Computers & Graphics 35 (2011) 288–298

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

A projected back-tracking line-search for constrained interactive inverse kinematics ☆

Morten Engell-Nørregård *, Kenny Erleben

University of Copenhagen, Universitetsparken 1, 2100, KBH Ø, Denmark

ARTICLE INFO

Article history:
Received 25 March 2010
Received in revised form
15 December 2010
Accepted 17 December 2010
Available online 4 January 2011

Keywords:
Projected line-search
Joint limits
Constraints
Non-linear optimization
Inverse kinematics

ABSTRACT

Inverse kinematics is the problem of manipulating the pose of an articulated figure in order to achieve a desired goal disregarding inertia and forces. One can approach the problem as a non-linear optimization problem or as non-linear equation solving. The former approach is superior in its generality and ability to generate realistic poses, whereas the latter approach is recognized for its low iteration cost. Therefore, many prefer equation solving over optimization for interactive applications. In this paper we present a projected-gradient method for solving an inverse kinematics problem interactively, which exhibit good performance and precision. The method is compared to existing work in terms of visual quality and accuracy. Our method shows good convergence properties and deals with joint constraints in a simple and elegant manner. Our main contribution lies in an explicit incorporation of joint limits in an interactive solver. This makes it possible to compute the pose in each frame without the discontinuities exhibited by existing key frame animation techniques.

1. Introduction

Inverse kinematics is used for a wide range of applications such as robot simulation or motion planning, creation of digital content for movies or commercials, or for synthesizing motion in an interactive applications such as computer games and other types of virtual worlds. In Fig. 1, we have illustrated animations created with our own interactive inverse kinematics method.

Inverse kinematics is a standard tool in many applications like Maya, 3D Studio Max [1] or Blender [2]. Recently inverse kinematics have been employed as a dimensionality reduction tool in a tracker of human motion [11]. In short, it is a well known and wide-spread technique, and better numerical methods will, therefore, be valuable to a large community. A generally applicable method should be easy to use and thus minimize the number and complexity of user-defined parameters, while giving as realistic a pose as possible. Furthermore, speed is essential whether the method is used interactively to pose a figure, or to simulate movement in a virtual world. Most applications have chosen one of two avenues. Either, they are specialized closed-form solutions for specific low-dimensional manipulators, the approach often taken in Robotics, or they are general type methods. Even though inverse kinematics has been around for quite some time, there seems to be very little work done in exploring methods which can bridge the gap between the two extremes, perhaps because the animation industry has not felt the need for further improving their methods, and the design of robots have followed the same line of thought. However, with the development of more and more humanoid robots and the move towards more physics-based animations in media, the need for interactive general purpose inverse kinematics methods is again topical. Our focus is on the underlying method of solving the inverse kinematics problem. This can be extended to handle more user control, but that is not our interest.

2. The problem

We focus on a general, interactive method which includes joint limits. To state the problem more formally: Given a serial mechanism, we can set up a coordinate transformation from one joint frame to the next. Thus, we can find one transformation that takes a point specified in the frame of the end-effector into the root frame of the mechanism. We write it in general as

$$y = F(\theta). \tag{1}$$

We can change the values of the joint-parameter θ and gain explicit control over the position and orientation of the end-effector, y. This is commonly known as forward kinematics. Given a desired goal position, g, one seeks the value of θ such that

$$\theta = F^{-1}(g). \tag{2}$$

☆This article was recommended for publication by O. Staadt.

* Corresponding author.
E-mail addresses: mort@diku.dk (M. Engell-Nørregård), kenny@diku.dk (K. Erleben).

0097-8493/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.12.011

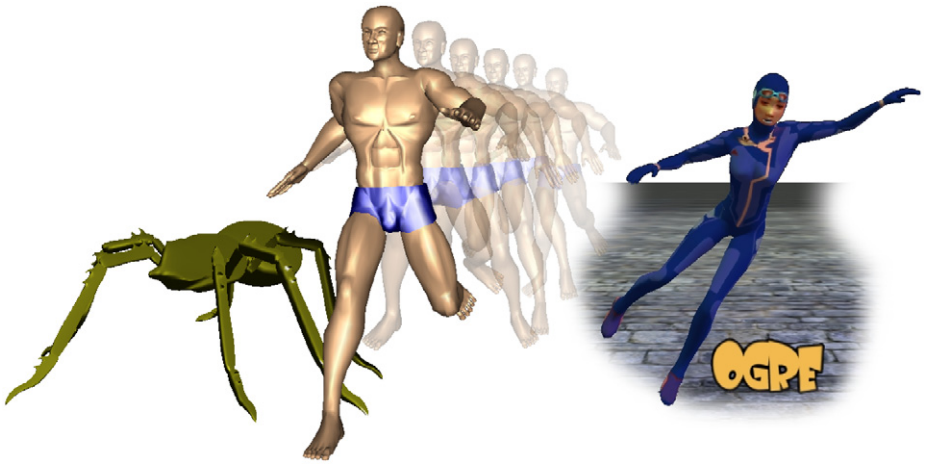

Fig. 1. Inverse kinematics solved figures, the method is general and poses any conceivable figure. The developed method was integrated in an Ogre demo.

This is known as inverse kinematics and it is the problem we address in this paper.

2.1. Related work

Inverse kinematics was introduced in robotics [7] and to the graphics community early in [10]. In robotics, the problem is usually phrased as a dynamic system which may lead to different schemes [14]. An overview of numerical methods used in computer graphics can be found in Buss [3].

Inverse kinematics methods. In Zhao and Badler [34] a new mathematical formalism is presented for solving inverse kinematics as a constrained non-linear optimization problem. This work allows for general types of constraints. Inverse kinematics is known to suffer from problems with redundancies and singularities [17]. In robotics, redundancy problems have been addressed by adding more constraints [6]. In animation the redundancy is most often handled by using the spatial temporal coherency of consecutive solutions. Thus, the correct solution closest to the previous solution is chosen. This is also true in our system (see, e.g. Fig. 2).

In Fêdor [9], three methods are revisited and evaluated for computer game usage: an algebraic method, cyclic-coordinate-descent [32], and a Newton–Raphson method inspired by Zhao and Badler [34]. The Newton-based method is used for complex manipulation and claimed to give the most realistic looking poses, but it is the slowest. However, joint limits were not dealt with in this work. Other formulations have been investigated for instance in Ho et al. [13] the problem is solved using linear programming. In Sumner et al. [31] a mesh-based inverse kinematics method was presented. This relies on example poses to manipulate the mesh directly and does not handle joint limits. Furthermore, the mesh-based approach runs with only one frame per second for even moderately sized meshes.

In robotics closed-form methods are popular [22]. Closed-form methods often result in an algebraic systems that can be solved very fast and reliable, but these methods are highly specialized for a specific low-dimensional manipulator (up to seven degrees of freedom).

In Computer Graphics which is our main focus in this paper, even a single arm will usually have more than seven degrees of freedom. In a simplified male human skeleton from Figs. 1 and 3 this is 11. The shoulder has five degrees of freedom (two in the collar bone, three in the shoulder joint) one in the elbow, three in the wrist and two in the hand. Thus, general purpose methods capable of dealing with many degrees of freedom are desired.

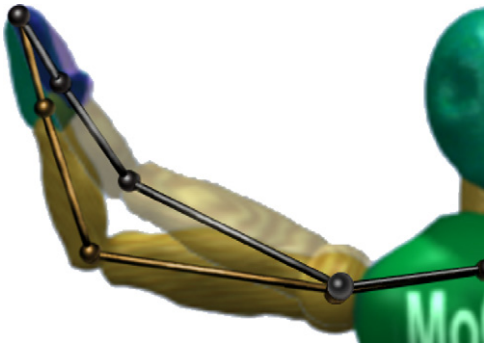

Fig. 2. An example of a mathematically correct solution which deviates from the reference due to redundancy. The Ghosted overlay with the gray skeleton is the projected line-search inverse kinematics solution. Notice the large difference on the wrist and elbow and the small but noticeable difference on the shoulder joint. A positional goal was used in this example.

