Fast and Robust Inverse Kinematics of Serial Robots using Halley's Method

Steffan Lloyd, Rishad Irani, *Member, IEEE,* and Mojtaba Ahmadi, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel numerical inverse kinematics algorithm called the *Quick Inverse Kinematics* or *QuIK* method. The QuIK method is a *third-order* algorithm that uses both the first and second-order derivative information to iteratively converge to a solution. Numerical inverse kinematics methods are readily implemented on any serial robot and do not rely on joint alignment. However, they typically are slower and less robust. The second-order derivative term allows the QuIK algorithm to converge more rapidly and more robustly than existing algorithms. A damped extension to the QuIK method is also proposed to increase reliability near singularities. The QuIK methods are tested in terms of evaluation speed, reliability, and singularity robustness against the Newton-Raphson method and several other modern algorithms. The proposed QuIK methods outperform all other tested algorithms in terms of speed and robustness, and have strong performance near singularities. The QuIK algorithms are proposed as faster and more robust "drop-in" replacements to the Newton-Raphson methods in inverse kinematics. C++ and Matlab codebases are made available.**

***Index Terms*—Kinematics, numerical inverse kinematics, simulation & animation, performance evaluation & benchmarking.**

I. Introduction

INVERSE kinematics is a ubiquitous problem in robotics and many other domains, such as computer graphics, animation, video games [1], and even biology [2]. Many kinematic chains do not admit closed-form solutions to their inverse kinematics problem. When closed-form solutions are possible, they are difficult to derive and rely on perfect joint alignment. Numerical inverse kinematic routines transform the inverse kinematics into either an optimization problem or a root-finding problem, and iterate from an initial guess of the joint variables to an exact solution. Numerical inverse kinematic routines are used in many robotics applications, such as

- *Kinematic calibration*: Calibrated kinematic parameters can give higher robot accuracy, but preclude the use of closed-form solutions since joint alignment is imperfect. However, numerical solutions have no such limitation.
- *Optimal or singularity-free design*: Robotic designs that allow for straightforward closed-form inverse kinematic solutions are not necessarily optimal by other design criteria. For example, the spherical wrist used by many manipulators to decouple translational and rotational motion can create pinch points which are hazardous for human-robot interactions [3].
- *Derivation-free analysis*: Numerical inverse kinematic solutions can be performed programmatically without a derivation step. This reduces implementational complexity, and is required in many applications such as computer graphics [4], reconfigurable robots [5], and more.
- *Redundant manipulators*: Closed-form inverse kinematic formulations of highly articulated robots are challenging and often impossible; however, a numerical approach is intuitive and allows for straightforward optimization of the pose within the null-space of the kinematic chain [1].

Due to the broad applicability of the inverse kinematics problem, many such iterative numerical algorithms have been developed. Existing methods can be broadly categorized into heuristic, optimization-based and Jacobian-based methods [4]. Heuristic methods implement simple update rules to solve the inverse kinematics problem. They are readily implemented and involve only basic computations. One of the most popular heuristic methods is the *Cyclic Coordinate Descent* (CCD) method [6]. CCD runs quickly and is ideal when an approximate solution must be obtained rapidly. However, it is slower than other methods when higher precision is needed and has issues converging in some cases [6].

In optimization methods, the problem is formulated as an optimization problem with a scalar cost function to be minimized. Conversely, Jacobian-based methods use the manipulator Jacobian function to estimate joint changes which will reduce the end-effector error to zero. Generally, Jacobian-based solutions are less computationally intensive and more easily implemented [4]; however, optimization methods can be more flexible in dealing with redundancy resolution [7]. The focus of the current work is on Jacobian-based methods.

A large number of methods have been proposed within the Jacobian-based category of numerical methods, including *first-order* Jacobian transpose techniques [8], and *second-order* methods such as Jacobian pseudoinverse methods [9] and

Manuscript received October 7, 2021; accepted March 15, 2022. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant RGPIN-2017-06967, Grant RGPIN-2015-04169, and Grant CRDPJ 514258-17. The work of Steffan Lloyd was supported by NSERC CGS-D. This article was recommended for publication by Associate Editor Q. Li and Editor E. Yoshida upon evaluation of the reviewers' comments. (*Corresponding author: Steffan Lloyd.*)

The authors are with the Department of Mechanical and Aerospace Engineer- ing, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: steffan.lloyd@ carleton.ca; rishad.irani@carleton.ca; mojtaba.ahmadi@carleton.ca).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2022.3162954.

Digital Object Identifier 10.1109/TRO.2022.3162954

variations thereof such as the damped least-squares (DLS) method (also known as the Levenberg-Marquardt algorithm) [10]–[12], or the selectively damped least squares method [13]. The Broyden-Fletcher-Goldfarb-Shannon (BFGS) algorithm has also been applied to inverse kinematics [1], and avoids explicit inversion of the Jacobian matrix — instead estimating its inverse through successive gradient evaluations. The BFGS algorithm has less computational load at each iteration; however, steps are not taken as accurately, so more iterations may be needed and reliability may not be as high. Some more recent works have also investigated using a machine-learning approach to inverse kinematics — using neural networks to learn the inverse relationship between the joint angles and the corresponding Cartesian coordinates [14], [15]. However, these methods give inexact answers, require a costly network training step before they can be used, and do not handle redundancy in an intuitive manner. These machine learning solutions, as well as just simple lookup tables, can also be used as an initial seed to numerical algorithms, thus improving the quality of the initial guess and increasing reliability [16].

The current state-of-the-art and defacto-usage of numerical inverse kinematics fall into variations of these aforementioned algorithms. The open-source *Orocos Kinematics and Dynamics Library* (KDL) [17] is arguably the most popular generic inverse kinematics solver [7], and features various solvers based on a Levenberg-Marquard optimization approach, as well as a Newton-Raphson pseudoinverse solver. The KDL library is used in the ROS libraries, along with another solver called Trac-IK that improves reliability by simultaneously running both a Newton-Raphson solver and a BFGS solver, and stopping when either of the two reaches a successful solution [7]. Mathworks' popular *Matlab Robotics Toolbox* features two solvers – one based on the Levenberg-Marquard algorithm (implemented as in [12]), and another based on the BFGS algorithm (implemented as in [1]).

At the core of these second-order Jacobian-based methods is the Newton-Raphson (NR) root-finding algorithm. For a C^1 continuous nonlinear function $f : \mathbb{R}^n \to \mathbb{R}^m$, the first-order Taylor series expansion is given by

$$f(\mathbf{x} + \delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\, \delta\mathbf{x}. \tag{1}$$

The NR method attempts to find the root of f, denoted by the symbol $\mathbf{x}^*$, by setting $f(\mathbf{x} + \delta\mathbf{x}) = \mathbf{0}$, then solving (1) for the required step $\delta\mathbf{x}$ to reach $\mathbf{x}^*$. This rearrangement yields an iterative equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla f(\mathbf{x})^{-1} f(\mathbf{x}) = \mathbf{x}_k - \nabla f(\mathbf{x}) \setminus f(\mathbf{x}), \tag{2}$$

where, for the remainder of this paper, we will use the backslash symbol, $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$, to denote $\mathbf{x}$ as the solution to the linear equation $\mathbf{A}\mathbf{x} = \mathbf{b}$. We use this specialized notation since this operation includes much hidden complexity. The linear system is not necessarily square, and as such an explicit inversion may not be possible. For singular systems, a specific solution must be selected, and for over-determined systems the solution $\mathbf{x}$ will not give exact results and a least-squares approach (or similar) is necessary. Linear systems can also be solved faster and more robustly through the pseudoinverse of $\mathbf{A}$ or through row operations such as a Lower-Upper (LU) or Orthogonal (QR) decomposition. How the linear system is solved affects the performance of the inverse kinematics algorithm, particularly for redundant chains or near singularities [13], [18].

The NR method is a *second-order* algorithm, and by most standards, is very fast. Once the algorithm is sufficiently close to the true solution $\mathbf{x}^*$, its convergence is quadratic — within a certain basin, the number of significant digits in the estimate approximately doubles with each iteration [19]. For the inverse kinematics problem, the function f is the error vector between the desired and current configuration, and the gradient term is the manipulator Jacobian matrix – a quantity that is not difficult to compute and is often readily available in robotics applications. However, a drawback of the NR method is that at singular, or near-singular points, the manipulator Jacobian becomes ill-conditioned, which can cause the iteration equation (2) to diverge. This issue is the rationale of the use of damped methods [10], [11], [13], in which the speed of the algorithm is sacrificed slightly to allow for more reliability near kinematic singularities. Another issue with the NR method is that, as is, it does not handle the addition of constraints to the problem. Extensions to handle these cases exist, such as task augmentation [20], null-space projection [21], [22] or task priority [23].

The NR method is the first member of a class of root-finding algorithms known as *Householder's methods* [24]. The second algorithm in this class is the *third-order* root finding method, known as *Halley's Method*. In Halley's method, the Taylor series is expanded one term further [25], as

$$f(\mathbf{x} + \delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\, \delta\mathbf{x} + \tfrac{1}{2}\, \nabla^2 f(\mathbf{x})\, \delta\mathbf{x}\, \delta\mathbf{x}. \tag{3}$$

Here, the second derivative term $\nabla^2 f(\mathbf{x})$ is, in fact, a rank-3 tensor in $\mathbb{R}^{m \times n \times n}$, and the associated multiplication operation is defined "naturally," such that $\left[\nabla^2 f(\mathbf{x})\, \delta\mathbf{x}\right] \in \mathbb{R}^{m \times n}$ and $\left[\nabla^2 f(\mathbf{x})\, \delta\mathbf{x}\, \delta\mathbf{x}\right] \in \mathbb{R}^m$, with elements defined respectively as

$$\begin{aligned} \left[\nabla^2 f(\mathbf{x})\, \delta\mathbf{x}\right]_{ij} &= \sum\nolimits_k \left[\nabla^2 f(\mathbf{x})\right]_{ijk} \delta x_k, \\ \left[\nabla^2 f(\mathbf{x})\, \delta\mathbf{x}\, \delta\mathbf{x}\right]_i &= \sum\nolimits_k \sum\nolimits_j \left[\nabla^2 f(\mathbf{x})\right]_{ijk} \delta x_k\, \delta x_j. \end{aligned} \tag{4}$$

To solve for the desired step size of (3), the left-hand side is set to zero, and the equation is solved for $\delta\mathbf{x}$. Here, however, the problem is more complex and solving the resulting quadratic equation is non-trivial in the multidimensional case. Instead, an estimate of $\delta\mathbf{x}$ is obtained as the NR step,

$$\delta\mathbf{x}_{\mathrm{nr}} = -\nabla f(\mathbf{x}) \setminus f(\mathbf{x}), \tag{5}$$

and substituted into (3), yielding

$$\mathbf{0} \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\, \delta\mathbf{x} + \nabla^2 f(\mathbf{x})\, \delta\mathbf{x}_{\mathrm{nr}}\, \delta\mathbf{x}. \tag{6}$$

