TRAC-IK: An Open-Source Library for Improved Solving of Generic Inverse Kinematics

Patrick Beeson and Barrett Ames

***Abstract*— The Inverse Kinematics (IK) algorithms implemented in the open-source Orocos Kinematics and Dynamics Library (KDL) are arguably the most widely-used generic IK solvers worldwide. However, KDL's only joint-limit-constrained IK implementation, a pseudoinverse Jacobian IK solver, repeatedly exhibits false-negative failures on various humanoid platforms. In order to find a better IK solver for generic manipulator chains, a variety of open-source, drop-in alternatives have been implemented and evaluated for this paper. This article provides quantitative comparisons, using multiple humanoid platforms, between an improved implementation of the KDL inverse Jacobian algorithm, a set of sequential quadratic programming (SQP) IK algorithms that use a variety of quadratic error metrics, and a combined algorithm that concurrently runs the best performing SQP algorithm and the improved inverse Jacobian implementation. The best alternative IK implementation finds solutions much more often than KDL, is faster on average than KDL for typical manipulation chains, and (when desired) allows tolerances on each Cartesian dimension, further improving speed and convergence when an *exact* Cartesian pose is not possible and/or necessary.**

I. INTRODUCTION

Kinematics equations form the basis for all humanoid manipulation research and are fundamental to the dynamics equations needed for balancing, walking, and real-world tool usage. Inverse Kinematics, when given the configuration of a robot, provides a set of joint values that reach a desired Cartesian pose for the robot's end effectors. There are both analytical and numerical solutions for Inverse Kinematics. Analytical solutions suffer from an inability to generalize to tool-use scenarios or changes in robot configuration, as the solver must be constructed beforehand. Typically, Numerical IK solvers are more generic in that they rely on a frequent, runtime approximation of the local inverse Jacobian in order to try to find joint solutions that come "close enough" to the desired Cartesian solution. Numerical IK methods use the Newton method or similar to iterate until the solution is found. While theoretically sound, numerical approaches can be quite slow compared to analytical approaches, and thus there is active research to try and speed up the computation of the Jacobian, speed up the matrix inversion, and converge to a quality solution without getting continuously stuck in bad local minima [1].

Arguably the most commonly used numerical IK implementation in the robotics community today is the joint-limit-constrained pseudoinverse Jacobian solver found in the Orocos Kinematics and Dynamics Library (KDL).[1] KDL contains the kinematics framework used by various popular ROS packages and by the popular MoveIt! planning library.[2]

Despite its popularity, KDL's IK implementation[3] exhibits numerous false-negative failures on a variety of humanoid and mobile manipulation platforms. In particular, KDL's IK implementation has the following issues:

1) frequent convergence failures for robots with joint limits,
2) no actions taken when the search becomes "stuck" in local minima,
3) inadequate support for Cartesian pose tolerances,
4) no utilization of tolerances in the IK solver itself.

As detailed in this paper, issues 2–4 above can be mitigated by simple implementation enhancements to KDL's inverse Jacobian solver. Issue 1, however, requires consideration of alternative IK formulations.

The remainder of this paper will describe a variety of IK implementations created for this work, then compare the algorithm solve rates and runtime speeds of these implementations on various humanoid platforms. Section II describes six implemented IK algorithms; included are simple enhancements that drastically improve the KDL solver, an alternative non-linear optimization formulation for IK with several different metrics, and a final algorithm that combines multiple metrics to consistently outperform the alternatives in both average IK solves and average runtime. The methodology for a rigorous quantitative comparison is discussed in Section III. Comparison results are detailed in Sections IV–VI. All of the algorithms are open-source, "drop in" replacements to KDL's `ChainIkSolverPos_NR_JL` class (also referred to as "Plain `KDL`" in Tables I–III) and are available for download at `https://bitbucket.org/traclabs/trac_ik/`.

II. CANDIDATE IK ALGORITHMS

Inverse Jacobian methods are quite elegant in their simplicity. Given a seed value for joints q_{seed} (often the current joint values), *Forward Kinematics* can be used to compute 1) the Cartesian pose for the seed, 2) the Cartesian error vector p_{err} between the seed pose and the target pose, and 3) the Jacobian J which defines the partial derivatives in Cartesian space with respect to the current joint values. After

This work was performed at TRACLabs Inc. in Houston, Texas, and was supported by NASA Contracts NNX14CJ19P & NNX15CJ06C.

The open-source extensions/alternatives to the KDL IK solver can be downloaded from `https://bitbucket.org/traclabs/trac_ik/`.

Please send any correspondence to `pbeeson@traclabs.com`.

[1]KDL: `http://www.orocos.org/wiki/orocos/kdl-wiki/`.

[2]MoveIt!: `http://moveit.ros.org/`.

[3]In this paper, the *KDL IK solver* specifically refers to the `ChainIkSolverPos_NR_JL` class, which handles robot chains that contain joints with hard limits.

inverting the Jacobian, J^{-1} defines the partial derivatives in joint space with respect to Cartesian space.[4] Consequently, an Inverse Kinematics solution is simply computed by iterating the function

$$q_{next} = q_{prev} + J^{-1} p_{err}, \quad (1)$$

where Forward Kinematics of q_{next} is used to compute the new value of p_{err}. When all elements of p_{err} fall below a stopping criteria, the current joint vector q is the returned IK solution.

