Computers & Graphics 78 (2019) 1–11

Contents lists available at ScienceDirect

Computers & Graphics

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

Special Section on Motion in Games 2017

Solving inverse kinematics using exact Hessian matrices

Kenny Erleben[a,*], Sheldon Andrews[b]

[a] Univeristy of Copenhagen, Copenhagen, Denmark
[b] École de technologie supérieure, Montreal, Canada

ARTICLE INFO

Article history:
Received 30 June 2018
Revised 12 October 2018
Accepted 22 October 2018
Available online 31 October 2018

Keywords:
Inverse kinematics
Hessian
Optimization
Motion tracking,

ABSTRACT

Inverse kinematics (IK) is a central component of systems for motion capture, character animation, robotics motion planning and control. The field of computer graphics has developed fast stationary point methods, such as the Jacobian Transpose method and cyclic coordinate descent. Most of the work that uses Newton's method and its variants avoids directly computing the Hessian, and instead approximations are sought, such as in the BFGS class of solvers.

In this work, we present a numerical method for computing the exact Hessian of an IK system with prismatic, revolute, and spherical joints. For the latter, formulations are presented for joints parameterized by Euler angles which can be represented for instance by using quaternions. Our method is applicable to human skeletons in computer animation applications and some, but not all, robots. Our results show that using exact Hessians can give performance advantages and higher accuracy compared to standard numerical methods used for solving IK problems. Furthermore, we provide code that allows other researchers to plug-in exact Hessians in their own work with little effort.

1. Introduction

Inverse kinematics is the problem of computing the configuration (i.e., joint angles) for a kinematic chain, skeleton, or mechanism, such that an end effector will reach a prescribed goal. IK methods are fundamental for many robotics and computer graphics applications, as evidenced by literature on the topic dating back several decades [1,2]. In robotics, the problem is usually phrased as a dynamic system which may lead to different schemes [3]. An overview of numerical methods used in computer graphics can be found in Buss [4], and a comprehensive overview of inverse kinematics can be found in the recent state-of-the-art report by Aristidou et al. [5].

Popular techniques for solving IK problems typically discount the use of exact Hessians, and prefer to rely on approximations of second-order derivatives. However, robotics work has shown that exact Hessians in 2D Lie algebra-based dynamical computations outperforms approximate methods [6]. Encouraged by these results, we examine the viability of exact Hessians for inverse kinematics of 3D characters. To our knowledge, no previous work in computer graphics has addressed the significance of using a closed form solution for the exact Hessian, which is surprising since IK is pertinent for many computer animation applications and there is a large body of work on the topic.

This paper presents the closed form solution in a simple and easy to evaluate geometric form using two world space cross-products. Compared to robotics work, we target joints with general Euler angles ordering and use the familiar homogeneous coordinates representation to describe the kinematics. This technical choice is based on the popularity of Euler angles in character animation applications, but the theory we present holds equally well for quaternion representation or other parameterizations of rotation angles. We consider the derivation and algorithm for the computation of the exact IK Hessian to be a novel theoretical contribution. Furthermore, we investigate numerically whether using the exact Hessian with a Newton's method type of approach improves the performance and accuracy for IK problems with many degrees of freedom and large displacements of end-effectors. Our results suggest that using exact Hessians can outperform using approximate Hessians, or even Hessian free methods, under certain conditions.

We supplement this paper with both Python and MATLAB code[1], allowing other researchers to harvest the benefits of using exact Hessians in their own IK solvers. Our supplementary code uses both quaternions and Euler angles, thereby demonstrating that our theory extends to multiple representations of

* Corresponding author.
E-mail address: kenny@di.ku.dk (K. Erleben).

[1] https://github.com/sheldona/hessianIK.

https://doi.org/10.1016/j.cag.2018.10.012
0097-8493/

rotations. This paper is an extended version of the Motion in Games 2017 paper [7]. This edition of the paper includes full algorithmic details, extensions to prismatic joints and moving roots as well as treatment of exponential maps. Furthermore, we present an analysis into the indefinite and singular nature of exact Hessians in Appendix A.

2. Related work

Many formulations for inverse kinematics (IK) have been investigated in previous work. For instance, Ho et al. [8] solve the problem using linear programming. In Sumner et al. [9], a mesh based inverse kinematics method was presented. This relies on example poses to manipulate the mesh directly and does not handle joint limits. In robotics, closed-form methods are popular [10]. Closed-form methods often result in algebraic systems that can be solved very fast and reliably, but these methods are highly specialized for a specific low-dimensional manipulator (up to 7 degrees of freedom). Safonova et al. [11] combine inverse kinematics with other techniques and solve a Sequential Quadratic Programming (SQP) problem. The running times are in minutes, which prohibits interactive usage. Motion synthesis using space-time optimization and machine learning has also been tried [12,13] although running times are not yet within the grasp of the real-time domain. Zhao and Badler [14] present a mathematical formalism for solving IK as a constrained non-linear optimization problem. This work allows for general types of constraints and real-time performance. Furthermore, the optimization model can be used to derive a large spectrum of known methods for solving the IK problem.

Our work takes at the outset the IK optimization model of Zhao and Badler [14]. In fact, there exists many methods for solving the IK optimization problem. Engell-Nørregård and Erleben [15] shows how the Jacobian Transpose (JT) method is equivalent to a Steepest Descent (SD) method without a line-search. Furthermore, they show that Damped Least Squares (DLS) is obtained from a Gauss-Newton (GN) approach, and the damping results in a Levenberg–Marquardt (LM) type method. Other quasi-Newton type methods are readily available, such as memory limited BFGS [14] and a wide range of methods using iterative methods like Preconditioned Conjugate Gradient (PCG) method for solving the sub-system or approximate the Hessians using low-rank updates. Even Cyclic-Coordinate Descent (CCD) [16] can be derived from using a matrix-splitting approach for solving the gradient equation that forms the basis for the JT method. Interestingly, solving the gradient equation with pseudo-inverses yields a system of equations similar to the Newton systems that form the basis for GN/LM type of methods. Hence, these quasi-Newton methods are linked to the JT method. Recent work by Harish et al. [17] investigates how to parallelize a DLS approach for solving the IK problem. The work focuses on creating a parallel line-search that can help eliminate the need for many iterations, which the JT method usually requires to converge for highly non-linear motions with many degrees of freedom. This is expected from convergence theorems stating that JT/SD method has linear convergence rate [18]. However, it has been observed in practice [19]. Other work incorporates higher order information, mimicking the effect of a Hessian, by non-linear conjugate gradients and adding non-smoothness allows a full coupling of joint limits too [20]. In Fedor [21] the quasi-Newton type method is used for complex manipulation and claimed to give the most realistic looking poses, but it is the slowest. Our work shows that a Newton method can be highly accurate and outperform other methods when the exact Hessian is used. To our knowledge, no other work in computer graphics have shown this. Furthermore, Zordan [22] notes the importance of tuning parameters for numerical methods in order to achieve good performance, and this is a relevant step in our experiments too.

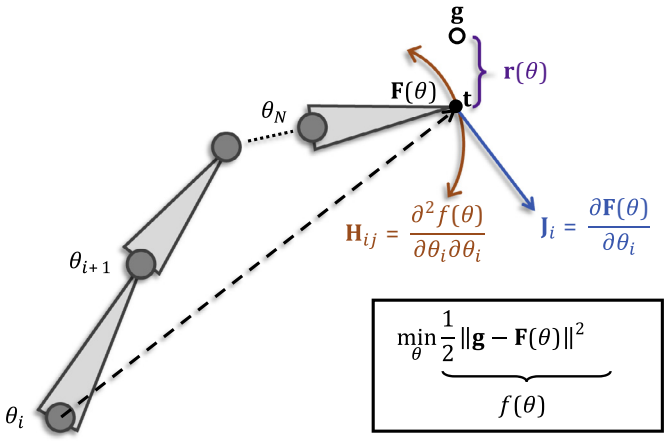

Fig. 1. A conceptualization of the IK problem: finding joint angles $\theta_0, \ldots, \theta_N$ that minimize the distance between end-effector position $F(\boldsymbol{\theta})$ and the goal position g. The distance is encoded by the residual $r(\boldsymbol{\theta})$. The non-linear optimization problem is solved using the gradient $\nabla f(\theta) = -J^T r$, and optionally the Hessian matrix H.

Inverse kinematics is known to suffer from problems with redundancies and singularities [23]. In robotics, redundancy problems have been addressed by adding more constraints [24]. 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 equivalent to adding regularization to the IK problem. In this work we address redundancies (singularities in the Hessian) by projection to nearest positive definite Hessian.

3. Theory

We begin by formally defining the inverse kinematic problem of a serial chain mechanism. Without loss of generality, we ignore branched structures and closed loops in this formalism and limit our presentation to serial chains. Fig. 1 conceptualizes many of the details we discuss in this section.

Let the *tool vector* in a serial chain be given by $\mathbf{t} \in \mathbb{R}^3$. We assume the serial chain is made of rigid links. The tool vector is placed in the last frame of the chain, which we call the end-effector frame. Note that we adopt robotics terminology throughout this section, and conceptually interpret $\mathbf{t}$ as the location of a tool being held by a robotic manipulator. However, in the context of computer graphics applications, $\mathbf{t}$ can be interpreted as a point with fixed position relative to a bone in a skeleton armature (e.g. an optical marker).

Each link describes a coordinate frame with respect to the parent link, or bone. The parent frame of the root link is the world coordinate system. Hence, a link is equivalent to a transformation of a point in the coordinate frame of the link to the coordinate frame of its parent. This means each link is equivalent to a coordinate transformation, $\mathcal{T}_k$, that transforms a vector $\mathbf{p}$ from frame k into frame $k-1$. We write this mathematically as

$$\left[\mathbf{p}\right]_{k-1} = \mathcal{T}_k \circ \left[\mathbf{p}\right]_k. \tag{1}$$

The subscripts on the vectors denote the frame of reference. The world coordinate system (WCS) is given the index $k = -1$. The operator $\circ$ is used to denote function composition. It means the coordinate transformation $\mathcal{T}_k$ is applied to the term on the right of the operator.