In Safonova et al. [29], inverse kinematics is combined with other techniques and a sequential quadratic programming problem is solved. The running times are in minutes which prohibits interactive usage. Motion synthesis using space-time optimization and machine learning has also been tried [8,16] although running times are not yet within the grasp of the real-time domain. An example of a widely used general method is the Blender Software [2] which uses a Jacobian Inverse scheme with the pseudo-inverse being computed using SVD. To sum up, there is still a need for fast general purpose methods for posing characters with direct manipulation.

Joint limits. Often constraints such as joint limits are omitted [9] or dealt with as a post-processing step [32]. The added value of handling joint limits is shown clearly in Fig. 5. Several approaches for handling joint limits has been proposed. Joint sinus curves was used in Maurel and Thalmann [19] to describe the boundaries of the feasible motion space of the joints. If a joint exceeds its boundary then it is projected back to the boundary and kept fixed at the boundary until the goal position is changed. In Herda et al. [12], quaternion boundary fields were created, and a bisection algorithm was used to back-project infeasible joint positions onto the closest point on the quaternion boundary field. In Meredith and Maddock [20] a back-projection is used after the joint-parameters have been updated. In Wilhelms and Gelder

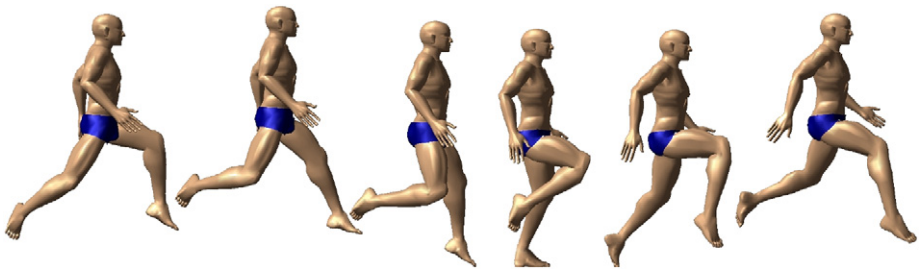

Fig. 3. A running animation made using the projected line-search optimization approach. The presented method supports interactive editing of animated characters.

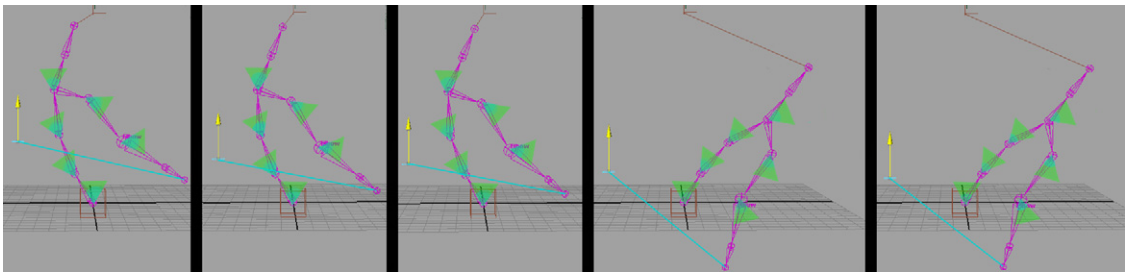

Fig. 4. The inverse kinematics system in Maya exhibiting a typical lack of smoothness in the motion. The skeleton has 16 degrees of freedom in total and two positional end effector goals. Joint limits are shown as cones. One goal (shown by vertical arrow) was moved vertically to force the system to move along the joint limits. The flip is clearly seen from the third to the fourth picture.

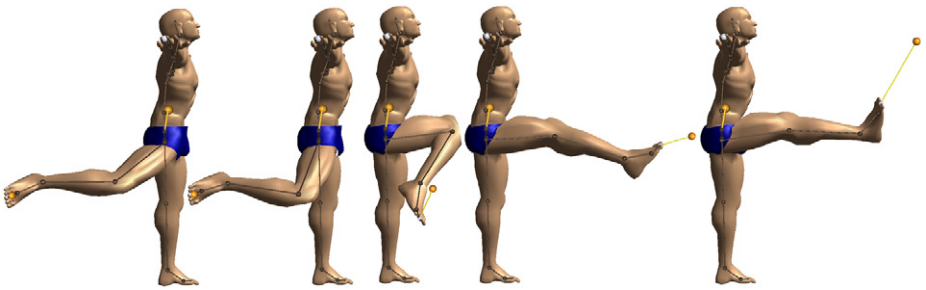

Fig. 5. Joint limits shown by posing a leg in its extreme positions using positional goals. The goal position of the end-effector is shown as a small ball. Notice in pose 3 that the goal would be reachable if no joint limits were present, and that both poses 4 and 5 would be different without joint limits.

[33], joint reach cones are introduced and later refined in Shao and Ng-Thow-Hing [30] to handle moving center of rotations. Here back-projection of infeasible joints is also used. In Zhao and Badler [34] they keep track of currently active joint limits and modify their scheme in such a way that joint limits will not be violated. An example of a method currently used in Commodity software, is the Jacobian Inverse method used in the Blender software, which uses a projection to move the solution unto a feasible region. This projection is performed as a separate step after an unconstrained solution has been found. It has not been possible to get information regarding the methods used in other major 3D systems, but their performance and quality are comparable. Thus, Blender has been chosen as an example of a 3D system in this work.

2.2. Contribution

In previous work, limits are dealt with as a post-processing step that simply back-projects infeasible iterates to a feasible iterate or an active set approach is used. These approaches either disregard a sufficient improvement in the solution or result in computationally expensive book-keeping. Our major contribution is a line-search method that guarantees an improvement of the solution, and robust handling of joint limits.

Our experience and the literature seem to indicate that constrained non-linear optimization is not the favorite choice for interactive applications. This is a shame since such a formalism offers more realistic motion and general constraints. This has motivated our work. We believe the task lies in creating a simple and elegant numerical method well suited for the purpose of inverse kinematics and which is easily implemented by programmers in industry.

In this paper, we will present and evaluate a numerical approach for solving the interactive inverse kinematics problem as a constrained non-linear optimization problem. We will demonstrate how our numerical approach can be used interactively and can deal with joint limits. Our method does not exhibit the flipping behavior of methods which solve the unconstrained inverse kinematics problem before projecting the solution unto the feasible set. It avoids this without having to resort to such pre- or post-processing as, e.g. the preferred pose of Maya. In Fig. 4, an example of the unwanted flipping behavior exhibited in Maya for a constrained solution with two positional goals and joint limits are shown.

Our focus has not been on developing a finished interactive animation suite but rather to present a method which can be used by others in their systems. We have done this by making all code available in the OpenTissue library [25]. From this the implementation details can be seen and the method may be included in any project.

Explicit comparison is performed with the inverse kinematics solver found in the Blender software package, and an example of the unwanted behavior of Blender has been chosen because it is fully comparable in functionality with the major 3D animation softwares Maya and 3D Max. Furthermore, Blender is open-source. Thus, we could compare the timings of the IK-solvers directly without, e.g. render-time disturbing the measurements.

2.3. Overview of the document

In Section 2, we introduce inverse kinematics as an equation solving problem and debate how popular methods are related. Hereafter, we present the non-linear optimization approach in Section 3 and give details of our method in Section 4. Next, we present our results in Section 5 and conclude in Section 6. In Appendix A, we give details on how to compute the Jacobian.