As mentioned in Section I, the widely-used KDL implementation of pseudoinverse Jacobian IK for robots with joint limits has several issues when used on real-world robotic configurations. Some of these issues can be overcome by simply re-implementing the KDL solver to provide a more usable API and more robust overall behavior. Other issues are more fundamental, in that they violate assumptions in the underlying Newton-method search. Six alternative IK algorithms implemented for this paper are described next, called `KDL-RR`, `SQP`, `SQP-DQ`, `SQP-SS`, `SQP-L2`, and `TRAC-IK`.

A. The `KDL-RR` Algorithm

Some simple enhancements can be made to KDL's pseudoinverse Jacobian solver that increase its performance dramatically. First, the KDL Inverse Kinematics API takes as input a maximum number of iterations to try when searching for an IK solution. In general, the computational time to compute and utilize the Jacobian will depend on the size and complexity of the manipulation chain; thus, there is no way for a user to really understand how the number of iterations for a chain might result in IK compute time. This might not only affect computational performance, but it makes it difficult to compare the KDL implementation with other IK solvers. It is quite easy to simply re-implement KDL's IK solver to loop for a maximum time rather than a maximum number of counts. This is functionally equivalent to running KDL with a number of iterations, but the maximum number of iterations is dynamically determined based on the user-desired solve time.

Second, and more importantly, Inverse Jacobian IK implementations can get stuck in local minima during the iteration process. This is especially true in cases where joints have physical limits on their range. In the KDL implementation, there is nothing to detect local minima or mitigate this scenario; however, local minima are easily detectable when $q_{next} - q_{prev} \approx 0$. In an improved implementation of pseudoinverse Jacobian IK, local minima are detected and mitigated by changing the next seed. Consequently, the performance of the KDL IK solver can be significantly improved by simply using random seeds for q to "unstick" the iterative algorithm when local minima are detected. This implementation of KDL with *random restarts* and a time-based run loop is what will be referred to as `KDL-RR` throughout the rest of this paper.

[4]In KDL, the Moore-Penrose inverse of J is used because it is more efficient to compute.

B. The `SQP` IK Algorithm

As will be detailed in Section IV, despite the drastic improvements that simply adding random restarts makes to the KDL IK solver, the failure rate for many robotic chains was observed to still be too high. Analysis of the failures in `KDL-RR` showed that many failures happen when the iterative algorithm seems to be making progress ($\Delta q \not\approx 0$) but is encountering joint limits in the robot configuration.

In theoretical applications, where joints have unbounded motion, inverse Jacobian algorithms work quite well; however, in practice, many robotic joints have hard limits. By iteratively multiplying the inverse Jacobian, it is possible to set the joint values beyond their limits. Thus after each iteration, the KDL implementation must clamp the values of q_{next} to the joint limits. Analysis from `KDL-RR` showed that this clamping actually occurs repeatedly, as the inverse Jacobian mathematics wants to push a joint through its limit to reduce the Cartesian error. Thus, joint limits make the search space non-smooth, lead the solver down "garden paths", and cause the IK solver to fail in scenarios where solutions do exist.

One way to avoid these failures is to solve IK using methods that better handle constraints like joint limits. For instance, IK as a nonlinear optimization problem can be solved locally using sequential quadratic programming (SQP) [2]. SQP is an iterative algorithm for nonlinear optimization, which can be applied to problems with objective functions and constraints that are twice continuously differentiable.

As a comparison algorithm to `KDL-RR`, an IK solver using previously cited SQP formulations [2], [3] was implemented. This algorithm is referred to as `SQP` throughout the rest of this paper. It is characterized as follows:

$$\begin{aligned} \arg\min_{q \in \mathbf{R}^n} \ & (q_{seed} - q)^T (q_{seed} - q), \quad (2) \\ s.t. \quad & f_i(q) \leq b_i, \ i = 1, ..., m, \end{aligned}$$

where q_{seed} is the n-dimensional seed value of the joints, and the inequality constraints $f_i(q)$ are the joint limits, the Euclidean distance error, and the angular distance error.

This SQP problem is evaluated using the NLopt library, which is open source, compiles out of the box on Ubuntu Linux 12.04, and can be installed via `apt-get` in the standard Ubuntu Linux 14.04.[5] Using the NLopt C++ API, it was possible to create a "drop in" replacement for KDL's IK solver that uses the SLSQP [4] algorithm to implement Equation 2.

NLopt's implementation of SLSQP uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm as its iterative search method. BFGS is a Newton-approximation method, so one might expect to see similar results to KDL's pseudoinverse Jacobian IK, which is using Newton's method; however unlike Newton's method, BFGS has proven to have good performance even for non-smooth optimizations [5]. Despite the improved performance in non-smooth optimizations, BFGS can still get stuck in local minima. In order to properly compare the sequential quadratic programming IK algorithm

[5]NLopt: `http://ab-initio.mit.edu/wiki/index.php`

with the alternatives in this paper, SQP also incorporates the same local minima detection and random restarts as implemented in KDL-RR.

C. The SQP-DQ IK Algorithm

As will be detailed in Section IV, the SQP algorithm performs poorly when thoroughly tested, despite its use in previously published work. Because SQP is focusing on minimizing the overall amount of joint movement and is only considering Cartesian error as a *constraint*, it is biasing the search around the joint space of the seed value, rather than making constant progress towards minimizing the overall Cartesian error.