Given a N link serial chain and a tool vector $\mathbf{t}$ and known joint parameters, we wish to compute the tool vector position in the world coordinate system $\left[\mathbf{t}\right]_{WCS}$. This is called the end-effector

position, and is given by

$$\left[\mathbf{e}\right] \equiv \left[\mathbf{t}\right]_{WCS} \equiv \underbrace{\mathcal{T}_0 \circ \mathcal{T}_1 \circ \cdots \circ \mathcal{T}_N \circ \left[\mathbf{t}\right]_N}_{\equiv \mathbf{F}(\theta)}. \tag{2}$$

This defines the end effector function $\mathbf{F}(\boldsymbol{\theta})$ where

$$\theta = \begin{bmatrix}\theta_0 & \theta_1 & \ldots & \theta_K\end{bmatrix}^T \tag{3}$$

is termed the joint angles or joint parameters. The dimension K depends on the joint types. If the serial chain consists of ball-and-socket joints then $K = 3N$. If it only consists of revolute and/or prismatic joints then $K = N$. Generally speaking the contiguous sub-block k of $\boldsymbol{\theta}$ will parameterize the k^{th} joint and hence define the coordinate transformation $\mathcal{T}_k$. The explicit form depends on the joint type and chosen representation of rotations, quaternions, Euler angles, exponential maps etc. In general $\mathcal{T}_k$ is an affine map in $\mathbb{R}^3$ and its derivative with respect to its joint parameters is a linear map.

We now have enough notation and terminology in place to define what we understand of the unconstrained inverse kinematics problem.

Given a desired goal position $\mathbf{g} \in \mathbb{R}^3$, we seek to find the joint angles $\boldsymbol{\theta}^*$ that minimize the distance between the goal position and the end-effector position

$$\theta^* = \arg\min_{\theta} \underbrace{\frac{1}{2}\|\mathbf{g} - \mathbf{F}(\theta)\|^2}_{\equiv f(\theta)}, \tag{4}$$

which defines the IK objective function $f(\boldsymbol{\theta})$. Joint angle limits are added easily. For instance, using box constraints

$$\theta^* = \arg\min_{\mathbf{l} \le \theta \le \mathbf{u}} f(\theta) \tag{5}$$

where $\mathbf{l} < \mathbf{u}$ are lower and upper bounds on the joint angles. We note that the IK problems in (4) or (5) are non-linear problems and require specialized iterative numerical methods which utilize the gradient, and possibly the Hessian, of $f(\boldsymbol{\theta})$. One may apply Newton's method in a root search setting

$$\underbrace{\mathbf{g} - \mathbf{F}(\theta)}_{\equiv \mathbf{r}(\theta)} = \mathbf{0}. \tag{6}$$

This defines a residual vector $\mathbf{r}(\boldsymbol{\theta})$ that is related to the minimization form by $f(\theta) \equiv \frac{1}{2}\|\mathbf{r}(\theta)\|^2$. Solving the residual Eq. (6) using an iterative Newton method implies solving the Newton equation

$$\underbrace{\frac{\partial \mathbf{F}(\theta)}{\partial \theta}}_{\equiv \mathbf{J}} \Delta\theta = \mathbf{g} - \mathbf{F}(\theta), \tag{7a}$$

$$\mathbf{J}\,\Delta\theta = \mathbf{r}(\theta). \tag{7b}$$

A steepest descent method for solving the optimization problem in Eq. (4) iteratively updates the current iterate $\boldsymbol{\theta}^k$ with the gradient $\nabla f(\boldsymbol{\theta}^k)$ as the search direction and a step length α, to give the next iterate

$$\theta^{k+1} = \theta^k - \alpha \nabla f(\theta^k). \tag{8}$$

The gradient of the IK minimization formulation is given as

$$\nabla f(\theta) = -\frac{\partial \mathbf{F}}{\partial \theta}^T \mathbf{r}(\theta) = -\mathbf{J}^T \mathbf{r}. \tag{9}$$

From Eqs. (7b) and (9) it shows that an efficient method for computation the Jacobian matrix $\mathbf{J}$ is desired.

Iterative solvers based on Newton's method use second derivative information contained in the Hessian matrix $\mathbf{H}$ by solving the linear system

$$\mathbf{H}\,\mathbf{p} = -\nabla f(\theta), \tag{10}$$

which is the form used in our experiments. Here, the descent direction $\mathbf{p}$ is used to update the current solution by means of a line search with step size α, such that $\theta \leftarrow \theta + \alpha\mathbf{p}$. The (i, j) entry of the Hessian is given by

$$\mathbf{H}_{ij} = \frac{\partial^2 f}{\partial\theta_j \partial\theta_i}, \tag{11}$$

$$= \mathbf{J}_j^T \mathbf{J}_i - \left(\frac{\partial \mathbf{J}_i}{\partial \theta_j}\right)^T \mathbf{r}, \tag{12}$$

$$= \mathbf{J}_j^T \mathbf{J}_i - \mathbf{K}_{ij}^T \mathbf{r} \tag{13}$$

We use a subscript on $\mathbf{J}$ to denote respective columns of the matrix and define $\mathbf{K}_{ij} \equiv \frac{\partial \mathbf{J}_i}{\partial \theta_j}$. Here, $\mathbf{K}$ is in fact a third order tensor. If we used tensor algebra notation and double contraction operator : then this can be written compactly as

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} - \mathbf{K} : \mathbf{r}. \tag{14}$$

Previous work has argued to ignore the term $\mathbf{K}$: $\mathbf{r}$, leading to Gauss-Newton type methods or using iterative techniques to approximate $\mathbf{H}$, such as the popular BFGS method. In this work, we examine computing the exact Hessian.

3.1. Computing the Jacobian

Let us define the coordinate transformation concatenation as

$$\mathcal{X}_a^b \equiv \mathcal{T}_a \circ \mathcal{T}_{a+1} \circ \cdots \circ \mathcal{T}_{b-1} \circ \mathcal{T}_b \tag{15}$$

where $a \le b$. Using this notation we can write compactly the Jacobian with respect to the i^{th} joint parameter as

$$\frac{\partial \mathbf{F}(\theta)}{\partial \theta_i} = \mathcal{X}_0^{k-1} \circ \frac{\partial \mathcal{T}_k}{\partial \theta_i} \circ \mathcal{X}_{k+1}^N \circ \left[\mathbf{t}\right]_N \tag{16}$$

We assume implicitly that the k^{th} joint is the only joint that depends on θ_i. We keep k distinct from i to underline that joints can depend on one or more joint parameters. From (16) we realize

$$\left[\mathbf{t}\right]_k = \mathcal{X}_{k+1}^N \circ \left[\mathbf{t}\right]_N \tag{17}$$

is simply the tool vector in the k^{th} joint frame. The term $\frac{\partial \mathcal{T}_k}{\partial \theta_i}$ depends on the joint type. There are two fundamental different types of joints, revolute and prismatic. One can build more complex joint types by combining these two fundamental types. In their most simple form we have

- For a revolute joint $\mathcal{T}_k$ is a rotation $\mathbf{R}_k \equiv \mathbf{R}(\theta_i, \mathbf{a}_k)$ around a fixed joint axis $\mathbf{a}_k$. That is $\mathcal{T}_k \equiv \mathbf{R}_k$. In many systems the revolute joint origin can be displaced relatively to the parent frame in which case one often have $\mathcal{T}_k \equiv \mathbf{T}(\mathbf{l}_k) \circ \mathbf{R}_k$ where $\mathbf{l}_k$ is a constant link vector.
- For a prismatic joint the transformation $\mathcal{T}_k$ is a translation of $\mathbf{T}_k \equiv \mathbf{T}(\theta_i \mathbf{a}_k)$ along the known fixed unit joint axis $\mathbf{a}_k$. Again in many systems one can locally orient the joint axis with respect to the parent frame by applying a constant known rotation. That is $\mathcal{T}_k \equiv \mathbf{T}_k \circ \mathbf{R}$.

The derivatives of applying these fundamental transformation types to any vector $\mathbf{p}$ is given by

$$\frac{\partial \mathcal{T}_k}{\partial \theta_i} \circ \mathbf{p} = \begin{cases} \mathbf{a}_k \times \mathbf{R}_k \circ \mathbf{p} & \text{if revolute} \\ \mathbf{a}_k & \text{if prismatic} \end{cases} \tag{18}$$

Notice the subtle effect of the revolute joint. The translation of the origin is lost. Hence $\mathbf{p}$ is rotated into the parent frame before taking the cross product. Combining this with the observation in (17) we have

$$\mathbf{J}_i = \frac{\partial \mathbf{F}(\theta)}{\partial \theta_i} \tag{19}$$

Table 1
Jacobian formulas for various fundamental joint types. It is assumed that $\mathcal{T}_k$ depends on θ_i.

k	Jacobian formula
Revolute	$\mathbf{J}_i = \mathbf{a}_{k,WCS} \times \Delta\mathbf{p}_{k,WCS}$
Prismatic	$\mathbf{J}_i = \mathbf{a}_{k,WCS}$

Table 2
Tensor formulas for various combinations of fundamental joint types. It is assumed that $\mathcal{T}_h$ depends on θ_j and $\mathcal{T}_k$ depends on θ_i.

h	k	Tensor formula
Revolute	Revolute	$\mathbf{K}_{i,j} = \mathbf{a}_{h,WCS} \times \mathbf{a}_{k,WCS} \times \Delta\mathbf{p}_{k,WCS}$
Revolute	Prismatic	$\mathbf{K}_{i,j} = \mathbf{a}_{h,WCS} \times \mathbf{a}_{k,WCS}$
Prismatic	Revolute	$\mathbf{K}_{i,j} = \mathbf{0}$
Prismatic	Prismatic	$\mathbf{K}_{i,j} = \mathbf{0}$

$$= \mathcal{X}_0^{k-1} \circ \frac{\partial \mathcal{T}_k}{\partial \theta_i} \circ \left[\mathbf{t}\right]_k \tag{20}$$

$$= \begin{cases} \mathbf{a}_{k,WCS} \times \Delta\mathbf{p}_{k,WCS} & \text{if revolute joint} \\ \mathbf{a}_{k,WCS} & \text{if prismatic joint} \end{cases} \tag{21}$$

where $\Delta\mathbf{p}_{k,WCS} \equiv \left[\mathbf{e}\right]_{WCS} - \left[\mathbf{o}_k\right]_{WCS}$ is the difference between the end-effector position and the position of the origin of the k^{th} joint frame $\mathbf{o}_k$ and $\mathbf{a}_{k,WCS} = \left[\mathbf{a}_k\right]_{WCS}$. Later we treat ball-and-socket joints in more details in Section 4. The Jacobian formulas are summarized in Table 1.

