the original distance between end effector and target $||\Delta\vec{e}||$ for small enough α, see [Buss 2009, 7].

Since by substitution of the Jacobian transpose matrix at the position of the Jacobian inverse matrix the Inverse Jacobian method remains the same, we end with the following algorithm for the Jacobian Transpose method.

Algorithm 9 The Jacobian Transpose method

Require: start configuration Φ^0, target $\vec{g}$, optimization factor α and threshold ϵ.
  $\vec{e}_0 = f(\Phi_0)$. Let $k = 0$.
  while $(|\vec{g} - \vec{e}_k| > \epsilon$ **or** $k < N)$ **do**
    $J = J(\vec{e}_k, \Phi^k)$
    $\Delta\vec{e}_k = \alpha(\vec{g} - \vec{e}_k)$, $\Delta\Phi^k = J^T \Delta\vec{e}_k$
    $\Phi^{k+1} = \Phi^k + \Delta\Phi^k$
    for i=1,i=m **do**
      $\vec{\Phi}_i^{k+1} = \log(\text{CLAMPED}(e^{\vec{\Phi}_i^{k+1}}))$
    end for
    $\vec{e}_{k+1} = f(\Phi^{k+1})$
    $k = k + 1$
  end while

  return Φ^k.

5.5.3 Pseudo Inverse Jacobian method

The Pseudo Inverse Jacobian method is a Jacobian method intended for the case where the Jacobian matrix has no inverse. The Pseudo Inverse Jacobian method is similar to the Inverse Jacobian method, aside from that it uses the pseudo inverse of the Jacobian matrix instead of the inverse of the Jacobian matrix to calculate the change in configuration $\Delta\Phi$ given a change in end effector position and orientation $\Delta\vec{e}$. This means that we use the following expression to calculate $\Delta\Phi$.

$$\Delta\Phi = J^{\dagger}\Delta\vec{e}$$

Here $J^{\dagger}$ is the pseudo inverse of the Jacobian matrix J, also called the Moore-Penrose inverse of J. This leads to the best solutions in the sense of least squares. This means that $\Delta\Phi = J^{\dagger}\Delta\vec{e}$ is the solution to $\Delta\vec{e} = J\Delta\Phi$ of smallest norm $||\Delta\Phi||$ which minimizes the error $||J\Delta\Phi - \Delta\vec{e}||^2$.

The algorithm for the Pseudo Inverse Jacobian method is similar to the algorithm of the Inverse Jacobian method and is given below.

Algorithm 10 The Pseudo Inverse Jacobian method

Require: start configuration Φ^0, target $\vec{g}$, optimization factor α and threshold ϵ.
  $\vec{e}_0 = f(\Phi_0)$. Let $k = 0$.
  while $(|\vec{g} - \vec{e}_k| > \epsilon$ **or** $k < N)$ **do**
    $J = J(\vec{e}_k, \Phi^k)$
    $\Delta\vec{e}_k = \alpha(\vec{g} - \vec{e}_k)$, $\Delta\Phi^k = J^{\dagger}\Delta\vec{e}_k$
    $\Phi^{k+1} = \Phi^k + \Delta\Phi^k$
    for i=1,i=m **do**
      $\vec{\Phi}_i^{k+1} = \log(\text{CLAMPED}(e^{\vec{\Phi}_i^{k+1}}))$
    end for
    $\vec{e}_{k+1} = f(\Phi^{k+1})$
    $k = k + 1$
  end while

  return Φ^k.

The Pseudo Inverse Jacobian method has stability problems in the neighborhood of singularities. At a singularity the Jacobian matrix has no full row rank, which corresponds to the fact that there is a direction of movement of the end effector which is not achievable. We can verify whether we are at a singularity by determining if the Jacobian matrix has a zero row.
When the configuration of a chain is exactly at a singularity, then the Pseudo Inverse method does not attempt to move in an impossible direction. When the configuration of a chain is close to a singularity, then the Pseudo Inverse method will lead to very large changes in joint angles while the movement to the target might be very small. A singularity can be detected by verifying the values in the Jacobian matrix to be near zero. In the next section we will give a method which avoids these singularity problems.

The Pseudo Inverse Jacobian method has the nice property that the matrix $(I - J^{\dagger}J)$ performs a projection on the nullspace of J. This means that $J(I - J^{\dagger}J)z = 0$ for all z in the $\Delta\Phi$-space. When

we now use the following expression to compute the change in configuration given the change in end effector position and orientation,

$$\Delta\Phi = J^{\dagger}\Delta\vec{e} + (I - J^{\dagger}J)z$$

then this is still a minimizer of the error $||J\Delta\Phi - \Delta\vec{e}||^2$.
This means that $(I - J^{\dagger}J)z$ gives us all the directions in which we can move our configuration without influencing the magnitude of the error.
This property can be used to solve secondary goals and mostly also reduces the singularity problems.

5.5.4 Damped Least Squares Jacobian method

The Damped Least Squares Jacobian method is a method which avoids the singularities that arise at the Pseudo Inverse Jacobian method. The method calculates the change in configuration $\Delta\Phi$ given a change in end effector position and orientation $\Delta\vec{e}$ by the following expression.

$$\Delta\Phi = J^T(JJ^T + \gamma^2 I)^{-1}\Delta\vec{e}$$

Here I is the 6x6 identity matrix and $\gamma \in \mathbb{R}$ is a nonzero damping constant.

The solution of the Pseudo inverse method $\Delta\Phi = J^{\dagger}\Delta\vec{e}$ was the minimizer of $||J\Delta\Phi - \Delta\vec{e}||^2$. The solution of the Damped Least Squares method $\Delta\Phi = J^T(JJ^T + \gamma^2 I)^{-1}\Delta\vec{e}$ gives the minimizer of $||J\Delta\Phi - \Delta\vec{e}||^2 + \gamma^2||\Delta\Phi||^2$.

The damping constant is chosen ad hoc. It has to be large enough to take away the singularity problems and become stable, but if it is too large the convergence rate becomes too slow.

This gives the following algorithm for the Damped Least Squares Jacobian method.

Algorithm 11 The Damped Least Squares method

Require: start configuration Φ^0, target $\vec{g}$, optimization factor α and threshold ϵ.
  $\vec{e_0} = f(\Phi_0)$. Let $k = 0$.
  while $(|\vec{g} - \vec{e}_k| > \epsilon$ **or** $k < N)$ **do**
    $J = J(\vec{e}_k, \Phi^k)$
    $\Delta\vec{e}_k = \alpha(\vec{g} - \vec{e}_k)$, $\Delta\Phi^k = J^T(JJ^T + \gamma^2 I)^{-1}\Delta\vec{e}_k$
    $\Phi^{k+1} = \Phi^k + \Delta\Phi^k$
    for i=1,i=m **do**
      $\vec{\Phi}_i^{k+1} = \log(\text{CLAMPED}(e^{\vec{\Phi}_i^{k+1}}))$
    end for
    $\vec{e}_{k+1} = f(\Phi^{k+1})$
    $k = k + 1$
  end while

  return Φ^k.

5.5.5 Computing the Jacobian matrix

To compute the Jacobian matrix for the end effector position and orientation $\vec{e}$ with respect to the configuration Φ we need to determine the column vectors $\frac{\partial \vec{e}}{\partial \Phi_{i,k}}$ for all $i \in \{1, \ldots, m\}$ and $k \in \{1, 2, 3\}$. Given a configuration Φ, fixed translations $\vec{t}_i^R$ and $\hat{q}_0^W$, we can compute the dual quaternions $\hat{q}_i^R$ and $\hat{q}_i^W$ for all $i \in \{1, \ldots, m\}$.

We will now derive $\frac{\partial \vec{e}}{\partial \Phi_i}$ for a given joint $i \in \{1, \ldots, m\}$. By the definition of the exponential map we can write q_i^R as a function depending on its log value $\vec{w} := (w_1, w_2, w_3) = \Phi_i$.

$$q_i^R(\vec{w}) = e^{\vec{\mathbf{w}}}$$

In section 4.3 we can find that the derivative of $q_i^R(\vec{w})$ is given by the following expressions.

$$\frac{\partial q_i^R(\vec{w})}{\partial w_k} = (a_k - w_k)\frac{\sin(||\vec{w}||)}{||\vec{w}||} + w_k \vec{\mathbf{w}} \left(\frac{\cos(||\vec{w}||)}{||\vec{w}||^2} - \frac{\sin(||\vec{w}||)}{||\vec{w}||^3} \right) \quad \text{for } ||\vec{w}|| > 0,\ k \in \{1, 2, 3\} \tag{1}$$

$$\frac{\partial q_i^R(\vec{w})}{\partial w_k} = a_k \quad \text{for } ||\vec{w}|| = 0,\ k \in \{1, 2, 3\} \tag{2}$$

$$\text{where } a_k = \mathbb{1}_{(k=1)}\mathbf{i} + \mathbb{1}_{(k=2)}\mathbf{j} + \mathbb{1}_{(k=3)}\mathbf{k}$$

The dual quaternion $\hat{q}_i^R(\vec{w})$ is given by $\hat{q}_i^R(\vec{w}) = (1 + \frac{1}{2}\epsilon \vec{\mathbf{t}}_i^R) q_i^R(\vec{w})$. We use the product rule for dual quaternions to obtain the derivative of $\hat{q}_i^R(\vec{w})$.

$$\frac{\partial \hat{q}_i^R(\vec{w})}{\partial w_k} = (1 + \frac{1}{2}\epsilon \vec{\mathbf{t}}_i^R) \frac{\partial q_i^R(\vec{w})}{\partial \vec{w}_k} \quad \text{for} \quad k \in \{1, 2, 3\} \tag{3}$$

Here we substitute $\frac{\partial q_i^R(\vec{w})}{\partial w_k}$ by equation 1 if $||\vec{w}|| > 0$, or by equation 2 if $||\vec{w}|| = 0$.

Since the relative orientations of the joints are independent of each other we can express the dual quaternion $\hat{q}_{\text{end}}^W(\vec{w})$ as follows.

$$\hat{q}_{\text{end}}^W(\vec{w}) = \hat{q}_0^W \hat{q}_1^R \cdots \hat{q}_{i-1}^R \hat{q}_i^R(\vec{w}) \hat{q}_{i+1}^R \cdots \hat{q}_{m-1}^R \hat{q}_m^R \hat{q}_{\text{end}}^R = \hat{q}_{i-1}^W \left(\hat{q}_i^R(\vec{w}) \right) \hat{q}_m^i \hat{q}_{\text{end}}^R \tag{4}$$

By the use of the product rule for dual quaternions we can find the derivative $\frac{\partial \hat{q}_{\text{end}}^W(\vec{w})}{\partial w_k}$ by a substitution of equation 3 in the following expression.

$$\frac{\partial \hat{q}_{\text{end}}^W(\vec{w})}{\partial w_k} = \hat{q}_{i-1}^W \frac{\partial \hat{q}_i^R(\vec{w})}{\partial \vec{w}} \hat{q}_m^i \hat{q}_{\text{end}}^R \quad \text{for} \quad k \in \{1, 2, 3\} \tag{5}$$

However, we are not interested in the derivative of the dual quaternion $\hat{q}_{\text{end}}^W(\vec{w})$, but in the derivative of $\vec{e}(\vec{w})$. We know that $\vec{e} = (\vec{t}_{\text{end}}^W, \vec{o}_{\text{end}}^W)$ with $\vec{o}_{\text{end}}^W = \log(q_{\text{end}}^W)$. Therefore we use the following equivalent relation of $\hat{q}_{\text{end}}^W(\vec{w})$.

$$\hat{q}_{\text{end}}^W(\vec{w}) = (1 + \frac{1}{2}\epsilon \vec{\mathbf{t}}_{\text{end}}^W(\vec{w})) q_{\text{end}}^W(\vec{w}) \tag{6}$$

Here we note that the rotation quaternion q_{end}^W and the translation vector $\vec{t}_{\text{end}}^W$ both depend on $\vec{w}$. By the use of the product rule of a dual quaternion this leads to the following derivative of $\hat{q}_{\text{end}}^W(\vec{w})$.

$$\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial w_k} = \frac{\partial}{\partial w_k}\left((1+\frac{1}{2}\epsilon\vec{\mathbf{t}}^W_{\text{end}}(\vec{w}))q^W_{\text{end}}(\vec{w})\right) \qquad \text{for} \quad k \in \{1,2,3\}$$

$$= \frac{\partial q^W_{\text{end}}(\vec{w})}{\partial w_k} + \frac{1}{2}\epsilon\left(\frac{\partial \vec{t}^W_{\text{end}}(\vec{w})}{\partial w_k} q^W_{\text{end}}(\vec{w}) + \vec{\mathbf{t}}^W_{\text{end}}(\vec{w})\frac{\partial q^W_{\text{end}}(\vec{w})}{\partial \vec{w}_k}\right) \qquad \text{for} \quad k \in \{1,2,3\} \tag{7}$$

We can use this expression to obtain the derivatives of $\vec{t}^W_{\text{end}}(\vec{w})$ and $q^W_{\text{end}}(\vec{w})$, for which we only need to substitute $\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial w_k}$ as in equation 5. $\vec{\mathbf{t}}^W_{\text{end}}(\vec{w})$ and $q^W_{\text{end}}(\vec{w})$ can be obtained from equation 6.

$$\frac{\partial \vec{\mathbf{t}}^W_{\text{end}}(\vec{w})}{\partial w_k} = \left(2\mathbb{D}\left(\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial \vec{w}_k}\right) - \vec{\mathbf{t}}^W_{\text{end}}(\vec{w})\mathbb{R}\left(\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial \vec{w}_k}\right)\right)\left(q^W_{\text{end}}(\vec{w})\right)^{-1} \qquad \text{for} \quad k \in \{1,2,3\} \tag{8}$$

$$\frac{\partial q^W_{\text{end}}(\vec{w})}{\partial w_k} = \mathbb{R}\left(\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial w_k}\right) \qquad \text{for} \quad k \in \{1,2,3\} \tag{9}$$

Here we use $\mathbb{R}(\cdot)$ to give the real part and $\mathbb{D}(\cdot)$ to give the dual part of a dual quaternion.