In another alternative IK algorithm, the nonlinear optimization formulation was changed to minimize Cartesian pose error directly. In this case, only the joint limits continue to be constraints to the SQP formulation in Equation 2, while Cartesian error is the function being minimized, with no regard as to the amount of joint motion needed. In order to do this, a scalar distance metric for Cartesian error is needed. A distance metric from screw theory that combines translation and orientation into a single-unit space using *dual quaternions* [6] was used for the SQP-DQ IK algorithm.

In SQP-DQ, the objective function for minimizing the distance between two poses defined by dual quaternions [7] is:

$$\phi_{DQ} = 4((\log \hat{e}) \cdot (\log \hat{e})^T), \tag{3}$$

where $\hat{e}$ is the dual quaternion that describes the error between the target and current poses. In essence, the $\log$ operation on the eight-element dual quaternion vector ($\hat{e}$) maps the pose to an eight-element axis and angle formulation. The dot product (including the scalar coefficient) then provides the squared magnitude of the difference between the two poses.

D. The SQP-SS and SQP-L2 IK Algorithms

SQP-DQ uses dual quaternion error as a combined measure of distance error and angular error in Cartesian space. Generating the dual quaternion error does take non-trivial computational time compared to simpler metrics. As such, SQP-DQ was compared against two alternative SQP formulations that minimize the Cartesian error using two very simple metrics—*Sum of Squares* error for the 6-element vector p_{err} and *L2-Norm* of that Cartesian error vector. Specifically, the objective function being minimized in the SQP formulation for SQP-SS is

$$\phi_{SS} = p_{err} \cdot p_{err}^T, \tag{4}$$

and the function being minimized for SQP-L2 is

$$\phi_{L2} = \sqrt{p_{err} \cdot p_{err}^T}. \tag{5}$$

Although choosing between two joint candidates based on minimum error will always yield the same answer when using *Sum of Squares* and *L2-Norm*, these different metrics do affect the gradients of the search space and can yield slightly different results in practice.

E. The TRAC-IK Algorithm

Section IV will detail the improved performance of the various SQP IK methods over inverse Jacobian methods on a variety of humanoid chains. In particular, SQP-SS (one of the algorithms that was implemented as a baseline that was not expected to outperform SQP-DQ) was the top overall algorithm in terms of IK solve rate—achieving comparable success rates across a large number of IK solves. However, SQP-SS (like all the SQP-based nonlinear optimization implementations) can have much longer solve times than the pseudoinverse Jacobian methods for certain robotic configurations. Consequently, a final IK solver, called TRAC-IK, was implemented to leverage concurrency to improve on the overall solve rate while keeping computation time relatively low. For a single IK solver request, TRAC-IK spawns two solvers, one running SQP-SS and one running KDL-RR. Once either finishes with a solution, both threads are immediately stopped, and the resulting solution is returned.

III. QUANTITATIVE TEST METHODOLOGY

In order to quantify the performance of the different IK methods, an offline automated test process was used. This allowed a large, statistically-valid number of random samples to be used as inputs. Kinematics models of several different robots were used to demonstrate the generality of the methods. The comparison entailed:

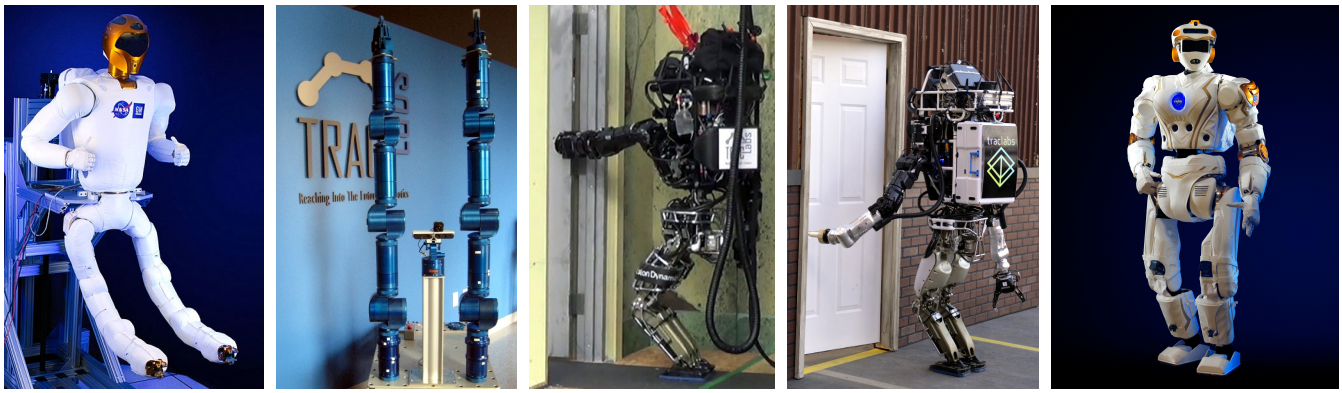
1) selecting a kinematics chain of joints to test;
2) computing a *nominal* set of joint angles for the chain by finding the midpoint between the joint limits;
3) selecting a random joint value (within joint limits) for each joint in the chain;
4) performing Forward Kinematics to determine the target Cartesian pose from the random joints;
5) calling each IK implementation for the target pose, using the nominal joint values as the seed;
6) repeating steps 3–5 above, while summing the number of successful IK solves for each IK implementation and summing the time in microseconds for each successful IK solve to complete;
7) averaging the success rate and solve time for each IK implementation after 10,000 random samples.