3. The traditional approach

Initially, we know the value of the joint-parameters, θ^k, and a desired goal state for the end-effector, g. The corresponding initial state of the end-effector is given by

$$y^k = F(\theta^k). \tag{3}$$

Writing

$$\theta = \theta^k + \Delta\theta^k, \tag{4}$$

where $\Delta\theta^k$ is the change in joint parameter values. Our task is now to compute $\Delta\theta^k$ such that

$$g = F(\theta^k + \Delta\theta^k). \tag{5}$$

We perform a Taylor series expansion of the right-hand side

$$g = F(\theta^k) + \frac{\partial F(\theta^k)}{\partial\theta}\Delta\theta^k + O(\|\Delta\theta^k\|^2). \tag{6}$$

We introduce the notation

$$J(\theta^k) = \frac{\partial F(\theta^k)}{\partial\theta}, \tag{7}$$

and call the matrix J the Jacobian. Next, we ignore the remainder term of the Taylor series expansion, to obtain the approximation

$$g \approx F(\theta^k) + J(\theta^k)\Delta\theta^k. \tag{8}$$

Recall that $y = F(\theta^k)$ and for the moment assume that $J(\theta^k)$ is invertible. Then we can isolate the unknowns of our problem

$$\Delta\theta^k = J(\theta^k)^{-1}(g - y^k). \tag{9}$$

This is a linear model for taking us as close to g as possible with a linear step. Thus, we may not get to g in one step. To solve this we will keep on taking more steps until we get sufficiently close. That is, we compute $\theta^{k+1} = \theta^k + \Delta\theta^k$ and repeat the above steps with k replaced by $k+1$. This results in a non-linear Newton method. The important thing to notice is that the method only needs to know how to evaluate the function-value, $F(\theta)$, and the Jacobian $J(\theta)$.

From Nocedal and Wright [23] we know that if F is continuously differentiable and the Newton sub-system is solved with sufficient accuracy then the non-linear Newton method will have quadratic convergence. We also know, that we are guaranteed to find a solution to the vector equation $g = F(\theta)$ given an initial iterate θ^1 is sufficiently close to the solution. If the initial iterate is not sufficiently close then we may only get linear convergence.

In practice, J is rarely invertible. To overcome these problems one may use the Moore–Penrose pseudo-inverse in which case the method is known as the Jacobian Inverse method [10]. The pseudo-inverse update is the solution of the least square problem of minimizing $\frac{1}{2}\|J\Delta\theta^k - (g - F(\theta^k))\|^2$. The residual function, r, can be written as

$$r(\Delta\theta) = y^{k+1} - g = F(\theta^k + \Delta\theta) - g. \tag{10}$$

Taking a first-order approximation yields

$$r(\Delta\theta) \approx J\Delta\theta - (g - F(\theta^k)). \tag{11}$$

Using this linear residual model we wish to minimize $\|J\Delta\theta - (g - F(\theta^k))\|$ or equivalently

$$f(\Delta\theta) = \tfrac{1}{2}\|J\Delta\theta - (g - F(\theta^k))\|^2. \tag{12}$$

From the first-order optimality conditions we have the minimizer

$$\nabla f(\Delta\theta^*) = J^T J\Delta\theta^* - J^T(g - F(\theta^k)) = 0. \tag{13}$$

Setting $\Delta\theta^k$ to be the minimizer and re-arranging terms while assuming full column-rank of J, we have

$$\Delta\theta^k = (J^T J)^{-1} J^T (g - y^k). \tag{14}$$

Thus, the pseudo-inverse is a Gauss–Newton type of method that yield the solution of a least square problem.

A major draw-back of the pseudo-inverse method is the discontinuity of the pseudo-inverse near a singularity [17]. A damped least square (Levenberg–Marquardt) type method can be used to overcome this problem. That is, one seek to minimize $\frac{1}{2}\|J\Delta\theta - (g - F(\theta^k))\|^2 + \lambda^2\|\Delta\theta\|^2$, where $\lambda > 0$ is a regularization/damping parameter. Performing a similar derivation as above results in the update formula,

$$\Delta\theta^k = (J^T J + \lambda^2 I)^{-1} J^T (g - y^k). \tag{15}$$

However, one must deal with the problem of selecting a regularization value. Actually, Levenberg–Marquardt is using $J^T J$ as the Hessian approximation and can be understood as a modified Newton method combined with a trust region [23]. Notice that in most work on inverse kinematics only a single Gauss–Newton or Levenberg–Marquardt iteration is taken to solve the Newton subsystem.

In some cases, the inverse Jacobian can be approximated by the transpose, $J^{-1} \approx J^T$, this variant of the method is known as the Jacobian Transpose method [32].

The Jacobian Transpose method has linear convergence to the unconstrained minimizer. One may also use singular value decomposition to obtain a minimum norm solution. Singular value decomposition has the benefit that one can deal with the singularities and ill-conditioning arising from the loss of freedom [17], while it retains the ability to handle secondary goals. The open source software package Blender [2] uses a singular value decomposition-based Jacobian Inverse method, which deals with joint limits by projection of the final solution unto the feasible region. If one uses a matrix splitting method [28] for solving the Newton equation then one would obtain the equivalent of the Cyclic-Coordinate-Descent method [32]. The iteration cost of this method is very low. However, it has poor convergence rate.

All of the above variants suffer from the following two drawbacks. First their extension to deal with joint limits is not an explicit part of the mathematical model and can be described as applying a back-projection of non-feasible iterates disregarding the optimality of the projected iterate. Second, the Newton subsystem is not well-posed and approximate solutions are needed in one way or the other. This often results in poor convergence rate and maybe even divergence.

Taking a non-linear optimization approach to the inverse kinematics problems allows one to model joint limits in the underlying mathematical model of the problem, and the numerical problem of singularities of the Jacobian is avoided.

4. A non-linear optimization approach

We use a squared weighted norm formulation measuring the distance between the goal positions and the end-effector positions. This formulation is similar to Zhao and Badler [34], and like them we can support numerous goal types including both position and orientation goals. The main difference being that we have agglomerated all goals and introduced a general weighting matrix instead of dealing with K square weighted summation terms. This formulation is well suited for calculation of the solution to a global kinematics problem with multiple end-effectors, because it takes into account an interdependency between various branches. The jumping motion in Fig. 6 is an example of such an animation with multiple dependencies.

Given a branched mechanism containing K kinematic chains, where each chain has exactly one end-effector. We agglomerate the K end-effector functions into one function

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_j \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} F_1(\theta) \\ \vdots \\ F_j(\theta) \\ \vdots \\ F_K(\theta) \end{bmatrix} = F(\theta), \tag{16}$$

where y_j is the world coordinate position of the jth end-effector, and $F_j(\theta)$ is the end-effector function corresponding to the jth kinematic chain. Using an agglomerated end-effector function, we create an objective function

$$f(\theta) = (g - F(\theta))^T W (g - F(\theta)), \tag{17}$$

where W is a symmetric positive definite and possible diagonal matrix and $g = [g_1^T \quad \cdots \quad g_K^T]^T$ is the agglomerated vector of end-effector goals. The optimization problem is

$$\theta^* = \underset{\theta}{\operatorname{argmin}} f(\theta) \tag{18}$$

subject to the linear box-constraints

$$\theta \geq l, \tag{19a}$$

$$\theta \leq u, \tag{19b}$$

which models the minimum and maximum joint-parameter values. Here l is a vector containing the minimum joint limits and u is a vector of the maximum joints limits. This implies $l \leq u$ at all times.

If F is sufficiently smooth and $\theta^k \to \theta^*$ as $k \to \infty$ then F behaves almost as a quadratic function at θ^*. This intuition suggest that when we get close to a solution the formulation behaves as a convex quadratic minimization problem. Further, by design all constraints are linear functions defining a convex feasible region. Thus, a simple constraint qualification for the first-order necessary Karush–Kuhn–Tucker optimality conditions is always fulfilled [23]. This would imply that a Newton method would be the method best suited for solving this problem. Unfortunately this approach is infeasible, since the interactivity requirements of the system prohibits the costly computation of a Hessian and even Quasi-Newton methods such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [23] would still be costly compared to methods relying solely on the first-order information. Originally, Zhao and Badler [34] combined a Quasi-Newton method with the active set idea used in the projected gradient method of Rosen [26,27]. The idea can be stated as modifying the Newton direction by applying a projection operator to the current Hessian approximation such that the resulting Newton search direction obtained from the Newton sub-system is kept inside the feasible region. Of course then one must search for blocking constraints when performing a line-search, and furthermore, one must update the set of active constraints and conduct the corresponding projections on the Hessian approximation. The approach of Zhao et al. does not exploit the fact that the above formulation has a convex feasible region. In fact the feasible region is a boxed feasible region. This suggest that all the book-keeping and modifications of the Hessian matrix can be omitted if the active set idea is replaced by a projected line-search. Unfortunately, preliminary testing showed us that projection of a Quasi-Newton direction was not a feasible approach, since the projected direction can no longer be guaranteed to give a reduction in the objective function. To be able to guarantee this reduction a method which is more perpendicular to the iso-contour of the objective function must be chosen.