We remain with extracting the derivative of $\vec{o}^W_{\text{end}}(\vec{w}) := (o_1(\vec{w}), o_2(\vec{w}), o_3(\vec{w})) = \log(q^W_{\text{end}}(\vec{w}))$ from the derivative of $q^W_{\text{end}}(\vec{w})$. We use the chain rule to obtain the following expression.

$$\frac{\partial q^W_{\text{end}}(\vec{w})}{\partial w_k} = \frac{\partial q^W_{\text{end}}(\vec{w})}{\partial \vec{o}_l(\vec{w})}\frac{\partial \vec{o}_l(\vec{w})}{\partial w_k} \quad \text{for} \quad k \in \{1,2,3\}, \, l \in \{1,2,3\}$$

Hence, we can find the derivative of $\vec{o}^W_{\text{end}}(\vec{w})$ by

$$\frac{\partial \vec{o}_l(\vec{w})}{\partial \vec{w}_k} = S\left(\left(\frac{\partial q^W_{\text{end}}(\vec{w})}{\partial \vec{o}_l}\right)^* \frac{\partial q^W_{\text{end}}(\vec{w})}{\partial w_k}\right) \quad \text{for} \quad k \in \{1,2,3\}, \, l \in \{1,2,3\} \tag{10}$$

Here the function $S(\cdot)$ gives the scalar part of a quaternion. The derivative $\frac{\partial \hat{q}^W_{\text{end}}(\vec{w})}{\partial \vec{o}^W_{\text{end}}}$ can be obtained by theorem 4.4.7.

We finally found the partial derivatives of $\vec{e}(\vec{w})$ by equation 8 and equation 10.

$$\frac{\partial \vec{e}(\vec{w})}{\partial w_k} = (\frac{\partial \vec{t}^W_{\text{end}}(\vec{w})}{\partial w_k}, \frac{\partial \vec{o}^W_{\text{end}}(\vec{w})}{\partial w_k}) \quad \text{for} \quad k \in \{1,2,3\}$$

This gives us 3 columns of the Jacobian matrix for each joint. When a joint i does not solve for the end effector orientation, then we let $\frac{\partial \vec{o}^W_{\text{end}}(\vec{w})}{\partial \vec{w}_k} = \vec{0}$ for every $k \in \{1,2,3\}$.

Before we give the algorithm which computes the Jacobian matrix $J(\vec{e}, \Phi)$ as we described now, we first give the algorithm which computes the derivative of a quaternion to its log values as we described by theorem 4.4.7.

Algorithm 12 Compute D(q), the derivative of $q = e^{\vec{\mathbf{w}}}$ with respect to $\vec{w}$

Require: $q \in H_0$

$\vec{\mathbf{w}} := (0, w_1, w_2, w_3) = \log(q)$, $||\vec{w}|| = \sqrt{w_1^2 + w_2^2 + w_3^2}$

if $||\vec{w}|| > 0$ **then**

$b = \frac{\sin(||\vec{w}||)}{||\vec{w}||}$, $c = \frac{\cos(||\vec{w}||)}{||\vec{w}||^2} - \frac{\sin(||\vec{w}||)}{||\vec{w}||^3}$

$\frac{\partial q}{\partial w_1} = (-w_1 b\,,\, b + w_1 w_1 c\,,\, w_1 w_2 c\,,\, w_1 w_3 c)$

$\frac{\partial q}{\partial w_2} = (-w_2 b\,,\, w_2 w_1 c\,,\, b + w_2 w_2 c\,,\, w_2 w_3 c)$

$\frac{\partial q}{\partial w_3} = (-w_3 b\,,\, w_3 w_1 c\,,\, w_3 w_2 c\,,\, b + w_3 w_3 c)$

else

$\frac{\partial q}{\partial w_1} = (0\,,\, 1\,,\, 0\,,\, 0)$

$\frac{\partial q}{\partial w_2} = (0\,,\, 0\,,\, 1\,,\, 0)$

$\frac{\partial q}{\partial w_3} = (0\,,\, 0\,,\, 0\,,\, 1)$

end if

return $\frac{\partial q}{\partial \vec{w}} = (\frac{\partial q}{\partial w_1}, \frac{\partial q}{\partial w_2}, \frac{\partial q}{\partial w_3})$

Algorithm 13 Compute $J(\vec{e}, \Phi)$

Require: configuration Φ (fixed variables are given $\vec{t}_i^R$, $\vec{t}_{\text{end}}^R$ and $\hat{q}_{\text{end}}^R$ and $\hat{q}_0^W$)

for $i = 1, i = m$ **do**

$\hat{q}_i^R = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_i^R)e^{\vec{\Phi}_i}$

$\hat{q}_i^W = \hat{q}_{i-1}^W \hat{q}_i^R$

end for

$\hat{q}_{\text{end}}^W = \hat{q}_m^W \hat{q}_{\text{end}}^R$, $q_{\text{end}}^W = \mathbb{R}(\hat{q}_{\text{end}}^W)$, $\vec{\mathbf{t}}_{\text{end}}^W = 2\mathbb{D}(\hat{q}_{\text{end}}^W)(q_{\text{end}}^W)^*$

$\hat{q}_{\text{tail}} = \hat{q}_{\text{end}}^R$

for $i = m, i = 1$ **do**

if $i < m$ **then**

$\hat{q}_{\text{tail}} = \hat{q}_{i+1}^R \hat{q}_{\text{tail}}$

end if

$\frac{\partial q_i^R}{\partial \vec{\Phi}_i} = D(e^{\Phi_i})$

$\frac{\partial \hat{q}_i^R}{\partial \vec{\Phi}_i} = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}_i^R)\frac{\partial q_i^R}{\partial \vec{\Phi}_i}$

$\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i} = \hat{q}_{i-1}^W \frac{\partial \hat{q}_i^R}{\partial \vec{\Phi}_i} \hat{q}_{\text{tail}}$

$\frac{\partial \vec{\mathbf{t}}_{\text{end}}^W}{\partial \vec{\Phi}_i} = (2\mathbb{D}(\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i}) - \vec{\mathbf{t}}_{\text{end}}^W \mathbb{R}(\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i}))(q_{\text{end}}^W)^*$

$\frac{\partial q_{\text{end}}^W}{\partial \vec{o}_{\text{end}}^W} := (\frac{\partial q_{\text{end}}^W}{\partial o_1}, \frac{\partial q_{\text{end}}^W}{\partial o_2}, \frac{\partial q_{\text{end}}^W}{\partial o_3}) = D(q_{\text{end}}^W)$

$\frac{\partial \vec{\mathbf{o}}_{\text{end}}^W}{\partial \vec{\Phi}_i} = \left(S(\left(\frac{\partial q_{\text{end}}^W}{\partial o_1}\right)^* \mathbb{R}(\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i})), S(\left(\frac{\partial q_{\text{end}}^W}{\partial o_2}\right)^* \mathbb{R}(\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i})), S(\left(\frac{\partial q_{\text{end}}^W}{\partial o_3}\right)^* \mathbb{R}(\frac{\partial \hat{q}_{\text{end}}^W}{\partial \vec{\Phi}_i}))\right)$

$\frac{\partial \vec{e}}{\partial \vec{\Phi}_i} = (\frac{\partial \vec{\mathbf{t}}_{\text{end}}^W}{\partial \vec{\Phi}_i}, \frac{\partial \vec{\mathbf{o}}_{\text{end}}^W}{\partial \vec{\Phi}_i})^T$

end for

return $J(\vec{e}, \Phi) = (\frac{\partial \vec{e}}{\partial \vec{\Phi}_1}, \ldots, \frac{\partial \vec{e}}{\partial \vec{\Phi}_m})$

5.6 Comparison of the inverse kinematics methods

In this chapter different inverse kinematics methods are discussed. We started with showing that the inverse kinematics problem is hard to solve analytically. Therefore iterative methods were given to solve the inverse kinematics problem. These are the Cyclic Coordinate Descent method, the Jacobian Inverse method, the Jacobian Transpose method, the Pseudo Inverse Jacobian method and the Damped Least Squares method. We compare the performance of those methods on whether they have singularity problems, on their computational speed, on the naturalness of the solutions, on whether the methods are applicable for general chains and on how difficult the methods are to implement. The properties of each method are shown in the next table.

	singularities	speed	naturalness	apllicable to general chains	difficulty
Cyclic Coordante Descent	no	++	−+	yes	++
Jacobian Inverse	no	+−	+	no	+−
Jacobian Transpose	no	+	+−	yes	+
Pseudo Inverse Jacobian	yes	+−	−+	yes	+−
Damped Least Squares	barely	+−	+	yes	−+

Table 2: Properties of the inverse kinematics methods

Furthermore, we should notice that the Pseudo Inverse Jacobian method performs bad when a solution comes near a singluraty. The chain will start flipping up and down till it gets in a stable solution again. Furthermore, a joint in the chain can get stuck in a certain configuration. When we for example use this method for the motion of a human arm, we can find that a joint in the arm gets stuck in an overstretched configuration which means that the arm can not bow anymore.
The Cyclic Coordinate Descent method is seen as less naturall than the other inverse kinematics methods. Partly this is because it starts oscillating when a solution is stuck in a local minimum. This happens for example when the target is out of reach. Furthermore, the Jacobian transpose method appears slightly less natural than the Damped Least Squares method when used for human motion.

An example of the motion obtained with the Jacobian Transpose method is represented in figure 39 which can be found in the appendix.

6 Collisions

We speak of a collision when two objects collide with each other. Since we do not want this to happen we have to define how to handle it. We first define how we detect a collision and locate the position of this collision on the objects. Once we know where we have a collision, we can develop a collision response method which manipulates the chain such that the objects do not collide with each other anymore.

6.1 Collision Detection

We speak of collision when the virtual character collides with an object in the game environment. The virtual character can be considered as an object also. The objects and the virtual character are decomposed in convex sets. When there is a collision, we define a contact as the position on the convex set of the virtual character with a highest depth value inside a convex set of the object. A pair of convex sets gives only one contact. There can be multiple contacts when there is collision between multiple convex sets from the virtual doctor and object. The use of convex sets instead of general sets makes the collision detection easier and faster.

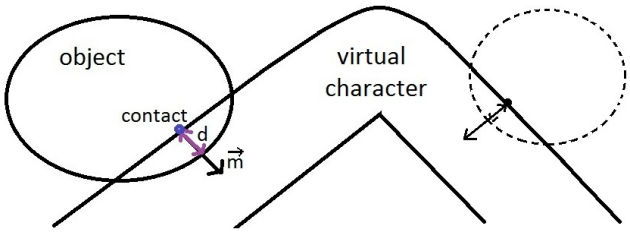


Figure 25: A contact with depth d and normal vector $\vec{m}$.

When we detect a collision and obtain a contact, then we will endow the contact with a position in world coordinates, with the contact depth inside the object and with the contact's normal vector which defines the direction the contact should move to get outside the object. We will assume that the contact normal vector is unit.

We will now show how we can create convex sets to represent the virtual character and to represent the obstacles.

The convex sets to represent the virtual character
To obtain convex sets for the virtual character we make use of bounding boxes. A bounding box is an easy convex shape which surrounds an object as closely as possible. Just using one bounding box for the complete virtual character will not work. We would rather define a bounding box for each rigid body part of the virtual character. We do this by putting a bounding box in the shape of a sigar around every link of the skeleton of the virtual character. The sigar shape is such that every point on this bounding box has the same shortest distance to the link, see figure 26. We have now obtained convex sets which stay close to the original virtual character, even when we give motion to it.