This expression can be rearranged for $\delta\mathbf{x}$ and yields the iterative equation [25]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\nabla f(\mathbf{x}) + \tfrac{1}{2}\, \nabla^2 f(\mathbf{x})\, \delta\mathbf{x}_{\mathrm{nr}}\right] \setminus f(\mathbf{x}). \tag{7}$$

Halley's method is perhaps more easily understood graphically. Fig. 1 shows both the NR method and Halley's method on a 1-dimensional sinusoidal function. For each iteration, the

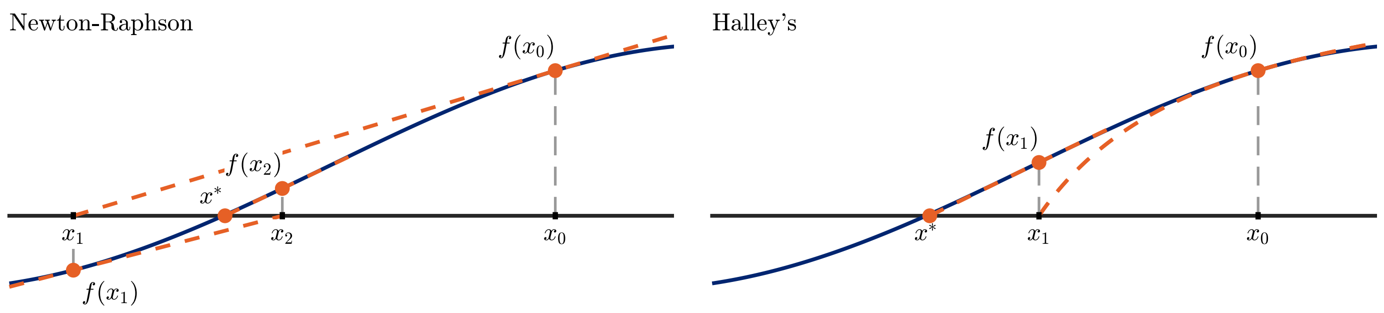

Fig. 1. Illustration of the Newton-Raphson method (left) and Halley's method (right) on a sinusoidal function. The Newton-Raphson method uses a first-order (linear) function approximation at x, whereas Halley's method uses a second-order approximation — matching both the function's slope and acceleration at x.

former fits a line to match the slope of $f(x)$ and uses it to find the next step of the iteration. Halley's method instead fits an *oscullating curve* that matches both the slope and the second derivative of $f(x)$, and uses the roots of this fitted curve to estimate the next point [19]. The additional higher-order information improves the estimate of the root location. Halley's method is a *third-order method*, and instead of the quadratic convergence of the NR method, Halley's method gives *cubic* convergence within the basin of convergence — approximately *tripling* the number of correct significant digits at each iteration [26].

Despite its advantages, Halley's method, and other similar third-order methods, do not see much widespread usage [26], [27]. Computation of both the first and second-order partial derivative terms can be prohibitively difficult, both in terms of derivation and evaluation. Computing the gradient for NR requires $(m \cdot n)$ partial derivative terms, whereas $\nabla^2 f(\mathbf{x})$ involves $(m{\cdot}n^2)$ second-order partial derivative terms and thus scales poorly to larger n [26]. If numerical differentiation must be used for the derivative terms, the complexity of evaluating both $\nabla f(\mathbf{x})$ and $\nabla^2 f(\mathbf{x})$ becomes intractable. Given that the NR or even first-order gradient descent methods are easier to implement and can achieve the same result by performing a larger number of simpler iterations, the effort of implementing Halley's method is often deemed unnecessary or even counterproductive [27].

However, certain problems are well-suited to higher-order root-finding methods, and have the following properties:

1) *Low dimensionality:* Unlike a typical artificial intelligence problem which may involve copious optimization parameters, problems that favor the use of Halley's method would have a relatively low order, decreasing the number of additional partial derivative terms in $\nabla^2 f(\mathbf{x})$ and lessening the cost of solving an additional set of linear equations.
2) *Expensive function evaluation*: If a single evaluation of f is computationally expensive, it is more desirable to minimize the number of iterations required by the algorithm.
3) *Easily computed, symmetric or sparse second derivative:* If the second derivative term $\nabla^2 f(\mathbf{x})$ is easily computed compared to the objective function f or the first derivative $\nabla f(\mathbf{x})$, it is more desirable to use Halley's method. If the second-derivative has additional symmetry or sparsity that can reduce computations and memory usage, then the method is additionally relevant.

Based on current literature, Halley's method does not seem to have been investigated as a method for solving the inverse kinematics problem in kinematic chains. However, the reader can note that the numerical inverse kinematics problem fills *all* the properties listed above. The problem dimensionality is typically relatively small (i.e., compared to machine learning problems). The forward kinematics of a manipulator is quite computationally intensive, so additional iterations are not cheap. The second-derivative term, known as the *Kinematic Hessian*, is surprisingly easy to compute, and features both significant symmetry and sparsity [28]. Finally, it will be shown that Halley's method is very robust in the inverse kinematics problem, converging more reliably and within a larger basin of convergence than the NR method. This increased robustness is significant in robotics, where reliability is critical — particularly in real-time applications.

This paper makes several contributions. Firstly, we show how Halley's method can be readily applied to the inverse kinematics problem, resulting in the novel *Quick Inverse Kinematics* (QuIK) method. Second, we derive a singularity-robust version of the QuIK method, called the *Damped Quick Inverse Kinematics* (DQuIK) method. Thirdly, we provide extensive benchmarking of the proposed methods against the Newton-Raphson algorithm, as well as several state-of-the-art inverse kinematics packages. These benchmarks show the superiority of the QuIK algorithms over traditional inverse kinematic methods in terms of convergence rate, robustness, and speed. Additionally, a full implementation of the proposed algorithms is developed in C++ and Matlab code, and made available for general use [29].

This paper is organized as follows. Section II explicitly states the inverse kinematics problem and defines notation for the paper. Section III presents the Quick Inverse Kinematics algorithm and associated derivations, extensions and details. Lastly, Section IV shows several benchmarking tests of the QuIK algorithms for convergence rate, overall algorithm efficiency, robustness, and behavior near singularities.

II. PROBLEM DEFINITION

Consider an n degree-of-freedom (DOF) manipulator with generalized coordinates $\mathbf{q} \in \mathbb{R}^n$. We define $\mathbf{T}_i$ as the 4×4 homogeneous transformation matrix from the i^{th} frame of the robot to the base coordinate system, composed from a rotation matrix $\mathbf{R}_i$ and displacement vector $\mathbf{p}_i$ as

$$\mathbf{T}_i(\mathbf{q}) = \left[\begin{array}{c:c} \mathbf{R}_i & \mathbf{p}_i \\ \hdashline \mathbf{0} & 1 \end{array}\right]. \tag{8}$$

$\mathbf{R}_i$ is further subdivided into three column vectors $\mathbf{n}_i$, $\mathbf{s}_i$ and $\mathbf{a}_i$, corresponding to unit vectors along the x–, y– and z–axes of the i^{th} coordinate system, as

$$\mathbf{R}_i = \left[\begin{array}{c:c:c} \mathbf{n}_i & \mathbf{s}_i & \mathbf{a}_i \end{array}\right]. \tag{9}$$

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function that reports the errors of m tasks for the manipulator. In the general case, the inverse kinematics problem seeks to find the joint angles $\mathbf{q}^*$ which yield $f(\mathbf{q}^*) = \mathbf{0}$. If these m tasks correspond to the spatial and orientation positioning of the last link of the chain to a desired transformation matrix $\mathbf{T}_d$, composed of a rotation $\mathbf{R}_d$ and position $\mathbf{p}_d$, then $f : \mathbb{R}^n \to \mathbb{R}^6$ as

$$f(\mathbf{q}) = \mathbf{e} = \begin{bmatrix} \mathbf{e}_{\text{lin}} \\ \mathbf{e}_{\text{rot}} \end{bmatrix}, \tag{10}$$

where $\mathbf{e}$ is a 6-vector of the positional and rotational error of the n^{th} link. It is convenient to represent this error using the six-component exponential coordinates, arranged into a spatial twist as [30]

$$\mathbf{e} = \begin{bmatrix} \mathbf{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \log\left(\mathbf{T}_n \mathbf{T}_d^{-1}\right), \tag{11}$$

where $\mathbf{v}_e$ correspond to the linear velocity vector required to go from $\mathbf{p}_n$ to $\mathbf{p}_d$ in unit time, and $\boldsymbol{\omega}_e$ corresponds to the angular velocity vector required to rotate from $\mathbf{R}_n$ to $\mathbf{R}_d$ in unit time [30]. With this definition, the manipulator geometric Jacobian $\mathbf{J}(\mathbf{q})$ can be used directly as the error gradient $\nabla\mathbf{e}(\mathbf{q})$ [12]. Computation of $\mathbf{e}$ is accomplished via the matrix logarithm operation [12], [30], which is described in the appendix. The inverse kinematics problem seeks to set $\mathbf{e}(\mathbf{q})$ identically to zero. In this paper, the solution is achieved iteratively from an initial guess $\mathbf{q}_0$ according to an update law

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \delta\mathbf{q}. \tag{12}$$

In the current work, we deal solely with the case of serial kinematic chains. The extension to tree-like chains would be possible with only minor modifications and involve augmentation of the error function and corresponding Jacobian.