5. Projected gradient

The simplest idea for a projected line-search method is to use the gradient descent as a basis. This method is called gradient projection or the projected gradient method. It is very well suited for non-linear optimization with box constraints and is robust. In fact, it is not even necessary to assume feasibility of the previous iterate to ensure feasibility of the current iterate, since the method relies on projection. The unprojected search direction of the gradient descent is orthogonal to the iso-contour of the objective function. Thus, we can guarantee that the projected search direction will always be a descent direction for a sufficiently small step-length.

5.1. Computing the gradient

From (17) we have

$$f(\theta) = (g - F(\theta))^T W (g - F(\theta)). \tag{20}$$

The differential can be computed as follows:

$$df = d(g - F(\theta))^T W (g - F(\theta)) + (g - F(\theta))^T W \, d(g - F(\theta)), \tag{21}$$

which reduces to

$$df = 2(g - F(\theta))^T W d(g - F(\theta)), \tag{22}$$

Fig. 6. Example of the visual quality showing 11 key-frames in an animation of a jump. The animation was edited manually using a simple editing suite made in connection with this paper.

where

$$d(g-F(\theta)) = -\frac{\partial F(\theta)}{\partial \theta} d\theta, \tag{23a}$$

$$= -J\, d\theta. \tag{23b}$$

Which means

$$df = -2(g-F(\theta))^T W J\, d\theta. \tag{24}$$

From this we have

$$\frac{df}{d\theta} = -2(g-F(\theta))^T W J, \tag{25}$$

and the gradient can now be written as

$$\nabla f = \frac{df^T}{d\theta} = -2J^T W(g-F(\theta)). \tag{26}$$

How to compute the Jacobian is treated in Appendix A.

5.2. Finding a step-length

Given this search direction we can update our parameter vector θ by

$$\theta^{k+1} = \theta^k - \tau \nabla f, \tag{27}$$

where τ is some scalar. If τ is a constant or given by a formula this is equivalent to solving the Jacobian Transpose method as can be seen from (26).

Several values of τ have been tried for the Jacobian Transpose method. In Buss [3], they compute the step-length according to

$$\tau = \frac{e^T z}{z^T z} \quad \text{where} \quad e = (g-y^k) \quad \text{and} \quad z = JJ^T e. \tag{28}$$

Whereas in Meredith and Maddock [21] they compute the step-length such that

$$\|(I-JJ^+\tau e)\| \le \varepsilon, \tag{29}$$

where $\varepsilon > 0$ is a user-specified constant. The rationale behind both approaches is to measure the deviation from the linear approximation. If the deviation is too big then the step-length is reduced. In Robotics, rate control is used [24,14] in which $\|\Delta\theta\|$ is clamped to a specified maximum, and only a single iteration is used. Others [4] clamp the goal-displacement if it exceeds a threshold. Due to linearity down-scaling, the goal-displacement is equivalent to using a smaller step-length.

A non-linear optimization method uses some dynamic scheme to find a suitable step-length in each iteration. Usually, a simple inexact line-search is used, such as the Armijo back-tracking. As our approach needs to satisfy the constraints we need a modification of this approach as we will describe next.

5.3. Projected Armijo back-tracking line-search

An important part of a projected method is the projected line-search since it is here that an actual projection and thus the constraining of the solution is done. Numerous ways of performing inexact line-searches exist and most of them could be combined with projection. We have chosen the Armijo back-tracking approach because of its beneficial properties with regard to speed and robustness, and because it guarantees good improvements in the objective function when such is possible.

We can think of $f(\theta)$ as being a function of the step-length parameter, τ, thus we may write

$$f(\tau) = f(\theta + \tau\Delta\theta). \tag{30}$$

A first-order Taylor approximation around $\tau = 0$ yields

$$f(\tau) \approx f(0) + f'(0)\tau. \tag{31}$$

The sufficient decrease condition, the Armijo condition [23], is

$$f(\tau) < f(0) + \alpha f'(0)\tau \tag{32}$$

for some $\alpha \in [0..1]$. Observe that

$$f' = \frac{d}{d\tau} f(\theta + \tau\Delta\theta) = \nabla f(\theta)^T \Delta\theta. \tag{33}$$

This is nothing more than the directional derivative of f taken at θ and in the direction of $\Delta\theta$. The idea is now to perform an iterative step reduction by setting $\tau^1 = 1$ and verify the above test. If the test fails one updates the step-length as

$$\tau^{k+1} = \beta\tau^k, \tag{34}$$

where $\alpha \le \beta < 1$ is the step-reduction parameter. Performing back-tracking ensures the longest possible step is taken. Therefore, there is no need for a curvature condition to avoid unnecessarily small steps. We can now rephrase the test as follows:

$$f(\theta + \tau^k\Delta\theta) < f(\theta) + (\alpha\nabla f(\theta)^T \Delta\theta)\tau^k. \tag{35}$$

This is the Armijo test used in an unprojected line-search. If a projected line-search is done, then we can think of θ as a function of τ^k so we write

$$\hat{\theta}(\tau^k) = \theta + \Delta\theta\tau^k, \tag{36}$$

moving some terms around results in

$$\Delta\theta\tau^k = \hat{\theta}(\tau^k) - \theta. \tag{37}$$

Using this in the original Armijo condition we have

$$f(\hat{\theta}(\tau^k)) < f(\theta) + \alpha\nabla f(\theta)^T(\hat{\theta}(\tau^k) - \theta). \tag{38}$$

Next we will keep $\hat{\theta}(\tau^k)$ feasible by doing a projection onto the feasible region

$$f(P(\hat{\theta}(\tau^k))) < f(\theta) + \alpha\nabla f(\theta)^T(P(\hat{\theta}(\tau^k)) - \theta), \tag{39}$$

where P is a projection operator, for instance it could be

$$P(\hat{\theta}(\tau^k)) = \max(\min(\hat{\theta}(\tau^k), u), l), \tag{40}$$

where the comparison is element-wise. The vectors l and u would be constant lower and upper bounds for θ. This ensures that even if a previous iterate was infeasible then the current iterate will be feasible. Given this projected line-search it is possible to perform a fast and robust computation of the pose.

6. Performance of projected back-tracking line-search

The projected line-search developed in this paper has been tested with two different methods: the Jacobian Transpose method, which is the same as a Steepest Descent method using a fixed step-length, and a projected line-search method with the projected Armijo back-tracking line-search we described in Section 4.3. We have compared our results with the results from the SVD based Jacobian Inverse method used in the Blender Software package.

6.1. Comparisons

Blender was chosen as an example of a widely used software package which lives up to current performance and quality demands. The reason for choosing Blender instead of, e.g. Maya or 3D Max was, that it is possible to perform actual measurements on Blender due to it being open source. Since the functionality, quality and performance of Blenders inverse kinematics system are very similar to the other solutions (see, e.g. Fig. 4 or the so-called cg survey by CGgenie [5]), we chose only this one.

Our reference implementation of the Jacobian Transpose method corresponds to the one used by Wellman [32], where a fixed sufficiently small step-length is used. Experiments were

done with the Jacobian Transpose method varying the fixed step-length. These experiments showed that a step-length of more than 0.005 would make the Jacobian Transpose method diverge even if the method were given an initial iterate comparatively close to a solution. Step-length below this limit slowed down the method without discernible improvement to the quality. Therefore, in all our results we used Jacobian Transpose method with a fixed step-length of 0.005. An example of the divergence exhibited with too large step-length is shown in Fig. 7.