3.2. Computing the Hessian

From (11) we seek an efficient way to compute the third order tensor $\mathbf{K}$ in order to compute the Hessian. The $\mathbf{K}$-tensor is defined as

$$\mathbf{K}_{i,j} = \frac{\partial}{\partial \theta_j}\left(\frac{\partial \mathbf{F}}{\partial \theta_i}\right), \tag{22a}$$

$$= \mathcal{X}_0^{h-1} \circ \frac{\partial \mathcal{T}_h}{\partial \theta_j} \circ \mathcal{X}_{h+1}^{k-1} \circ \frac{\partial \mathcal{T}_k}{\partial \theta_i} \circ \mathcal{X}_{k+1}^{N} \circ \left[\mathbf{t}\right]_N, \tag{22b}$$

where we assume $\mathcal{T}_k$ depends on joint parameter θ_i and $\mathcal{T}_h$ on θ_j. Again notation is kept to make it explicit that coordinate transforms can depend on more than a single joint parameter. We notice (22) is simply a recursive application of the transformation rules we used for computing the Jacobian in the first place. Table 2 summarizes the formulas for computing $\mathbf{K}_{i,j}$. We will now derive these in more details.

If we assume k and h to both be revolute joints then the first derivative will change the position into a vector direction and apply a cross-product with the instantaneous joint axis from bone k. The second derivative will add a cross-product with the instantaneous joint axis from bone h. Hence, we may now immediately write up a world-coordinate system equation for computing $\mathbf{K}_{i,j}$. Finally, we have

$$\mathbf{K}_{i,j} = \mathbf{a}_{h,WCS} \times \mathbf{a}_{k,WCS} \times \Delta\mathbf{p}_{k,WCS}, \tag{23a}$$

$$= \mathbf{a}_{h,WCS} \times \mathbf{J}_i, \tag{23b}$$

If k and h are both prismatic then we have $\mathbf{K}_{i,j} = \mathbf{0}$. To prove this we will use homogenous coordinates and write

$$\frac{\partial \mathcal{T}_k}{\partial \theta_i} = \begin{bmatrix} \mathbf{0} & \mathbf{a}_k \\ \mathbf{0}^T & 0 \end{bmatrix} \quad \text{and} \quad \frac{\partial \mathcal{T}_h}{\partial \theta_j} = \begin{bmatrix} \mathbf{0} & \mathbf{a}_h \\ \mathbf{0}^T & 0 \end{bmatrix}. \tag{24}$$

By (22)

$$\mathbf{K}_{i,j} = \mathbf{X}_0^{h-1}\begin{bmatrix} \mathbf{0} & \mathbf{a}_h \\ \mathbf{0}^T & 0 \end{bmatrix}\mathbf{X}_{h+1}^{k-1}\begin{bmatrix} \mathbf{0} & \mathbf{a}_k \\ \mathbf{0}^T & 0 \end{bmatrix}\mathbf{X}_{k+1}^{N}\begin{bmatrix} \mathbf{t} \\ 1 \end{bmatrix}, \tag{25}$$

$$= \mathbf{X}_0^{h-1}\begin{bmatrix} \mathbf{0} & \mathbf{a}_h \\ \mathbf{0}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{a}'_k \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix} \tag{26}$$

where $\mathbf{a}'_h = \mathbf{X}_{h+1}^{k-1} \circ \mathbf{a}_k$ and we use concatenation definition loosely as either a homogenous matrix and as a abstract notation of transformation.

If h is revolute and k is prismatic then we have $\mathbf{K}_{i,j} = \mathbf{a}_{h,WCS} \times \mathbf{a}_{k,WCS}$. The proof follows easily using homogenous coordinates and substituting into (22).

$$\frac{\partial \mathcal{T}_h}{\partial \theta_j} = \begin{bmatrix} C(\mathbf{a}_h)\mathbf{R}_h & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} \quad \text{and} \quad \frac{\partial \mathcal{T}_k}{\partial \theta_i} = \begin{bmatrix} \mathbf{0} & \mathbf{a}_k \\ \mathbf{0}^T & 0 \end{bmatrix} \tag{27}$$

where $C(\mathbf{a}_k)$ denotes the skew symmetric matrix such that $C(\mathbf{a}_k)\,\mathbf{p} \equiv \mathbf{a}_k \times \mathbf{p}$ for any $\mathbf{p}$.

$$\mathbf{K}_{i,j} = \mathbf{X}_0^{h-1}\begin{bmatrix} C(\mathbf{a}_h)\,\mathbf{R}_h & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}\mathbf{X}_{h+1}^{k-1}\begin{bmatrix} \mathbf{0} & \mathbf{a}_k \\ \mathbf{0}^T & 0 \end{bmatrix}\mathbf{X}_{k+1}^{N}\begin{bmatrix} \mathbf{t} \\ 1 \end{bmatrix}, \tag{28}$$

$$= \mathbf{X}_0^{h-1}\begin{bmatrix} C(\mathbf{a}_h)\,\mathbf{R}_h & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{a}'_k \\ 0 \end{bmatrix}, \tag{29}$$

$$= \begin{bmatrix} \mathbf{X}_0^{h-1} \circ C(\mathbf{a}_h)\,\mathbf{R}_h\,\mathbf{a}'_k \\ 0 \end{bmatrix}, \tag{30}$$

$$= \begin{bmatrix} \mathbf{a}_{h,WCS} \times \mathbf{a}_{k,WCS} \\ 0 \end{bmatrix}. \tag{31}$$

From this proof it is trivial to see that if we reverse the roles of the revolute and prismatic joints then $\mathbf{K}_{i,j} = \mathbf{0}$.

3.3. Adding a moving root bone

For character animation it may be desirable to have support for a moving root bone. In principle one can make the character walk and jump simply by controlling the end-effectors. Here we outline how to extend the joint angle parameter vector to include the translational parameters of the root bone. First we extend the joint parameter vector $\boldsymbol{\theta}$ with $\mathbf{t}_o = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T$ to give us the extended version $\boldsymbol{\theta}'$

$$\theta' \equiv \begin{bmatrix} t_x & t_y & t_z & \theta^T \end{bmatrix}^T. \tag{32}$$

The root bone has the nice property that the bone vector lives in the world coordinate system and will be independent of any rotational degree of freedom in the system. Hence, the first three columns of the IK Jacobian will be

$$\frac{\partial \mathbf{F}(\theta')}{\partial \mathbf{t}_x} = \mathbf{i}, \tag{33a}$$

$$\frac{\partial \mathbf{F}(\theta')}{\partial \mathbf{t}_y} = \mathbf{j}, \tag{33b}$$

$$\frac{\partial \mathbf{F}(\theta')}{\partial \mathbf{t}_z} = \mathbf{k}. \tag{33c}$$

Further, from Table 2 we have

$$\mathbf{K}_{ij} = \mathbf{0} \quad \text{if } i \in \{0,1,2\} \text{ or } j \in \{0,1,2\} \tag{34}$$

Hence, we can add the extension trivially to the unmodified $\mathbf{J}$ and $\mathbf{H}$ to get the root-translation extensions $\mathbf{J}'$ and $\mathbf{H}'$, by letting $\mathbf{I}_{3\times 3} \equiv \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \end{bmatrix}$ and define

$$\mathbf{J}' \equiv \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{J} \end{bmatrix}, \tag{35a}$$

$$\mathbf{H}' \equiv \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix}. \tag{35b}$$

4. Ball and socket joints

A ball and socket joint is special in the way that it does not have a natural defined rotation axis. One may parametrize it using revolute joints. Euler angles is one such convention where one conceptually can think of a ball-and-socket joint as three consecutive revolute joints sharing the same origin. The benefit of Euler angles is that one can straightforwardly apply our previous results. The drawback is that such parameterizations may suffer from Gimbal locks causing Hessians to become singular. An alternative to Euler angles that is suggested for particular shoulder joints is to an exponential map for the stance motion of the joint. By careful design one can place the singularity of the exponential maps at locations such that gimbal lock is unlikely to occur for human shoulder joint. Thereby limit the number of cases singular Hessians is encountered.

4.1. Using Euler angles

In this section, we show how $\mathcal{T}_k$ is parameterized using the Euler angles α_k, β_k, and γ_k. Homogeneous coordinates are used for deriving the equations, and when presenting our final results we implicitly homogenize our formulas. The theory is generally applicable to any chosen representation of rotations. Our choice here is motivated by the goal to make the derivations more easy accessible to a larger audience.