III. THE QUICK INVERSE KINEMATICS ALGORITHM

To properly define the QuIK method and associated equations, we must first discuss how the Newton-Raphson method is adapted to the inverse kinematics problem. In the NR method, we adapt (2) as

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \nabla f(\mathbf{q}_k) \setminus f(\mathbf{q}_k) = \mathbf{q}_k - \mathbf{J}(\mathbf{q}_k) \setminus \mathbf{e}(\mathbf{q}_k), \tag{13}$$

where $\mathbf{J}(\mathbf{q}_k)$ is the *geometric Jacobian* function of the chain, to the last frame (frame n) and expressed in world frame coordinates, satisfying the relation

$$\begin{bmatrix} \mathbf{v}_n \\ \boldsymbol{\omega}_n \end{bmatrix} = \mathbf{J}\,\dot{\mathbf{q}}. \tag{14}$$

For kinematic chains with frame assignments such that the joint axes coincide with the z–axes of each link, computation of the Jacobian $\mathbf{J}$ is straightforward from the forward kinematics $\mathbf{T}_i(\mathbf{q})$, as [31]

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_v \\ \mathbf{J}_\omega \end{bmatrix} = \begin{bmatrix} \mathbf{j}_{v1} & \cdots & \mathbf{j}_{vn} \\ \mathbf{j}_{\omega 1} & \cdots & \mathbf{j}_{\omega n} \end{bmatrix}, \tag{15}$$

where the submatrices $\mathbf{J}_v \in \mathbb{R}^{3\times n}$ and $\mathbf{J}_\omega \in \mathbb{R}^{3\times n}$ are called the *linear* and *angular velocity Jacobians*, respectively, with columns computed as

$$\begin{aligned} \mathbf{j}_{vj} &= \begin{cases} \mathbf{a}_{j-1} \times \left(\mathbf{p}_n - \mathbf{p}_{j-1}\right), & \text{if } j \text{ is revolute,} \\ \mathbf{a}_{j-1}, & \text{if } j \text{ is prismatic,} \end{cases} \\ \mathbf{j}_{\omega j} &= \begin{cases} \mathbf{a}_{j-1}, & \text{if } j \text{ is revolute,} \\ \mathbf{0}, & \text{if } j \text{ is prismatic.} \end{cases} \end{aligned} \tag{16}$$

To extend the NR method to the QuIK method, we add an additional derivative term to the iterative equation (13), effectively adapting the update equation (7) for Halley's method to the inverse kinematics problem, giving

$$\begin{aligned} \mathbf{q}_{k+1} &= \mathbf{q}_k - \left[\nabla f(\mathbf{q}_k) + \tfrac{1}{2}\,\nabla^2 f(\mathbf{q}_k)\,\delta\mathbf{q}_{\text{nr}}\right] \setminus f(\mathbf{q}_k) \\ &= \mathbf{q}_k - \left[\mathbf{J}(\mathbf{q}_k) + \tfrac{1}{2}\,\mathsf{H}(\mathbf{q}_k)\,\delta\mathbf{q}_{\text{nr}}\right] \setminus \mathbf{e}(\mathbf{q}_k), \end{aligned} \tag{17}$$

where the multiplication $\left[\mathsf{H}(\mathbf{q}_k)\,\delta\mathbf{q}_{\text{nr}}\right]$ is performed analogous to (4), $\delta\mathbf{q}_{\text{nr}}$ is the Newton-Raphson step

$$\delta\mathbf{q}_{\text{nr}} = -\mathbf{J}(\mathbf{q}_k) \setminus \mathbf{e}(\mathbf{q}_k), \tag{18}$$

and $\mathsf{H}(\mathbf{q}) \in \mathbb{R}^{6\times n\times n}$ is the rank-3 tensor corresponding to the gradient of the robot Jacobian,

$$\mathsf{H}(\mathbf{q}) \triangleq \nabla\mathbf{J}(\mathbf{q}) = \tfrac{\partial}{\partial\mathbf{q}}\mathbf{J}(\mathbf{q}), \tag{19}$$

which is the the Kinematic Hessian of the manipulator [32]. The H term should not be confused with the "Hessian matrix" which is sometimes defined when solving inverse kinematics as an optimization problem of a scalar cost function [1]. In these cases, as the cost function is scalar, the "Hessian" is a matrix and the resulting algorithm is *second-order*. Instead, H is a rank-3 tensor of second-order partial derivatives for the vector function $\mathbf{e}(\mathbf{q})$, formed by the partial derivatives of the geometric Jacobian $\mathbf{J}(\mathbf{q})$, and the resulting algorithm is *third-order*.

Practically, the QuIK method is straightforward to implement with (17) and (18). The difficulty is then to compute the kinematic Hessian H efficiently. Eq. (19) is not in a convenient form for numerical evaluation. However, closed-form derivation of H is possible by applying various kinematic identities. This derivation is omitted for brevity, but a thorough treatment can be found in [28]; the results are summarized in the following surprisingly concise formulae. Let $\mathbf{H}_i \in \mathbb{R}^{6\times n}$ denote the i^{th} "page" of H, such that $\mathbf{H}_i(\mathbf{q}) = \frac{\partial}{\partial q_i}\mathbf{J}(\mathbf{q})$. Then,

$$\mathbf{H}_i(\mathbf{q}) = \tfrac{\partial}{\partial q_i}\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \frac{\partial}{\partial q_i}\mathbf{j}_{v1} & \cdots & \frac{\partial}{\partial q_i}\mathbf{j}_{vn} \\ \frac{\partial}{\partial q_i}\mathbf{j}_{\omega 1} & \cdots & \frac{\partial}{\partial q_i}\mathbf{j}_{\omega n} \end{bmatrix}, \tag{20}$$

for $i = 1, \ldots, n$, where

$$\frac{\partial \mathbf{j}_{vj}}{\partial q_i} = \begin{cases} \mathbf{a}_{i-1} \times (\mathbf{a}_{j-1} \times [\mathbf{p}_n - \mathbf{p}_{j-1}]), & \\ & \text{if } i \le j \text{ and } i, j \text{ are revolute,} \\ \mathbf{a}_{j-1} \times (\mathbf{a}_{i-1} \times [\mathbf{p}_n - \mathbf{p}_{i-1}]), & \\ & \text{if } i > j \text{ and } i, j \text{ are revolute,} \\ \mathbf{a}_{i-1} \times \mathbf{a}_{j-1}, & \text{if } i \le j,\ i \text{ is revolute, } j \text{ is prismatic,} \\ \mathbf{a}_{j-1} \times \mathbf{a}_{i-1}, & \text{if } i > j,\ i \text{ is revolute, } j \text{ is prismatic,} \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

$$\frac{\partial \mathbf{j}_{\omega j}}{\partial q_i} = \begin{cases} \mathbf{a}_{i-1} \times \mathbf{a}_{j-1}, & \text{if } i \le j \text{ and } i, j \text{ are revolute,} \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{21}$$

Each of these terms is straightforward to compute directly. However, comparison with (16) reveals that many of the cross products above have already been computed in the manipulator Jacobian $\mathbf{J}$. Substitution of these terms directly into (21) results in further simplifications, summarized as

$$\frac{\partial \mathbf{j}_{vj}}{\partial q_i} = \begin{cases} \mathbf{j}_{\omega i} \times \mathbf{j}_{vj}, & \text{if } i \le j \text{ and } i \text{ is revolute,} \\ \mathbf{j}_{\omega j} \times \mathbf{j}_{vi}, & \text{if } i > j \text{ and } i \text{ is revolute,} \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

$$\frac{\partial \mathbf{j}_{\omega j}}{\partial q_i} = \begin{cases} \mathbf{j}_{\omega i} \times \mathbf{j}_{\omega j}, & \text{if } i \le j \text{ and } i, j \text{ are revolute,} \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{22}$$

Two final simplifications are possible. First, we can note the symmetry in the entries of the expression for $\frac{\partial}{\partial q_i}\mathbf{j}_{vj}$ and avoid redundant calculations. Second, the cross product of any vector with itself is always zero, so the term $\frac{\partial}{\partial q_i}\mathbf{j}_{\omega j} = \mathbf{0}$ if $i = j$. Thus, an equivalent but more efficient formulation is

$$\frac{\partial \mathbf{j}_{vj}}{\partial q_i} = \begin{cases} \mathbf{j}_{\omega i} \times \mathbf{j}_{vj}, & \text{if } i \le j \text{ and } i \text{ is revolute,} \\ \frac{\partial}{\partial q_j}\mathbf{j}_{vi}, & \text{if } i > j \text{ and } i \text{ is revolute,} \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

$$\frac{\partial \mathbf{j}_{\omega j}}{\partial q_i} = \begin{cases} \mathbf{j}_{\omega i} \times \mathbf{j}_{\omega j}, & \text{if } i < j \text{ and } i, j \text{ are revolute,} \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{23}$$

Eq. (23) is interesting for several reasons. First, the kinematic Hessian $\mathbf{H}$ is much simpler to compute than one may think, involving at most two cross products per column, and often fewer. Second, $\mathbf{H}$ benefits both from significant sparsity *and* significant symmetry, meaning that the number of computations required to evaluate it is significantly reduced. The number of cross products required to evaluate (23) is n^2 in the worst case (if all joints are revolute). In fact, only *half* of the elements of $\mathbf{H}$ require any computation at all. This sparsity and symmetry can be leveraged to reduce the requirements of computing the potentially large tensor $\mathbf{H}$, as well as in reducing the number of multiplications in the product $\left[\mathbf{H}(\mathbf{q}_k)\,\delta\mathbf{q}_{\text{nr}}\right]$. Finally, and perhaps most interesting, the kinematic Hessian $\mathbf{H}$ can be computed *solely from the Jacobian* $\mathbf{J}$ *itself*. This computational property reduces the complexity of evaluating $\mathbf{H}$, since $\mathbf{J}$ must be computed for the NR method regardless. It also means that implementing the QuIK method is straightforward — regardless of how $\mathbf{J}$ is computed, the extension to $\mathbf{H}$ involves only a small number of extra calculations.

A. Damping the QuIK Method

In inverse kinematics, the Newton-Raphson method is often not used in its pure form, as it can experience numerical instabilities when the manipulator is near a singularity. These instabilities arise in the step of solving the linear set of equations $\mathbf{J} \setminus \mathbf{e}$ in (13). One method of solving this system is through the Moore-Penrose pseudoinverse $\mathbf{J}^\dagger$, defined as

$$\mathbf{J}^\dagger = \mathbf{J}^\mathsf{T}\left(\mathbf{J}\,\mathbf{J}^\mathsf{T}\right)^{-1}. \tag{24}$$

While this solution involves more work than through the direct use of row operations such as LU decomposition, it provides a minimum-norm solution. While the minimum norm solution is well behaved when $\mathbf{J}$ is *exactly* singular, in practice exact singularities are highly unlikely, except by design [33]. However, if the pseudoinverse is evaluated near, but not exactly at a singularity, it can result in a poorly behaved solution, giving large commanded joint angles and destabilizing the iterative algorithms [13].

Although many methods exist for handling the convergence problems of the NR algorithm, one of the most common is the *Damped Least-Squares* or *Damped Newton-Raphson* (DNR) method [10], [11]. In optimization, this method is also known as the Levenberg-Marquardt algorithm. A damping factor λ is introduced into the pseudoinverse solution in (24), giving the DNR step as

$$\delta\mathbf{q}_{\text{dnr}} = -\mathbf{J}^\mathsf{T}\left(\mathbf{J}\,\mathbf{J}^\mathsf{T} + \lambda^2\mathbf{I}\right) \setminus \mathbf{e}, \tag{25}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. This damping factor acts as a norm-2 regularization term. Whereas (24) minimizes the residual error $||\mathbf{J}\delta\mathbf{q} - \mathbf{e}||^2$, instead (25) will minimize

$$||\mathbf{J}\delta\mathbf{q} - \mathbf{e}||^2 + \lambda||\delta\mathbf{q}||^2, \tag{26}$$

simultaneously attempting to solve the linear system while minimizing the step size $||\delta\mathbf{q}||$, as weighted by λ. When $\mathbf{e}$ is large, the damping term will have little or no effect on the resulting $\delta\mathbf{q}$. However, when $\mathbf{e}$ approaches zero, the damping term has a more significant effect and ensures that large steps are not taken to attempt to eliminate small errors.