Figure 26: The bounding box of a link

However, there is one drawback in defining multiple convex sets for the virtual character. The convex sets overlap each other around the joints, see figure 27.
When we detect collisions between these convex sets and an object, this can lead to the detection of

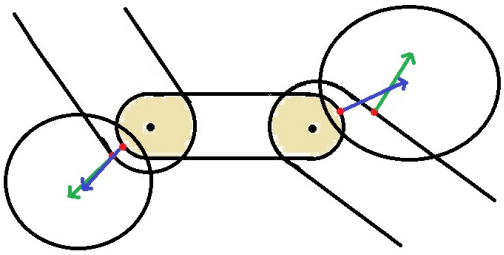
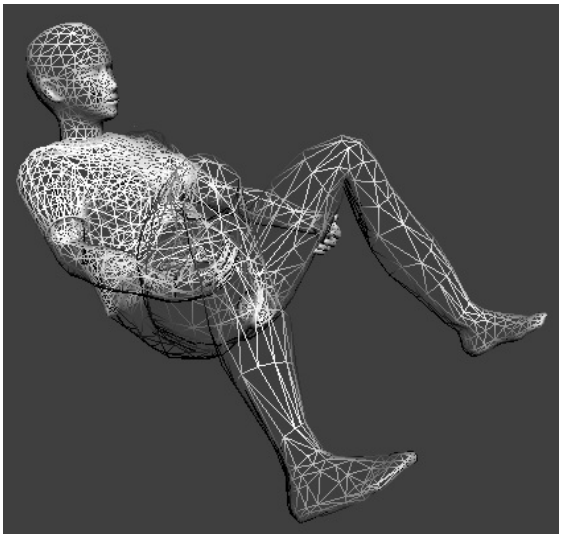


Figure 27: Overlapping convex sets in the skeleton model

two contacts which are similar or two conflicting contacts. However, when we can ensure that the collision depth remains small, then the probability of finding two conflicting contacts is relatively small.

The convex sets to represent the obstacles
An obstacle can be represented in different ways within a game environment. The most used way is creating a mesh that represents the object and place it in the environment. In case the object involves another virtual character, then again a skinned skeleton can be used to represent it.



(a) A triangular mesh

(b) A skinned skeleton

Figure 28

Before we define how we will cover the objects with convex sets, we first need to know how close we want the virtual character to be able to move along the object. This will define how precise we need to cover the object with convex sets. We can cover the obstacle with a single smooth convex bounding box when we are fine with restricting the motion of the virtual character outside this bounding box. The use of a single convex set instead of many convex sets to cover an obstacle will benefit the computational speed of the collision detection.
However, when we want the virtual character to be able to move closely along an obstable, we need to be more precise in covering this obstacle.

We can cover the object with simple smooth convex sets which overlap each other as little as possible. We will not discuss how this is done, since this depends on the characteristics of the object. When the obstacle has a mesh, then it is not necessary to find a cover of convex sets in order to perform collision detection. In this case the small shapes which create the mesh of an object can be used as convex sets for the collision detection. In figure 28a we can see the mesh of an obstable built out of small triangles.
The use of the mesh gives the most precise convex sets for the collision detection, but it also has some drawbacks. One of the drawbacks is the fact that a mesh is generally not smooth. By this a small change of the contact position can bring a big change to the direction of the normal vector which gives the direction a contact on the virtual character should move to become collision free. This means that the motion of the virtual character can be unsmooth as the result of the collision response with a mesh. When we bring a small change to the contact position on a smooth object, then the direction of the normal vector does not change that much. We also had this problem of large changes in normal vector with the object cover of smooth convex sets in the area where those sets were overlapping. The union of two smooth convex sets may not be smooth. However, mesh normals can be interpolated to form a smooth manifold.

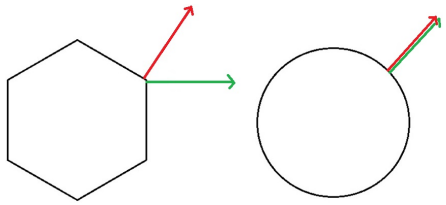


Figure 29: The change in direction of the normal vector for a small change in contact position

Moreover, we will get multiple conflicting contacts when the local area of collision involves more than one of the pieces of the mesh. Aside from this, the mesh of an object is mostly built out of a large number of small convex shapes, which will lead to high computational cost for the collision detector.

The collision detector
For the collision detection we use a method based on the Gilbert-Johnson-Keerthi algorithm, also called the GJK-algorithm. This GJK-algorithm is an iterative method which computes the closest distance between two arbitrary convex sets and which can also return a closest pair of points. We use this algorithm first of all to determine whether two convex sets are colliding. Here one of the convex sets corresponds to the virtual character and the other corresponds to the obstable. When two convex sets collide, then the closest distance between them is zero. When two convex sets are not colliding, then the closest distance between them is larger than zero. Once a collision is detected, we let the GJK-algorithm determine a pair of closest points on the boundary of the convex sets and the distance between them. This provides us the contact position on the virtual character and the depth value inside the obstacle. Aside from this the methodology of the GJK-algorithm provides a vector which gives the direction that the convex set representing the virtual character should move in to become collision free from the obstacle. We can find a full detail description and efficient implementation of the GJK-algorithm for collision detection in [van den Bergen, 1999].

6.2 The contact Jacobian matrix

The collision detector detects given a configuration whether there is a collision between the virtual character and an obstacle. When there is a collision, the collision detector determines the contact points. We do not want our virtual character to collide with obstacles. Therefore, we need to change the configuration of the virtual character such that the virtual character becomes collision free again. In chapter 5 we saw how the Jacobian method determines a change in the configuration given a desired change in end effector position and orientation. The Jacobian method uses a Jacobian matrix which represents the change in end effector position and orientation given the change in configuration. In a similar way, we can define a Jacobian matrix for a contact which represents the change in contact depth with respect to the change in configuration. By this we can define a Jacobian method which determines the change in configuration of the chain given a desired change in contact depth.

We let the Jacobian matrix represent the change in depth value with respect to the change in configuration because we know that decreasing the contact depth means that the contact is moving away from the obstacle. We could also have chosen to let the Jacobian matrix represent the change in contact position with respect to the change in configuration, since we know in which direction we want the position of the contact to move to become collision free by the contact's normal vector. The reason why we prefer to use the first representation of the Jacobian matrix is that it gives more freedom to the direction in which a contact can move in order to become collision free than the latter representation. This larger freedom in direction is useful when we have to make multiple contacts collision free. Still, it remains that the change in contact depth depends on the direction of the normal vector.

The change in contact depth Δd equals the dot product of the change in contact position $\Delta\vec{p}$ with the contact's normal vector $\vec{m}$, as we can see in figure 30. This means that we first need to determine the change in contact position with respect to the change in configuration in order to determine the change in contact depth with respect to the change in configuration. We first define how we locate the contact on the chain. We say that a contact belongs to joint k when k is the last joint in the chain which has influence on the contacts position for a change in configuration. This means that the contact lays on the rigid body part which corresponds to the link which connects joint k with joint $k+1$. Here, joint $m+1$ can be seen as the virtual joint which represents the end effector connected to end joint m. A joint can hold multiple contacts.

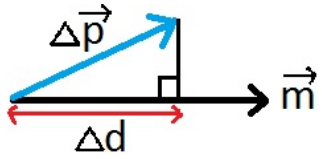


Figure 30

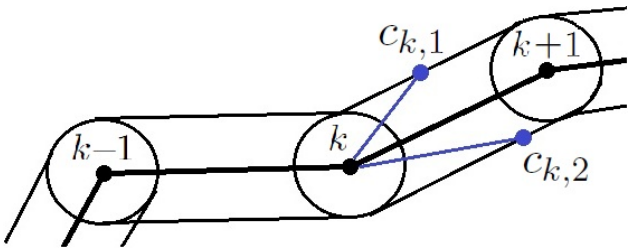


Figure 31: Contact location

This leads us to the following notations for the contacts.

Notation 6.2.1. *Define the following symbols for the contacts.*

- $c_{k,l}$ *is the* l*-th contact of joint* k*,*
- c_k *is the set of contacts of joint* k*, which has* $|c_k|$ *elements,*
- $\vec{p}^{\,R}_{k,l}$ *is the position of contact* $c_{k,l}$ *in the coordinate system of joint* k*,*
- $\vec{p}^{\,W}_{k,l}$ *is the position of contact* $c_{k,l}$ *in the world coordinate system,*
- $\vec{m}^{\,W}_{k,l}$ *is the normal vector of contact* $c_{k,l}$ *in world coordinates which gives the direction we should move our contact to get outside the object,*
- $d_{k,l}$ *is the depth value of contact* $c_{k,l}$*,*
- d_k *is the vector of depth values corresponding to the set of contacts* c_k*.*

We defined a contact $c_{k,l}$ such that k is the last joint which has influence on the contact position $p^W_{k,l}$ when we change the configuration. This means that the change in contact position $p^W_{k,l}$ given a change in configuration $\vec{\Phi}_i$ equals zero if $i > k$. This also means that the change in contact depth $d_{k,l}$ given a change in configuration $\vec{\Phi}_i$ equals zero if $i > k$.

$$\frac{\partial \vec{p}^{\,W}_{k,l}}{\partial \vec{\Phi}_i} = \vec{0} \quad \text{and} \quad \frac{\partial \vec{d}_{k,l}}{\partial \vec{\Phi}_i} = \vec{0} \quad \text{for} \quad i > k$$

We determine the change in position $p^W_{k,l}$ given a change in configuration $\vec{\Phi}_i$ with the help of the change of the dual quaternion $\hat{q}^W_{c_{k,l}}$ with respect to the change in configuration $\vec{\Phi}_i$ for $i \leq k$. Here $\hat{q}^W_{c_{k,l}}$ represents the position of contact $c_{k,l}$.

A contact has a position, but it has no orientation. Therefore, we define the dual quaternion which represents the position of contact $c_{k,l}$ in the coordinate system of joint k by the following expression.

$$\hat{q}^R_{c_{k,l}} = (1 + \frac{1}{2}\epsilon \vec{\mathbf{p}}^R_{k,l})$$

Since the position $p^R_{k,l}$ of $c_{k,l}$ in the coordinate system of joint k is a fixed variable, we have that $\hat{q}_{c_{k,l}}$ is also a fixed variable. We can now determine $\hat{q}^W_{c_{k,l}}$.

$$\hat{q}^W_{c_{k,l}}(\Phi) = \hat{q}^W_{c_{k,l}}(\vec{\Phi}_1, \ldots, \vec{\Phi}_k) = \hat{q}^w_0 \hat{q}^R_1(\vec{\Phi}_1) \cdots \hat{q}^R_k(\vec{\Phi}_k) \hat{q}^R_{c_{k,l}}$$

This dual quaternion has an orientation given by $(q^W_k)^{-1}$ which we can ignore. $\hat{q}^W_{c_{k,l}}$ depends only on the configurations $\Phi_i, \ldots, \vec{\Phi}_k$. We can calculate the following partial derivatives of $\hat{q}^W_{c_{k,l}}$.

$$\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i} = \hat{q}^w_0 \hat{q}^R_1 \cdots \hat{q}^R_{i-1} \frac{\partial \hat{q}^R_i(\vec{\Phi}_1)}{\partial \vec{\Phi}_i} \hat{q}^R_{i+1} \cdots \hat{q}^R_k \hat{q}^R_{c_{k,l}} \quad \text{for} \quad i \leq k$$

Here, $\frac{\partial \hat{q}^R_i(\vec{\Phi}_1)}{\partial \vec{\Phi}_i}$ can be obtained with $\frac{\partial q^R_i(\vec{\Phi}_1)}{\partial \vec{\Phi}_i}$, which can be computed by algorithm 12, together with the fixed translation $\vec{t}^{\,R}_i$. We can derive $\frac{\partial \vec{\mathbf{p}}^W_{k,l}}{\partial \vec{\Phi}_i}$ from $\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i}$ for all $i \leq k$.

$$\frac{\partial \vec{\mathbf{p}}_{k,l}}{\partial \vec{\Phi}_i} = \left(2\mathbb{D}\left(\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i}\right) - \vec{\mathbf{p}}^W_{k,l}\mathbb{R}\left(\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i}\right)\right)(\mathbb{R}(\hat{q}^W_{c_{k,l}}))^* \quad \text{for} \quad i \leq k$$

Finally, we derive $\frac{\partial d_{k,l}}{\partial \vec{\Phi}_i}$ from $\frac{\partial \vec{p}_{k,l}}{\partial \vec{\Phi}_i}$ with the help of the normal vector $\vec{m}_{k,l}$.

$$\frac{\partial d_{k,l}}{\partial \vec{\Phi}_i} = \frac{\partial \vec{p}_{k,l}}{\partial \vec{\Phi}_i} \cdot \vec{m}_{k,l} \quad \text{for} \quad i \leq k$$

We can now define the Jacobian matrix $J_{c_{k,l}}$ for contact $c_{k,l}$, which gives the change in depth value $d_{k,l}$ with respect to the change in configuration Φ. Algorithm 14 gives the computation of $J_{c_{k,l}}$.