A solve was considered successful if the difference in each dimension between the pose and the target pose is no more than ϵ=1E-6 in any of the dimensions.[6] The maximum timeout used in all comparison testing was 5 milliseconds per solve. The robot models that were used for comparison are shown in Figure 1.

IV. QUANTITATIVE IK RESULTS

Below are detailed comparisons of the six alternative IK algorithms discussed in Section II using the methodology detailed in Section III. All experimental data is averaged over

[6]While some humanoids, like the hydraulic-actuated Atlas, cannot achieve 1E-6 precision in control, other robots, like industrial robotic arms and the TRACBot arm, can achieve such precision. Thus IK algorithms are compared using high precision error.



(a) Robonaut2 (b) TRACBot (c) Atlas 2013 (d) Atlas 2015 (e) Valkyrie

Fig. 1. The five robot models used for testing. The Robonaut2 humanoid has 7-DOF arms and 7-DOF "legs" (used as arms to navigate the International Space Station) and a 1-DOF waist. The TRACBot mobile manipulation robot has modular N-DOF arms (here 6- and 7-DOF arms are tested) mounted on a differential drive base. The Atlas robot (circa the 2013 DRC Trials) has 6-DOF arms and a 3-DOF set of back/waist joints. The Atlas robot (circa the 2015 DRC Finals) has 7-DOF arms and a 3-DOF set of back/waist joints. The Valkyrie robot has 7-DOF arms and a 3-DOF back/waist.

10,000 random samples for each robot configuration. The desired Cartesian error was 1E−6, and the timeout was 5 ms.

A. Solving IK for Typical Manipulation Chains

Table I details the comparison of the success rates of the six Inverse Kinematic algorithms for standard manipulation chains of different humanoid platforms.

1) `KDL-RR`*:* It is clear (from the first two gray columns) that many more IK solutions can be found by simply adding random restarts when the regular KDL solver stops making progress. For example, just for the 6-DOF arms on the 2013 Atlas humanoid (see Figure 1(c)), the percentage of IK solves (to 10,000 known-reachable Cartesian poses from the nominal joint space with a maximum of 5 ms allowed) went from 75.53% to 90.66%. Likewise, there was an increase from 44.83% success using the stock KDL IK implementation to 82.1% success with `KDL-RR` on the 7-DOF arms of the Valkyrie humanoid (see Figure 1(e)).

It is clear that despite improving upon the stock KDL IK implementation, `KDL-RR` still performs poorly on many test cases. Some examples of the poor performance of `KDL-RR` are for the 7-DOF arm chains of the Robonaut2 grasping "legs" (77.37% solve rate), the Atlas 2015 7-DOF arms (76.95% solve rate), and the TRACBot 6-DOF arms (71.86% solve rate).

2) SQP variants: For the most part, the various SQP algorithms demonstrate the improvements made by using nonlinear optimization algorithms, which better handle discontinuities in the search space caused by joint limits, over inverse Jacobian methods that mathematically assume no joint limits.

The exception, `SQP`, performed poorly as expected. Despite its use in previously cited work, the `SQP` algorithm spends a lot of time attempting to minimize the overall joint motion rather than trying to minimize the Cartesian error. Initial, unpublished experimental results show that `SQP` performs much better when the desired Cartesian error is high ($\epsilon \geq$1E−3) and when the number of degrees of freedom are small.

On the other hand, `KDL-RR` was outperformed by the all the SQP methods that minimize Cartesian error directly. In particular, `SQP-SS` seems to consistently outperform all the SQP-based algorithms. In fact, `SQP-SS` and `SQP-L2`, which conflate orientation and position errors in a single error metric, were not expected to outperform `SQP-DQ`, which in theory better models a unified distance metric of 3D pose error. More analysis would be needed to fully explore why dual quaternions were outperformed by a simple dot product of the Cartesian error vector. An initial hypothesis is that this may be from the large computational overhead needed to compute the dual quaternion error (and gradients) on each iteration of the algorithm; thus, while `SQP-DQ` often converges very quickly to the right answer, sometimes it is stopped by the 5 ms timeout.

3) Runtime analysis: In addition to solve rates, the runtimes associated with each of these algorithms was recorded. Table I (non-gray columns) details that `SQP-L2` linearly increases in computation time with the length of the chain.[7] On the other hand, the runtime of `SQP-DQ` and `SQP-SS` is not a function of chain length (or degrees of freedom).

Of additional importance, the runtime results show that when `KDL-RR` does have a high success rate, it is correlated with a low solve time. That is, in the cases where pseudoinverse Jacobian solutions can be found without hitting joint limits in the search, those solutions are found very quickly. This, along with the fact that `SQP-SS` is the clear winner in terms of solve rate despite being relatively constant in runtime over different length chains on the same robot, inspired the combined `TRAC-IK` algorithm that attempts to equal or surpass the solve rates of `SQP-SS` while achieving solve speeds closer to `KDL-RR`.

[7]Presumably, Forward Kinematics from longer chains yield flatter Cartesian spaces in the L2-Norm space, which result in smaller gradients, thus increasing the gradient-based search time.