To resolve the singularity issues in the novel QuIK method, we propose using the same damping methodology, in what we name the *Damped Quick Inverse Kinematics* (DQuIK) algorithm. Rather than finding an estimate to the root of the quadratic equation $\mathbf{J}\,\delta\mathbf{q} + \frac{1}{2}\,\mathbf{H}\,\delta\mathbf{q}\,\delta\mathbf{q} - \mathbf{e} = \mathbf{0}$, we instead attempt to minimize the quantity

$$||\mathbf{J}\,\delta\mathbf{q} + \tfrac{1}{2}\,\mathbf{H}\,\delta\mathbf{q}\,\delta\mathbf{q} - \mathbf{e}||^2 + \lambda||\delta\mathbf{q}||^2, \tag{27}$$

simultaneously solving the quadratic system while minimizing the step size $\delta\mathbf{q}$. As before, we first find an approximate solution from the NR method, except now the damping λ is included as $\mathbf{q}_{\text{dnr}}$ as in (25). Then, the DQuIK step size is computed with the same damped pseudoinverse solution as

$$\begin{aligned} \delta\mathbf{q}_k &= -\mathbf{A}^\mathsf{T}\left(\mathbf{A}\,\mathbf{A}^\mathsf{T} + \lambda^2\mathbf{I}\right) \setminus \mathbf{e}(\mathbf{q}_k), \\ \mathbf{A} &= \mathbf{J}(\mathbf{q}_k) + \tfrac{1}{2}\,\mathbf{H}(\mathbf{q}_k)\,\delta\mathbf{q}_{\text{dnr}}. \end{aligned} \tag{28}$$

The downside of the damped DQuIK and DNR methods is that they require an appropriate selection of the damping parameter λ. If λ is too low, the algorithm may experience issues near singularities, whereas if it is too large, the step $\delta\mathbf{q}$ is overly warped by the effect of λ and the algorithm converges slower than necessary. A thorough investigation into proper selection of λ is beyond the scope of the current work. However, there is a good base of literature on the subject. Many methods have been proposed for the NR method that involve dynamic selection of λ at each algorithm iteration, for optimal performance. These methods can be equally well applied to the QuIK method, by selecting λ based on a measure of manipulability, such as in [10], [34], by limiting the minimum singular values of $\mathbf{J}$ to be above a given threshold [13], [35], or by similarly limiting the condition number of $\mathbf{J}$ under a threshold [18], [36], [37]. Comparisons of many of these methods are given in [12], [18].

B. Extension to High-DOF & Redundant Chains

As with the Newton-Raphson methods, the QuIK methods can be extended to higher-DOF kinematic chains. As is, the given equations are perfectly suited to handle longer kinematic chains. With longer kinematic chains, a valid concern is that the third-order QuIK methods may not scale as ideally as the second-order NR methods, due to the exponential increase in partial derivative terms to be computed. Indeed, the calculation of the kinematic Hessian in the QuIK methods is of complexity $O(n^2)$, whereas the error and Jacobian terms in second-order methods such as the Newton-Raphson method have linear complexity, $O(n)$. Practically, however, this is not a significant problem. The computations of both methods are dominated computationally by the linear system solution steps, which, if done by LU or Cholesky decomposition, scales as $O(m^{2.8})$, or for Singular Value Decomposition (SVD) or QR decomposition, $O(m^2 n)$ [38] — in other words, the algorithms are dominated by the number of constraints, not the number of joints. Moreover, even the longest kinematic chains are still relatively small, by computational standards, so non-ideal scaling of the complexity is not as severe a problem as in other applications. In robotics, kinematic redundancy is often exploited to perform additional tasks or optimize the inverse kinematics solution within the null-space of the manipulator. These techniques could be accomplished as in the NR method through task augmentation [20] or null-space projection [21], [22]; however, this is beyond the scope of the current work.

IV. BENCHMARKING AND ANALYSIS

The final section of this work will seek to validate the proposed QuIK method in experimental testing. We have written a C++ library implementing the QuIK/DQuIK and NR/DNR methods for this testing, which is made available for general use [29].

In the majority of our testing, we compare the QuIK/DQuIK methods against the NR/DNR methods. Of course, many numerical inverse kinematics algorithms exist; however, the NR method is very common in modern inverse-kinematic algorithms, albeit often with various tweaks and modifications to the damping, exit conditions, etc. As these various implementations have too many variations and parameters to test and comment on in the current work, we instead use our implementations of both the NR and QuIK methods. These implementations are identical in all aspects except in their computations of the step size $\delta\mathbf{q}$. As such, they can be meaningfully compared to ascertain the fundamental performance of the two algorithms. Other algorithmic variations, such as dynamic or selective damping [12], [18] or more advanced exit conditions, can be equally well applied to either the QuIK or NR methods. Additionally, we include the quasi-Newton BFGS method, implemented as in [1], for comparison in the benchmarks. The BFGS method avoids explicit inversion of $\mathbf{J}$ such that each iteration is faster, but potentially less accurate, than the NR method.

This section is organized as follows. Section IV-A describes the implementational details of the algorithms for benchmarking, how sample inverse kinematics problems are generated, and an error saturation step that can further improve the algorithm performance. In Section IV-B, we briefly compare the convergence of the QuIK and NR algorithms. Section IV-C will compare the overall speed and reliability of the QuIK, NR, and BFGS methods, and show the QuIK method to outperform the other algorithms in nearly all tests. Section IV-D investigates the singularity robustness of the algorithms. Finally, to properly situate these comparisons among existing research, Section IV-E compares the QuIK method against other state-of-the-art algorithms commonly used today. For Sections IV-B to IV-D, we use a 6-DOF KUKA KR6 industrial robot as an example kinematic chain; however, in Section IV-E, we will provide results for an assortment of sample kinematic chains, including redundant manipulators.

A. Algorithm & Benchmark Implementation

The C++ codebase for the QuIK and NR algorithms can be viewed and downloaded at [29]. This codebase is well commented and can clarify any technical implementational details not covered in this paper. It can also be used to reproduce the benchmarks presented in this work. The algorithms take as input a desired end-effector transform $\mathbf{T}_n(\mathbf{q}^*)$ and an initial guess of the joint angles $\mathbf{q}_0$. The algorithms run until convergence is achieved, as determined by the condition

$$||\mathbf{e}|| < \varepsilon, \tag{29}$$

where ε is the desired target Cartesian accuracy, or after a maximum of 200 iterations. Double-precision (52-bit accuracy) arithmetic is used in all calculations. All testing is done on a desktop computer with an Intel i7 CPU@3.2 GHz with 32 GB memory. The code was compiled using the MinGW-w64 GCC compiler.

1) Sample problem generation: To test the algorithms, it is necessary to have a means of generating a large number of inverse kinematics "problems" to solve programmatically, such that speed, reliability and convergence can be quantified statistically with a good sample size. Each sample problem is defined by a target joint configuration $\mathbf{q}^*$ and a starting joint configuration $\mathbf{q}_0$. Target configurations $\mathbf{q}^*$ are generated randomly from a uniform distribution within $\pm\pi$. The target end-effector transform is then computed from the forward kinematics of the chain, $\mathbf{T}(\mathbf{q}^*)$. To obtain an appropriate starting pose $\mathbf{q}_0$, we randomly perturb $\mathbf{q}^*$ by a vector $\delta\mathbf{q}_{\text{pert}}$. The perturbations $\delta\mathbf{q}_{\text{pert}}$ are generated from a uniform distribution in the range $[-1, 1]$ and normalized by the average absolute value of its elements before being scaled by a desired mean initial joint error, denoted by σ. Mathematically, this can be written as

$$\mathbf{q}^* \sim \text{unif}(-\pi, \pi), \qquad \mathbf{r} \sim \text{unif}(-1, 1),$$
$$\delta\mathbf{q}_{\text{pert}} = \sigma\left(\frac{||\mathbf{r}||_1}{n}\right)^{-1}\mathbf{r}, \qquad \mathbf{q}_0 = \mathbf{q}^* + \mathbf{q}_{\text{pert}}. \tag{30}$$

This formulation is useful since the difficulty of the inverse kinematics problem can be selected by adjusting the mean initial joint error σ. Intuitively, σ can be interpreted as the mean magnitude of the error in each joint, i.e., $\sigma = \frac{1}{n}\sum|\delta q_{\text{pert},i}|$.

2) Error saturation: In our implementation, we additionally include a basic error saturation step, as described in [13], [33]. When the error $\mathbf{e}$ is large, the Taylor series expansion of (1) or (3) is no longer an accurate function estimate, which can cause the algorithms to take inaccurate steps. Moving the target position closer, in these cases, can help [33]. The saturated error is calculated as in [13], [33], as

$$\mathbf{e}_{\text{sat}} = \begin{cases} \mathbf{e}, & \text{if } ||\mathbf{e}|| < d, \\ d\,\mathbf{e}/||\mathbf{e}||, & \text{otherwise}, \end{cases} \tag{31}$$

where d is an adjustable parameter of the maximum step size. In the current work, (31) was implemented separately for the linear and rotational elements of $\mathbf{e}$, as they differ in magnitude (i.e., with parameters d_{lin} and d_{rot}).

This strategy dramatically increases performance in the NR algorithm, and a good initial guess is d_{lin} as half the average link lengths of the chain and $d_{\text{rot}} = \pi/4$ [33]. However, for the QuIK algorithms, due to their better approximation of the kinematics, error saturation was found to have a minor effect only. Regardless, it is necessary to include it to provide a fair comparison between both methods.

In the current work, these parameters were optimized by running both algorithms on $N = 10^5$ randomly generated samples with $\sigma = 1\,\text{rad}$, and quantifying the resulting error rate. This calculation, relating the error saturation parameters to the resulting error rate, was fed into a Nelder-Mead Simplex optimization routine [39], implemented via Matlab's `fminsearch` method, giving the optimal values for the KUKA KR6 robot of $d_{\text{lin}} = 140\,\text{mm}$ and $d_{\text{rot}} = 0.86\,\text{rad}$ (for NR) and $d_{\text{lin}} = 340\,\text{mm}$ and $d_{\text{rot}} = 1.00\,\text{rad}$ (for QuIK).