Fig. 7. An example of what may happen if the step-length of the Jacobian Transpose method is chosen too high. The motion flips between the ghosted poses, superimposed on the desired result. Step-length in this example was set at 0.05. This example used five positional goals and joint limits.

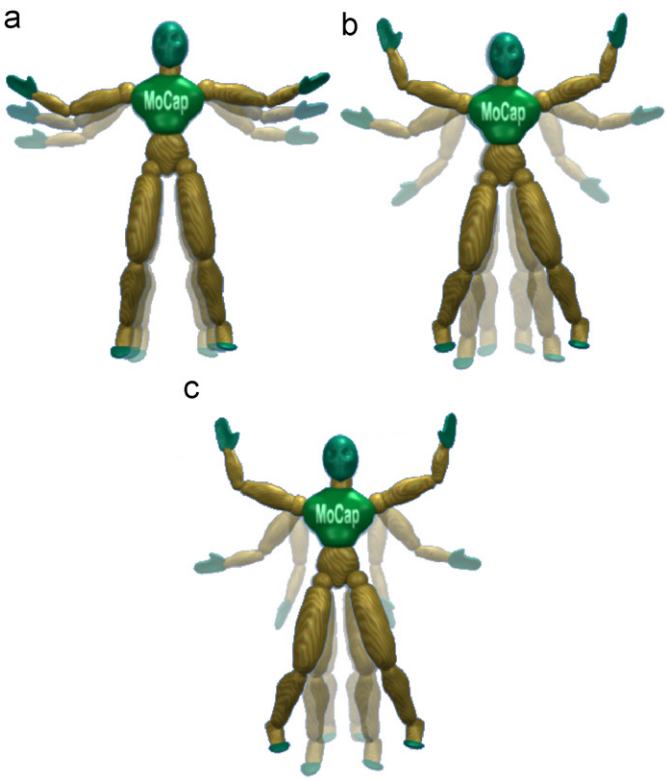

Fig. 8. The difference between the three sampling settings of motion capture data. Notice that spatial-temporal coherence is decreased from left to right. (a) Animation using 28 frames per cycle. (b) Animation using seven frames per cycle. (c) Animation using five frames per cycle.

6.2. Test framework

The methods were tested by running a number of repeated tests under different conditions. The test scenario comprised a fixed setup using a motion captured animation of a person doing a gymnastics exercise. A time lapse of the animation showing an exercise can be seen in Fig. 12. The positions of the hands, feet, pelvis and head of the motion capture animation were used as goals for the inverse kinematics skeleton. The orientation of the goals was not used in these tests. The goals were updated each frame. The skeleton consists of 71 degrees of freedom, and joint limit constraints on all joints, giving a total of 142 constraints (71 upper and 71 lower).

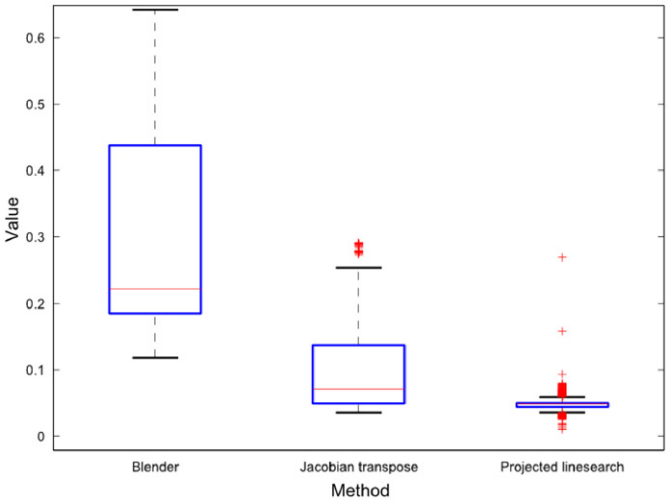

Fig. 9. Plot of the distribution corresponding to the animation shown in Fig. 8(a). Crosses denote outliers in the data set. Notice that the projected line-search has much lower median as well as a more compact distribution.

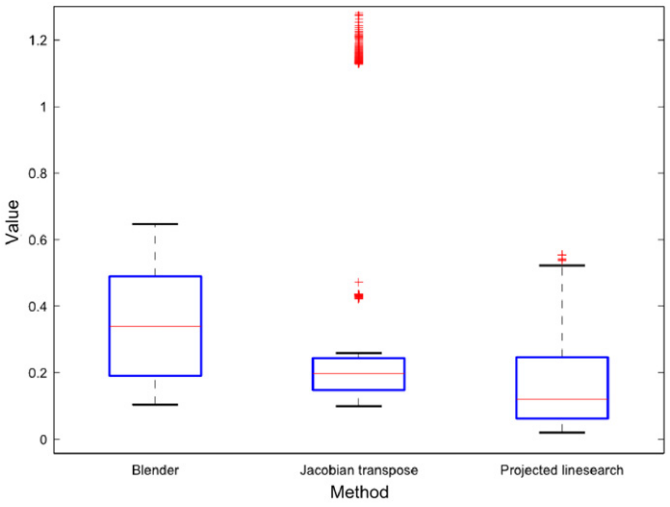

Fig. 10. Plot of the distribution corresponding to the animation shown in Fig. 8(b). Crosses denote outliers in the data set.

Since interactive performance was a major factor in the evaluation we chose to compare the quality of the method with a fixed time slot at their disposal giving each a maximum number of iterations which would make it converge in approximately 0.015 s or less, corresponding to approximately 66 frames/s.

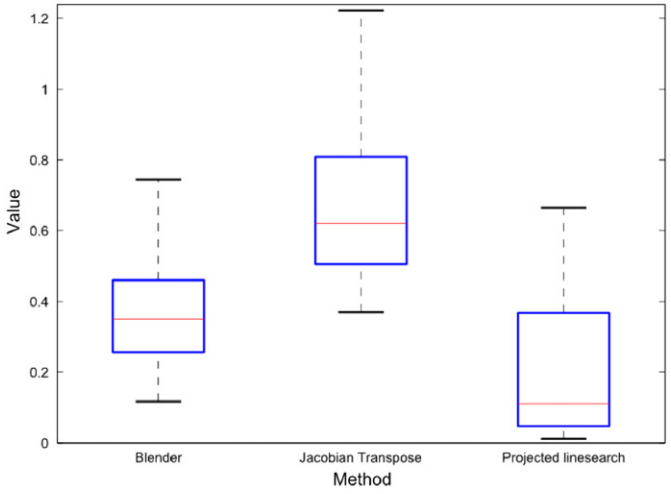

Fig. 11. Plot of the distribution corresponding to the animation shown in Fig. 8(c). Notice that the Jacobian Transpose method is beginning to have problems in this case due to the lack of coherence. Some significant outliers with values of around 80 has been omitted from the Jacobian Transpose method plot to make comparison possible.

We varied how far the goal positions were placed from the end-effector positions. This was done by sampling the motion captured data with varying intervals. The settings used were 0.05, 0.7, and 1.0 s. Fig. 8 shows three consecutive frames of the reference animation, using the three settings.

The test cases were run on an Intel® dual core 1.66 GHz architecture with 1 GB memory utilizing only one core.

An absolute tolerance of 0.05 units was set. The tolerance was chosen to be small enough not to interfere with the visual quality of the animation. The 0.05 units is approximately 1 cm if the skeleton corresponds to a person that is approximately 1.80 m high. The function-value in Figs. 9, 10, and 11 are the sums of all end-effector squared errors.

6.3. *Visual quality*

The visual quality is graded by the closeness to the reference animation as well as the smoothness of the animation. In both cases the gradient projection gives superior results to what is being computed by the SVD-based Jacobian Inverse method. This can be seen from the plots in Figs. 9, 10 and 11. The figures show the median value as a red line. The edges of the blue boxes denote the 25th and 75th percentiles, the whiskers denote the boundary of the inliers of the data set. Outliers are shown individually as red crosses. The smoothness of the animation and the handling of joint limits and orientation/position of intermediate joints can be seen from the frames in Fig. 12 notice frames 7 and 9, where the animation made by the reference method clearly jitters.