The i^{th} index of $\boldsymbol{\theta}$ can be either the α, β or γ angle of the k^{th} joint. That is,

$$\theta_i \in \{\alpha_k, \beta_k, \gamma_k\}. \tag{36}$$

Let us consider a single IK joint, and we label this the k^{th} joint. This joint is equivalent to a rigid body transformation, $\mathcal{T}_k$, such that

$$\begin{bmatrix}\mathbf{p}\\1\end{bmatrix}_{k-1} = \mathcal{T}_k \begin{bmatrix}\mathbf{p}\\1\end{bmatrix}_k = \underbrace{\begin{bmatrix}\mathbf{R}_k(\alpha_k,\beta_k,\gamma_k) & \mathbf{t}_k\\ \mathbf{0}^T & 1\end{bmatrix}}_{\equiv \mathcal{T}_k} \begin{bmatrix}\mathbf{p}\\1\end{bmatrix}_k. \tag{37}$$

This transforms a vector $\mathbf{p}$ from the k^{th} joint frame into the $(k-1)^{\text{th}}$ joint frame. The rigid body transformation consists of a rotation $\mathbf{R}_k(\alpha_k, \beta_k, \gamma_k)$ that we parametrize using the joint angles α_k, β_k, and γ_k, and a joint vector $\mathbf{t}_k$ (also called the link vector). Let *ABC* denote the Euler angle convention which defines the chosen axes of rotation. We adopt a *ZYZ* Euler angle convention implying $ABC = ZYZ$, which gives the rotation sequence

$$\mathbf{R}_k \equiv \mathbf{R}_A(\alpha)\,\mathbf{R}_B(\beta)\,\mathbf{R}_C(\gamma). \tag{38}$$

We label the respective local rotation axes $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and derive the Jacobian due to each angle as

$$\frac{\partial \mathbf{F}(\theta)}{\partial \alpha_k} = \mathbf{X}_0^{k-1} \frac{\partial \mathbf{T}_k}{\partial \alpha_k} \begin{bmatrix}\mathbf{t}\\1\end{bmatrix}_k = \mathbf{X}_0^{k-1} \begin{bmatrix}\mathbf{a}\times\Delta\mathbf{p}\\0\end{bmatrix} \tag{39a}$$

$$\frac{\partial \mathbf{F}(\theta)}{\partial \beta_k} = \mathbf{X}_0^{k-1} \frac{\partial \mathbf{T}_k}{\partial \beta_k} \begin{bmatrix}\mathbf{t}\\1\end{bmatrix}_k = \mathbf{X}_0^{k-1} \begin{bmatrix}\mathbf{b}'\times\Delta\mathbf{p}\\0\end{bmatrix} \tag{39b}$$

$$\frac{\partial \mathbf{F}(\theta)}{\partial \gamma_k} = \mathbf{X}_0^{k-1} \frac{\partial \mathbf{T}_k}{\partial \gamma_k} \begin{bmatrix}\mathbf{t}\\1\end{bmatrix}_k = \mathbf{X}_0^{k-1} \begin{bmatrix}\mathbf{c}'\times\Delta\mathbf{p}\\0\end{bmatrix} \tag{39c}$$

where $\Delta\mathbf{p}$ is the tool vector with respect to the k^{th} joint frame and $\mathbf{a}$, $\mathbf{b}'$ and $\mathbf{c}'$ are the instantaneous joint axes of our ABC Euler rotation expressed in the $(k-1)^{\text{th}}$ frame. We notice that the homogeneous coordinates in all above equations have become zero. Hence the effect of $\mathbf{X}_0^{k-1}$ is to rotate the directional cross-product vector from frame $(k-1)$ into the world coordinate frame. This important observation allows us to make the shortcut of simply computing the cross product using world coordinate system quantities. Using this observation, and dropping homogeneous coordinates, we finally obtain

$$\frac{\partial \mathbf{F}(\theta)}{\partial \alpha_k} = \big[\mathbf{a}\big]_{WCS} \times \Delta\mathbf{p}_{WCS}\,, \tag{40a}$$

$$\frac{\partial \mathbf{F}(\theta)}{\partial \beta_k} = \big[\mathbf{R}_A(\alpha_k)\mathbf{b}\big]_{WCS} \times \Delta\mathbf{p}_{WCS}\,, \tag{40b}$$

$$\frac{\partial \mathbf{F}(\theta)}{\partial \gamma_k} = \big[\mathbf{R}_A(\alpha_k)\mathbf{R}_B(\beta_k)\mathbf{c}\big]_{WCS} \times \Delta\mathbf{p}_{WCS} \tag{40c}$$

where $\Delta\mathbf{p}_{WCS} = \big(\mathbf{e} - \big[\mathbf{o}_k\big]_{WCS}\big)$ is the vector difference between the end-effector position and the position of the joint origin in the world coordinate system.

Let the instantaneous joint axis of bone k be given by

$$\mathbf{v} = \begin{cases}\mathbf{a} & \text{if } \theta_i = \alpha_k\\ \mathbf{R}_A(\alpha_k)\mathbf{b} & \text{if } \theta_i = \beta_k\\ \mathbf{R}_A(\alpha_k)\mathbf{R}_B(\beta_k)\mathbf{c} & \text{if } \theta_i = \gamma_k\\ \mathbf{0} & \text{otherwise}\end{cases} \tag{41}$$

and similarly for bone h by

$$\mathbf{w} = \begin{cases}\mathbf{a} & \text{if } \theta_j = \alpha_h\\ \mathbf{R}_A(\alpha_h)\mathbf{b} & \text{if } \theta_j = \beta_h\\ \mathbf{R}_A(\alpha_h)\mathbf{R}_B(\beta_h)\mathbf{c} & \text{if } \theta_j = \gamma_h\\ \mathbf{0} & \text{otherwise}\end{cases}. \tag{42}$$

Finally, we have

$$\mathbf{K}_{i,j} = \mathbf{w}_{WCS} \times \mathbf{v}_{WCS} \times \Delta\mathbf{p}_{WCS} \tag{43a}$$

$$= \mathbf{w}_{WCS} \times \mathbf{J}_i\,, \tag{43b}$$

where $\mathbf{v}_{WCS} = \mathbf{R}^k_{WCS}\mathbf{v}$ and $\mathbf{w}_{WCS} = \mathbf{R}^h_{WCS}\mathbf{w}$ are the instantaneous joint axes transformed in the coordinates of the world coordinate system.

4.2. Using exponential map

From [25] the exponential map $e^{\mathbf{w}}$ is defined as a mapping from $\mathbb{R}^3 \mapsto SO(3)$. Efficient and robust methods for computing the exponential map and its derivative $\frac{\partial \mathbf{R}}{\partial \mathbf{w}}$ are described in [25]. For a ball and socket joint the twist is handled as a single revolute joint and the swing rotation is represented by

$$\mathbf{w} \equiv \alpha\hat{s} + \beta\hat{t} \tag{44}$$

where $\hat{s}$ and $\hat{t}$ are any two orthogonal unit vectors in the perpendicular plane to the major axis of the limb. Let us denote the swing rotation by the coordinate transform $\mathcal{T}_k$ then from [25] we find

$$\frac{\partial \mathcal{T}_k}{\partial \alpha} = \begin{bmatrix}\frac{\partial \mathbf{R}_k}{\partial \mathbf{w}}\hat{s} & \mathbf{0}\\ \mathbf{0}^T & 0\end{bmatrix} \tag{45a}$$

$$\frac{\partial \mathcal{T}_k}{\partial \beta} = \begin{bmatrix}\frac{\partial \mathbf{R}_k}{\partial \mathbf{w}}\hat{t} & \mathbf{0}\\ \mathbf{0}^T & 0\end{bmatrix} \tag{45b}$$

where $\mathbf{R}_k$ is the rotation matrix of $e^{\mathbf{w}}$. This can be used directly to evaluate (19) and (22). Assuming the k^{th} joint uses the exponential map parameterization and that θ_i is one of the stance parameters then we can apply parentheses to show the order of evaluation.

$$\mathbf{K}_{i,j} = \mathcal{X}_0^{h-1} \circ \left(\frac{\partial \mathcal{T}_h}{\partial \theta_j} \circ \left(\mathcal{X}_{h+1}^{k-1} \circ \left(\frac{\partial \mathcal{T}_k}{\partial \theta_i} \circ \big[\mathbf{t}\big]_k\right)\right)\right), \tag{46a}$$

This demonstrates that it is still trivial to evaluate the derivatives. However, there is no geometric interpretation allowing us to directly compute the derivatives in the world coordinate setting, as we did for the revolute joints.

5. Implementation

A tree branched IK structure essentially consists of several serial chains that all share the same root and may have overlapping "trunks" of the tree structure in common.

Given such an IK structure, serial chains are easily created by traversal over the bones as shown in Algorithm 1. This should be run once at initialization, or whenever the tree-structure changes its connectivity or becomes re-rooted.

Algorithm 1: MAKECHAINS: Chains of a tree-branched kinematic structure are generated by iterating over all bones to identify end-effector bones and then followed by back-traversal from each end-effector to the single root bone of the tree structure. Running time is $\mathcal{O}(N + LH)$ where N is number of bones in structure, $L \ll N$ is the number of leaves/chains, and $H < N$ is the height of the tree-structure. Hence, we have $\mathcal{O}(N)$.

Data: $\mathcal{S}$: All bones in a tree IK structure
Result: $\mathcal{C}$: Set of all IK chains of the IK structure

$\mathcal{L} \leftarrow$ empty set of bones;
foreach $\mathcal{B} \in \mathcal{S}$ **do**
  if $\mathcal{B}$ *has no children* **then**
    add B to $\mathcal{L}$;
  end
end
$\mathcal{C} \leftarrow$ empty set of chains;
foreach $\mathcal{B} \in \mathcal{L}$ **do**
  $H \leftarrow$ empty chain;
  while $\mathcal{B}$ *exist* **do**
    add $\mathcal{B}$ to front of H;
    $\mathcal{B} \leftarrow$ Parent bone of $\mathcal{B}$;
  end
  add H to $\mathcal{C}$;
end

A recursive traversal can be used to perform forward kinematics and compute the world orientations and positions of all bones, as shown in Algorithm 2. It should be invoked prior to any Jacobian or Hessian computation. The Jacobian computation is outlined in Algorithm 3, and the Hessian computation in Algorithm 4.

Algorithm 2: FORWARDKINEMATICSUPDATE: Recursive forward sweep through a hierarchical kinematics structure quickly updates all bones with their current orientation $\mathbf{R}$ and position $\mathbf{t}$ of joint frames in the WCS. Running time is $\mathcal{O}(N)$, where N is number of bones in the structure.