B. Algorithm Convergence

The first benchmark will demonstrate the differences in convergence speeds between the NR and QuIK methods. For this test we generate $N = 10^6$ random target configurations as in (30), with $\sigma = \frac{\pi}{4}\,\text{rad}$, for the KUKA KR6 manipulator shown in Fig. 2 and with Denavit-Hartenberg (DH) parameters as in Table I. We run each set of sample joint configurations through both the NR and QuIK algorithms and track the normed errors $||\mathbf{e}||$ at each iteration. In Fig. 3, we plot the results of this test, showing at each iteration of the algorithms the full distribution of the iteration errors and their median. This figure shows the rapid convergence of the QuIK algorithm, and illustrates the quadratic and cubic convergence of the NR and QuIK algorithms, respectively. For the NR method, it can be noted that accuracy approximately doubles with each iteration ($||\mathbf{e}|| \approx 10^{-0.5} \to 10^{-0.8} \to 10^{-1.2} \to 10^{-2} \to 10^{-3.4} \to 10^{-6.1}$), and for the QuIK method, it approximately triples instead ($||\mathbf{e}|| \approx 10^{-0.9} \to 10^{-2.4} \to 10^{-6.5} \to 10^{-15.4}$).

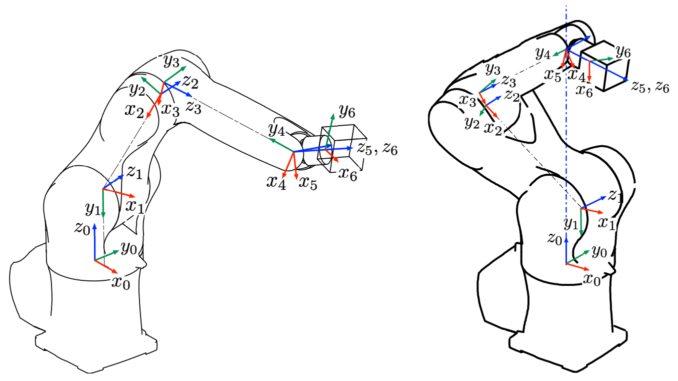

Fig. 2. The KUKA KR6 R700 robot, shown in a typical working configuration (left) and in the singular configuration of Section IV-D (right).

TABLE I
DH PARAMETERS FOR KUKA KR6 R700 MANIPULATOR

	Joint Angle [rad]	Link Offset [mm]	Link Length [mm]	Link Twist [rad]
1	q_1	183	25	$-\pi/2$
2	q_2	0	-315	0
3	q_3	0	-35	$\pi/2$
4	q_4	365	0	$-\pi/2$
5	q_5	0	0	$\pi/2$
6	q_6	80	0	0

Tool transform: $\log(\mathbf{T}_{\text{tool}}) = \left[150,\ 250,\ 100,\ 0,\ \frac{\pi}{2},\ 0\right]^{\mathsf{T}}$.

C. Overall Algorithm Complexity & Robustness

The QuIK method may converge faster than the Newton-Raphson method on an iteration-to-iteration basis, but this result does not necessarily imply that it will give better overall performance given that each iteration requires more computations. Better indicators are *total evaluation time* (CPU time to convergence) and *robustness* (how often the algorithms fail). To test these metrics, joint samples are generated as before but over a range of initial joint errors σ. Algorithms are run to with an exit error tolerance of $\varepsilon = 10^{-8}\,\text{rad}$. Results are averaged over $N = 10^7$ samples. This test is run for both the QuIK and NR methods, as well as the damped DQuIK and DNR algorithms, with $\lambda^2 = 10^{-5}$ (heavily damped) and $\lambda^2 = 10^{-7}$ (lightly damped). The BFGS algorithm is also included in this comparison. These results are plotted in Fig. 4, where the top plot shows the mean number of iterations required to converge and the middle plot shows the mean CPU time to convergence. Finally, the bottom plot shows the percentage of samples which failed to converge. For this test,

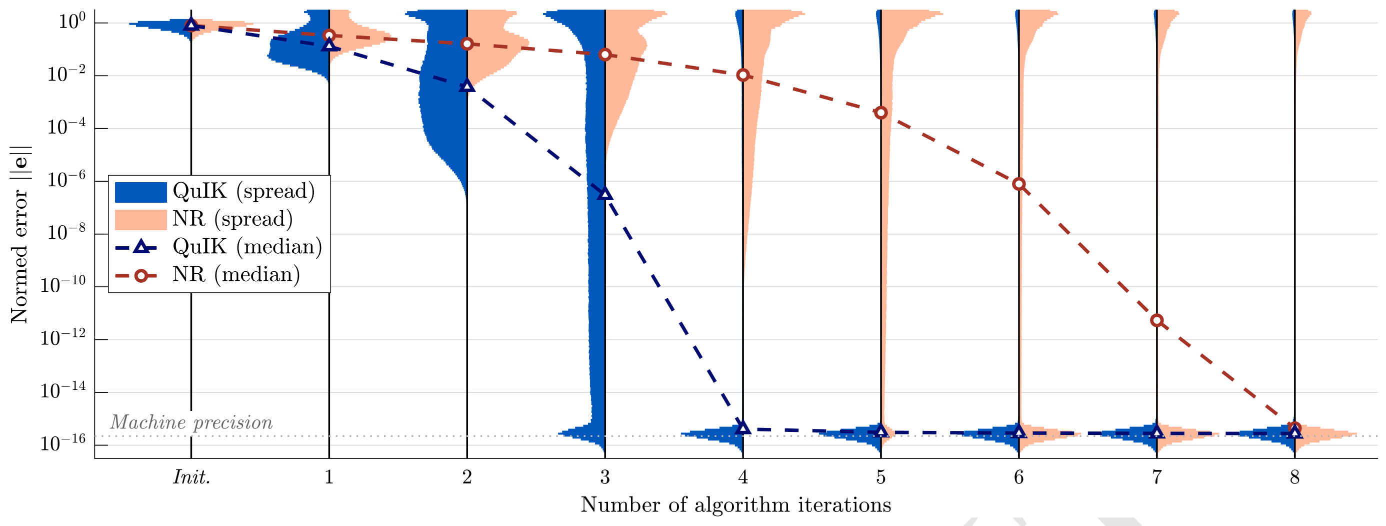

Fig. 3. Convergence of the Newton-Raphson and QuIK algorithms on a 6-DOF manipulator, with a mean initial guess error $\sigma = \pi/4$ and with $N = 10^6$ random poses. *Note: the histograms are scaled individually so that error distributions can be observed; thus, they are not to-scale relative to each other.*

a sample is considered converged if

$$||\mathbf{e}|| < \varepsilon' \, ||\mathbf{J}(\mathbf{q}^*)||, \tag{32}$$

where $\varepsilon' = 10^{-5}$ rad is a modified tolerance, and $||\mathbf{J}(\mathbf{q}^*)||$ is the norm-2 condition number of the geometric Jacobian at the target point. This modification avoids categorizing ill-conditioned configurations as "failed" when the algorithm in fact did converge successfully, within numerical limitations.

The results show the QuIK and DQuIK methods to be superior to their NR counterparts in all tests. The top figure shows the QuIK algorithms to require many fewer steps to converge than the NR methods, as expected. This difference is small for low initial joint error, but significant when σ is larger. This difference can be attributed to the larger basin of convergence of the third-order QuIK methods. Since the estimation at each step is more accurate than for the NR method, its convergence is stronger when further from the true solution. In the lightly damped case ($\lambda^2 = 10^{-7}$), the damped algorithms converged almost identically to their respective undamped versions. However, the heavily damped algorithms ($\lambda^2 = 10^{-5}$) are slower, and the difference between the DNR and DQuIK algorithms is also narrowed as the higher-order benefits are "masked" by the damping effects.

The middle plot in Fig. 4 shows that, despite the added complexity at each step of the iterations, the QuIK methods also outperforms the NR methods in terms of pure computational speed. As with the top plot, this difference is smaller for initial joint error, but more significant at larger σ. The lightly damped algorithms also performed nearly identically to the undamped versions, however for the heavily damped cases, speeds were slower, and the DNR algorithm performs better as the added accuracy of the DQuIK method is lost to the damping effects. The BFGS algorithm was slower in all cases, but was relatively better at higher initial errors.

In the bottom plot of Fig. 4, we show the reliability of the algorithms, as measured by the percentage of samples that failed to converge. Here, we see the clear superiority of the QuIK and DQuIK methods. The QuIK method failed 8 to 35x less frequently than the NR method, depending on the initial joint error σ. Within robotics, this trait is perhaps the most significant advantage of the QuIK and DQuIK methods. While they are faster and more efficient, they also provide a higher chance of convergence. For real-time control scenarios, where an inverse kinematics error is highly problematic, the lower chance of failures is a significant boon. Moreover, whereas algorithm speed can vary significantly based on implementational details, reliability is more intrinsically connected to the algorithm itself.

The lightly damped methods had better reliability than the undamped methods for low initial joint error, and had similar or slightly worse performance for high σ. The heavily damped algorithms had even better reliability for low σ, but were considerably less reliable for high initial joint error. The reason for this difference is that errors at good initial guesses are primarily caused by ill-conditioned configurations, which algorithm damping helps to correct. However, for poor initial guesses, errors are also caused by convergence to the wrong solution branch. Damping does not help with this issue, and instead only masks the valuable higher-order derivative information of the QuIK methods. Thus, both of the highly damped algorithms behave similarly to the undamped NR algorithm in these cases. The BFGS method had the worst reliability in all cases.