Notation 6.2.2. *The contact Jacobian matrix of a contact $c_{k,l}$ is given by*

$$J_{c_{k,l}} = J(d_{k,l}, \Phi) = (\frac{\partial d_{k,l}}{\partial \vec{\Phi}_1}, \ldots, \frac{\partial d_{k,l}}{\partial \vec{\Phi}_k}, \frac{\partial d_{k,l}}{\partial \vec{\Phi}_{k+1}}, \ldots, \frac{\partial d_{k,l}}{\partial \vec{\Phi}_m})$$

with $\frac{\partial d_{k,l}}{\partial \vec{\Phi}_i} = \vec{0}$ for $i > k$.

Algorithm 14 Compute $J_{c_{k,l}}$, the Jacobian matrix of contact $c_{k,l}$ with $k \in \{1, \ldots, m+1\}$

Require: configuration Φ and contact $c_{k,l}$ (fixed variables are given $\vec{t}^R_i$, $\hat{q}^W_0$, $\vec{p}^R_{k,l}$ and $\vec{m}^W_{k,l}$)
  for $i = 1, i = k$ **do**
    $\hat{q}^R_i = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}^R_i)e^{\vec{\Phi}_i}$
    $\hat{q}^W_i = \hat{q}^W_{i-1}\hat{q}^R_i$
  end for
  $\hat{q}^R_{c_{k,l}} = (1 + \frac{1}{2}\epsilon\vec{p}^R_{k,l})$, $\hat{q}^W_{c_{k,l}} = \hat{q}^W_k\hat{q}^R_{c_{k,l}}$
  $\hat{q}_{\text{tail}} = \hat{q}^R_{c_{k,l}}$
  for $i = k, i = 1$ **do**
    if $i < k$ **then**
      $\hat{q}_{\text{tail}} = \hat{q}^R_{i+1}\hat{q}_{\text{tail}}$
    end if
    $\frac{\partial q^R_i}{\partial \vec{\Phi}_i} = D(\vec{\Phi}_i)$ by algorithm 12
    $\frac{\partial \hat{q}^R_i}{\partial \vec{\Phi}_i} = (1 + \frac{1}{2}\epsilon\vec{\mathbf{t}}^R_i)\frac{\partial q^R_i}{\partial \vec{\Phi}_i}$
    $\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i} = \hat{q}^W_{i-1}\frac{\partial \hat{q}^R_i}{\partial \vec{\Phi}_i}\hat{q}_{\text{tail}}$
    $\frac{\partial \vec{\mathbf{p}}^W_{k,l}}{\partial \vec{\Phi}_i} = (2\mathbb{D}(\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i}) - \vec{\mathbf{p}}^W_{k,l}\mathbb{R}(\frac{\partial \hat{q}^W_{c_{k,l}}}{\partial \vec{\Phi}_i}))(\mathbb{R}(\hat{q}^W_{c_{k,l}}))^*$
    $\frac{\partial d_{k,l}}{\partial \vec{\Phi}_i} = \frac{\partial \vec{p}^W_{k,l}}{\partial \vec{\Phi}_i} \cdot \vec{m}^W_{k,l}$
  end for
  for i=k+1,i=m **do**
    $\frac{\partial d_{k,l}}{\partial \vec{\Phi}_i} = \vec{0}$
  end for
  return $J_{c_{k,l}} = (\frac{\partial d_{k,l}}{\partial \vec{\Phi}_1}, \ldots, \frac{\partial d_{k,l}}{\partial \vec{\Phi}_m})$

Furthermore, we can define a contact Jacobian for all the contacts of the joints in a chain.

Definition 6.2.3. *Given a chain* $(1, \ldots, m)$*, we define the set of contacts by* $\{c\} = (c_1, \ldots, c_m)$ *and we define the following contact Jacobian matrix for multiple contacts* $\{c\}$ *by*

$$J_{\{c\}} = (J_{c_1}, J_{c_2}, \ldots, J_{c_m})^T$$

where J_{c_i} *is the contact Jacobian for the set of contacts of joint* $i \in (1, \ldots, m)$ *given by*

$$J_{c_i} = (J_{c_{i,1}}, J_{c_{i,2}}, \ldots, J_{c_{i,|c_i|}})^T = (J(d_{i,1}, \Phi), J(d_{i,2}, \Phi), \ldots, J(d_{i,|c_i|}, \Phi))^T.$$

Furthermore, we define $d_{\{c\}}$ *as the vector with all depth values of the contacts* $\{c\}$ *given by*

$$d_{\{c\}} = (d_1, d_2, \ldots, d_m) \quad \textit{with} \quad d_i = (d_{i,1}, \ldots, d_{i,|c_i|}) \quad \forall i \in (1, \ldots, m)$$

Algorithm 15 determines the contact Jacobian $J_{\{c\}}$ for a set of contacts $\{c\}$ which represents the change in depth values of the contacts with respect to the change in configuration Φ.

Algorithm 15 Compute $J_{\{c\}}$

Require: Φ, $m_{k,l}, \vec{p}_{k,l}, d_{k,l}$ for all contacts $c_{k,l} \in \{c\}$
  for $k = 1, k = m$ **do**
    $l = 1$
    while $l \leq |c_k|$ **do**
      $J_{c_{k,l}} = J(d_{k,l}, \Phi)$ by algorithm 14
      $l = l + 1$
    end while
    $J_{c_k} = (J_{c_{k,1}}, \ldots, J_{c_{k,|c_k|}})$
  end for
  return $J_{\{c\}} = (J_{c_1}, J_{c_2}, \ldots, J_{c_m})$

6.3 Collision Response

In this section we will develop an inverse kinematics method based on the Jacobian methods which can give a collision response. We want this method to derive a stable solution which solves for a given change in end effector position, but which will also move the contacts out of the obstacles when there is a collision.

In the previous section we derived the contact Jacobian matrix $J_{\{c\}}$ for a set of contacts $\{c\}$ which represents the change in contact depths $\Delta d_{\{c\}}$ given a change in configuration $\Delta\Phi$. In order to know how we should change the configuration of a chain in order to change the depth values in a desired way, we use the following linear approximation.

$$\Delta d_{\{c\}} \approx \frac{\partial d_{\{c\}}}{\partial \Phi}\Delta\Phi = J_{\{c\}}\Delta\Phi \tag{11}$$

This means that a small change of $\Delta\Phi$ to the current configuration Φ will bring a small change of $\Delta d_{\{c\}}$ to the current depth values $d_{\{c\}}$. However, what we need is the other way around. We need to know how we should bring a small change $\Delta\Phi$ to the current configuration Φ in order to change the current depth values $d_{\{c\}}$ by a desired amount $\Delta d_{\{c\}}$. Since the contact Jacobian matrix in general has no inverse, we have to use one of the following expressions to determine the change in configuration $\Delta\Phi$.

$$\Delta\Phi = J_{\{c\}}^T \Delta d_{\{c\}} \quad \text{or,} \tag{12}$$

$$\Delta\Phi = J_{\{c\}}^\dagger \Delta d_{\{c\}} \quad \text{or,} \tag{13}$$

$$\Delta\Phi = J_{\{c\}}^T (J_{\{c\}} J_{\{c\}}^T + \gamma^2 I)^{-1} \Delta d_{\{c\}} \tag{14}$$

We will represent $\Delta\Phi$ by expression 12 in the remainder of this section. However, we should know that we can also use one of the other two expressions.

To move the contacts out of the object, we need to change the contact depths $d_{\{c\}}$ such that they become zero. This means that we desire a change of $\Delta d_{\{c\}} = d_{\{c\}} - 0 = d_{\{c\}}$ to the depth values $d_{\{c\}}$. The approximation in equation 12 is only true for small changes $\Delta d_{\{c\}}$. Therefore, we change $d_{\{c\}}$ only with a fraction $\beta \in (0, 1)$ of $\Delta d_{\{c\}}$.

$$\Delta\Phi = \beta J_{\{c\}}^T \Delta d_{\{c\}}$$

When we solved for a desired change in end effector position and orientation given a configuration, we used the following expression.

$$\Delta\Phi = \alpha J^T \Delta\vec{e}$$

As a first attempt of a collision response method, we decide to solve for the end effector position and orientation when there is no collision with an object and we solve for the depth values when there is collision. We can find this method in algorithm 16. We notice that we do not check for collisions during the iterations of the algorithm.
When we look at the performance of this algorithm for a trajectory of target positions for the end effector, then we observe that the virtual character sticks for a moment on an object when there

is a collision. With sticking on an object we mean that the configuration of the chain remains such that a contact does not get free from the object, even when the target position for the end effector is moving in a direction away from the object. This happens even when there is only a single contact.

Algorithm 16 Collision response 1

Require: Φ^0, $\vec{g}$, $\vec{p}^{\,W}_{k,l}$, $\vec{m}_{k,l}$ and $d_{k,l}$ $\forall c_{k,l} \in \{c\}$, N, t_i^R $\forall i \in (1,\ldots,m)$, $\hat{q}_0^W$ and $\hat{q}^R_{\text{end}}$

```
j=0
if {c} ≠ ∅ then
  for i=1,i=m do
    q̂_i^R = (1 + ½ t⃗_i^R) e^{Φ^0}
    q̂_i^W = q̂_{i-1}^W q̂_i^R, q_i^W = ℝ(q̂_i^W), t⃗_i^W = 2𝔻(q̂_i^W)(q_i^W)*
  end for
  p⃗^R_{k,l} = (q̂_k^W)*(p⃗^W_{k,l} − t⃗_k^W) q_k^W,   p⃗^old_{k,l} = p⃗^W_{k,l}   ∀c_{k,l} ∈ {c}
  d^0_{{c}} = d_{{c}}
  while j < N do
    J_{{c}} = J(d^j_{{c}}, Φ) by algorithm 15
    Δd^j_{{c}} = d^j_{{c}}
    ΔΦ^j = β J^T_{{c}} Δd^j_{{c}}
    Φ^{j+1} = Φ^j + ΔΦ^j
    for i=1,i=m do
      Φ⃗_i^{j+1} = log(CLAMPED(e^{Φ⃗_i^{j+1}}))
      q̂_i^R = (1 + ½ t⃗_i^R) e^{Φ⃗_i^{j+1}}
      q̂_i^W = q̂_{i-1}^W q̂_i^R, q_i^W = ℝ(q̂_i^W), t⃗_i^W = 2𝔻(q̂_i^W)(q_i^W)*
    end for
    p⃗^W_{k,l} = t⃗_k^W + q_k^W p⃗^R_{k,l} (q_k^W)*   ∀c_{k,l} ∈ {c}
    d^{j+1}_{k,l} = d^0_{k,l} − (p⃗^W_{k,l} − p⃗^old_{k,l}) · m_{k,l}   ∀c_{k,l} ∈ {c}
    j = j + 1
  end while
else
  e⃗_0 = f(Φ^0).
  while j < N) do
    J = J(e⃗_j, Φ^j)
    Δe⃗_j = α(g⃗ − e⃗_j), ΔΦ^j = J^T Δe⃗_j
    Φ^{j+1} = Φ^j + ΔΦ^j
    for i=1,i=m do
      Φ⃗_i^{j+1} = log(CLAMPED(e^{Φ⃗_i^{j+1}}))
    end for
    e⃗_{j+1} = f(Φ^{j+1})
    j = j + 1
  end while
end if
return Φ^j
```

The method tries to bring the contact depth to exactly zero. A contact remains till its depth is smaller than or equal to zero. By this the contact can not or takes long to get outside an object. When we decrease the depth with approximately $\beta d_{k,l}$ at every iteration, then after N iterations we have decreased the depth to $(1-\beta)^N d_{k,l} > 0$. Only the small error in the change of depth value, caused by the fact that equation 11 is a linear approximation, can help a contact to get outside an object. Furthermore, the method is expected to oscillate when it succeeds to push a contact outside the object. This since the method does not solve for the end effector position and orientation while solving the contacts outside the objects.

To solve the problem of sticking on an object, we need to change the depth by $\Delta d_{k,l} = d_{k,l} + \delta$ for a small $\delta > 0$ in order to make contact $c_{k,l}$ free from the object. This means that we should solve for $\Delta\Phi = \beta J_{\{c\}}^T(\Delta d_{\{c\}} + \delta)$ instead of $\Delta\Phi = \beta J_{\{c\}}^T \Delta d_{\{c\}}$ in algorithm 16.

Even this method has some problems. When we consider a trajectory of target position for the end effector which moves in the oposite direction of a contacts normal vector, then the virtual character will eventually still move through the object. In figure 32 we can see how this happens. At a first contact, the method moves the contacts outside the object. When all joints in the chain are contact free again, the method solves the end effector to the new target. Solving for this new target might lead to a configuration with contacts which have an even larger depth inside the object than the previous time. Even the number of contacts might increase.