Data: $\mathcal{B}_k$: Current bone

$(\alpha, \beta, \gamma) \leftarrow$ current Euler angles of $\mathcal{B}_k$;
$\mathbf{R}_k \leftarrow \mathbf{R}_Z(\alpha)\,\mathbf{R}_Y(\beta)\,\mathbf{R}_Z(\gamma)$;
$\mathbf{t}_k \leftarrow$ local position of $\mathcal{B}_p$;
$\mathbf{R}_p \leftarrow$ identity;
$\mathbf{t}_p \leftarrow \mathbf{0}$;
if *Parent bone of* $\mathcal{B}_k$ *exist* **then**
  $\mathcal{B}_p \leftarrow$ Parent bone of $\mathcal{B}_k$;
  $\mathbf{R}_p \leftarrow$ world orientation of $\mathcal{B}_p$;
  $\mathbf{t}_p \leftarrow$ world position of $\mathcal{B}_p$;
end
world orientation of $\mathcal{B}_k \leftarrow \mathbf{R}_p\,\mathbf{R}_k$;
world position of $\mathcal{B}_k \leftarrow \mathbf{t}_p + \mathbf{R}_p\,\mathbf{t}_k$;
foreach $\mathcal{B}_c \in$ *children of* $\mathcal{B}_k$ **do**
  update($\mathcal{B}_c$);
end

Algorithm 3: COMPUTEJACOBIAN: The numerical method for computing the IK Jacobian of a tree-branched kinematic structure. We use blocked indexing into $\mathbf{J}$ which we consider to consist of $\mathbf{R}^3$ blocks. The running time is $\mathcal{O}(NL)$ with N being the total number of bones and $L \ll N$ the number of leaves/chains. Hence, we may argue for an overall $\mathcal{O}(N)$. Tool-vectors are ignored in this algorithm, but line 7 can be modified to include non-zero tool-vectors if needed.

Data: $\mathcal{C}$: set of chains, $\mathcal{S}$: All bones in a tree IK structure
Result: $\mathbf{J}$: The IK Jacobian

1 FORWARDKINEMATICSUPDATE(root bone of $\mathcal{S}$);
2 $\mathcal{C} \leftarrow$ MAKECHAINS($\mathcal{S}$);
3 $E \leftarrow$ number of chains;
4 $N \leftarrow$ number of bones;
5 $\mathbf{J} \leftarrow$ zero matrix of $3E \times 3N$;
6 **foreach** $H_i \in \mathcal{C}$ **do**
7 $\mathbf{e} \leftarrow$ world position of end-effector bone in H_i;
8 **foreach** $\mathcal{B}_k \in \mathcal{H}$ **do**
9 $\mathbf{R}_p \leftarrow$ identity;
10 **if** *Parent bone of* $\mathcal{B}_k$ *exist* **then**
11 $\mathbf{R}_p \leftarrow$ world orientation of parent boneof $\mathcal{B}_k$;
12 **end**
13 $(\alpha, \beta, \gamma) \leftarrow$ current Euler angles of $\mathcal{B}_k$;
14 $\Delta\mathbf{p} \leftarrow \mathbf{e} -$ world position of $\mathcal{B}_k$;
15 $\mathbf{u} \leftarrow \mathbf{R}_p\,\mathbf{k}$;
16 $\mathbf{v} \leftarrow \mathbf{R}_p\,\mathbf{R}_Z(\alpha)\,\mathbf{j}$;
17 $\mathbf{w} \leftarrow \mathbf{R}_p\,\mathbf{R}_Z(\alpha)\,\mathbf{R}_Y(\beta)\,\mathbf{k}$;
18 $\mathbf{J}_{i,3k} \leftarrow \mathbf{u} \times \Delta\mathbf{p}$;
19 $\mathbf{J}_{i,3k+1} \leftarrow \mathbf{v} \times \Delta\mathbf{p}$;
20 $\mathbf{J}_{i,3k+3} \leftarrow \mathbf{w} \times \Delta\mathbf{p}$;
21 **end**
22 **end**

Observe that in order to get the IK gradient one must use the result of the Jacobian in (9).

6. Results

This section presents comparisons of tracking real motion data using quasi-Newton and exact Hessian-based methods. All experiments were performed on a Windows PC with Intel i7 2.80 GHz processor and 32 GB of memory.

A MATLAB implementation of the objective function, and Jacobian and Hessian computations is used in our evaluation. The exact Newton method uses the interior-reflective trust region algorithm [26], and the quasi-Newton method used the BFGS algorithm [27,28]. The non-linear optimization is performed using the built-in *fminunc* function, or *fmincon* in the case of a constrained optimization. We compare our results for unconstrained optimization with the LM algorithm, which uses the Hessian approximation $\mathbf{J}^T\mathbf{J}$ and is equivalent to the damped least squares method proposed in Buss [4].

6.1. Motion capture data

We evaluate the performance of solving IK problems, with and without the exact Hessian, using motion examples from the HDM05 database [29]. This database provides skeletal motion and optical marker trajectories for a variety of motion classes. Examples of motions contained in this database are shown in Figs. 2 and 6.

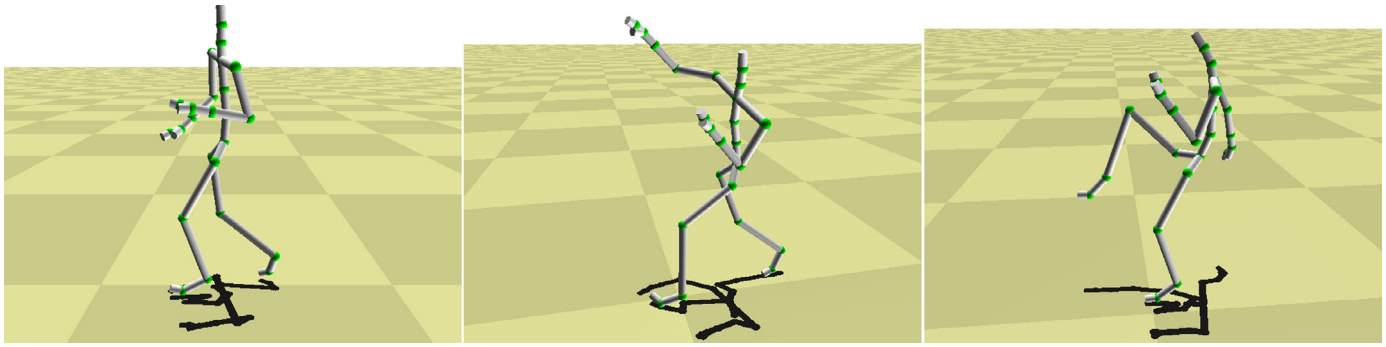

Fig. 2. Exemplary frames of motion data used in our comparison: slow walking (left), punching (middle), and kicking (right). Convergence plots for these frames (and more) are shown in Fig. 3–5.

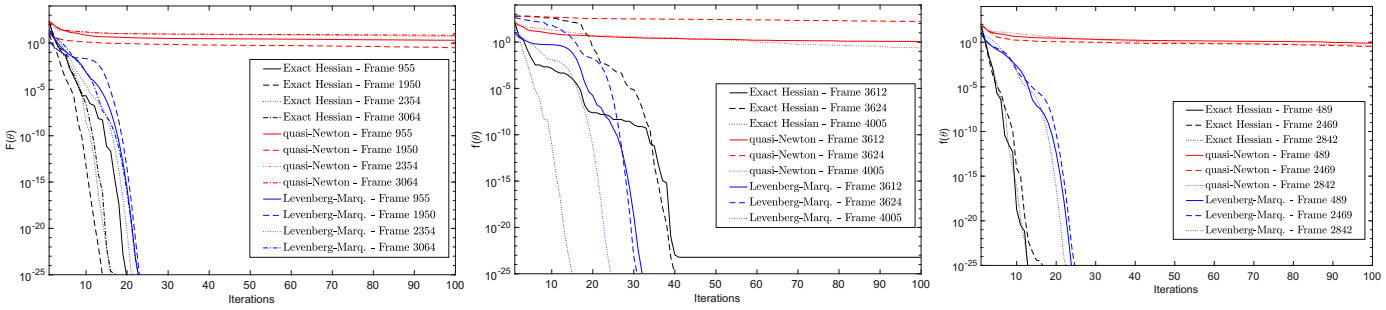

Fig. 3. Convergence of the unconstrained optimization in (4). The vertical axis shows the residual, $f(\boldsymbol{\theta})$. Frames are selected from the *dg_03-02_01* motion in the HDM05 database. Three types of motion are used: walking (left), punching (middle), and kicking (right).

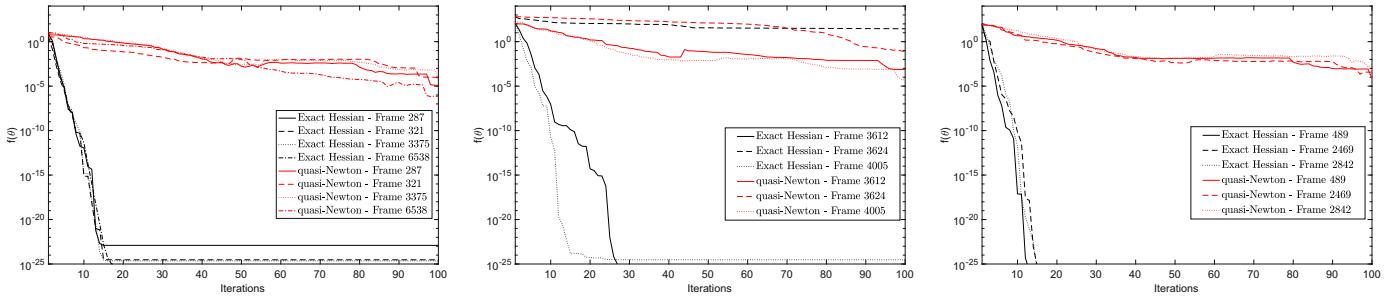

Fig. 4. Convergence of the constrained optimization in (5). The vertical axis shows the residual, $f(\boldsymbol{\theta})$. Frames are selected from the *dg_03-02_01* motion in the HDM05 database. Three types of motion are used: walking (left), punching (middle), and kicking (right). Results for the LM algorithm are excluded since the implementation does not handle bounds.