TABLE I

A COMPARISON OF THE SOLVE RATES FOR THE SIX IK ALGORITHMS TESTED ACROSS VARIOUS MANIPULATIONS CHAINS ON FOUR DIFFERENT ROBOT PLATFORMS. THE POSES WERE RANDOM (YET KNOWN REACHABLE) CARTESIAN POSES, SEEDED BY THE NOMINAL JOINT CONFIGURATION. FOR ROBONAUT2, IK ALGORITHMS WERE TESTED ON THE 7-DOF ARMS WITH AND WITHOUT THE WAIST JOINT. THE 7-DOF ROBONAUT2 "LEGS" WERE ALSO TESTED, AGAIN WITH AND WITHOUT AN EXTRA DEGREE OF FREEDOM IN THE WAIST. FOR TRACBOT, EVALUATION IS PERFORMED FOR BOTH THE 7-DOF ARM AND A 6-DOF VERSION OF THE ARM, WHICH IS OFTEN USED IN PRACTICE TO REDUCE POTENTIAL COLLISIONS OF THE DUAL ARM SYSTEM. THE 6-DOF ARMS FROM THE 2013 VERSION OF ATLAS, ALONG WITH 3 LONGER CHAINS COMPRISED OF VARIOUS BACK JOINTS, WAS EVALUATED. SIMILARLY, THE 2015 ATLAS AND THE VALKYRIE HUMANOIDS THAT BOTH HAVE 7-DOF ARMS AND 3 BACK JOINTS HAD 7–10 DOF CHAINS TESTED.

Humanoid Kinematics Chain			IK Error	IK Technique													
				Plain KDL		KDL-RR		SQP		SQP-DQ		SQP-SS		SQP-L2		TRAC-IK	
Robot	Chain Description	DOFs	Position/ Rotation Error	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)
Robonaut2	Arm + Waist	8	1E-6 / 1E-6	78.18	0.61	85.36	0.75	17.93	3.43	96.46	1.06	97.76	0.91	97.12	1.81	98.72	0.68
	Arm	7	1E-6 / 1E-6	85.82	0.61	91.21	0.50	27.66	2.95	97.76	0.88	98.85	0.64	98.29	1.31	99.54	0.40
	Leg + Waist	8	1E-6 / 1E-6	76.05	0.71	81.21	0.83	17.41	3.52	96.93	1.34	97.99	1.16	97.08	2.17	99.13	0.71
	Leg	7	1E-6 / 1E-6	60.82	0.58	77.37	0.87	22.31	3.42	97.59	1.32	98.36	1.13	97.81	1.88	99.26	0.73
TRACBot	Long Arm	7	1E-6 / 1E-6	78.88	0.39	90.13	0.59	26.85	3.24	99.85	0.93	99.88	0.75	99.83	1.73	99.95	0.44
	Short Arm	6	1E-6 / 1E-6	46.09	0.17	71.86	0.63	26.91	2.92	98.55	1.09	97.45	1.01	97.08	1.56	98.81	0.68
Atlas 2013	Arm + Back (r,p,y)	9	1E-6 / 1E-6	89.59	0.79	91.95	0.84	31.00	3.04	99.57	0.65	99.88	0.47	99.72	1.37	99.89	0.39
	Arm + Back (p,y)	8	1E-6 / 1E-6	87.76	0.78	91.21	0.85	28.69	2.75	99.82	0.60	99.91	0.41	99.88	1.14	99.90	0.38
	Arm + Back yaw	7	1E-6 / 1E-6	85.36	0.59	90.84	0.69	30.53	2.63	99.84	0.53	99.92	0.36	99.89	0.95	99.95	0.34
	Arm	6	1E-6 / 1E-6	75.53	0.15	90.66	0.25	27.04	2.25	99.71	0.60	99.11	0.43	99.23	0.83	99.85	0.26
Atlas 2015	Arm + Back (r,p,y)	10	1E-6 / 1E-6	93.93	0.75	94.18	0.76	19.93	3.35	98.90	0.94	99.44	0.81	99.37	1.73	99.65	0.56
	Arm + Back (p,y)	9	1E-6 / 1E-6	91.22	0.83	91.99	0.85	22.00	3.45	99.28	0.86	99.52	0.78	99.54	1.47	99.72	0.53
	Arm + Back yaw	8	1E-6 / 1E-6	83.26	0.66	86.60	0.72	27.65	3.13	99.20	0.79	99.53	0.68	99.57	1.25	99.50	0.51
	Arm	7	1E-6 / 1E-6	75.39	0.39	85.50	0.55	28.38	3.02	98.98	0.78	99.32	0.65	99.28	1.09	99.45	0.42
Valkyrie	Arm + Back (r,p,y)	10	1E-6 / 1E-6	84.32	1.08	85.43	1.10	24.97	3.26	99.63	0.85	99.82	0.63	99.72	1.69	99.90	0.57
	Arm + Back (p,y)	9	1E-6 / 1E-6	77.62	1.25	80.40	1.32	24.91	3.36	99.38	0.82	99.65	0.60	99.44	1.45	99.85	0.58
	Arm + Back yaw	8	1E-6 / 1E-6	66.73	1.08	77.49	1.28	27.90	3.06	99.17	0.82	99.69	0.58	99.52	1.26	99.67	0.59
	Arm	7	1E-6 / 1E-6	44.83	0.60	82.10	1.16	31.67	2.96	99.05	0.90	99.61	0.62	99.40	1.16	99.83	0.51

4) TRAC-IK*:* By running a pseudoinverse Jacobian IK method and a nonlinear optimization IK method concurrently, an increase in solve rate was achieved as expected. As shown in Table I, TRAC-IK consistently outperforms all other IK algorithms tested in terms of solve rate. The three (out of eighteen total) chains where it is beat by another algorithm (Atlas 2013 8-DOF chain, Atlas 2015 8-DOF chain, Valkyrie 8-DOF chain) is only by a total of 10 solves out of 30,000. Such small numbers are within the expected variation due to the randomness of the nonlinear search methods and the randomness of the restarts when local minima are detected. This result, given the 180,000 samples over 18 chains, is statistically significant, with the successes of TRAC-IK being unquestionably different than those of SQP-SS with p-value$<5\text{E}{-7}$ (McNemar's test χ^2=274.72). Thus, TRAC-IK is the clear winner in terms of IK solve rate.