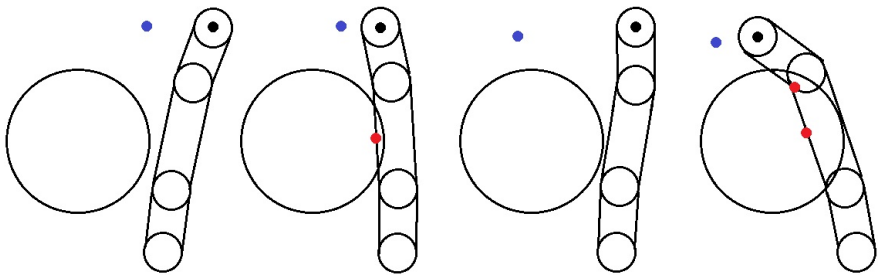


Figure 32

There are several tricks with which we can avoid that we get contacts which lay deep inside an object. The first one is retargeting. With retargeting we mean that we replace the target when the distance between the target and end effector is too large. When the distance is larger than a maximum distance M_d, then we linearly place the new target between the end effector and target with a distance M_d from the end effector. This means that we only allow the target to be a distance M_d away from the end effector. We give the retarget function in the next algorithm.

Algorithm 17 Retarget$(\vec{g}, \vec{e}, M_d)$

Require: end effector $\vec{e} := (\vec{t}^W_{\text{end}}, \vec{o}^W_{\text{end}})$, target $\vec{g} := (\vec{t}^W_{\text{target}}, \vec{o}^W_{\text{target}})$, maximum distance M_d
  if $||\vec{t}^W_{\text{target}} - \vec{t}^W_{\text{end}}|| > M_d$ **then**
    $\vec{t}^W_{\text{target}} = \vec{t}^W_{\text{end}} + M_d \frac{\vec{t}^W_{\text{target}} - \vec{t}^W_{\text{end}}}{||\vec{t}^W_{\text{target}} - \vec{t}^W_{\text{end}}||}$
  end if
  return $\vec{g} = (\vec{t}^W_{\text{target}}, \vec{o}^W_{\text{target}})$

Another trick to avoid that we get contacts with a high depth value is to constrain the joint velocities. With the joint velocity we mean the change in rotation angle when we rotate a joint. The change in rotation angle of joint i is given by $2||\Delta\vec{\Phi}_i||$. We give each joint i its own maximum velocity M_i. This way we can prioritize the change of a certain joint above other joints to create a more natural motion. We give the function to constrain the joint velocities in the next algorithm.

Algorithm 18 ClampVelocities($\Delta\Phi$)

Require: $\Delta\Phi := (\Delta\vec{\Phi}_1, \ldots, \Delta\vec{\Phi}_m)$, $M_1, \ldots, M_m$
  for i=1,i=m **do**
    if $2||\Delta\vec{\Phi}_i|| > M_i$ **then**
      $\Delta\vec{\Phi}_i = \Delta\vec{\Phi}_i \frac{M_i}{2||\Delta\vec{\Phi}_i||}$
    end if
  end for
  return $\Delta\Phi = (\Delta\vec{\Phi}_1, \ldots, \Delta\vec{\Phi}_m)$

The maximum distance M_d and maximum velocities $M_1, \ldots, M_m$ should be chosen small enough such that the contact depths can not get to large and large enough such that it does not create a slow motion to the motion of the chain. Here, we mean with a slow motion that the configuration of the chain can not let the end effector catch up with the target positions.
Aside from this, we can only define the maximum distance M_d and maximum velocities $M_1, \ldots, M_m$ as constants if the solutions are rendered with a constant rate. Otherwise, we should let the variables M_d and $M_1, \ldots, M_m$ depend on this rate.

For the case where we only have a single contact, the additional depth δ and the constrainment of the joint velocities and distance between the end effector and target give a big improvement on the performance of algorithm 16. When a collision free configuration excists, the method will move the contact out of the object. Also, the method avoids that a contact can get deep inside the object.
For the case where there are multiple contacts present, we should realize that those contacts can conflict with each other. There might not be a solution which brings each contact out of the object on a distance which is exactly δ. Therefore, we need to give more freedom to the solutions. We increase the freedom of the solutions by allowing solutions which bring a contact outside the object on a distance larger than zero from the object. We consider the distance that the contact has gone outside the object within an iteration as the distance we desire a contact to be outside the object. This means that the next iteration we will consider the desired change in depth of the contact as zero. This gives the other contacts more freedom to also get free from the object.

In algorithm 19, we give the new method for collision response with all improvements. This method works well when we consider the fact that it makes all contacts free from the objects when there is a collision. Also, this method prevents that there arise contacts with a high depth value. The fact remains that the motion of the virtual character obtained by this method is unstable; the chain oscillates when the virtual character collides with an object. This is caused by the fact that we do not solve for the end effector position and orientation while making the contacts free from the object. The amplitude of this oscillation depends on the amount with which a contact can get inside an object and on the amount with which the contact is moved out of the object again. Therefore, we need to take the δ's as small as possible (but large enough to avoid sticking) and

we need to take very small step sizes per solution such that the contact depths remain as small as possible to make this method as stable as possible. When we want to use this method for real time motion it require a very high computational speed to obtain a stable motion.

Algorithm 19 collision response 2

Require: Φ^0, $\vec{g}$, $\vec{p}^{\,W}_{k,l}$, $\vec{m}_{k,l}$ and $d_{k,l}$ $\forall c_{k,l} \in \{c\}$, N, δ, M_d, M_i, t_i^R $\forall i \in (1,\ldots,m)$, $\hat{q}_0^W$ and $\hat{q}^R_{\text{end}}$

j=0

if $\{c\} \neq \emptyset$ **then**

for i=1,i=m **do**

$\hat{q}_i^R = (1 + \frac{1}{2}\vec{\mathbf{t}}_i^R)e^{\Phi^0}$

$\hat{q}_i^W = \hat{q}_{i-1}^W \hat{q}_i^R, \quad q_i^W = \mathbb{R}(\hat{q}_i^W), \quad \vec{t}_i^{\,W} = 2\mathbb{D}(\hat{q}_i^W)(q_i^W)^*$

end for

$\vec{p}^{\,R}_{k,l} = (\hat{q}_k^W)^*(\vec{p}^{\,W}_{k,l} - \vec{t}_k^{\,W})q_k^W, \quad \vec{p}^{\,\text{old}}_{k,l} = \vec{p}^{\,W}_{k,l} \quad \forall c_{k,l} \in \{c\}$

$d^0_{\{c\}} = d_{\{c\}}$

while $j < N$ **do**

$J_{\{c\}} = J(d^j_{\{c\}}, \Phi)$ by algorithm 15

if $d^j_{\{c\}} < 0$ **then**

$\Delta d_{\{c\}} = 0$

else

$\Delta d_{\{c\}} = d_{\{c\}} + \delta$

end if

$\Delta\Phi^j = \beta J^T_{\{c\}} \Delta d^j_{\{c\}}, \quad \Phi^{j+1} = \Phi^j + \Delta\Phi^j$

for i=1,i=m **do**

$\vec{\Phi}_i^{j+1} = \log(CLAMPED(e^{\vec{\Phi}_i^{j+1}}))$

$\hat{q}_i^R = (1 + \frac{1}{2}\vec{t}_i^R)e^{\vec{\Phi}_i^{j+1}}$

$\hat{q}_i^W = \hat{q}_{i-1}^W \hat{q}_i^R$, $q_i^W = \mathbb{R}(\hat{q}_i^W)$, $\vec{t}_i^{\,W} = 2\mathbb{D}(\hat{q}_i^W)(q_i^W)^*$

end for

$\vec{p}^{\,W}_{k,l} = \vec{t}_k^{\,W} + q_k^W \vec{p}^{\,R}_{k,l}(q_k^W)^* \quad \forall c_{k,l} \in \{c\}$

$d^{j+1}_{k,l} = d^0_{k,l} - (\vec{p}^{\,W}_{k,l} - \vec{p}^{\,\text{old}}_{k,l}) \cdot m_{k,l} \quad \forall c_{k,l} \in \{c\}$

$j = j + 1$

end while

else

$\vec{e}_0 = f(\Phi^0)$. $\vec{g} = \text{Retarget}(\vec{g}, \vec{e}_0, M_d)$

while $j < N$) **do**

$J = J(\vec{e}_j, \Phi^j), \quad \Delta\vec{e}_j = \alpha(\vec{g} - \vec{e}_j)$

$\Delta\Phi^j = J^T\Delta\vec{e}_j$, $\Delta\Phi^j = \text{ClampVelocities}(\Delta\Phi^j)$

$\Phi^{j+1} = \Phi^j + \Delta\Phi^j$

for i=1,i=m **do**

$\vec{\Phi}_i^{j+1} = \log(\text{CLAMPED}(e^{\vec{\Phi}_i^{j+1}}))$

end for

$\vec{e}_{j+1} = f(\Phi^{j+1}), \quad j = j + 1$

end while

end if

return Φ^j

Finally, we notice that there might be multiple solutions which bring the contacts outside the object. In this case, we would like to take the solution which brings the end effector the closest to the target. We know that a general solution to the equation $J_{\{c\}}\Delta\Phi = \Delta d_{\{c\}}$ is given by the next expression.

$$\Delta\Phi = J^{\dagger}_{\{c\}}\Delta d_{\{c\}} + (I - J^{\dagger}_{\{c\}}J_{\{c\}})z$$

Here, z is an arbitrary vector in the $\Delta\Phi$-space and $(I - J^{\dagger}_{\{c\}}J_{\{c\}})$ is the projection operator to the nullspace of $J_{\{c\}}$. This means that $(I - J^{\dagger}_{\{c\}}J_{\{c\}})z$ gives us the directions in which we can change the configuration without influencing the change in depth values obtained by $J_{\{c\}}\Delta\Phi$. Since z is an arbitrary vector in $\Delta\Phi$ space, we can use it to solve for a secndary goal. This secondary goal is to solve the end effector towards the target. This means we have a primary goal of solving $J_{\{c\}}\Delta\Phi = \Delta d_{\{c\}}$ and a secondary goal to solve $J\Delta\Phi = \Delta\vec{e}$.

We already know the general solution to our primary goal. When we substitute this general solution in the secondary goal, then we can determine z.

$$\Delta d_{\{c\}} = J\Delta\Phi = J(J^{\dagger}_{\{c\}}\Delta d_{\{c\}} + (I - J^{\dagger}_{\{c\}}J_{\{c\}})z) = JJ^{\dagger}_{\{c\}}\Delta d_{\{c\}} + J(I - J^{\dagger}_{\{c\}}J_{\{c\}})z$$

This leads to the following expression for z.

$$z = (J(I - J^{\dagger}_{\{c\}}J_{\{c\}}))^{\dagger}(\Delta\vec{e} - JJ^{\dagger}_{\{c\}}\Delta d_{\{c\}})$$

When we use this z in the general solution of the primary goal, then we have found a solution which uses its freedom to solve the end effector as close as possible to the target. This means that we get the following solution to $\Delta\Phi$.

$$\begin{aligned}\Delta\Phi &= J^{\dagger}_{\{c\}}\Delta d_{\{c\}} + (I - J^{\dagger}_{\{c\}}J_{\{c\}})(J(I - J^{\dagger}_{\{c\}}J_{\{c\}}))^{\dagger}(\Delta\vec{e} - JJ^{\dagger}_{\{c\}}\Delta d_{\{c\}}) \\ &= J^{\dagger}_{\{c\}}\Delta d_{\{c\}} + (J(I - J^{\dagger}_{\{c\}}J_{\{c\}}))^{\dagger}(\Delta\vec{e} - JJ^{\dagger}_{\{c\}}\Delta d_{\{c\}})\end{aligned}$$

Here, the last equality holds since the projection operator $(I - J^{\dagger}_{\{c\}}J_{\{c\}})$ is both Hermitian and Idempotent. A proof of this equality can be found in [Maciejewski and Klein 1985, 116].
In the expression of $\Delta\Phi$ we can interpret $J(I - J^{\dagger}_{\{c\}}J_{\{c\}})$ as the freedom available to move the end effector without bringing a change to the depth values $d_{\{c\}}$ and we can interpret $(\Delta\vec{e} - JJ^{\dagger}_{\{c\}}\Delta d_{\{c\}})$ as the desired change in end effector position and orientation minus the change in end effector position and orientation caused by satisfying the primary goal of making the contacts free from the objects.

In algorithm 20 we can find the method which as primary goal makes the contacts free and as secondary goal brings the end effector closer to the target.