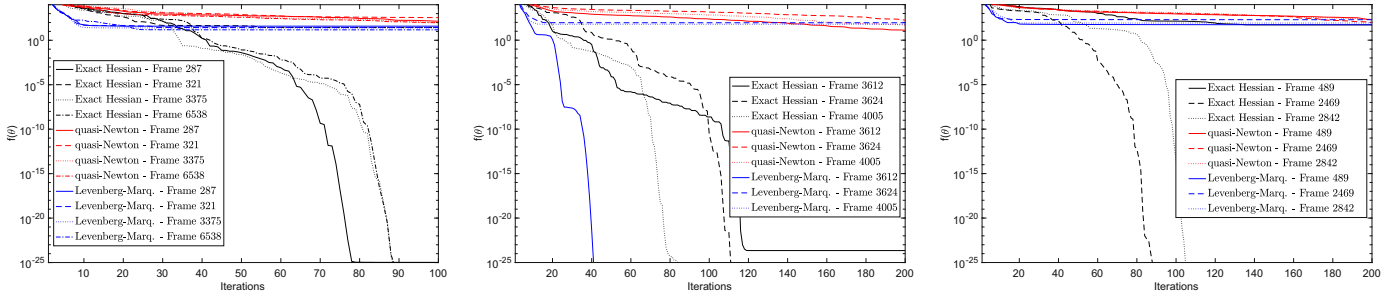

Fig. 5. Convergence of the unconstrained optimization when initialized with $\theta = \mathbf{0}$. Both methods demonstrate slower convergence compared to initializing with the joint angles from the previous frames. In particular, the exact Newton method demonstrates linear convergence early on. Note that $k_{max} = 200$ was used to generate some of these plots since more iterations were required to observe convergence behavior.

A total of 41 optical marker trajectories sampled at 120 Hz are used for tracking. We solve for the motion of a skeleton with 58 degrees of freedom (DOF). The skeleton initialization in Algorithm 1 and the forward kinematic update in Algorithm 2 are done using code provided with the database.

We evaluate the methods in our experiments using a variety of motion styles. In addition to sequences involving walking and simple locomotion, we include fast moving motions (punching, kicking) and agile movements (cartwheels, exercises, jumping) since these are likely to be more challenging for cases where the posture at the previous frame is used to initialize the solution. Furthermore, we include motions where one or limbs are nearly straightened, since these are postures that are likely to introduce kinematic singularities. In all experiments we use a

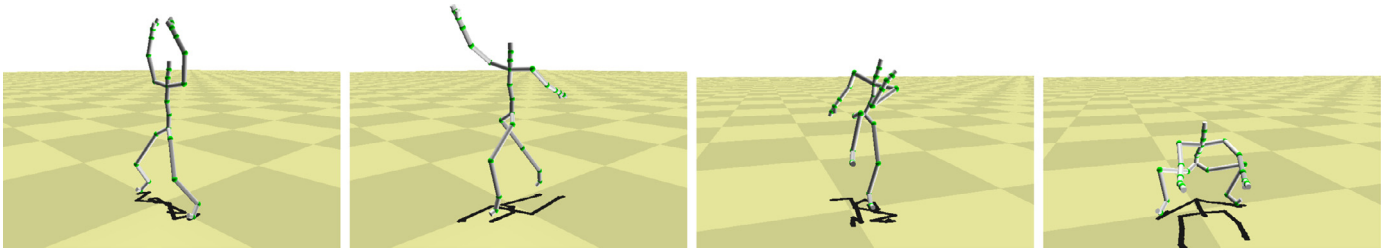

Fig. 6. Additional frames of motion data from the *mm_03-05_02* sequence used in our convergence comparison. Left to right: jumping jacks, skiing exercise, elbow-to-knee, and squat. Convergence plots for these frames (and more) are shown in Fig. 7.

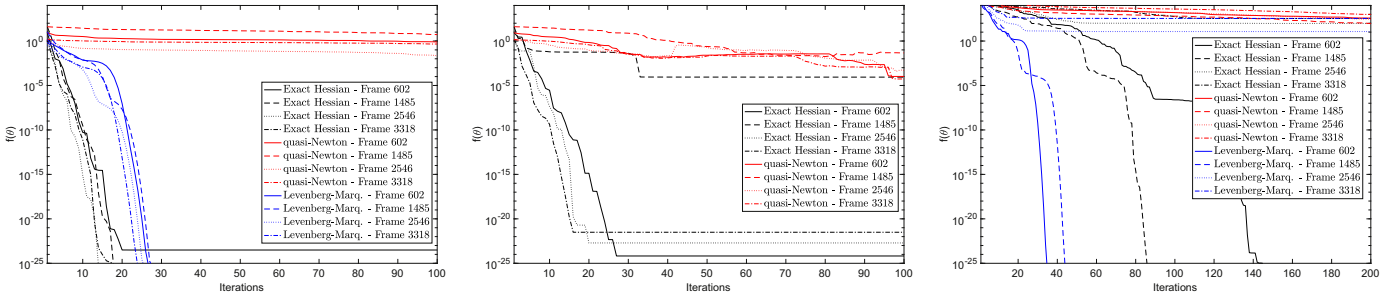

Fig. 7. Convergence for frames selected from the *mm_03-05_02* motion in the HDM05 database. The vertical axis shows the residual of the optimization problem, $f(\boldsymbol{\theta})$. From left to right: unconstrained optimization, constrained optimization, and the unconstrained optimization when initialized with $\theta = \mathbf{0}$.

moving root bone which is numerically the same as using 3 prismatic joints.

6.2. Unconstrained optimization

Fig. 3 shows the convergence when performing the unconstrained optimization in (4) for selected frames of motion. The frames contain three types of motion: walking, kicking, and punching. In each case, the joint angles and root position of the previous frame of motion were used as the initial solution to the IK solver in order to give a reasonable initial estimate for both the gradient and Hessian since we found that both methods exhibited linear or sub-linear convergence when initialized with a configuration dissimilar to the current frame.

The average timings to solve for the examples shown in Fig. 3 using $k_{max} = 100$ are 2.6 s for the quasi-Newton BFGS solver and 19.2 s for trust region with an exact Hessian. In the latter case, the longer timing is due to the overhead of computing the matrix of second-order partial derivatives. However, using a stopping tolerance $f(\theta) < 10^{-2}$, we observe the average timings shown in Table 3. For Newton's method based algorithms it's clear that the exact Hessian gives a much lower residual with fewer iterations and less wall-clock time. However, the LM algorithm clearly gives the fastest performance when a good initial solution is provided.

6.3. Constrained optimization

Fig. 4 shows the convergence when performing the constrained optimization in (5), which accounts for joint limits. Compared to the unconstrained convergence shown in Fig. 3, the constrained optimizer seems to give slightly better performance in some instances, which is surprising. We hypothesize that the box constraints can sometimes help to avoid solutions where indefiniteness or singularities may occur, and this is a point for future investigation.

6.4. Initial solution

Here we set $\theta = \mathbf{0}$ as the initial solution and re-solve for the motion frames used in our previous analysis. This allows us to evaluate the performance of each method when given a naive or poor initial solution. As shown by Fig. 5, the convergence across all of the tested methods becomes worse. Specifically, the trust region method with exact Hessian exhibits linear, or even sub-linear, convergence early on. The quasi-Newton and LM algorithms are also unable to achieve a low error solution in many of the example frames. The LM algorithm tends to get stuck in a local minimum when not initialized with a good solution.

The trust region with exact Hessian was the only method to consistently reach low error residual in instances of poor initial solution. We note that as progress is made towards a solution, the expected quadratic convergence behavior is observed.

6.5. Additional examples

We perform convergence analysis for specific frames from an additional motion sequence shown in Fig. 6, which involves various exercise motions during a workout. The convergence plots shown in Fig. 7 demonstrate similar behavior to the walking, punching, and kicking examples for each of the unconstrained, constrained, and zero initialized cases. However, we do note that in these examples when $\theta = \mathbf{0}$ is used to initialize the frames, the LM algorithm demonstrates better convergence.

6.6. Reconstructing motion sequences

We evaluate the IK solvers when reconstructing a complete motion sequence contained more than several thousand frames. The average timing and average error per frame can be found in Table 4, where the error is measured as the sum of the difference between the real and reconstructed marker positions (in cm). A video of the reconstructed motions, with side-by-side comparison to the original motion from the HDM05 database, can be found in the supplementary material. During the reconstruction, each frame is initialized with the solution of the previous frame as generated by the specific solver method, except for the initial frame when the solution is initialized as $\theta = \mathbf{0}$.

The exact Newton solver with $k_{max} = 10$ gives the lowest error when fitting to marker data, and qualitatively the best

Algorithm 4: ComputHessian: The numerical method for computing the Hessian of a tree branched IK structure. The algorithm only fills in the upper triangular part of the Hessian **H**-matrix. Notice that the tool vectors are assumed to be the zero vectors in line 3. The running time is $\mathcal{O}(N^2L + L^2)$ where N is the total number of bones and $L < N$ is the number of leaves/chains in the structure. Overall this will be $\mathcal{O}(N^2)$.

```
   Data: C: set of chains, J: Jacobian matrix
   Result: H: The IK Hessian matrix
1  H ← symmetric matrix J^T J;
2  foreach H_c ∈ C do
3      e ← world position of end-effector bone in H_c;
4      r ← goal of H_c - e ;
5      foreach B_k ∈ H do
6          foreach B_h ∈ H with h ≤ k do
7              R_p ← identity;
8              if Parent bone of B_h exist then
9                  R_p ← world orientation of parent of B_h;
10             end
11             (α, β, γ) ← current Euler angles of B_h;
12             u ← R_p k;
13             v ← R_p R_Z(α) j;
14             w ← R_p R_Z(α) R_Y(β) k;
15             foreach i ∈ (3k, 3k + 2) do
16                 H_{i,3h} ← H_{i,3h} − (u × J_i)^T r;
17                 H_{i,3h+1} ← H_{i,3h+1} − (v × J_i)^T r;
18                 H_{i,3h+2} ← H_{i,3h+2} − (w × J_i)^T r;
19             end
20         end
21     end
22 end
```

Table 3
Average computational time / function calls per frame for the examples in Fig. 3 with stopping criterion $f(\theta) < 10^{-2}$.