As planned, TRAC-IK not only outperforms in terms of solve rate, but also improves upon the overall runtime of all the other IK methods tested. It largely outperforms SQP-SS, except for a few instances on the Valkyrie robot configurations, where the algorithms are essentially the same speed. Similarly, TRAC-IK outperforms KDL-RR on all chains $>$6 degrees of freedom, and even those smaller 6-DOF chains have equal runtimes for the two IK implementations.

B. Solving IK for Longer Manipulation Chains

The results above demonstrate the poor performance of KDL-RR versus SQP methods on typical manipulation chains of 6–9 degrees of freedom. However, there are robots, like the Robonaut2 humanoid, that use longer chains for manipulation. The Robonaut2 robot uses its "legs" and "feet" as arms and hands to move around the International Space Station in a quasi-static fashion. So, for Robonaut2, computing the IK of a foot or hand with respect to another "static" foot that is grasping a handrail is not uncommon. KDL typically performs reasonably well on long, redundant chains (here 14–15 DOFs), as joint limits do not typically affect overall progress towards the target pose. If KDL performs well on longer chains, the TRAC-IK algorithm is expected to handle these scenarios

TABLE II

IK SOLVER EVALUATIONS ON LONGER MANIPULATION CHAINS. TESTS WERE RUN ON A 14-DOF CHAIN, WHICH STARTS ON THE LEFT FOOT OF THE ROBONAUT2 HUMANOID AND ENDS AT THE RIGHT ROOT, AND A 15-DOF CHAIN THAT STARTS AT THE LEFT FOOT AND ENDS AT THE LEFT HAND.

Humanoid Kinematics Chain			IK Error	IK Technique													
Robot	Chain Description	DOFs	Position/ Rotation Error	Plain KDL		KDL-RR		SQP		SQP-DQ		SQP-SS		SQP-L2		TRAC-IK	
				Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)
Robonaut2	Arm + Waist + Leg	15	1E-6 / 1E-6	97.62	0.78	97.65	0.78	0.0	–	98.37	1.41	99.14	1.25	88.86	4.64	99.87	0.70
	Left Leg + Right Leg	14	1E-6 / 1E-6	96.55	0.52	96.59	0.52	0.0	–	95.90	1.27	96.96	1.10	93.07	3.94	99.03	0.56

as well; thus, the algorithms were evaluated on the 14-DOF chain between the two feet of the Robonaut2 humanoid and the 15-DOF chain between the left foot and left hand of the Robonaut2 humanoid.

Table II shows the solve rates and run times of the six algorithms on the two long manipulation chains. Note that KDL-RR performs better on these higher-DOF chains, where limitations on joint range are less of an issue in achieving a solution. Of the 20,000 random Cartesian poses evaluated on these two chains, TRAC-IK solved for 99.45% of them while SQP-SS only solved 98.05% and KDL-RR solved 97.1%. On the other hand, KDL-RR took an average of 0.65 ms per solve and SQP-SS took an average of 1.175 ms per solve, while TRAC-IK took only 0.63 ms per solve. This illustrates that even as the chain length increases, which is a problem for methods like SQP-L2 in terms of finding solutions quickly and SQP in terms of finding solutions at all, TRAC-IK still finds the most solutions in the shortest amount of time.

V. SYSTEM INTEGRATION RESULTS

The TRAC-IK algorithm has been integrated into a trajectory control system for the simulated Robonaut2 robot for an end-to-end evaluation of the algorithm. TRAC-IK was tested both against a timer-wrapped version (again, all 5 ms timeouts) of KDL's pseudoinverse Jacobian IK algorithm and against KDL-RR. This test used a simulated Robonaut2 humanoid, where the Robonaut2 control system was given a predetermined pose trajectory that moves a tooltip across the surface of a stationary rectangular object.

A video demonstrating this comparison side-by-side has been made available.[8] While the video shows only a single run for each comparison, in fact *every* experimental run demonstrated that the TRAC-IK algorithm will solve each point faster than the stock KDL IK algorithm. Additionally, with TRAC-IK, *all* points along both trajectories are always considered reachable, whereas KDR-RR and the stock KDL inverse Jacobian IK algorithm sometimes fail to solve for reachable poses.

VI. ADDING TOLERANCES TO INVERSE KINEMATICS

The results in Section IV detail that the SQP methods described in Section II improve the IK solve rates over inverse Jacobian methods; however, these results were all gathered by solving IK for *exact* (within 1E−6) 3D Cartesian poses that were known to be reachable. As discussed in Section I, it is desirable to allow for tolerances on the Cartesian poses in the IK solution. Ensuring that the IK algorithms from Section II adequately handle tolerances on the Cartesian pose improves the robustness of IK solutions for many common scenarios, including straight-line motion, tool use, or Cartesian motion with a robot whose low-level control system is not extremely precise (e.g., the hydraulic arms of the Atlas humanoid robot).