Algorithm 20 Collision response 3

Require: Φ^0, $\vec{g}$, $\vec{p}^{W}_{k,l}$, $\vec{m}_{k,l}$ and $d_{k,l}$ $\forall c_{k,l} \in \{c\}$, N, δ, M_d, M_i, t_i^R $\forall i \in (1, \ldots, m)$, $\hat{q}_0^W$ and $\hat{q}^R_{\text{end}}$

```
j=0
```
$\vec{e}_0 = f(\Phi^0)$. $\vec{g} = \text{Retarget}(\vec{g}, \vec{e}_0, M_d)$

for i=1,i=m **do**

$\hat{q}_i^R = (1 + \frac{1}{2}\vec{\mathbf{t}}_i^R)e^{\Phi^0}$

$\hat{q}_i^W = \hat{q}_{i-1}^W \hat{q}_i^R, \quad q_i^W = \mathbb{R}(\hat{q}_i^W), \quad \vec{t}_i^W = 2\mathbb{D}(\hat{q}_i^W)(q_i^W)^*$

end for

if $\{c\} \neq \emptyset$ **then**

$\vec{p}^{R}_{k,l} = (\hat{q}_k^W)^*(\vec{p}^{W}_{k,l} - \vec{t}_k^W)q_k^W, \quad \vec{p}^{\text{old}}_{k,l} = \vec{p}^{W}_{k,l} \quad \forall c_{k,l} \in \{c\}$

$d^0_{\{c\}} = d_{\{c\}}$

end if

while $j < N$ **do**

$J = J(\vec{e}_j, \Phi^j)$

$\Delta\vec{e}_j = \alpha(\vec{g} - \vec{e}_j)$

if $\{c\} \neq \emptyset$ **then**

$J_{\{c\}} = J(d^j_{\{c\}}, \Phi)$ by algorithm 15

if $d^j_{\{c\}} < 0$ **then**

$\Delta d_{\{c\}} = 0$

else

$\Delta d_{\{c\}} = \beta(d_{\{c\}} + \delta)$

end if

$\Delta\Phi^j = J^\dagger_{\{c\}}\Delta d^j_{\{c\}} + (J(I - J^\dagger_{\{c\}}J_{\{c\}}))^\dagger(\Delta\vec{e} - JJ^\dagger_{\{c\}}\Delta d_{\{c\}})$

else

$\Delta\Phi^j = J^T\Delta\vec{e}_j$

$\Delta\Phi^j = \text{ClampVelocities}(\Delta\Phi^j)$

end if

$\Phi^{j+1} = \Phi^j + \Delta\Phi^j$

for i=1,i=m **do**

$\vec{\Phi}_i^{j+1} = \log(CLAMPED(e^{\vec{\Phi}_i^{j+1}}))$

$\hat{q}_i^R = (1 + \frac{1}{2}\vec{t}_i^R)e^{\vec{\Phi}_i^{j+1}}$

$\hat{q}_i^W = \hat{q}_{i-1}^W \hat{q}_i^R$, $q_i^W = \mathbb{R}(\hat{q}_i^W)$, $\vec{t}_i^W = 2\mathbb{D}(\hat{q}_i^W)(q_i^W)^*$

end for

$\vec{e}_{j+1} = f(\Phi^{j+1})$

if $\{c\} \neq \emptyset$ **then**

$\vec{p}^{W}_{k,l} = \vec{t}_k^W + q_k^W \vec{p}^{R}_{k,l}(q_k^W)^* \quad \forall c_{k,l} \in \{c\}$

$d^{j+1}_{k,l} = d^0_{k,l} - (\vec{p}^{W}_{k,l} - \vec{p}^{\text{old}}_{k,l}) \cdot m_{k,l} \quad \forall c_{k,l} \in \{c\}$

end if

$j = j + 1$

end while

return Φ^j

When we look at the performance of this method we find that the motion is more stable than the

previous method. This since the method does also solve for the end effector position and orientation while making the contacts free from the objects. Also, in certain cases the chain can nicely surround an object which looks like a natural motion. Unfortunately, this method can not assure that we end with a contact free solution. Even though the contacts of the current chain can be made free, the new solution might contain contacts again. We can find an example in which this method goes wrong in figure 33.

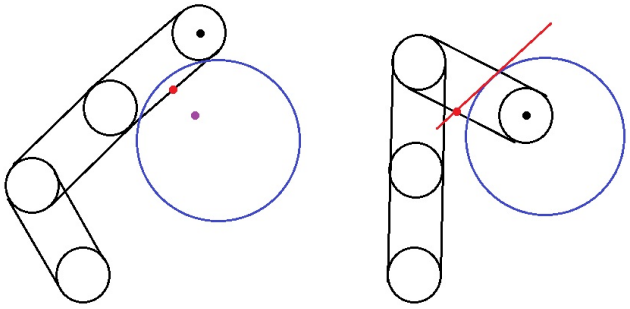


Figure 33

The increased freedom in order to obtain a solution which not only makes the contacts free also brings the end effector closer to the target enlarges the probability of obtaining new contacts in the solution. When there are many conflicting contacts in the same area or when a contact's depth is to high, then the method can not find a collision free solution. The only way to increase the performance of this method is by performing additional collision detection within the iterations of this method.

6.4 Conclusion

All the three methods which we developed all have their advantages and disadvantages. The first collision response method was stable and going from one solution to the other a contact depth could not increase due to the fact that the contact is maintained. The disadvantage of this method was that the virtual character can stick for a moment to the object. The second method was unstable since the solution oscillates when the virtual character touches the object. However, it makes all contacts free from the object when a collision free solution excists and the contact depths can not become to high. The last method gives the most natural motion to the virtual character and is quite stable. The disadvantage of this method is that the obtained solution might not be collision free which causes that the virtual character can still move through an object in certain cases.
We can conclude that it is not that easy to find a collision response method which is all natural, stable and returns a collision free solution.

7 The multilink mechanism for the virtual doctor

This chapter describes how the Razer Hydra motion controllers are used to control the virtual obstetrician in the BIRTH game. It gives an explanation on the mapping from the controllers to the hands of the virtual obstetrician in the game environment. Also, it explains the inverse kinematics mechanism behind this control where we can also let the virtual obsetrician grip an object.

7.1 The skeleton of the virtual doctor

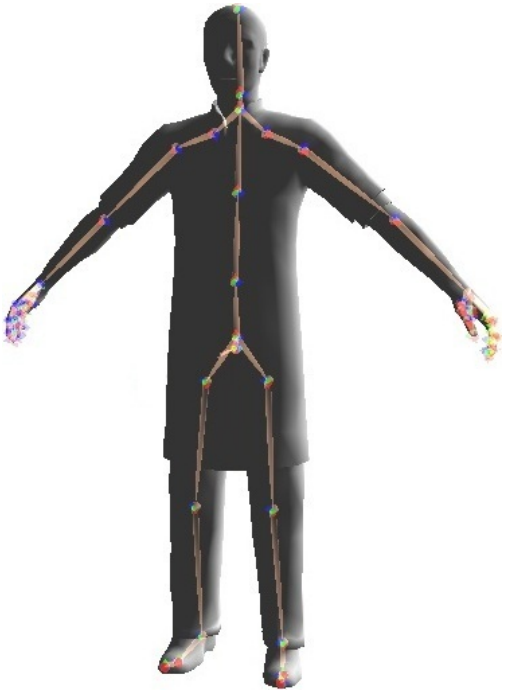


Figure 34: Skeleton of the virtual doctor

In chapter 3 we described how we use the skeleton structure to obtain a rigid body structure. For our virtual doctor this skeleton stucture consisting of joints and links is given in figure 34. Let $\mathcal{J}$ be the set of all joints in this skeleton stucture. All joints have a local coordinate sytem as we defined in chapter 4.5. From those local coordinate systems we can derive the dual quaternions $\hat{q}_i^R$ for all $i \in \mathcal{J}$. Here, $\hat{q}_i^R$ represents the relative orientation between the local coordinate system of joint i and the local coordinate system of the parent of joint i. To derive the dual quaternions $\hat{q}_i^W$ for all $i \in \mathcal{J}$ we first need to define our root joint which holds the world coordinate system. Instead of selecting one of the joints from $\mathcal{J}$ we place a virtual root joint in front of the virtual doctor and link this joint to the joint in the pelvis of the doctors skeleton. This gives freedom to the motion of the complete skeleton while the world coordinate system can remain fixed. We obtain the dual quaternion $\hat{q}_i^W$ for any $i \in \mathcal{J}$ by applying forward kinematics to the chain from the virtual root to joint i. We store the values of th dual quaternions $\hat{q}_i^W$ and $\hat{q}_i^R$ at a fixed location in the memory of the computer for all $i \in \mathcal{J}$. By this we can easily obtain the current configuration of the chain by reading their values from the memory or update a new configuration by saving the new dual quaternion values at the same memory location.

On the skeleton we have to define chains which we will manipulate in order to get motion. A chain $\mathcal{C} \subset \mathcal{J}$ is an ordered list of joints with the first joint in the list the base joint and the last joint in the list the end joint. To give motion to this chain it should have an end effector $\mathcal{C}$end which is linked to the endjoint of $\mathcal{C}$ and it should have a target $\mathcal{C}$target where the end effector should go to. We predetermine the chains for our skeleton together with their targets and end effectors. In the memory we store the chain $\mathcal{C}$ together with the joint where the base joint of the chain is rooted on. In addition we store the dual quaternions $\hat{q}_{\mathcal{C}\text{end}}^R$ as constant and the dual quaternion $\hat{q}_{\mathcal{C}\text{target}}^W$ as a variable in the memory. Here, $\hat{q}_{\mathcal{C}\text{end}}^R$ represents the orientation and position of end effector $\mathcal{C}$end relative to the endjoint of chain $\mathcal{C}$ and $\hat{q}_{\mathcal{C}\text{target}}^W$ represents the orientation and position of the target $\mathcal{C}$target for chain $\mathcal{C}$ relative to the world coordinate system.

Furthermore, for every joint we have an address in the memory where we store its contacts. The function CollisionDetection() performs a collision detection and updates the contacts in the memory for every joint.

For every chain $\mathcal{C}$ we define the function SolveIK which is dependent on the chain $\mathcal{C}$, the target dual quaternion $\hat{q}_{\mathcal{C}\text{target}}^W$ and the end effector dual quaternion $\hat{q}_{\mathcal{C}\text{end}}^R$.

$$\text{SolveIK}(\mathcal{C}, \hat{q}^W_{\mathcal{C}\text{target}}, \hat{q}^R_{\mathcal{C}\text{end}})$$

The function SolveIK is an inverse kinematics function. In case we do not want to consider collisions for chain $\mathcal{C}$, the function uses the Jacobian Transpose or Damped Least Square method from chapter 5 to solve the end effector $\mathcal{C}$end towards the target $\mathcal{C}$target. In case we do want to consider collisions for chain $\mathcal{C}$, the function uses the second or thirth collision response method from chapter 6. Those collision response methods solve as a primary goal the contacts of the chain out of the objects when there is a collision and as secondary goal solve the end effector $\mathcal{C}$end towards the target $\mathcal{C}$target. The function SolveIK can read and update variables from the memory. For example, the configuration of chain $\mathcal{C}$ in the inverse kinematics methods can be obtained from the dual quaternions $\hat{q}^R_i \quad \forall i \in \mathcal{C}$.

On this skeleton we have to define chains in order to give motion to it. Here it is important to first define what we want our virtual character to be able to do. In our case the virtual character is controlled with two controllers. Therefore we have to define a chain which gets influenced by the controllers. We also want the virtual person to be able to grip an object with his hands, therefore we also have to define how we do this and how we define a chain for the hand.

The next section explains how we can let a game player control the arms of the virtual doctor using a set of Razer Hydra controllers with the help of an inverse kinematics method. Section 7.3 explains how we can create an inverse kinematics based mechanism by which the game player can in addition let the virtual doctor grip an object.

7.2 Motion by controllers

The game player holds two 6-DOF controllers in his hands which give their positions and orientations corresponding to the basis station. The basis station is placed in front of the game player. With these controllers the game player has to give motion to the arms of the virtual doctor. Here, the motion of the arms of the game player should correspond as much as possible to the motion of the arms of the virtual doctor. The controller positions and orientations correspond to the position and orientation of the wrists of the game player. We map the position and orientation from each controller to the position and orientation of the wrists of the virtual character. An inverse kinematics method will determine the configuration of the arms of the virtual doctor. This means that the position and orientation obtained from the controllers determine the position and orientation of the wrists of the virtual doctor in the game environment.
To map the position and orientation of the controllers to a position and orientation in the world coordinate system in which the virtual doctor is standing, we first of all need to give a correspondence between the basis station and the world coordinate system of the virtual doctor. The world coordinate system is placed in front of the virtual doctor. Similarly, the basis station is placed in front of the game player. We can more precisely locate the world coordinate system at the position where the arms of the virtual doctor end when he stretches them. To avoid that the game player collides with the basis station when he wants to let the arms of the virtual character be stretched, we let the world coordinate system of the virtual character correspond with a location 10 cm in front of the basis station (towards the game player). Furthermore, we need to scale the distances obtained by the controllers to distances in the game environment. This scale corresponds to the scale that the virtual character has in the game when we consider an average human size. At last we map the orientations of the controllers to the orientations of the hands of the virtual character which should correspond to the orientations that the hands have when they hold the controllers.