Type	quasi-Newton	exact Hessian	LM
Walking	5.30 s/213 calls	0.59 s/3.7 calls	0.25 s/7.8 calls
Punching	6.05 s/299.3 calls	2.54 s/13.3 calls	0.49 s/17.3 calls
Kicking	4.68 s/232.3 call	0.64 s/4.7 calls	0.29 s/10 calls
Workout	5.05 s/219.8 calls	0.78 s/4.8 calls	0.25 s/9 calls

Table 4
The average time / error per frame when running the IK solver over a complete motion sequence with stopping tolerance $f(\theta) < 10^{-2}$ and prescribed maximum iterations k_{max}.

Motion seq. (HDM05)	quasi-Newton $k_{max} = 10$	quasi-Newton $k_{max} = 100$	exact Hessian $k_{max} = 10$
dg_03-05_02	0.38 s/23.78 cm	2.63 s/1.61 cm	1.01 s/0.5 cm
dg_03-02_03	0.31 s/15.25 cm	4.25 s/1.65 cm	0.69 s/0.13 cm
bk_05-03_03	0.33 s/308.06 cm	2.19 s/5.96 cm	0.91 s/0.10 cm
mm_03-05_02	0.32 s/43.3 cm	2.29 s/2.80 cm	0.77 s/0.09 cm

reconstruction. The exact method typically required fewer than 4 iterations to achieve the prescribed error of $f(\theta) < 10^{-2}$. The quasi-Newton method with $k_{max} = 10$ does a very poor job of reconstructing the motion, and for some frames the residual is quite high. Increasing the maximum iteration count to $k_{max} = 100$ improved the reconstruction, but it still does not achieve the low error of the exact Newton method, even though the computational time is comparable.

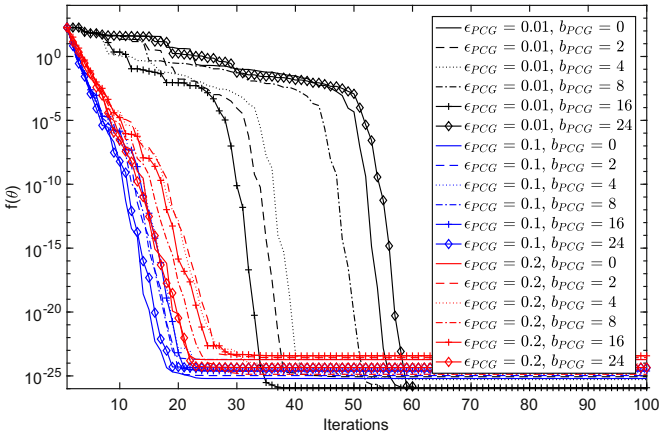

Fig. 8. A parameter sweep is performed to evaluate the effect of preconditioned conjugate gradient (PCG) parameters: stopping tolerance, ϵ_{PCG}, and preconditioner bandwidth, b_{PCG}.

6.7. Tuning the optimization

Here we delve into some specific details about the trust region and BFGS algorithm used in our experiments. While the BFGS method gave consistent rates of convergence across the exemplary frames, we found that the trust region method was sensitive to selection of parameters related to the PCG algorithm that is used to solve the trust region subproblem.

The default Jacobi, or diagonal, preconditioner is simple and efficient to compute. However, it assumes that the Hessian is diagonally dominant, which is not guaranteed in our case. We found that using a larger bandwidth for the preconditioning matrix helped convergence in some instances. Furthermore, we found that setting a very small stopping tolerance for the PCG iterations could sometimes lead to needless exploration of regions of the solution space, especially when the Hessian matrix in these regions was indefinite. This occurred most often when the residual was large (e.g. $\|r(\boldsymbol{\theta})\| > 1$). This is not unexpected since our objective function is essentially a sum of squared errors function, and the Hessian will only be positive definite near the final solution. In such cases, setting a low function tolerance was detrimental and gave linear or sub-linear rate of convergence, similar to what we observed in Fig. 5.

We tune these parameters empirically by sweeping the function tolerance $\epsilon_{PCG} \in [0.01, 0.1, 0.2]$ and preconditioner bandwidth $b_{PCG} \in [0, 2, 4, 8, 16, 24]$. Fig. 8 shows the convergence when running this sweep for a selected frame of motion from the punching sequence. We found that overall $\epsilon_{PCG} = 10^{-1}$ and $b_{PCG} = 24$ gave good performance across our examples, and therefore we use these values in all of our experiments. We note that changing these values slightly could lead to better or worse performance depending on the specific IK problem.

We also observed that the sub-solver sometimes required many iterations to converge. This is not unexpected, since there is no guarantee that the Hessian matrix is well-conditioned. In fact, sometimes the condition number of the Hessian can become quite large (larger than 10^5). We therefore give the PCG solver adequate opportunity to solve the sub-problem by setting the maximum iterations to M^2, where M is the number of skeletal degrees of freedom.

Finally, in cases where the Hessian matrix is indefinite, we found that applying the technique described in Higham [30] to compute a "nearby" semi-positive definite Hessian matrix was helpful. For the results in Fig. 5, this helped to improve convergence when the initial solution was not close to the true solution.

7. Discussion and conclusion

The computation of exact Hessians for solving IK problems is often dismissed based on conventional pragmatism, but rarely examined in detail. In this paper and the accompanying supplementary material, we have presented a mathematical framework for computing exact Hessians which is useful for computer graphics researchers. We have also evaluated the benefits of including exact Hessians based on experimentation with real-world motion capture data.

We observe from our convergence plots in Figs. 3 and 4 that with fewer than 10 iterations, the exact method usually achieves several orders of magnitude better accuracy than the quasi-Newton method. For Newton's method class of algorithms, the added information from the exact Hessian results in very few required iterations to achieve a specified accuracy, and in many cases the low number of iterations outweighs the per iteration cost of computing the exact Hessian. Hence, one gets a faster overall numerical method.

However, the LM algorithm achieves comparable convergence at much less computational cost, but as Fig. 5 illustrates convergence to a low error solution is dependent on having a good initial solution. All methods benefit when temporal coherence can be exploited (i.e. using joint angles from the previous frame of motion), but only the exact Hessian method consistently give good accuracy when a poor initial solution was provided. In our experiments, the Hessian approximation methods could often not reach the same level of accuracy in these instances, even after more than 200 iterations.

We note that our MATLAB implementation is unoptimized, and variables shared during computation of the Jacobian and Hessian are recomputed. Computing these just once and reusing their value would certainly improve computational performance. Performance could also be improved by realizing a parallelized implementation of the Jacobi and Hessian computations. Currently, a single thread is used to compute matrices and minimize the objective function. We note that although computing the Hessian has quadratic time complexity, the for-loops in this algorithm are straightforward to parallelize. Hence, a parallel numerical method with exact Hessians will have low computational time constants. In fact, the speedup factor of computing the Hessian is likely to be in favor of a GPU as the fill-in of the Hessian is one order of magnitude larger than the data on which its computation depends on. Hessian free and approximate methods do not have this potential for speed-up.

As noted in Section 6.7, the exact Newton method regularly encounters solutions where the Hessian matrix is indefinite. This is a limitation of our approach, and even simple skeletons produce many configurations for which the Hessian matrix is indefinite, or even negative definite. We provide further analysis and discussion about the definiteness of the Hessian matrix in Appendix A. In this work we proposed using a technique to enforce semi-positive definiteness.

Future work will investigate the conditions under which this happens and how to avoid them. A hybrid approach that switches between exact and approximate modes would also be an interesting research trajectory to explore. For instance, choosing a gradient- based, quasi-Newton, or exact Newton method based-on performance expectations given the current solution and residual.

Acknowledgments

Thanks to Jernej Barbič and Yili Zhao for releasing their ASF/AMC viewer, which we use to create animations in the supplementary video.

Appendix A. Definiteness of Hessian matrices

The goal of this section is to provide the reader with further intuition about the definiteness of the Hessian matrix and configurations that are desirable for performance and convergence. For this purpose, the example shown in Fig. A.9 is analyzed and configurations where the matrix is positive definite, indefinite, and negative definite are explored.

We sample the skeleton by sweeping each joint angle in the range $[-180^\circ, +180^\circ]$ and computing the Hessian at each configuration. Three different goal locations are used in our analysis since the Hessian matrix built using Eq. (14) is dependent on the residual. We use goal locations $\mathbf{g} = [10]^T$, $\mathbf{g} = [20]^T$, and $\mathbf{g} = [30]^T$, which are located inside the workspace, the workspace perimeter, and outside the workspace, respectively.

Fig. A.10 shows the definiteness of the Hessian across the configuration landscape. Regions where the Hessian matrix is positive definite, indefinite, negative definite, and semi-definite are identified. The Hessian matrix is positive definite when all of its eigenvalues are greater than zero; indefiniteness occurs when it contains both negative and positive eigenvalues; and the matrix is negative definite when all eigenvalues are less than zero.

We observe that the Hessian matrix for our toy example is positive definite only in regions where the residual is small. This is not surprising since we are effectively minimizing a sum of squared errors function, which exhibits convexity only in the region near a local or global minimum. Conversely, the Hessian is negative definite in regions where the residual is large. This can be confirmed by inspection of the 3D surface plots, which uses the value of the objective function $f(\boldsymbol{\theta})$ as the height of the surface.

The matrix is semi-definite when there are zero eigenvalues. This essentially means loss of degrees of freedom such as would occur in a gimbal lock (e.g. a 2D linkage with three revolute joints can loose a degree of freedom if two consecutive links collapse). This case does not occur frequently for the example we provide, but semi-definiteness happens more easily when the number of links in a kinematic chain grows due to increasing the redundancy in the degrees of freedom.