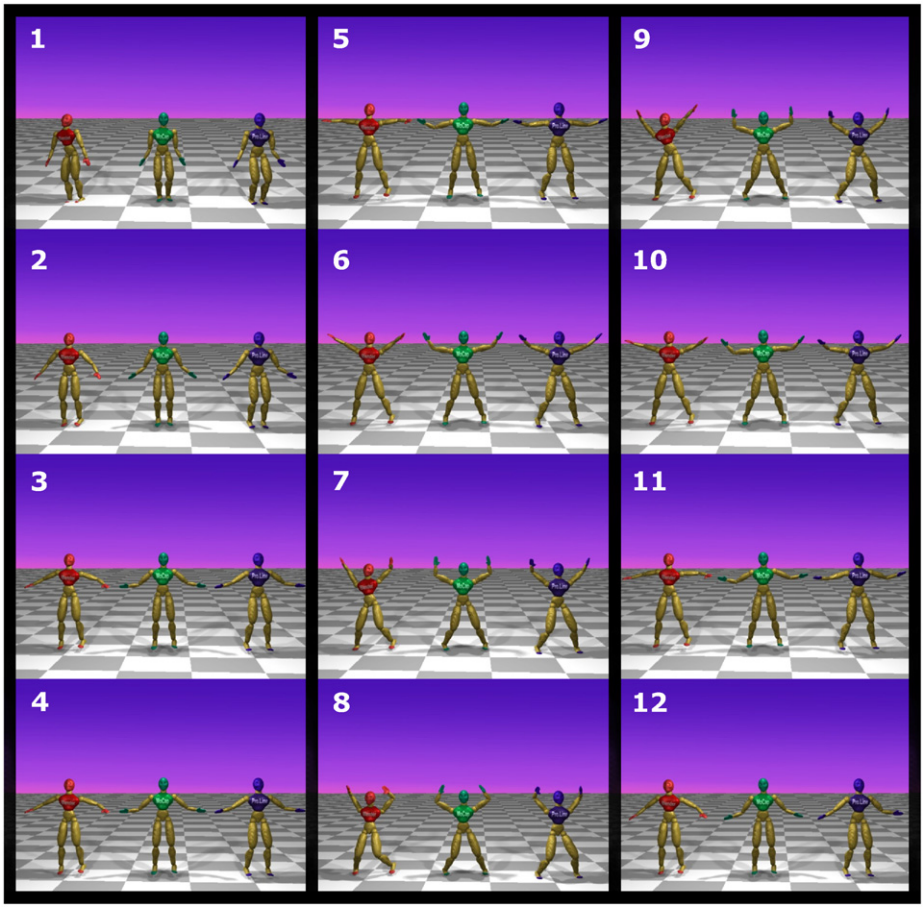

Fig. 12. Selected frames from the animation used in testing the methods, each frame shows from left to right: the Blender version, the motion capture reference, the projected line-search method. Notice that the Blender method jitters from frame-to-frame and that it differs significantly from the reference (notice especially frames 7 and 9).

6.4. Robustness

Since the method guarantees a reduction in the objective function, the method is very robust. No restrictions on previous poses are necessary. The robustness can be seen from the tight distributions of the values in Figs. 9, 10 and 11.

6.5. Generality

The method can be used with any conceivable skeleton. Examples of skeletons we have tested are shown in Fig. 1. The method is easily integrable in other systems. The method is implemented in the free open-source meta-library OpenTissue [25], and can be included in any source code under the Zlib license [35]. As shown in Fig. 1 the method has been included in an ogre demo by a third party.

7. Discussion and conclusion

The non-linear optimization approach has shown beneficial behavior as a stronger mathematical formulation of the inverse kinematics problem. We have presented an efficient and robust numerical approach for solving the problem. We believe our approach is simple and elegant and easy to implement, even for non-specialist in numerical optimization. Further, we have demonstrated that our approach is sufficiently exact and responsive for interactive manipulation of multiple end-effectors while handling joint limits.

We have shown the connection between using a projected gradient method for the non-linear optimization and the traditional Jacobian Transpose method. Further, we showed how the Jacobian Transpose method could be improved by using our projected line-search method.

The shaky and jittery motion often reported near singular configurations can be avoided completely when using an optimization approach, since the variable step-length cancels out the adverse effect of the high angular velocity. It is thus only a matter of a sufficiently exact line-search and a sufficiently low minimum step-length.

In our opinion, we have only touched upon the subject of seeking the best suited mathematical formulation for interactive inverse kinematics and matching numerical methods. There are many possible avenues for further work. It could be interesting to reformulate the first-order necessary optimality conditions into a complementarity problem and further reformulate this as a possible smooth or non-smooth non-linear equation solving problem. This would in a sense take us back to the equation solving approaches but with the difference that constraints are dealt with implicitly without the need for projections at all. Quasi-Newton and Steepest Descent methods are but a few out of many methods for solving the constrained minimization problem we have stated. Trust region methods or hybrids may be other interesting methods to adapt to the formulation given.

Acknowledgment

The Motion capture data used in this project was obtained from mocap.cs.cmu.edu.

Appendix A. A geometric approach to the differential

To give the reader a more full picture of the method we have included this appendix explaining details of the Jacobian computation. The derivations in this appendix are based on previous work such as Zhao and Badler [34], but describes this in more detail for the multiple end-effector case.

Without loss of generality, we will choose homogeneous coordinates to develop the theory in the following. Given a chain with n links, we have ${}^0T_1,\ldots,{}^{n-1}T_n$ homogeneous coordinate matrices. We will assume that the ith joint depends on the parameters θ_i. That is ${}^{i-1}T_i$ can be thought of as a function of θ_i. We will specify the tool held by the end-effector and the goal placement of the tool by the agglomerated vectors

$$[y]_n = \begin{bmatrix} p \\ \hat{i} \\ \hat{j} \end{bmatrix}_n, \quad [g]_0 = \begin{bmatrix} p_{\text{goal}} \\ \hat{i}_{\text{goal}} \\ \hat{j}_{\text{goal}} \end{bmatrix}_0 \in \mathbb{R}^3 \times S^2 \times S^2, \tag{A.1}$$

where p is the position while $\hat{i}$ and $\hat{j}$ are unit vectors specifying the orientation. The bracket notation $[\cdot]_i$ is used to make it explicit that vectors are expressed in the coordinates of the ith joint frame. Using homogeneous coordinates we can write the instantaneous position of the tool as

$$y = \begin{bmatrix} p \\ \hat{i} \\ \hat{j} \end{bmatrix}_0 = \begin{bmatrix} {}^0T_n & 0 & 0 \\ 0 & {}^0T_n & 0 \\ 0 & 0 & {}^0T_n \end{bmatrix} \begin{bmatrix} p \\ \hat{i} \\ \hat{j} \end{bmatrix}_n = F(\theta), \tag{A.2}$$

where ${}^0T_n = {}^0T_1 \cdots {}^{n-1}T_n$. Often one would use the practical choices

$$[p]_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad [\hat{i}]_n = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad [\hat{j}]_n = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \tag{A.3}$$

which could greatly simplify all expressions. However, in the following we will keep things general. Let us investigate the differential of the end-effector function

$$dF = \underbrace{\frac{\partial F(\theta)}{\partial \theta}}_{J} d\theta = [J_1 \quad \cdots \quad J_i \quad \cdots J_n] \begin{bmatrix} d\theta_1 \\ \vdots \\ d\theta_i \\ \vdots \\ d\theta_n \end{bmatrix}, \tag{A.4}$$

where J is the Jacobian. The above equation tell us what the differential change of the end-effector would be if we induced some differential change in the joint parameters. This opens up for an intuitive way of computing J. We observe that $J_i d\theta_i$ describes how the end-effector position is influenced by a change in θ_i. Without loss of generality, we will only focus on the position term. The remaining terms follows in a similar fashion. We have

$$\frac{\partial}{\partial \theta_i} \begin{bmatrix} p \\ 1 \end{bmatrix}_0 d\theta_i = \underbrace{{}^0T_1 \cdots {}^{i-2}T_{i-1}}_{{}^0T_{i-1}} \frac{\partial({}^{i-1}T_i)}{\partial \theta_i} \underbrace{{}^iT_{i+1} \cdots {}^{n-1}T_n}_{{}^iT_n} \begin{bmatrix} p \\ 1 \end{bmatrix}_n d\theta_i \tag{A.5a}$$

$$= {}^0T_{i-1} \frac{\partial({}^{i-1}T_i)}{\partial \theta_i} \begin{bmatrix} p \\ 1 \end{bmatrix}_i d\theta_i. \tag{A.5b}$$