Often, as in MoveIt!, such Cartesian tolerances are handled at the planning stage. That is, a higher-level algorithm knows that there are tolerances for x, y, z, $roll$, $pitch$, and yaw, and searches for an IK answer by calling an IK solver repeatedly on *exact* Cartesian poses that are within the tolerances until an answer is found (if one exists at all). In doing this, the underlying IK solver is actually repeatedly searching a lot of the same space, making this search very inefficient. Instead, it is preferable that the IK solver itself incorporate these tolerances. This means 1) the IK solver only has to be called once per desired Cartesian pose and 2) the IK solver can use the tolerances to bias the search and get even faster performance.

If the stock KDL IK implementation is used as is, it expects a full 6-dimensional Cartesian *pose* in order to run. The "max error" ϵ can be changed, but affects all dimensions equally, which is often not desired. Furthermore, ϵ is only used to check for termination ($p^i_{err} \leq \epsilon$) and does not bias the IK solver search. However, with minor changes to the definition of p_{err}, tolerances can be used to not only evaluate IK solutions, but also to bias the iterative IK search.

In Equations 1–5, if tolerances t_{err} are provided, the i elements in p_{err} can be redefined prior to multiplication with an inverse Jacobian for KDL-RR, to evaluating the constraints in SQP, to computing the dual quaternion error in SQP-DQ, or to computing the error metric during non-linear optimization for SQP-SS and SQP-L2:

$$p^i_{err} = \begin{cases} 0, & \text{if } |p^i_{err}| \leq |t^i_{err}| \\ p^i_{err}, & \text{otherwise.} \end{cases} \quad (6)$$

This new p_{err} is also used to evaluate against ϵ for termination.

The benefits of adding tolerances to the IK solver is

[8] http://traclabs.com/~pbeeson/CRAFTSMAN/videos/TRACIKvsKDL.mp4

TABLE III

TRAJECTORY TESTS FOR THE ATLAS 2013 6-DOF ARM AND ATLAS 2015 7-DOF ARM. AT FIRST, DENSE TRAJECTORIES THAT SPECIFIED EXACT POSES BETWEEN RANDOMLY GENERATED TARGET POINTS WERE TESTED. SECOND THE SAME TRAJECTORIES WERE RUN THROUGH THE VARIOUS IK SOLVERS (MODIFIED TO USE TOLERANCES), BUT WITH UNBOUNDED TOLERANCES ON THE $roll, pitch, yaw$ ERROR.

Humanoid Kinematics Chain			IK Error	IK Technique													
Robot	Chain Description	DOFs	Position/ Rotation Error	Plain KDL		KDL-RR		SQP		SQP-DQ		SQP-SS		SQP-L2		TRAC-IK	
				Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)
Atlas 2013	Arm	6	1E-6 / 1E-6	27.53	0.03	37.35	0.03	5.96	2.57	37.75	0.31	37.75	0.19	37.75	0.54	37.75	0.05
	Arm	6	1E-6 / ∞	–	–	92.42	0.05	93.89	0.47	93.97	0.15	93.96	0.10	93.96	0.28	93.96	0.06
Atlas 2015	Arm	7	1E-6 / 1E-6	71.11	0.19	94.13	0.22	21.11	2.74	96.57	0.40	96.55	0.25	96.52	0.66	96.56	0.10
	Arm	7	1E-6 / ∞	–	–	98.05	0.07	99.66	0.54	99.77	0.19	99.77	0.15	99.77	0.33	99.77	0.06

demonstrated by an experiment using the Atlas 2013 6-DOF arm model. Two scenarios were tested, one with exact poses required (desired error is $\leq$1E$-$6 for x,y,z,$roll$,$pitch$,yaw) for a 6-DOF arm and one where only the 3D location is desired ($roll$,$pitch$,yaw error is unbounded). Each was tested on the same data, which was generated by taking random, reachable Cartesian poses and creating dense, straight line trajectories, interpolating both location and angle between the initial and target poses at 1 cm/1 degree increments. New target poses were considered until 10,000 of these dense waypoints had been specified. The IK algorithms were then run for each waypoint along the trajectory, seeding the solvers with the solution from the previous waypoint—if a waypoint IK solve failed, the next waypoint along the trajectory was given the seed from the last successful IK solve.

The 6 different IK solvers were tested, and the results are shown in Table III for Atlas 2013. In the fully constrained case, because both position and orientation were interpolated between randomly selected Cartesian points, the IK implementations were expected to largely fail, as many of the requested poses are simply unreachable by the 6-DOF arm. On the other hand, if only a straight line trajectory is desired, the 6-DOF arm can follow this trajectory reliably by using TRAC-IK. Note that $\sim$6% of the points along the randomly computed trajectories actually fall outside the configuration space of the robot; however, by ignoring those points, a reasonable approximation of straight line motion is computed by KDL-RR 98.46% of the time and TRAC-IK 100% of the time. Additionally, Table III illustrates (albeit an extreme example) the drastic decrease in compute time that incorporating tolerances can provide to IK solvers. Table III also shows the results on the 7-DOF version of the Atlas 2015 arm, which reinforces the points above.