A positive definite matrix is preferred since it permits the use of efficient linear solvers, such as the conjugate gradient method. Furthermore, in regions where the Hessian is positive definite, Newton's method will converge quickly. However, for indefinite configurations, Newton's method may become attracted to a saddle point and converge to a local minimum. This is problematic for more complex examples, as other work observes that the ratio of saddle points to local minima increases exponentially with the dimensionality of the problem [31].

Two possible approaches for dealing with semi-definiteness and indefiniteness: (i) model joint constraints more accurately by adding limits, or (ii) add regularization terms to the IK objective function. The intuition behind joint limits is that the feasible region is significantly reduced and disallows iterative numerical methods to search in regions with semi- and indefinite Hessians. Regularization helps by adding extra terms to the objective function and thereby changing the Hessian. If done correctly, the modified Hessian always remains positive definite.

Fig. A.9. A simple 2D skeleton with 2 DOF shown in the neutral configuration, with $\theta_1 = 0$ and $\theta_2 = 0$. Each bone has length 1, and the root is fixed to the origin of the global coordinate frame.

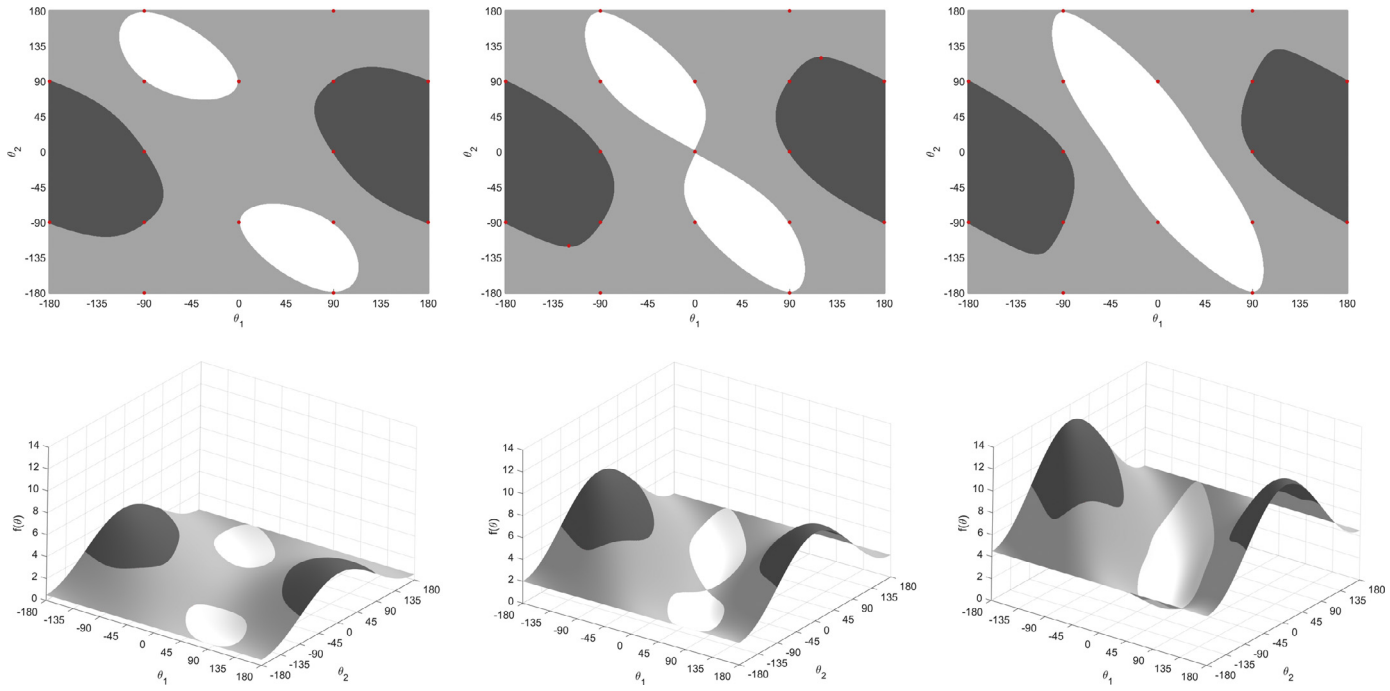

Fig. A.10. Examining the Hessian matrix for the linkage shown in Fig. A.9. Top row: regions in the configuration space of a two joint linkage where the Hessian is positive definite (white), indefinite (light gray), negative definite (dark gray) and semi-definite (red). Bottom row: the same regions represented as a 3D surface where the height is determined by the objective function $f(\boldsymbol{\theta})$. Since the Hessian matrix is dependent on the residual, plots for $\mathbf{g} = [10]^T$ (left), $\mathbf{g} = [20]^T$ (middle), and $\mathbf{g} = [30]^T$ (right). Note that positive definite Hessian matrices are found in regions where the residual is small.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2018.10.012.

References

[1] Craig JJ. Introduction to robotics: mechanics and control. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 1989.

[2] Girard M, Maciejewski AA. Computational modeling for the computer animation of legged figures. In: Proceedings of the 12th annual conference on computer graphics and interactive techniques, SIGGRAPH '85. New York, NY, USA: ACM Press; 1985. p. 263–70.

[3] Hsia TC, Guo ZY. New inverse kinematic algorithms for redundant robots. J Robot Syst 1991;8(1):117–32.

[4] Buss SR. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods. Unpublished Survey. 2009. https://www.math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf.

[5] Aristidou A, Lasenby J, Chrysanthou Y, Shamir A. Inverse kinematics techniques in computer graphics: a survey. Comput Graph Forum 2018;37:35–58. doi:10.1111/cgf.13310.

[6] Lee S-H, Kim J, Park FC, Kim M, Bobrow JE. Newton-type algorithms for dynamics-based robot movement optimization. IEEE Trans. Robot. 2005;21(4):657–67.

[7] Erleben K, Andrews S. Inverse kinematics problems with exact hessian matrices. In: Proceedings of the tenth international conference on motion in games, MIG '17. New York, NY, USA: ACM; 2017 14:1–14:6.

[8] Ho ESL, Komura T, Lau RWH. Computing inverse kinematics with linear programming. In: Proceedings of the ACM symposium on virtual reality software and technology, VRST '05. New York, NY, USA: ACM; 2005. p. 163–6.

[9] Sumner RW, Zwicker M, Gotsman C, Popovic J. Mesh-based inverse kinematics. ACM Trans. Graph. 2005;24(3):488–95.

[10] 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. Proceedings of the international conference on advances in intelligent computing, ICIC 2005, Hefei, China, August 23–26, 2005, Part II. Lecture Notes in Computer Science, 3645. Springer; 2005. p. 423–32.

[11] Safonova A, Hodgins JK, Pollard NS. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. ACM Trans Graph 2004;23(3):514–21.

[12] Fang AC, Pollard NS. Efficient synthesis of physically valid human motion. ACM Trans Graph 2003;22(3):417–26.

[13] Liu CK, Hertzmann A, Popovic; Z. Learning physics-based motion style with nonlinear inverse optimization. ACM Trans Graph 2005;24(3):1071–81.

[14] Zhao J, Badler NI. Inverse kinematics positioning using nonlinear programming for highly articulated figures. ACM Trans Graph 1994;13(4):313–36.

[15] Engell-Nørregård M, Erleben K. Technical section: a projected back-tracking line-search for constrained interactive inverse kinematics. Comput Graph 2011;35(2):288–98.

[16] Wellman C. Inverse kinematics and geometric constraints for articulated figure manipulation. Simon Fraser University; 1993.

[17] Harish P, Mahmudi M, Callennec BL, Boulic R. Parallel inverse kinematics for multithreaded architectures. ACM Trans Graph 2016;35(2) 19:1–19:13.

[18] Nocedal J, Wright SJ. Numerical optimization. New York: Springer-Verlag; 1999.

[19] Callennec BL. Interactive techniques for motion deformation of articulated figures using prioritized constraints. École Polytechnique Fédérale de Lausanne (EPFL); 2006.

[20] Engell-Nørregård M, Erleben K. A projected non-linear conjugate gradient method for interactive inverse kinematics.. In: Proceedings of the 6th Vienna international conference on mathematical modelling, MATHMOD; 2009.

[21] Fedor M. Application of inverse kinematics for skeleton manipulation in real-time. In: Proceedings of the 19th spring conference on computer graphics, SCCG '03. New York, NY, USA: ACM Press; 2003. p. 203–12.

[22] Zordan VB. Solving computer animation problems with numeric optimization. Technical Report; 2002.

[23] Maciejewski AA. Motion simulation: dealing with the ill-conditioned equations of motion for articulated figures. IEEE Comput Graph Appl 1990;10(3):63–71.

[24] Chiacchio P, Siciliano B. A closed-loop jacobian transpose scheme for solving the inverse kinematics of nonredundant and redundant wrists. J Robot Syst 1989;6(5):601–30.

[25] Grassia FS. Practical parameterization of rotations using the exponential map. J Graph Tools 1998;3(3):29–48.

[26] Coleman TF, Li Y. An interior trust region approach for nonlinear minimization subject to bounds. SIAM J Optim 1996;6(2):418–45.

[27] Broyden CG. The convergence of a class of double-rank minimization algorithms 1. general considerations. IMA J Appl Math 1970;6(1):76–90.

[28] Fletcher R. A new approach to variable metric algorithms. Comput J 1970;13(3):317–22.

[29] Müller M, Röder T, Clausen M, Eberhardt B, Krüger B, Weber A. Documentation MOCAP database HDM05. Technical Report; 2007. http://resources.mpi-inf.mpg.de/HDM05/.

[30] Higham NJ. Computing a nearest symmetric positive semidefinite matrix. Linear Algebra Appl. 1988;103:103–18.

[31] Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Proceedings of the advances in neural information processing systems (NIPS); 2014. p. 2933–41.