Assume that the ith joint is a revolute joint with the unit joint axis $[u_i]_{i-1} = [x_i \; y_i \; z_i]^T$ specified as a constant vector in the i-1th frame. One can show

$$\frac{\partial({}^{i-1}T_i)}{\partial \theta_i} = \begin{bmatrix} U_i^{\times} R_i & 0 \\ 0^T & 0 \end{bmatrix}, \tag{A.6}$$

where R_i is the 3-by-3 upper part of ${}^{i-1}T_i$

$${}^{i-1}T_i = \begin{bmatrix} R_i & t_i \\ 0^T & 1 \end{bmatrix}, \tag{A.7}$$

where t_i is the translational part and

$$U_i^{\times} = \begin{bmatrix} 0 & y_i & -z_i \\ -y_i & 0 & x_i \\ z_i & -x_i & 0 \end{bmatrix} \tag{A.8}$$

is the skew-symmetric cross-product matrix. That is $[u_i]_{i-1} \times p = U_i^{\times} p$ for some $p-$vector. This means we have

$$\frac{\partial}{\partial\theta_i}\begin{bmatrix} p \\ 1 \end{bmatrix}_0 d\theta_i = {}^0T_{i-1}\begin{bmatrix} U_i^{\times}R_i & 0 \\ 0^T & 0 \end{bmatrix}\begin{bmatrix} p \\ 1 \end{bmatrix}_i d\theta_i, \tag{A.9a}$$

$$= {}^0T_{i-1}\begin{bmatrix} U_i^{\times}R_i[p]_i \\ 0 \end{bmatrix} d\theta_i. \tag{A.9b}$$

Notice that $R_i[p]_i = [p]_{i-1} - [t_i]_{i-1}$, so

$$\frac{\partial}{\partial\theta_i}\begin{bmatrix} p \\ 1 \end{bmatrix}_0 d\theta_i = {}^0T_{i-1}\begin{bmatrix} [u_i]_{i-1} \times ([p]_{i-1} - [t_i]_{i-1}) \\ 0 \end{bmatrix} d\theta_i, \tag{A.10a}$$

$$= \begin{bmatrix} [u_i]_0 \times ([p]_0 - [t_i]_0) \\ 0 \end{bmatrix} d\theta_i. \tag{A.10b}$$

Using similar derivations for i and j terms allow us to conclude that we can obtain the ith column of the Jacobian corresponding to a revolute joint by

$$J_i = \begin{bmatrix} [u_i]_0 \times ([p]_0 - [t_i]_0) \\ [u_i]_0 \times [\hat{i}]_0 \\ [u_i]_0 \times [\hat{j}]_0 \end{bmatrix}. \tag{A.11}$$

If the ith joint was a prismatic joint with sliding along the joint axis given by the unit vector $[u_i]_{i-1}$ then one would have

$$\frac{\partial({}^{i-1}T_i)}{\partial\theta_i} = \begin{bmatrix} 0 & [u_i]_{i-1} \\ 0^T & 0 \end{bmatrix} \tag{A.12}$$

from this we have

$$\frac{\partial}{\partial\theta_i}\begin{bmatrix} p \\ 1 \end{bmatrix}_0 d\theta_i = {}^0T_{i-1}\begin{bmatrix} 0 & [u_i]_{i-1} \\ 0^T & 0 \end{bmatrix} {}^iT_n \begin{bmatrix} p \\ 1 \end{bmatrix}_n d\theta_i, \tag{A.13a}$$

$$= {}^0T_{i-1}\begin{bmatrix} u_i \\ 0 \end{bmatrix}_{i-1} d\theta_i, \tag{A.13b}$$

$$= \begin{bmatrix} u_i \\ 0 \end{bmatrix}_0 d\theta_i. \tag{A.13c}$$

Note that the sub-parts corresponding to orientation, $\hat{i}$ and $\hat{j}$, are zero

$$\frac{\partial}{\partial\theta_i}\begin{bmatrix} \hat{i} \\ 0 \end{bmatrix}_0 = {}^0T_{i-1}\begin{bmatrix} 0 & [u_i]_{i-1} \\ 0^T & 0 \end{bmatrix} {}^iT_n \begin{bmatrix} \hat{i} \\ 0 \end{bmatrix}_n d\theta_i, \tag{A.14a}$$

$$= {}^0T_{i-1}\begin{bmatrix} 0 & [u_i]_{i-1} \\ 0^T & 0 \end{bmatrix}\begin{bmatrix} \hat{i} \\ 0 \end{bmatrix}_i d\theta_i, \tag{A.14b}$$

$$= 0 \tag{A.14c}$$

similar for the $\hat{j}-$term. Thus for the case of the prismatic joint we have

$$J_i = \begin{bmatrix} [u_i]_0 \\ 0 \\ 0 \end{bmatrix}. \tag{A.15}$$

The extension to more than one-translation axis is trivial.

As seen from all the above derivations the "effect" of the tool is computed from the geometry expressed in the world coordinate system, and we can conclude that the computation of the Jacobian is totally independent of what type of coordinate representation one uses. We exploit this to use a quaternion representation rather than a homogeneous coordinate representation. Since spherical linear interpolation quaternions are better suited for interpolation of the in-between key-frames and superior skinning techniques exist based on quaternions as well [15]. Thus, one can avoid conversion between quaternions and matrices completely. Further, the computational effort in computing the absolute transformations of each joint is less expensive using a quaternion representation than using a matrix representation.

A.1. A ball-and-socket joint

In the above, we only dealt with having one rotation axis and multiple translation axes. Thus, a natural question is what to do with multiple rotation axes? Traditionally, this has been handled by creating imaginary multiple joints. Thus a joint with rotation around three axes can be modeled as three revolute joints placed on the top of each other. In the following, we will derive equations for dealing with such a joint in a canonical way.

Euler angles are a popular choice as parameterization in motion capture and animation formats, therefore, we will use an Euler angle parameterization of a ball-and-socket joint. Inspired by the robotics community [7,18], we will adopt the *ZYZ* Euler angle convention. This means that if the ith joint is a ball-and-socket joint then it will be parametrized by the angles θ_i, ϕ_i, and ψ_i. Further, the relative transformation is given as

$${}^{i-1}T_i = \begin{bmatrix} R_Z(\phi_i)R_Y(\psi_i)R_Z(\theta_i) & t_i \\ 0^T & 0 \end{bmatrix}, \tag{A.16}$$

where R_Z and R_Y are rotations around the z and y axes of the $(i-1)$th joint frame. Trivially we have

$$\frac{\partial({}^{i-1}T_i)}{\partial\phi_i} = \begin{bmatrix} (Z^{\times}R_Z(\phi_i))R_Y(\psi_i)R_Z(\theta_i) & 0 \\ 0^T & 0 \end{bmatrix}, \tag{A.17a}$$

$$\frac{\partial({}^{i-1}T_i)}{\partial\psi_i} = \begin{bmatrix} R_Z(\phi_i)(Y^{\times}R_Y(\psi_i))R_Z(\theta_i) & 0 \\ 0^T & 0 \end{bmatrix}, \tag{A.17b}$$

$$\frac{\partial({}^{i-1}T_i)}{\partial\theta_i} = \begin{bmatrix} R_Z(\phi_i)R_Y(\psi_i)(Z^{\times}R_Z(\theta_i)) & 0 \\ 0^T & 0 \end{bmatrix}, \tag{A.17c}$$