It should be noted that although expanding TRAC-IK to use tolerances had a positive affect on finding solutions, the current solution does not also constrain the amount of joint motion that could occur; thus, it is not guaranteed that actual joint trajectories will have minimal motion—though the trajectories in the integration tests discussed in Section V always resulted in smooth joint motion with no "rollouts". Such secondary criteria for acceptable IK solutions are left for future work; however, the fast solve times of TRAC-IK make it an ideal IK method to use if multiple IK solutions are desired for further comparison.

VII. CONCLUSIONS

This article investigated how to improve upon several failure points of KDL's implementation of joint-limited pseudoinverse Jacobian inverse kinematics. Specifically, the low success rates of the stock KDL IK implementation are outperformed by adding local minimum detection and mitigation, by reformulating IK as a sequential quadratic programming problem, and by utilizing Cartesian tolerances (when desired) to speed up the IK search. The various IK algorithms detailed in Section II were implemented, and the results demonstrated in Sections IV–VI show improved numerical convergence, increasing IK solve rates, and improved runtime.

The presented results clearly show the benefit of nonlinear-optimization-based methods for performing IK on robots with limited joint ranges, with SQP-SS achieving a 99.21% IK success rate over 180,000 random samples, and KDL-RR only reaching 85.86% success. Furthermore, the best overall algorithm, TRAC-IK, demonstrated the benefit of concurrently running inverse Jacobian and simple non-linear optimization to achieve a 99.59% IK success rate over 180,000 random samples while improving on the average time needed for a single IK solution to converge. Finally, it was demonstrated that by allowing dimensional tolerances in the IK solver directly, adequate solutions for many manipulation domains can be found reliably and quickly without need for a higher-level search over feasible Cartesian poses. This is useful in scenarios where control between exact Cartesian poses is not wanted or not particularly useful.

All of the algorithms discussed in Section II are open-source, "drop in" replacements to KDL's ChainIkSolverPos_NR_JL class (also referred to as "Plain KDL" in Tables I–III) and are available at https://bitbucket.org/traclabs/trac_ik/.

VIII. Future Work

The algorithms presented here are meant to be a beginning. They were created out of necessity, due to a lack of quality solutions provided by the stock KDL IK implementations for joint-limited humanoid robots. While they work well, the proposed implementations were created to minimize overall runtime—returning the first solution (of possibly many) that meets the specified input requirements. However, the non-linear optimization framework for IK seems to be fertile ground for further exploration. Specifically, it allows further objectives to be specified to meet secondary criteria relevant to the robot or task configuration. These objectives may bias the robot toward areas of the workspace that are kinematically well conditioned, both with respect to intrinsic qualities of the robot (e.g., always keep the elbow pointed away from body), or with respect to desired task-space requirements (e.g., move to where the robot can exert wrenches most effectively to tighten a bolt; move to where the robot has the best command fidelity to place a peg in a tightly constrained hole). Robot and task conditioning approaches have been well studied in the literature and have been deployed in multiple contexts [8], [9], [10], [11], [12], [13], typically through the use of prioritized, null space control [14] that can provide a level of response or reactivity more difficult to achieve in planning algorithms. The integration of all of these secondary criteria into the `TRAC-IK` implementation is left to future investigation.

ACKNOWLEDGMENT

The authors would like to thank Seth Gee and Stephen Hart for their help in creating the simulation setup and video described in Section V.

References

[1] R. Nilsson, "Inverse kinematics," Master's thesis, Luleå University of Technology, 2009.

[2] S. Kumar, N. Sukavanam, and R. Balasubramanian, "An optimization approach to solve the inverse kinematics of redundant manipulator," *International Journal of Information and System Sciences (Institute for Scientific Computing and Information)*, vol. 6, no. 4, pp. 414–423, 2010.

[3] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, T. T. Koolen, P. Marion, M. Posa, A. Valenzuela, K.-T. Yu, J. Shah, K. Iagnemma, R. Tedrake, and S. Teller, "An architecture for online affordance-based perception and whole-body planning," *Journal of Field Robotics*, vol. 32, no. 2, pp. 229–254, March 2015.

[4] D. Kraft, "A software package for sequential quadratic programming," DLR German Aerospace Center–Institute for Flight Mechanics, Köln, Germany, Tech. Rep. DFVLR-FB 88-28, July 1988.

[5] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical Optimization: Theoretical and Practical Aspects*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[6] B. Kenwright, "A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d character hierarchies," in *International Conference on Computer Graphics, Visualization and Computer Vision*, 2012, pp. 1–13.

[7] D.-P. Han, Q. Wei, and Z.-X. Li, "Kinematic control of free rigid bodies using dual quaternions," *International Journal of Automation and Computing*, vol. 5, no. 3, pp. 319–324, 2008.

[8] T. Yoshikawa, "Manipulability of Robotic Mechanisms," *The International Journal of Robotics Research*, vol. 4, pp. 3–9, 1985.

[9] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Robotics & Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[10] S. Chiu, "Control of redundant manipulators for task compatability," in *International IEEE Conference on Robotics and Automation (ICRA)*, vol. 4, March 1987, pp. 1718–1724.

[11] ——, "Kinematic characterization of manipulators: An approach to defining optimality," in *International IEEE Conference on Robotics and Automation (ICRA)*, vol. 2, April 1988, pp. 828–833.

[12] L. Sentis and O. Khatib, "Task-oriented control of humanoid robots through prioritization," in *IEEE International Conference on Humanoid Robots*, 2004.

[13] S. Hart and R. Grupen, "Natural task decomposition with intrinsic potential fields," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007, pp. 2507–2512.

[14] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.