We let the controllers only manipulate the configuration of the arms of the virtual character. It is also possible to let the controllers additionally manipulate the spine of the virtual doctor, but only the arms give a more natural motion when we consider that the virtual doctor only needs to perform actions in front of him with his arms and hands.
We define for each arm and controller one chain. Figure 35 shows the joints which are contained in the chains of the arms.

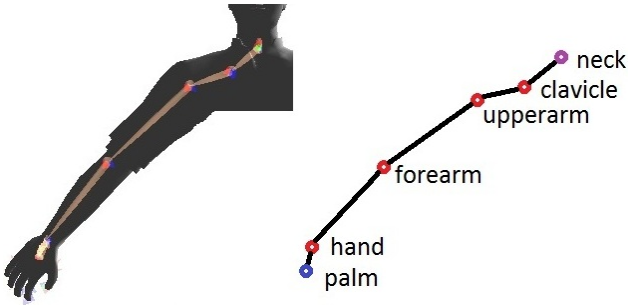


Figure 35: The chain of the arm

Each arm has a clavicle, upperarm, forearm and hand joint. We specify the joints for the left and right arm by adding the term left and right to them. Here, the clavicle joint is the base joint of the

chain linked to the neck joint and the hand joint is the end joint of the chain. The end effector is denoted by the palm joint and is linked to the hand join. We give the end effector the same position as the end joint, since they both correspond with the location of the wrist of the virtual character. This does not make the end effector and the end joint the same, since a change in configuration of the endjoint can still change the orientation of the hand with respect to the chain. We let LeftArm be the chain corresponding to the left arm and RightArm be the chain corresponding to the right arm of the virtual character.
We call the controller position and orientation in world coordinates the target position and orientation of the chain, since we want the end effector position and orientation to correspond to the position and orientation obtained from the controllers. We let LeftController be the target corresponding to the left controller and RightController be the target corresponding to the right controller.
This means that we have to solve the following functions to obtain a new configuration for the virtual character which solves the end effectors to their targets.

$$\text{SOLVEIK(LeftArm, } \hat{q}^W_{\text{LeftController}}\text{, } \hat{q}^R_{\text{LeftPalm}}\text{)}$$
$$\text{SOLVEIK(RightArm, } \hat{q}^W_{\text{RightController}}\text{, } \hat{q}^R_{\text{RightPalm}}\text{)}$$

Here, we can use as inverse kinematics method the Jacobian transpose or Damped Least Squares method, when we do not want to consider collisions of the arm. We can use the second collision response method from chapter 6 when obtaining a collision free solution is most important and we do not mind the arm of the virtual doctor to shiver slightly when he touches an object. Alternatively we can use the thirth collision response method from chapter 6 when we find more importance in the stability of the motion but do not give the hard constrain of obtaining a collision free solution. Furthermore, we let the inverse kinematics method solve the end joints of the chains of the arms for the end effector position and orientation, but the other joints only for the end effector position. This creates a more natural motion when we only let the end joint solve for the orientation obtained from the controller. It makes the orientation of the wrist of the virtual doctor correspond to the orientation obtained from the controller which is the orientation of the wrist of the game player.

7.3 Gripping

We have now found a way to let the game player control the motion of the arms of the virtual character. Aside from this, the game player has to be able to grip an object once he has moved the hands of the virtual doctor near the object. With gripping we mean that the fingers of the virtual doctor have to get placed on the object. We also want the hands of the virtual doctor to be able to move along objects without gripping them. For this reason, the game player has to pull the trigger of the controller to verify that he attempts to grip an object. The game player can let the virtual character grip with the right hand when he pulls the right controller trigger and/or with the left hand when he pulls the left controller trigger.

The objects have predefined grips. We define $\mathcal{G}$ as the set of grips of all objects. A grip $G \in \mathcal{G}$ consists of a target position for the palm joint represented by $\hat{q}^W_{PMG}$ and target positions for the fingertips of the hand represented by $\hat{q}^W_{TG}$, $\hat{q}^W_{IFG}$, $\hat{q}^W_{MFG}$, $\hat{q}^W_{RFG}$ and $\hat{q}^W_{PG}$ which correspond to the thumb, index finger, middle finger, ring finger and pinky respectively. We specify the grips for the left hand by $\mathcal{G}^L$ and the grips for the right hand by $\mathcal{G}^R$.

Our goal is now to find a configuration for the virtual doctor, such that the palm joint of the virtual doctor corresponds with the target palm of grip G and the fingertips of the virtual doctor correspond with the target fingertips of G.

When the game player pulls the trigger, we want to obtain one of the grips of which the target position for the palm lays within a given distance τ_{Lin} from the palm joint of the virtual character. When there is no grip of which the target position for the palm lays within a distance τ_{Lin} from the palm joint of the virtual character, we say that the attempt to grip failed. Otherwise, we want to obtain the closest grip within the grips of whose target position for the palm joint lays within a distance τ_{Lin} from the palm joint of the virtual character. We define the closest grip as the grip whose orientation of the target palm is closest to the orientation of the palm joint of the virtual character. We define the distance between two orientations as the smallest rotation angle that we need to rotate the orientations to each other.

Algorithm 21 determines the closest grip out of the set of all grips in the game environment when the game player pulls the controller trigger. The algorithm returns $\emptyset$ when the attempt to grip failed.

The moment the game player releases the controller trigger is the moment we stop solving the hand of the virtual doctor towards a grip when there is a closest grip. When the game player succeeds to grip an object, then he should keep the trigger pulled till he wants to release the object again.

When we have found a closest grip G, then we solve for the chain of the arm the end effector which is the palm joint of the virtual doctor to the target for the palm of G instead of to the target obtained from the controller. This leads to one of the following functions.

$$\text{SOLVEIK(LeftArm, } \hat{q}^W_{\text{PMG}^L}\text{, } \hat{q}^R_{\text{LeftPalm}})$$
$$\text{SOLVEIK(RightArm, } \hat{q}^W_{\text{PMG}^R}\text{, } \hat{q}^R_{\text{RightPalm}})$$

Algorithm 21 Compute ClosestGrip($\mathcal{G}, \hat{q}^W_{\text{palm}}$)

Require: $\mathcal{G} := \{G_1, \ldots, G_z\}$, $\hat{q}^W_{PMG_i} \quad \forall i \in (1, \ldots, z)$, $\hat{q}^W_{\text{palm}}$, τ_{Lin} and τ_{Ang}.
Ensure: GRIP=ClosestGrip($\mathcal{G}$,$\hat{q}^W_{\text{palm}}$,τ_{Lin}, τ_{Ang})
  $q^W_{\text{palm}} = \mathbb{R}(\hat{q}^W_{\text{palm}})$ $\vec{\mathbf{t}}^W_{\text{palm}} = 2\mathbb{D}(\hat{q}^W_{\text{palm}})(q^W_{\text{palm}})^*$
  for i=1,i=z **do**
    $q^W_{PMG_i} = \mathbb{R}(\hat{q}^W_{PMG_i})$ $\vec{\mathbf{t}}^W_{PMG_i} = 2\mathbb{D}(\hat{q}^W_{PMG_i})(q^W_{PMG_i})^*$
    if $q^W_{\text{palm}} \cdot q^W_{PMG_i} < 0$ **then**
      $q^W_{PMG_i} = -q^W_{PMG_i}$
    end if
    $d_{\text{Ang}} = \arccos((q^W_{\text{palm}})^* q^W_{PMG_i})$
    $d_{\text{Lin}} = ||\vec{t}^W_{\text{palm}} - \vec{t}^W_{PMG_i}||$
    if $d_{\text{Ang}} < \tau_{\text{Ang}}$ **and** $d_{\text{Lin}} < \tau_{\text{Lin}}$ **then**
      GRIP=G_i
      $\tau_{\text{Ang}} = d_{\text{Ang}}$
    end if
  end for
  return GRIP

Here, we can use the same inverse kinematics method as when we solved to the controller position and orientation. When an attempt to grip failed or the game player has not pulled the trigger, we solve the end effector of the arm to the target obtained from the controller.

Furthermore, we need to solve the hand such that the fingertips of the virtual doctor get placed on the targets for the fingertips of the closest grip G. To do this we first need to define the chains in the hands of the virtual character. We illustrate those chains in the next figure.

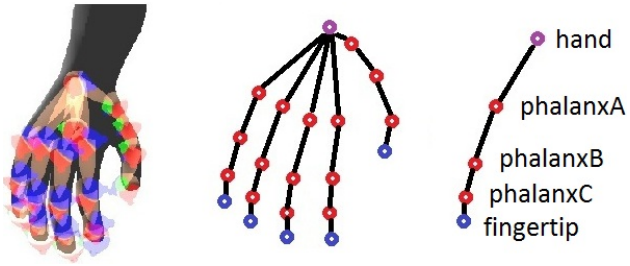


Figure 36: The chain of the hand

Each finger of the virtual character is represented with a chain consisting of three joints called phalanxA, phalanxB and phalanxC. The fingertip is the end effector linked to the end joint which is phalanxC. PhalanxA is the base joint and is linked to the hand joint of the virtual character. The five chains of the fingers are called Thumb,IndexFinger, MiddleFinger, RingFinger and Pinky. The dual quaternions which represent the end effector positions of the fingers are denoted by $\hat{q}^R_{\text{Tend}}$, $\hat{q}^R_{\text{IFend}}$, $\hat{q}^R_{\text{MFend}}$, $\hat{q}^R_{\text{RFend}}$ and $\hat{q}^R_{\text{Pend}}$ for the thumb, index finger, middle finger, ring finger and pinky respectively.

When there is a grip G we solve for each finger of the hand the end effector to the the corresponding target fingertip from G. To keep the notation short we define the following functions.

$$\text{SolveIK(LeftHand,}G^L) = \begin{cases} \text{SolveIK(LeftThumb, } \hat{q}^W_{TG^L}, \hat{q}^R_{LeftTend}) \\ \text{SolveIK(LeftIndexFinger, } \hat{q}^W_{IFG^L}, \hat{q}_{LeftIFend})^R \\ \text{SolveIK(LeftMiddleFinger, } \hat{q}^W_{MFG^L}, \hat{q}^R_{LeftMFend}) \\ \text{SolveIK(LeftRingFinger, } \hat{q}^W_{RFG^L}, \hat{q}^R_{LeftRFend}) \\ \text{SolveIK(LeftPinky, } \hat{q}^W_{PG^L}, \hat{q}^R_{LeftPend}) \end{cases}$$

$$\text{SolveIK(RightHand,}G^R) = \begin{cases} \text{SolveIK(RightThumb, } \hat{q}^W_{TG^R}, \hat{q}^R_{RightTend}) \\ \text{SolveIK(RightIndexFinger, } \hat{q}^W_{IFG^R}, \hat{q}_{RightIFend})^R \\ \text{SolveIK(RightMiddleFinger, } \hat{q}^W_{MFG^R}, \hat{q}^R_{RightMFend}) \\ \text{SolveIK(RightRingFinger, } \hat{q}^W_{RFG^R}, \hat{q}^R_{RightRFend}) \\ \text{SolveIK(RightPinky, } \hat{q}^W_{PG^R}, \hat{q}^R_{RightPend}) \end{cases}$$

As an inverse kinematics method for the SolveIK function for the hand the Cyclic Coordinate Descent method is used. This method gives a natural motion for the grip motion of a hand and is computationally fast. Moreover, it is not necessary to encounter collisions for the fingers. The game player has already maneuvered the hand of the virtual doctor near the object in a collision free configuration before we solve for the fingers. The joints in the fingers are constrained in such a way that they can not collide with each other when we solve them successively.

We start with solving the hand as soon as the palm joint of the virtual character is on a distance $\tau_d < \tau_{Lin}$ from the palm target of the closest grip. From this moment on we iteratively solve the arm followed by the hand towards their targets. If we would start solving the fingers, while not yet being close enough to the target palm, the fingers might be solved to their finger targets, after which the palm gets solved and pushes the fingers through the object.

7.4 The inverse kinematics mechanism

It is important to be careful with the order in which we solve for the arms and hands. Also, it is important that we use the right method on the right time. Before we give an overview of the inverse kinematics mechanism for the control of the virtual doctor with the controllers as we described in the previous two sections, we first give some additional functions.

The function ReadTriggers() sets the variable RightTrigger on "ON" when the game player has pulled the right trigger and "OFF" otherwise and sets the variable LeftTrigger on "ON" when the game player has pulled the left trigger and "OFF" otherwise.

The function ReadControllers() reads the new controller positions and orientations and sets the value for the dual quaternions $\hat{q}^W_{LeftController}$ and $\hat{q}^W_{RightController}$.