where

$$Y^{\times}p = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times p \quad \text{and} \quad Z^{\times}p = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times p \tag{A.18}$$

for some vector p. Next we define the three vectors

$$[u_i]_{i-1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_{i-1}, \tag{A.19a}$$

$$[v_i]_{i-1} = R_Z(\phi_i)\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}_{i-1}, \tag{A.19b}$$

$$[w_i]_{i-1} = R_Z(\phi_i)R_Y(\psi_i)\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_{i-1}, \tag{A.19c}$$

and by straightforward computation we have

$$\frac{\partial[p]_0}{\partial\phi_i} d\phi_i = ({}^0T_{i-1}[u_i]_{i-1}) \times ([p]_0 - [t_i]_0)\, d\phi_i, \tag{A.20a}$$

$$\frac{\partial[p]_0}{\partial\psi_i} d\psi_i = ({}^0T_{i-1}[v_i]_{i-1}) \times ([p]_0 - [t_i]_0)\, d\psi_i, \tag{A.20b}$$

$$\frac{\partial [p]_0}{\partial \theta_i}\, d\theta_i = ({}^0T_{i-1}[w_i]_{i-1}) \times ([p]_0 - [t_i]_0)\, d\theta_i, \tag{A.20c}$$

now renaming the vector $([p]_0 - [t_i]_0)$

$$r_0 = ([p]_0 - [t_i]_0), \tag{A.21}$$

we get the Jacobian entry

$$J_i = \begin{bmatrix} [u_i]_0 \times r_0 & [v_i]_0 \times r_0 & [w_i]_0 \times r_0 \\ [u_i]_0 \times [\hat{i}]_0 & [v_i]_0 \times [\hat{i}]_0 & [w_i]_0 \times [\hat{i}]_0 \\ [u_i]_0 \times [\hat{j}]_0 & [v_i]_0 \times [\hat{j}]_0 & [w_i]_0 \times [\hat{j}]_0 \end{bmatrix}. \tag{A.22}$$

References

[1] Autodesk. web page, ⟨http://usa.autodesk.com⟩; 2010.
[2] Blender. The blender foundation, web page, ⟨http://www.blender.org⟩; 2010.
[3] Buss SR. Introduction to inverse kinematics with Jacobian transpose, pseudo-inverse and damped least squares methods. unpublished survey; April 2004.
[4] Buss SR, Kim J-S. Selectively damped least squares for inverse kinematics. Journal of Graphics Tools 2005;10(3):37–49.
[5] CGgenie. Cg genie. web page, ⟨http://cgenie.com/⟩; 2010.
[6] Chiacchio P, Siciliano B. A closed-loop Jacobian transpose scheme for solving the inverse kinematics of nonredundant and redundant wrists. Journal of Robotic Systems 1989;6(5):601–30.
[7] Craig JJ. Introduction to robotics: mechanics and control. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 1989.
[8] Fang AC, Pollard NS. Efficient synthesis of physically valid human motion. ACM Transactions on Graphics (TOG) 2003;22(3):417–26.
[9] Fêdor M. Application of inverse kinematics for skeleton manipulation in real-time. In: SCCG '03: proceedings of the 19th spring conference on computer graphics. New York, NY, USA: ACM Press; 2003. p. 203–12.
[10] Girard M, Maciejewski AA. Computational modeling for the computer animation of legged figures. In: SIGGRAPH '85: proceedings of the 12th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM Press; 1985. p. 263–70.
[11] Hauberg S, Lapuyade J, Engell-Nørregård M, Erleben K, Pedersen KS. Three dimensional monocular human motion analysis in end-effector space. In: Cremers D, editor. Energy minimization methods in computer vision and pattern recognition. Lecture notes in computer science. Springer; August 2009. p. 235–48.
[12] Herda L, Urtasun R, Hanson A, Fua P. Automatic determination of shoulder joint limits using quaternion field boundaries. International Journal of Robotics Research 2003;22(6):419–34.
[13] Ho ESL, Komura T, Lau RWH. Computing inverse kinematics with linear programming. In: VRST '05: proceedings of the ACM symposium on virtual reality software and technology. New York, NY, USA: ACM; 2005. p. 163–6.
[14] Hsia TC, Guo ZY. New inverse kinematic algorithms for redundant robots. Journal of Robotic Systems 1991;8(1):117–32.
[15] Kavan L, Zara J. Spherical blend skinning: a real-time deformation of articulated models. In: 2005 ACM SIGGRAPH symposium on interactive 3D graphics and games. ACM Press; April 2005. p. 9–16.
[16] Liu CK, Hertzmann A, Popović Z. Learning physics-based motion style with nonlinear inverse optimization. ACM Transactions on Graphics 2005;24(3): 1071–81.
[17] Maciejewski AA. Motion simulation: dealing with the ill-conditioned equations of motion for articulated figures. IEEE Computer Graphics and Applications 1990;10(3):63–71.
[18] Mason M. Mechanics of robotic manipulation. Cambridge, MA, London, England: MIT Press; August 2001. Intelligent robotics and autonomous agents series, ISBN 0-262-13396-2.
[19] Maurel W, Thalmann D. Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones. Computers & Graphics 2000;24(2): 203–18.
[20] Meredith M, Maddock S. Using a half-Jacobian for real-time inverse kinematics. In: Proceedings of the fifth international conference on computer games: artificial intelligence, design and education (CGADIDE). United Kingdom: Reading; November 2004. p. 81–8.
[21] Meredith M, Maddock S. Adapting motion capture data using weighted real-time inverse kinematics. Computers in Entertainment 2005;3(1):5.
[22] Moradi H, Lee S. Joint limit analysis and elbow movement minimization for redundant manipulators using closed form method. In: Huang D-S, Zhang X-P, Huang G-B, editors. Advances in intelligent computing, Proceedings of international conference on intelligent computing, ICIC 2005, Hefei, China, August 23–26, 2005, Part II, Lecture notes in computer science. Springer, vol. 3645; 2005. p. 423–32.
[23] Nocedal J, Wright SJ. Numerical optimization. Springer series in operations research. New York: Springer-Verlag; 1999.
[24] Oh S-Y, Orin D, Bach M. An inverse kinematic solution for kinematically redundant robot manipulators. Journal of Robotic Systems 1984;1(3):235–49.
[25] OpenTissueBoard. Opentissue. web page, ⟨http://www.opentissue.org⟩; 2010.
[26] Rosen JB. The gradient projection method for nonlinear programming. Part I. Linear constraints. Journal of the Society for Industrial and Applied Mathematics 1960;8(1):181–217. URL ⟨http://www.jstor.org/stable/2098960⟩.
[27] Rosen JB. The gradient projection method for nonlinear programming. Part II. Nonlinear constraints. Journal of the Society for Industrial and Applied Mathematics 1961;9(4):514–32. URL ⟨http://www.jstor.org/stable/2098878⟩.
[28] Saad Y. Iterative methods for sparse linear systems. 2nd ed.. Philadelphia, PA: SIAM; 2003.
[29] Safonova A, Hodgins JK, Pollard NS. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. ACM Transactions on Graphics 2004;23(3):514–21.
[30] Shao W, Ng-Thow-Hing V. A general joint component framework for realistic articulation in human characters. In: I3D '03: proceedings of the 2003 symposium on interactive 3D graphics. New York, NY, USA: ACM; 2003. p. 11–8.
[31] Sumner RW, Zwicker M, Gotsman C, Popovic J. Mesh-based inverse kinematics. ACM Transactions on Graphics 2005;24(3):488–95.
[32] Wellman C. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University; 1993.
[33] Wilhelms J, Gelder AV. Fast and easy reach-cone joint limits. Journal of Graphics Tools 2001;6(2):27–41.
[34] Zhao J, Badler NI. Inverse kinematics positioning using nonlinear programming for highly articulated figures. ACM Transactions on Graphics 1994;13(4):313–36.
[35] Zlib. Zlib-libpng license. web page, ⟨http://opensource.org/licenses/zlib-license.php⟩; 2010.