In practice, many applications allow for a good initial guess at the joint angles, and as such, the initial error σ that the algorithm needs to overcome is small, and the speed and reliability can be very good. For kinematically calibrated manipulators, an approximated closed-form solution can give a nearby configuration from which to iterate. In trajectory-following operations, the previously commanded joint angles are never very far from the joint angles for the current step. Other researchers have also proposed using pre-trained lookup tables or machine learning models to obtain approximate solutions, which could be given to these numerical algorithms

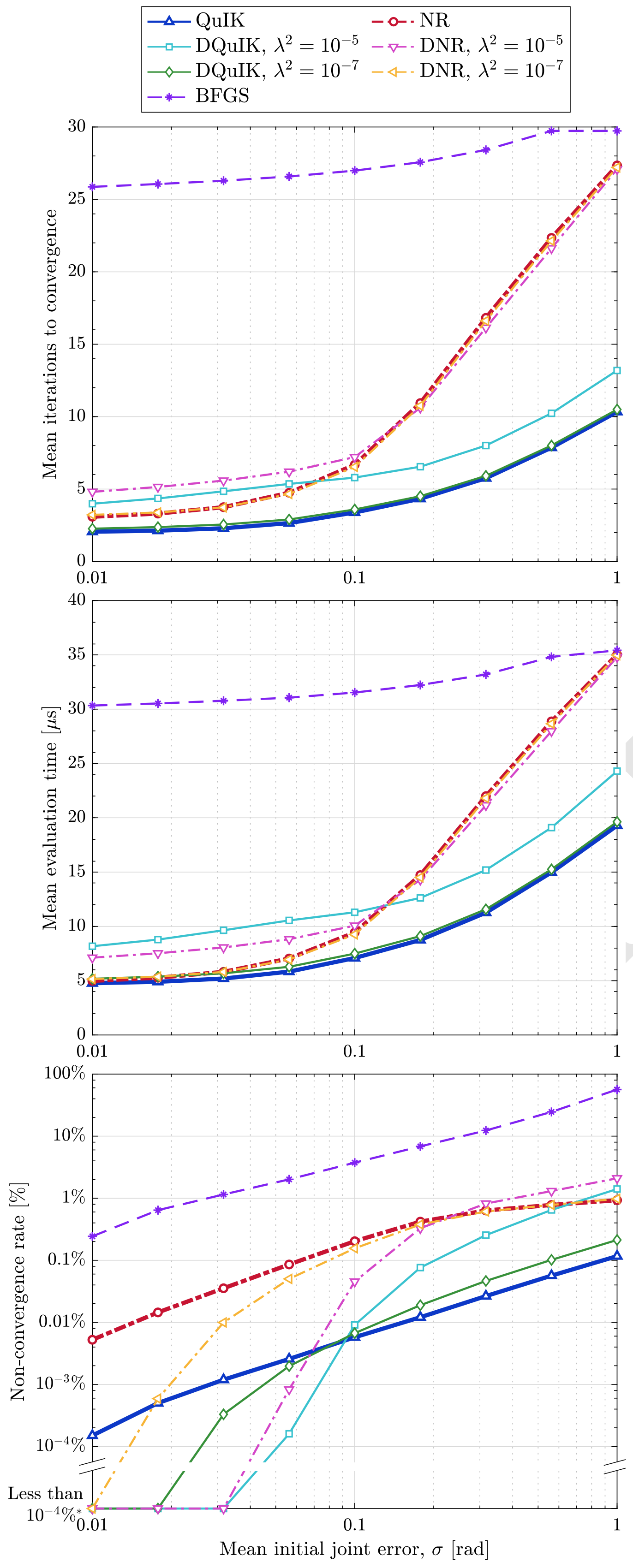

Fig. 4. Comparison of the mean iterations to convergence (top), mean evaluation time to convergence (middle) and convergence robustness (bottom), with a target tolerance $\varepsilon = 10^{-8}$ rad and averaged over $N = 10^7$ samples. * Error rates below 10^{-4}% are considered not statistically significant at the tested sample size, and are lumped with zero.

to converge quickly and reliably to a more exact answer [16].

The most common reasons given against the use of numerical inverse kinematic solutions are slow speed and lower reliability. However, with good initial guesses (e.g., $\sigma \approx 0.01$ rad), the speed of the QuIK method in our tests approached 5 µs, and the error rate approached 10^{-4}% (i.e., one error for every million samples). The lightly damped DQuIK method had similar speed and saw no errors in 10 million samples. This performance is both sufficiently fast and reliable for even high-rate control scenarios. Moreover, the performance comes very close to closed-form solutions. For example, the closed-form solution to the KUKA KR6 manipulator, in our testing, evaluated in approximately 0.7 µs — still outperforming the numerical algorithms, but not by a significant margin.

D. Singularity Handling

Singularities can affect iterative inverse kinematics in two ways. The first case is if the inverse kinematics algorithm starts at an ill-conditioned point, or encounters one as an intermediate point. This type of singularity can cause the algorithms to take large "jumps" and potentially diverge, or iterate into the incorrect solution basin. The second case is when the target point itself is ill-conditioned.

The first case is more straightforward to deal with, since the algorithm only needs to "get past" the given point without making large errors. One reason why the QuIK method may converge more reliably is its natural handling of singular or near-singular points. Consider Fig. 5, which plots, in one dimension, the behavior of the Newton-Raphson and Halley's method on a sinusoidal function, where the function gradient at x_0 is nearly zero. Here, the step size in the NR algorithm becomes very large, and the algorithm diverges. In Halley's method, however, the effect of the singularity is reversed. As $\nabla f(\mathbf{x})$ approaches zero,

$$\begin{aligned}\delta\mathbf{x} &= \lim_{\nabla f(\mathbf{x})\to\mathbf{0}} -\left[\nabla f(\mathbf{x}) + \tfrac{1}{2}\nabla^2 f(\mathbf{x})\,\delta\mathbf{x}_{\text{nr}}\right] \setminus f(\mathbf{x}) \\ &= \lim_{\nabla f(\mathbf{x})\to\mathbf{0}} -\left[\nabla f(\mathbf{x}) + \tfrac{1}{2}\nabla^2 f(\mathbf{x})\,\nabla f(\mathbf{x}) \setminus f(\mathbf{x})\right] \setminus f(\mathbf{x}) \\ &= \left[\mathbf{0} + \mathbf{0} \setminus f(\mathbf{x})\right] \setminus f(\mathbf{x}) \\ &\approx \left[\infty\right] \setminus f(\mathbf{x}) = \mathbf{0}. \end{aligned} \tag{33}$$

Thus, rather than taking large steps, Halley's method instead takes small steps until better knowledge of the function can be obtained.

The case where the target point $\mathbf{q}^*$ is singular is more challenging to deal with, and in these cases, the higher-order information of the QuIK methods is less helpful. Here, the damping methods presented in Section III-A show their worth. To quantify singular configuration reliability, we generate a single singular configuration where the wrist center of the manipulator is directly above the first joint axis, as shown in Fig. 2. This configuration is then perturbed using randomly generated values which are scaled using a numerical solver to achieve any arbitrary condition number, $||\mathbf{J}(\mathbf{q}^*)||$. A total of $N = 10^6$ samples are generated in this manner. These configurations are then further perturbed as in (30) with $\sigma = 0.1$ rad, to obtain a second set of configurations. We then run the algorithms twice – once using the singular configurations as

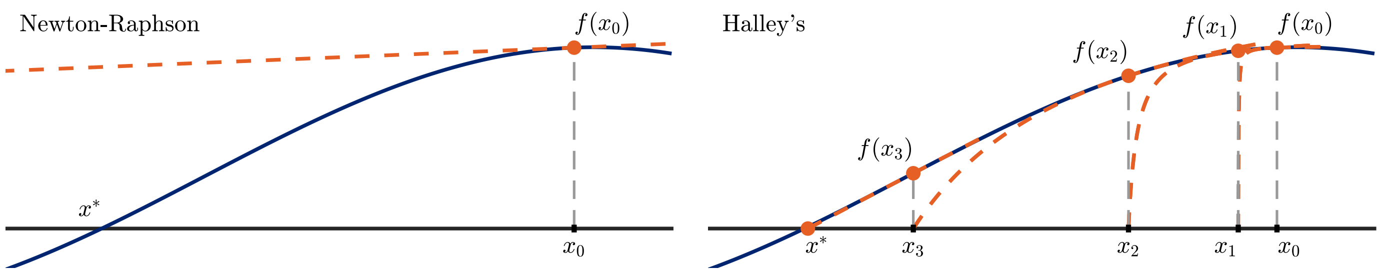

Fig. 5. Behavior of the Newton-Raphson (left) and Halley's method (right) when x_0 is near a singularity, $\nabla f(\mathbf{x}_0) = \mathbf{0}$. The NR method immediately diverges, however Halley's method instead takes small steps until the basin of convergence is reached. In this example, Halley's method will converge to the root even when x_0 is arbitrarily close to the singularity, so long as the gradient is not *exactly* zero.

the target, and the perturbed configurations as the initial guess, then again using the singular poses as the initial guess and the perturbed as the target. This setup allows quantification of the algorithm performance in both singularity situations — where the singularity is a midpoint of the algorithm (*singular start*) and where the singularity is the target (*singular target*). Fig. 6 plots the reliability for both situations over varying condition numbers $||\mathbf{J}(\mathbf{q})||$ using the same seven algorithms as in the previous section, with a requested accuracy of $\varepsilon = 10^{-12}$. Here we use a slightly different metric for convergence than in the previous section, since the condition numbers are so high as to render (32) meaningless. Instead, we use the condition $||\mathbf{e}|| < \varepsilon'$, with $\varepsilon' = 10^{-3}$ rad.

These results show that the undamped NR and QuIK methods both suffer from reliability issues near singularities, for both singular starting configurations and singular target configurations. The reliability is worst when the pose is nearly, but not identically singular ($||\mathbf{J}|| \approx 10^6$ to 10^{10}). This trend is expected, as the pseudoinverse is known to be well behaved exactly at singularities, but can command large joint adjustments when in the neighborhood of a singularity [13]. As expected, Fig. 6 shows the undamped QuIK method to be more robust than the NR method in handling singular starting configurations.

The damped algorithms generally avoid this unstable region. In all cases, higher damping improved the algorithm performance near singularities. For the DNR method, the lightly and heavily damped algorithms still occasionally failed in testing. However, the lightly damped DQuIK method experienced only rare failures, and the heavily damped DQuIK method did not experience any failures at all during testing. The downside of the damping term, of course, is slower general performance and a reduction in reliability when the initial guesses are poor and the configurations are non-singular. However, with proper tuning of the λ parameter, as discussed in further depth in [10], [12], [13], [35], damping is a beneficial trade-off to avoid poor performance in the undamped cases near the singularities. Methods of dynamically adjusting the damping can improve results in both scenarios [10], [12], [13], [18], [34]–[37].