The function DrawVirtualDoctor() visualizes the virtual doctor in the game environment for a new configuration obtained by the inverse kinematics methods. When the rate with which we render the solutions is constant, we can visualize the virtual doctor straight after a new configuration is obtained. Otherwise, we can let the function DrawVirtualDoctor() place a new configuration into a queue which ensures that the virtual doctor is visualized at a constant rate.

In figure 37 the scheme which gives the overview of the inverse kinematics mechanism is given. It can be seen that when the game player does not pull the trigger of the controller, then the mechanisms solves for the Arm the end effector to the target given by the controllers. When the game player pulls the trigger, but he is not close enough to one of the grips, the system still solves the Arm to the target given by the controllers and does not change the configuration of the hand. When the game player pulls the trigger and he is close enough to one of the grips, then the system will solve the Arm to the target palm obtained from the closest grip. When the palm of the hand is almost on its grip position, then the system will also solve the fingertips of the hand to their targets of the grip.

An example of a matched grip for the virtual doctor on the baby can be found in figure 38 in the appendix.

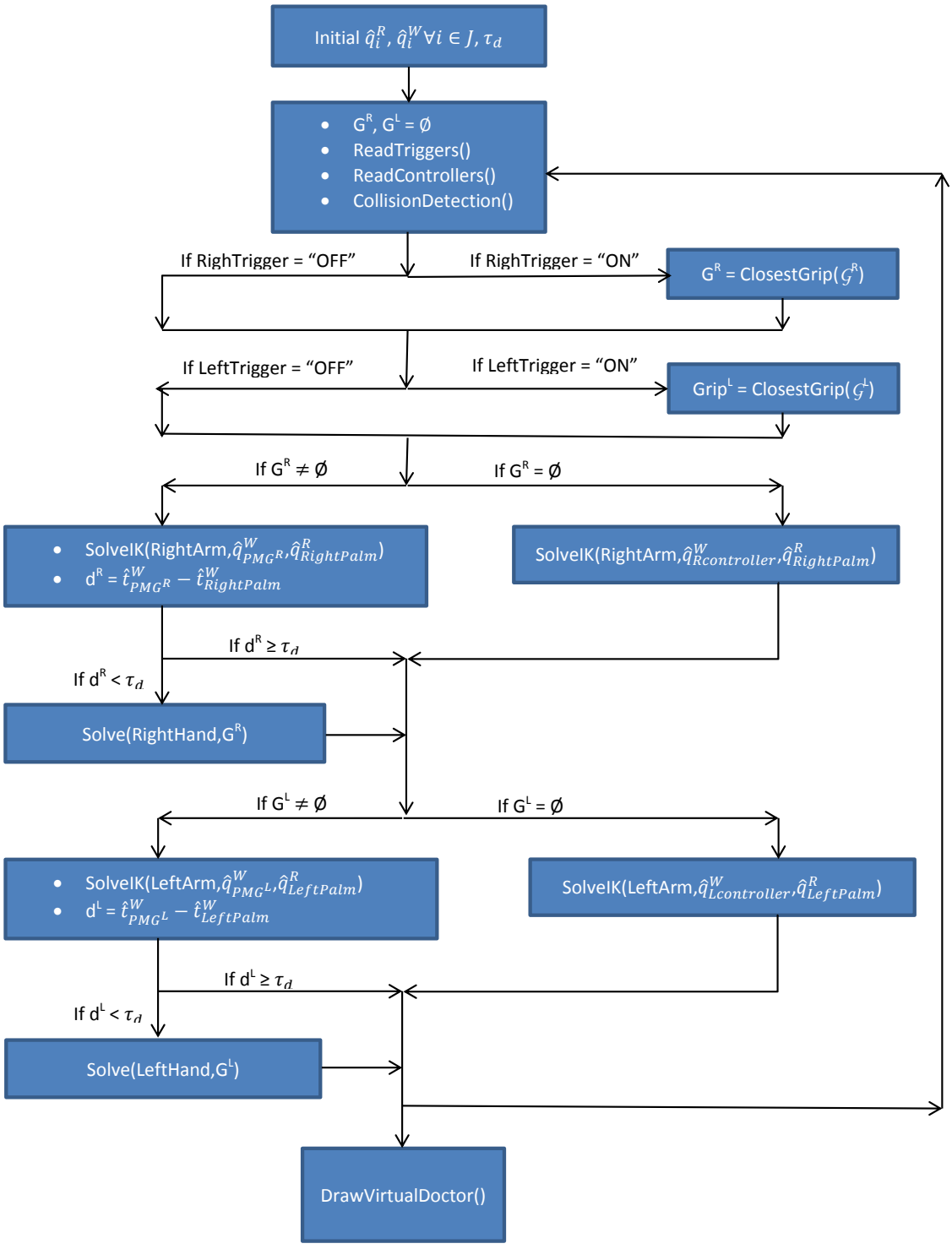


Figure 37: The inverse kinematics mechanism

8 Conclusions

Inverse kinematics techniques are used to control the virtual obstetrician in the BIRTH game with a set of Hydra Razer controllers. This report gave new quaternion based Jacobian methods. This Jacobian methods use a Jacobian matrix which represents the change in end effector position and orientation given the change in the log values of the relative quaternions of the joints. This led to successful methods which performs well. Moreover, the quaternions do not suffer from singularities in the representation in the range we work in. Furthermore, in the log space of the quaternions there is no dependency between the parameters. The only singularities which can still arise are in the Pseudo Jacobian method when we take the pseudo inverse of the Jacobian matrix. Aside from this fact, the Jacobian methods are efficient and give a natural and stable motion. Furthermore, the properties of the quaternions could be used to constrain the rotational freedom of the joints. The addition of these constraints only improved the methods by the increased naturalness of the solutions.

The collision response methods which are proposed in this report are an extension to this new inverse kinematics method. When there is a collision of the virtual character with an object, the collision methods give an additional contact Jacobian matrix which represents the change in depth value inside the object given a change in configuration. The collision response methods have as primary goal to make a solution collision free and as secondary goal to bring the end effector closer to a given target. The collision methods have promising results, but it remains a challenge to find a collision response method for the real time motion control of a virtual character which can guarantee all the following: a collision free solution, a stable solution and a natural solution.

Finally, it is shown how these inverse kinematics techniques are used to control the virtual doctor with the Razer Hydra controllers within the BIRTH game. The controllers can be used to give motion to the arms of a virtual doctor, but also they suffice to let the virtual doctor grip an object with his hands. This makes the Razer Hydra controllers a useful device for the real time motion control of a virtual character.

9 Recommendations for future research

There are many challenges in finding an inverse kinematics method for the real time motion control of a virtual character which can deal with constraints and contacts. We have found some promising methods in this report. These methods can still be extended and improved.

First of all, the inverse kinematics method described in this report was used for the control of a chain which only has rotational freedom for the joints. The method can be extended to the control of a chain which has aside from rotational freedom also translational freedom for the joints. We defined the parameters which determine the configuration of the chain by the log values of the quaternions which represent the relative orientations of the joints. We can extend this by defining the configuration of the chain by the log values of the dual quaternions which represent the relative orientation and translation of the joints. This also needs the extension of the Jacobian method.

Furthermore, the contacts in the collision response methods all have equal weights. A system which prioritizes certain contacts above others might reduce the problems which arise when contacts are conflicting.

Also, the amplitude of oscillation in the second collision response method described in this report appears to decrease when we reduce the step sizes between one configuration and the other. An experiment on this correspondence can be carried out, to find out how small those step sizes should be in order to minimize the oscillation. This might lead to a more natural method.

In addition, we can experiment whether the frequency of collision detection within the iterations of the collision response methods will benefit the performance of those methods.

Moreover, the research can be extended to be able to deal with the inequaltities that arise from using contacts and joint limits. The joint limits were simply clamped by which contacts might still stick in the proposed collision response methods. The joint limits could be handled in the same way as the contacts.

Finally, it has to be considered whether adding contact constraint to the motion control of the virtual obstetrician in the BIRTH game improves the skill training of the game player or that it only creates a barrier for the game player. Positioning the hands of the virtual obstetrician near the correct grip on the baby might be harder when there are contact constraints than without.

10 References

Inverse Kinematics

Aristidou, A., Lasenby, J., 2009, Inverse kinematics: a review of existing techniques and introduction of a new faster iterative solver, *CUED/F-INFENG/TR-632*, 1-74.

Baerlocher, P., 2001, Inverse kinematics techniques of the interactive posture control of articulated figures, *Ecole Polytechnique Federale de Lausanne*, Switzerland, PhD thesis, 1-156.

Buss, S.R., 2009, Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods, *University of California, San Diego* unpublished survey article, 1-19.

Engell-Norregard, M.P., Niebe, S.M., Bonding, M.B., 2007, Inverse kinematics with constraints, *University of Copenhagen*, Denmark, student project, 1-58.

Engell-Norregard, M.P., 2007, Inverse Kinematics The state of art, *University of Copenhagen*, Denmark, graduate project, 1-23.

Welman, C., 1993, Inverse kinematics and geometric constraints for articulated figure manipulation, *Simon Fraser University*, Canada, master thesis, 1-77.

Zhang, L., Brunnett, G., Rusdorf, F., 2011, Real-time Human Motion Capture with Simple Marker Sets and Monocular Video, *Journal of Virtual Reality and Broadcasting* 8, no. 1, 1-14.

Quaternions

Dam, E.B., Koch, M. and Lillholm, M., 1998, Quaternions, Interpolation and Animation, *University of Copenhagen*, Denmark, Technical report, 1-98.

Kavan, L., Collins, S., ra, J., O'Sullivan, C., 2008, Geometric skinning with approximate dual quaternion blending, *ACM Trans. Graph.* 27, No. 4, 1-23.

Kavan, L., Collins, S., ra, J., O'Sullivan, C., 2007, Skinning with dual quaternions, *ACM Press* 1, no. 212, 1-39.

Kavan, L., Collins, S., ra, J., O'Sullivan, C., 2006, Dual quaternions for rigid transformation blending, *TCD-CD-2006-46*, 1-10.

Pennestri, E., Valentini, P. P., 2009, Dual quaternions as a tool for rigid body motion analysis: a tuturial with an application to biomechanics, *Archive of Mechanical Engineering* 57, july, Versita, 1-17.

Perez, A. M. and McCarthy, J.M, 2004, Dual quaternion synthesis of constrained robotic systems, *ASME Journal of Mechanical Design* 126(3), 425-435.

Pham, H.L., Perdereau V., Adorno, B.V. and Fraisse, P., 2001, Position and orientation control of robot manipulators using dual quaternion feedback, *IEEE IROS 2001*, Tapei (Taiwan), 18-22.

Grassia, F.S., 1998, Practical parameterization of rotations using the exponential map, *The Journal of Graphics Tools* 3(3), 1-13.

Collision

Bergen van den, G., 1999, A fast and robust GJK implementation for collision detection of convex objects, *Journal of Graphics Tools*, 4(2), 7-25.

Bertram, D., Kuffner, J., Dillmann, R., Asfour, T., 2006, An integrated approach to inverse kinematics and path planning for redundant manipulators, Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'06), 1874-1879.

Ding, H., Chan, S. P., 1996, A real-time planning algorithm for obstacle avoidance of redundant robots, *Journal of Intelligent and Robotic systems* 16, 229-243.

Rahmanian-Shahri, N., Troch, I., 1996, Collision-avoidance for redundant robots through control of the self-motion of the manipulator, *Journal of Intelligent and Robotic systems* 16, 123-149.

Fratu, A.,Breth, J.F. , Fratu, M., 2010, Redundant inverse kinematics system for obstacle avoidance, *RECENT* Vol. 11, no. 1(28), 29-32.

Maciejewski, A.A., Klein, C.A., 1985, Collision avoidance for kinematically redundant manipulators in dynamically varying environments, *The International Journal of Robotics Research* 4, 109-116.

Choi, S.I., Kim, B.K., 2000, Obstacle avoidance for redundant manipulators using directional-collidability/temporal-collidability measure, Journal of Intelligent and Robotic Systems 28, 213-229.

Kallmann, M., 2008, Analytical inverse kinematics with body posture control, *Computer Animation and Virtual Worlds* 19, 79-91.

Figures

[1](a)(c) http://www.razerzone.com/eu-en/gaming-controllers/razer-hydra-portal-2-bundle (Accessed: 17 juli 2012)
(b) http://www.maximumpc.com/article/reviews/razer_hydra_review/ (Accessed: 17 juli 2012)

[2] http://around-the-corner.typepad.com/ (Accessed: 15 august 2012)

[3] [Zhang *et al.*, 2010]

[4] http://www.mathworks.nlhelp/toolbox/physmod/mech/ref/spherical.html (Accessed: 20 juli 2012)

[5] http://en.wikipedia.org/wiki/Right-hand_rule (Accessed: 12 june 2012)

[6] [Pennestri and Valentini, 2009]

[7] [Kavan *et al.*, 2008]

Appendix: Screenshots of the BIRTH game

Figure 38: Screenshot of a grip on the baby

Figure 39: Screenshots of the Jacobian Transpose method