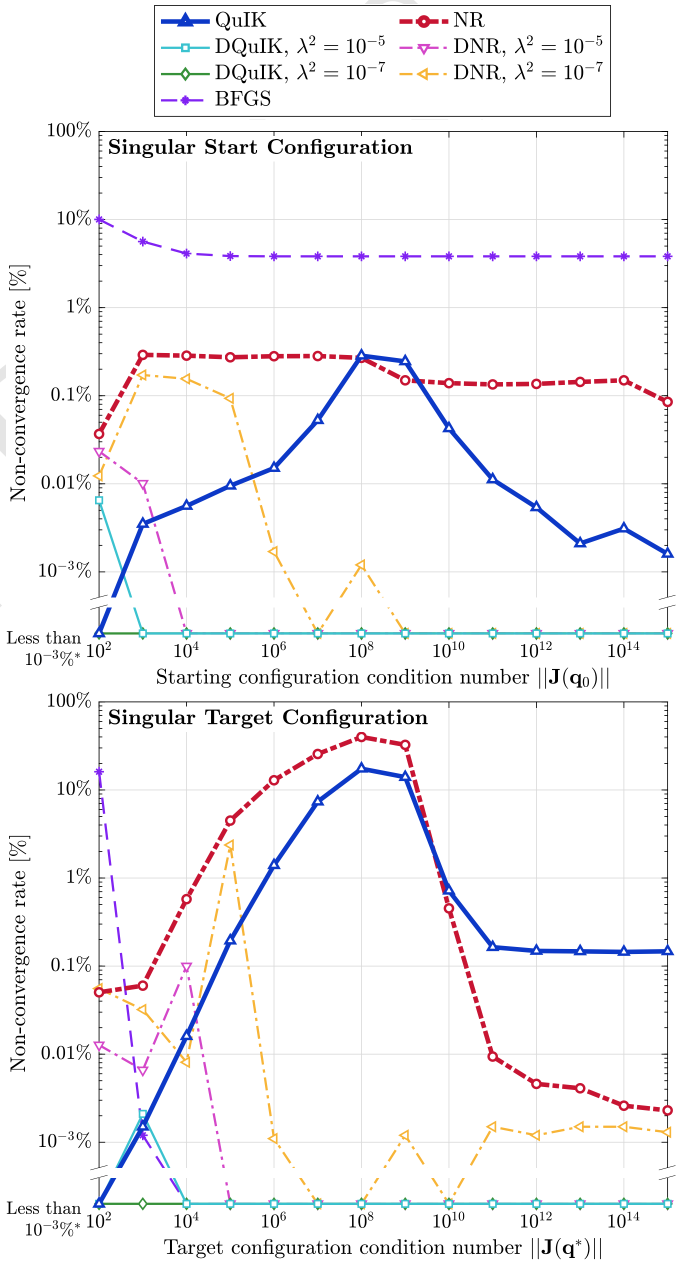

Fig. 6. Reliability of tested inverse kinematics algorithms at near-singular starting configurations (top), and at near-singular target configurations (bottom), with a requested accuracy $\varepsilon = 10^{-12}$ rad, an initial joint error $\sigma = 0.1$ rad, and averaged over $N = 10^6$ samples.
* Error rates below 10^{-3}% are considered not statistically significant at the tested sample size, and are lumped with zero.

E. Comparisons with Other Packages

In the final section of this paper, we show the performance of the proposed QuIK method against other state-of-the-art methods and on a variety of test manipulators. For this test, we emulate the testing procedure given in [7]. For each manipulator, we generate 10^6 random target configurations $\mathbf{q}^*$, uniformly distributed within the joint-space of each manipulator. The starting configurations $\mathbf{q}_0$ are taken as the home configuration of the tested robot, and is fixed for all samples. Requested accuracy is $\varepsilon = 10^{-8}$ rad and reliability is computed as in Section IV-C. We test the QuIK, DQuIK (with $\lambda^2 = 10^{-7}$), NR and BFGS algorithms from earlier, along with the following additional algorithms:

- *KDL-LMA* and *KDL-NR*: The open-source *Orocos Kinematics and Dynamics Library* (KDL) is arguably the most widely-used generic inverse kinematic solvers [7], [17], and is used in the ROS packages. KDL-LMA is their implementation of the automatically-damped Levenberg-Marquard algorithm. This algorithm is their primary inverse-kinematics algorithm. We also include KDL-NR, which is their implementation of the Newton-Raphson method, implemented without damping.
- *ML-BFGS* and *ML-LMA*: Mathworks' *Matlab Robotics Toolbox* features two inverse kinematics solvers. ML-BFGS is a BFGS solver, implemented as in [1] with some modifications. ML-LMA is a version of the Levenberg-Marquardt algorithm, implemented as in [12].

All algorithms are run from compiled C++ code. For the Matlab algorithms, automatic code generation (provided in Matlab) reduces the interpreted code into compiled C++ code prior to benchmarking. Each of the algorithms is tested on the KUKA KR6 manipulator defined earlier, as well as the following additional four manipulators.

- *KUKA KR6 (Perturbed DH)*: The KUKA KR6 robot from Fig. 2, with a "perturbed" DH table. Each parameter is modified by a small but non-negligible random number in the range ± 0.01. This chain gives an approximation of a manipulator after kinematic calibration, and will demonstrate that joint alignment is not required for these numerical algorithms.
- *KUKA iiwa7 r800*: A 7-DOF revolute-joint redundant manipulator.
- *Kinova Jaco*: A 6-DOF manipulator with a nonspherical wrist, which complicates closed-form solutions [3].
- *Boston Dynamics Atlas*: The Atlas humanoid robot [40] is used as a sample hyper-redundant chain. Testing is performed on the 16-DOF chain from the robot's right foot to right hand.

The full kinematic parameters of these manipulators can be found in the linked codebase [29]. Note that, as discussed in Section III-B, we do not implement any specific optimization strategy for redundant manipulators to select an "ideal" pose within the null-space of the manipulator — instead, we search for any solution which reduces the Cartesian error to zero. Further optimization could be accomplished through techniques such as task augmentation [20] or null-space projection [21], [22]; however, this is beyond the scope of the current work.

The results of this testing are given in Table II. Across all manipulators, and in terms of both evaluation speed and reliability, the QuIK and DQuIK methods outperforms other methods — often by one to two full orders of magnitude. In terms of speed, the QuIK algorithm computes in 21 to 35 µs, depending on manipulator. The next nearest algorithms were BFGS (1.1 to 3.1x slower), NR (1.0 to 1.8x slower), and KDL-LMA (3.9 to 7.2x slower). Both Matlab algorithms performed exceedingly slowly, with ML-BFGS being 329 to 941x slower and ML-LMA being 32 to 52x slower. It is to be expected that our implementations may be more efficient, as

TABLE II
RELATIVE SPEED AND RELIABILITY OF THE QUIK METHOD

	Mean Time*		Error Rate*	
KUKA KR6 (6-DOF)				
QuIK	21 µs		0.13%	
DQuIK	22 µs	(x 1.0)	0.24%	(x 1.8)
NR	38 µs	(x 1.8)	1%	(x 7.7)
BFGS	34 µs	(x 1.6)	61%	(x 460)
KDL-LMA	148 µs	(x 7.0)	5.3%	(x 40)
KDL-NR	425 µs	(x 20)	3.5%	(x 27)
ML-BFGS	14 794 µs	(x 697)	61%	(x 467)
ML-LMA	670 µs	(x 32)	1.1%	(x 8.6)
KUKA KR6 (Perturbed DH, 6-DOF)				
QuIK	24 µs		0.38%	
DQuIK	25 µs	(x 1.0)	0.82%	(x 2.2)
NR	41 µs	(x 1.7)	1.3%	(x 3.4)
BFGS	36 µs	(x 1.5)	62%	(x 164)
KDL-LMA	149 µs	(x 6.2)	6.8%	(x 18)
KDL-NR	452 µs	(x 19)	4.2%	(x 11)
ML-BFGS	15 015 µs	(x 628)	62%	(x 163)
ML-LMA	1 255 µs	(x 52)	1.4%	(x 3.7)
KUKA iiwa7 R800 (7-DOF)				
QuIK	16 µs		0%	
DQuIK	17 µs	(x 1.0)	0%	—
NR	25 µs	(x 1.5)	0.001%	(x ∞)
BFGS	51 µs	(x 3.1)	34%	(x ∞)
KDL-LMA	118 µs	(x 7.2)	0.005%	(x ∞)
KDL-NR	337 µs	(x 20)	0.085%	(x ∞)
ML-BFGS	15 520 µs	(x 941)	63%	(x ∞)
ML-LMA	670 µs	(x 41)	0.004%	(x ∞)
Kinova Jaco (Nonspherical Wrist, 6-DOF)				
QuIK	35 µs		0.66%	
DQuIK	37 µs	(x 1.0)	1.4%	(x 2.1)
NR	64 µs	(x 1.8)	4.4%	(x 6.8)
BFGS	41 µs	(x 1.1)	59%	(x 90)
KDL-LMA	177 µs	(x 5.0)	17%	(x 26)
KDL-NR	774 µs	(x 22)	14%	(x 21)
ML-BFGS	17 504 µs	(x 495)	64%	(x 98)
ML-LMA	1 268 µs	(x 36)	2.1%	(x 3.2)
Boston Dynamics Atlas (Foot to Hand, 16-DOF)				
QuIK	32 µs		0%	
DQuIK	32 µs	(x 1.0)	0%	—
NR	33 µs	(x 1.0)	0%	—
BFGS	110 µs	(x 3.4)	1.8%	(x ∞)
KDL-LMA	126 µs	(x 3.9)	0%	—
KDL-NR	267 µs	(x 8.4)	0%	—
ML-BFGS	10 498 µs	(x 329)	13%	(x ∞)
ML-LMA	1 484 µs	(x 47)	0%	—

* Factors are relative to the QuIK algorithm, which is the fastest and most robust across all tests.

we limit ourselves to serial manipulators, whereas Matlab and KDL work on any tree-structure robot, which introduces some overhead. Additionally, the KDL algorithms use an SVD-based linear solver, which is more numerically stable but slower. However, the speed differences are nonetheless considerable, and the SVD-based algorithms were still less reliable than our Cholesky-based implementations.

In terms of reliability, the QuIK and DQuIK methods again had the highest performance. The next nearest contenders were ML-LMA (3.2 to 8.6x more failures) and NR (3.4 to 7.7x more failures). That these algorithms would perform similarly is expected as they are mathematically very similar — however, Matlab's implementation is significantly slower. The BFGS algorithms (both BFGS and ML-BFGS) were not reliable, failing 90 to 467x more often, and the KDL algorithms (KDL-LMA and KDL-NR) had only moderate performance (11 to 40x more failures). The perturbed KUKA KR6 chain and the Kinova Jaco chain both are slightly more difficult to solve. All algorithms take longer and have slightly lower reliability for these robots, but the differences to the "aligned" KUKA KR6 robot are nonetheless relatively minor. We note that the testing methodology for Table II primarily tests the algorithms with poor initial guesses. Thus, per the trends observed in Fig. 4, we expect these results to slightly amplify the relative speed of the QuIK methods, but also reduce their relative reliability.

Finally, we can note that the performance gains of the QuIK and DQuIK method extend to the two redundant manipulators. It is noticeable that the inverse kinematics is more reliable on these redundant manipulators. This increased reliability is expected, as the number of constraints m has not changed, but more joint variables n are available to eliminate the error, thus facilitating the problem.

V. CONCLUSION

A novel inverse kinematics algorithm, the *Quick Inverse Kinematics* (QuIK) method, has been introduced. The QuIK algorithm is a *third-order*, iterative, numerical algorithm which adds a second-order derivative term to the Taylor series approximation of the forward kinematics function. The second-order derivative information was shown to exhibit significant symmetry and sparsity, speeding up computations. An extension to the QuIK method, called the *Damped Quick Inverse Kinematics* (DQuIK) method, was introduced to allow for stable behavior near ill-conditioned target configurations. The proposed algorithms were benchmarked in terms of evaluation speed, reliability, and singularity robustness against the Newton-Raphson method and several other modern inverse kinematics algorithms. The QuIK and DQuIK algorithms were able to converge faster and more reliably than the other tested methods. The damped DQuIK method was generally more reliable than the damped NR method reliable near singular configurations. The QuIK algorithms are proposed as faster and more robust "drop-in" replacements to the Newton-Raphson methods in inverse kinematics. A codebase is provided for public use, featuring C++ and Matlab implementations of the proposed algorithms, and allowing readers to reproduce the results given in the current work [29].

APPENDIX
THE MATRIX LOGARITHM

The matrix logarithm operation is included here, for completeness. These formulas originate from [12], [30] but have been modified for computational accuracy and notational consistency. The matrix logarithm operation is used to convert a homogeneous transformation matrix $\mathbf{T} \in SE(3)$ to a 6-element spatial twist $\mathbf{e} \in \mathbb{R}^6$ of the form

$$\mathbf{e} = \log(\mathbf{T}) = \begin{bmatrix}\mathbf{p}^\top & \boldsymbol{\omega}^\top\end{bmatrix}^\top, \tag{34}$$

where $\mathbf{p}$ is the linear translation of $\mathbf{T}$, and $\boldsymbol{\omega}$ is the angle-axis representation of the rotational component, $\mathbf{R} \in SO(3)$. Define the intermediate 3-vector $\boldsymbol{\epsilon}$ from the entries of $\mathbf{R}$ as

$$\boldsymbol{\epsilon} = \begin{bmatrix}r_{32} - r_{23} & r_{13} - r_{31} & r_{21} - r_{12}\end{bmatrix}^\top. \tag{35}$$

If $\mathbf{R}$ is not a diagonal matrix, then $\boldsymbol{\epsilon}$ is non-zero and $\boldsymbol{\omega}$ can be computed directly as [12]

$$\boldsymbol{\omega} = \operatorname{atan2}\big(||\boldsymbol{\epsilon}||, \operatorname{trace}(\mathbf{R}) - 1\big)\, ||\boldsymbol{\epsilon}||^{-1}\, \boldsymbol{\epsilon}. \tag{36}$$

If $||\boldsymbol{\epsilon}||$ is nearly zero, then $\mathbf{R}$ is either nearly the identity, in which case $\operatorname{trace}(\mathbf{R}) \approx 3$, or nearly rotation of π radians about the x–, y– or z–axes, in which case $\operatorname{trace}(\mathbf{R}) \approx -1$. In the former case, we use a Taylor-series approximation to avoid divide-by-zero errors which reduces to

$$\boldsymbol{\omega} \approx \big(\tfrac{3}{4} - \tfrac{1}{12}\operatorname{trace}(\mathbf{R})\big)\boldsymbol{\epsilon}. \tag{37}$$

In the latter case, numerical precision is less important since for rotational errors this large, numerical algorithms will not take accurate steps regardless of $\boldsymbol{\omega}$, and a rotation of π rad about *any* axis will reduce error nearly to zero. An acceptable approximation in this case is [12]

$$\boldsymbol{\omega} \approx \tfrac{\pi}{2}\big[\operatorname{diag}(\mathbf{R}) + 1\big]. \tag{38}$$

REFERENCES

[1] J. Zhao and N. Badler, "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures," *ACM Trans. Graph.*, vol. 13, no. 4, pp. 313–336, 1994.

[2] A. A. Canutescu and R. L. Dunbrack, "Cyclic coordinate descent: A robotics algorithm for protein loop closure," *Protein Sci.*, vol. 12, no. 5, pp. 963–972, 2003.

[3] A. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. C. L'Ecuyer, "Kinova Modular Robot Arms for Service Robotics Applications," *Int. J. Robot. Appl. Technol.*, vol. 5, no. 2, pp. 49–71, 2018.

[4] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse Kinematics Techniques in Computer Graphics: A Survey," *Comput. Graph. Forum*, vol. 37, no. 6, pp. 35–58, sep 2018.

[5] I. M. Chen, G. Yang, and I. G. Kang, "Numerical inverse kinematics for modular reconfigurable robots," *J. Robot. Syst.*, vol. 16, no. 4, pp. 213–225, 1999.

[6] L.-C. Wang and C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *IEEE Trans. Robot. Autom.*, vol. 7, no. 4, pp. 489–499, 1991.

[7] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," *IEEE-RAS Int. Conf. Humanoid Robot.*, vol. 2015-Decem, pp. 928–935, 2015.

[8] W. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *23rd IEEE Conf. Decis. Control*, no. December. IEEE, dec 1984, pp. 1359–1363.

[9] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Trans. Man-Machine Syst.*, vol. 10, no. 2, pp. 47–53, 1969.

[10] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 108, no. 3, pp. 163–171, 1986.
[11] C. W. Wampler, "Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods," *IEEE Trans. Syst. Man Cybern.*, vol. 16, no. 1, pp. 93–101, 1986.
[12] T. Sugihara, "Solvability-Unconcerned Inverse Kinematics by the Levenberg–Marquardt Method," *IEEE Trans. Robot.*, vol. 27, no. 5, pp. 984–991, oct 2011.
[13] S. R. Buss and J.-S. Kim, "Selectively Damped Least Squares for Inverse Kinematics," *J. Graph. Tools*, vol. 10, no. 3, pp. 37–49, 2005.
[14] A. El-Sherbiny, M. A. Elhosseini, and A. Y. Haikal, "A comparative study of soft computing methods to solve inverse kinematics problem," *Ain Shams Eng. J.*, vol. 9, no. 4, pp. 2535–2548, 2018.
[15] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, "A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots," *2017 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2017*, vol. 2018-Janua, pp. 1818–1823, 2018.
[16] N. Vahrenkamp, D. Muth, P. Kaiser, and T. Asfour, "IK-Map: An enhanced workspace representation to support inverse kinematics solvers," *IEEE-RAS Int. Conf. Humanoid Robot.*, vol. 2015-Decem, no. 611909, pp. 785–790, 2015.
[17] H. Bruyninckx, "Open robot control software: The OROCOS project," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 2523–2528, 2001.
[18] A. Colome and C. Torras, "Redundant inverse kinematics: Experimental comparative review and two enhancements," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 5333–5340, 2012.
[19] T. R. Scavo and J. B. Thoo, "On the Geometry of Halley's Method," *Am. Math. Mon.*, vol. 102, no. 5, p. 417, may 1995.
[20] O. Egeland, "Task-Space Tracking with Redundant Manipulators," *IEEE J. Robot. Autom.*, vol. 3, no. 5, pp. 471–475, 1987.
[21] A. Leigeois, "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," *IEEE Trans. Syst. Man. Cybern.*, vol. 7, no. 12, pp. 868–871, 1977.
[22] T. Chan and R. Dubey, "A weighted least-norm solution based scheme for avoiding joint limits for redundant manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE Comput. Soc. Press, 1993, pp. 395–402.
[23] Y. Nakamura and H. Hanafusa, "Optimal Redundancy Control of Robot Manipulators," *Int. J. Rob. Res.*, vol. 6, no. 1, pp. 32–42, 1987.
[24] A. S. Householder, *The Numerical Treatment of a Single Nonlinear Equation*. New York: McGraw-Hill, 1970.
[25] G. Gundersen and T. Steihaug, "On large-scale unconstrained optimization problems and higher order methods," *Optim. Methods Softw.*, vol. 25, no. 3, pp. 337–358, jun 2010.
[26] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic Press, 1970.
[27] W. Rheinboldt, *Methods for Solving Systems of Nonlinear Equations*. SIAM, 1998.
[28] M. Thomas and D. Tesar, "Dynamic modeling of serial manipulator arms," *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 104, no. 3, pp. 218–228, 1982.
[29] S. Lloyd, R. Irani, and A. Mojtaba, "Github Repository for the QuIK Algorithm," 2022. [Online]. Available: https://github.com/CarletonABL/QuIK
[30] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1994, vol. 29.
[31] M. W. Spong and M. Vidyasagar, *Robot dynamics and control*. John Wiley and Sons, 2008.
[32] A. Hourtash, "The kinematic Hessian and higher derivatives," *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom. CIRA*, pp. 169–174, 2005.
[33] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.
[36] R. V. Mayorga, N. Milano, and A. K. Wong, "A fast procedure for manipulator inverse kinematics evaluation and pseudoinverse robustness," *Proc. IEEE Int. Work. Intell. Motion Control. IMC 1990*, vol. 2, no. 4, pp. 787–792, 1990.
[34] L. Kelmar and P. Khosla, "Automatic generation of kinematics for a reconfigurable modular manipulator system," in *Proceedings. 1988 IEEE Int. Conf. Robot. Autom.* IEEE Comput. Soc. Press, 1988, pp. 663–668.
[35] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator," *IEEE Trans. Control Syst. Technol.*, vol. 2, no. 2, pp. 123–134, 1994.
[37] A. A. Maciejewski and C. A. Klein, "Numerical filtering for the operation of robotic manipulators through kinematically singular configurations," *J. Robot. Syst.*, vol. 5, no. 6, pp. 527–552, 1988.
[38] J. R. Bunch and J. E. Hopcroft, "Triangular factorization and inversion by fast matrix multiplication," *Math. Comput.*, vol. 28, no. 125, pp. 231–231, jan 1974.
[39] C. Lagarias, Jeffrey, A. Reeds, James, H. Wright, Margaret, and E. Wright, Paul, "Convergence properties of the nelder–mead simplex method in low dimensions," *SIAM J. Optim.*, vol. 9, no. 1, pp. 112–147, 1998.
[40] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Auton. Robots*, vol. 40, no. 3, pp. 429–455, 2016.

Steffan Lloyd received his B. Eng. degree in mechanical engineering from Carleton University, Ottawa, Canada in 2015. He worked as a mechanical designer in aerospace robotics at Advanced Integration Technologies (AIT) in Umeå, Sweden from 2016-2018. He is currently pursuing a Ph.D. in mechanical engineering at Carleton University under the supervision of M. Ahmadi and R. Irani.

Rishad A. Irani received the B.A.Sc. degree in mechanical engineering from the University of Windsor, Windsor, Canada, in 2003, and the M.A.Sc. and Ph.D. degrees in mechanical engineering from Dalhousie University, Halifax, Canada, in 2006 and 2011, respectively. He joined Carleton University in January 2016. Previously, he worked with Rolls-Royce Canada Limited. His current research focuses on modeling and mechatronic applications for marine systems.

M. Ahmadi received the B.Sc. degree from Sharif University of Technology, Tehran, Iran, in 1989, the M.Sc. degree from the University of Tehran, Iran, in 1992, and the Ph.D. degree in mechanical engineering from McGill University, Montreal, Canada, in 1998. He joined Carleton University, Ottawa, ON, Canada in 2005, where he is currently a Professor. He founded the Advanced Biomechatronics and Locomotion Lab. His current interests include rehabilitation robotics, human-robot interaction, design of mechatronics systems.